

Roteiro Nº 05	ESP32 como Servidor Web em uma LAN – Controle de inversor de frequência	
Curso Engenharia Elétrica	Disciplina ELT 1119 – Redes e Aplicações IoT	Professor Carlos Alberto Vasconcelos Bezerra
Nome do Estudante		Data

1. Objetivos da Aprendizagem

- Desenvolver um servidor web embarcado no ESP32.
- Compreender como usar um slider HTML para entrada de dados em tempo real.
- Controlar uma saída digital do ESP32 remotamente por meio de um botão na interface.
- Aplicar os conceitos de comunicação HTTP, conversão digital-analógica (DAC) e controle de dispositivos.

2. Material Necessário

- Placa de desenvolvimento ESP32-DevKitC-V4;
- Computador com ambiente de desenvolvimento Thonny instalado;
- Cabo USB para comunicação entre o ESP32 e o computador;
- Inversor de frequência;
- Motor de indução trifásico;
- Placa de relé 24V

3. Fundamentação Teórica

O **ESP32** é um microcontrolador moderno com conectividade Wi-Fi integrada, ideal para aplicações em IoT (Internet das Coisas). Ele permite o controle de dispositivos físicos através de interfaces web hospedadas no próprio chip.

Neste experimento, utilizaremos dois recursos principais:

1. Conversor Digital-Analógico (DAC)

O ESP32 possui conversores DAC embutidos nos pinos GPIO25 e GPIO26, capazes de gerar uma tensão analógica entre 0 V e 3,3 V. Essa tensão pode ser usada para controlar um inversor de frequência que irá controlar a velocidade de um motor de indução trifásico.

2. Servidor Web Local

O ESP32 será configurado como um servidor web simples, que fornecerá uma página HTML com:

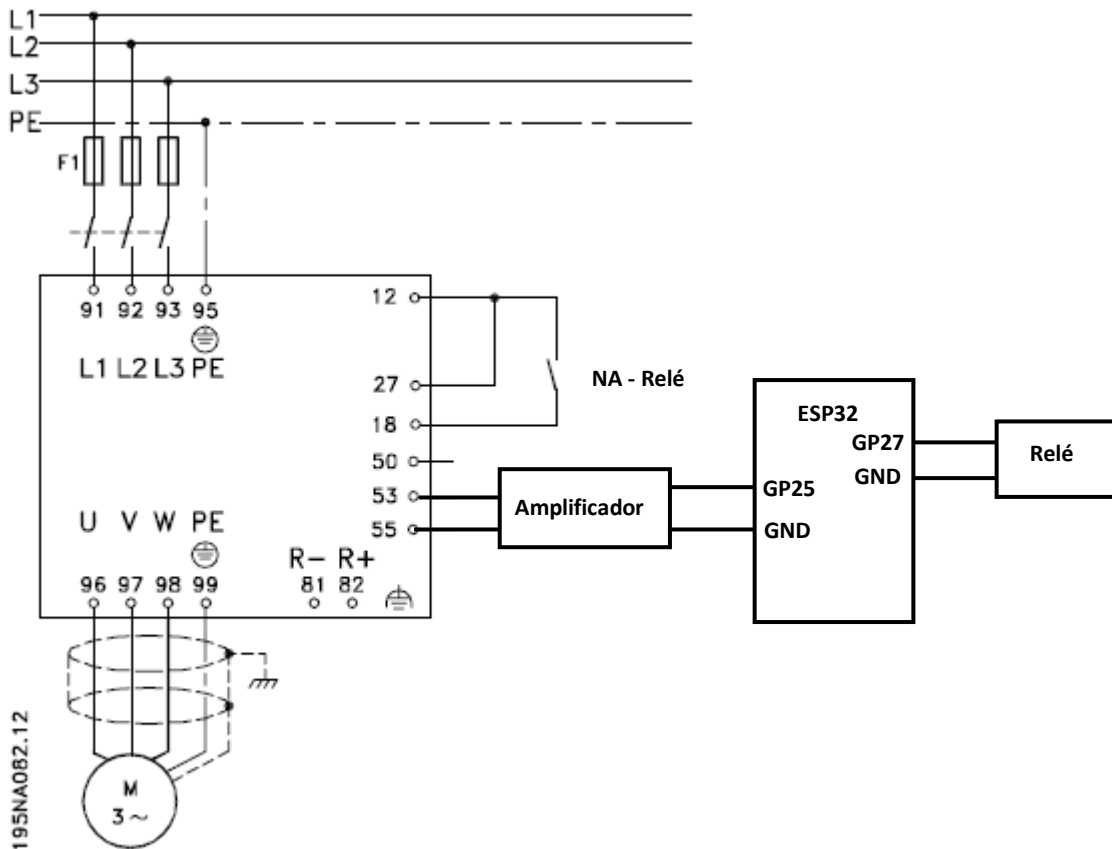
- Um **controle deslizante** que permitirá ao usuário selecionar um valor de frequência entre 0 e 120 Hz.
- Um **botão liga/desliga**, que ativará um relé em uma saída digital.
- Um **indicador visual** de estado da saída (verde = ligado, vermelho = desligado).

O valor do slider será convertido para uma tensão proporcional no DAC (0 a 255), que irá variar a frequência do inversor. O botão liga/desliga servirá para ligar e desligar um relé que dará partida/parada do motor.

4. Procedimento Experimental

4.1 Montagem do circuito:

Monte o circuito conforme diagrama abaixo, conectando a saída DAC(Pin 25) do ESP32 como indicado.



4.2 Configuração do Ambiente

- Abra o **Thonny IDE** e conecte-se ao ESP32.
- Digite o código abaixo no ambiente Thonny.

```
import network
import socket
from machine import DAC, Pin
import time

# Conectando-se à rede Wi-Fi
ssid = 'Alencar Oi 2.4G'
password = 'Carloshenrique'

station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)

while not station.isconnected():
    pass

print('Conectado à rede Wi-Fi')
print(station.ifconfig())

# Inicializa DAC no pino GPIO25
dac = DAC(Pin(25))
```

```
# Saída digital para controle liga/desliga (GPIO27)
saida = Pin(27, Pin.OUT)
estado_saida = False # Desligado inicialmente
saida.value(0)
```

```
# Função que mapeia frequência (0-120 Hz) para valor DAC (0-255)
def freq_to_dac(freq):
    return int((freq / 120) * 255)
```

```
# Página HTML com slider, botão e indicador
def web_page(freq, ligado):
    cor = "green" if ligado else "red"
    texto = "LIGADO" if ligado else "DESLIGADO"
    return f'<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Controle de Inversor</title>
```

```
<style>
```

```
body {{
```

```
    display: flex;
```

```
    justify-content: center;
```

```
    align-items: center;
```

```
    height: 100vh;
```

```
    margin: 0;
```

```
    background-color: #f4f4f4;
```

```
    font-family: Arial, sans-serif;
```

```
}}
```

```
.container {{
```

```
    width: 40%;
```

```
    padding: 30px;
```

```
    background-color: #ffffff;
```

```
    border: 2px solid #ccc;
```

```
    border-radius: 10px;
```

```
    text-align: center;
```

```
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
```

```
}}
```

```
input[type=range] {{
```

```
    width: 100%;
```

```
}}
```

```
h2 {{
```

```
    margin-top: 0;
```

```
}}
```

```
.botao {{
```

```
    padding: 10px 20px;
```

```
    font-size: 16px;
```

```
    margin-top: 20px;
```

```
    cursor: pointer;
```

```
}}
```

```
.indicador {{
```

```
    margin: 10px auto;
```

```
    width: 20px;
```

```
    height: 20px;
```

```
    border-radius: 50%;
```

```
    background-color: {cor};
```

```
    }}
</style>
</head>
<body>
  <div class="container">
    <h2>Controle de Frequência do Inversor</h2>
    <p>Frequência: <span id="freq_val">{freq}</span> Hz</p>
    <input type="range" min="0" max="120" value="{freq}" id="freq_slider" oninput="updateFreq(this.value)">

    <br><br>
    <button class="botao" onclick="toggleSaida()">Ligar/Desligar</button>
    <div class="indicador" id="estado_indicador"></div>
    <p>{texto}</p>
  </div>

  <script>
    function updateFreq(val) {{
      document.getElementById("freq_val").innerText = val;
      fetch("/?freq=" + val);
    }}

    function toggleSaida() {{
      fetch("/toggle").then(() => {{
        location.reload();
      }});
    }}
  </script>
</body>
</html>"""
```

```
# Cria socket do servidor
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]
s = socket.socket()
s.bind(addr)
s.listen(1)

print('Servidor rodando em: http://%s' % station.ifconfig()[0])

# Loop principal do servidor
freq = 0

while True:
  conn, addr = s.accept()
  print('Conexão de:', addr)
  request = conn.recv(1024).decode()

  # Atualiza frequência se necessário
  if 'GET /?freq=' in request:
    try:
      i = request.find('/?freq=')
      j = request.find(' ', i)
      freq_val = request[i+7:j]
      freq = int(freq_val)
      dac_value = freq_to_dac(freq)
      dac.write(dac_value)
```

```
print(f"Frequência: {freq} Hz | Valor DAC: {dac_value}")
except:
    pass

# Alterna estado da saída digital
elif 'GET /toggle' in request:
    estado_saida = not estado_saida
    saida.value(1 if estado_saida else 0)
    print("Saída:", "Ligada" if estado_saida else "Desligada")

# Envia a página atualizada
response = web_page(freq, estado_saida)
conn.send('HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n')
conn.sendall(response)
conn.close()
```

5. Execução do servidor web:

- 5.1 Configure o ESP32 para conectar-se a uma rede Wi-Fi local.
- 5.2 Observe o endereço IP do ESP32 exibido no terminal.
- 5.3 Acesse esse endereço em um navegador web na mesma rede.
- 5.4 Teste o botão e verifique se o relé liga e desliga o inversor.
- 5.5 Teste o controle deslizante e verifique se a tensão muda no pino DAC, variando a frequência do inversor.

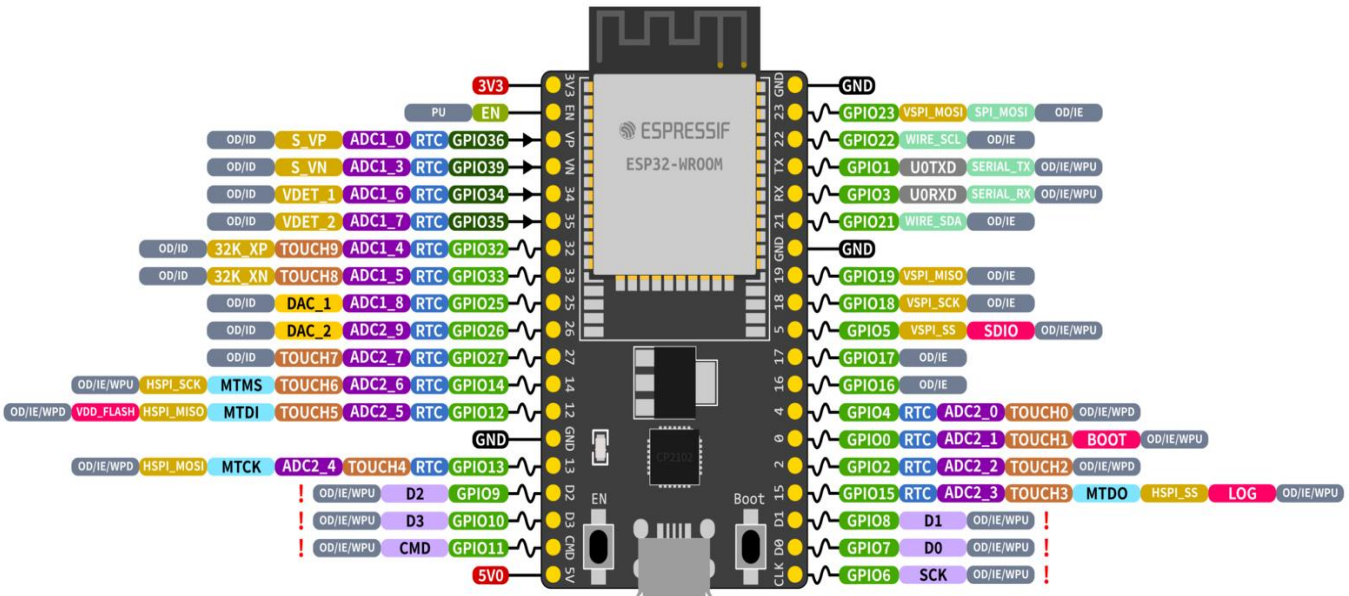
6. Exercícios Propostos:

- 1. **(Análise de Código)**
Explique como o valor do slider foi capturado pela interface HTML e enviado ao ESP32.
- 2. **(Mapeamento de Sinal)**
Sabendo que o DAC varia de 0 a 255 (8 bits), qual valor digital será enviado para 60 Hz? Qual será a tensão correspondente?
- 3. **(Modificação)**
Altere o código para que a frequência vá de 0 a 200 Hz em vez de 0 a 120 Hz.
- 4. **(Validação com Instrumentação)**
Meça a tensão na saída DAC com o multímetro para 0 Hz, 60 Hz e 120 Hz. Compare os valores obtidos com os esperados.
- 5. **(Expansão do Projeto)**
Sugira uma aplicação real para esse tipo de controle (por exemplo: bomba d'água, ventilador industrial etc.) e descreva o funcionamento.

Bibliografia

- ASCHER, D.; LUTZ, M. Aprendendo Python. Porto Alegre: Bookman, 2007.
- MENEZES, N. N. C. Introdução à programação com Python. São Paulo: Ed. Novatec, 2014.
- OLIVEIRA, S. Internet das coisas com ESP8266, Arduino e Raspberry Pi. São Paulo: Ed. Novatec, 2017.
- OLIVEIRA, C. L. V. IoT com Micropython e NodeMCU. São Paulo: Ed. Novatec, 2022.

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
Bluetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

