OPTA-PROG-MODBUSRS485-HMIDELTA

PT-BR

Programação OPTA com integração via Modbus RTU com a IHM Delta

1. Informações Gerais

- Versão / Revisão: Revisão 1
- Data de Desenvolvimento: 19/07/2024
- Autor: Daniel Arcos
- Linguagem: C++
- Plataforma / Hardware: Finder OPTA

2. Objetivo da Programação

O objetivo desta programação é estabelecer comunicação via Modbus do OPTA com a IHM da Delta para um exemplo simples de controle de nível. Para ter acesso aos arquivos da IHM, entrar em contato com a Engenharia Finder Brasil solicitando o mesmo.

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 bibliotecas utilizadas na programação
- IHM Delta
- Medidor de Energia
- Potenciometro para variação de nível
- Fonte de alimentação
- Software de Configuração para a IHM Delta

4. Passo a Passo da Implementação

Configuração da IHM Delta:

1. Configurar comunicação Modbus RTU:
    - Baud rate: 19200
    - Parity: None
    - Stop bits: 2
    - Endereço da IHM: 2
2. Mapear variáveis nos registradores Modbus:
    - Registrador 1: CTR (contador)
    - Registrador 4: V (tensão)
    - Registrador 5: P1 (potência)
    - Registrador 6: voltageA0 (tensão A0)
    - Registrador 7: liga (estado liga/desliga)
    - Registrador 8: b1 (botão 1)
    - Registrador 9: b2 (botão 2)
    - Registrador 10: voltageA1 (tensão A1)

Verifique o Datasheet do modelo da IHM escolhido para identificar os terminais corretos do Modbus RTU, ligue-os e lembre-se de utilizar um resistor de terminação nas extremidades.

5. Código completo

Todo o arquivo da programação está disponível para download nos arquivos.

Visão Geral

Definições e Variáveis Globais:

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

// Variáveis para armazenamento de dados
unsigned int ctr = 0;
float v = 0;          // Tensão do medidor
float p = 0;          // Potência do medidor
float p1 = 0;         // Potência multiplicada (para display)
float voltageA0 = 0;  // Tensão do sensor A0
float voltageA1 = 0;  // Tensão do sensor A1
int liga = 0;         // Estado liga/desliga
int b1 = 0;           // Botão 1
int b2 = 0;           // Botão 2

// Configuração Modbus
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 };
```

Função setup (Configuração Inicial):

```
void setup() {
  Serial.begin(115200);
  delay(400);

  // Configuração RS485
```

```
  RS485.setDelays(predl, postdl);

  // Inicialização Modbus RTU
  if (!ModbusRTUClient.begin(baudrate, SERIAL_8N2)) {
    Serial.println("Erro Modbus");
    while (1); // Loop infinito em caso de erro
  }

  // Configuração de pinos
  analogReadResolution(12); // Alta resolução para leituras analógicas
  pinMode(LED_D0, OUTPUT);
  pinMode(D0, OUTPUT);
  pinMode(LED_USER, OUTPUT);
  pinMode(LEDR, OUTPUT);
}
```

Função loop (Loop Principal):

```
void loop() {
  ctr++; // Incrementa contador

  // Transmissão de dados para IHM Delta
  ModbusRTUClient.beginTransmission(2, HOLDING_REGISTERS, 1, 10);
  ModbusRTUClient.write(ctr);      // Registrador 1
  ModbusRTUClient.write(2);        // Registrador 2
  ModbusRTUClient.write(3);        // Registrador 3
  ModbusRTUClient.write(v);        // Registrador 4 - Tensão
  ModbusRTUClient.write(p1);       // Registrador 5 - Potência
  ModbusRTUClient.write(voltageA0);// Registrador 6 - Tensão A0
  ModbusRTUClient.write(liga);     // Registrador 7 - Estado
  ModbusRTUClient.write(b1);       // Registrador 8 - Botão 1
  ModbusRTUClient.write(b2);       // Registrador 9 - Botão 2
  ModbusRTUClient.write(voltageA1);// Registrador 10 - Tensão A1

  // Leitura de comando da IHM
  auto estadoSaida = ModbusRTUClient.coilRead(2, 1);
```

```
  // Finaliza transmissão
  if (!ModbusRTUClient.endTransmission()) {
    Serial.print("failed! ");
    Serial.println(ModbusRTUClient.lastError());
  }

  // Leitura de dados do medidor de energia
  p = readdata(0x21, 0X9BA); // Potência ativa total
  v = readdata(0x21, 0X9C4); // Tensão
  float i = readdata(0x21, 0X9D4); // Corrente (não usada no display)

  // Exibição no Serial Monitor
  Serial.println(String(p, 1) + "W " + String(v, 1) + "V " + String(i, 3) + "A ");
  delay(400);

  // Processamento de dados
  p1 = p * 10; // Ajuste de escala para display

  // Leitura de sensores analógicos
  voltageA0 = readAnalogVoltage(A0);
  voltageA1 = readAnalogVoltage(A1);

  // Controle de saídas baseado no estado da IHM
  controlOutputs(estadoSaida);
}
```

Funções Auxiliares:

```
// Leitura de dados Modbus (medidor de energia)
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; // Conversão para float
  }
  return res;
}

// Leitura de tensão analógica
float readAnalogVoltage(int pin) {
  int sensorValue = analogRead(pin);
  return sensorValue * (3.0 / 4095.0) / 0.3; // Conversão para tensão
}

// Controle de saídas baseado no estado da IHM
void controlOutputs(bool estadoSaida) {
  if (estadoSaida) {
    // Modo "acionado" - piscar LEDs
    digitalWrite(LED_USER, HIGH);
    digitalWrite(LEDR, LOW);
    delay(300);
    digitalWrite(LED_USER, LOW);
    digitalWrite(LEDR, HIGH);
    delay(50);
    digitalWrite(LED_D0, LOW);
    digitalWrite(D0, LOW);
  } else {
    // Modo "repouso" - LEDs acesos
    digitalWrite(LED_D0, HIGH);
    digitalWrite(D0, HIGH);
    digitalWrite(LED_USER, HIGH);
    digitalWrite(LEDR, HIGH);
  }
}
```

6. Funcionamento do Sistema

Este código implementa um sistema completo de monitoramento e controle com:

Comunicação Modbus RTU:

- **Mestre**: Finder OPTA
- **Escravos**: IHM Delta (endereço 2) e Medidor de Energia (endereço 0x21)
- **Configuração**: baud rate 19200, 8N2

Fluxo de Dados:

1. **Leitura do Medidor**: Obtém tensão, corrente e potência do medidor de energia
2. **Leitura de Sensores**: Lê valores analógicos das entradas A0 e A1
3. **Transmissão para IHM**: Envia dados para display na IHM Delta
4. **Recebimento de Comandos**: Lê estado de botões/controles da IHM
5. **Controle de Saídas**: Aciona LEDs e saída digital baseado no estado da IHM

Variáveis Monitoradas:

- **Dados do Medidor**: Potência (W), Tensão (V), Corrente (A)
- **Sensores Analógicos**: Tensões nas entradas A0 e A1
- **Estado do Sistema**: Variáveis de controle (liga, b1, b2)

Aplicações Típicas:

- Sistemas de monitoramento de energia
- Painéis de controle industrial
- Sistemas de automação predial
- Monitoramento de processos industriais

Características Importantes:

1. **Operação Offline**: Funciona sem dependência de nuvem
2. **Tempo Real**: Atualizações rápidas (400ms)
3. **Debug Completo**: Informações detalhadas no Serial Monitor
4. **Controle Visual**: Feedback através de LEDs
5. **Escalável**: Pode ser expandido com mais variáveis

EN

OPTA programming with integration via Modbus RTU with Delta HMI

1. General Information

- Version / Revision: Revision 1
- Development Date: July 19, 2024
- Author: Daniel Arcos
- Language: C++
- Platform / Hardware: Finder OPTA

2. Programming Objective

The objective of this programming is to establish Modbus communication between the OPTA and Delta's HMI for a simple level control example. To access the HMI files, contact Finder Brasil Engineering and request them.

3. Requirements and Dependencies

- Install the latest version of the Arduino IDE
- Install the OPTA hardware drivers in the Arduino IDE
- Install the OPTA library
- Install libraries used in programming
- Delta HMI
- Energy Meter
- Level Variation Potentiometer

- Power Supply
- Configuration Software for the Delta HMI

4. Implementation Step-by-Step

Delta HMI Configuration:

1. Configure Modbus RTU communication:

- Baud rate: 19200
- Parity: None
- Stop bits: 2
- HMI address: 2

1. Map variables to Modbus registers:

- Register 1: CTR (counter)
- Register 4: V (voltage)
- Register 5: P1 (power)
- Register 6: voltageA0 (voltage A0)
- Register 7: power (on/off state)
- Register 8: b1 (button 1)
- Register 9: b2 (button 2)
- Register 10: voltageA1 (voltage A1)

Check the datasheet for the chosen HMI model to identify the correct Modbus RTU terminals, connect them, and remember to use a terminating resistor at the ends.

5. Complete Code

The entire programming file is available for download in the archives.

Overview

Definitions and Global Variables:

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

// Variáveis para armazenamento de dados
unsigned int ctr = 0;
float v = 0;          // Tensão do medidor
float p = 0;          // Potência do medidor
float p1 = 0;         // Potência multiplicada (para display)
float voltageA0 = 0;  // Tensão do sensor A0
float voltageA1 = 0;  // Tensão do sensor A1
int liga = 0;         // Estado liga/desliga
int b1 = 0;           // Botão 1
int b2 = 0;           // Botão 2

// Configuração Modbus
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 };
```

Setup function (Initial Configuration):

```
void setup() {
  Serial.begin(115200);
  delay(400);

  // Configuração RS485
  RS485.setDelays(predl, postdl);

  // Inicialização Modbus RTU
  if (!ModbusRTUClient.begin(baudrate, SERIAL_8N2)) {
    Serial.println("Erro Modbus");
    while (1); // Loop infinito em caso de erro
```

```
  }

  // Configuração de pinos
  analogReadResolution(12); // Alta resolução para leituras analógicas
  pinMode(LED_D0, OUTPUT);
  pinMode(D0, OUTPUT);
  pinMode(LED_USER, OUTPUT);
  pinMode(LEDR, OUTPUT);
}
```

Loop function (Main Loop):

```
void loop() {
  ctr++; // Incrementa contador

  // Transmissão de dados para IHM Delta
  ModbusRTUClient.beginTransmission(2, HOLDING_REGISTERS, 1, 10);
  ModbusRTUClient.write(ctr);      // Registrador 1
  ModbusRTUClient.write(2);        // Registrador 2
  ModbusRTUClient.write(3);        // Registrador 3
  ModbusRTUClient.write(v);        // Registrador 4 - Tensão
  ModbusRTUClient.write(p1);       // Registrador 5 - Potência
  ModbusRTUClient.write(voltageA0);// Registrador 6 - Tensão A0
  ModbusRTUClient.write(liga);     // Registrador 7 - Estado
  ModbusRTUClient.write(b1);       // Registrador 8 - Botão 1
  ModbusRTUClient.write(b2);       // Registrador 9 - Botão 2
  ModbusRTUClient.write(voltageA1);// Registrador 10 - Tensão A1

  // Leitura de comando da IHM
  auto estadoSaida = ModbusRTUClient.coilRead(2, 1);

  // Finaliza transmissão
  if (!ModbusRTUClient.endTransmission()) {
    Serial.print("failed! ");
    Serial.println(ModbusRTUClient.lastError());
  }
```

```
  // Leitura de dados do medidor de energia
  p = readdata(0x21, 0X9BA); // Potência ativa total
  v = readdata(0x21, 0X9C4); // Tensão
  float i = readdata(0x21, 0X9D4); // Corrente (não usada no display)

  // Exibição no Serial Monitor
  Serial.println(String(p, 1) + "W " + String(v, 1) + "V " + String(i, 3) + "A ");
  delay(400);

  // Processamento de dados
  p1 = p * 10; // Ajuste de escala para display

  // Leitura de sensores analógicos
  voltageA0 = readAnalogVoltage(A0);
  voltageA1 = readAnalogVoltage(A1);

  // Controle de saídas baseado no estado da IHM
  controlOutputs(estadoSaida);
}
```

Auxiliary Functions:

```
// Leitura de dados Modbus (medidor de energia)
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; // Conversão para float
  }
  return res;
}
```

```
// Leitura de tensão analógica
float readAnalogVoltage(int pin) {
  int sensorValue = analogRead(pin);
  return sensorValue * (3.0 / 4095.0) / 0.3; // Conversão para tensão
}

// Controle de saídas baseado no estado da IHM
void controlOutputs(bool estadoSaida) {
  if (estadoSaida) {
    // Modo "acionado" - piscar LEDs
    digitalWrite(LED_USER, HIGH);
    digitalWrite(LEDR, LOW);
    delay(300);
    digitalWrite(LED_USER, LOW);
    digitalWrite(LEDR, HIGH);
    delay(50);
    digitalWrite(LED_D0, LOW);
    digitalWrite(D0, LOW);
  } else {
    // Modo "repouso" - LEDs acesos
    digitalWrite(LED_D0, HIGH);
    digitalWrite(D0, HIGH);
    digitalWrite(LED_USER, HIGH);
    digitalWrite(LEDR, HIGH);
  }
}
```

6. System Operation

This code implements a complete monitoring and control system with:

Modbus RTU Communication:

- **Master**: Finder OPTA
- **Slaves**: Delta HMI (address 2) and Energy Meter (address 0x21)
- **Configuration**: 19200 baud rate, 8N2

Data Flow:

1. **Meter Reading**: Obtains voltage, current, and power from the energy meter
2. **Sensor Reading**: Reads analog values from inputs A0 and A1
3. **Transmission to HMI**: Sends data to the display on the Delta HMI
4. **Command Receiving**: Reads the status of HMI buttons/controls
5. **Output Control**: Activates LEDs and digital outputs based on the HMI status

Monitored Variables:

- Meter Data: Power (W), Voltage (V), Current (A)
- Analog Sensors: Voltages at inputs A0 and A1
- System Status: Control variables (on, b1, b2)

Typical Applications:

- Energy Monitoring Systems
- Industrial Control Panels
- Building Automation Systems
- Industrial Process Monitoring

Important Features:

1. Offline Operation: Works without cloud-based functionality
2. Real-Time: Fast updates (400ms)
3. Full Debugging: Detailed information on the Serial Monitor
4. Visual Control: Feedback via LEDs
5. Scalable: Can be expanded with more variables