6 Sistemas domóticos basados en electrónica. Arduino

Contenidos

Los sistemas domóticos basados en electrónica y, en especial, con las placas Arduino permiten establecer una domótica de bajo coste. Esta cualidad, junto con el hecho de la facilidad de su programación y la gran documentación y ejemplos que hay en internet, hace que esta opción tenga cada vez una mayor aceptación. Con las placas Arduino, se puede establecer un sistema domótico completo en cuanto al *hardware* y, además, permiten poder desarrollar proyectos personalizados para sensores y actuadores que, posteriormente, se pueden utilizar en otros sistemas domóticos.

Objetivos

- Distinguir las diferentes placas de modelos de Arduino.
- Saber configurar la programación de las placas.
- Realizar pequeños programas de uso de diferentes tipos de sensores y actuadores.
- Establecer comunicación mediante el monitor serie.
- Entender la necesidad de instalar un servidor domótico.

6.1. Los sistemas domóticos basados en electrónica

Los sistemas basados en electrónica supusieron el punto de partida de la domótica. Comenzaron como sistemas cerrados en los cuales solo las empresas que los desarrollaban conocían el funcionamiento interno. No estaban estandarizados y ello conllevaba a que, cuando el usuario de la instalación necesitaba realizar algún cambio o ampliación, debía recurrir a estas mismas empresas.

Esta dependencia beneficiaba a estas empresas desarrolladoras, puesto que se aseguraban de que el cliente no podía irse con la competencia. Por el contrario, el usuario final dependía de ellos y, en muchas ocasiones, la empresa cerraba y desaparecía, perdiendo el usuario un soporte técnico en el cual, ante un fallo de un dispositivo, no tenía acceso a repuestos ni a posibilidades de ampliación de la instalación. A pesar de todos estos inconvenientes, fueron muy utilizados entre los usuarios que querían tener en sus viviendas un sistema domótico.

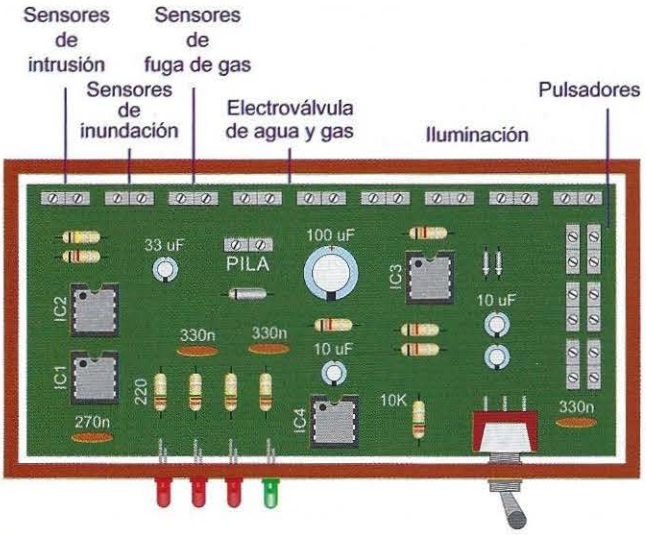


Figura 6.1. Sistema electrónico de domótica.

6.2. El sistema de desarrollo basado en Arduino

Arduino es una plataforma de desarrollo de proyectos electrónicos de código abierto. Esto quiere decir que hay una comunidad de desarrolladores que contribuyen a mejorar y ampliar constantemente este sistema.

Nació como un instrumento de ayuda al aprendizaje de la electrónica y la programación y, desde entonces, va teniendo cada vez una mayor aceptación por su versatilidad, facilidad de uso y una gran información en internet. Por lo tanto, el sistema de Arduino se compone de dos elementos: por un lado, la placa electrónica de desarrollo y, por otro lado, un entorno (IDE) de programación basado en el lenguaje C.

Actualmente, existen viviendas que se han domotizado empleando el sistema Arduino, aunque a un nivel particular y desarrollado por los propios usuarios que tienen los conocimientos necesarios. A pesar de ello, es posible ampliar los sistemas estudiados añadiéndole una placa Arduino para resolver una necesidad.

6.3. Tipos de placas Arduino

Arduino es un sistema que está basado en el microcontrolador ATMEL de *hardware* abierto, es decir, cualquiera puede utilizar los esquemas electrónicos y desarrollar su propio *hardware*. Entre algunos de estos diseños que se han popularizado y los modelos oficiales, hay más de dos docenas de placas:

- **Arduino Uno.** Es la más popular y con la cual se suele empezar. Por ello aparece en muchos paquetes de iniciación. Tiene 15 pines digitales y 5 analógicos. El procesador es el ATMEGA328P de 32 kB de memoria.
- **Arduino Leonardo.** Es similar a Arduino Uno. Tiene el procesador ATMEGA32U4 y dispone de 20 pines de entradas y salidas digitales. Cuenta con una entrada microUSB. Esta placa emula a un ratón, *joystick* o teclado.
- **Arduino 101.** Incorpora un microprocesador Intel Curie, con lo que se obtiene una placa de alto rendimiento con un bajo consumo. Incorpora, además, comunicación *bluetooth,* acelerómetro y giroscopio, aspectos que son apreciados en un proyecto IoT (internet de las cosas).
- **Arduino Explora.** Evoluciona desde la placa Leonardo, integrando un micrófono, avisador acústico, sensor de temperatura, acelerómetro, led RGB, cuatro pulsadores y un *joystick* analógico.
- **Arduino Zero.** Es una evolución del Arduino Uno con más potencia y con un microprocesador SAMD21, que permite desarrollar aplicaciones de 32 bits. A diferencia de la mayoría de las placas Arduino que funcionan a 5 V, esta placa trabaja con una tensión de 3,3 V.
- **Arduino Mega.** Es la placa que se utiliza cuando el Arduino Uno se queda corto en prestaciones. Incorpora el microprocesador ATMEGA256, que es más potente. Dispone de 70 pines en total para funciones digitales y analógicas y 256 kB de memoria *flash*.
- **Arduino Mini pro.** Placa basada en el ATMEGA328P de reducidas dimensiones. Tiene un total de 14 pines de entradas y salidas. Hay dos modelos de 3,3 V y 5 V.

Arduino se relaciona con el exterior mediante una serie de pines denominados *puertos*. Hay dos tipos de puertos:

1. **Entradas.** Por estos pines, Arduino recibe o lee las señales provenientes de pulsadores y sensores.

2. **Salidas.** Por estos pines o puertos, el microprocesador escribe o envía señales para activar el receptor conectado en ellos.

Figura 6.2. Placas Arduino más significativas.

Recuerda

No todas las placas Arduino funcionan a 5 V, las hay de 3,3 V.

El número de puertos de entradas o salidas varía en función del modelo de Arduino empleado. Por ejemplo, el modelo de Arduino Uno dispone de 14 pines numerados del 0 al 13. Muchos de los puertos pueden ser de entrada o de salida y, antes de utilizarlos, se deben configurar para indicar cómo se deben comportar. Algunos de los pines poseen varias funciones, las cuales se deben indicar antes de usarlos.

NOTA

En la exposición de esta unidad, se utilizará el Arduino Mega, aunque se detallarán las modificaciones para ser utilizado en un Arduino Uno. La versión Mega tiene más prestaciones y, económicamente, es muy poca la diferencia.

6.4. El lenguaje de programación

Un programa informático es un conjunto de instrucciones que le indican al microprocesador cómo debe actuar con los siguientes criterios:

- Ser preciso.
- Ser inequívoco y, por tanto, solo puede ser interpretado de una única manera y sin posibilidad de duda.
- Conciso, con instrucciones cortas.

De todos los lenguajes de programación existentes, Arduino se programa con una variante del lenguaje C++, el cual es muy conocido y utilizado. Es rígido en cuanto a la programación y, por ello, si algo no está bien en cuanto a la sintaxis, el programa lo indicará provocando un error y obligando a su corrección.

Este lenguaje C++ empleado, el microprocesador no lo entiende tal cual y, para que lo comprenda, se debe traducir. Esta traducción de un lenguaje entendible por los programadores (código fuente) a instrucciones entendibles por el microprocesador (código maquina) se denomina *compilación.*

6.5. El entorno de programación

El entorno de programación, también llamado *integrated development enviroment* (IDE), es la aplicación informática sobre la cual se realiza el programa o código fuente. Este entorno proporciona un conjunto de herramientas que ayudan en esta tarea como, por ejemplo, el compilador, el generador de información de errores, la administración de bibliotecas, el monitor para el puerto serie, etcétera.

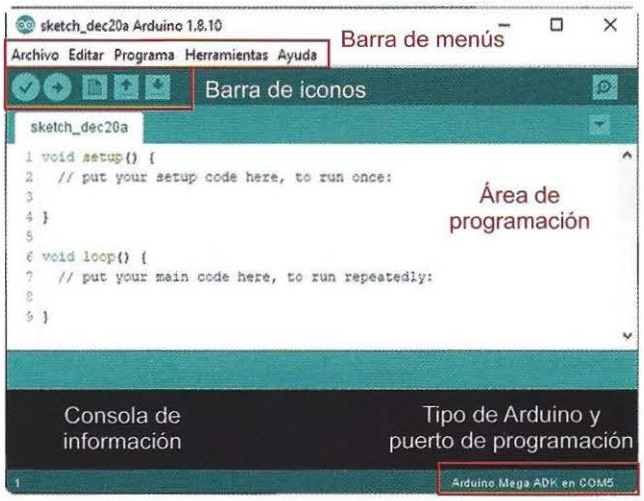


Figura 6.3. Partes más significativas del entorno de programación.

El IDE de Arduino se puede descargar desde su web www.arduino.cc. Una vez instalado, al ejecutarse, muestra el IDE, el cual consta de las siguientes partes:

- **Barra de menús.** Contiene las diversas opciones que puede realizar el IDE.
- **Barra de iconos.** Con los accesos a las funciones de mayor uso, como son la parte de compilación y transferencia del programa al Arduino, entre otras.
- **Área de programación.** Es el espacio destinado a la realización de la programación.
- **Consola de información.** En este espacio, se muestra información sobre el proceso de compilación y

transferencia del programa. En caso de algún error, se muestra en diferente color para resaltarlo.

- **Tipo de Arduino y puerto de programación.** Informa sobre qué tipo de placa se está realizando la programación y, además, sobre el número del puerto COM del ordenador sobre el cual se conecta la placa para realizar la programación.

6.6. La programación de la placa Arduino

La programación de una placa de Arduino comienza con la configuración del *hardware* –cuando se le indica al programa el tipo de placa que va a emplear–, a continuación, se realiza la programación del código para que responda a las necesidades planteadas y, por último, se compila y se transfiere a la placa de Arduino mediante un cable USB.

6.6.1. Configuración de *hardware*

El entorno IDE se puede utilizar para programar diversos tipos de placas, por ello es necesario indicar sobre cuál se va a trabajar. Esta operación se realiza desde el menú de herramientas, donde se selecciona la placa, que, en este caso, es una Arduino Mega (Figura 6.4).

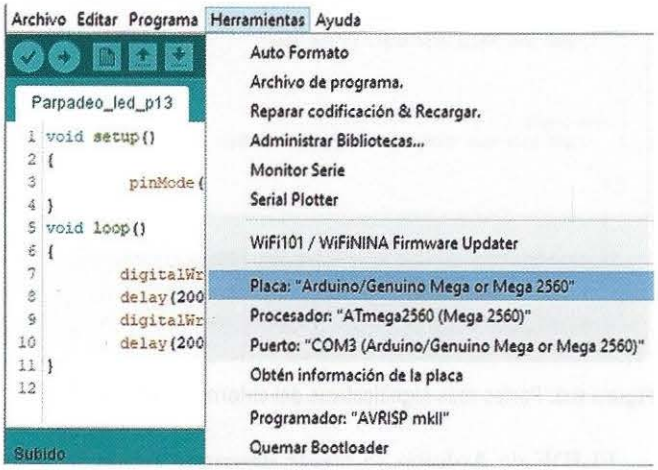


Figura 6.4. Configuración de la placa.

Otro dato que se necesita es indicar sobre qué puerto USB, que se indica como *COM,* se realiza la comunicación. La manera más sencilla de averiguar este dato es conectar y desconectar el Arduino y ver qué puerto aparece o desaparece, y este será el que utiliza.

6.6.2. La estructura de un programa en Arduino

Un programa o *sketch* de Arduino consiste en dos secciones o funciones básicas:

1. ***Setup.*** En esta función, se indican las instrucciones que se ejecutan solo una vez cuando se inicia el programa (al arrancar Arduino o realizar un reset). Aquí se realizan las inicializaciones.
2. ***Loop.*** Esta sección se repite indefinidamente. Se comienza ejecutando las instrucciones en orden y, al terminar la última, comienza automáticamente de nuevo por la primera instrucción de esta sección.

Estas dos secciones se crean automáticamente cuando se inicia un nuevo programa en el IDE.

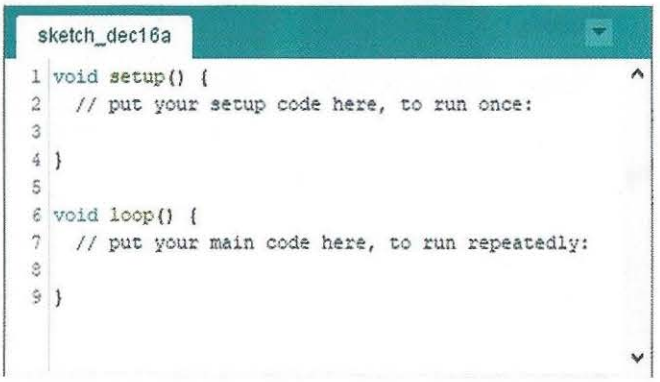
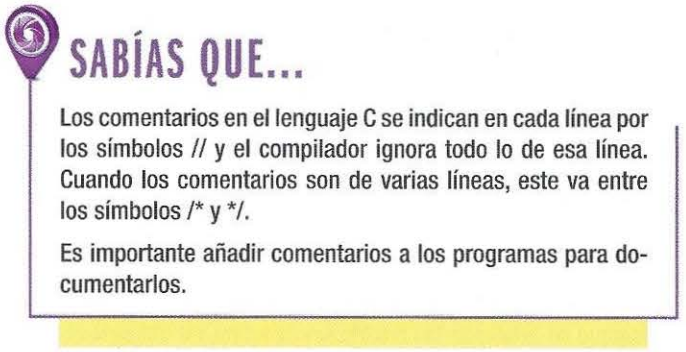


Figura 6.5. IDE de Arduino con las funciones *setup* y *loop.*

En el lenguaje C, el contenido de las funciones comienzan por una llave { y terminan por otra llave }.



Al guardar o salvar un programa de Arduino, este tiene las extensión *.ino* y se guarda en una carpeta con el mismo nombre que el programa.

6.6.3. La compilación y transferencia

Una vez realizado el programa, el siguiente paso es verificar que el código no contiene errores. En el IDE de Arduino, se puede compilar desde el menú programa y la opción verificar/compilar o desde un icono directo.

Si no hay ningún error en la parte inferior, informa de ello, quedando listo el programa para ser transferido a la placa de Arduino.

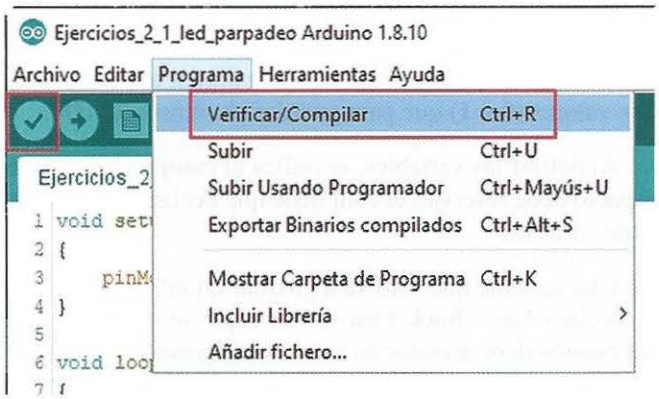


Figura 6.6. Compilación.

La transferencia se realiza desde el menú de programa y la opción subir o desde un icono directo.

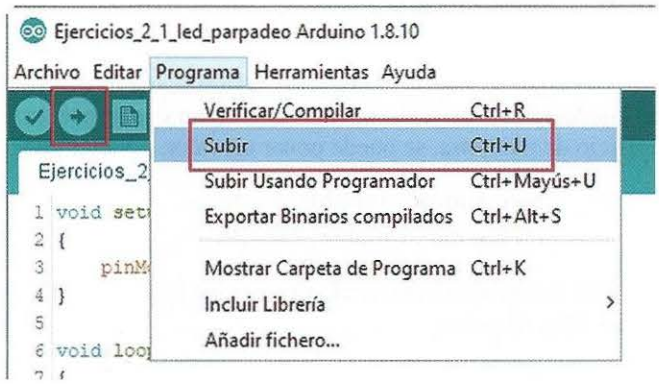


Figura 6.7. Transferir o subir.

6.7. Las entradas y salidas digitales

Las entradas y salidas digitales trabajan con valores binarios y se limitan a leer un nivel de tensión –correspondiente a un 1 o un 0 en las entradas– y a poner esos mismos valores en sus salidas.

6.7.1. Las salidas digitales

Las salidas digitales permiten poner un valor binario de 0 o 1 en un puerto o pin de salida. Muchos de los pines o puertos de Arduino pueden configurarse para trabajar de diversas maneras, por ello lo primero es configurar el puerto. La configuración del puerto se realiza mediante la instrucción **pinMode.** Esta instrucción necesita dos valores, que son el número de puerto y cómo debe trabajar. Si se desea utilizar un puerto configurable como salida, se le indica mediante el valor OUTPUT. Por ejemplo, para poner el puerto o pin 13 como salida, se realiza mediante la siguiente orden:

```
pinMode (13, OUTPUT);
```

Es importante respetar las mayúsculas y las minúsculas, así como el punto y coma (;) final, ya que, en caso contrario, dará un error en su compilación.

Esta configuración solo se realiza una única vez para el programa y, por ello, se debe poner en la función *setup*.

Para escribir un valor en ese pin de salida del Arduino, se utiliza la instrucción **digitalWrite,** a la cual se le indica el número de puerto y el valor que ha de poner. Por ejemplo, para poner en el puerto 13 un valor binario de 1, se indicaría mediante la siguiente orden:

```
digitalWrite (13 , 1);
```

El compilador de Arduino entiende que, si se pone el parámetro HIGH, lo toma como un 1 digital y, si se pone el parámetro LOW, como un 0 digital.

```
digitalWrite (13 , HIGH);
```

ACTIVIDAD RESUELTA 6.1

Realiza para tu Arduino un programa que genere un parpadeo en el led insertado que lleva en la placa. El parpadeo debe ser de 0,2 segundos encendido y 0,2 segundos apagado.

Solución:

Respecto al *hardware,* se va a encender y apagar el led que viene integrado en las placas. Este pin varía según el modelo de Arduino, en el caso del Uno y del Mega, es el pin 13.

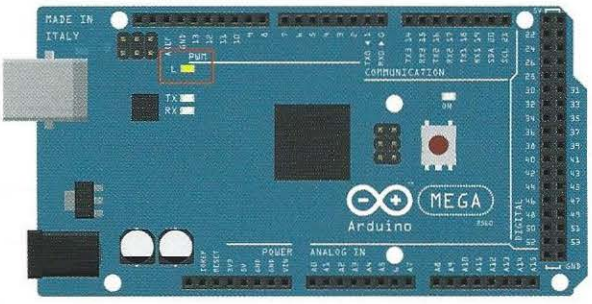


Figura 6.8. Arduino Mega con el pin del led integrado encendido (pin 13).

Para realizar el parpadeo, las instrucciones que se van a dar son las siguientes:

- Enciende el led.
- Haz una pausa.
- Apaga el led.
- Haz una pausa.

Estas órdenes las repetirá indefinidamente, por ello se debe poner en la función *loop*. Pero antes se debe configurar el puerto en el cual está el led. La orden pausa se realiza mediante la instrucción *delay*, a la cual se le indica el tiempo en milisegundos.

```
void setup()
{
    pinMode( 13 , OUTPUT); // Pin 13
}
void loop()
{
    digitalWrite(13 , HIGH); // Enciende el LED
    delay(200); // Esperar un 0,2 segundos
    digitalWrite(13 , LOW); // Apagar el LED
    delay(200); // Esperar otros 0,2 segundos
}
```

Código 6.1. Parpadeo del led del Arduino.

6.7.2. Las entradas digitales

Las entradas digitales o binarias pueden tener dos valores: 1/0, sí/no o *true/false*.

Cuando se tiene un pulsador, se hace que el valor de tensión conectado a ese pin digital de entrada oscile entre un nivel de tensión de 5 V o de 0 V correspondientes a los valores digitales o binarios de 1 y 0.

Para forzar esos valores y no dejar el pin al aire, se emplea una resistencia, que, según dónde se coloque, se denomina *resistencia pullup* o *pulldown*. Con una resistencia *pullup*, se consigue un valor alto (1) en vacío o reposo y, con una resistencia *pulldown*, se consigue un nivel bajo (0) en vacío o reposo.

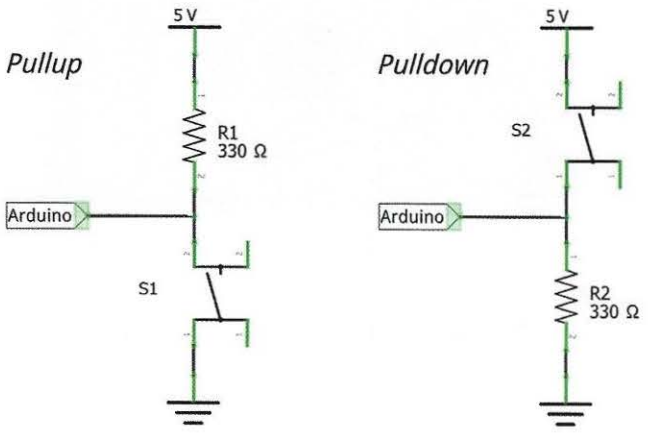


Figura 6.9. Resistencia *pullup* y *pulldown*.

Cuando se lee un valor de una de las entradas, este se guarda en algún lugar de la memoria; este espacio reservado se denomina *variable*. El tamaño reservado para cada variable depende del valor de esta, por ejemplo, no se necesita el mismo espacio para guardar un valor digital que solo toma dos valores (0 o 1) que para guardar el número 20 000.

Al definir las variables, se indica al compilador cuánto espacio debe reservar, el cual tiene que declararse antes de poder usarlas.

Una variable que solo va a guardar un bit (un 1 o un 0) se declara como **bool.** Una variable que se declara como **int** cuando debe guardar un valor correspondiente a un número entero entre –32 768 y 32 767. La variable se declara colocando la palabra clave y, a continuación, el nombre de esa variable, terminando con un punto y coma (;), por ejemplo:

```
bool nombre_variable_1;
int nombre_variable_2;
```

Los nombres de las variables no se pueden repetir, puesto que es el que se utilizará para poder identificarlas. Además, se puede inicializar, es decir, cuando se crean y se reserva ese espacio de memoria, se puede poner un valor, por ejemplo:

```
bool nombre_variable_1 = false;
int nombre_variable_2 = 123;
```

Con las palabras clave, el compilador las colorea para poder identificarlas.

No se puede utilizar como nombre de variable una palabra reservada o clave.

Es frecuente utilizar las variables para definir los pines que se utilizan, de esta manera, en caso de necesitar cambiar el pin, solo se debe corregir esta variable en vez de buscar por todo el código (Código 6.2).

La lectura de un valor digital se realiza mediante la orden **digitalRead** y, entre paréntesis, el número de pin que lee. Esta lectura se debe guardar en una variable, por ejemplo:

```
int valor;
valor = digitalRead(6) ;
```

En este caso, se lee el pin número 6 y se guarda en la variable entera denominada *valor*.

SABÍAS QUE...

Es posible declarar y usar una variable en una misma línea de código, por ejemplo:

```
int valor = digitalRead(6) ;
```

ACTIVIDAD RESUELTA 6.2

Desarrolla para tu Arduino un programa que encienda un led situado en el puerto 10 cuando se accione un pulsador situado en el pin 6.

Solución:

Se realiza el esquema electrónico teniendo en cuenta que, para la entrada, se coloca una resistencia *pulldown* de 220 Ω en el puerto D6 (D de digital). La salida será el puerto D10, que se conectará un led con una resistencia de 330 Ω.

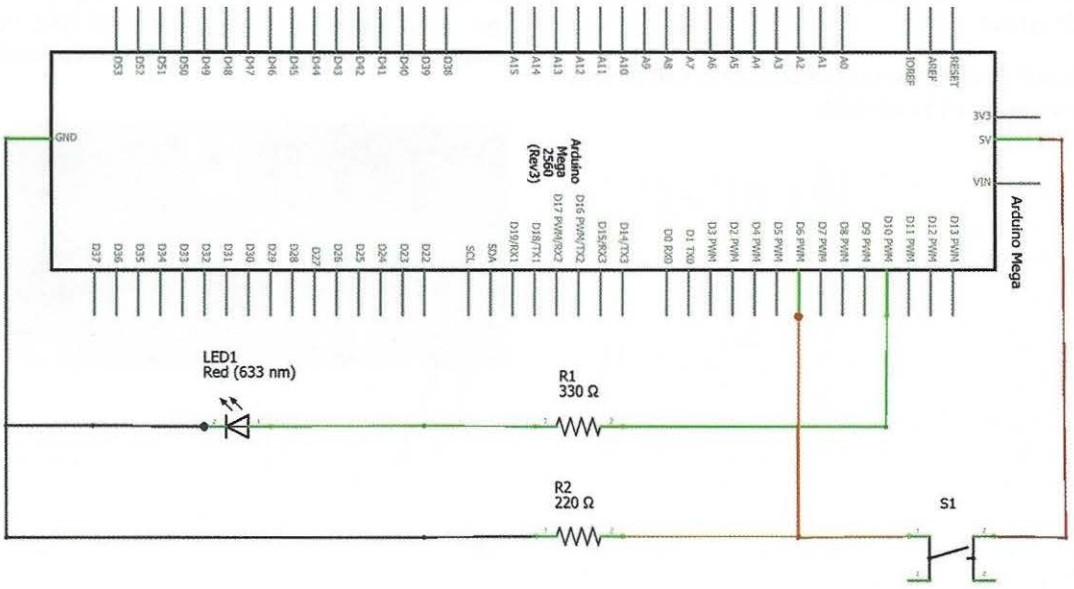


Figura 6.10. Esquema electrónico.

El montaje quedaría para un Arduino Mega de la siguiente manera:

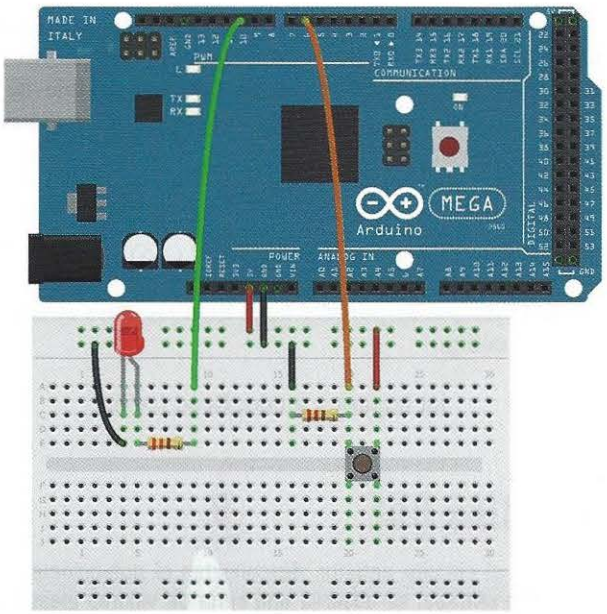


Figura 6.11. Montaje electrónico.

```
int LED = 10;
int boton = 6;
void setup()
{
    pinMode( LED, OUTPUT) ;
    pinMode( boton , INPUT) ;
}
void loop()
{
    int valor = digitalRead(boton) ;
    digitalWrite( LED, valor) ;
}
```

Código 6.2. Control de un led según el accionamiento de un pulsador.

En el Código 6.2, Arduino entiende INPUT como entrada y OUTPUT como salida.

6.7.3. El monitor serie

El Arduino puede comunicarse por el puerto serie, en este caso, entre él y el ordenador, y para ello dispone de una librería llamada *Serial*.

Esta librería se inicializa indicándole a qué velocidad se realiza la comunicación que puede estar comprendida entre los 300 y los 115 200 bits por segundo, aunque lo normal es emplear la velocidad de 9600 bits por segundo. Para enviar un dato, se emplea la orden **println,** junto al dato que ha de enviar.

En el Código 6.3, se envía constantemente el valor 1234, que se ha almacenado en la variable *i*.

```
void setup()
{
    Serial.begin (9600);
}
void loop()
{
    int i = 1234 ;
    Serial.println ( i );
}
```

Código 6.3. El monitor serie.

Para leer los datos que envía Arduino por comunicación, se debe abrir el monitor serie que contiene el IDE de Arduino y ajustarlo a esa velocidad

Figura 6.12. Apertura del monitor serie.

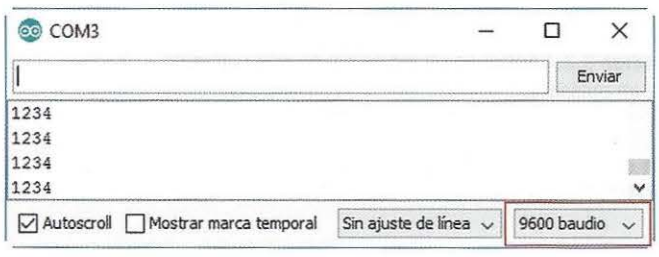


Figura 6.13. Resultados en el monitor serie.

6.8. Las entradas analógicas

Arduino dispone de puertos analógicos, 6 en el modelo Uno y 16 en el Mega, etiquetados como A0-A15 con rotulación en la placa como ANALOG IN. Estos puertos contienen un conversor de analógico a digital (ADC), que puede leer el valor entre 0 V y 5 V.

Para ver cómo funcionan estas entradas, se coloca un potenciómetro con la alimentación en sus extremos y el cursor se conecta a uno de los puertos analógicos, de tal manera que, en el puerto analógico, tendrá un nivel de tensión que oscilará entre 5 V y 0 V en función de la posición del cursor.

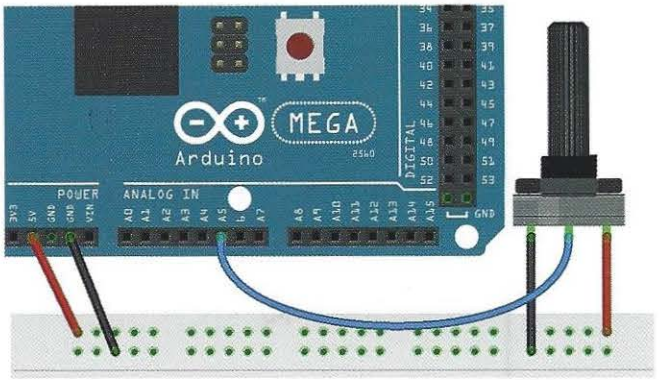


Figura 6.14. Montaje del potenciómetro conectado a una entrada analógica.

Los convertidores de Arduino Uno y Mega son de 10 bits de resolución, por lo que devuelve valores entre 0 y 2^{10} = = 1024 para tensiones entre 0 V y 5 V.

La instrucción para leer por un puerto analógico es **analogRead** y se acompaña con el número del puerto entre paréntesis. La lectura se guarda en una variable de tipo entero, en este caso, la variable se denomina *lectura* y el puerto analógico es A5:

```
int lectura;
lectura = analogRead (A5);
```

También puede ponerse estas instrucciones en una sola línea de código

```
int lectura = analogRead (A5);
```

SABÍAS QUE...

Si, en el circuito anterior, desconectas el potenciómetro y lees la entrada, se observa una serie de valores aleatorios que no tienen sentido, lo que se debe a interferencias y al ruido de radiofrecuencia.

ACTIVIDAD RESUELTA 6.3

Conecta un potenciómetro de 10 kΩ a la entrada analógica A5 y, utilizando el monitor serie, muestra el valor leído.

Solución:

El montaje electrónico es el que se muestra en la Figura 6.14 y el programa es el siguiente:

```
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int lectura = analogRead (A5);
    Serial.println (lectura);
    delay (200);
}
```

Código 6.4. Lectura de una entrada analógica.

Al probar el programa, se observa que proporciona valores entre 0 y 1023 –es decir, no indica el nivel de tensión directamente–, para ello se debe hacer una regla de tres y obtener su conversión.

6.9. La conexión de cargas

Las salidas del Arduino trabajan con corrientes muy pequeñas, que son insuficientes para poder activar una carga de cierta potencia. Para solucionar este inconveniente, se emplea algún elemento que trabaje como preactuador, es decir, un elemento que necesite de una corriente pequeña de mando, pero pueda manejar corrientes más elevadas (algo similar a lo que hace un contactor). Se tienen básicamente dos elementos: el relé y el transistor.

6.9.1. El relé

Los relés para Arduino trabajan a una tensión de mando de 5 V y, para sus contactos, tensiones de hasta 250 V con corrientes entre 5 A y 10 A según modelos. Suelen tener un solo contacto de fuerza. Tienen tres bornes para la parte de mando (positivo, negativo y señal) y otros tres para la salida (común, normalmente abierto y cerrado).

El control de cargas de potencia a través de un relé es muy sencillo y solamente consiste en activar una salida digital, a la cual se conecta el pin de señal del relé y, en los contactos de potencia, se conecta la carga que quiere controlarse, que puede ser, por ejemplo, una lámpara de iluminación. Hay que tener la precaución de no sobrepasar la corriente máxima que soportan los contactos del relé.

Figura 6.15. Relé para Arduino.

6.9.2. El transistor

Un transistor se puede utilizar para el control de cargas. Es de un tamaño más reducido que un relé y tiene velocidades de conmutación más rápidas. Existen dos tipos de transistores bipolares, el tipo PNP y el NPN, y, en cada uno de ellos, la conexión de la carga varía.

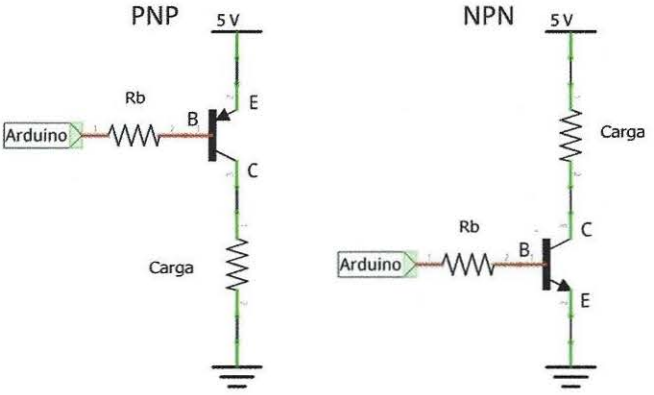


Figura 6.16. Conexión de la carga según el tipo de transistor bipolar.

Cuando la carga es de naturaleza inductiva, como puede ser un motor, se debe colocar un diodo en paralelo con la carga. Este diodo de protección recibe el nombre *flyback*.

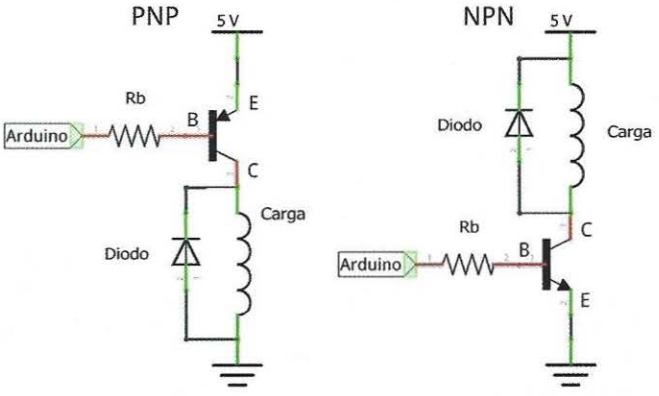


Figura 6.17. Conexión del diodo en paralelo con la carga por proteger.

El valor óhmico de la resistencia de base (Rb) depende de la tensión de alimentación (Vcc), que, para una tensión de 5 V, el valor es de 220 Ω o 330 Ω.

El transistor puede ser cualquiera y hay un amplio surtido, pero lo normal es colocar uno de uso genérico, que son baratos, como puede ser el 2N2222 o el BC546.

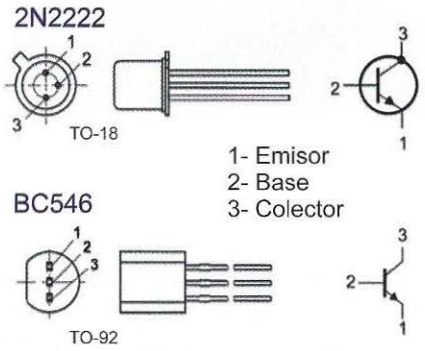


Figura 6.18. Patillaje del transistor 2N2222 y BC546.

Es importante conectar correctamente las patillas del transistor, así como verificar que el transistor elegido puede trabajar a la tensión y con la corriente máxima permitida para no dañarlo. Para ello es conveniente usar las hojas de características proporcionadas por el fabricante del transistor.

6.10. Actuadores

Hay muchos tipos de actuadores que se conectan a la salida del Arduino, aunque los más destacables son los que se estudian a continuación.

6.10.1. El diodo led y la regulación PWM

Hasta ahora, se ha estudiado cómo activar y desactivar un diodo led –que consiste en controlar una salida digital–, pero lo que se pretende ahora es ver cómo controlar el nivel de iluminación de un led. Casi todos los Arduino carecen de salidas analógicas, pero hacen uso de la técnica PWM (sigla del inglés *pulse width modulation*) –que consiste en generar una onda cuadrada e ir variando su ancho en cada ciclo, de tal manera que el valor eficaz de la salida es equivalente a una salida analógica–.

De la gráfica de la Figura 6.19, se deduce que, para un ciclo de trabajo del 50%, la salida correspondería al equivalente de una señal analógica de 2,5 V.

Para usar esta técnica, se deben utilizar los pines adecuados, que en la placa de Arduino vienen etiquetados como *PWM* o con el símbolo ~ delante del número del puerto. La orden para utilizar es **analogWrite,** a la cual le sigue el número de puerto y el valor que ha de poner en su salida.

analogWrite (puerto, valor);

El valor debe encontrarse entre 0 y 255 y el puerto, uno que permita el PWM.

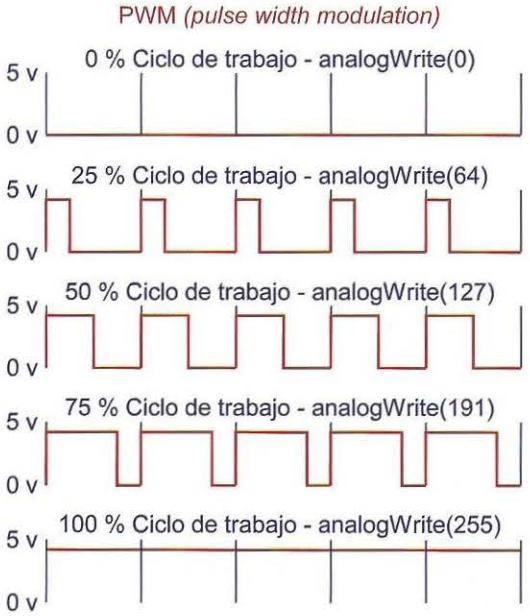


Figura 6.19. Señal PWM con diferentes porcentajes de ciclo de trabajo.

ACTIVIDAD RESUELTA 6.4

Conecta un diodo led a un puerto PWM y haz que varíe su iluminación.

Solución:

Se va a emplear el puerto PWM 9. El montaje electrónico es el siguiente con una resistencia de 330 Ω:

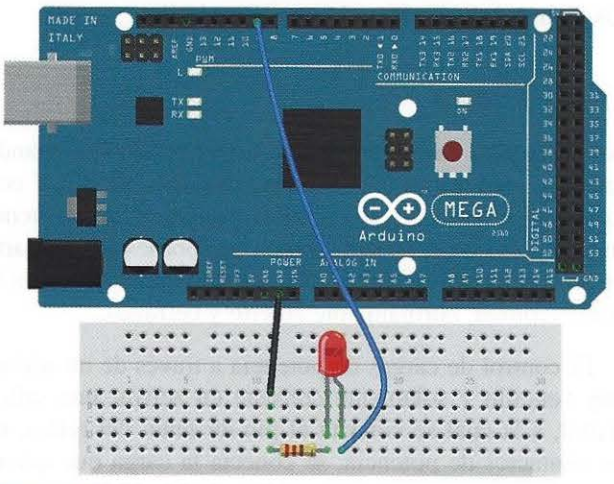


Figura 6.20. Montaje de un led a un puerto PWM.

El código del programa es el siguiente:

```
void setup()
{
    pinMode( 9, OUTPUT) ;
}
void loop()
{
    for ( int i= 0 ; i<255 ; i++)
    {
      analogWrite (9, i) ;
      delay( 10);
    }
}
```

Código 6.5. Regulación PWM sobre un led.

Se ha empleado una instrucción *for,* que lo que hace es repetir las instrucciones que tiene dentro un valor que va desde 0 hasta 255. Estas instrucciones son escribir en el puerto de salida, que es el número 9, el valor *i,* que es el que varía. Posteriormente, se lleva a cabo una pausa de 10 ms. Cuando llega a su valor máximo, se apaga y vuelve a comenzar.

Se puede realizar una modificación para que varíe constantemente sin ese apagón brusco, que consiste en contar desde –255 hasta 255 y tomar el valor absoluto **(abs),** es decir, sin el signo. Lo que realiza la parte que va desde –255 hasta 0 es apagarlo lentamente y la parte que va desde 0 hasta 255 es encenderlo lentamente.

```
void setup()
{
    pinMode( 9, OUTPUT) ;
}
void loop()
{
    for ( int i= -255 ; i<255 ; i++)
    {
      analogWrite (9, abs(i)) ;
      delay( 10);
    }
}
```

Código 6.6. Regulación PWM sobre un led.

6.10.2. El diodo led RGB

Un diodo emite luz en un color que depende de sus características de fabricación. Para conseguir que un diodo emita cualquier color, se emplean en realidad tres diodos bajo una misma carcasa. Estos diodos son de color rojo, verde y azul y, variando la intensidad de cada uno de ellos y sumándolos, se obtienen los diferentes colores.

El led RGB tiene cuatro pines, que son uno para cada color, y uno común (cátodo), que se conecta al negativo.

Figura 6.21. Mezcla de colores primarios.

SABÍAS QUE...

El nombre *RGB* proviene de la sigla en inglés de los tres colores primarios, rojo *(red),* verde *(green)* y azul *(blue).*

ACTIVIDAD RESUELTA 6.5

Conecta un diodo led RGB y haz que varíe su color pasando por rojo, verde y azul.

Solución:

Este diodo RGB se va a conectar a los puertos PWN 9, 10 y 11, junto con una resistencia de 330 Ω.

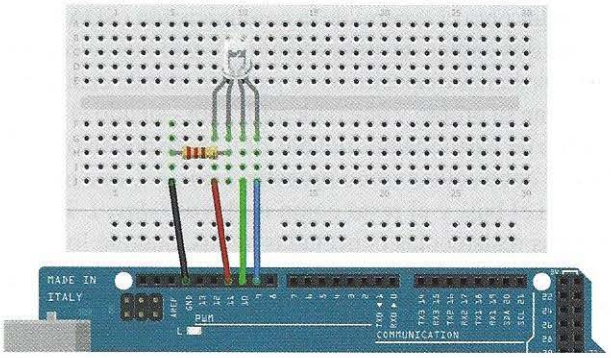


Figura 6.22. Conexión de un led RGB.

En el código del programa (Código 6.7), en la función *setup,* se configuran los puertos para utilizarlos como salidas.

Se ha utilizado una función que se ha creado denominada *Color,* a la cual se le siguen los valores de las tres componentes del color y hace una pausa de 500 ms. La creación de funciones propias es una de las características del lenguaje de programación que facilita la reutilización del código y lo clarifica.

```
void setup()
{
    pinMode(9, OUTPUT);
    pinMode(10, OUTPUT);
    pinMode(11, OUTPUT);
}

void loop()
{
    Color(255 ,0 ,0) ;
    delay(500);
    Color(0,255 ,0) ;
    delay(500);
    Color(0 ,0 ,255) ;
    delay(500);
    Color(0,0,0);
    delay(1000);
}

void Color(int R, int G, int B)
{
    analogWrite(9 , R) ; // Rojo
    analogWrite(10, G) ; // Verde
    analogWrite(11, B) ; // Azul
}
```

Código 6.7. Uso de un led RGB.

6.10.3. Pantalla LCD y el bus I2C

El bus I2C es una norma de comunicación entre componentes electrónicos que simplifica el cableado, ya que necesita dos cables para el control de la comunicación (SDA y SCL) más los dos cables de alimentación. A este bus se conectan todos los demás dispositivos en paralelo, donde cada uno de ellos tiene su propio y único identificador –siendo el número máximo de participantes del bus de 128 dispositivos–. Uno de ellos debe actuar como **maestro** *(master)*, que es el que controla el bus, y el resto como **esclavos** *(slave)*. También lleva dos resistencias conectadas al bus, aunque lo normal es que estas resistencias ya estén conectadas en el propio componente

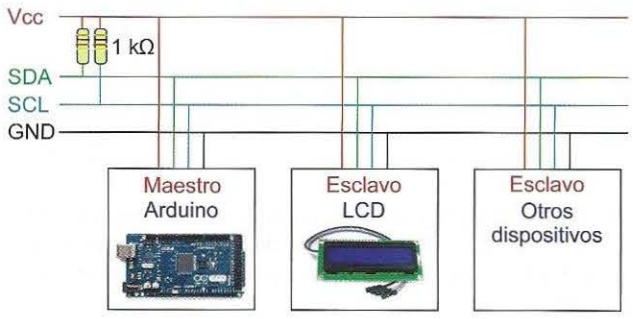


Figura 6.23. El bus I2C.

Según la placa de Arduino, los pines son los siguientes:

- En el Arduino Uno, los pines I2C están en los pines analógicos A4 (SDA) y A5 (SCL).
- En el Arduino Mega, son el 20 (SDA) y el 21 (SCL).

Al conectar un dispositivo al bus I2C, si se desconoce su dirección, se puede recurrir a un escáner.

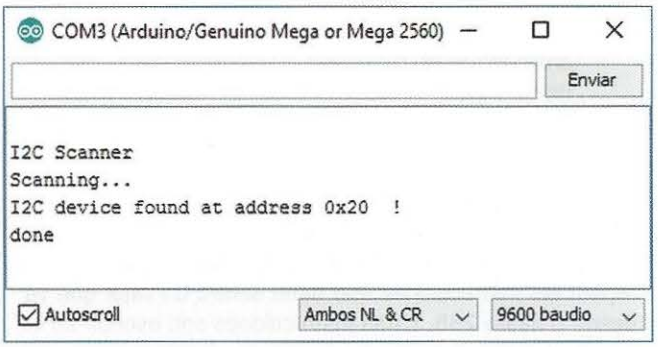


Figura 6.24. Resultado del escáner del bus I2C.

NOTA

En los Recursos digitales, previo registro de esta obra en www.paraninfo.es, está disponible el fichero I2C_scan.ino, así como las librerías necesarias.

Algunos componentes electrónicos necesitan las denominadas *librerías,* que son un conjunto de ficheros con código que indican a Arduino cómo debe trabajar y, de esta manera, se simplifica el uso de estos componentes.

En el caso de la pantalla LCD, se necesita la librería denominada *NewLiquidCrystal_1.3.4.zip* y, para instalarla desde el IDE de Arduino, hay que ir al menú: programa, incluir librería y añadir biblioteca .ZIP.

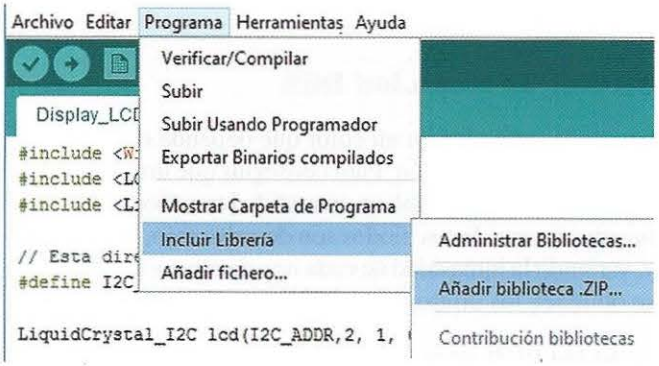


Figura 6.25. Añadir biblioteca .ZIP.

ACTIVIDAD RESUELTA 6.6

Conecta una pantalla LCD al bus I2C del Arduino y muestra un mensaje cualquiera.

Solución:

La conexión electrónica de la pantalla al Arduino consiste en conectar los dos hilos de alimentación y los hilos de comunicación SDA y SCL, verifica tu Arduino para asegurarte de los pines que han de emplearse (4 y 5 para Uno y 20 y 21 para Mega).

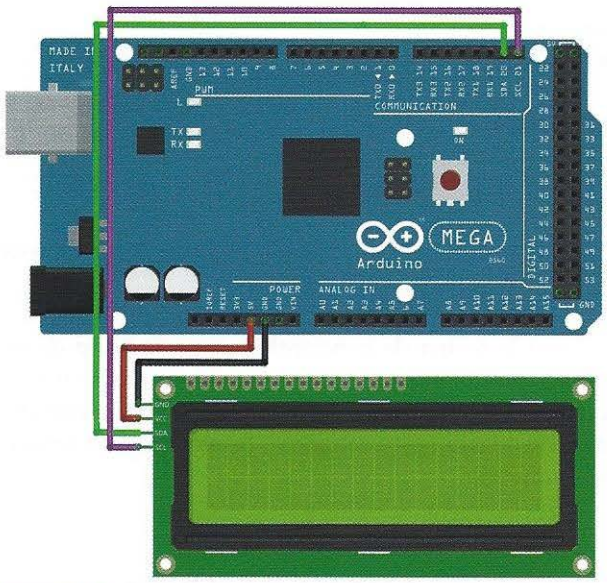


Figura 6.26. Montaje de una pantalla LCD al bus I2C.

```
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>

// Esta dirección se debe averiguar.
#define I2C_ADDR 0x20

LiquidCrystal_I2C lcd(I2C_ADDR,2, 1, 0, 4, 5, 6, 7);

void setup()
{
    lcd.begin (16,2);
    lcd.setBacklightPin(3,POSITIVE);
    lcd.setBacklight(HIGH);

    lcd.home ();
    lcd.print("Hola Mundo");
    lcd.setCursor ( 0, 1 );
    lcd.print("Paraninfo");
}
void loop()
{
}
```

Código 6.8. Uso de una pantalla LCD.

Se deben incluir los tres ficheros de cabecera con extensión .h para que el código tome las librerías. También se debe averiguar cuál es la dirección de la pantalla en el bus I2C, para ello se ha utilizado el escáner de I2C dando la dirección 0x20.

Se crea una instancia mediante el siguiente código:

```
LiquidCrystal_I2C lcd (I2C_ADDR,2, 1, 0, 4, 5, 6, 7);
```

A partir de aquí, ya se puede utilizar la pantalla LCD. Primero, se inicializa indicando que tiene 16 caracteres y dos líneas con la orden *begin*. Luego, es solo cuestión de situarse en la posición deseada para escribir el mensaje. Con home, se sitúa en la primera posición de la primera línea y, con setCursor, se le indica la posición y la línea, teniendo en cuenta que se empieza a contar desde 0. Con *printf*, se le indica el texto que debe escribir.

Como el código se ha puesto en la función *setup*, solo se mostrará una vez cuando arranque la placa.

6.10.4. El zumbador

Un zumbador es un elemento destinado a dar un aviso de tipo acústico. Basa su funcionamiento al efecto de la piezoelectricidad, en la cual, al ser sometido a una corriente eléctrica, el elemento se pone a vibrar generando un tono audible.

Estos elementos poseen polaridad, basta con conectar el borne positivo a una de las salidas digitales PWM y el otro borne al negativo. Existe otro tipo de zumbador que cuenta con tres bornes, dos para alimentación y el tercero para la señal.

Para la activación, se pone en el puerto PWM un valor. Como es un poco molesto, se crea una función denominada *beep*, la cual pone un valor, por ejemplo, 20, realiza una pausa y, luego, lo apaga.

ACTIVIDAD RESUELTA 6.7

Conecta un zumbador a tu Arduino y realiza el programa para simular el sonido de un despertador, es decir, ha de generar zumbidos de manera continua.

Solución:

Se conecta el zumbador entre una salida PWM como, por ejemplo, el puerto 9 del Arduino Mega y el negativo y, si el módulo del zumbador necesita positivo de alimentación, se conecta.

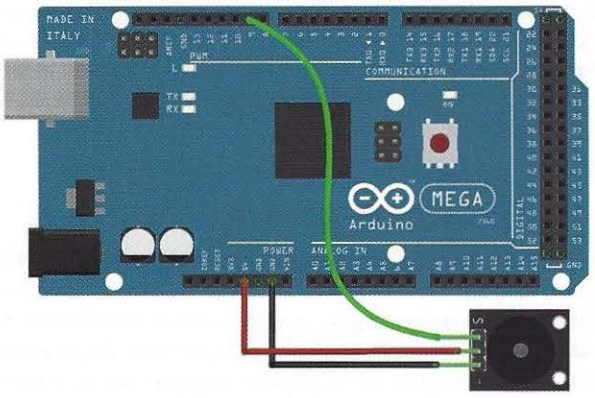


Figura 6.27. Montaje de un zumbador.

Para la función *beep*, se ha usado una variable de tipo *char*, que permite almacenar números de −127 a 128. Se le ha añadido el prefijo *unsigned* para indicarle que los números son sin signo y, por tanto, puede almacenar desde 0 hasta 255.

```
void setup()
{
pinMode(9, OUTPUT);
}
void loop()
{
    beep(250);
}
void beep(unsigned char pausa)
{
    analogWrite(9, 20);
    delay(pausa);
    analogWrite(9, 0);
    delay(pausa);
}
```

Código 6.9. Uso de un zumbador.

6.11. Sensores

En el mundo de Arduino, hay sensores para prácticamente medir cualquier variable física. El funcionamiento de la lectura de la variable del sensor es similar para todos ellos y, por tanto, solo se estudiarán algunos.

6.11.1. El sensor de temperatura y humedad

Existen varios tipos de sensores que, de una manera conjunta y bajo un mismo encapsulado, proporcionan la lectura de las magnitudes físicas de temperatura y humedad. Entre los más conocidos, están los modelos DHT11 y DHT22.

Tabla 6.1. Comparativa entre los sensores DHT11 y DHT22

	DHT11	DHT22
Humedad	Del 20 % al 80 % HR	Del 0 % al 100 % HR
Precisión de la humedad	±5 % HR	±2 % HR
Temperatura	De los 0 °C a los 50 °C	De los −40 °C a los 80 °C
Precisión de la temperatura	±2 °C	±0,5 °C
Número de lecturas	1 por segundo	2 por segundo

Tanto la forma de conectarlos como la parte de programación es la misma. Estos dispositivos incorporan una electrónica que se encarga de realizar la conversión de la medición de analógico a digital y proporciona su valor por un pin digital.

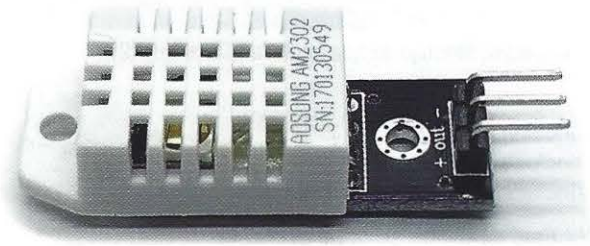


Figura 6.28. Sensor DHT22.

En el modelo de cuatro pines, uno de ellos no se conecta. Dos pines son de alimentación y uno de señal, que se conecta en un puerto digital. Además, entre el pin de señal (data) y el positivo de alimentación (Vcc), se conecta una resistencia de un valor entre 4,7 kΩ y 10 kΩ. Por último, para utilizar este sensor con Arduino, se debe instalar la librería DHT11, que sirve para ambos modelos de sensor.

Recuerda

Para insertar una librería en Arduino, hay que ir al menú programa, incluir librería y añadir biblioteca. Después, seleccionar el fichero comprimido zip o la carpeta.

ACTIVIDAD RESUELTA 6.8

Realiza la medición de la temperatura y la humedad con tu Arduino. Muestra la lectura por el monitor serie.

Solución:

Va a emplearse un sensor de temperatura y humedad DHT22. Con el modelo DHT11, hay que realizar los cambios indicados en el código, que se limitan a señalar el tipo de sensor (DHT11 o DHT22) y el puerto donde se conecta, que en este caso es el 5.

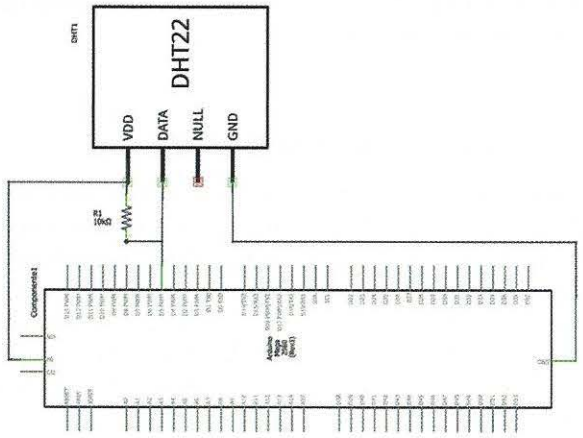


Figura 6.29. Esquema electrónico de conexionado del sensor DHT22.

En la parte de configuración *(setup)*, se inicia la comunicación serie a 9600 bps y se arranca el sensor *(begin)*.

La lectura se realiza con **readHumidity** y con **readTemperature** y se guardan en unas variables de tipo *float*, que son las empleadas para guardar números decimales. Finalmente, se envían al monitor serie y se realiza una pausa de 2 segundos.

```
#include "DHT.h"
#define DHTpin 5 // Puerto 5
#define DHTtype DHT22 // o DHT11
DHT dht(DHTpin, DHTtype);

void setup()
{
    Serial.begin(9600);
    dht.begin(); //Se inicia el sensor
}
void loop()
{
    // Se lee humedad y temperatura
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    //Se muestran las lecturas
    Serial.println("Humedad: ");
    Serial.println(h);
    Serial.println("Temperatura: ");
    Serial.println(t);
    delay(2000);
}
```

Código 6.10. Uso de un sensor de temperatura y humedad.

6.11.2. El sensor de ultrasonidos

El sensor de ultrasonidos se utiliza para detectar objetos que se encuentran enfrente de él y proporciona su distancia. Se basa en enviar una onda de ultrasonidos, la cual rebota con el objeto que ha de detectar y vuelve al sensor. La electrónica que contiene se encarga de medir el tiempo transcurrido desde que envía esa onda hasta que la recibe, funcionando con el mismo principio que el sónar o que la ecografía.

Entre los modelos más conocidos, se encuentra el SRF05, que dispone de un emisor junto a un receptor y que funciona a una tensión de 5 V. Este sensor envía un tren de 8 pulsos a una frecuencia de 40 kHz y espera un máximo de unos 30 ms para recibir el eco; si no lo recibe, es porque no hay ningún objeto delante. Este valor se define por *software*.

La velocidad del sonido en el aire es de 340 m/s o, lo que es lo mismo, 29,1 μS/cm, y hay que tener en cuenta que el sonido va hacia el objeto y vuelve, por tanto, el tiempo invertido es el doble. Este valor se necesita para obtener la distancia.

La conexión consiste en dos hilos para la alimentación eléctrica, un hilo para el *trigger* o pulso de inicio y otro para el eco, donde se lee la anchura del pulso del eco. Este modo es compatible con el modelo anterior (SRF04). Existe otra modalidad que emplea un solo terminal para las funciones de *trigger* y de eco.

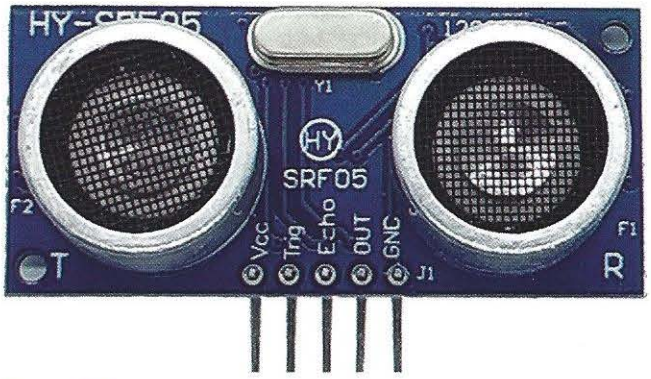


Figura 6.30. Sensor de ultrasonidos SRF05.

ACTIVIDAD RESUELTA 6.9

Realiza un medidor de distancias empleando un sensor de ultrasonidos para tu Arduino y muestra el resultado por el monitor serie

Solución:

Tanto el sensor SRF04 como el SRF05 se conectan de igual manera, dos hilos de alimentación, el pin del *trigger* y el pin de eco, estos dos últimos en los puertos digitales 7 y 6, respectivamente.

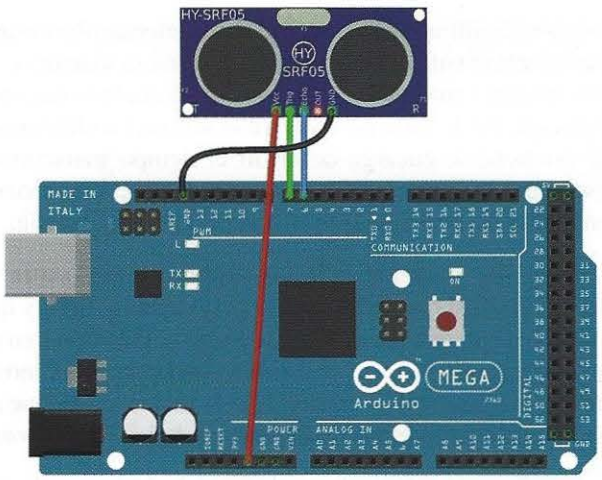


Figura 6.31. Montaje del sensor de ultrasonidos SRF05 al Arduino.

El código es el siguiente:

```
#define PIN_TRIGGER 7
#define PIN_ECHO 6
#define ESPERA_ENTRE_LECTURAS 500
#define TIMEOUT_PULSO 25000

long cronometro;
long reloj=0;

void setup()
{
    Serial.begin(9600);
    pinMode(PIN_ECHO,INPUT);
    pinMode(PIN_TRIGGER,OUTPUT);
    digitalWrite(PIN_TRIGGER,LOW);
    delayMicroseconds(2);
}

void loop()
{
    long duracion, distancia;

    cronometro=millis()-reloj;
    if(cronometro>ESPERA_ENTRE_LECTURAS)
    {
      digitalWrite(PIN_TRIGGER, LOW);
      delayMicroseconds(2);
      digitalWrite(PIN_TRIGGER, HIGH);
      delayMicroseconds(12);
      digitalWrite(PIN_TRIGGER, LOW);
      duracion = pulseIn(PIN_ECHO, HIGH);
      distancia = duracion / (2.0 * 29.1);
      Serial.println(String(distancia) + " cm.");
      reloj=millis();
    }
}
```

Código 6.11. Uso de un sensor de ultrasonidos.

En el Código 6.11, además de definir los pines de conexión del *trigger* y el eco, se fija un tiempo entre una lectura y la siguiente de 500 ms y se le indica un tiempo máximo para recibir el eco de 25 ms.

En el *setup*, además de configurar los puertos, el pin del *trigger* se pone a cero *(low)* para inicializarlo.

El funcionamiento es el siguiente *(loop):* se pone el puerto del *trigger* a cero para desactivarlo y luego se activa *(high)* para enviar el tren de pulsos de ultrasonido y se corta *(low)* y, a continuación, se espera a recibir el eco, cuyo valor se convierte a distancias en centímetros, todo ello dentro de una función para que emita la onda cada cierto tiempo (ESPERA_ENTRE_LECTURAS), ya que no es necesario que esté continuamente trabajando.

6.12. Servidor domótico

En esta unidad, se ha estudiado el empleo básico de un Arduino, pero se puede profundizar mucho más, por ello el siguiente paso es emplear un servidor domótico que se encargue de gestionar todos los dispositivos que se han montado con el Arduino. Este servidor, además, sirve para poder enlazar con internet y desde cualquier lugar y equipo –como puede ser un teléfono inteligente–, supervisar y gestionar todo el sistema.

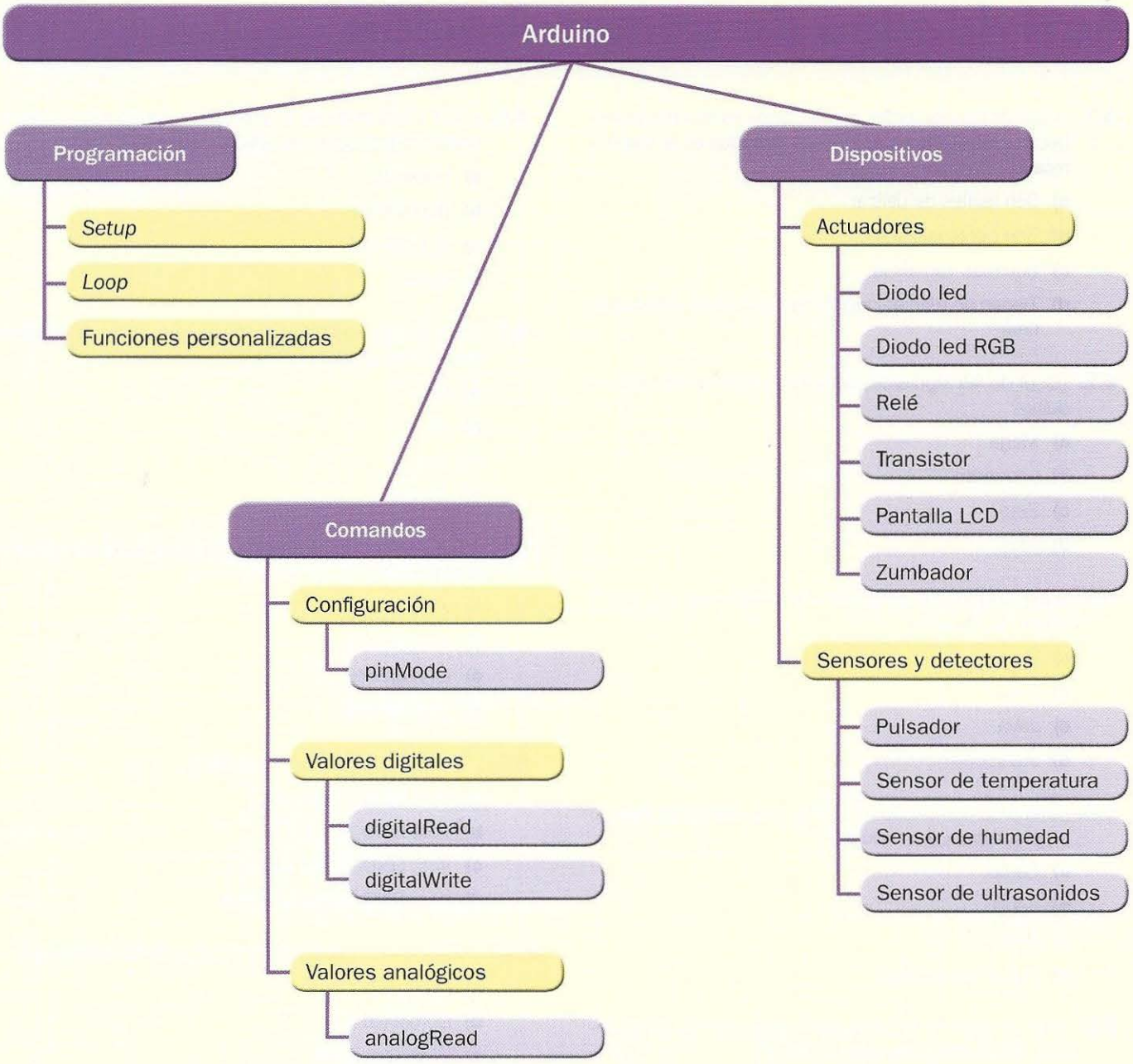
El servidor domótico puede ser un ordenador, pero eso implica tenerlo en funcionamiento todo el tiempo. La solución que se emplea es utilizar un dispositivo que realice esta función, pero con menos recursos, como puede ser una placa **Raspberry Pi.** Para el *software* del servidor, hay varias opciones, por destacar alguna, puede ser OpenHab, Home Assistant o Ubidots.

Figura 6.32. Logo del *software* OpenHab.

Figura 6.33. Logo del *software* Home Assistant.

Figura 6.34. Logo del *software* Ubidots.

Mapa conceptual



Actividades de comprobación

6.1. ¿Cuál de las siguientes afirmaciones es incorrecta respecto a los sistemas domóticos basados en la electrónica?

a) Son fáciles de utilizar.

b) Son personalizables.

c) Son más versátiles.

d) Tienen un mantenimiento fácil por parte del usuario final.

6.2. ¿Cuál de las siguientes opciones no es un tipo de Arduino?

a) Mega.

b) Leonardo.

c) Zero.

d) Plus.

6.3. ¿Qué lenguaje de programación se emplea con el IDE oficial de Arduino?

a) C.

b) Cobol.

c) Java.

d) JavaScript.

6.4. ¿Cómo se llaman las funciones que lleva por defecto un programa para Arduino?

a) *Setup*.

b) *Loop*.

c) *Setup* y *loop*.

d) *Setup*, *loop* y *start*.

6.5. ¿En qué función se coloca la inicialización de las variables de un programa para Arduino?

a) *Start*.

b) *Setup*.

c) *Loop*.

d) *Function*.

6.6. ¿Qué instrucción se emplea para configurar un pin o puerto digital como de salida o entrada?

a) pinMode.

b) digitalWrite.

c) digitalRead.

d) *delay*.

6.7. ¿Qué tipo de variable se emplea para guardar un valor binario de manera eficiente?

a) *bool*.

b) *int*.

c) *float*.

d) *double*.

6.8. ¿Qué instrucción se emplea para leer el valor en un pin analógico?

a) writeAnalog.

b) readAnalog.

c) analogWrite.

d) analogRead.

6.9. ¿Cuántos hilos tiene un bus I2C?

a) Vcc, GND, SDA y SLC.

b) Vcc, GND, ICA e ICB.

c) Vcc, GND, Com1 y Com2.

d) Vcc, GND, ComA y ComB.

6.10. ¿Cuál de los siguientes no es un sistema domótico?

a) OpenHab.

b) DomoHome.

c) Home Assistant.

d) Ubidots.

Actividades de aplicación

6.11. Realiza un programa que encienda un led situado en el puerto 2 y otro en puerto 3 cuando se accione un pulsador situado en el pin 8.

6.12. Desarrolla un programa que, al accionar un pulsador situado en el pin 7, muestre por el monitor serie la suma de los números 123 y 87.

6.13. Diseña un programa que lea el valor de tensión que proporciona un conmutador situado en el pin analógico 3 y lo muestre en el monitor serie.

6.14. Modifica el programa de la Actividad 6.13 para que se genere un sonido desde un zumbador cada vez que se accione el pulsador.

6.15. Elabora un programa que encienda un led situado en el puerto digital 2. Al pulsar una vez, se debe encenderse el led a máxima intensidad. Si, a continuación, se pulsa, debe poner el led al 50 % de iluminación y, si acto seguido se pulsa el led, tiene que apagarse y quedar listo para comenzar de nuevo.

6.16. Establece un programa que muestre por una pantalla LCD tu nombre y apellidos.

6.17. Lleva a cabo un programa que muestre por una pantalla LCD el valor de un led conectado en el pin digital 4 que sea controlado por un pulsador conectado en el pin digital 5.

6.18. Define un programa que lea el valor de la temperatura y la humedad. Cuando la temperatura esté por encima de 25 °C, ha de activar un led y, cuando la humedad sea mayor del 60 %, tiene que activar otro led. Además, debe mostrar los valores de temperatura y humedad en una pantalla LCD.

Actividades de ampliación

6.19. Entra en la página web oficial de Arduino y descarga e instala el IDE de programación.

6.20. Busca en internet diferentes tipos de pantallas LCD para Arduino. Fíjate en el número de filas y columnas que puede representar y también si el cableado es mediante el bus I2C.

6.21. Localiza los sensores de temperatura y humedad DHT22 y DHT11 y compara sus prestaciones. Fíjate en si el número de pines es de tres o cuatro.

6.22. Halla el sensor de ultrasonidos para Arduino, fíjate en si es el modelo SRF04 o el SRF05 y compara las características y prestaciones entre ambos modelos.