UNIDAD 2: I/O DIGITALES E INTERRUPCIONES

OBJETIVOS

No vamos a hablar de lo que son las entradas o salidas digitales. A estas Alturas es de suponer que las conoces y las habrás utilizado. Sí vamos a aclarar algunas ideas acerca del control de dispositivos digitales, y a explicar lo que son las cargas “Pull-Up” o las cargas “Pull-Down”.

En esta unidad vamos a darle una especial importancia a las interrupciones. Arduino es capaz de cancelar la ejecución del programa en curso y pasar a ejecutar otro programa o tarea diferente. Finalizada esta, Arduino retorna nuevamente al programa original. Todo ello puede ocurrir cuando por ciertas patillas de entrada se detecta una determinada señal. Ya lo vas a ver.

TEORÍA

- SALIDAS DIGITALES; NIVELES LÓGICOS DE ACTIVACIÓN
- ENTRADAS DIGITALES; CARGAS PULL-UP Y PULL-DOWN
- LAS INTERRUPCIONES

PRÁCTICAS

- **EJEMPLO 2-1: Activando unos leds**
- **EJEMPLO 2-2: Monitorizando entradas**
- **EJEMPLO 2-3: Monitorizando entradas con Pull-Up’s**
- **EJEMPLO 2-4: Monitorizando entradas sin interrupción**
- **EJEMPLO 2-5: Monitorizando entradas con interrupción**
- **EJEMPLO 2-6: Control de dos interrupciones**

MATERIALES

- ✓ Ordenador portátil o de sobremesa.
- ✓ Entorno de trabajo Arduino IDE que se incluye en el material complementario y que se supone instalado y configurado.
- ✓ Tarjeta controladora Arduino UNO
- ✓ Cable USB.

TABLA DE CONTENIDOS

TEORÍA

1. SALIDAS DIGITALES; NIVELES LÓGICOS DE ACTIVACIÓN

Aunque a veces nos puede asombrar, no es muy difícil entender lo que se va a explicar en este apartado. Es de suponer que ya conoces las funciones relativas al manejo de las salidas digitales. Las recordamos:

- **pinMode(n,OUTPUT):** Permite configurar cualquiera de las patillas digitales D0:D13 como salidas.
- **digitalWrite(n,LOW)**: Saca un nivel lógico "0" de 0 V por la patilla n indicada.
- **digitalWrite(n,HIGH)**: Saca un nivel lógico "1" de +5 V por la patilla n indicada.

Ahora bien, imagina que quieres controlar el encendido de un diodo led que está conectado con la patilla de salida digital D4. Seguro que escribirías una secuencia de funciones similar a esta:

```
pinMode(4,OUTPUT);   //Patilla D4 salida

digitalWrite(4,HIGH);   // Sacar nivel "1" por la patilla D4
```

¿Por qué se habla de sacar un nivel "1" (+5 V) cada vez que se quiere activar algo? ¿Es que el nivel "0" (0 V) sólo sirve para desactivar? ¿No puede ser al revés? Pues sí, efectivamente, puede ser al revés. Tanto el nivel "1" como el nivel "0" son igual de representativos. Mira los esquemas de la siguiente figura. ¿Qué nivel lógico habría que sacar por la señal D4 (patilla 7) para que se ilumine el led en el circuito de la izquierda? ¿y para que se active el led en el de la derecha?

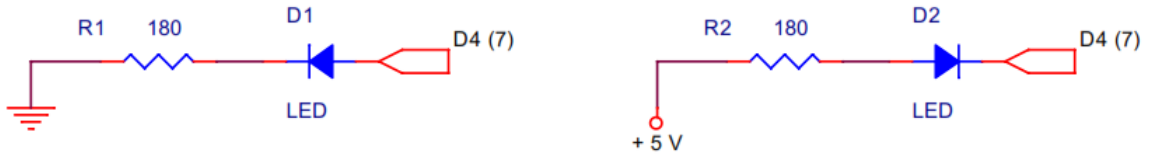


Figura 1

Sabiendo que para que un led se ilumine, el ánodo debe ser positivo respecto al cátodo, en el circuito de la izquierda por D4 (patilla 7) hay que sacar un “1” (+5 V) al ánodo ya que el cátodo se conecta con GND (0 V) a través de la resistencia de absorción. Sin embargo, en el circuito de la derecha, por D4 se debe sacar un “0” (0 V) al cátodo ya que el ánodo está conectado con +5 V a través de su resistencia. ¿Lo entiendes?

Lo mismo que hablamos de un led, podríamos hablar de otros dispositivos como la bobina de un relé, un motor, un zumbador, etc… Estudia los esquemas de la siguiente figura. Quizá mencionar que en el caso particular del motor notarás que el sentido de giro es diferente cuando lo activas con nivel “1” que cuando lo haces con nivel “0”.

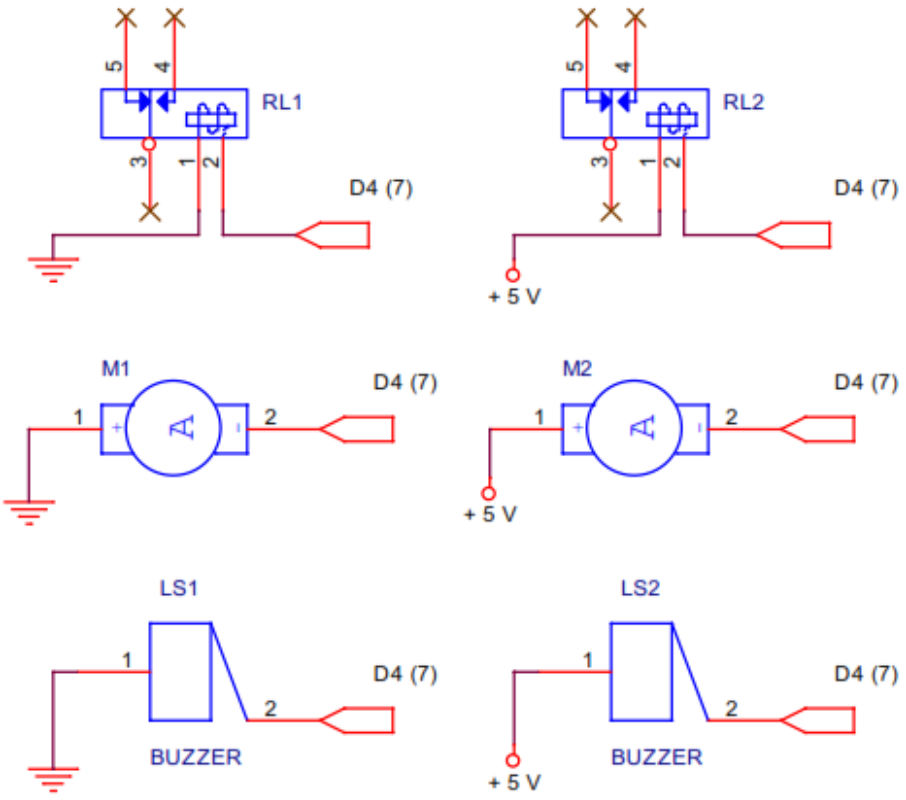


Figura 2

En resumidas cuentas. Hay periféricos de salida que se activan cuando les aplicas un nivel “1”. Se dice que son “activos al nivel 1”. También los hay que se activan al aplicarles un nivel “0”. Los puedes encontrar de ambos tipos. En el fondo todo depende de cómo esté construido y conectado dicho periférico.

2. ENTRADAS DIGITALES; CARGAS PULL-UP Y PULL-DOWN

Aunque creas saberlo todo sobre las entradas digitales, vamos a tratar un problema que, aunque lo hayas padecido, quizá no sepas el porqué. Para empezar recordemos que los periféricos de entrada digital más sencillas y económicas son unos simples pulsadores y/o interruptores. Los puedes encontrar de infinidad de formas y tamaños. Algunos están pensados para aplicaciones industriales, máquina herramienta, paneles de control, etc...

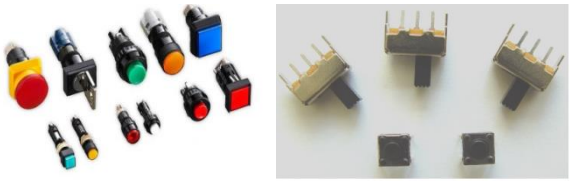
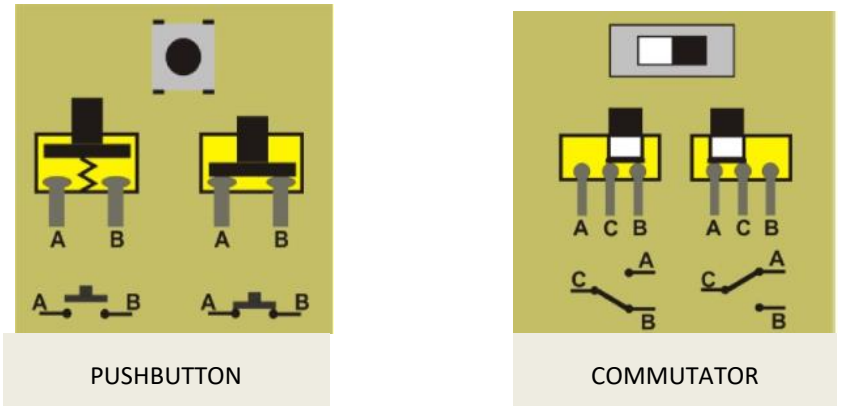


Figura 3

Hay otros mucho más sencillos (y baratos) muy utilizados en aplicaciones domésticas, educativas, para soldar sobre tarjetas impresas, en pequeños aparatos, etc… Nosotros vamos a usar unos pequeños pulsadores como los dos que se ven en la parte inferior de la fotografía.

En cualquier caso, sean del tipo que sean, su funcionamiento es bastante elemental. Cuando se accionan, una lámina metálica abre o cierra un contacto eléctrico entre dos o más patillas del dispositivo. Se dice que un interruptor o un conmutador tiene “enclavamiento”. Cuando lo accionas se queda donde lo dejas hasta que vuelvas a moverlo a otra posición (p.e. el interruptor de la luz). Sin embargo, los pulsadores no. Estos abren o cierran uno o más circuitos sólo en el momento de ser accionados. Luego vuelven a su posición de reposo cuando los sueltas (p.e. el timbre de tu casa). En las siguientes imágenes tienes la representación de un conmutador deslizante y de un pulsador. También tienes sus correspondientes símbolos eléctricos.



PUSHBUTTON COMMUTATOR

Figura 4

El conmutador de la figura tiene tres patillas. Una de ellas, la C es común a las otras dos. Cuando se desliza hacia la derecha la lámina cierra el circuito entre las patillas C y B. El circuito C y A queda abierto, sin conexión. Si se desliza hacia la izquierda se cierra el circuito entre C y A y queda abierto entre C y B.

El pulsador tiene básicamente dos patillas, A y B. Se dice que en reposo está normalmente abierto. Las patillas están sin conexión. Cuando lo accionas se cierra el circuito y ambas patillas quedan conectadas. Al soltarlo, un muelle se encarga de llevar la lámina metálica a su posición de reposo. Que sepas que también hay pulsadores que normalmente están cerrados y sólo se abren al accionarlos.

Tampoco es nada nuevo recordar las dos funciones del lenguaje Arduino para el control de las entradas digitales:

- **pinMode(n,INPUT)**: Permite configurar cualquiera de las patillas digitales D0:D13 como entradas. Realmente no es necesario ponerla. Cada vez que inicias el sistema, al conectar la alimentación o pulsar el RESET, todas las patillas se configuran por defecto como entradas
- **digitalRead(n)**: Lee el nivel lógico que hay en la patilla n que desees.

Sin embargo, analiza los circuitos de los siguientes esquemas, en el que un pulsador se ha conectado, por ejemplo, con D2 (patilla 5) que se supone configurada como entrada.

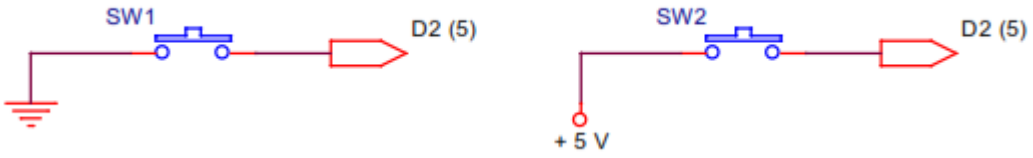


Figura 5

Está claro que, en el circuito de la izquierda, cada vez que accionas el pulsador, la patilla D2 se conecta con GND (0 V) y por tanto está a nivel "0". El de la derecha es al revés. Cuando accionas el pulsador la patilla D2 se conecta con +5 V y por lo tanto queda a nivel "1".

Y ahora la pregunta del millón ¿cómo quedan las patillas si en ninguno de los dos circuitos se acciona el pulsador? Seguro que algunos responderían… "justo al contrario que cuando se accionaba". Es decir, en el circuito de la izquierda, si no se pulsa, la patilla D2 queda a nivel "1" y en el de la derecha a nivel "0".

¡¡Error!! Vamos a ver. Si en ninguno de los dos circuitos se acciona el pulsador, bajo el punto de vista de la patilla D2 ¿qué diferencia hay? Ninguna. En ambos casos D2se queda sin conexión. Se dice que está "al aire". ¿Quién gana, el nivel "1" o el nivel "0"? No hay una respuesta clara. Algunas veces prevalecerá el "1" y otras el "0". Lógicamente esta situación no es deseable.

A nivel eléctrico lo correcto es añadir al circuito una resistencia. Si esta se conecta con el positivo +5 V se le llama resistencia o carga **"PULL-UP".** Si se conecta con GND o 0 V, se le llama **"PULL-DOWN".** Observa ahora estos dos circuitos. (Figura 6).

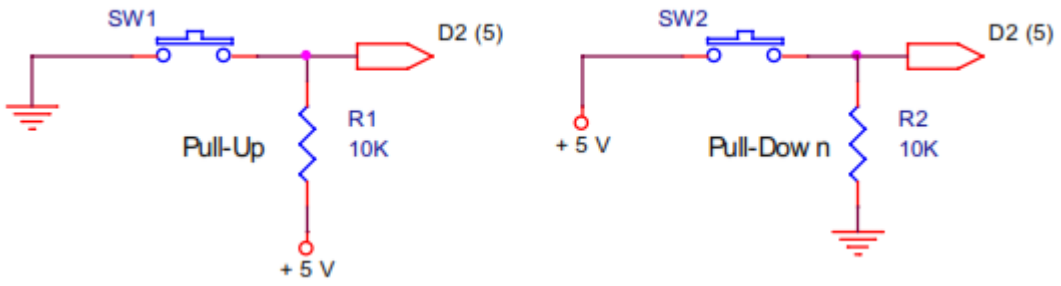


Figura 6

En el de la izquierda, la resistencia Pull-Up R1 “amarra” la señal D2 (patilla 5) a nivel “1” (+5 V) cuando el pulsador SW1 está sin accionar, o sea abierto. Cuando se acciona la patilla D2 pasa a nivel “0”. En el circuito de la derecha, la resistencia Pull Down R2 “amarra” la patilla a nivel “0” (GND) cuando el pulsador SW2 está sin accionar. Cuando se acciona la patilla D2 pasa a nivel “1”.

Vale, ¿pero de qué valor deben ser esas resistencias? No es algo crítico. Piensa que cada vez que acciones un pulsador y cierres el circuito, se va a producir un flujo de electrones entre GND y +5 V, es decir, circula una intensidad (I). Interesa que esta intensidad, este consumo, sea mínimo. Por ejemplo, si R1 o R2 las pusieras de 100Ω, el consumo sería de:

$$I = \frac{V}{R} = \frac{5}{100} = 0.05\ A = 50\ mA$$

Si eliges un valor de 10 KΩ, como en el ejemplo, el consumo es de:

$$I = \frac{V}{R} = \frac{5}{10000} = 0.0005\ A = 0,5\ mA$$

Ahora bien, no debes pasarte. Puedes tener la tentación de poner una resistencia muy grande para reducir el consumo al mínimo. Lo que ocurrirá entonces es que con ese valor tan grande, la patilla D2 queda sin “amarrar”, aislada del +5 V (en el caso del Pull-Up) o de GND (en el caso del Pull-Down). Si es así, te quedas como al principio. Para eso mejor no pones ninguna resistencia. Suele ser muy frecuente emplear valores comprendidos entre 4700 y 10 KΩ.

Nuevamente hay que resaltar que el nivel activo, es decir cuando se acciona el pulsador, no tiene porqué ser el “1”, también puede ser el “0”. Todo depende de cómo conectes ese pulsador.

¿Qué nos aporta el Arduino? La posibilidad de quela resistencia Pull-Up la ponga él mismo, evitando así quela tengas que poner tú externamente en tus circuitos. Observa la Figura 7: la resistencia está integrada dentro delpropio controlador Arduino.

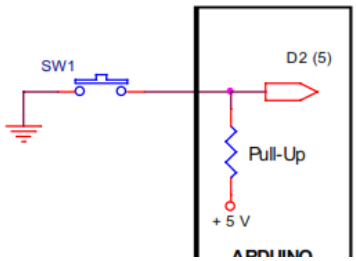


Figura 7

Esta opción la puedes configurar mediante la función:

- **pinMode (n, INPUT_PULLUP)**: n representa a la patilla de entrada a la que deseas asociar su correspondiente resistencia Pull-Up.

Lo puedes hacer con cualquiera de las patillas D0:D13 que vayan a ser configuradas como entrada. No tiene sentido aplicarlo a las salidas. Tampoco tienes posibilidad de configurar una entrada con resistencia Pull-Down.

Quizá no te parezca importante la posibilidad de ahorrar una resistencia. Al fin y al cabo, cuesta unos pocos céntimos de euro. Sin embargo, piensa en una aplicación comercial. Si tu proyecto precisa de varias resistencias Pull-Up, no sólo ahorras en el precio de estas, también ahorras en:

- No tener que mantener un stock de las mismas.

- El tamaño de la placa o tarjeta de circuito impreso. Esta se cotiza por decímetro cuadrado de superficie y a más componentes mayor tamaño..
- El tiempo de diseño de las pistas de esa tarjeta impresa. A más componentes más tardarás en diseñarlas. El tiempo es oro.
- Montaje de la tarjeta. A más componentes más tiempo de mano de obra.

Si a lo anterior lo multiplicas por el número de tarjetas o equipos que vas a comercializar, el ahorro puede suponer varias decenas o cientos de euros. Ya ves que no es ninguna tontería. Ahora que lo sabes, utiliza la función Pull-Up siempre que lo consideres oportuno.

3. LAS INTERRUPCIONES

Vamos a tratar otro tema: las interrupciones. Con este nombre quizá pienses que esto consiste en parar al controlador, detenerlo y que deje de ejecutar el programa. Nada de eso.

Imagina una máquina herramienta que está haciendo piezas. En un momento determinado salta una alarma por alguno de estos motivos:

- La pieza está mal colocada y sale defectuosa.
- El motor que mueve la cinta que transporta las piezas se ha averiado y estas se están amontonando.
- Un sensor detecta que hay un exceso de calor en un determinado componente de la máquina.
- Un operario mete la mano donde no debe.

¿Qué debe hacer el controlador que gobierna el funcionamiento de esa máquina?

a) Nada en especial, seguir ejecutando el programa habitual como si nada.

b) Pararse y dejar de ejecutar el programa habitual inmediatamente.

c) Dejar de ejecutar el programa habitual y ejecutar otro programa que atienda a la causa que provocó la alarma, dando los avisos y llevando a cabo las tareas oportunas: retirar la pieza defectuosa, parar la producción de piezas, activar un sistema de refrigeración, dar una señal de aviso al operario, etc…

La respuesta está clara ¿no? Una interrupción no implica que el controlador se detenga, sino que interrumpa la tarea que está ejecutando en ese momento y pase a ejecutar otra distinta.

¿Cómo se provoca una interrupción? Pueden ser muchos los eventos o sucesos que Producen una interrupción. Depende del modelo de controlador del que se trate. Lo típico es que sean periféricos externos que aplican señales por determinadas patillas del controlador. En el caso del Arduino UNO que nos ocupa, dispone de dos fuentes o patillas de Interrupción diferentes: D2/INT0 y D3/INT1.

Seguro que recuerdas que estas patillas las has usado en múltiples ocasiones y que jamás han provocado interrupción alguna. En el fondo son patillas de entrada/salida digital igual que todas las demás D0:D13. Es cierto. Lo que ocurre es que para que las patillas D2 y D3 actúen como entradas de interrupción INT0 e INT1 respectivamente, es necesario configurarlas de forma adecuada y, de alguna manera, dotarles de una especie de permiso. Provocar una interrupción no es algo baladí. Es precisamente lo que vas a aprender en este apartado.

¿Qué ocurre en el controlador cuando recibe una interrupción? Observa la Figura 8 donde se resume el esquema de trabajo:

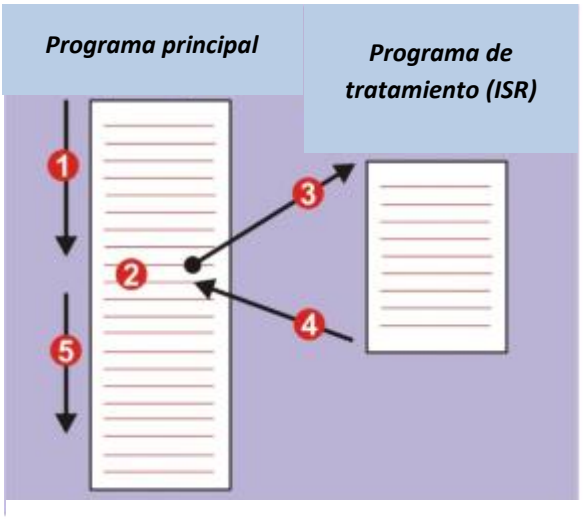


Figura 8

1. El controlador está ejecutando su programa habitual llamado “Programa principal”.
2. En un momento dado un periférico solicita y provoca una interrupción.
3. El controlador deja el programa principal y pasa a ejecutar el programa de tratamiento. También se le conoce como ISR (Interrupt Service Routine – Rutina de servicio a la interrupción).
4. Finalizada la ejecución de la ISR, el controlador retorna al programa principal.
5. La ejecución se reanuda a partir del punto en que se abandonó.

¿Qué hay de nuevo? Para que puedas gestionar y usar las interrupciones de Arduino UNO dispones de cuatro funciones:

Función attachInterrupt()

Configura cómo debe funcionar una interrupción.

Sintaxis:

attachInterrupt(pin, ISR, modo);

pin: Representa a la patilla que se va a configurar como entrada de interrupción. En el caso de Arduino NANO puede ser INT0 (D2) o bien INT1 (D3).

ISR: Nombre de la función o rutina que se debe ejecutar cada vez que se produce esa interrupción.

modo: Cuándo se debe producir la interrupción:

LOW: Cuando por el pin se recibe un nivel "0".

CHANGE: Cuando se detecta un cambio de estado en el pin.

RISING: Cuando se detecta un flanco de subida por el pin ("0" → "1").

FALLING: Cuando se detecta un flanco de bajada por el pin("1"→ "0").

Función detachInterrupt()

Desactiva una interrupción.

Sintaxis:

detachInterrupt(pin);

pin: Representa a la patilla cuya interrupción se va a desactivar. En el caso de Arduino UNO puede ser INT0 (D2) o bien INT1 (D3).

Funcióninterrupts()

Habilita las interrupciones. Considérala como una especie de permiso global que permite aquellas interrupciones que previamente hubieran sido configuradas mediante attachInterrupt().

Sintaxis:

interrupts();

FunciónnoInterrupts()

Deshabilita a todas las interrupciones. Considérala como una prohibición globalque impide el funcionamiento de todas las interrupciones.

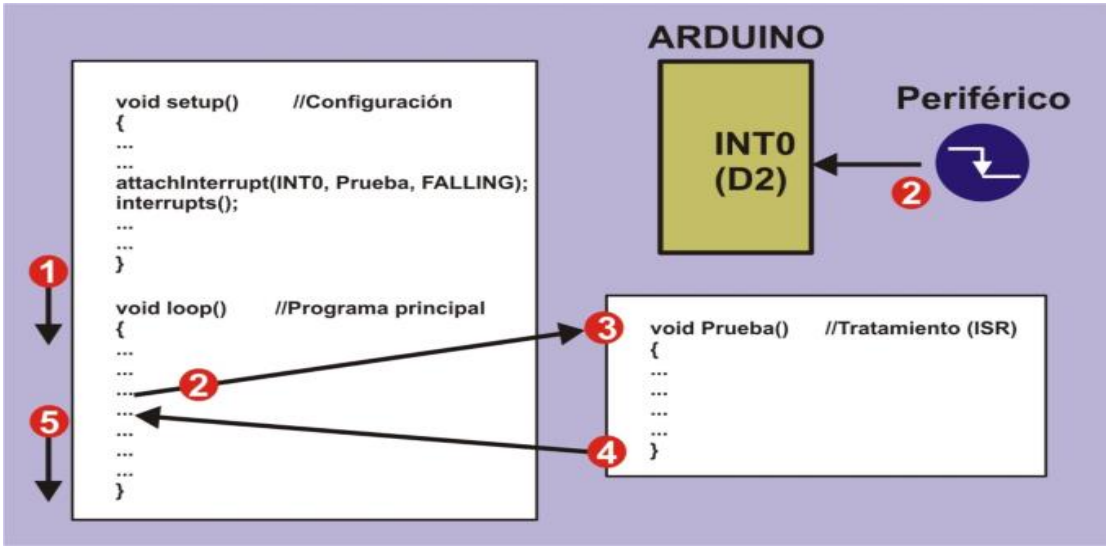
Sintaxis:

noInterrupts();

La siguiente figura te puede dar una idea del aspecto que puede tener un programa que emplea interrupciones.

En la función setup() de configuración, mediante attachInterrupt(), se configura la interrupción por la patilla D2 (INT0) cada vez que se detecte un flanco descendente(FALLING). Si esto sucede se ejecutará la función Prueba() que corresponde al tratamiento de la misma (ISR). Acto seguido se da permiso global mediante interrupts(). En estas condiciones…

1. Comienza a ejecutarse el programa principal loop().
2. En un momento dado un periférico aplica una señal de interrupción con flanco descendente por la patilla INT0 (D2). El programa principal deja de ejecutarse.
3. El controlador pasa a ejecutar la función Prueba() (tratamiento o ISR).
4. Al final de la misma se retorna al programa principal.
5. Este reanuda su ejecución a partir de donde se quedó.



Limitaciones

Por la propia naturaleza de Arduino, su sistema de interrupciones está algo restringido. Por ejemplo, las conocidas funciones delay() y millis() no funcionan si se encuentran dentro del programa de tratamiento (ISR). Esto es porque esas funciones emplean a su vez el propio sistema interno de interrupciones. Arduino no puede atender una interrupción dentro de otra.

Tampoco puedes pasar parámetros a una función de tratamiento (ISR) ni esta te puede devolver nada. Si lo necesitaras, tendrás que emplear variables globales para pasar los datos que deben ser usados tanto por el programa principal como por la función del programa de tratamiento.

Piensa que una interrupción es un suceso que eventualmente puede o no ocurrir. Se puede producir una situación de alarma o no. Un periférico puede enviar la señal de interrupción o puede que no. Es recomendable que la función de tratamiento (ISR) se pueda ejecutar lo más rápidamente posible. Ten en cuenta que mientras ésta se ejecuta, la ejecución del programa principal queda “abandonada”.

Ventajas

También tienen ventajas. En general se dice que un programa en el que se emplean interrupciones es bastante más eficiente. Eso sí, debes emplearlas sabiamente. Imagina que en tu proyecto tienes que hacer un determinado trabajo cada vez que un periférico te mande una señal. Tienes dos opciones. Una, mediante la función digitalRead(), leer y esperar a que se produzca dicha señal. Eso sí, mientras esperas no puedes hacer otra cosa.

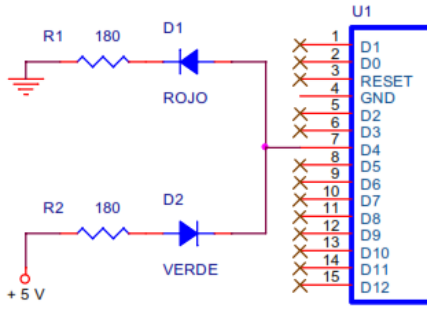
Otra solución, la buena, es emplear una interrupción. Tu programa principal puede estar realizando lo que sea. Cuando le llegue la señal, y sólo entonces, el controlador “atiende” a la interrupción y desarrolla el trabajo que corresponda.

Digamos que no hay “tiempos muertos” ya que el controlador siempre está haciendo algo útil, como si hiciera varias tareas casi al mismo tiempo.

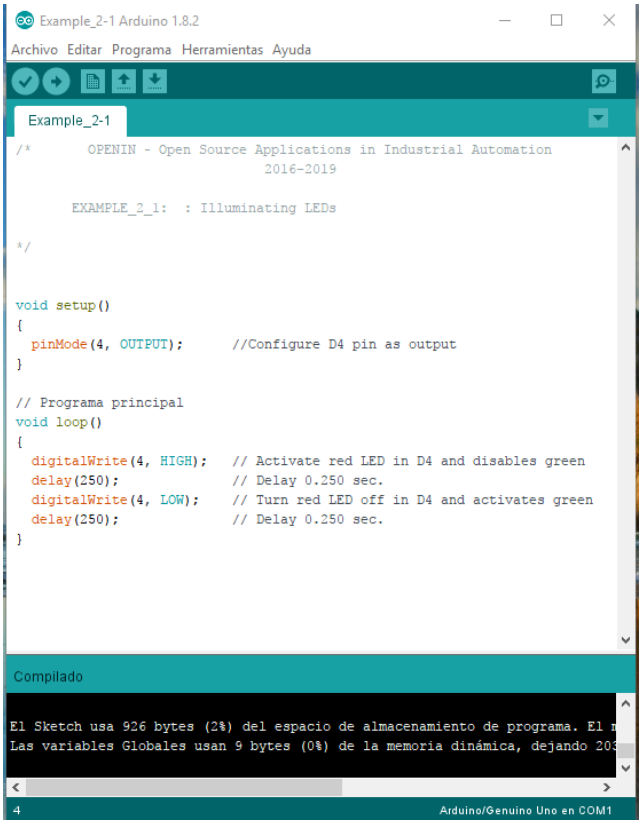
PRÁCTICA

4. EJEMPLO 1: ACTIVANDO UNOS LEDS

Con este ejemplo no vas a aprender nada nuevo en cuanto a programación se refiere. A estas alturas ya debes saber cómo activar y desactivar uno o varios leds. Lo que sí vas a hacer es el montaje eléctrico necesario para comprobar que un led (u otros periféricos) se puede activar al aplicarle un nivel lógico“1” o un nivel “0”.Observa el esquema de la figura.



El programa es muy sencillo. La patilla D4 se configura como salida en la función setup().

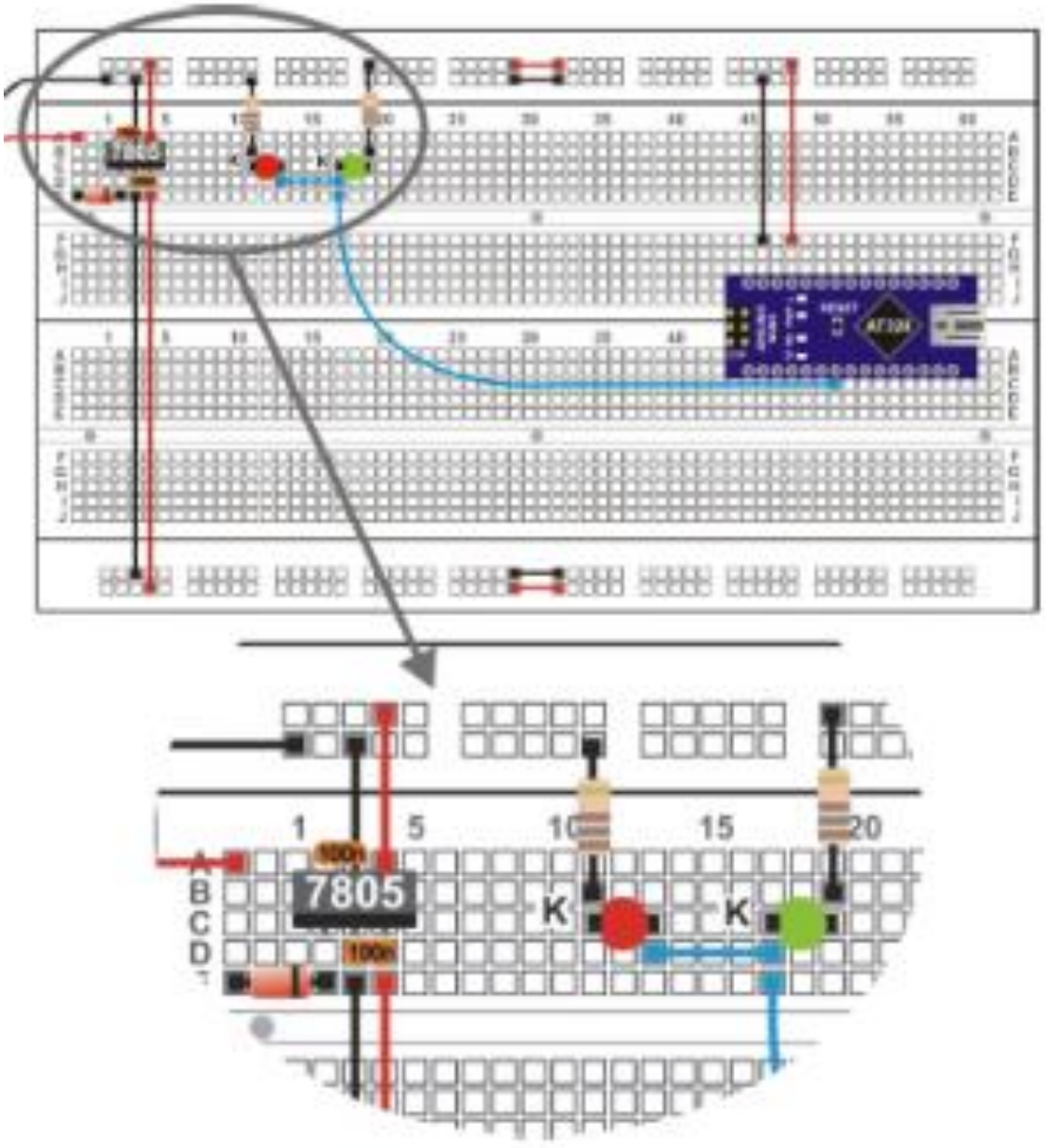


El programa principal es un bucle infinito. En él la patilla D4 se pone a nivel “1”. Se enciende el led rojo mientras que el led verde se apaga.

Tras una temporización de 0.25”,la patilla D4 se pone a nivel ”0”. El led rojo se apaga y ahora se enciende el verde. Tras otra temporización de 0.25”

el ciclo se vuelve a repetir.

Aquí tienes una representación del montaje práctico sobre el módulo board. Procura prestar atención a la disposición y orientación de los componentes.



Ahora tú.

El programa es muy sencillo así que no vas a hacer nada al respecto. Lo que sí puedes hacer es modificar el valor de, por ejemplo, la resistencia de absorción R2, la del led verde. Su valor es de 180 Ω, pues cámbiala por otra de 10 KΩ.

¿Qué observas?

-

¿Por qué?

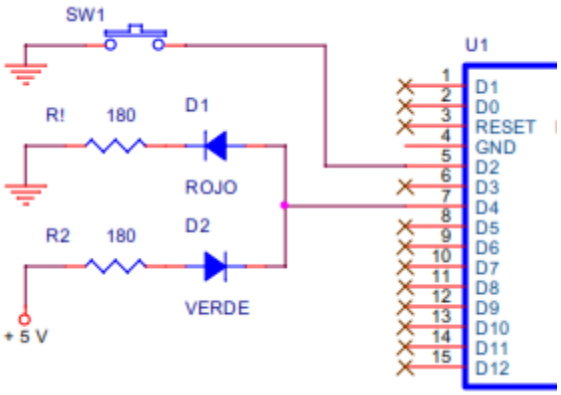
-

Prueba a calcular la intensidad que circula ahora por el led verde:

$$I = \frac{V}{R} = \frac{V - V_{AK}}{R} = \frac{5 - 1.5}{10000} = 0.00035\ A = 0,35\ mA$$

Ya ves que la intensidad I que circula por el led ahora es de 0.35 mA cuando elfabricante indica que debe ser de 20 mA aprox. Ya conoces otra forma de regular elbrillo con que se ilumina un led. Vuelve a colocar la resistencia R2 con su valor originalde 180 Ω.

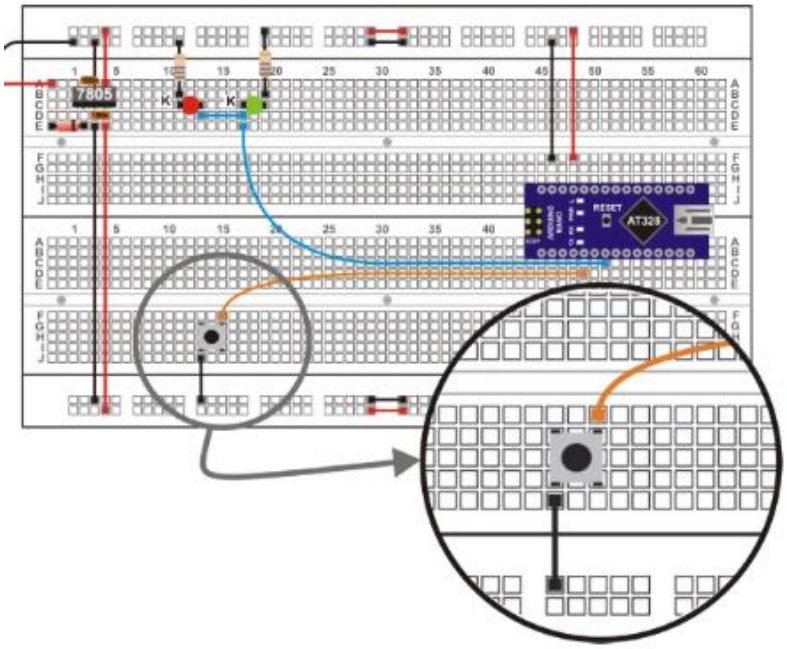
5. EJEMPLO 2: MONITORIZANDO ENTRADAS



Aquí tienes el esquema. Es idéntico al del ejemplo anterior al que se le ha añadido el pulsador SW1 conectado con D2.

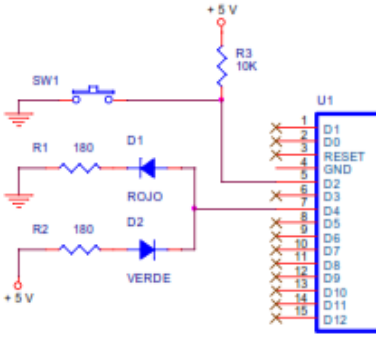
El ejemplo tampoco aporta ninguna novedad en lo que a programación se refiere. Se trata de leer el nivel lógico de la patilla de entrada D2 y reflejarlo sobre los leds rojo y verde. Si la entrada está a "1"se ilumina el rojo, y si está a "0" se ilumina el verde. Ambos leds se controlan mediante la patilla de salida D4Para insertar correctamente el pulsador sobre el módulo board, te recomiendo que endereces sus patillas con ayuda de un alicate de puntas planas como en la figura. Las patillas deben quedar totalmente perpendiculares al cuerpo del pulsador.

En la siguiente figura dispones del aspecto del montaje práctico. Poco a poco debes familiarizarte con el montaje de circuitos sobre el módulo board.

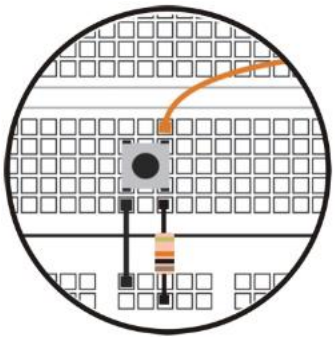


Ahora tú.

Como ya decía, el programa del Ejemplo_2-2 no tiene nada que no conozcas. Limítate a grabarlo sobre Arduino UNO y comprueba su funcionamiento. A lo mejor te llevas una sorpresa. Está claro que cuando accionas el pulsador introduces un nivel “0” por la patillaD2, por lo que el led verde debe iluminarse. Pero si no lo accionas ¿qué observas? Es más, si tocas con los dedos el cable del pulsador, o el del led, o directamente las patillas del controlador Arduino UNO, ¿qué pasa?



Parece que los leds se encienden de forma aleatoria. Esto es porque cuando no pulsas SW1 la entrada D2 está “al aire” y no tiene un nivel lógico definido.



Modifica ligeramente elcircuito eléctrico añadiendo una resistencia de 10 KΩ como se muestra en el esquema teórico y en

el de montaje. Se trata de la resistencia Pull-Up. Gracias a ella la patilla D2 se mantiene "amarrada" a nivel "1" cuando el pulsador esté sin accionar.

Comprueba que el mismo programa Ejemplo_2-2, sin modificación alguna, ahora funciona correctamente y el encendido de los leds no presentan ninguna anomalía. Ya ves que el detalle es importante.

6. EJEMPLO 3: MONITORIZANDO ENTRADAS CON PULL-UPS

Con este ejemplo queda resuelto definitivamente el asunto de las resistencias Pull-Up. La patilla D2 se configura como entrada con resistencia Pull-Up, por lo que ya no la tienes que poner externamente. Ahora el circuito eléctrico queda exactamente igual que el del primer esquema del ejemplo anterior. Lo único que ha cambiado es el programa.

```
void setup()
{
  pinMode(4, OUTPUT);      //Configure D4 pin as output
  pinMode(2, INPUT_PULLUP);
}
```

Mediante la función pinMode(2, INPUT_PULLUP) dentro de setup(), seconfigura a la patilla D2 para que actúe como entrada con Pull_Up. Antes de grabar el programa retira la resistencia Pull-Up de 10 K que pusiste en el ejemplo anterior. No debe haber más que una resistencia y

en este caso la poneel Arduino. Ahora graba el programa y comprueba que funciona correctamente. Los ledsconectados en la salida D4 monitorizan el estado lógico del pulsador conectado en lapatilla D2. El rojo se ilumina cuando está sin accionar (nivel “1” gracias a la Resistencia Pull-Up interna) y el verde cuando lo acciones (nivel “0”).Como resumen a estos tres primeros ejemplos podemos sacar las siguientesconclusiones:

- Cualquier periférico digital de salida sepuede activar mediante nivel “1” omediante nivel “0”. Depende de cómo estéconectado.
- Igualmente, los periféricos digitalespueden producir un nivel “1” o bien unnivel “0” cuando están activos. Tambiéndepende de cómo estén conectados.
- El nivel “0” es tan válido como el nivel “1”.Ambos son igual de representativos.
- Algunos periféricos de entrada puedenincluir o no la resistencia Pull-Up. En caso negativo deberás actuar enconsecuencia poniéndola tu externamente o bien que la ponga el propio Arduino.

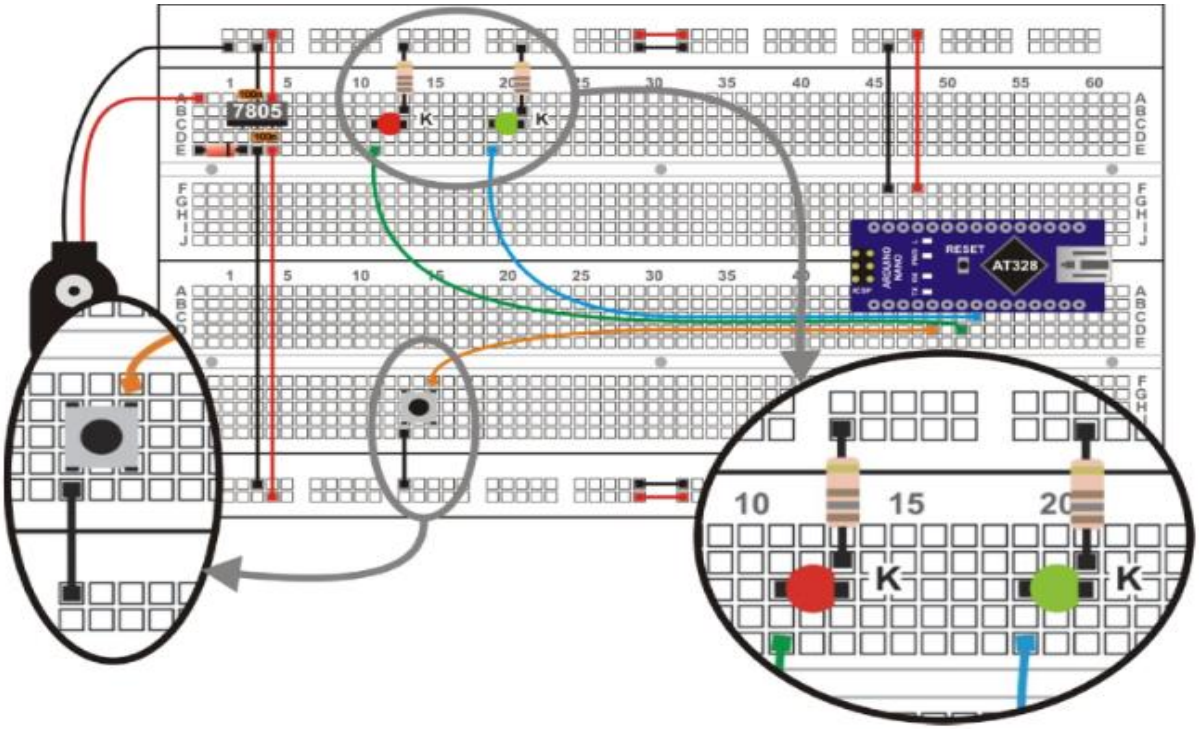
7. EJEMPLO 4: MONITORIZANDO ENTRADAS SIN INTERRUPCIÓN

De nuevo tienes otro ejemplo que no te va a enseñar nada nuevo en cuanto a programación se refiere, y sin embargo te va a permitir valorar la conveniencia o no de emplear interrupciones.

En la patilla de salida D4 se ha conectado un led, el rojo, que va a monitorizar en todo momento el estado de la patilla de entrada conectada en D2. Al mismo tiempo se pretende que otro led, el verde, conectado en la salida D5, parpadee y cambie de estado cada 0.5”. ¿Fácil no?

En la figura tienes el esquema eléctrico. El led rojo se controla desde D4 y monitoriza el estado del pulsador conectado en D2. El led verde se controla desde D5 y parpadea a intervalos de 0.5”.Observa que ambos leds se activarán ahora con nivel “1”.

Este sería el montaje práctico que debes realizar sobre el módulo board.



Una vez que realices el montaje y grabes el programa, comprueba con cuidados o funcionamiento. ¿Qué observas?

Efectivamente, la monitorización sobre el led rojo del estado del pulsador parece que va con "retraso". Incluso, si haces una rápida pulsación, a veces el led rojo "ni se entera". ¿Por qué? Mientras el controlador está ejecutando una de las dos funciones delay (500), no puede estar ejecutando la función digitalWrite(4, digitalRead(2)). Es decir, no puede estar al tanto de lo que ocurre con el pulsador de la entrada D2 ni por tanto monitorizar su estado en D4. Realmente esa monitorización sólo se ejecuta cada vez que finaliza un ciclo de encendido/apagado de la salida D5, y esto ocurre cada 1 ". ¿Te das cuenta? Vamos a tratar de resolver el problema en el siguiente ejemplo, mediante el uso de una interrupción.

8. EJEMPLO 5: MONITORIZANDO ENTRADAS CON INTERRUPCIÓN

El ejemplo trata de hacer exactamente lo mismo que el anterior, pero bien hecho. Sobre la salida D5 se produce un parpadeo de forma que el led se enciende durante 0.5” y se apaga otro tanto. Al mismo tiempo la salida D4 monitoriza el estado lógico de la entrada D2.

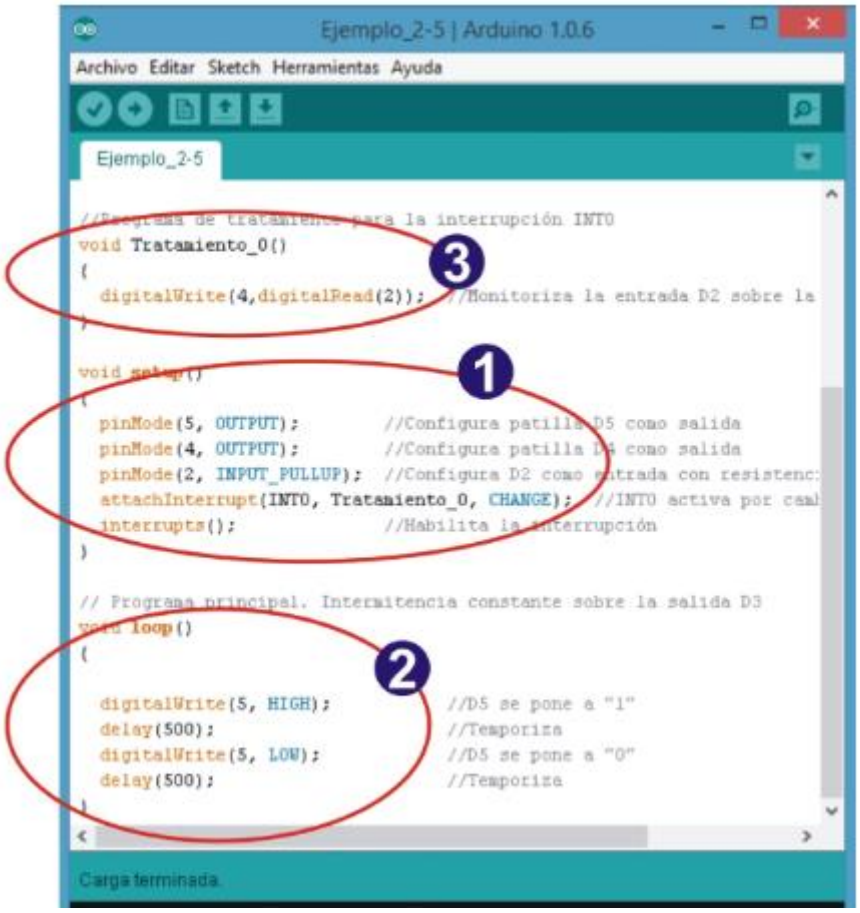
Utiliza el mismo esquema eléctrico y el mismo montaje práctico. Cambia el programa que tiene tres secciones bien diferenciadas. Observa la figura siguiente.

Setup():

Las patillas D4 y D5 se configuran como salidas. D2 se configura como entrada con resistencia Pull-Up. Mediante la función attachInterrupt() se configura la interrupción. Esta se asocia con la patilla INT0 (D2), se activa cuando se detecte un cambio de estado en la misma y, cada vez que se produzca, se ejecuta la funciónTratamiento_0(). La función interrupts() habilita y concede permiso a las interrupciones, en este caso a INT0.

loop():

Es el programa principal. Consiste en producir una intermitencia sobre la salida D5. El led se enciende durante 0.5” y se apaga otro tanto. No hay ninguna novedad.



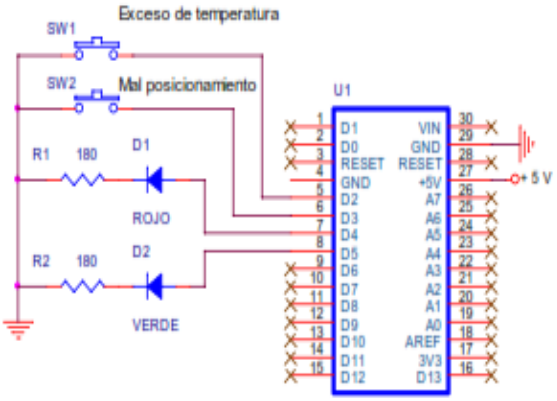
Treatment_0():

Es el programa de tratamiento de la interrupción (ISR). Cada vez que se detecta un cambio de estado en la patilla INT0 (D2), el controlador deja lo que esté haciendo y pasa a ejecutar la función Tratamiento_0(). En ella se lee el estado de la propia entrada D2 y se monitoriza sobre la salida D4. Al finalizar retorna al programa principal en el punto en que lo abandonó.

Ahora, cuando grabes el programa, puedes comprobar que la monitorización sobre el led rojo del estado de la entrada es instantánea. A pesar de ello el led verde sigue parpadeando como si nada hubiera ocurrido. Ten en cuenta que el programa de tratamiento, el que monitoriza el estado de la entrada, se ejecuta en unos pocos microsegundos y no afecta de forma apreciable al programa principal.

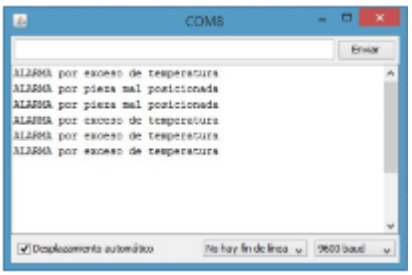
9. EJEMPLO 6: CONTROL DE LAS DOS INTERRUPCIONES

Vamos a acabar la Unidad 2 con un nuevo ejemplo que utiliza las dos interrupciones disponibles en Arduino NANO: INT0 (D2) e INT1 (D3). Imagina una máquina herramienta encargada de controlar dos salidas, D4 y D5, que realizan una determinada secuencia.



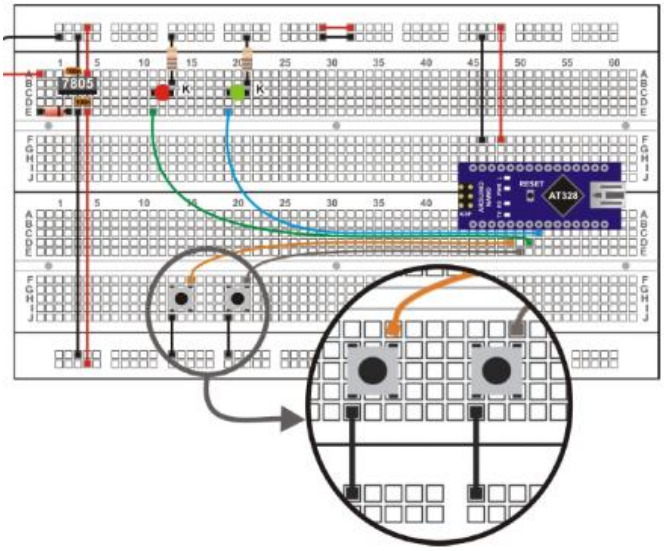
Dos sensores detectan dos posibles situaciones de alarma: exceso de temperatura y que una pieza está mal posicionada. Estos sensores, simulados mediante los pulsadores SW1 ySW2, están conectados con las patillas D2 y D3 y provocan las interrupciones INT0 e INT1respectivamente. Mira el esquema eléctrico de la figura

El programa principal realiza una secuencia de encendido de los leds rojo y verde. Cuando se detecta una de las dos interrupciones el programa de tratamiento oportuno transmite vía serie un mensaje de aviso de alarma, como los que ves en la ventana adjunta del monitor serie de comunicaciones.



Ambas interrupciones están configuradas para que sean activas al flanco descendente. Esto ocurre cada vez que pulses el correspondiente pulsador. Ten en cuenta que las patillas D2 (INT0) y D3 (INT1) están configuradas como entradas con resistencia Pull-UP. Cuando los pulsadores están sin accionar, las patillas están a “1”.

Aquí puedes ver el montaje práctico. Es muy similar al del ejemplo anterior. Seha añadido el pulsador SW2 conectado con D3 (INT1)..



Una vez que grabes el programa comprueba su funcionamiento. Abre el monitor serie. Sin accionar ninguno de los dos pulsadores, el programa principal se limita a realizar una secuencia en el encendido de los leds rojo y verde. Si quieres puedes cambiarla.

Al accionar cualquiera de los dos pulsadores, en la ventana del monitor serie verás el mensaje correspondiente. Hemos comprobado que ocasionalmente, cuando se provoca cualquiera de las dos interrupciones, el sistema se bloquea. Hay que reiniciarlo pulsando el RESET. Viene motivado por el “efecto rebote” de los pulsadores. Efectivamente, una única pulsación puede interpretarse como varias interrupciones consecutivas y Arduino no puede tratarlas debidamente. También hemos comprobado en el laboratorio que si las señales de interrupción (o alarma) proceden de un generador libre de rebotes, el sistema funciona correctamente y no se bloquea. Hemos llegado a provocar hasta 10interrupciones por segundo durante 5 minutos y la respuesta ha sido correcta.

Puedes experimentar con otro tipo de periféric os diferentes a los pulsadores. Manos a la obra.

REFERENCIAS

LIBROS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

PÁGINAS WEB

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com