Guía Completa: Configuración de Pines GPIO en ESP32-S3

Tabla de Contenidos

Introducción

Los pines GPIO (General Purpose Input/Output) del ESP32-S3 son fundamentales para conectar sensores, actuadores, LEDs, botones y otros componentes externos. Esta guía te enseñará paso a paso cómo configurarlos correctamente.

Requisitos Previos

- ESP32-S3 DevKit
- ESP-IDF configurado
- Visual Studio Code con extensión ESP-IDF
- Conocimientos básicos de programación en C

Paso 1: Incluir Librerías

El primer paso es incluir las librerías necesarias en tu archivo `main.c`:

```
#include <stdio.h>
#include "driver/gpio.h"          // Librería principal para GPIO
#include "freertos/FreeRTOS.h"    // Sistema operativo en tiempo real
#include "freertos/task.h"        // Para delays y tareas
```

¿Por qué estas librerías?

- **driver/gpio.h**: Contiene todas las funciones para manejar pines GPIO
- **freertos/FreeRTOS.h**: Base del sistema operativo
- **freertos/task.h**: Permite usar delays (vTaskDelay)

Paso 2: Definir los Pines

Define los pines que vas a usar con constantes para facilitar el mantenimiento:

```
// Definir pines usando constantes
#define LED_INTERNO     GPIO_NUM_2   // LED interno del ESP32-S3
#define BOTON_USER      GPIO_NUM_0   // Botón BOOT del dispositivo
#define PIN_SENSOR      GPIO_NUM_4   // Pin para conectar sensores
#define PIN_RELAY       GPIO_NUM_5   // Pin para controlar relés
#define PIN_BUZZER      GPIO_NUM_6   // Pin para buzzer/alarma
```

Buenas Prácticas:

- Usa nombres descriptivos
- Agrupa pines por funcionalidad
- Documenta qué conectarás a cada pin

Paso 3: Entender la Estructura gpio_config_t

Cada pin se configura usando la estructura `gpio_config_t`:

```
gpio_config_t configuracion = {
    .pin_bit_mask = (1ULL << NUMERO_PIN),    // Máscara del pin
    .mode = GPIO_MODE_OUTPUT,                // Modo del pin
    .pull_up_en = GPIO_PULLUP_DISABLE,       // Pull-up interno
    .pull_down_en = GPIO_PULLDOWN_DISABLE,   // Pull-down interno
    .intr_type = GPIO_INTR_DISABLE           // Tipo de interrupción
};
```

Parámetros explicados:

pin_bit_mask

- Especifica qué pin(es) configurar
- Usa `(1ULL << NUMERO_PIN)` para un solo pin
- Para múltiples pines: `(1ULL << PIN1) | (1ULL << PIN2)`

mode

Modo	Descripción	Cuándo usar
`GPIO_MODE_OUTPUT`	Solo salida	LEDs, relés, buzzers
`GPIO_MODE_INPUT`	Solo entrada	Botones, sensores digitales
`GPIO_MODE_INPUT_OUTPUT`	Bidireccional	Comunicación, buses de datos

pull_up_en / pull_down_en

Configuración	Cuándo usar
GPIO_PULLUP_ENABLE	Botones, señales digitales
GPIO_PULLDOWN_ENABLE	Señales que necesitan estado bajo por defecto
GPIO_PULLUP_DISABLE	Sensores analógicos, señales externas

Paso 4: Configurar Pines de Salida

Para pines que controlan LEDs, relés, buzzers, etc.:

```
void configurar_pines_salida(void) {
    // Configuración para LED interno
    gpio_config_t config_led = {
        .pin_bit_mask = (1ULL << LED_INTERNO),
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_led);

    // Configuración para relé
    gpio_config_t config_relay = {
        .pin_bit_mask = (1ULL << PIN_RELAY),
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_relay);

    // Inicializar en estado apagado
    gpio_set_level(LED_INTERNO, 0);
    gpio_set_level(PIN_RELAY, 0);

    printf("Pines de salida configurados\n");
}
```

Paso 5: Configurar Pines de Entrada

Para botones, sensores, interruptores:

```
void configurar_pines_entrada(void) {
    // Configuración para botón con pull-up interno
    gpio_config_t config_boton = {
        .pin_bit_mask = (1ULL << BOTON_USER),
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = GPIO_PULLUP_ENABLE,    // Importante para botones
```

```
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_boton);

    // Configuración para sensor digital
    gpio_config_t config_sensor = {
        .pin_bit_mask = (1ULL << PIN_SENSOR),
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_sensor);

    printf("Pines de entrada configurados\n");
}
```

Paso 6: Usar los Pines Configurados

Escribir a Pines de Salida:

```
// Encender LED
gpio_set_level(LED_INTERNO, 1);

// Apagar LED
gpio_set_level(LED_INTERNO, 0);

// Alternar estado
static int estado = 0;
estado = !estado;
gpio_set_level(LED_INTERNO, estado);
```

Leer Pines de Entrada:

```
// Leer estado del botón
int estado_boton = gpio_get_level(BOTON_USER);

// Verificar si está presionado (considerando pull-up)
if (estado_boton == 0) {
    printf("Botón presionado\n");
} else {
    printf("Botón no presionado\n");
}
```

Ejemplo Completo

Aquí tienes un ejemplo completo que puedes usar en tu `main.c`:

```
#include <stdio.h>
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

// Definir pines
#define LED_INTERNO     GPIO_NUM_2
#define BOTON_USER      GPIO_NUM_0
#define PIN_SENSOR      GPIO_NUM_4
#define PIN_RELAY       GPIO_NUM_5

void configurar_todos_los_pines(void) {
    printf("Configurando pines GPIO...\n");

    // Configurar LED (salida)
    gpio_config_t config_led = {
        .pin_bit_mask = (1ULL << LED_INTERNO),
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_led);

    // Configurar botón (entrada con pull-up)
    gpio_config_t config_boton = {
        .pin_bit_mask = (1ULL << BOTON_USER),
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = GPIO_PULLUP_ENABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_boton);

    // Configurar relé (salida)
    gpio_config_t config_relay = {
        .pin_bit_mask = (1ULL << PIN_RELAY),
        .mode = GPIO_MODE_OUTPUT,
        .pull_up_en = GPIO_PULLUP_DISABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&config_relay);

    // Inicializar salidas en estado apagado
    gpio_set_level(LED_INTERNO, 0);
    gpio_set_level(PIN_RELAY, 0);

    printf("✓ Todos los pines configurados correctamente\n");
}
```

```
void app_main(void)
{
    printf("=== Sistema de Monitoreo de Granjas ===\n");
    printf("Iniciando configuración de pines...\n");

    // Configurar todos los pines
    configurar_todos_los_pines();

    // Variables para el ejemplo
    int contador = 0;
    int estado_led = 0;

    printf("Iniciando bucle principal...\n");

    while(1) {
        // Leer estado del botón
        int boton_presionado = !gpio_get_level(BOTON_USER); // Invertido por pull-
up

        // Si el botón está presionado, activar relé
        if (boton_presionado) {
            gpio_set_level(PIN_RELAY, 1);
            printf("🟢 Botón presionado - Relé activado\n");
        } else {
            gpio_set_level(PIN_RELAY, 0);
        }

        // Parpadear LED cada segundo
        if (contador % 10 == 0) { // Cada 1 segundo (100ms * 10)
            estado_led = !estado_led;
            gpio_set_level(LED_INTERNO, estado_led);
            printf("💡 LED: %s | Contador: %d\n",
                   estado_led ? "ENCENDIDO" : "APAGADO", contador/10);
        }

        contador++;
        vTaskDelay(100 / portTICK_PERIOD_MS); // Esperar 100ms
    }
}
```

Troubleshooting

Problemas Comunes y Soluciones

1. El LED no enciende

```
// Verificar configuración
printf("Estado del pin %d: %d\n", LED_INTERNO, gpio_get_level(LED_INTERNO));

// Verificar que esté configurado como salida
```

```
printf("Pin configurado como: %s\n",
       gpio_get_direction(LED_INTERNO) ? "SALIDA" : "ENTRADA");
```

2. El botón siempre lee el mismo valor

- Verificar que el pull-up esté habilitado
- Revisar conexiones físicas
- Verificar que el pin no esté usado por otro periférico

3. Errores de compilación

```
# Error: gpio.h no encontrado
# Solución: Verificar que ESP-IDF esté correctamente instalado

# Error: GPIO_NUM_X no definido
# Solución: Usar números directos o verificar la definición
```

4. Comportamiento errático

```
// Agregar delays para estabilizar lecturas
vTaskDelay(pdMS_TO_TICKS(50)); // 50ms de delay

// Verificar interferencias eléctricas
// Usar filtros de software para señales ruidosas
```

Comandos para Probar

Compilar y Flashear:

```
# Limpiar proyecto
idf.py clean

# Compilar
idf.py build

# Flashear y monitorear
idf.py flash monitor
```

Monitorear Salida:

```
# Solo monitorear (si ya está flasheado)
idf.py monitor
```

```
# Salir del monitor: Ctrl+]
```

Próximos Pasos

1. **Añadir más sensores**: Temperatura, humedad, pH
2. **Implementar PWM**: Para control de velocidad/brillo
3. **Configurar interrupciones**: Para respuesta inmediata a eventos
4. **Implementar I2C/SPI**: Para sensores más complejos
5. **Añadir WiFi**: Para monitoreo remoto

Referencias Útiles

- ESP32-S3 Pinout Diagram
- GPIO API Reference
- FreeRTOS Task Reference

Autor: Diego
Fecha: 26 de octubre de 2025
Proyecto: PantallaGranjas261025