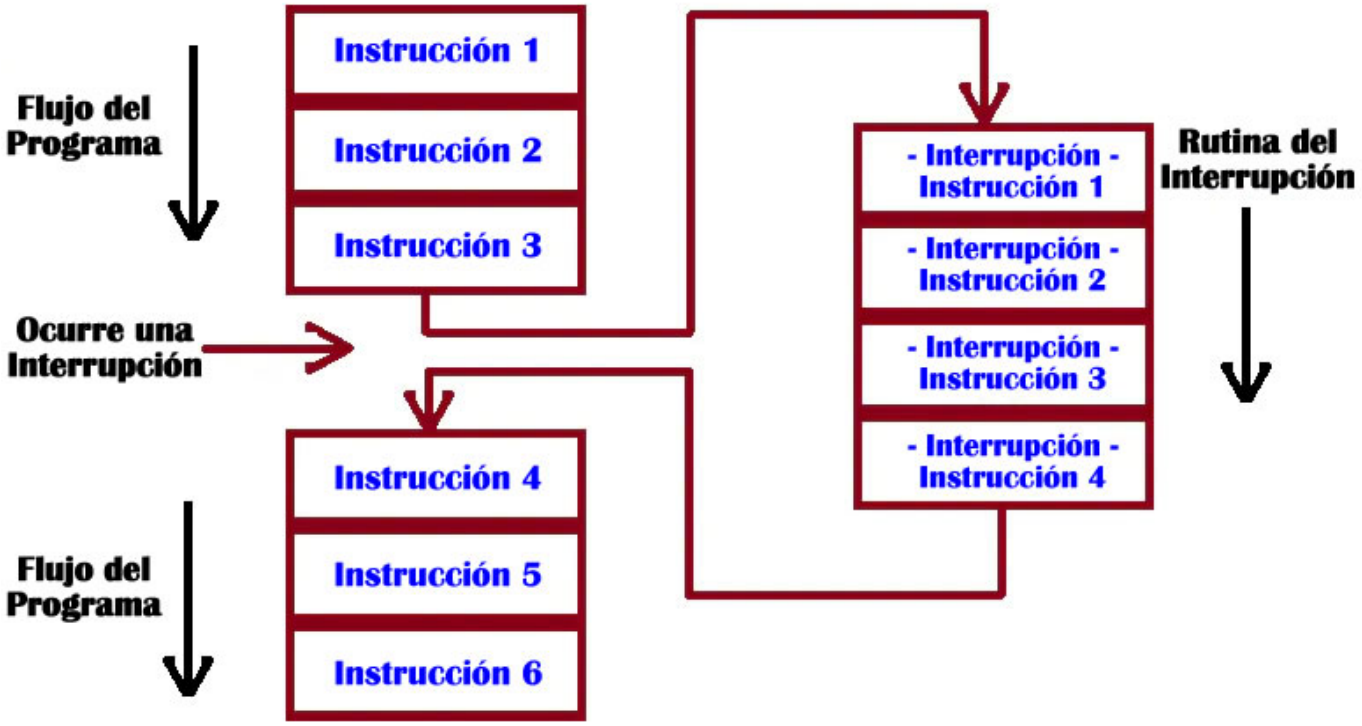
Interrupciones en Arduino

Flujo de ejecución de una interrupción

El funcionamiento es semejante a cuando llamamos a una subrutina desde una línea del programa. Solo que en este caso no existe tal linea sino que la subrutina de interrupción se ejecutará en cualquier instante del programa, cuando se cumplan unas condiciones muy precisas.



Habilitación y configuración de la interrupción

Esta instrucción se pone en `void setup(){}` para habilitar y definir la interrupción:

```
attachInterrupt ( digitalPinToInterrupt ( P ), Funcion, RISING );
```

Veamos sus parámetros a continuación:

- `digitalPinToInterrupt` → Si damos el pin, el programa se encarga de buscar la interrupción.
- P → Es el pin que detecta la interrupción. **En el Arduino Nano es el 2**.
- `Funcion` → Función asociada a la interrupción. Se ejecuta al producirse la interrupción. En otro lugar del programa deberemos declararla `void Funcion(){}`.
- `flanco de activación`
  - `RISING` → Detecta el **flanco de subida** en el pin P, llamando a la interrupción.
  - `FALLING` → Si se pone en lugar de `RISING`, llama a la interrupción en el **flanco de bajada** de la señal de entrada al pin P.

Función ISR

- La función asociada a una interrupción se denomina **ISR** (*Interruption Service Routines*) y, por definición, tiene que ser una función que no recibe nada y no devuelva nada.
- Dos ISR no pueden ejecutarse de forma simultánea. En caso de dispararse otra interrupción mientras se ejecuta una ISR, la función ISR se ejecuta una a continuación de otra.
- **La ISR, cuanto más corta mejor**. Al diseñar una ISR debemos mantener el menor tiempo de ejecución posible, dado que mientras se esté ejecutando el bucle principal y todo el resto de funciones se encuentran detenidas.
    - Imaginemos, por ejemplo, que el programa principal ha sido interrumpido mientras un motor acercaba un brazo para coger un objeto. Una interrupción larga podría hacer que el brazo no para a tiempo, tirando o dañando el objeto.
- Frecuentemente la función de la ISR se limitará a activar un flag, incrementar un contador, o modificar una variable. Esta modificación será atendida posteriormente en el hilo principal, cuando sea oportuno.
- No emplear en una ISR un proceso que consuma tiempo. Esto incluye cálculos complejos, comunicación (serial, I2C y SPI) y, en la medida de lo posible, cambio de entradas o salidas tanto digitales como analógicas.

Variables volátiles

- **Las variables de la ISR como volátiles**. Para poder modificar una variable externa a la ISR dentro de la misma debemos declararla como `volatile`.
- El indicador `volatile` indica al compilador que la variable tiene que ser consultada siempre antes de ser usada, dado que puede haber sido modificada de forma ajena al flujo normal del programa (lo que, precisamente, hace una interrupción).
- Al indicar una variable como `volatile` el compilador desactiva ciertas optimizaciones, lo que supone una pérdida de eficiencia. Por tanto, sólo debemos marcar como `volatile` las variables que realmente lo requieran, es decir, las que se usan tanto en el bucle principal como dentro de la ISR.

Efectos de la interrupción y la medición del tiempo

Las interrupciones tienen efectos en la medición del tiempo de Arduino, tanto fuera como dentro de la ISR, porque Arduino emplea interrupciones de tipo Timer para actualizar la medición del tiempo.

Efectos fuera de la ISR

- Durante la ejecución de una interrupción Arduino no actualiza el valor de la función millis y micros. Es decir, el tiempo de ejecución de la ISR no se contabiliza y Arduino tiene un desfase en la medición del tiempo.
- Si un programa tiene muchas interrupciones y estas suponen un alto tiempo de ejecución, la medida del tiempo de Arduino puede quedar muy distorsionada respecto a la realidad (nuevamente, un motivo para hacer las ISR cortas).

Efectos dentro de la ISR. Dentro de la ISR el resto de interrupciones están desactivadas. Esto supone:

- La función millis no actualiza su valor, por lo que no podemos utilizarla para medir el tiempo dentro de la ISR. (sí podemos usarla para medir el tiempo entre dos ISR distintas)
- Como consecuencia la función delay() no funciona, ya que basa su funcionamiento en la función millis()
- La función micros() actualiza su valor dentro de una ISR, pero empieza a dar mediciones de tiempo inexactas pasado el rango de 500us.
- En consecuencia, la función delayMicroseconds funciona en ese rango de tiempo, aunque debemos evitar su uso porque no deberíamos introducir esperas dentro de una ISR.

Consulta

Contenidos

1. Qué son y como usar interrupciones en Arduino
2. Arduino y las interrupciones (25-8-2017)

(*sólo para alumnos matriculados)*

3. **Variables volátiles**. Declaración y uso de variables volátiles: Cuando una variable está en la función asociada a la interrupción y en la función `loop` , tenemos que declararla como `volatile int variable` para que se sitúe en la RAM.
    1. Referencia para el programador del lenguaje Arduino
    2. Introducción a la programación del Arduino: Referencia del lenguage C en Arduino