B1 – Sensores

Objetivo general del bloque B

El objetivo del bloque B es implementar una serie de montajes sencillos, basados en el microcontrolador Arduino UNO, y aprender a programar su funcionamiento.

Objetivo de la sesión B1

El objetivo de esta sesión de práctica es analizar el funcionamiento de varios *sensores* y realizar la programación ***Arduino*** adecuada para medir con ellos distintos parámetros físicos.

Advertencia de precaución

Hay que poner mucha atención a la hora de polarizar (conectar) los sensores. Si la referencia (GND) y la alimentación positiva no se conectan a los pines adecuados, tanto el sensor como el microcontrolador se pueden dañar de forma irreversible.

Tipos de variables

En el lenguaje Arduino, cuando se declara una variable es imprescindible especificar su tipo. La sintaxis general de una declaración es siempre una línea escrita de la siguiente manera:

tipoVariable nombreVariable;

Existen varios tipos de variables/datos:

Data Types	Size in Bytes	Can contain:
boolean	1	true (1) or false (0)
char	1	ASCII character or signed value between -128 and 127
unsigned char, byte, uint8_t	1	ASCII character or unsigned value between 0 and 255
int, short	2	signed value between -32,768 and 32,767
unsigned int, word, uint16_t	2	unsigned value between 0 and 65,535
long	4	signed value between -2,147,483,648 and 2,147,483,647
unsigned long, uint32_t	4	unsigned value between 0 and 4,294,967,295
float, double	4	floating point value between -3.4028235E+38 and 3.4028235E+38 (Note that double is the same as a float on this platform.)

El tipo de una variable lo debemos elegir según el tipo de datos que queramos almacenar en esa variable. Asignar un valor a una variable que sea de un tipo diferente al especificado provoca un error del sketch.

El valor que puede tener una **variable de tipo "int"** es un número entero.

El valor que puede tener una **variable de tipo "float"** es un número decimal. Los números decimales se deben escribir en el sketch utilizando la notación anglosajona (es decir, utilizando el punto decimal en vez de la coma).

Función *pinMode()*

La función ***pinMode()*** configura un pin digital (cuyo número se ha de especificar como primer parámetro) como entrada o como salida, según si el valor de su segundo parámetro es la constante predefinida INPUT o OUTPUT, respectivamente.

Los pines digitales a priori pueden actuar como entrada o como salida. En el sketch, hay que definir previamente si queremos que actúen de una forma o de otra. Por ello, la función ***pinMode()*** se suele escribir dentro de "***setup()***".

Ejemplo:

```
void setup() {
  pinMode(13, OUTPUT);    // poner el digital 13 como salida
}
```

Función *digitalWrite()*

La función ***digitalWrite()*** envía un valor ALTO (HIGH) o BAJO (LOW) a un pin digital.

El pin al que se le envía la señal, se especifica como primer parámetro (escribiendo su número), y el valor concreto de esta señal se especifica como segundo parámetro (escribiendo HIGH o LOW).

Si el pin especificado en ***digitalWrite()*** está configurado como salida, el valor HIGH equivale a una señal de salida de hasta 40 mA y 5 V (o bien 3,3 V en las placas que trabajen a ese voltaje) y el valor LOW equivale a una señal de salida de 0 V.

Función *delay()*

La función ***delay()*** sirve para pausar ("congelar") el sketch cierto tiempo.

Ejemplo:

```
void setup() {
  pinMode(13, OUTPUT);    // Configurar el pin 13 como salida
}

void loop() {
  digitalWrite(13, HIGH); // activa el pin digital 13
  delay(1000);            // Espera un segundo (1000 ms)
  digitalWrite(13, LOW);  // desactiva el pin digital 13
  delay(1000);            // Espera un segundo
}
```

Función *delayMicroseconds()*

La función ***delayMicroseconds()*** pausa el programa durante el tiempo (en microsegundos) especificado por el parámetro.

Función *analogRead()*

La función ***analogRead()*** devuelve un valor proporcional a la señal analógica de entrada de un pin cuyo número se especifica como parámetro. La señal analógica de entrada debe estar comprendida entre 0 y un voltaje de referencia que, por defecto, es 5 V. El valor devuelto por la función ***analogRead()*** es un valor, de tipo entero, comprendido entre 0 y 1023 (2^{10} valores posibles). El mapeado de valores lo realiza un conversor analógico-digital de 10 bits de la propia placa Arduino. Si el voltaje de referencia es 5V, la resolución de "lectura" es 5V/1024 = 0,0049 V ≈ 5 mV.

Ejemplo:

```
int analogPin = A3;             // Lectura desde la entrada analógica A3
int val = 0;                    // variable donde se guarda el valor.

void setup() {
  Serial.begin(9600);           // setup serial
}

void loop() {
  val = analogRead(analogPin);  // lectura del valor de A3 y asignación a val.
  Serial.println(val);          // impresión del valor val
}
```

Función *analogWrite()*

La función ***analogWrite()*** envía una señal PWM (*Pulse Width Modulation*) a un pin digital, configurado como OUTPUT. El pin digital se especifica mediante el primer parámetro de la función. La señal PWM (Figura 1) queda definida por el valor que se especifica como segundo parámetro de la función.

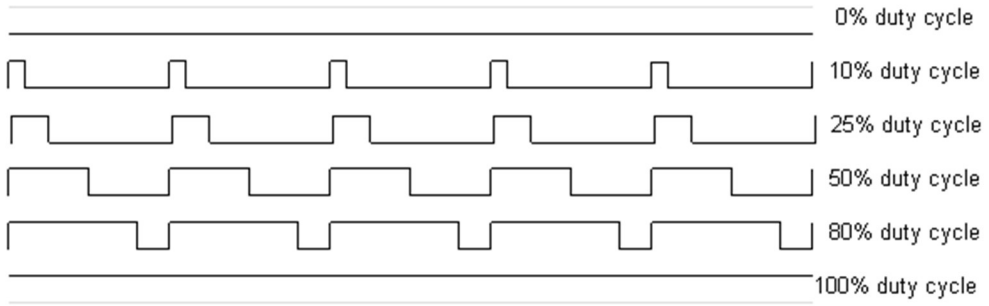

Figura 1

Las salidas PWM tienen una resolución de 8 bits, y por tanto solo pueden ofrecer 2^8=256 valores diferentes (de 0 a 255). Si el segundo parámetro de la función ***analogWrite()*** es 0, el pulso no dura nada (no hay señal) y el valor analógico correspondiente es mínimo (0 V). Si el segundo parámetro es 255, el pulso dura todo el período de la señal PWM (señal continua) y el valor analógico correspondiente es máximo (5 V). Cualquier valor entre los extremos (0 y 255) implica un pulso de una longitud intermedia.

Ejemplo:

```
int ledPin = 9;       // LED conectado al pin digital 9
int analogPin = 3;    // potenciómetro conectado al pin analógico 3
int val = 0;          // variable donde se guarda el valor
```

```
void setup() {
  pinMode(ledPin, OUTPUT);       // asignar como pin de salida
}

void loop() {
  val = analogRead(analogPin);  // leer desde el pina 3
  analogWrite(ledPin, val / 4); // valor del analogWrite desde 0 a 255
}
```

Instrucciones *map()*; y *constrain()*;

Cuando leemos una señal analógica usando ***analogRead ()***, será un número comprendido entre 0 y 1023. Pero cuando queremos manejar un pin PWM usando ***analogWrite ()***, se quiere un número de 0 a 255. Podemos "encorsetar" el rango más grande dentro del rango más pequeño usando la función ***map()***. La instrucción ***map()*** se utiliza en multitud de proyectos, para adecuar las señales de entrada generadas por sensores a un rango numérico óptimo para trabajar.

La instrucción ***map();*** modifica un valor (especificado como primer parámetro) que inicialmente está dentro de un rango (delimitado por su mínimo –segundo parámetro– y su máximo –tercer parámetro–) para que esté dentro de otro rango (delimitado por otro mínimo –cuarto parámetro– y otro máximo –quinto parámetro–) de forma que la transformación del valor sea lo más proporcional posible.

Debido a que ***map ()*** aún puede devolver números fuera de rango, se usa la función ***constrain()*** para “ajustar” números dentro de rango. Si el número está fuera del rango, lo convertirá en el número más grande o más pequeño. Si está dentro del rango, permanecerá igual. La instrucción ***constrain()*** sirve para contener un valor determinado entre dos extremos mínimo y máximo.

La instrucción ***constrain();*** recalcula el valor pasado como primer parámetro (“x”) dependiendo de si está dentro o fuera del rango delimitado por los valores pasados como segundo y tercer parámetro (“a” y “b” respectivamente, donde “a” es menor que “b”).

Función *pulseIn()*

La función ***pulseIn()*** lee un pulso (ALTO o BAJO) en un pin. Por ejemplo, si el valor es ALTO, ***pulseIn ()*** espera a que el pin pase de BAJO a ALTO, comienza a medir, luego espera a que el pin pase a BAJO y detiene el tiempo. Devuelve la longitud del pulso en microsegundos o se rinde y devuelve 0 si no se recibió un pulso completo dentro del tiempo de espera.

Sintaxis:

```
pulseIn(pin, value)
```

```
pulseIn(pin, value, timeout)
```

Instrucción *Serial.begin()*

La instrucción ***Serial.begin();*** abre el canal serie para que pueda empezar la comunicación por él. Se suele escribir dentro de la sección “*void setup()*”. Su único parámetro –obligatorio– , especifica la velocidad en bits/s a la que se producirá la transferencia serie de los datos. Para la comunicación con una computadora, se suele utilizar el valor 9600, pero se puede especificar cualquier otra velocidad.

Instrucciones *Serial.print()* y *Serial.println()*

La instrucción ***Serial.print();*** envía un dato (especificado como parámetro) desde el microcontrolador hacia el exterior a través del canal serie.

Serial.println() hace lo mismo que ***Serial.print()***, pero, además, añade al final de los datos enviados dos caracteres extra: el de retorno de carro (código ASCII nº 13) y el de nueva línea (código ASCII nº 10), de manera que al final de la ejecución de ***Serial.println()*** se efectúa un salto de línea.

Prácticas a realizar

B1a - LED parpadeante

Se pide implementar un montaje, similar al de la Figura 2, con un LED (protegido con una resistencia de 330 Ω o más, montada en serie) y conseguir, mediante programación, que el LED parpadee, controlando el tiempo que permanece encendido y el tiempo que permanece apagado.

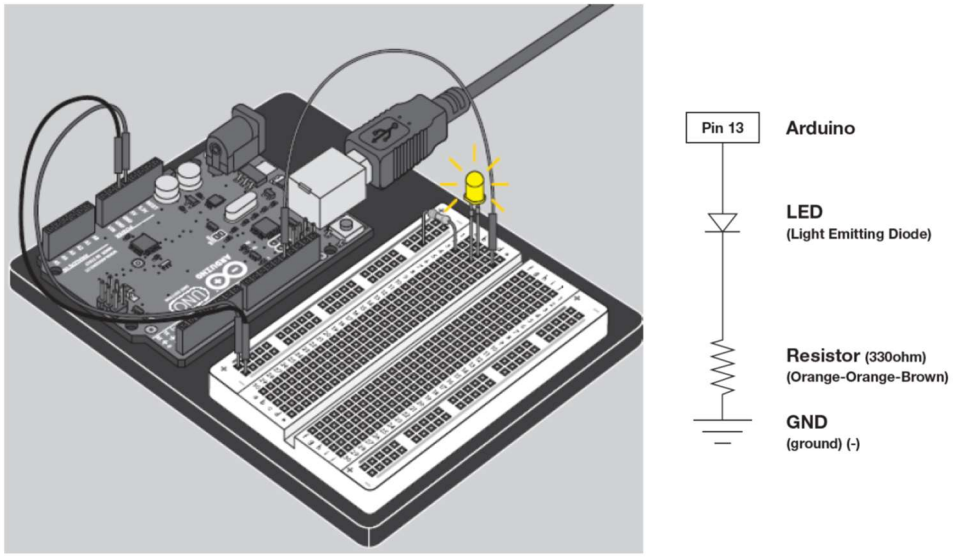

Figura 2

B1b - Detector de luz/oscuridad

Se trata de realizar el montaje de la Figura 3, correspondiente a un detector de luz/oscuridad con entrada y salida analógicas. Se debe utilizar el valor del voltaje leído en el pin analógico de entrada (A0) para iluminar el LED y visualizar el valor leído empleando el puerto serie. Se debe ver que el LED se vuelve más brillante o más tenue dependiendo de la cantidad de luz que está leyendo la fotorresistencia (montaje similar al de la sesión A3).

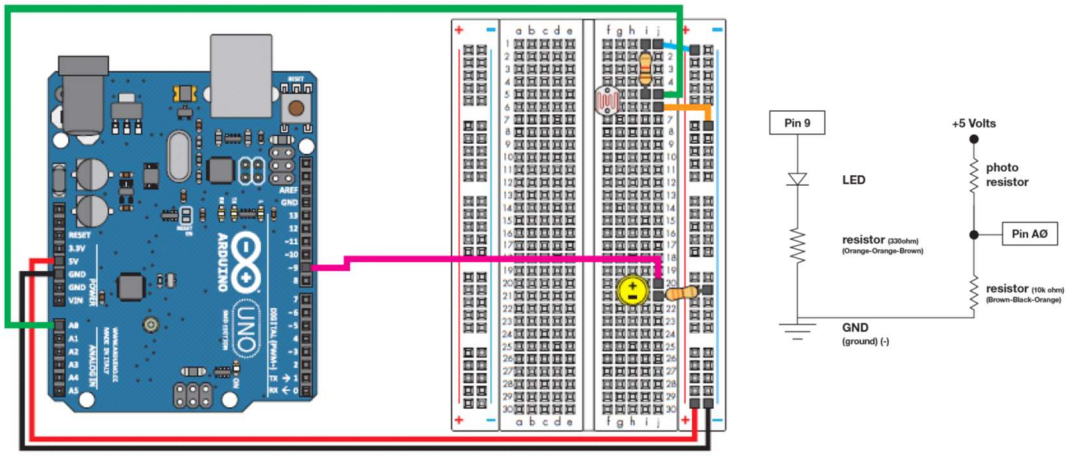

Figura 3

B1c – Sensor de temperatura TMP36

Un sensor de temperatura es simplemente un chip que nos devuelve un valor de tensión proporcional a la temperatura a la que está expuesto (Figura 4). El **TMP36** funciona entre -50°C y 125°C. Típicamente un sensor de temperatura tiene tres terminales: uno positivo (5 V), uno de tierra y uno de salida (señal). El pin central es el de señal, pero para saber cuál es GND y cuál 5V, nos debemos fijar en el encapsulado que tiene una cara plana y otra curva. Poniendo la cara plana mirando hacia vosotros con las patas hacia abajo (de modo que podáis leer el modelo), el pin de la izquierda es alimentación 5V y el otro es GND.

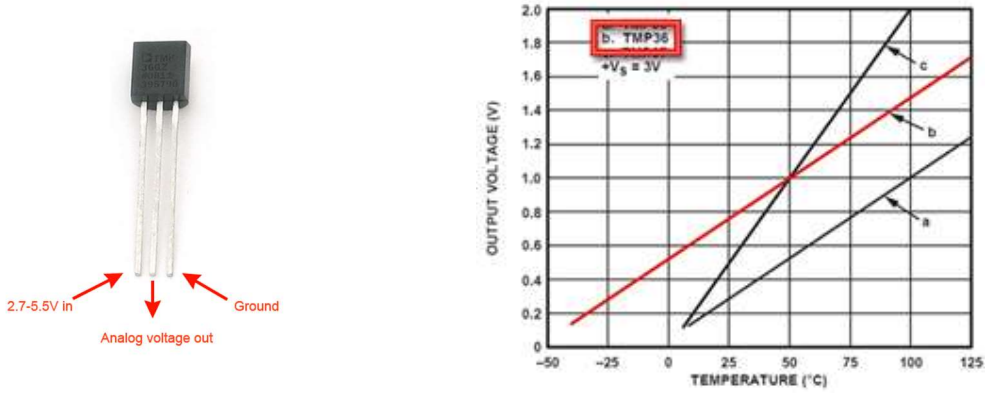

Figura 4

La conexión circuital debe ser la adecuada (como se muestra en la Figura 5). En caso contrario, el sensor se quemará de forma irreversible. (Si por error conectáis la tensión al revés, tendréis cierto tiempo de reaccionar y cambiarla, pero mucho cuidado porque se calentará y puede produciros una quemadura dolorosa. Si veis que está caliente no tratéis de sacarlo con los dedos, simplemente desconectad el cable de Arduino y esperad un rato para que se enfríe.)

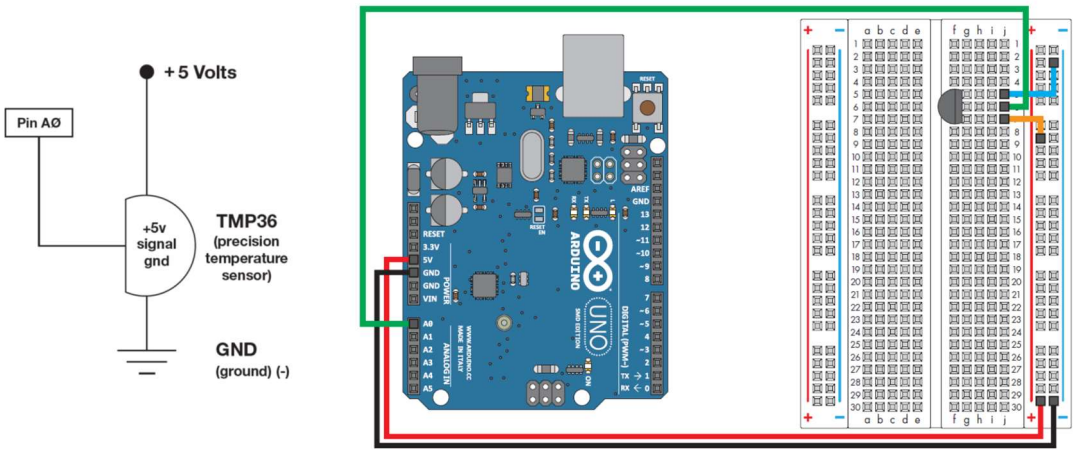

Figura 5

Sabemos que nuestro Arduino UNO mide en las entradas analógicas un máximo de 1.024 para 5V (y 0 para 0V). Por tanto, para una lectura dada, el valor en voltios es:

$$Volt = \frac{5}{1024} * lectura$$

El fabricante del TMP36 nos dice que la salida de tensión del sensor es de 10 mV por cada grado de temperatura (1 V por cada 100°C). También nos dice que 0 V no corresponden a 0°C sino a -50°C. Por tanto, la temperatura en grados Celsius está dada por:

$$TempC = \frac{5}{1024} * lectura * 100 - 50$$

Se pide, mediante programación:

- Leer la señal del sensor mediante una entrada analógica.
- Mostrar en el puerto serie la temperatura en grados centígrados.

B1d – Sensor de distancia mediante ultrasonidos HC-SR04

Cada módulo HC-SR04 (Figura 6) incluye un transmisor ultrasónico (T), un receptor (R) y un circuito de control. Este sensor envía ondas sonoras desde el transmisor, que rebotan en los objetos y regresan al receptor. Si se coloca enfrente de un objeto, se puede determinar lo lejos que está este último, por el tiempo que tardan las ondas sonoras en volver al sensor. El módulo HC-SR04 permite medir (sin contacto) distancias de 2 cm a 4 m, con una precisión de 3 mm.

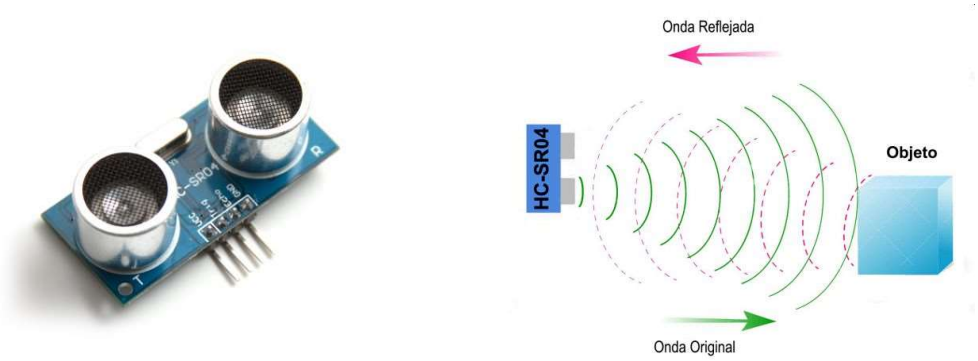

Figura 6

Las conexiones son muy sencillas (Figura 7): **V_{cc}** a 5V; **GND** a GND; **Trig** al pin 9 (por ejemplo); ***Echo*** al pin 8 (por ejemplo).

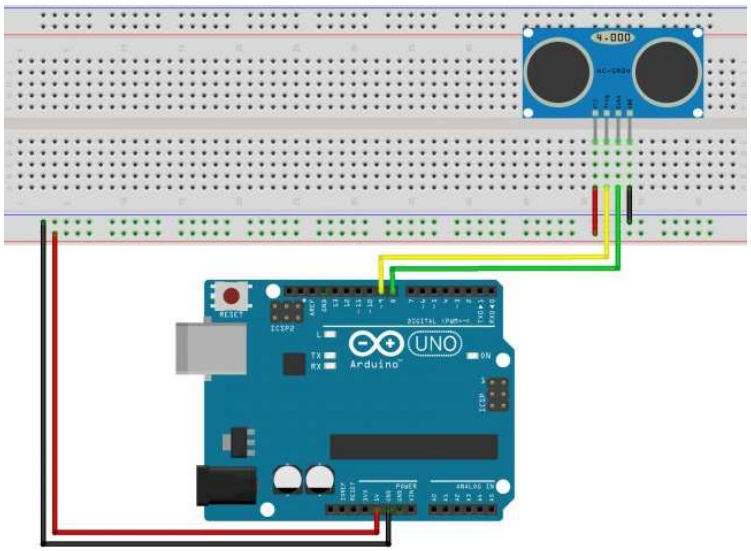

Figura 7

La Figura 8 muestra las señales que se emplean para poder determinar la distancia. Cada ciclo de medición comienza suministrando un pulso corto (señal a nivel alto durante 10 µs) a la entrada del disparador (trigger). El módulo envía entonces (automáticamente) una ráfaga de ultrasonidos de 8 ciclos a 40 kHz y detecta si hay una señal de retorno a nivel alto (eco). El tiempo que dura la señal de retorno a nivel alto corresponde al tiempo desde el envío de la señal de disparo hasta el retorno.

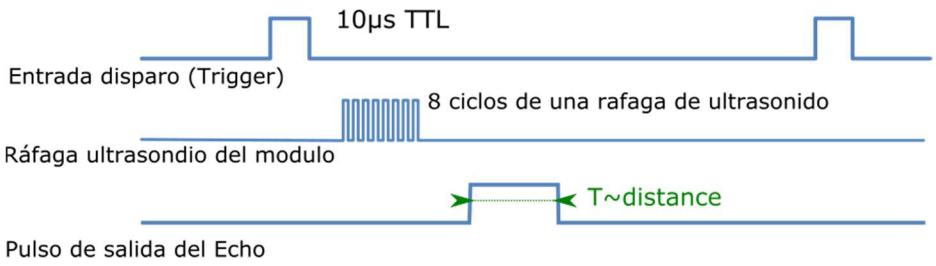

Figura 8

La distancia a la que está el objeto se puede calcular a partir del intervalo de tiempo (T) que dura la señal de retorno a nivel alto (eco):

$$distancia = T * V_{sonido} / 2$$

donde V_{sonido} es la velocidad del sonido (V_{sonido}=343 m/s).

Conviene utilizar un ciclo de medición (trigger input) de más de 60 ms, para evitar que la señal de disparo se "mezcle" con la señal de eco. La superficie del objeto no debe ser inferior a 0,5 m y debe ser lo más plana posible; de lo contrario, afectará a los resultados de la medición. Los materiales acústicamente "blandos", como tela o lana, pueden llegar a ser difíciles de detectar porque absorben el sonido. El funcionamiento del sensor no se ve afectado por la luz.

Se pide:

- Controlar (mediante programación) la ráfaga de ultrasonidos que envía el módulo hacia el objeto, y la señal (eco) que detecta el módulo tras el reflejo en el objeto.
- Mostrar, en el puerto serie, la distancia a la que está situado el objeto (expresada en cm).