Tutorial_14.- Arduino_Interrupciones del microcontrolador

Contenido

Material necesario

- Placa Arduino
- Cable micro USB
- Placa Protoboard
- Cables Jumper-Wire
- Componentes:
  - Pulsadores
  - Resistencias

IMPORTANTE:

Comprobaremos que nuestra placa **Arduino** está **desconectada** y sin energía, puesto de no ser así podría dañarse tanto la placa, como el equipo. Una vez hemos realizado esta comprobación, pasaremos a realizar el montaje.

1. Contenido teórico

Contenido teórico

Qué es una interrupción

Una interrupción es una señal capaz de parar la actividad principal del procesador para ejecutar otra función distinta.

La interrupción **puede ser**:

1) Externa: El microcontrolador cuenta con un pin que recibe la petición de interrupcion externa por algún periférico externo. Por ejemplo apretar un pulsador.
2) Interna: Si se produce un evento concreto en algún recurso interno del microcontrolador, como un temporizador, un registro, etc. Por ejemplo un temporizador termina su cuenta.

Una vez que se ha activado la interrupción, la rutina principal del procesador (En Arduino la función "loop()") se pausa y se ejecuta otra función conocida como **Rutina de Interrupción de Servicio**, o ISR por sus siglas en Inglés, *Interrupt Service Routine*. Una vez que la ISR ha finalizado, el programa vuelve a lo que estaba haciendo antes.

Las interrupciones pueden desencadenarse por distintos **tipos de evento**:

- LOW: La interrupción se activa cuando el pin toma un valor LOW o bajo (0 V)
- CHANGE: La interrupción se activa cuando el pin cambia de valor, detecta un flanco. Tanto si pasa de LOW a HIGH como de HIGH a LOW.
- RISING: Se activa únicamente cuando el valor del pin pasa de LOW a HIGH (Flanco positivo o de subida).
- FALLING: Es el caso opuesto al modo RISING. Se activa la interrupción cuando el valor pasa de HIGH a LOW (Flanco negativo o de bajada).
- HIGH: Este modo solo está disponible en la placa DUE y funciona de manera opuesta al modo LOW, es decir, se activa cuando el valor de voltaje del pin es alto.

Número de interrupciones y pines asociados según modelo de Arduino

Cada modelo de placa Arduino cuenta con un microcontrolador diferente y presentan, por tanto, diferencias en sus interrupciones. La siguiente tabla especifica el número de interrupciones disponibles en cada modelo asi como el numero de pin asociado a cada una de las interrupciones.

Placa Arduino	Pin de cada interrupción					
	Interrupcion0	Interrupcion1	Interrupcion2	Interrupcion3	Interrupcion4	Interrupcion5
UNO	2	3	-	-	-	
Leonardo	3	2	0	1	7	-
Mega	2	3	21	20	19	18

Rutina de servicio de la interrupción (ISR)

Es el código asociado a las interrupciones, el conjunto de líneas de código que se ejecutarán cuando se active la interrupción.

Su sintaxis es similar a la de cualquier otra función de Arduino como "void setup()" o "void loop()". Sin embargo, tiene una serie de restricciones y características especiales, a saber:

- **No** puede tener **parámetros de entrada**, es decir, no puede ser una función del tipo void my_interrupt (int b).
- No puede devolver ningún valor, por tanto debe ser una función **de tipo void**.
- Si la RSI modifica alguna **variable** que es necesario usar en el resto del programa, hay que definirla como **volatile**.

Usando las interrupciones

Pasos para utilizar una interrupción:

1. Tener claro el **número de interrupción** a utilizar y conectar a su pin asociado el periférico que provocará la interrupción. Si se está usando Arduino Leonardo y se quiere trabajar con la interrupción 0, conectar,por ejemplo, un pulsador al pin número 3.
2. Especificar qué función se llamará en caso de que se produzca la interrupción, ISR y cual es el evento que la desencadena. Para ello se escribe la función **attachInterrupt** dentro de setup(), su sintaxis es

```
attachInterrupt(interrupt, ISR, mode);
```

Los argumentos de la función attachInterrupt son:

- **Interrupt**: Debe especificarse el numero de interrupción asociado al pin donde esté conectado el periferico que interrumpe, p.e. el pulsador.
- **ISR**: nombre de la función, ISR, que se llamará en caso de que se produzca la interrupción.
- **Mode**: Define el tipo de evento que lanzará la ISR , es decir, qué tiene que pasar para que se ejecute la interrupción. Los diferentes modos ya se han visto y son:
  - LOW: se ejecuta siempre que el valor en el pin sea 0.
  - CHANGE: se ejecuta siempre que se produzca un cambio, ya sea de 0 a 1 ó de 1 a 0.
  - RISING: se ejecuta mientras el valor va de 0 a 1 (o de 0 a 5 Voltios).
  - FALLING: se ejecuta mientras el valor va de 1 a 0 (o de 5 a 0 Voltios).

```
attachInterrupt(0, ISR_0, FALLING);
```

3. Definir la función ISR.

```
void ISR_0(){
    numInterrupt++;
}
```

4. Este paso debe realizarse en el caso de que la ISR modifique alguna variable necesaria también para el programa principal. Entoces la variable deberá ser global y declararse en el programa principal como **volatile.**

```
volatile int numInterrupt = 0;
```

A tener en cuenta...

- Las ISR deben ser lo más cortas posibles al objeto de detener el menor tiempo posible el programa principal.
- Sólo se puede ejecutar una interrupción cada vez, esto es, si se está ejecutando una interrupción y se activa una segunda, no tendrá efecto hasta que la primera termine.
- Las **funciones de tiempo** definidas en "loop()", programa principal, dejan de funcionar cuando se activa una interrupción.
    - Delay() conjuntamente con las interrupciones
    - La función millis() no incrementa su valor durante el transcurso de una interrupción.
    - **SI** se puede utilizar la función delayMicroseconds().
- Si se está recibiendo o transmitiendo datos (Serial.begin()) cuando se activa una interrupción, es posible que se pierdan.
- Puedes deshabilitar las interrupciones en cualquier momento utilizando la instrucción **detachInterrupt()**.
- Existen instrucciones que permiten desactivar las interrupciones en general o alguna en concreto. Pueden usarse si un trozo de código presente en "loop()" debe ser "insensible" a las interrupciones.
    - noInterrupts(): desactiva la ejecución de interrupciones hasta nueva orden.
    - Interrupts(): reinicia las interrupciones definidas con attachInterrupt().
    - detachInterrupt(num Interrupt): anula la interrupción indicada.

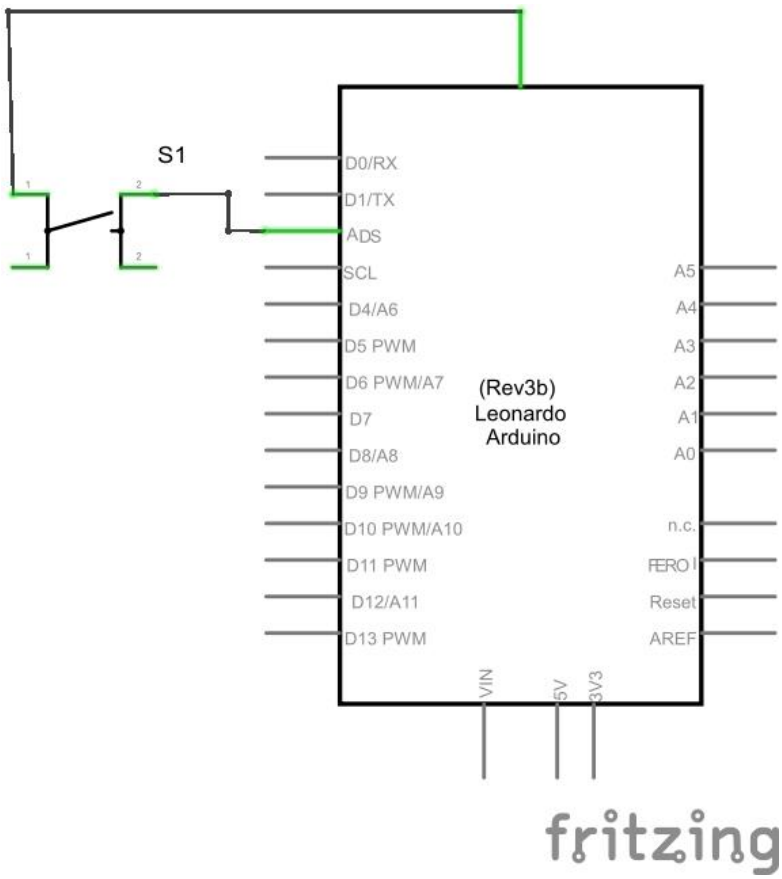
Proyecto14_01. Provocando una interrupción en ARduino

El siguiente proyecto imprime cuando se provoca una interrupción al apretar el pulsador, se muestra en pantalla el número de veces que se ha activado la interrupción (se ha apretado el pulsador).

Esquema

El pulsador se conecta mediante resistencia de Pull-up interna a 5V de modo que HIGH será el valor por defecto en el terminal 2 de Arduino (pin de la interrupción). Cuando se apriete el pulsador LOW será el valor de entrada en el pin 2, ya que el otro terminal del pulsador está conectado a tierra.

Asi el evento que produce la interrupción es un flanco negativo, un cambio de HIGH a LOW, modo FALLING.



Enlaces sobre interrupciones

https://www.sparkfun.com/tutorials/326

Código

```
/*  Variable volatile
To modify the way in which the compiler treats the variable.
This tells the compiler that such variables might change at any time,
and thus the compiler must reload the variable from the RAM each time
you reference,
it won´t be sotored in a processor register.
*/

volatile int numInterrupt = 0;

void setup() {
  pinMode(2,INPUT_PULLUP);
  Serial.begin(9600);

  // Sintaxis interrupciones:
  // attachInterrupt(numero_interrupt,función_a_ejecutar,modo);
  // Modos LOW,CHANGE,RISING,FALLING


  //Pin 3 goes from '1' to '0' when pushbuttom is pressed
  attachInterrupt(0, ISR_0, FALLING);
}

void loop() {
  Serial.println(numInterrupt);
  delay (100);
 }

void ISR_0() {
  numInterrupt++;
 }
```

Ejercicios

1. Ensambla el circuito oportuno y escribe el cógio necesario para encender/apagar un LED cada vez que actúes sobre un pulsador. En esta ocasión, usa interrupciones en lugar de la técnica de “polling” empleada en el tutorial 01.