Cap 4. Interrupciones

Que es una interrupción?

A grandes rasgos, una interrupción es una señal que interrumpe la del actividad del procesador. Existen dos formas de activarlas interrupciones:

- Evento interno -> Un cronómetro o señal software
- Evento externo -> Un cambio de estado en un pin

Las interrupciones, se deben declarar en el código del programa en *una rutina de servicio de interrupción* (ISR). Esto se hace así porque en el momento en el que se active la interrupción, durante la ejecución del programa principal, esta provocará que el programa se ponga en pausa y realizará las instrucciones que se hayan declarado en la rutina anteriormente descrita. Una finalizada dicha rutina, el programa volverá a ejecutarse desde el punto en el que se encontraba antes de activarse la interrupción.

Las interrupciones de los chips AVR (los que usan nuestros arduinos) son "Asíncronas", la cual cosa significa que ocurre fuera del flujo normal del programa, por lo que nos es muy útil a la hora de programar, ya que **no** tenemos que estar pendiente de cómo estará el estado de dicho pin (si fuese externa) o del tiempo que queda para que ocurra (si fuese interna), sino que es ella la que nos avisa de cuando ocurre el suceso.

·Interrupciones externas.

Para activarlas, tenemos cuatro alternativas de detección:

Detección por *estado bajo* o *LOW*

Detección por *cambio de estado* o *CHANGE*

Detección por *flanco ascendente* o *RISING*

Detección por *flanco descendente* o *FALLING*

NOTA: Las marcas en rojo dentro de las señales indica el momento en el que se activaría la detección al estar activo su modo correspondiente.

Dependiendo de la placa Arduino que estemos usando (en nuestro caso Arduino UNO), tendremos más o menos pins dedicados para las interrupciones externas (Consultar datasheet del chip de la placa para saber el patillaje dedicado a las interrupciones Externas).

En nuestro caso tan solo tenemos dos pins dedicado para el uso de las interrupciones externas que son:

- Pin digital 2 -> Corresponde a la *Interrupción 0*
- Pin digital 3 -> Corresponde a la *Interrupción 1*

··Configuración de los registro para el uso de las interrupciones.

Para poder usar las interrupciones deberemos activar el uso de la interrupción externa y después asignarle un modo de activación.

Para seleccionar la interrupción que queremos usar, debemos modificar el registro **EIMSK** *(External Interrupt Mask Register)*.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

En la trama de este registro, modificaremos los bits 0 y 1, en función de la interrupción que queramos activar.

Una vez seleccionada la interrupción a usar, seleccionaremos el modo de disparo para dicha interrupción. Esto se hace mediante el registro **MCUCR** *(MCU General Control Register)*.

TRAMA DEL REGISTRO EN LA PRÓXIMA PÁGINA

En esta trama variaremos los primeros 4 bits (del 0 al 3), activando un modo de disparo u otro en función de la configuración.

MCUCR bits	7	6	5	4	3	2	1	0
	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00

MCUCR Bit	Nombre	Sentido	Oportunidades
7	SRE	Habilitar Ext.SRAM	0 = No SRAM externo conectado
			1 = externa SRAM conectado
6	SRW	Ext.SRAM estados de espera	0 = Sin estado de espera adicional en SRAM externa
			1 = estado de espera adicional sobre SRAM externa
5	SE	Habilitar el sueño	0 = Ignorar comandos SUEÑO
			1 = SLEEP en el mando
4	SM	Modo de espera	0 = modo inactivo (la mitad del sueño)
			1 = Apagado Modo (sueño completo)
3	ISC11	Interrumpir el control de pin INT1 (conectado a GIMSK)	00: de bajo nivel inicia de interrupción
			01: Indefinido
2	ISC10		10: Flanco descendente provoca interrupción
			11: Flanco ascendente provoca interrupción
1	ISC01	Interrumpir el control Pin INT0 (conectado a GIMSK)	00: de bajo nivel se inicia la interrupción
			01: Indefinido
0	ISC00		10: Flanco descendente provoca interrupción
			11: Flanco ascendente provoca interrupción

También se puede usar el registro **EICRA** (*External Interrupt Control Register A*) para ajustar el modo de activación. Su trama es la siguiente:

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Donde:

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Una vez configurado el sistema, debemos crear la función **ISR**. La función ISR, tal como se explica en la primera parte del documento, se activará una vez la interrupción este ejecutada, poniendo en pausa el código y ejecutando lo que haya en su interior. Una vez finalice su código, devolverá la línea de ejecución a la parte del código en la cual se encontraba antes de la llamada.

La función ISR tendrá una apariencia así:

```
ISR({vector}_vect){

}
```

Dónde "{vector}" deberá ser substituido por el vector de interrupción seleccionado. Para saber que vector usar nos iremos a la página:

http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

En esta página, encontramos un cuadro que nos indica la totalidad de los vectores a usar por nuestro chips AVR. En nuestro ejemplo vamos a usar el vector **INT0**, que nos permitirá trabajar con la interrupción externa "0", por lo que la función ISR quedará de la siguiente forma:

```
ISR(INT0_vect){

}
```

El código usado hasta el momento para la habilitación y uso de las interrupciones, es más cercano al *C* puro que al *C* de la IDE Arduino. Lo hemos hecho así, porque la intención es aprender, desde un nivel más abajo, como se comporta el procesador y porque en el uso de las Interrupciones internas, deberemos modificar los Timers y hasta el momento, tan solo se puede realizar de esta forma.

A pesar de ello, la librería de Arduino si contempla la activación y uso de las interrupciones y se hace de la siguiente forma.

Las únicas diferencias existentes aparecen a la hora de configurar la interrupción externa a usar, junto a su modo de activación, y el nombre que deberemos ponerle a la función ISR.

Para seleccionar la interrupción y asignar el modo de activación, usaremos:

attachInterrupt(*interrupción, función, modo****);***

Dónde:

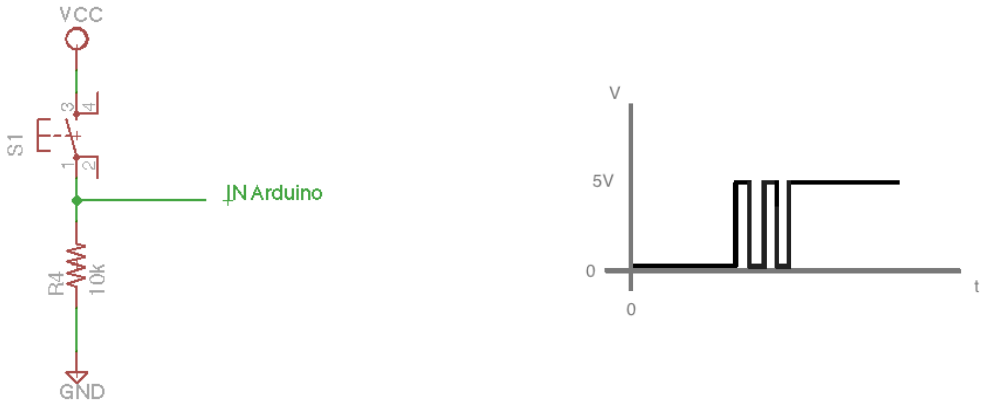
Interupción: Número de la interrupción a usar (0 o 1), debe ser "int".

Función: La función a la que debe invocar cuando se activa la interrupción. Esta función no debe devolver ningún valor, tan solo realizar un código (función ISR).

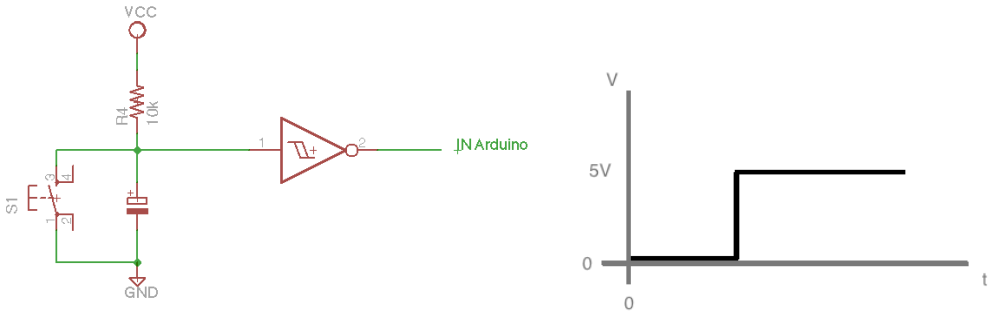
Modo: Uno cuatro modos anteriormente nombrados. (LOW, CHANGE, RISING o FALLING).

Es posible que a la hora de realizar diferentes circuitos nos encontremos con algún montaje que nos dé problemas a la hora de detectar la señal de activación de las interrupciones. Esto viene dado por el rebote que se introduce en la señal de entrada.

Como lo podemos evitar? El error que podemos sufrir en el montaje normal, aun conectándole una resistencia en paralelo a masa es el siguiente:



Una alternativa para solucionarlo es el uso de un Smith Trigger externo, en concreto el **HEF40106BP**.



Al conectar la resistencia y el condensador en serie, reduciremos la constante de carga del condensador, y el Smith trigger se encargará de limpiar e invertir la señal.

NOTA: El condensador es de 10uF