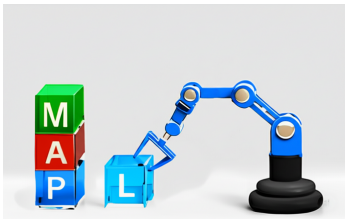
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA
FEMEC-33101 INTRODUÇÃO A ENGENHARIA MECATRÔNICA



IIoT – Internet das Coisas Industriais

Álisson Carvalho Vasconcelos

José Jean Paul Zanlucchi de Souza Tavares

2025

Sumário

1 Introdução à IoT e MQTT **3**
- 1.1 O que é IoT? . . . 3
- 1.2 O que é MQTT? . . . 3
  - 1.2.1 Arquitetura Publish/Subscribe: . . . 3
  - 1.2.2 Componentes Principais: . . . 4
  - 1.2.3 Qualidade de Serviço (QoS): . . . 4

2 ESP32 **4**
- 2.1 Introdução ao ESP32 . . . 4
- 2.2 Características Técnicas Detalhadas: . . . 5
- 2.3 Pinagem do ESP32 DevKit V1: . . . 5
- 2.4 Modelos e Variantes: . . . 6

3 Preparação do Ambiente **6**
- 3.1 Instalação do Driver CP210x (Windows) . . . 6
- 3.2 Instalação do ESP32 no Arduino IDE . . . 8
- 3.3 Instalação das Bibliotecas . . . 10

4 Programação Básica com Arduino IDE **11**
- 4.1 Estrutura Fundamental de um Sketch . . . 11
- 4.2 Controle Avançado de GPIO . . . 13
- 4.3 Comunicação Serial Avançada . . . 13
- 4.4 Controle de Tempo Profissional . . . 14
- 4.5 Exemplo Completo: Sistema de Monitoramento . . . 15
- 4.6 Técnicas Avançadas de Debug . . . 17

5 Comunicação MQTT com PubSubClient usando EWiFi **17**
- 5.1 Visão Geral do Código . . . 17
- 5.2 Estrutura do Código . . . 18
  - 5.2.1 Parte 1: Inclusão de Bibliotecas e Definições . . . 18
  - 5.2.2 Parte 2: Declaração de Objetos . . . 18
  - 5.2.3 Parte 3: Função de Callback MQTT . . . 18
  - 5.2.4 Parte 4: Função de Reconexão MQTT . . . 19
  - 5.2.5 Parte 5: Setup - Configuração Inicial . . . 20
  - 5.2.6 Parte 6: Loop Principal . . . 21
- 5.3 Arquivo WiFiPassword.h . . . 21
- 5.4 Fluxo de Execução do Programa . . . 22
- 5.5 Possíveis Erros e Soluções . . . 22

6 Exercícios Propostos **22**
- 6.1 Exercício 1: Controle de LED com Botão Remoto . . . 22
- 6.2 Exercício 2: Controle de Acesso por Tag RFID . . . 23

7 Referências **25**
7.1 Documentação Oficial 25
7.2 Tutoriais e Guias 25
7.3 Ferramentas e Bibliotecas 25

1 Introdução à IoT e MQTT

1.1 O que é IoT?

A Internet das Coisas (IoT) representa uma revolução tecnológica onde objetos físicos do cotidiano estão conectados à internet, coletando e trocando dados em tempo real. Na engenharia mecatrônica, a IoT permite:

- **Monitoramento Remoto**: Sensores coletam dados de máquinas e processos
- **Controle à Distância**: Atuadores podem ser controlados de qualquer lugar
- **Integração de Sistemas**: Conexão entre componentes mecânicos, eletrônicos e software
- **Automação Inteligente**: Sistemas que aprendem e se adaptam automaticamente

Aplicações Práticas em Mecatrônica:

- Linhas de produção industrial com manutenção preditiva
- Robôs colaborativos conectados em tempo real
- Sistemas de segurança monitorados via smartphone
- Dispositivos médicos para telemedicina
- Veículos autônomos e sistemas de transporte inteligente

1.2 O que é MQTT?

MQTT (Message Queuing Telemetry Transport) é um protocolo de comunicação leve e eficiente, especialmente projetado para dispositivos com recursos limitados. Desenvolvido inicialmente pela IBM em 1999, tornou-se padrão OASIS e é amplamente utilizado em aplicações IoT.

1.2.1 Arquitetura Publish/Subscribe:

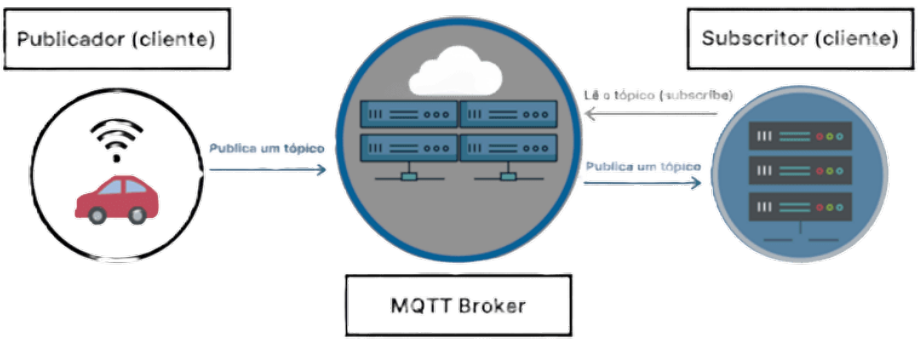
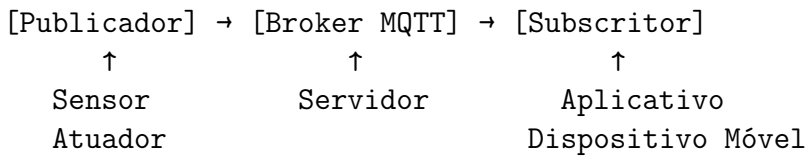


Figura 1: Arquitetura MQTT - Publicador/Subscritor adquirido de `https://pt.m.wikipedia.org/wiki/Ficheiro:Arquitetura_MQTT_exemplo.png`



1.2.2 Componentes Principais:

- **Broker**: Servidor intermediário que gerencia e distribui mensagens
- **Tópicos**: Canais temáticos para organização das mensagens (ex: "casa/sala/temperatura")
- **Publicadores**: Dispositivos que enviam mensagens para tópicos
- **Subscritores**: Dispositivos que recebem mensagens de tópicos

1.2.3 Qualidade de Serviço (QoS):

- **QoS 0 - At most once**: Entrega rápida, sem confirmação (ideal para dados não críticos)
- **QoS 1 - At least once**: Entrega garantida, pode ter duplicatas (balance entre velocidade e confiabilidade)
- **QoS 2 - Exactly once**: Entrega exata uma vez, lenta mas totalmente confiável (para dados críticos)

Vantagens do MQTT para Mecatrônica:

- Baixo consumo de energia e recursos
- Eficiente em redes com largura de banda limitada
- Tolerante a conexões instáveis
- Fácil implementação em microcontroladores
- Ideal para comunicação Machine-to-Machine (M2M)

2 ESP32

2.1 Introdução ao ESP32

O ESP32 é um microcontrolador revolucionário que combina alta performance com baixo custo, tornando-se a escolha preferida para projetos de IoT e aplicações mecatrônicas.

2.2 Características Técnicas Detalhadas:

Especificação	Detalhes	Aplicação em Mecatrônica
Processador	Dual-core Xtensa LX6 até 240MHz	Processamento paralelo para controle em tempo real
Memória	520KB RAM, 4-16MB Flash	Armazenamento de programas e dados de sensores
Wireless	Wi-Fi 802.11 b/g/n, Bluetooth 4.2	Conexão sem fio para monitoramento remoto
GPIO	34 pinos programáveis	Interface com sensores e atuadores
Interfaces	SPI, I2C, UART, I2S, PWM, ADC	Comunicação com periféricos diversos
Tensão	3.3V operação	Compatível com maioria dos sensores

Tabela 1: Características técnicas do ESP32

2.3 Pinagem do ESP32 DevKit V1:

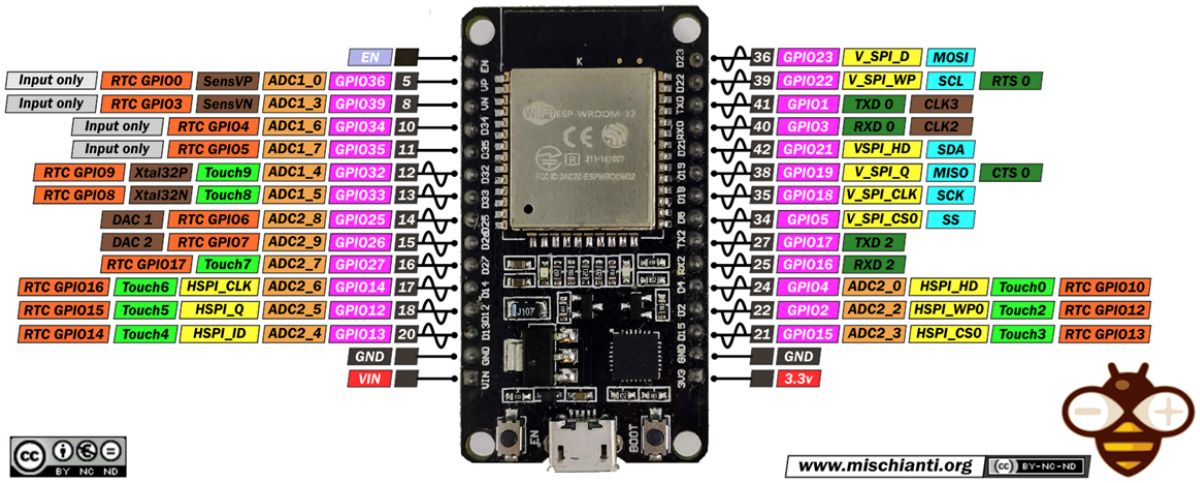


Figura 2: Pinagem ESP-WROOM-32 DevKit V1 adquirido de `https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/`

Pinos Digitais de Saída (Output):

- GPIO 23, 22, 1, 3, 21, 19, 18
- LED interno: GPIO 2 (ideal para testes e debug)

Pinos de Entrada Apenas (Input Only):

- GPIO 36 (VP), 39 (VN), 34, 35
- Ideais para leitura de sensores analógicos

Pinos Especiais:

- **GPIO 1 (TX0) e 3 (RX0)**: Comunicação serial UART
- **GPIO 21 (SDA) e 22 (SCL)**: Interface I2C para sensores
- **GPIO 18 (SCK), 19 (MISO), 23 (MOSI)**: Comunicação SPI
- **GPIO 2, 4, 12-15, 25-27, 32-33**: Saídas PWM para controle de motores

2.4 Modelos e Variantes:

ESP32-WROOM-32:

- Módulo base com antena PCB integrada
- 4MB flash, ideal para produtos finais
- Temperatura operacional: -40°C to 85°C

ESP32-DevKitC:

- Placa de desenvolvimento para prototipagem
- Pinagem exposta, conexão USB para programação
- Cristal de 40MHz, regulador de tensão

ESP32-PICO-D4:

- Design ultra-compacto (7mm × 7mm)
- Tudo em um chip - ideal para espaços reduzidos
- 4MB flash embutido

ESP32-S:

- Versões com recursos de segurança avançados
- Ideal para aplicações industriais críticas

3 Preparação do Ambiente

3.1 Instalação do Driver CP210x (Windows)

Passo a Passo Detalhado:
1. **Download do Driver**:

- Acesse: `https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads`
- Baixe a versão mais recente do driver CP210x

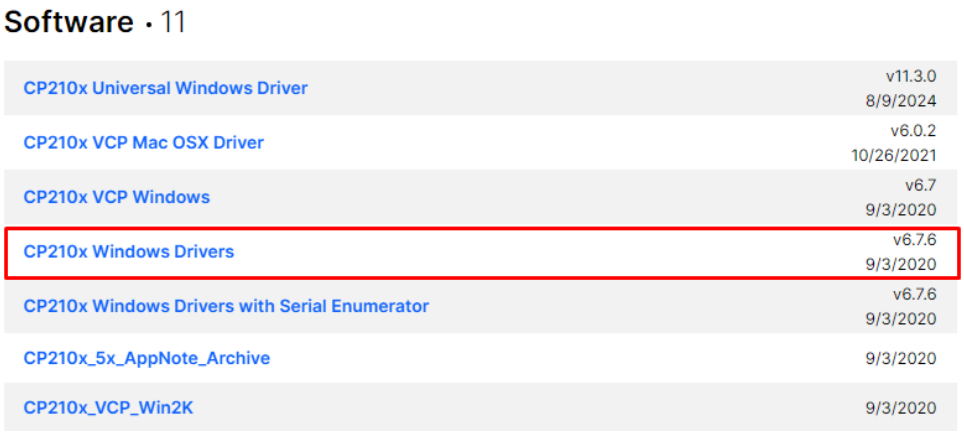


Figura 3: Drive Universal para Chips CP210x adquirido de `https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads`

2. **Instalação**:

```
# Para Windows 64-bit
Execute: CP210x_Windows_Drivers_x64.exe

# Para Windows 32-bit
Execute: CP210x_Windows_Drivers_x86.exe

# Siga o assistente de instalação
# Reinicie o computador se solicitado
```

3. **Verificação da Instalação**:
   - Conecte o ESP32 via cabo USB
   - Abra Gerenciador de Dispositivos:
   - Tecla Windows + R
   - Digite: `devmgmt.msc`
   - Enter
   - Verifique em "Portas (COM e LPT)"→ "Silicon Labs CP210x USB to UART Bridge (COMX)"

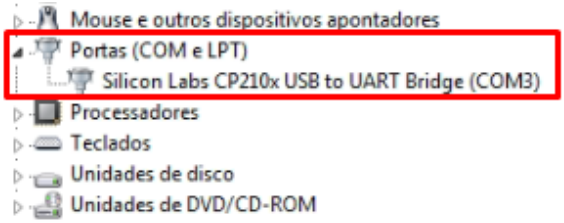


Figura 4: Porta do Chip CP210x USB instalada e disponível na COM3

4. **Solução de Problemas Comuns**:

- Se não aparecer: tente outra porta USB
- Driver não assinado: pressione "Instalar mesmo assim"
- No Linux: drivers geralmente incluídos no kernel

3.2 Instalação do ESP32 no Arduino IDE

Configuração Passo a Passo:
1. **Abrir Arduino IDE**:

- Versão 2.x recomendada
- Disponível em: `https://www.arduino.cc/en/software`

Figura 5: Página de Download do Arduino adquirido de `https://www.arduino.cc/en/software`

2. **Configurar URLs Adicionais**:

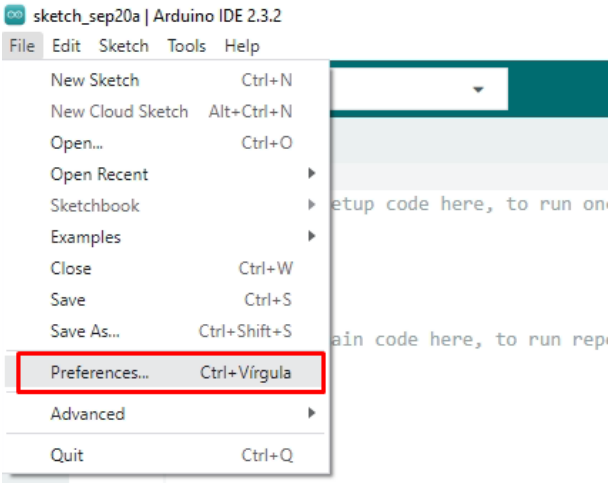


Figura 6: Acesso as preferências do Arduino IDE

- **Arquivo → Preferências**

- No campo "URLs Adicionais para Gerenciadores de Placas", adicione:

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
https://dl.espressif.com/dl/package_esp32_index.json
https://espressif.github.io/arduino-esp32/package_esp32_index.json
```

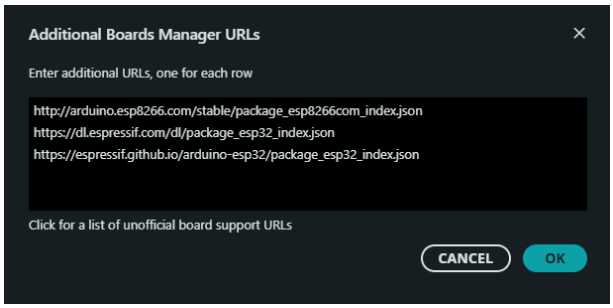


Figura 7: Janela de inserção de URLs de gerenciador de placas

- Clique em "OK"

3. **Instalação da Placa**:

   - **Ferramentas → Placa → Gerenciador de Placas** ou **Barra Lateral → Icone de placas**
   - Pesquise "ESP32"
   - Instale "ESP32 by Espressif Systems"(versão mais recente)
   - Aguarde o download e instalação (pode demorar alguns minutos)
   - Repita o mesmo processo para instalar "esp8266 by ESP8266 Community"(versão mais recente)

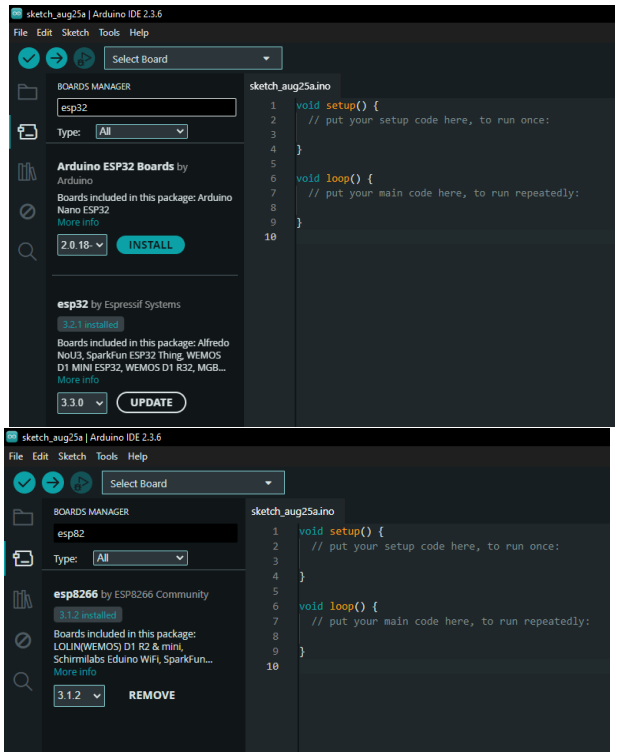


Figura 8: Instalações das placas ESP32 e ESP8266 no Gerenciador de Placa do Arduino IDE

4. **Seleção da Placa**:
   - **Ferramentas → Placa → ESP32 Arduino**
   - Selecione: "DOIT ESP32 DEVKIT V1"
5. **Configurações Adicionais**:
   - **Ferramentas → Porta**: Selecione a porta COM do ESP32
   - **Ferramentas → Upload Speed**: 921600 (para upload mais rápido)
   - **Ferramentas → Core Debug Level**: None (para melhor performance)

3.3 Instalação das Bibliotecas

Bibliotecas Essenciais via Library Manager:

```
// No Arduino IDE:
Sketch → Include Library → Manage Libraries ou Barra Lateral → Icone
de Livraria

// Biblioteca para instalar:
1. PubSubClient by Nick O'Leary (versão 2.8+)
```

Instalação de Bibliotecas Personalizadas:
1. **Criar Pasta de Bibliotecas**:

- Windows: `C:\Users\SeuUsuario\Documents\Arduino\libraries\`
- Linux: `~/Arduino/libraries/`

2. **Instalar LiquidCrystal**:

```
# Extrair o arquivo ZIP (LiquidCrystal.zip)
# Direcionar para o caminho .../Arduino/libraries/
```

3. **Instalar EWiFi**:

```
# Extrair o arquivo ZIP (EWiFi.zip)
# Direcionar para o caminho .../Arduino/libraries/
```

4. **Instalar CLPIoT**:

```
# Extrair o arquivo ZIP (CLPIoT.zip)
# Direcionar para o caminho .../Arduino/libraries/
```

5. **Instalar PN532**:

```
# Extrair o arquivo ZIP (PN532.zip)
# Direcionar para o caminho .../Arduino/libraries/
```

Verificação da Instalação:

- Reinicie o Arduino IDE
- **Sketch → Include Library**: Deve aparecer "LiquidCrystalEWiFi", "CLPIoT"e "PN532"lib na lista
- Compile um sketch vazio para testar se não há erros

4 Programação Básica com Arduino IDE

4.1 Estrutura Fundamental de um Sketch

```
// ================================================
// SECAO 1: DEFINICOES E INCLUDES
// ================================================
#include <Arduino.h>          // Biblioteca principal
#define LED_PIN 2             // Define pino do LED
#define BOTAO_PIN 0           // Define pino do botao

// Constantes e defines
const int TEMPO_PISCA = 500;  // 500ms
const float VERSION = 1.0;    // Versao do firmware

// ================================================
// SECAO 2: VARIAVEIS GLOBAIS
```

```
// ============================================
int contador = 0;              // Contador global
unsigned long ultimoTempo = 0; // Para timing

// ============================================
// SECAO 3: CONFIGURACAO INICIAL (SETUP)
// ============================================
void setup() {
  // Inicializa comunicacao serial
  Serial.begin(115200);
  Serial.println("Iniciando Sistema Mecatronico...");

  // Configura pinos
  pinMode(LED_PIN, OUTPUT);
  pinMode(BOTAO_PIN, INPUT_PULLUP);

  // LED inicial apagado
  digitalWrite(LED_PIN, LOW);

  // Mensagem de inicializacao completa
  Serial.println("Sistema pronto!");
  Serial.print("Versao: ");
  Serial.println(VERSION);
}

// ============================================
// SECAO 4: LOOP PRINCIPAL
// ============================================
void loop() {
  // Controle do LED com timing nao-bloqueante
  unsigned long tempoAtual = millis();

  if (tempoAtual - ultimoTempo >= TEMPO_PISCA) {
    ultimoTempo = tempoAtual;

    // Alterna estado do LED
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    contador++;

    Serial.print("Piscadas: ");
    Serial.println(contador);
  }

  // Leitura do botao com debounce
  lerBotao();

  // Pequeno delay para nao sobrecarregar
  delay(10);
}

// ============================================
// SECAO 5: FUNCOES PERSONALIZADAS
// ============================================
void lerBotao() {
  static unsigned long ultimoDebounce = 0;
  const int DEBOUNCE_DELAY = 50;

  if (millis() - ultimoDebounce > DEBOUNCE_DELAY) {
```

```
    if (digitalRead(BOTAO_PIN) == LOW) {
      Serial.println("Botao pressionado!");
      // Aqui viria a acao do botao
    }
    ultimoDebounce = millis();
  }
}
```

Listing 1: Estrutura básica de um sketch Arduino

4.2 Controle Avançado de GPIO

Configuração de Pinagem:

```
// Configuracoes basicas
pinMode(PINO, INPUT);           // Entrada simples
pinMode(PINO, OUTPUT);          // Saida digital
pinMode(PINO, INPUT_PULLUP);    // Entrada com pull-up interno
pinMode(PINO, INPUT_PULLDOWN);  // Entrada com pull-down

// Para ESP32 especifico
pinMode(PINO, INPUT_PULLUP);    // Usar sempre que possivel
```

Controle Digital:

```
// Leitura digital
int estado = digitalRead(PINO);
bool estadoBool = (digitalRead(PINO) == HIGH);

// Escrita digital
digitalWrite(PINO, HIGH);     // 3.3V - Ligado
digitalWrite(PINO, LOW);      // 0V - Desligado

// Toggle (alternancia)
digitalWrite(PINO, !digitalRead(PINO));
```

Controle Analógico:

```
// ESP32 tem ADC de 12 bits (0-4095)
int valorRaw = analogRead(PINO);

// Converter para tensao (0-3.3V)
float tensao = (valorRaw / 4095.0) * 3.3;

// Converter para porcentagem
int porcentagem = map(valorRaw, 0, 4095, 0, 100);

// PWM output (0-255)
analogWrite(PINO, 128); // 50% duty cycle
```

4.3 Comunicação Serial Avançada

Configuração e Métodos Úteis:

```
void setup() {
  // Inicializa serial com baud rate
  Serial.begin(115200);

  // Aguarda conexao serial (para debugging)
```

```
  while (!Serial) {
    delay(10);
  }
}

void enviarDados() {
  // Diferentes metodos de envio
  Serial.println("Mensagem completa"); // Com quebra de linha
  Serial.print("Valor: ");             // Sem quebra
  Serial.println(42);                 // Numero com quebra

  // Formatacao avancada
  Serial.printf("Temperatura: %.2f C\n", 23.45);
  Serial.printf("Umidade: %d%%\n", 65);
}

void lerComandos() {
  if (Serial.available() > 0) {
    String comando = Serial.readString();
    comando.trim(); // Remove espacos extras

    if (comando == "LED_ON") {
      digitalWrite(LED_PIN, HIGH);
      Serial.println("LED ligado");
    }
    else if (comando == "LED_OFF") {
      digitalWrite(LED_PIN, LOW);
      Serial.println("LED desligado");
    }
  }
}
```

4.4 Controle de Tempo Profissional

Evitando delay():

```
// NAO FAZER ISSO (bloqueante):
void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);           // Congela todo o sistema
  digitalWrite(LED_PIN, LOW);
  delay(1000);           // Nada mais funciona
}

// FAZER ASSIM (nao-bloqueante):
unsigned long previousMillis = 0;
const long interval = 1000; // 1 segundo

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    // Executa a cada 1 segundo
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
  }

```

```
  // Outras tarefas executam livremente aqui
  lerSensores();
  processarDados();
  verificarConexao();
}
```

Múltiplos Timers:

```
// Estrutura para multiplos timers
struct Timer {
  unsigned long previous;
  long interval;
  bool enabled;
};

Timer ledTimer = {0, 1000, true};
Timer sensorTimer = {0, 2000, true};
Timer commTimer = {0, 5000, true};

void checkTimer(Timer &timer, void (*function)()) {
  if (timer.enabled && millis() - timer.previous >= timer.interval) {
    timer.previous = millis();
    function();
  }
}

void piscaLED() { digitalWrite(LED_PIN, !digitalRead(LED_PIN)); }
void leSensor() { /* ler sensor */ }
void verificaConexao() { /* verificar conexao */ }

void loop() {
  checkTimer(ledTimer, piscaLED);
  checkTimer(sensorTimer, leSensor);
  checkTimer(commTimer, verificaConexao);
}
```

4.5 Exemplo Completo: Sistema de Monitoramento

```
#include <Arduino.h>

// Definicoes de pinos
#define LED_PIN 2
#define BOTAO_PIN 0
#define SENSOR_A0 36

// Variaveis globais
int contador = 0;
float temperatura = 0;
bool sistemaAtivo = true;

void setup() {
  Serial.begin(115200);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BOTAO_PIN, INPUT_PULLUP);

  Serial.println("=== SISTEMA MECATRONICO IoT ===");
  Serial.println("Comandos: START, STOP, STATUS");
```

```
}

void loop() {
  // Timing nao-bloqueante
  static unsigned long lastRead = 0;
  if (millis() - lastRead >= 1000) {
    lastRead = millis();
    if (sistemaAtivo) {
      lerSensores();
      enviarDados();
    }
  }

  // Verificar comandos serial
  verificarComandos();

  // Verificar botao
  verificarBotao();

  delay(10); // Pequeno delay
}

void lerSensores() {
  int raw = analogRead(SENSOR_A0);
  temperatura = (raw / 4095.0) * 100; // Simulacao
  contador++;
}

void enviarDados() {
  Serial.printf("Leitura #%d: %.2f C\n", contador, temperatura);
}

void verificarComandos() {
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();

    if (cmd == "START") {
      sistemaAtivo = true;
      Serial.println("Sistema ATIVADO");
    }
    else if (cmd == "STOP") {
      sistemaAtivo = false;
      Serial.println("Sistema DESATIVADO");
    }
    else if (cmd == "STATUS") {
      Serial.printf("Status: %s\n", sistemaAtivo ? "ATIVO" : "INATIVO");
      Serial.printf("Temperatura: %.2f C\n", temperatura);
    }
  }
}

void verificarBotao() {
  static unsigned long lastPress = 0;
  if (digitalRead(BOTAO_PIN) == LOW && millis() - lastPress > 200) {
    lastPress = millis();
    sistemaAtivo = !sistemaAtivo;
    Serial.printf("Botao pressionado! Sistema: %s\n",
```

```
                    sistemaAtivo ? "ATIVO" : "INATIVO");
  }
}
```

Listing 2: Exemplo completo de sistema de monitoramento

4.6 Técnicas Avançadas de Debug

Sistema de Logging:

```
enum LogLevel { DEBUG, INFO, WARNING, ERROR };

void log(LogLevel level, const char* message) {
  const char* levels[] = {"DEBUG", "INFO", "WARNING", "ERROR"};
  Serial.printf("[%s] %s\n", levels[level], message);

  // LED pisca diferente conforme nivel
  if (level == ERROR) {
    for(int i=0; i<3; i++) {
      digitalWrite(LED_PIN, HIGH); delay(100);
      digitalWrite(LED_PIN, LOW); delay(100);
    }
  }
}

// Uso:
log(INFO, "Sistema iniciado");
log(DEBUG, "Temperatura lida: 23.5C");
log(ERROR, "Sensor nao respondendo!");
```

Monitoramento de Performance:

```
void medirPerformance() {
  static unsigned long loopCount = 0;
  static unsigned long lastReport = 0;

  loopCount++;

  if (millis() - lastReport >= 5000) {
    lastReport = millis();
    Serial.printf("Loops por segundo: %.2f\n",
                  loopCount / 5.0);
    loopCount = 0;
  }
}
```

5 Comunicação MQTT com PubSubClient usando EWiFi

5.1 Visão Geral do Código

Este código demonstra como conectar um ESP32 a uma rede WiFi (doméstica ou enterprise) usando a biblioteca EWiFi e estabelecer comunicação MQTT com um broker para controle remoto de dispositivos.

5.2 Estrutura do Código

5.2.1 Parte 1: Inclusão de Bibliotecas e Definições

```
#include <EWiFi.h>         // Biblioteca personalizada para WiFi
#include <PubSubClient.h> // Biblioteca para MQTT
#include <WiFiPassword.h> // Arquivo com credenciais

// Configuracoes MQTT
const char* mqtt_server = "test.mosquitto.org";
const int mqtt_port = 1883;

// Topicos MQTT
const char* topic_subscribe = "device/led";
const char* topic_publish = "device/status";
```

Explicação:

- `EWiFi.h`: Biblioteca personalizada que simplifica a conexão WiFi
- `PubSubClient.h`: Biblioteca oficial para comunicação MQTT
- `WiFiPassword.h`: Arquivo header com credenciais (mantido separado por segurança)
- `mqtt_server`: URL do broker MQTT público para testes
- `topic_subscribe`: Tópico para receber comandos
- `topic_publish`: Tópico para enviar status

5.2.2 Parte 2: Declaração de Objetos

```
WiFiClient espClient;
PubSubClient client(espClient);
```

Explicação:

- `WiFiClient`: Objeto para gerenciar conexão WiFi
- `PubSubClient`: Objeto para comunicação MQTT, usando o cliente WiFi como base

5.2.3 Parte 3: Função de Callback MQTT

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Mensagem recebida no topico: ");
  Serial.print(topic);
  Serial.print(" - Mensagem: ");

  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  Serial.println(message);

  if (String(topic) == topic_subscribe) {
```

```
    if (message == "ON") {
      digitalWrite(2, HIGH);
      client.publish(topic_publish, "LED LIGADO");
      Serial.println("LED ligado via MQTT");
    }
    else if (message == "OFF") {
      digitalWrite(2, LOW);
      client.publish(topic_publish, "LED DESLIGADO");
      Serial.println("LED desligado via MQTT");
    }
  }
}
```

Explicação Passo a Passo:

1. **Recebimento de Mensagem**: Esta função é chamada automaticamente quando chega uma mensagem MQTT
2. **Processamento do Payload**: Converte o array de bytes em String para facilitar manipulação
3. **Verificação do Tópico**: Confirma se a mensagem é para o tópico esperado
4. **Execução de Comando**:
   - Se mensagem = "ON"→ acende LED e publica confirmação
   - Se mensagem = "OFF"→ apaga LED e publica confirmação
5. **Feedback no Serial**: Exibe informações para debug

5.2.4 Parte 4: Função de Reconexão MQTT

```
void reconnect() {
  Serial.println("Conectando ao broker MQTT...");

  while (!client.connected()) {
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);

    if (client.connect(clientId.c_str())) {
      Serial.println("Conectado ao broker MQTT!");
      client.subscribe(topic_subscribe);
      Serial.print("Inscrito no topico: ");
      Serial.println(topic_subscribe);
      client.publish(topic_publish, "ESP32 Conectado");
    } else {
      Serial.print("Falha na conexao MQTT, rc=");
      Serial.print(client.state());
      Serial.println(" Tentando novamente em 5 segundos...");
      delay(5000);
    }
  }
}
```

Explicação Passo a Passo:

1. **Loop de Reconexão**: Tenta conectar repetidamente até conseguir

2. **ID Único do Cliente**: Gera um ID randomico para evitar conflitos
3. **Tentativa de Conexão**: Usa `client.connect()` para estabelecer conexão
4. **Subscribe no Tópico**: Uma vez conectado, se inscreve no tópico para receber mensagens
5. **Mensagem de Boas-Vindas**: Publica mensagem indicando que está online
6. **Tratamento de Erro**: Se falhar, exibe código do erro e tenta novamente após 5s

5.2.5 Parte 5: Setup - Configuração Inicial

```
void setup() {
  Serial.begin(115200);
  Serial.println("Iniciando ESP32...");

  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);

  Serial.println("Conectando ao WiFi...");

  if (password != "XXXX") {
    ewifi.setWiFi(SSID, password);
  } else {
    ewifi.setWiFi(SSID, WPA2_AUTH_PEAP, anonymous, username,
   userpassword);
  }

  if (ewifi.connect(30)) {
    Serial.println("Conectado ao WiFi!");
    Serial.print("IP: ");
    Serial.println(ewifi.getIP(WIFI_IPv4));
    Serial.print("MAC: ");
    Serial.println(ewifi.getmacAddress());
  } else {
    Serial.println("Falha na conexao WiFi!");
    while(1) {
      digitalWrite(2, HIGH);
      delay(100);
      digitalWrite(2, LOW);
      delay(100);
    }
  }

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  Serial.println("Configuracao inicial concluida!");
}
```

Explicação Passo a Passo:

1. **Inicialização Serial**: Configura comunicação serial para monitoramento (115200 baud rate)
2. **Configuração do LED**: Define pino 2 como saída e inicia desligado

3. **Seleção Automática de Rede**: Verifica se é rede doméstica ou enterprise
4. **Conexão WiFi**: Usa `ewifi.connect(30)` com timeout de 30 segundos
5. **Exibição de Informações**: Mostra IP e MAC address após conexão bem-sucedida
6. **Configuração MQTT**: Define servidor e função de callback
7. **Tratamento de Falha**: Pisca LED infinitamente em caso de falha WiFi

5.2.6 Parte 6: Loop Principal

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }

  client.loop();
  delay(100);
}
```

Explicação:

1. **Verificação de Conexão**: Chama `reconnect()` se desconectado do broker
2. **Processamento MQTT**: `client.loop()` mantém a comunicação ativa
3. **Delay Não-Bloqueante**: Pequeno delay para evitar sobrecarga do CPU

5.3 Arquivo WiFiPassword.h

```
/** Input WiFi data
 * Definition of macros SSID, username and password
*/
#define SSID "UFU-Institucional"
#define username "SEU_USUARIO@ufu.br"
#define userpassword "SUA_SENHA"
#define password "XXXX"  // Use "XXXX" para rede enterprise
#define anonymous "anonymous@ufu.br"

/** Input MQTT data
 * Definition of macros MQTT server and port
*/
#define MQTTServer "test.mosquitto.org"
#define MQTTPort 1883

/* DECLARATION OF TOPICS VARIABLES */
const char* topicLed = "device/led";
const char* topicStatus = "device/status";
```

Listing 3: Arquivo de configuração WiFiPassword.h

Instruções de Uso:

- **Para Rede Doméstica**: Altere `password` para sua senha real

- **Para Rede Enterprise**: Mantenha `password` como "XXXX"e preencha `username` e `userpassword`
- **Broker MQTT**: Pode ser alterado para um broker local ou cloud

5.4 Fluxo de Execução do Programa

1. **Inicialização** → Setup() configura hardware e conexões
2. **Conexão WiFi** → EWiFi conecta automaticamente conforme configuração
3. **Conexão MQTT** → Conecta ao broker e se inscreve nos tópicos
4. **Loop Principal** → Mantém conexões ativas e processa mensagens
5. **Callback** → Executa ações quando recebe mensagens MQTT

5.5 Possíveis Erros e Soluções

- **Falha na Conexão WiFi**:
  - Verifique credenciais no `WiFiPassword.h`
  - Confirme se a rede está disponível
- **Falha na Conexão MQTT**:
  - Verifique se o broker está online
  - Teste com broker público alternativo
- **Mensagens Não Chegam**:
  - Confirme se está publicando no tópico correto
  - Verifique QoS e retain flags

Este código oferece uma base robusta para projetos IoT com tratamento de erros e reconexão automática, ideal para aplicações mecatrônicas que requerem comunicação confiável.

6 Exercícios Propostos

6.1 Exercício 1: Controle de LED com Botão Remoto

Objetivo: Criar um sistema de comunicação utilizando o protocolo MQTT para que o aplicativo do Android (IntroEMT) controle um LED da CLPIoT Didática.
Requisitos:

- Android #1:
  - 1º Passo: Precionar botão "ABRIR SETUP";
  - 2º Passo: Preencher as informações do Broker:

        * Endereço (domínio ou IP);
        * Porta (padrão 1883);
        * Tópico de botão precionado ("GrupoX/led");
        * Tópico do status do LED ("GrupoX/status").
    - 3º Passo: Precionar botão "TESTAR BROKER", para testar a conexão com o Broker;
    - 4º Passo: Precionar botão "CONFIRMAR", para configurar os dados da comunicação MQTT;
    - 5º Passo: Precionar botão "EXECUTAR LED!", para entrar no contrele remoto do LED;

- ESP32 #2:

    - 1º Passo: Carregar os códigos "LED_ON_OFF.ino"e "WiFiPassword.h"no software Arduino IDE;
    - 2º Passo: Preencher as informações para conexão:
        * SSID: Informações do nome da rede domestica ou empresarial;
        * username: Informações do nome do usuário de uma rede empresarial;
        * userpassword: Informações da senha do usuário de uma rede empresarial;
        * password: Informações da senha de uma rede doméstica.

    Observação: Caso seja uma rede doméstica não é obrigatório o preenchimento dos dados empresariais, bem como, se for uma rede empresárial não precisa preencher o "password".

    - 3º Passo: Preencher as infromações do Broker:
        * MQTTServer: Informações do endereço do broker (dominío ou IP);
        * MQTTPort: Informações da porta do broker (padrão 1883);
        * topicLed: Informações do tópico de botão precionado ("GrupoX/led");
        * topicStatus: Informações do tópico do status do LED ("GrupoX/status").
    - 4º Passo: Precionar o botão de upload (seta) ou precionar as teclas "Ctrl+U", para carregar o código;

- Com ESP32 já funcinando, precionar a o botão "ON"/"OFF"ou a imagem do led para enviar mensagem via protocolo MQTT para que o ESP32 acione o led, e aguardar o retorno do ESP32 informando que o led foi ligado;

- Feedback visual com LED de status no ESP32 e no aplicativo IntroEMT.

6.2 Exercício 2: Controle de Acesso por Tag RFID

Objetivo: Criar um sistema de comunicação utilizando o protocolo MQTT para que o aplicativo do Android (IntroEMT) controle o acesso por meio da leitura de uma Tag RFID realizada pela CLPIoT Didática.
Requisitos:

- Android #1:

  - 1º Passo: Precionar botão "ABRIR SETUP";
  - 2º Passo: Preencher as informações do Broker:
    * Endereço (domínio ou IP);
    * Porta (padrão 1883);
    * Tópico de botão precionado ("GrupoX/led");
    * Tópico do status do LED ("GrupoX/status").
  - 3º Passo: Precionar botão "TESTAR BROKER", para testar a conexão com o Broker;
  - 4º Passo: Precionar botão "CONFIRMAR", para configurar os dados da comunicação MQTT;
  - 5º Passo: Precionar botão "EXECUTAR RFID!", para entrar no contrele de acesso;
  - 6º Passo: Precionar botão "Setup Tag Id";
  - 7º Passo: Preencher as infromações da Tag:
    * Tópico de recebimento do Tag ID ("GrupoX/tagId");
    * Chave de acesso.
  - 8º Passo: Precionar botão "CONFIRMAR", para configurar os dados de acesso;

- ESP32 #2:
  - 1º Passo: Carregar o código "Full_Tag_Format.ino"e executar para formatar a Tag para o formato NDEF.
  - 2º Passo: Após fechar o arquivo "Full_Tag_Format.ino", carregar os códigos "Access_Point.ino"e "WiFiPassword.h"no software Arduino IDE;
  - 3º Passo: Preencher as informações para conexão:
    * SSID: Informações do nome da rede domestica ou empresarial;
    * username: Informações do nome do usuário de uma rede empresarial;
    * userpassword: Informações da senha do usuário de uma rede empresarial;
    * password: Informações da senha de uma rede doméstica.

  Observação: Caso seja uma rede doméstica não é obrigatório o preenchimento dos dados empresariais, bem como, se for uma rede empresárial não precisa preencher o "password".

  - 4º Passo: Preencher as infromações do Broker:
    * MQTTServer: Informações do endereço do broker (dominío ou IP);
    * MQTTPort: Informações da porta do broker (padrão 1883);
    * topicLed: Informações do tópico de botão precionado ("GrupoX/led");
    * topicStatus: Informações do tópico do status do LED ("GrupoX/status")
    * topicTagId: Informações do tópico de recebimento do Tag ID ("GrupoX/-tagId").
  - 5º Passo: Precionar o botão de upload (seta) ou precionar as teclas "Ctrl+U", para carregar o código;

- Com ESP32 já funcinando, posicionar a Tag sobre o leitor PN532 instalado na placa CLPIoT Didática onde sera enviado mensagem via protocolo MQTT para o aplicativo IntroEMT onde ele verifica a chave inserida e retorna para o ESP32 para liberar ou não o acesso;
- Feedback visual com LED de status no ESP32 ligado e mensagem no LCD de acesso liberado, e no aplicativo IntroEMT a chave e a confirmação de acesso liberado.

7 Referências

7.1 Documentação Oficial

- **ESP32 Official Documentation**: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/`
  - Guia completo do fabricante
  - API references e examples
- **Arduino Core for ESP32**: `https://github.com/espressif/arduino-esp32`
  - Repositório oficial no GitHub
  - Issues e contribuições
- **MQTT Official Specification**: `https://mqtt.org/mqtt-specification/`
  - Especificação completa do protocolo
  - Melhores práticas

7.2 Tutoriais e Guias

- **Random Nerd Tutorials**: `https://randomnerdtutorials.com/`
  - Tutoriais ESP32 e MQTT
  - Projetos práticos passo a passo
- **ESP32.net**: `https://esp32.net/`
  - Compilação de recursos ESP32
  - Exemplos de código

7.3 Ferramentas e Bibliotecas

- **PubSubClient Library**: `https://github.com/knolleary/pubsubclient`
  - Documentação da biblioteca
  - Exemplos de uso
- **MQTT Explorer**: `http://mqtt-explorer.com/`
  - Cliente MQTT para teste
  - Monitoramento de tópicos