SƠ ĐỒ NỐI DÂY ESP32-S3 - TRẠM KIỂM SOÁT LŨ

Phiên bản: v4 - Rút gọn + ESP-IDF v5.5.1
Âm thanh: DFPlayer Mini + Buzzer 5V nhỏ
Nguồn: USB Type-C
Trạng thái: ✅ Đã tối ưu

---

1. KẾT NỐI MODULE LORA RA-02 (SPI)

Chân LoRa	ESP32-S3	Chức năng
VCC	3V3	Nguồn 3.3V
GND	GND	Mass
NSS	GPIO 10	FSPICS0
MOSI	GPIO 11	FSPID
MISO	GPIO 13	FSPIQ
SCK	GPIO 12	FSPICLK
RST	GPIO 14	Reset + pull-up 4.7kΩ
DIO0	GPIO 2	Interrupt

```
3V3 ---|4.7kΩ|--- RST --- GPIO 14
```

---

2. LCD I2C

Chân LCD	ESP32-S3
VCC	5V/3V3
GND	GND
SDA	GPIO 8
SCL	GPIO 9

---

3. LED CẢNH BÁO

LED	GPIO	Màu	Cảnh báo
LED 1	GPIO 15	🟢 Xanh	Thấp
LED 2	GPIO 16	🟡 Vàng	TB
LED 3	GPIO 4	🔴 Đỏ	Cao

Mỗi LED qua điện trở 220Ω xuống GND.

---

4. DFPLAYER MINI

```
DFPlayer   ESP32-S3
VCC        5V
RX         GPIO 17 (TX)
TX         GPIO 18 (RX)
SPK+       Loa (+)
SPK-       Loa (-)
GND        GND
```

```
DFPlayer        ESP32-S3
VCC   ──────────────── 5V
RX    ──────────────── GPIO 17 (U1TXD)
TX    ──────────────── GPIO 18 (U1RXD)
SPK+  ──────────────── Loa (+)
SPK-  ──────────────── Loa (-)
GND   ──────────────── GND
  │
[Thẻ SD FAT32: mp3/001.mp3, 002.mp3, 003.mp3]
```

---

5. BUZZER 5V

```
GPIO 5 ---|--- [Buzzer 5V] ---|--- GND
```

⚠️ Active Buzzer 5V, dòng < 30mA

---

6. BẢNG TỔNG HỢP

GPIO	Chức năng
2	LoRa DIO0
4	LED Đỏ + 220Ω
5	Buzzer
8	LCD SDA
9	LCD SCL
10	LoRa NSS
11	LoRa MOSI
12	LoRa SCK
13	LoRa MISO
14	LoRa RST
15	LED Xanh + 220Ω
16	LED Vàng + 220Ω
17	DFPlayer TX
18	DFPlayer RX

---

7. CODE ESP-IDF v5.5.1

CMakeLists.txt (project root)

cmake

```cmake
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(tram_kiem_soat)
```

main/CMakeLists.txt

cmake

```cmake
idf_component_register(
    SRCS "main.c" "lcd_i2c.c" "dfplayer.c"
    INCLUDE_DIRS "."
)
```

---

main/lcd_i2c.h

c

```c
#ifndef LCD_I2C_H
#define LCD_I2C_H

#include "driver/i2c_master.h"
#include "esp_err.h"

#define LCD_ADDR           0x27
#define I2C_MASTER_SDA_IO   8
#define I2C_MASTER_SCL_IO   9
#define I2C_MASTER_FREQ_HZ  100000

esp_err_t lcd_init(void);
void lcd_clear(void);
void lcd_set_cursor(uint8_t col, uint8_t row);
void lcd_print(const char *str);
void lcd_deinit(void);

#endif
```

---

main/lcd_i2c.c

c

```c
#include "lcd_i2c.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"

static const char *TAG = "LCD_I2C";
static i2c_master_bus_handle_t bus_handle = NULL;
static i2c_master_dev_handle_t dev_handle = NULL;

static esp_err_t lcd_send_byte(uint8_t data, uint8_t mode) {
    uint8_t high_nibble = (data & 0xF0) | mode | 0x08; // EN=1
    uint8_t low_nibble = ((data << 4) & 0xF0) | mode | 0x08;

    uint8_t buffer[4];
    buffer[0] = high_nibble | 0x04; // EN=1, Backlight=1
    buffer[1] = high_nibble;        // EN=0
    buffer[2] = low_nibble | 0x04;  // EN=1
    buffer[3] = low_nibble;         // EN=0

    return i2c_master_transmit(dev_handle, buffer, 4, 1000);
}

esp_err_t lcd_init(void) {
    // Cấu hình I2C Master Bus (ESP-IDF v5.x)
    i2c_master_bus_config_t bus_config = {
        .i2c_port = I2C_NUM_0,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .clk_source = I2C_CLK_SRC_DEFAULT,
        .glitch_ignore_cnt = 7,
        .flags.enable_internal_pullup = true,
    };
    ESP_ERROR_CHECK(i2c_new_master_bus(&bus_config, &bus_handle));

    // Thêm thiết bị LCD
    i2c_device_config_t dev_config = {
        .dev_addr_length = I2C_ADDR_BIT_LEN_7,
        .device_address = LCD_ADDR,
        .scl_speed_hz = I2C_MASTER_FREQ_HZ,
    };
    ESP_ERROR_CHECK(i2c_master_bus_add_device(bus_handle, &dev_config, &dev_handle));
    // Khởi tạo LCD theo datasheet HD44780
    vTaskDelay(pdMS_TO_TICKS(50));
    lcd_send_byte(0x30, 0x00);
    vTaskDelay(pdMS_TO_TICKS(5));
    lcd_send_byte(0x30, 0x00);
    vTaskDelay(pdMS_TO_TICKS(1));
    lcd_send_byte(0x30, 0x00);
    vTaskDelay(pdMS_TO_TICKS(10));
    lcd_send_byte(0x20, 0x00); // 4-bit mode

    lcd_send_byte(0x28, 0x00); // 2 lines, 5x8 font
    lcd_send_byte(0x08, 0x00); // Display off
    lcd_send_byte(0x01, 0x00); // Clear
    vTaskDelay(pdMS_TO_TICKS(2));
    lcd_send_byte(0x06, 0x00); // Entry mode
    lcd_send_byte(0x0C, 0x00); // Display on, cursor off

    ESP_LOGI(TAG, "LCD initialized");
    return ESP_OK;
}

void lcd_clear(void) {
    lcd_send_byte(0x01, 0x00);
    vTaskDelay(pdMS_TO_TICKS(2));
}

void lcd_set_cursor(uint8_t col, uint8_t row) {
    uint8_t row_offsets[] = {0x00, 0x40};
    lcd_send_byte(0x80 | (col + row_offsets[row]), 0x00);
}

void lcd_print(const char *str) {
    while (*str) {
        lcd_send_byte(*str++, 0x01); // RS=1 for data
    }
}

void lcd_deinit(void) {
    if (dev_handle) {
        i2c_master_bus_rm_device(dev_handle);
    }
    if (bus_handle) {
        i2c_del_master_bus(bus_handle);
    }
}
```

---

main/dfplayer.h

c

```c
#ifndef DFPLAYER_H
#define DFPLAYER_H

#include <stdint.h>
#include "esp_err.h"

#define DFPLAYER_TXD  17
#define DFPLAYER_RXD  18
#define UART_NUM      UART_NUM_1

esp_err_t dfplayer_init(void);
void dfplayer_set_volume(uint8_t volume);
void dfplayer_play(uint8_t track);
void dfplayer_deinit(void);

#endif
```

---

main/dfplayer.c

c

```c
#include "dfplayer.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include "esp_log.h"

static const char *TAG = "DFPLAYER";
static const int UART_BUF_SIZE = 1024;

static void dfplayer_send_cmd(uint8_t cmd, uint16_t arg) {
    uint8_t buffer[10];
    buffer[0] = 0x7E; // Start
    buffer[1] = 0xFF; // Version
    buffer[2] = 0x06; // Length
    buffer[3] = cmd;  // Command
    buffer[4] = 0x00; // Feedback (0=no, 1=yes)
    buffer[5] = (arg >> 8) & 0xFF;
    buffer[6] = arg & 0xFF;

    // Checksum
    int16_t checksum = -(buffer[1] + buffer[2] + buffer[3] + buffer[4] + buffer[5] + buffer[6]);
    buffer[7] = (checksum >> 8) & 0xFF;
    buffer[8] = checksum & 0xFF;
    buffer[9] = 0xEF; // End

    uart_write_bytes(UART_NUM, (const char*)buffer, 10);
}

esp_err_t dfplayer_init(void) {
    const uart_config_t uart_config = {
        .baud_rate = 9600,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_DEFAULT,
    };

    ESP_ERROR_CHECK(uart_param_config(UART_NUM, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_NUM, DFPLAYER_TXD, DFPLAYER_RXD,
                                UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE));
    ESP_ERROR_CHECK(uart_driver_install(UART_NUM, UART_BUF_SIZE * 2, 0, 0, NULL, 0));
    vTaskDelay(pdMS_TO_TICKS(500)); // DFPlayer cần thời gian khởi động

    ESP_LOGI(TAG, "DFPlayer initialized");
    return ESP_OK;
}

void dfplayer_set_volume(uint8_t volume) {
    if (volume > 30) volume = 30;
    dfplayer_send_cmd(0x06, volume);
}

void dfplayer_play(uint8_t track) {
    dfplayer_send_cmd(0x03, track);
}

void dfplayer_deinit(void) {
    uart_driver_delete(UART_NUM);
}
```

---

main/main.c

c

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "lcd_i2c.h"
#include "dfplayer.h"

static const char *TAG = "MAIN";

// Định nghĩa GPIO
#define LED_LOW     15
#define LED_MED     16
#define LED_HIGH    4
#define BUZZER_PIN  5

// Hàm khởi tạo GPIO
static void gpio_init_all(void) {
    gpio_config_t io_conf = {
        .pin_bit_mask = (1ULL << LED_LOW) | (1ULL << LED_MED) |
                        (1ULL << LED_HIGH) | (1ULL << BUZZER_PIN),
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE,
    };
    gpio_config(&io_conf);

    // Tắt tất cả LED và buzzer
    gpio_set_level(LED_LOW, 0);
    gpio_set_level(LED_MED, 0);
    gpio_set_level(LED_HIGH, 0);
    gpio_set_level(BUZZER_PIN, 0);
}

// Hàm bíp buzzer
static void beep(int duration_ms) {
    gpio_set_level(BUZZER_PIN, 1);
    vTaskDelay(pdMS_TO_TICKS(duration_ms));
    gpio_set_level(BUZZER_PIN, 0);
}
```

```
// Xử lý mức nước
static void handle_water_level(float level) {
    // Tắt tất cả LED
    gpio_set_level(LED_LOW, 0);
    gpio_set_level(LED_MED, 0);
    gpio_set_level(LED_HIGH, 0);

    // Hiển thị LCD
    lcd_clear();
    lcd_print("Muc nuoc:");
    lcd_set_cursor(0, 1);
    char buffer[16];
    snprintf(buffer, sizeof(buffer), "%.1f cm", level);
    lcd_print(buffer);

    // Xử lý theo mức
    if (level < 50) {
        ESP_LOGI(TAG, "✅ An toan");
        gpio_set_level(LED_LOW, 1);

    } else if (level >= 50 && level < 100) {
        ESP_LOGW(TAG, "⚠️ Canh bao muc 1");
        gpio_set_level(LED_MED, 1);
        dfplayer_play(1);

    } else if (level >= 100 && level < 150) {
        ESP_LOGW(TAG, "⚠️⚠️ Canh bao muc 2");
        gpio_set_level(LED_MED, 1);
        gpio_set_level(LED_HIGH, 1);
        dfplayer_play(2);
        beep(100);
        vTaskDelay(pdMS_TO_TICKS(100));
        beep(100);

    } else {
        ESP_LOGE(TAG, "🚨 Khan cap!");
        gpio_set_level(LED_HIGH, 1);
        dfplayer_play(3);
        for (int i = 0; i < 3; i++) {
            beep(100);
            vTaskDelay(pdMS_TO_TICKS(50));
        }
    }
}

void app_main(void) {
    ESP_LOGI(TAG, "Khoi dong he thong...");

    // Khởi tạo GPIO
    gpio_init_all();

    // Test LED + Buzzer
    gpio_set_level(LED_LOW, 1);
    beep(100);
    vTaskDelay(pdMS_TO_TICKS(200));
    gpio_set_level(LED_LOW, 0);

    gpio_set_level(LED_MED, 1);
    beep(100);
    vTaskDelay(pdMS_TO_TICKS(200));
    gpio_set_level(LED_MED, 0);

    gpio_set_level(LED_HIGH, 1);
    beep(100);
    vTaskDelay(pdMS_TO_TICKS(200));
    gpio_set_level(LED_HIGH, 0);

    // Khởi tạo LCD
    ESP_ERROR_CHECK(lcd_init());
    lcd_clear();
    lcd_print("Tram Kiem Soat");
    lcd_set_cursor(0, 1);
    lcd_print("Khoi dong...");

    // Khởi tạo DFPlayer
    ESP_ERROR_CHECK(dfplayer_init());
    dfplayer_set_volume(25);

    // TODO: Khởi tạo LoRa ở đây

    lcd_clear();
    lcd_print("San sang!");
    beep(100);
    vTaskDelay(pdMS_TO_TICKS(100));
    beep(100);

    ESP_LOGI(TAG, "He thong san sang!");

    // Main loop - Test
    while (1) {
        // Mô phỏng nhận dữ liệu LoRa
        float test_levels[] = {30, 70, 120, 180};

        for (int i = 0; i < 4; i++) {
            beep(50); // Bíp khi "nhận LoRa"
            handle_water_level(test_levels[i]);
            vTaskDelay(pdMS_TO_TICKS(5000));
        }
    }
}
```

---

8. HƯỚNG DẪN BUILD

bash

```
# Cài đặt ESP-IDF v5.5.1
cd ~/esp
git clone -b v5.5.1 --recursive https://github.com/espressif/esp-idf.git esp-idf-v5.5.1
cd esp-idf-v5.5.1
./install.sh esp32s3

# Activate
. ./export.sh

# Build project
cd your_project_folder
idf.py set-target esp32s3
idf.py build
idf.py -p /dev/ttyUSB0 flash monitor
```

---

9. CHECKLIST

- ☐ Tất cả kết nối GND
- ☐ LoRa: GPIO 10-14, 2
- ☐ LCD: GPIO 8 (SDA), 9 (SCL)
- ☐ DFPlayer: GPIO 17 (TX), 18 (RX)
- ☐ LED: GPIO 15, 16, 4 + điện trở 220Ω
- ☐ Buzzer: GPIO 5
- ☐ Thẻ SD FAT32 có mp3/001.mp3, 002.mp3, 003.mp3
- ☐ Pull-up 4.7kΩ cho LoRa RST

✅ HOÀN TẤT

Sơ đồ đã tối ưu với:

- ✅ LoRa SPI: GPIO 10-13 (FSPI hardware)
- ✅ LCD I2C: GPIO 8, 9
- ✅ DFPlayer UART: GPIO 17, 18 (UART1)
- ✅ LED 3 màu: GPIO 15, 16, 4
- ✅ Buzzer: GPIO 5
- ✅ Code ESP-IDF v5.5.1 đầy đủ cho LCD I2C và DFPlayer