Entradas digitales

Supongamos que deseamos detectar cuando el valor de una señal externa al microcontrolador es cero lógico. Pero que además de esto, estamos realizando otras tareas que duran X segundos.

```
#define SECONDS X
delay(SECONDS*1000);
```

Encuesta

La manera común de resolver este problema es encuestando esta señal, o sea, preguntar si el valor de la misma es cero lógico. Este procedimiento se muestra a continuación:

```
if (digitalRead(INPUT_PIN) == LOW)
    Serial.println("Digital signal detected");
```

Este método funciona para la mayoría de las ocasiones, pero no siempre es la solución óptima.

```
void loop()
{
  // put your main code here, to run repeatedly:
  delay(SECONDS*1000);
  if (digitalRead(INPUT_PIN) == LOW)
    Serial.println("Digital signal detected");
}
```

El problema es que mientras que el microcontrolador esté realizando otras tareas (delay en el código anterior), nunca podrá verificar si cambia el estado de la señal. Por lo tanto es muy probable que existan cambios de estado que no se detecten. Esto ocurre porque la detección del estado de la señal se realiza por software y no por hardware.

Interrupciones externas

Las interrupciones son más eficientes que la encuesta,precisamente porque el evento **se detecta por hardware y no por software**. Esto significa que el microcontrolador puede estar realizando otras tareas e incluso así detectar el evento. Las interrupciones **permiten asociar una funcion a la ocurrencia de un determinado evento**. Para definir una interrupción externa se necesita:

- Un pin de Arduino que detectará el evento
- Una condición de disparo
- Una subrutina de atención a la interrupción

Los pines que pueden ser usados por interrupciones varían entre las placas Arduino como se muestra a continuación:

Placa	**Pines digitales usados para interrupción**
Uno, Nano (Basados en AtMega 328)	2, 3
Mega, Mega ADK, Mega 2560	2, 3, 18, 19, 20, 21
Micro, Leonardo (Basados en AtMega 32u4)	0, 1, 2, 3, 7
Zero	Todos los pines, excepto el 4
Due	Todos los pines

Las condiciones de disparo más comunes son:

- CHANGE: Se dispara cuando exista tanto un frente de subida como uno de bajada
- FALLING: Se dispara cuando exista un frente de bajada, o sea, cuando pase de HIGH a LOW
- RISING: Se dispara cuando exista un frente de subida, o sea, cuando pase de LOW a HIGH

Para resolver el problema inicial de la manera más eficiente, hacemos lo siguiente:
Se le indica al microcontrolador que ejecutará el método *signalDetected()* cuando haya un frente de bajada en el pin *INTERRUPT_PIN*.

```
attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), signalDetected, FALLING);
```

En el método *signalDetected()* se activa una 'bandera' que indica que se detectó el evento deseado.

```
void signalDetected()
{
  isDetected = true;
}
```

En el *loop()* después de realizar las otras tareas (delay en este caso),se pregunta por esta bandera.

```
void loop()
{
  // put your main code here, to run repeatedly:
  delay(1000*SECONDS);
  if (isDetected)
  {
    Serial.println("Digital signal detected");
    isDetected = false;
  }
}
```

Esta solución es más eficiente que la anterior porque nunca se dejará de detectar un evento. Siempre que la señal pase de uno a cero lógico, el microcontrolador ejecutará la subrutina de atención, y desde el programa principal se sabrá preguntando por la 'bandera'.

Aspectos a tener en cuenta con el uso de las interrupciones

- Dentro de la subrutina de atención, la mayoría de funciones relacionadas a los temporizadores no funcionarán correctamente (*millis()*, *delay()*, etc). Esto ocurre debido a que estas funciones utilizan interrupciones que no se disparan mientras se esté ejecutando una interrupción externa.
- Cualquier variable que sea modificada dentro de una subrutina de atención, debe declararse como 'volatile'
- Los datos recibidos por puerto serie pueden perderse mientras se esté ejecutando una interrupción externa
- Como práctica sana, las interrupciones deben ser lo más cortas posible.