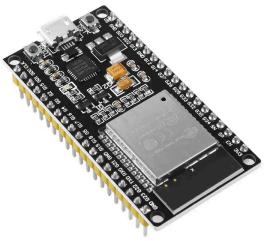
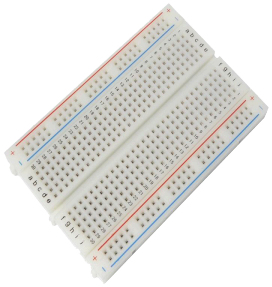
PRACTICA 1 : BLINK

MATERIAL

Para esta practica necesitaremos el microcontrolador ESP32, luces leds, protoboard (opcional), cables, resistencias.

OBJETIVO Y FUNCIONALIDAD DE LA PRACTICA

El objetivo de esta practica es producir el parpadeo periodico de un led.
Para ello se utilizara la salida serie para depurar el programa, conectaremos la led a un pin de salida del microprocesador y crearemos un bucle infinito para encender y apagar el led con un delay entre cada

EJERCICIOS

1. Generar el programa y subir el codigo al github de cada uno

2. Modificar el programa para que incluya el envio de datos (ON y OFF) al puerto serie. Añadir la inicialización del puerto serie y el envio cada vez que cambia el estado del led.

En estos ejercicios tenemos que utilizar la función ***Serial.println(****"mensaje a mostrar por el monitor"****)*** *para mostrar el mensaje por el monitor serie.*

Antes hay que iniciarlo con la función Serial.begin(velocidad del monitor, normalmente 115200). También hay que incluir en el archivo .ini la velocidad del monitor de la siguiente forma: monitor_speed = 115200. Y por último el microcontrolador tiene que estar conectado al ordenador por el puerto COM, no el USB.

```
**Código comentado:**
```

```ino
#include <Arduino.h>
#define LED_BUILTIN 2
// Define una constante que representa el pin del LED en la placa.

// Define una constante que representa el tiempo de espera en
// milisegundos entre los cambios de estado del LED.
#define DELAY 500


void setup() {
  // Configuramos el pin del LED como salida.
  pinMode(LED_BUILTIN, OUTPUT);

  // Inicia la comunicación serie a 115200 baudios.
  Serial.begin(115200);
}

// Función en bucle, que se repite infinitas veces
void loop() {
  // Enciende el LED (pone el pin en nivel alto).
  digitalWrite(LED_BUILTIN, HIGH);

  // Espera 500 milisegundos.
  delay(DELAY);

  // Envía el mensaje "OFF" al monitor serie.
  Serial.println("OFF");

  // Apaga el LED (pone el pin en nivel bajo).
  digitalWrite(LED_BUILTIN, LOW);

  // Espera 500 milisegundos.
  delay(DELAY);

  // Envía el mensaje "ON" al monitor serie.
  Serial.println("ON");
}
```

3. Modificar el programa para que actue directamente sobre los registros de los puertos de entrada y salida

Para este ejercicio implementamos el siguiente codigo.

```
// Establecer un puntero al registro de I/O
uint32_t *gpio_out = (uint32_t *)GPIO_OUT_REG;

// Activar el bit correspondiente al pin 2
*gpio_out |= (1 << 2);

// Alternar el estado del bit correspondiente al pin 2
*gpio_out ^= (1 << 2);
```

*Substituimos todos los digitalWrite por *gpio_out |= (1<<2)*

Código comentado:

```
#include <Arduino.h>
#include "soc/gpio_reg.h" // Necesari perque funcioni

#define LED_BUILTIN 2
#define DELAY 500

void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
    Serial.begin(115200);
}

void loop() {
    Serial.println("Iniciando...");

    uint32_t *gpio_out = (uint32_t *)GPIO_OUT_REG;
    *gpio_out |= (1 << LED_BUILTIN);  // Enciende el LED
    Serial.println("ON");
    delay(DELAY);

    *gpio_out ^= (1 << LED_BUILTIN);  // Apaga el LED
    Serial.println("OFF");
    delay(DELAY);
}
```

4. Eliminar los delay modificar el pin de salida a uno cualquiera de los que estan disponibles i medir con el osciloscopio cual es la màxima frecuencia de apagado encendido que permite el microcontrolador. Medir la frecuencia en estos cuatro casos:

- Con el envio por el puerto série del mensaje i utilizando las funciones de Arduino

```
#include <Arduino.h>
#define LED_BUILTIN 4

void setup() {
pinMode(LED_BUILTIN, OUTPUT);
Serial.begin(115200);
}

void loop() {
Serial.begin(115200);
digitalWrite(LED_BUILTIN, HIGH);
Serial.println("OFF");
digitalWrite(LED_BUILTIN, LOW);
Serial.println("ON");
}
```

Utilizamos el digitalWrite para encender o apagar el LED y el Serial-println para mostrar el mensaje por el monitor serie.

- Con el envio por el puerto série y accedirendo directamente a los registros

```
#include <Arduino.h>
#include "soc/gpio_reg.h"

#define LED_BUILTIN 4

void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
    Serial.begin(115200);
}

void loop() {

    uint32_t *gpio_out = (uint32_t *)GPIO_OUT_REG;
    *gpio_out |= (1 << LED_BUILTIN);  // Enciende el LED
    Serial.println("ON");

    *gpio_out ^= (1 << LED_BUILTIN);  // Apaga el LED
    Serial.println("OFF");
}
```

*Utilizamos *gpio_out para encender o apagar el LED y el Serial-println para mostrar el mensaje por el monitor serie.*

- Sin el envio por el puerto série del mensaje i utilizando las funciones de Arduino

En este caso seria el mismo que el primero pero sin la linea de Serial.begin(115200) y Serial.println("mensaje")

- Sin el envio por el puerto série y accedirendo directamente a los registros

En este caso seria el mismo que el segundo pero sin la linea de Serial.begin(115200) y Serial.println("mensaje")

5. Generar un informe fichero informe.MD (markdown) donde se muestre el codigo, un diagrama de flujo y un diagrama de tiempos

Código

```
#include <Arduino.h>

#define LED_BUILTIN 2
#define DELAY 1000

void setup() {
    Serial.begin(115200);  // Inicializa la comunicación serie
    pinMode(LED_BUILTIN, OUTPUT);  // Configura el LED como salida
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.println("ON"); // Mensaje cuando el LED está encendido
    delay(DELAY);

    digitalWrite(LED_BUILTIN, LOW);
    Serial.println("OFF"); // Mensaje cuando el LED está apagado
    delay(DELAY);
}
```

Diagrama de flujo

graph TD; Start --> |Iniciar comunicación serie| Init_Serial Init_Serial --> |Configurar pin LED| Config_LED Config_LED --> |Bucle infinito| Loop Loop --> |Encender LED| LED_ON LED_ON --> |Enviar ON al puerto serie| Serial_ON Serial_ON --> |Esperar 1000ms| Delay1 Delay1 --> |Apagar LED| LED_OFF LED_OFF --> |Enviar OFF al puerto serie| Serial_OFF Serial_OFF --> |Esperar 1000ms| Delay2 Delay2 --> Loop

Diagrama de Tiempos

Serial TX representa el momento en que el ESP32 envía datos al monitor serie.

En el código, cada vez que el LED cambia de estado, el ESP32 imprime "ON" o "OFF" en la consola serie con Serial.println().

6. Responder a la siguiente pregunta en el programa que se ha realizado cual es el tiempo libre que tiene el procesador ?

En este caso, el código usa delay(1000);, lo que significa que el procesador se queda inactivo esperando durante 1000 ms en cada ciclo.

Cálculo del tiempo libre:
• Cada ciclo del loop dura 2000 ms (1000 ms encendido + 1000 ms apagado).
• Durante 2000 ms, el ESP32 no ejecuta ninguna otra tarea porque está esperando en delay().
• Por lo tanto, el tiempo libre es casi nulo, ya que el microcontrolador no hace nada más útil durante el delay().

EJERCICIO VOLUNTARIO

- Leer el valor de un convertidor A/D de entrada ; sacarlo por el puerto serie y sacar el mismo valor por otro pin D/A

Vamos a usar el ADC (Convertidor Analógico a Digital) para leer un valor analógico de un pin y luego usar el DAC (Convertidor Digital a Analógico) para enviar ese mismo valor a otro pin.

Paso 1: Leer valor del ADC (entrada analógica)

El ESP32 tiene varios pines ADC (pines de entrada analógica) que permiten leer valores analógicos (de 0 a 3.3V en la mayoría de las placas). En este caso, puedes usar un pin como el 2 para la entrada analógica.

Paso 2: Enviar el valor a un DAC (salida analógica)

El ESP32 también tiene pines DAC (Digital to Analog Converter) que te permiten generar una señal analógica de 0 a 3.3V a partir de un valor digital. Usaremos el pin 25 como salida DAC.

Código comentado

```
#include <Arduino.h>

#define ADC_PIN 2       // Pin de entrada analógica (ADC)
#define DAC_PIN 25       // Pin de salida analógica (DAC)

// Configurar el canal de PWM
#define PWM_CHANNEL 0
#define PWM_FREQ 5000
#define PWM_RESOLUTION 8 // Resolución de 8 bits (0-255)

void setup() {
    Serial.begin(115200);  // Iniciar la comunicación por el puerto serie

    // Configurar el ADC
    analogReadResolution(12); // Resolución de 12 bits (0-4095)
    analogSetAttenuation(ADC_0db);  // Atenuación para que el rango de entrada sea de 0-

    // Configurar el PWM para el DAC
    ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RESOLUTION);  // Configurar el canal PWM
    ledcAttachPin(DAC_PIN, PWM_CHANNEL);  // Asignar el pin al canal PWM
}

void loop() {
    // Leer valor del ADC (entrada analógica)
    int adcValue = analogRead(ADC_PIN);

    // Enviar el valor por el puerto serie
    Serial.print("ADC Value: ");
    Serial.println(adcValue);

    // Mapeamos el valor de ADC (0-4095) a (0-255) para la salida PWM
    int pwmValue = map(adcValue, 0, 4095, 0, 255);

    // Escribir el valor al pin DAC a través del PWM
    ledcWrite(PWM_CHANNEL, pwmValue);

    delay(100);  // Pausa de 100 ms
}
```

Serial Monitor

Deberia mostrar algo parecido a:

```
ADC Value: 733
```