

LABORATORIO DE SISTEMAS EMBEBIDOS

PRÁCTICA #6

Interrupciones

1. Objetivos

- Configurar la ESP32 para utilizar sus distintos interruptores.
- Analizar las diferencias de la programación sin interruptores y con interruptores.

2. Recursos

- Wokwi
- Visual Studio Code
- ESP32
- Leds
- Display 7 segmentos

3. Marco teórico

3.1. Interrupciones

Las interrupciones son eventos que pueden interrumpir el flujo normal de ejecución del programa para atender una tarea específica de manera inmediata. Las interrupciones permiten al microcontrolador Arduino responder a eventos externos en tiempo real, como pulsaciones de botones, cambios de estado en pines de entrada, señales de temporizadores, entre otros.

Cuando se produce una interrupción, el procesador Arduino suspende temporalmente la ejecución del programa principal y ejecuta un fragmento de código conocido como "rutina de interrupción" o "manejador de interrupción". Esta rutina se encarga de atender la tarea relacionada con el evento que generó la interrupción. Una vez que se completa la rutina de interrupción, el procesador Arduino retoma la ejecución del programa principal desde el punto en que se interrumpió.

Las interrupciones son útiles en situaciones donde se requiere una respuesta rápida ante eventos externos. Por ejemplo, cuando se tiene una cortadora de madera, se colocan varios sensores capaces de detectar brazos humanos, entonces se considera que dicha cortadora tiene un programa principal donde gira a gran velocidad, pero posee un interruptor que se acciona cuando el sensor indica que un brazo se acercó mucho a la cierra, en ese momento la cortadora desciende por un orificio y evita que la persona sea lastimada. Luego cuando ya no se detecta el brazo de la persona, la cierra sube y continua con su operación principal.

3.2. Tipos de interrupciones

Interrupción externa

Se refiere a la capacidad de la placa ESP32 de detectar y responder a cambios de señales externas en los pines específicos, estos cambios pueden producirse por cualquier tipo de señal enviada a los pines seleccionados como entradas de botón, sensores o cualquier tipo de accionamiento externo. Esta característica permite que el ESP32 interrumpa su flujo de ejecución normal y ejecute una rutina de interrupción designada en respuesta a un evento externo específico.

Para utilizar la interrupción externa es necesario tener ciertas consideraciones a nivel de código como:

- Crear una función **“void IRAM_ATTR isr()”** donde se almacenará las instrucciones o código a ejecutar durante la interrupción.
- En el setup, se debe añadir obligatoriamente la función **“attachInterrupt(pin_GPIO,isr,modo)”** que será la que inicializará la interrupción con las configuraciones ingresadas. De manera que en el primer parámetro se ingresa el pin GPIO que recibirá la interrupción, en el segundo parámetro el nombre de la función que contiene las instrucciones a ejecutar durante la interrupción y en el tercer parámetro se coloca el modo de activación que puede variar entre:
 - HIGH: Ejecuta la interrupción cuando el pin detecta un alto.
 - FALLING: Ejecuta la interrupción cuando se detecta un cambio de estado alto a bajo.
 - RISING: Ejecuta la interrupción cuando se detecta un cambio de estado bajo a alto.
- En caso de que se requiera desactivar una interrupción se coloca **“detachInterrupt(pin_GPIO)”**

Interrupción interna

Se refiere a la capacidad de generar interrupciones en el microcontrolador ESP32 en respuesta a eventos internos. El ESP32 tiene un avanzado sistema de interrupciones que permite activar una rutina de interrupción cuando transcurre un tiempo específico. De manera que el programa principal ejecuta su código mientras el temporizador interno llega al tiempo de desborde definido que es cuando la sentencia de interrupción se ejecuta y cuando este proceso culmina, se continúa recorriendo el código principal desde la línea donde se detuvo.

Para utilizar la interrupción interna es necesario tener ciertas consideraciones a nivel de código como:

- Crear una función **“void IRAM_ATTR isr()”** donde se almacenará las instrucciones o código a ejecutar durante la interrupción.
- En el setup, se debe añadir obligatoriamente la función **“attachInterrupt(temporizador,&isr,estado)”** que será la que inicializará la interrupción con las configuraciones ingresadas. De manera que en el

primer parámetro se ingresa el temporizador definido, en el segundo parámetro se tiene la dirección del puntero que contiene el código de la interrupción y en el último parámetro se coloca el estado de activación como true(activado) y false(desactivado).

- En la función “**timerAlarmWrite(timer, TIMER_INTERVAL_MS * 1000, true)**” se define el tiempo en que la interrupción será ejecutada, siendo que el segundo parámetro corresponde al tiempo mencionado.
- Por último, habilita la generación de interrupciones por alarma en el temporizador especificado con “**timerAlarmEnable(timer)**”

3.3. Material de apoyo teórico

Interrupción externa

- https://www.youtube.com/watch?v=up-goNfJ_EY&t=938s

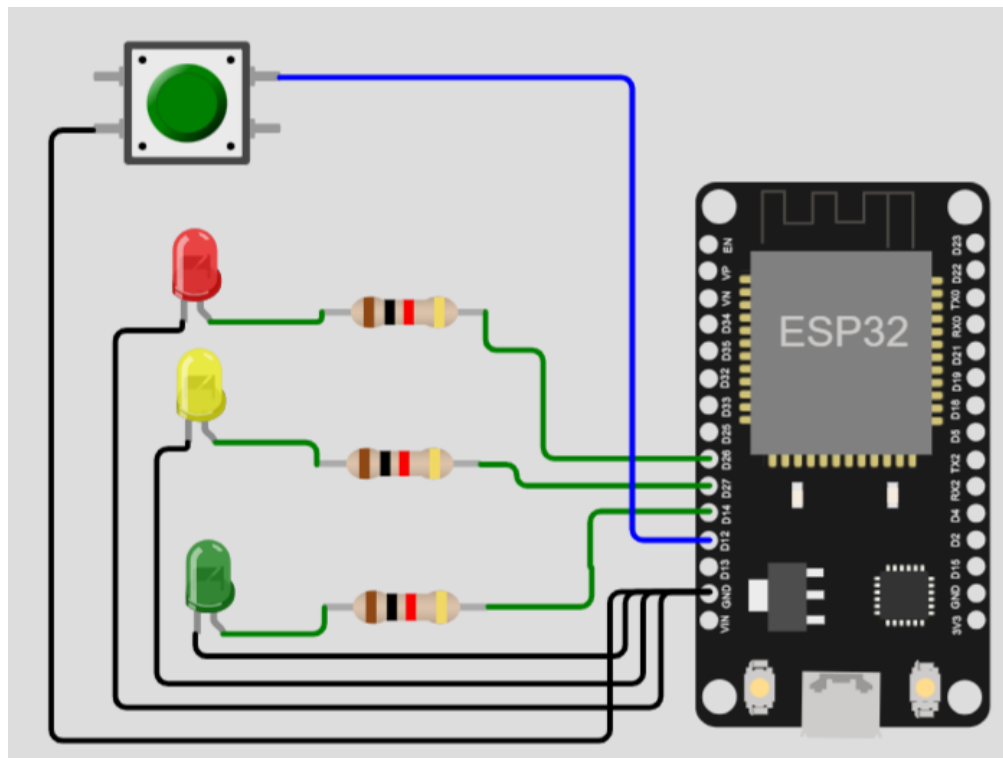
Interrupción interna

- <https://descubrearduino.com/interrupciones-esp32-gpio/>

4. Actividad por desarrollar

4.1. Semáforo con interrupción externa

Diagrama esquemático



Enlace de simulación: <https://wokwi.com/projects/367324510944280577>

Código utilizado

- Se definen los pines de los 3 leds y de pulsador.

```
int luz_verde = 14;
int luz_amarilla=27;
int luz_roja=26;
int pulsador=12;
```

- Se crea la función isr con extensión IRAM_ATTR para colocar el código en la memoria RAM interna, de lo contrario se almacena en la memoria flash de la ESP32 que carece de velocidad con respecto a la memoria RAM. Entonces, es recomendable utilizar la memoria RAM, ya que una interrupción debe ejecutarse lo más rápido posible. En este caso, durante la interrupción los tres leds se encenderán al mismo tiempo.

```
void IRAM_ATTR isr(){ //Funcion de la interrupcion
    digitalWrite(luz_verde,HIGH);
    digitalWrite(luz_amarilla,HIGH);
    digitalWrite(luz_roja,HIGH);
}
```

- En el setup configura los pines de los leds en modo de salida y el pulsador en modo de entrada con PULLUP. Además, en la última línea se tiene la función attachInterrupt que será la que inicializará la interrupción con las configuraciones ingresadas. En este caso, se indica que el pin digital “pulsador” será una entrada de interrupción externa que cuando reciba un “Falling” es decir un cambio de 1 a 0 en la entrada, se ejecutarán las instrucciones contenidas en la función isr().

```
void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
    pinMode(luz_verde,OUTPUT);
    pinMode(luz_amarilla,OUTPUT);
    pinMode(luz_roja,OUTPUT);
    pinMode(pulsador,INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(pulsador),isr,FALLING);
}
```

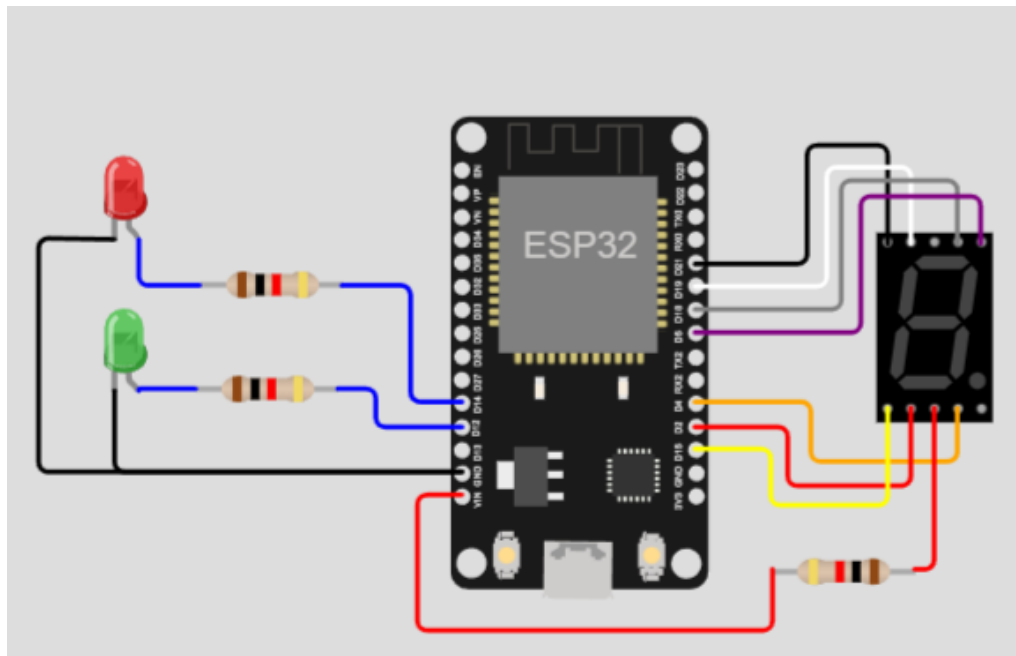
- Por otra parte, en el código principal se estará ejecutando un código de encendido de los tres leds encendiendo uno a la vez cada segundo.

```
void loop() {
    // put your main code here, to run repeatedly:
    Serial.println(digitalRead(pulsador));
    digitalWrite(luz_roja,HIGH);
    digitalWrite(luz_amarilla,LOW);
    digitalWrite(luz_verde,LOW);
    delay(1000);

    digitalWrite(luz_roja,LOW);
    digitalWrite(luz_amarilla,HIGH);
    digitalWrite(luz_verde,LOW);
    delay(1000);
    digitalWrite(luz_roja,LOW);
    digitalWrite(luz_amarilla,LOW);
    digitalWrite(luz_verde,HIGH);
    delay(1000);
}
```

4.2. Sincronización de leds y conteo con interrupción interna

Diagrama esquemático



Enlace de simulación: <https://wokwi.com/projects/367492234087362561>

Código utilizado

- Se definen los leds que controlarán al display de ánodo común y en arreglos diferentes se define los estados de alto y bajo de las salidas para formar cada uno de los números en el display de 7 segmentos.

```
int LEDs[] = {18,5,4,2,15,19,21}; // Pines del display anodo comun
```

```
// Estado de pines para formar los numeros del display
```

```
int cero[] = {0,0,0,0,0,0,1};
int uno[] = {1,0,0,1,1,1,1};
int dos[] = {0,0,1,0,0,1,0};
int tres[] = {0,0,0,0,1,1,0};
int cuatro[] = {1,0,0,1,1,0,0};
int cinco[] = {0,1,0,0,1,0,0};
int seis[] = {0,1,0,0,0,0,0};
int siete[] = {0,0,0,1,1,1,1};
int ocho[] = {0,0,0,0,0,0,0};
int nueve[] = {0,0,0,0,1,0,0};
```

- Se define el tiempo de demora para el encendido del led verde que es comandado por el código principal. Por otra parte, se define el intervalo de tiempo de la interrupción, luego se define el pre-escalador a utilizar y la asignación de pines para el led verde y para el led rojo. Además, se define el `hw_timer_t *timer=NULL` para almacenar el valor de tiempo del temporizador y por último se desbloquea el `portMUX_TYPE timerMux` para permitir la modificación de variables que se encuentren tanto en el programa principal como en el programa de la interrupción.

```
int demora=3000;
```

```
#define TIMER_INTERVAL_MS 3000 // Intervalo de tiempo para la interrupcion
```

```
#define TIMER_DIVIDER 80 // Divisor para una frecuencia de 1 MHz
```

```
#define LED_PIN 14 //Define el led de la interrupcion
```

```
#define LED_VERDE 12 // Define el led del programa principal
```

```
hw_timer_t *timer = NULL; // Define el tiempo del temporizador
```

```
//sincroniza el bucle principal y la ISR cuando se modifique una variable compartida
```

```
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
```

- Se establece las variables `ledState` y estados correspondientes a los cambios de estado del led rojo activado en la interrupción y al led verde activado en el programa principal respectivamente.

```
volatile bool ledState = false;
```

```
volatile bool estado = false;
```

- Se define la función `timerInterrupt` de tipo `IRAM_ATTR` donde se almacenará el código de la interrupción, pero hay que considerar el uso de `portENTER_CRITICAL_ISR` y `portEXIT_CRITICAL_ISR` que se utilizan para habilitar y deshabilitar las interrupciones en una región crítica dentro de un manejador de interrupción. Esto se realiza para evitar que otra interrupción ocurra al mismo tiempo que la interrupción actual, con lo cual se limita al sistema a que solo ejecute una interrupción a la vez. Entonces con `portENTER_CRITICAL_ISR` se deshabilitan las interrupciones en una región

crítica, pero con portEXIT_CRITICAL_ISR se habilitan nuevamente las interrupciones después de haber finalizado la región crítica.

```
// Se define la funcion de la interrupcion
void IRAM_ATTR timerInterrupt() {
    portENTER_CRITICAL_ISR(&timerMux); //Declara el inicio de la interrupcion
    ledState = !ledState;
    digitalWrite(LED_PIN, ledState);
    portEXIT_CRITICAL_ISR(&timerMux); // Declara que la interrupcion termino
}
```

- En el setup se define como salida a los pines de ambos leds al igual que se utiliza un ciclo for para definir a los 7 pines del display de 7 segmentos como salidas. Por otra parte, se configura la interrupción de la siguiente manera:
 - Se define el timer con “timerBegin(timer,pre-escalador,autorecarga)” que recibe:
 - Timer: Se define el número del timer a utilizar, estos pueden ser 0,1,2,3.
 - Pre-escalador: Especifica el divisor de frecuencia.
 - Autorecarga: Si es true, se habilita que el temporizador se resetee automáticamente, si fuera false este debería tener un medio de reseteo manual del temporizador.
 - Se define el “timerAttachInterrupt(timer,&timerInterrupt,estado)” que recibe:
 - Timer: Se define el número del timer a utilizar, estos pueden ser 0,1,2,3.
 - ,&timerInterrupt: Especifica el nombre de la función de interrupción que deseas adjuntar al temporizador.
 - Estado: Cuando es true se habilita el modo de interrupción de nivel alto para el temporizador especificado. Esto significa que la interrupción se generará cuando el temporizador alcance su valor máximo y se reinicie automáticamente.
 - Se define timerAlarmWrite(timer, intervalo, autoconfiguración) que recibe:
 - Timer: Se define el número del timer a utilizar, estos pueden ser 0,1,2,3.
 - Intervalo: Este parámetro especifica el valor de comparación, en microsegundos, para la alarma del temporizador. En este caso, TIMER_INTERVAL_MS es una constante que representa el intervalo de tiempo deseado en milisegundos. Al multiplicarlo por 1000, se convierte a segundos. El temporizador generará una interrupción cuando alcance este valor.
 - autoconfiguración: Indica si se debe configurar la alarma para que se repita automáticamente o no. Si se pasa true, la alarma se configurará en modo repetitivo, lo que significa que generará una interrupción periódica

cada vez que se alcance el valor de comparación. Si se pasa false, la alarma solo generará una interrupción una vez y deberá ser reconfigurada manualmente para generar más interrupciones.

- Se define `timerAlarmEnable(timer)` que habilita la alarma del temporizador especificado. Esto significa que el temporizador comenzará a contar y generará una interrupción cuando alcance el valor de comparación que se estableció previamente utilizando `timerAlarmWrite`.

```
void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(LED_VERDE, OUTPUT);
    for (int i = 0; i < 7; i++) pinMode(LEDs[i], OUTPUT);

    timer = timerBegin(1, TIMER_DIVIDER, true); // Inicializa el timer
    timerAttachInterrupt(timer, &timerInterrupt, true); // Declara la interrupcion
    // Define el tiempo en que se activa la interrupcion
    timerAlarmWrite(timer, TIMER_INTERVAL_MS * 1000, true);
    timerAlarmEnable(timer); // Habilita la alarma del timer
}
```

- Finalmente, el código principal realizará un conteo del 1 al 9 en el display de 7 segmentos, pero al mismo tiempo encenderá y apagará el led verde en cada cambio numérico. En esta etapa se considera que el led rojo cambiará su estado cada 3 segundos en la interrupción mientras que el led verde cambiará su estado cada 3 segundos en el programa principal.

```
void loop() {
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], uno[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], dos[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], tres[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], cuatro[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], cinco[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], seis[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], siete[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], ocho[i]);
    delay(demora);
    estado = !estado;
    digitalWrite(LED_VERDE, estado);
    for (int i = 0; i < 7; i++) digitalWrite(LEDs[i], nueve[i]);
}
```



1. Hacer que un led, parpadee en interrupción cada 2 segundos, el segundo led parpadee en otra interrupción cada 3 segundos, y en el loop que siga parpadeando el tercer led cada 1 segundo además de mostrarse el contador en el display de 7seg.
2. Hacer que 5 leds parpadeen uno tras otro en orden ascendente, y con interrupción externa cuando se detecta un RISSING, el parpadeo se vuelve ascendente en su próximo ciclo. Usar botón PULLUP, se debe realizar el cambio al momento de soltar el botón no al momento de presionarlo.
3. Hacer que 5 leds parpadeen uno tras otro en orden ascendente, y con interrupción externa cuando se detecta un FALLING, el parpadeo se vuelve ascendente en su próximo ciclo. Usar botón PULDOWN, se debe realizar el cambio al momento de soltar el botón no al momento de presionarlo.