Practica A

#include <Arduino.h>

struct Button {

const uint8\_t PIN;

uint32\_t numberKeyPresses;

bool pressed;

};

Button button1 = {18, 0, false};

void IRAM\_ATTR isr() {

button1.numberKeyPresses += 1;

button1.pressed = true;

}

void setup() {

Serial.begin(115200);

pinMode(button1.PIN, INPUT\_PULLUP);

attachInterrupt(button1.PIN, isr, FALLING);

}

void loop() {

if (button1.pressed) {

Serial.printf("Button 1 has been pressed %u times\n",

button1.numberKeyPresses);

button1.pressed = false;

}

//Detach Interrupt after 1 Minute

static uint32\_t lastMillis = 0;

if (millis() - lastMillis > 60000) {

lastMillis = millis();

detachInterrupt(button1.PIN);

Serial.println("Interrupt Detached!");

}

}

[env:esp32-s3-devkitm-1]

platform = espressif32

board = esp32-s3-devkitm-1

framework = arduino

monitor\_speed = 115200

Este código configura un botón conectado al pin 18 del ESP32 y utiliza una interrupción para contar cuántas veces se ha presionado el botón.

* Se define una estructura Button que contiene el pin del botón, el número de presiones y el estado del botón (pressed).
* La función isr() se ejecuta cuando el botón se presiona, incrementando el contador numberKeyPresses y marcando que el botón fue presionado (pressed = true).
* En el setup(), se configura el pin 18 como entrada con resistencia pull-up y se adjunta una interrupción al pin para detectar la caída de señal (cuando el botón se presiona).
* En el loop(), si el botón ha sido presionado, el código imprime cuántas veces se ha presionado el botón en el Monitor Serie. Luego, la variable pressed se restablece a false.
* Después de 1 minuto (60000 milisegundos), se desactiva la interrupción, y se imprime un mensaje indicando que la interrupción ha sido detenida.

El archivo de configuración en platformio.ini establece la plataforma y la velocidad del monitor serie.

El contador se desfasa debido a los rebotes que suceden

Hay veces que no cuenta todos los rebotes en el flanco.

En resumen, el programa cuenta las veces que se presiona un botón, lo muestra por el Monitor Serie y desactiva la interrupción después de un minuto.

Practica B

#include <Arduino.h>

volatile int interruptCounter = 0; // Se declara como volatile

int totalInterruptCounter = 0; // No necesita ser volatile

hw\_timer\_t \*timer = NULL;

portMUX\_TYPE timerMux = portMUX\_INITIALIZER\_UNLOCKED;

void IRAM\_ATTR onTimer() {

portENTER\_CRITICAL\_ISR(&timerMux); // Entramos en el modo crítico en la ISR

interruptCounter++; // Incrementamos el contador de interrupciones

portEXIT\_CRITICAL\_ISR(&timerMux); // Salimos del modo crítico en la ISR

}

void setup() {

Serial.begin(115200);

// Inicialización del temporizador (1us = 80)

timer = timerBegin(0, 80, true); // Timer 0, prescaler 80 (para 1us)

timerAttachInterrupt(timer, &onTimer, true); // Adjuntamos la interrupción

timerAlarmWrite(timer, 1000000, true); // Alarma en 1 segundo (1000000 us)

timerAlarmEnable(timer); // Habilitamos la alarma

}

void loop() {

if (interruptCounter > 0) { // Si ha ocurrido una interrupción

portENTER\_CRITICAL(&timerMux); // Entramos en el modo crítico

interruptCounter--; // Decrementamos el contador de interrupciones

portEXIT\_CRITICAL(&timerMux); // Salimos del modo crítico

totalInterruptCounter++; // Contamos la interrupción total

Serial.print("An interrupt has occurred. Total number: ");

Serial.println(totalInterruptCounter); // Imprimimos el total

}

}

Este código configura un temporizador hardware en el ESP32 que genera una interrupción cada 1 segundo. La interrupción se maneja en la función onTimer(), que es una rutina de interrupción (ISR) donde se incrementa el contador interruptCounter cada vez que el temporizador se activa.

En el setup(), se inicializa el temporizador con un prescaler de 80 (para que cuente en microsegundos) y se configura para generar una interrupción cada 1 segundo. Además, se habilita la interrupción y se adjunta la función onTimer() como la rutina a ejecutar cuando ocurra la interrupción.

En el loop(), el programa verifica si el contador interruptCounter ha sido incrementado por la ISR. Si es así, el contador se decrementa de forma segura dentro de una sección crítica (portENTER\_CRITICAL y portEXIT\_CRITICAL), y el contador total de interrupciones (totalInterruptCounter) se incrementa. Luego, el número total de interrupciones se imprime en el Monitor Serie.

El uso de las secciones críticas asegura que el acceso a las variables compartidas entre la ISR y el código principal esté sincronizado correctamente, evitando condiciones de carrera. Este proceso se repite continuamente, y el monitor serie muestra el número total de interrupciones ocurridas.

EXTRA

#include <Arduino.h>

const int ledPin = 2; // Pin donde se conecta el LED

const int buttonPin1 = 4; // Pin del primer pulsador

const int buttonPin2 = 5; // Pin del segundo pulsador

volatile int blinkDelay = 1000; // Frecuencia inicial de parpadeo (en milisegundos)

volatile bool ledState = LOW; // Estado del LED

volatile unsigned long lastDebounceTime1 = 0; // Tiempo para debouncing del pulsador 1

volatile unsigned long lastDebounceTime2 = 0; // Tiempo para debouncing del pulsador 2

unsigned long debounceDelay = 50; // Tiempo de espera para evitar rebotes (en milisegundos)

hw\_timer\_t \*timer = NULL; // Temporizador

portMUX\_TYPE timerMux = portMUX\_INITIALIZER\_UNLOCKED;

void IRAM\_ATTR onTimer() {

// Cambiar el estado del LED solo cuando el tiempo de parpadeo haya pasado

portENTER\_CRITICAL\_ISR(&timerMux);

static unsigned long previousMillis = 0; // Variable para llevar el control del tiempo

unsigned long currentMillis = millis();

// Solo alternamos el LED si ha pasado el tiempo configurado en blinkDelay

if (currentMillis - previousMillis >= blinkDelay) {

ledState = !ledState; // Alternar el estado del LED

digitalWrite(ledPin, ledState); // Aplicamos el cambio de estado al LED

previousMillis = currentMillis; // Actualizamos el tiempo del último cambio

}

portEXIT\_CRITICAL\_ISR(&timerMux);

}

void IRAM\_ATTR buttonInterrupt1() {

if (millis() - lastDebounceTime1 > debounceDelay) {

portENTER\_CRITICAL\_ISR(&timerMux);

blinkDelay = max(100, blinkDelay - 100); // Aumenta la frecuencia (reduce el delay)

portEXIT\_CRITICAL\_ISR(&timerMux);

lastDebounceTime1 = millis();

}

}

void IRAM\_ATTR buttonInterrupt2() {

if (millis() - lastDebounceTime2 > debounceDelay) {

portENTER\_CRITICAL\_ISR(&timerMux);

blinkDelay += 100; // Disminuye la frecuencia (aumenta el delay)

portEXIT\_CRITICAL\_ISR(&timerMux);

lastDebounceTime2 = millis();

}

}

void setup() {

Serial.begin(115200);

pinMode(ledPin, OUTPUT);

pinMode(buttonPin1, INPUT\_PULLUP);

pinMode(buttonPin2, INPUT\_PULLUP);

// Configurar interrupciones para los pulsadores

attachInterrupt(buttonPin1, buttonInterrupt1, FALLING);

attachInterrupt(buttonPin2, buttonInterrupt2, FALLING);

// Configuración del temporizador para generar interrupciones a intervalos

timer = timerBegin(0, 80, true); // Timer 0, prescaler 80 para generar interrupciones a 1ms

timerAttachInterrupt(timer, &onTimer, true); // Asociamos la interrupción

timerAlarmWrite(timer, blinkDelay, true); // Configuramos el temporizador con el delay inicial

timerAlarmEnable(timer); // Habilitamos la alarma

}

void loop() {

// El código en el loop es muy simple ya que todo está manejado por las interrupciones

}