Interrupciones

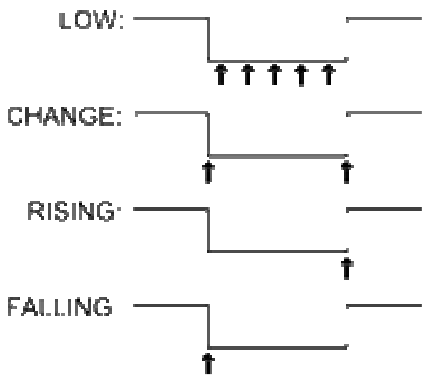
Una petición de interrupción IRS ("Interrupt Request Service") es una señal que se origina en un dispositivo hardware (por ejemplo, un periférico), para indicar al procesador que algo requiere su atención inmediata; se solicita al procesador que suspenda lo que está haciendo para atender la petición.

Las interrupciones juegan un papel fundamental, en especial en la operación de dispositivos E/S. Sin ellas el sistema debería chequear constantemente los dispositivos para comprobar su actividad, pero las interrupciones permiten que los dispositivos puedan permanecer en silencio hasta el momento que requieren atención del procesador. ¿Podría figurarse un sistema telefónico donde hubiera que levantar periódicamente el auricular para comprobar si alguien nos llama?

En el Arduino UNO, hay dos interrupciones externas disponibles. Estos se encuentran en los pines digitales 2 y 3 de la placa Arduino UNO, aunque estas interrupciones se llaman 0 y 1.

Las interrupciones externas (INT0 e INT1) de arduino pueden ejecutarse porque la señal de entrada de un pin está:

- Está en nivel bajo (tiene el valor de 0) LOW
- Ha cambiado de situación (tanto para flanco de subida como de bajada) CHANGE
- Cambia a nivel alto (flanco de subida) RISING
- Cambia a nivel bajo (flanco de bajada) FALLING



Interrupciones

En este ejemplo veremos como actúa la interrupción externa (int0) del pin 2. Cuando pulsemos el pulsador el led se apagará y cuando lo soltemos se encenderá. Para poder apreciar la diferencia se programará el ejercicio 2 veces. Al principio se hará sin interrupciones y tras ver como responde se volverá a hacer el mismo ejercicio con la interrupción arriba mencionada.

Hardware Necesario

- Placa Arduino
- Pulsador
- LED
- Resistecia de 330 ohms
- Resistecia de 10K ohms

Circuito

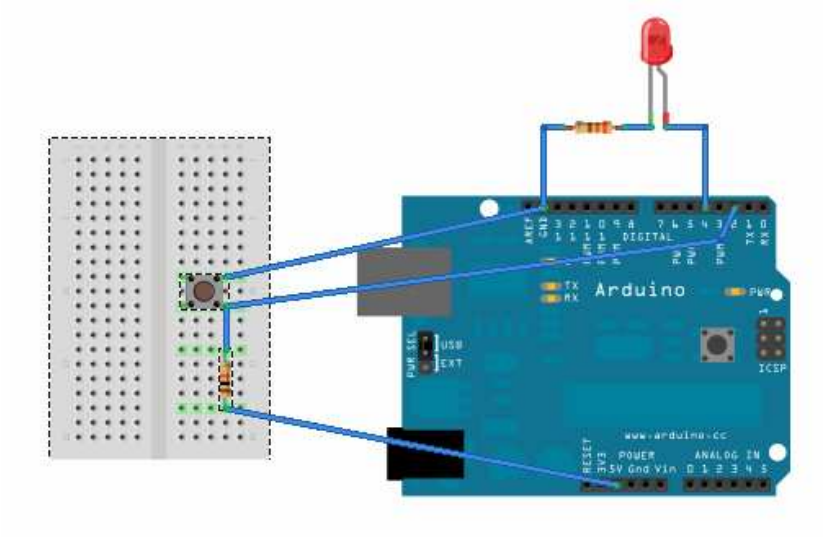
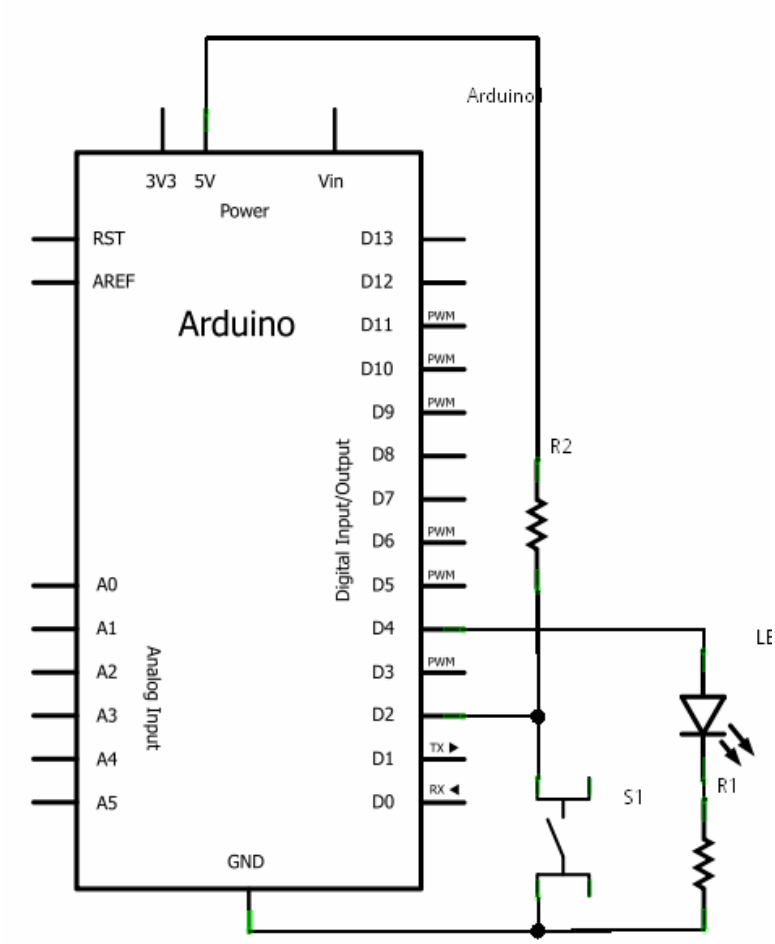


imagen desarrollada utilizando Fritzing. Para mas circuitos de ejemplo, visita la página del proyecto Fritzing

Conectar el pin de 5V a uno de los extremos de la resistencia de 10K ohm. El otro extremo de la resistencia hay que conectarlo a el pin digital 2 (INT0) y a uno de los extremos del pulsador. El otro extremo del pulsador hay que conectarlo a masa.

El pin digital 4 hay que conectarlo a la pata larga del diodo (el ánodo) y la corta mediante una resistencia de 330 ohm en serie a masa.

Esquema



Code

Cuando el pulsador está en estado de reposo el led está apagado, pero al pulsar el pulsador se apaga. Al final del programa hay un bucle donde pierde el tiempo ya que se supone que hace alguna función compleja que requiere mucho tiempo. Este hecho produce que cuando el pulsador cambia de estado, el led no lo hará a la vez. Normalmente habrá que mantener cierto tiempo el pulsador en su nueva situación para que el programa salga del bucle complejo y cambie la situación del led.

```
int pbIn = 2;          // Digital input on pin 2
int ledOut = 4;        // The output LED pin
int state = LOW;       // The input state

void setup()
{
 // Set up the digital Pin 2 to an Input and Pin 4 to an Output
  pinMode(pbIn, INPUT);
  pinMode(ledOut, OUTPUT);
}


void loop()
{
  state = digitalRead(pbIn);      //Read the button
  digitalWrite(ledOut, state);    //write the LED state

  //Simulate a long running process or complex task
  for (int i = 0; i < 100; i++)
```

```
  {
     // do nothing but waste some time
     delay(10);
  }
}
```

En el ejercicio anterior se ha podido apreciar que ante unos cambios de estado del pulsador muy rápidos el programa no detectaba este hecho, por lo que el led no cambiaba de estado. Este problema se soluciona mediante una interrupción que lee el estado del pulsador (pin 2). La interrupción se activa mediante la función attachInterrupt()

```
attachInterrupt(pbIn, stateChange, CHANGE);
```

La función attachInterrupt () recibe 3 argumentos: La interrupción que se activará, la función que se ejecutará al activarse la interrupción, el estado de la señal por la que se activará la interrupción.

```
int pbIn = 0;          // Digital input on pin 2
int ledOut = 4;        // The output LED pin
volatile int state = HIGH;       // The input state

void setup()
{
 // Set up the digital pin 2 to an Interrupt and Pin 4 to an Output
  pinMode(ledOut, OUTPUT);
  digitalWrite(ledOut, state);    //Led On

  //Attach the interrupt to the input pin and monitor for ANY Change
  attachInterrupt(pbIn, stateChange, CHANGE);
}

void loop()
{
  //Simulate a long running process or complex task
  for (int i = 0; i < 100; i++)
  {
    // do nothing but waste some time
    delay(10);
  }
}

void stateChange()
{
  state = !state;
  digitalWrite(ledOut, state);
}
```

El circuito electrónico puede crear rebotes que engañen a la interrupción y a veces no responda adecuadamente. Esto se soluciona sustituyendo el pulsador por un circuito antirebotes.

Nota

Dentro de la función enlazada, la función delay() no funciona y el valor devuelto por la función millis() no se incrementará. Los datos serie recibidos en el transcurso de esta interrupción pueden perderse. Deberías declarar como volátil cualquier variable que modifiques dentro de la función.

attachInterrupt(interrupcion, funcion, modo)

Descripción

Especifica la función a la que invocar cuando se produce una interrupción externa. Reemplaza cualquier función previa que estuviera enlazada a la interrupción. La mayoría de las placas Arduino tienen dos interrupciones externas: Las número 0 (en el pin digital 2) y la 1 (en el pin digital 3). La Arduino Mega tiene otras cuatro: Las número 2 (pin 21), 3 (pin 20), 4 (pin 19) y 5 (pin 18).

Parámetros

interrupcion: el número de la interrupción (*int*)
funcion: la función a la que invocar cuando la interrupción tiene lugar; esta función no debe tener parámetros ni devolver nada. Esta función es a veces referenciada como *rutina de interrupción de servicio*
modo define cuando la interrupción debe ser disparada. Hay cuatro constantes predefinidas como valores válidos:

- **LOW** para disparar la interrupción en cualquier momento que el pin se encuentre a valor bajo(LOW).
- **CHANGE** para disparar la interrupción en culquier momento que el pin cambie de valor.
- **RISING** para disparar la interrupción cuando el pin pase de valos alto (HIGH) a bajo (LOW).
- **FALLING** para cuando el pin cambie de valor alto (HIGH) a bajo (LOW)

Retorno

Ninguno

Nota

Dentro de la función enlazada, la función delay() no funciona y el valor devuelto por la función millis() no se incrementará. Los datos serie recibidos en el transcurso de esta interrupción pueden perderse. Deberías declarar como volátil cualquier variable que modifiques dentro de la función.

Usando las interrupciones

Las interrupciones son útiles para hacer que las cosas sucedan automáticamente en programas para microcontroladores, y puedan ayudar a resolver problemas de temporización. Una buena tarea en la que utilizar interrupciones podría ser leer un encoder rotacional, monitorizando la entrada del usuario.

Si quisieras asegurarte de que un programa siempre captura los pulsos de un encoder rotacional, sin perder nunca un pulso, sería muy complicado escribir un programa que haga otra cosa, puesto que el programa debería estar constantemente consultando las líneas del sensor del encoder, de forma que capture los pulsos cuando tienen lugar. Otros sensores tienen un interfaz dinámico similar, como intentar leer un sensor de sonido que intenta capturar un click, o un sensor de ranuras por infrarrojos (fotointerruptor) tratando de capturar el paso de una moneda. En todas estas situaciones, usar una interrupción, libera al microcontrolador de realizar otras tareas sin echar en falta el "timbre".

Ejemplo

```
int pin = 13;
volatile int estado = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, parpadeo, CHANGE);
}

void loop()
{
  digitalWrite(pin, estado);
}

void parpadeo()
{
  estado = !estado;
}
```

noInterrupts()

Descripción

Desactiva las interrupciones (pueden reactivarse usando interrupts()). Las interrupciones permiten que las operaciones importantes se realicen de forma transparente y están activadas por defecto. Algunas funciones no funcionarán y los datos que se reciban serán ignorados mientras que las interrupciones estén desactivadas. Las interrupciones pueden perturbar ligeramente el tiempo de temporizado, sin embargo puede que sea necesario desactivarlas para alguna parte crítica del código.

Parámetros

Ninguno

Devuelve

Nada

Ejemplo

```
void setup() {}

void loop()
{
  noInterrupts();
  // Código dependiente críticamente del tiempo
  interrupts();
  // resto del código aquí
}
```

interrupts()

Descripción

Activa las interrupciones (después de haberlas desactivado con noInterrupts(). Las interrupciones permiten que se ejecuten ciertas tareas en segundo plano que están activadas por defecto. Algunas funciones no funcionarán correctamente mientras las interrupciones estén desactivadas y la comunicación entrante puede ser ignorada. Las interrupciones pueden perturbar ligeramente la temporización en el código y deben ser desactivadas sólo para partes particularmente críticas del código.

Parámetros

Ninguno

Devuelve

No devuelve nada

Ejemplo

```
void setup() {}

void loop()
{
  noInterrupts();
```

```
  // código crítico y sensible al tiempo
  interrupts();
  // otro código
}
```

detachInterrupt(interrupt)

Descripción

Apaga la interrupción dada.

Parámetros

interrupt: el número de interrupción a invalidar (0 o 1).