REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DO DESENVOLVIMENTO, INDÚSTRIA, COMÉRCIO E SERVIÇOS
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512024003399-8**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1° de janeiro subsequente à data de 19/08/2024, em conformidade com o §2°, art. 2° da Lei 9.609, de 19 de Fevereiro de 1998.

Título: LESS: Low Energy System for Sensors

Data de publicação: 19/08/2024

Data de criação: 19/08/2024

Titular(es): INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE

Autor(es): JADER LUGON JUNIOR; ELIAS FERNANDES DE SOUSA; VICENTE DE PAULO SANTOS DE OLIVEIRA; ANDRÉ LEONARDO TAVARES PAULA; THIAGO RODRIGUES FARIA

Linguagem: C++

Campo de aplicação: AG-01; MA-04

Tipo de programa: IT-01

Algoritmo hash: SHA-256

Resumo digital hash: e3d6c91763aeee947888effdb89e0fb21e9f4649a62075d10ecb21aceb48f4bd

Expedido em: 24/09/2024

Aprovado por:
Carlos Alexandre Fernandes Silva
Chefe da DIPTO

DESCRIÇÃO TÉCNICA DO PROGRAMA

LSL ESP32 - Land Safety Logger para ESP32

1. Introdução

Programado na linguagem C++ com bibliotecas do Arduino IDE, com aplicação na área de monitoramento ambiental. O Land Safety Logger ESP32 é um firmware desenvolvido para ser embarcado em microcontroladores ESP32, destinado a aquisição, transmissão e retenção de dados de sensores ambientais. Este sistema é projetado para que o dispositivo não dependa de energia solar nem de internet, podendo operar em ambientes remotos, com robustez e capacidade de camuflagem.

O sistema permite basicamente seis funcionalidades: Gestão do tempo; coleta de dados (leitura de sensores e processamento de dados); Armazenamento seguro; Comunicação via Bluetooth (transmissão de dados, Configuração e Controle); Gestão de Energia com modo de adormecimento profundo; e Interface de usuário.

2. Características e descrição do funcionamento

O Land Safety Logger ESP32 apresenta como características principais a robustez e simplicidade, sendo projetado para operar em ambientes remotos e desafiadores, como florestas e áreas subterrâneas. Além disso, possibilita uma fácil integração, sendo simples de instalar e configurar, com um design robusto que minimiza a necessidade de manutenção. O sistema oferece segurança dos dados, protegendo os dados coletados contra perda e adulteração, assegurando a integridade das informações. A solução utiliza modos de deep sleep e configuração de alarmes para minimizar o consumo de energia, prolongando a vida útil do sistema em ambientes remotos, sendo ideal para monitoramento ambiental e agrícola adaptando-se a diferentes necessidades de coleta e análise de dados.

A leitura de sensores pode ser configurada para leitura analógica em 8 pinos de entrada, sendo associados a 8 pinos de saída utilizados para ligar os sensores individualmente. O processamento de dados possibilita a coleta e faz a média de 5 leituras de até 8 sensores analógicos, armazenando e transmitindo a cada o tempo especificado, os valores médios de cada sensor e a leitura de referência. Esta referência pode ser obtida por uma bateria usada somente para esta leitura.

O armazenamento seguro possibilita o armazenamento com o sistema de arquivos LittleFS, armazenando os dados em formato CSV, garantindo a integridade e fácil acesso aos registros.

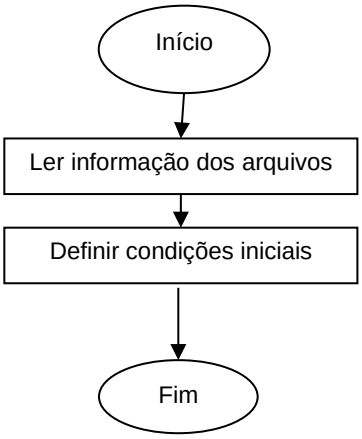
A comunicação de dados via Bluetooth pode transmitir dados via módulo LoRa se estiver conectado aos pinos RxTx (opcional). Envia os dados coletados e recebe comandos via

Bluetooth, permitindo interação com dispositivos móveis sem a necessidade de conexão à internet. O sistema de configuração e controle do Bluetooth permite a configuração remota do sistema e o envio de comandos para ajuste das funcionalidades.

A gestão de energia em modo de adormecimento profundo utiliza modos de deep sleep para otimizar o consumo de energia, acordando em intervalos programados ou por eventos específicos configurados no RTC (Real-Time Clock).

A interface de usuário oferece uma interface intuitiva para configuração e controle, facilitando a instalação e operação do sistema.

3. Fluxograma de operação



4. Entrada e Saída de Dados

A fim de simplificar a entrada e saída de dados, o método ASCII foi adotado. O usuário fará a inserção de dados em um arquivo ".dat", conforme o exemplo da seguinte configuração:
Input.dat

```
<BegincalculateSettling>
MP_DENSITY: 1500.0
WATER_DENSITY: 1025.0
Z: 0
DIAMETER: 0.001
SALINITY: 0.4
TEMPERATURE: 20
CORREL: 2
<EndcalculateSettling>
```

5. O Programa computacional

```
#include <Wire.h>
#include <RTClib.h>
#include <BluetoothSerial.h>
#include <LittleFS.h>
#include <esp_bt.h>
#include <ESP32Time.h>

#define uS_TO_S_FACTOR 1000000ULL
```

```
RTC_DATA_ATTR int TEMPO_PARA_DORMIR = 20; // Tempo para dormir em segundos
#define PINO_DE_ACORDAR GPIO_NUM_27 // Pino para acordar o ESP32
#define TEMPO_PARA_ESPERAR_SERIAL 120000 // Tempo de espera para comandos via
serial (ms)
#define LED_INTERNO 22 // LED embutido (PNP LoRa)
#define PINO_REFERENCIA 19 // Pino de referência
#define SDA 12 // Pino SDA
#define SCL 14 // Pino SCL

int SAIDAS[8] = {13, 15, 2, 0, 4, 5, 18, 23}; // Pinos de saída
int ENTRADAS[8] = {26, 25, 33, 32, 35, 34, 39, 36}; // Pinos de entrada
const char DESCRICAO[] = "Land Safety Logger v3, gpios 27,26,25,33,32,35,34,39 e Vcc para
LoRa gpio 22";
const char IDENTIFICACAO[] = "Land Safety Logger 1";

RTC_DATA_ATTR int contador = 0;
RTC_DS3231 rtcExterno;
ESP32Time rtcInterno;
bool usarRtcExterno = false;
BluetoothSerial SerialBT;

void inicializarSistemaDeArquivos() {
    if (!LittleFS.begin()) {
        Serial.println("Erro ao montar o LittleFS. Tentando formatar...");
        if (LittleFS.format()) {
            Serial.println("LittleFS formatado com sucesso!");
        } else {
            Serial.println("Falha ao formatar LittleFS!");
            return;
        }
    }
}

void configurarPinos() {
    for (int i = 0; i < sizeof(SAIDAS)/sizeof(SAIDAS[0]); i++) {
        pinMode(SAIDAS[i], OUTPUT);
        digitalWrite(SAIDAS[i], LOW);
    }
    for (int i = 0; i < sizeof(ENTRADAS)/sizeof(ENTRADAS[0]); i++) {
        pinMode(ENTRADAS[i], INPUT);
    }
    pinMode(PINO_DE_ACORDAR, INPUT_PULLDOWN);
    pinMode(LED_INTERNO, OUTPUT);
    digitalWrite(LED_INTERNO, 0); delay(500); digitalWrite(LED_INTERNO, 1);
```

```
}

void inicializarRTC() {
   Wire.begin(SDA, SCL);
   Serial.println(DESCRICAO);
   if (rtcExterno.begin()) {
      usarRtcExterno = true;
      if (rtcExterno.lostPower()) {
         rtcExterno.adjust(DateTime(F(__DATE__), F(__TIME__)));
      }
      // Atualiza o RTC interno com o tempo do RTC externo
      rtcInterno.setTime(rtcExterno.now().unixtime());
   } else {
      Serial.println("Não foi possível encontrar o RTC externo. Usando RTC interno.");
   }
}

void leituraDeSensores() {
   contador++;
   Serial.println("Leitura: " + String(contador));

   // Obtém o tempo atual do RTC interno
   unsigned long unix = rtcInterno.getEpoch();
   String leituras = String(unix);
   // Percorre os interruptores dos sensores, ligando, medindo, atribuindo valor a N[] e
deligando cada sensor.
   int N[8];
   for (int i = 0; i < 8; i++) {
      digitalWrite(SAIDAS[i], HIGH);
      delay(100);

      int sum = 0;
      for (int j = 0; j < 5; j++) {
         sum += analogRead(ENTRADAS[i]);
         delay(10);
      }
      // Calcula a média de 5 leituras
      N[i] = sum / 5;
      digitalWrite(SAIDAS[i], LOW);
      yield();
      leituras += "," + String(N[i]);
   }
   int REF = analogRead(PINO_REFERENCIA); //Bateria de referência para identificação de
ruídos.
```

```
    leituras += "," + String(REF);
    leituras += ",";
    leituras += IDENTIFICACAO;

    File file = LittleFS.open("/dados.csv", "a");
    if (file) {
        file.println(leituras);
        file.close();
    } else {
        Serial.println("Erro ao abrir arquivo para escrita");
    }
    digitalWrite(LED_INTERNO, LOW); // Ativa módulo LoRa
    Serial2.begin(115200, SERIAL_8N1, 16, 17); // Inicia Serial2 para comunicação LoRa
conectado aos pinos 16 e 17.
    Serial.println("Dados: " + leituras);
    Serial2.println(leituras);
    Serial.flush();
    Serial2.flush();
    if (Serial2.available()) {
    String response = Serial2.readStringUntil('\n');
    if (eNumero(response)) {
        long novoTempo = response.toInt();
        if (novoTempo > 100000) {
            Serial.println("Comando recebido. Ajustando tempo epoch para " + response);
            rtcExterno.adjust(DateTime(novoTempo));
            rtcInterno.setTime(novoTempo);
  }
    }
      }
    digitalWrite(LED_INTERNO, HIGH);
}

void baixarDados() {
    File file = LittleFS.open("/dados.csv", "r");
    if (!file) {
        Serial.println("Erro ao abrir arquivo para leitura");
        SerialBT.println("Erro ao abrir arquivo para leitura");
        return;
    }

    while (file.available()) {
        String line = file.readStringUntil('\n');
        Serial.println(line);
        SerialBT.println(line);
```

```
    }
    file.close();
}

void deletarArquivo() {
    if (LittleFS.remove("/dados.csv")) {
        Serial.println("Arquivo deletado com sucesso");
    } else {
        Serial.println("Erro ao deletar arquivo");
    }
}

bool eNumero(const String &str) {
    for (char c : str) {
        if (!isDigit(c)) {
            return false;
        }
    }
    return true;
}

void tratarComando(String comando) {
    comando.trim();

    if (comando == "download") {
        Serial.println("Comando recebido. Enviando dados...");
        baixarDados();
    } else if (comando == "del") {
        Serial.println("Comando recebido. Deletando arquivo...");
        deletarArquivo();
    } else if (comando == "exit") {
        Serial.println("Saindo do modo de comando.");
        return;
    } else if (eNumero(comando)) {
        long novoTempo = comando.toInt();
        if (novoTempo > 10000) {
            Serial.println("Comando recebido. Ajustando tempo epoch para " + String(novoTempo));
            rtcExterno.adjust(DateTime(novoTempo));
            rtcInterno.setTime(novoTempo);
        } else if (novoTempo > 0) {
            Serial.println("Comando recebido. Ajustando tempo de sono para " + String(novoTempo)
+ " segundos.");
            TEMPO_PARA_DORMIR = novoTempo;
        }
```

```
    } else {
        Serial.println("Comando desconhecido.");
    }
}

void entradaDeComandos() {
    SerialBT.begin("ESP32_BT");
    esp_bredr_tx_power_set(ESP_PWR_LVL_N12, ESP_PWR_LVL_N12);
    digitalWrite(LED_INTERNO, LOW);
    unsigned long startMillis = millis();
    Serial.println("Comandos: download, del, exit. Entradas: Tempo desligado e unixtime");
    SerialBT.println("Comandos: download, del, exit. Entradas: Tempo desligado e unixtime");
    while (millis() - startMillis < TEMPO_PARA_ESPERAR_SERIAL) {
        if (Serial.available() > 0) {
            String comando = Serial.readStringUntil('\n');
            tratarComando(comando);
        }
        if (SerialBT.available() > 0) {
            String comando = SerialBT.readStringUntil('\n');
            tratarComando(comando);
        }
    }
    digitalWrite(LED_INTERNO, HIGH);
}

void setup() {
    Serial.begin(115200);
    inicializarSistemaDeArquivos();
    configurarPinos();
    inicializarRTC();

    esp_sleep_wakeup_cause_t wakeup_reason = esp_sleep_get_wakeup_cause();
    switch (wakeup_reason) {
        case ESP_SLEEP_WAKEUP_EXT0:
            Serial.println("Acordado por pino externo (EXT0).");
            entradaDeComandos();
            break;
        case ESP_SLEEP_WAKEUP_TIMER:
            Serial.println("Acordado por timer.");
            leituraDeSensores();
            break;
        default:
            Serial.println("Acordado por outra razão.");
            break;
```

```
    }

    esp_sleep_enable_ext0_wakeup(PINO_DE_ACORDAR, 1);
    esp_sleep_enable_timer_wakeup(TEMPO_PARA_DORMIR * uS_TO_S_FACTOR);

    Serial.println("Entrando em sono profundo");
    delay(1000);
    esp_deep_sleep_start();
}

void loop() {
    // Loop vazio, pois tudo é tratado no setup
}
```

Referências

KOOI, M. et al. Ups and Downs in the Ocean: Effects of Biofouling on Vertical Transport of Microplastics. Environmental Science & Technology, v. 51, n. 14, p. 7963–7971, 18 jul. 2017.

MILLERO, F. J.; HUANG, F. The density of seawater as a function of salinity (5 to 70 g kg−1) and temperature (273.15 to 363.15 K). Ocean Science, v. 5, n. 2, p. 91–100, 5 mai. 2009.