INSTITUTO FEDERAL
Ceará
Campus Caucaia

E V O F R E

CURSO TÉCNICO INTEGRADO EM ELETROELETRÔNICA

Disciplina: SDCD

Facilitador: Professor José Tarcízio Gomes Filho
Professor José Rogério Maciel Ferreira Filho
Professor Pedro Henrique Almeida Miranda
Chat GPT

Atividade:

Autores: Marcos Nicolau de Oliveira Costa
Gilvan Gomes de Araújo Filho

Nome do Projeto: EVOFRE

2024.2

1. INTRODUÇÃO

Os inversores de frequência desempenham um papel fundamental no cenário industrial e comercial. Esses dispositivos são responsáveis por controlar a velocidade e o torque de motores elétricos, otimizando o consumo de energia, prolongando a vida útil dos equipamentos e promovendo maior flexibilidade operacional. Sua importância vai além da eficiência energética, pois também possibilitam processos mais precisos e sustentáveis em diversos setores, sendo uma ferramenta muito eficiente.

Ao longo das últimas décadas, a sociedade contemporânea vêm observado o avanço da tecnologia em todos os segmentos e nesse sentido surgiu o projeto EVOFRE, acrônimo de “Evolução da Frequência”, automatizando inversores de frequência permitindo operações remotas por meio de sistemas supervisórios.

2. JUSTIFICATIVA

Tendo em vista o amplo uso de inversores de frequência em várias aplicações, esse projeto foi escolhido com a finalidade controlar e monitorar motores elétricos através do inversor de frequência.

3. LISTA DE MATERIAL:

Item	Descrição	QTD	Valor Unitário	Valor Total
01	Módulo WiFi ESP32 Bluetooth	1	R$ 49,99	R$ 49,99
02	Conversor TTL RS485	1	R$ 7,90	R$ 7,90
03	Inversor de Frequência Schneider ATV312 ATV312H037N4 0,5 CV	1	R$ 2.290	R$ 2.290
04	Motor Elétrico Trifásico AC WEG W22 ⅓ CV 220/380V	1	R$ 1.465	R$ 1.465
05	Cabo De Rede RJ45 Montado 30 metros Cat5	1	R$ 48,85	R$ 48,85
06	Kit Jumpers 20cm - 40 Vias	1	R$ 27,90	R$ 27,90
07	Mão de obra	2	R$ 15.512	R$ 31.024

*	**Total**	**8**	*	**34.913,64**

4. FUNCIONAMENTO:

Com as conexões estabelecidas (Item 6), o inversor de frequência é controlado por meio da comunicação Modbus RS-485, permitindo o controle e o monitoramento do motor. A conexão é feita a partir da conexão de um cabo RJ45 no inversor, que é conectado a um ESP32, por meio de um conversor TTL. O ESP32 possui uma programação que monitora e controla o sistema através de um supervisório web, hospedado pelo próprio microcontrolador.

No supervisório, o operador dispõe de controles intuitivos no painel para operar o motor elétrico. É possível acioná-lo por meio de botões e controlar a sua velocidade, garantindo simplicidade e eficiência no controle do sistema.

5. PROGRAMAÇÃO:

A programação do ESP32 foi elaborada na Linguagem C, confira-a logo abaixo:

```
#include <WiFi.h>
#include <ModbusMaster.h>
#include <ESPAsyncWebServer.h>
#include <AsyncTCP.h>

// Configuração do RS485
#define RX_PIN 16
#define TX_PIN 17
#define RE_DE_PIN 15

// Configuração do Wi-Fi
const char *ssid = "SSID";
const char *password = "SENHA";

// ModbusMaster e servidor
ModbusMaster node;
AsyncWebServer server(80);

// Variáveis de controle
uint16_t frequencyValue = 600;  // Frequência padrão (60 Hz *
10)

// Funções para controlar o RS485
void preTransmission() {
```

```
  digitalWrite(RE_DE_PIN, HIGH);  // Ativa transmissão
}

void postTransmission() {
  digitalWrite(RE_DE_PIN, LOW);  // Ativa recepção
}

void setup() {
  // Inicialização serial e RS485
  Serial.begin(115200);
  Serial1.begin(19200, SERIAL_8E1, RX_PIN, TX_PIN);

  pinMode(RE_DE_PIN, OUTPUT);
  digitalWrite(RE_DE_PIN, LOW);  // Modo recepção padrão

  node.begin(1, Serial1);
  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);

  // Conexão Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Conectando ao Wi-Fi...");
  }
  Serial.println("Conectado ao Wi-Fi. IP: " +
WiFi.localIP().toString());

  // Página inicial
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
{
    String html = R"rawliteral(
      <!DOCTYPE html>
      <html lang="en">
      <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Controle ESP32</title>
        <style>
          body { font-family: Arial, sans-serif;
background-color: #373737; margin: 0; padding: 0; display:
```

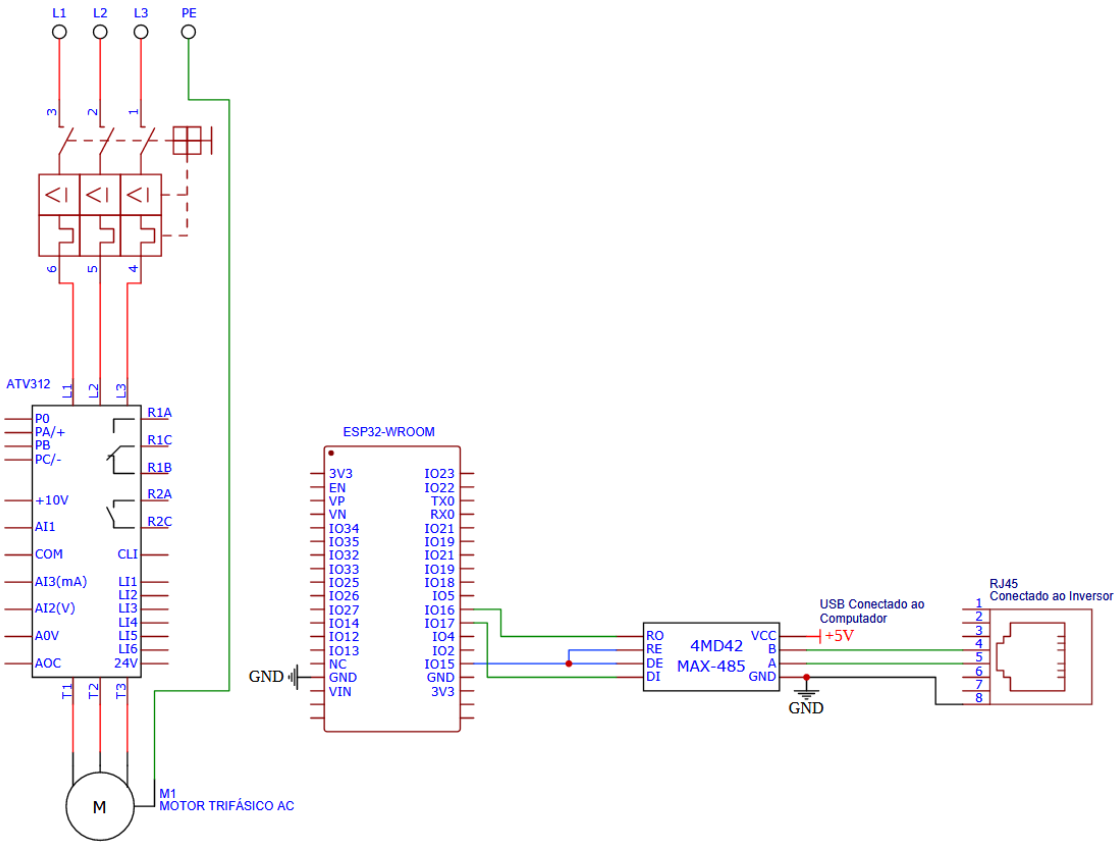
```
flex; flex-direction: column; justify-content: center;
align-items: center; min-height: 100vh; }
          .logo { margin-bottom: 20px; width: 40%; max-width:
400px; height: auto; }
          .gradient-border { position: relative; background:
linear-gradient(90deg, #131313, #3d3d3d); background-size:
300% 300%; border-radius: 12px; padding: 6px; animation:
gradient-animation 10s ease infinite; }
          .container { background-color: #ffffff;
border-radius: 10px; padding: 20px; max-width: 400px; width:
90%; text-align: center; }
          h1 { font-size: 24px; color: #333333; }
          button { background-color: #2b2b2b; color: white;
border: none; padding: 10px 20px; margin: 5px; border-radius:
5px; cursor: pointer; font-size: 16px; transition:
background-color 0.3s ease; }
          button:hover { background-color: #00b3d3; }
          input { padding: 8px; margin: 5px 0; width:
calc(100% - 20px); font-size: 16px; border: 1px solid #ccc;
border-radius: 5px; }
          label { font-size: 14px; color: #555555; display:
block; margin-bottom: 5px; }
          p { font-size: 14px; color: #333333; }
          @keyframes gradient-animation { 0% {
background-position: 0% 50%; } 50% { background-position: 100%
50%; } 100% { background-position: 0% 50%; } }
        </style>
      </head>
      <body>
        <img
src="https://i.postimg.cc/63cb4VkV/LOGO-WHITE.png"
class="logo" alt="Logo">
        <div class="gradient-border">
          <div class="container">
            <h1>Controle do Motor via Modbus</h1>
            <div>
              <button
onclick="sendCommand('RESET')">RESET</button>
              <button
onclick="sendCommand('SISTEMA')">INÍCIO</button>
            </div>
            <br>
```

```
            <div>
              <button
onclick="sendCommand('HORARIO')">HORÁRIO</button>
              <button
onclick="sendCommand('ANTIHORARIO')">ANTI-HORÁRIO</button>
            </div>
            <br>
            <div>
              <label for="frequency">Frequência (Hz): </label>
              <input id="frequency" type="number" min="0"
max="500" step="0.1">
              <button onclick="setFrequency()">ALTERAR
FREQUÊNCIA</button>
            </div>
          </div>
        </div>
        <script>
          function sendCommand(command) {
            fetch('/command?cmd=' + command)
              .then(response => response.text())
              .then(data => alert(data));
          }
          function setFrequency() {
            const freq =
document.getElementById('frequency').value;
            if (freq >= 0 && freq <= 500) {
              fetch('/setfreq?freq=' + (freq * 10))
                .then(response => response.text())
                .then(data => alert(data));
            } else {
              alert("Frequência inválida. Insira um valor
entre 0 e 500 Hz.");
            }
          }
        </script>
      </body>
      </html>
    )rawliteral";
    request->send(200, "text/html", html);
  });

  // Endpoint para comandos
```

```
  server.on("/command", HTTP_GET, [](AsyncWebServerRequest
*request) {
    String cmd = request->getParam("cmd")->value();
    uint8_t result;
    if (cmd == "RESET") result =
node.writeSingleRegister(0x2135, 0x0080);
    else if (cmd == "SISTEMA") result =
node.writeSingleRegister(0x2135, 0x0006);
    else if (cmd == "HORARIO") result =
node.writeSingleRegister(0x2135, 0x000F);
    else if (cmd == "ANTIHORARIO") result =
node.writeSingleRegister(0x2135, 0x080F);
    else result = 0xFF;  // Comando inválido

    if (result == node.ku8MBSuccess) {
      request->send(200, "text/plain", "Comando executado: " +
cmd);
    } else {
      request->send(500, "text/plain", "Erro no comando: " +
cmd);
    }
  });

  // Endpoint para alterar frequência
  server.on("/setfreq", HTTP_GET, [](AsyncWebServerRequest
*request) {
    String freqStr = request->getParam("freq")->value();
    frequencyValue = freqStr.toInt();
    uint8_t result = node.writeSingleRegister(0x2136,
frequencyValue);
    if (result == node.ku8MBSuccess) {
      request->send(200, "text/plain", "Frequência ajustada
para " + String(frequencyValue / 10.0) + " Hz.");
    } else {
      request->send(500, "text/plain", "Erro ao ajustar
frequência.");
    }
  });

  // Inicia o servidor
  server.begin();
}
```

```
void loop() {
}
```

6. ESQUEMA ELÉTRICO:



7. SUPERVISÓRIO:

O supervisório utilizado foi feito em HTML e CSS, com o back-end sendo viabilizado pela biblioteca Esp Async WebServer. O sistema supervisório possui apenas uma tela contendo botões de reset, início e de controle do sentido motor, além de um espaço em que você pode configurar a frequência. Abaixo você pode ver a tela inicial, que é exibida quando a aplicação inicia. Abaixo é possível ver a tela do supervisório:

8. COMUNICAÇÃO:

O ESP32 comunica com o inversor de frequência ATV312 da Schneider, através do protocolo Modbus RTU RS485. O ESP32 atua como mestre Modbus, controlando o dispositivo escravo, o inversor, com comandos enviados pela biblioteca Modbus Master através dos pinos RX, TX e RE/DE. Paralelamente, um servidor web assíncrono gerado pela biblioteca ESP Async WebServer fornece uma interface HTML onde o usuário pode enviar comandos ou configurar a frequência do motor. As requisições HTTP feitas pela interface acionam funções no ESP32 que traduzem os comandos para o protocolo Modbus, enviando ou recebendo dados do dispositivo e retornando as respostas ao navegador para exibição.

Foram utilizados as seguintes entradas nos respectivos endereços de memória:

Endereço de Memória (dec/hex)	**Função do Endereço de Memória**	**Comando (hex)**	**Função do Comando**
8501/0x2135	Palavra de comando	128/0x0080	Reset do inversor
8501/0x2135	Palavra de comando	6/0x0006	Início do sistema de controle
8501/0x2135	Palavra de comando	15/0x000F	Partida no sentido horário
8501/0x2135	Palavra de comando	2063/0x080F	Partida no sentido anti-horário
8502/0x2136	Ajuste do valor de frequência	*	Frequência desejada

9. CONCLUSÃO:

O desenvolvimento do projeto de monitoramento e controle do inversor de frequência EVOFRE demonstrou como a evolução tecnológica pode ser benéfica por facilitar o gerenciamento de sistemas e processos. Através da implementação das funcionalidades de monitoramento em tempo real e controle otimizado, foi possível alcançar maior eficiência no gerenciamento do inversor, garantindo precisão e segurança para possíveis operadores.

10. REFERÊNCIAS:

SCHNEIDER. Manual em português do Altivar 312 (ATV312). 2016.

SCHNEIDER. ATV312_communication variables_EN_V1. 2006.

OPENAI. *ChatGPT: Assistente virtual baseado em inteligência artificial*. Disponível em: https://chat.openai.com/.

DRIVE AUTOMAÇÃO E TECNOLOGIA. ATV312: Monitoramento em tempo real (Google Planilhas). YouTube, 3 de dezembro de 2024. 10min10s. Disponível em: https://www.youtube.com/watch?v=c4bASWKxWns&.

Plataforma IOT para controle do ATV312, como você nunca viu: Arduino Mega + Ethernet + Blynk. YouTube, 24 de março de 2021. 10min11s. Disponível em: https://www.youtube.com/watch?v=8WXOhrNj62Y.