S.E.P. **S.E.S.** **TecNM**

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Aguascalientes

REPORTE DE CODIGO 5 PARA ESP32

Materia:

Arquitectura de Computadoras

Alumnos:

Arantza Darina Gómez Hernández - 23151198

José Andrés Rodríguez Marmolejo - 23151344

Aguascalientes, Ags., 17 de noviembre de 2025

Reporte de Código: Señal SOS, Blink Multicolor y Efecto Pulse seleccionados por Botones además de interrupciones externas y guardados en la memoria (ESP32)

1. Introducción

En este reporte se explica el funcionamiento de un programa para el ESP32 donde se controlan tres efectos de luz (SOS, Blink y Pulse) usando botones físicos. Además, se utiliza una tarea controlable mediante interrupciones y un botón especial que permite pausar o reanudar el efecto que esté activo. Este tipo de prácticas nos ayuda a comprender cómo funciona la programación con tareas, cómo controlar un LED RGB y cómo usar interrupciones para hacer el programa más interactivo y eficiente.

2. Código

```
// Constantes a las que se les asigna el numero de pin físico que estan
conectados a los botones
int BOTON_SOS = 4;   // Pin 4 conectado al boton que ejecuta 'SOS()'
int BOTON_Blink = 5; // Pin 5 conectado al boton que ejecuta 'Blink()'
int BOTON_Pulse = 6; // Pin 6 conectado al boton que ejecuta 'Pulse()'
int BOTON_Int = 7;   // Pin 7, ejecuta el metodo de interrupcion

// Es un manejador que permite interruptir y resumir tareas
TaskHandle_t xControllableTaskHandle = NULL;

// Metodo que revisa si el boton de interrupcion es presionado
void pausa(void *pvParameters) {
  bool pausado = false;

  for (;;) {
    if (digitalRead(BOTON_Int) == LOW) {
      vTaskDelay(pdMS_TO_TICKS(50)); // Desactiva el rebote de la entrada
para evitar múltiples activaciones con una sola presión
      if (digitalRead(BOTON_Int) == LOW) { // Se confirma la pulsacion
```

```
        // Se revisa el bool de pausado para revisar si la tarea debe ser
resumida o pausada
        if (pausado) {
          vTaskResume(xControllableTaskHandle); // Se continua con la
tarea
          pausado = false;
        } else {
          vTaskSuspend(xControllableTaskHandle); // Se pausa la tarea
          pausado = true;
        }
        while (digitalRead(BOTON_Int) == LOW); // La función espera a que
el boton deje de ser pulsado para iniciar
      }
    }
    vTaskDelay(pdMS_TO_TICKS(10)); // Pequeño retraso para evitar una
espera activa donde constantemente se espera confirmación
  }
}

void setup() {
  // Comandos para detectar el LED y los botones conectados
  pinMode(LED_BUILTIN, OUTPUT);

  pinMode(BOTON_SOS, INPUT_PULLUP);
  pinMode(BOTON_Blink, INPUT_PULLUP);
  pinMode(BOTON_Pulse, INPUT_PULLUP);
  pinMode(BOTON_Int, INPUT_PULLUP);

  xTaskCreate(
    actividadesLED,             // Nombre del metodo que se convierte en
tarea
    "Actividades con el LED",   // Un nombre descriptivo para la tarea
    2048,                       // El tamaño de la pila (stack) de la
tarea, especificado en bytes en el entorno ESP32
    NULL,                       // Un puntero que pasa parámetros o datos
a la función de la tarea cuando esta se inicia.
    1,                          // La prioridad en la que se ejecutará la
tarea.
    &xControllableTaskHandle
    // Un puntero a una variable de tipo TaskHandle_t*. Si la creación es
exitosa, se utiliza para devolver un identificador
    // mediante el cual la tarea puede ser referenciada y manipulada
  );

  xTaskCreate(
```

```
    pausa,
    "Pausa",
    2048,
    NULL,
    2,
    NULL
  );
}

void loop() {
  // El contenido de loop se movio al metodo de actividadesLED para
realizar las pausas y continuaciones.
}

// Metodo que se convierte en una tarea que contiene la selección de
botones.
void actividadesLED(void *pvParameters) {
  // Ciclo infintito para que la función que se convierte en un tarea
siempre este funcionando
  for(;;){
    // Condicionales que detectan los estados de los botones para ejecutar
los metodos especificados
    // Si el boton conectado al pin 4(BOTON_SOS) se presiona: Ejecuta el
metodo del efecto de destellos SOS
    if (digitalRead(BOTON_SOS) == LOW) {
      SOS();
    }

    // Si el boton conectado al pin 5(BOTON_Blink) se presiona: Ejecuta el
metodo del efecto Blink
    else if (digitalRead(BOTON_Blink) == LOW) {
      Blink();
    }

    // Si el boton conectado al pin 6(BOTON_Pulse) se presiona: Ejecuta el
metodo del efecto Pulse
    else if (digitalRead(BOTON_Pulse) == LOW) {
      Pulse();
    }
  }

}

void SOS() {
  // Simboliza S en codigo morse
```

```
  rgbLedWrite(LED_BUILTIN, 255,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,255,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,255);
  delay(200);

  rgbLedWrite(LED_BUILTIN, 0,0,0); //Apagado del LED para simbolizar el
nuevo bloque de parpadeos que representan una letra nueva
  delay(600);

  // Simboliza O en codigo morse
  rgbLedWrite(LED_BUILTIN, 255,0,0);
  delay(800);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,255,0);
  delay(800);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,255);
  delay(800);

  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(600);

  // Simboliza S en codigo morse
  rgbLedWrite(LED_BUILTIN, 255,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,255,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,255);
  delay(200);

  rgbLedWrite(LED_BUILTIN, 0,0,0);
}
```

```
void Blink() {
  rgbLedWrite(LED_BUILTIN, 255,0,0); // Rojo
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 255,165,0); // Naranja
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 255,255,0); //Amarillo
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,255,0); // Verde
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,255); // Azul
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 25,25,112); // Azul oscuro
  delay(200);
  rgbLedWrite(LED_BUILTIN, 0,0,0);
  delay(200);
  rgbLedWrite(LED_BUILTIN, 128,0,128); // Morado
  delay(200);

  rgbLedWrite(LED_BUILTIN, 0,0,0);
}

void Pulse() {
  for (int r=0; r<=255; r++){
    rgbLedWrite(LED_BUILTIN, r,r,r);
    delay(10);
    }

    for (int r=255; r>=0; r--){
    rgbLedWrite(LED_BUILTIN, r,r,r);
    delay(10);
    }
}
```

3. Explicación general del código

Constantes:

Las líneas iniciales definen tres constantes enteras (BOTON_SOS, BOTON_Blink y BOTON_Pulse) que indican los números de pines digitales del ESP32 a los que están conectados los botones. Cada botón tiene la función de ejecutar una rutina distinta dependiendo de cuál sea presionado. También se declara un botón que sirve para pausar o continuar la tarea principal gracias al uso de interrupciones y del sistema FreeRTOS que contiene el ESP32.

Función pausa():

Es el método que se encarga de pausar y reanudar las tareas (conversiones de métodos) por medio de funciones especificadas por el sistema FreeRTOS. Su estructura esta basada en los momentos de presionar el botón y la lectura de un valor booleano que define si el sistema debe de pausar o reanudar las actividades.

Función setup():

Esta función se ejecuta una sola vez al iniciar o reiniciar la placa. Dentro de setup() se configuran los modos de los pines y se crea una tarea a partir de otro método llamado actividadesLED, esta tarea es manipulada (pausada y resumida) por medio del método pausa(), la cual también es creada como una tarea secundaria.

Función actividadesLED():

Contiene un ciclo infinito donde se revisa constantemente el estado de los botones y permite un funcionamiento continuo. Dependiendo del que se presione, se ejecuta la función SOS(), Blink() o Pulse(). Cada una de estas funciones controla el LED RGB por medio de diferentes colores y tiempos de delay para crear los efectos deseados.

Función rgbLedWrite(pin, rojo, verde, azul):

Esta función controla los tres componentes de color del LED RGB. Cada parámetro (rojo, verde y azul) acepta valores entre 0 y 255, lo que permite generar una amplia gama de colores combinando distintas intensidades.

Bloque SOS():

Representa la señal internacional de emergencia “SOS” (· · · — — — · · ·) en colores rojo, verde y azul. Los destellos cortos simbolizan los puntos y los largos las rayas. Se usa delay() para establecer la duración de cada destello y los intervalos entre letras.

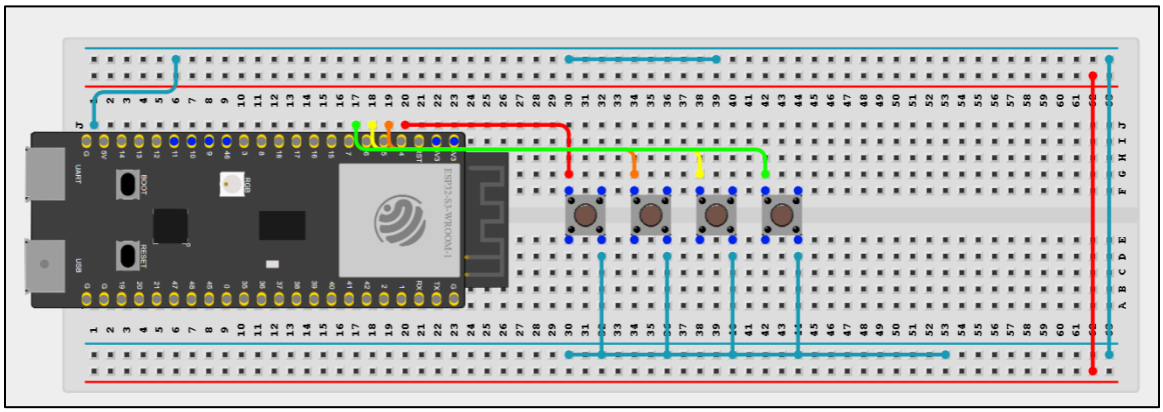
Bloque Blink():

Crea una secuencia de siete colores (rojo, naranja, amarillo, verde, azul, azul oscuro y morado), encendiendo el LED durante 200 milisegundos y apagándolo brevemente entre cada cambio. Este efecto muestra el uso de la función rgbLedWrite() para generar distintos tonos RGB.

Bloque Pulse():

Genera un efecto de 'respiración' o 'pulso luminoso' mediante dos bucles for: el primero incrementa el brillo del LED de 0 a 255, y el segundo lo reduce de 255 a 0. Se usan los mismos valores para los tres canales RGB para obtener luz blanca, cuya intensidad varía suavemente.

4. Diagrama del circuito



5. Conclusión

Este proyecto nos permitió comprender mejor cómo manejar tareas, botones y efectos visuales en el ESP32. Gracias al uso de interrupciones, el programa puede reaccionar rápidamente sin tener que depender solo del loop principal. Además, trabajar con un LED RGB nos ayudó a entender cómo se combinan los colores y cómo controlar intensidades. En general, este código demuestra cómo varios elementos pueden trabajar juntos para crear un sistema más completo y funcional.