INFORME PRÁCTICA 2: Interrupciones

Para esta práctica, primeramente hemos definido qué es una interrupción hardware y los tipos de interrupciones que existen. Después hemos visto como expresar estas interrupciones en nuestro ESP32-S3 así como la sintaxis que se utiliza.

Interrupción por GPIO

1er código

En este primer código se muestra como crear una interrupción con el ESP32-S3 donde intervinene el componente físico del interruptor para hacerla. En nuestro caso, al no diponer, utilizamos 2 jumpers: uno conectado al pin elegido (18), y otro conectado al GND del procesador. Una vez los juntas, se hace la interrupción. Por pantalla del terminal se podrá ver las veces que se ha hecho una interrupción e indica, después de un cierto tiempo, cuando se dejan de contar estas interrupciones.

```
#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!");
}
```

```
}
```

Inclusión de bibliotecas y declaración de variables globales

En este código, declaramos la librería *Arduino.h* para poder utilizar la sintaxis correspondiente. Creamos una estructura llamada *Button* donde dentro de ella encontramos:

+ ***const uint8_t PIN***

  Corresponde a la declaraciónd del pin del botón.

+ ***uint32_t numberKeyPresses***

  Se declara la variable que dará el número de veces que se ha presionado el botón.

+ ***bool pressed***

  Se declara una variable booleana para indicar si el botón está presionado.

Declaración e inicialización de variables

Se declara un objeto ***button1*** con estructura de la variable global *Button* para después inicializarla. En este caso, está asociada al pin 18, sin presionar y sin estar presionado.

Función ***void IRAM_ATTR isr()***

Dentro del void encontramos las siguientes declaraciones:

+ ***button1.numberKeyPresses += 1***

  Se incrementa el contador de pulsaciones cada vez que haya una interrupción.

+ ***button1.pressed = true***

  Indica el bóton como que ha sido presionado.

Configuración de la función ***setup()***

+ ***Serial.begin(115200)***

  Inicializa la comunicación serial a una velocidad de 115200 baudios

+ ***pinMode(button1.PIN, INPUT_PULLUP)***

  Es una función que configura el pin del botón como una entrada con pull-up interno, es decir, se activa internamente una resistencia pull-up, lo que garantiza que el pin tenga un valor lógico alto cuando no esté conectado a nada externamente.

+ ***attachInterrupt(button1.PIN, isr, FALLING)***

  Adjunta la función de interrupción isr() al flanco de bajada del pin del botón.

Configuración de la función ***loop()***

+ Dentro del *if* en funcionamiento cuando el botón está pulsado:
    - ***Serial.printf("Button 1 has pressed %u times \n")***

Muestra por pantalla el mensaje entre "".

- ***button1.numberKeyPresses***

  Imprime el número de pulsaciones del botón cuando se presiona.

- ***button1.pressed = false***

  Reinicia el estado del botón a no presionado.

+ Fuera del *if*

  - Desactiva la interrupción después de 1 minuto
  - ***detachInterrupt(button1.PIN)***

    Desactiva la interupción asociada al botón

  - ***Serial.println("Interupt Detached!")***

    Muestra por pantalla que la interrupción se ha desactivado.

Interrupción por timer

2ndo código

El siguiente código muestra como crear una interrupción como en el anterior pero en vez de hacerla físicamente, es decir, con un interruptor y cada vez que se apriete la muestre, va haciendo interrupciones cada cierto tiempo. En este caso cada 4 segundos. En la pantalla del terminal nos cuenta, también, el numero de interrupciones que se hacen.

```
#include <Arduino.h>

volatile int interruptCounter;
int totalInterruptCounter;

hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

void IRAM_ATTR onTimer() {
    portENTER_CRITICAL_ISR(&timerMux);
    interruptCounter++;
    portEXIT_CRITICAL_ISR(&timerMux);
}

void setup() {
    Serial.begin(115200);
    timer = timerBegin(0, 80, true);
    timerAttachInterrupt(timer, &onTimer, true);
    timerAlarmWrite(timer, 4000000, true);
    timerAlarmEnable(timer);
}
void loop() {
    if (interruptCounter > 0) {
        portENTER_CRITICAL(&timerMux);
        interruptCounter--;
```

```
        portEXIT_CRITICAL(&timerMux);
    totalInterruptCounter++;

    Serial.print("An interrupt as occurred. Total number: ");
    Serial.println(totalInterruptCounter);
        }
}
```

Inclusión de bibliotecas y declaración de variables globales

En este código, la declaración de librerías es la misma que el anterior, con *Arduino.h*. Nos encontramos con una función llamada *volatile*, la cual consiste ordenar al compilador que optimice y así no intentará reducir el espacio, declarada al lado de un contador de interrupciones y de otro contador del total de interrupciones. Declaramos también un puntero a la estructura del temporizador con la función ***hw_timer_t***.

En el ***void IRAM_ATTR onTimer()***

+ ***portENTER_CRITICAL_ISR(&timerMux)***

  Entra en la sección crítica de la ISR.

+ ***interruptCounter++***

  Incrementa el contador de interrupciones.

+ ***portEXIT_CRITICAL_ISR(&timerMux)***

  Sale de la sección crítica de la ISR.

Configuración del *setup()*

+ ***Serial.begin(115200) timer = timerBegin(0, 80, true)***

  Inicializamos la comunicación serial, como antes y se inicializa el temporizador en el grupo 0 y el divisor de reloj 80, con el contador en modo ascendente (*true*).

+ ***timerAlarmWrite(timer, 4000000, true)***

  Configura el temporizador para generar una interrupción cada 4 segundos.

+ ***timerAlarmEnable(timer)***

  Habilita la alarma del temporizador.

En el *loop()*

+ Dentro del *if*
  - ***interruptCounter > 0***

    Determina si ha ocurrido al menos 1 interrupción.

  - ***portENTER_CRITICAL(&timerMux)***

    Entra en la sección crítica del bucle principal.

- ***interruptCounter--***

  Disminuye el contador de interrupciones.

- ***portEXIT_CRITICAL(&timerMux)***

  Sale de la sección crítica del bucle principal.

- ***totalInterruptCounter++***

  Incrementa el contador de interrupciones.

+ Fuera del *if*

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

    Imprime un mensaje indicando que se ha producido una interrupción.

  - ***Serial.println(totalInterruptCounter)***

    Imprime el número total de interrupciones hasta ahora.

Capturas de pantalla de lo que muestra el Terminal

* **CÓDIGO 1**

```
Button 1 has been pressed 1865 times
Button 1 has been pressed 1866 times
Button 1 has been pressed 1868 times
Button 1 has been pressed 2200 times
Button 1 has been pressed 2201 times
Button 1 has been pressed 2202 times
Button 1 has been pressed 2501 times
Button 1 has been pressed 2506 times
Button 1 has been pressed 2512 times
Button 1 has been pressed 2518 times
Interrupt Detached!
```

* **CÓDIGO 2**

```
entry 0x403c98d0
An interrupt as occurred. Total number: 1
An interrupt as occurred. Total number: 2
An interrupt as occurred. Total number: 3
An interrupt as occurred. Total number: 4
An interrupt as occurred. Total number: 5
An interrupt as occurred. Total number: 6
An interrupt as occurred. Total number: 7
An interrupt as occurred. Total number: 8
An interrupt as occurred. Total number: 9
An interrupt as occurred. Total number: 10
An interrupt as occurred. Total number: 11
```