Dirección General de Asuntos
del Personal Académico

TUTORIAL SOBRE ARDUINO

MIGUEL ANGEL BAÑUELOS SAUCEDO

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLÓGICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Este material fue elaborado para el proyecto PAPIME DGAPA-UNAM PE106816 (2016)

ÍNDICE

INTRODUCCIÓN

El sistema Arduino se ha convertido en una herramienta popular para desarrollar diversos proyectos electrónicos, y para introducir al usuario al uso de microcontroladores. Existe una gran cantidad de información sobre esta plataforma en internet o en libros técnicos; sin embargo, esta misma abundancia dificulta en ocasiones encontrar los datos que se requieren para comenzar a utilizar rápidamente una tarjeta Arduino. Decidimos elaborar este breve tutorial, con la intención de invitar al lector a utilizar esta herramienta electrónica, mediante una introducción sencilla a su operación. Los contenidos fueron seleccionados después de haber impartido un curso introductorio al sistema Arduino en junio de 2015. Se espera que el lector esté familiarizado con los sistemas de numeración binaria y tenga nociones de programación. De igual manera, es conveniente tener experiencia en la interpretación de diagramas de circuitos eléctricos.

1. LA TARJETA ARDUINO

El sistema Arduino consiste en una tarjeta electrónica y un ambiente de programación. Arduino está formado por una familia de tarjetas con diferentes capacidades y costos. Arduino UNO es una tarjeta de gama baja, adecuada para proyectos sencillos y para un primer contacto con este tipo de sistemas. El microcontrolador es el dispositivo encargado de almacenar y ejecutar el programa desarrollado por el usuario. La tarjeta Arduino UNO está basada en un microcontrolador de la compañía ATMEL (ver Fig. 1). La tarjeta, además del microcontrolador, cuenta con componentes necesarios para la operación del mismo, entre ellos un circuito para alimentación de voltaje, así como puerto USB (que se utiliza para la programación desde una computadora personal) y terminales de conexión para conectarlo a sensores, motores, focos, etc.

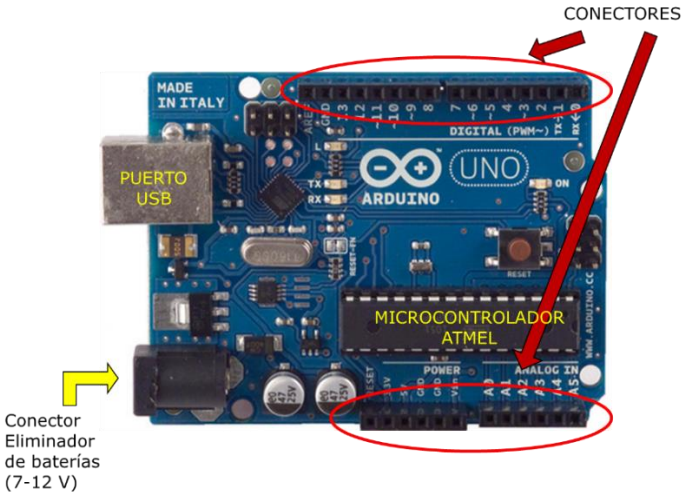


Fig. 1. Fotografía de la tarjeta Arduino UNO donde se señalan los conectores principales.

La tarjeta Arduino cuenta con una entrada para conectarle un eliminador de baterías. Mediante ese conector es posible alimentar también con una pila de 9 V.

El puerto USB sirve para conectar la tarjeta Arduino a una computadora personal y desde ahí programarla utilizando un lenguaje propio del Arduino. Este programa (Arduino Software IDE) es gratuito y se puede descargar de la dirección electrónica www.arduino.cc . Existen versiones para Windows, Linux y Mac OS X. En el caso de Windows se recomienda descargar la versión Windows Installer. El puerto USB también le proporciona voltaje a la tarjeta Arduino, entonces como alimentación de energía se puede usar el puerto USB, o el conector de alimentación que acepta voltajes de 7 a 12 V corriente directa. Otra opción, es utilizar la conexión etiquetada V_{in}, por donde también se puede suministrar un voltaje de 7 a 12 V de corriente directa.

Los programas para el Arduino reciben el nombre de Skecht. Cuando se crea un nuevo programa, el Arduino IDE incluye los bloques básicos del programa: *setup* y *loop* (ver Fig. 2. Ventana del Arduino IDE que muestra los bloques de programación básicos.). El bloque *setup* se ejecuta primero y una sola vez. Este bloque sirve para inicializar el programa y la tarjeta Arduino. Por

ejemplo, se pueden inicializar variables, definir el modo de operación de los pines (conexiones) y llamar bibliotecas de funciones. Se utiliza un par de llaves para delimitar las líneas de programa que corresponden al bloque (ver Fig. 3). El bloque *loop* se ejecuta continuamente, es decir, una vez que se ejecuta la última línea de programa de ese bloque, vuelve a comenzar por ejecutar la primera línea y se continua con todas las siguientes, hasta que el proceso se repite. Los pares de paréntesis redondos después de las palabras *setup* y *loop* implican que no se les pasa ningún dato a las funciones *setup* y *loop.* La palabra *void* que antecede a las palabras *setup* y *loop* significan que dichas funciones no regresan ningún dato. El texto a la derecha de la doble diagonal '//' es un comentario y no se interpreta como instrucciones del programa. Una breve descripción del concepto de funciones en programación se puede encontrar en el apéndice A.

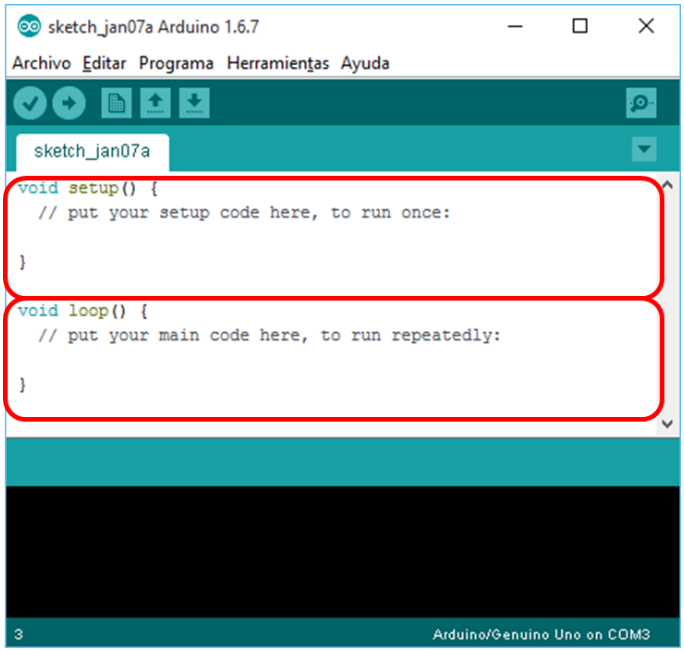


Fig. 2. Ventana del Arduino IDE que muestra los bloques de programación básicos.

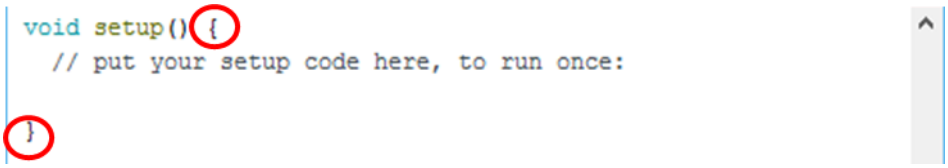


Fig. 3. Bloque de código setup *(inicialización). Los círculos rojos señalan a las llaves que delimitan el espacio del programa que contiene las instrucciones de inicialización.*

2. ENTRADAS Y SALIDAS DIGITALES

Los conectores de la tarjeta Arduino proporcionan acceso para señales digitales de entrada o salida de manera indistinta (pero que se debe configurar de manera adecuada antes de utilizar la señal). Una señal digital es aquella que solo puede tener dos valores: encendido o apagado, justo como funciona un foco en una lámpara común. El estado de encendido o apagado se puede asignar indistintamente a los números '0' y '1' de un sistema binario. En el caso de la tarjeta Arduino UNO, estos estados están representados por niveles de voltaje. En tarjetas cuyo microcontrolador opera a 5 V, los voltajes cercanos a 5 V se consideran como encendido, y los valores cercanos a 0 V como apagado.

En la Fig. 4 se muestran los conectores de la tarjeta Arduino donde se ubican las señales de entrada y salida digital. Estos se aprecian numerados del 0 al 13. Cada una de estas conexiones pueden funcionar como una entrada digital o una salida digital. Como entrada podrían utilizarse para detectar si un botón ha sido presionado o no, y como salida pueden usarse para prender un LED.

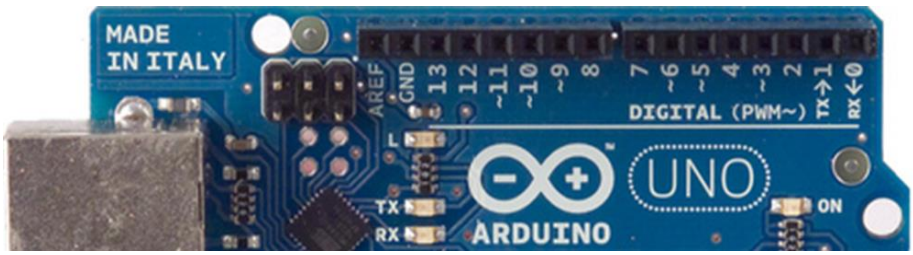


Fig. 4. Detalle de la tarjeta Arduino UNO mostrando los conectores de entrada y salida digital.

En la Fig. 4 también se observa que algunas terminales de conexión llevan una tilde ' ~ ' junto al número de conexión. Esta tilde indica las conexiones que pueden generar una señal PWM (Pulse Width Modulation), la cual es una señal cuadrada que se puede usar para, por ejemplo, controlar la velocidad de un motor. Una explicación breve de estas señales se proporciona en el apéndice B.

Una forma sencilla de construir una entrada digital es utilizando un interruptor de presión (*push-button)*, como se muestra en la Fig. 5A. Cuando el interruptor es presionado, el circuito se cierra y la terminal de salida queda conectada a 5 V (ver Fig. 5B). Por otro lado, cuando el interruptor es liberado, se abre la conexión a 5 V. La terminal de salida está entonces conectada a tierra (0 V) a través de la resistencia R; esto produce un voltaje de salida cercano a los 0 V, donde el valor exacto depende de los valores de la resistencia R, y de la configuración que presente el circuito que se conecte en el punto de salida. Un valor típico de la resistencia que se emplea en estos casos es R= 10 kΩ. El utilizar un valor menor puede resultar en una demanda excesiva de corriente a la fuente de alimentación.

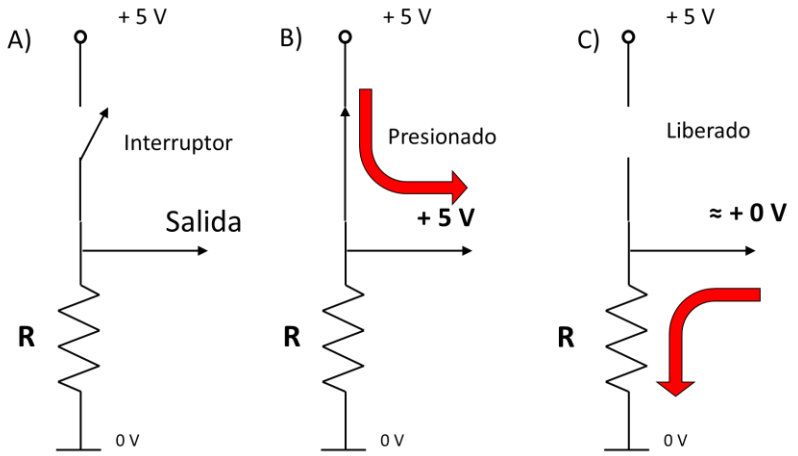


Fig. 5. Generación de una señal digital con un interruptor de presión. A) Diagrama esquemático. B) La salida es 0 V cuando el interruptor es presionado. C) La salida es casi 5 V cuando el interruptor está liberado.

Uno de los modelos de interruptor de presión más común se muestra en la Fig. 6, junto con sus diagramas esquemáticos físicos y de conexión. Debe tomarse en cuenta que las cuatro patillas se encuentran conectadas por pares, es decir, es necesario identificar las patillas que entrarán en contacto eléctrico cuando se presione el botón. Para el interruptor de la figura, las patas más cercanas son las que se conectan al presionar el botón.

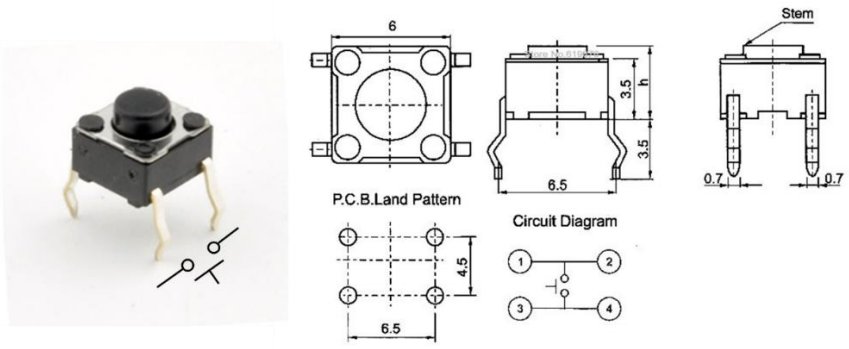


Fig. 6. Fotografía de un interruptor de presión y su correspondiente diagrama físico y eléctrico (Tomado de Internet).

El ejemplo más sencillo de la utilización de una señal de salida digital consiste en encender un foquito, o más comúnmente un LED. El LED es un dispositivo de que emite luz, pero presenta diferencias de funcionamiento respecto a un foquito. Los LED pueden emitir luz de diferentes colores, y en versiones microscópicas forman parte de muchas pantallas digitales en la actualidad. Un foco puede conectarse de manera indistinta a las terminales positiva y negativa de una fuente de voltaje. Pero esto no sucede con un LED, el cual requiere una conexión particular a las terminales positiva y negativa, y además necesita una resistencia para limitar el flujo de corriente eléctrica (ver Fig. 7). Aplicar más de 5 V a un LED con la conexión invertida puede destruirlo.

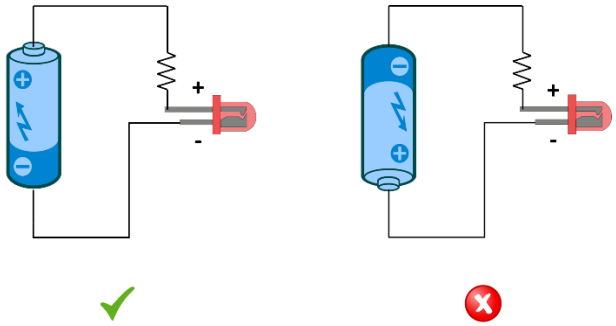


Fig. 7. Ilustración de la conexión correcta de un LED.

Un LED tiene dos terminales llamadas ánodo y cátodo, que corresponden a una terminal positiva y negativa respectivamente. La identificación de las terminales es posible por dos métodos: En el primero, si se observa el LED desde la parte superior, se puede ubicar una muesca plana que señala al cátodo. En el segundo, y considerando que no se han trozado las patillas del dispositivo, entonces la más larga indicará al ánodo (+) y la más corta al cátodo (-) (ver Fig. 8).

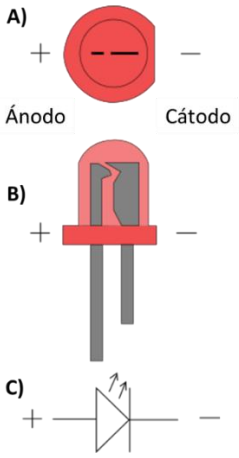


Fig. 8. Detalle de las terminales de un LED. A) Vista superior. B) Vista lateral. C) Símbolo eléctrico.

Para calcular el valor adecuado de la resistencia asociada al LED, es necesario conocer dos valores: el voltaje típico de operación del LED y la corriente típica de operación. Valores comunes son 2.2 V para el voltaje de operación y 10 mA para la corriente. De esta manera, si se considera un voltaje de alimentación de 5 V, la resistencia estimada es de 280 Ω, según se muestra en la Fig. 9. En la práctica es común utilizar resistencias de 220 Ω, o 330 Ω; aunque 270 Ω es un valor más cercano al calculado y también corresponde a un valor comercial. Debe recordarse que las resistencias se fabrican con ciertos valores que en el caso más común corresponden a la serie: 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, y 82 (estándar E12). El valor de una resistencia se identifica mediante unas bandas de colores (ver apéndice C).

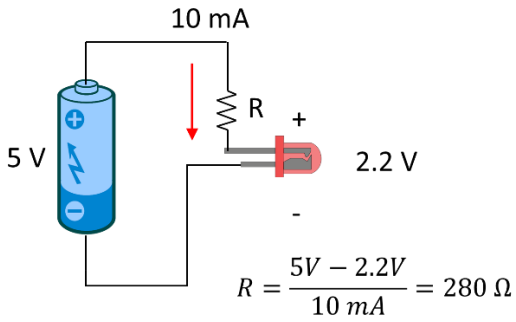


Fig. 9. Cálculo de la resistencia asociada a un LED.

Un ejemplo sencillo de uso de señales digitales de entrada y salida con el Arduino consiste en encender el LED cuando el Arduino detecta que se ha presionado el botón (ver Fig. 10). Para armar el circuito se requiere una resistencia de 10 kΩ (bandas: café, negro, naranja, oro), y una de 220 Ω (bandas: rojo, rojo, café, oro).

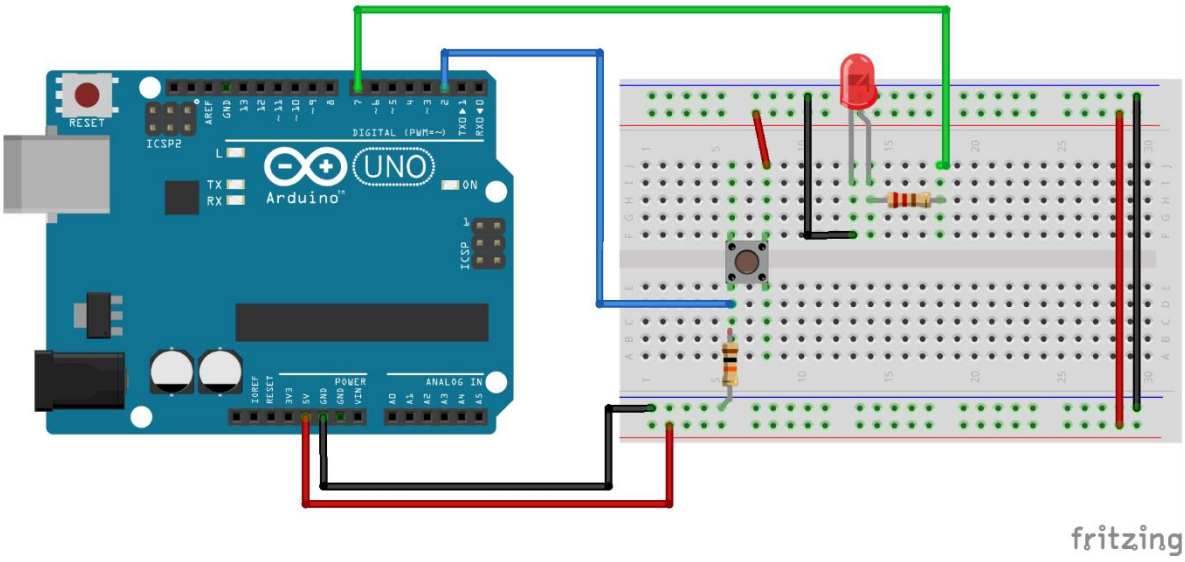


Fig. 10. Diagrama de conexiones de un circuito con interruptor y LED.

La placa donde se insertan los componentes se denomina *protoboard* y tiene un arreglo de conexiones internas según se muestra en la Fig. 11. Las líneas verticales están conectadas por grupos de 5 orificios, y las líneas horizontales por grupos de 25 (o más dependiendo del tamaño).

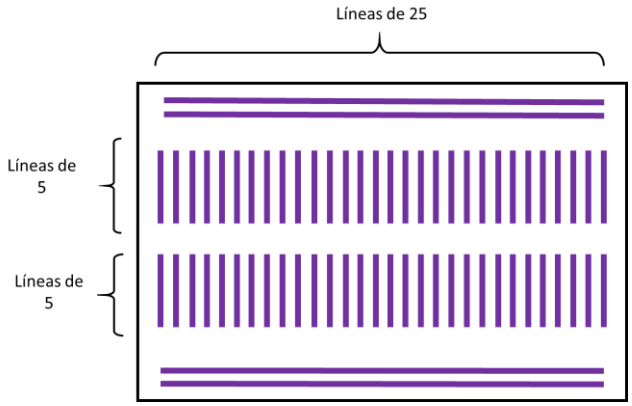


Fig. 11. Diagrama de conexiones internas de una protoboard.

El programa para este ejercicio se encuentra en el catálogo de ejemplos del Arduino, para cargarlo basta con seguir el menú *Archivo-> Ejemplos -> Digital -> Button* (ver Fig. 12).

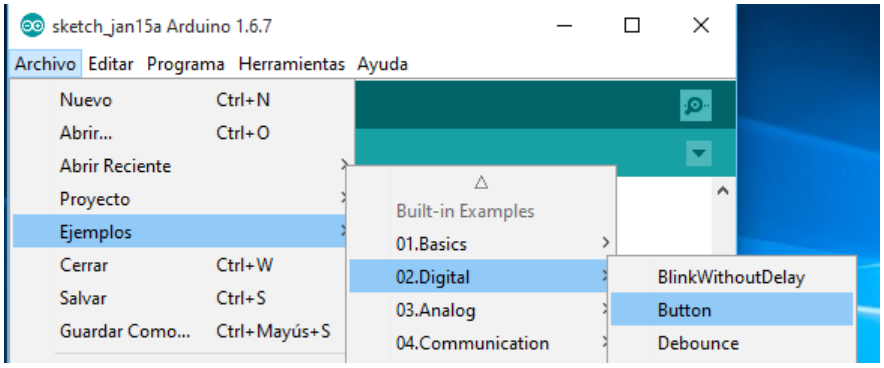


Fig. 12. Cargando un programa ejemplo.

La primera parte del programa es una sección de comentarios (texto que no es interpretado como instrucciones). Esta sección está delimitada por los pares de caracteres /* y */, los cuales marcan el inicio y fin de la sección de comentarios (ver Fig. 13).

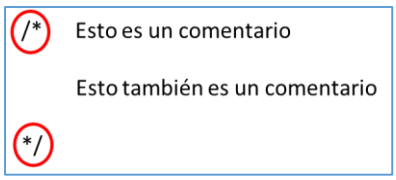


Fig. 13. Ejemplo de un bloque de comentarios de texto.

Después de la sección de comentarios, aparece una sección donde se definen algunas constantes del programa (ver Tabla 1). Se define una constante de valor entero *buttonPin=2.* Se denomina constante, porque su valor no cambiará en el resto del programa. Esto nos ayudará a definir la terminal digital número 2, como la encargada de recibir la señal del interruptor de presión. En lo sucesivo el programa, cada vez que encuentre la palabra *buttonPin,* la reemplazará por número entero 2. De manera similar se define la constante entera *ledPin=13.* Para hacer compatible el programa con el circuito mostrado en la Fig. 10, deberemos cambiar la instrucción para que quede *ledPin=7.*

A continuación aparece una sección de declaración de variables globales, es decir, serán variables válidas en todos los demás bloques de programa (ver Tabla 2). Con la instrucción *buttonState=0*, se declara una variable entera y se le asigna un valor inicial de cero (esta asignación es opcional). Observe que ya no se utiliza la palabra *const*, la cual fue usada para definir las constantes del bloque precedente. Esta variable será utilizada para almacenar el valor del botón.

Tabla 1. Bloque de declaración de constantes.

```
// Las constantes no cambian
// Se definen los pines que se utilizarán:
const int buttonPin = 2;     // El pin 2 corresponde al botón
const int ledPin =  13;      // El pin corresponde al LED
```

Tabla 2. Bloque de declaración e inicialización de variables.

```
// Las variables sí cambian:
int buttonState = 0;         // Variable para leer el estado del botón
```

A continuación podemos grabar el programa del Arduino en nuestra computadora utilizando el menú *Archivo -> Guardar como*, y seleccionando un nombre adecuado a nuestro proyecto (p. ej. LED_boton). Después podemos verificar el código de nuestro programa con el botón izquierdo de la barra de botones (ver Fig. 14). En caso de que el código no tenga errores, se procede a subir el programa a la tarjeta Arduino utilizando el botón que tiene un dibujo de una flecha hacia la derecha. Para evitar errores de comunicación con la tarjeta, es conveniente verificar que está correctamente seleccionada en el menú "Herramientas -> Placa", donde debe aparecer el modelo de nuestra tarjeta (p. ej. Arduino UNO). También debemos verificar que se ha seleccionado el puerto correcto de comunicación USB, mediante el menú "Herramientas -> Puerto:" (p. ej. COM4 (Arduino/Genuino UNO). Después de revisar esta configuración ya podemos subir el programa a la tarjeta Arduino (con el botón -flecha a la derecha-). Finalmente probaremos que el LED se enciende cada vez que presionamos el botón.

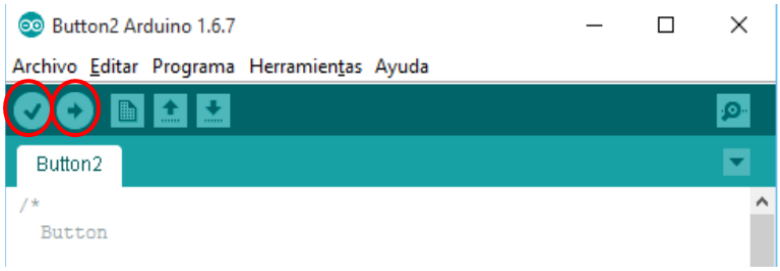


Fig. 14. Barra de botones del ambiente Arduino. Se señalan con un círculo rojo los botones para verificar (izquierda) y subir el programa(derecha).

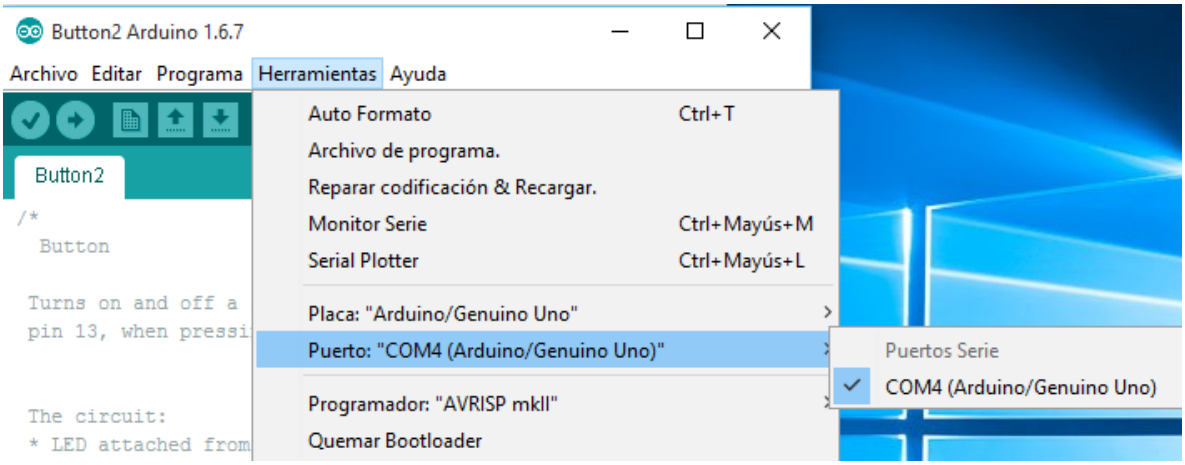


Fig. 15. Menú de herramientas mostrando las opciones Placa Placa *y* Puerto *configuradas para utilizar una tarjeta Arduino UNO.*

La rutina principal del programa se muestra en la Tabla 4. La primera instrucción

buttonState = digitalRead(buttonPin);

Lee el pin 2 de la tarjeta Arduino (recordar que se definió previamente la constante buttonPin=2) y el resultado se asigna a la variable buttonState. La lectura de un pin digital solo puede arrojar dos resultados cero o uno, que se interpretan como LOW o HIGH por el programa, respectivamente.

La instrucción

if (buttonState == HIGH)

sirve para preguntar si el botón fue presionado (y por lo tanto se lee un voltaje ALTO en el pin). En caso afirmativo se ejecuta la instrucción que enciende el LED:

digitalWrite(ledPIN, HIGH);

y en caso contrario la instrucción para apagar el LED:

digitalWrite(ledPin, LOW);

El programa puede modificarse para hacer que el LED parpadee cada vez que es presionado el botón. Para ello, deberán cambiarse las instrucciones dentro de la sentencia IF y para que quede como se muestra en la Tabla 3. El resto del programa se conserva sin cambios.

Tabla 4. Rutina principal del programa de encendido de un LED.

```
void loop() {
 // Lee el estado del botón
 buttonState = digitalRead(buttonPin);

 // Revisa si el botón fue presionado
 if (buttonState == HIGH)  {       // Pregunta si buttonState es HIGH:
  // Entonces enciende el LED:
  digitalWrite(ledPin, HIGH);
 }
else
{               // En caso contrario
  // Apaga el LED
  digitalWrite(ledPin, LOW);
 }
}
```

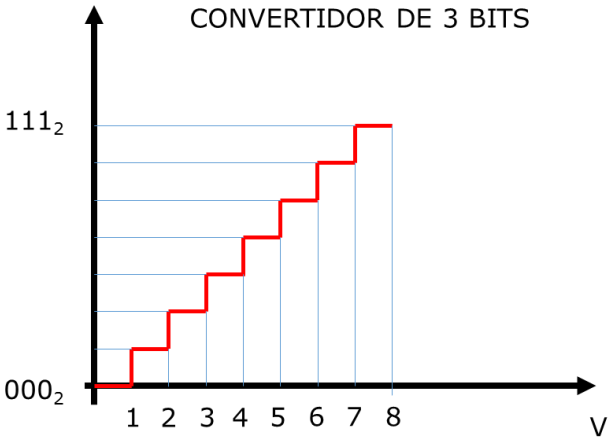
Tabla 3. Código que se modifica para hacer que el LED parpadee.

```
if (buttonState == HIGH) {
   // Se enciende el LED y parpadea:
   digitalWrite(ledPin, HIGH);   // Se enciende el LED)
   delay(300);            // Espera 0.3 segundos
   digitalWrite(ledPin, LOW);    // Apaga el LED
   delay(300);            // Espera 0.3 segundos
  }
else {
```

3. ENTRADAS Y SALIDAS ANALÓGICAS

La tarjeta Arduino UNO cuenta con 6 pines de entrada analógica, etiquetados de A0 a A5. Usualmente se utilizan para leer voltajes entre 0 y 5 V CD. La lectura se hace por medio de un convertidor analógico digital interno de 10 bits, es decir, entrega códigos que van de 0 000 000 000 a 1 111 111 111, es decir, entre 0 y 1023 en base decimal.

Una señal analógica puede tomar cualquier valor entre sus límites. Por ejemplo, si la señal varía entre 0 y 5 V, entonces un valor posible es 2.5, y otro sería 2.500001. Los valores de una señal digital, en cambio, están limitados por el número de bits que se utilizan. En el caso de un convertidor analógico-digital de 3 bits, solo hay 8 posibles combinaciones, tal y como se muestra en la columna derecha de la tabla de la Fig. 16. En esa misma figura, se ilustra el caso en que una señal analógica de 0 a 8 V se convierte a su valor digital. Cualquier valor de voltaje entre 1 V y 2 V generará el mismo código binario (001_2), y ocurre algo similar para cualquier valor entre los voltajes 2 V y 3 V. Esto produce una respuesta escalonada. Al proceso de tomar una señal analógica (con un número infinito de valores) y convertirla en una señal con solo un número limitado de valores se le denomina *cuantización.*



VOLTAJE	CÓDIGO BINARIO
0-1	000
1-2	001
2-3	010
3-4	011
4-5	100
5-6	101
6-7	110
7-8	111

Fig. 16. Ejemplo del funcionamiento de un convertidor analógico-digital de 3 bits.

En el caso del convertidor analógico-digital de la tarjeta Arduino UNO, el proceso de cuantización a 10 bits se vuelve tan fino, que la gráfica no se vería escalonada, sino continua (ver Fig. 17).

Para ejemplificar el uso de una entrada analógica utilizaremos un sensor de temperatura. El circuito integrado LM35 es un dispositivo que se puede alimentar con 5 V y entrega en su terminal de salida un voltaje que es proporcional a la temperatura en grados centígrados (10 mV/°C) (ver Fig. 18).

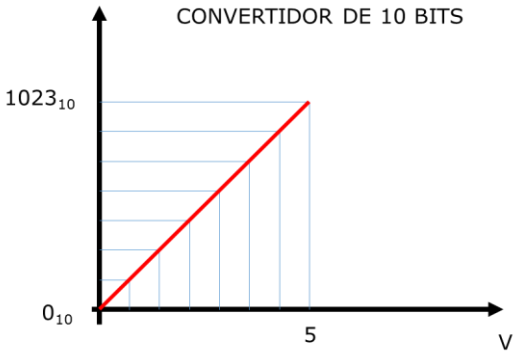


Fig. 17. Gráfica de cuantización de un convertidor analógico-digital de 10 bits.

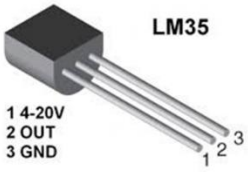


Fig. 18. Sensor de temperatura LM35 (Fuente: Internet).

Para su utilización podemos seguir el diagrama de conexiones mostrado en la Fig. 19. La terminal de salida del sensor de temperatura (terminal central) se conecta a una de las entradas analógicas de la tarjeta Arduino (A0).

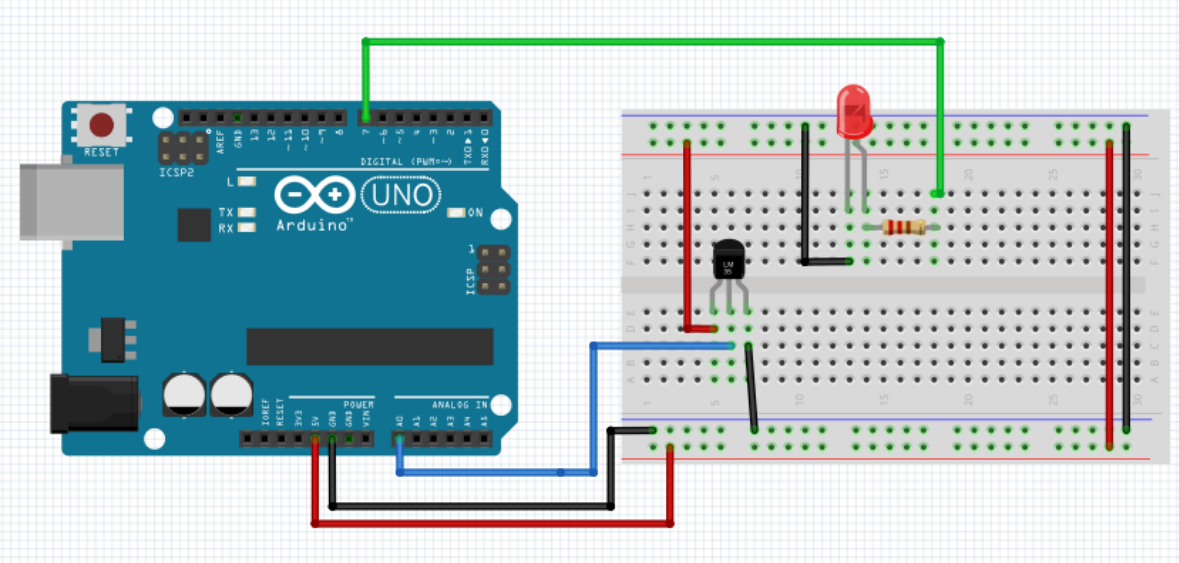


Fig. 19. Esquema de conexiones del sensor de temperatura y un LED.

En la Tabla 5, se muestra el código de un programa que lee el sensor de temperatura y enciende un LED cuando la temperatura es mayor a 25 °C. Para comprobarlo, basta con presionar con los dedos el sensor para que su temperatura aumente. La instrucción que se encarga de leer al sensor es

```
valorSensor = analogRead(pinSensor);
```

El valor que toma la variable *valorSensor* estará entre 0 y 1023, aunque en la práctica corresponda a un intervalo menor debido a que no se alcanzan temperaturas lo suficientemente altas como para que el sensor entregue una señal de 5 V. Obsérvese que la primera sentencia del bloque *loop* (programa principal) es

```
float milivolts;
```

esta instrucción declara (define) una variable de punto flotante (y que solo puede ser usada dentro del bloque *loop*).

Tabla 5. Programa que lee un sensor de temperatura LM35 y detecta un umbral

```
/*
    Ejemplo con el sensor de temperatura LM35
    Si la temperatura supera un valor umbral se encenderá un LED
 */
// En esta sección se definen las constantes
            const int pinSensor = A0;    // Entrada analógica. Se conecta al sensor de temperatura
            const int pinLed    = 7;     // Pin de salida digital para el LED
            const int umbral    = 25;    // Umbral de la temperatura en grados centígrados
            int valorSensor = 0;         // Variable para guardar el valor leído del sensor
            float temperatura = 0;       // Variable para guardar la temperatura
// Sección de inicialización
            void setup() {
            pinMode(pinLed, OUTPUT);   // Se define el pin 7 como salida
            }
// Programa principal
            void loop() {
              float milivolts;   // Se define una variable de punto flotante
                valorSensor = analogRead(pinSensor);   // leemos el valor del sensor
                float milivolts = 5000*(valorSensor / 1023.0);  // Se convierte a milivolts
                temperatura = milivolts/10;    // Se convierte a grados centígrados
                if (temperatura > umbral){     // Se compara la temperatura con el umbral
                  digitalWrite(pinLed, HIGH);  // Si se supera el umbral se enciende el LED
                } else {
                  digitalWrite(pinLed, LOW);     // En caso contrario se apaga el LED
                }
              delay(1000);   // Se introduce un retardo de 1 segundo antes de repetir el programa
            }
```

El Arduino puede generar señales de pulsos modulados (PWM, *pulse width modulation)* que funcionan como *salidas analógicas* en ciertas condiciones. Estas señales de PWM se pueden utilizar para controlar la intensidad de un LED o la velocidad de un motor de corriente directa. En el Apéndice B se encuentra una breve explicación de este tipo de señales.

Como ejemplo de salida analógica se propone controlar la intensidad de un LED por medio de un potenciómetro. El potenciómetro es un dispositivo resistivo con tres terminales. El símbolo eléctrico se muestra en la Fig. 20. Al rotar el vástago se modifica la resistencia entre las terminales A-W y B-W. El diagrama de conexiones se muestra en la Fig. 21. El circuito del LED se ha conectado al pin 6 que tiene la capacidad de producir una señal PWM (se indica con una tilde ~ junto al número).

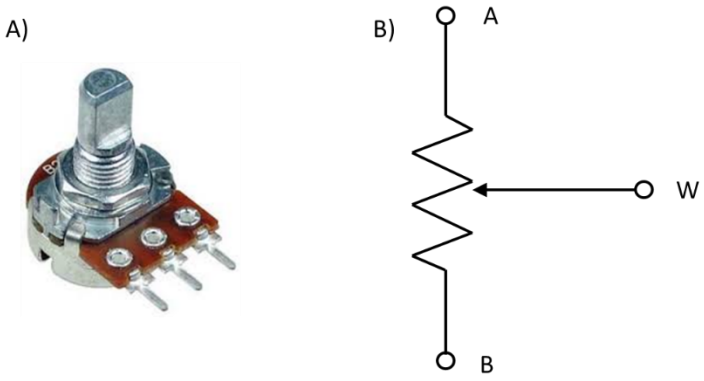


Fig. 20. Potenciómetro. A) Fotografía (Fuente: Internet). B) Diagrama eléctrico.

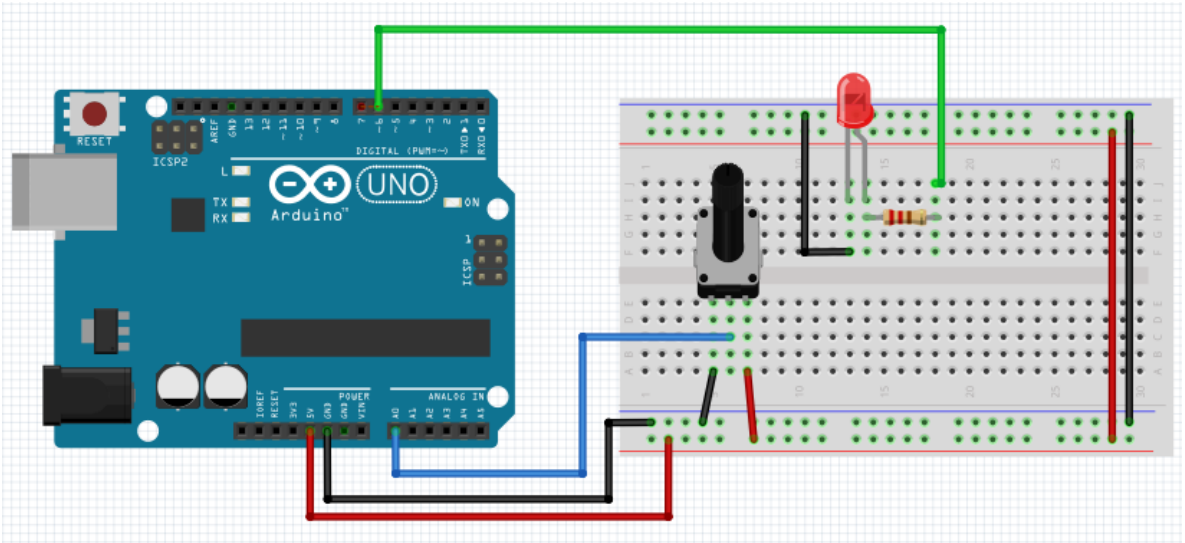


Fig. 21. Esquema de conexiones para el LED y el potenciómetro.

El programa se muestra en la Tabla 6. La instrucción

```
analogWrite(pinLed, brilloLed);
```

es la encargada de generar la señal de PWM. Esta instrucción acepta valores entre 0 y 255. Como la lectura del potenciómetro produce un dato entre 0 y 1023, se hace un re-escalamiento del valor mediante la instrucción

```
brilloLed = map(valorPotenciometro, 0, 1023, 0, 255);
```

Tabla 6.Código del programa que controla la intensidad de un LED con un potenciómetro.

```
/*   Programa que el valor de un potenciómetro
          y lo usa para ajustar el brillo a un led usando una señal PWM    */
// Declaración de constantes
          const int pinSensor = 0;   // Sensor analógico conectado al potenciómetro
          const int pinLed    = 6;   // Salida PWM conectada al LED
// Declaración de variables
          int brilloLed        = 0;  // Almacena el valor de brillo que se dará al LED
          int valorPotenciometro = 0;  // Almacena el valor leído del potenciómetro

// Inicialización
          void setup() {
            pinMode(pinLed, OUTPUT);    // Define al pin 6 como salida PWM
          }
// Programa principal
          void loop() {
            valorPotenciometro = analogRead(pinSensor);  // Lee el valor del potenciómetro
            brilloLed = map(valorPotenciometro, 0, 1023, 0, 255);  // Reescala el valor del pot
            analogWrite(pinLed, brilloLed);  // Se ajusta la salida del pin PWM
            delay(100);     // Retardo de 100 milisegundos
          }
```

Es conveniente mencionar que la instrucción *analogWrite* no tiene nada que ver con los pines de entrada analógica ni con la instrucción *analogRead.*

4. COMUNICACIÓN SERIAL

El puerto USB que se utiliza para programar el Arduino también se puede usar para establecer comunicación con una computadora personal (PC). De esta manera, con solo algunas instrucciones y un programa instalado en la PC es posible enviar y recibir cadenas de caracteres.

Como ejemplo elaboraremos un programa que envíe a la PC la temperatura registrada por un sensor de temperatura. Para ello, necesitamos el mismo circuito de la Fig. 19 y el programa de la Tabla 7. Se utiliza la instrucción *Serial.print* para enviar letreros y datos a la PC. La instrucción

Serial.println

añade una instrucción de cambio de línea (retorno de carro).

Tabla 7. Código del programa para leer un sensor LM35 y enviar el dato por puerto serie a la PC.

```
/*  Programa que utiliza un sensor de temperatura LM35, lee
 *  la temperatura y envía el dato por el puerto serial a la PC */

// Declaración de constantes
   const int pinSensor = A0;     // pin del sensor de temperatura LM35

// Declaración de variables
   int valorSensor = 0;         // Guarda el valor leído del sensor
   float temperatura = 0;       // Guarda el valor de la temperatura

// Inicialización
   void setup() {
      pinMode(pinLed, OUTPUT);  // Define el pin de salida
      Serial.begin(9600);       // Velocidad de la comunicación serial
   }

// Programa principal
   void loop() {
      valorSensor = analogRead(pinSensor);   // Lee el valor del sensor de temperatura
      float milivolts = 5000*(valorSensor / 1023.0);   // Convierte el valor a miliVolt
      temperatura = milivolts/10;   // Calcula el valor en grados centígrados
      Serial.print("La temperatura es de: ");     // Envía un letrero a la PC
      Serial.print(temperatura);                  // Envía el dato de la temperatura
      Serial.println(" grados centigrados");      // Envía un letrero a la PC
      delay(1000);  // Retardo de 1000 milisegundos
   }
```

Para poder visualizar los datos que recibe la PC se activa la función de monitor serial, dentro de la barra de funciones (ver Fig. 22). Los resultados entonces serán desplegados como se muestra en la Fig. 23.

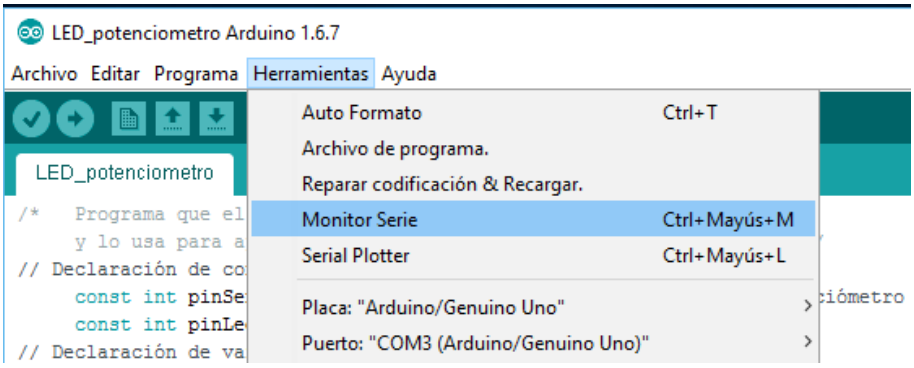


Fig. 22. Activación del Monitor Serie dentro del menú de Herramientas.

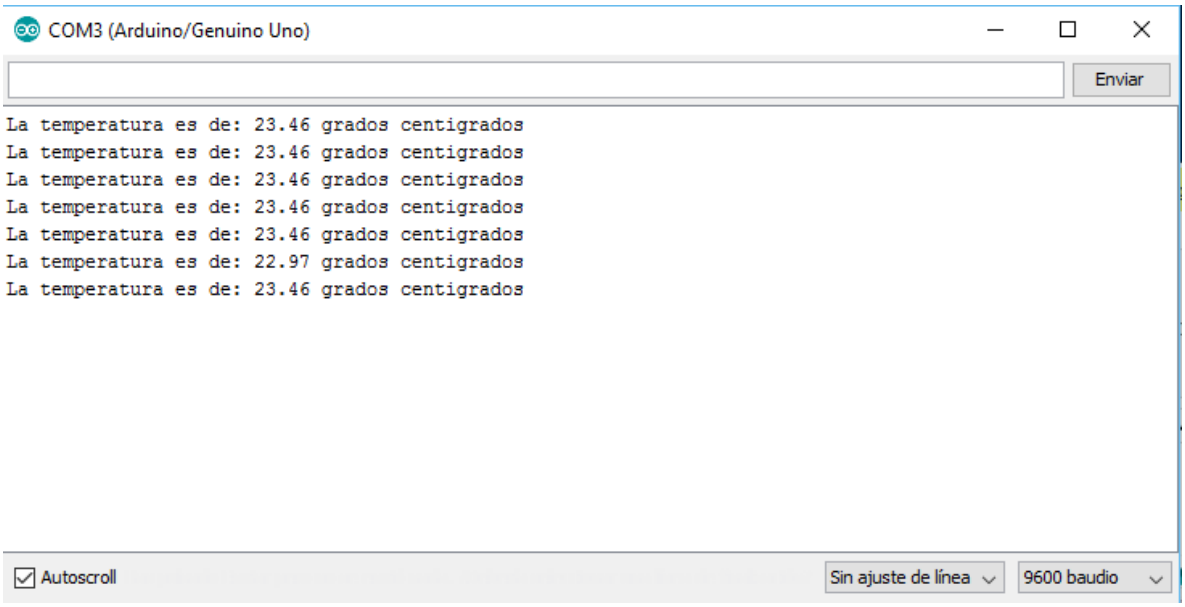


Fig. 23. Ventana del Monitor Serial desplegando los datos de la temperatura.

Se puede utilizar el mismo circuito de la Fig. 19 y elaborar un programa para controlar el encendido del LED mediante las teclas S y N, en el teclado de la computadora. El código se muestra en la Tabla 8.

Tabla 8. Código de un programa para controlar el encendido de un LED desde la PC.

```
/*  Programa que utiliza la comunicación serial
 *  para encender un LED */
// Declaración de constantes
   const int pinLed    = 7;     // pin del LED
// Declaración de variables
   int valorLed = 0;         // Guarda el estado del LED, encendido o apagado
   int datoRecibido = 0;       // Es el dato que se recibe por el puerto serial
// Inicialización
   void setup() {
      pinMode(pinLed, OUTPUT);  // Define el pin de salida
      Serial.begin(9600);      // Velocidad de la comunicación serial
      Serial.println(" Encender el LED (Y/N)");   // Envía un letrero a la PC
   }
// Programa principal
   void loop() {
      if (Serial.available() ) {
         datoRecibido = Serial.read();
         if (datoRecibido == 'S' or datoRecibido == 's') {
           digitalWrite(pinLed, HIGH);    // Si se recibe S o s enciende el LED
           Serial.println("LED encendido");
         }
         else if (datoRecibido == 'N' or datoRecibido == 'n') {
           digitalWrite(pinLed, LOW);     // Si se recibe N o n se apaga el LED
           Serial.println("LED apagado");  }
         else {
           Serial.println("Tecla incorrecta");
           }
         }
      delay(1000);   // Espera un segundo
   }
```

Una opción más interesante que el Monitor Serie del Arduino es el programa CoolTerm, el cual cuenta con funciones de registro de datos (es decir, se pueden guardar todos los datos capturados en un archivo), y se puede descargar de la siguiente liga

http://freeware.the-meiers.org/

Esto nos permite descargar el archivo CoolTerm_Win.zip. En el Apéndice D, se encuentra una breve explicación de cómo descomprimir el archivo.

Para utilizar el nuevo programa de comunicación serial podemos modificar el programa que lee la temperatura de un sensor LM35 y envía el dato por el puerto serial (Tabla 7), con unos pequeños cambios en la rutina principal, para hacerlo compatible, y que se muestran en la Tabla 9.

Tabla 9. Modificación al código del programa que lee un sensor LM35 y envía el dato a una PC.

```
// Programa principal
   void loop() {
      valorSensor = analogRead(pinSensor);   // Lee el valor del sensor de temperatura
      float milivolts = 5000*(valorSensor / 1023.0);   // Convierte el valor a miliVolt
      temperatura = milivolts/10;   // Calcula el valor en grados centígrados
   //    Serial.print("La temperatura es de: ");     // Envía un letrero a la PC
      Serial.println(temperatura);                // Envía el dato de la temperatura
   //    Serial.println(" grados centigrados");     // Envía un letrero a la PC
      delay(1000);  // Retardo de 1000 milisegundos
   }
```

Las modificaciones consisten en eliminar los textos que se envían (añadiendo caracteres de comentario al principio), y modificando la instrucción que envía el valor de la temperatura, para que incluya el código de fin de línea (Serial.println).

A continuación ejecutamos el programa CoolTerm.exe, y damos clic al botón de conexión. En la ventana deberán mostrarse los datos recibidos. Para activar la opción de registro de datos usamos el menú *Connection-> Capture textfile -> Start* (ver Fig. 24). Se utiliza la opción Stop para finalizar el registro. También es posible añadir la hora a la que se adquirió el dato usando el menú *Connection-> Options -> Receive -> Add timestamps to received data*. Adicionalmente se puede seleccionar que sean marcas de tiempo relativo en la opción *Type*.

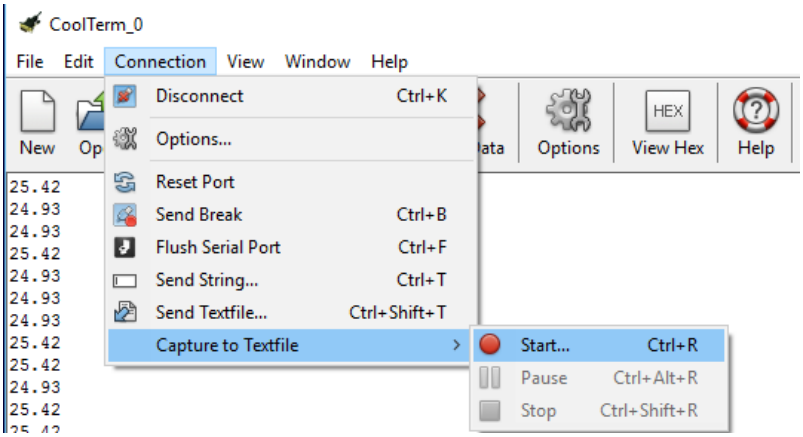


Fig. 24. Activación del registro de datos en el programa CoolTerm.

Los datos son guardados en formato texto (.txt) pudiendo abrirse y graficarse utilizando una hoja de cálculo como Excel (ver Fig. 25)

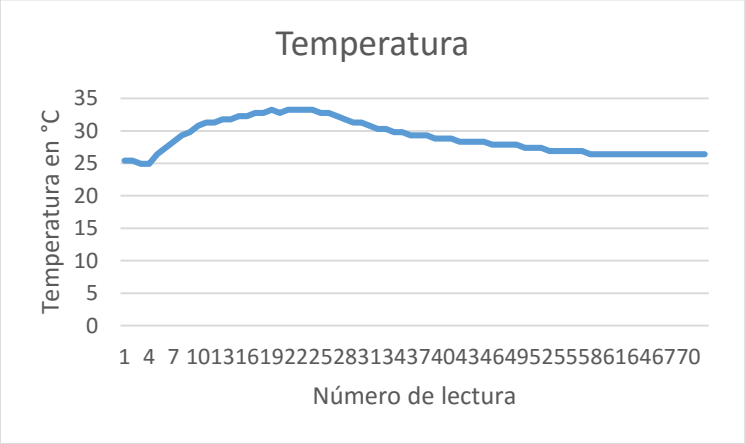


Fig. 25. Ejemplo de graficación de datos de temperatura adquiridos usando el programa CoolTerm.

5. MANEJO DE UN DISPLAY LCD

El manejo de un display de cristal líquido (LCD) se simplifica utilizando las bibliotecas de funciones del sistema Arduino. Como ejemplo, se propone construir un termómetro que muestre la temperatura en la pantalla del display. El sensor de temperatura que se utiliza es un circuito integrado LM35. Sólo cuenta con tres terminales, las cuales corresponden al voltaje de alimentación, la conexión a tierra, y la salida (ver Fig. 26). Este circuito integrado genera un voltaje en su terminal de salida que es proporcional a la temperatura, de acuerdo con la siguiente expresión

$$V_{salida} = T \cdot 10\ mV/°\text{C}$$

donde *T* es la temperatura en grados centígrados. Por ejemplo, si la temperatura ambiente es de 27 °C, entonces la salida queda dada por

$$V_{salida} = (27\ °\text{C})10\frac{mV}{°\text{C}} = 270\ mV$$

El LM35DZ puede medir temperaturas

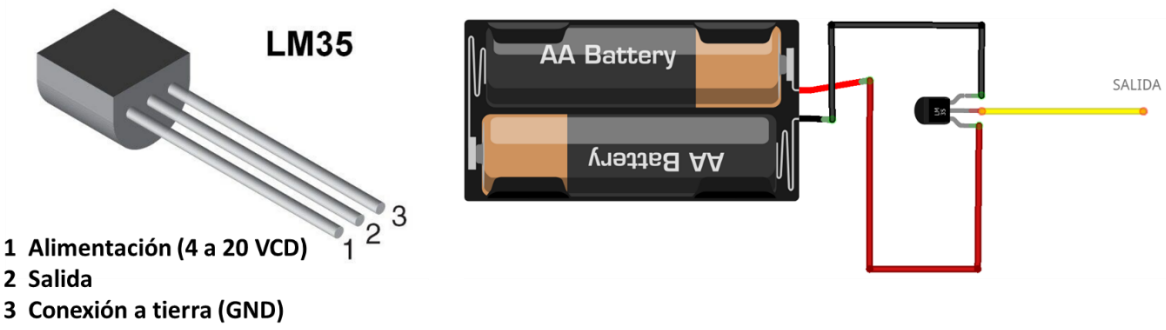


Fig. 26. Identificación de las terminales del sensor de temperatura LM35 (a la izquierda), y ejemplo de conexión (a la derecha).

Existen diferentes tipos de pantallas de cristal líquido (LCD, por sus siglas en inglés), como existen diversos fabricantes, es común que simplemente se especifiquen por el número de letras y renglones que es posible desplegar. En nuestro ejemplo utilizaremos una de 16x2, es decir, dos renglones de 16 letras (caracteres) cada uno. El diagrama de conexiones con una tarjeta Arduino UNO se muestra en la Fig. 27. Se utiliza una resistencia de 220 Ω (rojo, rojo, café) para encender la luz de retroiluminación. También se utilizan dos resistencias (8.2 kΩ y 1 kΩ), para regular el contraste de la pantalla. En la Tabla 10, se muestra el código de un programa para leer la temperatura y desplegarla en la pantalla.

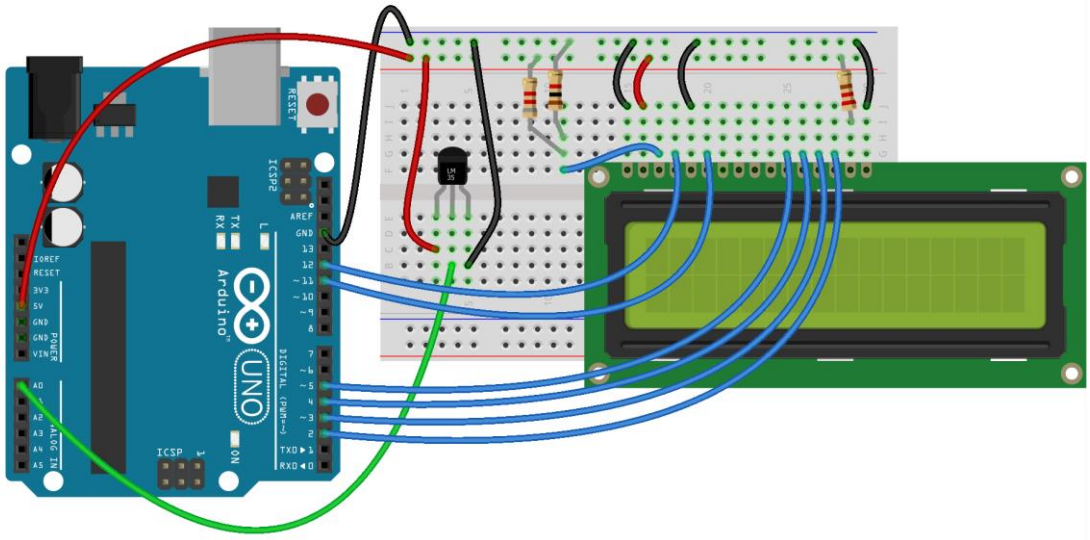


Fig. 27. Conexiones para el ejemplo del manejo de un LCD y un sensor de temperatura (modificado de los ejemplos del programa Fritzing

Para poder conectar el display en la *protoboard* es necesario soldarle una tira de pines como se muestra en la Fig. 28.

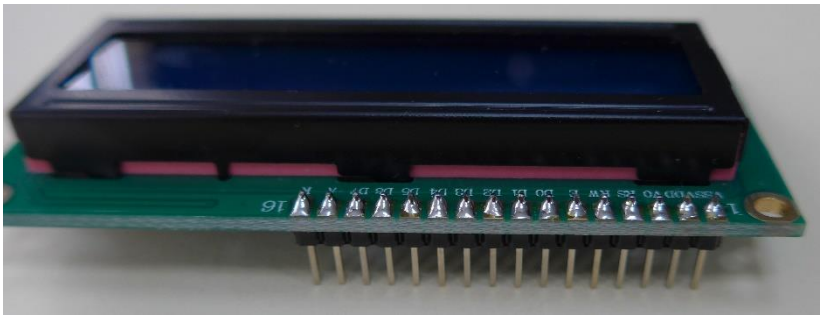


Fig. 28. Tira de pines soldada al display LCD.

A continuación, se comentan algunos detalles del programa. Para manejar el display se requiere utilizar una biblioteca y eso se logra con la instrucción

```
#include <LiquidCrystal.h>
```

El display cuenta con varios pines de conexión que sirven para suministrar alimentación al propio display, y dos pines especiales para alimentar la retroiluminación. Además, se usa un pin para

Tabla 10. Código del programa que usa un sensor de temperatura LM35 y muestra el resultado en un display.

```
// Se requiere la biblioteca de uso del display
            #include <LiquidCrystal.h>
// Se definen los pines que se comunican con el display
            LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
            const int pinSensor = 0;   // A0 es el pin del sensor analógico
  int valorSensor = 0;   // Guarda el valor del sensor
  float temperatura = 0;
// Inicialización
            void setup() {
              lcd.begin(16, 2);   // LCD es de 2 renglones y 16 caracteres
              // Manda un letrero al display
   pinMode(pinSensor, INPUT);
              lcd.print("La temp es:");
}
            void loop() {        // Programa principal
  // lcd.setCursor(col, row)
   lcd.setCursor(0, 1);   // Selecciona caracter 0, fila 1 (segunda)
   valorSensor = analogRead(pinSensor);
   temperatura = 500*(valorSensor/1023.0);
       lcd.print("      ");
       lcd.setCursor(0, 1);
            lcd.print(temperatura);
       delay(300);
            }
```

regular el contraste de la pantalla, cuatro para mandarle datos, y dos más para enviarle señales de control. La secuencia de valores de estas señales es controlada por la biblioteca *LiquidCrystal.h* y para poder enviar mensajes al display solo se requiere seleccionar la ubicación del texto, y enviarlo. La ubicación se hace con la instrucción

lcd.setCursor(*columna, fila*);

y el envío de texto con la instrucción

lcd.print(*cadena*);

La biblioteca incluye otros comandos que pueden consultarse en
https://www.arduino.cc/en/Reference/LiquidCrystalPrint

Un ejemplo del funcionamiento del programa se muestra en la Fig. 29. En este caso, se utilizó un display modelo 1602A, el cual tiene retroiluminación azul.

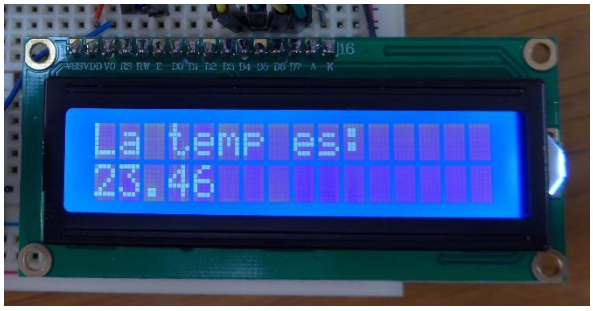


Fig. 29. El display LCD mostrando la temperatura ambiente.

6. BIBLIOTECAS PARA EL MANEJO DE SENSORES

Una de las ventajas de la popularidad del sistema Arduino es que muchos fabricantes de sensores han desarrollado bibliotecas compatibles con esta plataforma. Como ejemplo utilizaremos el sensor digital de temperatura DS18B20, el cual es un circuito integrado digital que proporciona un dato de 9 bits a 12 bits que representa la temperatura en grados centígrados. Tiene una exactitud de ±0.5 °C y un tiempo de conversión a 12 bits de 750 ms. Además, es posible conseguir una versión a prueba de agua, la cual se puede sumergir en líquidos cuya temperatura sea inferior a 100 °C. Este dispositivo se comunica mediante un protocolo llamado 1-wire, por ello, es necesario instalar dos bibliotecas: una para la comunicación 1-wire, y otra para manejar el sensor DS18B20.

La biblioteca 1-wire se puede descargar de: https://github.com/PaulStoffregen/OneWire dando clic en el botón que dice Download ZIP. Con esto se descarga el archivo *OneWire-master.zip*.

De la misma manera, se puede descargar la biblioteca del sensor DS18B20 de:

https://github.com/milesburton/Arduino-Temperature-Control-Library dando clic en el botón que dice Download ZIP. Con esto se descarga el archivo *Arduino-Temperature-Control-Library-master.zip.*

Las bibliotecas se instalan usando el menú *Programa->Incluir Librería->Añadir Librería .ZIP*, según se muestra en la Fig. 30.

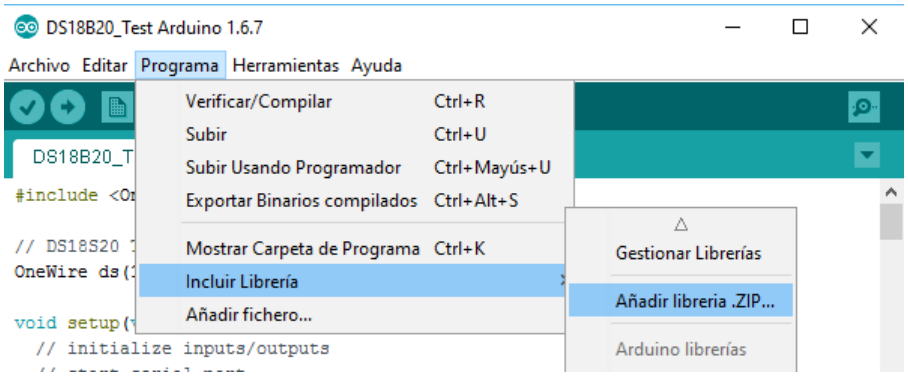


Fig. 30. Menú para instalar una nueva biblioteca en formato ZIP.

Información adicional sobre cómo instalar una biblioteca se puede encontrar en

http://arduino.cc/en/Guide/Libraries

El listado del programa se muestra en la Tabla 11. En las primeras líneas se encuentran las sentencias

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

las cuales se requieren para utilizar las funciones de las bibliotecas OneWire y del sensor de temperatura digital DS18B20, respectivamente. Otros detalles del programa se mencionan en los comentarios incrustados en el código.

El diagrama de conexiones del circuito que incluye el display LCD y el sensor de temperatura DS18B20 se muestra en la Fig. 31. La resistencia que se conecta entre Vcc (+5 V) y la terminal amarilla del sensor es de 4.7 kΩ.

Tabla 11. Programa que lee un sensor de temperature digital DS18B20 y envía el resultado a un display LCD.

```
/* Programa que lee un sensor de temperatura DS18B20 y
  *  despliega el resultado en un display LCD   */
// Se requiere la biblioteca de uso del display
            #include <LiquidCrystal.h>
// Se requiere la biblioteca de comunicación 1-wire, y del sensor
  #include <OneWire.h>
  #include <DallasTemperature.h>
// Se seleciona el pin 8 para lectura del sensor
#define ONE_WIRE_BUS 8

// Se active un controlador de protocolo OneWire.
OneWire oneWire(ONE_WIRE_BUS);

// Se asigna el controlador de protocol OneWire al sensor DS18B20 y se le asigna el nombre de
// sensors
DallasTemperature sensors(&oneWire);

// Se definen los pines que se comunican con el display
            LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
            const int pinSensor = 0;   // A0 es el pin del sensor analógico
  int valorSensor = 0;   // Guarda el valor del sensor
  float temperatura = 0;
// Inicialización
            void setup() {
 // Se inicializa el controlador que se denominó sensors
    sensors.begin();
             lcd.begin(16, 2);   // LCD es de 2 renglones y 16 caracteres
             // Manda un letrero al display
    pinMode(pinSensor, INPUT);
             lcd.print("La temp es:");
}
// Programa principal
            void loop() {
             // Selecciona el renglón 0, caracter 1
             // (Nota: el renglón 1 es el segundo
             lcd.setCursor(0, 1);
   sensors.requestTemperatures(); // Se envía el comando para leer las temperaturas
       lcd.print("      ");
       lcd.setCursor(0, 1);
             lcd.print(sensors.getTempCByIndex(0));  // Se despliega la temperatura del sensor 0
       delay(300);
            }
```

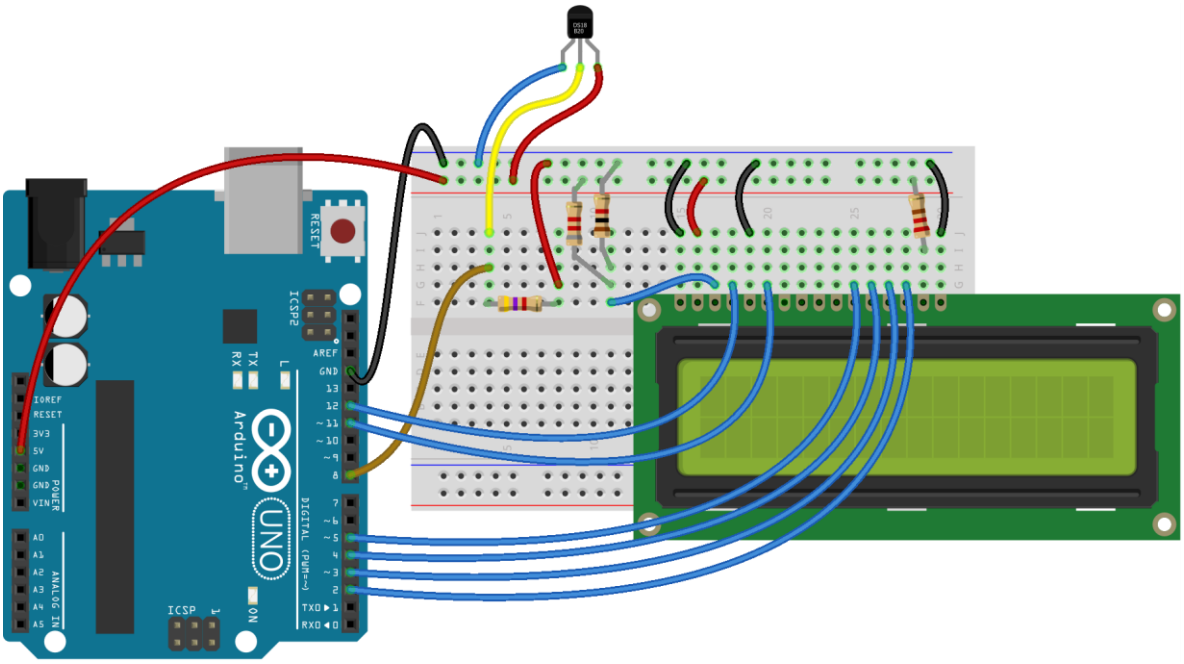


Fig. 31. Diagrama de conexiones del LCD y un sensor de temperatura DS18B20.

7. IMPLEMENTACIÓN DE UN CRONÓMETRO

Ahora utilizaremos la experiencia adquirida en el manejo de entradas y salidas digitales y lo combinaremos con el uso de un display LCD para implementar un cronómetro. Para ello utilizaremos una clase de sensores ópticos que se denominan optointerruptores. Empleando dos de ellos, es posible arrancar el cronómetro al detectar que un objeto atraviesa frente a un primer sensor y detener el cronómetro cuando el objeto atraviesa frente al segundo sensor.

Los opto-interruptores que utilizaremos son del tipo reflexivo (ver Fig. 32A). Consisten de un LED infrarrojo, y un fototransistor. El fototransistor detecta cuando la luz infrarroja que emite el LED es reflejada por un objeto (ver Fig. 32B). En la Fig. 32C se muestra el símbolo electrónico del opto-interruptor, el cual se compone del símbolo de un LED y del símbolo de un fototransistor agrupados.

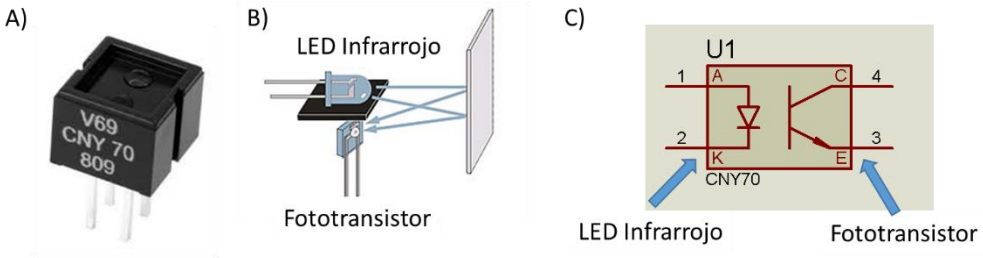


Fig. 32. A) Imagen de un opto-interruptor reflexivo. B) Funcionamiento de un opto-interruptor. C) Símbolo electrónico. (Fuente A y B: Internet).

El opto-interruptor requiere dos resistencias para su funcionamiento, según se muestra en la Fig. 33. El LED infrarrojo se encuentra siempre encendido. Cuando no hay algún objeto que refleje la luz infrarroja, el fototransistor se encontrará "apagado" y eso equivale a que se comporte como un circuito abierto (ver Fig. 33A). En esta condición el voltaje de salida es aproximadamente 5 V. Si se coloca frente al opto-interruptor un objeto, éste reflejará la luz infrarroja y el transistor cambiará a "encendido", lo cual equivale a que se comporte como un circuito cerrado (ver Fig. 33B). En este caso, la salida será aproximadamente 0 V.

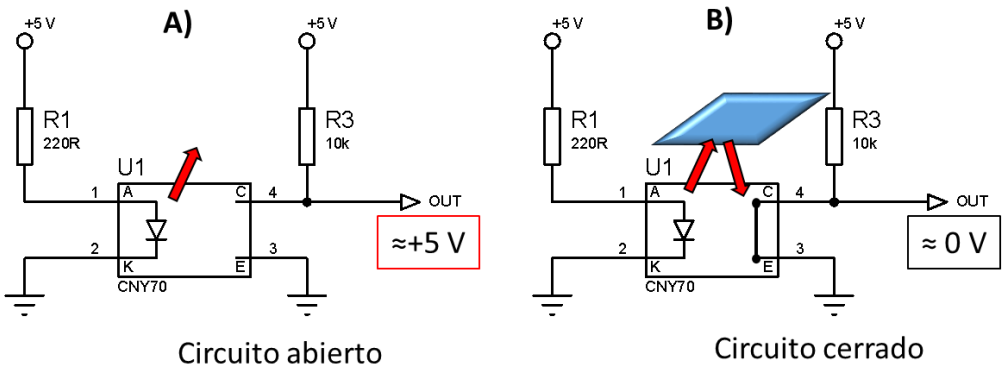


Fig. 33. A) Operación cuando no hay un objeto reflejante. B) Operación cuando sí hay un objeto reflejante.

En la Fig. 34, se muestra el diagrama de conexiones de los dos opto-interruptores, y de dos LEDs que servirán como comprobación (testigos) de que se han activado los sensores ópticos. Este circuito se deberá conectar a las terminales D2 a D5 de la tarjeta Arduino, según se indica. El código del programa cronómetro se muestra en la Tabla 12.

Tabla 12. Código del cronómetro optoelectrónico.

```
int const pinOpto1 = 2;   // Pin 2 recibe la señal del opto-interruptor 1
int const pinOpto2 = 4;   // Pin 4 recibe la señal del opto-interruptor 2
int const pinLed1 = 3;    // Pin 3 es la salida para el LED 1
int const pinLed2 = 5;    // Pin 5 es la salida para el LED 2
int valorOpto1, valorOpto2;   // Almacenan el estado del opto-interruptor (ON/OFF)
unsigned long tiempo, tiempo1, tiempo2;   //  tiempo1 es el tiempo inicial, tiempo2 es el tiempo final

void setup() {
  pinMode(pinOpto1, INPUT);
  pinMode(pinOpto2, INPUT);
  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  valorOpto1 = 0;
  while(!valorOpto1){   // Espera a que se active opto1
    valorOpto1 = digitalRead(pinOpto1);   // Lee el primer opto
  }
  while(valorOpto1){
    valorOpto1 = digitalRead(pinOpto1); // Espera a que se desactive opto1
  }
    tiempo1 = millis();    // Guarda el inicial tiempo en milisegundos
    digitalWrite(pinLed1, HIGH);    // Enciende el LED de arranque
    valorOpto2 = 0;
    while(!valorOpto2) {
    valorOpto2 = digitalRead(pinOpto2);  // Espera a que se active opto2
  }
  while(valorOpto2){
    valorOpto2 = digitalRead(pinOpto2);  // Espera a que se desactive opto2
  }
    tiempo2 = millis();     // Guarda el tiempo final en milisegundos
    digitalWrite(pinLed2, HIGH);   // Enciende el LED de paro

  if(tiempo2 > tiempo1)  {
    tiempo = tiempo2 - tiempo1;
    Serial.print("Tiempo: ");
    Serial.println(tiempo);
    delay(1000);   // Espera un segundo
    digitalWrite(pinLed1, LOW);    // Apaga el led de arranque
    digitalWrite(pinLed2, LOW);    // Apaga el led de paro
  }
}
```

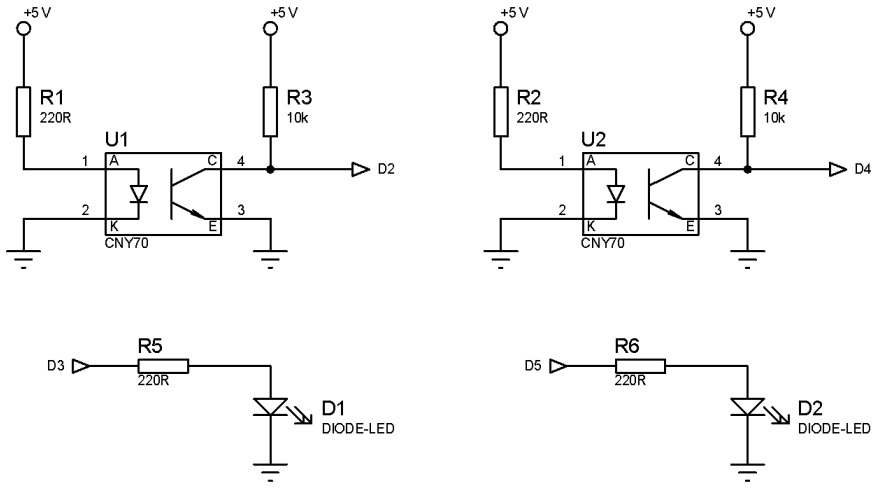


Fig. 34. Diagrama electrónico de los sensores y LEDs para realizar un cronómetro. Se indican las conexiones D2 a D5 hacia la tarjeta Arduino.

El programa espera a que se active el opto-interruptor 1 y cuando se desactiva, lo cual indica que el objeto ha dejado de estar enfrente, se lee el tiempo inicial mediante la instrucción

tiempo1=mills();

La función *mills()* proporciona un número en milisegundos que representa el tiempo que lleva la tarjeta Arduino ejecutando el programa actual. La misma función se utiliza para detectar el paso de un objeto sobre el opto-interruptor 2 y se almacena en la variable tiempo2. La diferencia entre tiempo1 y tiempo2, es el tiempo cronometrado y se envía a la PC por el puerto serial. El resultado se puede examinar utilizando el Monitor Serie del ambiente Arduino.

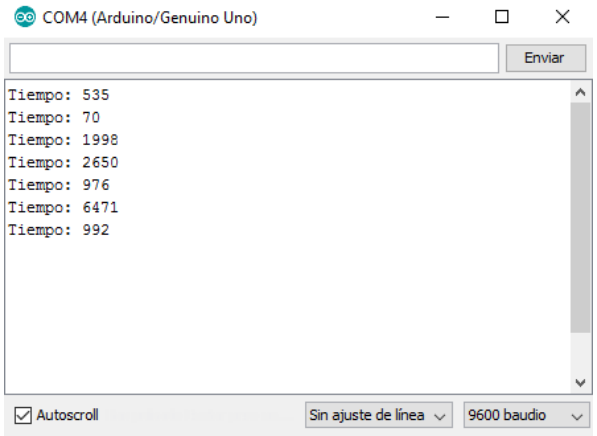


Fig. 35. Ventana del Monitor Serie recibiendo datos del cronómetro.

8. REFERENCIAS

Torrente A., Oscar. Arduino curso práctico de formación. Ed. Alfaomega. 2013.

McRobets, Michael. Beginning Arduino. Apress. 2010.

Arduino programming language reference. https://www.arduino.cc/en/Reference/HomePage

APÉNDICE A. INSTALACIÓN DEL AMBIENTE DE PROGRAMACIÓN DEL ARDUINO

En primer lugar debemos entrar a la página web www.arduino.cc, y seleccionar la opción *download* (ver Fig. 36). Esto nos llevará a la página de descargas que se muestra en la Fig. 37.

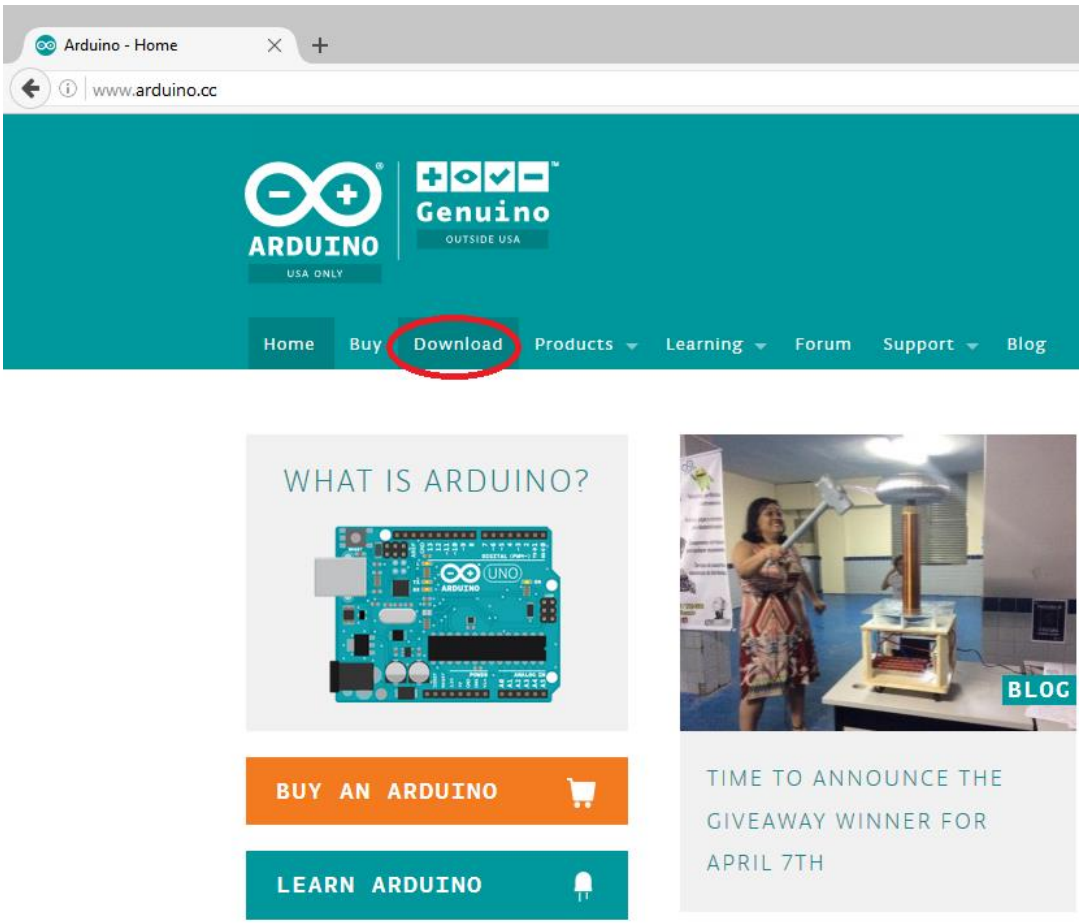


Fig. 36. Página principal del sitio oficial de Arduino.

Si nuestra computadora tiene instalado el sistema operativo Windows, podemos entonces dar clic en la opción *Windows Installer*. Eso nos llevará a una página donde se nos pregunta si deseamos contribuir económicamente con los desarrolladores, si no se desea contribuir en ese momento, basta con seleccionar la opción *Just Download* (ver Fig. 38).

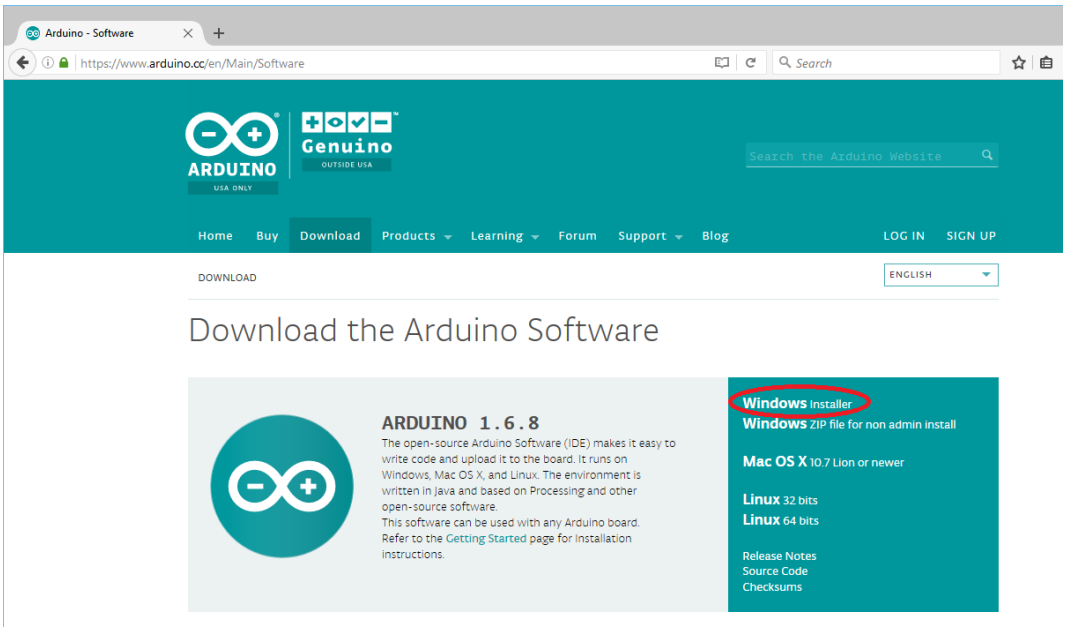


Fig. 37. Página de descargas del ambiente de programación del Arduino.

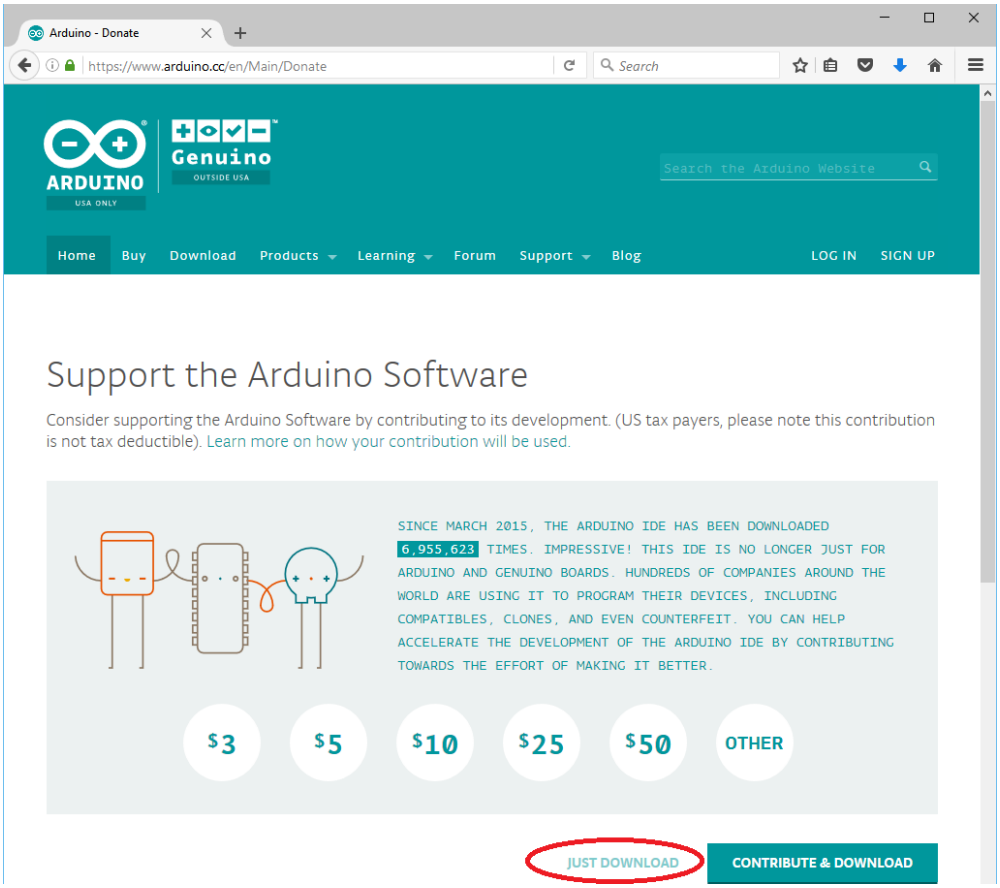


Fig. 38. Ventana de solicitud de contribución económica.

APÉNDICE B. PROGRAMACIÓN DE FUNCIONES

Para explicar el concepto de una función en programación, supongamos que queremos calcular la raíz cuadrada de un número, y para ello se crea una función llamada *squareroot().* Dentro de los paréntesis debemos indicar el número del cual queremos obtener la raíz cuadrada. El resultado del cálculo es otro número, por lo que la función podría definirse como

int squareroot(int)

De esta manera se declara una función a la cual se le pasa como parámetro un número entero (*int*) y entrega como resultado otro número entero.

Un ejemplo de utilización sería:

```
int a;

a=squareroot(16);
```

donde la variable entera **a** recibe el dato que entrega la función *squareroot().*

La definición completa de la función podría ser:

```
int squareroot(int x) {

int result;

result=x^0.5;

return result;

}
```

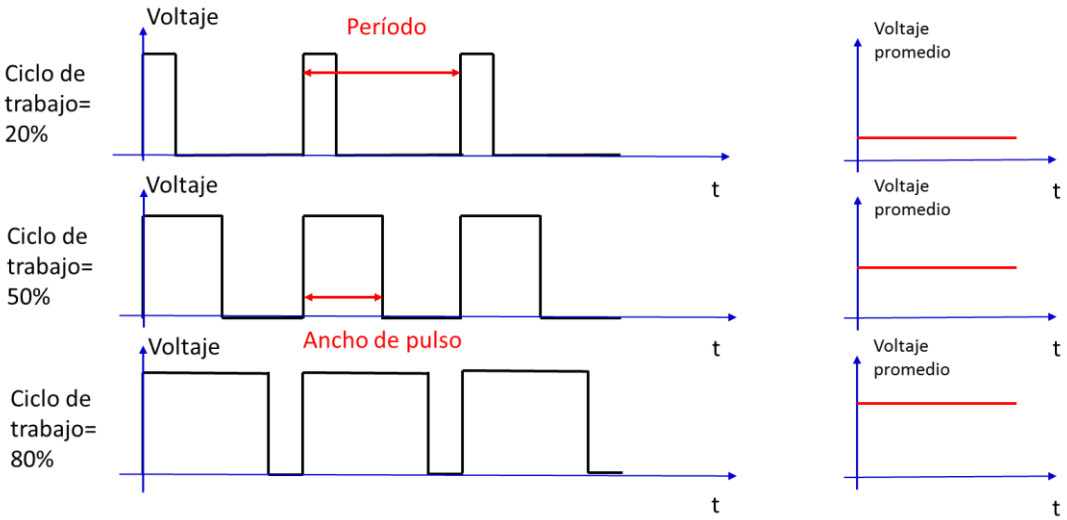
Las funciones se deben declarar afuera de los bloques *setup* y *loop.*

Se puede encontrar más información en:

https://www.arduino.cc/en/Reference/FunctionDeclaration

APÉNDICE C. PWM (PULSE WIDTH MODULATION)

La modulación por ancho de pulso o PWM por sus siglas en inglés, consiste en la generación de una señal cuadrada de voltaje con una frecuencia constante, pero cuya relación entre el tiempo que la señal está en alto con respecto a bajo cambia. En la Fig. 39 se muestran tres ejemplos de una señal PWM. El período, es decir el tiempo que tarda la señal en repetir su patrón, es constante. Sin embargo, el ancho de pulso, es decir el tiempo que la señal permanece en un valor alto, cambia. La relación entre el tiempo que el pulso está en alto, y la duración del período se denomina ciclo de trabajo. Si esta señal PWM es aplicada a un sistema lento (es decir, que no puede cambiar de encendido a apagado a la misma velocidad de la señal PWM), entonces el sistema interpretará la señal como un voltaje promedio, tal como se muestra en las gráficas del lado derecho de la Fig. 39. De esta manera se está produciendo el efecto de una salida analógica continua.



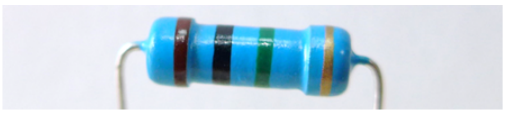
$$Ciclo\ de\ trabajo = \frac{ancho\ de\ pulso}{período}$$

Fig. 39. Ejemplos de una señal de modulación por ancho de pulso.

Información adicional se puede consultar en

https://www.arduino.cc/en/Tutorial/PWM

APÉNDICE D. TABLA DE COLORES DE RESISTENCIAS



Color	1ra. banda	2da. banda	3ra. banda	Tolerancia
Negro	0	0	x 10^0	
Café	1	1	x 10^1	1 %
Rojo	2	2	x 10^2	2 %
Naranja	3	3	x 10^3	
Amarillo	4	4	x 10^4	
Verde	5	5	x 10^5	
Azul	6	6	x 10^6	
Violeta	7	7	x 10^7	
Gris	8	8	x 10^8	
Blanco	9	9	x 10^9	
				Dorado 5%
				Plata 10%

APÉNDICE E. CÓMO DESCOMPRIMIR UN ARCHIVO .ZIP

Para descomprimir un archivo .zip se pueden utilizar las funciones propias del sistema operativo o bien tener instalado un programa que maneje archivos comprimidos. Para esta última opción se sugiere instalar el programa WinRAR, descargándolo de la siguiente liga:

http://www.rarlab.com/download.htm

Se debe escoger la versión de acuerdo al idioma y al tipo de sistema operativo (32 bits o 64 bits). En caso de duda se puede instalar la versión de 32 bits. Como ejemplo, hemos descargado la versión 5.3 en español de 64 bits:

winrar-x64-530es.exe

Se instala con las opciones que marca el programa por omisión. El programa activa unas opciones que aparecerán en el menú contextual del sistema operativo (es decir, las que aparecen al hacer clic con el botón derecho del ratón).

Ahora es posible seleccionar el archivo que queremos descomprimir, por ejemplo *CoolTerm_Win.zip*, y usando el botón derecho del ratón aparece un menú donde se selecciona la opción extraer en CoolTerm_Win\ como se muestra en la Fig. 40. Esto crea una carpeta donde quedará descomprimido el contenido del archivo .zip.

Fig. 40. Menú contextual (botón derecho del ratón) para extraer el contenido de un archivo .ZIP en una carpeta.

Si se desea, se puede mover la carpeta a otra ubicación, o crear un acceso directo del programa CoolTerm.exe (mediante el menú contextual, botón derecho del ratón) y moverlo al escritorio. En otro caso, basta con dar doble clic en *CoolTerm.exe,* para abrir el programa.