INSTITUTO TECNOLÓGICO DE SONORA
Dirección de Ingeniería y Tecnología

Departamento de Computación y Diseño

Sistemas Empotrados

Práctica 08 - Interrupciones Externas en el Microcontrolador ESP32

Interrupciones por Hardware en el Microcontrolador ESP32

Una interrupción es una señal que detiene temporalmente lo que el CPU está ejecutando para saltar a ejecutar una rutina llamada rutina de atención a una interrupción.

Las interrupciones por hardware son generadas por fuentes o periféricos de hardware. Las interrupciones por hardware pueden ser internas o externas,

Las **interrupciones internas** son generadas por algún dispositivo dentro del microcontrolador, por ejemplo, los temporizadores del microcontrolador.

Las interrupciones **externas** son generadas por algún dispositivo externo al microcontrolador que cambia el estado en uno de los pines GPIO del microcontrolador.

En esta práctica se estudiarán programas que respondan a interrupciones externas. Se recomienda usar interrupciones para detectar entradas del usuario o eventos externos que queremos asegurarnos de detectar siempre.

En el microcontrolador ESP32 todos sus pines GPIO pueden configurarse para que acepten interrupciones.

El procedimiento para crear un programa en Microcontrolador ESP32 que responda a interrupciones externas es el siguiente:

1. Establezca que pin va a usarse para lanzar la interrupción y configúrelo como de entrada digital usando la función `pinMode()`:

   ```
   pinMode(pin, INPUT);
   ```

2. Establezca el nombre de la rutina que atenderá a la interrupción, *`ISR`* y el tipo de cambio de estado, *`mode`* en el pin dado por pin que generará la interrupción, usando la función `attachInterrupt()`:

   ```
   attachInterrupt(pin, ISR, mode)
   ```

   Los diferentes valores que puede tomar *`mode`* son:

   - **`LOW`** para lanzar la interrupción cuando el pin esté en el estado `LOW`.
   - **`HIGH`** para lanzar la interrupción cuando el pin esté en el estado `HIGH`.
   - **`CHANGE`** para lanzar la interrupción cuando el pin cambie de estado de `LOW` a `HIGH` o de `HIGH` a `LOW`.
   - **`RISING`** para lanzar la interrupción cuando el pin cambie de estado de `LOW` a `HIGH`.

- **FALLING** para lanzar la interrupción cuando el pin cambie de estado de `LOW` a `HIGH`.

3. Escriba la rutina que atenderá a la interrupción, *`ISR`*. Su sintaxis es:

```
void IRAM_ATTR ISR() {
  sentencias;
}
```

Espressif, la compañía que fabrica el microcontrolador ESP32 recomienda que en la sintaxis de la rutina que atenderá a la interrupción se use el modificador **`IRAM_ATTR`** que le indica al compilador que coloque la rutina en la memoria RAM interna en lugar de la memoria flash. De esa manera la la rutina que atenderá a la interrupción ejecuta más rápido.

Las rutinas de atención a las subrutinas, ISR, son funciones con ciertas características:

1. No tienen parámetros.
2. No regresan valores.
3. Deben ser tan pequeñas y tan rápidas como sea posible.
4. Dentro de una ISR no funciona la función `delay()`.
5. Dentro de una ISR el valor de la función `millis()` no se incrementa.
6. Se pueden perder datos recibidos por el puerto serie mientras se esté dentro de una ISR.
7. El intercambio de información entre la ISR y el resto del programa es mediante variables globales.
8. Las variables globales que se modifiquen dentro de la ISR deben calificarse con el calificador `volátil`. Esto le indica al compilador que su valor puede cambiar en cualquier momento y que no debe optimizar su acceso.

Objetivo

Implementar programas con interrupciones externas con el Microcontrolador ESP32.

Equipo y Materiales

1 Microcontrolador ESP32 DEVKIT DOIT de 30 pines
1 Base para el Microcontrolador ESP32 DEVKIT DOIT de 30 pines
1 Cable USB a micro USB
1 Protoboard
1 Resistencia de 220 Ω
1 Resistencia de 10 KΩ
1 Interruptor de botón NO
1 LED
Cables Dupont macho – hembra
Cables Dupont macho – macho

Procedimiento

Encender y apagar un LED con un interruptor de botón NO usando un Microcontrolador ESP32, sin usar interrupciones.

1. Arme el circuito de las figuras 1 y 2:

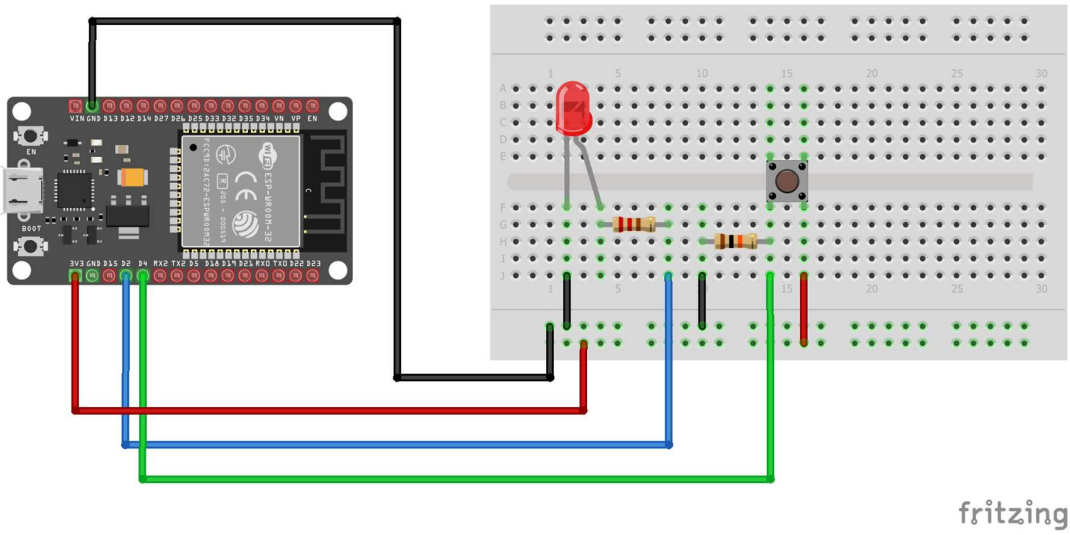

Figura 1

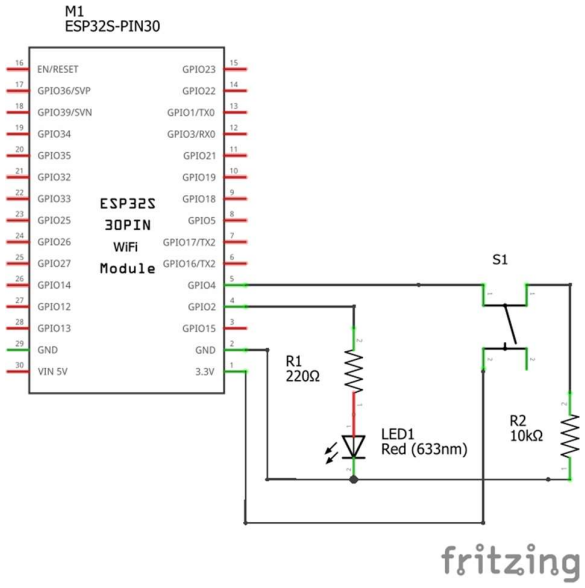

Figura 2

2. Usando el IDE 2.0 de Arduino, edite el siguiente programa. Guárdelo con el nombre de **botonLedSinInt** y compílelo.

botonLedSinInt.ino

```
/*
 * botonLedSinInt.ino
 *
 * Este programa hace que el led de status del microcontrolador
 * ESP32 DEVKIT DOIT de 30 pines o un LED conectado al pin 2,
 * cambie de estado (de apagado a prendido o viceversa) cada
 * vez que se presiona el interruptor de botón.
 *
 * Uno de los polos del interruptor de botón está conectado
 * al pin 4 y el otro al pin 3.3V. Hay una resistencia de
 * 10 KOhm entre tierra y el pin 4.
 *
 * Entre checadas del estado del boton, suponemos que el
```

```
 * microcontrolador ESP32 realiza una tarea que le lleva
 * TIEMPO_TAREA ms. Durante ese tiempo si el usuario presiona
 * el boton, no es detectado por el microcontrolador ESP32.
 *
 * No usa la funcion delay(). En lugar de ello utiliza la
 * biblioteca NoDelay. Esto permite que otro codigo ejecute al
 * mismo tiempo que se encuentra en el periodo de espera.
 *
 * Este programa utiliza la biblioteca Bounce2 para eliminar
 * el ruido en la conmutación de los interruptores
 */

#include <NoDelay.h>
#include <Bounce2.h>

// Pin a la que esta conectado el interruptor, GPIO4
const unsigned int PIN_BOTON = 4;
// Pin a la que esta conectado el LED, GPIO2
const unsigned int PIN_LED = 2;
// Duracion en ms de la tarea que ejecuta el Microcontrolador ESP32
const long TIEMPO_TAREA = 1000;
// Estado actual del pin del LED
int estadoLed = LOW;
// Velocidad de transmisión del puerto serie
const unsigned int BAUD_RATE = 115200;

// Crea una instancia de la clase noDelay
// que determina si han transcurrido TIEMPO_TAREA ms
noDelay pausa(TIEMPO_TAREA);
// Crea una instancia de la clase Bounce
Bounce debouncer = Bounce();

void setup() {
    // Establece el pin PIN_LED (GPIO2) como de salida.
    pinMode(PIN_LED,OUTPUT);
    // Establece el pin PIN_BOTON (GPIO4) como de entrada.
    pinMode(PIN_BOTON, INPUT);

    // Establece el interruptor de boton al que se le
    // eliminara el ruido
    debouncer.attach(PIN_BOTON);

    // Establece el intervalo (en ms) de espera para eliminar
    // el ruido
    debouncer.interval(25);

    // Establece la velocidad de transmisión del puerto serie al
    // valor BAUD_RATE
    Serial.begin(BAUD_RATE);

    digitalWrite(PIN_LED, estadoLed);
    Serial.println("Inicia tarea");
}

void loop() {
    // Verifica si ss termino de ejecutar la tarea
    if (pausa.update()) {
        Serial.println("Termina tarea");
      // Actualiza el estadoBoton de la instancia del objeto Bounce
      debouncer.update();
```

```
      // Cambia el estado del led solo si el estado del boton
      // cambio de HIGH a LOW
      if (debouncer.fell()) {
          estadoLed = !estadoLed;
          digitalWrite(PIN_LED, estadoLed);
      }

      Serial.println("Inicia tarea");
    }
}
```

3. Conecte el Microcontrolador ESP32 a la computadora mediante el cable USB a micro USB.
4. Cargue el programa al Microcontrolador ESP32.
5. Abra el monitor serie.
6. Verifique que el programa no siempre detecta el cambio de estado del botón.
7. Cierre el monitor serie.
8. Desconecte el Microcontrolador ESP32 de la computadora.

Encender y apagar un LED con un interruptor de botón NO usando un Microcontrolador ESP32, con interrupciones y rebote.

En el programa anterior fue difícil lograr presionar el botón cuando el programa leía el estado del botón. Para solucionar este problema haremos que sea el cambio del estado en el botón el que interrumpa al programa.

1. Usando el IDE 2.0 de Arduino, edite el siguiente programa. Guárdelo con el nombre de **botonLedIntRebote** y compílelo.

botonLedIntRebote.ino

```
/*
 * botonLedIntRebote.ino
 *
 * Este programa hace que el led de status del microcontrolador
 * ESP32 DEVKIT DOIT de 30 pines o un LED conectado al pin 2,
 * cambie de estado (de apagado a prendido o viceversa) cada
 * vez que se presiona el interruptor de botón.
 *
 * Uno de los polos del interruptor de botón está conectado
 * al pin 4 y el otro al pin 3.3V. Hay una resistencia de
 * 10 KOhm entre tierra y el pin 4.
 *
 * Entre checadas del estado del boton, suponemos que el
 * microcontrolador ESP32 realiza una tarea que le lleva
 * TIEMPO_TAREA ms.
 *
 * Si se presiona el boton se genera una interrupcion
 *
 * Debido al rebote del boton, se puede generar varias veces la
 * interrupcion del boton y el estado del LED no quedar en el
 * valor deseado.
 *
 * No usa la funcion delay(). En lugar de ello utiliza la
 * biblioteca NoDelay. Esto permite que otro codigo ejecute al
 * mismo tiempo que se encuentra en el periodo de espera.
 */
```

```
#include <NoDelay.h>

// Pin a la que esta conectado el interruptor, GPIO4
const unsigned int PIN_BOTON = 4;
// Pin a la que esta conectado el LED, GPIO2
const unsigned int PIN_LED = 2;
// Duracion en ms de la tarea que ejecuta el Microcontrolador ESP32
const long TIEMPO_TAREA = 1000;
// Estado actual del pin del LED
volatile int estadoLed = LOW;
// Velocidad de transmisión del puerto serie
const unsigned int BAUD_RATE = 115200;

// Crea una instancia de la clase noDelay
// que determina si han transcurrido TIEMPO_TAREA ms
noDelay pausa(TIEMPO_TAREA);

void IRAM_ATTR botonIsr();

void setup() {
    // Establece el pin PIN_LED (GPIO2) como de salida.
    pinMode(PIN_LED,OUTPUT);
    // Establece el pin PIN_BOTON (GPIO4) como de entrada.
    pinMode(PIN_BOTON, INPUT);

    // Establece la velocidad de transmisión del puerto serie al
    // valor BAUD_RATE
    Serial.begin(BAUD_RATE);

    // Instala la rutina de atención llamada botonIsr al evento
    // de un cambio de HIGH a LOW en el pin PIN_BOTON (GPIO4)
    attachInterrupt(PIN_BOTON, botonIsr, FALLING);

    digitalWrite(PIN_LED, estadoLed);
    Serial.println("Inicia tarea");
}

void loop() {
    // Verifica si ss termino de ejecutar la tarea
    if (pausa.update()) {
        Serial.println("Termina tarea");

        Serial.println("Inicia tarea");
    }
 }

 /*
 * Esta funcion es la rutina de atencion a la interrupcion
 * generada cuando el pin conectado al boton cambia de estado
 * de HIGH a LOW.
 */
void IRAM_ATTR botonIsr() {
    // Si se presiono el boton conectado al pin PIN_BOTON (GPIO4),
    // cambia el estado del LED de HIGH a LOW o viceversa
     estadoLed = !estadoLed;
     digitalWrite(PIN_LED, estadoLed);
}
```

2. Conecte el Microcontrolador ESP32 a la computadora mediante el cable USB a micro USB.

3. Cargue el programa al Microcontrolador ESP32.

4. Abra el monitor serie.

5. Verifique que, debido al rebote en el interruptor de botón, el led no siempre queda en el estado esperado.

6. Cierre el monitor serie.

7. Desconecte el Microcontrolador ESP32 de la computadora.

Encender y apagar un LED con un interruptor de botón NO usando un Microcontrolador ESP32, con interrupciones y sin rebote.

En el programa anterior, debido al rebote en el botón, podían ocurrir varias interrupciones y debido a eso el estado del led no necesariamente era el esperado. Para solucionar este problema haremos que la rutina de atención a la subrutina elimine esas falsas interrupciones.

1. Usando el IDE 2.0 de Arduino, edite el siguiente programa. Guárdelo con el nombre de **botonLedInt** y compílelo.

botonLedInt.ino

```
/*
 * botonLedInt.ino
 *
 * Este programa hace que el led de status del microcontrolador
 * ESP32 DEVKIT DOIT de 30 pines o un LED conectado al pin 2,
 * cambie de estado (de apagado a prendido o viceversa) cada
 * vez que se presiona el interruptor de botón.
 *
 * Uno de los polos del interruptor de botón está conectado
 * al pin 4 y el otro al pin 3.3V. Hay una resistencia de
 * 10 KOhm entre tierra y el pin 4.
 *
 * Entre checadas del estado del boton, suponemos que el
 * microcontrolador ESP32 realiza una tarea que le lleva
 * TIEMPO_TAREA ms.
 *
 * Si se presiona el boton se genera una interrupcion
 *
 * Debido al rebote del boton, se puede generar varias veces la
 * interrupcion del boton y el estado del LED no quedar en el
 * valor deseado. Para evitar eso, la rutina de atencion a la
 * interrupcion elimina las falsas interrupciones debido al
 * rebote del boton
 *
 * No usa la funcion delay(). En lugar de ello utiliza la
 * biblioteca NoDelay. Esto permite que otro codigo ejecute al
 * mismo tiempo que se encuentra en el periodo de espera.
 */

#include <NoDelay.h>

// Pin a la que esta conectado el interruptor, GPIO4
const unsigned int PIN_BOTON = 4;
// Pin a la que esta conectado el LED, GPIO2
const unsigned int PIN_LED = 2;
// Duracion en ms de la tarea que ejecuta el Microcontrolador ESP32
const long TIEMPO_TAREA = 1000;
// Intervalo en que el rebote se acaba
const long INTER_REBOTE = 250;
// Estado actual del pin del LED
```

```
volatile int estadoLed = LOW;
// Almacena la hora en la que se genero la ultima interrupcion

volatile unsigned long lapsoIntAnterior = 0;

// Velocidad de transmisión del puerto serie
const unsigned int BAUD_RATE = 115200;

// Crea una instancia de la clase noDelay
// que determina si han transcurrido TIEMPO_TAREA ms
noDelay pausa(TIEMPO_TAREA);

void IRAM_ATTR botonIsr();
void setup() {
    // Establece el pin PIN_LED (GPIO2) como de salida.
    pinMode(PIN_LED,OUTPUT);
    // Establece el pin PIN_BOTON (GPIO4) como de entrada.
    pinMode(PIN_BOTON, INPUT);

    // Establece la velocidad de transmisión del puerto serie al
    // valor BAUD_RATE
    Serial.begin(BAUD_RATE);

    // Instala la rutina de atención llamada botonIsr al evento
    // de un cambio de HIGH a LOW en el pin PIN_BOTON (GPIO4)
    attachInterrupt(PIN_BOTON, botonIsr, FALLING);

    digitalWrite(PIN_LED, estadoLed);
    Serial.println("Inicia tarea");
}

void loop() {
    // Verifica si ss termino de ejecutar la tarea
    if (pausa.update()) {
        Serial.println("Termina tarea");

        Serial.println("Inicia tarea");
    }
 }

 /*
 * Esta funcion es la rutina de atencion a la interrupcion
 * generada cuando el pin conectado al boton cambia de estado
 * de HIGH a LOW.
 */
void IRAM_ATTR botonIsr() {
    // Obtiene la hora actual
    unsigned long lapsoIntActual = millis();

    // Si transcurrio el tiempo suficiente para que se
    // elimine el ruido de conmutacion
    if(lapsoIntActual - lapsoIntAnterior >= INTER_REBOTE) {
        // cambia el estado del LED de HIGH a LOW o viceversa
        estadoLed = !estadoLed;
        digitalWrite(PIN_LED, estadoLed);
        lapsoIntAnterior = lapsoIntActual;
    }
}
```

2. Conecte el Microcontrolador ESP32 a la computadora mediante el cable USB a micro USB.

3. Cargue el programa al Microcontrolador ESP32.
4. Abra el monitor serie.
5. Verifique que el programa ejecuta correctamente.
6. Cierre el monitor serie.
7. Desconecte el Microcontrolador ESP32 de la computadora.