**PROJETO RAIZ ALERTA - SISTEMA DE MONITORAMENTO DE UMIDADE**

**Relatório Técnico de Desenvolvimento**

**1. CIRCUITO ELETRÔNICO - MONTAGEM COMPLETA**

**1.1 COMPONENTES UTILIZADOS**

* **1x ESP32 DevKit V1** (Microcontrolador principal)
* **2x Sensores de Umidade de Solo** (Modelo analógico capacitivo)
* **1x Buzzer Passivo** (5V, 2KHz)
* **1x Protoboard** (830 pontos)
* **10x Jumpers Macho-Macho** (cores variadas)

**1.2 ESQUEMA DE CONEXÕES**

**SENSOR DE UMIDADE 1:**

* **VCC** → Pino **3.3V** do ESP32 (fio vermelho)
* **GND** → Pino **GND** do ESP32 (fio preto)
* **SINAL** → Pino **GPIO34** do ESP32 (fio amarelo)

**SENSOR DE UMIDADE 2:**

* **VCC** → Pino **3.3V** do ESP32 (fio vermelho)
* **GND** → Pino **GND** do ESP32 (fio preto)
* **SINAL** → Pino **GPIO35** do ESP32 (fio verde)

**BUZZER:**

* **POSITIVO (+)** → Pino **GPIO25** do ESP32 (fio laranja)
* **NEGATIVO (-)** → Pino **GND** do ESP32 (fio preto)

**1.3 DIAGRAMA DE MONTAGEM NA PROTOBOARD**

PROTOBOARD - VISTA SUPERIOR

═══════════════════════════════════════════════════════

A B C D E || F G H I J

1 [+] [+] [+] [+] [+]|| [-] [-] [-] [-] [-]

2

3 [====== ESP32 DEVKIT V1 ======]

4 [3V3] [GND]

5 [EN ] [GPIO23]

6 [VP ] [GPIO22]

7 [VN ] [TX0]

8 [GPIO34]←---SENSOR1 [RX0]

9 [GPIO35]←---SENSOR2 [GPIO21]

10 [GPIO32] [GND]

11 [GPIO33] [GPIO19]

12 [GPIO25]←---BUZZER(+) [GPIO18]

13 [GPIO26] [GPIO5]

14 [GPIO27] [GPIO17]

15 [GPIO14] [GPIO16]

16 [GPIO12] [GPIO4]

17 [GND] [GPIO0]

18 [GPIO13] [GPIO2]

19 [D2 ] [GPIO15]

20 [D3 ] [D1]

21 [CMD] [D0]

22 [5V ] [CLK]

**1.4 ESPECIFICAÇÕES TÉCNICAS**

* **Tensão de Operação:** 3.3V (IMPORTANTE: Não usar 5V nos sensores!)
* **Consumo Estimado:** ~150mA em operação normal
* **Alcance WiFi:** 2.4GHz, até 30 metros em ambiente interno
* **Precisão dos Sensores:** ±3% de umidade relativa
* **Tempo de Resposta:** < 1 segundo

**2. CÓDIGO DO FIRMWARE - ESP32**

**2.1 ARQUIVO PRINCIPAL: RaizAlerta.ino**

/\*

\* =====================================================

\* RAIZ ALERTA - Sistema de Monitoramento de Umidade

\* =====================================================

\* Versão: 1.0.0

\* Data: 17/12/2024

\* Desenvolvido para: ESP32 DevKit V1

\* =====================================================

\*/

#include <WiFi.h>

#include <WebServer.h>

#include <ArduinoJson.h>

#include <Preferences.h>

#include "secrets.h"

// ===== DEFINIÇÕES DE HARDWARE =====

#define SENSOR1\_PIN 34 // GPIO34 - ADC1\_CH6

#define SENSOR2\_PIN 35 // GPIO35 - ADC1\_CH7

#define BUZZER\_PIN 25 // GPIO25 - Buzzer

#define LED\_BUILTIN 2 // LED interno do ESP32

// ===== CONFIGURAÇÕES DO SISTEMA =====

#define INTERVALO\_LEITURA 10000 // 10 segundos

#define TAMANHO\_BUFFER\_MEDIA 6 // 6 leituras = 1 minuto

#define LIMITE\_SECO 30 // 0-30% = Seco

#define LIMITE\_MODERADO 50 // 31-50% = Moderado

#define LIMITE\_UMIDO 70 // 51-70% = Úmido

// 71-100% = Crítico

// ===== VARIÁVEIS GLOBAIS =====

WebServer server(80);

Preferences preferences;

float bufferSensor1[TAMANHO\_BUFFER\_MEDIA];

float bufferSensor2[TAMANHO\_BUFFER\_MEDIA];

int indiceBuffer = 0;

float umidadeAtual1 = 0;

float umidadeAtual2 = 0;

float umidadeMedia = 0;

String faixaAtual = "Desconhecido";

unsigned long ultimaLeitura = 0;

bool alarmeLigado = false;

// Limites configuráveis

int limiteSeco = LIMITE\_SECO;

int limiteModerado = LIMITE\_MODERADO;

int limiteUmido = LIMITE\_UMIDO;

// Calibração

int valorSecoSensor1 = 4095;

int valorMolhadoSensor1 = 0;

int valorSecoSensor2 = 4095;

int valorMolhadoSensor2 = 0;

// ===== FUNÇÕES PRINCIPAIS =====

void setup() {

Serial.begin(115200);

Serial.println("\n=== RAIZ ALERTA INICIANDO ===");

// Configurar pinos

pinMode(BUZZER\_PIN, OUTPUT);

pinMode(LED\_BUILTIN, OUTPUT);

digitalWrite(BUZZER\_PIN, LOW);

// Carregar configurações salvas

carregarConfiguracoes();

// Conectar WiFi

conectarWiFi();

// Configurar servidor web

configurarRotas();

server.begin();

Serial.println("Sistema pronto!");

Serial.print("Acesse: http://");

Serial.println(WiFi.localIP());

}

void loop() {

server.handleClient();

// Realizar leitura a cada intervalo

if (millis() - ultimaLeitura >= INTERVALO\_LEITURA) {

realizarLeitura();

ultimaLeitura = millis();

}

// Verificar alarme

verificarAlarme();

}

// ===== FUNÇÕES DE LEITURA =====

void realizarLeitura() {

// Ler sensores

int leitura1 = analogRead(SENSOR1\_PIN);

int leitura2 = analogRead(SENSOR2\_PIN);

// Converter para porcentagem usando calibração

umidadeAtual1 = map(leitura1, valorSecoSensor1, valorMolhadoSensor1, 0, 100);

umidadeAtual2 = map(leitura2, valorSecoSensor2, valorMolhadoSensor2, 0, 100);

// Limitar valores entre 0 e 100

umidadeAtual1 = constrain(umidadeAtual1, 0, 100);

umidadeAtual2 = constrain(umidadeAtual2, 0, 100);

// Adicionar ao buffer

bufferSensor1[indiceBuffer] = umidadeAtual1;

bufferSensor2[indiceBuffer] = umidadeAtual2;

indiceBuffer = (indiceBuffer + 1) % TAMANHO\_BUFFER\_MEDIA;

// Calcular média

umidadeMedia = calcularMedia();

// Determinar faixa

faixaAtual = determinarFaixa(umidadeMedia);

// Log no Serial

Serial.print("S1: ");

Serial.print(umidadeAtual1);

Serial.print("% | S2: ");

Serial.print(umidadeAtual2);

Serial.print("% | Média: ");

Serial.print(umidadeMedia);

Serial.print("% | Faixa: ");

Serial.println(faixaAtual);

}

float calcularMedia() {

float soma1 = 0, soma2 = 0;

int contador = 0;

for (int i = 0; i < TAMANHO\_BUFFER\_MEDIA; i++) {

if (bufferSensor1[i] > 0 || bufferSensor2[i] > 0) {

soma1 += bufferSensor1[i];

soma2 += bufferSensor2[i];

contador++;

}

}

if (contador == 0) return 0;

return (soma1 + soma2) / (contador \* 2);

}

String determinarFaixa(float umidade) {

if (umidade <= limiteSeco) return "Seco";

else if (umidade <= limiteModerado) return "Moderado";

else if (umidade <= limiteUmido) return "Úmido";

else return "Crítico";

}

// ===== FUNÇÕES DE ALARME =====

void verificarAlarme() {

if (faixaAtual == "Crítico" && !alarmeLigado) {

ligarAlarme();

} else if (faixaAtual != "Crítico" && alarmeLigado) {

desligarAlarme();

}

}

void ligarAlarme() {

alarmeLigado = true;

tone(BUZZER\_PIN, 1000);

digitalWrite(LED\_BUILTIN, HIGH);

Serial.println("⚠️ ALARME ATIVADO!");

}

void desligarAlarme() {

alarmeLigado = false;

noTone(BUZZER\_PIN);

digitalWrite(LED\_BUILTIN, LOW);

Serial.println("✅ Alarme desativado");

}

// ===== ROTAS WEB =====

void configurarRotas() {

server.on("/", HTTP\_GET, handleRoot);

server.on("/status", HTTP\_GET, handleStatus);

server.on("/config", HTTP\_GET, handleGetConfig);

server.on("/config", HTTP\_POST, handleSetConfig);

server.on("/calibrate", HTTP\_POST, handleCalibrate);

server.on("/test-alarm", HTTP\_POST, handleTestAlarm);

}

void handleRoot() {

String html = "<h1>Raiz Alerta</h1>";

html += "<p>Sistema de Monitoramento Ativo</p>";

html += "<p>Endpoints disponíveis:</p>";

html += "<ul>";

html += "<li>GET /status - Status atual</li>";

html += "<li>GET /config - Configurações</li>";

html += "<li>POST /config - Atualizar configurações</li>";

html += "<li>POST /calibrate - Calibrar sensores</li>";

html += "</ul>";

server.send(200, "text/html", html);

}

void handleStatus() {

StaticJsonDocument<512> doc;

doc["sensor1"] = umidadeAtual1;

doc["sensor2"] = umidadeAtual2;

doc["media"] = umidadeMedia;

doc["faixa"] = faixaAtual;

doc["alarme"] = alarmeLigado;

doc["timestamp"] = millis();

JsonArray historico = doc.createNestedArray("historico");

for (int i = 0; i < TAMANHO\_BUFFER\_MEDIA; i++) {

JsonObject leitura = historico.createNestedObject();

leitura["s1"] = bufferSensor1[i];

leitura["s2"] = bufferSensor2[i];

}

String response;

serializeJson(doc, response);

server.send(200, "application/json", response);

}

void handleGetConfig() {

StaticJsonDocument<256> doc;

doc["limiteSeco"] = limiteSeco;

doc["limiteModerado"] = limiteModerado;

doc["limiteUmido"] = limiteUmido;

doc["intervaloLeitura"] = INTERVALO\_LEITURA;

String response;

serializeJson(doc, response);

server.send(200, "application/json", response);

}

void handleSetConfig() {

if (!server.hasArg("plain")) {

server.send(400, "application/json", "{\"erro\":\"Corpo vazio\"}");

return;

}

StaticJsonDocument<256> doc;

DeserializationError error = deserializeJson(doc, server.arg("plain"));

if (error) {

server.send(400, "application/json", "{\"erro\":\"JSON inválido\"}");

return;

}

// Atualizar limites

if (doc.containsKey("limiteSeco")) limiteSeco = doc["limiteSeco"];

if (doc.containsKey("limiteModerado")) limiteModerado = doc["limiteModerado"];

if (doc.containsKey("limiteUmido")) limiteUmido = doc["limiteUmido"];

// Salvar configurações

salvarConfiguracoes();

server.send(200, "application/json", "{\"status\":\"Configurações atualizadas\"}");

}

void handleCalibrate() {

if (!server.hasArg("plain")) {

server.send(400, "application/json", "{\"erro\":\"Corpo vazio\"}");

return;

}

StaticJsonDocument<256> doc;

DeserializationError error = deserializeJson(doc, server.arg("plain"));

if (error) {

server.send(400, "application/json", "{\"erro\":\"JSON inválido\"}");

return;

}

String tipo = doc["tipo"];

int umidade = doc["umidade"];

int leitura1 = analogRead(SENSOR1\_PIN);

int leitura2 = analogRead(SENSOR2\_PIN);

if (tipo == "seco" && umidade == 0) {

valorSecoSensor1 = leitura1;

valorSecoSensor2 = leitura2;

} else if (tipo == "molhado" && umidade == 100) {

valorMolhadoSensor1 = leitura1;

valorMolhadoSensor2 = leitura2;

}

salvarCalibracao();

StaticJsonDocument<256> response;

response["status"] = "Calibração realizada";

response["leitura1"] = leitura1;

response["leitura2"] = leitura2;

String jsonResponse;

serializeJson(response, jsonResponse);

server.send(200, "application/json", jsonResponse);

}

void handleTestAlarm() {

tone(BUZZER\_PIN, 1000);

delay(1000);

noTone(BUZZER\_PIN);

server.send(200, "application/json", "{\"status\":\"Teste concluído\"}");

}

// ===== FUNÇÕES AUXILIARES =====

void conectarWiFi() {

Serial.print("Conectando ao WiFi");

WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nWiFi conectado!");

Serial.print("IP: ");

Serial.println(WiFi.localIP());

}

void carregarConfiguracoes() {

preferences.begin("raiz-alerta", false);

limiteSeco = preferences.getInt("limiteSeco", LIMITE\_SECO);

limiteModerado = preferences.getInt("limiteModerado", LIMITE\_MODERADO);

limiteUmido = preferences.getInt("limiteUmido", LIMITE\_UMIDO);

valorSecoSensor1 = preferences.getInt("seco1", 4095);

valorMolhadoSensor1 = preferences.getInt("molhado1", 0);

valorSecoSensor2 = preferences.getInt("seco2", 4095);

valorMolhadoSensor2 = preferences.getInt("molhado2", 0);

preferences.end();

}

void salvarConfiguracoes() {

preferences.begin("raiz-alerta", false);

preferences.putInt("limiteSeco", limiteSeco);

preferences.putInt("limiteModerado", limiteModerado);

preferences.putInt("limiteUmido", limiteUmido);

preferences.end();

}

void salvarCalibracao() {

preferences.begin("raiz-alerta", false);

preferences.putInt("seco1", valorSecoSensor1);

preferences.putInt("molhado1", valorMolhadoSensor1);

preferences.putInt("seco2", valorSecoSensor2);

preferences.putInt("molhado2", valorMolhadoSensor2);

preferences.end();

}

**2.2 ARQUIVO DE CONFIGURAÇÃO: secrets.h**

#ifndef SECRETS\_H

#define SECRETS\_H

// Configurações de WiFi - ALTERAR COM SEUS DADOS

const char\* WIFI\_SSID = "NOME\_DA\_SUA\_REDE";

const char\* WIFI\_PASSWORD = "SENHA\_DA\_SUA\_REDE";

#endif

**2.3 ENDPOINTS DA API REST**

| **Método** | **Endpoint** | **Descrição** |
| --- | --- | --- |
| GET | /status | Retorna leituras atuais e histórico |
| GET | /config | Retorna configurações atuais |
| POST | /config | Atualiza limites de umidade |
| POST | /calibrate | Calibra sensores |
| POST | /test-alarm | Testa buzzer por 1 segundo |

**2.4 EXEMPLO DE RESPOSTA - GET /status**

{

"sensor1": 25.5,

"sensor2": 28.3,

"media": 26.9,

"faixa": "Seco",

"alarme": false,

"timestamp": 123456,

"historico": [

{"s1": 25.5, "s2": 28.3},

{"s1": 24.8, "s2": 27.9},

{"s1": 25.1, "s2": 28.1},

{"s1": 25.3, "s2": 28.0},

{"s1": 25.0, "s2": 28.2},

{"s1": 25.2, "s2": 28.4}

]

}

**3. PRÓXIMOS PASSOS**

**3.1 DESENVOLVIMENTO DO DASHBOARD WEB**

* Implementar interface responsiva
* Integrar com API REST do ESP32
* Criar visualizações em tempo real
* Implementar sistema de notificações

**3.2 TESTES DE CAMPO**

* Instalar em ambiente real
* Calibrar com solo local
* Validar alcance WiFi
* Testar autonomia energética

**3.3 MELHORIAS FUTURAS**

* Adicionar backup de dados
* Implementar modo offline
* Criar app mobile
* Integrar com Defesa Civil