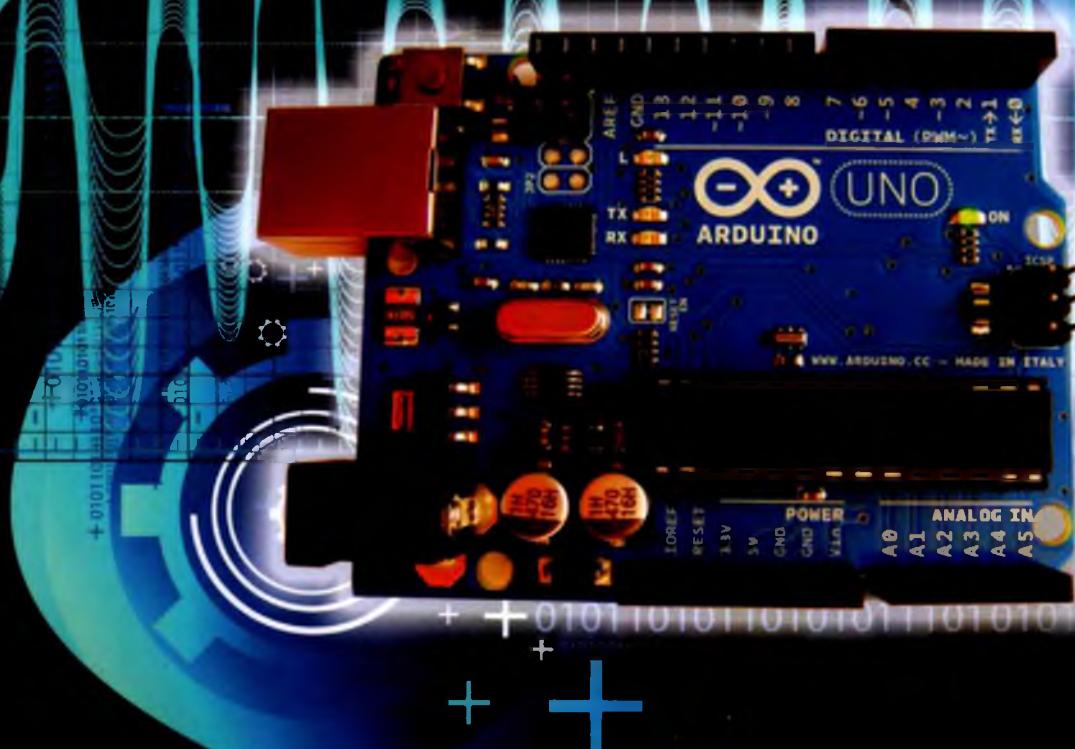


30 PROYECTOS CON ARDUINO



Construya y programe sus propios proyectos de Arduino

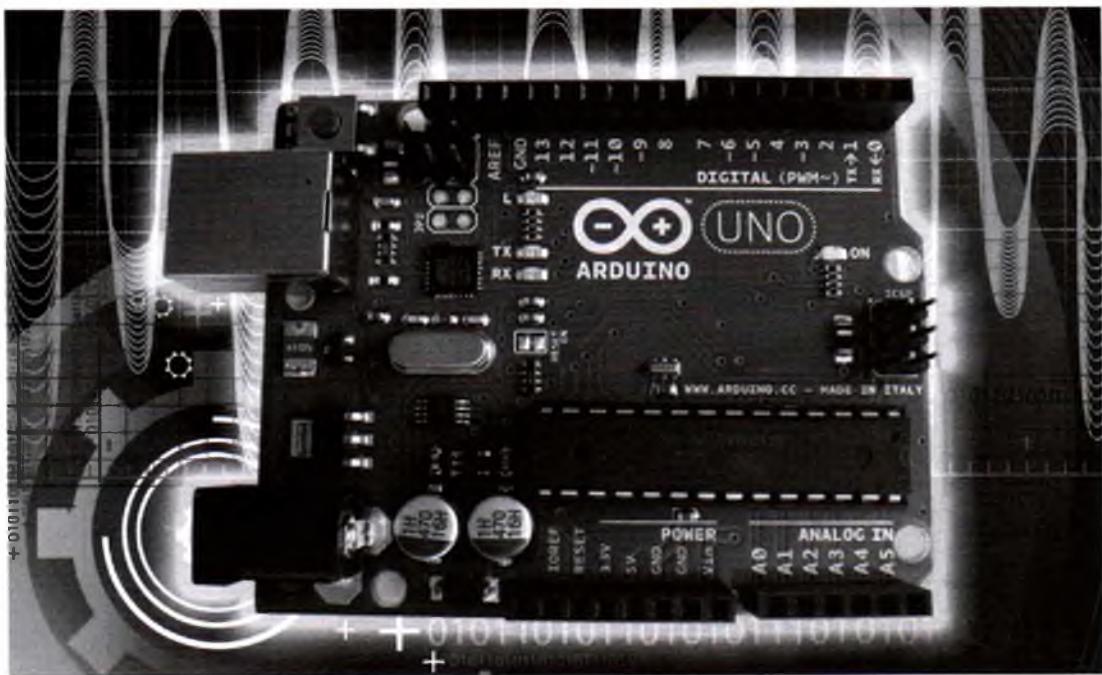
Compatible con múltiples plataformas; Mac, Windows y Linux

**Un completo libro sobre Arduino; con ilustraciones, esquemas, fotografías,
e ilustraciones paso a paso**

SIMON MONK

EDITORIAL ESTRIBOR

30 Proyectos con Arduino™



30 Proyectos con Arduino™

Simon Monk

Revisión técnica en español:

JAVIER POMPA

Prof. IES Tecnología y FP en Electrónica

Editorial Esteribor

“Arduino” es una marca comercial y registrada del equipo Arduino.

30 PROYECTOS CON ARDUINO™, 1ª Edición

No está permitida la reproducción total o parcial de esta publicación, ni su tratamiento informático, ni la transmisión de ninguna forma o por cualquier medio, ya sea electrónico, mecánico, por fotocopia, por registro u otros métodos, sin el permiso previo y por escrito del editor.

DERECHOS RESERVADOS © 2012, respecto a la edición en español por:

EDITORIAL ESTRIBOR, S.L.

c/ Alba, 6B
28043 Madrid
ESPAÑA

Traducido bajo licencia de McGraw-Hill, Inc. de la edición en inglés de:

30 ARDUINO™ PROJECTS FOR THE EVIL GENIUS™

Copyright versión original © MMX, por McGraw-Hill, Inc.

(ISBN-13 978-0-07-174133-0). Todos los derechos reservados.

Copyright versión en español © MMXII, por Editorial Estripor, S.L. Todos los derechos reservados.

ISBN: 978-84-940030-0-4

Depósito legal: M-10999-2012

Diseño de portada

Francisco Luque Ulloa

Foto de portada

Javier Pompa

Maquetación e impresión

Montesinos Artes Gráficas (Madrid)

Adaptación gráficos y fotos (ver pág. ix)

Javier Pompa

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

La información ha sido obtenida por la Editorial de fuentes que se creen que son fiables. No obstante, debido a la posibilidad de error humano o mecánico por parte de las fuentes de información utilizadas en la elaboración del libro, Editorial Estripor no garantiza la exactitud, adecuación o integridad de la información y no se hace responsable de los errores u omisiones o los resultados obtenidos con el uso de dicha información. El libro se proporciona “tal cual”.

*A mi difunto padre, Hugh Monk, de quién heredé el amor por la electrónica.
Él se habría divertido mucho con todo esto.*

Acerca del Autor

Dr. Simon Monk es titulado en Cibernética y en Informática y Doctor en Ingeniería de Software. Ha pasado varios años como profesor universitario antes de regresar al sector industrial. Es cofundador de la empresa de software móvil Momote, Ltd. Es aficionado activo a la electrónica desde sus días de escuela y colabora ocasionalmente con revistas de electrónica para aficionados.

Acerca del revisor técnico en español:

Javier Pompa es Licenciado en Física y Profesor de IES de Tecnología y FP en Electrónica. Es un enamorado de la electrónica y de la enseñanza y en la actualidad se encuentra preparando una ambiciosa serie de libros sobre electrónica.

Tabla de Contenido

Lista de Figuras	ix
Prefacio	xiii
Agradecimientos	xvii
Introducción	xix
1 Arranque rápido	1
Encendido	1
Instalación del Software	1
Configuración del entorno Arduino.....	6
Descarga del software del proyecto.....	6
Proyecto 1 LED intermitente	8
Placa de pruebas de inserción de componentes (<i>protoboard</i>)	11
Resumen.....	13
2 Un recorrido por Arduino.....	15
Microcontroladores	15
¿Qué hay en una placa Arduino?.....	15
La familia Arduino.....	20
El lenguaje C	21
Resumen.....	25
3 Proyectos con LEDs	27
Proyecto 2 Intermitente S.O.S. de código Morse	27
Loops (bucles)	29
Arrays (matrices)	30
Proyecto 3 Traductor de código Morse	31
Proyecto 4 Traductor de código Morse de gran intensidad.....	35
Resumen.....	40
4 Más proyectos con LEDs.....	41
Entradas y salidas digitales	41
Proyecto 5 Modelo de semáforo	41
Proyecto 6 Luz Estroboscópica	44
Proyecto 7 Luz para el T.A.E.....	47
Proyecto 8 Luz estroboscópica de alta potencia	52
Generación de números aleatorios.....	55
Proyecto 9 Dado de LEDs	55
Resumen.....	59
5 Proyectos con sensores	61
Proyecto 10 Código de seguridad con el teclado	61
Codificadores giratorios.....	67
Proyecto 11 Modelo de semáforo basado en codificador giratorio.....	68
Detección de la luz	72

Proyecto 12 Monitor de pulsaciones	73
Medición de la temperatura	77
Proyecto 13 Registrador de temperaturas USB	77
Resumen	83
6 Proyectos de luz	85
Proyecto 14 LED multicolor	85
LEDs de siete segmentos	89
Proyecto 15 Dado doble con display de siete segmentos	91
Proyecto 16 Matriz de LEDs	95
Displays LCD	101
Proyecto 17 Placa de mensajes USB	102
Resumen	105
7 Proyectos de sonido	107
Proyecto 18 Osciloscopio	107
Generación de sonido	111
Proyecto 19 Reproductor de música	112
Proyecto 20 Arpa de luz	117
Proyecto 21 Medidor VU	120
Resumen	124
8 Proyectos de energía eléctrica	125
Proyecto 22 Termostato LCD	125
Proyecto 23 Ventilador controlado por ordenador	132
Controladores de puente H	134
Proyecto 24 Hipnotizador	134
Servomotores	138
Proyecto 25 Láser servo-controlado	138
Resumen	142
9 Proyectos varios	145
Proyecto 26 Detector de mentiras	145
Proyecto 27 Cerradura magnética	148
Proyecto 28 Mando a distancia por infrarrojos	153
Proyecto 29 Reloj Lilypad	159
Proyecto 30 Temporizador de cuenta atrás	163
Resumen	168
10 Sus Proyectos	169
Circuitos	169
Componentes	171
Herramientas	175
Ideas para proyectos	179
Apéndice: Componentes y suministros	181
Proveedores	181
Kit de componentes para principiantes	185
Index	187

Lista de figuras

- Figura 1-1** Placa Arduino UNO alimentada y con el LED encendido. *
- Figura 1-2** Descarga del software de Arduino para Windows. *
- Figura 1-3** Extracción del software de Arduino en Windows.
- Figura 1-4** La opción de menú Extraer todo en Windows.
- Figura 1-5** Establecer el directorio para la extracción. *
- Figura 1-6** Asistente de Nuevo Hardware Encontrado de Windows. *
- Figura 1-7** Establecer la ubicación de los controladores USB. *
- Figura 1-8** Inicio del software Arduino desde Windows. *
- Figura 1-9** Instalación del software Arduino en Mac OS X.
- Figura 1-10** Instalación de los controladores USB en Mac OS X.
- Figura 1-11** Configuración del puerto serie del Mac.
- Figura 1-12** Configuración del puerto serie en Windows.
- Figura 1-13** Configuración de la placa.
- Figura 1-14** Carga del sketch Blink de ejemplo.
- Figura 1-15** Carga del sketch en la placa Arduino.
- Figura 1-16** Esquema electrónico de un LED conectado a la placa Arduino. *
- Figura 1-17** LED conectado en serie con una resistencia.
- Figura 1-18** LED conectado directamente a la placa Arduino. *
- Figura 1-19** Proyecto 1 en la placa de pruebas. *
- Figura 1-20** Diseño de la placa de pruebas del Proyecto 1.
-
- Figura 2-1** Componentes de una placa Arduino. *
- Figura 2-2** La ley de Ohm. *
- Figura 2-3** LED y resistencia en serie. *
- Figura 2-4** Diagrama de bloques del Atmega328.
- Figura 2-5** Placa Arduino Lilypad.
-
- Figura 3-1** Traductor de código Morse. *
- Figura 3-2** Ejecución de Serial Monitor. *
- Figura 3-3** La ventana de Serial Monitor. *
- Figura 3-4** Funcionamiento de un transistor bipolar NPN. *
- Figura 3-5** Esquema electrónico del manejo de un LED de gran potencia. *
- Figura 3-6** Diseño de la placa de pruebas del Proyecto 4.
- Figura 3-7** Conexión de cables sin soldadura en el LED Luxeon
- Figura 3-8** Fotografía de una placa de pruebas completa para el Proyecto 4.
- Figura 3-9** Protoshield en forma de kit.
- Figura 3-10** Parte inferior de la Protoshield.
- Figura 3-11** Montaje de Protoshield básico.
- Figura 3-12** Diseño de Protoshield del Proyecto 4. *
- Figura 3-13** Shield Luxeon completa conectada a una placa Arduino.

- Figura 4-1** Esquema electrónico del Proyecto 5.
- Figura 4-2** Proyecto 5 . Modelo de semáforo.
- Figura 4-3** Diseño de la placa de pruebas del Proyecto 5
- Figura 4-4** Esquema electrónico del Proyecto 6. *
- Figura 4-5** Funcionamiento interno de un potenciómetro.
- Figura 4-6** Diseño de la placa de pruebas del Proyecto 6.
- Figura 4-7** Diseño de la Protoshield para el Proyecto 6. *
- Figura 4-8** Montaje de un cable para alimentación externa con pila.
- Figura 4-9** Esquema electrónico del Proyecto 7. *
- Figura 4-10** Proyecto 7. Montaje de luz de gran potencia.
- Figura 4-11** Distribución de la placa perforada. *
- Figura 4-12** Proyecto 7. Luz para el T.A.E.
- Figura 4-13** Esquema electrónico del Proyecto 9. *
- Figura 4-14** Distribución de la placa de pruebas para el Proyecto 9.
- Figura 4-15** Proyecto 9. Dado de LEDs.

- Figura 5-1** Esquema electrónico del Proyecto 10. *
- Figura 5-2** Teclado matricial de 12 botones
- Figura 5-3** Soldadura de pines al teclado.
- Figura 5-4** Comprobación de las conexiones del teclado.
- Figura 5-5** Diseño de la placa de pruebas del Proyecto 10.
- Figura 5-6** Proyecto 10. Código de seguridad con teclado
- Figura 5-7** Instalación de la biblioteca para Windows.
- Figura 5-8** Instalación de la biblioteca para Mac.
- Figura 5-9** Pulso de un codificador giratorio.
- Figura 5-10** Esquema electrónico del Proyecto 11. *
- Figura 5-11** Disposición de componentes del Proyecto 11 en la placa de pruebas.
- Figura 5-12** Utilización de una LDR para medir la luz.
- Figura 5-13** Esquema electrónico del Proyecto 12. *
- Figura 5-14** Tubo sensor para el monitor cardíaco.
- Figura 5-15** Disposición de los componentes en la placa de pruebas del Proyecto 12.
- Figura 5-16** Proyecto 12. Monitor de pulsaciones.
- Figura 5-17** Datos de prueba del monitor cardíaco en una hoja de cálculo.
- Figura 5-18** Circuito electrónico del Proyecto 13.
- Figura 5-19** Placa Arduino alimentada con el LED encendido.
- Figura 5-20** Emisión de comandos a través del Serial Monitor.
- Figura 5-21** Datos a copiar y pegar en una hoja de cálculo.
- Figura 5-22** Datos de temperatura importados en una hoja de cálculo.

- Figura 6-1** Circuito electrónico del Proyecto 14. *
- Figura 6-2** Diseño del circuito del Proyecto 14 sobre la placa de pruebas.
- Figura 6-3** Proyecto 14. Led multicolor.
- Figura 6-4** Display de siete segmentos.

- Figura 6-5** Placa Arduino manejando un display de siete segmentos.
- Figura 6-6** Manejo de más de un display de siete segmentos desde una placa Arduino.
- Figura 6-7** Circuito electrónico del Proyecto 15.
- Figura 6-8** Diseño del circuito del Proyecto 15 sobre la placa de pruebas.
- Figura 6-9** Proyecto 15. Display doble de LEDs de siete segmentos.
- Figura 6-10** Proyecto 16. Matriz de LEDs.
- Figura 6-11** Circuito electrónico del Proyecto 16.
- Figura 6-12** Diseño del circuito del Proyecto 16 sobre la placa de pruebas.
- Figura 6-13** Módulo LCD de 2 por 16 caracteres.
- Figura 6-14** Circuito electrónico del Proyecto 17. *
- Figura 6-15** Diseño del circuito del Proyecto 17 sobre la placa de pruebas.
-
- Figura 7-1** Ruido de una señal de 50-Hz en un osciloscopio. *
- Figura 7-2** Esquema electrónico del Proyecto 18. *
- Figura 7-3** Diseño del circuito del Proyecto 18 sobre la placa de pruebas. *
- Figura 7-4** Proyecto 18. Osciloscopio.
- Figura 7-5** Ondas cuadradas y senoidales.
- Figura 7-6** CDA utilizando una escalera R-2R. *
- Figura 7-7** Esquema electrónico del Proyecto 19.
- Figura 7-8** Diseño del circuito del Proyecto 19 sobre la placa de pruebas.
- Figura 7-9** Representación gráfica del array sin16.
- Figura 7-10** Esquema electrónico del Proyecto 20.
- Figura 7-11** Diseño del circuito del Proyecto 20 sobre la placa de pruebas.
- Figura 7-12** Proyecto 20. Arpa de luz.
- Figura 7-13** Proyecto 21. Medidor VU.
- Figura 7-14** Esquema electrónico del Proyecto 21.
- Figura 7-15** Diseño del circuito del Proyecto 21 sobre la placa de pruebas.
-
- Figura 8-1** Esquema electrónico del Proyecto 22. *
- Figura 8-2** Diseño del circuito del Proyecto 22 sobre la placa de pruebas.
- Figura 8-3** La histéresis en los sistemas de control. *
- Figura 8-4** Proyecto 22. Termostato LCD.
- Figura 8-5** Proyecto 23. Ventilador controlado por ordenador.
- Figura 8-6** Esquema electrónico del Proyecto 23. *
- Figura 8-7** Diseño del circuito del Proyecto 23 sobre la placa de pruebas.
- Figura 8-8** Un puente H.
- Figura 8-9** Proyecto 24. Hipnotizador.
- Figura 8-10** Esquema electrónico del Proyecto 24. *
- Figura 8-11** Diseño del circuito del Proyecto 24 sobre la placa de pruebas.
- Figura 8-12** Espiral para el hipnotizador.
- Figura 8-13** Proyecto 25. Láser servo-controlado.
- Figura 8-14** Esquema electrónico del Proyecto 25. *
- Figura 8-15** Montaje del servo y del láser.

Figura 8-16 Escritura de la letra A con el láser.

Figura 8-17 Shield servo láser.

Figura 8-18 Lado inferior del shield servo láser.

Figura 9-1 Proyecto 26. Detector de mentiras. *

Figura 9-2 Esquema electrónico del Proyecto 26.

Figura 9-3 Diseño del circuito del Proyecto 26 sobre la placa de pruebas.

Figura 9-4 Proyecto 27. Cerradura magnética.

Figura 9-5 Esquema electrónico del Proyecto 27. *

Figura 9-6 Diseño del circuito del Proyecto 27 sobre la placa de pruebas.

Figura 9-7 Proyecto 28. Mando a distancia por infrarrojos. * (Ver nota pág. XX)

Figura 9-8 Esquema electrónico del Proyecto 28. *

Figura 9-9 Diseño del circuito del Proyecto 28 sobre la placa de pruebas.

Figura 9-10 Código de infrarrojos en el osciloscopio.

Figura 9-11 Proyecto 29. Reloj binario con Lilypad.

Figura 9-12 Esquema electrónico del Proyecto 29.

Figura 9-13 Primer plano de LED conectado a una resistencia.

Figura 9-14 Lado inferior de la placa Lilypad.

Figura 9-15 Proyecto 30. Temporizador de cuenta atrás.

Figura 9-16 Esquema electrónico del Proyecto 30. *

Figura 9-17 Placa Arduino alimentada con el LED encendido.

Figura 10-1 Ejemplo de esquema eléctrico. *

Figura 10-2 Ejemplo de diseño de placa de pruebas.

Figura 10-3 Símbolos electrónicos.

Figura 10-4 Código de colores para resistencias. *

Figura 10-5 Circuito de commutación básico de un transistor. *

Figura 10-6 Alicates de corte y alicates de punta fina.

Figura 10-7 Soldador y material de soldadura.

Figura 10-8 Polímetro.

Figura 10-9 Medición de corriente.

Figura 10-10 Osciloscopio.

* **Nota:** Las Figuras que aparecen con un * al final de la descripción han sido adaptadas o modificadas de algún modo para la versión en español por el revisor técnico.

Prefacio

ES UNA REALIDAD que el mundo tecnológico está liderado por el mundo de habla inglesa, y sobre todo por Estados Unidos. Esto es obvio. En sus universidades y centros de investigación se desarrolla lo más avanzado e innovador de la tecnología actual. Muchos de sus centros de posgrado son, sencillamente, de quitarse el sombrero. Y, en gran medida, lo son porque en ellos desarrollan su actividad muchos de los mejores profesionales del mundo en su campo.

El acceso a todos los textos y documentación que producen estos autores y estos centros de excelencia supone, no hace falta decirlo, una ventaja competitiva para todo el mundo que comparte esa misma lengua, el inglés. Y, por la misma razón, una desventaja para el resto de los idiomas.

Recientemente escuchaba en un vídeo a **David Cuartielles**, ingeniero español y uno de los fundadores del equipo **Arduino**, decir que, en su opinión, en temas de tecnología, España y Latinoamérica llevan un retraso de 5 años sobre el mundo anglosajón, por la falta de libros traducidos al español. Obviamente, es difícil cuantificar el “daño”, pero éste, sin duda, existe.

Desafortunadamente, eso hace que nos estemos perdiendo cosas importantes.

Por poner un ejemplo sencillo, pensemos en el mundo de la literatura. Imaginemos que no se hubiera traducido a **Balzac**, a **Stendhal**, a **Goethe**, **Dostoievski**, **Faulkner**, **Canetti**..., a **Roth** o a **Coetzee**. ¡A **Shakespeare**! o a tantos y tantos otros clásicos de la literatura. ¡Cuánto más pobres seríamos!

Pues eso, desgraciadamente, es lo que ocurre, entre otros, en el sector de la tecnología. ¡Cuántas obras importantes nos estamos perdiendo! Aunque no sean clásicos de la literatura, muchas de estas obras, cada una en su campo, también son “clásicos” u obras de referencia en su sector: en informática, en robótica, en electrónica. Obras que en muchas ocasiones van por la 8^a o la 10^a edición, y que aquí seguimos sin poder disfrutar y aprender de ellas porque, simplemente, no están traducidas a nuestro idioma.

Ser conscientes de esa carencia ha sido, sin duda, una de nuestras principales motivaciones al iniciar este apasionante viaje que supone poner en marcha un proyecto editorial. Y, para nosotros es apasionante por muchas razones: porque se trata de libros (da igual el formato, papel o digital), de información, de cultura, de diálogo entre autores y lectores e, incluso, a veces, de ayudar a despertar una inspiración o un interés en los más jóvenes.

Aunque nuestra posible aportación sólo pueda suponer apenas nada ante la magnitud de la tarea, el reto creemos que merece la pena: intentar ayudar a hacer que ese hueco entre lo que se publica cada año en inglés y lo que llega en español a nuestros alumnos, profesores, profesionales o aficionados sea un poco más pequeño. Ese es nuestro Reto y nuestro Compromiso.

Nuestro interés, nuestro foco, por tanto, será rastrear el mercado internacional en busca de obras que creemos no deberían faltar en ninguna biblioteca técnica. Afortunadamente, eso, en estos tiempos de la globalización y de Internet, es una labor relativamente sencilla. Ver qué libros de estudio y de referencia están utilizando en **Carnegie Mellon**, en **Stanford**, en **Berkeley**, o en el **MIT**, resulta hoy por hoy de lo más viable. Por eso las referencias son perfectamente conocidas.

La barrera del idioma no debería impedir que podamos aprender y conozcamos a autores como **D. Knuth**, como **W. Stallings**, como **Aho, Hopcroft, Silberschatz** o **Tanenbaum**, por sólo citar a algunos “clásicos” de la tecnología y de la informática. Porque eso, como hemos comentado, nos hace más pobres.

Thomas Woll, en su clásico libro sobre el mundo editorial, menciona que la aspiración última de cualquier proyecto editorial es conseguir Credibilidad. Y dice que ningún proyecto podrá funcionar sin lo que él denomina la C³: Compromiso, Coherencia y Credibilidad.

Casi con toda seguridad, el Compromiso y la Coherencia son dos atributos que pueden partir de uno mismo. Sin embargo, la Credibilidad, como la confianza, es algo que uno se gana con el tiempo; algo que a uno le confieren los demás. En este caso, ustedes, los lectores.

Ya hemos hablado de nuestro Compromiso. La Coherencia pretendemos conseguirla centrándonos en dos aspectos. Primero, focalizando nuestra atención en un nicho concreto: libros técnicos de referencia, normalmente procedentes del mundo anglosajón, que no han sido traducidos a nuestro idioma. Segundo, procurando planificar nuestras tareas para que las cosas que decimos que vamos a hacer, se hagan, en tiempo y en forma, como hemos dicho que las haríamos. Y eso incluye publicar un determinado número de libros al año, de estos temas concretos.

Y como menciona **Woll**, cuando se trabaja duro y de manera honesta en el Compromiso y en la Coherencia, la Credibilidad suele llegar con el tiempo.

Para este primer título, hemos apostado por el libro que tienen en sus manos: **30 Proyectos con Arduino**, del **Dr. Simon Monk**, sobre un tema que se está convirtiendo en un cierto fenómeno a nivel mundial: el mundo **Arduino**.

Como muchos de ustedes ya saben, **Arduino** es una tecnología sencilla y asequible que permite hacer muchas cosas en el mundo de la electrónica y la robótica. Es una magnífica herramienta de enseñanza en esos campos. Permite, tanto a profesores, alumnos y profesionales como a aficionados que no tienen por qué saber nada de tecnología para sus quehaceres, como artistas o diseñadores multimedia, crear dispositivos y entornos interactivos.

En el mundo de la enseñanza, sin duda, tiene también un papel de primera magnitud. Su enorme versatilidad, su facilidad de uso, su software abierto, el precio de los componentes y la gran comunidad de personas que ya trabajan con esta pequeña gran plataforma tecnológica la convierten en una herramienta básica para profesores y alumnos y para todos aquellos que se inician en el aprendizaje de la electrónica y la robótica.

Un libro es un diálogo abierto entre muchas personas, desde el autor o todos los que participamos en su elaboración, hasta ustedes, a quiénes va dirigido. Y lo es más, si cabe, cuando se trata de tecnología. Cambian rápidamente las versiones, surgen nuevos avances, etc.

Para esta primera edición en español hemos realizado una importante labor de revisión y, en parte, de adaptación a nuestro entorno y a nuestro idioma. Eso quiere decir que se han modificado algunas fotos, algunos gráficos y se han tenido en cuenta los comentarios aportados en la web, hasta el momento, por otros lectores de la versión inglesa. El autor, asimismo, tiene abierta también una página web (www.arduinoevilgenius.com) donde va colocando las actualizaciones y las cosas que mencionan los lectores que pueden corregirse.

El libro, por supuesto, no es perfecto. Ningún libro, en cuanto diálogo abierto, lo es. Sí, en cambio, hemos puesto todo nuestro empeño en que sea lo mejor posible.

Para finalizar, deseamos recordar de nuevo las palabras de **T. Woll**: Compromiso, Coherencia y Credibilidad. ¡En eso estamos!

¡Sean bienvenidos!

Madrid, abril de 2012

Agradecimientos

Me gustaría dar las gracias a mis hijos, Stephen y Mathew Monk, por su interés y el ánimo que me han infundado mientras escribía este libro, por sus útiles sugerencias y por las pruebas de campo que han realizado de los proyectos. Tampoco habría podido escribir este libro sin la paciencia y el apoyo de Linda.

Agradezco a Chris Fitzer el préstamo de su osciloscopio y el talante con el que se tomó... ¡que lo rompiera! También quiero dar las gracias a todos los "frikies" de Momote por su interés en el proyecto y por su apoyo.

Finalmente, quiero agradecer a Roger Stewart y a Joya Anthony, de McGraw-Hill, por su enorme apoyo y entusiasmo, y con los que ha resultado un placer trabajar.

Introducción

LAS PLACAS DE INTERFAZ ARDUINO proporcionan una tecnología de bajo coste y fácil de usar para crear proyectos. En la actualidad pueden construirse toda una nueva generación de proyectos que pueden controlarse desde un ordenador. En poco tiempo podrá disponer, entre otros, de un láser controlado mediante un servo y manejado por el ordenador, de modo que los más traviesos ¡podrán poner el mundo a sus pies!

Este libro le mostrará cómo conectar una placa Arduino a su ordenador, para programarla, y para que pueda conectarle todo tipo de componentes electrónicos para crear proyectos, incluyendo el láser controlado por ordenador, mencionado anteriormente, un ventilador controlado por USB, un arpa de luz, un registrador de temperaturas USB, un osciloscopio para sonidos, y mucho más.

Con cada proyecto se proporciona el esquema electrónico completo y detalles para su construcción. La mayoría se puede montar sin necesidad de soldar y sin herramientas especiales.

No obstante, los aficionados más avanzados tal vez deseen transferir los proyectos desde la placa de pruebas de inserción de componentes (**protoboard**) a una placa de circuito impreso definitiva, para lo que también se proporcionan las instrucciones adecuadas.

Entonces, ¿qué es Arduino?

Bueno, **Arduino** es una pequeña placa de microcontrolador con un puerto USB para conectar al ordenador y diversos zócalos de conexión que se pueden conectar mediante cableado a todo tipo de componentes electrónicos externos, como motores, relés, sensores de luz, diodos láser, altavoces, micrófonos, etc. Se puede alimentar mediante la conexión USB del ordenador o con una pila de 9 V. La placa se puede controlar directamente desde el ordenador o programarla con éste y posteriormente desconectarla para trabajar de forma autónoma.

Llegados a este punto, puede que algunos estén pensando en qué organización gubernamental "top secret" tienen que colarse para adquirir una de estas placas. Sin embargo, afortunadamente, no es necesario hacer nada especial para obtener uno de estos dispositivos. Ni siquiera tendrá que separarse de su buscador favorito o de su distribuidor de componentes electrónicos online. Puesto que **Arduino** es un diseño de hardware de código abierto, cualquier persona es libre de tomar los diseños y crear sus propios "clones" de **Arduino** y venderlos, por lo que el mercado de las placas es bastante competitivo. Una placa **Arduino** oficial cuesta alrededor de 23 €, y las llamadas "clon" suelen costar menos de 20 €.

El nombre **Arduino** está reservado por los creadores originales. Sin embargo, los diseños "clones" de Arduino con frecuencia llevan el nombre duino al final de su nombre, como por ejemplo, **Freeduino** o **DFRduino**.

El software para programar Arduino es fácil de usar e igualmente está disponible libremente para equipos **Windows**, **Mac** y **Linux** sin coste alguno.

Arduino

Aunque Arduino es un diseño de código abierto para una placa interfaz de microcontrolador, en realidad es más que eso, ya que abarca tanto las herramientas de desarrollo de software que son necesarias para programar la placa Arduino, como la propia placa. Existe una gran comunidad de aficionados al montaje, programación, electrónica e incluso de aficionados al arte dispuestos a compartir sus conocimientos y experiencia en Internet.

Para comenzar a utilizar Arduino, vaya primero a la página oficial de Arduino (www.arduino.cc) (en inglés) o a la página en español, www.arduino.cc/es y descargue el software para Mac, PC o LINUX. Podrá incluso comprar una placa Arduino haciendo clic en el botón **Comprar una placa Arduino** o

pasar algún tiempo con su buscador favorito o en algún sitio de subastas en línea para encontrar opciones más económicas. En el siguiente capítulo se proporcionan instrucciones paso a paso para la instalación del software sobre las tres plataformas.

De hecho, existen diferentes diseños de placas Arduino destinados a diferentes tipos de aplicaciones. Todas las placas se pueden programar desde el mismo software de desarrollo de Arduino y, en general, los programas que funcionan en una placa, funcionan en todas.

En este libro usamos indistintamente las placas Arduino **UNO** y Arduino **DueMilanove**, siendo cada una de ellas una actualización de las populares placas anteriores (la placa **Diecimila**). La idea es resaltar que todas son compatibles y que para la ejecución de los proyectos del libro pueden utilizarse indistintamente cualquiera de ellas. **DueMilanove** significa 2009 en italiano, el año de su lanzamiento. **UNO** es la última versión publicada. El nombre de las antiguas **Diecimila** (10.000 en italiano) hacía referencia a las primeras 10.000 placas que se habían fabricado. Las placas clones más compatibles, como las **Freeduino**, se han basado en los diseños originales (UNO, DueMilanove, Diecimila, etc.).

La mayoría de los proyectos de este libro funcionan tanto con las placas originales **UNO**, **DueMilanove** o **Diecimila**, como con sus clones de diseño, aparte de un proyecto que utiliza la **Arduino Lilypad**.

Cuando esté realizando proyectos con una Arduino, necesitará descargar programas en la placa utilizando un cable USB entre su ordenador y la Arduino. Ésta es una de las cosas más prácticas de la utilización de Arduino. Muchas placas microcontroladoras usan un hardware específico de programación para descargar los programas al microcontrolador. En el caso de Arduino, todo está incluido en la propia placa. Esto también tiene la ventaja de que puede utilizar la conexión USB para pasar datos en un sentido y en otro entre la placa Arduino y su equipo. Por ejemplo, se puede conectar un sensor de temperatura a la Arduino y hacer que su ordenador le diga la temperatura.

En las antiguas placas **Diecimila**, había un puente **jumper** justo debajo del conector USB. Con el puente colocado en los dos pines superiores, la placa recibirá la alimentación eléctrica de la conexión USB. Si se conecta entre los dos pines inferiores, la placa será alimentada por la fuente de alimentación externa conectada al conector de alimentación inferior. En las placas más recientes **UNO** y **DueMilanove**, no existe este puente y la alimentación cambia automáticamente desde el USB al conector de 9 V cuando introducimos éste.

La fuente de alimentación puede ser de cualquier tensión entre 7 y 12 voltios. Por lo tanto, una pequeña pila de 9 V funcionará perfectamente para las aplicaciones portátiles. Normalmente, mientras esté montando su proyecto, probablemente la alimente desde el USB por razones de comodidad. Cuando esté listo para cortar el "cordón umbilical" (desconectar el cable USB), seguramente desee alimentar la placa de forma independiente. Esto puede hacerse con un adaptador o fuente de alimentación externa, o simplemente con una pila de 9 V conectada a un cable adaptado al conector de alimentación.

En los bordes de la placa hay dos filas de conectores. La fila de la parte superior son en su mayoría pines digitales (on/off), aunque cualquier pin marcado con **PWM** puede ser utilizado también como salida analógica. La fila inferior de conectores dispone en su lado izquierdo de unos pines muy útiles con diferentes tensiones de alimentación, y entradas analógicas en el lado derecho.

Los conectores están organizados de este modo para que las llamadas placas shields puedan enchufarse a la placa base de forma superpuesta. Pueden adquirirse shields ya montadas para muchos propósitos diferentes, incluyendo:

- Conexión a redes Ethernet
- Displays LCD y pantallas táctiles
- XBee (comunicaciones de datos inalámbricas)
- Sonido
- Control de motores
- Seguimiento por GPS
- Y muchas más

También puede usar placas **shields básicas** para crear sus propios diseños de shields específicos. Nosotros utilizaremos estas **Protoshields** en algunos de nuestros proyectos. Las shields suelen tener conectores que permiten que las placas se puedan conectar unas encima de otras. Por ejemplo, un diseño podría tener tres capas: una placa Arduino en la parte inferior, una **shield GPS** conectada encima, y una **shield display LCD** en la parte superior.

Los proyectos

Los proyectos de este libro son bastante diversos. Comenzamos con algunos ejemplos simples con LEDs estándar y con LEDs ultra brillantes Luxeon.

En el Capítulo 5 veremos diversos proyectos de sensores para registrar la temperatura y medir la luz y la presión. La conexión USB a la placa Arduino hace posible que podamos tomar las lecturas del sensor de estos proyectos y se las pasemos al ordenador, donde se pueden importar a una hoja de cálculo y obtener gráficos de los datos.

A continuación, nos ocupamos de varios proyectos que utilizan diversos tipos de **displays** de diferentes tecnologías, incluyendo un panel alfanumérico **LCD** de mensajes (de nuevo utilizando el USB para trasmisitir los mensajes desde el ordenador), así como LEDs multicolores y **displays** de siete segmentos.

El Capítulo 7 contiene cuatro proyectos que utilizan el sonido, así como un simple osciloscopio. Tenemos un proyecto que hace sonar melodías por un altavoz, y construimos un arpa de luz que cambia el tono y el volumen del sonido cuando se desplaza la mano sobre los sensores de luz. Esto produce un efecto bastante parecido al famoso sintetizador **Theremin**. El proyecto final de este capítulo utiliza un micrófono como entrada de sonido. Se trata de un **vúmetro** o medidor de VU, que muestra la intensidad del sonido en un **display**.

Los últimos capítulos contienen una mezcla de proyectos. Entre otros, un original reloj binario que utiliza la placa **Arduino Lilypad** y que indica la hora de una extraña forma binaria que sólo podrán leer las mentes más brillantes, un detector de mentiras, un disco giratorio para hipnotizar controlado por motor y, por supuesto, el láser servo-guiado controlado

desde el ordenador.

La mayoría de los proyectos de este libro se pueden construir sin necesidad de soldar; en su lugar, utilizamos una placa de pruebas **protoboard**. Las placas protoboard consisten en un bloque de plástico con agujeros y láminas metálicas de conexión por detrás. Los componentes electrónicos se insertan en los orificios de la parte superior. Además, no son caras. En el Apéndice hemos incluido una placa de pruebas que sirve para los propósitos de este libro. Sin embargo, si desea montar sus diseños para que resulten permanentes, el libro le muestra también cómo hacerlo, utilizando una placa de circuito impreso para prototipos.

En el apéndice se han incluido fuentes de suministro de los componentes utilizados, junto con los nº de referencia de catálogo de algunos proveedores especialmente útiles. Lo único que necesitará además de estos componentes son una placa Arduino, un ordenador, algunos cables y una placa de pruebas. El software para todos los proyectos puede descargarlo de la siguiente dirección web del autor: www.arduinoevilgenius.com.

Sin más preámbulos

Las personas a quienes les pierde la curiosidad no se suelen distinguir precisamente por su paciencia, por lo que en el próximo capítulo les mostraremos cómo empezar con Arduino lo antes posible. Este capítulo contiene las instrucciones para la instalación del software y para programar la placa Arduino, incluida la descarga del software de los proyectos, por lo que tendrá que leerlo antes de poder embarcarse en los mismos.

En el Capítulo 2 echaremos un vistazo a algunos de los conocimientos esenciales que le ayudarán a crear los proyectos que se describen en este libro y para que pueda continuar desarrollando los suyos propios. La mayoría de la teoría que contiene el libro está contenida en este capítulo, así es que si es del tipo de personas que sólo desea realizar los proyectos y, posteriormente, averiguar cómo funcionan, quizás prefiera, tras leer el Capítulo 1, elegir un proyecto y comenzar su montaje. Y si se queda atascado siempre podrá utilizar el índice o leer algunos de los capítulos anteriores.

CAPÍTULO 1

Arranque rápido

ESTE ES UN CAPÍTULO para las mentes curiosas más impacientes. Ya tiene en sus manos su nueva placa Arduino y está ansioso por ponerla a funcionar.

Por tanto, sin más dilación...

Puesta en marcha

Cuando se compra una placa **Arduino UNO** o **Duemilanove**, por lo general suelen venir con un programa **Blink** de muestra preinstalado que hará parpadear el pequeño LED que trae integrado. La Figura 1-1 muestra una placa Arduino UNO con el LED encendido.

El LED (diodo emisor de luz) marcado con una L está conectado a uno de los zócalos de entrada-salida digitales de la placa. En concreto, está conectado al **pin digital 13**, lo cual limita que éste sólo pueda utilizarse como salida, pero como el LED sólo utiliza una pequeña cantidad de corriente, todavía podemos conectar otras cosas en ese conector.

Todo lo que necesita hacer para que su Arduino funcione es proporcionarle la alimentación. La manera más fácil de alimentarla es conectarla al puerto **USB** (Bus Serie Universal) del ordenador. Necesitará un cable USB de conexión de tipo A a tipo B. Este es el cable que normalmente se utiliza para conectar un ordenador a una impresora.

Si está utilizando la placa antigua de Arduino, la Diecimila, asegúrese de que el puente (jumper) de alimentación está en la posición **USB**. El puente debe conectar los dos pines superiores para que la

placa reciba la alimentación de la conexión USB. Las nuevas placas Arduino UNO o Duemilanove no tienen este puente y seleccionan la fuente de alimentación automáticamente.

Si todo funciona bien, el LED de la placa debe parpadear una vez cada dos segundos. La razón por la cual las nuevas placas Arduino tienen este programa **Blink** (parpadeo) ya instalado es para verificar que la placa funciona. Si la placa no comienza a parpadear cuando se conecta, compruebe la posición del puente de alimentación (si lo tiene) y pruebe con otra entrada USB, si es posible en otro ordenador diferente, ya que algunas conexiones USB suministran más corriente que otras. Además, si se hace clic en el botón **Reset** (reinicio) el LED debería parpadear momentáneamente. Si al pulsar **Reset** el LED no parpadea, puede ser también que la placa no tenga preinstalado el programa **Blink**: no se desespere, ya que, en cualquier caso, una vez que esté todo preparado, vamos a instalar y modificar ese programa en nuestro primer proyecto.

Instalación del software

Ahora que ya tenemos la placa Arduino funcionando, vamos a instalar el software para que podamos modificar el programa **Blink** (parpadeo) y enviarlo a la placa. El procedimiento exacto depende del sistema operativo que esté utilizando en su ordenador. Pero el principio básico es el mismo para todos.

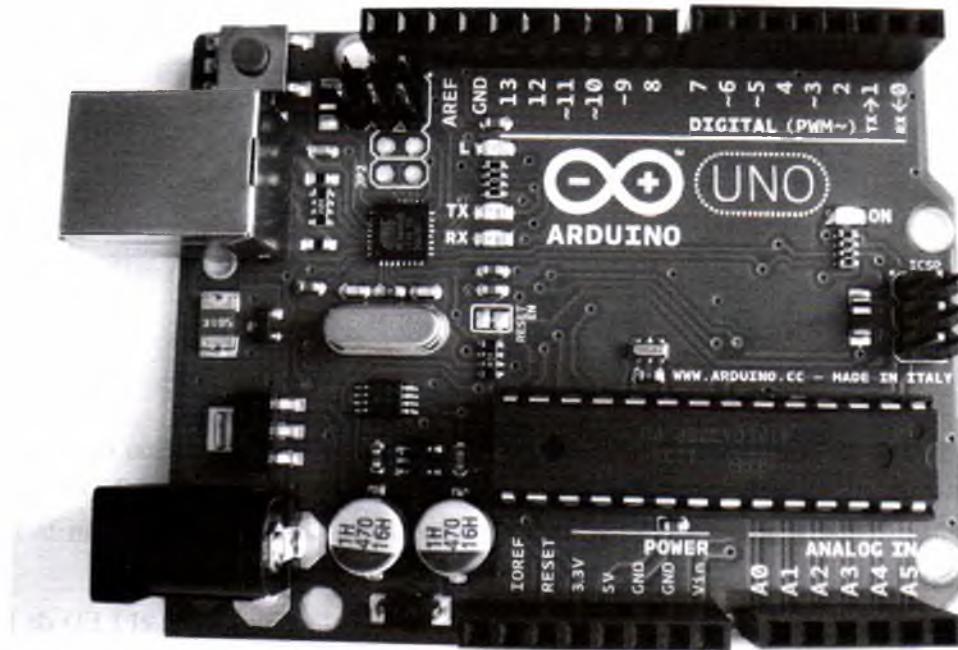


Figura 1-1 Placa Arduino UNO alimentada y con el LED encendido.

Instale el controlador USB que permite al ordenador comunicarse con el puerto USB de la placa Arduino. Este lo utiliza para la programación y para enviar mensajes.

Instale el entorno de desarrollo Arduino, que es el programa que se ejecuta en el ordenador y que permite crear los programas y descargarlos en la placa. La página web de Arduino, www.arduino.cc/es para la versión en español y www.arduino.cc (inglés) contiene la última versión del software.

Instalación en Windows

En la página web de Arduino (www.arduino.cc/es), vaya a la pestaña **Descarga** y seleccione la descarga para Windows. Esto iniciará la descarga del **archivo zip** que contiene el software de Arduino, tal como se muestra en la Figura 1-2.

El software de Arduino no distingue entre las distintas versiones de Windows. La descarga debe funcionar para todas las versiones de Windows, desde XP en adelante. Las instrucciones que mostramos son para Windows XP.

Seleccione la opción **Guardar** del diálogo, y guarde el archivo zip en su equipo. La carpeta contenida en el archivo Zip se convertirá en su directorio principal Arduino, así que ahora puede descomprimir (unzip) el archivo en **C:\Archivos de programa\Arduino**.

En Windows XP puede hacer esto haciendo clic con el botón secundario en el archivo **zip** para mostrar el menú de la Figura 1-3 y seleccionar la opción **Extraer todo**. Esto abrirá el **Asistente de Extracción**, como se muestra en la Figura 1-4.

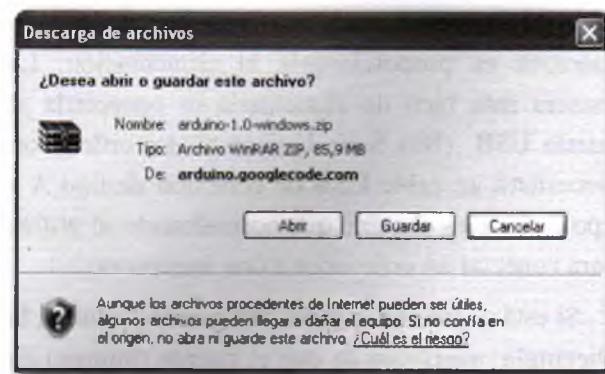


Figura 1-2 Descarga el Software de Arduino para Windows

Haga clic en **Siguiente** y, a continuación, modifique la carpeta para extraer los archivos en: **C:\Archivos de programa\Arduino** como se muestra en la Figura 1-5. A continuación, haga clic otra vez en **Siguiente**.

Esto creará un nuevo directorio para esta versión de Arduino (en este caso, la Arduino 1.0) en la carpeta **C:\Archivos de programa\Arduino-1.0**. Esto le permite tener varias versiones de Arduino instaladas al mismo tiempo, cada una en su propia carpeta. Las actualizaciones de Arduino son bastante poco frecuentes e históricamente siempre se ha mantenido la compatibilidad con las versiones anteriores del software. Así que, a menos que exista una nueva característica del software que desea utilizar, o si está teniendo problemas, no es en absoluto esencial mantenerse al día con la versión más reciente.

Ahora que ya tenemos la carpeta Arduino en el lugar correcto, necesitamos instalar los controlado-

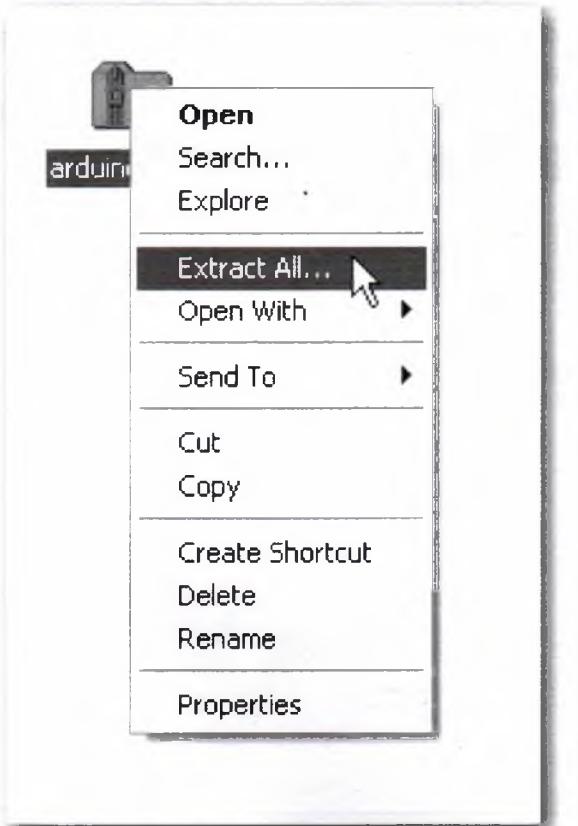


Figura 1-3 Extracción del software de Arduino en Windows.



Figura 1-4 La opción de menú Extraer todo en Windows.



Figura 1-5 Establecer el directorio para la extracción.

res USB. Si está usando una placa **Arduino UNO**, solo necesita informar a Windows sobre el nuevo hardware. Así es que conecte la placa Arduino y espere hasta que Windows comience el proceso de instalación del driver. Despues de un tiempo el proceso fallará y Windows anunciará que no se puede instalar el driver. Luego pulse **Inicio/Panel de control/Sistema/Hardware**, y abra el **Administrador de dispositivos** de Windows.

Busque **Otros dispositivos/USB device**. Haga click con el botón derecho sobre **Arduino UNO**



Figura 1-6 Asistente de Nuevo Hardware Encontrado de Windows.

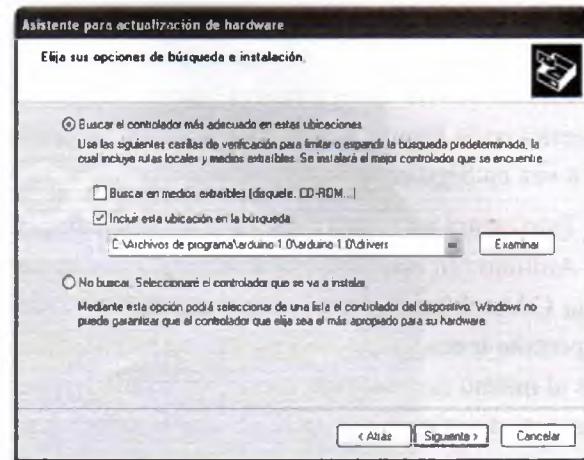


Figura 1-7 Establecer la ubicación de los controladores USB.

(Com xx) y elija **actualizar controlador**. Elija **Instalar desde una lista específica** y le saldrá el directorio donde está instalado Arduino, que debe ser algo parecido a **C:\Archivos de programa\arduino-1.0\arduino-1.0\drivers**. Cuando esté en la carpeta **drivers** seleccione el archivo **“ArduinoUNO.inf”**, con lo que Windows finalizará la instalación del driver.

En el caso de las placas más antiguas, siga las instrucciones específicas para cada modelo que se indican en www.arduino.cc/es.

Observe que no hay un acceso directo creado para el programa **Arduino**, por lo que, si lo desea, puede seleccionar el ícono de programa **Arduino**, hacer clic con el botón secundario y crear un acceso directo que luego puede arrastrar a su escritorio.



Figura 1-8 Inicio del software Arduino desde Windows.

Las dos secciones siguientes describen este mismo procedimiento para la instalación en equipos Mac y Linux, por lo que, si es usuario de Windows, puede omitir estas secciones.

Instalación en Mac OS X

El proceso para instalar el software Arduino en Mac es mucho más fácil que en el PC.

Como antes, el primer paso es descargar el archivo. En el caso del Mac, se trata de un archivo de imagen de disco. Una vez descargado, montará la imagen del disco y abrirá una ventana **Finder**, como se muestra en la Figura 1-9. La propia aplicación **Arduino** se instala de la forma habitual de Mac, arrastrándola desde la imagen de disco a la carpeta **Aplicaciones**. La imagen de disco también contiene dos paquetes de instalación de los controladores USB (consulte la Figura 1-10). Asegúrese de seleccionar el paquete para la arquitectura de su sistema. A menos que esté usando un Mac construido antes de marzo de 2006, tendrá que usar la versión **Intel** en lugar de la versión **PPC**.

Cuando ejecuta el instalador, puede simplemente hacer clic en **Continuar** hasta llegar a la pantalla **Seleccionar disco**, donde debe seleccionar el disco duro antes de hacer clic en **Continuar**. Como el software instala una extensión de kernel, le pedirá



Figura 1-9 Instalación del software Arduino en Mac OS X.

que introduzca su contraseña antes de completar la instalación.

Ahora puede buscar e iniciar el software **Arduino** en la carpeta **Aplicaciones**. Como se va a utilizar con frecuencia, si lo desea, puede hacer clic con el botón secundario en su ícono en el dock y configurarlo para **Guardar en Dock**.

Ahora puede pasar por alto la siguiente subsección, que es para la instalación de **LINUX**.

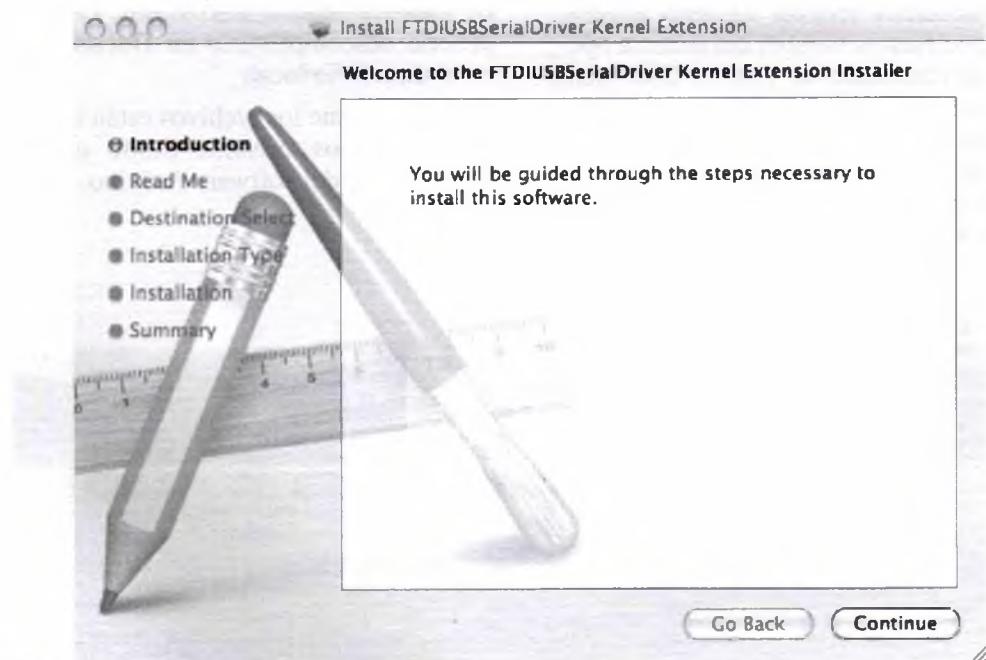


Figura 1-10 Instalación de los controladores USB en Mac OS X.

Instalación en LINUX

Hay muchas versiones diferentes de LINUX, por lo que, para obtener la información más reciente, consulte la página web de Arduino. Sin embargo, para la mayoría de las versiones de LINUX, la instalación es muy sencilla. Probablemente LINUX ya tendrá instalados los controladores USB, las bibliotecas AVR-GCC y el entorno Java que necesita el software Arduino.

Así es que, si tiene suerte, todo lo que necesita hacer es descargar el **archivo TGZ** para el software Arduino desde la página web de Arduino <http://www.arduino.cc/es/>, extraerlo, y ese será su directorio de trabajo Arduino.

Si, por otro lado, no tiene tanta suerte, como usuario de LINUX probablemente ya estará habituado a buscar apoyo de la comunidad de LINUX para configurar el sistema. Los pre-requisitos que necesitará instalar son **Java runtime 5** o posterior y las últimas bibliotecas **AVR-GCC**.

Introduciendo en Google la frase "Installing Arduino on SUSE LINUX" ("Instalar arduino en SUSE LINUX") o cualquiera que sea su versión de LINUX encontrará, sin duda, mucho material de ayuda.

Configuración del entorno Arduino

Sea cual fuere el tipo de ordenador que utiliza, ya debe tener el software Arduino instalado en el mismo. Ahora debemos realizar algunos ajustes. Tenemos que especificar el nombre del sistema operativo que está conectado al puerto USB para comunicarse con la placa Arduino, y tenemos que especificar el tipo de placa que estamos utilizando. Pero, primero, es necesario conectar Arduino al equipo mediante el **puerto USB** o no podrá seleccionar el **puerto serie**.

El puerto serie se establece desde el menú **Herramientas**, como se muestra en la Figura 1-11 para Mac y en la Figura 1-12 para Windows; la lista de puertos para LINUX es similar a la de Mac.

Si utiliza muchos dispositivos USB o Bluetooth con Mac, es probable que tenga bastantes opciones en esta lista. Seleccione el elemento de la lista que comienza con **dev/tty.usbserial**.

En Windows, el puerto serie se puede configurar a **COM3**.

Desde el menú **Tools**, podemos seleccionar la placa que vamos a utilizar, como se muestra en la Figura 1-13. Probablemente será la placa **UNO** pero si estuviera utilizando alguna de las placas más antiguas, seleccione la opción correspondiente.

Descarga del software del proyecto

El software de los **sketches** de este libro se puede descargar de la página web del libro. La descarga es menor de un megabyte, por lo que tiene sentido descargar el software para todos los proyectos, incluso si tiene intención de utilizar sólo algunos de ellos. Para descargarlos, vaya a la página web "www.arduinoevilgenius.com" (en inglés) y haga clic en **Downloads** en la parte superior de la pantalla.

Haga clic en el enlace **evil_genius.zip** para descargar un archivo Zip con todos los proyectos. Si está utilizando Windows, descomprimir el archivo en **Mis Documentos\Arduino**. En Mac o Linux, debería descomprimirlo en **Documentos/Arduino** en el directorio local.

Una vez que los archivos están instalados, podrá acceder a los mismos desde el menú **File | Sketchbook** del software Arduino.

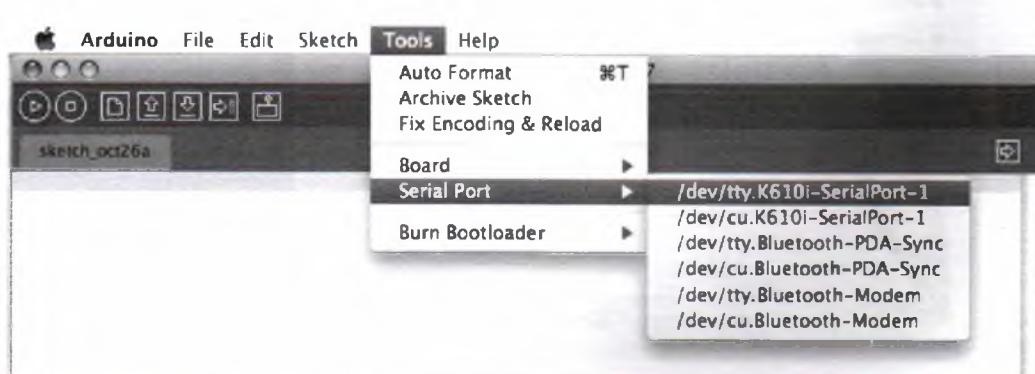


Figura 1-11 Configuración del puerto serie del Mac.

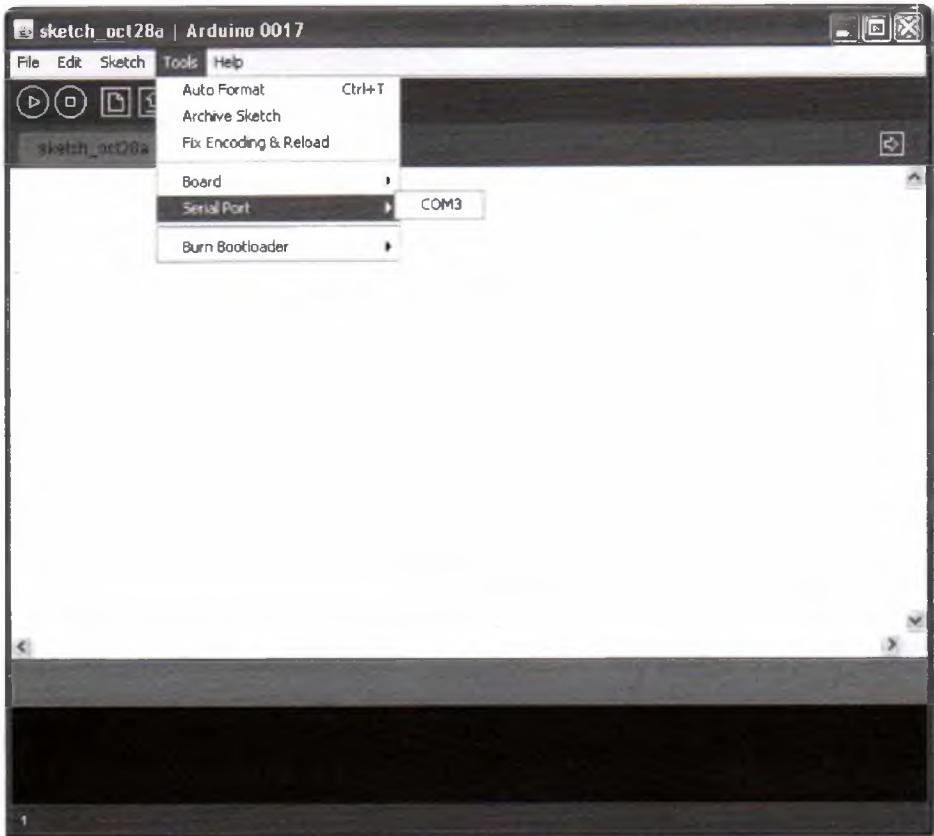


Figura 1-12 Selección de puerto en Windows con placas Duemilanove y versión de software 17 o 19.

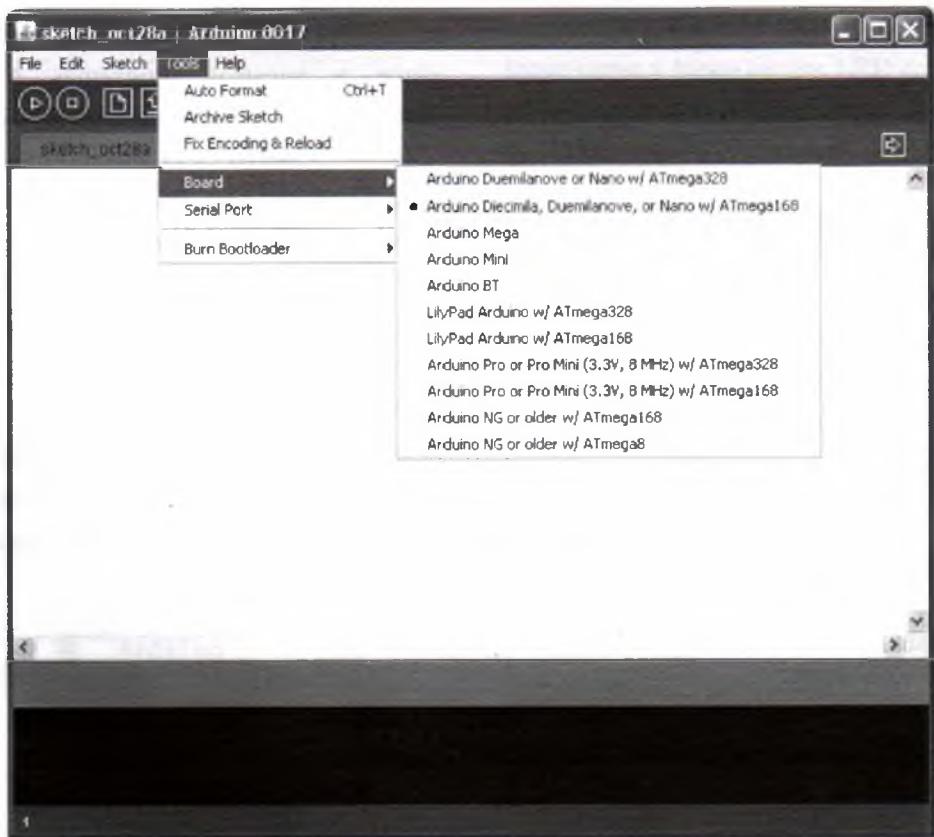


Figura 1-13 Selección de placa en Windows para placas Duemilanove o anteriores.

Proyecto 1

LED intermitente

Una vez comprobado que hemos instalado correctamente el software, ¡por fin podemos empezar con nuestro primer proyecto! En realidad, no es tan emocionante, pero tenemos que empezar por algún sitio, y con éste nos aseguramos que tenemos todo configurado correctamente y listo para utilizar nuestra placa Arduino.

Vamos a modificar el esquema **Blink** de ejemplo que viene con Arduino. Vamos a aumentar la frecuencia de intermitencia y, a continuación, instalar el **sketch** (programa) modificado en nuestra placa Arduino. En vez de parpadear lentamente, nuestra placa hará parpadear el LED rápidamente. Luego, adelantaremos un paso más utilizando un LED y resistencia externos de mayor tamaño que el que viene incorporado en la placa.

Software

En primer lugar, tenemos que cargar el **sketch** **Blink** en el software Arduino. El **sketch** **Blink** se

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
D1 LED rojo de 5 mm	23
R1 Resistencia de 270 Ω 0,5W	6
<ul style="list-style-type: none"> ■ De hecho, pueden servir prácticamente cualquier LED y resistencia de 270 Ω que haya disponibles. ■ No hacen falta más herramientas que un par de alicates o cortacables. ■ El número que aparece en la columna de la derecha en Apéndice se refiere al listado de componentes del Apéndice, que enumera las referencias de pieza de varios distribuidores. 	

incluye como ejemplo al instalar el entorno Arduino. Así que podemos cargarlo usando el menú **File**, como se muestra en la Figura 1-14.

La mayor parte del texto de este **sketch** es en forma de comentarios. Los comentarios no son realmente parte del programa pero explican lo que está haciendo el programa a cualquiera que lo lea.

Los comentarios pueden ser comentarios de una sola línea que comienzan tras // y continúan hasta el final de la línea, o pueden ser comentarios de varias líneas que comienzan con /* y finalizan algunas líneas más abajo con */.

Si se eliminaran todos los comentarios de un **sketch**, éste seguiría funcionando exactamente del mismo modo, pero la utilización de comentarios permite explicar qué es lo que hace el programa en cada momento.

Antes de empezar, son necesarios algunos comentarios sobre el vocabulario. La comunidad Arduino utiliza la palabra **sketch** en lugar de "programa", así que, de ahora en adelante, voy a referirme a nuestros programas Arduino como **sketches**. En ocasiones, puede que lo llame "código". **Código** es el término empleado por los programadores para referirse a una sección de un programa o incluso como término genérico de lo que se escribe cuando se crea un programa. Por eso, alguien podría decir: "*He escrito un programa para hacer eso*", o podría decir: "*He escrito un código para hacer eso*".

Para modificar la velocidad a la que parpadea el LED, tenemos que cambiar el valor del retardo (**delay**), de manera que en los dos lugares del sketch donde tenemos:

```
delay(1000);
```

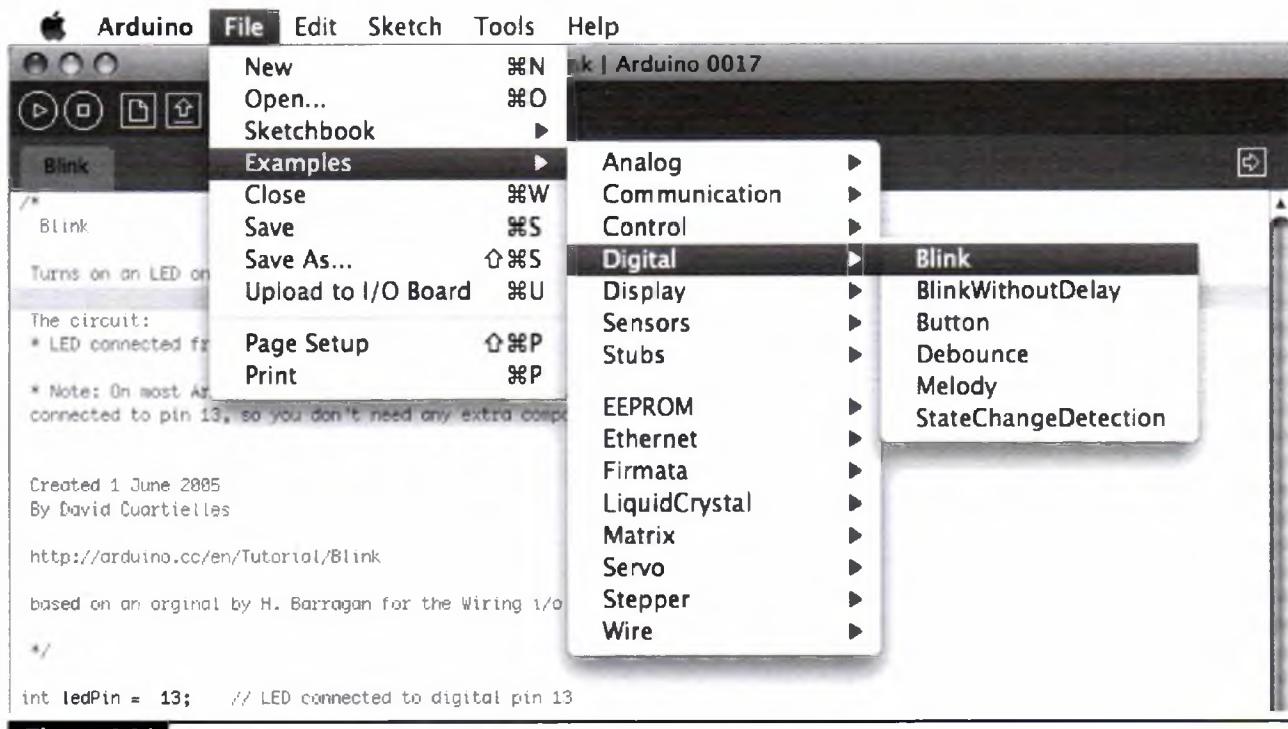


Figura 1-14 Carga del esquema Blink de ejemplo.

cambiar el valor entre paréntesis a 200 para que aparezca:

```
delay(200);
```

Esto cambiará el retardo (**delay**) entre encender y apagar el LED desde 1000 milisegundos (1 segundo) a 200 milisegundos (1/5 de un segundo). En el Capítulo 3 exploraremos este **sketch** con más detalle, pero, por ahora, vamos a cambiar el retardo y a descargar el nuevo sketch en la placa Arduino. Con la placa conectada a su equipo, haga clic en el

botón **Upload** (cargar) en Arduino. Esto se muestra en la Figura 1-15. Si todo es correcto, habrá una breve pausa y luego los dos LED de color rojo de la placa comenzarán a parpadear rápidamente una vez que el **sketch** se haya cargado en la placa. Esto debe tardar de 5 a 10 segundos.

Si esto no ocurre, compruebe los parámetros **puerto serie** y **tipo de placa** como se ha descrito en las secciones anteriores.

Cuando se haya instalado el **sketch** completo, la placa se reiniciará automáticamente y, si todo ha funcionado, verá el LED del puerto digital 13 parpadear mucho más rápidamente que antes.



Figura 1-15 Carga del sketch en la placa Arduino.

Hardware

Por el momento, esto no parece realmente electrónica, ya que el hardware va todo incluido en la placa Arduino. En esta sección, vamos a agregar un LED externo a la placa.

Los LED no se pueden conectar directamente a la alimentación sin más; deben tener conectado una resistencia que limite la corriente. Ambos componentes están disponibles en cualquier distribuidor de componentes electrónicos. Los códigos de pedido de algunos componentes para diversos proveedores se detallan en el Apéndice.

Los conectores de la placa Arduino están diseñados para que se le conecten directamente encima las llamadas placas **shield**. Sin embargo, para hacer montajes de prueba no definitivos también se pueden insertar directamente en los zócalos los cables o las patillas de contacto de los componentes electrónicos.

La Figura 1-16 muestra el esquema eléctrico de un LED conectado a la placa Arduino.

Este tipo de diagrama o esquema eléctrico utiliza símbolos especiales para representar los componentes electrónicos. Como vemos, el del LED se parece bastante a una flecha, lo que indica que los diodos emisores de luz (LED), al igual que el resto de los diodos, sólo permiten que la corriente

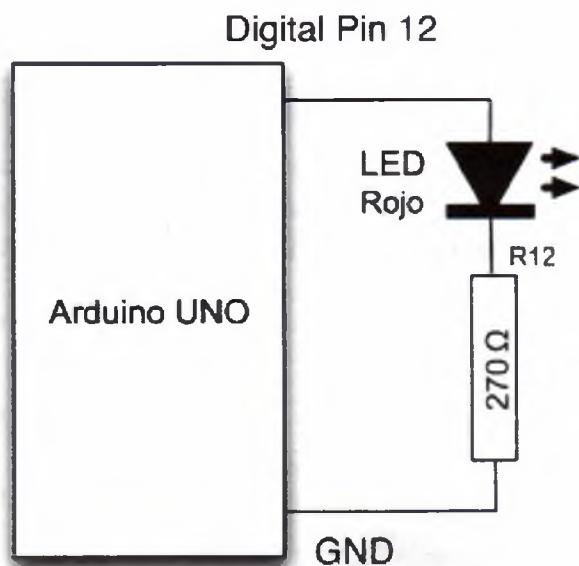


Figura 1-16 Esquema eléctrico de un LED conectado a la placa Arduino.

fluya en una dirección. Las flechas pequeñas que aparecen al lado del símbolo de diodo LED indican que emite luz.

Las resistencias se representan como un rectángulo. Las resistencias se dibujan también a menudo como una línea en zigzag. El resto de las líneas del diagrama representan las conexiones eléctricas entre los componentes. Estas conexiones pueden ser trozos de cable o pistas conductoras en una placa de circuito impreso. En este caso, éstas serán los terminales o patitas de los componentes.

Podemos conectar los componentes directamente a los zócalos de Arduino entre el pin digital **12** y el pin **GND**, pero primero tenemos que conectar una de las patillas del LED a la otra patilla de la resistencia.

No importa cuál de las patillas de la resistencia se conecta al LED; sin embargo, el LED debe conectarse de modo correcto. El LED tendrá una pata ligeramente más larga que la otra. La pata más larga es la que debe conectarse al **pin digital 12** y la más corta se conecta a la resistencia. En el caso de los LEDs y otros componentes se utiliza la convención de que el terminal más largo es el positivo.

Para conectar la resistencia al terminal o pata más corta del LED, separar con cuidado las dos patillas y liar la más corta alrededor de una de las patas de la resistencia, como se muestra en la Figura 1-17.

A continuación, introducir la pata más larga del LED (+) en el **pin digital 12** y la patilla libre de la

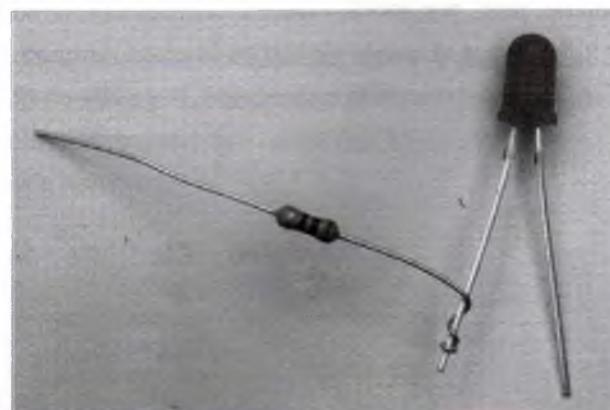


Figura 1-17 Un LED conectado en serie con una resistencia.

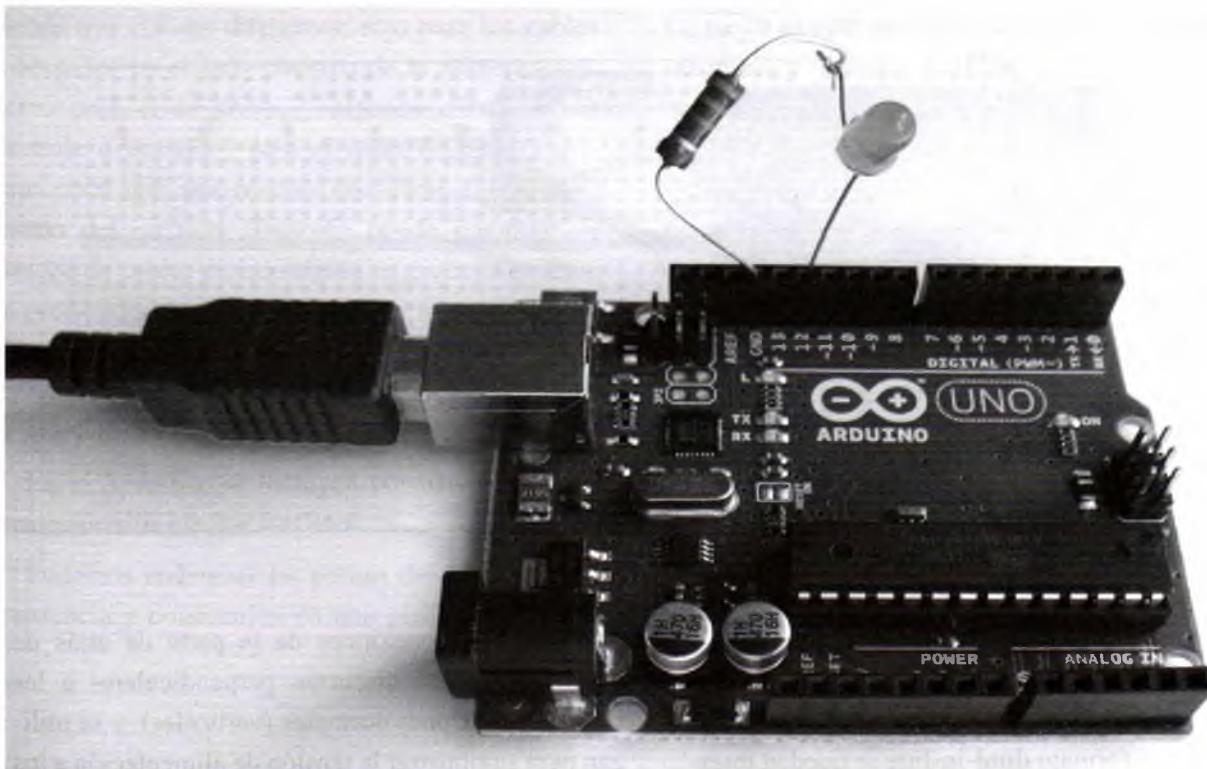


Figura 1-18 El LED conectado directamente a la placa Arduino.

resistencia en uno de los dos zócalos marcados **GND**. Esto se muestra en la Figura 1-18. A veces ayuda doblar ligeramente el extremo del conductor de manera que entre más firmemente en los zócalos.

Ahora podemos modificar nuestro **sketch** para utilizar el LED externo que acabamos de conectar. Todo lo que tenemos que hacer es cambiar el **sketch** para que utilice para el LED el **pin digital 12** en lugar del 13. Para ello, debemos cambiar la línea:

```
int ledPin = 13;  
// LED conectado al pin digital 13
```

para que indique:

```
int ledPin = 12;  
// LED conectado al pin digital 12
```

Ahora cargue el sketch haciendo clic en el botón **Upload To IO Board** (cargar en tarjeta de E/S) de la misma forma que lo hizo cuando modificó la velocidad de los destellos.

Placa de pruebas de inserción de componentes (**protoboard**)

Empalmar componentes retorciendo entre sí los terminales de los mismos sólo es práctico cuando se trata de conectar un único LED. Las **placas o módulos de prueba**, también llamadas placas de inserción de componentes o **protoboard**, nos permiten construir complicados circuitos sin necesidad de soldaduras. De hecho, es una buena idea construir primero todos los circuitos en una placa de pruebas hasta conseguir que funcione como queremos, y, posteriormente, pasarlo a una placa de circuito impreso soldada una vez que el montaje es definitivo.

Un módulo **protoboard** está formado por un bloque de plástico con agujeros en la parte superior y láminas metálicas de conexión en la parte inferior. Los componentes electrónicos se insertan a través de los orificios de la parte superior.

Por debajo de los orificios de la placa de pruebas hay tiras de conectores, de forma que todos los agu-

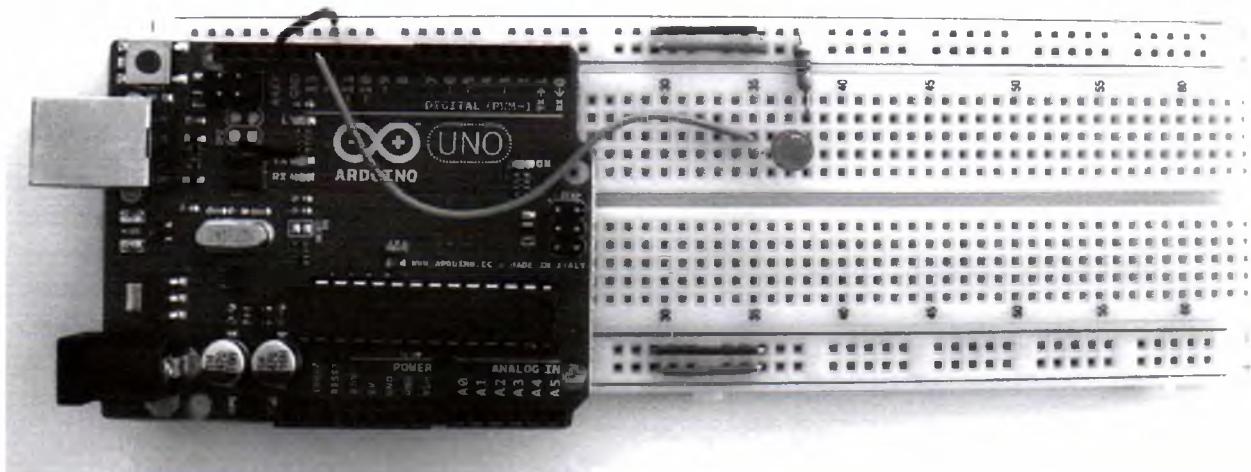


Figura 1-19 Proyecto 1 en la placa de pruebas.

jeros de una misma tira están conectados entre sí.

Las tiras están separadas por un espacio que discurre entre ellas para que los circuitos integrados que vienen en formato **dual-in-line** se puedan insertar sin que los contactos que se encuentren en la misma fila se cortocircuiteen entre sí.

Podemos montar este proyecto en una placa de pruebas en lugar de empalmar los componentes mediante retorcimiento de sus patillas. En la Figura 1-19 se puede ver la placa **protoboard** con los componentes insertados. La Figura 1-20 permite ver con claridad el conexionado interno de la placa de pruebas y la posición de los componentes.

Notará que en los bordes de la placa de pruebas (parte superior e inferior), hay dos largas tiras hori-

zontales. Las conexiones de la parte de atrás de estas tiras largas discurren perpendiculares a las tiras de conexiones normales (verticales) y se utilizan para suministrar la tensión de alimentación a los componentes de la placa de pruebas. Normalmente, hay una para tierra (**0 V o GND**) y una para el suministro de voltaje positivo (generalmente **5 V**). Como las tiras horizontales para la alimentación sólo llegan hasta el centro de la placa, hemos puesto unos cables de unión con la otra mitad para que la alimentación también llegue a la otra mitad de la placa.

Además de la placa de pruebas, necesitará algunos cables de hilo rígido y unos pelacables o alicates para cortar y eliminar el material aislante de los extremos del cable. Es una buena idea tener al

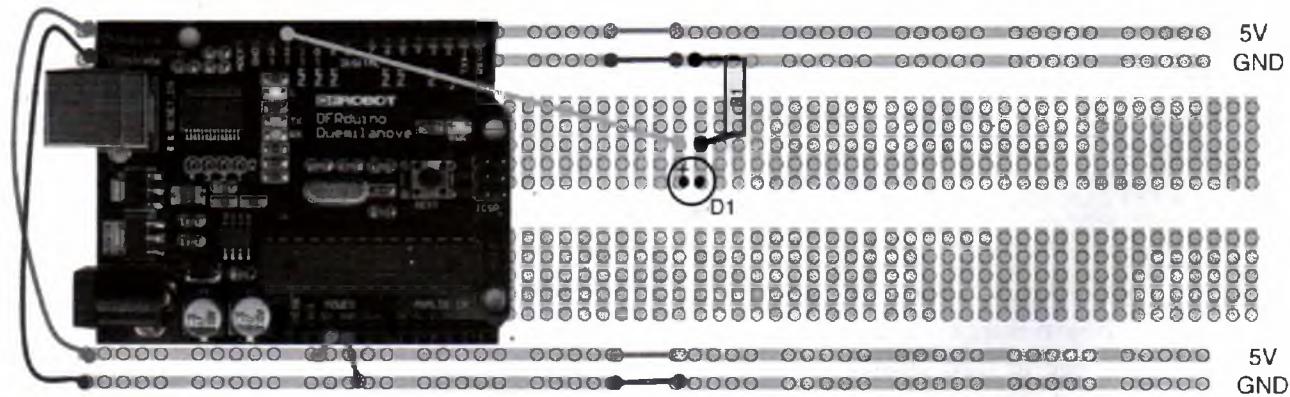


Figura 1-20 Diseño de la placa de pruebas del Proyecto 1.

menos tres colores diferentes: rojo para los cables conectados en el lado positivo de la alimentación, negro para el negativo, y algunos otros colores (naranja o amarillo) para el resto de conexiones. Esto hace que sea mucho más fácil entender el diseño del circuito. También puede comprar un estuche de cables de hilo rígido en diversos colores ya preparados. Tenga en cuenta que no es aconsejable utilizar cable flexible, ya que tenderá a abrirse cuando intente meterlos en los agujeros de la placa de pruebas.

En el Apéndice se incluyen posibles comercios donde adquirir estos materiales.

Podemos enderezar las patitas del LED y de la resistencia y conectarlos en una placa de pruebas.

Lo mejor es usar una placa de pruebas de un tamaño razonable y ponerle encima la placa Arduino. Si no quiere que esta conexión sea permanente utilice un poco de masilla adhesiva para pegarla. Sin embargo, puede que le resulte más fácil dedicar una placa Arduino para que sea su placa de diseño y quiera dejarla conectada permanentemente a la placa de pruebas.

Resumen

Hemos creado nuestro primer proyecto, aunque sea muy simple. En el siguiente capítulo obtendremos un poco más de información sobre Arduino antes de pasar a otros interesantes proyectos.

CAPÍTULO 2

Un recorrido por Arduino

EN ESTE CAPÍTULO, nos fijaremos en el hardware de la placa Arduino y del microcontrolador que contiene en su corazón. De hecho, la placa básicamente sólo presta apoyo al microcontrolador, extendiendo los pines de los conectores para que pueda conectar hardware en ellos y proporcionar un enlace USB para descargar *sketches*, etc.

También aprenderemos algunas cosas sobre el lenguaje C usado para programar la placa Arduino, algo sobre lo que profundizaremos en capítulos posteriores cuando comencemos algunos proyectos de trabajo práctico.

Aunque este capítulo en algunos momentos pueda resultar un poco teórico, le ayudará a entender el funcionamiento de los proyectos. Sin embargo, si lo que prefiere es avanzar con sus proyectos, puede sólo echarle un vistazo.

Microcontroladores

El corazón de Arduino es un microcontrolador. En realidad, el resto de la placa se ocupa de facilitar la alimentación y permitir que se comunique con el ordenador al que está conectado.

Entonces, ¿exactamente qué es lo que nos dan cuando compramos uno de estos pequeños equipos para utilizar en nuestros proyectos?

La respuesta es que lo que recibimos es un pequeño ordenador en un chip. Tiene todo y más de lo que contenían los primeros ordenadores domésticos. Dispone de un procesador, un kilobyte de RAM (memoria de acceso aleatorio) para contener

datos, unos cuantos kilobytes de memoria EPROM (ROM programable borrable) o de memoria Flash para contener nuestros programas, y tiene pines de entrada y salida. Estos pines de entrada/salida son los que conectan el microcontrolador con el resto de nuestra electrónica.

Las entradas pueden leer señales digitales (¿el interruptor está abierto o cerrado?) y analógicas (¿cuál es la tensión en un pin?). Esto nos permite conectar innumerables tipos de sensores de luz, temperatura, sonido, etc.

Las salidas también pueden ser analógicas o digitales. Así, se puede establecer que un pin esté activado o desactivado (5 V o 0 V) y esto puede encender o apagar los LED directamente, o se puede usar la salida para controlar dispositivos más potentes, como motores. También pueden proporcionar tensión de salida analógica. Es decir, se puede fijar la salida de un pin a una determinada tensión, lo que permite controlar la velocidad de un motor o el brillo de una bombilla, por ejemplo, en lugar de solo encenderlo o apagarlo.

¿Qué hay en una placa Arduino?

En la Figura 2-1 se muestra uno de los tipos de placa Arduino, en este caso una placa **Arduino UNO**. Echemos un vistazo rápido a los diversos componentes de la placa.

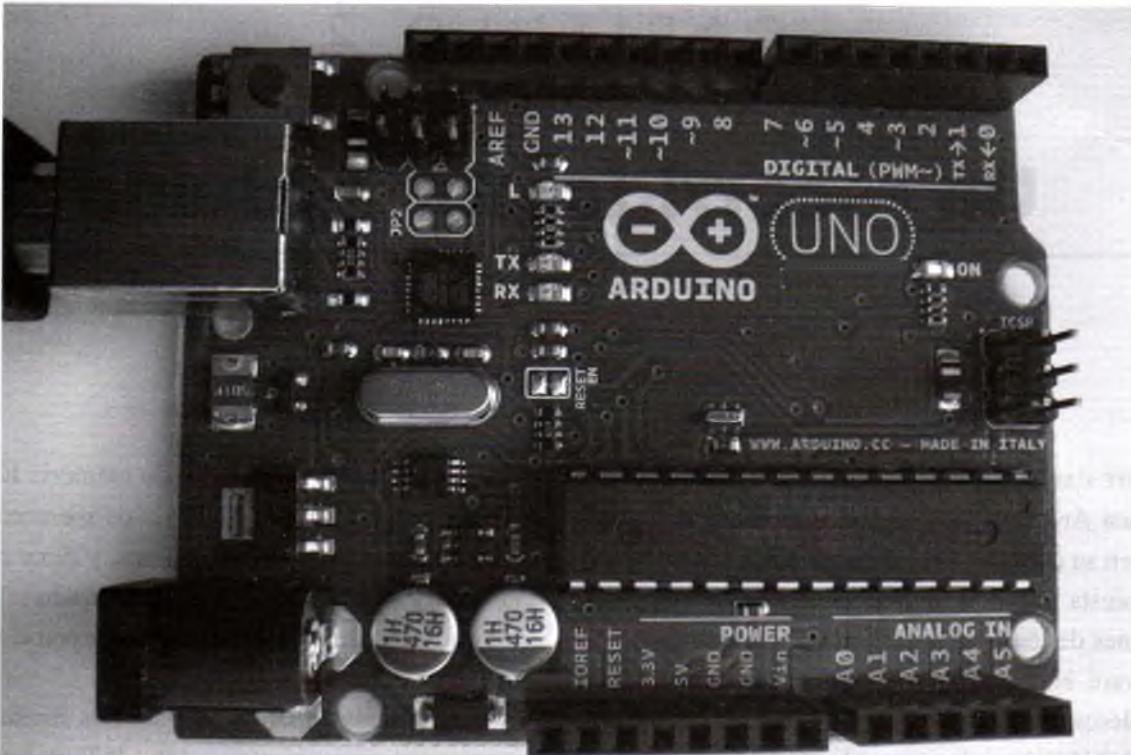


Figura 2-1 Los componentes de una Placa de Arduino.

Fuente de alimentación

Directamente debajo del **conector USB** se encuentra el **regulador de tensión de 5 V**. Regula a 5 V estabilizados cualquier tensión (entre 7 y 12 voltios) que se suministre desde el conector de alimentación.

5 V (junto con 3 V, 6 V, 9 V y 12 V) es una de las tensiones estándar en electrónica. 3, 6 y 9 V son estándar debido a que la tensión que se obtiene de una sola pila alcalina es 1,5 V, y todos ellos son prácticos múltiplos de 1,5 V, que es lo que se obtiene cuando se colocan en "batería" o en serie dos, tres, seis u ocho pilas de 1,5 V.

Si es así, se preguntará ¿entonces, por qué 5 V? Esa cifra no se puede conseguir utilizando pilas de 1,5 V. Pues bien, la respuesta se debe al hecho de que en los primeros días de la informática, se crearon un conjunto de chips, cada uno de los cuales contenía puertas lógicas. Estos chips usaban una tecnología llamada TTL (Transistor-Transistor Logic, lógica transistor-transistor), que era un poco delicada en cuanto a sus requisitos de tensión y requería una tensión entre 4.5 V y 5.5 V. Con lo

cual, 5 V se convirtió en el estándar de tensión en electrónica digital.

En nuestros días, el tipo de puertas lógicas utilizado en los chips ha cambiado y ahora son mucho más tolerantes con las diferentes tensiones.

El chip regulador de tensión de 5 V de nuestra placa es bastante grande para ser un componente de montaje en superficie. Esto es así para que pueda disipar el calor necesario para regular la tensión a una corriente razonablemente alta, que sirva para controlar los componentes electrónicos conectados a la placa.

Conexiones eléctricas

A continuación, veamos los conectores de la parte inferior de la Figura 2-1. Como puede ver, cada patilla está identificada con su nombre impreso junto a cada uno de los pines.

El primero es **Reset** (reiniciar). Esto hace la misma función que presionar el botón **Reset** en Arduino. Bastante parecido a reiniciar un PC, reinicia el microcontrolador, haciendo que comience su programa desde el principio. El pin **Reset** permite

reiniciar el microcontrolador momentáneamente estableciendo este pin alto (conectándolo a +5V).

El resto de los pines de esta sección proporcionan distintas tensiones (3.3 V, 5 V, GND, y 9 V), tal como aparece indicado. GND, o tierra, significa cero voltios. Es la tensión de referencia con la que se comparan de forma relativa el resto de tensiones de la placa.

Y, llegados a este punto, sería conveniente recordar al lector la diferencia entre **tensión** y **corriente**. No existe una analogía perfecta para explicar el comportamiento de los electrones en un cable, pero el autor cree que lo más parecido a esto es el símil hidráulico, especialmente apropiado para explicar los conceptos de tensión, corriente y resistencia. La relación entre estos tres parámetros se llama **Ley de Ohm**.

La Figura 2-2 resume la relación entre tensión, corriente y resistencia. La parte izquierda del dibujo muestra un circuito de tuberías, donde la parte superior está más alta que la parte inferior. Por lo que, naturalmente, el agua fluirá desde esa parte superior a la parte inferior. Dos factores determinan la cantidad de agua (la corriente) que pasa por cualquier punto del circuito en un momento dado:

- La altura del agua (o si se prefiere, la presión generada por la bomba). Esto sería como la **tensión o voltaje** en electrónica.

- La resistencia al flujo de agua ofrecida por la constricción de la tubería.

Cuanto más potente sea la bomba, más alto podrá bombarse el agua y mayor será la corriente que fluye a través del sistema. Por otro lado, cuanto mayor sea la resistencia de las tuberías, menor será la corriente.

En el lado derecho de la Figura 2-2, podemos ver el equivalente electrónico de nuestras tuberías. En este caso, la corriente es en realidad una medida de cuántos electrones fluyen por un punto por segundo. Y sí, la resistencia es la resistencia u oposición al flujo de los electrones.

En lugar de altura o presión, tenemos el concepto de tensión. La parte inferior del diagrama se encuentra a 0 V, o tierra, y hemos indicado que la parte superior del diagrama está a 5 V. Por lo que la corriente que fluye (I) será la diferencia de tensión (5) dividida por la resistencia R .

La **Ley de Ohm** usualmente se escribe como $V = I \times R$.

Normalmente, sabemos lo que vale V e intentamos calcular R o I , por lo que podemos hacer una pequeña reorganización para presentarlo de forma

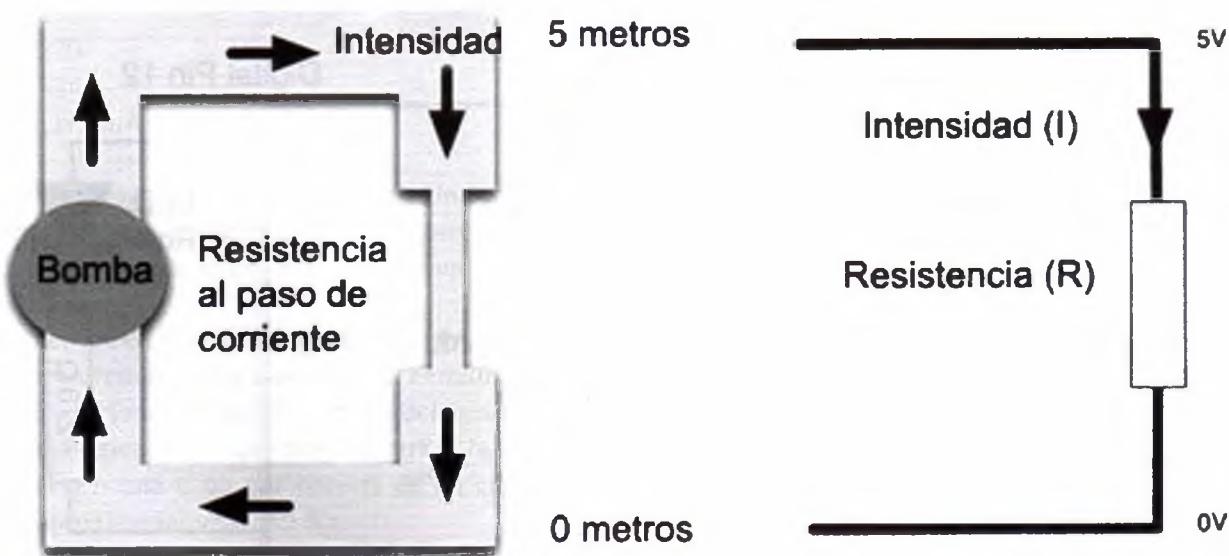


Figura 2-2 La ley de Ohm

más conveniente: $I = V/R$ y $R = V/I$.

Es muy importante hacer unos cuantos cálculos usando la Ley de Ohm cuando se conectan cosas a Arduino, o la placa podría resultar dañada si se le pide que proporcione demasiada corriente. No obstante, en general, las placas Arduino son bastante tolerantes con los abusos accidentales.

Así es que, volviendo a los pines de alimentación de Arduino, podemos ver que la placa nos proporciona útiles tensiones de alimentación de 3.3 V, 5 V y 9 V. Podemos utilizar cualquiera de estas tensiones para obtener corriente, siempre y cuando tengamos cuidado de no cortocircuitarlas (resistencia cero al paso de corriente), lo que podría provocar un exceso de corriente que podría causar daños. En otras palabras, tenemos que asegurarnos de que cualquier cosa que se conecte a la alimentación tiene suficiente resistencia para evitar que fluya demasiada corriente. Además de suministrar una determinada tensión, cada una de las conexiones de alimentación tiene un límite de corriente máxima que puede suministrar. Estas corrientes son 50 mA (milésimas de amperio) para la alimentación de 3.3 V y, aunque no está concretado en las especificaciones de Arduino, probablemente alrededor de 300 mA para los 5 V.

Entradas analógicas

La siguiente sección de conexiones analógicas, **ANALOG IN**, está marcada con los números 0 a 5. Estos seis pines se pueden utilizar para medir la tensión o voltaje que se les ha conectado, de forma que el valor se pueda utilizar en los **sketches**. Observe que miden una tensión y no una corriente. Por ellos sólo fluirá una pequeña corriente que luego irá a masa, debido a su gran resistencia interna.

Si bien aparecen etiquetadas como entradas analógicas, estas conexiones pueden utilizarse también como entradas o salidas digitales, aunque, por defecto, son entradas analógicas.

Conexiones Digitales

Cambiamos ahora al conector superior y comenzamos por el lado derecho (Figura 2-1). Tenemos pines denominados **Digital 0 a 13**. Estos se pueden utilizar como entradas o salidas. Cuando se utilizan como

salida, se comportan de forma bastante parecida a las tensiones de alimentación de las que hemos hablado antes, salvo que todas son de 5 V y se pueden activar o desactivar desde nuestro **sketch**. Por tanto, si los activamos desde nuestro **sketch** tendrán 5 V y si los desactivamos, tendrán 0 V. Al igual que con los pines de alimentación, debemos tener cuidado de no exceder su máxima capacidad de corriente.

Estas conexiones pueden suministrar 40 mA a 5 V. Eso es más que suficiente para encender un LED estándar, pero no para manejar directamente un motor eléctrico.

Como ejemplo, veamos cómo conectaríamos un LED a una de estas conexiones digitales. De hecho, volvamos al **Proyecto 1** del Capítulo 1.

Como recordatorio, la Figura 2-3 muestra el esquema eléctrico para manejar el LED que se utilizó por primera vez en el capítulo anterior. Si no utilizamos una resistencia con nuestro LED, sino que simplemente conectamos el LED entre el pin 12 y GND, cuando activemos el pin de salida digital 12 (5 V), podríamos quemar el LED, destruyéndolo.

Esto se debe a que los LEDs tienen una resistencia muy baja, lo que hace que fluya por ellos una corriente muy alta, a menos que sean protegidos conectándoles una resistencia para limitar el flujo de corriente.

Un LED necesita unos 10 mA para iluminarse razonablemente. La placa Arduino puede suministrar 50 mA, por lo que en este aspecto no hay

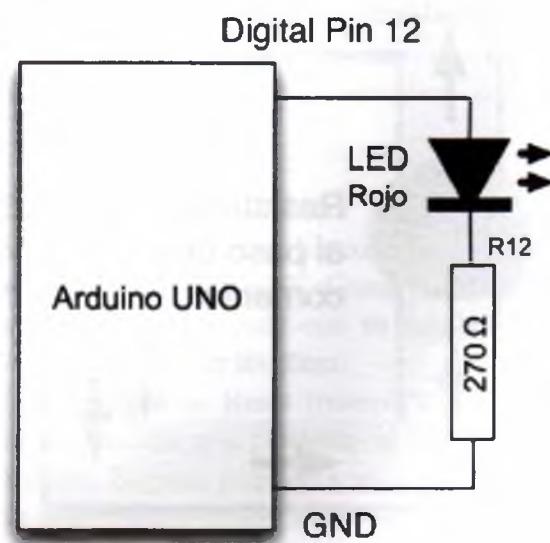


Figura 2-3 LED y resistencia en serie.

ningún problema; lo único que necesitamos es elegir una resistencia adecuada.

Los LEDs tienen la interesante propiedad de que, con independencia de cuánta corriente fluya por ellos, siempre habrá alrededor de 2V entre sus pines. Podemos utilizar este hecho y la ley de Ohm para averiguar el valor correcto de la resistencia a utilizar.

Sabemos que (al menos cuando está activado) el pin de salida está suministrando 5 V. Bien, acabamos de decir que 2 V son "utilizados" por nuestro LED, lo que deja 3 V ($5 - 2$) circulando a través de nuestra resistencia de limitación de corriente. Queremos que la corriente que fluya por el circuito sea 10 mA, por lo que podemos ver que el valor de la resistencia debe ser:

$$R = V/I$$

$$R = 3 \text{ V}/10 \text{ mA}$$

$$R = 3 \text{ V}/0,01 \text{ A}$$

$$R = 300 \Omega$$

Las resistencias vienen en valores estándar, y el valor más próximo a 300Ω es de 270Ω . Esto significa que en lugar de 10 mA, la corriente será en realidad:

$$I = V/R$$

$$I = 3/270$$

$$I = 11,111 \text{ mA}$$

Estas cosas no son críticas, y el LED funcionará más o menos aceptablemente con cualquier valor entre 5 y 30 mA, por lo que con 270Ω funcionará perfectamente.

También podemos configurar una de estas conexiones digitales para que sea una entrada, en cuyo caso, su funcionamiento será parecido a una entrada analógica, salvo que nos indicará si la tensión en el pin está o no por encima de un cierto umbral (aproximadamente 2,5 V).

Algunas de las conexiones digitales (3, 5, 6, 9, 10 y 11) están rotuladas **PWM**. Estas se pueden uti-

lizar para proporcionar una tensión de salida variable, en lugar de simplemente 5 V o 0 V.

En el lado izquierdo del conector superior de la Figura 2-1, hay otra conexión **GND** y una conexión denominada **AREF**. **AREF** se puede utilizar para escalar las lecturas de las entradas analógicas. Se utiliza muy raramente, y por el momento nos podemos olvidar de ella.

Microcontrolador

Volviendo a nuestro recorrido de la placa Arduino, el chip **microcontrolador** es el dispositivo rectangular negro con 28 pines. Está insertado en un zócalo DIL (Dual in-line) con lo que se puede reemplazar fácilmente. El microcontrolador usado en las placas **Arduino UNO** o **Duevelopment** es el **ATmega328**. En la Figura 2-4 podemos ver un diagrama de bloques que muestra las principales características de este dispositivo.

El corazón, o mejor dicho, el cerebro del dispositivo es la **CPU** (unidad central de procesamiento). Controla todo lo que ocurre dentro del dispositivo. Obtiene instrucciones de programa almacenadas en la **memoria Flash** y las ejecuta. Esto puede suponer obtener datos de la memoria de trabajo (RAM), procesarlos y luego volver a colocarlos de nuevo. O bien, puede significar cambiar una de las salidas digitales de 0 a 5 voltios.

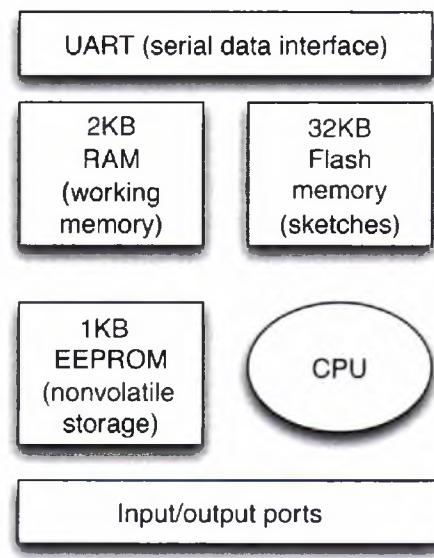


Figura 2-4 Diagrama de bloques del Atmega328.

La memoria ROM programable y borrible eléctricamente (memoria **EEPROM**) se asemeja a la memoria **Flash** en que es no volátil. Es decir, se puede apagar y encender el dispositivo sin perder lo que hubiera en la EEPROM. Mientras que la memoria **Flash** se destina a almacenar instrucciones de programa (de los sketches), la **EEPROM** se utiliza para almacenar datos que no queremos perder en caso de reinicio o de fallo de la alimentación.

Las antiguas placas Diecimila utilizan el **ATmega168**, que funciona de la misma manera que el Atmega328 excepto que tiene la mitad de cantidad de cada tipo de memoria. Tiene 16 KB de memoria Flash, 1KB de RAM y 512 bytes de memoria EEPROM.

Otros componentes

Encima y a la izquierda del microcontrolador se encuentra un pequeño componente rectangular plateado conocido como **oscilador de cristal de cuarzo**. Éste oscila 16 millones de veces por segundo, y en cada una de ellas, el microcontrolador puede realizar una operación: una adición, sustracción, etc.

En la esquina superior izquierda, junto al conector USB, se encuentra el pulsador de **Reset** (reinicio). Si lo pulsamos, éste envía un impulso lógico al pin **Reset** del microcontrolador, haciendo que éste inicie su programa desde cero y borre su memoria. Tenga en cuenta que cualquier programa guardado en el dispositivo se conserva debido a que éstos se guardan en la memoria **Flash** no volátil, es decir, la memoria que recuerda incluso cuando el dispositivo no está encendido.

En la parte central del lado derecho de la placa se encuentra el conector de programación serie, formado por seis pines. Este nos permite programar la Arduino sin utilizar el puerto USB. Pero dado que disponemos de una conexión USB y del software que facilita su utilización, no lo utilizaremos.

En la parte superior izquierda de la placa, junto al conector USB, se encuentra el chip de interfaz USB. Este convierte los niveles de señal utilizados por el estándar USB a niveles que pueden ser utilizados directamente por la placa Arduino.

La familia Arduino

Llegados a este punto, puede resultar útil conocer un poco los diferentes modelos de placas Arduino. Nosotros utilizaremos la **UNO** o la **Duemilanove** para la mayoría de nuestros proyectos; sin embargo, también trabajaremos más adelante con la interesante placa **Arduino Lilypad**.

La **Lilypad** (Figura 2-5), es una pequeña y delgada placa Arduino que se puede coser en las prendas de vestir para utilizarla en aplicaciones que se conocen con el nombre de wearable computing o ropa tecnológica. No dispone de conexión USB, y es necesario utilizar un adaptador aparte para programarla. Es un diseño excepcionalmente bello. Inspirado por su apariencia de reloj, la utilizaremos en el Proyecto 29 (Reloj binario incomprensible).

En el otro extremo del espectro se encuentra la **Arduino Mega**. Esta placa tiene un procesador más rápido, con más memoria y un mayor número de pines de entrada/salida.

Habilmente diseñada, la Arduino Mega puede utilizar las **shields** construidas para placas más pequeñas como Arduino UNO, Duemilanove o Diecimila. Las shields se instalan en la parte delantera de la placa, dejando el acceso libre a la doble fila de conectores adicionales que se encuentran en la parte trasera de la Mega. En realidad, sólo algunos proyectos muy exigentes necesitan utilizar una placa tan potente como la Arduino Mega.

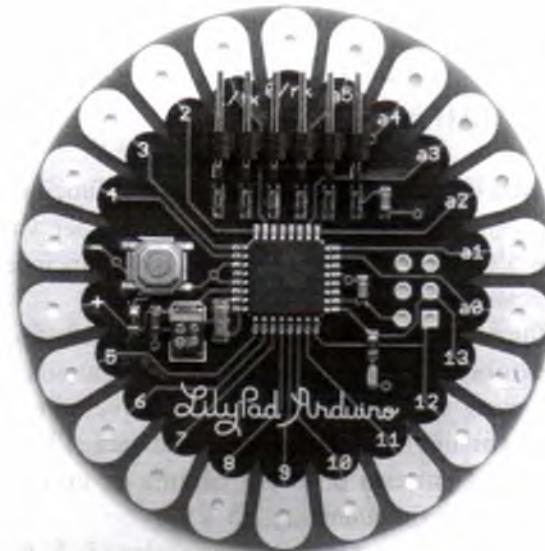


Figura 2-5 Una placa Arduino Lilypad.

El lenguaje C

Existen muchos lenguajes para programar microcontroladores, desde el complejo lenguaje **Assembly** (Ensamblador) a los lenguajes de programación gráfica como **Flowcode**. Arduino se sitúa en algún punto intermedio entre estos dos extremos y utiliza el lenguaje de programación C, aunque eliminando parte de su complejidad, lo cual hace que sea fácil iniciarse con él.

El lenguaje **C** es, en términos informáticos, un lenguaje clásico y venerable. Se adapta bien a la programación de los microcontroladores porque se inventó en un momento en el que, comparado con los "monstruos" actuales, los ordenadores de entonces eran mucho menos sofisticados.

C es un lenguaje fácil de aprender, que se compila en un eficiente "código máquina" y que requiere poco espacio en nuestra limitada memoria de Arduino.

Un ejemplo

Vamos a examinar ahora más detalladamente el **sketch** del **Proyecto 1**. Aquí se muestra el listado del código de este **sketch** para hacer parpadear un LED. Hemos ignorado todas las líneas que comienzan con // o los bloques de líneas que comienzan con /* y terminan con */ porque estas son las líneas de comentarios, que no tienen ningún efecto sobre el programa y están allí para aportar información.

```
int ledPin = 13;
// LED conectado al pin digital 13
void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    digitalWrite(ledPin, HIGH);
    // apaga el LED
    delay(1000);
    // espera un segundo
```

```
    digitalWrite(ledPin, LOW);
    // apaga el LED
    delay(1000);
    // espera un segundo
}
```

Es práctica habitual incluir dicho texto de información en la parte superior de cualquier archivo de programa, para indicar lo que hace. También puede incluir comentarios que describan alguna parte complicada del código, o cualquier cosa que requiera una cierta explicación.

El entorno de desarrollo Arduino utiliza algo llamado un **compilador**, el cual convierte la secuencia de comandos en **código máquina**, que será el que se ejecutará en el microcontrolador.

Por lo tanto, si vamos a la primera línea real de código, tenemos:

```
int ledPin = 13;
```

Esta línea de código da un nombre al pin digital de salida que vamos a conectar al LED. Si analizamos detenidamente la placa Arduino, verá el terminal del **pin 13** entre **GND** y el **pin 12** en el conector superior de la placa. La placa trae de fábrica un pequeño LED verde incorporado en la misma y conectado al **pin 13**. Vamos a cambiar la tensión de este pin entre 0 V y 5 V para que el LED parpadee.

Vamos a utilizar un nombre para el pin, de forma que sea fácil de cambiar y poder utilizar uno diferente. Puede ver que nos referimos a "**ledPin**" al final del **sketch**. Puede que prefiera utilizar el **pin 12** y el LED externo que utilizó con la placa de pruebas (**protoboard**) en el Capítulo 1. Pero, por ahora, vamos a suponer que está usando el LED que trae incorporado conectado al **pin 13**.

Observe que no escribimos simplemente:

```
led pin = 13
```

Esto se debe a que los compiladores son algo quisquillosos y precisos en lo referido a cómo escribimos nuestros programas. Los nombres que utilicemos en los programas no pueden usar espacios, por lo que convencionalmente se utiliza lo que se

denomina la nomenclatura “Mayúsculas en segunda palabra”. De esta forma, comenzamos cada palabra (excepto la primera) con una letra mayúscula y eliminamos los espacios; esto nos da:

```
ledPin = 13
```

La palabra **ledPin** es lo que se denomina **una variable**. Cuando se deseé utilizar una variable por primera vez en el **sketch**, hay que indicar al compilador qué tipo de variable es. Puede ser de tipo **int** (del inglés “integer”), como es el caso aquí, o de tipo **float** (decimal), o cualquiera de los restantes tipos que describiremos más adelante en este capítulo.

Un **int** es un entero, es decir, un número entero, que es justo lo que necesitamos para referirnos a un determinado pin de la Arduino. Después de todo, no existe un pin 12.5, por lo que no sería apropiado utilizar un número decimal o de coma flotante (**float**).

La sintaxis para una declaración de **variable** es:

```
tipo nombreVariable = valor;
```

Así es que, primero tenemos el tipo (**int**), luego un espacio, luego un nombre de variable con esta nomenclatura de introducir la mayúscula en la segunda palabra (**ledPin**), luego un signo igual (=), luego un valor y, por último, un punto y coma para indicar el final de la línea:

```
int ledPin = 13;
```

Como ya he mencionado, los compiladores son quisquillosos, por lo que si se olvida introducir el punto y coma, recibirá un mensaje de error al compilar el sketch. Pruebe a eliminar el punto y coma y luego haga clic en el botón Play (reproducir). Debería ver un mensaje como este:

```
error: expected unqualified-id before
numeric constant
```

No es exactamente “se te olvidó un punto y coma”, y no es raro que los mensajes de error sean muchos de ellos bastante confusos.

Las siguientes líneas del **sketch** son:

```
void setup()
{
    // ejecutar una vez, cuando se inicia el
    // sketch
    {
        pinMode(ledPin, OUTPUT);
        // establece el pin digital como salida
    }
}
```

Esto es lo que se llama una **función** y, en este caso, la función se llama **setup**. Todos los sketches deben contener una función **setup**, y las líneas de código dentro de la función que van dentro de las llaves se ejecutarán en el orden en el que están escritas. En este caso, es justo la línea que empieza con **pinMode**.

Un buen punto de partida para cualquier proyecto nuevo es copiar este proyecto de ejemplo y luego modificarlo para que se adapte a sus necesidades.

No nos preocuparemos demasiado de las funciones en esta etapa, más que para decir que la función **setup** se ejecutará cada vez que se reinicie la Arduino, incluido cuando se conecta la alimentación por primera vez. También se ejecutará cada vez que se descargue un nuevo **sketch**.

En este caso, la única línea de código en **setup** es:

```
pinMode(ledPin, OUTPUT);
// establece el pin digital como salida
```

Debemos mencionar que al final de esta línea tenemos un tipo diferente de comentario. Es decir, un comentario de una sola línea, el cual comienza con una // y termina al final de la línea.

La línea puede interpretarse como una orden que damos a Arduino para que utilice ledPin como una salida digital. Si tuviéramos un interruptor conectado a ledPin, podríamos configurarlo como una entrada utilizando:

```
pinMode(ledPin, INPUT);
```

No obstante, deberíamos llamar a la variable con un nombre más adecuado, como **switchPin**.

Las palabras **INPUT** y **OUTPUT** son lo que se denominan **constantes**, las cuales siempre se definirán en C como números. **INPUT** puede

definirse como 0 y **OUTPUT** como 1, pero no tendrá que ver realmente qué número se utiliza, ya que siempre se referirá a ellas como **INPUT** y **OUTPUT**. Más adelante en este capítulo, veremos dos constantes más, **HIGH** y **LOW**, que se utilizan cuando se establece la salida de un pin digital a +5 V ó 0 V, respectivamente.

La siguiente sección de código es otra función que deben tener todos los **sketches** Arduino; se denomina **loop** (bucle):

```
void loop()
{
    digitalWrite(ledPin, HIGH);
    // enciende el LED
    delay(1000);
    // espera un segundo
    digitalWrite(ledPin, LOW);
    // apaga el LED
    delay(1000);
    // espera un segundo
}
```

La función **loop** se ejecuta de manera continua hasta que se apague la Arduino. Es decir, tan pronto como termina la ejecución de los comandos que contiene, empieza de nuevo. Recuerde que una placa Arduino es capaz de ejecutar 16 millones de comandos por segundo, por lo que las cosas dentro de **loop** (bucle) se repetirán con mucha frecuencia mientras no se indique lo contrario.

En este caso, lo que queremos que Arduino siga haciendo continuamente es encender el LED, esperar un segundo, apagar el LED y luego esperar otro segundo. Cuando haya terminado, comenzará de nuevo encendiendo el LED. De esta manera, continuará ejecutando el bucle (**loop**) indefinidamente.

A estas alturas, la sintaxis de comando de **digitalWrite** y **delay** resultarán cada vez más familiares. Aunque podemos pensar en ellos como comandos que se envían a la placa Arduino, en realidad son funciones similares a **setup** y **loop**, aunque en este caso tienen lo que se denominan parámetros. En el caso de **digitalWrite**, se dice que toma dos parámetros: el

pin de Arduino en el que escribe y el valor a escribir.

En nuestro ejemplo, pasamos los parámetros **ledPin** y **HIGH** para activar el LED y, a continuación, **ledPin** y **LOW** para apagarlo de nuevo.

Tipos de datos y variables

Ya conocemos la variable **ledPin** y hemos declarado que es de tipo **int** (entero). La mayoría de las variables que vaya a utilizar en los **sketches** probablemente sean también del tipo **int**. Un **int** contiene un número entero entre -32.768 y +32.767. Esto utiliza sólo dos bytes de datos para cada uno de los números almacenados de los 1024 bytes disponibles para almacenamiento en una Arduino. Si ese rango no fuera suficiente, se puede utilizar un número **long** (largo), que utiliza cuatro bytes por cada número y le proporcionará un intervalo de números entre -2.147.483.648 y +2.147.483.647.

En la mayoría de los casos, un **int** (entero) representa un compromiso entre precisión y cantidad de memoria usada.

Si está comenzando a programar, yo usaría **int** para prácticamente todo y gradualmente iría ampliando el repertorio de tipos de datos a medida que aumentara mi experiencia.

En la Tabla 2-1 se resumen otros tipos de datos disponibles.

Una cosa a tener en cuenta es que si los tipos de datos exceden su rango de valores, ocurren cosas extrañas. Así, si tiene una variable **byte** con 255, y agrega 1 a la misma, obtendrá 0. O mucho peor, si tiene una variable **int** con 32.767 y le agrega 1, terminará con -32.768.

Hasta que se familiarice con el uso de diferentes tipos de datos, le recomendaría que utilizara sólo los de tipo **int**, ya que sirven prácticamente para todo.

Aritmética

Es bastante raro que sea necesario emplear mucha aritmética en los **sketches**. En ocasiones, puede que, por ejemplo, tenga que escalar una entrada analógica para convertirla en una temperatura o, con más frecuencia, agregar 1 a una variable de recuento.

TABLA 2-1 Tipos de datos en C

Tipo	Memoria (byte)	Rango	Notas
boolean	1	Verdadero o falso (0 or 1)	
char	1	-128 a +128	Se utiliza para representar un código de carácter ASCII (es decir, "A" se representa como 65). Normalmente, sus números negativos no se utilizan.
byte	1	0 a 255	
int	2	-32,768 a +32,767	
unsigned int	2	0 a 65,536	Se pueden utilizar para disponer de precisión extra, cuando no se necesitan números negativos. Deben utilizarse con cuidado ya que las operaciones aritméticas con ints pueden producir resultados inesperados.
long	4	-2,147,483,648 a +2,147,483,647	Sólo se necesitan para representar números muy grandes.
unsigned long	4	0 a 4,294,967,295	Ver "unsigned int".
float	4	-3.4028235E+38 a + 3.4028235E+38	
double	4	igual que float	Normalmente, esto serían 8 bytes y mayor precisión que los valores "float", con un rango superior. Sin embargo, en Arduino son iguales a "float".

Cuando esté realizando cálculos, necesitará poder asignar el resultado de dicho cálculo a una variable.

Las siguientes líneas de código tienen dos misiones. La primera proporciona a la variable **y** el valor 50 y la segunda proporciona a la variable **x** el valor de **y + 100**.

```
y = 50;
x = y + 100;
```

Strings (cadenas)

Cuando los programadores hablan de **strings** (cadenas), se están refiriendo a una cadena de caracteres, como, por ejemplo, el mensaje "Hola a todos". En el mundo de Arduino, hay un par de situaciones en las que puede que tenga que utilizar **strings**: al escribir mensajes en una pantalla LCD (cristal líquido) o cuando se devuelven datos de texto serie a través de

la conexión USB.

Las **strings** (cadenas) se crean usando la siguiente sintaxis:

```
char* message = "Hello World";
```

La palabra **char*** indica que la variable **message** es un puntero a un carácter. Por ahora, no tenemos que preocuparnos mucho acerca de su funcionamiento. Lo veremos más adelante en el libro cuando echemos un vistazo a la interfaz con pantallas de texto LCD.

Instrucciones condicionales

Las instrucciones condicionales son una forma de tomar decisiones en un **sketch**. Por ejemplo, el **sketch** puede encender el LED si el valor de una variable de temperatura cae por debajo de un determinado umbral.

El código para esto es el siguiente:

```
if (temperature < 15)
{
    digitalWrite(ledPort, HIGH);
}
```

La línea o líneas de código dentro de las llaves sólo se ejecutarán si fuera cierta la condición tras la palabra clave.

La condición tiene que ir entre paréntesis, y es a lo que los programadores llaman una expresión lógica. Una **expresión lógica** es como una expresión matemática que siempre debe devolver uno de dos valores posibles: verdadero (**true**) o falso (**false**).

La siguiente expresión devuelve **true** si el valor de la variable de temperatura es inferior a 15:

```
(temperature < 15)
```

Al igual que <, tenemos: >, <= y >=.

Para ver si dos números son iguales, se puede usar **==**, y para probar si no son iguales, puede usar **!=**.

Por tanto, la siguiente expresión devolverá **true** si la variable de temperatura tuviera cualquier valor que no fuera 15:

```
(temperature != 15)
```

También se pueden crear condiciones complejas mediante lo que se denominan operadores lógicos. Los principales operadores son **&&** (y) y **||** (o).

Por lo tanto, un ejemplo que enciende el LED si la temperatura es inferior a 15 o superior a 20 podría tener un aspecto similar a esto:

```
if ((temperature < 15) || (temperatura
> 20))
{
    digitalWrite(ledPort, HIGH);
}
```

A menudo, cuando se utiliza una instrucción con el condicional **if** (si..) se desea hacer una cosa si la condición es verdadera y otra cosa si es falsa. Esto se puede hacer utilizando la palabra clave **else**, como se muestra en el ejemplo siguiente. Note el uso de los paréntesis anidados para dejar claro a qué se le está aplicando el **o**.

```
if ((temperature < 15) || (temperatura
> 20))
{
    digitalWrite(ledPort, HIGH);
}
else
{
    digitalWrite(ledPort, LOW);
}
```

Resumen

En este capítulo hemos explorado el hardware de las placas Arduino y actualizado un poco nuestros conocimientos de electrónica elemental.

También hemos iniciado nuestra exploración del lenguaje de programación C. No se preocupe si encuentra algunas de estas cosas difíciles de seguir. Hay mucho que aprender si no está familiarizado con la electrónica y, si bien el objetivo del autor es explicar cómo funciona todo, puede empezar con los proyectos en primer lugar, y luego volver a la teoría cuando lo desee.

En el siguiente capítulo vamos a ocuparnos de la programación de nuestra placa Arduino y nos embarcaremos en algunos proyectos más serios.

CAPÍTULO 3

Proyectos con LEDs

EN ESTE CAPÍTULO, vamos a construir algunos proyectos basados en LED. La parte electrónica será muy simple para que podamos concentrarnos en la programación de Arduino.

La programación de los microcontroladores puede ser un asunto delicado que requiere un conocimiento profundo de las interioridades de los dispositivos: registros, punteros, etc. Esto se debe, en parte, a que los modernos microcontroladores pueden configurarse casi hasta el infinito. Arduino ha normalizado su configuración de hardware, lo que, a cambio de una pequeña pérdida de flexibilidad, ha hecho que los dispositivos sean mucho más fáciles de programar.

Proyecto 2 Intermitente S.O.S de código Morse

El código Morse solía ser un método de comunicación vital en los siglos XIX y XX. Su codificación de letras en una serie de puntos y rayas permitió que pudiera ser enviado a través de cables telegráficos, enlaces radiofónicos y utilizando luces de señalización. Las letras S.O.S. (*Save Our Souls*, Salvar nuestras almas) siguen siendo reconocidas como una señal internacional de socorro.

En este proyecto, haremos que nuestro LED parpadee la secuencia S.O.S. una y otra vez. Para este proyecto, necesitará los mismos componentes que para el Proyecto 1.

COMPONENTES Y EQUIPOS	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
D1 LED rojo de 5 mm	23
R1 Resistencia 270 Ω 0,5W	6
<ul style="list-style-type: none">■ Pueden servir prácticamente cualquier LED y resistencia de 270 Ω que haya disponibles.■ No hacen falta más herramientas que un par de alicates o cortacables.	

Hardware

El hardware es exactamente el mismo que para el Proyecto 1. Por lo tanto, puede simplemente conectar la resistencia y el LED directamente en los conectores de Arduino o utilizar una placa de pruebas (**protoboard**) (véase el Capítulo 1).

Software

En lugar de comenzar a escribir este proyecto desde cero, utilizaremos el Proyecto 1 como punto de partida. De esta forma, si no lo ha hecho ya, por favor, complete el Proyecto 1.

Si no lo ha hecho anteriormente, descargue el código del proyecto de www.arduinoevil-genius.com; a continuación, también puede cargar el **sketch** del

Proyecto 1 de su **Arduino Sketchbook** y descargarlo en la placa (véase el Capítulo 1). No obstante, creo que le ayudará a entender mejor su placa Arduino si modifica el sketch del Proyecto 1 como se sugiere a continuación.

Modifique la función **loop** del Proyecto 1, para que aparezca como se muestra aquí. Lo más recomendable en este caso es utilizar las opciones de "copiar y pegar".

```
void loop()
{
    digitalWrite(ledPin, HIGH);
    // S (...) primer punto
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(200);
    digitalWrite(ledPin, HIGH);
    // segundo punto
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(200);
    digitalWrite(ledPin, HIGH);
    // tercer punto
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(500);
    digitalWrite(ledPin, HIGH);
    // O (—) primera raya
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
    digitalWrite(ledPin, HIGH);
    // segunda raya
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
    digitalWrite(ledPin, HIGH);
    // tercera raya
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
    digitalWrite(ledPin, HIGH);
    // S (...) primer punto
    delay(200);
    digitalWrite(ledPin, LOW);
    delay(200);
    digitalWrite(ledPin, HIGH);
    // segundo punto
    delay(200);}
```

```
digitalWrite(ledPin, LOW);
delay(200);
digitalWrite(ledPin, HIGH);
// tercer punto
delay(200);
digitalWrite(ledPin, LOW);
delay(1000);
// Espere 1 segundo antes de que
// empecemos de nuevo
}
```

Aunque este **sketch** funciona, y no debe dudar en probarlo, vamos a cambiarlo para mejorarlo y, al mismo tiempo, hacerlo mucho más corto.

Podemos reducir el tamaño del **sketch** creando nuestra propia función que reemplace las cuatro líneas de código involucradas en cualquier parpadeo con una sola línea.

Después de la llave final de la función **loop**, agregue el código siguiente:

```
void flash(int duration)
{
    digitalWrite(ledPin, HIGH);
    delay(duration);
    digitalWrite(ledPin, LOW);
    delay(duration);
}
```

Ahora modifique la función **loop** para que se parezca a esto:

```
void loop()
{
    flash(200); flash(200); flash(200);
    // S
    delay(300);
    // En caso contrario los destellos se
    // ejecutan juntos
    flash(500); flash(500); flash(500);
    // O
    flash(200); flash(200); flash(200);
    // S
    delay(1000);
    // Espere 1 segundo antes de que
    // empecemos de nuevo
}
```

LISTADO DE PROYECTO 2

```

int ledPin = 13;

void setup()                      // ejecutar una vez, cuando se inicia sketch
{
  pinMode(ledPin, OUTPUT);          // establece el pin digital como salida
}

void loop()
{
  flash(200); flash(200); flash(200); // S
  delay(300);                      // si no, los destellos se ejecutan juntos
  flash(500); flash(500); flash(500); // O
  flash(200); flash(200); flash(200); // S
  delay(1000);                     // espere un segundo antes de empezar de nuevo
}

void flash(int duration)
{
  digitalWrite(ledPin, HIGH);
  delay(duration);
  digitalWrite(ledPin, LOW);
  delay(duration);
}

```

El código final completo se muestra en el Listado del Proyecto 2.

Esto hace el **sketch** mucho más corto y fácil de leer.

Pongamos todo junto

Así concluye el Proyecto 2. Ahora trataremos algunos aspectos más sobre la programación de Arduino antes de pasar a ver el Proyecto 3, donde vamos a utilizar nuestro mismo hardware para escribir un traductor de código Morse, en el que podemos escribir frases en nuestro ordenador y hacer que se transmitan como código Morse. En el Proyecto 4, mejoraremos la luminosidad de nuestro LED transmisor sustituyendo éste por un LED de alta potencia del tipo Luxeon.

Pero antes, veremos un poco más de teoría para comprender mejor los Proyectos 3 y 4.

Loops (bucles)

Los **loops** (bucles) nos permiten repetir un grupo de comandos un número determinado de veces o hasta que se cumpla cierta condición.

En el Proyecto 2, sólo queríamos que el LED parpadease con los tres puntos de una S, por lo que no supone gran dificultad repetir el comando **flash** tres veces. Sin embargo, esto sería poco práctico si tuviéramos que hacer parpadear el LED 100 o 1000 veces. En ese caso, en C podemos utilizar el comando de lenguaje **for**.

```

for (int i = 0; i < 100; i++)
{
  flash(200);
}

```

El bucle **for** es parecido a una función que toma tres argumentos aunque, aquí, en lugar de las habituales comas (,), esos argumentos se separan

mediante un punto y coma (;). Esto es una de las peculiaridades del lenguaje C, pero no se preocupe. el compilador le avisará si se equivoca.

Lo primero que aparece en los paréntesis después de **for** es una declaración de variable. En este caso especifica que la variable se va a utilizar como una **variable counter** y le asigna un valor inicial, en este caso, 0.

La segunda parte es una condición que debe ser verdad para permanecer en el bucle. En este caso, nos quedaremos en el bucle mientras **i** sea inferior a 100, pero tan pronto como **i** sea 100 o superior, dejaremos lo que estuvíramos haciendo dentro del bucle.

La parte final es lo que se debe hacer una vez que se han ejecutado todos los comandos del bucle. En este caso, es incrementar **i** en 1, de manera que, después de 100 vueltas dentro del bucle, **i** deja de ser inferior a 100 y, por tanto, debe salir del bucle.

Otra forma de hacer un bucle en C es usar el comando **while**. El mismo ejemplo mostrado anteriormente se podría llevar a cabo usando un comando **while**, como se muestra aquí:

```
int i = 0;
while (i < 100)
{
    flash(200);
    i++;
}
```

La expresión entre paréntesis detrás de **while** debe ser verdadera para permanecer en el bucle. Cuando deja de ser cierta, el **sketch** continuará ejecutando los comandos que aparecen tras la última llave.

Las llaves se utilizan para agrupar un grupo de comandos. En el argot de programación se les llama un **bloque**.

Arrays (matrices)

Los **arrays** (matrices) son una forma de contener una lista de valores. Las variables que hemos encontrado hasta la fecha sólo incluían un único valor,

generalmente de tipo int. Por el contrario, un **array** (matriz) contiene una lista de valores, y se puede acceder a cualquiera de esos valores por su posición en la lista.

C, al igual que la mayoría de los lenguajes de programación, comienza sus posiciones de índice en 0 en lugar de 1. Esto significa que el primer elemento es en realidad el elemento cero.

Para ilustrar el uso de los **arrays**, podríamos cambiar nuestro código Morse de ejemplo y emplear un **array** de duraciones de destello. Luego podemos usar un bucle **for** para recorrer cada uno de los elementos de la matriz.

En primer lugar, vamos a crear un **array** de valores **int** que contengan las duraciones:

```
int durations[] = {200, 200, 200, 500, 500,
500, 200, 200, 200}
```

Puede indicar que una variable contiene un array colocando [] tras el nombre de la variable. Si especifica el contenido de la matriz al mismo tiempo que lo está definiendo, como en el ejemplo anterior, no necesita especificar el tamaño de la matriz. Si no establece su contenido inicial, entonces deberá especificar el tamaño de la matriz dentro de los corchetes. Por ejemplo:

```
int durations[10];
```

Ahora podemos modificar nuestro método del **loop** para utilizar el **array**:

```
void loop()
// ejecutar una y otra vez
{
    for (int i = 0; i < 9; i++)
    {
        flash(durations[i]);
    }
    delay(1000);
    // espere 1 segundo antes de que
    // empiezemos de nuevo
}
```

Una ventaja obvia de este enfoque es que es fácil cambiar el mensaje simplemente alterando la matriz de duraciones (`durations[i]`). En el Proyecto 3, llevaremos el uso de los arrays un paso más allá para crear un destello de código Morse de propósito más general.

Proyecto 3

Traductor de código Morse

En este proyecto vamos a usar el mismo hardware que en los Proyectos 1 y 2, pero vamos a escribir un nuevo **sketch** que nos va a permitir escribir una frase en nuestro ordenador y hacer que la placa Arduino lo convierta en los puntos y rayas apropiados del código Morse.

La Figura 3-1 muestra el traductor de código Morse en acción. El contenido del cuadro de mensaje será transmitido como puntos y rayas mediante el LED.

Para ello, haremos uso de lo que hemos aprendido acerca de **arrays** (matrices) y **strings** (cadenas), y también aprenderemos algo acerca de cómo

enviar mensajes desde el ordenador a la placa Arduino a través del cable USB.

Para este proyecto, necesitará los mismos componentes que para el Proyecto 1 y 2. De hecho, el hardware es exactamente el mismo; vamos simplemente a modificar el **sketch** del Proyecto 1.

COMPONENTES Y EQUIPO	
Descripción	Apéndice A
Placa Arduino UNO o Duemilanove o similar	1
D1 LED rojo de 5 mm	23
R1 Resistencia 270 Ω 0,5W	6

Hardware

Por favor, remítase al Proyecto 1 para el conexionado de este proyecto.

Puede simplemente conectar la resistencia y el LED directamente en los conectores de Arduino o utilizar la placa de pruebas (**protoboard**) (véase el Capítulo 1). Puede incluso cambiar sólo la variable

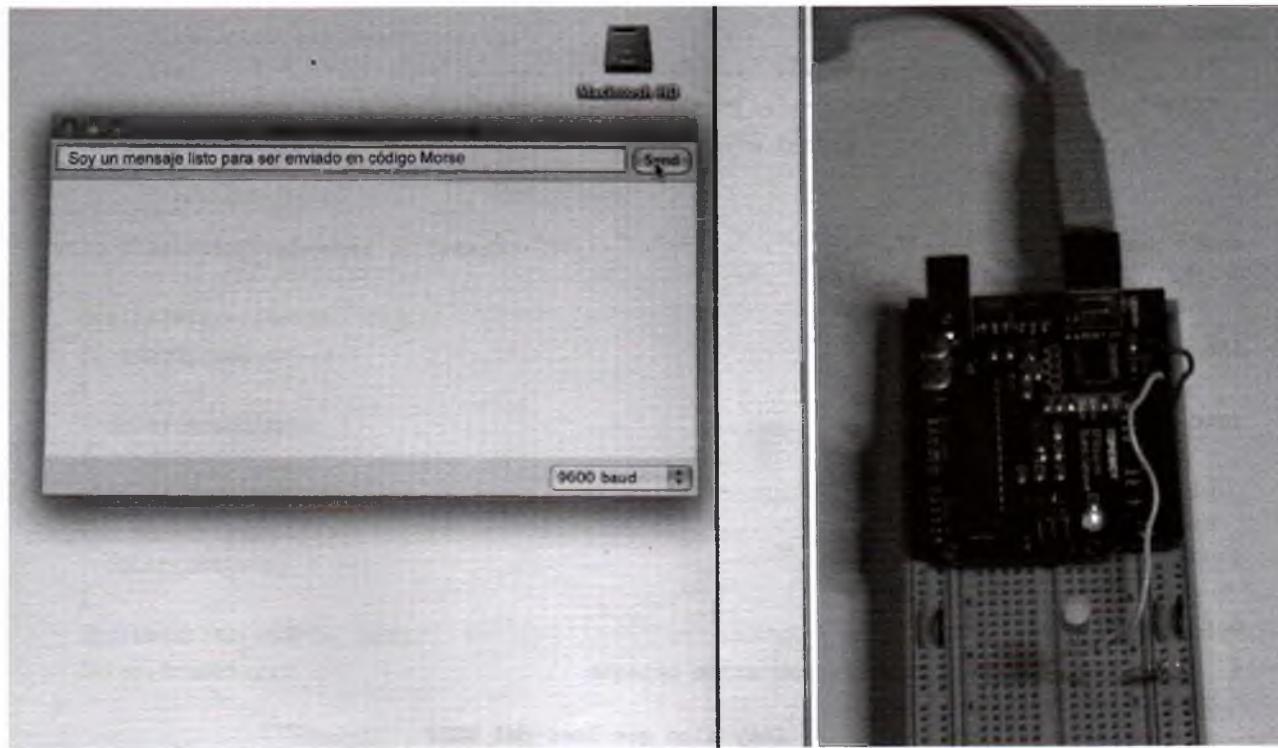


Figura 3-1 Traductor de código Morse

ledPin en el **sketch** para que sea el pin 13, y así utilizar el LED integrado en la placa, no necesitando entonces ningún componente externo.

Software

Las letras del código Morse se muestran en la Tabla 3-1.

Algunas de las normas del código Morse son que la duración de una raya es tres veces la de un punto, el tiempo que transcurre entre cada raya o punto es igual a la duración de un punto, el espacio entre dos letras tiene la misma longitud que la raya, y el espacio entre dos palabras tiene la misma duración que siete puntos.

En aras de este proyecto, no nos vamos a preocupar de la puntuación, aunque sería un interesante ejercicio el que intentara añadir esto al **sketch**.

Para obtener una lista completa de todos los caracteres Morse, véase:

http://en.wikipedia.org/wiki/Morse_code.

TABLA 3-1 Alfabeto del código Morse

A	-	N	-.	O	---
B	-..	O	--	1	---
C	-.-.	P	.--.	2	.---
D	-..	Q	.---.	3	...-
E	.	R	..	4
F	.-.	S	...	5
G	--.	T	-	6
H	U	..-	7	-----
I	..	V	...-	8	-----
J	----	W	9	-----
K	-.-	X	-.-		
L	-..-	Y	.---		
M	--	Z	--.		

Puede ver el **sketch** en el Listado del Proyecto 3. A continuación damos una explicación de su funcionamiento.

LISTADO DE PROYECTO 3

```
int ledPin = 12;

char* letters[] = {
    ".-", "-...", "-.-.", "-..", ".-", "...-", "-.-.", "-..-", "...-", "...-", "...-", // A-I
    "-.--", "-.-", "-.-.", "---", "-.-", "---.", "-.-.", "-.-.", "-.-.", "-.-.", // J-R
    "...", "-.", "-..", "...-", "---", "-.-.", "-.-.", "-.-.", "-.-." // S-Z
};

char* numbers[] = {"-----", ".----", "----.", "----", "----.", "----.", "----.", "----.", "----.", "----.", "----.", "----."};

int dotDelay = 200;

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}

void loop()
{
    char ch;
    if (Serial.available()) // ¿hay algo que leer del USB?
}
```

LISTADO DE PROYECTO 3 (continuación)

```
{  
    ch = Serial.read();           // leer una letra  
    if (ch >= 'a' && ch <= 'z')  
    {  
        flashSequence(letters[ch - 'a']);  
    }  
    else if (ch >= 'A' && ch <= 'Z')  
    {  
        flashSequence(letters[ch - 'A']);  
    }  
    else if (ch >= '0' && ch <= '9')  
    {  
        flashSequence(numbers[ch - '0']);  
    }  
    else if (ch == ' ')  
    {  
        delay(dotDelay * 4);      // espacio entre palabras  
    }  
}  
}  
  
void flashSequence(char* sequence)  
{  
    int i = 0;  
    while (sequence[i] != NULL)  
    {  
        flashDotOrDash(sequence[i]);  
        i++;  
    }  
    delay(dotDelay * 3);          // espacio entre letras  
}  
  
void flashDotOrDash(char dotOrDash)  
{  
    digitalWrite(ledPin, HIGH);  
    if (dotOrDash == '.')  
    {  
        delay(dotDelay);  
    }  
    else // debe ser una -  
    {  
        delay(dotDelay * 3);  
    }  
    digitalWrite(ledPin, LOW);  
    delay(dotDelay);            // espacio entre destellos  
}
```

Hacemos el seguimiento de nuestros puntos y rayas usando matrices de cadenas (**arrays de strings**). Contamos con dos de estos, uno para las letras (**letters**) y otro para los números. Por lo tanto, para averiguar lo que necesitamos para hacer parpadear el LED con la primera letra del alfabeto (A), obtendremos la primera posición de **string letters[0]** -recuerde, el primer elemento de una matriz es el elemento 0, no el 1.

Hemos definido la variable **dotDelay**, por lo tanto, si queremos hacer que nuestro código Morse parpadee más rápido o más lento, podemos cambiar este valor, ya que todas las duraciones se han definido como múltiplos del tiempo de duración de un punto.

La función **setup** es prácticamente igual a la de nuestros proyectos anteriores; sin embargo, esta vez estamos recibiendo comunicaciones del puerto USB, por lo que debemos añadir el comando:

```
Serial.begin(9600);
```

Esto indica a la placa Arduino que configure la velocidad de comunicación a través de USB para que sean **9600 baudios**. Esto no es que sea muy rápido, pero es suficiente para nuestros mensajes de código Morse. También es una buena velocidad de configuración porque ésa es la velocidad predeterminada utilizada por el software Arduino en el ordenador.

En la función **loop**, vamos a ver repetidamente si hemos enviado letras a la conexión USB y si tenemos que procesar la letra. La función de **Arduino Serial.available()** será verdadera si hay un carácter para convertir en código Morse y la función **Serial.read()** nos dará ese carácter, que asignaremos a una variable llamada **ch** que acabamos de definir dentro del **loop**.

Luego tenemos una serie de sentencias **if** (si..) que determinan si el carácter es una letra mayúscula, minúscula, o un carácter espacio de separación de dos palabras. Si miramos a la primera instrucción **if**, estamos comprobando si el valor del carácter es mayor o igual a "a" e inferior o igual a "z". Si es el caso, podemos buscar la secuencia de rayas y puntos de destello utilizando la matriz de letras (**letters[]**) que hemos definido en la parte superior del **sketch**.

Debemos determinar qué secuencia utilizar de la matriz restando "a" del carácter de **ch**. A primera vista, podría parecer extraño restar una letra de otra, pero es perfectamente aceptable hacer esto en C. Así, por ejemplo, "a" - "a" es igual a 0, mientras que "d" - "a" nos dará una respuesta de 3. Por lo tanto, si la letra que leemos en las conexiones USB fuera f, calculamos "f" - "a", lo que nos da 5 como la posición en el array **letters[]**. Si ahora miramos **letters[5]**, nos dará la cadena "...". A continuación pasamos esta cadena (string) a una función llamada **flashSequence**.

La función **flashSequence** lo que hace es recorrer cada una de las partes de la secuencia y realizar un destello en forma de raya o punto. En C, las **strings** (cadenas) tienen un código especial en el extremo que indica el final de la cadena, y esto se conoce como **NULL**. Por tanto, la primera cosa que hace **flashSequence** consiste en definir una variable llamada **"i"**. Esto indicará la posición actual en la cadena de puntos y rayas, comenzando en la posición 0. El bucle **while** seguirá ejecutándose hasta que llegue a **NULL**, al final de la cadena.

Dentro del bucle **while**, primero hacemos que destelle el punto o raya en el que nos encontramos utilizando una función que vamos a tratar en un momento y, a continuación, agregaremos 1 a "**i**" para, sin interrupción, continuar dando vueltas en el **loop**, haciendo parpadear cada punto o raya en su momento hasta que lleguemos al final de la cadena.

La función final que hemos definido es **flashDotOrDash**; esto sólo enciende el LED y luego utiliza una instrucción **if** para: o bien crear un retardo durante la duración de un solo punto (si el carácter es un punto), o un retardo de un período tres veces superior a esa duración (si el carácter es una raya), antes de volver a apagar el LED de nuevo.

Pongamos todo junto

Cargue el **sketch** terminado del Proyecto 3 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Para utilizar el traductor de código Morse, necesitamos emplear una parte del software Arduino llamado **Serial Monitor**. Esta ventana le permite escribir mensajes que se envían a la placa Arduino, así como ver los mensajes con que la placa Arduino

Hardware

El LED que utilizamos en el Proyecto 3 necesitaba unos 10 mA a 2 V. Podemos utilizar estos datos para calcular su potencia mediante la fórmula:

$$P = I \times V$$

La **Potencia** es igual a la tensión entre los extremos de algo multiplicado por la corriente que circula a través de ese algo, siendo la unidad de potencia el **Vatio (W)**. Por tanto, ese LED sería aproximadamente de 20 mW, o una cincuentaava parte de la potencia de nuestro LED de 1 W. Si bien Arduino no tiene problemas manejando un LED de 20 mW, no será capaz de manejar directamente un LED de 1 W.

Este es un problema común en electrónica, que se podría resumir de la siguiente forma: cómo conseguir que una corriente pequeña controle una corriente mayor, algo que se conoce como **amplificación**. El componente electrónico más utilizado para amplificar es el **transistor**, y es el que usaremos para encender y apagar nuestro LED Luxeon.

El funcionamiento básico del transistor se muestra en la Figura 3-4. Hay muchos tipos diferentes de transistores, y probablemente el más común y el tipo que vamos a utilizar se llama **transistor bipolar NPN**.

Este transistor tiene tres terminales llamados **emisor**, **colector** y **base**. El principio básico de funcionamiento es que una pequeña corriente circulando por la base permite que una corriente mucho mayor circule desde el emisor al colector.

Cuánto mayor será esta relación de corrientes depende de cada transistor, pero normalmente es un factor de 100. Por lo tanto, una corriente de 10 mA circulando por la base podría hacer circular hasta 1 A por el colector. Por tanto, si tenemos la resistencia de $270\ \Omega$ que hemos usado para manejar el LED a 10 mA, podemos esperar que sea más que suficiente para que el transistor commute los 350 mA necesarios para el LED Luxeon.

El esquema electrónico de nuestro circuito de control se muestra en la Figura 3-5.

La resistencia de $270\ \Omega$ (R_1) limita la corriente que circula por la base. Podemos calcular la corriente con la fórmula $I = V/R$. V serán 4.4 V en lugar de 5 V porque los transistores tienen normalmente una tensión de 0.6 V entre la base y el emisor, y el voltaje más alto que puede suministrar la Arduino desde el pin de salida es de 5 V. Por tanto, la corriente será $4.4/270 = 16\text{ mA}$.

R_2 limita la corriente que circula por el LED a alrededor de 350 mA. Hemos llegado a la cifra de $4\ \Omega$ utilizando la fórmula $R = V/I$. V será aproximadamente $5 - 3 - 0.6 = 1.4\text{ V}$. 5 V es la tensión de ali-

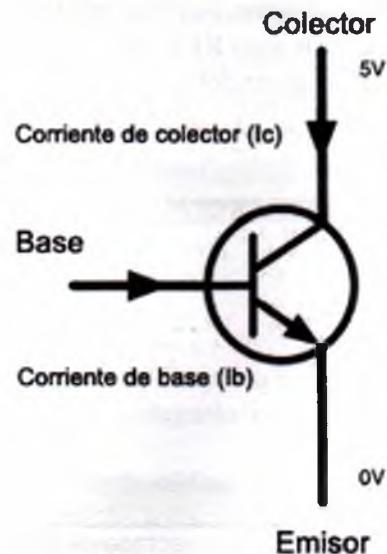
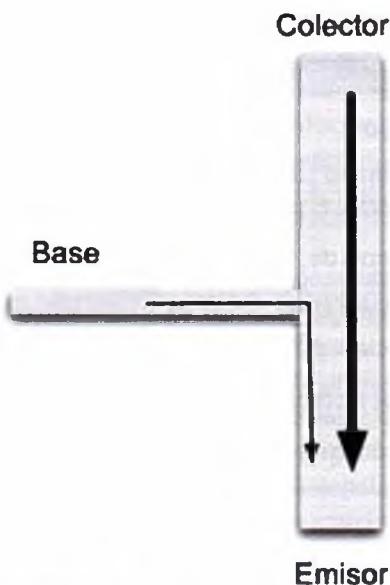


Figura 3-4 Funcionamiento de un transistor bipolar NPN

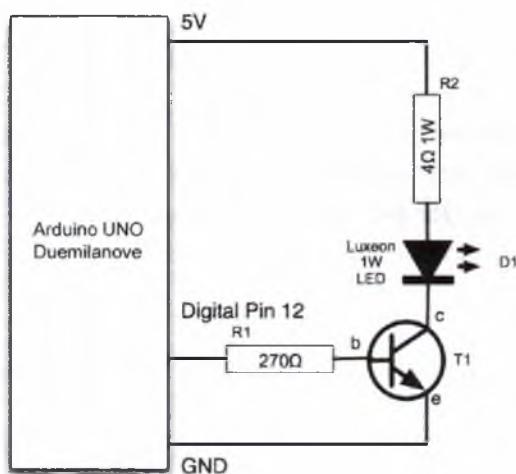


Figura 3-5 Diagrama esquemático del manejo de un LED de gran potencia.

mentación, el LED reduce aproximadamente 3 V y el transistor 0.6 V, por lo que la resistencia debe ser $1.4 \text{ V} / 350 \text{ mA} = 4 \Omega$. También debemos utilizar una resistencia que pueda hacer frente a esta corriente relativamente alta. La energía que la resistencia disipará como calor es igual a la tensión entre sus terminales multiplicado por la corriente que circula por ella. En este caso, es de $350 \text{ mA} \times 1.4 \text{ V}$, lo que da 490 mW. Para estar seguros, hemos seleccionado una resistencia de 1 W.

De la misma manera, cuando se escoge un transistor, tenemos que asegurarnos de que puede soportar la potencia que va a manejar. Cuando se activa, el transistor consumirá energía igual a la corriente multipli-

cada por el voltaje. En este caso, la corriente de base es lo suficientemente pequeña como para ignorarla, por lo que, la potencia será tan sólo $0.6 \text{ V} \times 350 \text{ mA}$, o 210 mW. Siempre es aconsejable escoger un transistor que pueda disipar holgadamente la potencia que necesitamos. En este caso, vamos a utilizar un BD139 que tiene una potencia superior a 12 W. En el Capítulo 10 puede encontrar una tabla con los transistores utilizados con más frecuencia.

Ahora necesitamos colocar los componentes en la placa **protoboard** siguiendo la disposición mostrada en la Figura 3-6, con la correspondiente de la Figura 3-8. Es fundamental identificar correctamente los terminales del transistor y del LED. El lado metálico del transistor debe mirar hacia la placa. La pata más larga del LED se corresponde con el terminal positivo.

Más adelante en este proyecto vamos a mostrar cómo se puede trasladar el proyecto desde la placa de pruebas a un diseño más permanente utilizando **Arduino Protoshield**. Esto requiere utilizar un soldador, así es que, si desea continuar para crear una **shield** y cuenta con el equipo para soldar, debería soldarle los cables al LED Luxeon. Suelde cables cortos de hilo rígido a los terminales marcados con + y -. Es una buena idea utilizar cable rojo para el terminal positivo y azul o negro para el negativo.

Si no desea soldar, no importa; lo único que tiene que hacer es enrollar con cuidado el cable alrededor de los conectores, como se muestra en la Figura 3-7.

La Figura 3-8 muestra el montaje completo de la placa de pruebas.

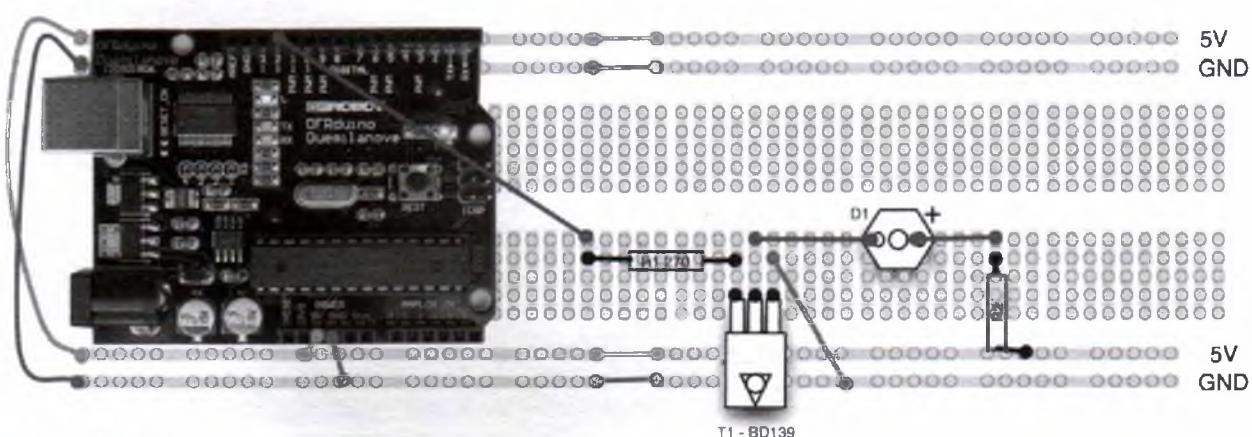


Figura 3-6 Diseño de la placa de pruebas del Proyecto 4.

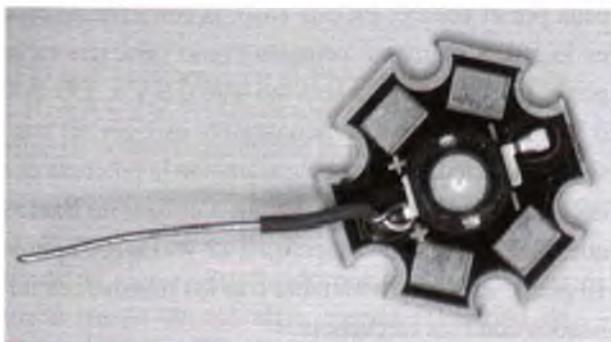


Figura 3-7 Conexión de cables sin soldadura en el LED Luxeon

Software

El único cambio en el software del Proyecto 3 es que estamos utilizando el pin de salida digital 11 en lugar del pin 12.

Pongamos todo junto

Cargue el **sketch** terminado del Proyecto 4 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

De nuevo, la prueba del proyecto es la misma que la del Proyecto 3. Necesitará abrir la ventana **Serial Monitor** y empezar a escribir.

El LED tiene un ángulo de emisión muy amplio, por lo que una variación de este proyecto podría ser adaptar el reflector de una linterna de LED para concentrar el haz.

Crear una shield

Este es el primer proyecto que hemos hecho que tiene suficientes componentes para justificar la creación de una placa de circuito impreso **Arduino Shield** para colocarla sobre la propia placa de Arduino. También vamos a usar este hardware con ligeras modificaciones en el Proyecto 6, por lo que quizás es el momento de crear una **shield LED Luxeon**.

Sin duda, en casa se pueden hacer sus propias placas de circuito impreso, pero esto requiere el uso de productos químicos nocivos y una cierta cantidad de equipo. Afortunadamente, hay otro gran componente de Arduino de hardware libre (open source) llamado Arduino **Protoshield**. Si echa un vistazo en las tiendas de alrededor, verá que se pueden obtener por 10 € o menos y le proporcionarán un kit con todo lo necesario para crear una **shield**. Esto incluye la propia placa, los conectores con los pines que se colocan en la Arduino, y algunos LEDs, pulsadores y resistencias. En cualquier caso, tenga en cuenta que existen distintas variantes de la placa **Protoshield**, por lo que, si su placa es ligeramente diferente, deberá adaptar el diseño que mostramos aquí.

Los componentes de una **Protoshield** se muestran en la Figura 3-9, siendo la parte más importante la placa de circuito impreso **Protoshield** (PCB). También se puede comprar sólo la placa de circuito

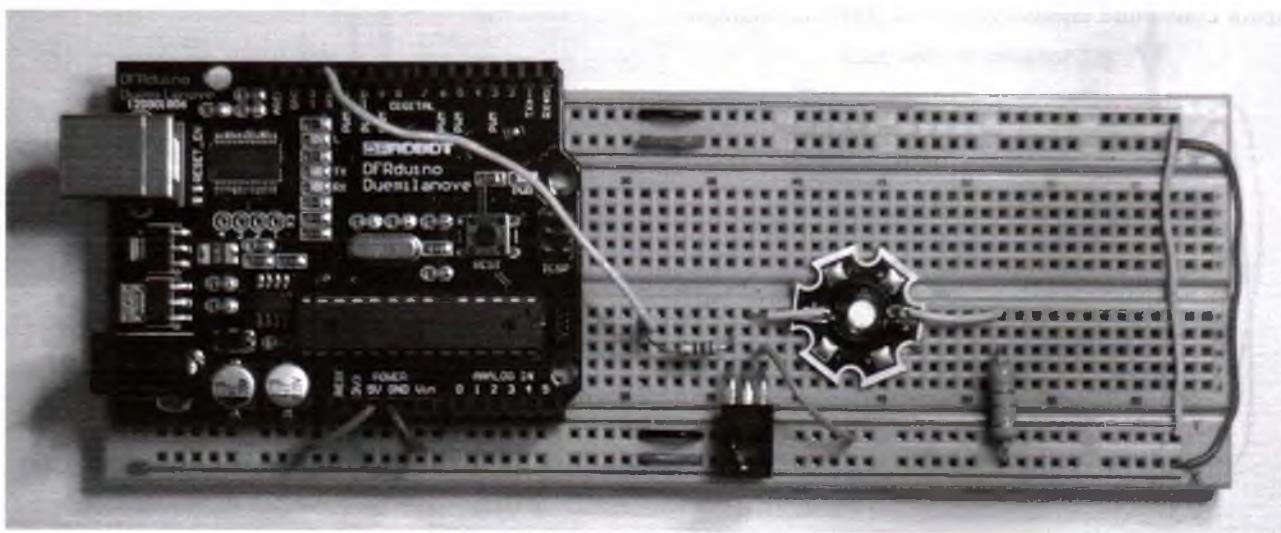


Figura 3-8 Fotografía de una placa de pruebas completa para el Proyecto 4.

Protoshield, que para muchos de los proyectos será todo lo que necesite.

En la placa no vamos a soldar todos los componentes que vienen con el kit. Vamos a limitarnos a agregar el LED de la alimentación, su resistencia, y sólo los pines de la parte inferior que se conectan a la placa Arduino, ya que no tendrá ninguna otra **shield** encima.

Una buena costumbre a la hora de montar los circuitos es soldar primero en su sitio los componentes que están más abajo. Así es que, en este caso, soldaremos las resistencias, el LED, el pulsador de reinicio (**reset**) y, a continuación, los conectores de los pines inferiores.

La resistencia de 1 K, el LED, y el pulsador se introducen desde la parte superior de la placa y se sueldan por debajo (Figura 3-10). La parte corta de los pines del conector se colocarán desde debajo de la placa y se soldarán en la parte superior.

Cuando suelde las patillas del conector, asegúrese de que estén alineadas correctamente, ya que hay dos filas paralelas para los conectores: una para la conexión a los pines que van por debajo y otra para los zócalos que no estamos utilizando, que están pensados para conectarse a otras **shields**.

Un buen método para garantizar que los conectores están en su sitio es conectarlos en una placa Arduino y, a continuación, colocar la placa **shield** encima y soldar los pines, lo que asegura que éstos queden rectos.

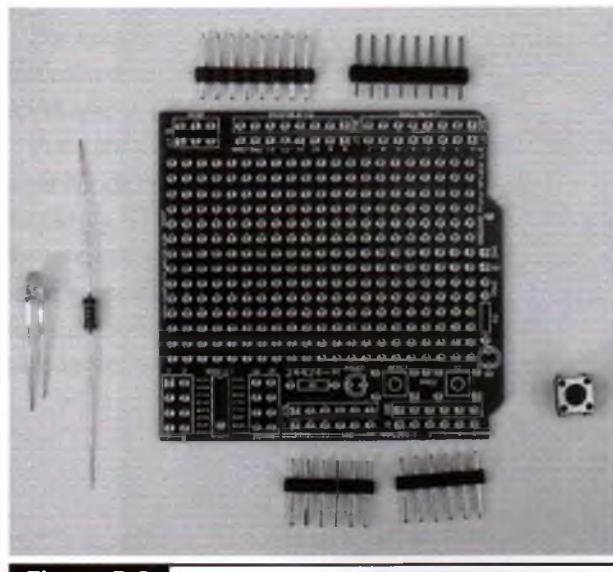


Figura 3-9 Protoshield en forma de kit.

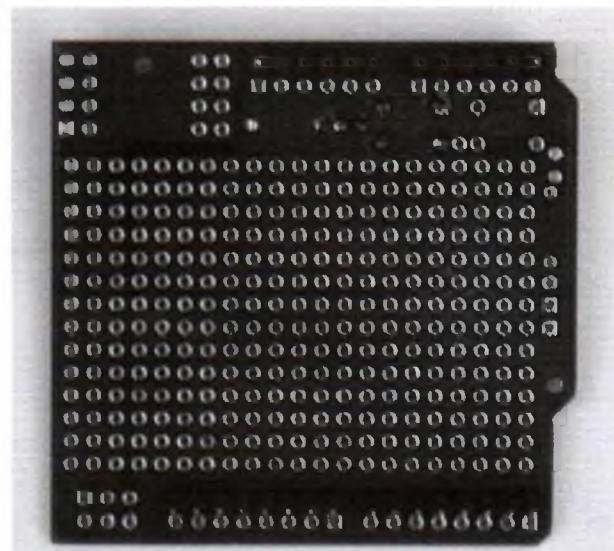


Figura 3-10 La parte inferior de la Protoshield.

Cuando haya soldado en su sitio los componentes del kit, debería tener una placa parecida a la que aparece en la Figura 3-11.

Ahora podemos añadir los componentes específicos de este proyecto que hemos estado usando en la placa **protoboard**. Primero, coloque todos los componentes en su sitio siguiendo la disposición de la Figura 3-12 para asegurarnos de que todo encaja en el lugar que le corresponde.

Este tipo de placa es de doble cara, es decir, se puede soldar por la parte superior o inferior de la placa. Como se puede ver en la Figura 3-12, algunas de las conexiones están dispuestas en forma de pistas conductoras que conectan varios puntos, igual que ocurre en nuestra placa de pruebas **proto-board**.

Vamos a montar todos los componentes en la parte superior, con las patillas introducidas en su sitio y soldadas en la parte inferior, por donde sobresalen de la placa. Las patas de estos componentes se conectan por debajo y luego se cortan los trozos que sobren. En caso necesario, se pueden utilizar trozos de cable rígido para alargar las patas de alguno de los componentes.

La Figura 3-13 muestra el montaje completo de la **shield**. Conecte la placa y pruébela. Si no funciona en el mismo momento de conectarla, desconéctela de inmediato de la alimentación y utilice un polímetro para comprobar cuidadosamente que no

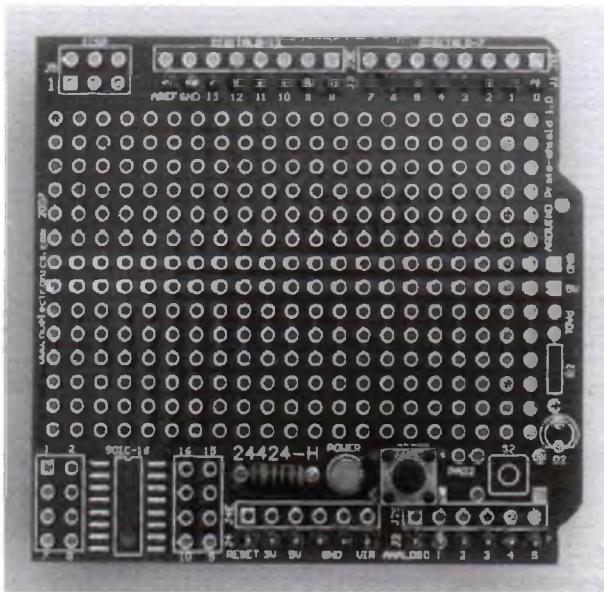


Figura 3-11 Montaje de Protoshield básico.

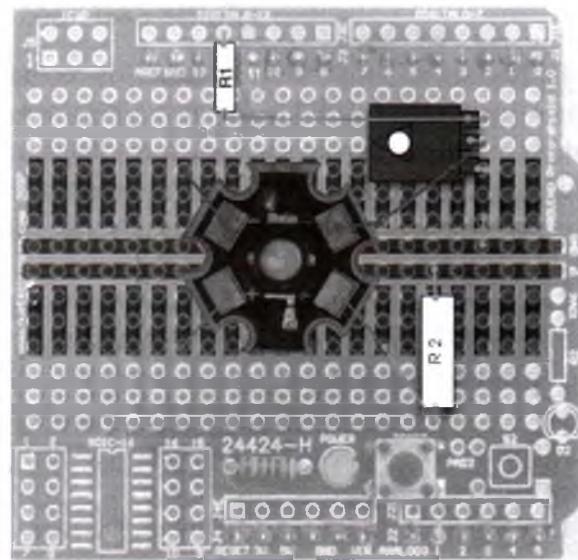


Figura 3-12 Diseño de Protoshield del Proyecto 4.

haya cortocircuitos o conexiones rotas en la shield.

¡Felicitaciones! Acaba de crear su primera shield de Arduino, y además la podremos reutilizar en proyectos posteriores.

Resumen

¡Bueno! Hemos dado un primer paso con algunos proyectos simples con LEDs y hemos descubierto cómo utilizar los LED Luxeon de alta potencia.

También hemos aprendido un poco más sobre cómo programar la placa Arduino en C.

En el siguiente capítulo, vamos a ampliar esto viendo más proyectos basados en LED, como un modelo de semáforo y una luz estroboscópica de gran potencia.

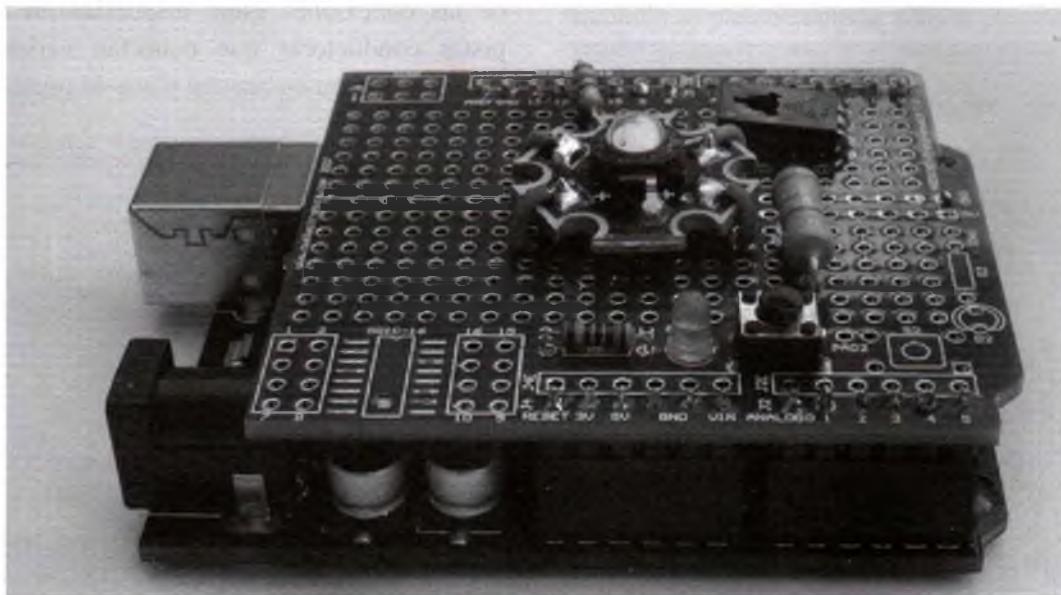


Figura 3-13 Shield Luxeon completa conectada a una placa Arduino.

CAPÍTULO 4

Más proyectos con LEDs

EN ESTE CAPÍTULO, vamos a seguir trabajando con esos pequeños y versátiles componentes, los LEDs, y vamos a aprender un poco más sobre las entradas y salidas digitales, incluyendo cómo utilizar los pulsadores.

Los proyectos que vamos a construir en este capítulo son un semáforo, dos proyectos de luz estroboscópica, y un módulo de potente iluminación que utiliza LEDs Luxeon de alta potencia.

Entradas y salidas digitales

Los pines digitales 0 a 12 pueden configurarse para ser utilizados como entrada o como salida. Esta asignación la determinamos en el **sketch**. Puesto que vamos a conectar componentes electrónicos en estos pines, es improbable que una vez establecidos queramos volver a cambiarlos. Es decir, una vez que hayamos configurado un pin como salida, no lo vamos a cambiar en mitad de un **sketch** para que sea una entrada.

Por esta razón, suele ser práctica habitual fijar la dirección de un pin digital en la función de configuración que debe definirse en cada **sketch**.

Por ejemplo, el siguiente código (a continuación de `pinMode`) establece el pin digital 10 como salida (`output`) y el pin digital 11 como entrada (`input`). Observe cómo utilizamos esta declaración de variables en nuestro **sketch** para hacer que resulte más fácil cambiar posteriormente el pin utilizado si así nos conviene más adelante.

```
int ledPin = 10;
int switchPin = 11;

void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(switchPin, INPUT);
}
```

Proyecto 5 Modelo de semáforo

Ahora que ya sabemos cómo establecer un pin digital para que actúe como entrada, podemos construir un proyecto de modelo de semáforo con la utilización de LEDs rojo, amarillo y verde. Cada vez que pulsemos el botón, la luz del semáforo cambiará al color siguiente de la secuencia prevista. En el Reino Unido, la secuencia de los semáforos es: rojo, rojo y amarillo juntas, verde, amarillo y, a continuación, vuelta al rojo.

Además de lo anterior, si en nuestro proyecto mantenemos presionado el botón, las luces cambiarán solas siguiendo la secuencia prevista, aunque con un cierto retardo entre cada paso.

Los componentes para el Proyecto 5 se muestran a continuación. Cuando trabaje con LEDs asegúrese de que éstos tengan la misma luminosidad para que los resultados sean óptimos.

COMPONENTES Y EQUIPO		
	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o similar	1
D1	LED rojo de 5 mm	23
D2	LED amarillo de 5 mm	24
D3	LED verde de 5 mm	25
R1-R3	Resistencia 270 Ω 0,5 W	6
R4	Resistencia 100 KΩ 0,5 W	13
S1	Pulsador en miniatura para conmutación	48

Hardware

El esquema electrónico del proyecto se muestra en la Figura 4-1.

Los LEDs están conectados de la misma manera que en nuestro proyecto anterior, cada uno con su resistencia limitadora de corriente. El pin digital 5 está puesto a masa (GND) a través de R4, y cuando pulsamos el interruptor, éste pasa a 5 V.

En la Figura 4-2 se muestra una fotografía del proyecto y, en la Figura 4-3, el diseño de la placa.

Software

El **sketch** del proyecto se muestra en el Listado del Proyecto 5.

El **sketch** es bastante auto explicativo. Una vez por segundo, el programa comprueba si se ha presionado el pulsador, de forma que si lo pulsamos varias veces con rapidez no cambiará la secuencia de luces. Sin embargo, si mantenemos presionado el pulsador, las luces harán la secuencia completa automáticamente.

El **sketch** también utiliza una función independiente, **setLights**, para establecer el estado de cada LED, reduciendo así tres líneas de código a una.

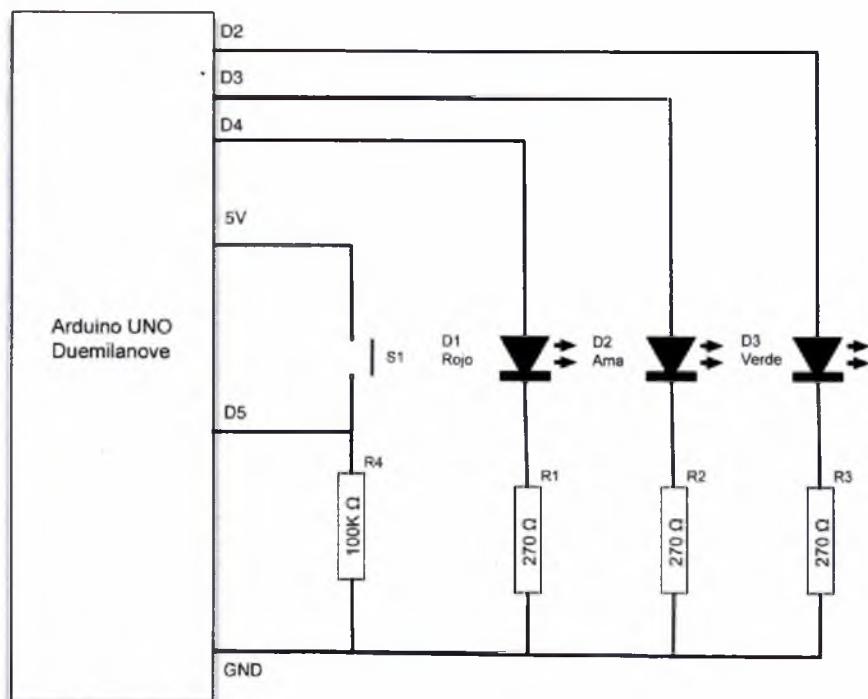


Figura 4-1 Diagrama esquemático del Proyecto 5.

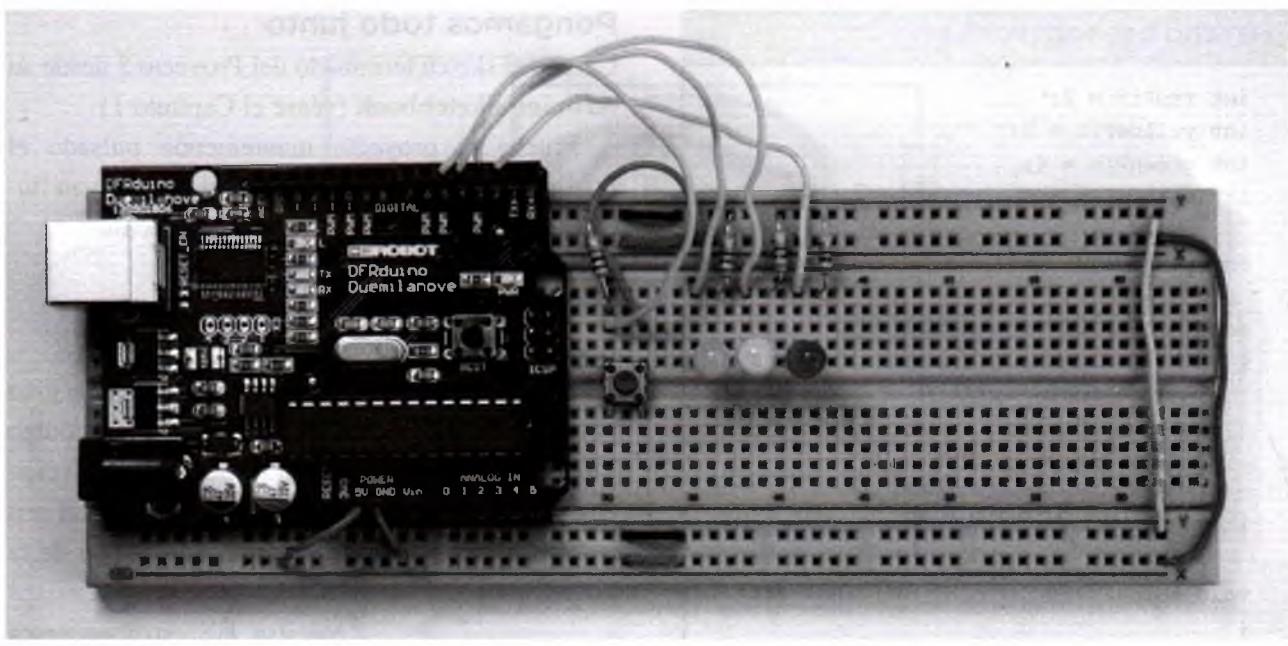


Figura 4-2 Diseño de la placa de pruebas del Proyecto 5. Un modelo de señales de tráfico.

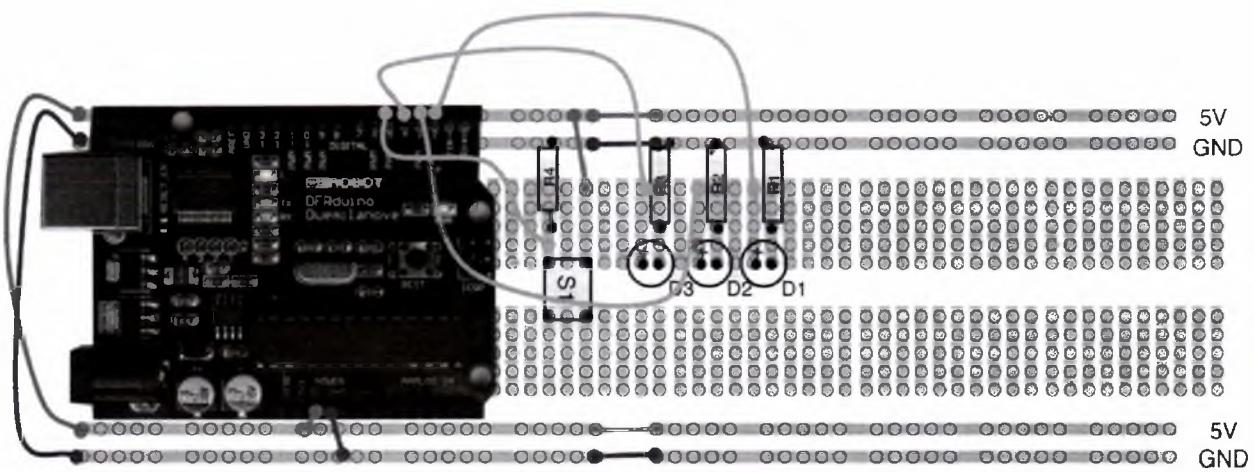


Figura 4-3 Diagrama de la placa de pruebas del Proyecto 5.

LISTADO DEL PROYECTO 5

```

int redPin = 2;
int yellowPin = 3;
int greenPin = 4;
int buttonPin = 5;

int state = 0;

void setup()
{
    pinMode(redPin, OUTPUT);
    pinMode(yellowPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop()
{
    if (digitalRead(buttonPin))
    {
        if (state == 0)
        {
            setLights(HIGH, LOW, LOW);
            state = 1;
        }
        else if (state == 1)
        {
            setLights(HIGH, HIGH, LOW);
            state = 2;
        }
        else if (state == 2)
        {
            setLights(LOW, LOW, HIGH);
            state = 3;
        }
        else if (state == 3)
        {
            setLights(LOW, HIGH, LOW);
            state = 0;
        }
        delay(1000);
    }

    void setLights(int red, int yellow, int
                  green)
    {
        digitalWrite(redPin, red);
        digitalWrite(yellowPin, yellow);
        digitalWrite(greenPin, green);
    }
}

```

Pongamos todo junto

Cargue el sketch terminado del Proyecto 5 desde su **Arduino Sketchbook** (véase el Capítulo 1).

Pruébe el proyecto manteniendo pulsado el botón y compruebe que todos los LEDs se van iluminando siguiendo la secuencia establecida.

Proyecto 6
Luz Estroboscópica

Este proyecto utiliza el mismo LED Luxeon de gran potencia que utilizamos en el traductor de código Morse. En este caso añadimos además una resistencia variable llamada potenciómetro, la cual nos permite el control sobre el ritmo de destello de la luz estroboscópica.

PRECAUCIÓN

Nuestra luz estroboscópica destella con gran intensidad luminosa. Si tiene algún problema de salud, como por ejemplo, epilepsia, puede que sea mejor saltarse este proyecto.

COMPONENTES Y EQUIPO

Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
D1 LED Luxon 1W	30
R1 Resistencia 270 Ω 0,5 W	6
R2 Resistencia 4 Ω 1 W	16
T1 Transistor potencia BD139	41
R3 Potenciómetro lineal 100K	17
Kit protoshield (opcional)	3
Enchufe eléctrico 2,1 mm (opc.)	49
Clip de batería 9 V (opcional)	50

Hardware

El hardware de este proyecto es básicamente el mismo que para el Proyecto 4, sino que a éste le hemos añadido un potenciómetro (véase la Figura 4-4).

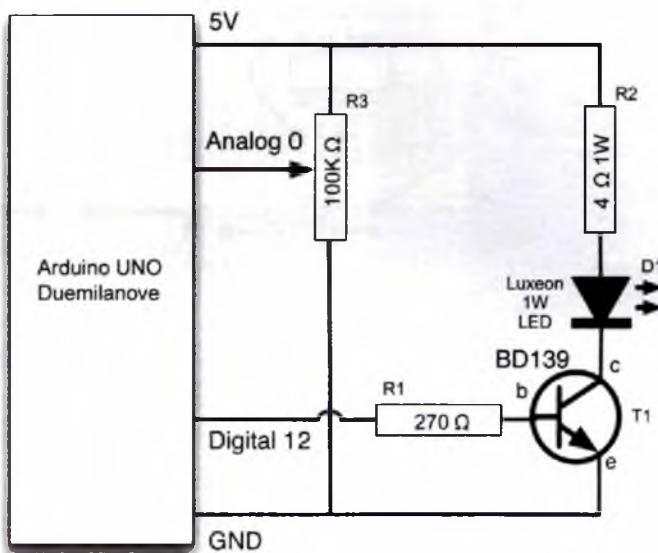


Figura 4-4 Esquema electrónico del Proyecto 6.

La placa Arduino está equipada con seis pines analógicos de entrada, numerados **Analog 0** a **Analog 5**. En ellos se mide el nivel de tensión en la entrada, dando como resultado un número entre 0 (0 V) y 1023 (5 V).

Esta característica la podemos aprovechar para conocer cuál es la posición del mando de un potenciómetro de manera que al variar su resistencia este actúe como un divisor de tensión para nuestro pin analógico. La Figura 4-5 muestra la estructura interna de un potenciómetro.

El potenciómetro es un componente electrónico que se utiliza habitualmente para el control de volumen. Está constituido por una pista circular de una cierta resistencia equipada con conexiones en ambos extremos. Un cursor deslizante proporciona una tercera conexión móvil.

La resistencia ajustable del potenciómetro nos puede proporcionar un voltaje variable al conectar uno de sus extremos a 0 V, y el otro a 5 V, con lo que el nivel de tensión en el cursor deslizante puede variar entre 0 y 5 V, según se gire el mando.

Como puede comprobarse, el diseño de la placa de pruebas (Figura 4-6) es similar al del Proyecto 4.

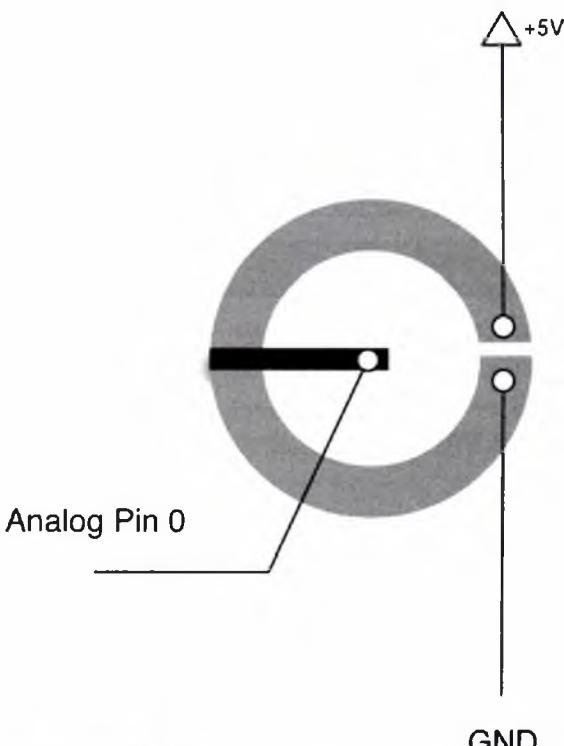


Figura 4-5 Funcionamiento interno de un potenciómetro.

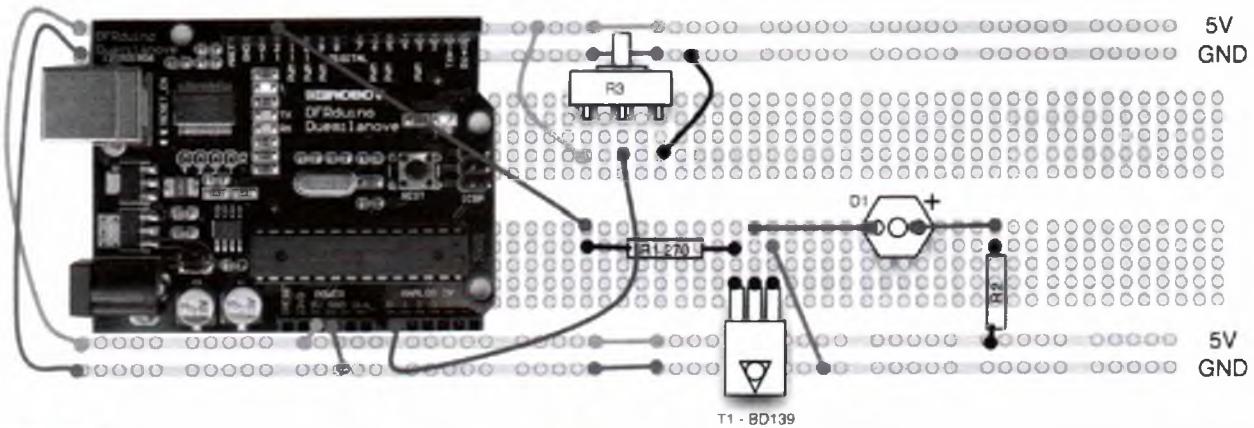


Figura 4-6 Diagrama de la placa de pruebas del Proyecto 6.

Software

El listado de código de este proyecto se muestra a continuación. La parte más interesante es la relacionada con la lectura de los valores de la entrada analógica y con el control del ritmo de destello.

En el caso de los pines analógicos no es necesario utilizar la función **pinMode**, por lo que no hay que añadir nada más a la función de configuración (**setup**).

Con los valores mostrados podemos variar la velocidad de destello entre 1 y 20 veces por

LISTADO DEL PROYECTO 6

```

int ledPin = 11;
int analogPin = 0;

void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    int period = (1023 -
        analogRead(analogPin)) / 2 + 25;
    digitalWrite(ledPin, HIGH);
    delay(period);
    digitalWrite(ledPin, LOW);
    delay(period);
}

```

segundo; y los tiempos entre el encendido y el apagado del LED serán de 500 y 25 milisegundos respectivamente.

Por lo tanto, si nuestra entrada analógica cambia de 0 a 1023, el cálculo que necesitamos para determinar el retardo de destello es aproximadamente:

```

flash_delay = (1023 - analog_value) / 2
+ 25

```

Como consecuencia, un **analog_value** de 0 daría un valor de **flash_delay** de 561, y un **analog_value** de 1023 daría un retardo de 25. En realidad, habría que dividir por un poco más de 2, pero los cálculos resultan más sencillos empleando solo números enteros.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 6 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Comprobará que girando el potenciómetro en el sentido de las agujas del reloj aumentará la velocidad de destello a medida que se incremente el voltaje en la entrada analógica. Si lo giramos en el sentido contrario disminuirá la velocidad de destello.

Montaje de una shield

Si desea hacer una **shield** para este proyecto, puede adaptar la que hicimos en el Proyecto 4 o empezar una nueva desde cero.

La disposición de componentes de la placa **Protoshield** se muestra en la Figura 4-7.

Ésta es básicamente la misma que para el Proyecto 4, excepto que hemos añadido el potenciómetro. Dado que los terminales de los potenciómetros son demasiado gruesos como para que entren en los agujeros de la **Protoshield**, lo mejor es conectarlos utilizando trozos de cables o, como hemos hecho aquí, soldando con cuidado hilos rígidos en sus terminales. Para proporcionar cierta resistencia mecánica al potenciómetro puede pegar éste en su sitio con una gota de pegamento Super Glue. El conexionado del potenciómetro a los terminales de **5 V**, **GND** y **Analog 0** se puede hacer por la parte de abajo de la placa, para que no se vea.

Tras haber realizado la shield, podemos independizar ésta del ordenador alimentándola con una pila o batería de 9 V.

Para alimentar el proyecto desde una pila, tenemos que hacernos un pequeño cable que tenga un clip de conexión del tipo PP3 (el utilizado normalmente con las pilas de 9 V) en un extremo y una clavija de alimentación de 2,1 mm en el otro. La figura 4-8 muestra el cable semi-montado.

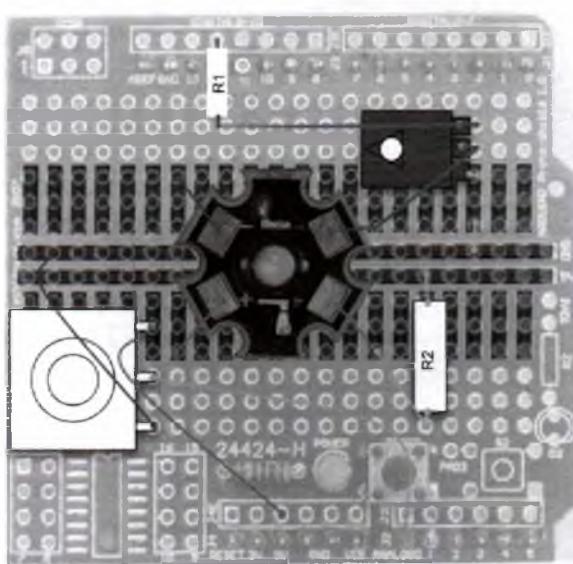


Figura 4-7 Diseño de la Protoshield para el Proyecto 6.



Figura 4-8 Montaje de un cable para alimentación externa con pila.

Proyecto 7 Luz para el T.A.E.

El Trastorno Afectivo Estacional (TAE) afecta a un gran número de personas; las investigaciones han demostrado que la exposición durante 10 o 20 minutos al día a una luz blanca brillante que imite la luz del día tiene efectos beneficiosos. Para utilizar este proyecto para tal fin, sugiero el uso de algún tipo de difusor, como el vidrio esmerilado, ya que no se debe mirar directamente a la luz de los LEDs.

Este es otro proyecto basado en los LEDs Luxeon de alto brillo. Utilizaremos una entrada analógica conectada a un potenciómetro que actúe como control de tiempo, encendiéndo el LED durante un determinado período fijado por la posición del cursor del potenciómetro. También utilizaremos una salida analógica para aumentar lentamente el brillo de los LEDs cuando se enciendan y luego bajarlo cuando se estén apagando. Para conseguir que la luz sea lo suficientemente brillante para que sea útil como luz para el TAE, vamos a utilizar seis LED Luxeon.

Puede que la naturaleza “protectora” de este proyecto esté causando a las mentes más rebeldes algo parecido a una crisis de identidad. Pero no, no tengan miedo: en el Proyecto 8 vamos a convertir este mismo hardware en una temible luz estroboscópica de alta potencia.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
D1-6 LED Luxon 1W	30
R1-3 Resistencia 1 KΩ 0,5 W	7
R4-5 Resistencia 4 Ω 2 W	16
R6 Potenciómetro lineal 100K	17
IC1-2 Regulador voltaje LM317	45
T1-2 FET 2N7000	42
Fuente alimentación regulada 15 V 1 A	51
Tarjeta perforada	53
Terminal de tornillo de tres vías	52

■ Nota: este es uno de los proyectos de este libro que requieren soldar.

■ Va a necesitar seis LED Luxeon para este proyecto. Si desea ahorrar algo de dinero, eche un vistazo en las tiendas online, donde debe poder conseguir 10 de estos LEDs por 10€ a 20€.

Hardware

Algunos de los pines digitales, y en concreto los números 5, 6, 9, 10 y 11, pueden proporcionar una salida variable en lugar de sólo 5 V o nada. Estos son los pines en los que aparece **PWM** junto a ellos en la placa. Esta es la razón por la que hemos empezado a usar el pin 11 para nuestro control de salida.

PWM son las siglas de **Pulse Width Modulation** (modulación por anchura de pulsos), y se refiere a la forma de controlar la cantidad de energía en la salida. Este control se hace activando y desactivando rápidamente la salida.

Los pulsos se realizan siempre al mismo ritmo (aproximadamente 500 por segundo), siendo la longitud de los mismos lo que varía. Si los pulsos son largos, nuestro LED estará encendido todo el tiempo. Sin embargo, si los pulsos son cortos, en realidad el

LED se enciende sólo durante un corto espacio de tiempo. Esto ocurre tan rápido que el observador no percibe que el LED en realidad está parpadeando, y solo cree observar que el LED está más o menos brillante.

Quizá los lectores quieran echar un vistazo en Wikipedia a una descripción más detallada de PWM.

El valor de la salida se puede ajustar utilizando la función **analogWrite**, que requiere un valor de salida entre 0 y 255, donde 0 indicará apagado y 255 plena potencia.

Como se puede ver en el esquema electrónico de la Figura 4-9, los LEDs están ordenados en dos columnas de a tres. Éstos se alimentan desde una fuente externa de 15 V en lugar de la alimentación de 5 V que hemos usado anteriormente. Puesto que cada LED consume alrededor de 300 mA, cada columna necesitará 300 mA, por lo que la alimentación debe poder proporcionar 0,6 A (1 A para mayor seguridad).

Este es el circuito más complejo de los realizados hasta ahora en nuestros proyectos. Estamos utilizando dos circuitos integrados que incorporan un regulador variable de voltaje, con objeto de limitar la corriente que llega a los LED. La salida de los reguladores de voltaje será normalmente de 1,25 V por encima del voltaje del pin **Ref** del chip. Esto significa que si manejamos nuestros LED con una resistencia de 4 Ω, habrá una corriente de aproximadamente $I = V/R$, ó $1,25/4 = 312$ mA circulando por ellos.

El **FET** (transistor de efecto campo) es similar a nuestro transistor bipolar normal en el sentido de que al igual que éste puede actuar como un interruptor, sino que la resistencia que presenta una vez desconectado es mucho más alta. Por lo tanto, cuando no está activado por algún voltaje en su puerta, es como si no existiera en el circuito. Sin embargo, cuando se activa, hará bajar el voltaje del pin **Ref** del regulador a una tensión lo suficientemente baja como para evitar que circule cualquier corriente por los LED, provocando su apagado. Ambos **FET** se controlan desde el mismo pin digital 11.

El montaje completo del circuito de LED se muestra en la Figura 4-10 y el diseño de la placa perforada, en la Figura 4-11.

El circuito se ha montado sobre una **placa perforada**, que es simplemente una placa con agujeros, sin ningún tipo de conexiones. Este tipo de placas sirve

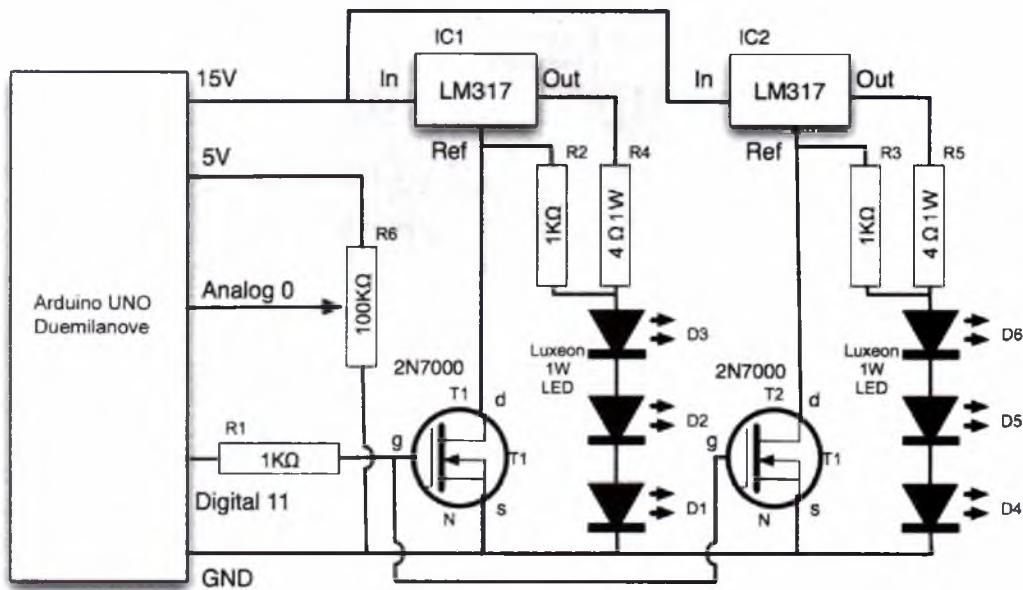


Figura 4-9 Esquema eléctrico del Proyecto 7.

de estructura para insertar los componentes, siendo necesario cablearlas por la parte de abajo, ya sea empalmando juntos los terminales o mediante cables.

Lo más sencillo es soldar dos hilos en cada LED antes de colocarlos en la placa. Es una buena idea marcar los terminales del LED mediante colores:

cable rojo para el positivo y negro o azul para el negativo, para que queden identificados de forma correcta.

Dado que los LEDs se calientan, es recomendable dejar un espacio entre ellos y la **placa perforada**, utilizando el aislamiento de los cables para que haga de separador. El regulador de tensión también se

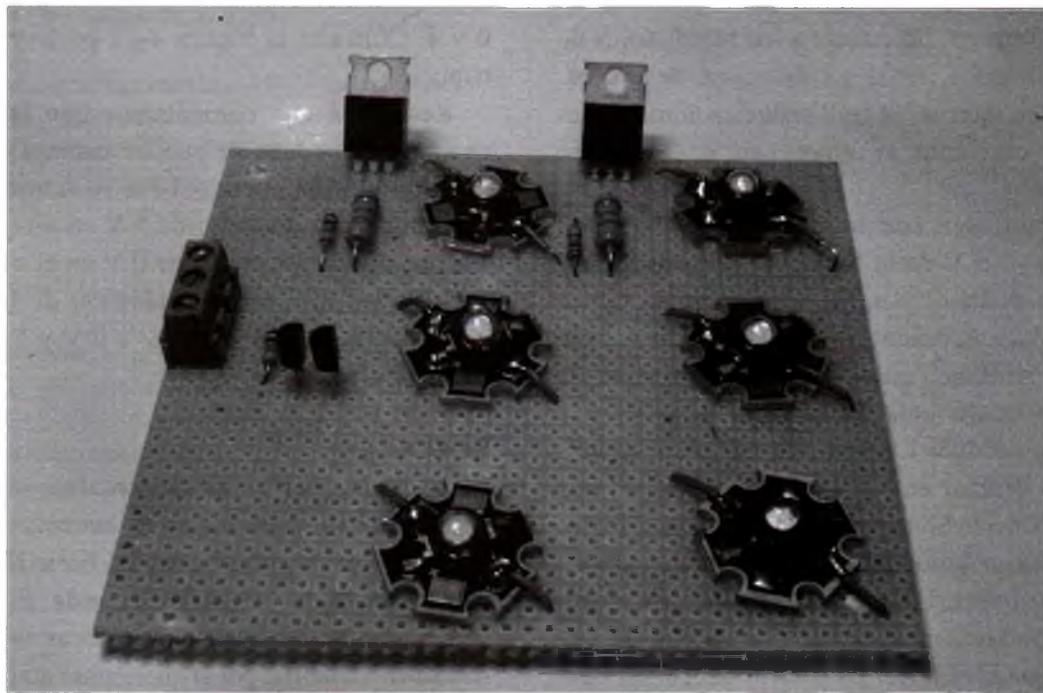


Figura 4-10 Proyecto 7. Montaje de luz de gran potencia.

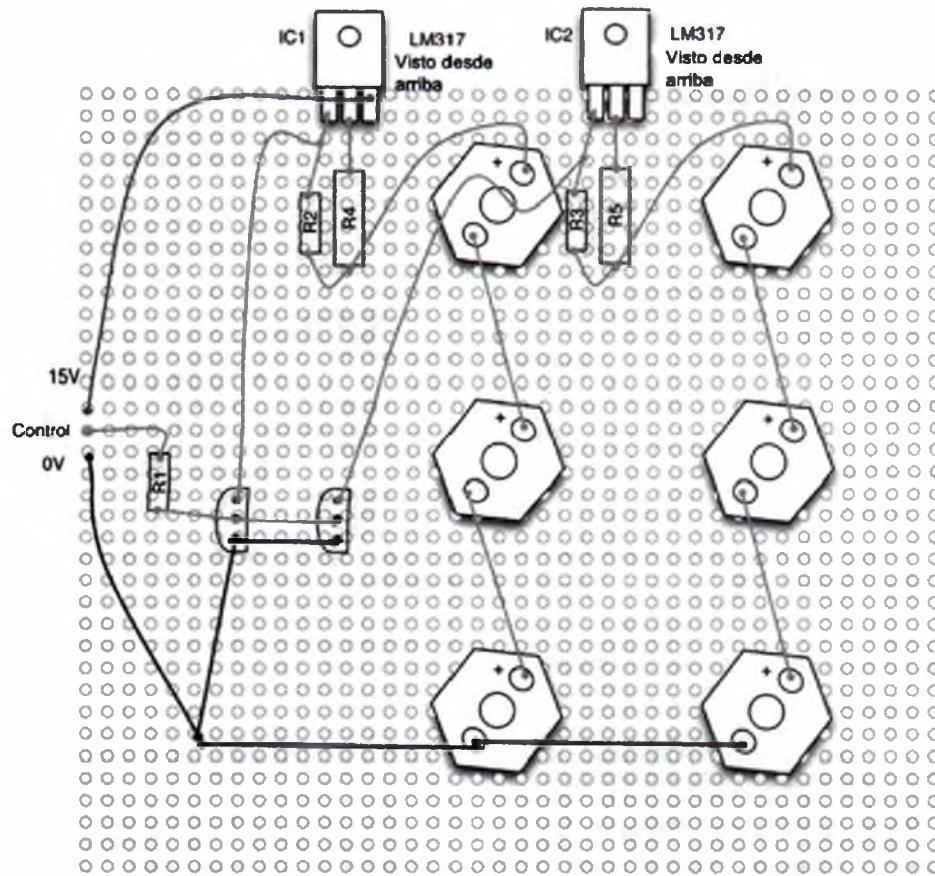


Figura 4-11 Distribución de la placa perforada.

caliente, pero no tanto como para necesitar un dissipador térmico. En cuanto a los reguladores de voltaje utilizados, éstos ya disponen de protección térmica interna, la cual reducirá automáticamente la corriente si empiezan a calentarse demasiado.

Los terminales con tornillo de la placa son para GND y 15 V de la fuente de alimentación y para una entrada de control. Cuando conectamos ésta a la placa Arduino, los 15 V vendrán del pin Vin de la Arduino, que a su vez se alimenta de una fuente de alimentación externa de 15 V.

Nuestro módulo LED de alta potencia lo volveremos a utilizar en otros proyectos, por lo que vamos a conectar directamente el potenciómetro en el conector **Analog In** (entrada analógica) de la placa Arduino. La separación entre los terminales del potenciómetro es de unos 5 mm, lo que significa que si el terminal central del cursor deslizante se encuentra en el pin **Analog 2**, los otros

dos terminales coincidirán con los pines **Analog 0** y **4**. Consulte la Figura 4-12 para ver esta distribución.

Recordará que comentamos que las entradas analógicas también se pueden utilizar como salidas digitales añadiendo 14 a su número de pin. Por tanto, para disponer de 5 V en un extremo de nuestro potenciómetro y de 0 V en el otro, vamos a establecer las salidas analógicas de los pines 0 y 4 (pines digitales 14 y 18) a 0 V y 5 V, respectivamente.

Software

En la parte superior del **sketch**, después de la variable utilizada para los pines, tenemos otras cuatro variables: **startupSeconds**, **turnOffSeconds**, **minOnSeconds** y **maxOnSeconds**. Es una práctica común en programación colocar estos valores dentro de variables, por si queremos cambiarlos posteriormente, y situarlas para que sean visibles en la

LISTADO DE PROYECTO 7

```
int ledPin = 11;
int analogPin = 2;

int startupSeconds = 20;
int turnOffSeconds = 10;
int minOnSeconds = 300;
int maxOnSeconds = 1800;

int brightness = 0;

void setup()
{
    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, HIGH);
    pinMode(14, OUTPUT);                      // Use pines Analog 0 y 4 para
    pinMode(18, OUTPUT);                      // el potenciómetro
    digitalWrite(18, HIGH);
    int analogIn = analogRead(analogPin);
    int onTime = map(analogIn, 0, 1023, minOnSeconds, maxOnSeconds);
    turnOn();
    delay(onTime * 1000);
    turnOff();
}

void turnOn()
{
    brightness = 0;
    int period = startupSeconds * 1000 / 256;
    while (brightness < 255)
    {
        analogWrite(ledPin, 255 - brightness);
        delay(period);
        brightness++;
    }
}

void turnOff()
{
    int period = turnOffSeconds * 1000 / 256;
    while (brightness >= 0)
    {
        analogWrite(ledPin, 255 - brightness);
        delay(period);
        brightness--;
    }
}

void loop()
{}
```

parte superior del programa, para que resulte más sencillo cambiarlas.

La variable **startupSeconds** determina cuánto tiempo se necesita para que el brillo del LED aumente gradualmente hasta alcanzar el máximo. Del mismo modo, **turnOffSeconds** determina el periodo de tiempo para atenuar los LEDs. Las variables **minOnSeconds** y **maxOnSeconds** determinan el intervalo de tiempos establecido por el potenciómetro.

En este programa no hay nada en la función **loop**. En su lugar todo el código está en **setup**. Por tanto, la luz iniciará automáticamente su ciclo cuando se encienda. Una vez que haya finalizado, permanecerá apagada hasta que se pulse el botón **Reset** (reinicio).

El encendido lento se consigue aumentando gradualmente el valor de la salida analógica en 1. Esto se lleva a cabo en un bucle **while**, donde el **delay** (retraso) se establece en 1/255 del tiempo de inicio, con lo que, tras 255 pasos, se habrá alcanzado el brillo máximo. El apagado lento funciona de forma parecida.

El período de tiempo de brillo máximo se determina por la entrada analógica. Suponiendo que queramos un intervalo de tiempo entre 5 y 30 minutos, necesitamos convertir el valor de 0 a 1023 en el número de segundos que hay entre 300 y 1800. Afortunadamente hay una práctica fun-

ción de Arduino que podemos utilizar para ello. La función **map** tiene cinco argumentos: el valor que desea convertir, el valor mínimo de entrada (0 en este caso), el valor máximo de entrada (1023), el valor mínimo de salida (300) y el valor máximo de salida (1800).

Pongamos todo junto

Cargue el sketch terminado del Proyecto 7 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Ahora tiene que conectar los cables desde **Vin**, **GND**, y el **pin digital 11** de la placa Arduino a los tres terminales de tornillo del módulo LED (Figura 4-12). Conecte una fuente de alimentación de 15 V en el conector de alimentación de la placa Arduino y estará listo para probarla.

Para iniciar de nuevo la secuencia de luz, haga clic en el botón de reinicio (reset).

Proyecto 8

Luz estroboscópica de alta potencia

Para este proyecto, puede utilizar el montaje de seis LED Luxeon del Proyecto 7 o puede usar el **shield Luxeon** que hemos creado para el Proyecto 4. El software será casi el mismo en ambos casos.

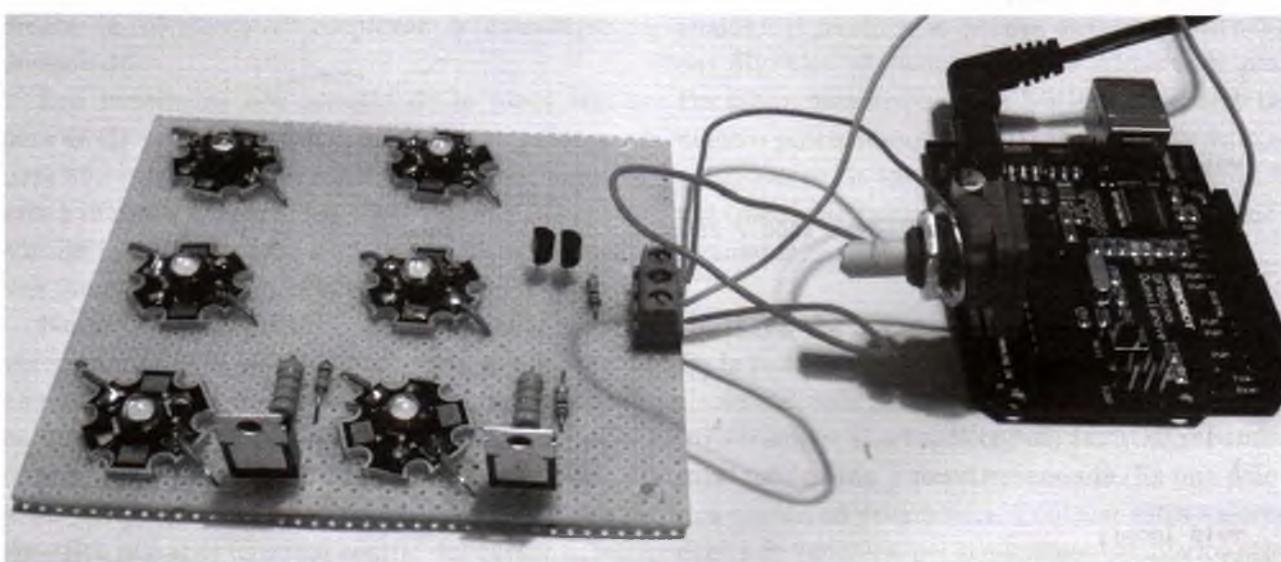


Figura 4-12 Proyecto 7. Luz para T.A.E.

En esta versión de luz estroboscópica, vamos a controlar con comandos, desde el ordenador, el efecto estroboscópico. Enviaremos los siguientes comandos a través de la conexión USB utilizando el **Serial Monitor**.

- 0-9** Establece la velocidad de los siguientes comandos de modo: 0 para apagado, 1 para lento y 9 para rápido.
- w** Efecto de onda que gradualmente se vuelve más clara y luego más oscuro.
- s** Efecto estroboscópico.

Hardware

Véase el Proyecto 4 (el traductor de código Morse que utiliza un único **shield LED Luxeon**) o el Proyecto 7 (matriz de seis LEDs Luxeon) para los componentes y detalles de su construcción.

Software

Este programa utiliza la función **sin** para crear un efecto que aumente el brillo de forma progresiva y agradable. Aparte de esto, casi todas las técnicas que utilizamos en este programa ya se han utilizado en proyectos anteriores.

LISTADO DE PROYECTO 8

```
int ledPin = 11;

int period = 100;

char mode = 'o'; // o-apagado, s-luz, w-onda

void setup()
{
    pinMode(ledPin, OUTPUT);
    analogWrite(ledPin, 255);
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available())
    {
        char ch = Serial.read();
        if (ch == '0')
        {
            mode = 0;
            analogWrite(ledPin, 255);
        }
        else if (ch > '0' && ch <= '9')
        {
            setPeriod(ch);
        }
        else if (ch == 'w' || ch == 's')
```

(continúa)

LISTADO DE PROYECTO 8 (continúa)

```
{  
    mode = ch;  
}  
}  
if (mode == 'w')  
{  
    waveLoop();  
}  
else if (mode == 's')  
{  
    strobeLoop();  
}  
}  
  
void setPeriod(char ch)  
{  
    int period1to9 = 9 - (ch - '0');  
    period = map(period1to9, 0, 9, 50, 500);  
}  
  
void waveLoop()  
{  
    static float angle = 0.0;  
    angle = angle + 0.01;  
    if (angle > 3.142)  
    {  
        angle = 0;  
    }  
    // analogWrite(ledPin, 255 - (int)255 * sin(angle)); // Placa de pruebas  
    analogWrite(ledPin, (int)255 * sin(angle)); // Shield  
    delay(period / 100);  
}  
  
void strobeLoop()  
{  
    //analogWrite(ledPin, 0); // Placa de pruebas  
    analogWrite(ledPin, 255); // shield  
    delay(10);  
    //analogWrite(ledPin, 255); // Placa de pruebas  
    analogWrite(ledPin, 0); // shield  
    delay(period);  
}
```

Pongamos todo junto

Cargue el sketch terminado del Proyecto 8 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Cuando haya instalado el programa y colocado el **shield Luxeon** o conectado el panel de seis LEDs, inicialmente las luces estarán apagadas. Abra la ventana **Serial Monitor**, introduzca **s** y pulse **RETURN**. Esto hará que la luz comience a parpadear. Pruebe los comandos de velocidad 1 a 9. A continuación, pruebe a escribir el comando **w** para cambiar al modo de onda.

Generación de números aleatorios

Los ordenadores son deterministas. Si les hace la misma pregunta dos veces, debería obtener la misma respuesta. Sin embargo, a veces, quiere sentir lo que es "llevar la mano". Evidentemente, esto es útil en el caso de los juegos.

También es útil en otras circunstancias. Imagínese lo siguiente: un "paseo aleatorio" (donde un robot realiza un giro al azar, luego se desplaza hacia adelante una distancia aleatoria o hasta que se golpea contra algo, luego da marcha atrás y se da la vuelta de nuevo) es mucho mejor para garantizar que el robot cubre toda la zona de una habitación que un algoritmo más fijo que puede dar como resultado que el robot se quede atrapado en una determinada secuencia.

La biblioteca Arduino incluye una función para generar números aleatorios.

Hay dos versiones de la función **random** (aleatoriedad). Puede tomar dos argumentos (mínimo y máximo) o un único argumento (**máximo**), en cuyo caso el valor mínimo se asume que es 0.

No obstante, tenga cuidado porque el argumento máximo es engañoso, ya que el número mayor que puede devolverle es el máximo menos uno.

Así, la siguiente línea dará a **x** un valor entre 1 y 6:

```
int x = random(1, 7);
```

y la siguiente línea dará a **x** un valor entre 0 y 9:

```
int x = random(10);
```

Como hemos señalado al inicio de esta sección, los ordenadores son deterministas y, en realidad, nuestros números aleatorios no son aleatorios en absoluto, sino una larga secuencia de números con una distribución aleatoria. Obtendrá la misma secuencia de números cada vez que ejecute el script.

Una segunda función (**randomSeed**) le permite controlar esto. La función **randomSeed** determina en qué parte de su secuencia de pseudo-números aleatorios empezará el generador de números aleatorios.

Un buen truco es utilizar el valor de una entrada analógica desconectada, ya que esto irá dando un valor diferente y con ello conseguirá un mínimo de 1000 puntos de partida diferentes para nuestra secuencia aleatoria. Esto no serviría para la lotería, pero es aceptable para la mayoría de las aplicaciones. Obtener números verdaderamente aleatorios es un tema muy complicado y requiere un hardware especial.

Proyecto 9 Dado de LEDs

Este proyecto utiliza lo que acabamos de aprender sobre números aleatorios para crear un dado electrónico con seis LEDs y un botón. Cada vez que se pulsa el botón, el LED "rueda" durante un tiempo antes de quedarse en un valor, para luego comunicarlo mediante destellos.

COMPONENTES Y EQUIPO

	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o similar	1
D1-7	LEDs rojos estándar	23
R1-7	Resistencia 270 Ω 0,5 W	6
S1	Pulsador en miniatura para conmutación	48
R8	Resistencia 100K Ω 0,5 W	13

Hardware

El esquema eléctrico del Proyecto 9 se muestra en la Figura 4-13. Cada LED se maneja con una salida digital independiente mediante una resistencia limitadora de corriente. Aparte de esto lo único que hace falta es el pulsador y la correspondiente resistencia asociada de puesta a masa.

Aunque los datos suelen tener un máximo de seis puntos, seguimos necesitando siete LEDs para conseguir la disposición normal de un punto en el centro para mostrar los números impares.

La Figura 4-14 muestra la distribución de la placa de pruebas y la Figura 4-15 la placa de pruebas finalizada.

Software

Este programa es bastante sencillo, pero aún así tiene pequeños detalles que hacen que el dado se comporte de manera similar a un dado real. Por ejemplo, cuando el dado "rueda", el número cambia pero va disminuyendo gradualmente. Además, la cantidad de tiempo que "rueda" el dado también es aleatoria.

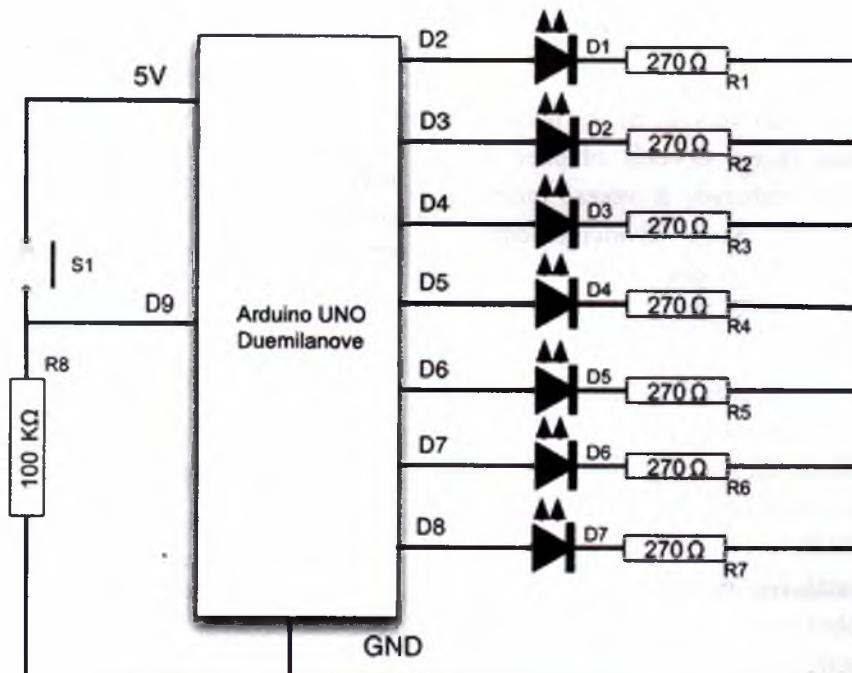


Figura 4-13 Esquema eléctrico del Proyecto 9.

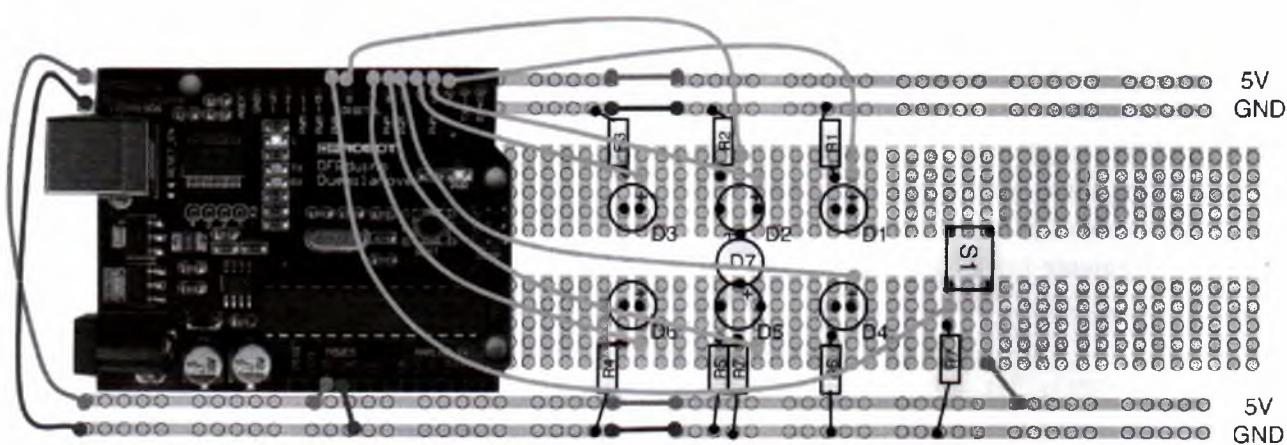


Figura 4-14 Distribución de la placa de pruebas para el Proyecto 9.

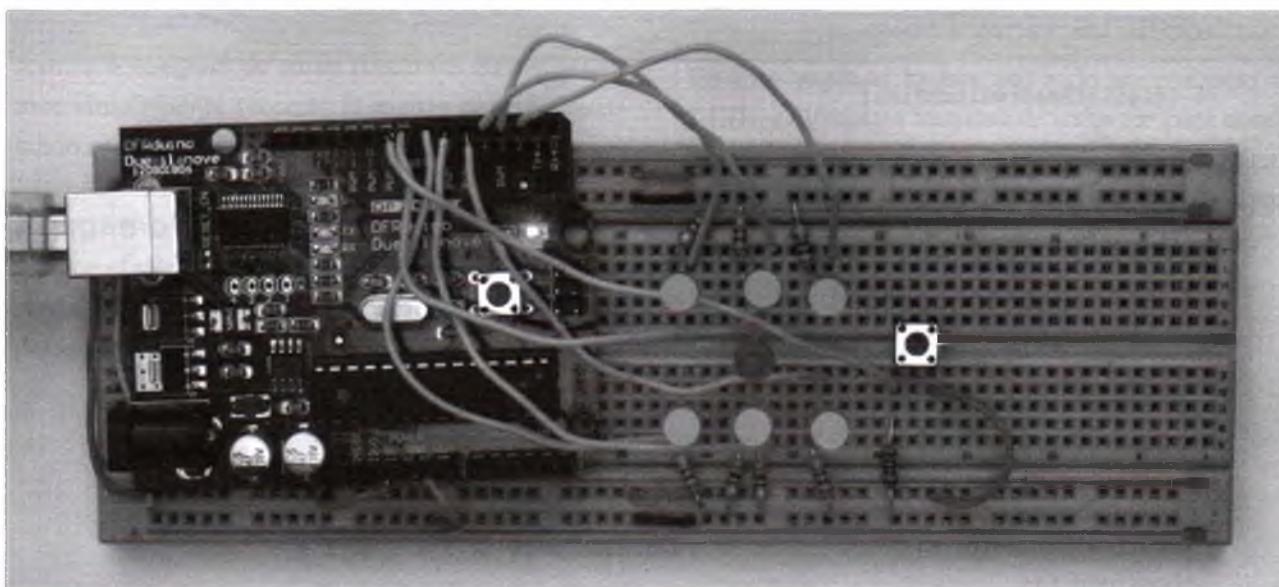


Figura 4-15 Proyecto 9. Dado de LEDs.

LISTADO DE PROYECTO 9

```
int ledPins[7] = {2, 3, 4, 5, 6, 7, 8};  
int dicePatterns[7][7] = {  
    {0, 0, 0, 0, 0, 0, 1}, // 1  
    {0, 0, 1, 1, 0, 0, 0}, // 2  
    {0, 0, 1, 1, 0, 0, 1}, // 3  
    {1, 0, 1, 1, 0, 1, 0}, // 4  
    {1, 0, 1, 1, 0, 1, 1}, // 5  
    {1, 1, 1, 1, 1, 1, 0}, // 6  
    {0, 0, 0, 0, 0, 0, 0} // En blanco  
};  
  
int switchPin = 9;  
int blank = 6;  
  
void setup()  
{  
    for (int i = 0; i < 7; i++)  
    {  
        pinMode(ledPins[i], OUTPUT);  
        digitalWrite(ledPins[i], LOW);  
    }  
    randomSeed(analogRead(0));  
}  
  
void loop()  
{
```

(continúa)

LISTADO DE PROYECTO 9 (continúa)

```

if (digitalRead(switchPin))
{
  rollTheDice();
}
delay(100);
}

void rollTheDice()
{
  int result = 0;
  int lengthOfRoll = random(15, 25);
  for (int i = 0; i < lengthOfRoll; i++)
  {
    result = random(0, 6);           // resultado será de 0 a 5 no 1 a 6
    show(result);
    delay(50 + i * 10);
  }
  for (int j = 0; j < 3; j++)
  {
    show(blank);
    delay(500);
    show(result);
    delay(500);
  }
}

void show(int result)
{
  .
  .

  for (int i = 0; i < 7; i++)
  {
    digitalWrite(ledPins[i], dicePatterns[result][i]);
  }
}

```

Ahora tenemos siete LEDs que inicializar en la función **setup**, por lo que vale la pena ponerlos en una matriz (**array**) y crear un bucle (**loop**) que vaya recorriendo la matriz para inicializar cada pin. También tenemos una llamada a **randomSeed** en **setup**, que si no estuviera allí, cada vez que hiciéramos un reset de la placa terminaríamos siempre con la misma secuencia de lanzamientos del dado. Como experimento, si lo desea, pruebe a convertir

esta línea en comentario colocando una **//** al principio de la misma para comprobarlo. ¡De hecho, puede que prefiera omitir esta línea y hacer trampas en el juego de La oca!

El **array dicePatterns** determina qué LEDs deben estar encendidos o apagados en una jugada determinada. De forma que, cada elemento de tiro de la matriz, es en realidad en sí mismo otra matriz de siete elementos, pudiendo ser cada uno de ellos

HIGH o LOW (1 ó 0). Cuando vamos a mostrar el resultado concreto de haber tirado los dados, podemos simplemente recorrer la matriz del tiro, ajustando cada LED en consecuencia.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 9 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Resumen

En este capítulo hemos utilizado un conjunto de LEDs y diferentes técnicas de software para conseguir interesantes efectos luminosos. En el siguiente capítulo vamos a investigar algunos sensores para utilizarlos como entrada en nuestros proyectos.

CAPÍTULO 5

Proyectos con sensores

LOS SENSORES CONVIERTEN las medidas del mundo real en señales electrónicas que podemos utilizar en nuestras placas Arduino. Todos los proyectos en este capítulo tratan sobre el uso de la luz y la temperatura.

También echaremos un vistazo a cómo establecer una interfaz con teclados y codificadores giratorios.

Proyecto 10 Código de seguridad con el teclado

Este proyecto seguramente ocuparía un lugar destacado en cualquier mente inquieta que se precie. Básicamente consiste en que al introducirse un código de seguridad en un teclado; si es correcto, se iluminará un LED verde; de lo contrario, se encenderá un LED rojo. En el Proyecto 27 volveremos sobre este proyecto y mostraremos cómo ampliarlo para que no sólo muestre la luz adecuada, sino que

también controle la cerradura de una puerta.

Dado que por lo general los teclados no suelen tener pines conectados, nosotros se los tendremos que soldar, por lo que en este proyecto tendremos de nuevo la posibilidad de practicar la soldadura.

Hardware

El esquema electrónico del Proyecto 10 se muestra en la Figura 5-1. En los proyectos anteriores nos habíamos familiarizado con los LED; y ahora le añadimos a estos un nuevo componente: el teclado.

Los teclados normalmente se organizan en una cuadrícula, de modo que cuando se pulsa una tecla, se conecta una fila a una columna. La Figura 5-2 muestra la distribución típica de un teclado matricial de 12-teclas, con números de 0 al 9 y las teclas * y #.

Los pulsadores de cada tecla se encuentran en la intersección de los cables de las filas y columnas. Al pulsar una tecla, se conecta una determinada fila a una determinada columna.

Organizar las teclas en una cuadrícula como ésta significa que sólo necesitamos usar 7 pines digitales (4 filas + 3 columnas) en lugar de 12 (uno por cada tecla).

No obstante, esto también implica que tenemos que trabajar un poco más el software para determinar qué teclas se presionan. El enfoque básico consiste en conectar cada fila a una salida digital y cada columna a una entrada digital. A continuación, activamos cada salida en orden y vemos qué entradas se han activado.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
D1 LED rojo de 5 mm	23
D2 LED verde de 5 mm	25
R1-2 Resistencia 270 Ω 0,5 W	6
K1 Teclado de 4 x 3	54
Tira cabecera de 2,5 cm	55

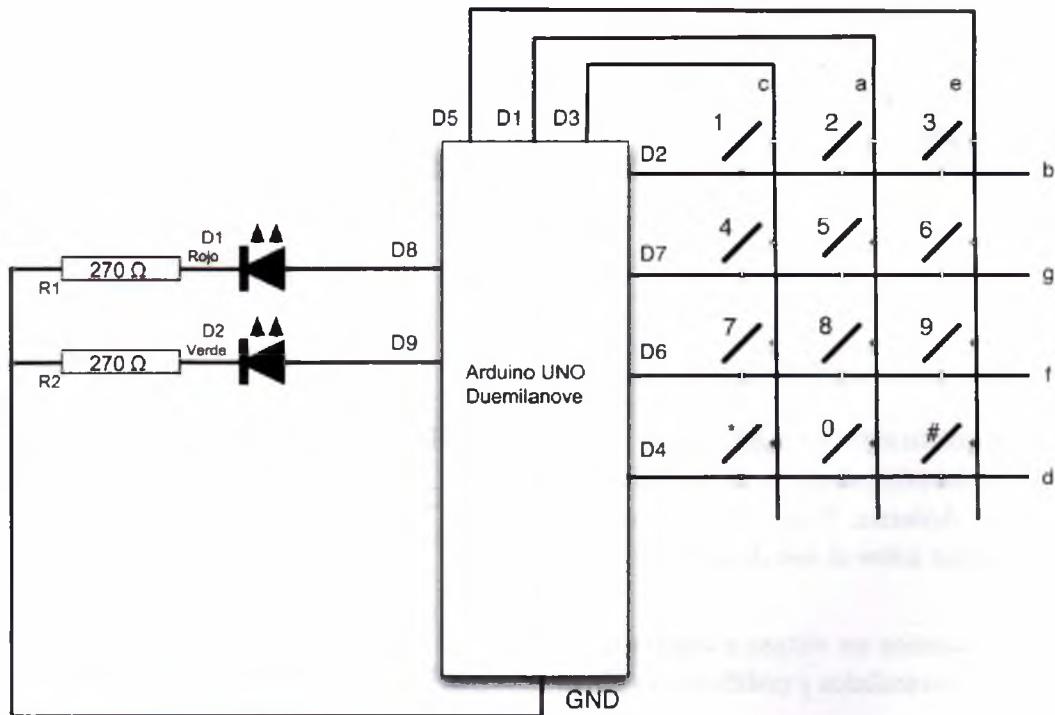


Figura 5-1 Esquema eléctrico del Proyecto 10.

La Figura 5-3 muestra cómo soldar un conector de siete pines al teclado para que luego pueda conectarlos a la placa de pruebas. Estos pines se

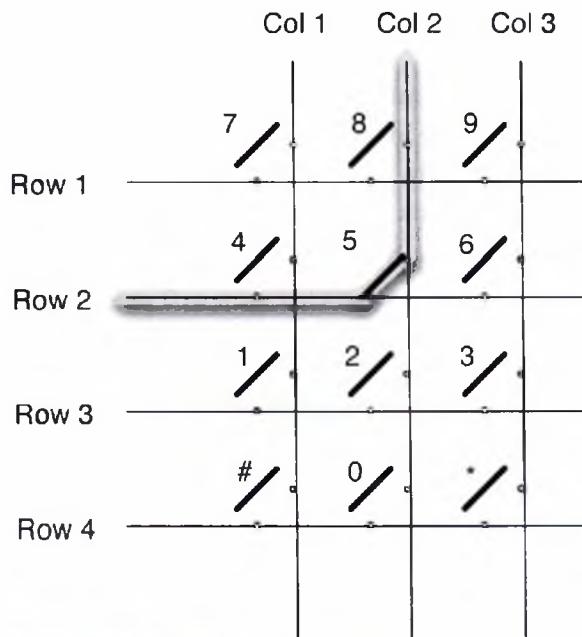


Figura 5-2 Teclado matricial de 12 botones.

compran en tiras, y de éstas se puede cortar el número de pines que necesitemos.

Ahora, lo único que necesitamos es averiguar qué pin del teclado numérico corresponde a cada fila o columna. Si tenemos suerte, el teclado vendrá con una hoja informativa que nos lo indique. Si no, tendremos que averiguarlo con la ayuda de un polímetro. Coloque el polímetro en continuidad para que suene un bip cuando juntemos las puntas de prueba. A continuación, tome un papel y dibuje un



Figura 5-3 Soldadura de pines al teclado.

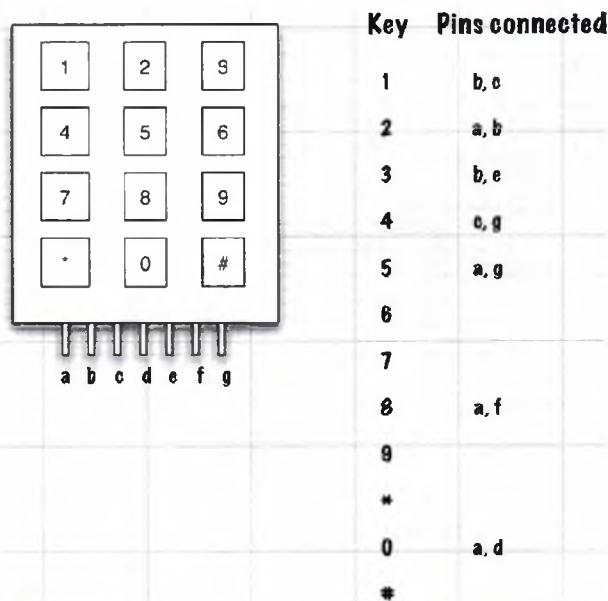


Figura 5-4 Comprobación de las conexiones del teclado.

diagrama de las conexiones del teclado y etiquete cada pin con una letra desde la **a** hasta la **g**. Luego, escriba una lista de todas las teclas. Entonces, manteniendo pulsada cada una de las teclas, una tras otra, busque el par de pines que hace sonar el polímetro, lo que indicará una conexión (Figura 5-4). Suelte la tecla para verificar que realmente ha encontrado el par correcto. Tras un cierto tiempo, aparecerá un patrón y podrá ver con facilidad la relación de los pines con las filas y columnas. La Figura 5-4 muestra la disposición del teclado numérico utilizado por el autor.

En la Figura 5-5 se muestra el diseño terminado de la placa de pruebas. Observe que el teclado tiene siete pines que se insertan directamente en los conectores **Digital Pin 1** a **7** de la placa Arduino (Figura 5-6), por lo que sólo necesitamos la placa de pruebas para los dos LED.

Puede que ya haya notado que junto a los pines digitales 0 y 1 aparecen las etiquetas **TX** y **RX**. Esto es debido a que también son utilizados por la placa Arduino para las comunicaciones serie, incluyendo la conexión **USB**. En este caso, no estamos utilizando el pin digital 0, pero hemos conectado el pin digital 1 a la columna central del teclado. Esto significa que podremos seguir programando la placa, pero que no podremos comunicarnos a través de la conexión USB mientras que el programa se esté ejecutando. Puesto que en cualquier caso tampoco nos hace falta, en realidad esto no representa ningún problema.

Software

Si bien podríamos escribir un programa que active la salida de cada fila, una tras otra, y que lea las entradas para obtener las coordenadas de cada tecla pulsada, en realidad es un poco más complicado, ya que los pulsadores no siempre se comportan de la manera adecuada cuando se pulsan. Tanto los teclados como el resto de pulsadores eléctricos son propensos a "rebotar". Es decir, cuando se pulsan, a veces no sólo se limitan a pasar de abierto a cerrado,

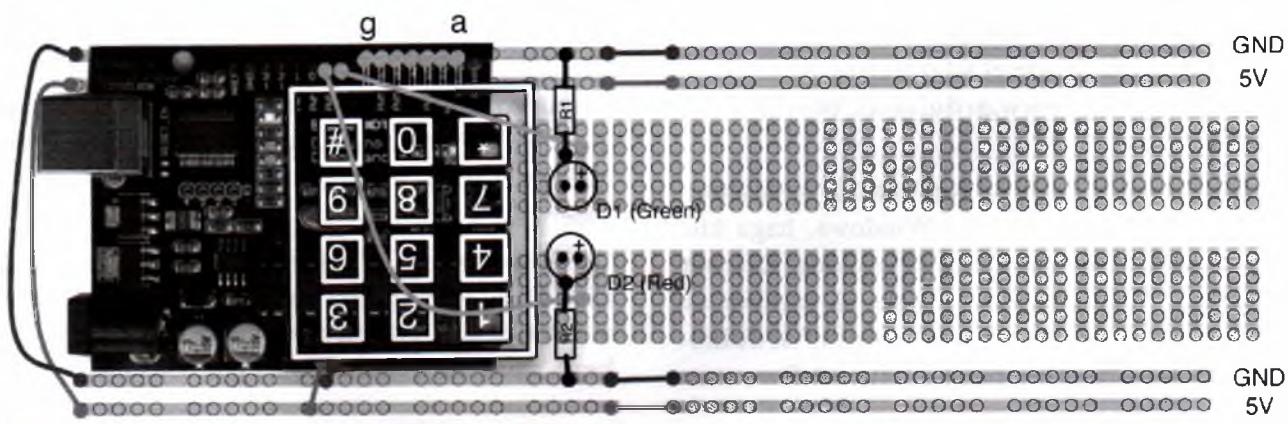


Figura 5-5 Diseño de la placa de pruebas del Proyecto 10.



Figura 5-6 Proyecto 10. Código de seguridad con teclado.

sino que al pulsar el botón se abren y cierran unas cuantas veces.

Afortunadamente, Mark Stanley y Alexander Brevig han creado una biblioteca para usarla con los teclados y resolver el problema del rebote, ahorrándonos a nosotros el trabajo de hacerlo. Esta es una buena oportunidad para demostrar cómo se instala una biblioteca en el software de Arduino.

Además de las bibliotecas que vienen con la placa Arduino, muchas personas han desarrollado sus propias bibliotecas y han publicado sus resultados para beneficio de la comunidad Arduino. Aunque a algunos puede que este altruismo les haga sonreír y que lo vean como una debilidad, esperemos que no se sientan tan superiores como para no utilizar esas bibliotecas para sus propios fines.

Para hacer uso de esta biblioteca, primero debemos descargarla desde el sitio web de Arduino en esta dirección: www.arduino.cc/playground/Code/Keypad.

Descargue el archivo **Keypad.zip** y descomprímalo. Si está utilizando Windows, haga clic con el botón secundario y seleccione **Extraer todo** y, a continuación, guarde el archivo en **C:\Archivos de programa\arduino\arduino-0019\hardware\bibliotecas** (Figura 5-7).

En Linux, busque el directorio de instalación de Arduino y copie la carpeta en **hardware/bibliotecas**.

En un Mac, no ponga la nueva biblioteca en la instalación de Arduino. En su lugar, cree una carpeta llamada **bibliotecas** en **Documentos/Arduino** (Figura 5-8) y coloque allí la carpeta **bibliotecas** completa. Por cierto, el directorio **Documentos/Arduino** es también la ubicación pre-determinada donde se almacenan sus **sketches**.

Una vez que hayamos instalado esta biblioteca en nuestro directorio Arduino, podremos usarla con cualquiera de los programas que escribamos. Pero recuerde que en Windows y Linux, si actualiza a una versión más reciente del software Arduino, tendrá que volver a instalar las bibliotecas que utilice.

Puede comprobar que la biblioteca está correctamente instalada reiniciando la Arduino, empezar un nuevo **sketch** y eligiendo la opción de menú **Sketch | Import Library | Keypad**. Esto debe insertar el texto "#include <Keypad.h>" en la parte superior del archivo.

El **sketch** de la aplicación se muestra en el Listado del Proyecto 10. Tenga en cuenta que puede que tenga que cambiar las matrices (**arrays**) de **keys**, **rowPins** y **colPins**, para que coincidan con la

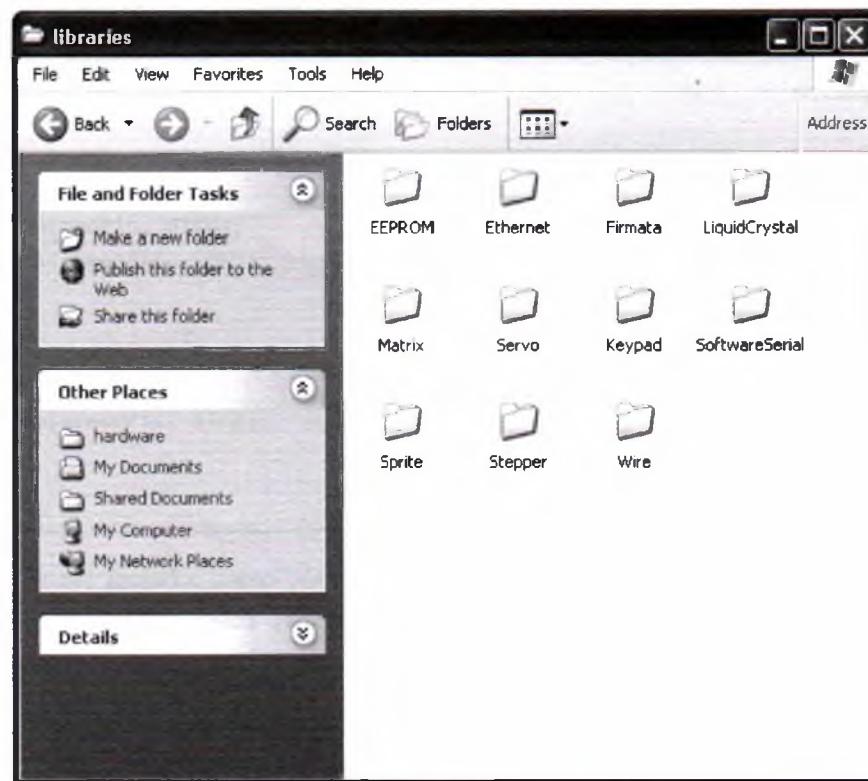


Figura 5-7 Instalación de la biblioteca para Windows.

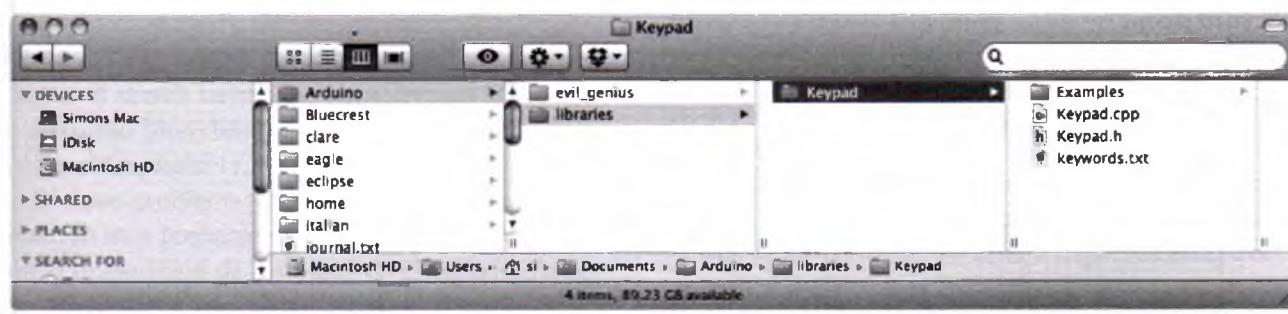


Figura 5-8 Instalación de la biblioteca para Mac.

LISTADO DEL PROYECTO 10

```
#include <Keypad.h>

char* secretCode = "1234";
int position = 0;

const byte rows = 4;
const byte cols = 3;
char keys[rows][cols] = {
    {'1','2','3'},
    {'4','5','6'},
    {'7','8','9'},
    {'*','0','#'}
};
byte rowPins[rows] = {2, 7, 6, 4};
byte colPins[cols] = {3, 1, 5};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, rows, cols);

int redPin = 9;
int greenPin = 8;

void setup()
{
    pinMode(redPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    setLocked(true);
}

void loop()
{
    char key = keypad.getKey();
    if (key == '*' || key == '#')
    {
        position = 0;
        setLocked(true);
    }
    if (key == secretCode[position])
    {
        position++;
    }
    if (position == 4)
    {
        setLocked(false);
    }
    delay(100);
}

void setLocked(int locked)
```

(continúa)

LISTADO DEL PROYECTO 10 (continúa)

```

    {
      if (locked)
      {
        digitalWrite(redPin, HIGH);
        digitalWrite(greenPin, LOW);
      }
      else
      {
        digitalWrite(redPin, LOW);
        digitalWrite(greenPin, HIGH);
      }
    }
  }
}

```

distribución de teclas del teclado, como comentamos en la sección del hardware.

Este sketch es bastante sencillo. La función **loop** comprueba si se ha pulsado una tecla. Si la tecla presionada es # o *, vuelve a establecer a 0 la posición de la variable. Si, por otro lado, la tecla que se pulsa es uno de los números, comprueba si la tecla pulsada es la siguiente tecla esperada (**secretCode[position]**) y, si es así, incrementa la posición en uno. Por último, el bucle comprueba si la posición es 4 y, si es así, coloca los LED en su estado de desbloqueado.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 10 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Si tiene problemas para conseguir que esto funcione, lo más probable es que haya algún problema con la disposición de los pines en el teclado. Así es que, compruebe con el polímetro el trazado de las conexiones de los pines.

Codificadores giratorios

Ya conocemos cómo funcionan los potenciómetros: a medida que giramos el mando, va cambiando la resistencia. Tradicionalmente, estos solían estar tras la mayoría de los mandos giratorios que pudiera manipular en los equipos electrónicos. Hoy día existe una alternativa, el codificador giratorio, y si tiene en casa algún aparato electrónico en el que se

pueda girar el mando indefinidamente sin que llegue a ningún tope de parada, probablemente detrás del mando haya un codificador giratorio.

Algunos codificadores giratorios también incorporan un botón, de forma que puede girar el mando y luego pulsar. Esta es una forma especialmente útil de hacer una selección de un menú cuando se utiliza con una pantalla de cristal líquido (LCD).

Un codificador giratorio es un dispositivo digital que tiene dos salidas (A y B) y, según se gira el mando, se obtiene un cambio en los resultados, que pueden indicar si el mando se ha girado hacia la derecha o hacia la izquierda.

La Figura 5-9 muestra cómo cambian las señales en A y B cuando se gira el codificador. Cuando gira hacia la derecha, los pulsos cambian, ya que estarían desplazándose de izquierda a derecha en el diagrama; cuando se mueve hacia la izquierda, los pulsos se moverían de derecha a izquierda en el diagrama.

Por tanto, si **A** está desactivado (bajo) y **B** está desactivado (bajo), y luego **B** se activa (alto) (pasa de la fase 1 a la 2), esto indicaría que hemos girado el mando hacia la derecha. También se indicaría un giro a la derecha si **A** está desactivado (bajo) y **B** se encuentra activado (alto) y, entonces, **A** se activa (alto) (pasando de la fase 2 a la fase 3), etc. Sin embargo, si **A** estuviera alto y **B** estuviera bajo y luego **B** pasara a alto, habríamos pasado de la etapa 4 a la etapa 3 y, por lo tanto, se habría girado hacia la izquierda.

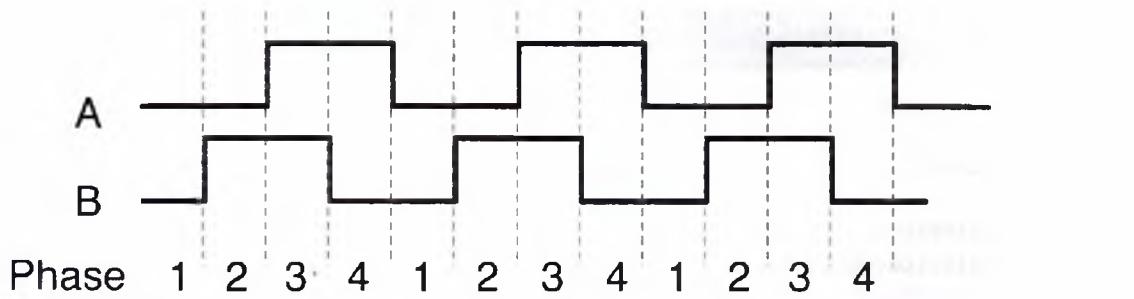


Figura 5-9 Pulsos de un codificador giratorio.

Proyecto 11

Modelo de semáforo basado en un codificador giratorio

Este proyecto utiliza un codificador giratorio con un pulsador incorporado para controlar la secuencia de las señales de un semáforo, y se basa en el Proyecto 5. Se trata de una versión mucho más realista de un semáforo y, realmente, no se encuentra muy alejado de la lógica que se encontraría en un semáforo auténtico.

Al girar el codificador giratorio cambiará la frecuencia de la secuenciación de las luces. Pulsar el botón hará que se prueben las luces, encendiéndolas todas al mismo tiempo mientras se mantiene pulsado.

Los componentes son los mismos que los del Proyecto 5, con la adición del codificador giratorio y de las resistencias de puesta a nivel alto en lugar del conmutador de botón original.

Hardware

El esquema eléctrico del Proyecto 11 se muestra en la Figura 5-10. La mayor parte del circuito es la misma que la del Proyecto 5, excepto que ahora tenemos un codificador giratorio.

El codificador giratorio funciona igual que si hubiera tres interruptores: uno para A, otro para B y otro para el botón pulsador. Cada uno de estos interruptores requiere de una resistencia de puesta a masa.

Dado que el circuito es prácticamente igual al del Proyecto 5, no será una gran sorpresa ver que la distribución de la placa de pruebas (Figura 5-11) es similar a la de ese proyecto.

Software

El punto de partida del **sketch** es el **sketch** del Proyecto 5. Hemos añadido código para leer el codificador y para responder a la pulsación del botón que enciende todos los LEDs. También hemos aprovechado la oportunidad para mejorar la lógica para que las luces se comporten de manera más realista, cambiando automáticamente. En el Proyecto 5, cuando se mantenía pulsado el botón, las luces cambiaban de secuencia aproximadamente una vez por segundo. En los semáforos reales, las luces permanecen verde y rojo mucho más tiempo de lo que lo hacen las de color amarillo.

Por lo tanto, nuestro **sketch** tiene ahora dos períodos: **shortPeriod**, que no cambia pero que se utiliza cuando las luces están cambiando, y **longPeriod**, que determina cuánto tiempo están iluminadas cuando

COMPONENTES Y EQUIPO

	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o similar	1
D1	LED rojo de 5 mm	23
D2	LED amarillo de 5 mm	24
D3	LED verde de 5 mm	25
R1-R3	Resistencia 270 Ω 0,5 W	6
R4-R6	Resistencia 100 KΩ 0,5 W	13
S1	Codificador giratorio con pulsador para conmutación	57

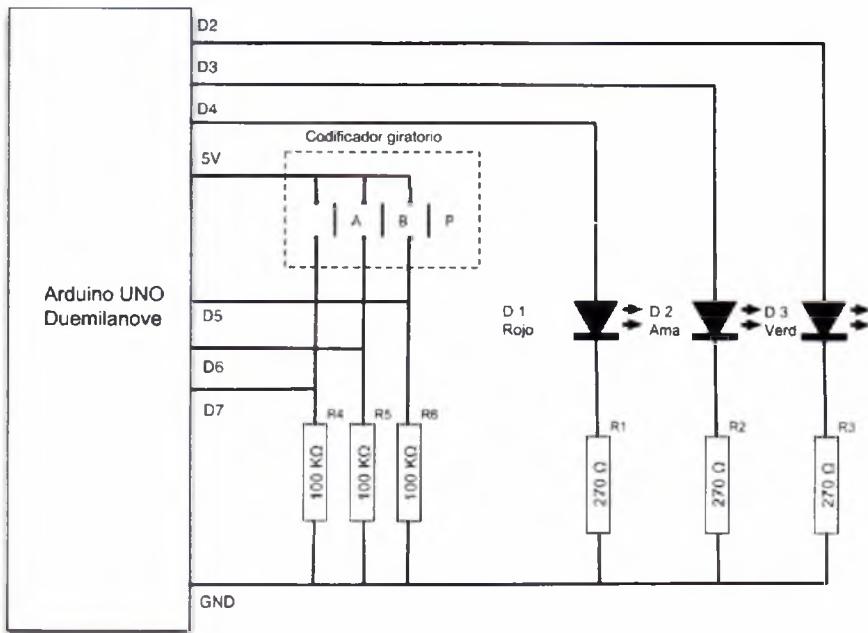


Figura 5-10 Esquema eléctrico del Proyecto 11.

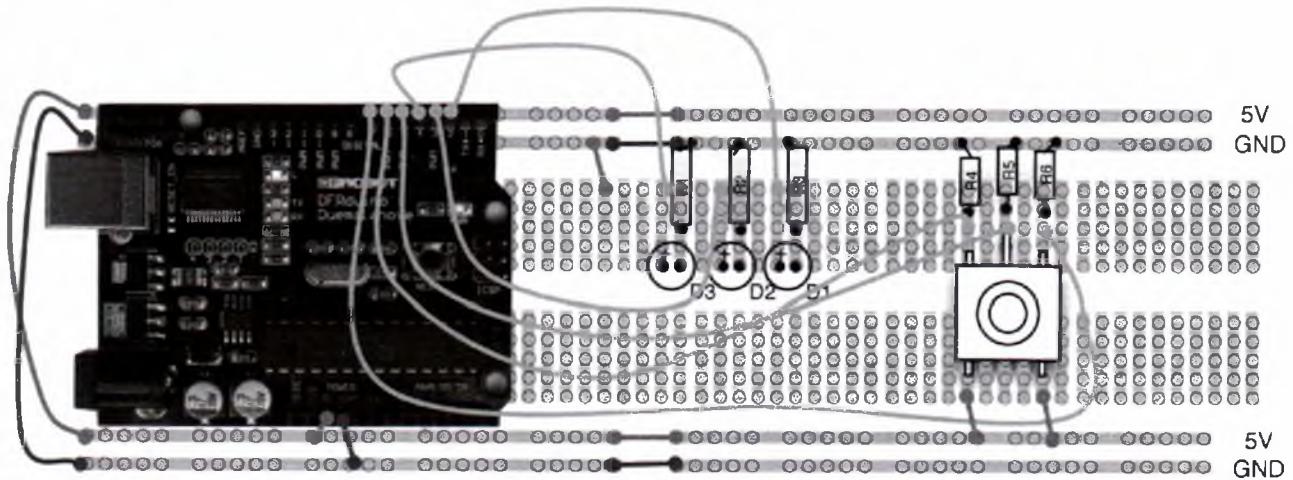


Figura 5-11 Disposición de componentes del Proyecto 11 en la placa de pruebas.

están verde o rojo. Este `longPeriod` es el período que se modifica girando el mando del codificador.

La clave para manejar el codificador giratorio se encuentra en la función `getEncoderTurn`. Cada vez que se llama a esta función, compara el estado anterior de A y B con su estado actual y, si algo ha cambiado, averigua si se ha girado hacia la derecha o hacia la izquierda y devuelve un -1 y 1, respectivamente. Si no hay ningún cambio (el mando no se ha girado), devuelve 0. Esta función debe ser llamada con frecuencia o, de lo contrario, si se girara con rapi-

dez el controlador giratorio resultaría que algunos cambios no serían reconocidos correctamente.

Si desea utilizar un codificador giratorio para otros proyectos, puede copiar esta función. Esta función utiliza el modificador `static` para las variables `oldA` y `oldB`. Esta es una técnica muy útil que permite que la función mantenga el valor entre una llamada a la función y la siguiente, cuando normalmente reiniciaría el valor de la variable cada vez que se llamaría a la función.

LISTADO DE PROYECTO 11

```

int redPin = 2;
int yellowPin = 3;
int greenPin = 4;
int aPin = 6;
int bPin = 7;
int buttonPin = 5;

int state = 0;
int longPeriod = 5000;      // Tiempo en verde o rojo
int shortPeriod = 700;       // Tiempo al cambiar
int targetCount = shortPeriod;
int count = 0;

void setup()
{
    pinMode(aPin, INPUT);
    pinMode(bPin, INPUT);
    pinMode(buttonPin, INPUT);
    pinMode(redPin, OUTPUT);
    pinMode(yellowPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
}

void loop()
{
    count++;
    if (digitalRead(buttonPin))
    {
        setLights(HIGH, HIGH, HIGH);
    }
    else
    {
        int change = getEncoderTurn();
        int newPeriod = longPeriod + (change * 1000);
        if (newPeriod >= 1000 && newPeriod <= 10000)
        {
            longPeriod = newPeriod;
        }
        if (count > targetCount)
        {
            setState();
            count = 0;
        }
    }
    delay(1);
}

int getEncoderTurn()
{
    // devuelve -1, 0, o +1
    static int oldA = LOW;
}

```

(continúa)

LISTADO DE PROYECTO 11 (continúa)

```
static int oldB = LOW;
int result = 0;
int newA = digitalRead(aPin);
int newB = digitalRead(bPin);
if (newA != oldA || newB != oldB)
{
    // algo ha cambiado
    if (oldA == LOW && newA == HIGH)
    {
        result = -(oldB * 2 - 1);
    }
}
oldA = newA;
oldB = newB;
return result;
}

int setState()
{
    if (state == 0)
    {
        setLights(HIGH, LOW, LOW);
        targetCount = longPeriod;
        state = 1;
    }
    else if (state == 1)
    {
        setLights(HIGH, HIGH, LOW);
        targetCount = shortPeriod;
        state = 2;
    }
    else if (state == 2)
    {
        setLights(LOW, LOW, HIGH);
        targetCount = longPeriod;
        state = 3;
    }
    else if (state == 3)
    {
        setLights(LOW, HIGH, LOW);
        targetCount = shortPeriod;
        state = 0;
    }
}

void setLights(int red, int yellow, int green)
{
    digitalWrite(redPin, red);
    digitalWrite(yellowPin, yellow);
    digitalWrite(greenPin, green);
}
```

Este **sketch** muestra una técnica útil que permite planificar eventos (encender un LED durante determinados segundos) al mismo tiempo que se comprueba si se ha girado el codificador giratorio o pulsado el botón. Si utilizamos la función **delay** de Arduino con, digamos, 20.000, para 20 segundos, en ese período no nos daría tiempo a comprobar el codificador giratorio o el conmutador.

Por tanto, lo que hacemos es utilizar un retardo muy breve (1 milisegundo) pero manteniendo un contador que se incrementa con cada ciclo del bucle. Así, si queremos un retardo de 20 segundos, nos paramos cuando el contador alcance 20.000. Esto es menos exacto que una única llamada a la función **delay**, debido a que 1 milisegundo es en realidad 1 milisegundo más el tiempo de procesamiento del resto de las operaciones que se hacen dentro del bucle.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 11 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Puede presionar el botón del codificador giratorio para probar los LEDs y girar el codificador giratorio para cambiar la duración en la que la señal permanece verde y roja.

Detección de la luz

Un dispositivo común y fácil de usar para medir la intensidad de la luz son las resistencias dependientes de la luz o **LDR** (Light Dependent Resistor). A veces, también se conocen como foto resistencias o foto resistores.

Cuanto mayor sea la cantidad de luz que incide sobre la superficie de una **LDR**, más baja será su resistencia. Una **LDR** típica tendrá una resistencia en la oscuridad de hasta $2\text{ M}\Omega$ y una resistencia al ser iluminada con luz del día brillante de quizás $20\text{ K}\Omega$.

Podemos convertir esta variación en la resistencia en una variación de tensión utilizando la **LDR**

junto con una resistencia fija para formar un divisor de tensión, conectando su salida a una de nuestras entradas analógicas. El esquema electrónico se muestra en la Figura 5-12.

Con una resistencia fija de $100\text{ K}\Omega$, podemos hacer algunos cálculos acerca del rango de tensiones que se puede esperar en la entrada analógica.

En la oscuridad, la **LDR** tendrá una resistencia de $2\text{ M}\Omega$, así que con una resistencia fija de $100\text{ K}\Omega$, tendremos una relación de tensión de alrededor de 20:1, con la mayoría de la tensión en la **LDR**, lo que se traduciría en alrededor de 4 V en la **LDR** y 1 V en el pin analógico.

Por otro lado, si exponemos la **LDR** a una fuerte intensidad lumínosa, su resistencia podría bajar a $20\text{ K}\Omega$. La proporción de tensiones sería entonces alrededor de 4:1 a favor de la resistencia fija, lo que da una tensión en la entrada analógica de alrededor de 4 V.

Un detector de luz más sensible es el **fototransistor**. Funciona como un transistor normal salvo que suele carecer de conexión en la base. En su lugar, la corriente del colector es controlada por la cantidad de luz que incide sobre el fototransistor.

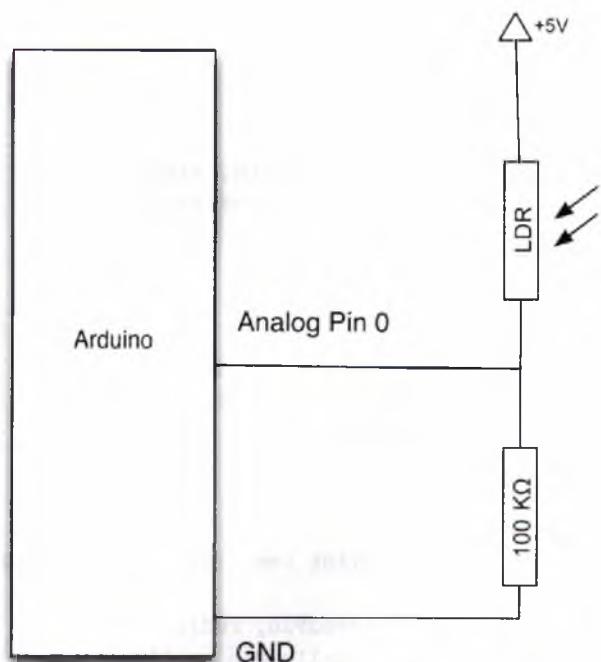


Figura 5-12 Utilización de una LDR para medir la luz.

Proyecto 12

Monitor de pulsaciones

Este proyecto utiliza un LED infrarrojo (IR) ultra-brillante y un fototransistor para detectar el pulso cardíaco en el dedo, haciendo parpadear un LED rojo al ritmo de las pulsaciones.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
D1 LED rojo de 5 mm	23
D2 Emisor 940 nm Infrarrojos	
LED de 5 mm	26
R1 Resistencia 56 KΩ 0,5 W	12
R2 Resistencia 270 Ω 0,5 W	6
R4 Resistencia 39 Ω 0,5 W	4
T1 Fototransistor de IR (misma longitud onda D2)	36

Hardware

El monitor de pulsaciones funciona de la siguiente manera: la luz del LED se transmite a través del dedo, siendo recibida en el otro lado por el fototransistor, cuya resistencia variará ligeramente en función de la sangre que fluya por el dedo.

El esquema eléctrico se muestra en la Figura 5-13 y la disposición de componentes en la placa de pruebas, en la Figura 5-15. Hemos elegido un elevado valor de resistencia para R1, debido a que la mayor parte de la luz que pasa a través del dedo será absorbida, y queremos que el fototransistor sea muy sensible. Quizás tenga que experimentar con el valor de la resistencia hasta conseguir buenos resultados.

Es importante proteger el fototransistor de la mayor cantidad de luz parásita que sea posible. Esto es especialmente importante en el caso de la iluminación doméstica, que en la realidad oscilan a 50 Hz o 60 Hz y agregan una considerable cantidad de “ruido” a nuestra débil señal del corazón.

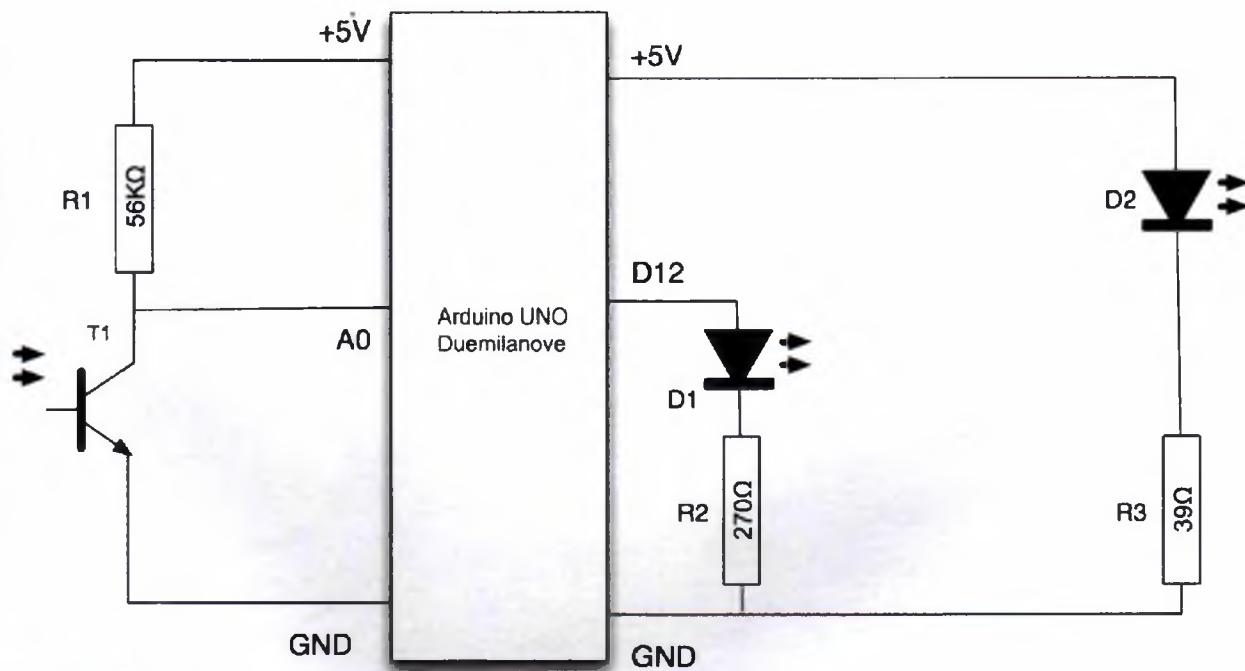


Figura 5-13 Esquema eléctrico del Proyecto 12.



Figura 5-14 Tubo sensor para el monitor cardíaco.

Por esta razón, tanto el fototransistor como el LED se han integrado en un tubo o cartón ondulado unido con cinta aislante, cuyo montaje se muestra en la Figura 5-14.

En el tubo se han perforado, uno frente al otro, dos agujeros de 5 mm, para así introducir el LED en un lado y el foto transistor en el otro. Soldamos unos cables cortos al LED y al fototransistor y, a continuación, colocamos otra capa de cinta aislante para envolverlo todo y para asegurar la posición del LED y el fototransistor. Asegúrese de comprobar los colores de los cables que ha conectado a cada pata del LED y del fototransistor antes de colocar la cinta aislante.

El diseño de la placa de pruebas de este proyecto (Figura 5-15) es muy sencillo.

El "tubo sensor de dedo" finalizado puede verlo en la Figura 5-16.

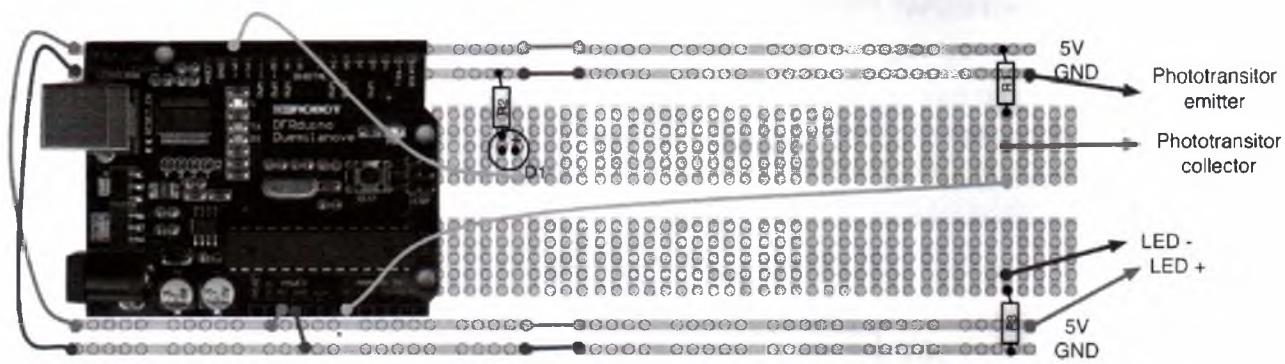


Figura 5-15 Disposición de los componentes en la placa de pruebas del Proyecto 12.

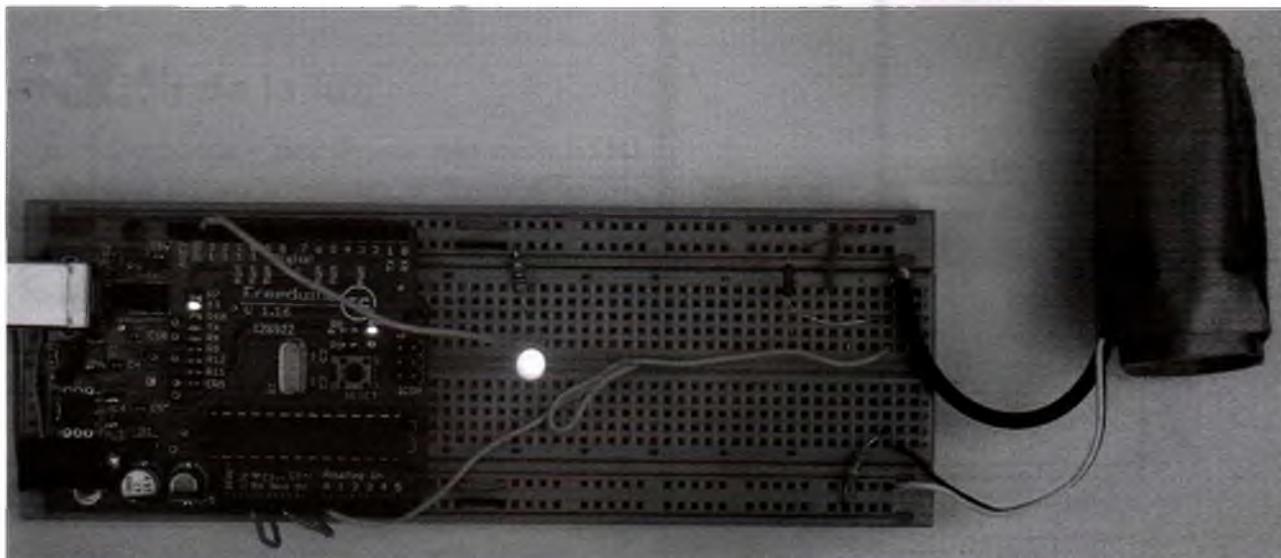


Figura 5-16 Proyecto 12. Monitor de pulsaciones.

Software

El software de este proyecto resulta un poco complicado conseguir que funcione. Por supuesto, el primer paso no es ejecutar todo el programa de comandos final sino, más bien, ejecutar un programa de comandos que recopile datos, que posteriormente podamos colocar en una hoja de cálculo y crear un gráfico para comprobar el algoritmo de "suavizado" (**smoothing**) (véanse las aclaraciones más adelante).

El programa de comandos (**script**) de prueba se proporciona en el Listado de Proyecto 12.

LISTADO DE PROYECTO 12 - SCRIPT DE PRUEBA

```
int ledPin = 13;
int sensorPin = 0;

double alpha = 0.75;
int period = 20;
double change = 0.0;

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(115200);
}

void loop()
{
    static double oldValue = 0;
    static double oldChange = 0;
    int rawValue = analogRead(sensorPin);
    double value = alpha * oldValue
        + (1 - alpha) * rawValue;

    Serial.print(rawValue);
    Serial.print(",");
    Serial.println(value);

    oldValue = value;
    delay(period);
}
```

Este programa de comandos lee la señal sin tratar de la entrada analógica y aplica la función de

"suavizado" y, a continuación, introduce ambos valores en el **Serial Monitor**, donde podemos capturarlos y pegarlos en una hoja de cálculo para análisis. Observe que las comunicaciones del **Serial Monitor** se han ajustado a su velocidad más rápida para reducir al mínimo el efecto del retraso provocado por la transmisión de datos. Cuando inicie el **Serial Monitor**, tendrá que cambiar la velocidad serie a **115200 baudios**.

La función **smoothing** (suavizado) utiliza una técnica llamada en inglés "leaky integration" (es una integración matemática imperfecta a propósito) y se puede ver en el código cuando hacemos este "suavizado" usando la línea:

```
double value = alpha * oldValue + (1 -
alpha) * rawValue;
```

La variable **alpha** es un número mayor que 0 pero menor de 1, y determina cuánto "suavizado" realizar.

Ponga el dedo en el tubo de sensor, inicie el **Serial Monitor**, y déjelo funcionando durante tres o cuatro segundos para capturar algunas pulsaciones.

A continuación, copie y pegue el texto capturado en una hoja de cálculo. Probablemente le pedirá el carácter delimitador de columna, que es una coma. En la Figura 5-17 se muestran los datos resultantes y un gráfico de líneas extraído de las dos columnas.

La curva más irregular corresponde a los datos en bruto leídos del puerto analógico, mientras que en la más uniforme se puede ver claramente que se ha eliminado la mayor parte del ruido. Si aún así todavía presentase exceso de ruido, -en concreto, picos falsos que confunden al monitor- aumente el nivel de "suavizado" disminuyendo el valor de **alpha**.

Una vez que encuentre el valor correcto de **alpha** para el montaje de su sensor, puede transferir este valor al **sketch** real y pasar a utilizar el **sketch** real en lugar del programa de prueba. El **sketch** real se presenta en el siguiente listado:

LISTADO DE PROYECTO 12

```

int ledPin = 13;
int sensorPin = 0;

double alpha = 0.75;
int period = 20;
double change = 0.0;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  static double oldValue = 0;
  static double oldChange = 0;
  int rawValue = analogRead(sensor-
Pin);
  double value = alpha * oldValue
    + (1 - alpha) * rawValue;
  change = value - oldValue;

  digitalWrite(ledPin, (change <
    0.0 && oldChange > 0.0));

  oldValue = value;
  oldChange = change;
  delay(period);
}

```

Con lo que ahora ya sólo queda el problema de detectar los picos. Si miramos la Figura 5-17, vemos que si hacemos un seguimiento de la lectura anterior, podemos comprobar que éstas van aumentando gradualmente hasta alcanzar su valor máximo, y luego descienden hacia valores negativos. Por tanto, si encendemos el LED cada vez que en la lectura se produce un cambio de valores de positivo a negativo o de negativo a positivo, obtendríamos un breve pulso del LED correspondiente al pico de cada pulso.

Pongamos todo junto

Tanto el **sketch** de prueba como el definitivo para el Proyecto 12 se encuentran en su **Arduino Sketchbook**. Para obtener instrucciones sobre cómo descargarlo a la placa, véase el Capítulo 1.

Como mencionamos al principio, conseguir que este proyecto funcione es un poco complicado. Además, ya verá que tiene que colocar el dedo justo en el lugar adecuado para comenzar a recibir el pulso. Si tiene algún problema, ejecute el **script** de prueba como se ha descrito anteriormente para

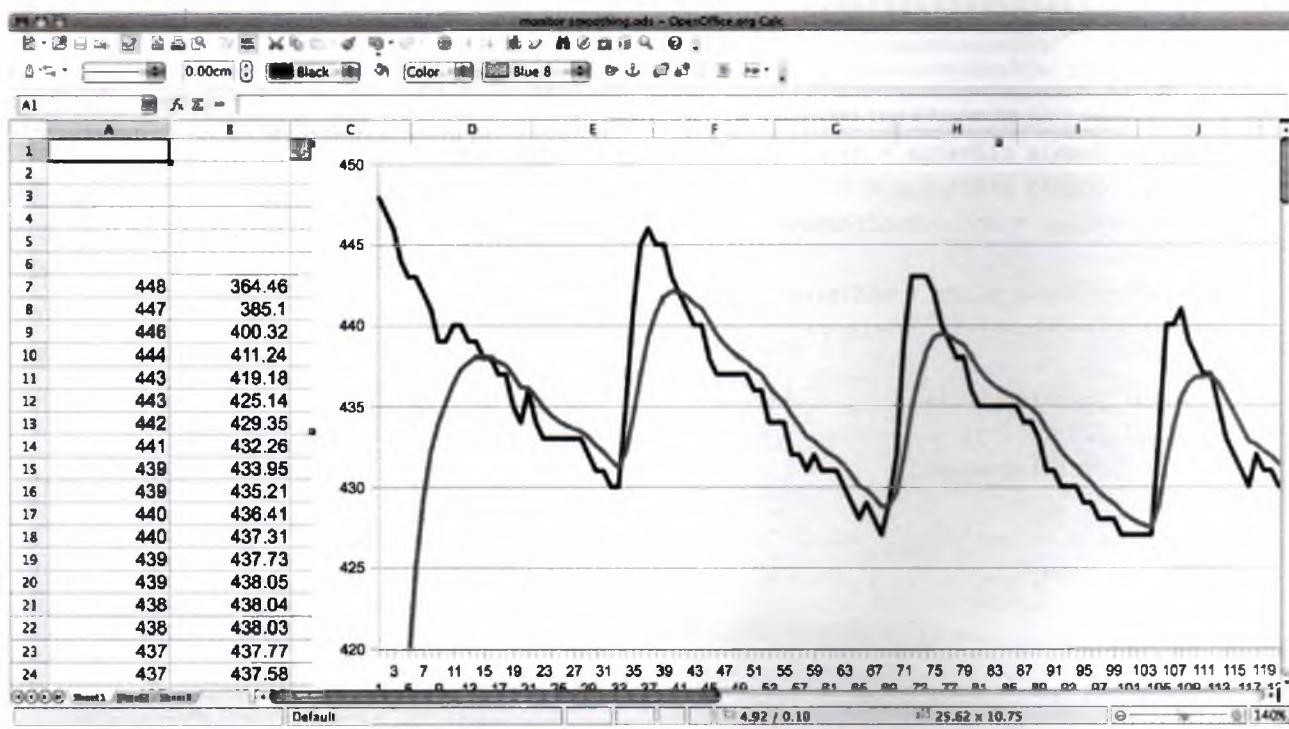


Figura 5-17 Datos de prueba del monitor cardíaco colocados en una hoja de cálculo.

verificar que el detector está obteniendo un pulso y que el factor de "suavizado" **alpha** es suficientemente bajo.

Al autor desea señalar que este dispositivo no debe utilizarse en modo alguno para ningún tipo de aplicación médica real.

Medición de temperatura

Medir la temperatura es un problema similar a la medición de la intensidad de la luz, pero en lugar de una LDR, se utiliza un dispositivo llamado **termistor**. Según aumenta la temperatura, también aumenta la resistencia del termistor.

Cuando se compra un **termistor** éste viene con una resistencia nominal específica. En este caso, el **termistor** elegido es de $33\text{ K}\Omega$. Esta será la resistencia del dispositivo a una temperatura de 25°C .

La fórmula para calcular la resistencia a una determinada temperatura viene dada por:

$$R = R_0 \exp(-\Beta/(T + 273) - \Beta/(T_0 + 273))$$

En este caso, **R_0** es la resistencia a una temperatura de 25°C ($33\text{ K}\Omega$) y **β** es un valor constante que encontrará en la hoja de características del termistor. En este caso, su valor es 4090.

Por lo tanto,

$$R = R_0 \exp(\Beta/(T + 273) - \Beta/298)$$

Reordenando esta fórmula, podemos obtener una expresión para la temperatura conociendo la resistencia.

$$R = R_0 \exp(\Beta/(T + 273) - \Beta/298)$$

Si utilizamos una resistencia fija de $33\text{ K}\Omega$, podemos calcular el voltaje en la entrada analógica utilizando la fórmula:

$$V = 5 * 33\text{K}/(R + 33\text{K})$$

así,

$$R = ((5 * 33\text{K})/V) - 33\text{K}$$

Dado que el valor analógico "**a**" viene dado por:

$$a = V * (1023/5)$$

entonces,

$$V = a/205$$

y

$$R = ((5 * 33\text{K}) * 205)/a - 33\text{K}$$

$$R = (1025 * 33\text{K}/a) - 33\text{K}$$

Podemos reorganizar nuestra fórmula para obtener una temperatura desde la entrada analógica, leyendo "**a**" como:

$$T = \Beta/(\ln(R/33) + (\Beta/298)) - 273$$

y, así, finalmente obtenemos:

$$T = \Beta/(\ln(((1025 * 33/a) - 33)/33) + (\Beta/298)) - 273$$

¡Guau! ¡Vaya montón de matemáticas!

Usaremos este cálculo en el siguiente proyecto para crear un registrador de temperaturas.

Proyecto 13 Registrador de temperaturas USB

Este proyecto lo controla el ordenador, pero una vez que se la han dado las instrucciones, se puede desconectar y utilizar con pilas para que almacene los datos. Cuando registra, guarda sus datos y, posteriormente, cuando el registrador se vuelve a conectar al PC, transferirá sus datos de nuevo utilizando la conexión USB, desde donde se pueden importar a una hoja de cálculo. De forma predeterminada, el registrador grabará 1 muestra cada cinco minutos, y puede registrar hasta 255 muestras.

Para instruir al registrador de temperaturas desde el ordenador, tendremos que definir algunos comandos que pueden ser emitidos desde el equipo. Vea estos comandos en la Tabla 5-1.

TABLA 5-1 Comandos del registrador de temperaturas

- R Lee los datos del registrador como texto CSV
- X Borra todos los datos del registrador
- C Modo Centígrados
- F Modo Fahrenheit
- 1-9 Establece el período de muestras en minutos, de 1 a 9
- G ¡Vamos! Empieza a registrar temperaturas
- ? Informa del estado del dispositivo, número de muestras tomadas, etc.

Este proyecto sólo necesita un termistor y una resistencia.

COMPONENTES Y EQUIPO

Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
R1 Termistor, 33K a 25°C, Beta 4090	18
R2 Resistencia 33 KΩ 0,5 W	10

- Si no puede conseguir un termistor con el valor correcto de beta, o la resistencia, puede cambiar estos valores en el sketch.

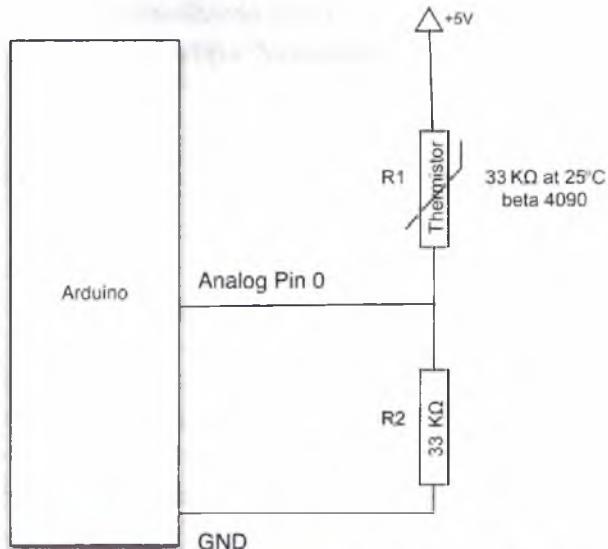
Hardware

El circuito electrónico del Proyecto 13 se muestra en la Figura 5-18.

El montaje es tan sencillo que podemos simplemente introducir los terminales del termistor y de la resistencia en la placa Arduino, como se muestra en la Figura 5-19.

Software

El software de este proyecto es un poco más complejo que el de otros anteriores (véase el Listado del Proyecto 13). Hasta la fecha, todas las variables que hemos utilizado en nuestros **sketches** (programas)

**Figura 5-18** Circuito eléctrico del Proyecto 13.

se "olvidan" tan pronto como la placa Arduino se reinicia o se desconecta de la alimentación. A veces puede ser interesante almacenar los datos de forma continua para que sigan allí la próxima vez que iniciamos la placa. Esto puede hacerse utilizando un tipo especial de memoria de la Arduino llamada **EEPROM**, que significa: memoria programable de sólo lectura borrable eléctricamente. La placa Arduino UNO cuenta con 1024 bytes de memoria EEPROM.

Este es el primer proyecto en el que hemos utilizado la **EEPROM** de la Arduino para almacenar datos, de modo que no se borren cuando la placa se reinicia o se desconecta de la alimentación. Esto significa que una vez que hayamos montado nuestro registro de almacenamiento de datos, podemos desconectarlo del cable USB y dejar que funcione con baterías. Incluso si se acaban las baterías, nuestros datos seguirán allí la próxima vez que lo conectemos.

Notará que en la parte superior de este **sketch** utilizamos el comando **#define** para lo que en el pasado hubiéramos usado variables. En realidad, esta es una manera más eficiente de definir constantes, es decir, valores que no van a cambiar durante la ejecución del **sketch**. Por ello, es realmente adecuado para la configuración de los pines y de constantes como **beta**. El comando **#define** es lo que se llama una directiva de pre-procesador, y lo que sucede es que, poco antes de que se compile el **sketch**, todas las apariciones con este nombre que se hallen en cualquier lugar del **sketch** serán sustituidas por el valor que se le ha asignado.

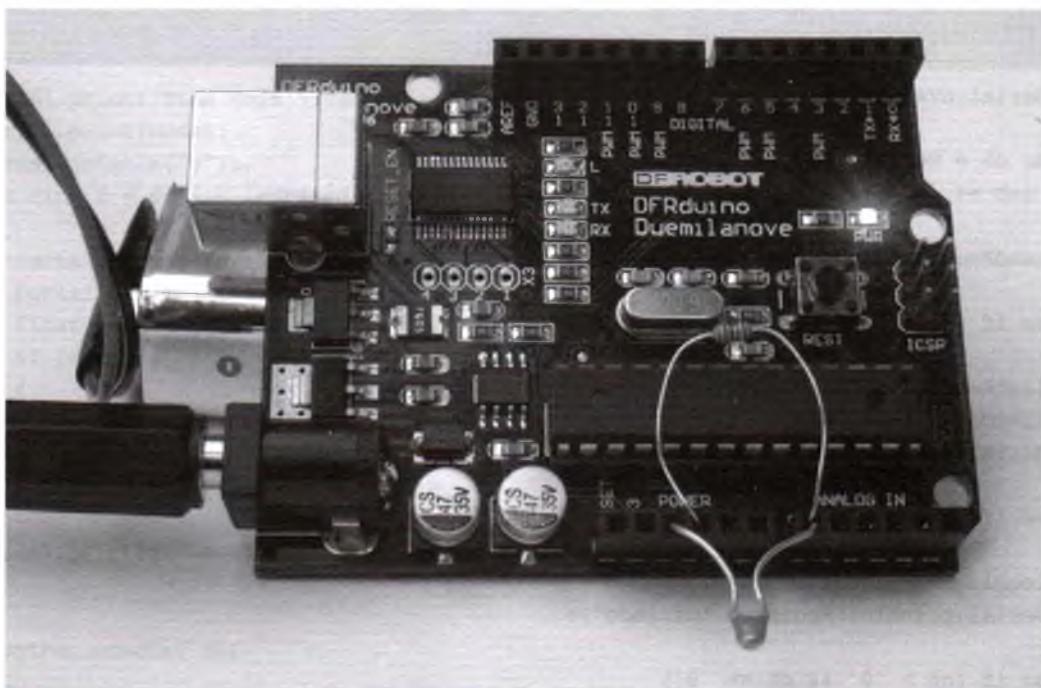


Figura 5-19 Una placa Arduino alimentada con el LED encendido.

LISTADO DE PROYECTO 13

```
#include <EEPROM.h>

#define ledPin 13
#define analogPin 0
#define maxReadings 255
#define beta 4090           // de hoja de características del termistor
#define resistance 33

float readings[maxReadings];
int lastReading = EEPROM.read(0);
boolean loggingOn = false;
long period = 300;
long count = 0;
char mode = 'C';

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
    Serial.println("Preparado");
}

void loop()
{
```

(continúa)

LISTADO DE PROYECTO 13 (continúa)

```
if (Serial.available())
{
    char ch = Serial.read();
    if (ch == 'r' || ch == 'R')
    {
        sendBackdata();
    }
    else if (ch == 'x' || ch == 'X')
    {
        lastReading = 0;
        EEPROM.write(0, 0);
        Serial.println("Datos borrados");
    }
    else if (ch == 'g' || ch == 'G')
    {
        loggingOn = true;
        Serial.println("Registro iniciado");
    }
    else if (ch > '0' && ch <= '9')
    {
        setPeriod(ch);
    }
    else if (ch == 'c' or ch == 'C')
    {
        Serial.println("Modo configurado a °C");
        mode = 'C';
    }
    else if (ch == 'f' or ch == 'F')
    {
        Serial.println("Modo configurado a °F");
        mode = 'F';
    }
    else if (ch == '?')
    {
        reportStatus();
    }
}
if (loggingOn && count > period)
{
    logReading();
    count = 0;
}
count++;
delay(1000);
}

void sendBackdata()
{
    loggingOn = false;
    Serial.println("Registro detenido");
    Serial.println("----- cortar aquí -----");
}
```

LISTADO DE PROYECTO 13 (continúa)

```
Serial.print("Time (min)\tTemp (");
Serial.print(mode);
Serial.println(")");
for (int i = 0; i < lastReading; i++)
{
    Serial.print((period * i) / 60);
    Serial.print("\t");
    float temp = getReading(i);
    if (mode == 'F')
    {
        temp = (temp * 9) / 5 + 32;
    }
    Serial.println(temp);
}
Serial.println("----- cortar aquí -----");
}

void setPeriod(char ch)
{
    int periodMins = ch - '0';
    Serial.print("Período de muestras establecido en: ");
    Serial.print(periodMins);
    Serial.println(" mins");
    period = periodMins * 60;
}

void logReading()
{
    if (lastReading < maxReadings)
    {
        long a = analogRead(analogPin);
        float temp = beta / (log(((1025.0 * resistance / a) - 33.0) / 33.0) +
            (beta / 298.0)) - 273.0;
        storeReading(temp, lastReading);
        lastReading++;
    }
    else
    {
        Serial.println("¡Llena! registro detenido");
        loggingOn = false;
    }
}

void storeReading(float reading, int index)
{
    EEPROM.write(0, (byte)index);           // almacena número muestras en byte 0
    byte compressedReading = (byte)((reading + 20.0) * 4);
    EEPROM.write(index + 1, compressedReading);
}
```

(continúa)

LISTADO DE PROYECTO 13 (continúa)

```

float getReading(int index)
{
    lastReading = EEPROM.read(0);
    byte compressedReading = EEPROM.read(index + 1);
    float uncompressesReading = (compressedReading / 4.0) - 20.0;
    return uncompressesReading;
}

void reportStatus()
{
    Serial.println("-----");
    Serial.println("Estado");
    Serial.print("Período muestras\t");
    Serial.println(period / 60);
    Serial.print("Núm. lecturas\t");
    Serial.println(lastReading);
    Serial.print("Modo grados\t");
    Serial.println(mode);
    Serial.println("-----");
}

```

tuidas por su valor. En realidad, utilizar **#define** o una variable, es una cuestión de gusto personal.

Afortunadamente, la lectura y escritura de la **EEPROM** se produce un byte tras otro. Por lo tanto, si queremos escribir una variable que sea un **byte** o un **char**, simplemente podemos utilizar las funciones **EEPROM.write** y **EEPROM.read**, tal como se muestra en el siguiente ejemplo:

```

char letterToWrite = 'A';
EEPROM.write(0, myLetter);

char letterToRead;
letterToRead = EEPROM.read(0);

```

Los 0 en los parámetros de lectura y escritura es la dirección en la **EEPROM** que se debe utilizar. Esto puede ser cualquier número entre 0 y 1023, y cada dirección representa un lugar donde se almacena un byte.

En este proyecto queremos almacenar tanto la posición de la última lectura tomada (en la variable **lastReading**) como el resto de las lecturas. Así que grabaremos **lastReading** en el primer byte de

EEPROM y, a continuación, los datos de lectura reales en los 256 bytes que siguen.

Cada lectura de la temperatura se guarda en un valor decimal (**float**) y, si lo recuerda del Capítulo 2, un decimal (**float**) ocupa 4 bytes de datos. Aquí teníamos que elegir: Podíamos o bien almacenar los 4 bytes o encontrar una manera de codificar la temperatura en un único byte. Decidimos tomar esta última vía, ya que es más fácil de hacer.

Para poder codificar la temperatura en un único byte, tenemos que hacer algunas concreciones. En primer lugar, vamos a asumir que cualquier temperatura en grados Centígrados será entre -20° y +40°. De todos modos, cualquier cosa que fuera más alto o más bajo probablemente dañaría nuestra placa Arduino. En segundo lugar, vamos a asumir que sólo necesitamos saber la temperatura con una precisión de un cuarto de grado.

Con estas dos premisas, podremos tomar cualquier valor de temperatura que obtengamos de la entrada analógica, agregarle 20, multiplicarlo por 4, y aún así estar seguro de que siempre tendremos un número entre 0 y 240. Puesto que un byte puede contener un número entre 0 y 255, encaja perfectamente.

Cuando sacamos nuestros números de la **EEPROM**, tenemos que volver a convertirlos en decimal (**float**), lo que podemos hacer invirtiendo el proceso, es decir, dividiendo por 4 y luego restando 20.

Tanto la codificación como la decodificación de los valores se han incluido en las funciones **storeReading** y **getReading**. Así, si decidimos utilizar un enfoque diferente para almacenar los datos, sólo tendríamos que cambiar estas dos funciones.

Pongamos todo junto

Cargue el **sketch** terminado del Proyecto 13 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Ahora abra **Serial Monitor** (Figura 5-20) y, para fines de ensayo, estableceremos el registrador de temperaturas para registrar cada minuto, lo que hacemos introduciendo 1 en **Serial Monitor**. La placa debería responder con el mensaje "Periodo de muestras establecido en: 1 mins." Si quisiéramos, podríamos cambiar el modo a Fahrenheit tecleando F en el **Serial Monitor**. Ahora podemos comprobar el estado del registrador introduciendo ?.

Si queremos que el registrador continúe registrando después de desconectar el cable USB, primero debemos alimentarlo con una fuente de alimentación

externa, como el cable para la pila de 9 V que hicimos en el Proyecto 6.

Por último, podemos introducir el comando G para iniciar el registro. Entonces ya podemos desenchufar el cable USB y dejar nuestro registrador funcionando con las pilas. Tras esperar unos 10 o 15 minutos, podemos volver a conectarlo y consultar los datos que tenemos abriendo **Serial Monitor** e introduciendo el comando R, cuyos resultados se muestran en la Figura 5-21. Seleccione todos los datos, incluyendo los encabezados de **Time** y **Temp** de la parte superior.

Copie el texto en el portapapeles (pulse **CTRL-C** en Windows y Linux, **ALT-C** en Mac), abra una hoja de cálculo en un programa como Microsoft Excel, y péguelos en una nueva hoja de cálculo (Figura 5-22).

Una vez que se encuentre en la hoja de cálculo, podemos incluso dibujar un gráfico utilizando nuestros datos.

Resumen

Ahora sabemos cómo manejar diversos tipos de sensores y dispositivos de entrada para acompañar a nuestros conocimientos sobre los LED. En la próxima sección veremos una serie de proyectos que utilizan la luz de diversas maneras y pondremos nuestras manos sobre algunas tecnologías de visualización más avanzadas, como los displays LCD de texto y los de siete segmentos.

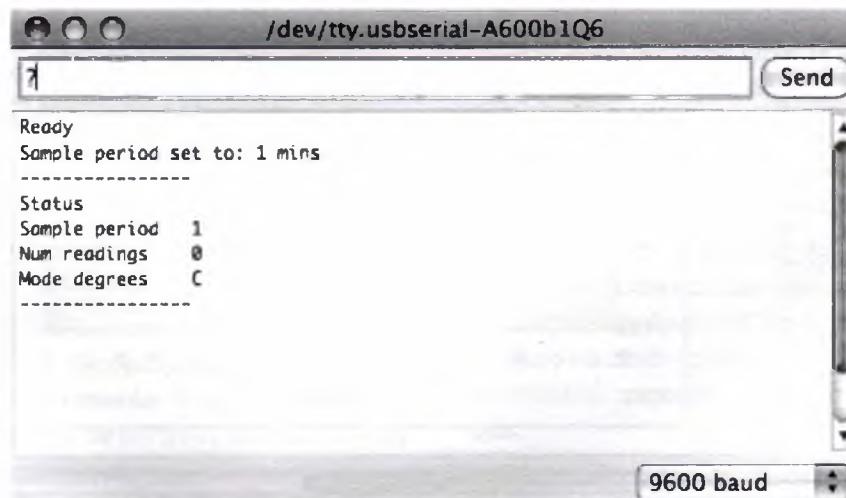


Figura 5-20 Emisión de comandos a través del **Serial Monitor**.

```

Logging stopped
----- cut here -----
Time (min) Temp (C)
0 19.50
5 19.25
10 19.25
15 19.00
20 19.00
25 18.75
30 18.75
35 19.00
40 19.00
45 18.75
50 18.75
55 18.75
60 18.75
65 18.75
70 19.00
75 18.75
80 18.75
85 18.50
90 18.75
95 18.50
100 18.75
105 18.50
110 18.50
115 18.75
120 18.50
125 18.50
130 18.50
135 18.25
140 18.25
145 18.25
150 18.25
155 18.25
----- cut here -----

```

9600 baud

Figura 5-21 Datos a copiar y pegar en una hoja de cálculo.

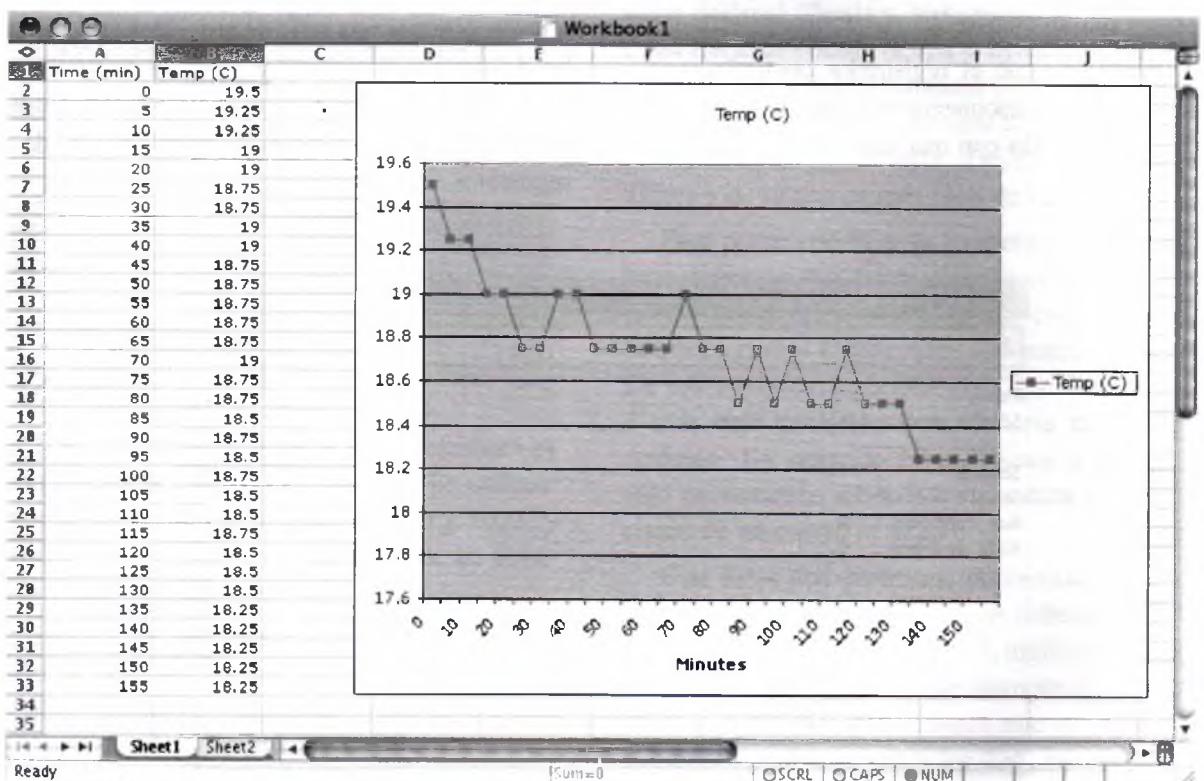


Figura 5-22 Datos de temperatura importados en una hoja de cálculo.

CAPÍTULO 6

Proyectos de luz

EN ESTE CAPÍTULO, vamos a ver algunos proyectos más basados en luces y en **displays**. En concreto, veremos la forma de utilizar LEDs multicolores, **displays de siete segmentos**, **displays de matrices de LEDs** y **displays de cristal líquido (LCD)**.

Proyecto 14 LED multicolor

Este proyecto utiliza un LED de tres colores de gran potencia en combinación con un codificador giratorio. Al girar el codificador giratorio cambia el color mostrado por el LED.

COMPONENTES Y EQUIPO

④	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o similar	1
D1	LED RGB (ánodo común)	31
R1-R3	Resistencia 100 Ω 0,5 W	5
R4-6	Resistencia 100 KΩ 0,5 W	13
S1	Codificador giratorio con pulsador	57

La lámpara LED es interesante porque tiene tres luces de LED en un paquete de cuatro pines. El LED es del tipo **ánodo común**, lo que significa que las conexiones positivas de los tres LED salen de un mismo pin (pin 2).

Si no pudiera encontrar un LED RGB (rojo, verde, azul) de cuatro pines, en su lugar puede utilizar uno de seis. Simplemente, consulte la hoja de características del componente para conectar entre sí los dos ánodos.

Hardware

La Figura 6-1 muestra el esquema electrónico del Proyecto 14, y la Figura 6-2 el diseño de la placa de pruebas.

Se trata de un esquema electrónico simple. El codificador giratorio tiene resistencias de puesta a masa para los sensores de dirección y el interruptor pulsador. Para obtener más detalles sobre los codificadores giratorios y su funcionamiento, véase el Capítulo 5.

Cada LED tiene su propia resistencia en serie para limitar la corriente a unos 30 mA por LED.

El encapsulado del LED tiene una parte plana en uno de los lados. El pin 1 es el más cercano al lado plano. Otra manera de identificar los pines es por su longitud. El pin 2 es el de mayor longitud, y corresponde al ánodo común.

En la Figura 6-3 se muestra el montaje completo del proyecto.

Cada uno de los LEDs (rojo, verde y azul) se maneja desde una de las salidas de **modulación por anchura de pulsos** (PWM) de la placa Arduino, por lo que, variando la salida de cada LED, podemos producir el espectro completo de los colores visibles.

El codificador giratorio se conecta exactamente igual que en el Proyecto 11: girándolo cambia el color y pulsándolo enciende y apaga el LED.

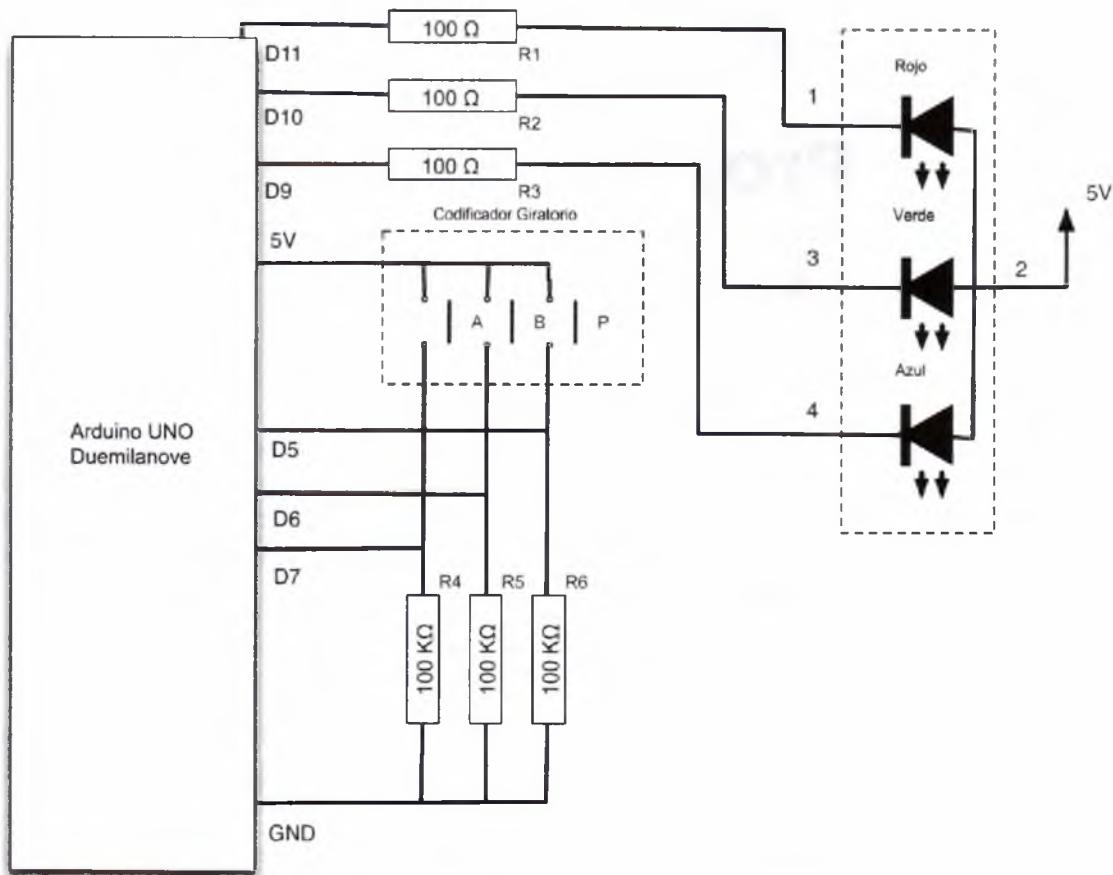


Figura 6-1 Circuito electrónico del Proyecto 14

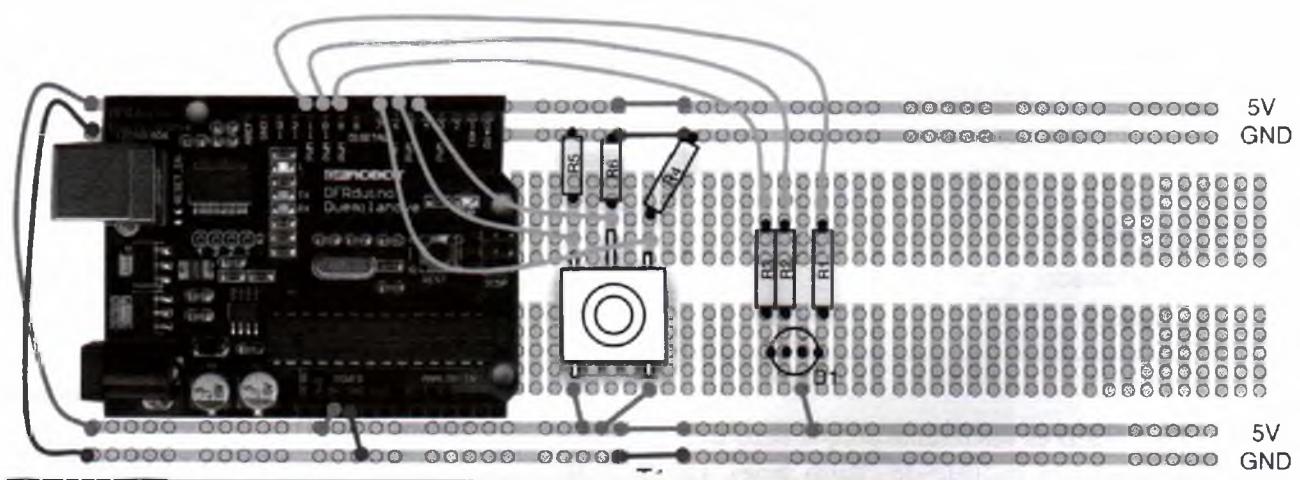


Figura 6-2 Diseño del circuito del Proyecto 14 sobre la placa de pruebas.

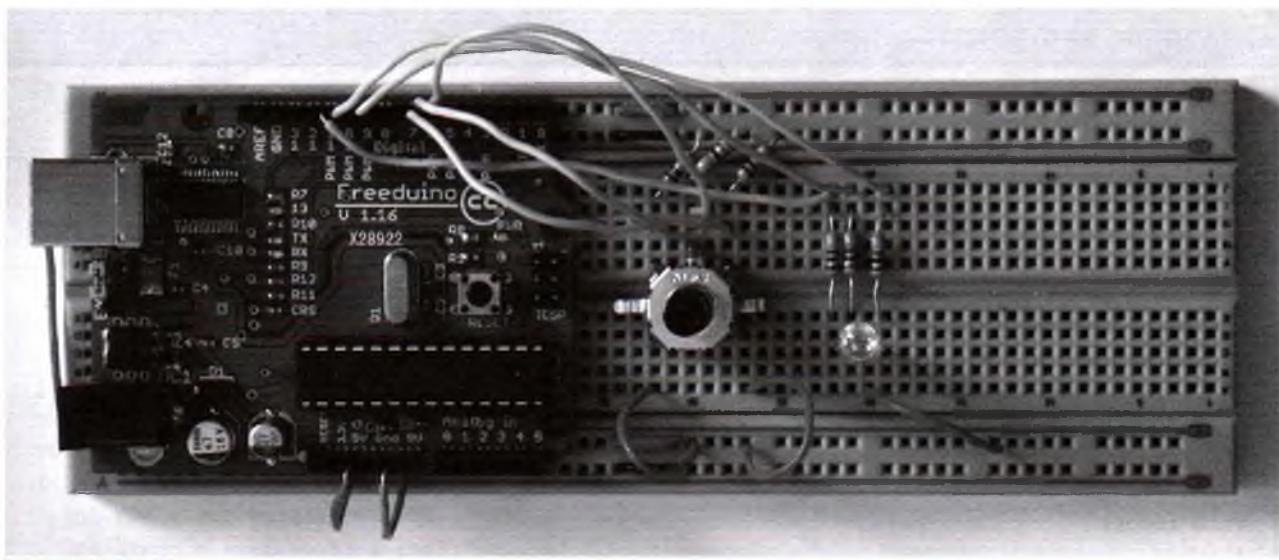


Figura 6-3 Proyecto 14. Led multicolor.

Software

Este **sketch** (Listado del Proyecto 14) utiliza una matriz (**array**) para representar los diferentes colores que serán mostrados por el LED. Cada uno de los elementos de la matriz es un número **long** de 32 bits. Tres de los bytes del número **long** se utilizan para representar los componentes rojo, verde y azul del color, lo que corresponde a la intensidad con la

que debe brillar cada uno de los elementos LED rojo, verde o azul. Los números del **array** se muestran en formato hexadecimal, que es uno de los formatos utilizados para representar los colores de 24-bit de las páginas web. Si desea crear algún color en particular, busque una tabla de colores web tecleando "tabla de colores html" en su buscador favorito. Allí podrá consultar el valor hexadecimal del color que desee.

LISTADO DEL PROYECTO 14

```

int redPin = 9;
int greenPin = 10;
int bluePin = 11;
int aPin = 6;
int bPin = 7;
int buttonPin = 5;

boolean isOn = true;
int color = 0;

long colors[48] = {
  0xFF2000, 0xFF4000, 0xFF6000, 0xFF8000, 0xFFA000, 0xFFC000, 0xFFE000, 0xFFFF00,
  0xE0FF00, 0xC0FF00, 0xA0FF00, 0x80FF00, 0x60FF00, 0x40FF00, 0x20FF00, 0x00FF00,
  0x00FF20, 0x00FF40, 0x00FF60, 0x00FF80, 0x00FFA0, 0x00FFC0, 0x00FFE0, 0x00FFF0,
  0x00E0FF, 0x00C0FF, 0x00A0FF, 0x0080FF, 0x0060FF, 0x0040FF, 0x0020FF, 0x0000FF,
  0x2000FF, 0x4000FF, 0x6000FF, 0x8000FF, 0xA000FF, 0xC000FF, 0xE000FF, 0xFF00FF,
}

```

(continúa)

LISTADO DEL PROYECTO 14 (continúa)

```
0xFF00E0, 0xFF00C0, 0xFF00A0, 0xFF0080, 0xFF0060, 0xFF0040, 0xFF0020, 0xFF0000
};

void setup()
{
    pinMode(aPin, INPUT);
    pinMode(bPin, INPUT);
    pinMode(buttonPin, INPUT);
    pinMode(redPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    pinMode(bluePin, OUTPUT);
}

void loop()
{
    if (digitalRead(buttonPin))
    {
        isOn = ! isOn;
        delay(200);      // de-bounce
    }
    if (isOn)
    {
        int change = getEncoderTurn();
        color = color + change;
        if (color < 0)
        {
            color = 47;
        }
        else if (color > 47)
        {
            color = 0;
        }
        setColor(colors[color]);
    }
    else
    {
        setColor(0);
    }
    delay(1);
}

int getEncoderTurn()
{
    // devuelve -1, 0, or +1
    static int oldA = LOW;
    static int oldB = LOW;
    int result = 0;
    int newA = digitalRead(aPin);
    int newB = digitalRead(bPin);
    if (newA != oldA || newB != oldB)
    {
```

(continúa)

LISTADO DEL PROYECTO 14 (continúa)

```

// algo ha cambiado
if (oldA == LOW && newA == HIGH)
{
    result = -(oldB * 2 - 1);
}
oldA = newA;
oldB = newB;
return result;
}

void setColor(long rgb)
{
    int red = rgb >> 16;
    int green = (rgb >> 8) & 0xFF;
    int blue = rgb & 0xFF;
    analogWrite(redPin, 255 - red);
    analogWrite(greenPin, 255 - green);
    analogWrite(bluePin, 255 - blue);
}

```

Los 48 colores de la matriz se han seleccionado de una de éstas tablas y representan una gama de colores que abarcan aproximadamente todo el rango desde el rojo al violeta.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 14 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

LEDs de siete segmentos

Hubo un tiempo en que estaba de moda llevar un reloj de LEDs. Esto exigía que el portador tuviera que pulsar un botón del reloj para que la hora apareciera por arte de magia en forma de cuatro dígitos rojos brillantes. Con el tiempo, el inconveniente de tener que utilizar ambos dedos para decir la hora superó la novedad del reloj digital, con lo que las mentes más inquietas se fueron de compras y en su lugar se pusieron un reloj LCD. Estos sólo podían leerse con una brillante luz solar.

Los LEDs de siete segmentos (consulte la Figura 6-4) han sido prácticamente sustituidos por **displays LCD** retroiluminados (véase más adelante en este capítulo), aunque todavía siguen resultando útiles de vez en cuando. Y también añaden sentido a ciertos proyectos ingeniosos.

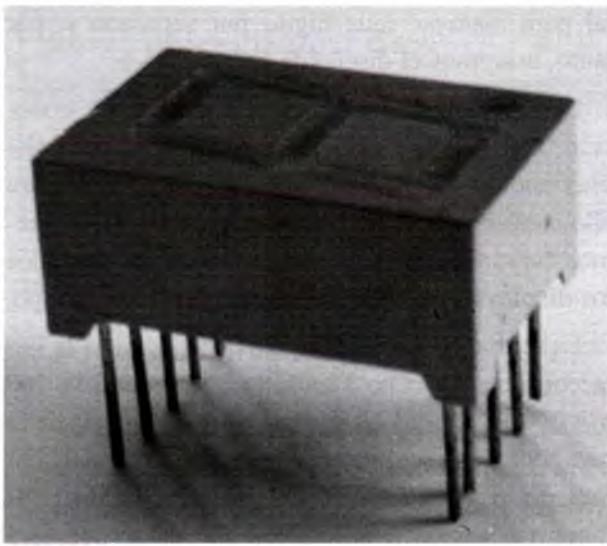


Figura 6-4 Display de siete segmentos.

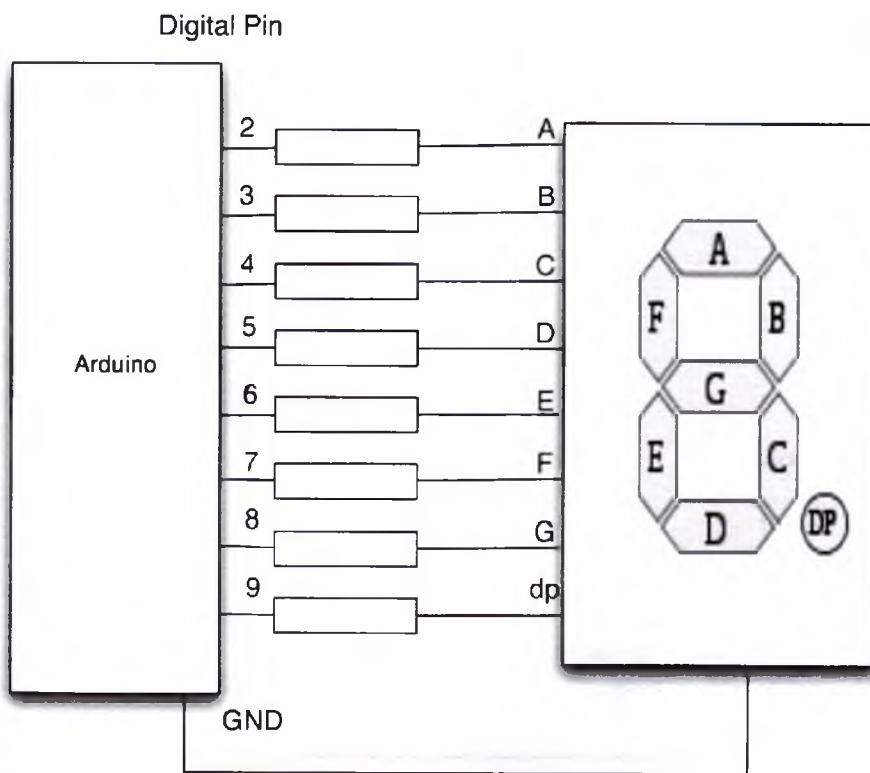


Figura 6-5 Placa Arduino manejando un display de siete segmentos.

La Figura 6-5 muestra el circuito para manejar un único **display** de siete segmentos.

Un sólo **display** de siete segmentos no es realmente de mucha utilidad. La mayoría de los proyectos necesitarán dos o cuatro dígitos. Pero, en ese caso, no tendremos suficientes pines de salida digital para manejar cada dígito por separado y, por tanto, usaremos el diseño de la Figura 6-6.

De forma bastante parecida a nuestro ya conocido "barrido" de teclado, vamos a activar cada **display** uno tras otro y a configurar los segmentos para ello antes de pasar al siguiente dígito. Esto lo haremos tan rápido que se creará la ilusión de que todos los **displays** se están encendiendo al mismo tiempo.

En teoría, cada **display** podría consumir a la vez la corriente de ocho LEDs, lo que requeriría 160 mA (a 20 mA por LED), que sería mucho más de lo que podemos sacar de un pin de salida digital. Por esta razón, para activar uno tras otro los distintos **displays**, vamos a utilizar un transistor que será controlado por una salida digital.

El tipo de transistor que estamos utilizando se llama **transistor bipolar**. Tiene tres conexiones: emisor, base y colector. Cuando una corriente circula a través de la base del transistor y pasa por el emisor, permite que desde el colector al emisor pueda circular una corriente mucho mayor. Este tipo de transistor ya lo vimos en el Proyecto 4, donde lo hemos usado para controlar la corriente a un LED Luxeon de alta potencia.

Dado que la corriente que circula del colector al emisor ya se encuentra limitada por las resistencias serie de los LED, no necesitamos limitarla de nuevo. Sin embargo, sí necesitamos limitar la corriente que circula por la base. La mayoría de los transistores multiplicarán la corriente de base por un factor de 100 o más, así que sólo necesitamos permitir que circulen por la base alrededor de 2 mA para activar completamente el transistor.

Los transistores tienen la interesante propiedad de que, en condiciones normales, la tensión entre la base y el emisor es de 0,6 V bastante constante, con independencia de la corriente que circule. Por tanto,

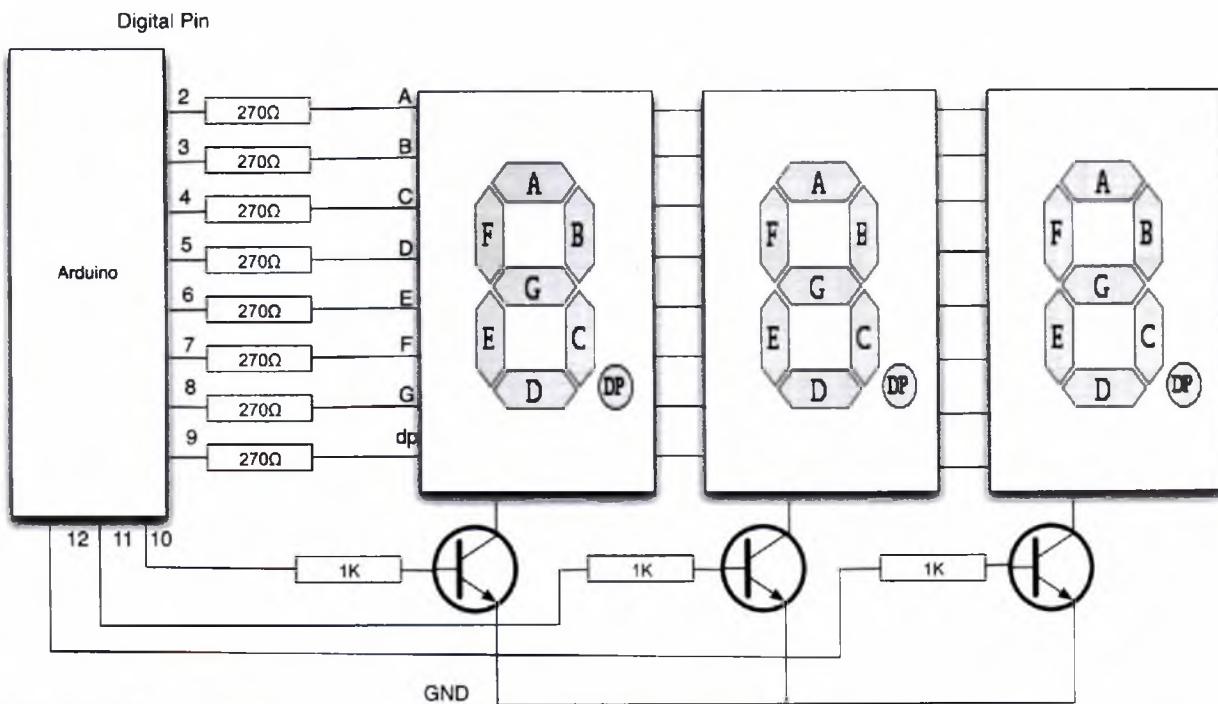


Figura 6-6 Manejo de más de un display de siete segmentos desde una placa Arduino.

si nuestro pin Arduino suministra 5 V, 0,6 V de esos circularán por el circuito base/emisor del transistor, lo que significa que nuestra resistencia debe tener un valor de alrededor de:

$$R = V/I$$

$$R = 4,4 / 2 \text{ mA} = 2,2 \text{ K}\Omega$$

De hecho, también sería correcto si dejamos circular 4 mA, porque la salida digital puede aguantar casi 40 mA. Por ello, vamos a elegir el socorrido valor estándar de 1 KΩ, que nos permitirá asegurarnos de que el transistor actuará como un interruptor y siempre lo activará y desactivará completamente.

COMPONENTES Y EQUIPO		
	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o similar	1
D1	Display LED siete segm. dos dígitos (ánodo común)	33
R1	Resistencia 100 KΩ 0,5 W	13
R4-13	Resistencia 270 Ω 0,5 W	6
R2, R3	Resistencia 1 KΩ 0,5 W	7
T1, T2	BC307	39
S1	Pulsador miniatura	48

Proyecto 15 Dado doble con display de siete segmentos

En el Proyecto 9 creamos un dado utilizando siete LEDs independientes. En este proyecto vamos a usar dos **displays LED** de siete segmentos para crear un dado doble.

Hardware

El sketch de este proyecto se muestra en la Figura 6-7.

El módulo **display LED** de siete segmentos que estamos utilizando es del tipo **ánodo común**, lo cual significa que todos los ánodos (terminales positivos) de los segmentos LED están conectados entre sí. Por lo tanto, para activar los **displays** uno tras otro, debe-

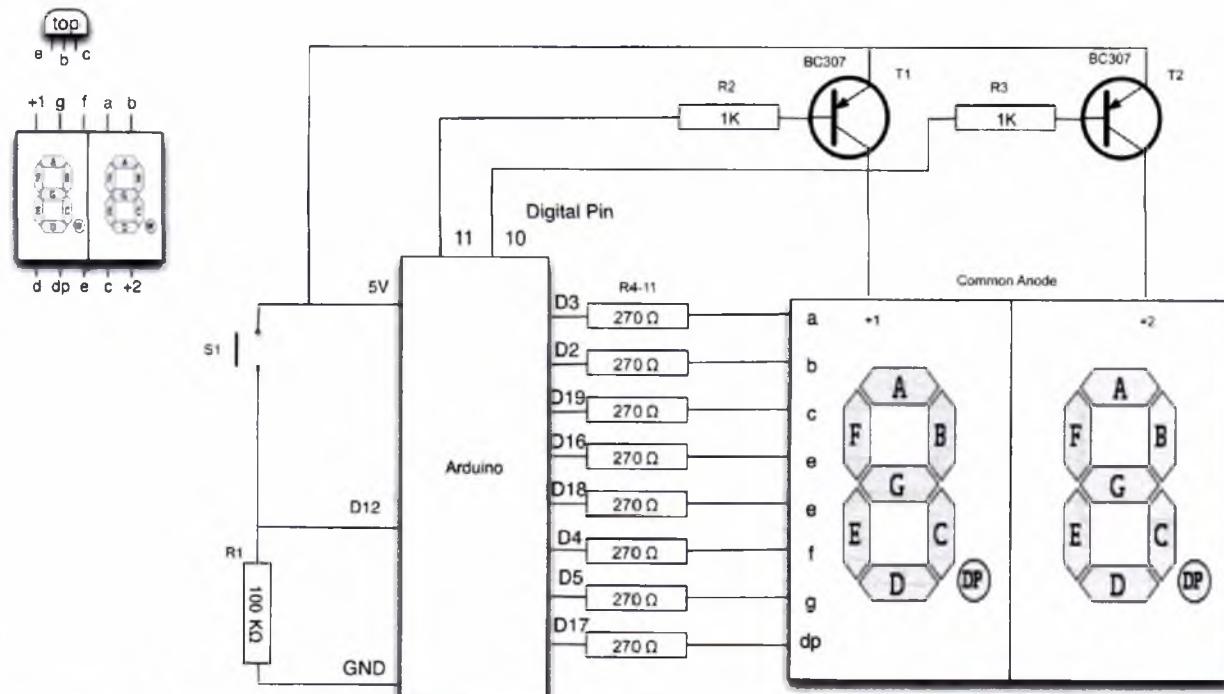


Figura 6-7 Circuito electrónico del Proyecto 15.

mos controlar la alimentación positiva, de forma secuencial, a cada uno de los dos ánodos comunes.

Como vemos, esto es lo contrario a como controlamos la alimentación al LED Luxeon del Proyecto 4, donde se controlaba la alimentación por el lado de masa del circuito. Y por eso vamos a utilizar otro tipo distinto de transistor. En lugar del **transistor NPN** (negativo-positivo-negativo) que utilizamos anteriormente, vamos a utilizar un **transistor PNP** (positivo-negativo-positivo). Para identificarlo, observe la diferente dirección de la flecha (hacia fuera o hacia dentro) en el símbolo del transistor.

Si estuviéramos utilizando un **display** de siete segmentos de **cátodo común**, entonces tendríamos que utilizar un **transistor NPN**, pero conectado en la parte inferior del circuito, en lugar de hacerlo en la parte superior.

El diseño y la fotografía de la placa de pruebas del proyecto se muestran en las Figuras 6-8 y 6-9.

Para reducir el número de cables necesarios, hemos colocado el **display** cerca de la placa Arduino, para que las resistencias entre los terminales de ésta y la

placa de pruebas puedan conectarse directamente. Como las conexiones de estas resistencias no llevan aislamiento y son relativamente largas, hay que tener cuidado para que no se toquen entre sí. Lo que es igualmente válido para las conexiones desde los pines de Arduino a cada uno de los segmentos del display. La colocación se ha hecho así para facilitar la conexión.

Software

Utilizaremos una matriz (**array**) que contenga los pines que se conectan a cada uno de los segmentos "a" a "g" y el punto decimal. También utilizaremos una matriz para determinar qué segmentos deben iluminarse para mostrar los distintos dígitos. Es un **array** de dos dimensiones, donde cada fila representa un dígito (0 a 9) y cada columna un segmento (consulte el Listado del Proyecto 15).

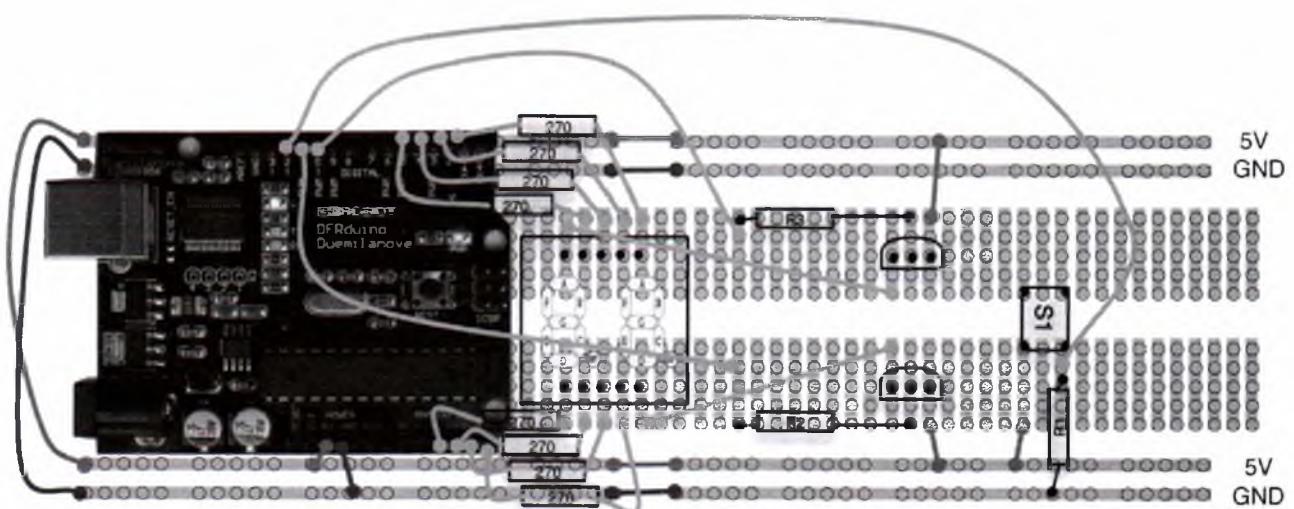


Figura 6-8 Diseño del circuito del Proyecto 15 sobre la placa de pruebas.

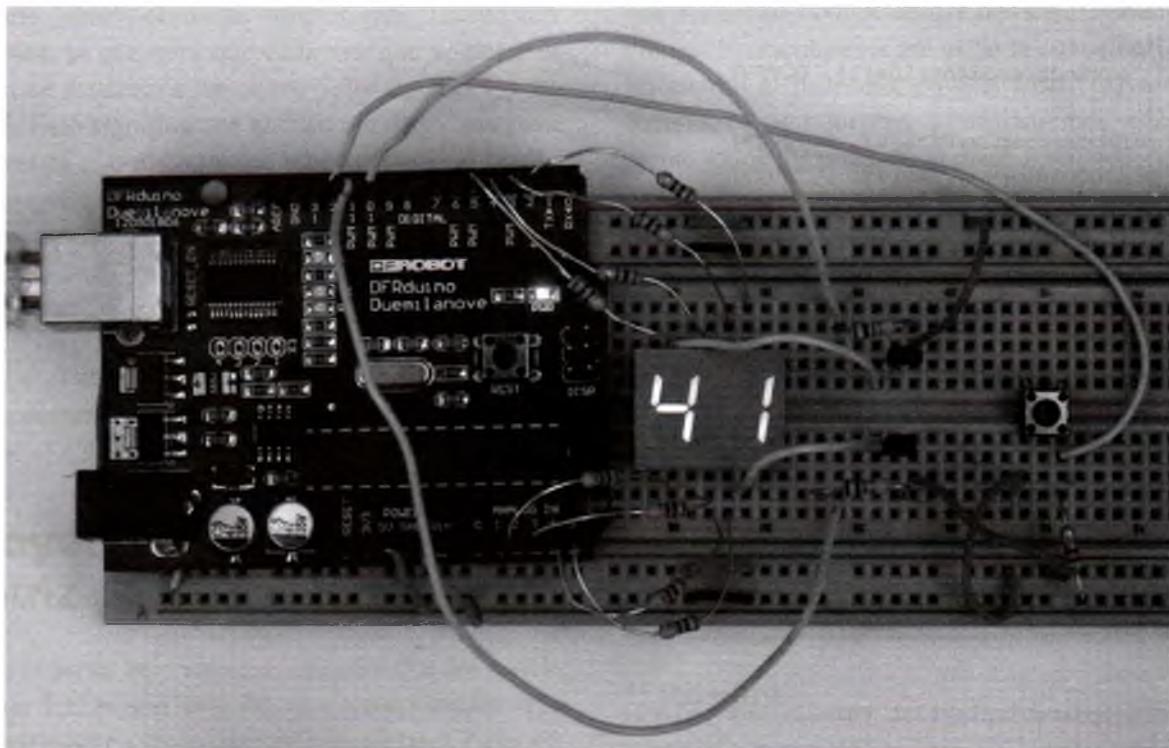


Figura 6-9 Proyecto 15. Display doble de LEDs de siete segmentos.

LISTADO DE PROYECTO 15

```
int segmentPins[] = {3, 2, 19, 16, 18, 4, 5, 17};
int displayPins[] = {10, 11};

int buttonPin = 12;

byte digits[10][8] =
// a b c d e f g .
{ {1, 1, 1, 1, 1, 1, 0, 0}, // 0
{ 0, 1, 1, 0, 0, 0, 0, 0}, // 1
{ 1, 1, 0, 1, 1, 0, 1, 0}, // 2
{ 1, 1, 1, 1, 0, 0, 1, 0}, // 3
{ 0, 1, 1, 0, 0, 1, 1, 0}, // 4
{ 1, 0, 1, 1, 0, 1, 1, 0}, // 5
{ 1, 0, 1, 1, 1, 1, 1, 0}, // 6
{ 1, 1, 1, 0, 0, 0, 0, 0}, // 7
{ 1, 1, 1, 1, 1, 1, 1, 0}, // 8
{ 1, 1, 1, 1, 0, 1, 1, 0} // 9
};

void setup()
{
    for (int i=0; i < 8; i++)
    {
        pinMode(segmentPins[i], OUTPUT);
    }
    pinMode(displayPins[0], OUTPUT);
    pinMode(displayPins[1], OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop()
{
    static int dice1;
    static int dice2;
    if (digitalRead(buttonPin))
    {
        dice1 = random(1,7);
        dice2 = random(1,7);
    }
    updateDisplay(dice1, dice2);
}

void updateDisplay(int value1, int value2)
{
    digitalWrite(displayPins[0], HIGH);
    digitalWrite(displayPins[1], LOW);
    setSegments(value1);
    delay(5);
    digitalWrite(displayPins[0], LOW);
}
```

(continúa)

LISTADO DE PROYECTO 15 (continúa)

```

    digitalWrite(displayPins[1], HIGH);
    setSegments(value2);
    delay(5);
}

void setSegments(int n)
{
    for (int i=0; i < 8; i++)
    {
        digitalWrite(segmentPins[i], ! digits[n][i]);
    }
}

```

Para manejar ambos **displays** tenemos que encender los **displays** uno tras otro, configurando sus segmentos de forma correcta. Por lo tanto, nuestra función **loop** debe mantener en distintas variables, **dice1** y **dice2**, los valores que vayan a mostrarse en cada uno de los **displays**.

Para “tirar” el dado, utilizaremos la función **random**, lo que hará que cada vez que se pulse el botón se establezca un nuevo valor en **dice1** y en **dice2**. Esto significa que al “tiro del dado” también le afectará el tiempo que se tenga pulsado el botón, por lo que no tendremos que preocuparnos de activar el generador de números aleatorios.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 15 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Proyecto 16

Matriz de LEDs

Si no estamos equivocados, creemos que las matrices de LEDs son uno de esos componentes que pueden gustar a las mentes más inquietas. Consisten en una matriz de LEDs, que en este caso es de 8 por 8. Estos dispositivos suelen tener un LED en cada una de las posiciones; sin embargo, en el dispositivo que vamos a utilizar, cada uno de estos LED son dos en realidad, uno rojo y uno verde, colocados bajo

una sola lente, por lo que aparecen como un único punto. De esta forma podemos encender uno o los dos LED para formar un color rojo, verde o naranja.

En la Figura 6-10 se muestra el proyecto completo.

Este proyecto utiliza una matriz de LEDs como la que acabamos de describir, y nos permitirá mostrar dibujos multicolores a través de la conexión USB. En cuanto al proyecto, en este caso emplearemos bastantes componentes y utilizaremos prácticamente todos los pines de la placa Arduino.

COMPONENTES Y EQUIPO

Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
Matriz LED 8 x 8 (2 colores)	34
R1-16 Resistencia 100 Ω 0.5 W	5
IC1 Contador de décadas 4017	46
T1-8 2N7000	42
Placa protoboard extra larga 72 x 2	

Hardware

La Figura 6-11 muestra el esquema electrónico del proyecto. Como era de esperar, los LEDs están organizados en filas y columnas, con todas las conexiones negativas de una determinada columna conectadas entre sí y una conexión positiva sepa-

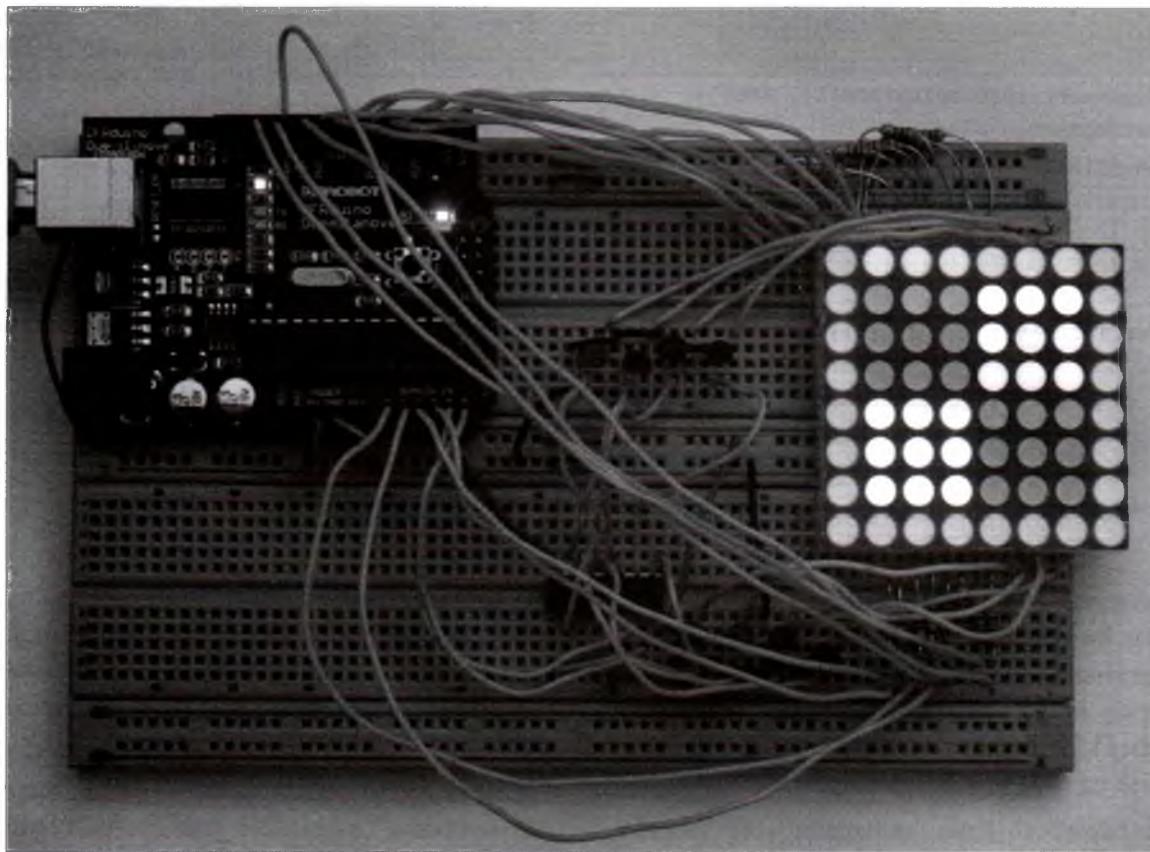


Figura 6-10 Proyecto 16. Matriz de LEDs.

rada para cada LED de la fila.

Para manejar la matriz tenemos que hacer lo mismo que hicimos con el **display** de dos dígitos del Proyecto 15, cambiando entre columnas y encendiendo y apagando sucesivamente la fila correspondiente de LEDs para crear la ilusión de que todos los LEDs están encendidos al mismo tiempo. En realidad, en cada momento habrá encendidos un máximo de 16 LEDs (8 rojos + 8 verdes).

En la matriz de LEDs hay 24 cables y en la placa Arduino sólo 17 pines que podemos utilizar con facilidad (D2-13 y A0-5). Así es que para controlar cada una de las columnas, una tras otra, vamos a utilizar un circuito integrado llamado un **contador de décadas**.

El **contador de décadas 4017** tiene diez pines de salida, que sucesivamente se van activando a nivel alto cada vez que hay un impulso en el pin de

reloj (**clock**). También tiene un pin reset para poner el contador a 0. Así pues, en lugar de necesitar una salida de la placa Arduino para cada fila, sólo necesitamos dos salidas: una para **clock** y una para **reset**.

Cada una de las salidas del 4017 va conectada a un transistor de efecto campo (**FET**). La única razón por la que hemos utilizado un **FET** en lugar de un transistor bipolar es que podemos conectar la puerta del **FET** directamente a una salida de la placa Arduino sin tener que utilizar una resistencia limitadora de corriente.

Observe que no utilizamos la primera salida del 4017. Esto se debe a que este pin se activa tan pronto como se reinicia el 4017 y esto nos llevaría a que esa columna estaría activada más tiempo del debido, haciendo que apareciera más brillante que las demás.

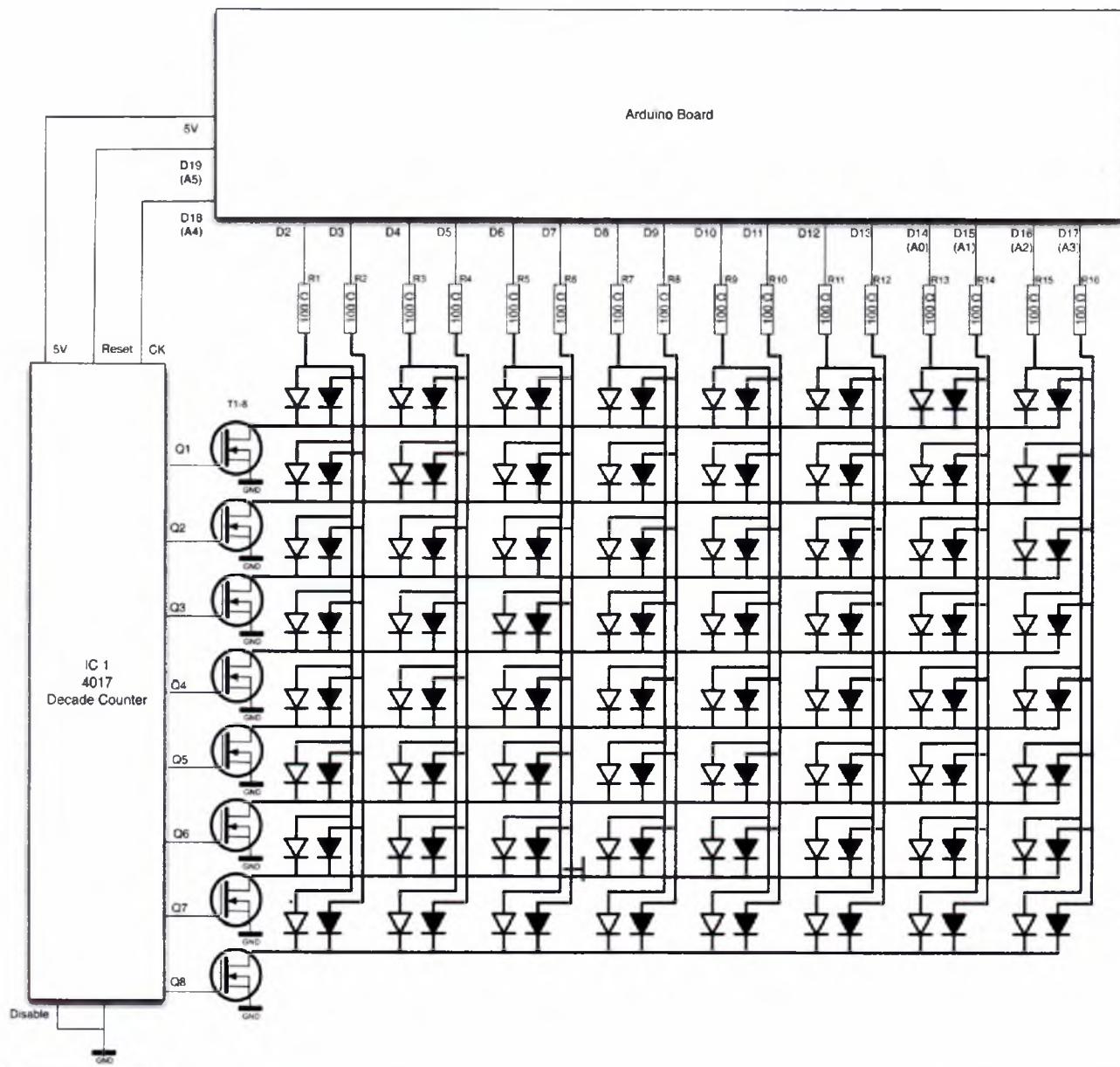


Figura 6-11 Circuito electrónico del Proyecto 16.

Para montar este proyecto en la placa de pruebas vamos a necesitar una placa mayor que las que hemos utilizado hasta ahora.

En la Figura 6-12 se muestra el diseño de este proyecto. Cuando conecte los cables preste atención y compruebe cuidadosamente cada una de las conexiones porque cualquier cambio accidental de las mismas puede producir resultados muy extraños y difíciles de depurar.

Software

El software de este proyecto es bastante corto (Listado del Proyecto 16). No obstante, el reto es conseguir que la temporización sea la adecuada, ya que si las cosas se hacen demasiado rápido, el 4017 no habrá tenido tiempo de activar la fila asignada cuando ya habrá llegado la señal para la siguiente columna. Esto provoca una imagen de colores borrosa. Por otro lado, si hace las cosas con demasiada lentitud, el **display** parpadea. Esta es precisa-

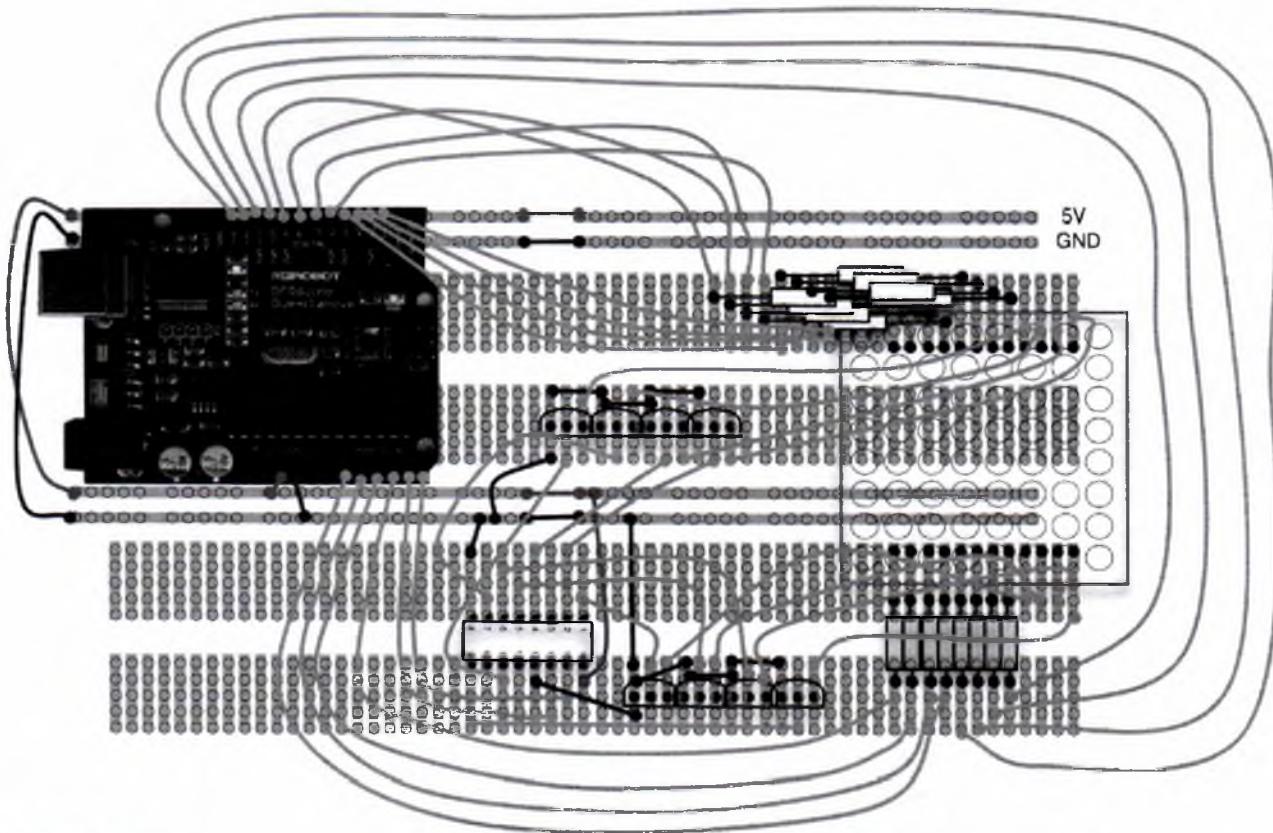


Figura 6-12 | Diseño del circuito del Proyecto 16 sobre la placa de pruebas.

LISTADO DE PROYECTO 16

```
int clockPin = 18;
int resetPin = 19;

int greenPins[8] = {2, 3, 4, 5, 6, 7, 8, 9};
int redPins[8] = {10, 11, 12, 13, 14, 15, 16, 17};

int row = 0;
int col = 0;

// colores: apagado = 0, verde = 1, rojo = 2, naranja = 3
byte pixels[8][8] = {
{1, 1, 1, 1, 1, 1, 1, 1},
{1, 2, 2, 2, 2, 2, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 3, 3, 3, 3, 2, 1},
{1, 2, 2, 2, 2, 2, 2, 1},
{1, 1, 1, 1, 1, 1, 1, 1}
};
```

(continúa)

LISTADO DE PROYECTO 16 (continúa)

```
void setup()
{
    pinMode(clockPin, OUTPUT);
    pinMode(resetPin, OUTPUT);
    for (int i = 0; i < 8; i++)
    {
        pinMode(greenPins[i], OUTPUT);
        pinMode(redPins[i], OUTPUT);
    }
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available())
    {
        char ch = Serial.read();
        if (ch == 'x')
        {
            clear();
        }
        if (ch >= 'a' and ch <= 'g')
        {
            col = 0;
            row = ch - 'a';
        }
        else if (ch >= '0' and ch <= '3')
        {
            byte pixel = ch - '0';
            pixels[row][col] = pixel;
            col++;
        }
    }
    refresh();
}

void refresh()
{
    pulse(resetPin);
    delayMicroseconds(2000);
    for (int row = 0; row < 8; row++)
    {
        for (int col = 0; col < 8; col++)
        {
            int redPixel = pixels[col][row] & 2;
            int greenPixel = pixels[col][row] & 1;
            digitalWrite(greenPins[col], greenPixel);
            digitalWrite(redPins[col], redPixel);
        }
    }
    pulse(clockPin);
```

(continúa)

LISTADO DE PROYECTO (continúa)

```

    delayMicroseconds(1500);
}

void clear()
{
    for (int row = 0; row < 8; row++)
    {
        for (int col = 0; col < 8; col++)
        {
            pixels[row][col] = 0;
        }
    }
}

void pulse(int pin)
{
    delayMicroseconds(20);
    digitalWrite(pin, HIGH);
    delayMicroseconds(50);
    digitalWrite(pin, LOW);
    delayMicroseconds(50);
}

```

mente la razón de las llamadas a **delayMicroseconds**. Esta función es como la función **delay**, pero permite retardos más cortos.

Aparte de eso, el código es bastante sencillo.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 16 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Ahora puede probar el proyecto. Tan pronto como se conecte al puerto USB y pulse **reset**, debería ver un patrón de prueba de un anillo exterior verde, con un anillo de color rojo dentro y, a continuación, un bloque naranja en el centro.

Abra **Serial Monitor** en el software de Arduino e introduzca **x**. Esto debería borrar la pantalla. Ahora puede cambiar cada línea de la pantalla introduciendo una letra para la fila (a-h), seguido inme-

diatamente de ocho dígitos. Cada dígito será **0** para apagado, **1** para el verde, **2** para el rojo y **3** para naranja. Por tanto, si introduce **a12121212** hará que en la fila superior se vayan alternando los colores rojo y verde. Cuando diseñe pautas para visualizar, lo mejor es escribir las líneas en un editor de texto o en un procesador de textos y, posteriormente, pegar toda la serie en **Serial Monitor**.

Si quiere, puede introducir el texto siguiente:

```

x
a11222211
b11222211
c11222211
d11111111
e11111111
f11222211
g11222211
h11111111

```

6

```
x 471 100 500 1000 10000  
a22222222  
b12333321  
c11211211  
d11122111  
e11233211  
f12333321  
g22222222  
h11111111
```

8

```
x  
a11111111  
b22212221  
c11212121  
d22212121  
e11212121  
f22212221  
g11111111  
h11111111
```

Este es realmente un buen proyecto para que experimenten las mentes más inquietas. Puede que también desee intentar producir un efecto de animación, cambiando la matriz **pixels** mientras continúa en el **loop**.

Displays LCD

Si nuestro proyecto necesita mostrar algo más que unos cuantos dígitos numéricos, es probable que quiera utilizar un módulo **display LCD**. Estos tienen la ventaja de que ya traen incorporado el controlador electrónico, por lo que gran parte del trabajo ya nos lo dan hecho y no tenemos que hacer un muestreo de cada dígito, configurando cada segmento.

Estos dispositivos ya empiezan a ser bastante estándar, por lo que ya hay muchos modelos de diferentes fabricantes que podemos utilizar de la misma manera. Los dispositivos que debemos buscar son lo que usan el chip controlador **HD44780**.

Los **displays LCD** pueden resultar bastante caros si se adquieren en los comercios de componentes electrónicos, pero buscando en Internet, a menudo se encuentran por unos euros, especialmente si está dispuesto a comprar unos cuantos de una vez.

La Figura 6-13 muestra un módulo que puede mostrar dos filas de 16 caracteres, estando cada carácter formado por una matriz de 7 por 5 segmentos. En este caso tampoco tenemos que manejar cada segmento por separado.



Figura 6-13 Módulo LCD de 2 por 16 caracteres.

El módulo de visualización incluye un conjunto de caracteres para que sepa qué segmentos activar para cada carácter. Esto significa que simplemente tenemos que decirle qué carácter mostrar y en qué posición de la pantalla.

Sólo necesitamos siete salidas digitales para manejar la pantalla. Cuatro de ellas son conexiones de datos y tres controlan el flujo de datos. Los detalles exactos de lo que se envía al módulo LCD se pueden pasar por alto, ya que existe una biblioteca estándar que podemos utilizar.

Lo mostramos en el siguiente proyecto.

Proyecto 17

Placa de mensajes USB

Este proyecto nos permitirá mostrar un mensaje de nuestro ordenador en un módulo LCD. No es necesario que el módulo LCD tenga que estar justo al lado del ordenador, de forma que puede utilizarlo colocándolo en el extremo de un cable USB largo para mostrar mensajes remotos - por ejemplo, al lado del portero electrónico de la puerta de su casa.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o similar	1
Módulo LCD (controlador HD44780)	58
R1 Resistencia 100 Ω 0,5 W	5
Tira de 16 pines de 2,54 mm de paso	56

Hardware

El esquema electrónico de la pantalla **LCD** se muestra en la Figura 6-14 y el diseño de la placa de pruebas, en la Figura 6-15. Como puede ver, los únicos componentes que se necesitan son el propio módulo **LCD** y una resistencia para limitar la corriente de la retroiluminación por LED.

El módulo **LCD** recibe cuatro bits de datos al mismo tiempo a través de las conexiones **D4-D7**. El módulo también tiene conectores para **D0-D3**, que sólo se utilizan para transferir datos, ocho bits al mismo tiempo. Pero, para reducir el número de pines necesarios, estos no los vamos a utilizar.

La manera más fácil de conectar el módulo **LCD** a la placa de pruebas es soldarle una tira de pines y conectar el módulo directamente a la placa de pruebas.

Software

El software de este proyecto es muy sencillo (Listado de Proyecto 17). Todo el trabajo de comunicación con el módulo **LCD** es realizado por la **biblioteca LCD**. Esta biblioteca se incluye como parte de la instalación de software estándar de Arduino, por lo que no es necesario descargar o instalar nada especial.

El bucle (**loop**) lee cualquier entrada y, cuando encuentra un carácter **#**, borra la pantalla. Si el carácter que encuentra es **/**, se desplaza a la segunda fila; en el resto de los casos, muestra el carácter que se haya enviado.

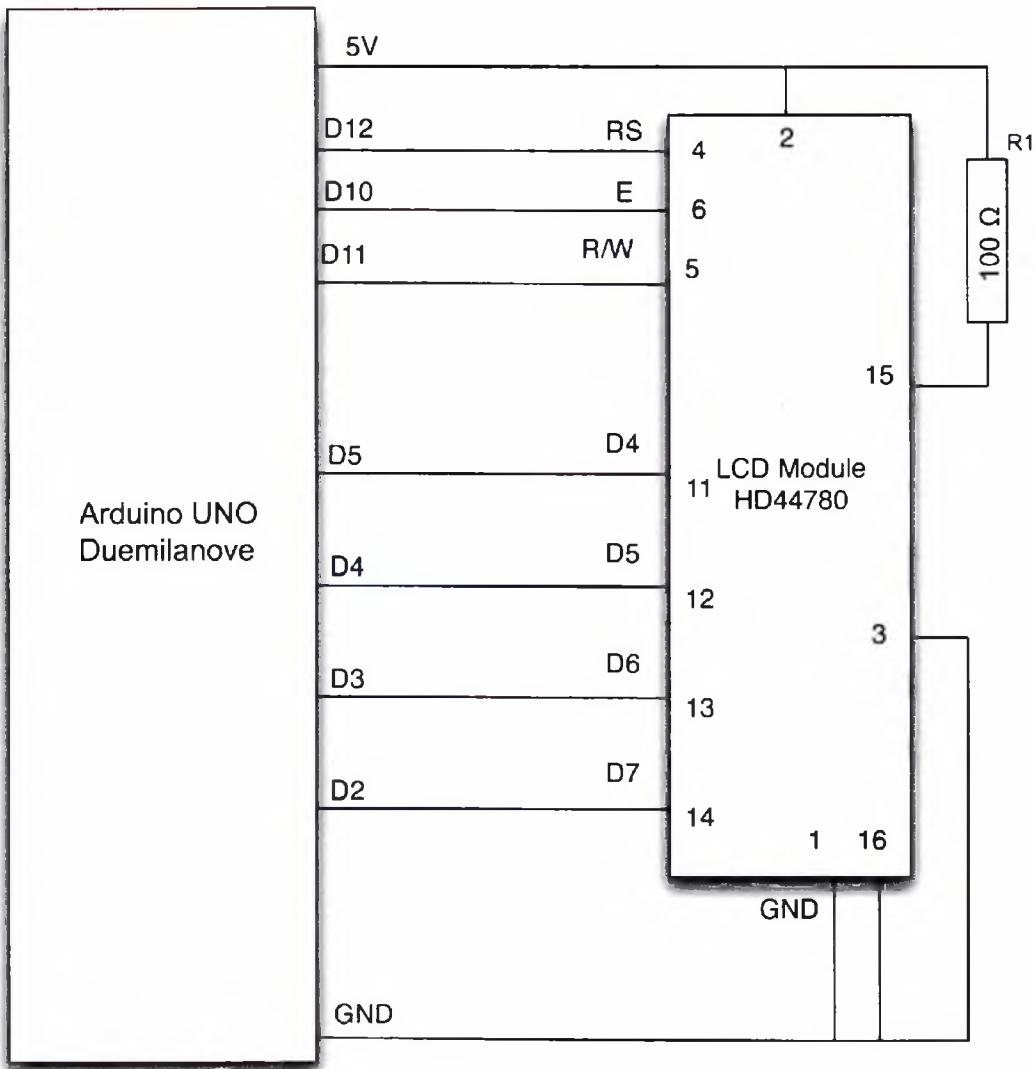


Figura 6-14 Circuito electrónico del Proyecto 17.

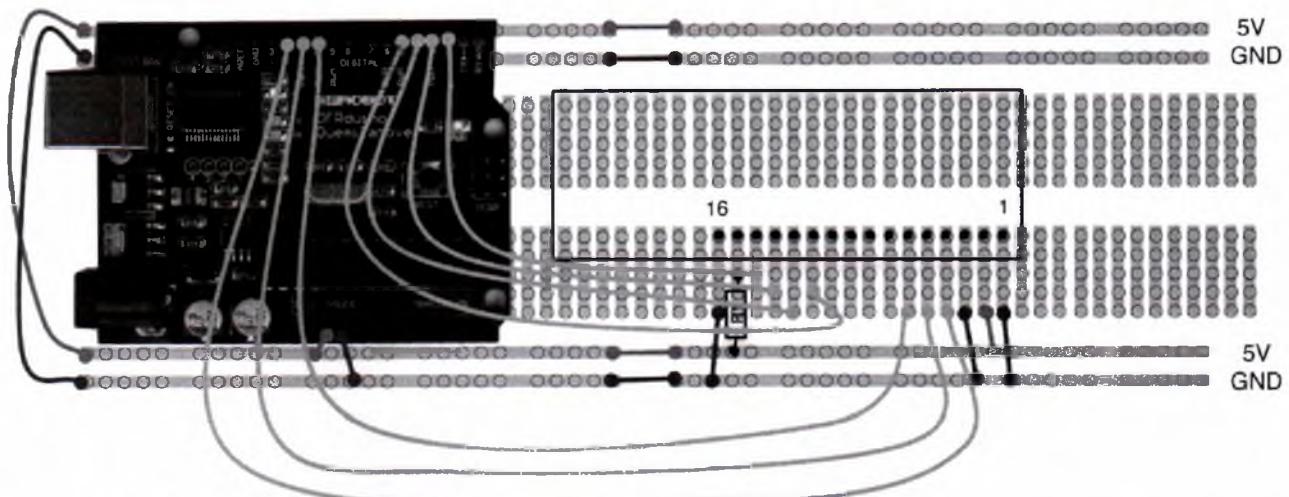


Figura 6-15 Diseño del circuito del Proyecto 17 sobre la placa de pruebas.

LISTADO DE PROYECTO 17

```
#include <LiquidCrystal.h>

// Pantalla LiquidCrystal con:
// rs on pin 12
// rw on pin 11
// activar en pin 10
// d4-7 on pins 5-2
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

void setup()
{
    Serial.begin(9600);
    lcd.begin(2, 20);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Evil Genius");
    lcd.setCursor(0,1);
    lcd.print("Rules");
}

void loop()
{
    if (Serial.available())
    {
        char ch = Serial.read();
        if (ch == '#')
        {
            lcd.clear();
        }
        else if (ch == '/')
        {
            lcd.setCursor(0,1);
        }
        else
        {
            lcd.write(ch);
        }
    }
}
```

Pongamos todo junto

Cargue el sketch terminado del Proyecto 17 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Ahora podemos probar el proyecto abriendo el **Serial Monitor** e introduciendo algún texto.

Más adelante, en el Proyecto 22, volveremos a utilizar el panel **LCD** con un termistor y un codificador giratorio para crear un termostato.

Resumen

Eso es todo para los proyectos relacionados con LEDs y luz. En el siguiente capítulo vamos a comenzar a trabajar con proyectos que utilizan sonido de una u otra forma.

CAPÍTULO 7

Proyectos de sonido

UNA PLACA ARDUINO se puede usar para generar sonidos como salida y recibir sonidos como entrada utilizando un micrófono. En este capítulo, tenemos varios proyectos de tipo "instrumento musical" y también proyectos que procesan entradas de sonido. Aunque no es estrictamente un proyecto de "sonido", nuestro primer proyecto consiste en crear un osciloscopio simple para que podamos ver la forma de onda de una entrada analógica.

Proyecto 18 Osciloscopio

Un osciloscopio es un dispositivo que permite ver una señal electrónica que aparece en forma de onda. Un osciloscopio tradicional funciona amplificando una señal para controlar la posición de un punto sobre el eje vertical (eje Y) de un tubo de rayos

catódicos, mientras un mecanismo de base de tiempos barre el eje horizontal de izquierda a derecha, repitiendo el ciclo al llegar al final. El resultado será algo parecido a la Figura 7-1.

En la actualidad, los tubos de rayos catódicos han sido sustituidos por osciloscopios digitales que utilizan pantallas LCD, pero los principios siguen siendo los mismos.

Este proyecto lee los valores de la entrada analógica y los envía al ordenador a través de una conexión USB. En lugar de ser recibidos por el **Serial Monitor**, son recibidos por un pequeño programa que los muestra como lo haría un osciloscopio. Según cambia la señal, también lo hace la forma de la onda.

En lo que se refiere al osciloscopio, obviamente éste no va a ganar ningún premio de precisión ni de velocidad, pero puede resultar divertido.

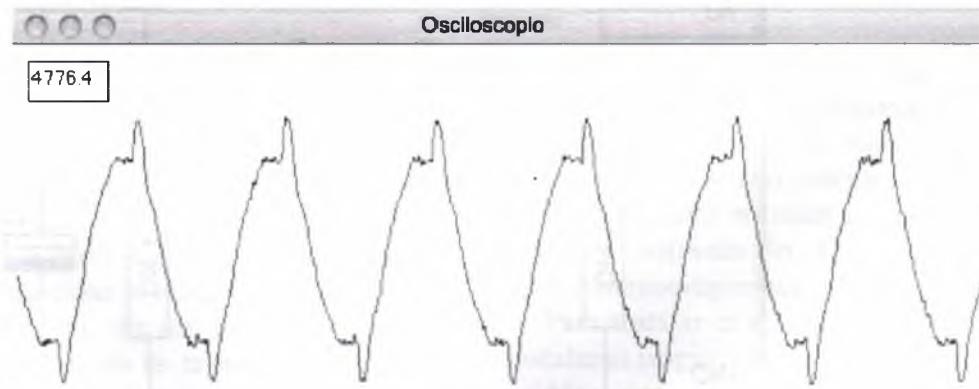


Figura 7-1 Ruido de una señal de 50-Hz en un osciloscopio.

COMPONENTES Y EQUIPO		
Descripción		Apéndice
Placa Arduino UNO o Duemilanove o clon		
C1 200nF no polarizado	1	21
C2, C3 Electrolítico de 100 μ F		22
R1, R2 Resistencia 1 M Ω 0,5 W		15
R3, R4 Resistencia 1 M Ω 0,5 W		7

Esta es la primera vez que vamos a utilizar **condensadores**. **C1** se puede conectar en cualquier sentido; sin embargo, **C2** y **C3** están polarizados, es decir, tienen polo positivo (+) y negativo (-) y, por tanto, deben conectarse de la forma correcta o es probable que resulten dañados. Como sucede con los LEDs, en los condensadores polarizados el polo positivo (marcado como un rectángulo blanco en el símbolo del esquema) es más largo que el polo negativo. El polo negativo también tiene a menudo un - (menos) o una forma de diamante junto al polo negativo.

Hardware

La Figura 7-2 muestra el esquema eléctrico del Proyecto 18 y la Figura 7-3 el Diseño del circuito sobre la placa de pruebas.

El circuito tiene dos partes. **R1** y **R2** son resistencias de alto valor que polarizan la señal que va a la entrada analógica a 2.5 V, haciendo de divisor de tensión. El condensador **C1** permite a la señal de AC (corriente alterna) pasar sin ninguna componente de corriente continua (DC) (modo AC tradicional en un osciloscopio).

R3, **R4**, **C2** y **C3** proporcionan un voltaje estable de referencia de 2.5 V. Esto es así para que nuestro osciloscopio pueda mostrar tanto señales positivas como negativas. Así, una de nuestras puntas de prueba queda fijada en 2.5 V, y cualquier señal que tengamos en la otra punta la tomará de referencia. Es decir, un voltaje positivo significará un valor en la entrada analógica de más de 2.5 V y un valor negativo de menos de 2.5 V.

La figura 7-4 muestra el osciloscopio completo.

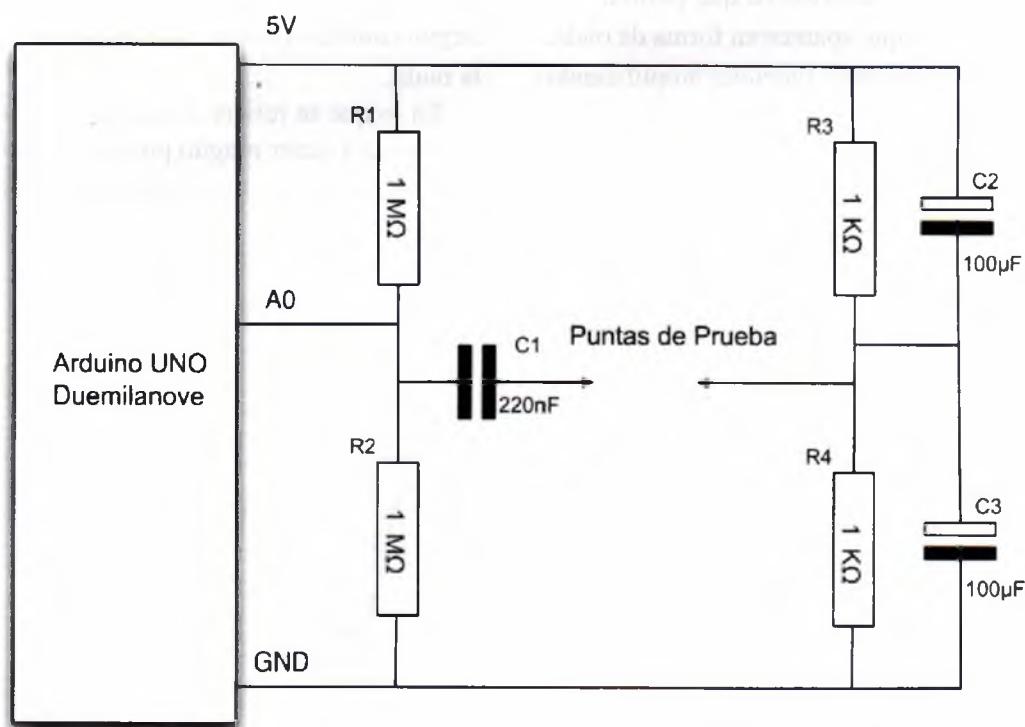


Figura 7-2 Esquema eléctrico del Proyecto 18.

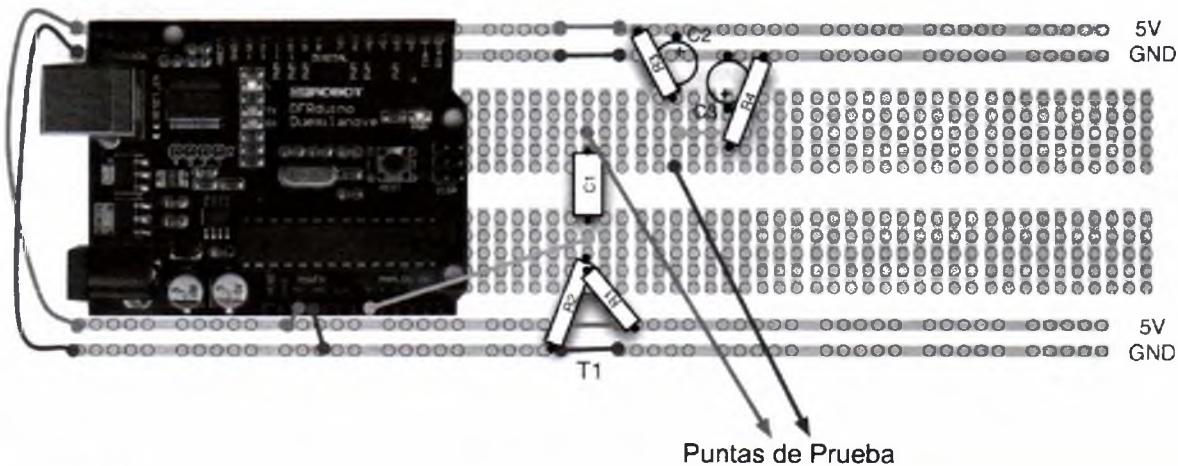


Figura 7-3 Diseño del circuito del Proyecto 18 sobre la placa de pruebas.

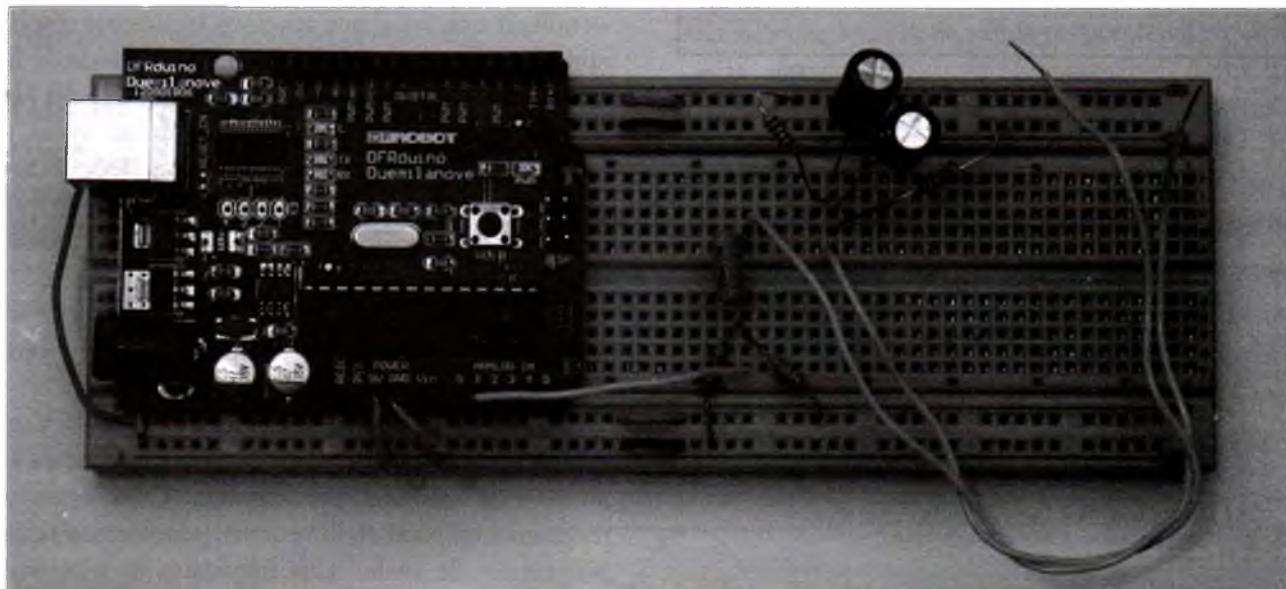


Figura 7-4 Proyecto 18. Osciloscopio.

Software

El **sketch** es corto y sencillo (Listado del Proyecto 18). Su único propósito es el de leer la entrada analógica y mandarla al puerto USB tan rápido como sea posible.

Lo primero a tener en cuenta es que hemos aumentado la velocidad a **115.200 baudios**, la más alta disponible. Para obtener la mayor cantidad posible de datos a través de la conexión, sin tener que recurrir a complejas técnicas de compresión, vamos a desplazar dos bits a la derecha ($>> 2$) nuestros valores en bruto de diez bits; el efecto de esto

es que lo divide por cuatro y, por tanto, hace que quepa en un solo byte.

Naturalmente, necesitaremos ejecutar algún tipo de software en nuestro ordenador para que podamos ver los datos enviados por la placa (Figura 7-1). Este software lo podemos descargar de www.arduinoevilgenius.com.

Para instalar el software, primero tendrá que instalar el lenguaje **Ruby** en su ordenador. Si usa un Mac, está de suerte, porque vienen con Ruby pre instalado. Si está utilizando Windows o Linux, por favor, siga las instrucciones de <http://www.ruby-lang.org/en/downloads>.

LISTADO DE PROYECTO 18

```
#define CHANNEL_A_PIN 0

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  int value =
    analogRead(CHANNEL_A_PIN);
  value = (value >> 2) & 0xFF;
  Serial.print(value, BYTE);
  delayMicroseconds(100);
}
```

Una vez que haya instalado **Ruby**, el siguiente paso es instalar un módulo opcional de **Ruby** para comunicarse con el puerto USB. Para instalar esto, abra el símbolo del sistema en Windows o un terminal para Mac y Linux y, si está utilizando Windows, introduzca:

```
gem install ruby-serialport
```

Si está usando Mac o Linux, escriba:

```
sudo gem install ruby-serialport
```

Si todo ha funcionado bien, debería ver un mensaje como éste:

```
Building native extensions. This could
take a while...
Successfully installed ruby-
serialport-0.7.0
1 gem installed
Installing ri documentation for
ruby-serialport-0.7.0...
Installing RDoc documentation for
ruby-serialport-0.7.0...
```

Para ejecutar el software, cambie el directorio en su terminal o intérprete de comandos al directorio donde descargó **scope.rb**. A continuación, escriba:

```
ruby scope.rb
```

Debe aparecer una ventana como la de la Figura 7-1.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 18 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1). Instale el software para su ordenador como se ha descrito anteriormente y ya estará listo para comenzar.

La manera más sencilla de probar el osciloscopio es utilizar una señal que tenemos fácilmente disponible a nuestro alrededor y que es el zumbido de la red eléctrica. La red eléctrica oscila a **50** ó **60** Hz (dependiendo de la parte del mundo donde viva), y cada aparato eléctrico emite radiación electromagnética a esta frecuencia. Para recogerla, todo lo que tiene que hacer es tocar el cable de prueba conectado a la entrada analógica y debería ver una señal similar a la de la Figura 7-1. Pruebe a agitar su brazo cerca de cualquier equipo eléctrico y verá cómo cambia esta señal.

En esta figura se muestra la **forma de onda** y un cuadro pequeño con un número dentro, que indica el número de muestras por segundo. Cada muestra representa un píxel en la ventana, y la ventana tiene 600 píxeles de ancho. Una frecuencia de muestreo de 4.700 muestras por segundo significa que cada muestra tiene una duración de 1/4.700 segundos, por lo que el ancho total de 600 muestras representa 600/4.700 ó, lo que es lo mismo, 128 milisegundos. La longitud de onda en la Figura 7-1 es de aproximadamente 1/6 de 128, o 21 milisegundos, lo que es igual a una frecuencia de 1/0.021 ó 47.6 Hz. Este resultado es lo suficientemente aproximado como para confirmar que lo que estamos viendo es un zumbido de frecuencia de red de 50 Hz.

La **amplitud de la señal**, como se muestra en la Figura 7-1, tiene una resolución de un píxel por paso de muestra, y hay 256 pasos. Por tanto, si conecta entre sí los dos cables de prueba, debería ver una línea horizontal en mitad de la ventana. Esto corresponde a 0 V y está a 128 píxeles desde la parte superior de la pantalla, ya que la ventana tiene 256 píxeles de alto. Esto significa que, puesto que la señal es aproximadamente dos tercios de la ventana, la amplitud de la señal es de alrededor de 3 V pico a pico.

Notará que la **tasa de muestreo** cambia bastante, algo que demuestra que este osciloscopio es de los más rudimentarios y que, por tanto, no debe utilizarse para nada crítico.

Si desea modificar la base de tiempos, cambie el valor de la función `delayMicroseconds`.

Generación de sonido

Desde una placa Arduino se puede generar sonido con sólo activar y desactivar uno de sus pines a la frecuencia correcta. Si lo hace, el sonido producido será irregular y chirriante. Esto se denomina una **onda cuadrada**. Para producir un tono más agrada-

ble, necesitará una señal que se parezca más a una **onda senoidal** (véase la Figura 7-5).

Generar una **onda senoidal** requiere un poco de razonamiento y esfuerzo. Una primera idea puede ser utilizar la salida analógica de uno de los pines para obtener la forma de onda. Sin embargo, el problema es que las salidas analógicas de una Arduino no son verdaderas salidas analógicas, sino que son salidas **PWM** que se activan y desactivan muy rápidamente. De hecho, su frecuencia de conmutación se encuentra dentro del espectro de audio, por lo que si no tenemos cuidado, nuestra señal sonará tan mal como una onda cuadrada.

La mejor manera de hacerlo es utilizar un convertidor digital-analógico, o **CDA** como se les conoce. Un **CDA** tiene diversas entradas digitales y produce una tensión de salida proporcional al valor de la entrada digital. Afortunadamente, hacer un **CDA simple** es fácil: todo lo que necesita son resistencias.

La Figura 7-6 muestra un **CDA** construido a partir de lo que se denomina una **escalera de resistencias R-2R**.

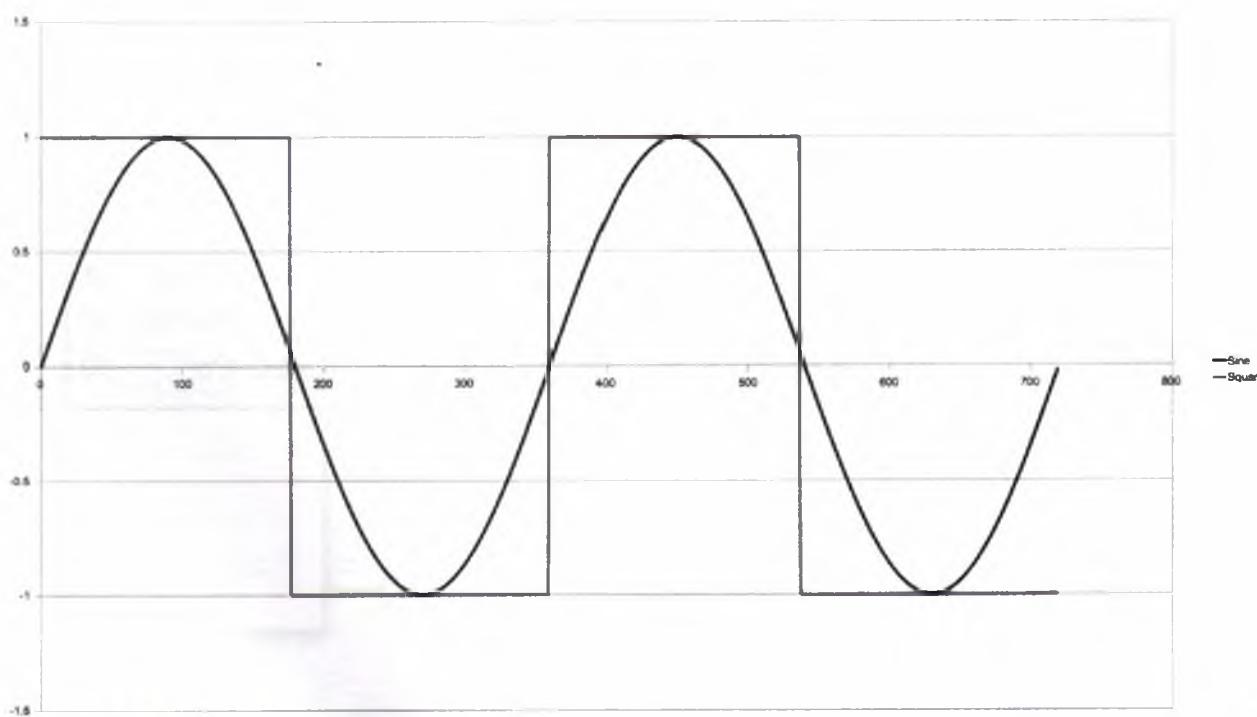


Figura 7-5 Ondas cuadradas y senoidales.

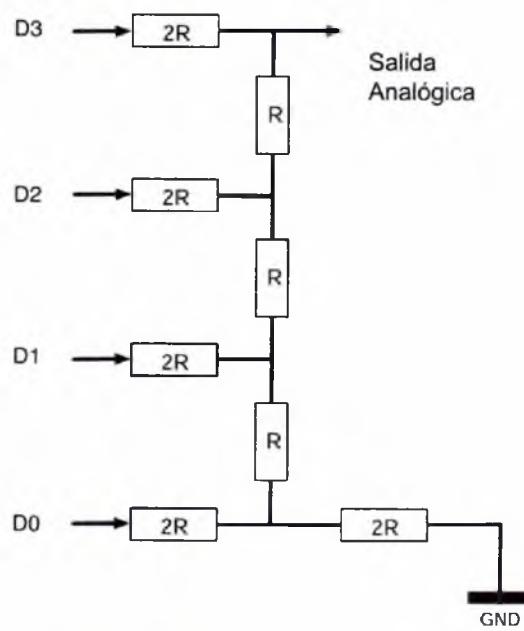


Figura 7-6 CDA utilizando una escalera R-2R.

Utiliza resistencias de un valor R y dos veces R , por lo que R podría ser $5\text{ K}\Omega$ y $2R$ $10\text{ K}\Omega$. Cada una

TABLA 7-1 SALIDA ANALÓGICA DE ENTRADAS DIGITALES

D3	D2	D1	D0	Salida
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

de las entradas digitales se conectarán a una salida digital de Arduino. Los cuatro dígitos representan los cuatro bits del número digital. Por tanto, esto nos da 16 salidas analógicas diferentes, como se muestra en la Tabla 7-1.

Proyecto 19 Reproductor de música

Utilizando un CDA, este proyecto reproduce una serie de notas musicales a través de un altavoz en miniatura para intentar crear algo parecido a una onda senoidal.

Si puede conseguir un pequeño altavoz con cables podría conectarlo directamente en la placa de pruebas. Si no, tendrá que, o bien soldarle cables rígidos a sus terminales o, si no tiene acceso a un soldador, empalmar los cables enrollándolos cuidadosamente en los terminales.

COMPONENTES Y EQUIPO

Descripción	Apéndice
Placa Arduino UNO o Duemilanove o clon	1
C1 100nF no polarizado	20
C2 Electrolítico 16V, 100 μF	22
R1-5 Resistencia 10 $\text{M}\Omega$ 0.5W,	9
R6-8 Resistencia 4.7 $\text{M}\Omega$ 0.5W,	8
R9 Resistencia 1 $\text{M}\Omega$ 0.5 W	15
R10 Potenciómetro lineal 100 $\text{M}\Omega$	13
IC1 Amplificador audio 1W TDA7052	47
Altavoz pequeño 8 Ω 1 W	59

Hardware

Para tratar de reducir el número de componentes al mínimo, hemos utilizado un circuito integrado para amplificar la señal y reproducirla por el altavoz. El **IC TDA7052** proporciona 1 W de potencia de salida en un pequeño chip de 8 pines fácil de usar.

La Figura 7-7 muestra el esquema electrónico del Proyecto 19, y el diseño del circuito sobre la placa de pruebas se muestra en la Figura 7-8.

Para conectar la salida del **CDA** a la entrada del amplificador vamos a utilizar **C1**; y como condensador de desacoplo para desviar a masa cualquier tipo de ruido de las líneas eléctricas, utilizaremos **C2**. Este debe ubicarse lo más cerca posible de **IC1**.

R9 y el potenciómetro **R10** forman un divisor de tensión para reducir la señal de la **escalera de resistencias** en, al menos, un factor de 10, dependiendo del ajuste del potenciómetro.

Software

Para generar una **onda senoidal**, el **sketch** recorre una serie de valores almacenados en la matriz **sin16**. Estos valores se muestran representados en la gráfica de la Figura 7-9. No es la **onda senoidal** más exacta del mundo, pero es una clara mejora comparada con la de una **onda cuadrada** (consulte el Listado del Proyecto 19).

La función **playNote** es la clave para generar la nota. El tono de la nota producida es controlado por el retardo tras cada paso de la señal. El bucle (loop) para generar la forma de onda se encuentra a su vez dentro de otro bucle que produce el suficiente número de ciclos como para hacer que cada nota tenga aproximadamente la misma duración.

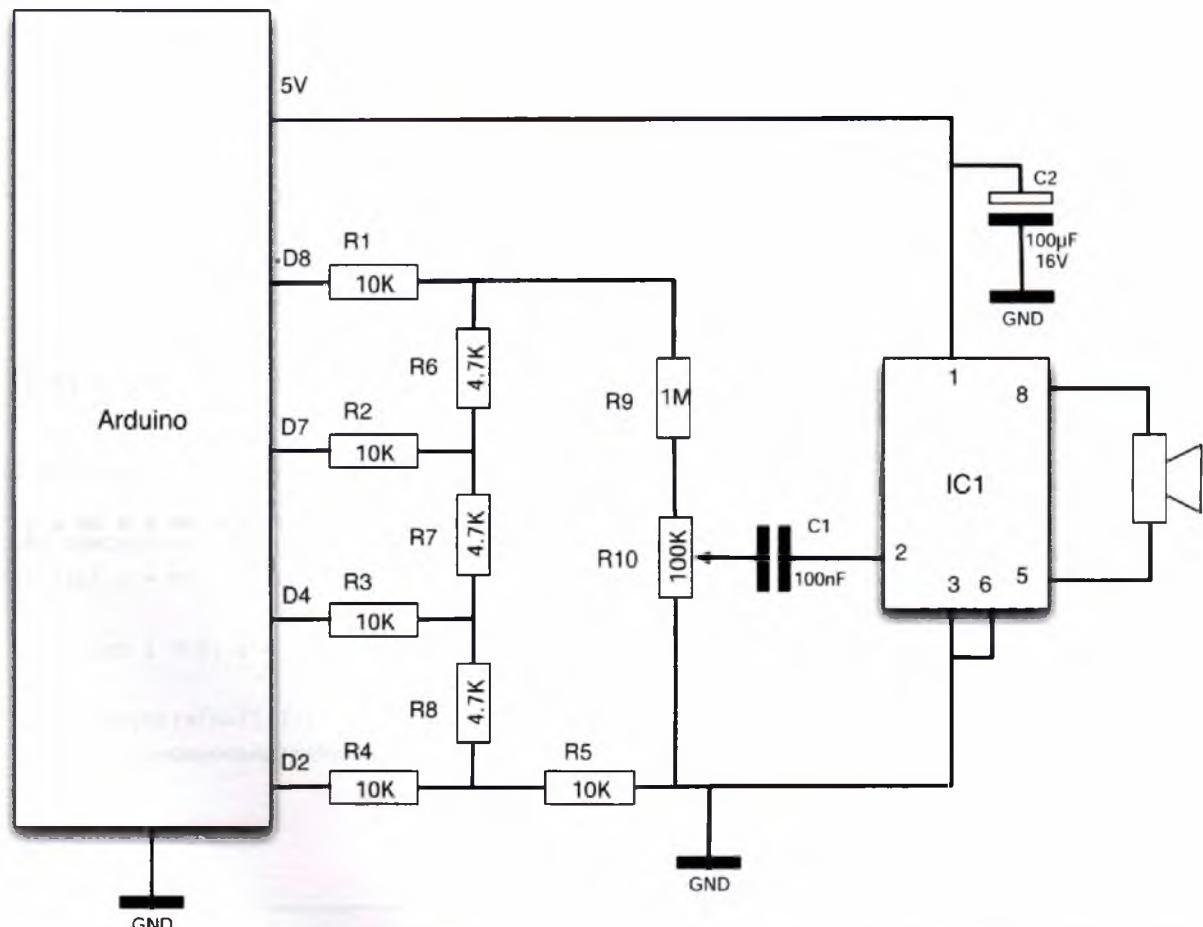


Figura 7-7 Esquema eléctrico del Proyecto 19.

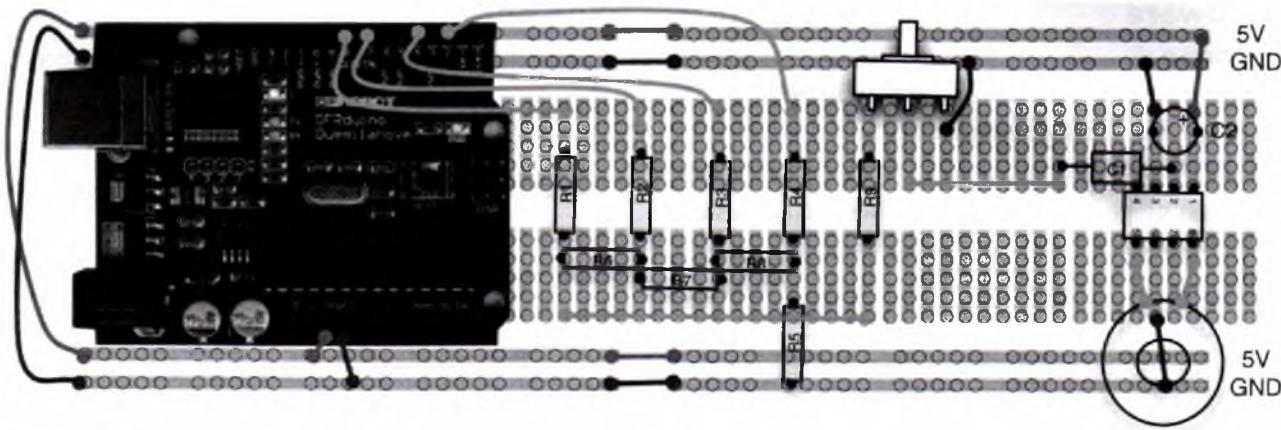


Figura 7-8 Diseño del circuito del Proyecto 19 sobre la placa de pruebas.

LISTADO DE PROYECTO 19

```

int dacPins[] = {2, 4, 7, 8};
int sin16[] = {7, 8, 10, 11, 12, 13, 14, 14, 15, 14, 14, 13, 12, 11,
               10, 8, 7, 6, 4, 3, 2, 1, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6};

int lowToneDurations[] = {120, 105, 98, 89, 78, 74, 62};
//          A   B   C   D   E   F   G
int highToneDurations[] = { 54, 45, 42, 36, 28, 26, 22 };
//          a   b   c   d   e   f   g

// Escala
//char* song = "A B C D E F G a b c d e f g";

// Jingle Bells
//char* song = "E E EE E E EE E G C D EEEE F F F F F E E E E D D E DD GG E E EE E E EE E
// G C D EEEE F F F F F E E E E G G F D CCCC";

// Jingle Bells - Más agudo
char* song = "e e ee e e ee e g c d eeee f f f f f e e e e d d e dd gg e e ee e e ee e g
c d eeee f f f f f e e e g g f d cccc";

void setup()
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(dacPins[i], OUTPUT);
  }
}

```

(continúa)

LISTADO DE PROYECTO 19 (continúa)

```
void loop()
{
    int i = 0;
    char ch = song[0];
    while (ch != 0)
    {
        if (ch == ' ')
        {
            delay(75);
        }
        else if (ch >= 'A' and ch <= 'G')
        {
            playNote(lowToneDurations[ch - 'A']);
        }
        else if (ch >= 'a' and ch <= 'g')
        {
            playNote(highToneDurations[ch - 'a']);
        }
        i++;
        ch = song[i];
    }

    delay(5000);
}

void setOutput(byte value)
{
    digitalWrite(dacPins[3], ((value & 8) > 0));
    digitalWrite(dacPins[2], ((value & 4) > 0));
    digitalWrite(dacPins[1], ((value & 2) > 0));
    digitalWrite(dacPins[0], ((value & 1) > 0));
}

void playNote(int pitchDelay)
{
    long numCycles = 5000 / pitchDelay + (pitchDelay / 4);
    for (int c = 0; c < numCycles; c++)
    {
        for (int i = 0; i < 32; i++)
        {
            setOutput(sin16[i]);
            delayMicroseconds(pitchDelay);
        }
    }
}
```

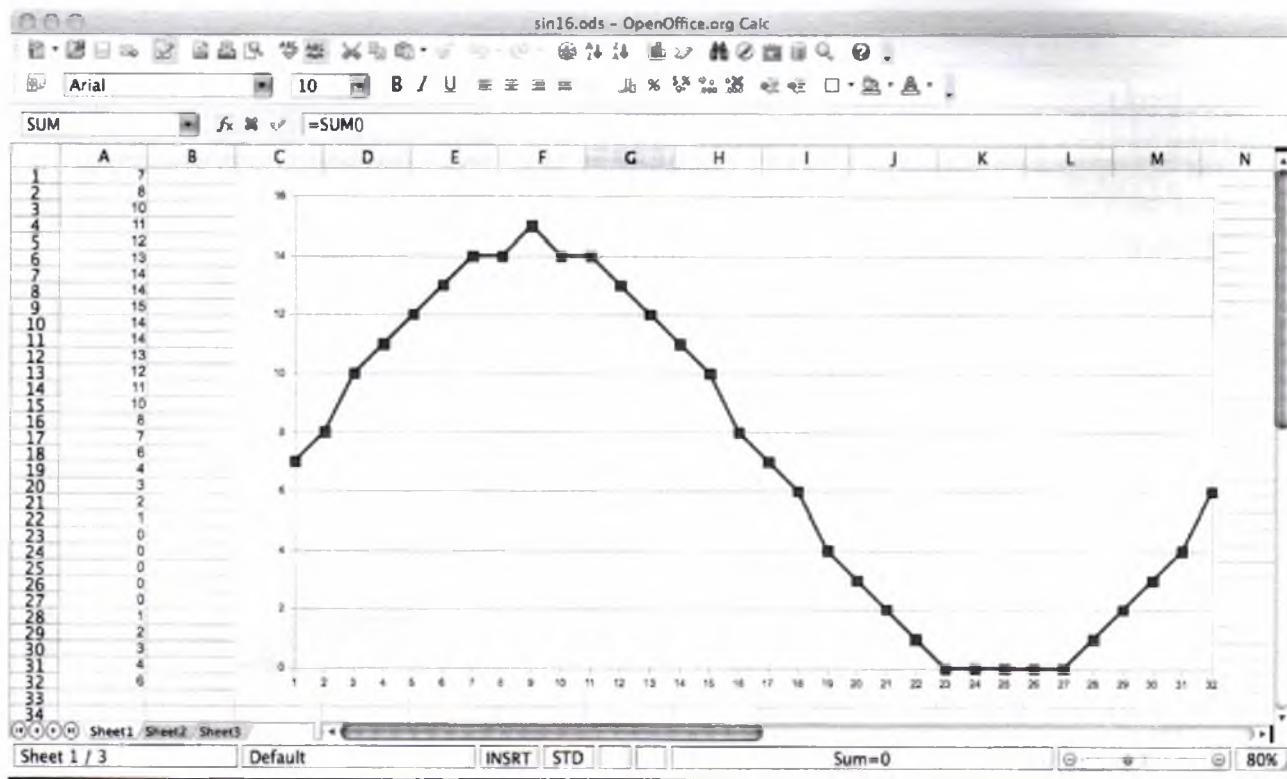


Figura 7-9 Una representación de la matriz sin16.

La función **playNote** llama a **setOutput** para establecer el valor de los cuatro pines digitales conectados a la **escalera de resistencias**. El operador **&** (**y**) se utiliza para enmascarar el valor, de forma que sólo veamos el valor del bit que nos interesa.

La música se reproduce desde una matriz de caracteres, siendo cada carácter una nota, y cada espacio representa el silencio entre notas. El bucle principal mira cada letra de la variable **song** y la reproduce. Cuando se ha reproducido la canción entera, hay una pausa de 5 segundos y luego la canción comienza de nuevo.

¡Puede que algunos encuentren este proyecto útil para vengarse de sus enemigos!

Pongamos todo junto

Cargue el sketch terminado del Proyecto 19 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Puede que desee cambiar la canción **Jingle Bells**. Para ello, simplemente coloque entre comen-

tarios la línea que empieza por "char* song =" colocando // delante de la misma y, a continuación, defina su propia matriz.

La notación funciona de la siguiente manera: hay dos octavas, las notas altas se escriben en minúscula, de la "a" a la "g" y las notas bajas de la "A" a la "G". Para una nota de mayor duración, simplemente repita la letra de la nota sin poner un espacio entre ellas.

Habrá notado que la calidad del sonido no es extraordinaria. En cualquier caso, sigue siendo mucho menos desgradable que utilizar una **onda cuadrada**, pero está lejos de lo armonioso de un instrumento musical real, donde cada nota tiene una "envolvente" donde la **amplitud** (volumen) de la nota varía de acuerdo con ésta al ser reproducida.

Proyecto 20

Arpa de luz

Este proyecto es en realidad una adaptación del Proyecto 19, pero utilizando dos sensores de luz (**LDRs**): uno que controla el tono del sonido y otro el volumen. Está inspirado en el instrumento musical Theremin, que se interpreta agitando misteriosamente las manos entre dos antenas. En realidad, este proyecto produce un sonido más parecido a una gaita que a un arpa, pero es bastante divertido.

COMPONENTES Y EQUIPO		
	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o clon	1
C1	100nF no polarizado	20
C2, C3	Electrolítico 16V, 100 μ F	22
R1-5	Resistencia 10 M Ω 0.5 W	9
R6-8	Resistencia 1 M Ω 0.5 W	8
R9, R11	Resistencia 1 M Ω 0.5 W	15
R10	Potenciómetro lineal 100 M Ω	13
R12, R13	Resistencia 47 M Ω 0.5 W	11
R14, R15	LDR	19
IC1	Amplificador audio 1W TDA7052	47
	Altavoz pequeño 8 Ω 1 W	59

Si puede conseguir un altavoz pequeño con cables podría conectarlo directamente en la placa de pruebas. Si no, tendrá que, o bien soldarle cables rígidos a sus terminales o, si no tiene acceso a un soldador, empalmar los cables enrollándoselos cuidadosamente en los terminales.

Hardware

El volumen del sonido se controla mediante una salida **PWM (D6)** conectada a la entrada de control de volumen del amplificador **TDA7052**. Queremos eliminar todos los restos de los impulsos **PWM** para que podamos pasar la salida a través del **filtro paso bajo** constituido por **R11** y **C3**, el cual sólo dejará pasar los cambios lentos de señal. Una forma de entender esto es pensar que el condensador **C3** es como un cubo que se llena con la carga de la resistencia **R11**. Las fluctuaciones rápidas de señal tendrán poco efecto debido a que se produce una integración de la señal.

Las Figuras 7-10 y 7-11 muestran el esquema electrónico y el diseño de la placa de pruebas para el proyecto y, en la Figura 7-12, puede ver el proyecto terminado.

Los **LDRs**, **R14** y **R15**, se han situado en los extremos opuestos de la placa de pruebas para hacer que resulte más fácil tocar el instrumento con las dos manos.

Software

El software de este proyecto tiene mucho en común con el Proyecto 19 (consulte el Listado del Proyecto 20).

Las principales diferencias son que el período de **pitchDelay** está determinado por el valor de la entrada analógica **0**. Esto luego se escala al rango adecuado utilizando la función **map**. Del mismo modo, el voltaje del volumen se ajusta leyendo el valor de la entrada analógica **1**, se escala usando la función **map** y, a continuación, se escribe en la salida **6** de **PWM**. Sería posible usar simplemente la **LDR R14** para controlar directamente la salida de la escalera de resistencias, pero de esta manera tenemos un mayor control sobre el escalado y compensado de la salida, y además queríamos ilustrar cómo suavizar o tratar una señal **PWM** para utilizarla para generar una salida constante.

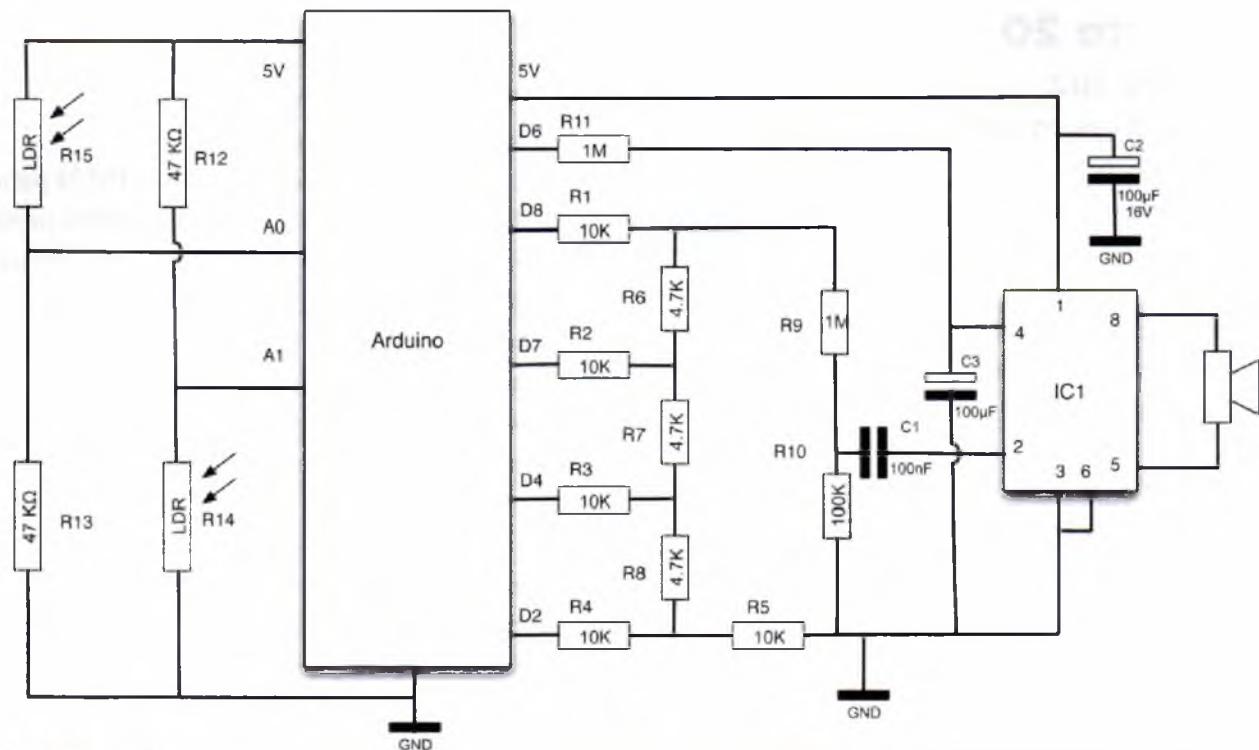


Figura 7-10 Esquema eléctrico del Proyecto 20.

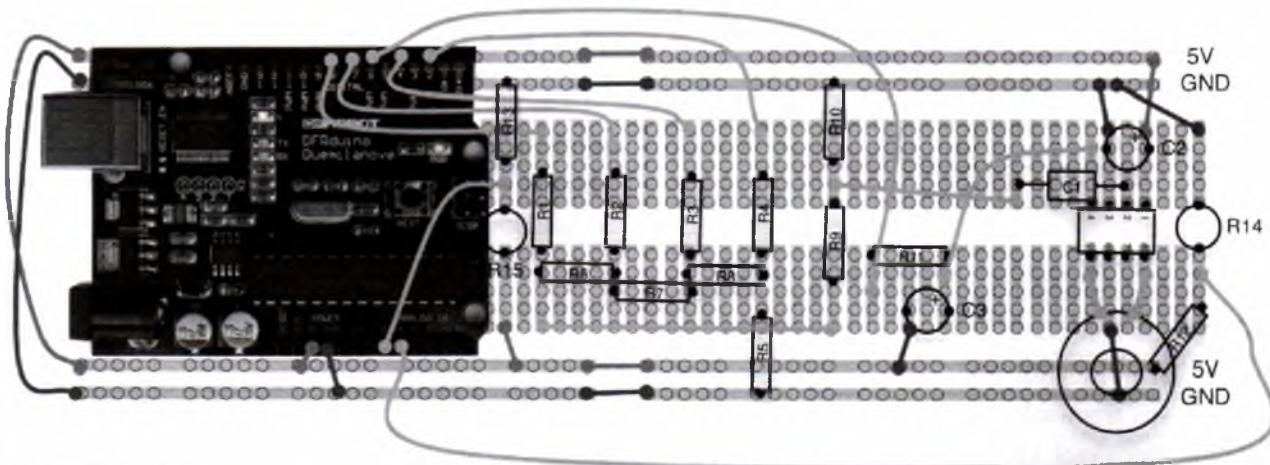


Figura 7-11 Diseño del circuito del Proyecto 20 sobre la placa de pruebas.

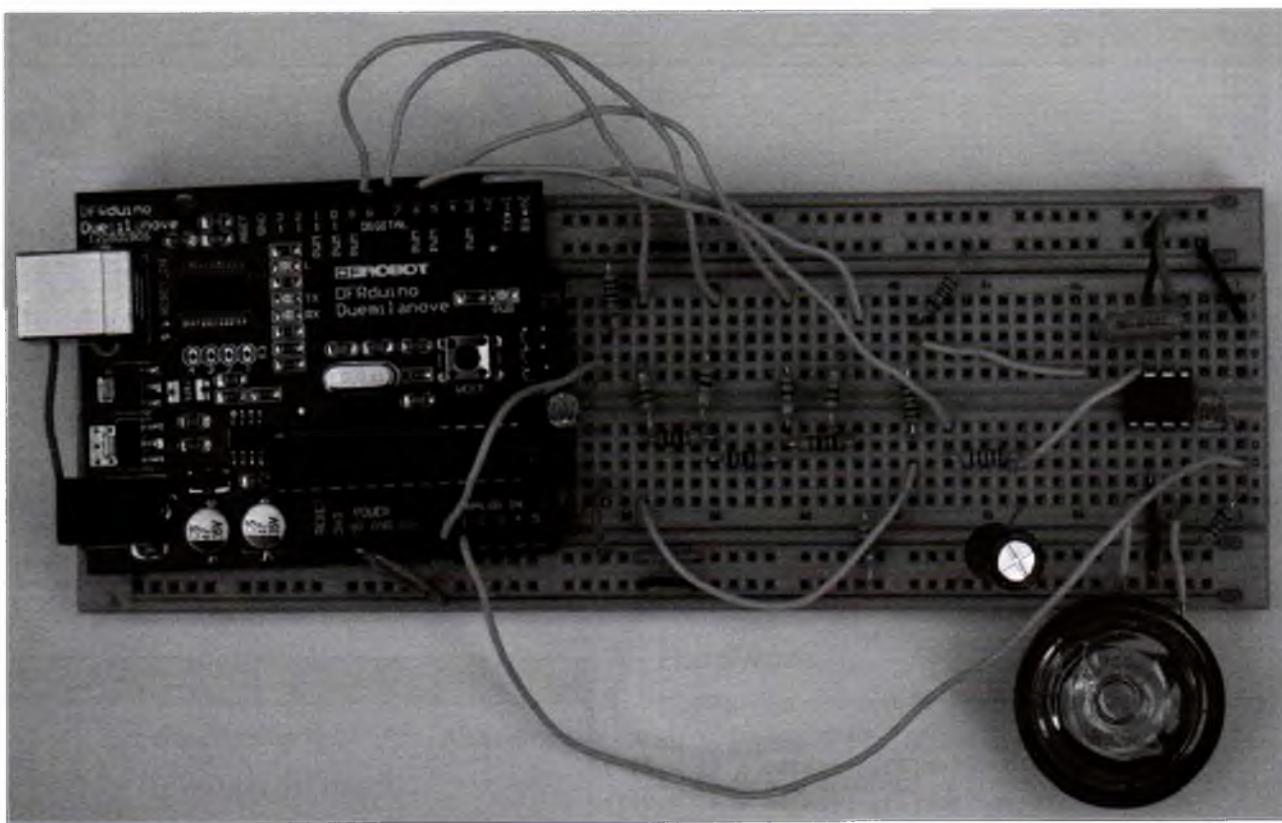


Figura 7-12 Proyecto 20. Arpa de luz.

LISTADO DE PROYECTO 20

```
int pitchInputPin = 0;
int volumeInputPin = 1;
int volumeOutputPin = 6;

int dacPins[] = {2, 4, 7, 8};
int sin16[] = {7, 8, 10, 11, 12, 13, 14, 14, 15, 14, 14, 13, 12, 11,
               10, 8, 7, 6, 4, 3, 2, 1, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6};

int count = 0;

void setup()
{
    for (int i = 0; i < 4; i++)
    {
        pinMode(dacPins[i], OUTPUT);
    }
    pinMode(volumeOutputPin, OUTPUT);
}
```

(continúa)

LISTADO DE PROYECTO 20 (continúa)

```

void loop()
{
    int pitchDelay = map(analogRead(pitchInputPin), 0, 1023, 10, 60);
    int volume = map(analogRead(volumeInputPin), 0, 1023, 10, 70);
    for (int i = 0; i < 32; i++)
    {
        setOutput(sin16[i]);
        delayMicroseconds(pitchDelay);
    }
    if (count == 10)
    {
        analogWrite(volumeOutputPin, volume);
        count = 0;
    }
    count++;
}

void setOutput(byte value)
{
    digitalWrite(dacPins[3], ((value & 8) > 0));
    digitalWrite(dacPins[2], ((value & 4) > 0));
    digitalWrite(dacPins[1], ((value & 2) > 0));
    digitalWrite(dacPins[0], ((value & 1) > 0));
}

```

Pongamos todo junto

Cargue el esquema terminado del Proyecto 20 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Para tocar el "instrumento", ponga la mano derecha sobre una de las **LDR** para controlar el volumen del sonido y la mano izquierda sobre la otra **LDR** para controlar el tono. Se pueden conseguir efectos interesantes moviendo las manos sobre las **LDR**.

Hay que tener en cuenta que, dependiendo de la luz ambiental, puede que tenga que ajustar en el **sketch** los valores de las funciones **map**.

Project 21

Medidor VU

Este proyecto (que se muestra en la Figura 7-13) utiliza una barra de LEDs de 10 segmentos para indicar el volumen del sonido captado por un micrófono. La barra de LED utilizada viene en un encapsulado **dual in line** (DIL).

El pulsador activa o desactiva el modo de funcionamiento del medidor **VU**. En el modo normal, el gráfico de barras simplemente se desplaza arriba y abajo con el volumen del sonido. En el modo máximo, el gráfico de barras registra el valor máximo y enciende el LED correspondiente, con lo que el nivel de sonido lo va subiendo gradualmente.

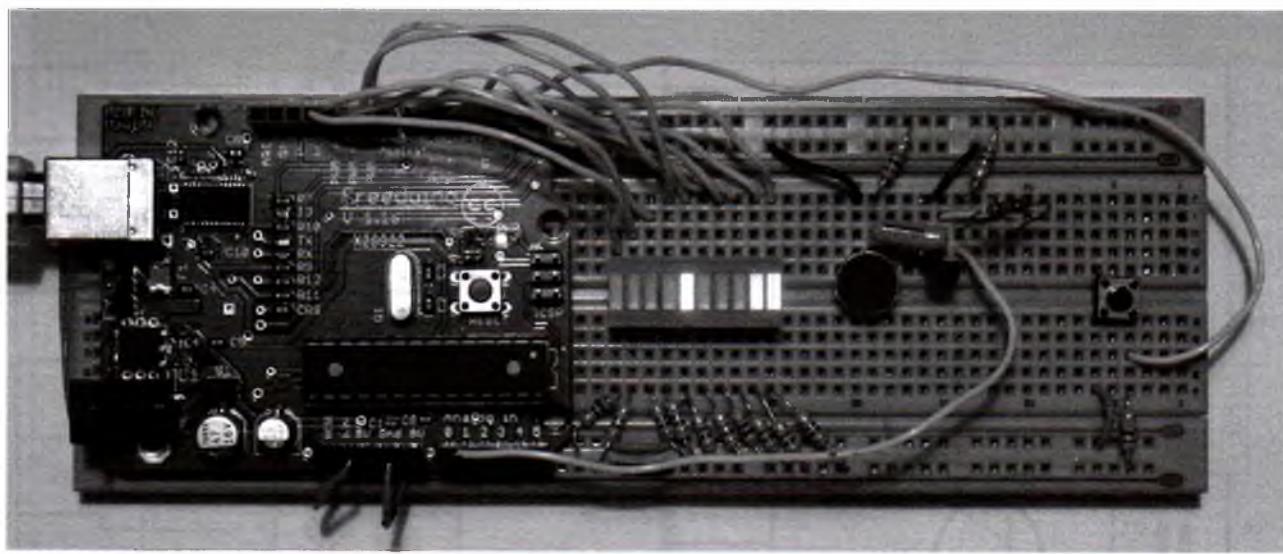


Figura 7-13 Proyecto 21. Medidor VU.

COMPONENTES Y EQUIPO		
	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o clon	1
R1, R3, R4	Resistencia 10 MΩ 0,5 W	9
R2	Resistencia 100 MΩ 0,5 W	13
R5-14	Resistencia 270 MΩ 0,5 W	6
R10	Resistencia 10 MΩ 0,5 W	9
C1	100 nF	20
	Barra de LED de 10 segmentos	35
S1	Pulsador	48
	Micrófono electret	60

Hardware

El esquema electrónico de este proyecto se muestra en la Figura 7-14. El encapsulado de la barra de LED trae un terminal de conexión por LED, necesitando cada uno su propia resistencia de limitación de corriente.

El micrófono no será capaz de entregar una señal lo suficientemente potente como para manejar la entrada analógica. Por tanto, para aumentar la señal, utilizaremos un sencillo amplificador de un transistor. Utilizaremos un montaje estándar denominado **polarización por realimentación de colector**, en la que una parte de la tensión del colector se utiliza para polarizar el transistor, consiguiendo así una amplificación lineal más holgada que si utilizáramos una comutada.

En la Figura 7-15 se muestra el diseño del circuito sobre la placa de pruebas. Puede observar que al necesitarse tantos LEDs se aumenta también el número de cables.

Software

El **sketch** para este proyecto (Listado del Proyecto 21) utiliza una matriz (**array**) de pines de LED para acortar la función **setup**. Esto también se utiliza en la función **loop**, donde vamos recorriendo cada LED y decidiendo si lo activamos o desactivamos.

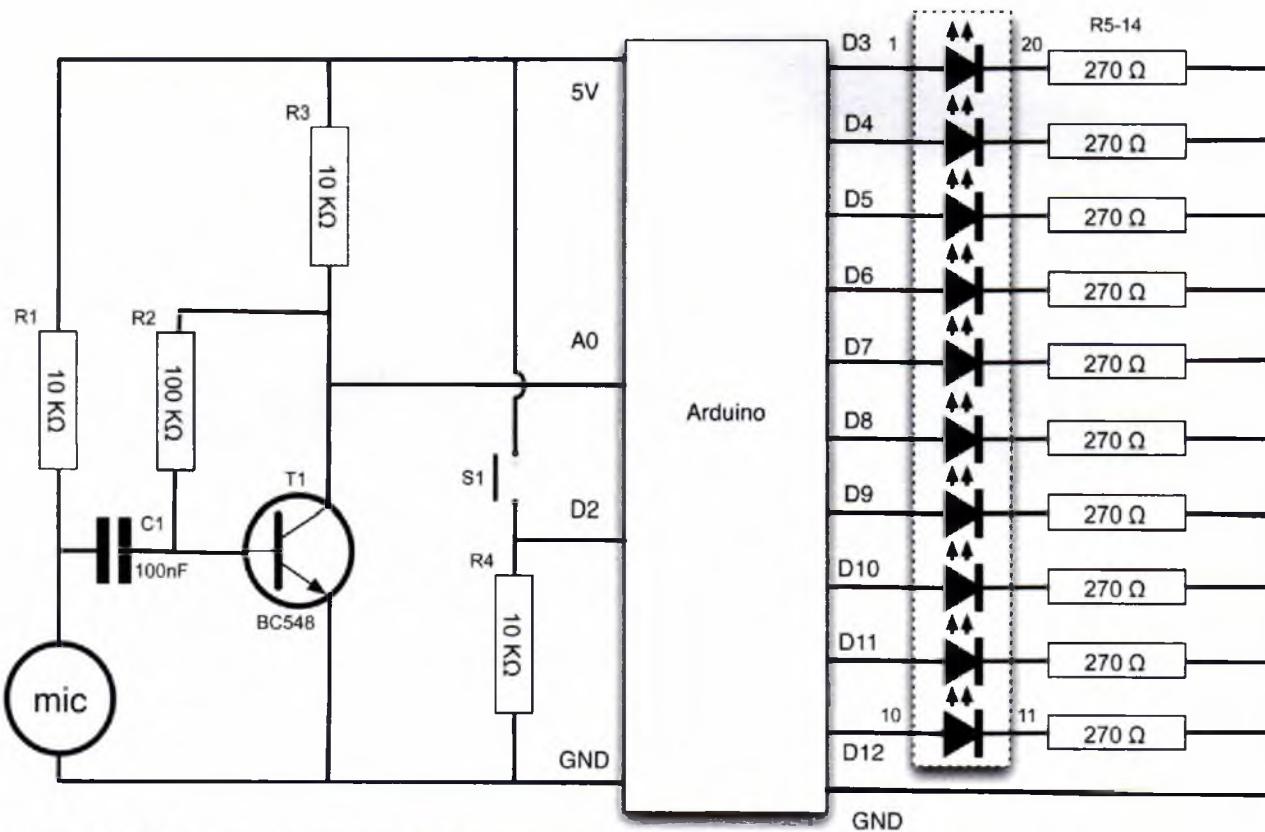


Figura 7-14 Esquema eléctrico del Proyecto 21.

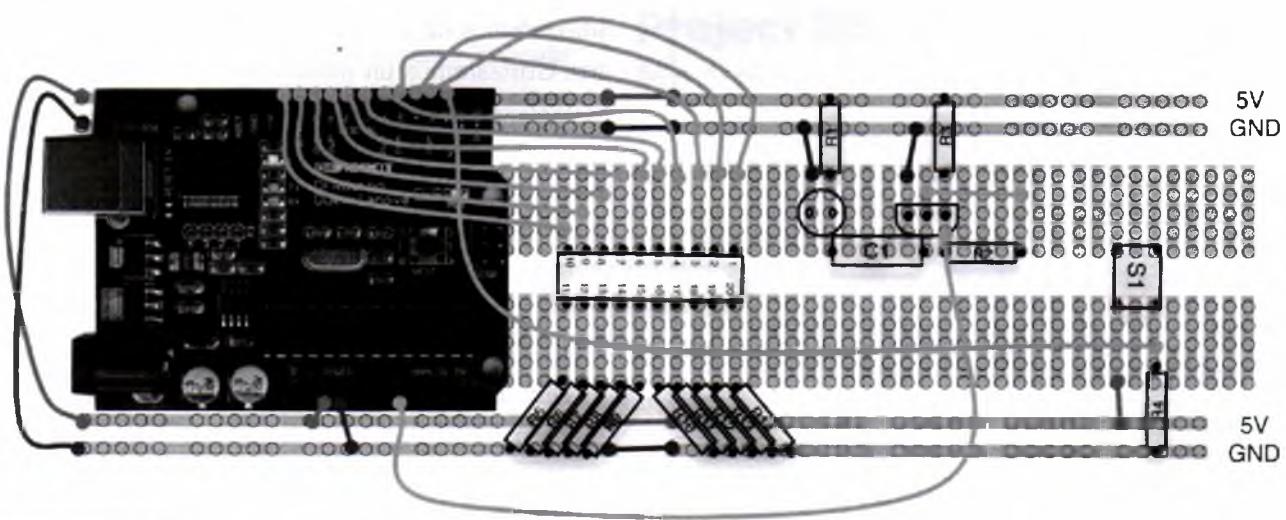


Figura 7-15 Diseño del circuito del Proyecto 21 sobre la placa de pruebas.

LISTADO DE PROYECTO 21

```

int ledPins[] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
int switchPin = 2;
int soundPin = 0;

boolean showPeak = false;
int peakValue = 0;

void setup()
{
    for (int i = 0; i < 10; i++)
    {
        pinMode(ledPins[i], OUTPUT);
    }
    pinMode(switchPin, INPUT);
}

void loop()
{
    if (digitalRead(switchPin))
    {
        showPeak = ! showPeak;
        peakValue = 0;
        delay(200); // debounce switch
    }
    int value = analogRead(soundPin);
    int topLED = map(value, 0, 1023, 0, 11) - 1;
    if (topLED > peakValue)
    {
        .
        peakValue = topLED;
    }
    for (int i = 0; i < 10; i++)
    {
        digitalWrite(ledPins[i], (i <= topLED || (showPeak && i == peakValue)));
    }
}

```

En la parte superior de la función **loop** comprobamos si el pulsador está pulsado; si es así, cambiamos el modo. El comando **!** invierte un valor, lo que convierte verdadero (**true**) en falso (**false**) y falso (**false**) en verdadero (**true**). Por esta razón, a veces se le conoce como el "operador de marketing". Tras cambiar el modo, reiniciamos el valor máximo a **0** y, a continuación, lo retardamos **200** milisegundos

para evitar que posibles rebotes del teclado vuelvan a cambiar otra vez el modo.

El nivel de sonido se lee del pin analógico **0** y, a continuación, utilizamos la función **map** para convertir un rango de **0** a **1023** a un valor entre **0** y **9**, que será el LED superior a iluminar. Esto se ajusta ligeramente ampliando el rango de **0** a **11** para, a continuación, restarle **1**. Así se evita que los dos

LED más bajos quedan permanentemente encendidos debido a la polarización del transistor.

Luego continuamos recorriendo los números **0** y **9** y utilizamos una expresión booleana que devuelve true (verdadero) (y, por lo tanto, enciende el LED) si "i" es menor o igual que el LED superior. En realidad, la cosa es más complicada, ya que también deberíamos mostrar ese LED si estamos en modo pico y da la casualidad de que ese LED resulta ser el **peakValue**.

Pongamos todo junto

Cargue el esquema terminado del Proyecto 21 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Resumen

Con este proyecto concluimos nuestros proyectos basados en sonido. En el siguiente capítulo vamos a ver cómo se utiliza la placa Arduino para controlar potencia, un tema que creemos que también le puede resultar interesante.

CAPÍTULO 8

Proyectos de energía eléctrica

TRAS HABER EXAMINADO LA LUZ y el sonido, dirigimos ahora nuestra atención a controlar la energía. En esencia, esto significa encender y apagar cosas y controlar su velocidad. Esto vamos a aplicarlo principalmente a motores y a dispositivos láser y al esperado proyecto del Láser servo-controlado.

Proyecto 22 Termostato LCD

La temperatura en el cuarto de trabajo de todo genio que se precie debe estar regulada, no siendo bueno que nadie se resfrie.

Este proyecto utiliza una pantalla LCD y un **termistor** como sensor de temperatura para mostrar la temperatura actual y la temperatura programada. Utiliza un **codificador giratorio** que permite cambiar la temperatura programada. El botón del codificador también actúa como conmutador de anulación.

Cuando la temperatura medida es inferior a la temperatura programada, se activa un **relé**. Los relés son componentes electromagnéticos clásicos que activan un interruptor mecánico cuando una corriente circula a través de una bobina. Tienen una serie de ventajas, como las de poder commutar altas tensiones y corrientes, lo que los hace adecuados para controlar equipos conectados a la red eléctrica. También separan eléctricamente el circuito de control (bobina), del circuito de commutación, de modo

que la tensión de la red jamás pueda mezclarse con la baja tensión del circuito, lo que es muy positivo desde el punto de vista de la seguridad.

Si el lector decide utilizar este proyecto para accionar dispositivos conectados a la red eléctrica, sólo debe lanzarse a realizarlo si realmente sabe lo que está haciendo, y además debe proceder con extrema cautela. La red eléctrica es muy peligrosa. Son muchas las personas que han perdido la vida por su causa y otras muchas las que han sufrido importantes quemaduras.

COMPONENTES Y EQUIPO

	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o clon	1
R1	Termistor beta = 4090 33 KΩ	18
R2	Resistencia 33 KΩ 0,5 W	10
R3-5	Resistencia 100 KΩ 0,5 W	13
R6	Resistencia 270 Ω 0,5 W	6
R7	Resistencia 1 KΩ 0,5 W	7
D1	LED rojo 5 mm	23
D2	1N4004	38
T1	BC548	40
	Relé 5V	61
	Módulo LCD HD44780	58
	Tira de pines 2.54 mm paso	55

LISTADO DE PROYECTO 22 (continúa)

```
boolean override = false;
float hysteresis = 0.25;

void setup()
{
    lcd.begin(2, 20);
    pinMode(ledPin, OUTPUT);
    pinMode(relayPin, OUTPUT);
    pinMode(aPin, INPUT);
    pinMode(bPin, INPUT);
    pinMode(buttonPin, INPUT);
    lcd.clear();
}

void loop()
{
    static int count = 0;
    measuredTemp = readTemp();
    if (digitalRead(buttonPin))
    {
        override = ! override;
        updateDisplay();
        delay(500); // debounce
    }
    int change = getEncoderTurn();
    setTemp = setTemp + change * 0.1;
    if (count == 1000)
    {
        updateDisplay();
        updateOutputs();
        count = 0;
    }
    count++;
}

int getEncoderTurn()
{
    // devuelve -1, 0, o +1
    static int oldA = LOW;
    static int oldB = LOW;
    int result = 0;
    int newA = digitalRead(aPin);
    int newB = digitalRead(bPin);
    if (newA != oldA || newB != oldB)
    {
        // algo ha cambiado
        if (oldA == LOW && newA == HIGH)
```

(continúa)

LISTADO DE PROYECTO 22 (continúa)

```
{  
    result = -(oldB * 2 - 1);  
}  
}  
oldA = newA;  
oldB = newB;  
return result;  
}  
  
float readTemp()  
{  
    long a = analogRead(analogPin);  
    float temp = beta / (log(((1025.0 * resistance / a) - 33.0) / 33.0) +  
        (beta / 298.0)) - 273.0;  
    return temp;  
}  
  
void updateOutputs()  
{  
    if (override || measuredTemp < setTemp - hysteresis)  
    {  
        digitalWrite(ledPin, HIGH);  
        digitalWrite(relayPin, HIGH);  
    }  
    else if (!override && measuredTemp > setTemp + hysteresis)  
    {  
        digitalWrite(ledPin, LOW);  
        digitalWrite(relayPin, LOW);  
    }  
}  
  
void updateDisplay()  
{  
    lcd.setCursor(0,0);  
    lcd.print("Actual: ");  
    lcd.print(adjustUnits(measuredTemp));  
    lcd.print(" °");  
    lcd.print(mode);  
    lcd.print(" ");  
  
    lcd.setCursor(0,1);  
    if (override)  
    {  
        lcd.print(" OVERRIDE ON ");  
    }  
    else
```

(continúa)

LISTADO DE PROYECTO 22 (continúa)

```

{
  lcd.print("Set:    ");
  lcd.print(adjustUnits(setTemp));
  lcd.print(" °");
  lcd.print(mode);
  lcd.print(" ");
}

float adjustUnits(float temp)
{
  if (mode == 'C')
  {
    return temp;
  }
  else
  {
    return (temp * 9) / 5 + 32;
  }
}

```

un sistema simple de control de encendido y apagado. Cuando la temperatura cae por debajo del nivel programado (llamado **punto de consigna**), la alimentación se conecta y la habitación se calienta hasta llegar a la temperatura por encima del punto de consigna programado; posteriormente, la habitación vuelve a enfriarse, hasta que la temperatura ambiente queda de nuevo por debajo del punto de consigna programado, en cuyo momento la calefacción se activa de nuevo, y así sucesivamente.

Pasado un cierto tiempo, cuando la temperatura se estabiliza cerca del punto de consigna programado, las conexiones y desconexiones empiezan a ser demasiado frecuentes, lo que no es deseable, además de que pueden producir un deterioro prematuro de los contactos del **relé**.

Una forma de minimizar este efecto es introducir algo llamado **histéresis**. Puede que incluso haya observado en el **sketch** una variable llamada **hysteresis**, que se ha establecido a un valor de 0,25°C.

La figura 8-3 muestra cómo utilizamos el valor de **histéresis** para evitar que se produzca el "hunting" con excesiva frecuencia.

A medida que aumenta la temperatura, con la alimentación encendida, nos iremos aproximando al punto de consigna; sin embargo, el equipo no se

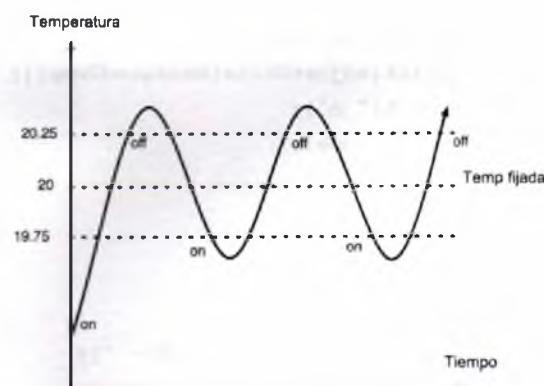


Figura 8-3 La histéresis en los sistemas de control.

apagará hasta que se haya superado este punto de consigna (valor programado) más el valor de **histéresis**. De igual modo, al disminuir la temperatura, la alimentación no volverá a aplicarse simplemente al caer justo por debajo del punto de consigna programado, sino cuando supera este punto, menos el valor de **histéresis**.

Si la actualización de la pantalla la hiciésemos de forma continua, es decir, cada vez que damos una vuelta al bucle principal, se produciría un indeseable efecto de parpadeo. Para evitarlo, la actualización la haremos cada 1000 vueltas del bucle. No obstante, esto significa que se actualizará tres o cuatro veces por segundo. Para ello, utilizamos la técnica de incrementar una variable **counter** (contador) cada vez que da la vuelta al bucle. Cuando llega a 1000, actualizamos la pantalla y volvemos a poner el contador a 0.

Utilizando **lcd.clear()**, cada vez que cambiamos la pantalla también hará que parpadee. Por lo tanto, basta con escribir las nuevas temperaturas encima de las antiguas. Esta es la razón por la que rellenamos el mensaje " OVERRIDE ON" con espacios, para que se borrara cualquier texto que hubiera podido aparecer anteriormente en los extremos.

Pongamos todo junto

Cargue el Sketch terminado del Proyecto 22 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

En la Figura 8-4 se muestra el proyecto completo. Para probar el proyecto, gire el **codificador giratorio**, y ajuste la temperatura establecida ligeramente por encima de la temperatura real. El LED debería encenderse. A continuación, ponga los dedos en el termistor para calentarlo. Si todo funciona correctamente, al aumentar la temperatura el LED debe apagarse y escucharse el clic del **relé**.

También puede probar el funcionamiento del **relé** conectando un polímetro en modo de continuidad (zumbador) en los contactos de salida del **relé**.

Por favor, es vital que recuerde que si tiene la intención de utilizar su **relé** para comutar la red eléctrica, primero coloque este proyecto en una **Protoshield** soldada correctamente. Segundo, tenga mucho cuidado y compruebe y revise dos veces lo que está haciendo. ¡La red eléctrica mata!

El **relé** sólo debe utilizarse en aplicaciones de baja tensión, a menos que vaya a utilizar este diseño para montarlo en una placa soldada correctamente.

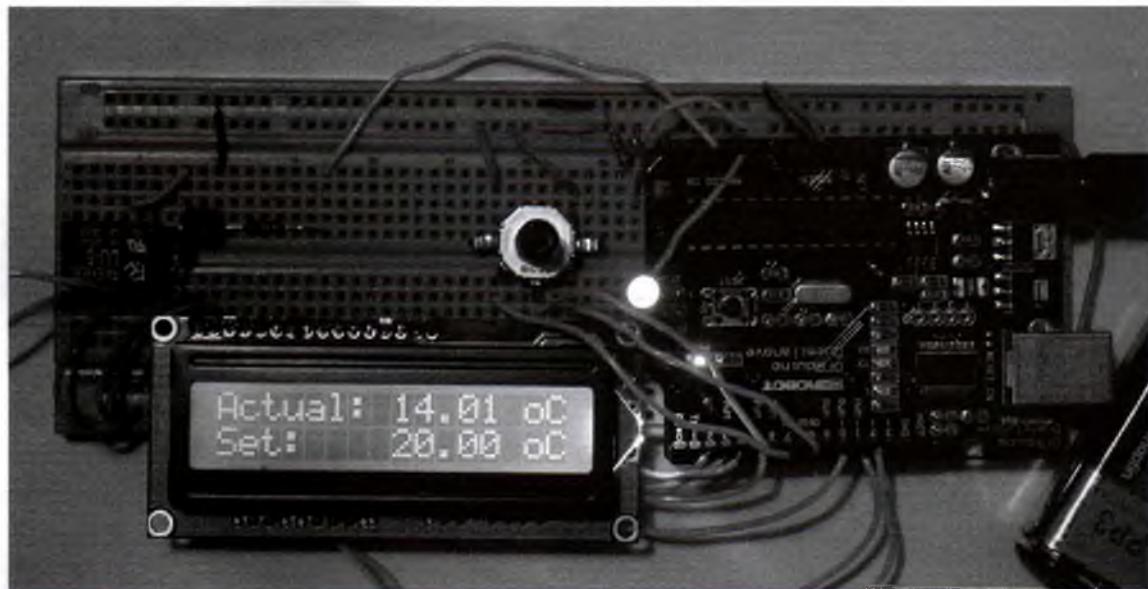


Figura 8-4 | Proyecto 22. Termostato LCD.

Proyecto 23

Ventilador controlado por ordenador

Una de las piezas útiles que se puede sacar de un viejo PC es el ventilador (Figura 8-5). En nuestro caso, vamos a utilizar uno de estos ventiladores para mantenernos frescos durante el verano. A estas alturas es evidente que un simple interruptor **on/off** no estaría en consonancia con nuestra manera de hacer las cosas; por ello, vamos a hacer que la velocidad del ventilador sea controlable desde nuestro ordenador.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o clon	1
R1 Resistencia 270 Ω 0,5 W	6
T1 Transistor de potencia BD139	41
M1 Ventilador ordenador 12 V	63
Fuente alimentación 12 V 1 A	62

Si resulta que no tiene ningún ordenador estropeado al que sacarle las tripas, no se preocupe: un ventilador nuevo se puede comprar bastante barato.

Hardware

Podemos controlar la velocidad del ventilador utilizando la salida analógica (**PWM**) y haciendo que maneje un transistor de potencia para controlar el motor. Puesto que estos ventiladores de ordenador suelen ser de **12 V**, vamos a utilizar una fuente de alimentación externa para proporcionar la energía para alimentar el ventilador.

La Figura 8-6 muestra el esquema electrónico del proyecto, y la Figura 8-7 el diseño de la placa de pruebas.

Software

En realidad, es un **sketch** muy simple (Listado del Proyecto 23). Esencialmente, lo único que necesitamos es leer un número del **0** al **9** del **USB** y ejecutar un **analogWrite** al **motorPin** de ese valor, multiplicado por **28**, para escalarlo hasta un número entre **0** y **252**.

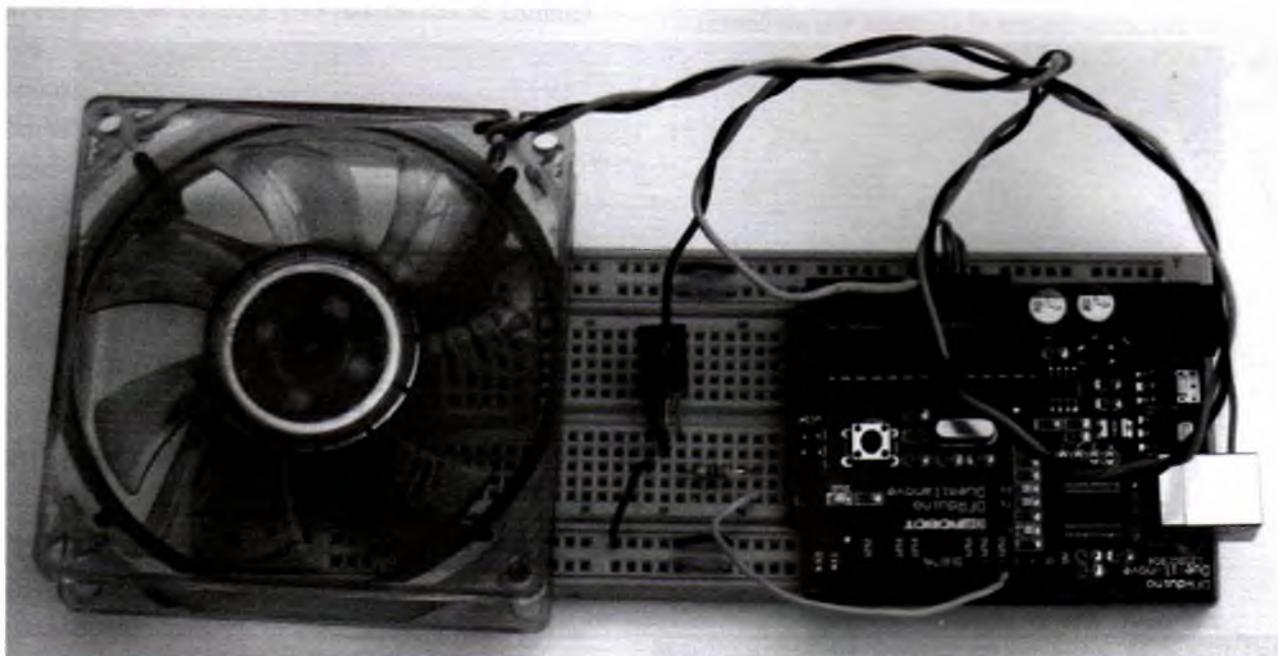


Figura 8-5 Proyecto 23. Ventilador controlado por ordenador.

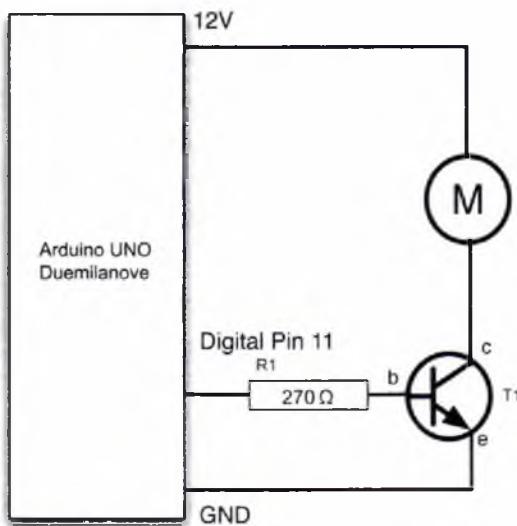


Figura 8-6 Esquema electrónico del Proyecto 23.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 23 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Hay tan pocos componentes en este proyecto que en realidad podríamos empalmar las patas de los mismos y colocarlos directamente en la placa Arduino, prescindiendo totalmente de la placa de pruebas.

LISTADO DE PROYECTO 23

```
int motorPin = 11;

void setup()
{
    pinMode(motorPin, OUTPUT);
    analogWrite(motorPin, 0);
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available())
    {
        char ch = Serial.read();
        if (ch >= '0' && ch <= '9')
        {
            int speed = ch - '0';
            analogWrite(motorPin, speed
                        * 28);
        }
    }
}
```

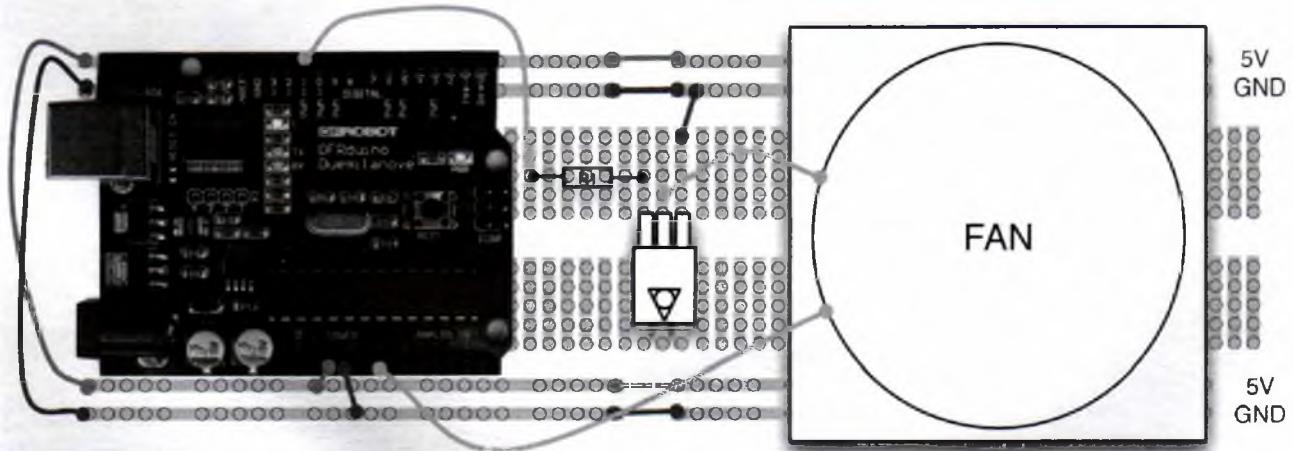


Figura 8-7 Diseño del circuito del Proyecto 23 sobre la placa de pruebas.

Controladores de puente H

Para cambiar la dirección en la que gira un motor, es necesario invertir la dirección en la que circula la corriente. Para ello se necesitan cuatro **interruptores o transistores**. La Figura 8-8 muestra cómo funciona este sistema, utilizando **interruptores** en una distribución llamada, por razones obvias, **puente H**.

En la Figura 8-8, S1 y S4 están cerrados y S2 y S3 están abiertos. Esto permite que la corriente circule por el motor, siendo el terminal A el positivo y el terminal B el negativo. Si invertimos los conmutadores, de forma que S2 y S3 estén cerrados y S1 y S4 abiertos, entonces B sería el positivo y A el negativo, y el motor girará en dirección opuesta.

Sin embargo, quizás haya divisado un peligro con este circuito. Y es que, si por casualidad S1 y S2 están cerrados, entonces la tensión positiva estará conectada directamente a la tensión negativa, lo que producirá un cortocircuito. Lo mismo ocurriría si S3 y S4 estuvieran ambos cerrados al mismo tiempo.

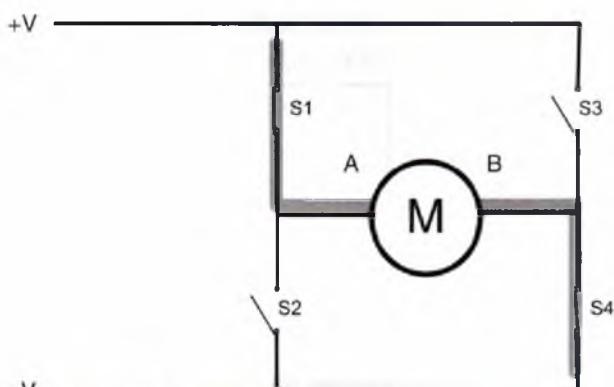


Figura 8-8 Un puente H.

En el siguiente proyecto, para controlar un motor eléctrico vamos a utilizar **transistores** en lugar de **conmutadores**.

Proyecto 24 Hypnotizer

El control de la mente es, sin duda, una de esas cosas que siempre ronda en la cabeza de los más inquietos. En este proyecto (véase la Figura 8-9) vamos a manejar el control completo de un motor con el fin de controlar no sólo su velocidad, sino

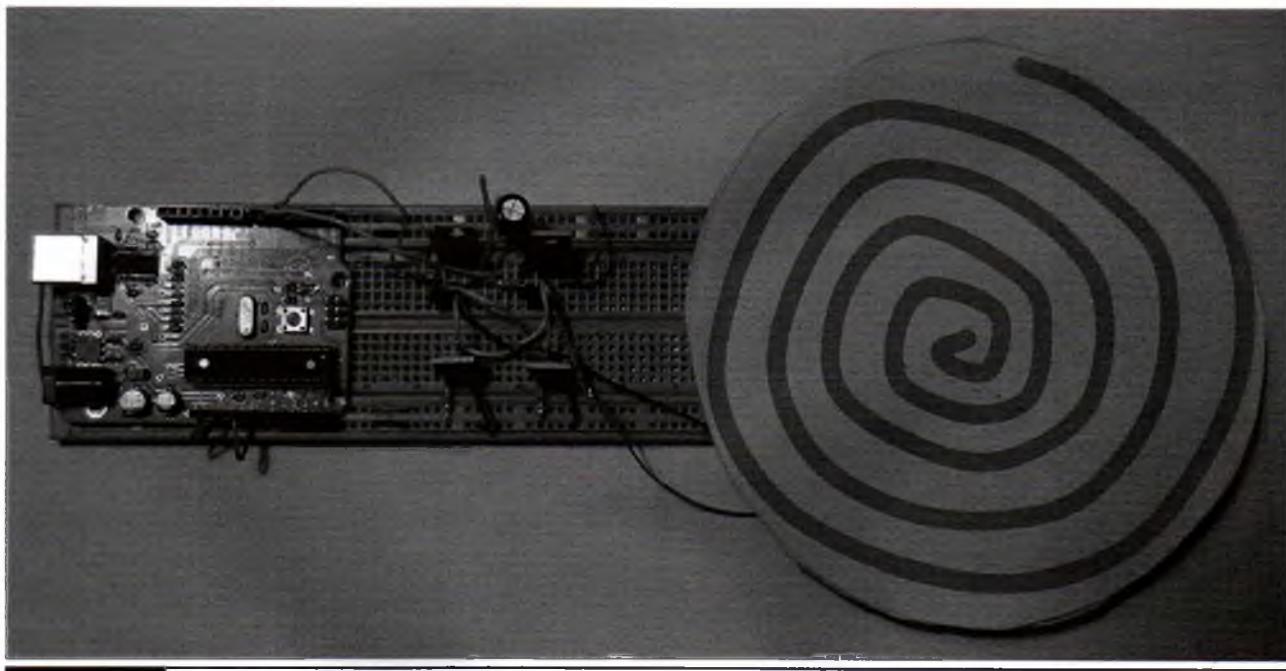


Figura 8-9 Proyecto 24. Hipnotizador.

COMPONENTES Y EQUIPO

Descripción	Apéndice
Placa Arduino UNO o Duemilanove o clon	1
T2, T4 MOSFET de potencia canal N FQP33N10	43
T1, T3 MOSFET de potencia canal P, FQP27P06	44
T4, T5 BC548	40
R1-6 Resistencia 10 KΩ 0,5 W	9
M1 Motor 6 V	64

también para que gire hacia la derecha o hacia la izquierda. Unido al motor colocaremos un disco espiral giratorio para fijar la atención y cautivar la mente de las desafortunadas víctimas.

El motor que hemos utilizado en este proyecto lo hemos recuperado de una unidad de CD de un ordenador estropeado. Una alternativa barata para el motor sería algún juguete viejo de niño que tenga un motor. Y para que podamos colocar en él el disco hipnótico, los más adecuados son los que cuentan con engranajes para mover el eje.

Hardware

El Esquema electrónico del hipnotizador se muestra en la Figura 8-10. El diseño es el de un **ponte H** estándar. Observe que para el control de potencia principal estamos utilizando lo que se llaman transistores de tipo **MOSFET** (Transistor de efecto de campo de puerta aislada) en lugar de los transistores **bipolares**. En teoría, esto nos permitirá controlar motores de bastante potencia, pero además tienen la ventaja de que los **MOSFETs** apenas se calientan con nuestro pequeño motor y, por tanto, no necesitaremos disipadores térmicos.

Las conexiones de puerta de los **MOSFETs** inferiores van astutamente conectadas a las salidas de

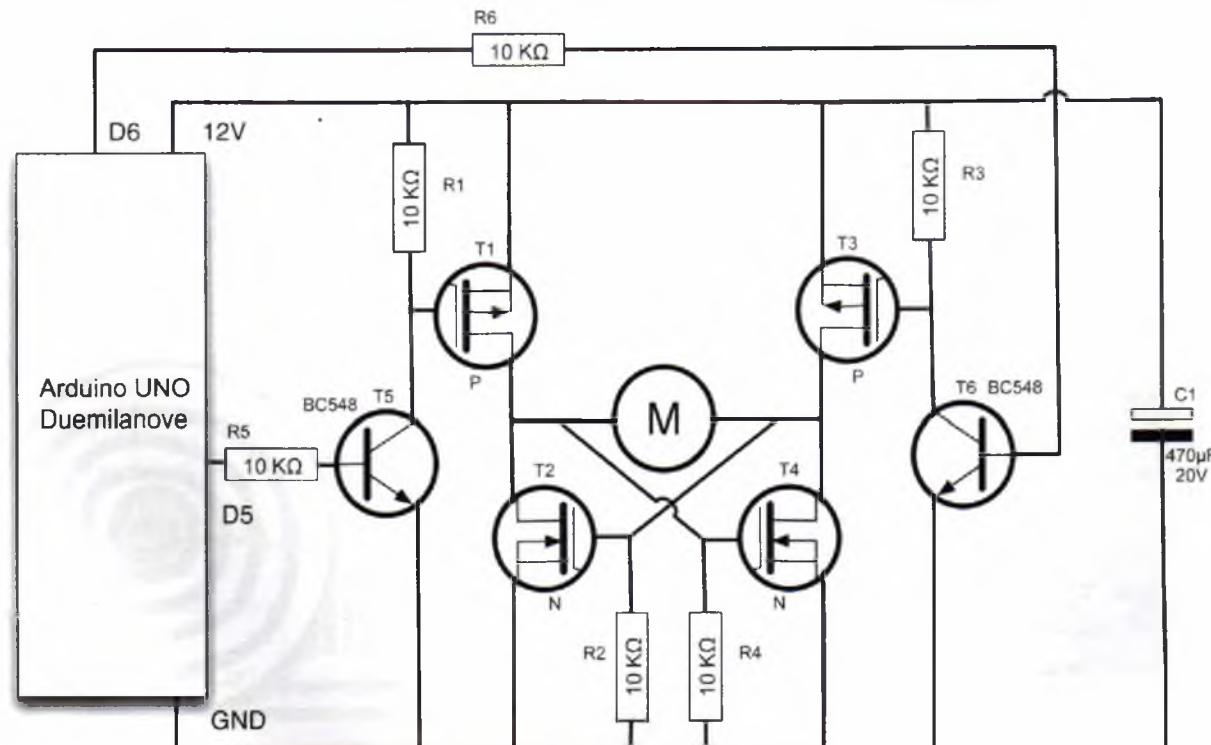


Figura 8-10 | Esquema electrónico del Proyecto 24.

los transistores colocados en el lado opuesto, de forma que cuando se activa **T1**, **T4** se activará también automáticamente; y cuando se activa **T3**, también lo hará **T2**.

Las resistencias **R1** a **R4** aseguran que el estado predeterminado de **T1** a **T4** sea apagado, colocando en estado alto las puertas de los **MOSFETs** del canal P y en estado bajo las de canal N.

T5 y **T6** son transistores **bipolares** de baja corriente que se utilizan para activar **T1** y **T3**, respectivamente. En este caso, podríamos prescindir de estos transistores y manejar las puertas de **T1** y **T3** directamente desde la placa Arduino. Sin embargo, para ello, habría que invertir la lógica (las puertas en estado alto apagarían el transistor). Aunque podríamos resolver esto mediante software, hay otra razón para utilizar estos dos componentes adicionales, y es que con ellos podemos utilizar este circuito para controlar motores de mayor tensión, simplemente aumentando la tensión de alimentación. Si estuviéramos manejando directamente los **MOSFETs**, entonces la salida positiva de la Arduino tendría que tener más de 5 V para desactivar el **MOSFET** si la alimentación del motor fuera de 9 V o más, algo que no es posible.

Esto hace que el circuito esté sobredimensionado, lo cual resulta especialmente interesante para todos aquellos con grandes requerimientos en el tema de motores.

Por último, **C1** filtra algunos de los impulsos eléctricos que se producen cuando se maneja un dispositivo como un motor.



Figura 8-12 Espiral para el hipnotizador.

La Figura 8-11 muestra el diseño del circuito del proyecto sobre la placa de pruebas.

Para funcionar, nuestro hypnotizador necesita un diseño en espiral. Si quiere, puede utilizar la Figura 8-12 para hacer una copia y pegarlo en el ventilador. O, como alternativa, en www.arduinoevilgenius.com tiene a su disposición para que lo imprima una versión más colorida de la espiral.

La espiral de papel la hemos recortado y pegado en un cartón un poco más duro y, posteriormente, la hemos pegado al engranaje del extremo del motor.

Software

Lo importante en este **sketch** (Listado del Proyecto 24) es asegurarse de la imposibilidad de que todos

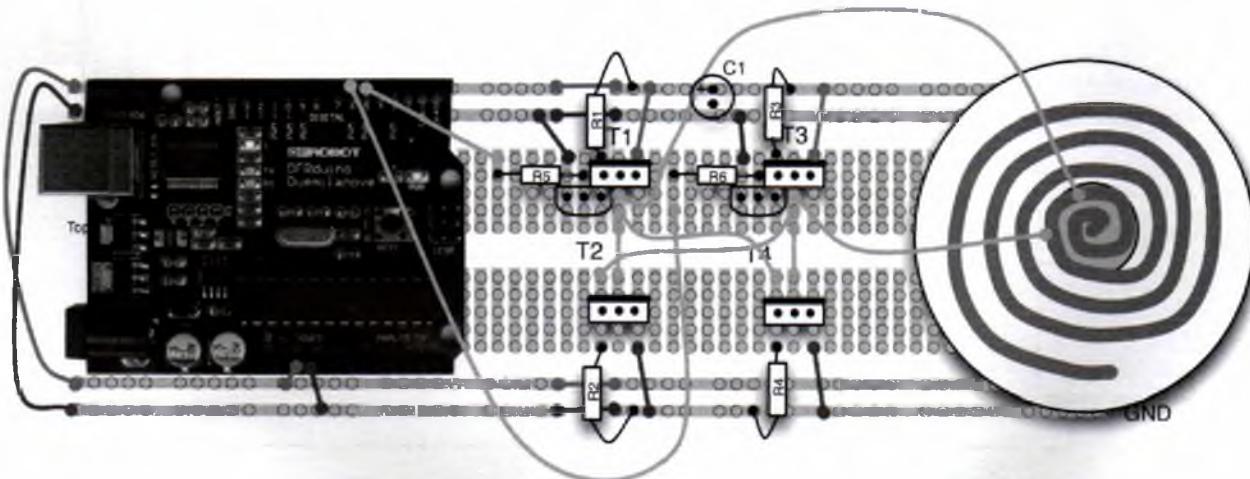


Figura 8-11 Diseño del circuito del Proyecto 24 sobre la placa de pruebas.

LISTADO DE PROYECTO 24

```
int t1Pin = 5;
int t3Pin = 6;

int speeds[] = {20, 40, 80, 120, 160, 180, 160, 120, 80, 40, 20,
                 -20, -40, -80, -120, -160, -180, -160, -120, -80, -40, -20};
int i = 0;

void setup()
{
    pinMode(t1Pin, OUTPUT);
    digitalWrite(t1Pin, LOW);
    pinMode(t3Pin, OUTPUT);
    digitalWrite(t3Pin, LOW);
}

void loop()
{
    int speed = speeds[i];
    i++;
    if (i == 22)
    {
        i = 0;
    }
    drive(speed);
    delay(1500);
}

void allOff()
{
    digitalWrite(t1Pin, LOW);
    digitalWrite(t3Pin, LOW);
    delay(1);
}

void drive(int speed)
{
    allOff();
    if (speed > 0)
    {
        analogWrite(t1Pin, speed);
    }
    else if (speed < 0)
    {
        analogWrite(t3Pin, -speed);
    }
}
```

los transistores estén activados al mismo tiempo. Si esto ocurriera, se produciría un cierto olor a quemado y algo se quemará y “morirá” para siempre.

Antes de activar cualquier transistor, debemos desconectar la totalidad de los mismos usando la función **allOff**. Además, la función **allOff** incluye un ligero retardo (**delay**) para asegurar que los transistores se han desactivado correctamente antes de encender nada.

El **sketch** utiliza una matriz de velocidades (**speed**), para controlar progresivamente la velocidad del disco. Esto hace que el disco gire cada vez más rápido en una dirección. Luego comenzará a girar cada vez más lento, hasta que, finalmente, invierte la dirección para, posteriormente, empezar a girar de nuevo, cada vez más rápido. Y así sucesivamente. Puede que tenga que ajustar ligeramente la matriz para adaptarla a su motor. Las velocidades que hay que especificar en la matriz pueden variar de un motor a otro, por lo que, probablemente, en su caso también necesite ajustarlas.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 24 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Antes de conectar la alimentación a este proyecto preste atención y compruebe bien el cableado. Puede comprobar cada uno de los circuitos del **puente H** conectando a masa los cables de control de los **pines digitales 5** y **6**. Luego conecte uno de los cables a 5 V, con lo que el motor debe comenzar a girar en un sentido. Vuelva a conectar el cable a tierra y luego conecte el otro cable a 5 V; el motor debe comenzar a girar en sentido contrario.

Servomotores

Los servomotores son unos pequeños y magníficos componentes que se utilizan con frecuencia en los coches teledirigidos por radio para controlar la dirección, y en los aviones para controlar los aletines. Los hay de distintos tamaños, para las distintas aplicaciones, y su amplia utilización en los modelos de aficionados hace que sean relativamente baratos.

A diferencia de los motores normales, estos no giran continuamente, sino que, utilizando una señal

PWM, pueden configurarse para que giren un determinado ángulo. Para ello incorporan toda la necesaria electrónica de control, con lo que lo único que hay que proporcionarles es alimentación (que para muchos dispositivos puede ser de 5 V), y una señal de control que podemos generar desde la placa Arduino.

A lo largo de los años, la interfaz para **servos** se ha estandarizado. El **servo** debe recibir un flujo continuo de pulsos al menos cada 20 milisegundos. El ángulo que vaya a mantener el **servo** viene determinado por la anchura del pulso. Una anchura de pulso de 1,5 milisegundos establecerá el **servo** en su punto medio, o **90 grados**. Una anchura de pulso de 1,75 milisegundos normalmente lo desplazará **180 grados**, y una menor, de 1,25 milisegundos, fijará un ángulo de **0 grados**.

Proyecto 25

Láser servo-controlado

Este proyecto (consulte la Figura 8-13) utiliza dos **servomotores** para dirigir un **diodo láser**. Verá que el láser se puede mover con bastante rapidez, por lo que podrá utilizarlo para “escribir” en paredes distantes.

Atención: este es un láser de verdad. No es de gran potencia, sólo 3 mW, pero, en cualquier caso, no debe dirigir el haz del láser ni a sus ojos ni a los de otras personas, pues podría causar daños en la retina.

COMPONENTES Y EQUIPO

	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o clon	1
D1	Diodo láser rojo 3 mW	32
M1, M2	Servo motores de 9 g	65
R1	Resistencia 100 Ω 0,5 W	5
	Protoshield Arduino (opcional)	3
	Tira 6 pines 2,5 mm (opcional)	55
	Tira zócalos (2 zóc.) 2,5 mm (opcional)	56

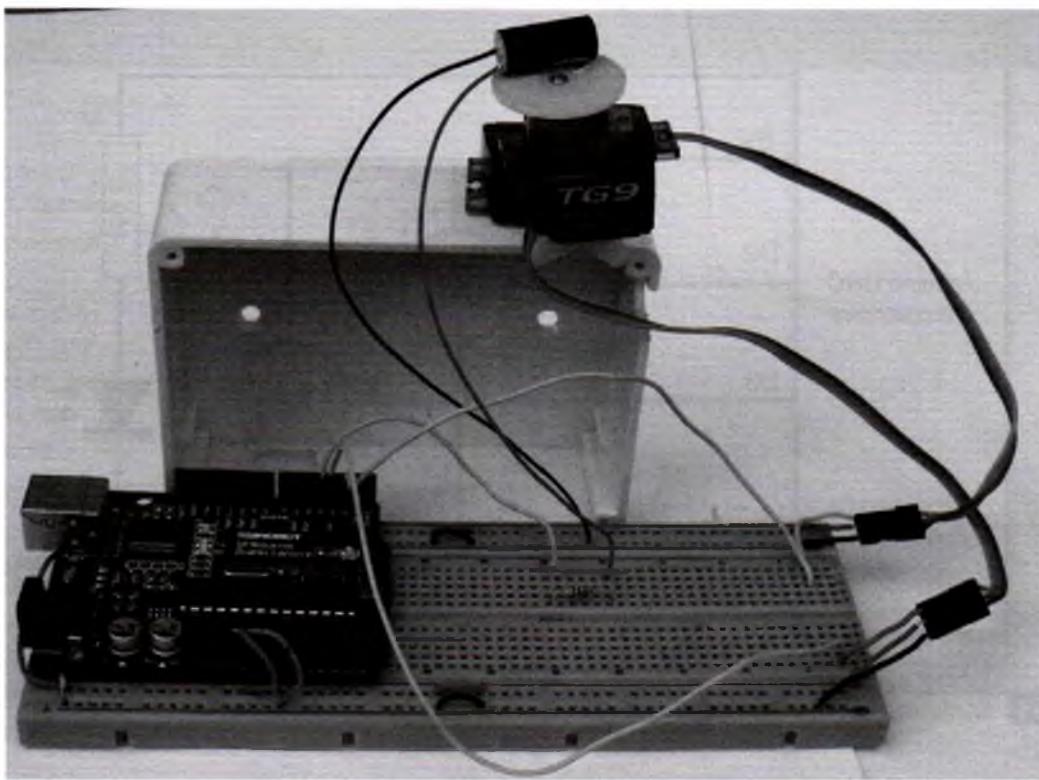


Figura 8-13 Proyecto 25. Láser servo-controlado.

Hardware

El circuito eléctrico del proyecto se muestra en la Figura 8-14, siendo éste bastante sencillo. Los **servos** sólo tienen tres cables. En cada servo, el cable marrón se conecta a tierra, el cable rojo a +5 V, y el naranja (control) a las **salidas digitales 2 y 3**. Los extremos de los servos tienen unos conectores diseñados para ser enchufados en las **tiras de pines**. Puede utilizar cable rígido para conectarlos a la placa de pruebas.

El **diodo láser** se maneja desde D4 exactamente igual que un LED normal, con una resistencia de limitación de corriente.

Los **servos** generalmente se suministran con una serie de "brazos" que se encajan en la rueda de plástico y que se fijan mediante un tornillo. Uno de los servos va pegado en uno de esos brazos (véase la Figura 8-15). Luego el brazo se instala en el otro servo. No coloque todavía el tornillo de sujeción, ya que necesitará ajustar el ángulo. Pegue el **diodo**

láser a un segundo brazo y colóquelo en el **servo**. Suele ser una buena idea pegar parte del cable del láser al brazo para impedir que se produzca tensión en el cable a la salida del láser. Para ello puede amarrarlo con un trozo de cable fijado sobre dos agujeros del brazo, como puede ver en la Figura 8-17.

Ahora tiene que colocar el **servo** inferior en una caja o en algo que le sirva de apoyo. En la Figura 8-15 se puede ver cómo se ha colocado en la caja de un antiguo proyecto. Asegúrese de entender bien el movimiento del **servo** antes de pegarlo definitivamente a algo. En caso de duda, espere hasta que haya instalado el software y pruebe el proyecto simplemente sujetando el **servo** inferior antes de pegarlo en su sitio. Una vez que esté seguro de que todo está en el lugar correcto, ajuste los tornillos de sujeción en los brazos del **servo**.

En la Figura 8-13 puede ver cómo hemos utilizado la placa de pruebas para fijar los distintos cables. No hay componentes, excepto la resistencia de la placa de pruebas.

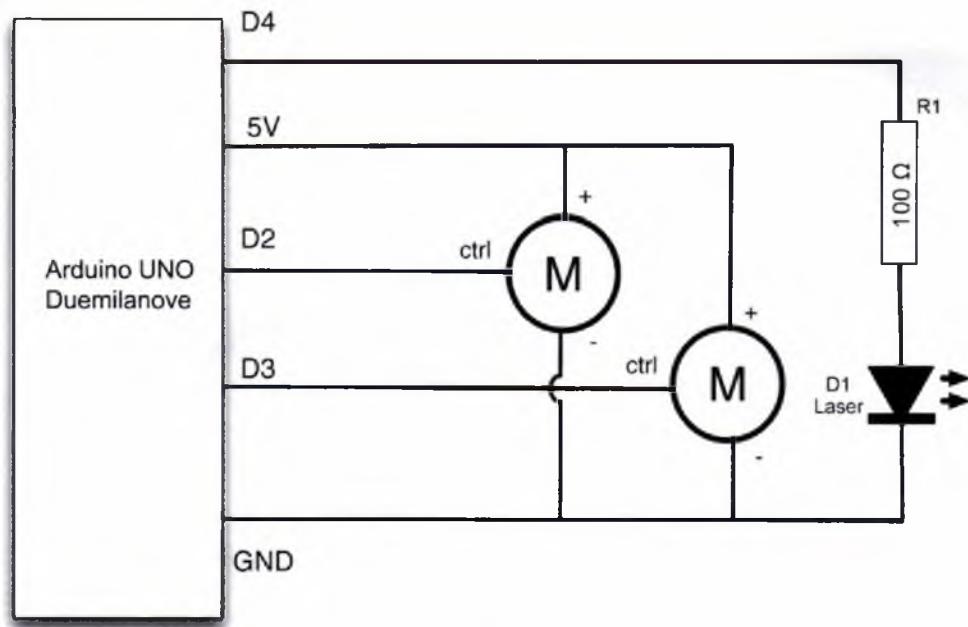


Figura 8-14 Esquema electrónico del Proyecto 25.

Software

Por suerte para nosotros, la biblioteca Arduino viene acompañada también de una **biblioteca servo**, por lo que todo lo que tenemos que hacer es decirle a cada servo a qué ángulo debe fijarse. Pero

está claro que no se trata sólo de esto; también queremos disponer de la forma de indicar a nuestro proyecto las coordenadas a las que dirigir el láser.

Para ello hacemos que se puedan enviar comandos a través del USB. Los comandos se envían en forma de letras. **R**, **L**, **U** y **D** dirigen el láser **cinco grados** hacia la derecha, izquierda, arriba o abajo, respectivamente. Para movimientos más finos, **r**, **l**, **u**, y **d** desplazan el láser sólo **un grado**. Para hacer una pausa y permitir que el láser deje de moverse, puede enviar el carácter **-** (guión). (Véase el Listado del Proyecto 25.)

Hay otros tres comandos. La letra **c** centrará el láser de nuevo en su posición de reposo, y los comandos **I** y **0** encienden y apagan el láser, respectivamente.

Pongamos todo junto

Cargue el **sketch** terminado del Proyecto 25 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).



Figura 8-15 Montaje del servo y del láser.

LISTADO DE PROYECTO 25

```
#include <Servo.h>

int laserPin = 4;
Servo servoV;
Servo servoH;

int x = 90;
int y = 90;
int minX = 10;
int maxX = 170;
int minY = 50;
int maxY = 130;

void setup()
{
    servoH.attach(3);
    servoV.attach(2);
    pinMode(laserPin, OUTPUT);
    Serial.begin(9600);
}

void loop()
{
    char ch;
    if (Serial.available())
    {
        ch = Serial.read();
        if (ch == '0')
        {
            digitalWrite(laserPin, LOW);
        }
        else if (ch == '1')
        {
            digitalWrite(laserPin, HIGH);
        }
        else if (ch == '-')
        {
            delay(100);
        }
        else if (ch == 'c')
        {
            x = 90;
            y = 90;
        }
        else if (ch == 'l' || ch == 'r' || ch == 'u' || ch == 'd')
        {

```

(continúa)

LISTADO DE PROYECTO 25 (continúa)

```

        moveLaser(ch, 1);
    }
    else if (ch == 'L' || ch == 'R' || ch == 'U' || ch == 'D')
    {
        moveLaser(ch, 5);
    }
}
servoH.write(x);
servoV.write(y);
}

void moveLaser(char dir, int amount)
{
    if ((dir == 'r' || dir == 'R') && x > minX)
    {
        x = x - amount;
    }
    else if ((dir == 'l' || dir == 'L') && x < maxX)
    {
        x = x + amount;
    }
    else if ((dir == 'u' || dir == 'U') && y < maxY)
    {
        y = y + amount;
    }
    else if ((dir == 'd' || dir == 'D') && y > minY)
    {
        y = y - amount;
    }
}
}

```

Abra el **Serial Monitor** e introduzca la siguiente secuencia. Debería ver que el láser traza la letra A, como se muestra en la Figura 8-16:

1UUUUUU—RRRR—DDDDDD—0UUU—1LLLL—0DDD

Montaje de una placa shield

Montar una **shield** para este proyecto no representa ningún problema. El **servo** inferior se puede pegar en su sitio en uno de los bordes de la placa. Los pines de la placa se sueldan en su sitio cerca de las líneas de **5 V** y **GND** que recorren el centro de la **shield** para que se puedan conectar fácilmente a los pines positivo y negativo de los conectores del **servo**.

Los lados superior e inferior de la placa shield se muestran en las Figuras 8-17 y 8-18.

Resumen

En los capítulos anteriores hemos ido aumentando nuestros conocimientos para entender cómo usar luz, sonido y diversos sensores en la placa Arduino. También hemos aprendido cómo controlar la potencia de motores y a utilizar relés. Esto debe cubrir casi todo lo que es probable que queramos hacer con nuestra placa Arduino, por lo que, en el próximo capítulo, podemos poner todas estas cosas juntas para crear algunos proyectos de mayor envergadura.

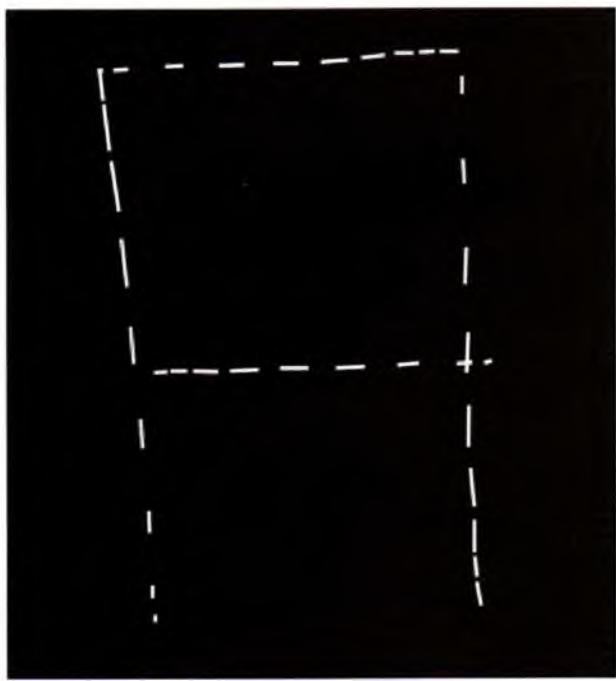


Figura 8-16 Escritura de la letra A con el láser.

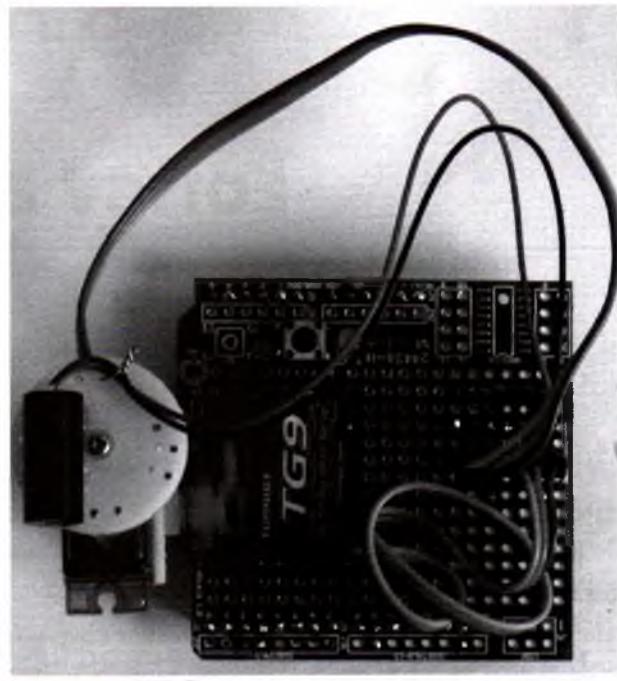


Figura 8-17 Shield servo láser.

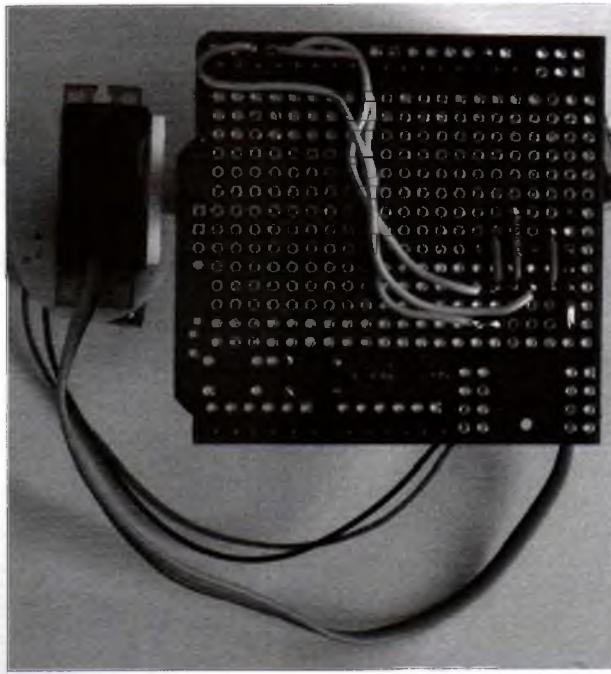


Figura 8-18 Lado inferior de la shield servo láser.

CAPÍTULO 9

Proyectos varios

ESTE CAPÍTULO ES una colección de proyectos que podemos construir. No se ilustra ningún aspecto determinado salvo resaltar que con la placa Arduino se pueden llegar a hacer grandes y divertidos proyectos.

Proyecto 26 Detector de mentiras

¿Cómo podría asegurarse de que sus invitados le dicen la verdad? Utilizando un detector de mentiras, por supuesto. Este detector de mentiras (véase la Figura 9-1) utiliza un efecto que se conoce como **respuesta galvánica de la piel**. Cuando una persona se pone nerviosa -por ejemplo, cuando dice una mentira- disminuye su resistencia cutánea. Utilizando una

entrada analógica podemos medir esta resistencia cutánea y utilizar un LED y un zumbador para indicar una falsedad.

Utilizaremos un LED multicolor, que mostrará rojo para indicar una mentira, verde para indicar una verdad y azul para mostrar que el detector de mentiras debe ajustarse, lo que haremos mediante un **potenciómetro**.

Hay dos tipos de **zumbadores piezoelectríficos**, los que solo tienen un **disco** o **transductor piezoelectrónico** y los que también incluyen el **oscilador electrónico** para manejarlos. En este proyecto usaremos el primer tipo, el que no lleva electrónica, ya que desde la propia placa Arduino vamos a generar la frecuencia necesaria.

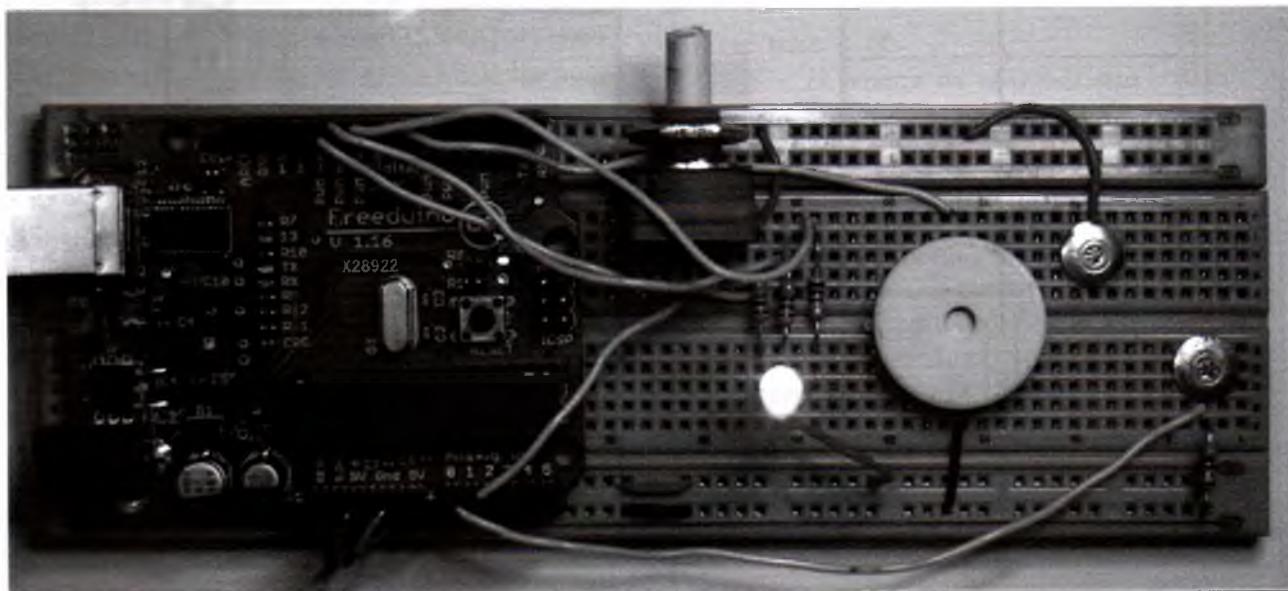


Figura 9-1 Proyecto 26. Detector de mentiras.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o clon	1
R1-3 Resistencia 100 Ω, 0,5 W	5
R4 Resistencia 470 Ω, 0,5 W	14
R5 Potenciómetro lineal 100 KΩ	17
D1 LED tricolor rojo-verde-azul (ánodo común)	31
S1 Disco piezoelectrónico (sin circuito oscilador)	67

Hardware

La resistencia cutánea de una persona se puede medir utilizando la resistencia de su piel como una de las resistencias de un divisor de tensión y una resistencia fija como la otra. Cuanto menor sea su resistencia, más llevaremos la **entrada analógica 0** hacia los

5 V; y cuanto mayor sea, más se acercará a **GND** o **cero voltios**.

El **piezozumbador**, a pesar del alto nivel de sonido que es capaz de generar, tiene realmente un consumo de corriente increíblemente bajo y se puede excitar directamente desde un pin digital de la Arduino.

Este proyecto utiliza el mismo LED multicolor que usamos en el Proyecto 14. En este caso, sin embargo, no vamos a mezclar colores diferentes sino que, simplemente, vamos a encender los LEDs de uno en uno para mostrar rojo, verde o azul.

La Figura 9-2 muestra el esquema electrónico del proyecto y la Figura 9-3 el diseño del circuito sobre la placa de pruebas.

El potenciómetro se utiliza para ajustar el punto de consigna de la resistencia, y los terminales de medición los hemos construido con dos chinchetas metálicas que se han colocado en la placa de pruebas.

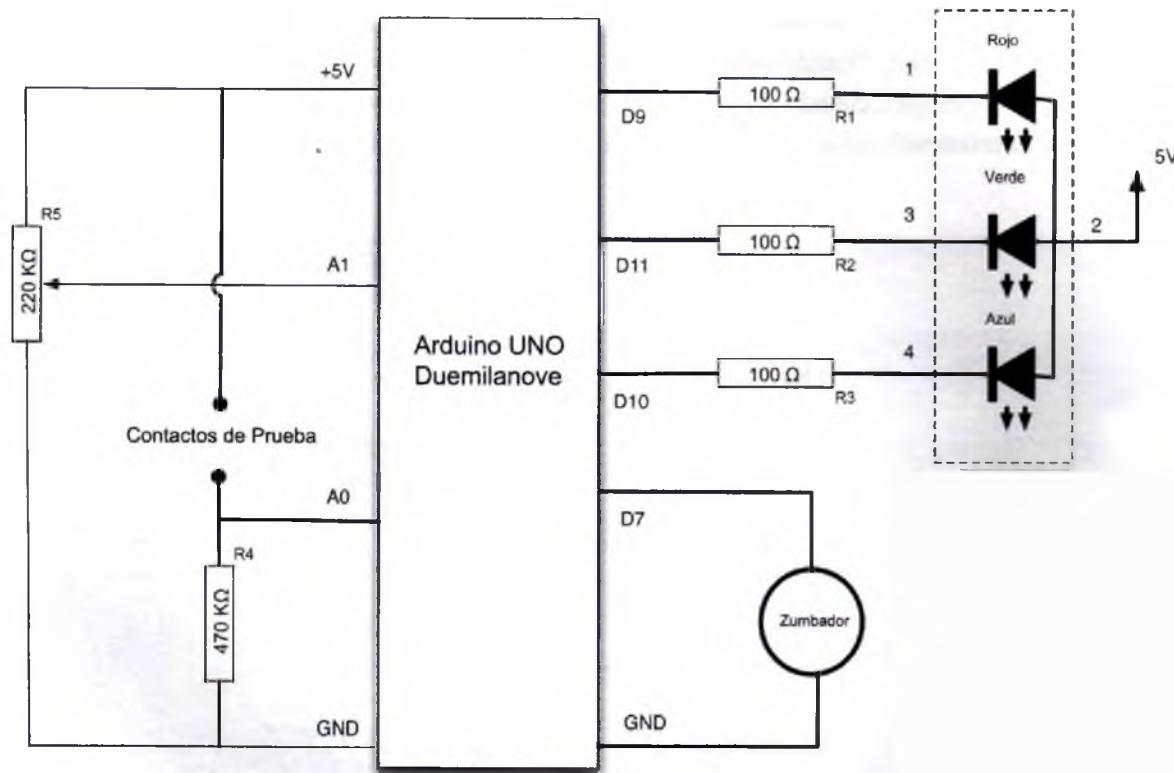


Figura 9-2 Esquema electrónico del Proyecto 26.

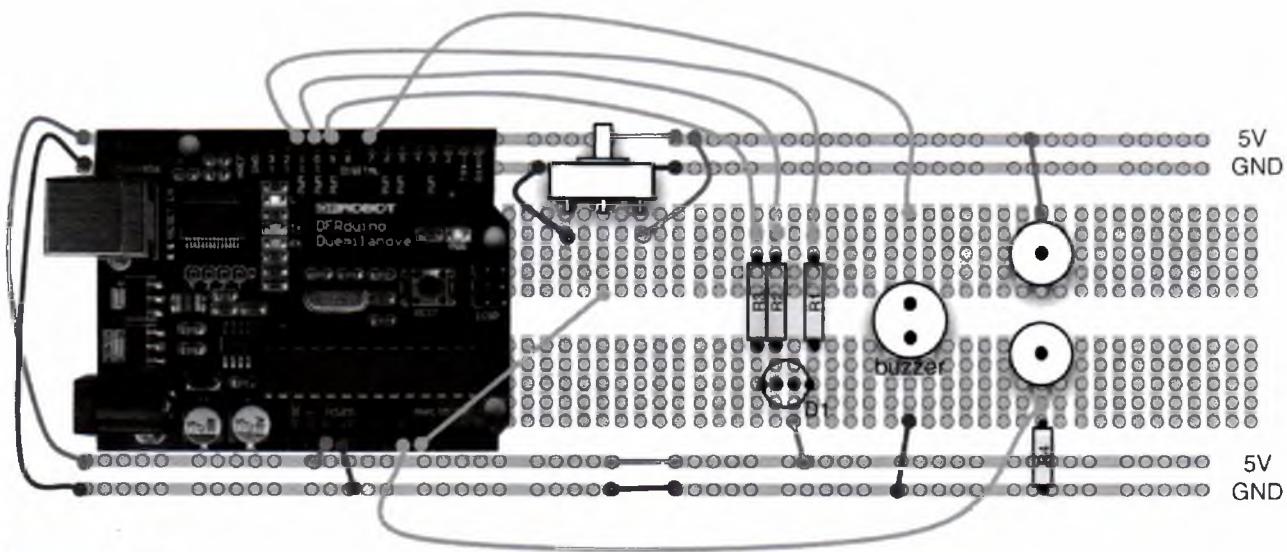


Figura 9-3 Diseño del circuito del Proyecto 26 sobre la placa de pruebas.

Software

El programa de este proyecto (Listado del Proyecto 26) sólo tiene que comparar el voltaje en **A0** y **A1**. Si son aproximadamente iguales, el LED será de color verde. Si el voltaje del terminal de medición (**A0**) es significativamente superior al de **A1**, el potenciómetro indicará un descenso en la resistencia cutánea, el LED cambiará a rojo y sonará el zumbador. Por otro lado, si **A0** es significativamente inferior a **A1**, el LED se tornará de color azul, lo que indica un aumento de la resistencia cutánea.

El zumbador requiere una frecuencia de alrededor de 5 KHz o 5000 hercios por segundo para excitarlo. Esto lo hacemos con un simple bucle **for loop** con comandos para encender y apagar el pin adecuado con retardo entre ellos.

Pongamos todo junto

Cargue el esquema terminado del Proyecto 26 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

LISTADO DE PROYECTO 26

```

int redPin = 9;
int greenPin = 10;
int bluePin = 11;
int buzzerPin = 7;

int potPin = 1;
int sensorPin = 0;

long red = 0xFF0000;
long green = 0x00FF00;
long blue = 0x000080;

int band = 10;
    // ajuste de sensibilidad

void setup()
{
    pinMode(potPin, INPUT);
    pinMode(sensorPin, INPUT);
    pinMode(redPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    pinMode(bluePin, OUTPUT);
    pinMode(buzzerPin, OUTPUT);
}

void loop()
{

```

(continúa)

LISTADO DE PROYECTO 26 (continúa)

```

int gsr = analogRead(sensorPin);
int pot = analogRead(potPin);
if (gsr > pot + band)
{
    setColor(red);
    beep();
}
else if (gsr < pot - band)
{
    setColor(blue);
}
else
{
    setColor(green);
}
}

void setColor(long rgb)
{
    int red = rgb >> 16;
    int green = (rgb >> 8) & 0xFF;
    int blue = rgb & 0xFF;
    analogWrite(redPin, 255 - red);
    analogWrite(greenPin, 255 - green);
    analogWrite(bluePin, 255 - blue);
}

void beep()
{
    // 5 KHz para 1/5th de segundo
    for (int i = 0; i < 1000; i++)
    {
        digitalWrite(buzzerPin, HIGH);
        delayMicroseconds(100);
        digitalWrite(buzzerPin, LOW);
        delayMicroseconds(100);
    }
}

```

Para probar el detector de mentiras, necesita poder probarlo con alguien, ya que necesitará una mano libre para ajustar el mando.

Primero, haga que la persona coloque dos dedos adyacentes encima de las chinchetas metálicas. A continuación, gire el mando del potenciómetro hasta que el LED se vuelva verde.

Ahora puede interrogar a su víctima. Si el LED cambia a rojo o azul, debe ajustar de nuevo el mando hasta que cambie de nuevo a verde y luego continuar el interrogatorio.

Proyecto 27

Cerradura magnética

Este proyecto (Figura 9-4) se basa en el Proyecto 10, pero lo amplía para que cuando se introduzca el código correcto, encienda un LED verde, además de activar un pequeño **solenoide**. El **sketch** también se ha mejorado para que el código secreto pueda ser cambiado sin tener que modificar e instalar una nueva secuencia de comandos. El código secreto se almacena en la memoria **EEPROM**, así que si se desconecta de la alimentación, éste no se perderá.

COMPONENTES Y EQUIPO

	Descripción	Apéndice
	Placa Arduino UNO o Duemilanove o clon	1
D1	LED rojo 5 mm	23
D2	LED verde 5 mm	25
R1-3	Resistencia 270 Ω, 0,5 W	6
K1	Teclado matricial 4 x 3	54
	Tira pines de 2,5 mm de paso	55
T1	BC548	40
	Solenóide 5V (< 100 mA)	66
D3	1N4004	38

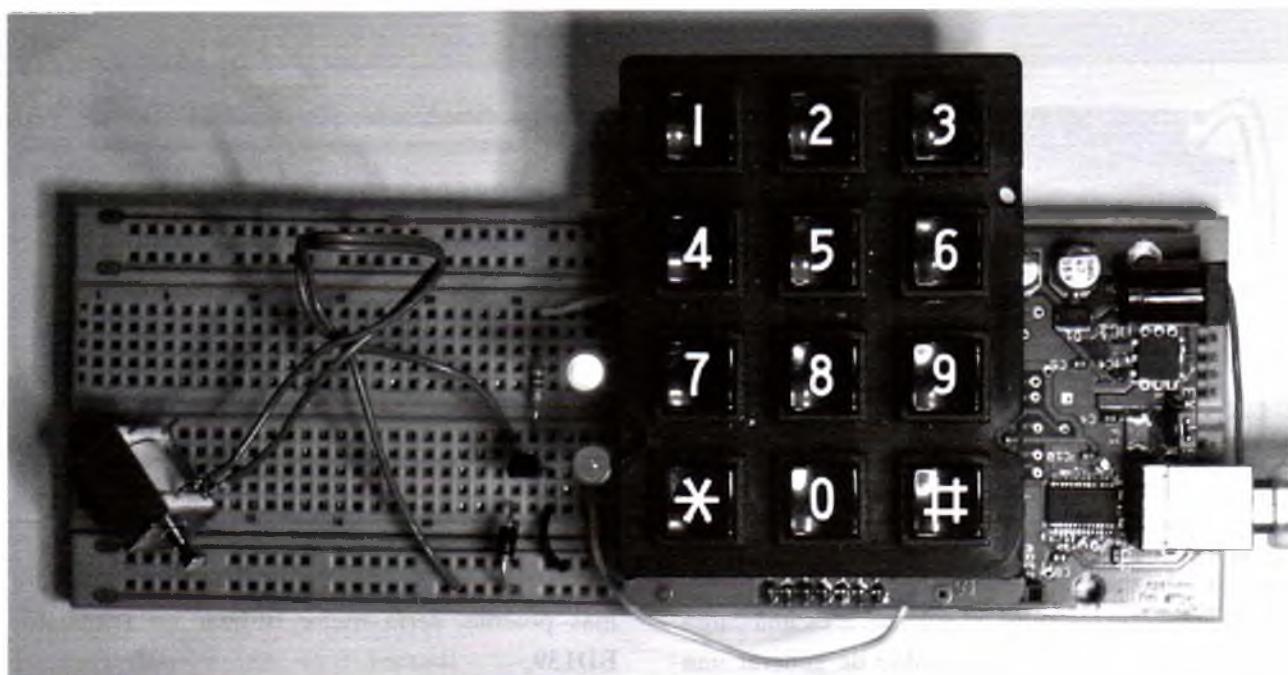


Figura 9-4 Proyecto 27. Cerradura magnética.

Cuando esté alimentado, el **solenoide** atraerá fuertemente la pieza metálica del centro, colocándola en su sitio. Cuando se quita la alimentación, ésta vuelve a su sitio libremente.

Hardware

El esquema electrónico (véase la Figura 9-5) y el diseño de la placa de pruebas (véase la Figura 9-6) son prácticamente los mismos que los del Proyecto 10, pero con algunos componentes adicionales.

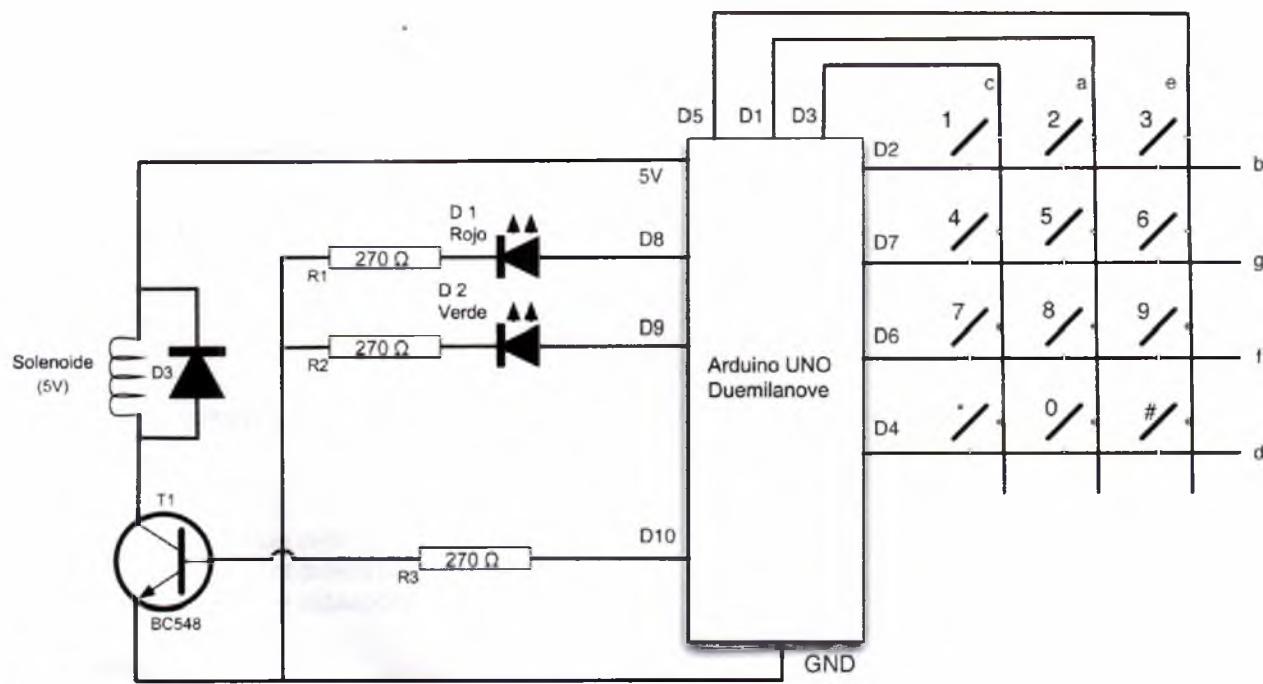


Figura 9-5 Esquema electrónico del Proyecto 27.

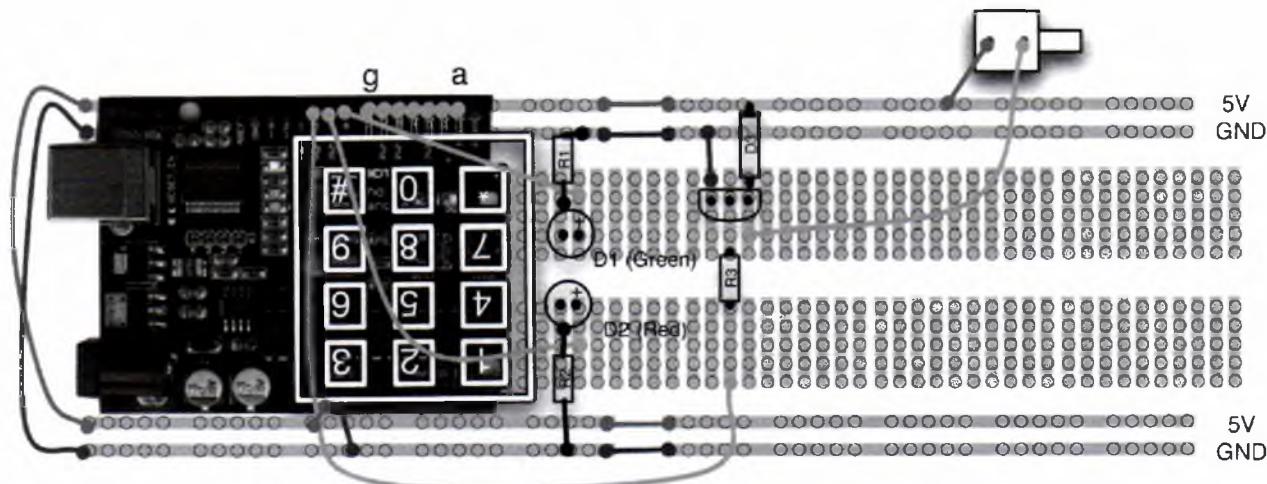


Figura 9-6 Diseño del circuito del Proyecto 27 sobre la placa de pruebas.

Al igual que los **relés**, el **solenoide** es una carga inductiva y, por tanto, susceptible de generar una **fuerza contraelectromotriz**, que es contra lo que protege el diodo **D3**. El **solenoide** es controlado por **T1**, así que asegúrese de seleccionar uno que no consuma más de 100 mA, que es la corriente máxima del colector del transistor.

Estamos utilizando un **solenoide** de muy baja potencia, y esto no sería suficiente para mantener a los intrusos fuera de nuestro cuarto. Si va a utilizar uno

más potente, sería mejor utilizar un **transistor BD139**.

Sería aconsejable que el **solenoide** se pudiese conectar directamente a la placa de pruebas; si no es así, tendrá que añadirle unos cables para conectarlo.

Software

El software de este proyecto es, como era de esperar, similar al del Proyecto 10 (véase el Listado del Proyecto 27).

LISTADO DEL PROYECTO 27

```
#include <Keypad.h>
#include <EEPROM.h>

char* secretCode = "1234";
int position = 0;
boolean locked = true;

const byte rows = 4;
const byte cols = 3;
char keys[rows][cols] = {
    {'1','2','3'},
    {'4','5','6'},
    {'7','8','9'},
    {'*','0','#'}
};
byte rowPins[rows] = {2, 7, 6, 4};
byte colPins[cols] = {3, 1, 5};
```

(continúa)

LISTADO DE PROYECTO 27 (continúa)

```
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, rows, cols);

int redPin = 9;
int greenPin = 8;
int solenoidPin = 10;

void setup()
{
    pinMode(redPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    loadCode();
    flash();
    updateOutputs();
}

void loop()
{
    char key = keypad.getKey();
    if (key == '*' && ! locked)
    {
        // desbloqueado y pulsado *, así que cambia el código
        position = 0;
        getNewCode();
        updateOutputs();
    }
    if (key == '#')
    {
        locked = true;
        position = 0;
        updateOutputs();
    }
    if (key == secretCode[position])
    {
        position++;
    }
    if (position == 4)
    {
        locked = false;
        updateOutputs();
    }
    delay(100);
}

void updateOutputs()
{
    if (locked)
    {
        digitalWrite(redPin, HIGH);
        digitalWrite(greenPin, LOW);
        digitalWrite(solenoidPin, HIGH);
    }
    else
```

(continúa)

LISTADO DE PROYECTO 27 (continúa)

```
{  
    digitalWrite(redPin, LOW);  
    digitalWrite(greenPin, HIGH);  
    digitalWrite(solenoidPin, LOW);  
}  
}  
  
void getNewCode()  
{  
    flash();  
    for (int i = 0; i < 4; i++)  
    {  
        char key;  
        key = keypad.getKey();  
        while (key == 0)  
        {  
            key = keypad.getKey();  
        }  
        flash();  
        secretCode[i] = key;  
    }  
    saveCode();  
    flash();flash();  
}  
  
void loadCode()  
{  
    if (EEPROM.read(0) == 1)  
    {  
        secretCode[0] = EEPROM.read(1);  
        secretCode[1] = EEPROM.read(2);  
        secretCode[2] = EEPROM.read(3);  
        secretCode[3] = EEPROM.read(4);  
    }  
}  
  
void saveCode()  
{  
    EEPROM.write(1, secretCode[0]);  
    EEPROM.write(2, secretCode[1]);  
    EEPROM.write(3, secretCode[2]);  
    EEPROM.write(4, secretCode[3]);  
    EEPROM.write(0, 1);  
}  
  
void flash()  
{  
    digitalWrite(redPin, HIGH);  
    digitalWrite(greenPin, HIGH);  
    delay(500);  
    digitalWrite(redPin, LOW);  
    digitalWrite(greenPin, LOW);  
}
```

Como ahora podemos cambiar el código secreto, hemos cambiado la función **loop** para que si se pulsa la tecla * mientras la cerradura está desbloqueada, las siguientes cuatro teclas pulsadas se conviertan en el nuevo código.

Puesto que cada carácter tiene exactamente un **byte** de longitud, el código se puede almacenar directamente en la memoria **EEPROM**. Utilizamos el primer **byte** de la **EEPROM** para indicar si se ha fijado el código. Si no se ha definido, el código por defecto será 1234. Una vez que se ha configurado el código, al primer **byte** de la **EEPROM** se le dará el valor de 1.

Pongamos todo junto

Cargue el esquema terminado del Proyecto 27 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Podemos comprobar que todo funciona conectando la alimentación e introduciendo el código 1234, en cuyo momento el LED verde debe encenderse y liberarse el **solenoide**. Podemos cambiar el código a uno no tan predecible pulsando la tecla * y, a continuación, introduciendo cuatro dígitos para el nuevo código. La cerradura se mantendrá abierta hasta que pulsemos la tecla #.

Lamentablemente, si olvida su código secreto, encender y apagar la alimentación del proyecto no lo reiniciará a 1234. En su lugar, tendrá que convertir en comentario (//) la línea:

```
loadCode();
```

de la función **setup**, para que aparezca como se muestra aquí:

```
// loadCode();
```

Ahora, reinstale el **sketch** y el código secreto volverá a ser 1234. Recuerde cambiar de nuevo su después de ajustar el código a algo que pueda recordar.

Proyecto 28 Mando a distancia por infrarrojos

Este proyecto (véase la Figura 9-7) permite controlar directamente desde el ordenador cualquier aparato del hogar que se maneje con un mando a distancia por infrarrojos. Con él, se puede grabar un mensaje de infrarrojos desde algún mando a distancia existente y luego reproducirlo desde el ordenador.

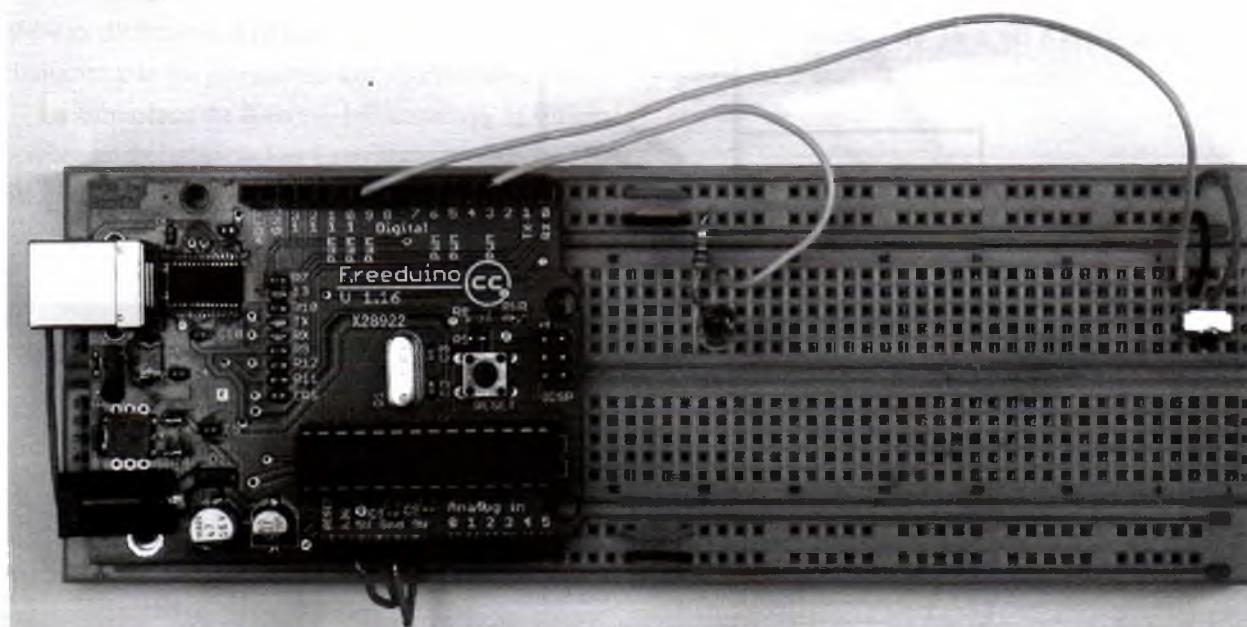


Figura 9-7 Proyecto 28. Mando a distancia por infrarrojos.

Nota del revisor: Puede apreciarse que no hay ninguna contradicción con el esquema de la Figura 9.8, pues aunque en éste la resistencia R1 va conectada al pin3 y aquí se ve conectada a masa, al estar en serie R1 y el LED, da igual en qué orden se conecten, siempre que la pata más larga del LED (ánodo o +) quede conectada al positivo (D3).

Utilizaremos la memoria **EEPROM** para almacenar los códigos de control remoto para que no se pierdan cuando se desconecte la placa Arduino.

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o clon	1
R1 Resistencia 100 Ω, 0,5 W	5
D1 Diodo emisor Infrarrojo 940 nm	26
IC1 Receptor de Infrarrojo de 940 nm	37

Hardware

El receptor de mando a distancia por infrarrojos (**IR**) es un pequeño módulo que combina un **fotodiodo infrarrojo** con el filtrado, acondicionado y amplificación necesarios para producir una salida digital a partir del mensaje de **IR**. Esta salida se pasa al **pin digital 9**. El esquema electrónico (véase la Figura 9-8) muestra lo fácil que es utilizar este chip, con sólo tres pines, **GND**, **+V** y la **señal de salida**.

El elemento transmisor de **IR** es un LED infrarrojo. Estos funcionan igual que un LED normal, pero en el extremo infrarrojo invisible del espectro. En algunos dispositivos se puede ver un ligero brillo rojo cuando se encienden.

La Figura 9-9 muestra el diseño del circuito del proyecto sobre la placa de pruebas.

Software

Ken Shirriff ha creado una biblioteca que puede utilizar para hacer casi cualquier cosa que desee hacer con un **IR** remoto. Nosotros, en lugar de reinventar la rueda, también vamos a usar esta biblioteca.

En el Capítulo 5 examinamos por primera vez cómo instalar una biblioteca. Para hacer uso de esta biblioteca, primero tenemos que descargarla. Para las placas Arduino UNO, extraiga la biblioteca modificada de:

<http://www.arduinoevilgenius.com/new-downloads>.

En el caso de las placas anteriores, puede hacerlo desde <http://arcfn.com/2009/08/multi-protocol-infrared-remote-library.html>.

Descargue el archivo **IRRemote.zip** y descomprímalo. Si está utilizando Windows, haga clic con el

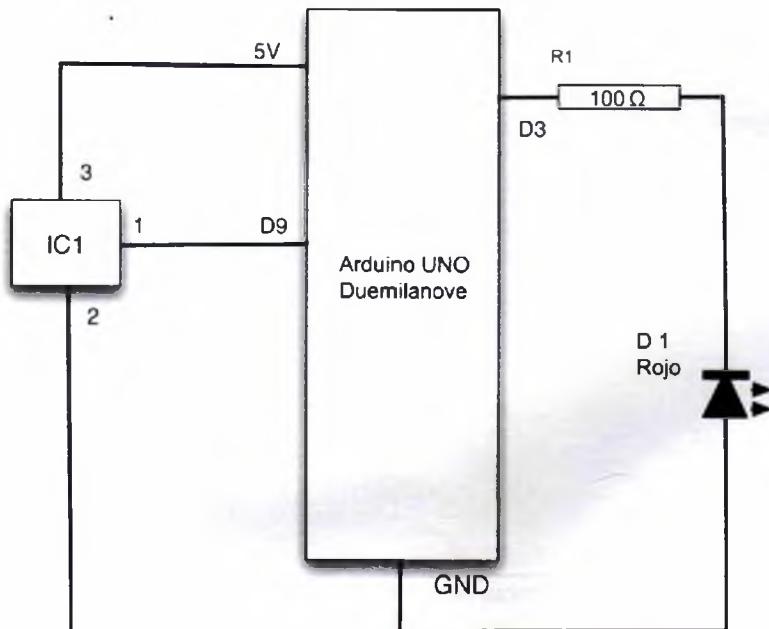


Figura 9-8 Esquema electrónico del Proyecto 28.

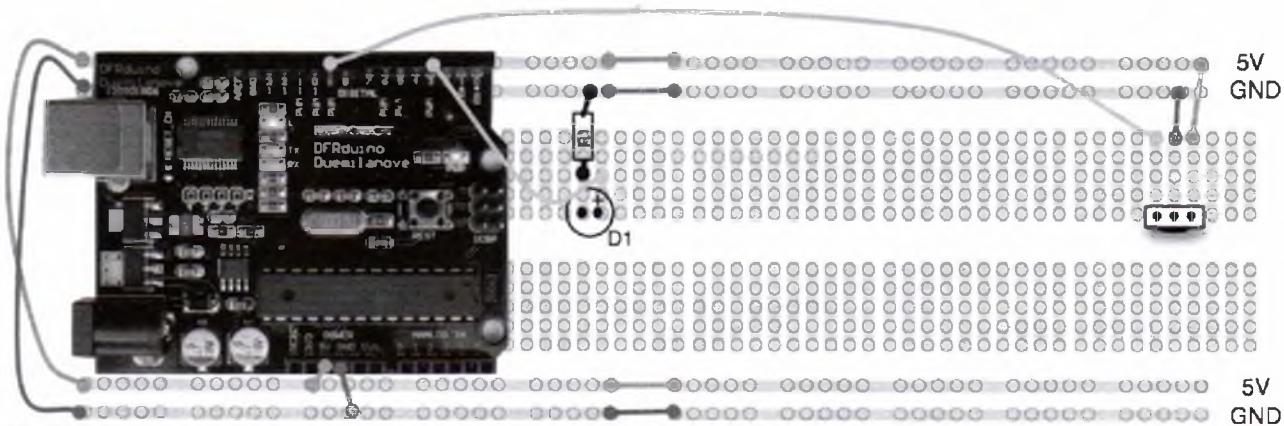


Figura 9-9 Diseño del circuito del Proyecto 28 sobre la placa de pruebas.

botón secundario y seleccione **Extraer todo** y, a continuación, guarde el archivo en **C:\Archivos de programa\Arduino\Arduino-0017\hardware\libraries**. En Linux, busque el directorio de instalación de Arduino y copie la carpeta en **hardware/libraries**.

En un Mac, no ponga la nueva biblioteca en la instalación de Arduino. En su lugar, cree una carpeta llamada **libraries** en **Documentos/Arduino** (Figura 5-8) y coloque allí la carpeta **libraries** completa.

Una vez que hayamos instalado esta biblioteca en nuestro directorio **Arduino**, podremos usarla con cualquiera de los programas que escribamos.

La biblioteca de **Ken** es, básicamente, la última palabra en **decodificación y envío de comandos IR**. Su intención es corresponder a las distintas normas

de protocolo de los distintos fabricantes. Nuestro **sketch** utiliza solamente una pequeña parte de la biblioteca, referida a la captura y envío de impulsos de datos sin tratar. (Consulte el Listado del Proyecto 28.)

Los mandos a distancia por infrarrojos envían una serie de impulsos a una **frecuencia** de entre 36 y 40 KHz. La Figura 9-10 muestra el trazado de los mismos en el osciloscopio.

Un valor de bit de **1** se representa por un impulso de onda cuadrada de 36 a 40 KHz y un **0** por una pausa en la que no se envía nada.

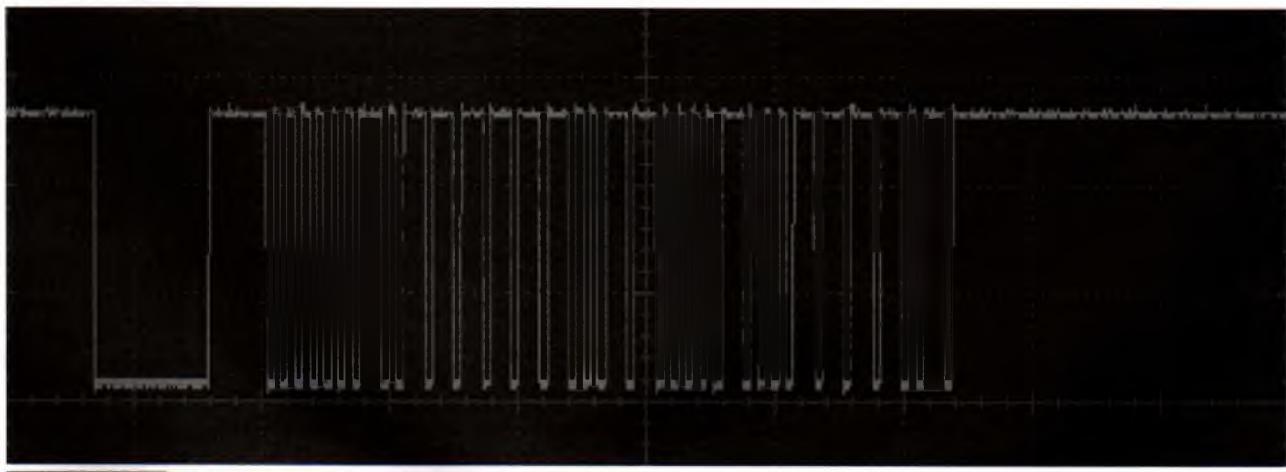


Figura 9-10 Código de infrarrojos en el osciloscopio.

LISTADO DE PROYECTO 28

```
#include <EEPROM.h>
#include <IRremote.h>

int irRxPin = 9;

int f = 38; // 40, 36, 38

IRrecv irrecv(irRxPin);
IRsend irsend;

decode_results results;
int codeLength = 0;
int currentCode = 0;

void setup()
{
    Serial.begin(9600);
    Serial.println("0-9 para configurar código memoria, s - para enviar");
    irrecv.enableIRIn();
    setCodeMemory(0);
}

void loop()
{
    if (Serial.available())
    {
        char ch = Serial.read();
        if (ch >= '0' && ch <= '9')
        {
            setCodeMemory(ch - '0');
        }
        else if (ch == 's')
        {
            sendIR();
        }
    }
    if (irrecv.decode(&results))
    {
        storeCode();
        irrecv.resume();
    }
}

void setCodeMemory(int x)
{
    currentCode = x;
    Serial.print("Establecer código memoria actual a: ");
}
```

(continúa)

LISTADO DE PROYECTO 28 (continúa)

```
Serial.println(currentCode);
irrecv.resume();
}

void storeCode()
{
    // escribe código en EEPROM, primer byte es la longitud
    int startIndex = currentCode * (RAWBUF + 1);
    int len = results.rawlen - 1;
    EEPROM.write(startIndex, (unsigned byte)len);
    for (int i = 0; i < len; i++)
    {
        if (results.rawbuf[i] > 255)
        {
            EEPROM.write(startIndex + i + 1, 255);
        }
        else
        {
            EEPROM.write(startIndex + i + 1, results.rawbuf[i]);
        }
    }
    Serial.print("Saved code, length: ");
    Serial.println(len);
}

void sendIR()
{
    // construir memoria temporal datos guardados en EEPROM y enviarlos
    int startIndex = currentCode * (RAWBUF + 1);
    int len = EEPROM.read(startIndex);
    unsigned int code[RAWBUF];
    for (int i = 0; i < len; i++)
    {
        int pulseLen = EEPROM.read(startIndex + i + 2);
        if (i % 2)
        {
            code[i] = pulseLen * USECPERTICK + MARK_EXCESS;
        }
        else
        {
            code[i] = pulseLen * USECPERTICK - MARK_EXCESS;
        }
    }
    irsend.sendRaw(code, len, f);
    Serial.print("Longitud de código enviado: ");
    Serial.println(len);
}
```

La biblioteca **IRRemote** requiere que el LED **IR** se maneje desde el **pin digital 3**. Por lo tanto, solo especificamos el pin de recepción en la parte superior del **sketch** (**irRxPin**).

En la función **setup**, comenzamos las comunicaciones serie y escribimos las instrucciones para usar el proyecto de nuevo en la **Serial Console**, que es desde donde lo vamos a controlar. También configuraremos la memoria del código actual en la memoria **0**.

La función **loop** sigue el esquema habitual de verificar cualquier entrada a través del puerto USB. Si se trata de un dígito entre **0** y **9**, convierte la memoria correspondiente en la memoria actual. Si se recibe un carácter "s" de **Serial Monitor**, enviará el mensaje que haya en la memoria de mensajes actual.

La función comprueba entonces si se ha recibido alguna señal **IR**; si es así, la escribe en la **EEPROM** utilizando la función **storeCode**. Almacena la longitud del código en el primer byte y, a continuación, el **número de ticks** de 50 microsegundos por cada pulso en bytes que se produce posteriormente. **RAWBUF** es una constante definida en la biblioteca como la longitud máxima del mensaje.

Observe que, como parte del proceso de enviar el código de la función **sendIR**, se crea una matriz de números enteros que representan las duraciones de los pulsos a partir de los bytes almacenados; las duraciones de los pulsos están, por tanto, en microsegundos, en lugar de en **ticks**, y se ajustan mediante **MARK_EXCESS** para compensar el hardware que lee las señales **IR**. Esto tiende a hacer las marcas un poco más largas de lo que deberían ser y los espacios un poco más cortos.

Asimismo, en **storeCode** y en **sendIR**, cuando accedemos a la memoria **EEPROM**, utilizamos una interesante técnica que nos permite usarla como si fuera una matriz para las memorias de mensajes. El punto de inicio para grabar o leer los datos de la memoria EEPROM se calcula multiplicando **currentCode** por la duración de cada código (más el byte que dice la longitud que tiene).

Pongamos todo junto

Cargue el esquema terminado del Proyecto 28 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Para probar el proyecto, busque un mando a distancia y el equipo que se controla con dicho mando. A continuación, ponga en marcha el proyecto.

Abra **Serial Monitor**; debería recibir el siguiente mensaje de saludo:

```
0-9 para configurar código memoria, s -  
para enviar  
Establecer código memoria actual a: 0
```

De forma predeterminada, cualquier mensaje que capturemos será grabado en la memoria **0**. Así es que dirija el mando a distancia al sensor de nuestro montaje y pulse un botón (encender un reproductor de DVD o abrir la bandeja del disco pueden ser algunas de las impresionantes acciones que podría hacer). Debería ver un mensaje como éste:

```
Código guardado, longitud: 67
```

Ahora dirija el LED **IR** de nuestro montaje al DVD y teclee **s** en **Serial Monitor**. En la pantalla del ordenador debería recibir un mensaje como éste:

```
Longitud código enviado: 67
```

Y lo que es más importante aún ¡el DVD debe obedecer a la orden que le ha mandado la placa Arduino!

Tenga en cuenta que el LED infrarrojo de nuestra placa de pruebas puede que no tenga mucha potencia, por lo que si no funciona, pruebe a acercarlo más al sensor de infrarrojos del aparato a controlar.

Ahora puede cambiar la posición de memoria introduciendo otro dígito en **Serial Monitor** y grabando distintos comandos del **IR**. Y tenga en cuenta que no hay ninguna razón para que los comandos tengan que ser del mismo aparato.

Proyecto 29

Reloj Lilypad

La placa **Lilypad** de Arduino funciona de forma muy parecido a la placa **UNO** o a la **Dueamilanove**, pero en lugar de una aburrida placa rectangular, **Lilypad** es circular y está diseñada para que se cosa en la ropa usando hilo conductor. Cualquiera apreciará su belleza cuando la vea. Por eso este proyecto se ha integrado en un portafotos, para mostrar la belleza natural de la electrónica (véase la Figura 9-11). Para ajustar el tiempo se ha utilizado un **interruptor magnético de tipo Reed**.

Dado que la placa Lilypad no lleva conectores de entrada y salida, las conexiones a ésta hay que hacerla mediante un soldador.

COMPONENTES Y EQUIPO

Descripción	Apéndice
Placa Arduino Lilypad y programador USB	2
R1-16 Resistencia 100 Ω 0,5 W	5
D1-4 LED rojo 5 mm	27
D5-10 LED azul 5 mm	29
D11-16 LED verde 5 mm	28
R17 Resistencia 100 K Ω , 0,5 W	13
S1 Interruptor de contactos Reed	9
Marco portafotos 13 x 18 cm	70
Fuente alimentación 5 V	71

Hardware

En este proyecto contamos con un LED y una resistencia en serie conectada a casi todas las conexiones de la **Lilypad**.

El interruptor magnético **Reed** es un pequeño componente muy útil que consiste simplemente en un interruptor con dos contactos que se encuentran en una envoltura de vidrio sellada. Cuando se aproxima un imán al interruptor, los contactos se atraen entre sí, cerrando el mismo.

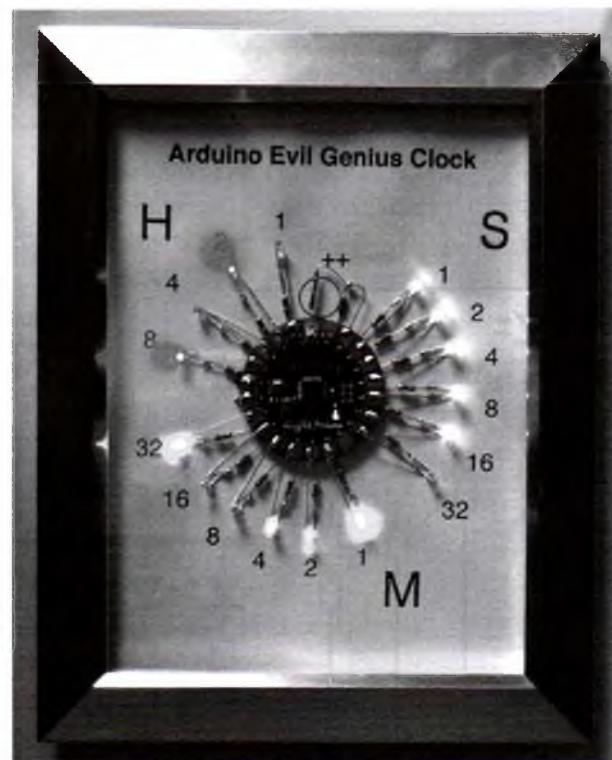


Figura 9-11 Proyecto 29. Reloj binario con Lilypad.

En nuestro caso, en lugar de un interruptor corriente, vamos a utilizar un **interruptor de contactos Reed** para que todo el proyecto se pueda montar en un marco de fotos, tras un cristal. La hora podremos ajustarla acercando un imán al interruptor.

La Figura 9-12 muestra el esquema electrónico del proyecto.

Cada LED tiene una resistencia soldada en la pata negativa, que es la más corta. El terminal positivo del LED se suelda luego al terminal de la Arduino Lilypad y el cable de la resistencia se pasa por debajo de la placa, donde se conecta al resto de los cables de las resistencias.

La Figura 9-13 muestra un primer plano del LED y de la resistencia, y el conexionado de los terminales bajo la placa se muestra en la Figura 9-14. Observe el disco de papel grueso que aísla la parte de atrás de la placa de los terminales de las resistencias soldadas.

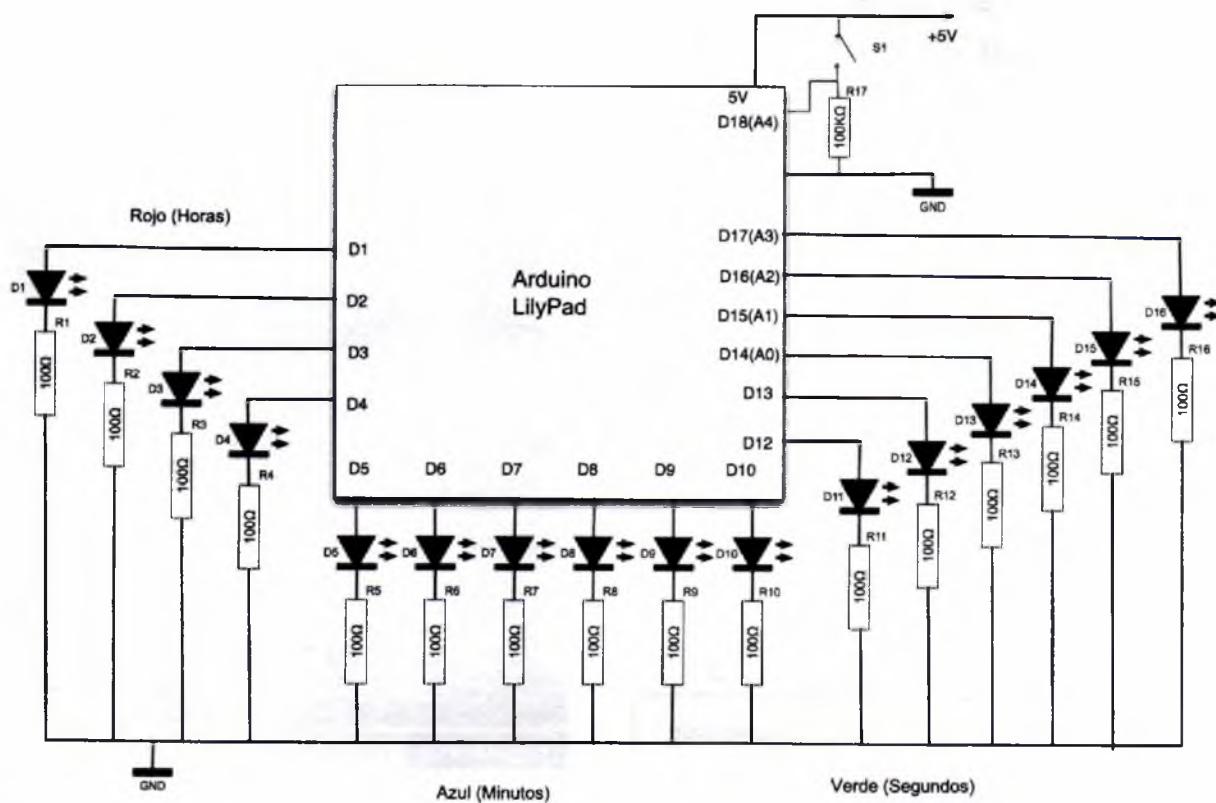


Figura 9-12 Esquema electrónico del Proyecto 29.

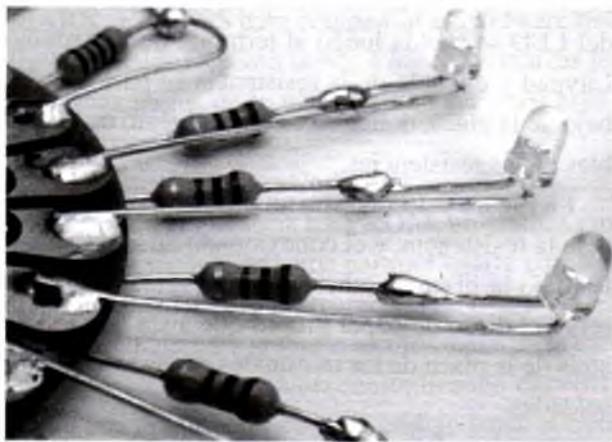


Figura 9-13 Primer plano de LED conectado a una resistencia.

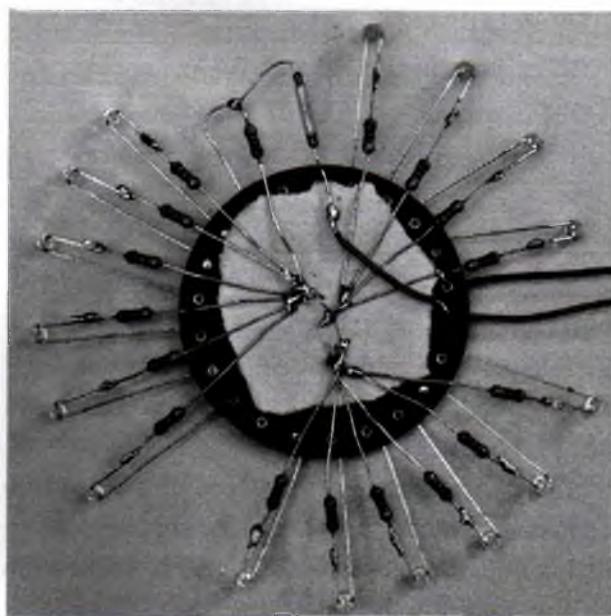


Figura 9-14 Lado inferior de la placa Lilypad.

En este montaje vamos a utilizar una **fuente de alimentación** de 5 V, ya que cuando todos los LEDs están encendidos se utiliza una cantidad significativa de electricidad y las pilas no durarían mucho tiempo. Los cables de alimentación salen por uno de los lados del portafotos, donde se sueldan a un conector.

Para el ejemplo que hemos presentado aquí, el autor utilizó como fuente de alimentación un cargador de teléfono móvil. Asegúrese de que cualquier fuente de alimentación que vaya a usar proporcione 5 V y, al menos, una corriente de 500 mA. Puede utilizar un polímetro para comprobar la polaridad de la fuente de alimentación.

Software

Este es otro proyecto en el cual hacemos uso de una biblioteca. Esta biblioteca facilita la manipulación de la hora y se puede descargar desde www.arduino.cc/playground/Code/Time.

Descargue el archivo **Time.zip** y descomprimalo. Si está utilizando Windows, haga clic con el botón secundario y seleccione **Extraer todo** y, a continuación, guarde el archivo en **C:\Archivos de programa\Arduino\Arduino-0017\hardware\libraries**.

En Linux, busque el directorio de instalación de Arduino y copie la carpeta en **hardware/libraries**.

En un Mac, no ponga la nueva biblioteca en la instalación de Arduino. En su lugar, cree una carpeta llamada **libraries** en **Documentos/Arduino** (Figura 5-8) y coloque allí la carpeta **libraries** completa.

Una vez que hayamos instalado esta biblioteca en nuestro directorio Arduino, podremos usarla con cualquiera de los programas que escribamos. (Consulte el Listado del Proyecto 29.)

Para hacer referencia a los diferentes conjuntos de LEDs vamos a utilizar matrices (**arrays**). Éstas se utilizan para simplificar la instalación y también en la función **setOutput**. Esta función establece los valores binarios de la matriz de LEDs, es decir, sirve para mostrar un valor binario. La función también recibe argumentos de la longitud de esa matriz y el valor que debe escribir en ella. Esto se utiliza en la función **loop** para establecer sucesivamente los LEDs para horas, minutos y segundos. Cuando se pasa una matriz a una función como ésta, debe prefijar el argumento en la definición de función con un *****.

LISTADO DE PROYECTO 29

```
#include <Time.h>

int hourLEDs[] = {1, 2, 3, 4};
// least significant bit first
int minuteLEDs[] = {10, 9, 8, 7, 6, 5};
int secondLEDs[] = {17, 16, 15, 14, 13,
12};

int loopLEDs[] = {17, 16, 15, 14, 13, 12,
10, 9, 8, 7, 6, 5, 4, 3, 2, 1};

int switchPin = 18;

void setup()
{
    for (int i = 0; i < 4; i++)
    {
        pinMode(hourLEDs[i], OUTPUT);
    }
    for (int i = 0; i < 6; i++)
    {
        pinMode(minuteLEDs[i], OUTPUT);
    }
    for (int i = 0; i < 6; i++)
    {
        pinMode(secondLEDs[i], OUTPUT);
    }
    setTime(0);
}

void loop()
{
    if (digitalRead(switchPin))
    {
        adjustTime(1);
    }
    else if (minute() == 0 && second() == 0)
    {
        spin(hour());
    }
    updateDisplay();
    delay(1);
}
```

(continúa)

LISTADO DE PROYECTO 29 (continúa)

```

void updateDisplay()
{
    time_t t = now();
    setOutput(hourLEDs, 4,
              hourFormat12(t));
    setOutput(minuteLEDs, 6, minute(t));
    setOutput(secondLEDs, 6, second(t));
}

void setOutput(int *ledArray, int num-
               LEDs, int value)
{
    for (int i = 0; i < numLEDs; i++)
    {
        digitalWrite(ledArray[i],
                    bitRead(value, i));
    }
}

void spin(int count)
{
    for (int i = 0; i < count; i++)
    {
        for (int j = 0; j < 16; j++)
        {
            digitalWrite(loopLEDs[j],
                        HIGH);
            delay(50);
            digitalWrite(loopLEDs[j],
                        LOW);
        }
    }
}

```

Una característica adicional de este reloj es que cada hora en punto gira los LED, iluminándolos todos de uno en uno. Así, a las 6 en punto, por ejemplo, girará seis veces antes de continuar funcionando normalmente.

Cuando se activa el contacto **Reed** se está llamando a la función **adjustTime** con un argumento de 1 segundo. Dado que la función se encuentra en la función **loop** con un milisecondo de retardo, los segundos van a pasar rápidamente.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 29 desde su **Arduino Sketchbook** y descárguelo en su placa. En una **Lilypad** esto se hace de una manera un poco diferente a lo que estamos acostumbrados. Antes de realizar la descarga, tendrá que seleccionar en el software Arduino un **tipo de placa** y un **puerto serie** diferentes.

Monte el proyecto, pero pruébelo conectado al programador USB antes de colocarlo en el marco de fotos.

Intente escoger un portafotos que tenga suficiente hueco para que los componentes puedan caber entre el cartón trasero y el cristal.

Si lo desea, puede diseñar una hoja de papel con etiquetas para los LED para hacer más fácil la lectura de la hora. En www.arduinoevilgenius.com puede encontrar un diseño adecuado.

Para leer la hora del reloj, se mira cada sección (horas, minutos y segundos) una tras otra y se agregan los valores próximos a los LED que se iluminan. Así, si se encienden los LED correspondientes a la hora próximos a 8 y 2, entonces la hora son las 10. A continuación, haga lo mismo con los minutos y segundos.

Proyecto 30

Temporizador de cuenta atrás

Ningún libro de proyectos que se precie debe terminar sin un completo **temporizador de cuenta atrás**, al más puro estilo Bond, con una maraña de cables de colores incluido (ver la Figura 9-15). Este temporizador también podríamos utilizarlo como **temporizador de cocina para huevos**, porque... ¡no hay nada más desagradable que un huevo pasado por agua demasiado cocido!

COMPONENTES Y EQUIPO	
Descripción	Apéndice
Placa Arduino UNO o Duemilanove o clon	1
D1-4 Display LED 7 segm 2 dig. (ánodo común)	33
R1-3 Resistencia 100 KΩ 0,5 W	13
R4-7 Resistencia 1 KΩ 0,5 W	7
R8-15 Resistencia 100 Ω 0,5 W	5
T1-4 BC307	39
Codificador giratorio	57
Zumbador Piezoelectrónico (con electrónica integrada)	68

Y para que la sonoridad sea la óptima, el **zumbador piezoelectrónico** que vamos a utilizar es del tipo de los que traen la electrónica integrada, con lo que todo lo que tendremos que hacer para que suene es alimentarlo con 5 V. En cualquier caso, tenga cuidado de conectarlo con la polaridad correcta.

Hardware

El proyecto es similar al Proyecto 15, aunque le hemos añadido el **display** de siete segmentos y los transistores asociados. También tenemos un **codificador giratorio** que utilizaremos para establecer la hora desde la que empezaremos la cuenta atrás. Ya hemos utilizado antes ambos componentes; para obtener más información sobre los **codificadores giratorios**, véase el Capítulo 6.

El esquema electrónico del proyecto se muestra en la Figura 9-16 y el diseño de la placa de pruebas, en la Figura 9-17.

Software

Fundamentalmente, el **sketch** de este proyecto (Listado del Proyecto 30) se ocupa de mantener el display actualizado y de crear la ilusión de que los cuatro displays se encienden al mismo tiempo cuando, en realidad, en cada momento sólo habrá encendido una de ellos. La forma de llevar esto a cabo ya lo describimos en el Proyecto 15.

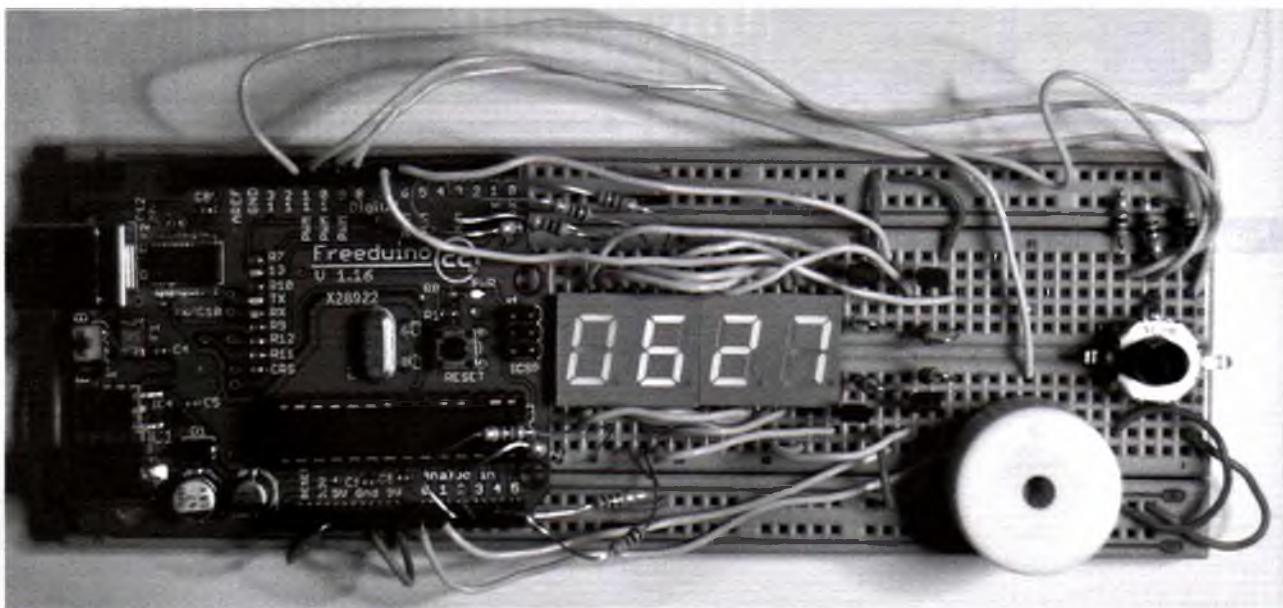


Figura 9-15 Proyecto 30. Temporizador de cuenta atrás.

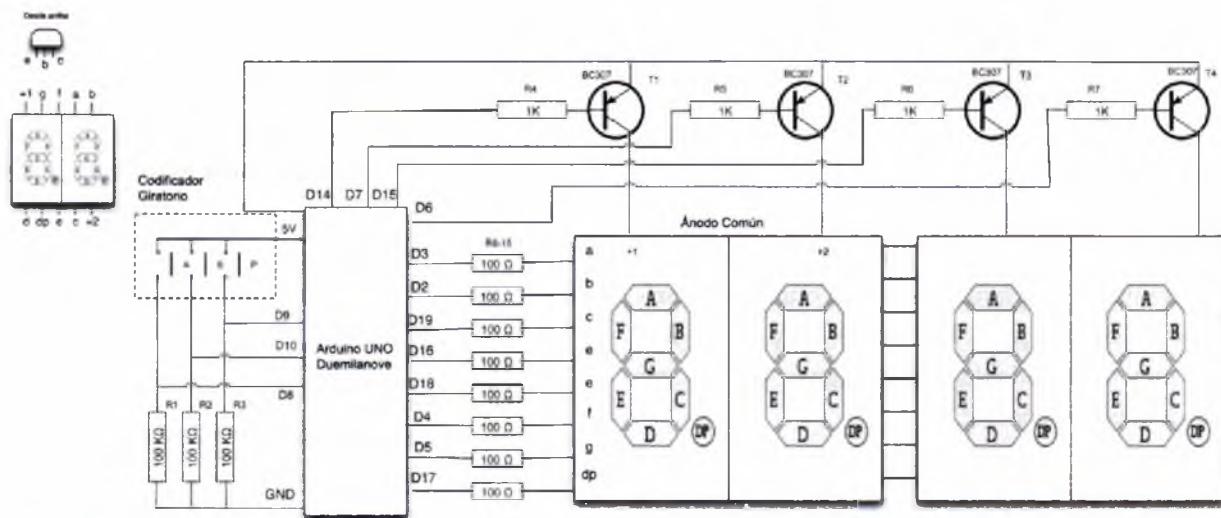


Figura 9-16 Esquema electrónico del Proyecto 30.

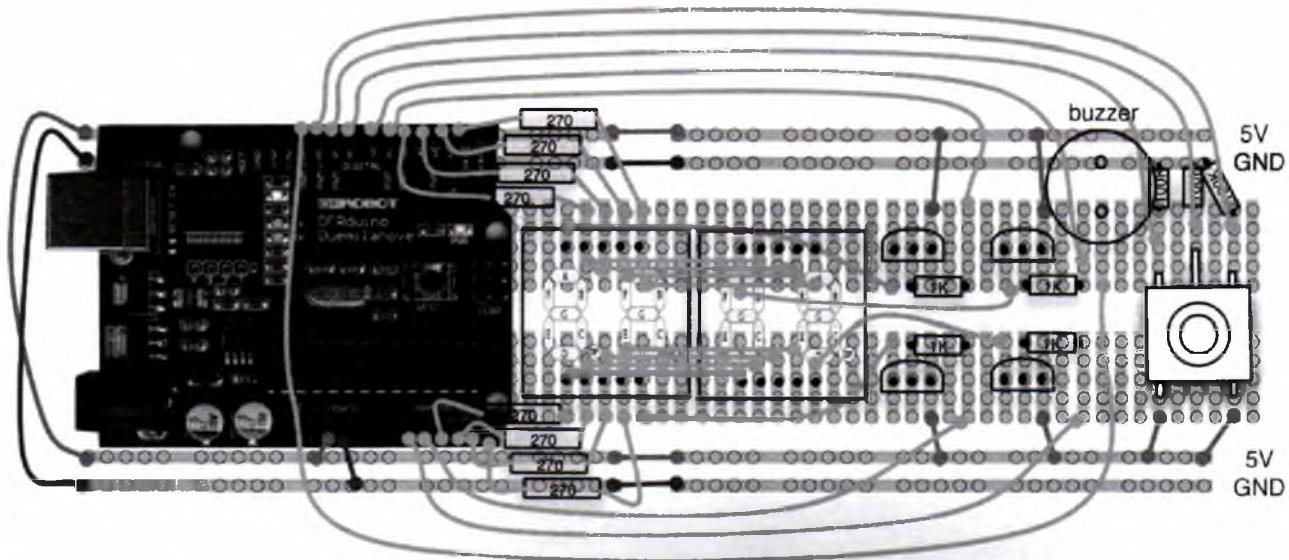


Figura 9-17 Placa Arduino alimentada con el LED encendido.

LISTADO DE PROYECTO 30

```
#include <EEPROM.h>

int segmentPins[] = {3, 2, 19, 16, 18, 4, 5, 17};
int displayPins[] = {14, 7, 15, 6};
int times[] = {5, 10, 15, 20, 30, 45, 100, 130, 200, 230, 300, 400, 500, 600, 700, 800,
    900, 1000, 1500, 2000, 3000};
int numTimes = 19;
byte selectedTimeIndex;
int timerMinute;
int timerSecond;

int buzzerPin = 11;
int aPin = 8;
int bPin = 10;
int buttonPin = 9;

boolean stopped = true;

byte digits[10][8] = {
// a b c d e f g .
{ 1, 1, 1, 1, 1, 1, 0, 0}, // 0
{ 0, 1, 1, 0, 0, 0, 0, 0}, // 1
{ 1, 1, 0, 1, 1, 0, 1, 0}, // 2
{ 1, 1, 1, 1, 0, 0, 1, 0}, // 3
{ 0, 1, 1, 0, 0, 1, 1, 0}, // 4
{ 1, 0, 1, 1, 0, 1, 1, 0}, // 5
{ 1, 0, 1, 1, 1, 1, 1, 0}, // 6
{ 1, 1, 1, 0, 0, 0, 0, 0}, // 7
{ 1, 1, 1, 1, 1, 1, 1, 0}, // 8
{ 1, 1, 1, 1, 0, 1, 1, 0} // 9
};

void setup()
{
    for (int i=0; i < 8; i++)
    {
        pinMode(segmentPins[i], OUTPUT);
    }
    for (int i=0; i < 4; i++)
    {
        pinMode(displayPins[i], OUTPUT);
    }
    pinMode(buzzerPin, OUTPUT);
    pinMode(buttonPin, INPUT);
    pinMode(aPin, INPUT);
    pinMode(bPin, INPUT);
    selectedTimeIndex = EEPROM.read(0);
    timerMinute = times[selectedTimeIndex] / 100;
    timerSecond = times[selectedTimeIndex] % 100;
}
```

(continúa)

LISTADO DE PROYECTO (continúa)

```

}

void loop()
{
    if (digitalRead(buttonPin))
    {
        stopped = ! stopped;
        digitalWrite(buzzerPin, LOW);
        while (digitalRead(buttonPin)) {};
        EEPROM.write(0, selectedTimeIndex);
    }
    updateDisplay();
}

void updateDisplay() // mmss
{
    int minsecs = timerMinute * 100 + timerSecond;
    int v = minsecs;
    for (int i = 0; i < 4; i++)
    {
        int digit = v % 10;
        setDigit(i);
        setSegments(digit);
        v = v / 10;
        process();
    }
    setDigit(5); // todos los dígitos apagados para evitar iluminación desigual
}

void process()
{
    for (int i = 0; i < 100; i++) // modificar este número entre intermitente y desenfoque
    {
        int change = getEncoderTurn();
        if (stopped)
        {
            changeSetTime(change);
        }
        else
        {
            updateCountingTime();
        }
    }
    if (timerMinute == 0 && timerSecond == 0)
    {
        digitalWrite(buzzerPin, HIGH);
    }
}

void changeSetTime(int change)

```

(continúa)

LISTADO DE PROYECTO (continúa)

```

{
    selectedTimeIndex += change;
    if (selectedTimeIndex < 0)
    {
        selectedTimeIndex = numTimes;
    }
    else if (selectedTimeIndex > numTimes)
    {
        selectedTimeIndex = 0;
    }
    timerMinute = times[selectedTimeIndex] / 100;
    timerSecond = times[selectedTimeIndex] % 100;
}

void updateCountingTime()
{
    static unsigned long lastMillis;
    unsigned long m = millis();
    if (m > (lastMillis + 1000) && (timerSecond > 0 || timerMinute > 0))
    {
        digitalWrite(buzzerPin, HIGH);
        delay(10);
        digitalWrite(buzzerPin, LOW);

        if (timerSecond == 0)
        {
            timerSecond = 59;
            timerMinute--;
        }
        else
        {
            timerSecond--;
        }
        lastMillis = m;
    }
}

void setDigit(int digit)
{
    for (int i = 0; i < 4; i++)
    {
        digitalWrite(displayPins[i], (digit != i));
    }
}

void setSegments(int n)
{
    for (int i = 0; i < 8; i++)
    {
        digitalWrite(segmentPins[i], ! digits[n][i]);
    }
}

```

(continúa)

LISTADO DE PROYECTO (continúa)

```

        }

}

int getEncoderTurn()
{
    // return -1, 0, or +1
    static int oldA = LOW;
    static int oldB = LOW;
    int result = 0;
    int newA = digitalRead(aPin);
    int newB = digitalRead(bPin);
    if (newA != oldA || newB != oldB)
    {
        // algo ha cambiado
        if (oldA == LOW && newA == HIGH)
        {
            result = -(oldB * 2 - 1);
        }
    }
    oldA = newA;
    oldB = newB;
    return result;
}

```

El temporizador se encontrará siempre en uno de estos dos estados: o bien estará **detenido**, en cuyo caso, al girar el codificador giratorio cambiará la hora, o puede estar **funcionando**, en cuyo caso estará contando regresivamente. Presionando el botón del codificador giratorio alternará entre los dos estados.

En lugar de hacer que el codificador giratorio cambie la hora un segundo por paso, tenemos una matriz de horas estándar que nos encaja perfectamente para lo que deseamos hacer. Esta matriz se puede modificar y ampliar pero, si cambia su longitud, debe modificar la variable **numTimes** según corresponda.

La biblioteca **EEPROM** se utiliza para almacenar la última hora marcada, de modo que cada vez que el proyecto se ponga en marcha, recordará esa última hora.

El proyecto emite un pequeño ruido con el clic de cada segundo. Si lo desea, puede desactivarlo. En la

función **updateCountTime** encontrará las líneas de código para convertir esto en comentario o eliminarlo.

Pongamos todo junto

Cargue el sketch terminado del Proyecto 30 desde su **Arduino Sketchbook** y descárguelo en su placa (véase el Capítulo 1).

Resumen

Este es el último capítulo que contiene proyectos. El autor espera que al intentar llevar a cabo los proyectos de este libro haya aumentado en el lector el apetito por la experimentación y el diseño, y esto haga que sienta la necesidad de diseñar sus propios proyectos.

Por ello, el siguiente capítulo tiene por objeto ayudarle en el proceso de desarrollo de sus propios proyectos.

CAPÍTULO 10

Sus proyectos

¡BUENO! ¡AQUÍ ESTAMOS! Queremos pensar que ha intentado reproducir algunos de los proyectos del libro y deseamos que haya aprendido mucho durante el trayecto. Ahora es el momento de empezar a desarrollar sus propios proyectos utilizando lo que han aprendido. Y, sin duda, podrá tomar prestado trozos del diseño de los proyectos en este libro. Pero para ayudarle en su tarea, este capítulo presenta algunas técnicas de diseño y construcción.

Circuitos

Al autor le gusta iniciar los proyectos con una vaga idea de lo que quiere lograr y, poco a poco, comenzar también a diseñar desde la perspectiva de la electrónica. Generalmente, el software viene después.

La forma de expresar un circuito electrónico es utilizar un **esquema electrónico**. El autor ha incluido esquemas electrónicos para todos los proyectos en este libro con lo que, incluso si no está muy familiarizado con la electrónica, llegados a este punto debería haber visto ya suficientes esquemas para comprender más o menos la relación que hay entre los **esquemas** y los **diagramas de distribución** de la placa de pruebas, que también se han incluido.

Esquemas electrónicos

En un **esquema electrónico**, las conexiones entre componentes se muestran como líneas. Estas conexiones utilizan las tiras conductoras que

existen debajo de la superficie de la placa de pruebas y los cables que conectan una placa de pruebas a otra. Para el tipo de proyectos que hemos desarrollado en este libro, normalmente no es demasiado importante la forma como se realiza la conexión. Es decir, la disposición de los cables no importa, siempre que se conecten todos los puntos.

Los esquemas electrónicos utilizan unas cuantas normas que merece la pena que destaquemos. Por ejemplo, es frecuente colocar las líneas de **GND** cerca de la parte inferior del diagrama y las **tensiones** mayores cerca de la parte superior del diagrama. Esto permite a aquellos que lean el esquema visualizar que el flujo de carga a través del sistema circula desde las tensiones altas a las tensiones más bajas.

Otra convención utilizada en los esquemas electrónicos es utilizar una pequeña barra para indicar una conexión a tierra (**GND**) cuando no hay suficiente espacio para dibujar todas las conexiones.

La Figura 10-1, mostrada originalmente en el Proyecto 5, muestra tres **resistencias**, todas con un cable conectado a la conexión de tierra (**GND**) de la placa Arduino. En la distribución de la placa de pruebas correspondiente (Figura 10-2), puede ver que las conexiones a **GND** van a través de tres cables y tres tiras conductoras de la placa de pruebas.

Existen muchas herramientas para dibujar esquemas electrónicos. Algunas de ellas son productos **CAD** (diseño asistido por computador) de electrónica integrada que son capaces de establecer por usted las pistas sobre una placa de

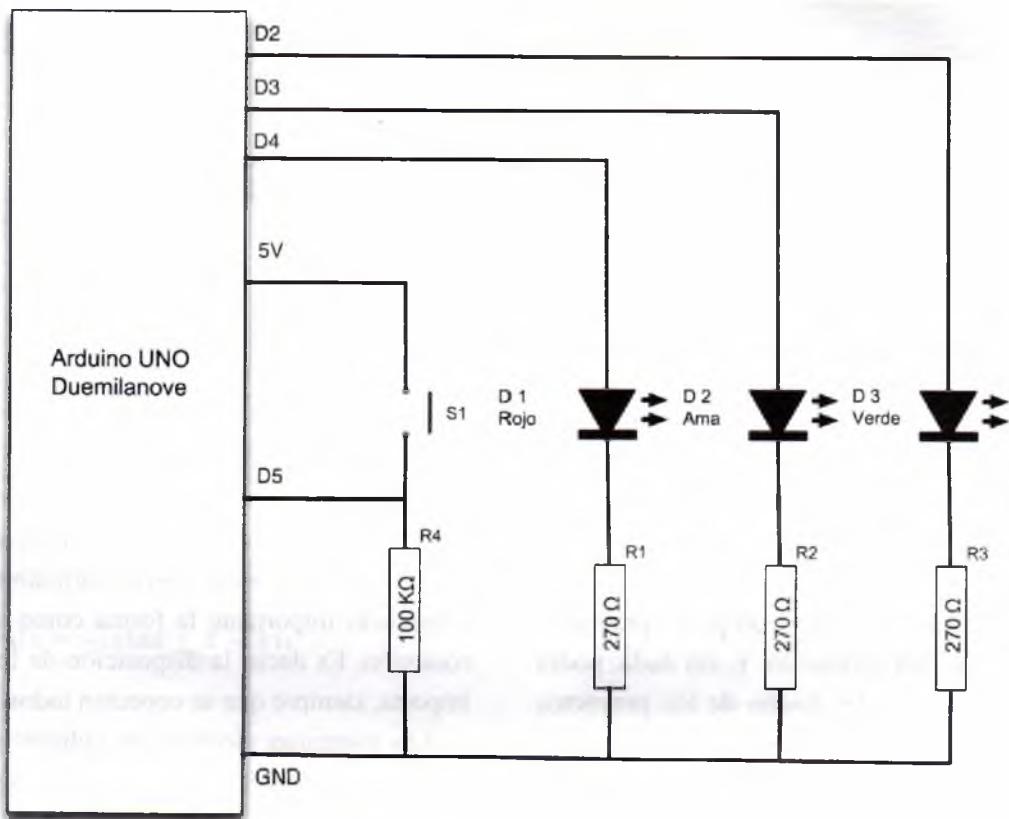


Figura 10-1 Ejemplo de esquema eléctrico.

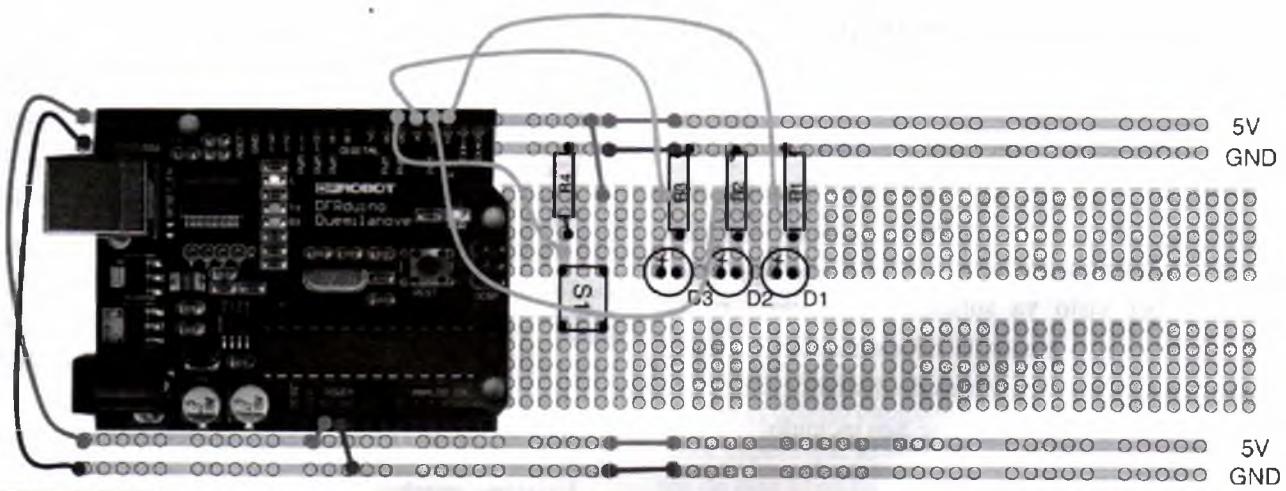


Figura 10-2 Ejemplo de diseño de placa de pruebas.

circuito impreso. Por lo general, estos crean **diagramas** bastante feos y el autor prefiere utilizar papel y lápiz o software de dibujo de propósito general. Todos los esquemas de este libro se han creado con el excelente software (aunque con un nombre raro) llamado **OmniGraffle**, de **Omni Group**, que sólo está disponible para plataformas Mac. Las plantillas de **OmniGraffle** para dibujar diagramas de placas de pruebas y esquemas electrónicos están disponibles para su descarga en www.arduinoevilgenius.com.

Símbolos de componentes

La Figura 10-3 muestra los **símbolos** de los componentes electrónicos que hemos utilizado en este libro. Existen diferentes estándares para los esquemas de los circuitos, pero los símbolos básicos pueden todos reconocerse de un estándar a otro. El conjunto utilizado en el presente libro no sigue ningún estándar particular. He elegido el enfoque que considero el más fácil de seguir para los esquemas.

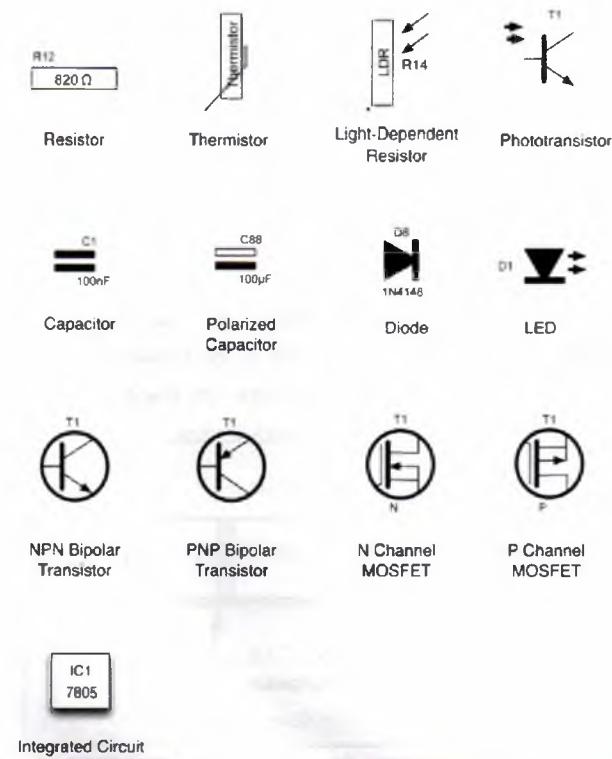


Figura 10-3 Símbolos electrónicos.

Componentes

En esta sección vamos a fijarnos en los aspectos prácticos de los componentes: qué hacen y cómo identificarlos, seleccionarlos y utilizarlos.

Fichas de características técnicas

Todos los fabricantes de componentes publican **fichas de características técnicas (datasheets)** de sus productos. Éstas suelen consistir en las especificaciones del comportamiento del componente. En el caso de las **resistencias** y de los **condensadores** la verdad es que no tienen demasiado interés; en cambio, sí resultan mucho más útiles para los **semiconductores** y **transistores** y, sobre todo, para los **circuitos integrados**. A menudo incluyen **notas de aplicación** que incluyen circuitos de ejemplo para utilizar los componentes.

Todas están disponibles en Internet. Pero tenga en cuenta que si escribe "BC158 datasheet" en su buscador favorito, encontrará que muchas de las entradas de mayor éxito son de empresas de publicidad que se aprovechan de que haya mucha gente buscando fichas técnicas (**datasheets**). Estas empresas colocan publicidad inútil junto a las fichas técnicas y finguen que agregan algo de valor por ver sus fichas de datos si se suscribe a su servicio. Estos sitios normalmente sólo llevan frustración y deben dejarse de lado en beneficio de los sitios web de los fabricantes. Por tanto, debe continuar la búsqueda hasta que encuentre URLs del tipo www.fairchild.com.

Por otra parte, muchos proveedores de componente al por menor (como **Farnell**) proporcionan **fichas técnicas** sin coste y sin publicidad de, prácticamente, todos los componentes que venden, lo que es de agradecer. También significa que puede comparar precios y comprar los componentes mientras se está informando.

Resistencias

Las **resistencias** son los componentes electrónicos más comunes y baratos. Sus usos más comunes son:

- Evitar un flujo excesivo de corriente (ver alguno de los proyectos que utilizan un LED)
- Su utilización en parejas o como **potenciómetro** para crear un divisor de tensión

En el Capítulo 2 explicamos la **ley de Ohm** y la utilizamos para calcular el valor de una resistencia en serie con un LED. Del mismo modo, en el Proyecto 19, redujimos la señal de nuestra **escalera de resistencias** usando dos **resistencias** como divisor de tensión.

Las **resistencias** tienen bandas de colores a su alrededor para indicar su valor. Sin embargo, si no está seguro del valor de una **resistencia**, siempre puede averiguarlo utilizando un **polímetro**. Una vez que se acostumbre a ello, es muy fácil leer los valores mediante los códigos de colores.

Cada banda de color tiene un valor asociado, como se muestra en la Tabla 10-1.

Normalmente habrá tres de estas bandas juntas, empezando desde un extremo de la **resistencia**, un hueco y luego una única banda en el otro extremo de la resistencia. La banda individual indica la exactitud del valor de la **resistencia**. Dado que ninguno de los proyectos en este libro requiere resistencias de

TABLA 10-1 Código de colores para resistencias

Negro	0
Marrón	1
Rojo	2
Naranja	3
Amarillo	4
Verde	5
Azul	6
Violeta	7
Gris	8
Blanco	9

precisión, no es necesario que escoja las resistencias en base a esto.

La Figura 10-4 muestra la disposición de las bandas de colores. El valor de la **resistencia** usa sólo tres bandas. La primera banda es el primer dígito, la segunda el segundo dígito y la tercera banda "multiplicadora" indica cuántos ceros hay que colocar detrás de los dos primeros dígitos.

Por lo tanto, una **resistencia** de **270 Ω** tendrá un primer dígito **2** (rojo), segundo dígito **7** (violeta), y un multiplicador de **1** (marrón). Del mismo modo, una resistencia de **10 KΩ** tendrá bandas de color marrón, negro y naranja (**1, 0, 000**).

La mayoría de nuestros proyectos utilizan **resistencias** de muy baja potencia. Se puede realizar un cálculo rápido para calcular la **corriente** que circula a través de la **resistencia** y, multiplicándola por la **tensión** que circula por ella, nos dirá la **potencia** utilizada por la resistencia. Las **resistencias** disipan este excedente de energía en forma de calor y, por tanto, se calientan si circula por ellas una cantidad significativa de **corriente**.

Sólo debe preocuparse de esto en el caso de las **resistencias** de menos de **100 Ω** o así, porque en las de valores más altos, la corriente que circule por ellas será muy baja.

Por ejemplo, una resistencia de **100 Ω** conectada directamente entre **5 V** y GND tendrá una **corriente** circulando por ella de $I = V / R$ ó $5 / 100$, es decir, **0,05 Amperios**. La **potencia** que utiliza será $I \times V$, es decir, $0,05 \times 5 = 0,25 W$.

La clasificación estándar de potencia de resistencias es **0.25 W** o **0.5 W** y, en nuestro caso, a menos que se indique lo contrario en los proyectos, las resistencias de **0.5 W** son suficientes.

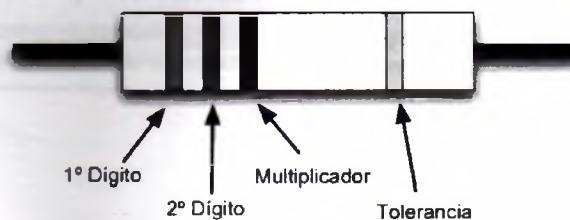


Figura 10-4 Código de colores para resistencias.

Transistores

Eche un vistazo a cualquier catálogo de componentes y verá que hay, literalmente, miles de tipos diferentes de **transistores**. En este libro hemos simplificado la lista a lo que se muestra en la Tabla 10-2.

El circuito básico de un **transistor** usado como interruptor se muestra en la Figura 10-5.

La **corriente** que circula desde la base al emisor (**b** a **e**) controla una corriente mucho mayor que circula desde el colector al emisor. Si no hay corriente que circule por la base, no habrá corriente que circule por la **carga**. En la mayoría de los transistores, si la **carga** tuviera resistencia cero, la corriente que circula hacia el colector sería 50 a 200 veces la corriente de la base. Sin embargo, nosotros queremos que nuestro **transistor** se active totalmente o se desconecte totalmente, por lo que la resistencia de carga limitará siempre la corriente del colector a la corriente requerida por la carga. Demasiada corriente de base dañará el **transistor** y además no permitirá cumplir el objetivo de controlar una corriente mayor con una más pequeña, y es por eso por lo que la base siempre tendrá conectada una **resistencia**.

Cuando se conmuta desde una placa Arduino, la **corriente** máxima de una salida es de **40 mA**, así es que deberíamos elegir una **resistencia** que permita circular unos **30 mA** de corriente cuando se active el pin de salida a **5 V**. Utilizando la **ley de Ohm**:

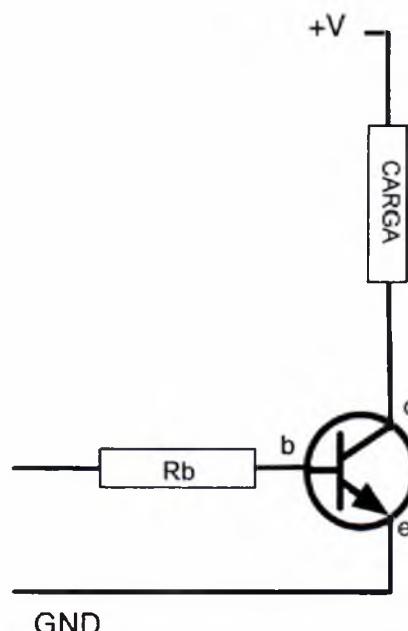


Figura 10-5 Circuito de conmutación básica de un transistor.

$$R = V / I$$

$$R = (5 - 0.6) / 0.03 = 147 \Omega$$

El **-0.6** es porque una de las características de los transistores bipolares es que siempre hay una tensión de alrededor de **0.6 V** entre la base y el emisor cuando el transistor está activado.

Por tanto, usando una resistencia de base de **150 Ω**, podríamos controlar una corriente de colector de 40 a 200 veces 30 mA, o, lo que es lo mismo, de **1,2 a 6 A**, lo que es más que suficiente en la mayoría de

TABLA 10-2 Transistores utilizados en este Libro

Transistor	Tipo	Objetivo
BC548	NPN bipolar	Conmutación de pequeñas cargas > 40 mA
BD139	NPN bipolar de potencia	Conmutación de cargas mayores (LED Luxeon). Véase Proyecto 6.
BC307	PNP bipolar	Conmutación de displays LED de ánodo común cuya corriente total es demasiado grande para la salida de Arduino (40 mA).
2N7000	FET canal N	Conmutación de baja potencia con muy baja resistencia de conexión. Véase Proyecto 7.
FQP33N10	MOSFET de potencia de canal N	Conmutación de alta potencia.
FQP27P06	MOSFET de potencia de canal P	Conmutación de alta potencia..

los casos. En la práctica, probablemente utilizaríamos una resistencia de $1\text{ K}\Omega$ o quizás de $270\ \Omega$.

El parámetro de valor máximo de los **transistores** no debe sobrepasarse o podría resultar dañado. Estos parámetros se pueden encontrar en la hoja de características del transistor. Por ejemplo, la del **BC548** contendrá muchos valores, aunque para nosotros los más interesantes son los que se resumen en la Tabla 10-3.

TABLA 10-3 Hoja de características de un transistor

Propiedad	Valor	Significado
I_c	100 mA	La corriente máxima que puede circular por el colector sin dañar al transistor.
β_{FE}	110-800	Ganancia de corriente en corriente continua. Es la relación entre la corriente de base y la corriente del colector. Como puede ver, varía entre 110 y 800 para este transistor.

Otros semiconductores

En los distintos proyectos hemos ido introduciendo diferentes tipos de componentes, desde LEDs a sensores de temperatura. En la Tabla 10-4 mostramos algunas indicaciones de los diversos proyectos realizados. Si desea desarrollar su propio proyecto, por ejemplo para indicar la temperatura, o cualquier otra cosa, puede que le resulte útil leer primero los proyectos que hemos desarrollado en el libro utilizando esos componentes.

Puede que incluso le resulte adecuado montar primero el proyecto de este libro y, a continuación, modificarlo para adaptarlo a sus propósitos.

Módulos y shields

No siempre tiene sentido hacer todo partiendo de cero. Después de todo, es la razón por la que com-

TABLA 10-4 Uso de componentes especializados en los Proyectos

Componente	Proyecto
LEDs de un color	Casi todos
LEDs multicolor	14
Panel matriz LEDs	16
Display 7 segmentos	15, 30
Chip amplificador audio	19, 20
LDR (célula fotosensible)	20
Termistor (sensor temperatura)	13
Regulador voltaje regulable	7

pramos una placa Arduino en lugar de hacernos la nuestra propia. Lo mismo es cierto en el caso de algunos módulos que queremos usar en nuestros proyectos.

Por ejemplo, el módulo de pantalla **LCD** que hemos utilizado en los Proyectos 17 y 22 contiene el chip controlador necesario para que funcione la pantalla **LCD**, reduciendo de este modo tanto el trabajo que tenemos que realizar en el **sketch** como el número de pines que tenemos que utilizar.

Si lo desea, existen otros tipos de módulos que puede utilizar en sus proyectos. Proveedores como **Sparkfun**, **Electan**, **Bricogeek** o **SuperRobótica.com** en España, son una gran fuente de ideas y módulos. Una muestra del tipo de módulos que se pueden obtener de estos proveedores incluye:

- **GPS**
- **Wi-Fi**
- **Bluetooth**
- **Zigbee** inalámbrica
- Módems **GPRS** para móviles.

Tendrá que dedicar tiempo a leer fichas, a planificar y a experimentar, pero sin duda, eso debería ser parte de la diversión.

Ligeramente menos complejo que utilizar un módulo desde cero es comprar una shield Arduino con el módulo ya instalado. Esto resulta especialmente una buena idea cuando los componentes que desea utilizar no van a ir colocados en una placa de pruebas (como los componentes de montaje en

superficie). Una **shield** ya montada puede darle un verdadero empujón al proyecto.

Continuamente siguen desarrollándose nuevas **shields** para las más variopintas aplicaciones y, en el momento de escribir este libro, puede comprar **shields** de Arduino para:

- **Ethernet** (conectar su Arduino a Internet)
- **XBee** (un estándar de conexión inalámbrica de datos utilizado en domótica, entre otras cosas)
- Controlador de motores
- **GPS**
- **Joystick**
- Interfaz de **tarjetas SD**.
- Pantalla gráfica táctil **LCD**
- **Wi-Fi**

Compra de componentes

Hace treinta años, los aficionados a la electrónica que vivían incluso en grandes pueblos o en pequeñas ciudades probablemente tenían a su disposición varias tiendas de repuestos o de reparación de radio/TV donde se podían comprar componentes y recibir consejos prácticos. En la actualidad, sólo quedan unos cuantos comercios que todavía venden componentes, como **Radioshack** en Estados Unidos, **Maplins** en el Reino Unido o algunos otros en nuestro país. No obstante, Internet ha entrado en escena para llenar el vacío y, ahora, comprar componentes es más fácil y más barato que nunca.

Con proveedores internacionales de componentes como **RS** y **Farnell** puede llenar una cesta de la compra online y recibir los componentes en un día o dos. Compare precios, ya que los precios varían considerablemente entre proveedores de los mismos componentes.

eBay puede ser una gran fuente para adquirir componentes. Si no le importa esperar unas semanas para recibir sus componentes, pueden conseguirse grandes gangas procedentes de China. A menudo tendrá que comprar grandes cantidades, pero también verá que puede ser más barato obtener 50 unidades de un componente de China que 5 uni-

dades adquiridas localmente. De esta manera dispondrá de algunas piezas de repuesto en su caja de componentes.

Herramientas

Para realizar sus proyectos necesitará un mínimo de herramientas. Si no tiene intención de soldar, entonces necesitará:

- **Cable rígido de distintos colores**, de un diámetro de alrededor de **0.6 mm** (23 SWG)
- **Alicates de punta fina y alicates de corte**, en particular para realizar los cables puente para la placa de pruebas
- **Placa de pruebas** de inserción de componentes (**protoboard**)
- **Polímetro**

Si tiene la intención de soldar, entonces también necesitará:

- **Un soldador**
- **Aleación de estaño** para soldar, sin plomo

Caja de componentes

Cuando comience a diseñar sus propios proyectos, le llevará algún tiempo acumular gradualmente su stock de componentes. Cada vez que termine un proyecto, algunos de los componentes volverán otra vez a su caja.

Es conveniente disponer de un stock de componentes básicos para que no tenga que estar comprando cosas cada vez que necesite una resistencia con un valor distinto de la que tenga. Habrá notado que la mayoría de los proyectos de este libro tienden a utilizar valores de resistencia como **100 Ω**, **1 KΩ**, **10 KΩ**, etc. En realidad, tampoco necesita tantos componentes diferentes para la mayoría de lo que es la base de un nuevo proyecto.

En el Apéndice se enumera lo que pensamos que puede ser un buen **kit de componentes básicos**.

Las cajas con diversos compartimentos que se pueden etiquetar ahorran mucho tiempo en la selección de componentes, especialmente en el caso de las resistencias que no tienen su valor escrito encima.

Alicates de corte y alicates de punta fina

Los **alicates de corte** son para cortar, y los **alicates de punta fina** se utilizan para sujetar las cosas (a menudo mientras las corta).

La Figura 10-6 muestra cómo se pella un cable. Suponiendo que sea diestro, sujeté los alicates con la mano izquierda y los alicates de corte con la derecha. Sujete el cable con los alicates de punta fina cerca de donde desea empezar a pelar el cable y luego pellizque suavemente el cable con los alicates de corte, dando despacio la vuelta al cable para, a continuación, tirar de la funda hacia fuera, para dejar el centro del cable pelado. A veces, el pellizco es demasiado fuerte y se corta o debilita el cable, y otras veces no se pellizca lo suficiente y la funda sigue intacta. Es simplemente una cuestión de práctica.

También puede comprar un pelador automático de cables, que pellizca y saca la funda en un solo movimiento. En la práctica, a menudo estos sólo funcionan bien para un determinado tipo de cable y, a veces, ¡simplemente ni funcionan!

Soldar

No es necesario gastar mucho dinero para conseguir un buen **soldador**. Las estaciones de soldadura con temperatura controlada, como la que se muestra en la Figura 10-7, son mejores, pero un **soldador eléctrico** de temperatura fija que se enchufa a la red es más que suficiente. Compre uno con la punta fina, y asegúrese de que está pensado para electrónica y no para usos de fontanería.

Para soldar, utilice **estaño fino** libre de plomo. Cualquier persona puede soldar cosas y hacer que funcionen; sin embargo, sólo algunos tienen la capacidad de hacer buenas soldaduras. No se preocupe si sus resultados no tienen una pinta tan curiosa como los que realizan los robots de construcción de circuitos impresos... ¡No la van a tener nunca!

Soldar es uno de esos trabajos en los que realmente se necesitarían tres manos: una mano para sostener el soldador, otra para el material de soldadura y otra para sostener lo que esté soldando. A veces, la cosa que se está soldando es lo suficientemente grande y pesada para quedarse en su lugar mientras se suelda; en otras ocasiones, es necesario mantenerlo en el aire. Los alicates pesados son buenos para esto, como lo son las pequeñas **mini prensas de ven-**



Figura 10-6 Alicates de corte y alicates de punta fina.



Figura 10-7 Soldador y material de soldadura.

tosa y los soportes del tipo "tercera mano" que utilizan **pinzas de cocodrilo** para sujetar las cosas.

Los pasos básicos para soldar son:

1. Humedecer la esponja del soporte del soldador.
2. Deje que el soldador alcance temperatura.
3. Estañe la punta del soldador presionándolo contra el estaño hasta que se derrita y cubra la punta.
4. Limpie la punta sobre una esponja húmeda esto produce un curioso y agradable sonido, pero también limpia el exceso de estaño. Ahora debe tener una pequeña punta plateada.
5. Toque con la punta del soldador el lugar donde desea soldar para calentarlo y, a continuación, tras una breve pausa (uno o dos segundos), coloque el estaño en el punto donde la punta del soldador se une a aquello que desea soldar. La soldadura debe fluir como si fuera líquido, creando una estupenda unión.
6. Retirar el estaño y el soldador, colocando el soldador en su soporte, teniendo mucho cuidado de no mover nada mientras se solidifica la soldadura. Si algo se mueve, vuelva a tocarlo con el soldador para volver a fundir el estaño; de lo contrario, puede producir una mala conexión, lo que se llama una **soldadura fría**.

Por encima de todo, trate de no calentar componentes sensibles (o caros) más tiempo del estrictamente necesario, especialmente si sus terminales de conexión son cortos.

Antes de empezar a trabajar en lo verdaderamente importante, practique la soldadura intentando unir trozos de cables usados, o soldando cables a alguna placa de circuitos antigua que tenga por ahí.

Polímetro

El gran problema de los electrones es que no se puede ver al "animal" que llevan dentro. Un **polímetro** permite ver qué es lo que están haciendo. Permite medir el **voltaje**, la **corriente**, la **resistencia** y, a menudo, también otros parámetros, como la **capacidad** o la **frecuencia**. Un **polímetro** barato de 10 € es perfectamente adecuado para casi cualquier tarea. Los profesionales utilizan medidores mucho más sólidos y precisos, pero para nosotros no es necesario en la mayoría de los casos.

Los **polímetros**, como el que se muestra en la Figura 10-8, pueden ser de tipo **analógico** o **digital**. Normalmente, se puede saber más con un medidor analógico que con uno digital, ya que se puede ver la rapidez del movimiento de la aguja y las alteraciones, algo que no es posible con un medidor digital, donde sólo cambian los números. Sin embargo, cuando tenemos una tensión estable, resulta mucho más fácil leer un medidor digital, ya que los analógicos tienen varias escalas y hay que decidir a qué escala hay que mirar antes de tomar la lectura.

También se pueden utilizar **polímetros** con selección automática de escala, en los que, una vez que haya seleccionado si está midiendo **corriente** o **tensión**, cambiará automáticamente el rango cuando aumenten la **tensión** o la **corriente**. Esto sin duda es útil, aunque puede que algunos argumenten que si ya nos encontramos en la fase de pensar en el rango de tensiones antes de medirlo, ya estamos haciendo en realidad una medida útil.



Figura 10-8 | Polímetro.

Para medir la tensión usando un polímetro:

1. Ponga el **polímetro** en un rango de **tensión** (comience utilizando una escala superior a la de la tensión que va a medir).
2. Conecte el cable negro del medidor a **GND** (tierra). Una **pinza de cocodrilo** montada sobre la punta de medición facilita la tarea.
3. Toque con la punta roja en el punto cuya **tensión** quiere medir. Por ejemplo, para ver si una salida digital de Arduino está activada o desactivada, puede tocar con la

punta roja en el pin y leer la tensión, que debe ser **5 V** o **0 V**.

Medir la **corriente** es diferente de la medición de la **tensión** porque, en el caso de la **corriente**, lo que se desea medir es la corriente que circula por algo y no la tensión en algún punto. Para medir, intercalamos el **polímetro** en el trayecto de la **corriente** que se está midiendo. Esto significa que cuando el **polímetro** se configura en una escala para medir **corriente**, habrá una resistencia muy baja entre sus puntas de prueba, por lo que debe tener cuidado de no cortocircuitar nada con las mismas.

La Figura 10-9 muestra cómo puede medir la **corriente** que circula por un LED.

Para medir la **corriente**:

1. Ponga el **polímetro** en una escala de **corriente** superior a la corriente esperada. Observe que los **polímetros** a menudo tienen una clavija aparte de alta corriente, que se utilizan para medir corrientes de hasta 10 A.
2. Conecte la punta positiva del medidor al lado más positivo de la corriente.
3. Conecte la punta negativa del medidor al

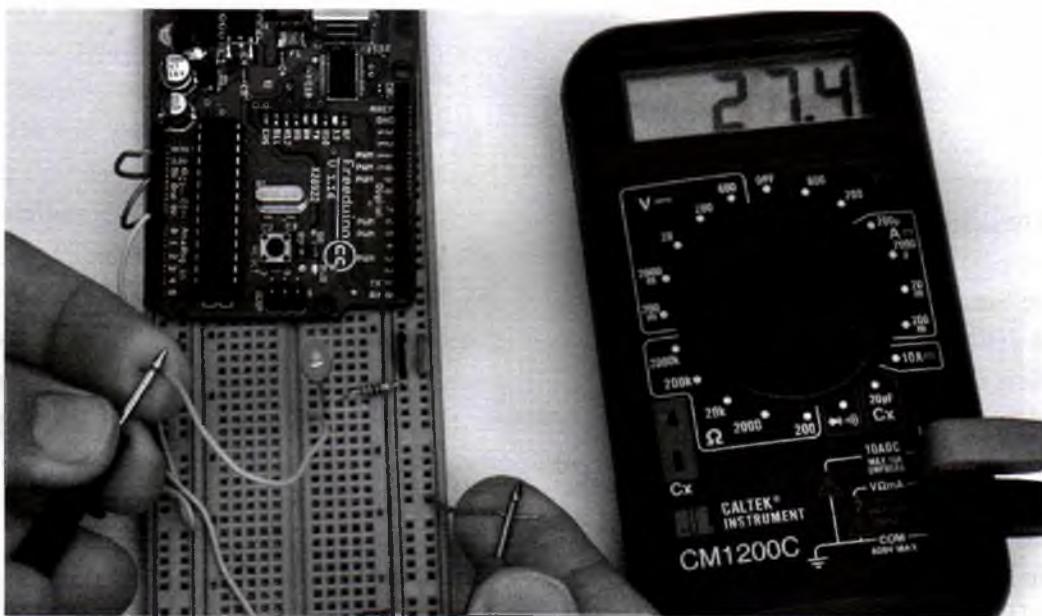


Figura 10-9 | Medición de corriente.

lado más negativo. Tenga en cuenta que si conecta las puntas al revés, un medidor digital sólo indicará una corriente negativa; pero, en uno analógico, puede dañarlo.

4. En el caso de un LED, el LED debe iluminarse con la misma fuerza que antes de colocar el medidor en el circuito, y podrá leer el consumo actual.

Otra característica de un **polímetro** que a veces resulta útil es la **prueba de continuidad**.

Normalmente esto hará que suene un "bip" cuando las puntas de prueba se tocan entre sí. Puede usarlo para probar fusibles, etc., pero también para comprobar si se ha producido algún cortocircuito accidentalmente en una placa de circuito o si hay conexiones rotas en un cable. En ocasiones, también resulta útil la medición de una resistencia; por ejemplo, cuando se desea saber el valor de una **resistencia** que no está marcada.

Algunos medidores también tienen conexiones para comprobar **diodos** y **transistores**, lo que puede ser útil para comprobarlos y desechar los **transistores** que se han quemado.

Osciloscopio

En el Proyecto 18 construimos un **osciloscopio** simple. Los **osciloscopios** son una herramienta indispensable para cualquier tipo de diseño o prueba de productos electrónicos en el que se esté tratando con una señal que cambia con el tiempo. Son equipos relativamente caros y los hay de varios tipos. Uno de los tipos que resulta más rentables es similar en concepto al del Proyecto 19. Ese **osciloscopio** simplemente envía sus lecturas a un ordenador, que es el encargado de mostrarlas.

Se han escrito libros enteros acerca de cómo utilizar eficazmente un **osciloscopio**, y cada **osciloscopio** es diferente, por lo que aquí sólo nos ocuparemos de los conceptos básicos.

En la Figura 10-10, puede verse la pantalla que muestra la **forma de onda**, en la parte superior de la cuadrícula. La cuadrícula vertical está en unidades de algunas fracciones de voltios, que en esta pantalla es de 2 V por división. Por tanto, el voltaje total de la **onda cuadrada** es 2.5×2 , es decir, 5 V.

El eje horizontal es el eje de tiempos, y este está calibrado en segundos, en este caso 500 microsegundos (**μs**) por división. Así es que la duración de

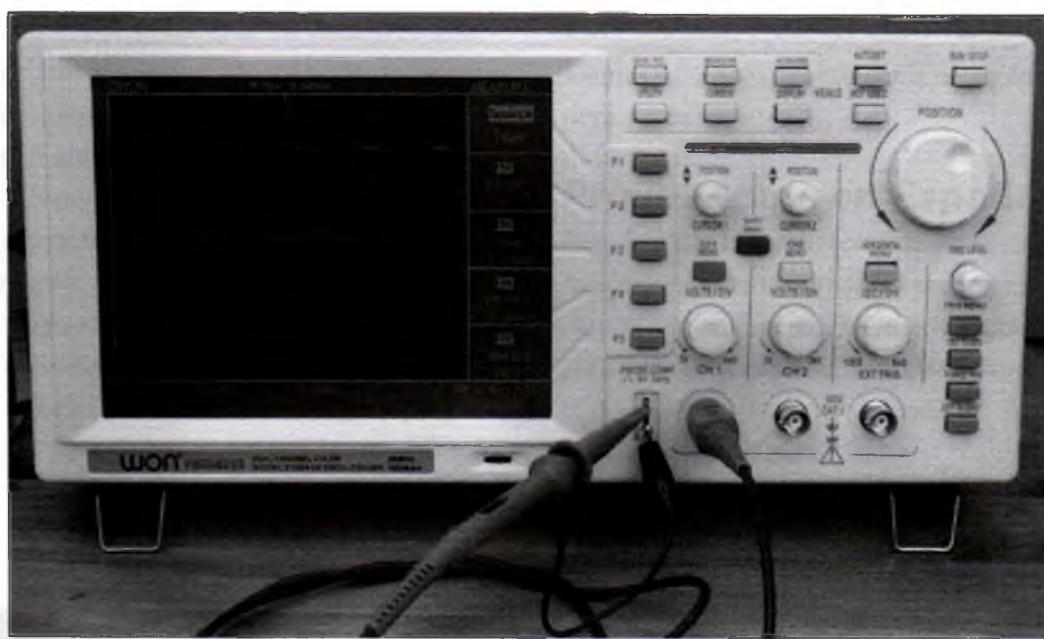


Figura 10-10 Osciloscopio.

un ciclo completo de la onda es **1000 µS**, es decir, **1 milisegundo**, lo que indica una frecuencia de **1 KHz**.

Ideas para proyectos

La zona **Arduino Playground** del sitio web principal de Arduino (www.arduino.cc) es una gran fuente de ideas para proyectos. De hecho, tiene incluso una sección específica para ideas de proyectos, dividido en fáciles, medios o difíciles.

Si escribe "proyectos Arduino" en su buscador favorito o en YouTube, encontrará un sinfín de proyectos interesantes que la gente ha emprendido.

Otra fuente de inspiración son los catálogos de componentes, ya sean en línea o en papel. Echándole un vistazo a estos catálogos puede que se encuentre con algún componente interesante que le haga preguntarse qué es lo que se podría hacer con él.

Dejar a nuestra mente pensar en posibles proyectos es algo saludable que debemos fomentar todos los interesados en esta interesantísima actividad. Tras explorar todas las opciones y darle mil vueltas, ¡el proyecto comenzará a tomar forma!

A P É N D I C E

Componentes y suministros

TODOS LOS COMPONENTES UTILIZADOS en este libro pueden adquirirse fácilmente a través de Internet. Sin embargo, a veces resulta un poco difícil encontrar exactamente lo que se está buscando. Por esta razón, este apéndice enumera los componentes junto con códigos de referencia para diversos proveedores. Esta es información que probablemente deje de estar actualizada con el tiempo, pero los grandes proveedores como **Farnell** o **RS** generalmente listarán los artículos "fuera de catálogo" y ofrecerán alternativas.

Proveedores

Existen tantos proveedores de componentes que parece un poco injusto hacer una lista con los pocos que el autor conoce. Por tanto, eche un vistazo en Internet, ya que los precios varían considerablemente entre los distintos proveedores.

Hemos enumerado códigos de piezas de **Farnell** y de **RS** porque son suministradores internacionales, pero también porque disponen de un fantástico catálogo de componentes. Habrá pocas cosas que no pueda comprarles. Pueden ser sorprendentemente baratos para componentes comunes, como resistencias y semiconductores, pero para los componentes menos comunes, como los módulos de diodo láser, sus precios pueden ser diez veces mayor de lo que pueda encontrar en otros sitios en Internet. Su principal función es suministrar a profesionales.

Algunos proveedores de menor tamaño se especializan en suministrar componentes para aficiona-

dos a la electrónica que crean proyectos de microcontroladores como los nuestros. Estos no suelen disponer de la misma gama de componentes, pero a menudo tienen componentes más exóticos y divertidos a precios razonables. Un buen ejemplo de esto es **Sparkfun Electronics**, en USA, o **Electan.com**, **BricoGeek.com** o **SuperRobótica.com** en España, pero sin duda existen otros muchos en Internet.

A veces, cuando eche únicamente en falta un par de componentes, será estupendo que pueda ir a una tienda local y comprarlos. **Radioshack** en Estados Unidos y **Maplins** en el Reino Unido disponen de una amplia gama de componentes, y son ideales para esto en sus respectivos países.

Las secciones que aparecen a continuación listan los componentes por tipo, junto con algunas posibles fuentes y códigos de referencia de algún suministrador.

Proveedores en España

Para facilitar la localización de componentes incluimos algunos suministradores por Internet que conocemos personalmente, habiendo muchos más, que por motivos de espacio no podemos referir.

- Arduino, shields y sensores: **Electan.com**, **BricoGeek.com**
- Transistores, sensores, LEDs: **Micropik.com**
- Motores, servos, sensores, software gratuito robótica, tutoriales: **SuperRobótica.com**
- Transistores, resistencias, componentes electrónicos en general: **mleon.com**

Arduino y clones			
Código	Descripción	RS	Otros
1	Arduino UNO	715-4081	Electan.com BricoGeek.com
2	Arduino Duemilanove	696-1655	
3	Arduino Lilypad	—	BricoGeek.com
4	Arduino Shield Kit	696-16	Electan.com BricoGeek.com

Otros proveedores a revisar incluyen [eBay](#), [Sparkfun](#), [Robotshop.com](#) y [Adafruit](#).

Nota de la edición española: Aunque en el siguiente listado las referencias para Farnell y RS corresponden a resistencias de 0.5 W, en realidad se pueden sustituir por ¼ W.

Resistencias			
Código	Descripción	Farnell	RS
4	Resistencia 39 Ω 0.5 W	9338756	683-3601
5	Resistencia 100 Ω 0.5 W	9339760	683-3257
6	Resistencia 270 Ω 0.5 W	9340300	148-360A
7	Resistencia 1 KΩ 0.5 W	9339779	477-7928
8	Resistencia 4.7 KΩ 0.5 W	9340629	683-3799
9	Resistencia 10 KΩ 0.5 W	9339787	683-2939
10	Resistencia 33 KΩ 0.5 W	9340424	683-3544
11	Resistencia 47 KΩ 0.5 W	9340637	506-5434
12	Resistencia 56 KΩ 0.5 W	9340742	683-4206
13	Resistencia 100 KΩ 0.5 W	9339795	683-2923
14	Resistencia 470 KΩ 0.5 W	9340645	683-3730
15	Resistencia 1 MΩ 0.5 W	9339809	683-4159
16	Resistencia 4 Ω 1 W	1155042	683-5477
17	Potenciómetro lineal 100 K Ω	1227589	249-9266
18	Termistor, NTC, 33 K a 25°C, beta 4090	1672317RL (beta = 3950)	188-5278 (R = 30 K, beta = 4100)
19	LDR	7482280	596-141

Condensadores			
Código	Descripción	Farnell	RS
20	100 nF no polarizado	1200414	538-1203A
21	220 nF no polarizado	1216441	107-029
22	Electrolítico 100 µF	1136275	501-9100

Semiconductores				
Código	Descripción	Farnell	RS	Otros
23	LED rojo 5-mm	1712786	247-1151	Tienda local
24	LED amarillo 5-mm	1612434	229-2554	Tienda local
25	LED verde 5-mm	1461633	229-2548	Tienda local
26	LED infrarrojo 5-mm 940 nm	1020634	455-7982	Tienda local
27	LED rojo 3-mm	7605481	654-2263	Tienda local
28	LED verde 3-mm	1142523	619-2852	Tienda local
29	LED azul 3-mm	1612441	247-1561	Tienda local
30	LED blanco Luxeon 1W	1106587	467-7698	Micropik.com ó desmontar de linterna led
31	RGB LED (ánodo común)	1168585 (nota: tiene cables separados en lugar de ánodo común)	247-1505 (tiene cables separados en lugar de ánodo común)	BricoGeek.com Micropik.com Electan.com
32	Módulo diodo láser rojo 3 mW	\$\$\$	\$\$\$	eBay o quitar de puntero láser barato
33	Display LED 7 segmentos, 2-díg (ánodo común)	1003316	195-136	BricoGeek.com Electan.com
34	Array LED 8 x 8 (2 colores)	—	—	BricoGeek.com
35	Display gráfico barra 10-segm.	1020492	—	BricoGeek.com Electan.com
36	Fototransistor IR 935 nm	1497882	195-530	Electan.com
37	Sensor de infrarrojos IR 940 nm o similar	4142822	315-387	SuperRobótica BricoGeek.com
38	Diodo 1N4004	9109595	628-9029	Tienda local
39	Transistor BC307/ BC556	1611157	544-9309A	Tienda local
40	Transistor BC548	1467872	625-4584	Tienda local
41	Transistor DB139	1574350	—	Tienda local
42	FET 2N7000	9845178	671-4733	Electan.com
43	MOSFET de potencia canal N FQP33N10	9845534	671-5095	
44	MOSFET de potencia canal P FQP27P06	9846530	671-5064	
45	Regulador tensión LM317	1705990	686-9717	Tienda local
46	Contador décadas 4017	1201278	519-0120	Tienda local
47	Amplificador audio TDA7052 1W	526198	658-485A	Tienda local

Otros proveedores a revisar, especialmente para LEDs, etc. incluyen eBay, Micropik.com, Sparkfun, Robotshop.com y Adafruit.

Otros				
Código	Descripción	Farnell	RS	Otros
48	Pulsador miniatura	1448152	102-406	Tienda local
49	Conector alimentación 2.1 mm	1200147	455-132	Tienda local
50	Clip de pila 9 V	1650667	489-021A	Tienda local
51	Fuente alim. regul. 15 V 1.5 A	1354828	238-151	SuperRobótica Tienda local
52	Terminal de tornillo 3 cables	1641933	220-4276	Tienda local
53	Placa perforada	1172145	206-8648	Tienda local
54	Teclado matricial 4 x 3	1182232	115-6031	BricoGeek.com
55	Pines macho 2.54 mm paso	1097954	668-9551	BricoGeek.com Electan.com
56	Terminal hembra 2.54 mm paso	1218869	277-9584	BricoGeek.com Electan.com
57	Codificador giratorio con pulsador	1520815	—	Electan.com
58	Módulo LCD (controlador HD44780)	1137380	532-6408	BricoGeek.com Sparkfun
59	Altavoz miniatura 8 Ω	1300022	628-4535	Tienda local
60	Micrófono Electret	1736563	—	Tienda local
61	Relé 5 V	9913734	499-6595	Tienda local
62	Fuente alimentación 12 V 1 A	1279478	234-238	SuperRobótica Tienda local
63	Ventilador de ordenador 12 V	1755867	668-8842	Tienda local
64	Motor 6 V	599128	238-9721	SuperRobótica Tienda local
65	Motor Servo 9 gr	—	—	SuperRobótica BricoGeek.com
66	solenoide 5V (< 100 mA)	9687920	533-2217	BricoGeek.com
67	Disco Piezoelectrónico (solo, sin electrónica)	1675548	511-7670	Tienda local
68	Zumbador Piezoelectrónico (con oscilador incluido)	1192513	—	Tienda local
69	Contacto magnético miniatura tipo Reed	1435590	289-7884	BricoGeek.com Tienda local
70	Marco fotos 13 x 18			Supermercado
71	Fuente alimentación 5 V, 500 mA	1297470	234-222	De cargador de teléfono móvil
72	Placa de pruebas	4692597	—	Tienda local Electan.com SuperRobótica

Las tiendas locales permiten ver los componentes antes de comprarlos, y suelen ser un buen sitio para compra de componentes como fuentes de alimentación o ventiladores. Además, son más baratas para éste tipo de suministros que los grandes proveedores profesionales.

Kit de componentes para principiantes

Es bueno disponer de un cierto stock de componentes comunes. La siguiente lista ofrece algunos de los componentes que probablemente se verá usando en sus distintos proyectos una y otra vez.

- Resistencias: Resistencia de 1/4 W, 100 Ω , 270 Ω , 1 K Ω , 10 K Ω y 100 K Ω
- LEDs rojos de 5 mm
- Transistores: BC548, BD139

Índice

Las referencias a las figuras están en cursiva.

Comando !, 123

A

actualizaciones, 3

alicates, 176

alimentación

hipnotizador (Proyecto 24), 134-138

láser controlado por servo (Proyecto 25), 138-143

termostato LCD (Proyecto 22), 125-131

ventilador controlado por ordenador

(Proyecto 23), 132-133

amplificación, 36

ánodos comunes, 91-92

Arduino Lilypad, 20

proveedores, 182

Arduino Mega, 20

Arduino Playground, 179

Arduino Protoshield, 37, 38-40

arpa de luz (Proyecto 20), 117-120

array de LEDs (Proyecto 16), 95-101

Asistente de Extracción, 2-3

Asistente de Nuevo Hardware Encontrado, 3, 4

ATmega168, 20

ATmega328, 19-20

B

bibliotecas, instalación en software de Arduino, 64, 65, 154-155, 161

botón **Reset**, 1

Brevig, Alexander, 64

C

cable USB, conexión de tipo A a tipo B, 1

caja de componentes, 175

CDA, 111

cerradura magnética (Proyecto 27), 148-153

chip de interfaz USB, 20

circuito

circuito eléctrico, 169-171

símbolos eléctricos, 171

circuito eléctrico, 169-171

véanse también los distintos proyectos

clones, 182

codificadores giratorios, 67, 68

código, 8

Comando !, 123

componentes

comprar, 175

kit de iniciación, 185

proveedores, 181-184

componentes de la placa, 16

chip de interfaz USB, 20

conector de programación serie, 20

conexiones digitales, 18-19

conexiones eléctricas, 16-18

entradas analógicas, 18

fuente de alimentación, 16

microcontroladores, 19-20

oscilador, 20

pulsador **Reset**, 20

comprar componentes, 175

condensadores, 108

proveedores, 182

conector de programación serie, 20

conector **Reset**, 16-17

conexiones digitales, 18-19

conexiones eléctricas, 16-18

configurar entorno de Arduino, 6, 7

constantes, 23

contador de décadas, 96

control remoto por infrarrojos (Proyecto 28), 153-158
controladores puente H, 134
controladores USB, instalación, 3-4
convertidor digital-analógico (CDA), 111
corriente, medición, 178
cortacables, 176

D

dado de LEDs (Proyecto 9), 55-59
dado doble de LEDs de siete segmentos (Proyecto 15), 91-95
dato, 55-59, 91-95
descarga de software de proyecto, 6-8
destello código Morse S. O. S. (Proyecto 2), 27-29
detector de mentiras (Proyecto 26), 145-148
Diecimila. Véase placa Arduino Diecimila
directivas de pre-procesador, 78
 diseño, 50
display de luces multicolor (Proyecto 14), 85-89
displays LCD, 101-102
Duemilanove. Véase placa Arduino Duemilanove

E

EEPROM, 20, 78, 82, 153
ejemplo, 21-23
EMF fuerza electromotriz, 126
entradas analógicas, 18
entradas y salidas digitales, 41
 salida analógica de entradas digitales, 112
entradas, 15
 analógico, 18
 digital, 41
EPROM, 15
escalera de resistencias R-2, 111, 112
expresiones lógicas, 25

F

FCEM o fuerza contraelectromotriz, 126, 150
FETs, 48, 96
fotorresistencias, 72
fototransistores, 72, 73-74
fuente de alimentación, 16
función **Alloff**, 138
función **getEncoderTurn**, 69
función **playNote**, 113-116
función **random**, 55

función **randomSeed**, 55
funciones, 22

G

generación de números aleatorios, 55
GND, 17
líneas en circuitos eléctricos, 169
gráfico de colores web, 87

H

herramientas, 175
 caja de componentes, 175
 cortacables y alicates, 176
 osciloscopios, 179
 polímetro, 177-179
 soldar, 176-177
hipnotizador (Proyecto 24), 134-138
Histéresis, 130
hojas de características técnicas, 171
"hunting", 127-130

I

ideas para proyectos, 179-180
imágenes de disco, 5
instalación del software, 1-2
 en Linux, 5-6
 en Mac OS X, 4-5
 en Windows, 2-4
instrucciones condicionales, 24-25
integers (enteros), 22
integración matemática imperfecta (*leaky*), 75
interruptor magnético Reed, 159

K

kit de iniciación, 185

L

láser controlado por servo (Proyecto 25), 138-143
láseres, láser controlado por servo (Proyecto 25), 138-143
LDRs, 72
LED intermitente (Proyecto 1), 8-11
LED intermitente (Proyecto 1), placa de pruebas, 12
Ledpin, 21, 22
LEDs
 añadir un LED externo, 10-11

- array de LEDs (Proyecto 16), 95-101
- dado de LEDs (Proyecto 9), 55-59
- dado doble de LEDs de siete segmentos (Proyecto 15), 91-95
- destello código Morse S. O. S. (Proyecto 2), 27-31
- LED parpadeante (Proyecto 1), 8-11
- LEDs de siete segmentos, 89-91
- Luxeon 1 W, 35
- luz estroboscópica (Proyecto 6), 44-47
- luz estroboscópica de alta potencia (Proyecto 8), 52-55
- Luz para T.A.E.(Proyecto 7), 47-52
- modelo de Semáforo (Proyecto 5), 41-44
- traductor de código Morse (Proyecto 3), 31-35
- traductor de código Morse de gran intensidad (Proyecto 4), 35-40
- LEDs de siete segmentos, 89-91
- Véase también LEDs*
- lenguaje C, 21
 - aritmética, 23-24
 - arrays** (matrices), 30-32
 - constantes, 23
 - ejemplo, 21-23
 - enteros, 22
 - expresiones lógicas, 25
 - funciones, 22
 - instrucciones condicionales, 24-25
 - loops** (bucles), 23, 29-30
 - nomenclatura Mayúscula en segunda palabra, 22
 - operadores lógicos, 25
 - parámetros, 23
 - punto y coma, 22
 - strings** (cadenas), 24
 - tipos de datos, 23, 24
 - variables, 22, 23
- lenguaje Ruby, instalación, 109-110
- ley de Ohm, 17-18
- Lilypad. *Véase Arduino Lilypad.*
- LINUX, instalación del software en, 5-6
- lógica transistor-transistor, 16
- loops (bucles), 23, 29-30
- luces
 - dado doble de LEDs de siete segmentos (Proyecto 15), 91-95
 - display de luces multicolor (Proyecto 14), 85-89
 - luz estroboscópica (alta potencia) (Proyecto 8), 52-55
 - luz estroboscópica (Proyecto 6), 44-46
 - matriz de LEDs (Proyecto 16), 95-101
 - placa de mensajes USB (Proyecto 17), 102-105
 - luz estroboscópica (alta potencia - Proyecto 8), 52-55
 - luz estroboscópica (Proyecto 6), 44-46
 - crear una **shield** para, 47
 - luz estroboscópica de alta potencia (Proyecto 8), 52-55
 - luz para T.A.E.(Proyecto 7), 47-52
- M**
 - Mac OS X, instalación del software en, 4-5
 - matrices, 30-32
 - medición de corriente, 178
 - medición de resistencia, 179
 - medición de resistencia, 179
 - medición de temperatura, 77
 - medición de voltaje, 177-178
 - medidor VU (Proyecto 21), 120-124
 - medidores analógicos, 177
 - medidores de escalado automático, 177
 - medidores digitales, 177
 - memoria, 15, 19-20
 - microcontroladores, 15, 19-20
 - modelo de semáforo (Proyecto 5), 41-44
 - modelo de semáforo utilizando codificador giratorio (Proyecto 11), 68-72
 - modulación por anchura de pulsos (PWM), 48
 - módulos, 174-175
 - monitor de pulsaciones (Proyecto 12), 73-77
 - MOSFETs, 135-136
 - multímetro, 177-179
- O**
 - OmniGraffle, 171
 - ondas cuadradas, 111
 - ondas sinusoidales, 111
 - operador de marketing, 123
 - operadores lógicos, 25
 - operadores, 25
 - operador de marketing, 123
 - oscilador, 20
 - osciloscopio (Proyecto 18), 107-111
 - osciloscopios, 179

P

- página web, 2
- parámetros, 23
- PCB. Véase placas de circuito Protoshield
pela-cables, 176
- piezozumbadores, 146-147
- Placa Arduino Duemilanove, 2
 - proveedores, 182
 - puesta en marcha, 1
 - seleccionar, 6, 7
- Placa Arduino Diecimila
 - proveedores, 182
 - puesta en marcha, 1
 - seleccionar, 6, 7
- Placa Arduino UNO
 - proveedores, 182
 - puesta en marcha, 1
 - seleccionar, 6, 7
- placa de mensajes USB (Proyecto 17), 102-105
- placa de mensajes, 102-105
- placas de circuito Protoshield, 39
- placas de prueba, 11-13
- polarización por realimentación de colector, 121
- potenciómetros, 45, 46, 47, 147
- programa Blink, 1
 - modificar, 8-11
- programas, 8
- proveedores, 181-184
- proyectos
 - arpa de luz, 117-120
 - cerradura magnética, 148-153
 - control remoto por infrarrojos, 153-158
 - dado de LEDs, 55-59
 - dado doble de LEDs de siete segmentos, 91-95
 - destello código Morse S. O. S., 27-29
 - detector de mentiras, 145-148
 - display de luces multicolor, 85-89
 - hipnotizador, 134-138
 - ideas, 179-180
 - láser controlado por servo, 138-143
 - LED parpadeante, 8-11
 - luz estroboscópica de alta potencia, 52-55
 - luz estroboscópica, 44-47
 - Luz para T.A.E., 47-52
 - matriz de LEDs, 95-101
 - medidor VU, 120-124

- modelo de semáforo utilizando un codificador giratorio, 68-72
- modelo de semáforo, 41-44
- monitor de pulsaciones, 73-77
- osciloscopio, 107-111
- placa de mensajes USB, 102-105
- reloj Lilypad, 159-162
- reproductor de música, 112-116
- teclado código de seguridad, 61-67
- temporizador de cuenta atrás, 163-168
- termostato LCD, 125-131
- traductor de código Morse gran intensidad, 35-38
- traductor de código Morse, 31-35
- USB registrador de temperaturas, 77-83
- ventilador controlado por ordenador, 132-133
- prueba de continuidad, 178-179
- puente (*jumper*) de alimentación, 1
- puerto serie, configuración, 6, 7
- pulsador **Reset**, 20
- PWM, 48

R

- RAM, 15
- red eléctrica, 110, 125
- registrador de temperaturas USB (Proyecto 13), 77-83
- regulador de voltaje, 16
- reloj Lilypad (Proyecto 29), 159-162
- reproductor de música (Proyecto 19), 112-116
- resistencias dependientes de la luz, 72
- resistencias, 10, 172
 - códigos de colores, 172
 - potenciómetros, 45, 46, 47, 147
- proveedores, 182
- resistencias dependientes de la luz (LDR), 72
- valores, 19

S

- salida analógica de entradas digitales, 112
- salidas, 15
 - salida analógica de entradas digitales, 112
- digital, 41
- semiconductores, proveedores, 183
- sensores
 - modelo de semáforo utilizando un codificador giratorio (Proyecto 11), 68-72

- monitor de pulsaciones (Proyecto 12), 73-77
registrar de temperaturas USB
(Proyecto 13), 77-83
teclado código de seguridad (Proyecto 10), 61-67
S
Serial Monitor, 34-35, 75
servomotores, 138
shields, 38-40, 47, 142, 174-175
Shirriff, Ken, 154
símbolos del código Morse, 32
sketches, 8
software
 instalar, 3-6
 descarga de software de proyecto, 6-8
 sketch Blink, 8-9
soldar, 176-177
solenoide, 148-150, 153
sonido
 arpa de luz (Proyecto 20), 117-120
 generación, 111-112
 osciloscopio (Proyecto 18), 107-111
 reproductor de música (Proyecto 19), 112-116
 Vúmetro o medidor VU (Proyecto 21), 120-124
Stanley, Mark, 64
Strings (cadenas), 24
- T**
tarjetas perforadas, 48
Teclado código de seguridad (Proyecto 10), 61-67
temperatura
 medición, 77
 registrar de temperaturas, 77-83
 termostato LCD, 125-131
temporizador de cuenta atrás (Proyecto 30), 163-168
Temporizador de cuenta atrás (Proyecto 30), 163-168
temporizador, 163-168
termistores, 77
 registrar de temperaturas USB
 (Proyecto 13), 77-83
termostato LCD (Proyecto 22), 125-131
termostato, 125-131
Theremin, 117
tipos de datos, 23, 24
traductor de código Morse (Proyecto 3), 31-35
traductor de código Morse de gran intensidad
(Proyecto 4), 35-38
- crear una **shield** para, 38-40
transistores bipolares, 90-91
transistores de efecto de campo de óxido metálico
(MOSFET), 135-136
transistores de efecto de campo, 48, 96
transistores PNP, 92
transistores, 173-174
 FETs, 48, 96
 hojas de características técnicas, 174
 MOSFETs, 135-136
 transistor bipolar NPN, 36
 transistores bipolares, 90-91
 transistores PNP, 92
 usados en este libro, 173
- TTL, 16
- U**
- UNO. *Véase placa Arduino UNO*
- V**
- variables, 22, 23
ventilador controlado por ordenador
(Proyecto 23), 132-133
voltaje, medir, 177-178
- W**
- Windows, instalación de software en, 2-4
- Z**
- zumbido de la red, 110

¡Diviértase creando 30 ingeniosos dispositivos controlados desde el ordenador!

Este curioso libro nos va mostrando de un modo fácil y ameno cómo realizar hasta 30 proyectos de electrónica y robótica utilizando como base el sistema de desarrollo del microcontrolador Arduino.

- Enseña de un modo sencillo y detallado los principios elementales de programación simplificada en C que va a necesitar -no se requiere ningún tipo de conocimientos previos en programación.
- Utiliza únicamente componentes y herramientas al alcance de todos, accesibles en cualquier distribuidor.
- Explica paso a paso cómo conectar una placa Arduino a su ordenador, cómo programarla y cómo conectarle distintos componentes electrónicos para crear hasta 30 curiosos dispositivos que podrá manejar con facilidad desde el ordenador.

Divertido, sorprendente y con el único límite de su propia imaginación!

30 Proyectos con Arduino:

- Incluye instrucciones paso a paso y prácticas ilustraciones.
- Proporciona detalles de esquemas y de construcción completos de cada proyecto.
- Explica los principios científicos detrás de cada proyecto.
- Elimina toda frustración por no poder encontrar las piezas: todos los componentes aparecen listados junto a posibles distribuidores.

Construya entre otros estos curiosos dispositivos:

- Traductor de código morse.
- Luz estroboscópica de gran potencia.
- Luz para T.A.E.
- Dado LED.
- Código de seguridad de teclado.
- Monitor de pulsaciones.
- Registrador de datos de temperatura USB.
- Osciloscopio.
- Arpa de luz.
- Termostato de LCD.
- Ventilador controlado por ordenador.
- Detector de mentiras.
- Cerrojo magnético.
- Control remoto por infrarrojos.



www.editorialestribor.com

