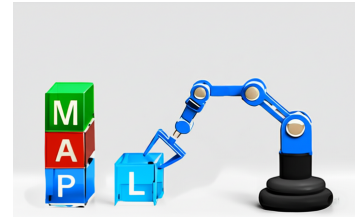




UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA
MECATRÔNICA
FEMEC-33101 INTRODUÇÃO A ENGENHARIA
MECATRÔNICA



IIoT – Internet das Coisas Industriais

Álison Carvalho Vasconcelos

José Jean Paul Zanlucchi de Souza Tavares

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	Desafio Extra: Sistema de Segurança com Múltiplos Comandos	23

7	Referências	23
7.1	Documentação Oficial	23
7.2	Tutoriais e Guias	23
7.3	Ferramentas e Bibliotecas	24

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:

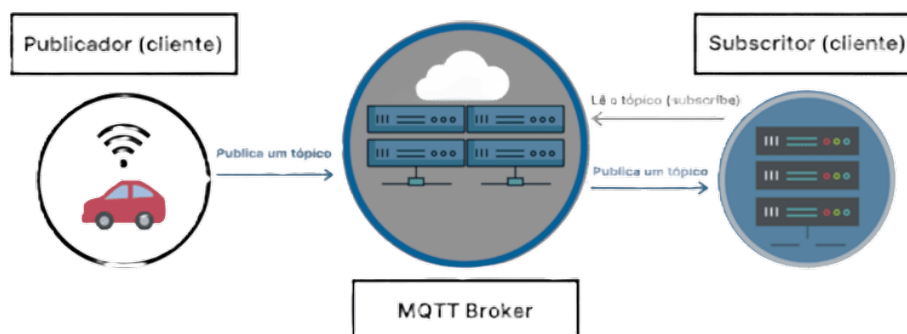
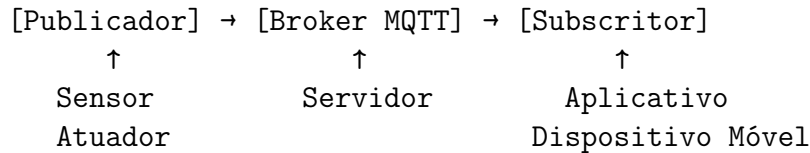


Figura 1: Arquitetura MQTT - Publicador/Subscriber adquirido de https://pt.m.wikipedia.org/wiki/Ficheiro:Arquitetura_MQTT_exemplo.png



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:

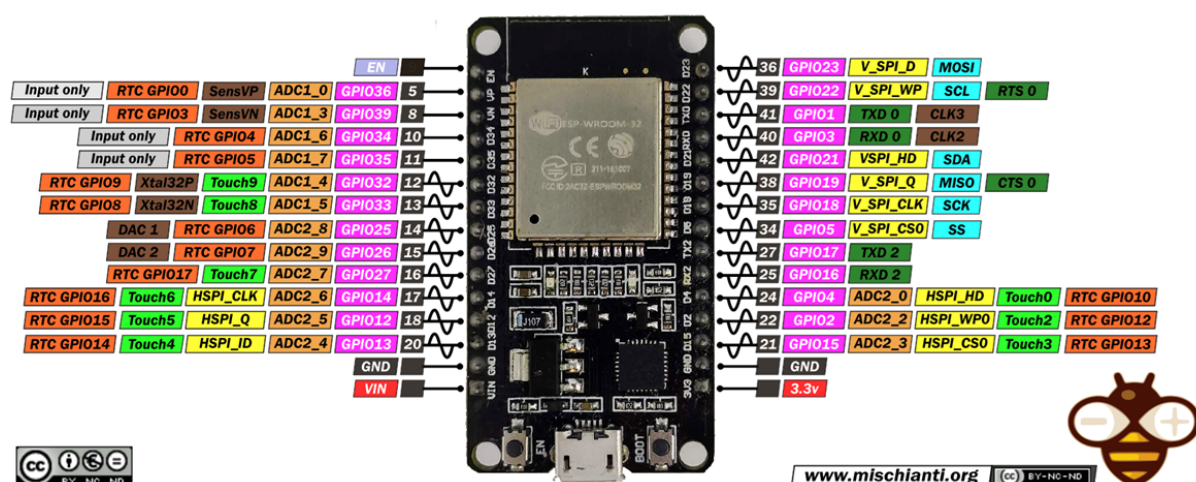


Figura 2: Pinagem ESP-WROOM-32 DevKit V1 adquirido de <https://mischianti.org/doit-esp32-dev-kit-v1-high-resolution-pinout-and-specs/>

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

Software • 11

CP210x Universal Windows Driver	v11.3.0 8/9/2024
CP210x VCP Mac OSX Driver	v6.0.2 10/26/2021
CP210x VCP Windows	v6.7 9/3/2020
CP210x Windows Drivers	v6.7.6 9/3/2020
CP210x Windows Drivers with Serial Enumerator	v6.7.6 9/3/2020
CP210x_5x_AppNote_Archive	9/3/2020
CP210x_VCP_Win2K	9/3/2020

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 (COM3)"

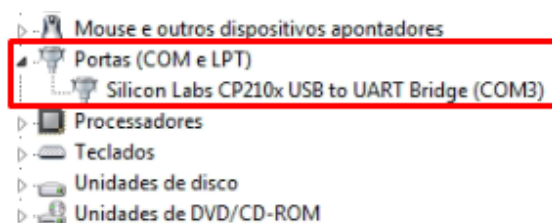


Figura 4: Porta do Chip CP210x USB instalada e disponível na COM3

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>

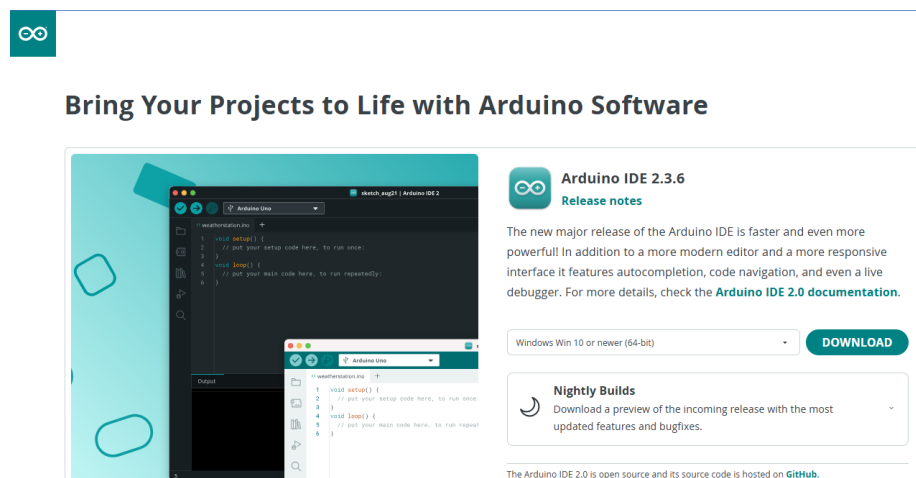


Figura 5: Página de Download do Arduino adquirido de <https://www.arduino.cc/en/software>

2. Configurar URLs Adicionais:

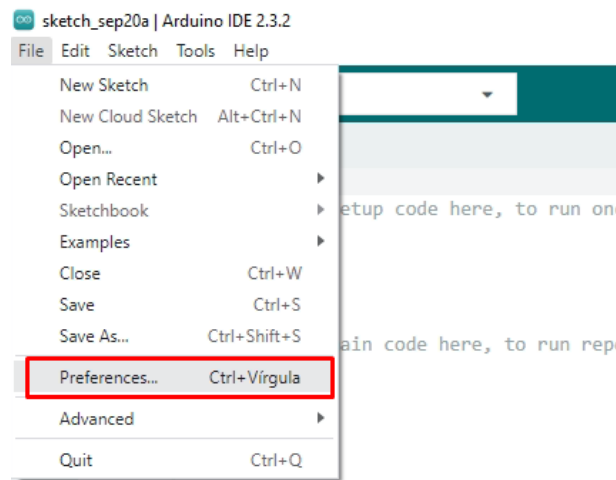


Figura 6: Acesso as preferências do Arduino IDE

- 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
```

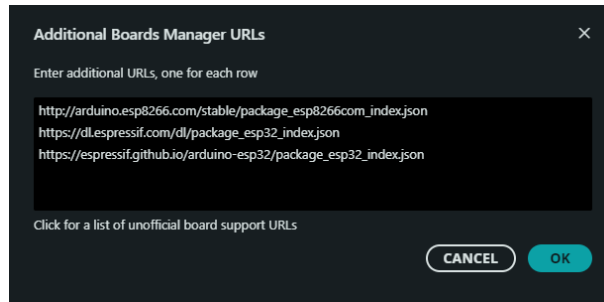


Figura 7: Janela de inserção de URLs de gerenciador de placas

- Clique em "OK"

3. Instalação da Placa:

- **Ferramentas → Placa → Gerenciador de Placas** ou **Barra Lateral → Ícone 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)

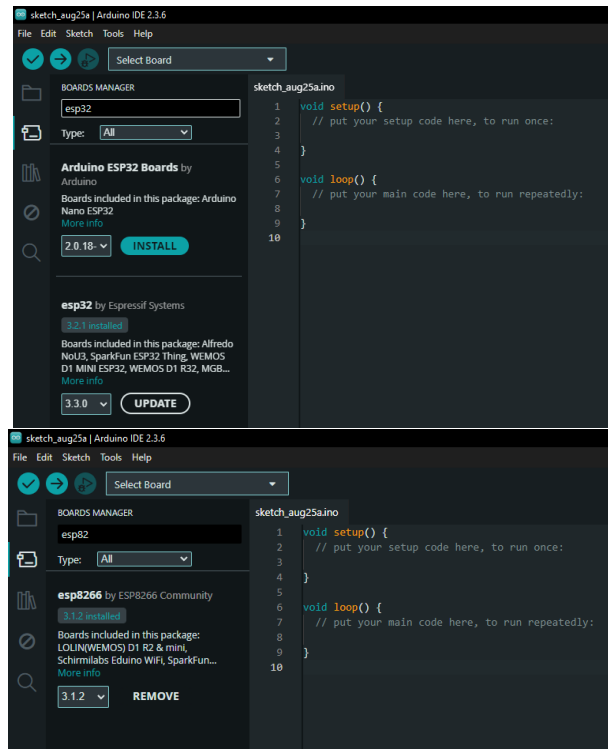


Figura 8: Instalações das placas ESP32 e ESP8266 no Gerenciador de Placa do Arduino IDE

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 → Ícone 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

```
1 // =====
2 // SECAO 1: DEFINICOES E INCLUDES
3 // =====
4 #include <Arduino.h>           // Biblioteca principal
5 #define LED_PIN 2             // Define pino do LED
6 #define BOTAO_PIN 0           // Define pino do botao
7
8 // Constantes e defines
9 const int TEMPO_PISCA = 500;  // 500ms
10 const float VERSION = 1.0;    // Versao do firmware
11
12 // =====
13 // SECAO 2: VARIAVEIS GLOBAIS
```

```

14 // =====
15 int contador = 0; // Contador global
16 unsigned long ultimoTempo = 0; // Para timing
17
18 // =====
19 // SECAO 3: CONFIGURACAO INICIAL (SETUP)
20 // =====
21 void setup() {
22     // Inicializa comunicacao serial
23     Serial.begin(115200);
24     Serial.println("Iniciando Sistema Mecatronico...");
25
26     // Configura pinos
27     pinMode(LED_PIN, OUTPUT);
28     pinMode(BOTAO_PIN, INPUT_PULLUP);
29
30     // LED inicial apagado
31     digitalWrite(LED_PIN, LOW);
32
33     // Mensagem de inicializacao completa
34     Serial.println("Sistema pronto!");
35     Serial.print("Versao: ");
36     Serial.println(VERSION);
37 }
38
39 // =====
40 // SECAO 4: LOOP PRINCIPAL
41 // =====
42 void loop() {
43     // Controle do LED com timing nao-bloqueante
44     unsigned long tempoAtual = millis();
45
46     if (tempoAtual - ultimoTempo >= TEMPO_PISCA) {
47         ultimoTempo = tempoAtual;
48
49         // Alterna estado do LED
50         digitalWrite(LED_PIN, !digitalRead(LED_PIN));
51         contador++;
52
53         Serial.print("Piscadas: ");
54         Serial.println(contador);
55     }
56
57     // Leitura do botao com debounce
58     lerBotao();
59
60     // Pequeno delay para nao sobrecarregar
61     delay(10);
62 }
63
64 // =====
65 // SECAO 5: FUNCOES PERSONALIZADAS
66 // =====
67 void lerBotao() {
68     static unsigned long ultimoDebounce = 0;
69     const int DEBOUNCE_DELAY = 50;
70
71     if (millis() - ultimoDebounce > DEBOUNCE_DELAY) {

```

```

72     if (digitalRead(BOTAO_PIN) == LOW) {
73         Serial.println("Botao pressionado!");
74         // Aqui viria a acao do botao
75     }
76     ultimoDebounce = millis();
77 }
78 }

```

Listing 1: Estrutura básica de um sketch Arduino

4.2 Controle Avançado de GPIO

Configuração de Pinagem:

```

1 // Configuracoes basicas
2 pinMode(PINO, INPUT);           // Entrada simples
3 pinMode(PINO, OUTPUT);          // Saida digital
4 pinMode(PINO, INPUT_PULLUP);    // Entrada com pull-up interno
5 pinMode(PINO, INPUT_PULLDOWN);  // Entrada com pull-down
6
7 // Para ESP32 especifico
8 pinMode(PINO, INPUT_PULLUP);    // Usar sempre que possivel

```

Controle Digital:

```

1 // Leitura digital
2 int estado = digitalRead(PINO);
3 bool estadoBool = (digitalRead(PINO) == HIGH);
4
5 // Escrita digital
6 digitalWrite(PINO, HIGH);      // 3.3V - Ligado
7 digitalWrite(PINO, LOW);       // 0V - Desligado
8
9 // Toggle (alternancia)
10 digitalWrite(PINO, !digitalRead(PINO));

```

Controle Analógico:

```

1 // ESP32 tem ADC de 12 bits (0-4095)
2 int valorRaw = analogRead(PINO);
3
4 // Converter para tensao (0-3.3V)
5 float tensao = (valorRaw / 4095.0) * 3.3;
6
7 // Converter para porcentagem
8 int porcentagem = map(valorRaw, 0, 4095, 0, 100);
9
10 // PWM output (0-255)
11 analogWrite(PINO, 128); // 50% duty cycle

```

4.3 Comunicação Serial Avançada

Configuração e Métodos Úteis:

```

1 void setup() {
2     // Inicializa serial com baud rate
3     Serial.begin(115200);
4
5     // Aguarda conexao serial (para debugging)

```

```

6  while (!Serial) {
7      delay(10);
8  }
9  }
10
11 void enviarDados() {
12     // Diferentes metodos de envio
13     Serial.println("Mensagem completa"); // Com quebra de linha
14     Serial.print("Valor: ");             // Sem quebra
15     Serial.println(42);                  // Numero com quebra
16
17     // Formatacao avancada
18     Serial.printf("Temperatura: %.2f C\n", 23.45);
19     Serial.printf("Umidade: %d%%\n", 65);
20 }
21
22 void lerComandos() {
23     if (Serial.available() > 0) {
24         String comando = Serial.readString();
25         comando.trim(); // Remove espacos extras
26
27         if (comando == "LED_ON") {
28             digitalWrite(LED_PIN, HIGH);
29             Serial.println("LED ligado");
30         }
31         else if (comando == "LED_OFF") {
32             digitalWrite(LED_PIN, LOW);
33             Serial.println("LED desligado");
34         }
35     }
36 }

```

4.4 Controle de Tempo Profissional

Evitando delay():

```

1  // NAO FAZER ISSO (bloqueante):
2  void loop() {
3      digitalWrite(LED_PIN, HIGH);
4      delay(1000);           // Congela todo o sistema
5      digitalWrite(LED_PIN, LOW);
6      delay(1000);           // Nada mais funciona
7  }
8
9  // FAZER ASSIM (nao-bloqueante):
10 unsigned long previousMillis = 0;
11 const long interval = 1000; // 1 segundo
12
13 void loop() {
14     unsigned long currentMillis = millis();
15
16     if (currentMillis - previousMillis >= interval) {
17         previousMillis = currentMillis;
18
19         // Executa a cada 1 segundo
20         digitalWrite(LED_PIN, !digitalRead(LED_PIN));
21     }
22 }

```

```

23 // Outras tarefas executam livremente aqui
24 lerSensores();
25 processarDados();
26 verificarConexao();
27 }

```

Múltiplos Timers:

```

1 // Estrutura para multiplos timers
2 struct Timer {
3     unsigned long previous;
4     long interval;
5     bool enabled;
6 };
7
8 Timer ledTimer = {0, 1000, true};
9 Timer sensorTimer = {0, 2000, true};
10 Timer commTimer = {0, 5000, true};
11
12 void checkTimer(Timer &timer, void (*function)()) {
13     if (timer.enabled && millis() - timer.previous >= timer.interval) {
14         timer.previous = millis();
15         function();
16     }
17 }
18
19 void piscaLED() { digitalWrite(LED_PIN, !digitalRead(LED_PIN)); }
20 void leSensor() { /* ler sensor */ }
21 void verificaConexao() { /* verificar conexao */ }
22
23 void loop() {
24     checkTimer(ledTimer, piscaLED);
25     checkTimer(sensorTimer, leSensor);
26     checkTimer(commTimer, verificaConexao);
27 }

```

4.5 Exemplo Completo: Sistema de Monitoramento

```

1 #include <Arduino.h>
2
3 // Definicoes de pinos
4 #define LED_PIN 2
5 #define BOTAO_PIN 0
6 #define SENSOR_A0 36
7
8 // Variaveis globais
9 int contador = 0;
10 float temperatura = 0;
11 bool sistemaAtivo = true;
12
13 void setup() {
14     Serial.begin(115200);
15     pinMode(LED_PIN, OUTPUT);
16     pinMode(BOTAO_PIN, INPUT_PULLUP);
17
18     Serial.println("=== SISTEMA MECATRONICO IoT ===");
19     Serial.println("Comandos: START, STOP, STATUS");

```



```

20 }
21
22 void loop() {
23     // Timing nao-bloqueante
24     static unsigned long lastRead = 0;
25     if (millis() - lastRead >= 1000) {
26         lastRead = millis();
27         if (sistemaAtivo) {
28             lerSensores();
29             enviarDados();
30         }
31     }
32
33     // Verificar comandos serial
34     verificarComandos();
35
36     // Verificar botao
37     verificarBotao();
38
39     delay(10); // Pequeno delay
40 }
41
42 void lerSensores() {
43     int raw = analogRead(SENSOR_A0);
44     temperatura = (raw / 4095.0) * 100; // Simulacao
45     contador++;
46 }
47
48 void enviarDados() {
49     Serial.printf("Leitura #%d: %.2f C\n", contador, temperatura);
50 }
51
52 void verificarComandos() {
53     if (Serial.available()) {
54         String cmd = Serial.readStringUntil('\n');
55         cmd.trim();
56
57         if (cmd == "START") {
58             sistemaAtivo = true;
59             Serial.println("Sistema ATIVADO");
60         }
61         else if (cmd == "STOP") {
62             sistemaAtivo = false;
63             Serial.println("Sistema DESATIVADO");
64         }
65         else if (cmd == "STATUS") {
66             Serial.printf("Status: %s\n", sistemaAtivo ? "ATIVO" : "INATIVO");
67             Serial.printf("Temperatura: %.2f C\n", temperatura);
68         }
69     }
70 }
71
72 void verificarBotao() {
73     static unsigned long lastPress = 0;
74     if (digitalRead(BOTAO_PIN) == LOW && millis() - lastPress > 200) {
75         lastPress = millis();
76         sistemaAtivo = !sistemaAtivo;
77         Serial.printf("Botao pressionado! Sistema: %s\n",

```

```

78         sistemaAtivo ? "ATIVO" : "INATIVO");
79     }
80 }

```

Listing 2: Exemplo completo de sistema de monitoramento

4.6 Técnicas Avançadas de Debug

Sistema de Logging:

```

1  enum LogLevel { DEBUG, INFO, WARNING, ERROR };
2
3  void log(LogLevel level, const char* message) {
4      const char* levels[] = {"DEBUG", "INFO", "WARNING", "ERROR"};
5      Serial.printf("[%s] %s\n", levels[level], message);
6
7      // LED pisca diferente conforme nivel
8      if (level == ERROR) {
9          for(int i=0; i<3; i++) {
10             digitalWrite(LED_PIN, HIGH); delay(100);
11             digitalWrite(LED_PIN, LOW); delay(100);
12         }
13     }
14 }
15
16 // Uso:
17 log(INFO, "Sistema iniciado");
18 log(DEBUG, "Temperatura lida: 23.5C");
19 log(ERROR, "Sensor nao respondendo!");

```

Monitoramento de Performance:

```

1  void medirPerformance() {
2      static unsigned long loopCount = 0;
3      static unsigned long lastReport = 0;
4
5      loopCount++;
6
7      if (millis() - lastReport >= 5000) {
8          lastReport = millis();
9          Serial.printf("Loops por segundo: %.2f\n",
10                      loopCount / 5.0);
11          loopCount = 0;
12      }
13 }

```

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

```
1 #include <EWiFi.h>           // Biblioteca personalizada para WiFi
2 #include <PubSubClient.h>    // Biblioteca para MQTT
3 #include <WiFiPassword.h>    // Arquivo com credenciais
4
5 // Configuracoes MQTT
6 const char* mqtt_server = "test.mosquitto.org";
7 const int mqtt_port = 1883;
8
9 // Topicos MQTT
10 const char* topic_subscribe = "device/led";
11 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

```
1 WiFiClient espClient;
2 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

```
1 void callback(char* topic, byte* payload, unsigned int length) {
2     Serial.print("Mensagem recebida no topico: ");
3     Serial.print(topic);
4     Serial.print(" - Mensagem: ");
5
6     String message;
7     for (int i = 0; i < length; i++) {
8         message += (char)payload[i];
9     }
10    Serial.println(message);
11
12    if (String(topic) == topic_subscribe) {
```

```

13     if (message == "ON") {
14         digitalWrite(2, HIGH);
15         client.publish(topic_publish, "LED LIGADO");
16         Serial.println("LED ligado via MQTT");
17     }
18     else if (message == "OFF") {
19         digitalWrite(2, LOW);
20         client.publish(topic_publish, "LED DESLIGADO");
21         Serial.println("LED desligado via MQTT");
22     }
23 }
24 }

```

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

```

1 void reconnect() {
2     Serial.println("Conectando ao broker MQTT...");
3
4     while (!client.connected()) {
5         String clientId = "ESP32Client-";
6         clientId += String(random(0xffff), HEX);
7
8         if (client.connect(clientId.c_str())) {
9             Serial.println("Conectado ao broker MQTT!");
10            client.subscribe(topic_subscribe);
11            Serial.print("Inscrito no topico: ");
12            Serial.println(topic_subscribe);
13            client.publish(topic_publish, "ESP32 Conectado");
14        } else {
15            Serial.print("Falha na conexao MQTT, rc=");
16            Serial.print(client.state());
17            Serial.println(" Tentando novamente em 5 segundos...");
18            delay(5000);
19        }
20    }
21 }

```

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

```
1 void setup() {
2   Serial.begin(115200);
3   Serial.println("Iniciando ESP32...");
4
5   pinMode(2, OUTPUT);
6   digitalWrite(2, LOW);
7
8   Serial.println("Conectando ao WiFi...");
9
10  if (password != "XXXX") {
11    ewifi.setWiFi(SSID, password);
12  } else {
13    ewifi.setWiFi(SSID, WPA2_AUTH_PEAP, anonymous, username,
14      userpassword);
15  }
16
17  if (ewifi.connect(30)) {
18    Serial.println("Conectado ao WiFi!");
19    Serial.print("IP: ");
20    Serial.println(ewifi.getIP(WIFI_IPv4));
21    Serial.print("MAC: ");
22    Serial.println(ewifi.getmacAddress());
23  } else {
24    Serial.println("Falha na conexao WiFi!");
25    while(1) {
26      digitalWrite(2, HIGH);
27      delay(100);
28      digitalWrite(2, LOW);
29      delay(100);
30    }
31  }
32
33  client.setServer(mqtt_server, mqtt_port);
34  client.setCallback(callback);
35
36  Serial.println("Configuracao inicial concluida!");
37 }
```

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

```

1 void loop() {
2     if (!client.connected()) {
3         reconnect();
4     }
5
6     client.loop();
7     delay(100);
8 }

```

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

```

1 /** Input WiFi data
2  * Definition of macros SSID, username and password
3  */
4 #define SSID "UFU-Institucional"
5 #define username "SEU_USUARIO@ufu.br"
6 #define userpassword "SUA_SENHA"
7 #define password "XXXX" // Use "XXXX" para rede enterprise
8 #define anonymous "anonymous@ufu.br"
9
10 /** Input MQTT data
11  * Definition of macros MQTT server and port
12  */
13 #define MQTTServer "test.mosquitto.org"
14 #define MQTTPort 1883
15
16 /* DECLARATION OF TOPICS VARIABLES */
17 const char* topicLed = "device/led";
18 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 (IntroEMC) controle um LED da plataforma ESP32.

Requisitos:

- Android #1: Botão disponível na janela "Executar" do aplicativo IntroEMC, onde publica "ON"/"OFF" no tópico "GrupoX/led"
- ESP32 #2: Subscrive o tópico "GrupoX/led", controla LED no `OUTPUT_1` e publica o status do LED no tópico "GrupoX/status"
- Feedback visual com LED de status no ESP32 e no aplicativo IntroEMC

6.2 Desafio Extra: Sistema de Segurança com Múltiplos Comandos

Objetivo: Implementar sistema com vários comandos via MQTT.

Comandos a Implementar:

- "LED_ON"/ "LED_OFF" → Controle básico
- "BLINK=3" → Piscar 3 vezes
- "STATUS" → Retornar status atual
- "RESET" → Reiniciar contadores

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