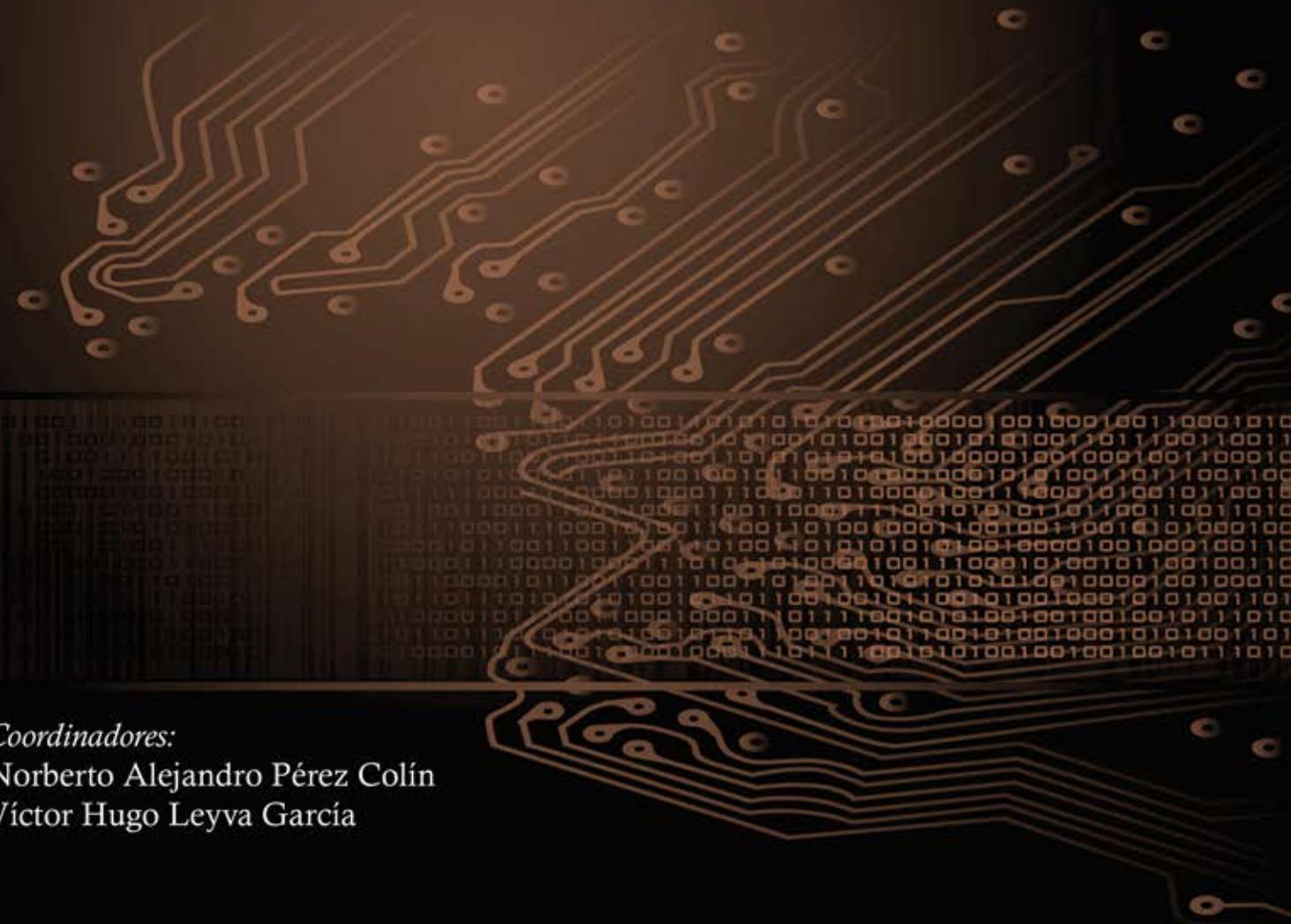


# Guía didáctica de robótica educativa con Arduino



*Aplicaciones para el bachillerato*



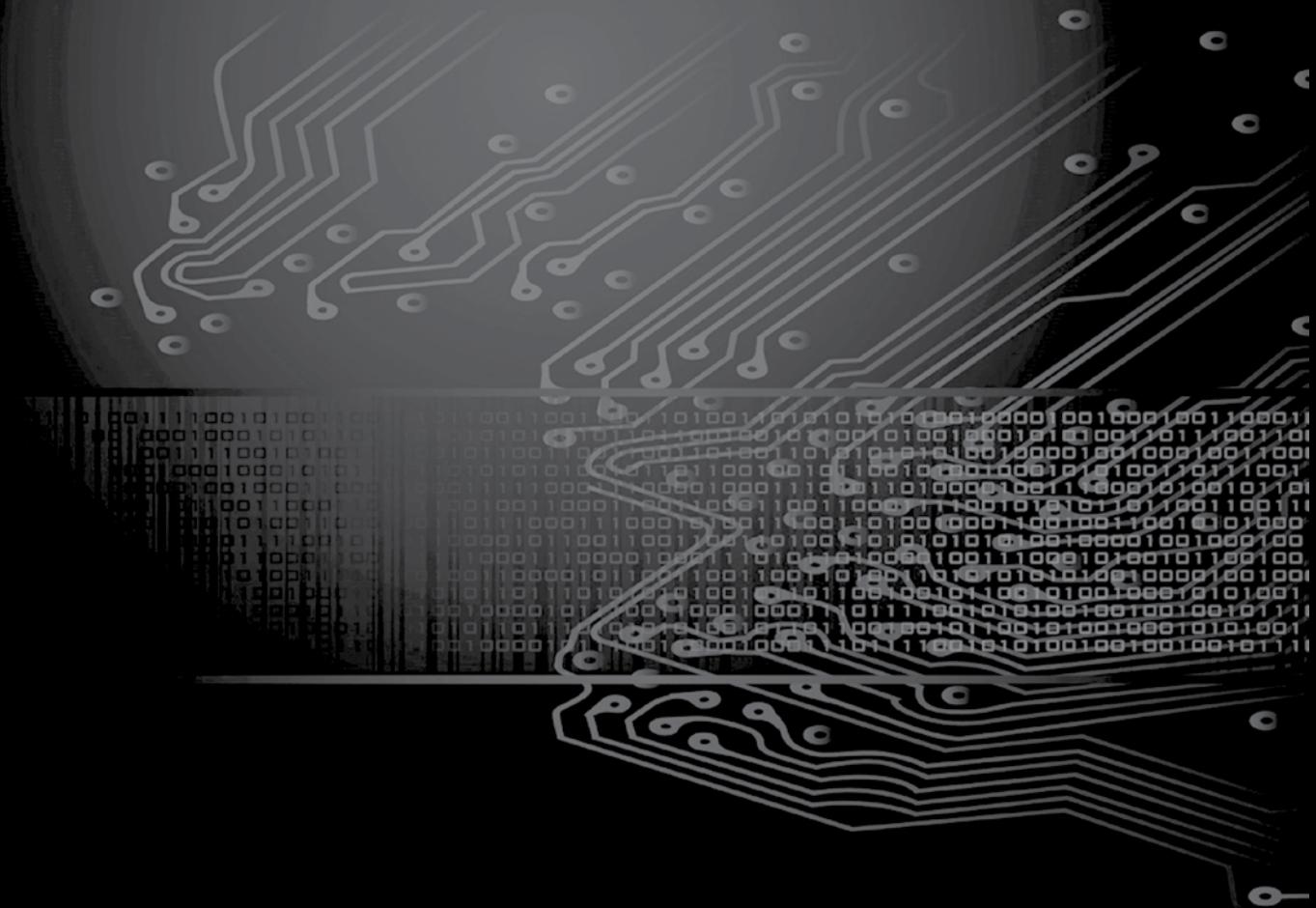
*Coordinadores:*

Norberto Alejandro Pérez Colín  
Víctor Hugo Leyva García

# Guía didáctica de robótica educativa con Arduino



*Aplicaciones para el bachillerato*



**UNIVERSIDAD NACIONAL AUTÓNOMA  
DE MÉXICO**

Dr. Enrique Luis Graue Wiechers  
Rector  
Dr. Leonardo Lomelí Vanegas  
Secretario General  
Ing. Leopoldo Silva Gutiérrez  
Secretario Administrativo  
Dr. Alberto Ken Oyama Nakagawa  
Secretario de Desarrollo Institucional  
Dr. César Iván Astudillo Reyes  
Secretario de Servicios a la Comunidad  
Dra. Mónica González Contró  
Abogada General  
Lic. Néstor Martínez Cristo  
Director General de Comunicación Social

**COLEGIO DE CIENCIAS Y HUMANIDADES**

Dr. Jesús Salinas Herrera  
Director General  
Ing. Miguel Ángel Rodríguez Chávez  
Secretario General  
Lic. José Ruiz Reynoso  
Secretario Académico  
Lic. Aurora Araceli Torres Escalera  
Secretaria Administrativa  
Lic. Delia Aguilar Gámez  
Secretaria de Servicios de Apoyo al Aprendizaje  
Mtra. Beatriz A. Almanza Huesca  
Secretaria de Planeación  
C. D. Alejandro Falcón Vilchis  
Secretario Estudiantil  
Dr. José Alberto Monzoy Vásquez  
Secretario de Programas Institucionales  
Lic. María Isabel Gracida Juárez  
Secretaria de Comunicación Institucional  
M. en I. Juventino Ávila Ramos  
Secretario de Informática

**Directores de los planteles**

Lic. Sandra Aguilar Fonseca  
Azcapotzalco  
Dr. Benjamín Barajas Sánchez  
Naucalpan  
Mtro. José Cupertino Rubio Rubio  
Vallejo  
Lic. Víctor Efraín Peralta Terrazas  
Oriente  
Mtro. Luis Aguilar Almazán  
Sur

*Guía didáctica de robótica educativa con Arduino.*

*Aplicaciones para el bachillerato*

es una publicación editada por el Colegio de Ciencias y Humanidades.

Ciudad Universitaria, 04510, Ciudad de México

Teléfonos: 5622 2499 ext. 393

Coordinadora editorial: Mtra. Ma. Elena Pigenutt Galindo

Formación: Lic. Mayra Monroy Torres y Lic. Verónica Espinosa Mata

Corrección: Fernando Velasco Gallegos y Lilia Cervantes Arias

Apoyo didáctico: Lizbeth Díaz Ayala

Fotografía: Patricia Becerril Coahuilazo

**DEPARTAMENTO DE ACTIVIDADES EDITORIALES  
SECRETARÍA DE SERVICIOS DE APOYO AL APRENDIZAJE  
COLEGIO DE CIENCIAS Y HUMANIDADES**

# **Guía didáctica de robótica educativa con Arduino**

*Aplicaciones para el bachillerato*

*Coordinadores:*

Norberto Alejandro Pérez Colín  
Víctor Hugo Leyva García

Proyecto INFOCAB PB 102313 de la Dirección General de Asuntos del Personal Académico

*Guía didáctica de robótica educativa con Arduino.  
Aplicaciones para el bachillerato*  
es una publicación auspiciada por la DGAPA,  
con el Proyecto INFOCAB PB 102313

Participantes:  
Norberto Alejandro Pérez Colín  
Víctor Hugo Leyva García  
Alberto Ávila Ramos  
Carmina Cecilia Espinosa Madrigal  
Clemente Efraim Ruiz Sánchez  
Darío Eduardo Rodríguez Palacios  
Edith López Martínez  
Érik Pineda Olvera  
Flor Clara Cubillas Hernández  
Gamar Zaid Joseph García Castillo  
José Miguel Baltazar Gálvez  
Juventino Ávila Ramos  
Libia Vargas Olvera

Primera edición: 7 de marzo de 2016.

DR © 2016 UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO  
Ciudad Universitaria, Delegación Coyoacán, CP 04510, Ciudad de México  
**ISBN para edición electrónica: 978-607-02-7760-3**

Esta edición y sus características son propiedad de la  
Universidad Nacional Autónoma de México.

Prohibida la reproducción total o parcial por cualquier medio  
sin la autorización escrita del titular de los derechos patrimoniales.  
Impreso y hecho en México.

# Índice

Presentación• 7

Instructivo• 8

Actividad 1. Primeros pasos en el manejo y comprobación del funcionamiento del microcontrolador Arduino• 9

Actividad 2. *Pin* 13 intermitente• 22

Actividad 3. *LED* intermitente y resistencia• 28

Actividad 4. Control de encendido de un *LED* a través de un interruptor (*push button*) conectado a una entrada digital• 37

Actividad 5. Interfaz de nave espacial• 44

Actividad 6. Lectura analógica y salida *PWM*• 53

Actividad 7. *LED* multicolor• 61

Actividad 8. Control de luminosidad con fotorresistencia• 70

Actividad 9. Detector de blanco• 78

Actividad 10. Detector de blanco con acoplamiento de corriente alterna• 85

Actividad 11. *Fading*• 91

Actividad 12. Molino activado con transistor• 97

Actividad 13. Semáforo basado en un potenciómetro• 109

Actividad 14. *Mood Cue*• 115

Actividad 15. Medidor VU (vúmetro)• 123

Actividad 16. Reloj de arena digital• 129

Actividad 17. Teclado musical• 135

**Actividad 18. *Color mixing lamp***• **141**

**Actividad 19. Control de bloqueo**• **150**

**Actividad 20. Dado de ledes**• **165**

**Actividad 21. Manejo de pantalla alfanumérica *LCD 16x2***• **179**

**Actividad 22. Arreglos unidimensionales**• **187**

**Actividad 23. Comunicación serial**• **198**

**Actividad 24. La estructura selectiva *switch*** • **207**

**Actividad 25. Secuencia *S.O.S.* en código *Morse*** • **215**

**Actividad 26. *Zoetrope* (máquina estroboscópica)**• **224**

**Actividad 27. Medición de la gravedad terrestre mediante un péndulo simple con Arduino**• **236**

**Evaluación. Modelo de semáforo**• **246**

**Solución de la evaluación. Modelo de semáforo**• **248**

## Presentación

En el Plan de Desarrollo Institucional 2011-2015 de la Universidad Nacional Autónoma de México se indica el compromiso institucional de impulsar el uso de las nuevas tecnologías de la información y la comunicación, para fortalecer la preparación y el desempeño escolar de los alumnos (UNAM, 2012; p. 16).

En este sentido, la *Guía didáctica de robótica educativa con Arduino* es una compilación de materiales impresos y multimedia que introducirá a los alumnos del bachillerato en el proceso de concepción, diseño y construcción de mecanismos robóticos. Los materiales de la Guía pretenden involucrar a los estudiantes en un proceso de aprendizaje heurístico de resolución de problemas que les permita, por medio de actividades guiadas, el aprendizaje inductivo, autónomo, independiente y permanente, motivándolos a tomar parte activa en la adquisición de aprendizajes científicos, matemáticos e informáticos inherentes de la robótica educativa, es decir, que los alumnos sean sujetos de su propia cultura, en conformidad con el modelo educativo del CCH.

La *Guía didáctica de robótica educativa con Arduino* es una herramienta digital que integra diferentes actividades guiadas para el diseño y construcción de prototipos robotizados basados en la tecnología Arduino, orientadas para los estudiantes del bachillerato de la UNAM. Estas actividades se fundamentan en el constructivismo, ya que promueven que los alumnos construyan su propio conocimiento a partir de las experiencias guiadas que se plantean, así como con el uso de materiales cognitivos destinados al diseño, construcción, manipulación y control de entornos robotizados.

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en *software* y *hardware* flexibles y de fácil uso e integración; se le puede conectar diversos dispositivos mecánicos, electrónicos, sensores y actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino y el entorno de desarrollo Arduino; el *software* puede de ser descargado de forma gratuita. Los archivos de diseño de referencia están disponibles bajo una licencia abierta, por lo que es posible adaptarlos a las características académicas de los estudiantes del bachillerato de la UNAM.

Norberto Alejandro Pérez Colín  
Víctor Hugo Leyva García

## Instructivo

Diversos profesores e investigadores de nuestra Universidad consideran que la construcción de sistemas robotizados facilita el desarrollo de las potencialidades cognitivas de los alumnos, así como la integración de distintas áreas de conocimiento como matemáticas, física, geometría, química, inteligencia artificial, mecánica, electricidad, electrónica e informática.

La *Guía didáctica de robótica educativa con Arduino* esta distribuida para la realización de las 27 actividades didácticas, cada una promueve el desarrollo de dos tipos de habilidades la primera en programación y la otra en electrónica. Además cuenta con una actividad de evaluación que permite la retroalimentación de los aprendizajes. Por lo tanto, cada actividad didáctica está conformada por dos prácticas: una en programación y la otra en electrónica. Aunque son de diferente área disciplinaria, las actividades didácticas que hemos propuesto permiten crear un prototipo electrónico.

El formato de las prácticas se muestra a continuación y describe en cada rubro su finalidad.

- Nombre de la actividad. Es un nombre relacionado con el prototipo a realizar, por su función o el aprendizaje.
- Antecedentes. Muestra cuáles son los conocimientos previos necesarios para la realización de la actividad.
- Introducción. Muestra el contexto de la actividad.
- Objetivo. Puede ser uno o varios, van relacionados con los aprendizajes a obtener.
- Material utilizado. Nos indica la cantidad y especificaciones de cada dispositivo a utilizar dentro de la actividad.
- Desarrollo. Muestra paso a paso el ensamblado y puesta en marcha del prototipo.
- Diagrama de flujo. Muestra gráficamente el algoritmo o proceso de cada uno de los estados del microcontrolador Arduino y el flujo de la información.
- Código. Es es un conjunto de líneas de texto que son las instrucciones que debe seguir el microcontrolador Arduino para ejecutar los fines del prototipo.
- Resultados y conclusiones. Muestra el análisis de ensamblado y la programación del prototipo.
- Referencias. Indica las referencias consultadas.



## Actividad 1. Primeros pasos en el manejo y comprobación del funcionamiento del microcontrolador Arduino

Norberto Alejandro Pérez Colín / Alberto Ávila Ramos

### Antecedentes

Los avances más significativos que han marcado el ritmo acelerado de la modernidad son fruto de la electrónica. A donde miremos existe un dispositivo electrónico en apoyo de la vida cotidiana, así como del quehacer social, educativo, político, científico, cultural, etcétera. El desarrollo tecnológico nos proporciona medios eficientes para hacer el mínimo esfuerzo en el desarrollo de nuestras actividades: pensemos en un sistema inalámbrico para controlar el funcionamiento de dispositivos diversos o las escaleras eléctricas. El diseño y programación de dichas herramientas tecnológicas se han simplificado, a tal grado que ya no se requieren conocimientos altamente especializados para su desarrollo. Basta tener un nivel básico de conocimientos en programación y electrónica y, por supuesto, muchas ganas de aprender para poder realizar un prototipo robotizado.

A partir de este momento nos introduciremos en el manejo y programación de un microcontrolador de bajo costo conocido con el nombre de Arduino, el cual nos permitirá desarrollar soluciones tecnológicas para aprovechar mejor el mundo que nos rodea.

#### *¿Qué es el microcontrolador Arduino?*

Es una placa electrónica fácil de utilizar, con la que se pueden crear prototipos, basados en *hardware* y *software* libre; el elemento principal es el microcontrolador ATMEGA 328, un circuito integrado programable que tiene la capacidad de ejecutar instrucciones que se encuentran en su memoria.

Arduino toma los datos por las terminales de entrada, conectadas a sensores que son los guías de la electrónica; procesando esta información en un programa se puede controlar todo tipo de motores, luces y actuadores. El *software* utilizado para la programación se puede adquirir en forma libre y gratuita de la página de Arduino <http://www.arduino.cc> (figura 1).

Arduino es un proyecto de *hardware* libre, es decir, podemos construir nuestro propio microcontrolador con ayuda de los diagramas de interconexión, los cuales podemos descargar de Internet.





Figura 1. Página web de Arduino. Imagen obtenida de <http://www.arduino.cc>

Existen varios modelos del microcontrolador Arduino, los cuales presentan diferentes posibilidades de aplicación debido a sus diferentes características de memoria y de procesamiento. Debido a su característica de *hardware libre*, todos los modelos de las placas de Arduino pueden ser hechos a mano o comprados con algún proveedor. En la figura 2 se muestran algunos modelos disponibles en México.



Figura 2. Algunos modelos de placas Arduino y algunas extensiones (*shields*)

El microcontrolador Arduino que vamos a utilizar para nuestras prácticas es el modelo ARDUINO UNO, como se muestra en la figura 3, que tiene las siguientes características: consta de 14 terminales digitales que pueden ser usados como entradas y salidas de la información; posee 6 entradas analógicas, cada una de ellas provee de 10 bits de resolución (1024 valores diferentes). También algunos de estas terminales poseen funciones especiales, las cuales se revisarán en este curso.



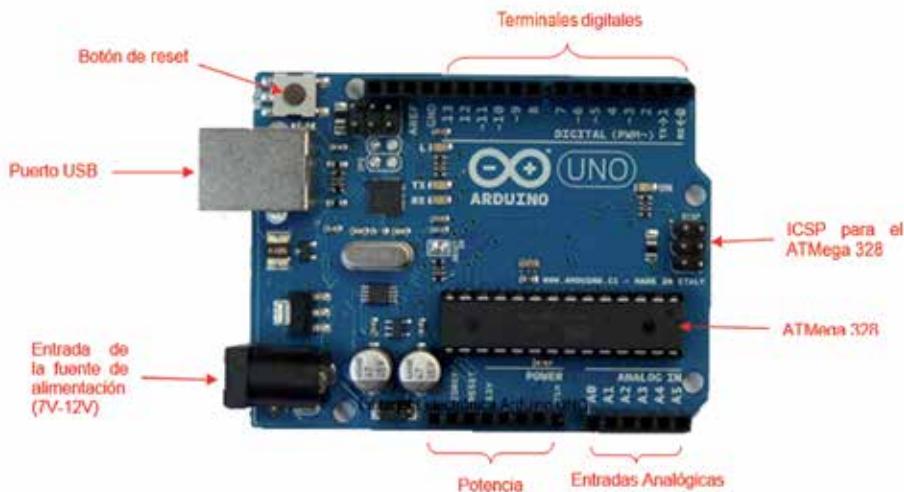


Figura 3. La tarjeta electrónica Arduino UNO y sus componentes internos

### Instalación y configuración del entorno de programación

Para comenzar a utilizar el microcontrador Arduino necesitamos descargar el *software* de programación e instalarlo en un equipo de cómputo; existen varias versiones adecuadas a diferentes sistemas: *Windows*, *Mac* y *Linux*.

Para descargar el *software*, accede a la pestaña Descarga de la página electrónica de Arduino <http://www.Arduino.cc/es/>, en la figura 4 se muestra el sitio web.

Este es un screenshot de la página web oficial de Arduino (<http://www.arduino.cc>). La barra superior muestra el logo de Arduino y una barra de búsqueda. La barra de menú tiene opciones como Compra, Descarga, Principio Puesta, Aprende, Reference, Hardware y FAQ.

### Descarga el Software de Arduino

El entorno de código abierto Arduino hace fácil escribir código y cargarlo a la placa E/S. Funciona en Windows, Mac OS X y Linux. El entorno está escrito en Java y basado en Processing, avr-gcc y otros programas también de código abierto.

El software Arduino se proporciona "COMO ES" Y NO MANIFESTAMOS NI IMPONEMOS GARANTIA CON RESPECTO A SU FUNCIONALIDAD, OPERATIVA O Uso, INCLUIDO, NINGUNA LIMITACION NI IMPLICACION RESPECTO GARANTIAS DE COMERCIALIZACION, ADECUACION PARA PROPÓSITOS PARTICULARES O INFRACCIONES. EXPLÍCATEMENTE NOSE DESvinculan De CUALQUIER CONSECUENCIA, INCIDENTE O DAÑO DIRECTO O INDIRECTO INCLUIDO, SIN LIMITE, PERDIDA DE BENEFICIOS, INTERRUPCIONES DE SERVICIO O PERDIDA DE DATOS, INDEPENDIENTEMENTE DEL FORMATO DE ACCION O TEORIA LEGAL EN QUE SE REALICE DICHA DEMANDA, INCLUSO SI HUBIERA ADVERTENCIA DE LA POSIBILIDAD O SINALDAD DE DICHOS DAÑOS.

Descargando el software desde esta página se aceptan los términos anteriormente citados.

**Descarga**  
Arduino 0019 (notas de la versión), hospedado en Google Code:  
+ Windows  
+ Mac OS X  
+ Linux: 32 bit - comprueba aquí las compatibilidades.

**Siguientes pasos**  
Empezando  
Referencias  
Entorno  
Ejemplos  
Fundamentos  
FAQ

Descarga según el sistema operativo

Figura 4. Página de descarga de Arduino. Imagen obtenida de: <http://www.arduino.cc>



Después de seleccionar el sistema operativo, el navegador nos abrirá una ventana de descarga, como se muestra en la figura 5, para guardar o ejecutar el archivo que contiene el *software* de Arduino.



Figura 5. Página web de Arduino, Descarga del *IDE* de Arduino. Imagen obtenida de: <http://www.arduino.cc>

Una vez que descarguemos el *software* del sitio de Arduino, se ejecuta el entorno de programación (*IDE*, *Integrated development environment*) (ver figura 6); el programa de instalación contiene librerías, ejemplos, herramientas y *drivers*, para su ejecución.

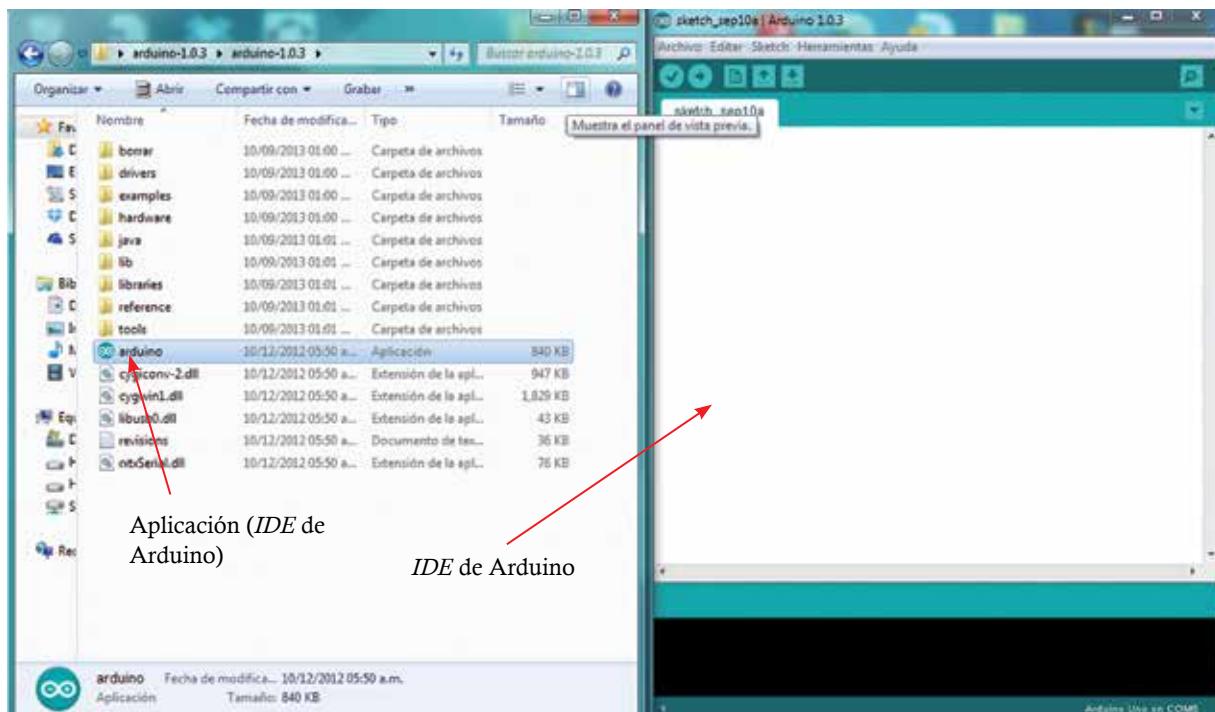


Figura 6. Izquierda. Carpeta con el *software* de Arduino y sus complementos. Derecha *IDE* de Arduino



En la figura 7 se muestran las áreas de trabajo y las herramientas del *IDE*.

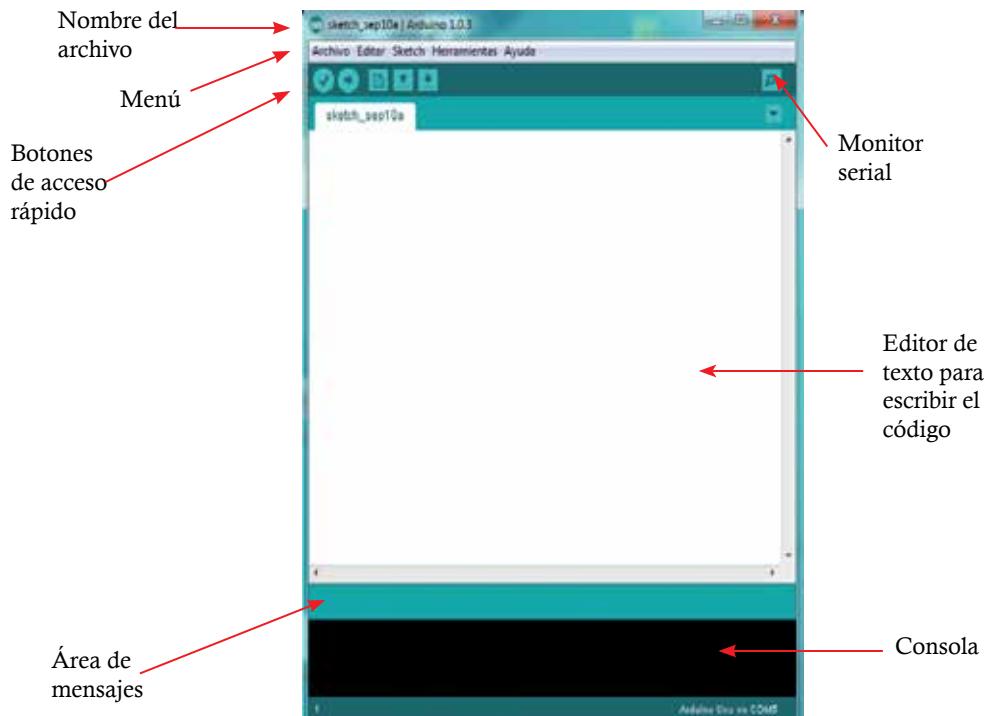


Figura 7. Distribución de las áreas de trabajo en el *IDE* de Arduino

#### Conectando nuestro controlador Arduino a la computadora

El microcontrolador Arduino se conecta mediante un cable *USB* a un puerto de la computadora, tal como se ilustra en la figura 8.



Figura 8. Conexión física de la computadora y la placa Arduino por *USB*



Cuando conectas la placa Arduino, *Windows* debería inicializar la instalación de los controladores (siempre y cuando no hayas utilizado el equipo de cómputo con una placa Arduino anteriormente). En *Windows*, se abrirá un cuadro de diálogo de instalación del nuevo *Hardware*.

Puedes comprobar que los controladores se han instalado correctamente abriendo la carpeta del Administrador del Dispositivos del panel de control del sistema operativo. Busca “*USB Serial Port*” (o Puerto *USB-Serie*) en la sección de puertos (esa es tu placa Arduino).

Una vez instalados los controladores necesitamos en el *IDE* seleccionar el tipo de placa en el menú *Herramientas > Tarjeta*, como se muestra en la figura 9.

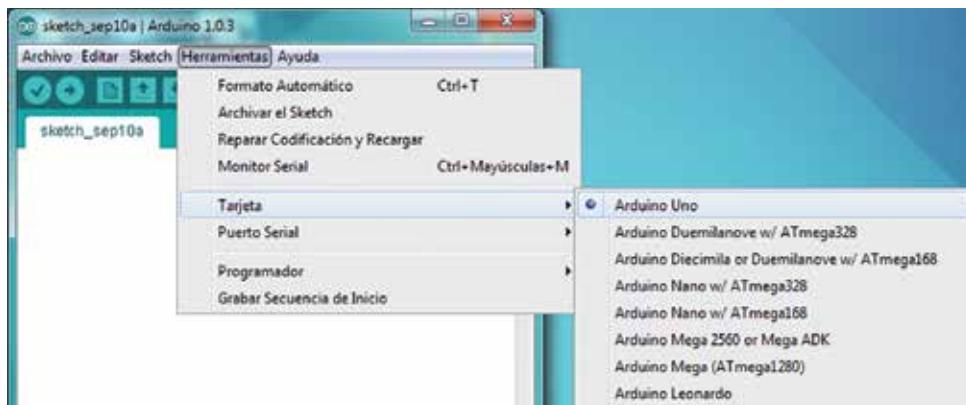


Figura 9. Selección de la tarjeta de trabajo Arduino en el *IDE*

El último paso para trabajar con la placa Arduino es seleccionar en el *IDE* el dispositivo serial de la placa Arduino en el menú *Herramientas>Puerto serial*, ver figura 10. Lo más probable es que sea *COM3* o mayor (*COM1* y *COM2* se reservan, por regla general para puertos serial de *hardware*). Abriendo la carpeta del Administrador del Dispositivos, del panel de control del sistema operativo. Busca “*USB Serial Port*”, nos indicará cuál es el puerto correcto.



Figura 10. Selección del puerto serial de comunicación con la tarjeta Arduino en el *IDE*

Para la programación de Arduino visita la *referencia extendida* en la página <http://arduino.cc/es/Reference/HomePage>, ver figura 11.



Los programas hechos con Arduino se dividen en tres partes principales: estructura, valores (variables y constantes) y funciones. El lenguaje de programación Arduino se basa en C/C++.



## Referencia del Lenguaje

Visita la [referencia extendida](#) para características más avanzadas del Lenguaje Arduino y la [página de librerías](#) para estudiar cómo conectar con distintos tipos de hardware.

Los programas hechos con Arduino se dividen en tres partes principales: estructura, valores (variables y constantes), y funciones. El Lenguaje de programación Arduino se basa en C/C++.

| Estructura  | Variables                         | Funciones                          |
|---|-----------------------------------|------------------------------------|
| + <code>setup()</code> (inicialización)               | <b>Constantes</b>                 | E/S Digitales                      |
| + <code>loop()</code> (bucle)                         | + <code>HIGH   LOW</code>         | + <code>pinMode()</code>           |
| <b>Estructuras de control</b>                         | + <code>INPUT   OUTPUT</code>     | + <code>digitalWrite()</code>      |
| + <code>if (comparador si...entonces)</code>          | + <code>true   false</code>       | + <code>digitalRead()</code>       |
| + <code>if...else (comparador si...sino)</code>       | <b>Constantes Numéricas</b>       |                                    |
| + <code>for (bucle con contador)</code>               | <b>Tipos de Datos</b>             |                                    |
| + <code>switch case (comparador multiple)</code>      | + <code>boolean (booleano)</code> | <b>E/S Analógicas</b>              |
| + <code>while (bucle por comparación booleana)</code> | + <code>char (carácter)</code>    | + <code>analogRead()</code>        |
|   | + <code>byte</code>               | + <code>analogWrite() - PWM</code> |
|   |                                   | (modulación por ancho de pulso)    |
|   |                                   | <b>E/S Avanzadas</b>               |

Figura 11. Vista de la [referencia extendida](#)

### Primer paso: verificación del buen funcionamiento de nuestra tarjeta Arduino

El microcontrolador Arduino es una placa electrónica utilizada para la creación de prototipos electrónicos con una filosofía de *software* y *hardware* flexibles y fáciles en su manejo.

En esta práctica vamos a realizar la prueba de comprobación de nuestro Arduino, con el fin de familiarizarnos con el interfaz de desarrollo, además de verificar el buen funcionamiento de nuestra placa.

En los lenguajes de programación y desarrollo casi siempre el profesor nos muestra como primer programa la impresión en pantalla de “HOLA MUNDO”. Para nuestra verificación del correcto funcionamiento de nuestra tarjeta Arduino, haremos parpadear un LED.

La placa Arduino UNO está diseñada con un LED incorporado directamente en la placa, y conectado al pin 13 a través de una resistencia de 1 KΩ.



## Objetivos

- Describirá las características generales del *hardware* y *software* del microcontrolador Arduino.
- Diferenciará los pines digitales, analógicos y de alimentación eléctrica, del microcontrolador Arduino.
- Identificará los componentes físicos del microcontrolador Arduino.
- Entenderá cómo el *hardware* y *software* del microcontrolador Arduino interactúan.

## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A a conexión B).
- Microcontrolador Arduino UNO.

## Desarrollo



Figura 12. Materiales y equipo utilizado

Interconectar el equipo de cómputo, el cable *USB* y el microcontrolador Arduino, de la figura 12.

Realizar la instalación de los controladores en la computadora (siempre y cuando no ha utilizado el equipo de cómputo con una placa Arduino anteriormente). En *Windows Vista*, *Windows 7* y *Windows 8*, los controladores deberían descargarse e instalarse automáticamente. En *Windows XP*, se abrirá un cuadro de diálogo de instalación del nuevo *hardware*.

Se puede comprobar que los controladores se han instalado correctamente abriendo la carpeta del Administrador del Dispositivo, del panel de control del sistema operativo. Busca “*USB Serial Port*” (o *Puerto USB-Serie*) en la sección de puertos (esa es tu placa Arduino).

Necesitamos en el *IDE* seleccionar el tipo de placa en el menú Herramientas > Tarjeta. En nuestro caso buscaremos la tarjeta UNO.



Seleccionar en el *IDE* el dispositivo serial de la placa Arduino en el menú Herramientas>Puerto serial. Lo más probable es que sea COM3 o mayor (COM1 y COM2 se reservan, por regla general para puertos *serial de hardware*). Abriendo la carpeta del Administrador del Dispositivos, del panel de control del sistema operativo. Busca “*USB Serial Port*”, que nos indicará cuál es el puerto correcto.

Una vez realizada la revisión anterior, buscaremos el código en el *IDE* de Arduino en el menú Archivo > Ejemplos > 01.Basics > Blink, como lo muestra la figura 13.

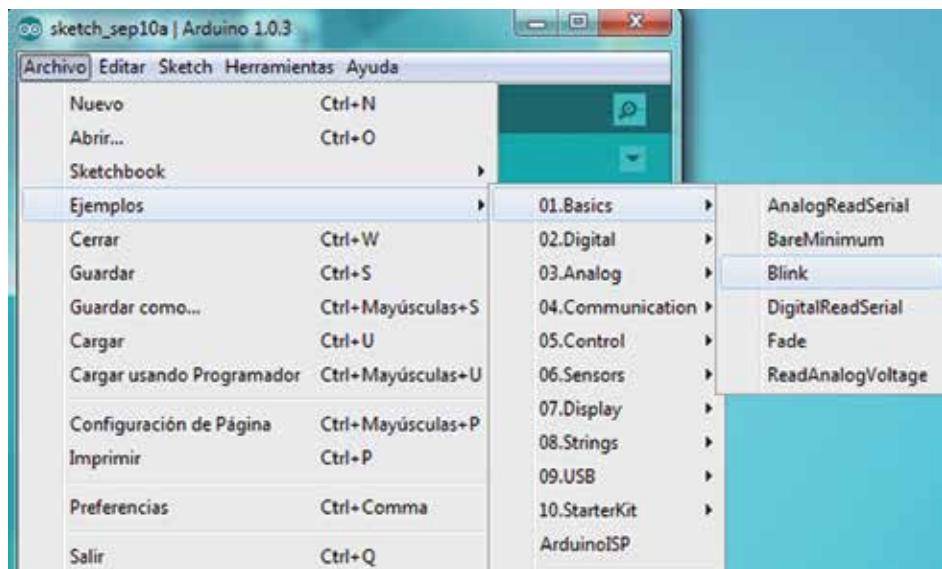


Figura 13. Selección del ejemplo “*Blink*” en el *IDE* de Arduino

Una vez que se tiene el código del programa “*Blink*”, en el *IDE*, procederemos a verificar si no hay error en la compilación y después a cargar el programa, como se muestra en la figura 14.



Figura 14. Compilación, carga el programa en el *IDE* de Arduino



Observemos el *LED* sobre la tarjeta Arduino UNO cómo comienza a parpadear continuamente. Si cambiamos el valor del retardo, podríamos observar espacios de tiempo más o menos cortos, según sea el caso.

## Circuito

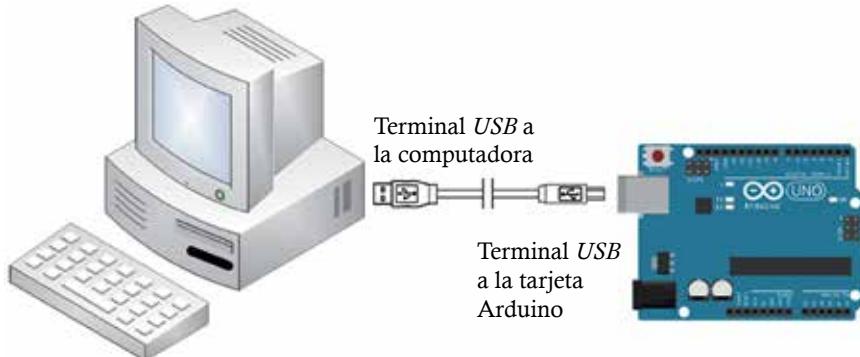
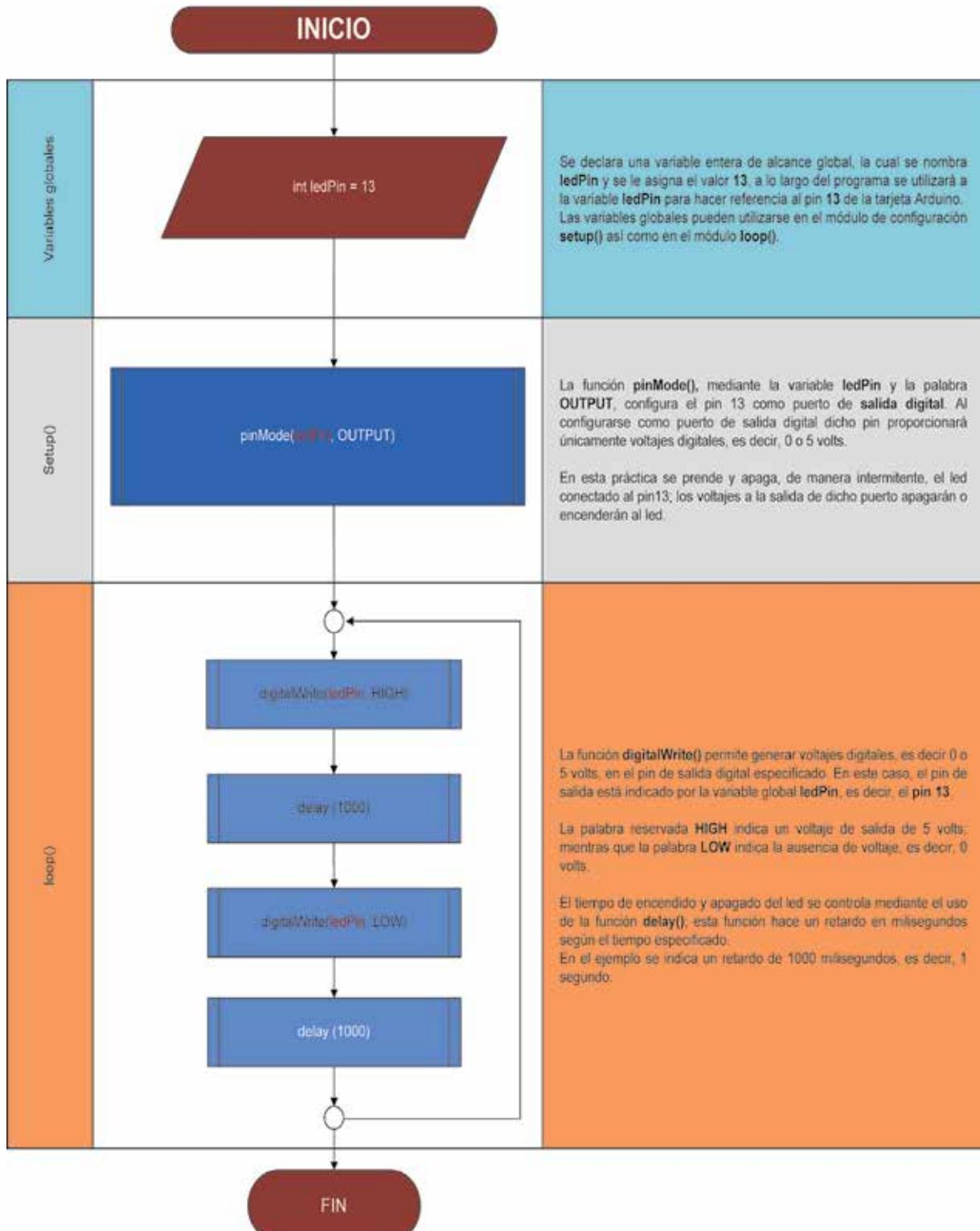


Figura 15. Conexión del equipo de cómputo, tarjeta Arduino por cable *USB*



## Diagrama de flujo



## Código

|                                   |  |
|-----------------------------------|--|
| Declaración de variables globales | <pre>/* Intermitencia de encendido y apagado de un LED  Se enciende y se apaga un LED; los tiempos de encendido y apagado son de un segundo  */ // el pin 13 está conectado a un LED integrado en la tarjeta Arduino UNO  int ledPin = 13;  // la función de configuración setup() se ejecuta una sola vez</pre>   |
| setup()                           | <pre>void setup() {      // se configura el pin 13 (ledPin) como pin de salida     pinMode(ledPin, OUTPUT); }</pre>  |
| loop()                            | <pre>// la función loop() se ejecuta cíclicamente y de manera ininterrumpida  void loop() {      digitalWrite(ledPin, HIGH); // enciende el LED (HIGH indica 5 volts de salida en el pin 13)     delay(1000); // retardo de un segundo     digitalWrite(ledPin, LOW); // se apaga el LED (LOW indica la ausencia de voltaje)     delay(1000); // retardo de un segundo }</pre> |



## Resultados y conclusiones

Se demuestra con esta práctica que nuestra tarjeta Arduino UNO se encuentra trabajando en óptimas condiciones, además podemos controlarla desde nuestra computadora mediante el *IDE* de Arduino.

Se cumplieron los propósitos de la práctica sobre el manejo del microcontrolador Arduino, donde la inducción al mundo Arduino es esencial para las otras prácticas, que es la intención fundamental de llamarla "Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino".

Se describieron las características generales del *hardware* y *software*, así como las diferencias entre los pines digitales, analógicos y de alimentación eléctrica, se pueden identificar correctamente los componentes físicos del microcontrolador Arduino.

Se entiende cómo el *hardware* y *software* del microcontrolador Arduino interactúan entre sí para poder realizar el control.

## Referencias

Getting Started with Arduino. (s.f.). Arduino LLC. Recuperado el 30 de agosto de 2013, de <http://arduino.cc/en/Guide/HomePage>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Arduino Projects Book*, Italia: Arduino LLC.



## Actividad 2. Pin 13 intermitente

Norberto Alejandro Pérez Colín / Alberto Ávila Ramos

### Antecedentes

*¿Por qué la placa Arduino?*

Arduino es una herramienta que permite controlar el mundo físico. Se puede utilizar para crear prototipos electromecánicos y en sistemas de automatización, leyendo datos de interruptores y sensores, para controlar luces, motores y actuadores. Algunas ventajas al trabajar con Arduino son su accesibilidad en el costo, su funcionamiento en diferentes sistemas operativos, su programación sencilla y el software sin licenciamiento.

*¿Qué es una señal digital?*

Es una señal generada por un fenómeno electromagnético, en donde el valor que toma cada magnitud codifica el contenido de ésta. Veamos la figura 1.

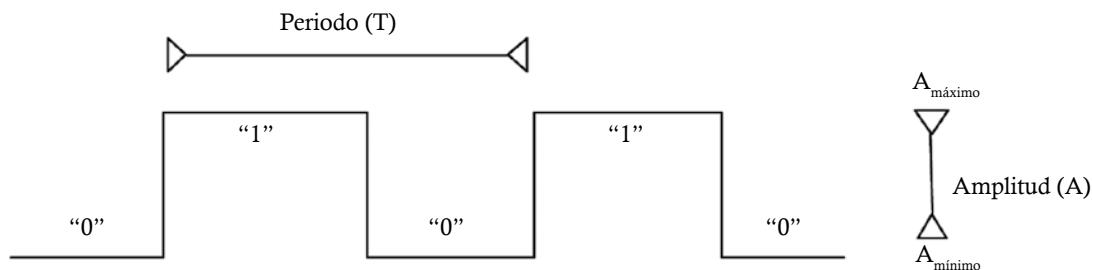


Figura 1. Señal digital

Las magnitudes que representan valores discretos, son los valores de  $A_{\text{máximo}}$  que nos representa un “1” o encendido y el valor de  $A_{\text{mínimo}}$  que nos representa un “0” o apagado. Por ejemplo, el interruptor de la luz de nuestras casas sólo puede tomar dos valores o estados: abierto o cerrado, o la misma lámpara: encendida o apagada.

Los sistemas digitales, como la computadora, usa lógica de dos estados representados por dos niveles de voltaje eléctrico, uno alto, H y otro bajo, L (*High* y *Low*, respectivamente, en inglés). Estos estados se sustituyen por ceros y unos, lo que facilita la aplicación de la lógica y la aritmética binaria.



## Introducción

En esta práctica vamos a conocer cómo es el funcionamiento de la salida digital (*pin 13*) de nuestro microcontrolador Arduino UNO. Para esto utilizaremos un diodo emisor de luz (*LED*, por sus siglas del inglés) conectado a una salida digital de nuestra tarjeta Arduino.

El *LED* es un componente semiconductor, utilizado como indicador luminoso en muchos dispositivos y en iluminación.

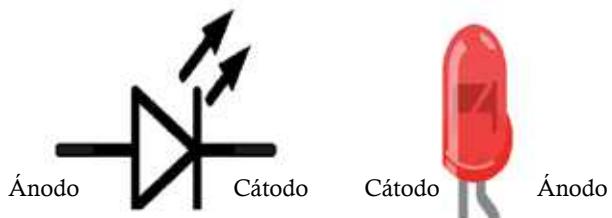


Figura 2. Símbolo eléctrico (izquierda) y *LED* rojo (derecha), imagen obtenida del software Fritzing

Principales formas de conocer la polaridad de un *LED*:

1. La pata más larga es el ánodo.
2. En el lado del cátodo, la base del *LED* tiene un borde plano.
3. Dentro de la capsula del *LED*, la placa indica el ánodo. Se puede reconocer porque es más pequeña que el yunque, que indica el cátodo.

Para la conexión de un *LED* veamos la figura 3, en donde debemos tomar en cuenta que debe estar polarizado directamente, es decir, el polo positivo de la fuente de alimentación conectado al ánodo y el polo negativo conectado al cátodo. Además, la fuente de alimentación debe suministrar un voltaje superior a su tensión umbral (voltaje mínimo de encendido). Y se debe garantizar que el voltaje no exceda los límites admisibles, lo que dañaría irreversiblemente al *LED*. (Esto se puede hacer de manera sencilla colocando una resistencia eléctrica en serie con el *LED*.)

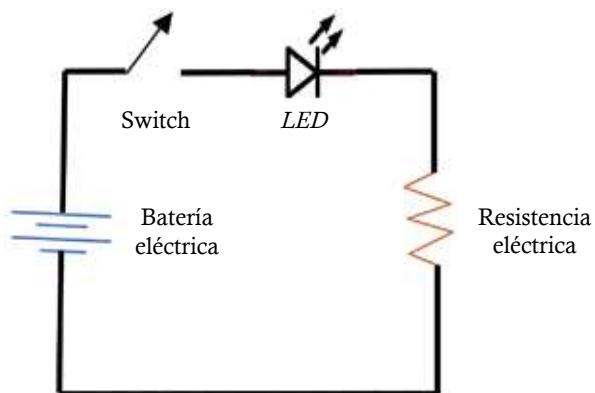


Figura 3. Circuito básico de conexión de un *LED*



Las ventajas de utilizar Diodos emisores de Luz son, principalmente, el bajo consumo de energía, tiempo de vida, tamaño reducido, durabilidad y reducción en la emisión de calor; por su versatilidad se pueden incorporar a todas las tecnologías de iluminación en un porcentaje mayor a 90 por ciento.

## Objetivos

- Describirá las características generales del *hardware* y *software* del microcontrolador Arduino.
- Diferenciará e identificará las terminales digitales, los analógicos y las terminales de alimentación eléctrica del microcontrolador Arduino.
- Entenderá cómo el *hardware* y *software* del microcontrolador Arduino interactúan a través de las terminales digitales.

## Material utilizado

- Equipo de cómputo.
- Cable estándar USB (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- *LED* de cualquier color.

## Desarrollo

Conecta el ánodo del *LED* en el *pin* 13 del Arduino, y el cátodo al *pin* de tierra (*GND*), como se muestra en la figura 4.

El *pin* 13 se configurará a través del código de programación (ver código en la sección correspondiente), como un *pin* de salida digital, por lo que proveerá una señal digital. Dicha señal digital proporcionará, cíclicamente, un voltaje de 5 volts (nivel *ALTO* o *HIGH*) durante cierto tiempo o, dicho en otras palabras, durante medio ciclo de la señal; después proveerá de un voltaje 0 volts (nivel *BAJO* o *LOW*) durante el restante medio ciclo. Esto significa que en el ánodo del *LED* estará a niveles *BAJOS* y *ALTOS* de voltaje, es decir, estará sometido a 0 volts y a 5 volts proporcionados por el *pin* 13.

Cuando en el *pin* 13 se tenga un nivel *ALTO*, entonces el *LED* encenderá y cuando en dicho *pin* se tenga un nivel bajo, entonces el *LED* se apagará.

El tiempo que el *LED* se mantenga encendido o apagado estará determinado en el código del programa a través de la función *DELAY()*. Esta función recibe como argumento un valor entero que corresponde a una pausa que hará el programa en milisegundos, es decir, nos indica cuánto tiempo permanecerá el *LED* encendido o apagado.

Ahora bien, observa que no se ha conectado una resistencia para limitar la corriente que atraviesa el *LED* y evitar que éste se dañe, como se indicó en la introducción. Esto se debe a que, internamente, el Arduino ya posee una resistencia de 1 KΩ conectada en el *pin* 13, la cual nos permite omitir la resistencia en serie que, en otras terminales, sí debemos conectar para proteger nuestro *LED*.



1. Conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.
  2. Ejecuta el *software* de programación de tu Arduino. Carga el programa de ejemplo “*Blink*” y cárgalo en tu Arduino. El código se explica en la siguiente sección.

El programa “*Blink*” lo encontrarás accediendo en el menú Archivo>Ejemplos>01. *Basics>Blink*.

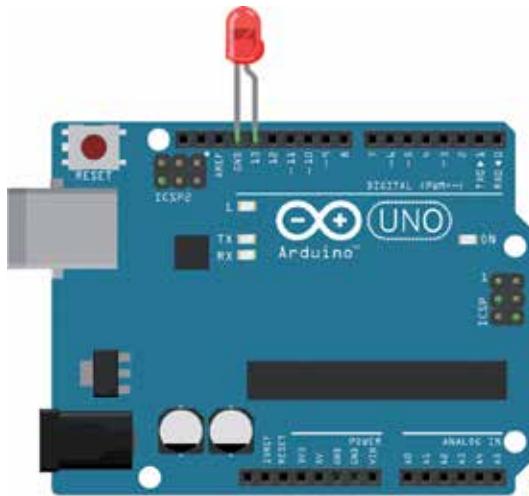
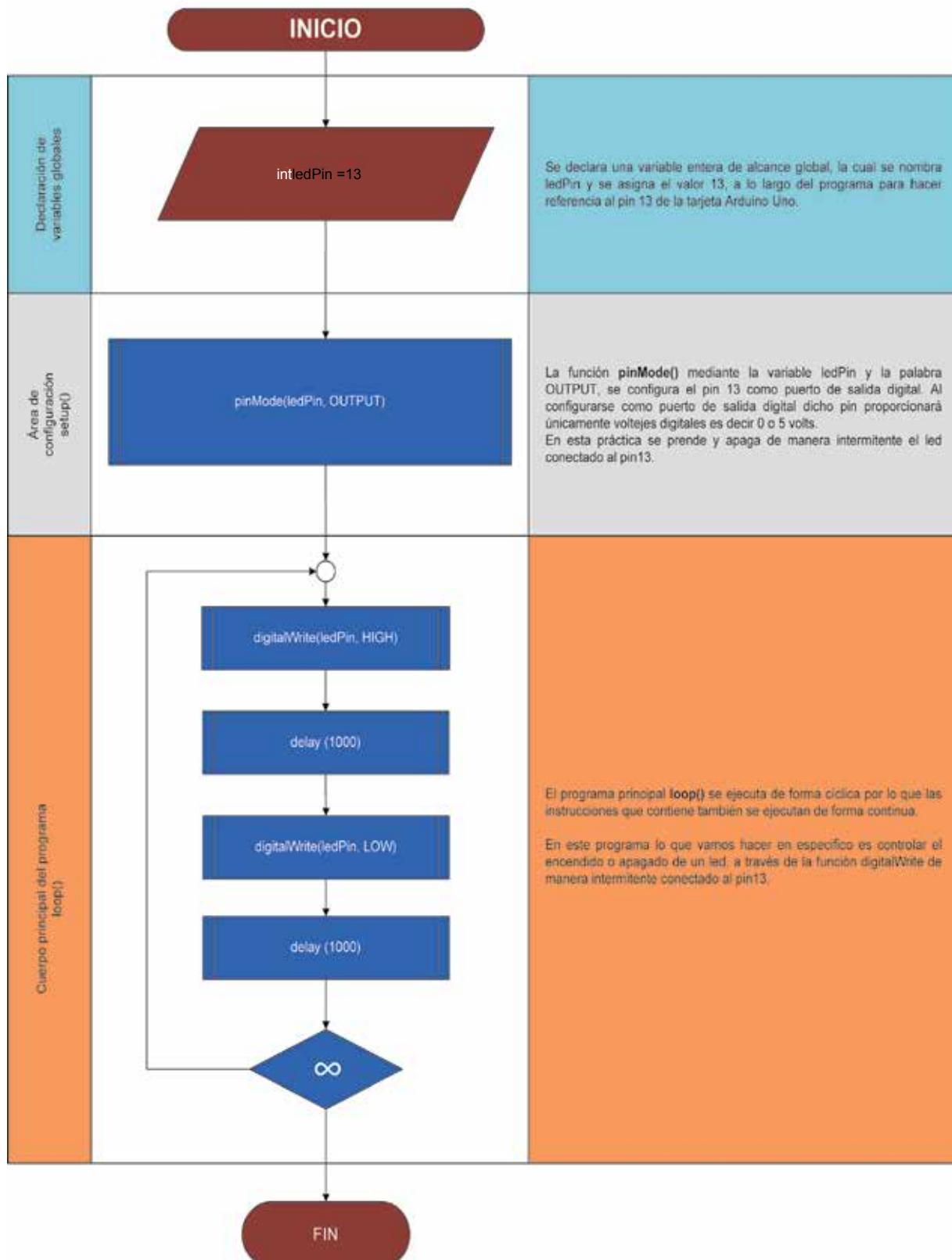


Figura 4. Esquema de conexión de un LED al pin 13 del Arduino. Imagen obtenida del software Fritzing

3. Observarás que el *LED* prende y apaga de manera intermitente debido a que el *pin* 13 proporciona una señal digital, donde los niveles de voltajes ALTO y BAJO se alternan; el tiempo de encendido y apagado se determina en el código del programa.



## Diagrama de flujo "Pin 13 intermitente"



## Código

|                                   |   |
|-----------------------------------|---|
| Declaración de variables globales | <pre>/* Intermitencia de encendido y apagado de un LED. Se enciende y se apaga un LED; los tiempos de encendido y apagado son de un segundo. */</pre>   |
| setup()                           | <pre>// el pin 13 está conectado a un LED integrado en la tarjeta Arduino UNO int ledPin = 13; // la función de configuración setup() se ejecuta una sola vez void setup() {     // se configura el pin 13 (ledPin) como pin de salida     pinMode(ledPin, OUTPUT); }</pre>   |
| loop()                            | <pre>// la función loop() se ejecuta ciclicamente y de manera ininterrumpida void loop() {     // enciende el LED (HIGH indica 5 volts de salida en el pin 13)     digitalWrite(ledPin, HIGH);     // retardo de un segundo     delay(1000);     // se apaga el LED (LOW indica la ausencia de voltaje)     digitalWrite(ledPin, LOW);     // retardo de un segundo      delay(1000); }</pre> |

## Resultados y conclusiones

Se demostró en esta práctica el funcionamiento de las terminales digitales y, específicamente, el *pin* 13 de nuestra tarjeta Arduino UNO, además de conocer el funcionamiento y manipulación de un *LED* mediante el *IDE* de Arduino.

Se describieron las características generales del *hardware* y *software* utilizado del microcontrolador Arduino, así como las de un *LED*, su simbología y la identificación física.

## Referencias

Blink. (s.f.). *Arduino LLC*. Recuperado el 5 de septiembre de 2013, de <http://arduino.cc/en/Tutorial/Blink>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). Get to know your tools. *Arduino Projects Book* (pp. 21-31). Italia: Arduino LLC.



## Actividad 3. LED intermitente y resistencia

Norberto Alejandro Pérez Colín / Alberto Ávila Ramos

### Antecedentes

*¿Qué es una protoboard?*

Es una tableta de material aislante, que contiene en su parte superior orificios interconectados por un material conductor entre sí, siguiendo patrones de líneas, donde pueden insertar componentes electrónicos y cables sin necesidad de soldarlos; es una herramienta muy útil en el armado de prototipos, también llamada placa de pruebas, por su facilidad y sencillez en su manejo de los circuitos electrónicos.



Figura 1. Aspecto real de la *protoboard*

En la figura 1 se muestra su aspecto físico, aunque se puede ilustrar mejor su operación en forma esquemática; en la figura 2 se muestra a la derecha cómo está constituida internamente.

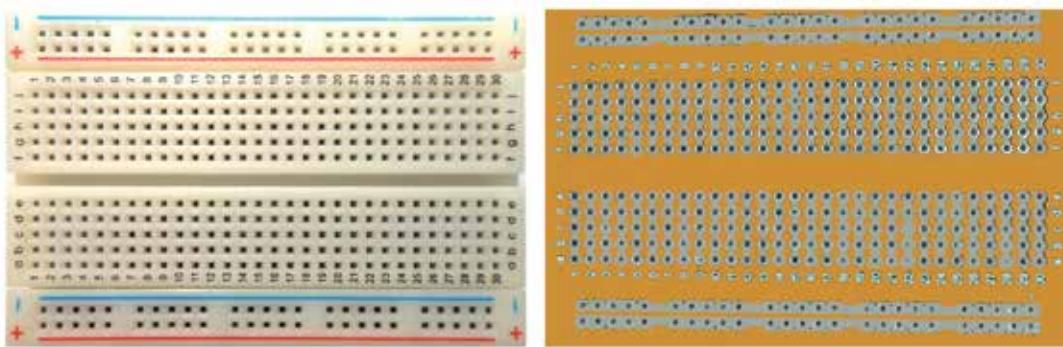


Figura 2. *Protoboard* físicamente (izquierda) y *protoboard* internamente (derecha)

La *protoboard* tiene una distribución en tres áreas, a los lados áreas de polarización y en el área central se ordena en forma de una matriz de puntos, por filas y columnas, donde las filas se encuentran nombradas por letras y las columnas por números.



El área de polarización está compuesta de dos nodos, un nodo para la parte positiva (+) y otro para la parte negativa (-), los nodos tienen una forma horizontal respectivamente. En la figura 2 de la derecha se pueden observar las dos laminillas de material conductor que componen esta área.

En el área central de la *protoboard* hay nodos comunes, formando una matriz de puntos, las columnas están compuestas de números y las filas ordenadas por letras; un ejemplo de la interconectividad en nuestra *protoboard* es el nodo compuesto por la columna 1 y las filas *abcde* y otro nodo lo compone el mismo número de columna con las filas *fghij*, causa por la cual existe un canal de división entre los grupos de letras. En la figura 2 de la derecha se pueden observar las laminillas de material conductor y su distribución en forma vertical.

En la periferia de la placa existen salientes y entrantes para que se puedan adaptar dos o más tabletas en forma plana para hacer un arreglo mayor.

### *¿Por qué una resistencia después de un LED?*

En la práctica anterior se trabajó con el circuito de la figura 3, donde se puede observar un *LED* acoplado con una resistencia eléctrica; en todos los casos éste es el circuito básico de conexión de un *LED*. Significa que cuando instalemos un *LED*, por regla, instalaremos una resistencia eléctrica. Esto con el fin específico de no producir un daño al *LED*, ya que podríamos quemarlo.

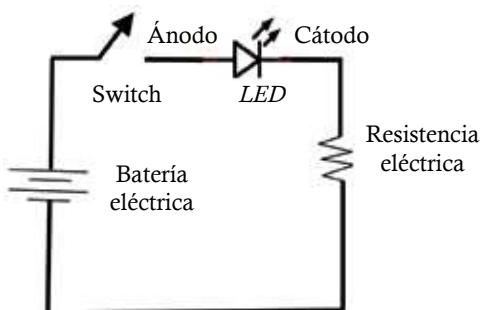


Figura 3. Circuito básico de conexión de un *LED*

Por lo tanto, y en forma redundante, cuando se conecte un *LED* en cualquier práctica de Arduino incorporaremos al circuito una resistencia. El cálculo de la resistencia lo indicaremos a continuación, basándonos en la figura 4.

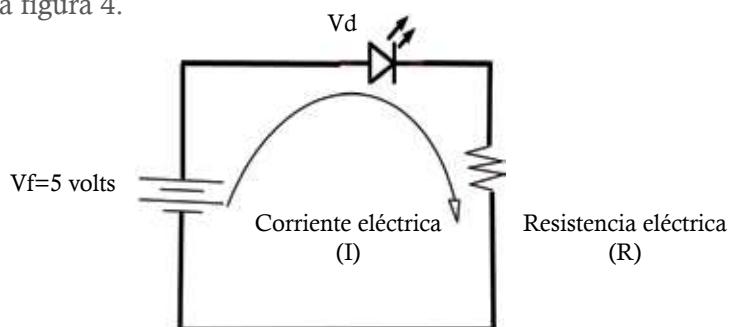


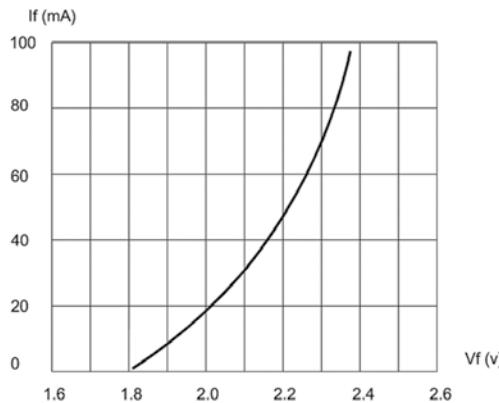
Figura 4. Circuito electrónico del *LED*



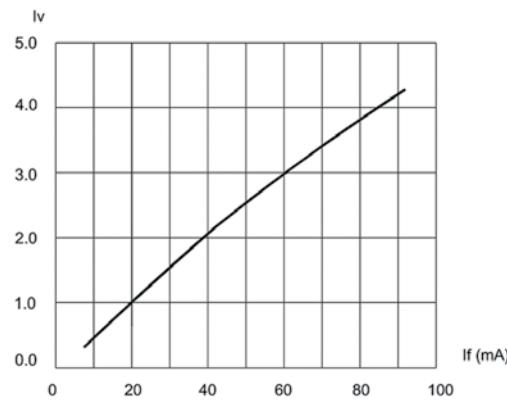
Cerrando el *switch*, se creará un flujo de corriente eléctrica a través del circuito electrónico con una batería de 5 voltios, los cuales son proporcionados por la placa Arduino.

El voltaje en cada *LED* varía de acuerdo con las especificaciones relacionadas con el color y la potencia soportada. La corriente eléctrica determinará el funcionamiento óptimo del *LED* siempre y cuando se encuentre dentro de los voltajes de trabajo, como se muestra en la figura 5 (izquierda), donde el mínimo de voltaje es 1.8 volts y el máximo 2.3 volts.

La corriente eléctrica determinará el valor para la intensidad luminosa que necesitamos. Lo común es de 10 mA para *LED* con baja luminosidad y 90 mA para *LED* con alta luminosidad, como se puede observar en la figura 5 (derecha).



Corriente directa vs voltaje



Intensidad luminosa vs corriente

| Características típicas eléctricas y ópticas (T=25°C) |                       |        |        |                   |
|---|-----------------------|--------|--------|-------------------|
| Información   | Condición             | Mínimo | Máximo | Unidad            |
| Voltaje directa                                       | $I_F = 20 \text{ mA}$ | 1.8    | 2.4    | Volt              |
| Corriente directa                                     | $V_R = 5V$            | -----  | 10     | $\mu\text{Amper}$ |
| Intensidad luminosa                                   | $I_F = 20 \text{ mA}$ | 1700   | 3500   | milicandela       |

Figura 5. Gráfica de características eléctricas (izquierda) y gráfica de características ópticas (derecha),

Obtenidas de <http://www.casadelled.com.ar/ZZ-GY-0001B,ZL503RCA2.pdf>

Mostraremos un ejercicio para la obtener la resistencia ( $R$ ), donde tomaremos el *LED* rojo, con el voltaje de trabajo de 1.8 volts mínimo, con  $I = 15 \text{ mA}$  que es una corriente baja y utilizando la ley de *Ohm*, aplicada a nuestro circuito.

$$R = \frac{(V_d - V_f)}{I}$$

Sustituyendo valores resulta:

$$R = \frac{(5 \text{ V} - 1.8 \text{ V})}{15 \text{ mA}} = 213 \Omega$$

Es conveniente utilizar el valor estándar del mercado superior para mayor seguridad, en este caso en el mercado encontraremos la de  $220 \Omega$ .

## Introducción

En esta práctica vamos a conocer cómo es el funcionamiento de las salidas digitales de nuestro microcontrolador Arduino UNO. Para esto utilizaremos un diodo emisor de luz, conectado a alguna de las salidas digitales de nuestra tarjeta Arduino.

Para la conexión de un *LED* nos apoyaremos en la figura 6, la fuente de alimentación es nuestra salida digital (*Pin 2*) conectado al ánodo y el común (*GND*) de la tarjeta Arduino conectado al cátodo. Con una resistencia eléctrica en serie con el *LED*, como protección, para su buen funcionamiento.

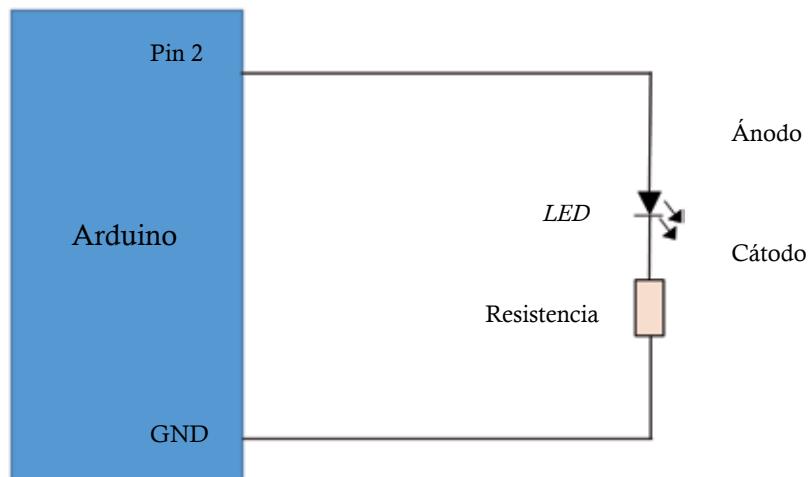


Figura 6. Esquema de conexión de un *LED* al puerto digital



## Objetivos

- Diferenciará e identificará las terminales digitales y las terminales de alimentación eléctrica del microcontrolador Arduino.
- Hará uso de alguno de las terminales digitales del Arduino para controlar el encendido y apagado de un *LED*.
- Hará uso de una resistencia para limitar la corriente que atraviesa un *LED*, cuando es conectado a alguno de los primeros 12 *pin* digitales.
- Aprenderá a calcular el valor de la resistencia de protección para el *LED*.
- Controlará el tiempo de encendido y apagado del *LED* mediante el uso de una variable global.

## Material utilizado

- Equipo de cómputo.
- Cable estándar USB (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- *LED* de cualquier color.
- Resistencia de  $220\ \Omega$ .

## Desarrollo

Como se ilustra en la figura 7 y con ayuda de una *protoboard*, conecta el ánodo del *LED* en el *pin* 2 del Arduino, y el cátodo al *pin* de tierra (*GND*) a través de la resistencia de  $200\ \Omega$ . El objetivo de dicha resistencia es limitar la corriente que atraviesa al *LED* y evitar que se queme.

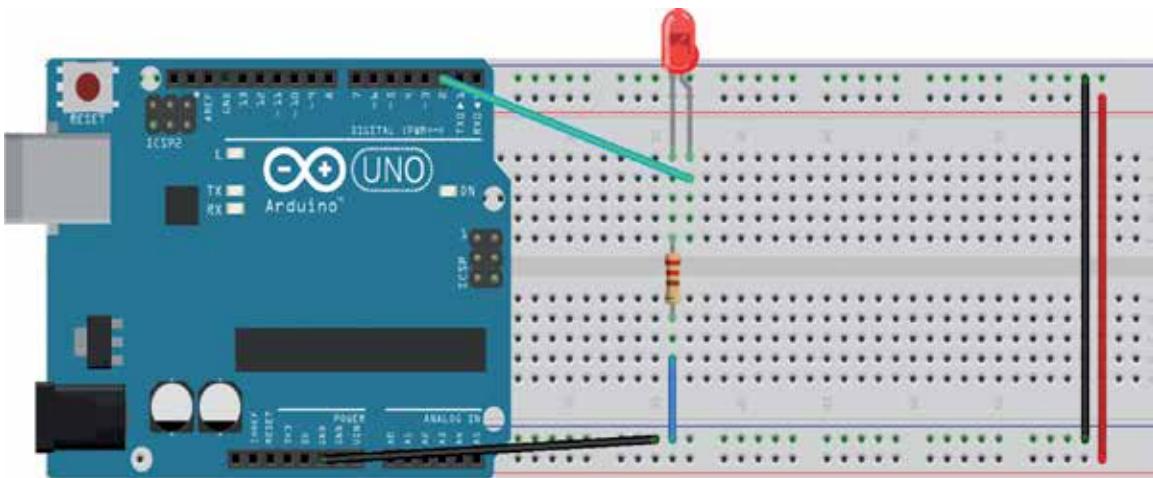


Figura 7. Prototipo de un *LED* al puerto digital. Imagen obtenida del software Fritzing

El *pin* 2 se configurará a través del código de programación (ver código en la sección correspondiente), como un *pin* de salida digital, por lo que proveerá una señal digital. Dicha señal digital proporcionará, cíclicamente, un voltaje de 5 volts (nivel ALTO o *HIGH*) durante cierto tiempo o, dicho en otras palabras, durante medio ciclo de la señal; después proveerá de un voltaje 0 volts (nivel BAJO o *LOW*) durante el restante medio ciclo. Esto significa que en el ánodo del *LED* estará a niveles BAJOS y ALTOS de voltaje, es decir, estará sometido a 0 volts y a 5 volts proporcionados por el *pin* 2.

Cuando en el *pin* 2 se tenga un nivel ALTO, entonces el *LED* encenderá; cuando dicho *pin* se tenga un nivel bajo, entonces el *LED* se apagará.

El tiempo que el *LED* se mantenga encendido o apagado estará determinado en el código del programa a través de la función *Delay()*. Esta función recibe como argumento un valor entero que corresponde a una pausa que hará el programa en milisegundos, es decir, nos indica cuánto tiempo permanecerá el *LED* encendido o apagado.

Conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.

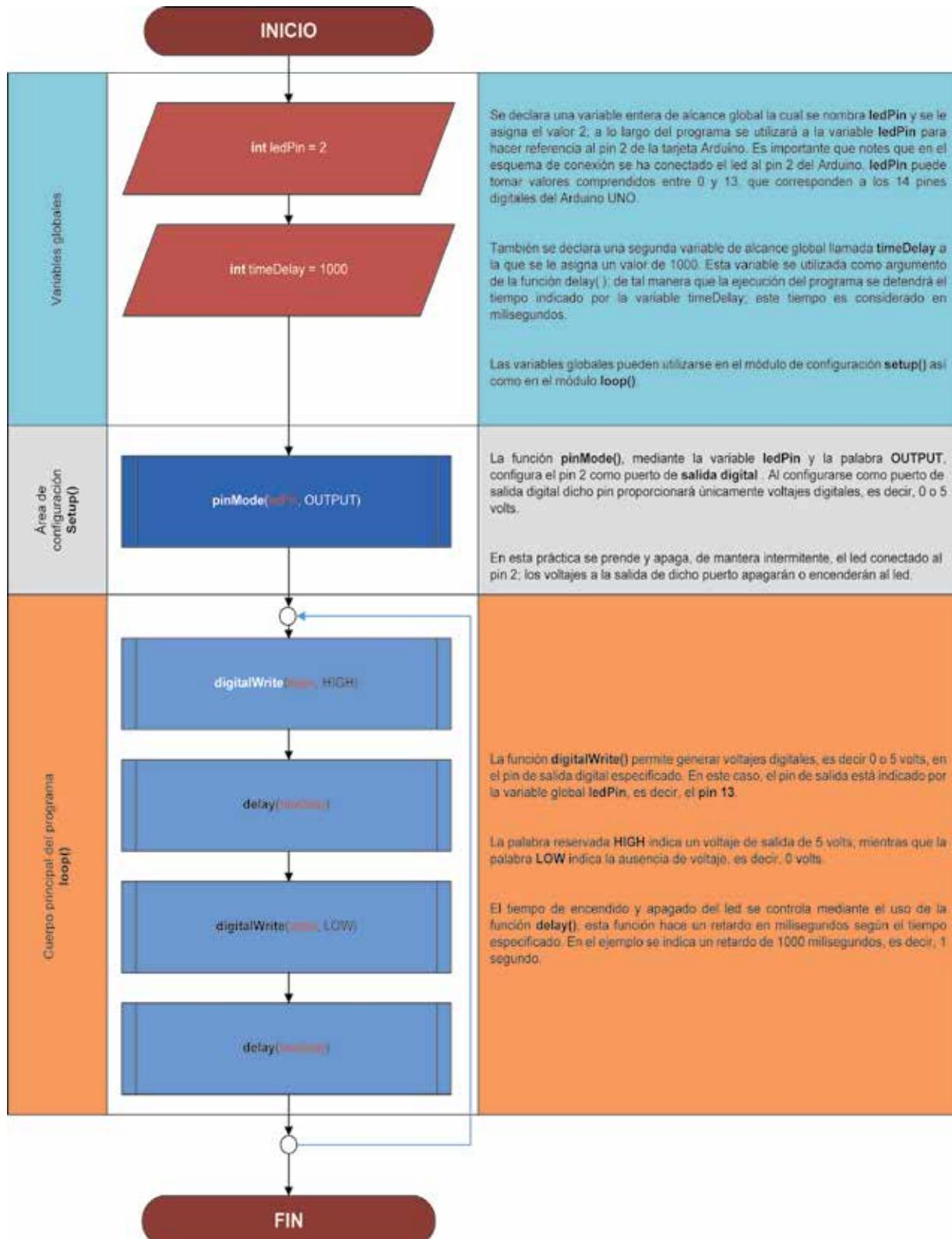
1. Ejecuta el *software* de programación de tu Arduino; carga el programa de ejemplo “*Blink*” y cárgalo en tu Arduino. El código se explica en la siguiente sección. Observa que se han hecho algunas modificaciones en el código original.

El programa “*Blink*” lo encontrarás accediendo en el menú Archivo>Ejemplos>01. Basics>*Blink*.

2. Observarás que el *LED* prende y apaga de manera intermitente debido a que el *pin* 2 proporciona una señal digital donde los niveles de voltajes ALTO y BAJO se alternan; el tiempo de encendido y apagado se determina en el código del programa por la variable *ledPin*.
3. Cambia el código que se te propone modificando el tiempo de encendido y apagado, a través del valor de la variable *timeDelay*, o modificando el *pin* de salida digital a través de la variables *ledPin*. Esta variable puede tomar valores del 0 al 13, que corresponden a las 14 terminales digitales del Arduino.



## Diagrama de flujo



## Código

|                                      |   |
|--------------------------------------|---|
| Declaración de variables globales    | <pre>/* Intermitencia de encendido y apagado de un LED. Se enciende y se apaga un LED; los tiempos de encendido y apagado son de un segundo. */ // Se conectará un led al pin indicado por la variable ledPin. // En esta práctica se conectará el led al pin 2 del Arduino. int ledPin = 2;  // La variable global timeDelay se empleará como argumento de la función delay( ), en el // cuerpo principal del programa, para controlar el tiempo de encendido y apagado del led. // Esta función provoca un retardo en la ejecución, la cantidad de milisegundos indicados por // el argumento int timeDelay = 1000;</pre> |
| Área de configuración setup()        | <pre>// la función de configuración setup() se ejecuta una sola vez void setup() {     // se configura el pin 13 (ledPin) como pin de salida     pinMode(ledPin, OUTPUT); }</pre>   |
| Cuerpo principal del programa loop() | <pre>// la función loop() se ejecuta ciclicamente y de manera ininterrumpida void loop() {     // enciende el LED (HIGH indica 5 volts de salida en el pin 13)     digitalWrite(ledPin, HIGH);     // retardo de un segundo     delay(timeDelay);     // se apaga el LED (LOW indica la ausencia de voltaje)      digitalWrite(ledPin, LOW);     // retardo de un segundo     delay(timeDelay); }</pre>   |



## Resultados y conclusiones

En esta práctica se conectó un *LED* en las terminales digitales del Arduino. Se observa que al conectarlo al *pin* 13 no se requiere colocar una resistencia de protección para el *LED*, debido a que el Arduino ya contiene una resistencia interna en dicho *pin*. Cuando se conecta el *LED* a cualquiera de las terminales restantes es necesario conectar una resistencia para limitar la corriente que atraviesa el *LED* y evitar daños en éste.

También se introdujo el uso de una variable global para controlar el tiempo de encendido y apagado del *LED*. Hacerlo hecho a través de la variable global nos permitió hacer un único cambio en el código, es decir, en el valor de la variable, y no en todas las funciones *delay ()* requerirían la modificación del tiempo de retardo.

## Referencias

Blink. (s.f.). Arduino LLC. Recuperado el 30 de Agosto de 2013, de <http://arduino.cc/en/Tutorial/Blink>

Specification for Zhongzhou led lamp (s.f.). Zhongzhou Electronics Ltd. Recuperado el 23 de septiembre de 2013, de <http://www.casadelled.com.ar/ZZ-GY-S0001B,ZL-503RCA2.pdf>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 21-31). Italia: Arduino LLC.



## Actividad 4. Control de encendido de un LED a través de un interruptor (*push button*) conectado a una entrada digital

Norberto Alejandro Pérez Colín / Alberto Ávila Ramos

### Antecedentes

¿Qué es un *push button*?

El control de diversos equipos eléctricos y electrónicos requiere de una interacción manual por parte del operador para encenderlos o apagarlos, para aumentar o disminuir el volumen, etcétera. Un dispositivo electro mecánico que nos permite controlar el funcionamiento de dichos aparatos es el botón o pulsador (*push button*).

Los botones son de diversas formas, tamaños y se encuentran en una gran diversidad de aparatos eléctricos y electrónicos. En la figura 1 se muestran algunos ejemplos de pulsadores.



Figura 1. Ejemplos de pulsadores (*push button*)

Los pulsadores controlan dos estados de operación de un circuito: circuito abierto y circuito cerrado. Un circuito abierto no permite el flujo de corriente mientras que un circuito cerrado sí permite el flujo de corriente.

En el mercado se encontrarán dos tipos de interruptores: los de contacto normalmente abierto en reposo (NA) y los de contacto normalmente cerrado en reposo (NC). Los interruptores del tipo NA se encuentran, en su posición de reposo, en circuito abierto, por lo cual no circula corriente a través de ellos; al ser oprimidos cambian su estado al de circuito cerrado, permitiendo así el flujo de corriente. Mientras que los interruptores del tipo NC se encuentran en circuito cerrado durante su posición de reposo, y al ser oprimidos se abre el circuito impidiendo el flujo de la corriente. Ambos tipos de pulsadores vuelven a su posición de reposo en el momento en que dejan de ser presionados, como se muestra en el esquema de la figura 2.

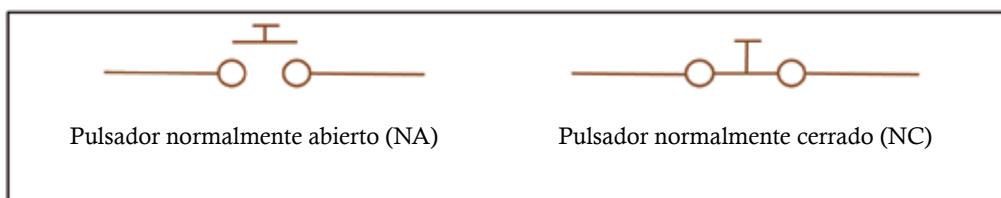


Figura 2. Esquema de tipos de pulsadores



## Introducción

¿Qué es una resistencia en pull-up?

Es un circuito electrónico que consta de una resistencia conectada por uno de sus extremos a una fuente de voltaje ( $V_{\text{entrada}}$ ) y por el otro lado a un interruptor del tipo NA conectado a tierra. Con este arreglo se está forzando a que durante la posición de reposo del interruptor se tenga un voltaje de salida ( $V_{\text{salida}}$ ) de 5 volts, es decir, igual al voltaje de entrada ( $V_{\text{entrada}}$ ), como se ilustra en la figura 3 (izquierda). Al presionar el *push button*, éste cambia su estado a circuito cerrado, por lo que el voltaje de salida será igual al voltaje de referencia ( $GND$ ) o 0 volts, como se ilustra en la figura 3 (derecha).

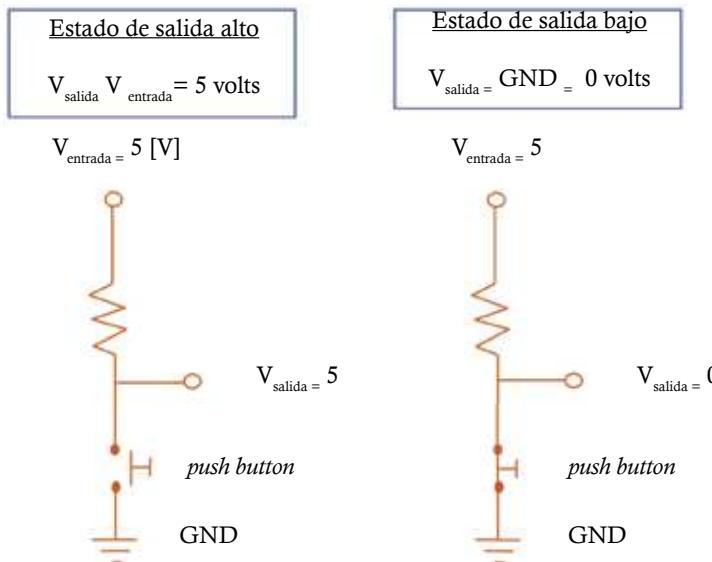


Figura 3. Resistencia en arreglo *pull-up*

En esta práctica vamos a conocer cómo se puede controlar el encendido y apagado de un *LED* a través de un *push-button*, ambos conectados al microcontrolador Arduino UNO; para ello utilizaremos un circuito con una resistencia en *pull-up*, como se muestra en la figura 4.

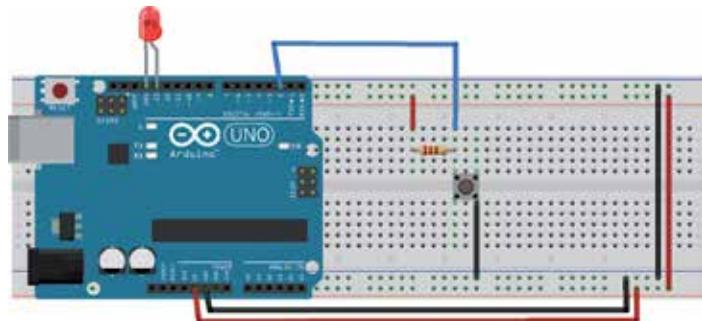


Figura 4. Esquema del control de encendido de un *LED* a través de un interruptor (*push button*) conectado a una entrada digital. Imagen obtenida del software *Fritzing*



## Objetivos

- Describirá las características generales de un *push button*.
- Describirá las características generales de un arreglo *pull-up*.
- Diferenciará e identificará las terminales digitales y las terminales de alimentación eléctrica del microcontrolador Arduino.
- Controlará el encendido y apagado del *LED* mediante el uso de una función digital (*digitalWrite*).

## Material utilizado

- Equipo de cómputo.
- Cable estándar USB (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- 1 *LED* de cualquier color.
- 1 resistencia de  $220\ \Omega$ .
- 1 *push-button*.

## Desarrollo

Conectamos nuestra *protoboard* a la fuente de alimentación de la tarjeta Arduino a 5 volts y tierra (figura 5).

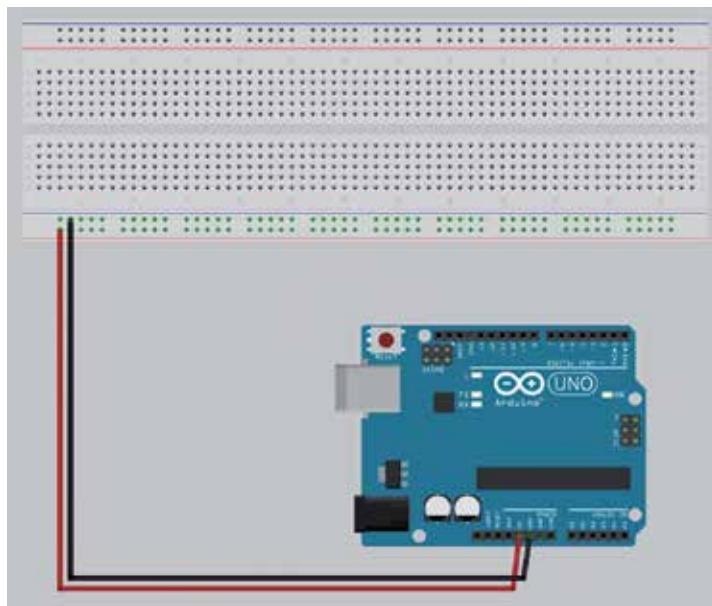


Figura 5. Esquema del control de encendido de un *LED* a través de un interruptor, inicio.

Imagen obtenida del software *Fritzing*



Agregamos el *push button* en serie con la resistencia de 220 ohms al circuito cerrado (figura 6).

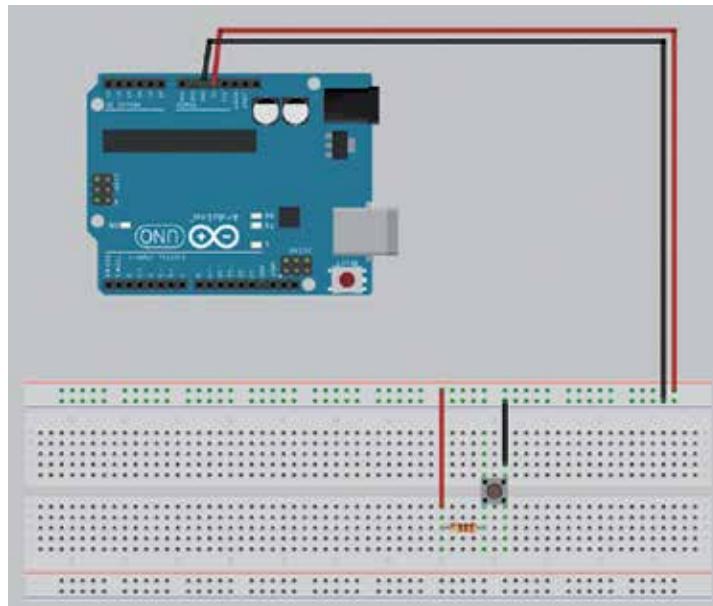


Figura 6. Esquema del control de encendido de un *LED* a través de un interruptor, intermedio.

Imagen obtenida del software *Fritzing*

Agregamos el *LED* en los pines *GND* y 13 de la tableta Arduino. En este *pin* se hace la depuración de los datos o procesos, y conectamos de la resistencia y su alimentación hacia el *pin* 3 de la tarjeta Arduino (figura7).

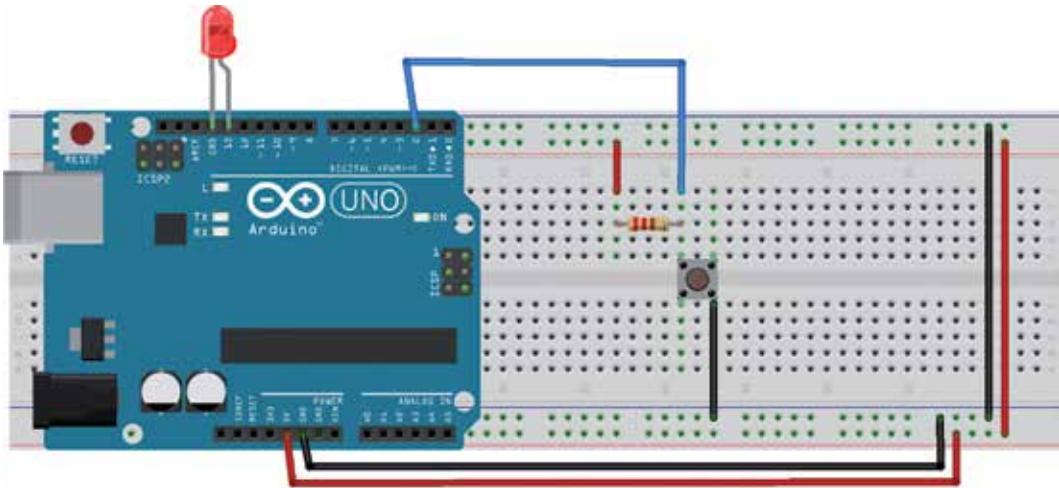


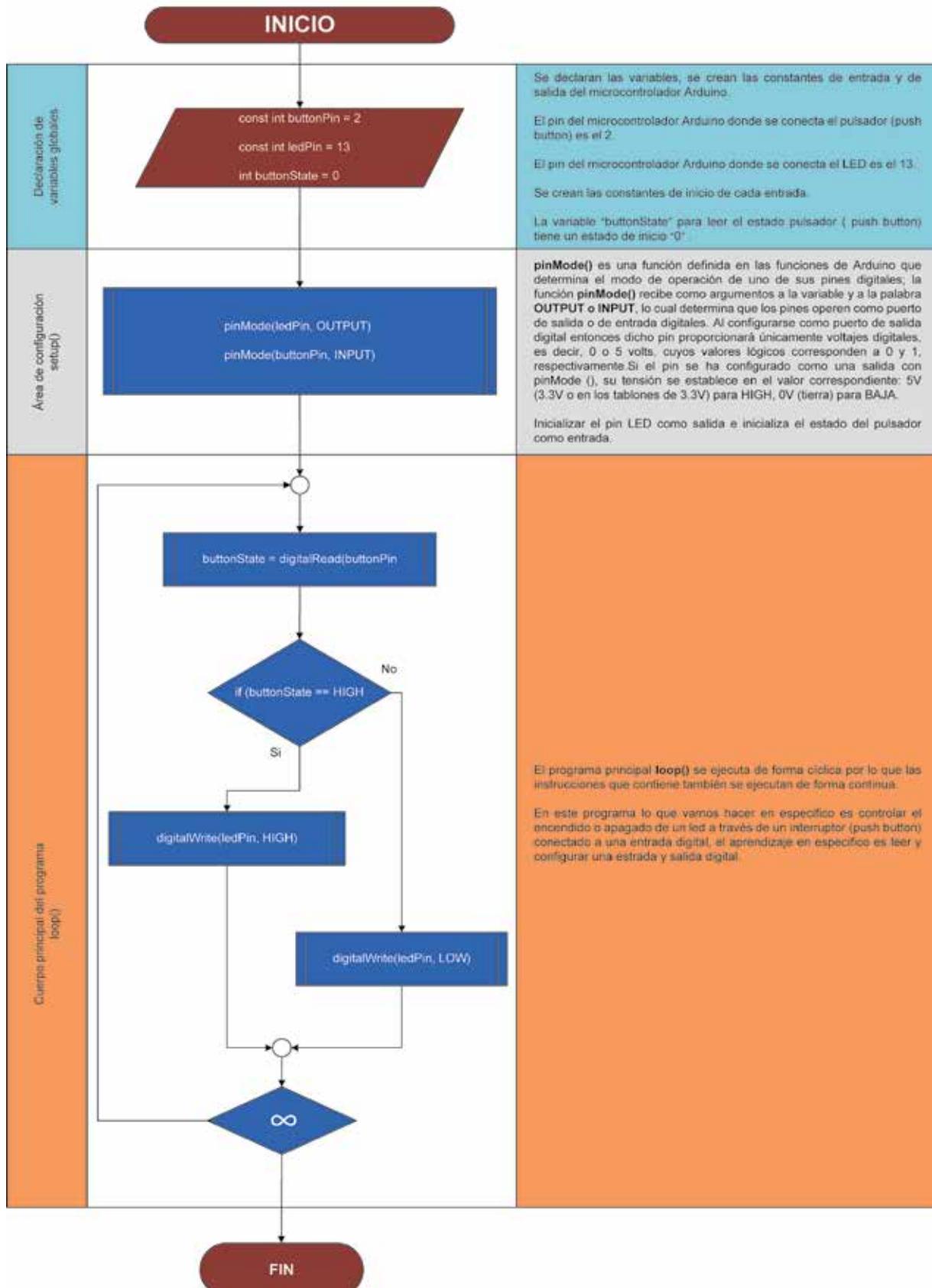
Figura 7. Esquema del control de encendido de un *LED* a través de un interruptor, final.

Imagen obtenida del software *Fritzing*



## Diagrama de flujo

**“Control de encendido de un led a través de un interruptor (push button) conectado a una entrada digital”**



## Código

|                                      |   |
|--------------------------------------|---|
| Declaración de variables globales    | <pre>//Práctica Control de encendido de un led a través de un interruptor (push button) conectado a una entrada digital</pre>   |
| Área de configuración setup()        | <pre>const int buttonPin = 2;//El número de pin donde se conecta el pulsador (push button) const int ledPin = 13;//El número de pin donde se conecta el LED</pre>   |
| Cuerpo principal del programa loop() | <pre>int buttonState = 0;//Variable para leer el estado pulsador ( push button)  void setup() {   pinMode(ledPin, OUTPUT);//Inicializar el pin LED como salida    pinMode(buttonPin, INPUT);//Inicializar el estado del pulsador como entrada }  void loop(){   buttonState = digitalRead(buttonPin); // Leer el estado del pulsador    // Comprobar si se presiona el pulsador.    // Si lo es, el ButtonState es ALTO    if (buttonState == HIGH) {      digitalWrite(ledPin, HIGH); // Cambia el Estado de LED a encendido   }    else {      digitalWrite(ledPin, LOW); //Cambia el Estado de LED a apagado   } }</pre> |



## Resultados y conclusiones

En esta práctica la manipulación de un *push button* y conocer sus características generales nos da la visión de cómo lograr controlar el microcontrolador Arduino. El *push button* es un interruptor o *switch* que regresa a su posición de origen, por eso solamente vamos a ver dos estados del *LED* (encendido y apagado).

El conocer un circuito básico con una resistencia, como el arreglo *pull-up*, nos ayuda a tener una visión de cómo trabajan los circuitos digitales eléctricamente, es decir, el valor cero digital tiene un valor de 0 volts y el uno digital tiene un valor de 5 volts.

Conocímos como funciona a nivel de programación para controlar el encendido y apagado del *LED* mediante el uso de una función digital (*digitalWrite*).

## Referencias

Digital Pins. (s.f.). Arduino LLC. Recuperado el 2 de octubre de 2013, de <http://arduino.cc/en/pmwiki.php?n=Tutorial/DigitalPins>

Taller de electrónica para usos creativos: *Prácticando con Arduino* (Mayo de 2008). Recuperado el 23 de septiembre de 2013, de

[http://wiki.medialab-prado.es/images/a/a0/Electronica\\_para\\_usos\\_creativos\\_dia3.pdf](http://wiki.medialab-prado.es/images/a/a0/Electronica_para_usos_creativos_dia3.pdf)

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 21-31). Italia: Arduino LLC.



## Actividad 5. Interfaz de nave espacial

Norberto Alejandro Pérez Colín / Alberto Ávila Ramos

### Antecedentes

*¿Cómo funcionan las terminales de potencia del microcontrolador Arduino?*

El microcontrolador Arduino UNO, puede funcionar como un suministro externo de energía eléctrica, sus terminales tienen las siguientes funciones y su ubicación se muestra en la figura 1.



Figura 1. La tarjeta electrónica Arduino UNO y sus terminales de potencia

- El pin V<sub>IN</sub>. Este pin se utiliza para conectar un voltaje de entrada a la placa Arduino, cuando se trata de utilizar una fuente de alimentación externa. Puede suministrar voltaje por este pin o a través de la toma de alimentación, sólo hay que tomar en cuenta que el GND o común es el mismo.
- El pin 5V. Este pin tiene un voltaje de 5V regulado.
- El pin de 3.3V. Un suministro de 3,3 volts generados por el regulador con un máximo de corriente de 50 mA.
- El pin GND. Los pines de tierra o común es la referencia cero de nuestros circuitos.

*¿Qué es una estructura de control, comparador?*

En la programación las estructuras de control selectivas se utilizan para resolver problemas donde es necesario tomar decisiones, es decir, donde se tengan diferentes caminos para llegar a la solución de un problema.

Las estructuras de control de selección ejecutan un bloque de instrucciones y saltan a un subprograma o subrutina, según se cumpla o no una condición.



Esta toma de decisión (se expresa en diagrama de flujo con un rombo como se observa en la figura 2) se basa en la evaluación de una o más condiciones que nos señalaran como alternativa o consecuencia la rama a seguir.

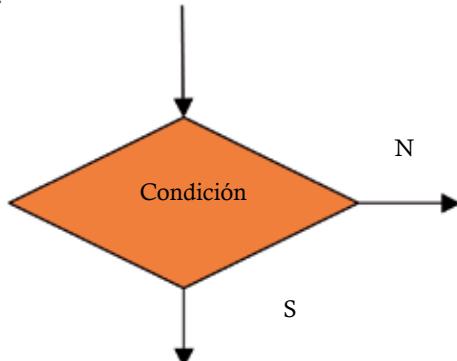


Figura 2. La estructura de control, función *if*

Las estructuras de control selectivas que se utilizan para la toma de decisiones lógicas las podemos clasificar de la siguiente forma:

#### *La estructura de control simple “si entonces”*

La estructura selectiva “si entonces” permite que el flujo del diagrama siga por un camino específico si se cumple una condición o conjunto de condiciones. Si al evaluar la condición o las condiciones el resultado es verdadero, entonces se ejecutan ciertas operaciones. Luego se continúa con la secuencia normal. Se muestra su estructura en un diagrama de flujo en la figura 3.

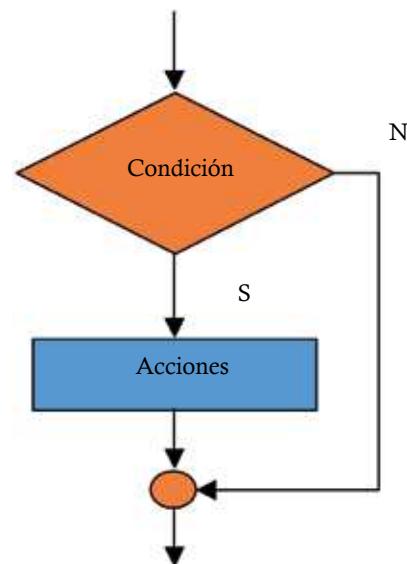


Figura 3. La estructura de control simple “si entonces”



| Estructura de control | Función | Sintaxis en <i>IDE</i> de Arduino     |
|-----------------------|---------|---------------------------------------|
| SI ENTONCES           | if      | if (Condición)<br>{<br>Acciones;<br>} |

*La estructura selectiva doble “si entonces/sino”*

La estructura selectiva “*si entonces/sino*” permite que el flujo del diagrama se bifurque por dos ramas diferentes en el punto de la toma de decisiones. Si al evaluar la condición o condiciones el resultado es verdadero, entonces se sigue por un camino específico y se ejecutan ciertas operaciones. Por otra parte, si el resultado es falso entonces se sigue por otro camino y se ejecutan otras operaciones. En ambos casos, luego de ejecutarse las operaciones indicadas, se continúa con la secuencia normal, como se muestra en la figura 4.

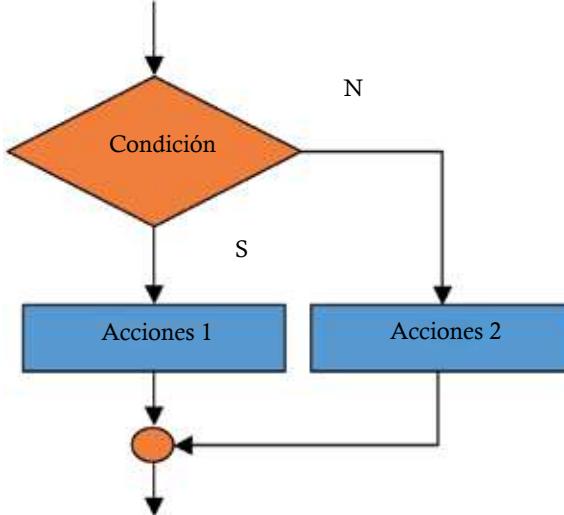


Figura 4. La estructura de control doble “si entonces / sino”

| Estructura de control | Función    | Sintaxis en <i>IDE</i> de Arduino  |
|-----------------------|------------|--|
| SI ENTONCES/SINO      | if ...else | if ... (Condición)<br>{<br>Acciones 1;<br>}<br>Else<br>{<br>Acciones 2;<br>} |

## Introducción

*¿Cómo utilizar la función if en Arduino?*

La función comparador *if* puede ser usado en conjunto con uno o más operadores de comparación, la instrucción “comprueba si cierta condición se cumple”, por ejemplo, si un valor de entrada posee un valor mayor a cierto número.

El formato para una comprobación *if* es la siguiente:

```
if (IntTiempo > 50)
```

Este programa comprueba si la variable *IntTiempo* es mayor a 50. Si lo es, el programa toma una acción. Dicho de otra forma, si la declaración escrita dentro de los paréntesis es verdadera (*true*), el código dentro de las llaves se ejecutará. Sino, el programa ignora dicho código.

Algunos ejemplos de su sintaxis son:

```
if (x > 120) digitalWrite(LEDpin, HIGH);
```

---

```
if (x > 120);  
digitalWrite(LEDpin, HIGH);
```

---

```
if (x > 120){ digitalWrite(LEDpin, HIGH); }
```

---

```
if (x > 120){  
    digitalWrite(LEDpin1, HIGH);  
    digitalWrite(LEDpin2, HIGH);  
}
```

---

Los operadores de comparación son los siguientes en Arduino:

*x == y* (x es igual a y)

*x != y* (x no es igual a y)

*x < y* (x es menor a y)

*x > y* (x es mayor a y)

*x <= y* (x es menor o igual a y)

*x >= y* (x es mayor o igual a y)

En esta práctica vamos a conocer cómo controlar las salidas digitales desde nuestro microcontrolador Arduino UNO, con una estructura de control en la programación, con la función *if* y un pulsador. En la figura 5 podemos observar el diagrama electrónico, para crear nuestro prototipo.



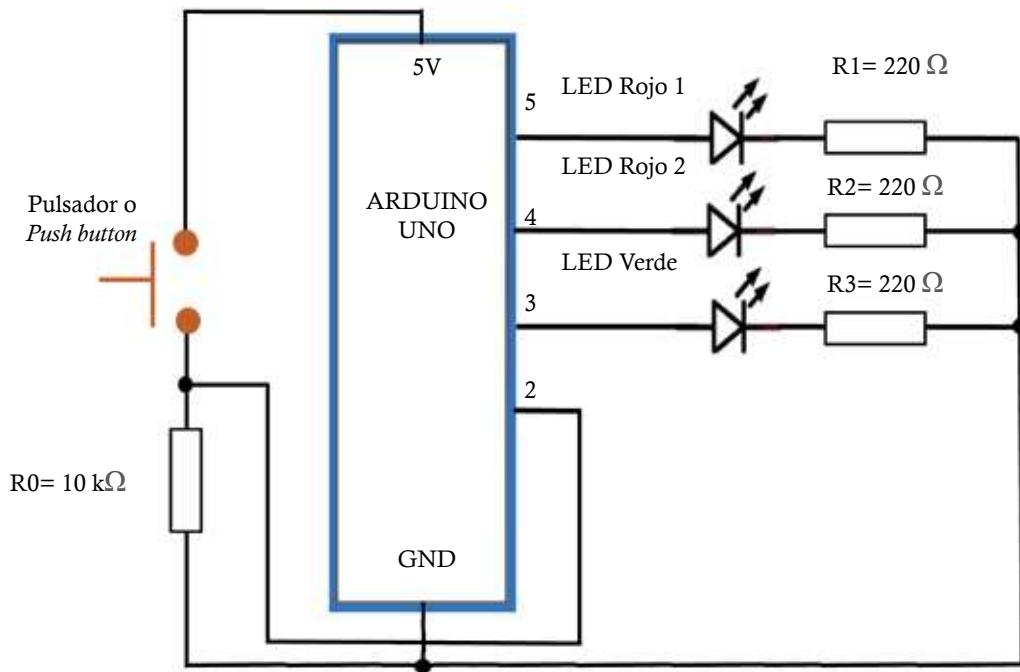


Figura 5. Esquema electrónico del prototipo interfaz de nave espacial

## Objetivos

- Describirá las características generales del *hardware* y *software* del microcontrolador Arduino utilizado en esta práctica.
- Describirá las características generales de una estructura de control con la función *if*.
- Diferenciará e identificará las terminales digitales y las terminales de alimentación eléctrica del microcontrolador Arduino.
- Hará uso de las 14 terminales digitales del Arduino para controlar los ledes, mediante la función *if*.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar USB (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 switch (*push button*).
- 2 LED rojo.
- 1 LED verde.
- 3 resistencias de  $220\Omega$ .
- 1 resistencia de  $10\text{ k}\Omega$ .



## Desarrollo

En la figura 6 comenzaremos por energizar la *protoboard* y aprender cómo se deben conectar los ánodos del *LED* rojo 1, *LED* rojo 2 y *LED* verde a la terminal 5, 4 y 3 del Arduino, respectivamente (aunque se puede haber seleccionado cualquier otra salida digital). El cátodo de cada *LED* debe conectarse al voltaje de referencia (*GND*) a través de una resistencia de  $220\ \Omega$ , la cual tiene el propósito de limitar la corriente que pasa por el *LED* y así evitar posibles daños a este dispositivo.

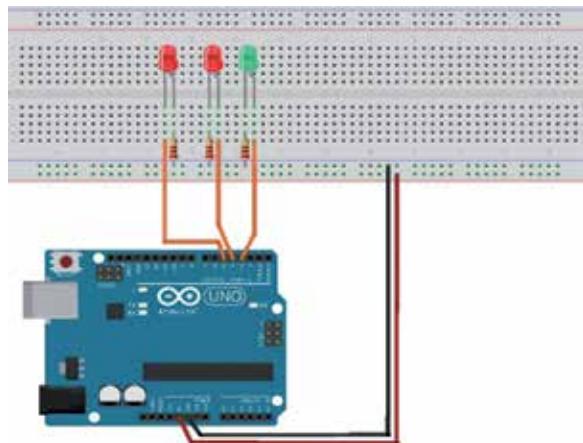


Figura 6. Esquema del prototipo interfaz de nave espacial

Ahora procederemos a conectar el pulsador (*push button*), como lo muestra la figura 7, donde un extremo va a la alimentación eléctrica de 5 V del Arduino y el otro extremo va conectado a una resistencia de  $10\ k\Omega$ , además va interconectado al *Pin 2* digital del Arduino.

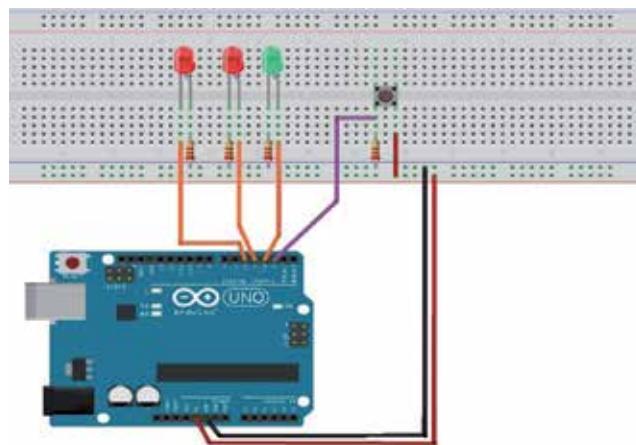


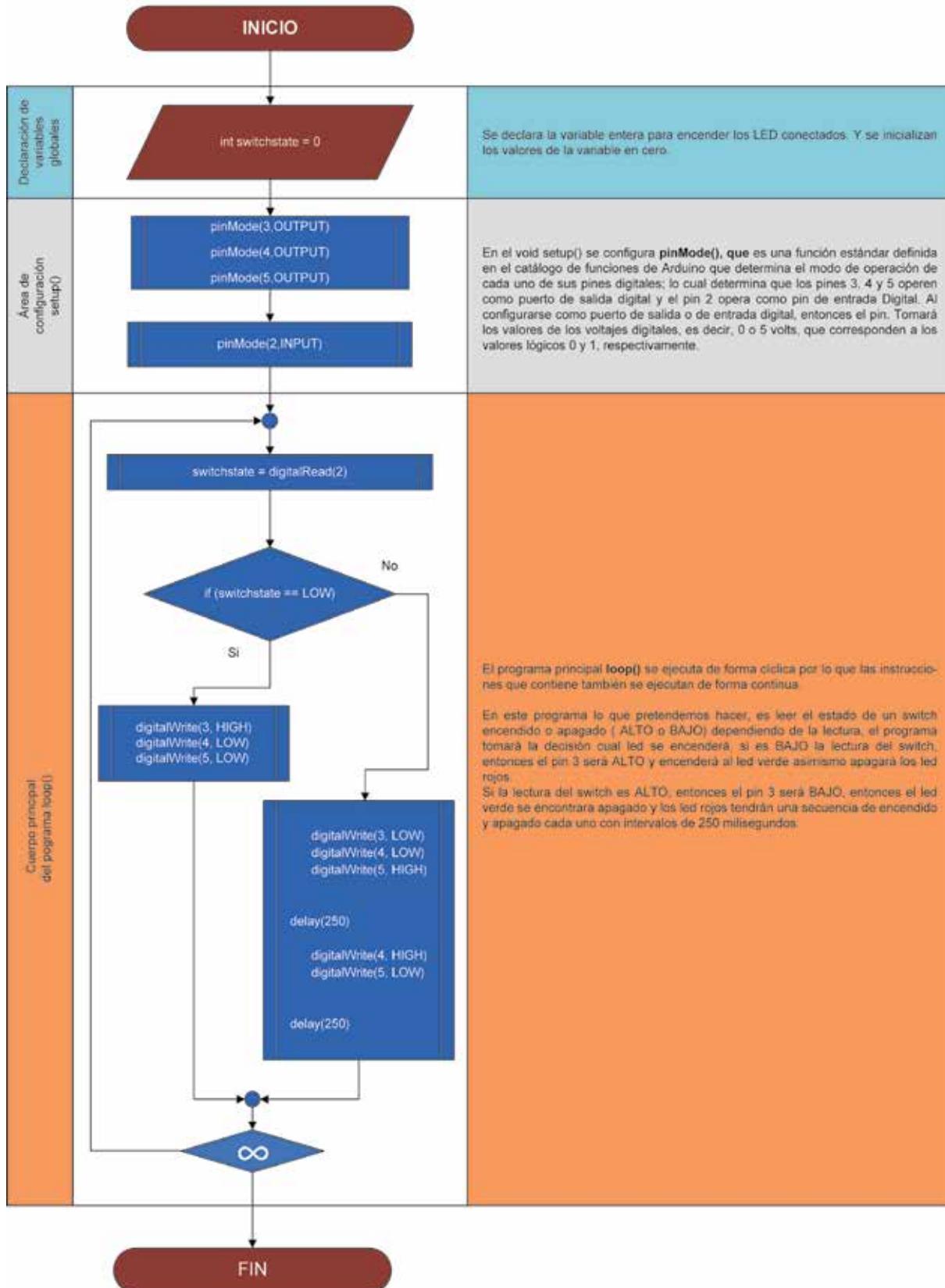
Figura 7. Esquema del prototipo interfaz de nave espacial

Conectamos la tarjeta Arduino a la computadora mediante el cable *USB*, Ejecutamos el entorno de programación de Arduino (*IDE*), y abrimos el *sketch* de los ejemplos del *StarterKit* “Proyecto -*Spaceship Interface*” y procedemos a cargar el código.



## Diagrama de flujo

### Interfaz de nave espacial



## Código

|                                      |  |
|--------------------------------------|--|
| Declaración de variables globales    | <pre>// Práctica: Interfaz de nave espacial  // Crear una variable global para mantener el estado del interruptor. Esta variable es //persistente en todo el programa. Cada vez que nos refierimos a SwitchState, //estamos hablando de la cantidad que posee  int switchstate = 0;</pre>  |
| Área de configuración setup()        | <pre>// Configuración setup()  void setup(){     // declarar los pines LED como salidas     pinMode(3,OUTPUT);     pinMode(4,OUTPUT);     pinMode(5,OUTPUT);      // declarar el pin del interruptor como entrada     pinMode(2,INPUT); }</pre>  |
| Cuerpo principal del programa loop() | <pre>void loop() {     // // Leer el valor del switch digitalRead () comprueba si hay voltaje en el pin o no     switchstate = digitalRead(2);      // Si no se presiona el botón enciende el LED verde y apaga los LEDs rojos     if (switchstate == LOW) {         digitalWrite(3, HIGH);         digitalWrite(4, LOW);         digitalWrite(5, LOW);     }      // Si el interruptor está en estado alto (se presiona el botón)     // se apagar el LED verde y parpadeará alternativamente los LEDs rojos     else {         digitalWrite(3, LOW);         digitalWrite(4, LOW);         digitalWrite(5, HIGH);     }      // Esperar un cuarto de segundo (250ms) antes de cambiar la luz     delay(250);     digitalWrite(4, HIGH);     digitalWrite(5, LOW);      // Esperar un cuarto de segundo (250ms) antes de cambiar la luz     delay(250); }</pre> |

## Resultados y conclusiones

En esta práctica se trabajaron específicamente las características generales de una estructura de control con la función *if*, donde nos da la pauta para el uso de la toma de decisiones en un algoritmo, además de utilizar en el prototipo electrónico los 14 terminales digitales del Arduino para controlar los ledes. Donde los aprendizajes que se manejan durante la práctica van enfocados a dejar un camino en la programación de Arduino.

Se demostró también que, con el armado de circuitos electrónicos, se combinan varios componentes electrónicos a la vez (en donde cada uno de ellos cumple una determinada función) y se pueden implementar diversos mecanismos de control que pueden permitir resolver necesidades de la vida cotidiana.

## Referencias

- if (conditional) (s.f.). Arduino LLC. Recuperado el 2 de octubre de 2013, de <http://arduino.cc/en/Reference/If>
- if / else (s.f.). Arduino LLC. Recuperado el 2 de octubre de 2013, de <http://arduino.cc/en/Reference/Else>
- Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 32-41). Italia: Arduino LLC.



## Actividad 6. Lectura analógica y salida PWM

Norberto Alejandro Pérez Colín / Alberto Ávila Ramos

### Antecedentes

¿Qué es un potenciómetro?

Un potenciómetro es un componente electrónico, al cual se le puede variar el valor de su resistencia permitiendo controlar la intensidad de corriente que fluye por un circuito. Un potenciómetro consiste de manera básica en una resistencia con una conexión intermedia y móvil. Se utilizan como divisores de tensión o como resistencias ajustables, cuando no se conecta uno de sus extremos. La figura 1 muestra algunos símbolos utilizados para representar al potenciómetro.

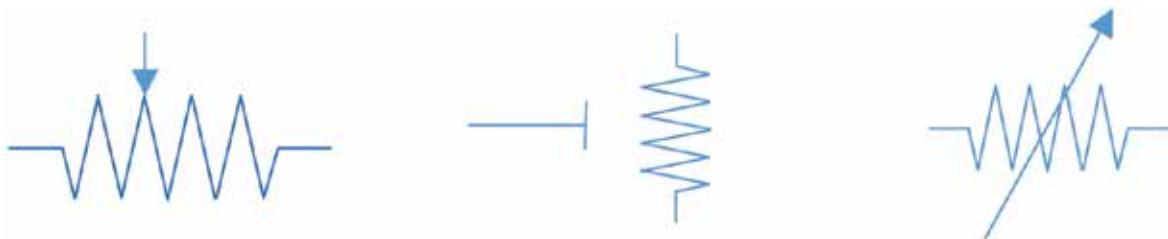


Figura 1. Símbolos utilizados para el potenciómetro

El potenciómetro se representa como una resistencia con dos contactos a los extremos, entre los que se mide su resistencia nominal y una toma intermedia que se desplaza manualmente; se suele representar por una flecha, como se observa en la figura 2, el comportamiento de los voltajes entre sus tres terminales.

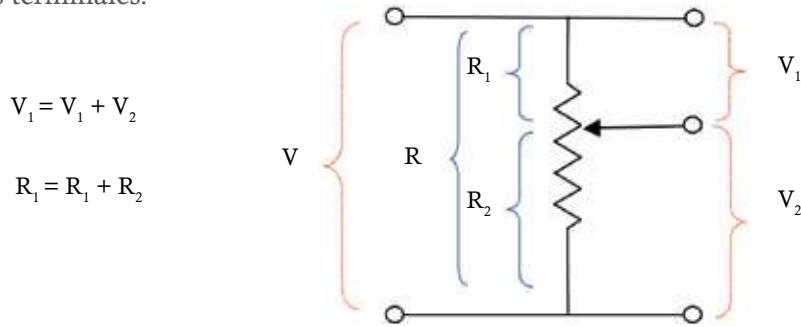


Figura. 2. Representación de potenciómetro manual

Los potenciómetros más usuales son los usados como controles de tono y volumen. Sin embargo, la parte visible es en realidad el mando que permite actuar sobre un eje, que mueve el cursor del potenciómetro. A este se le llama perilla. Cabe mencionar que sólo se presentan en la figura 3 algunos potenciómetros de los más habituales, pues existen diferentes tipos. No todos los potenciómetros tienen accionamiento giratorio, los hay de accionamiento longitudinal en los que el cursor se desplaza en línea recta.





Figura 3. Algunos potenciómetros comunes

### *¿Qué es modulación por ancho de pulso (PWM)?*

La modulación por ancho de pulso (*PWM* por sus siglas en inglés) es una técnica para obtener resultados analógicos en medios digitales. En control digital se utiliza para crear una onda cuadrada, esta señal cambia entre encendido y apagado. Este patrón de encendido y apagado puede simular voltajes en el medio completo, encendido para 5 volts y apagado para 0 volts. La porción de tiempo de la señal que pasa entre encendido y apagado se llama ancho de pulso. Para obtener diferentes valores analógicos se cambia o modula el ancho de pulso. Si se repite este patrón de encendido y apagado lo suficientemente rápido con un *LED*, el resultado es como si la señal tuviera una tensión constante entre 0 y 5 volts que controla el brillo del *LED*.

La modulación por ancho de pulso (*PWM*) es el voltaje promedio del valor de salida de un pin entre 0 y 5 volts con un ancho de pulso variable durante un periodo de tiempo determinado. Es una técnica para transferir información o energía, simulando una salida analógica con una salida digital.

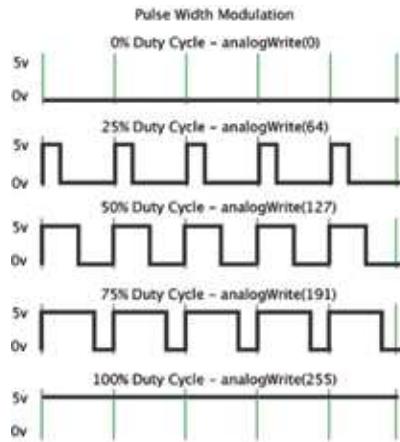


Figura 4. *PWM* y porcentajes de energía

La figura 4 nos muestra cómo va cambiando la función *analogWrite* (desde 0 hasta máximo 255) y cómo va cambiando la energía en porcentajes, gracias a esta técnica podemos simular una señal analógica.

La principal desventaja que presenta *PWM* es la eventualidad de que haya interferencias generadas por radiofrecuencia. Éstas pueden disminuir ubicando el controlador cerca de la carga y realizando un filtrado de la fuente de alimentación.



## Introducción

En esta práctica vamos a conocer cómo interpreta las entradas analógicas el microcontrolador Arduino, ya que es un elemento totalmente digital que tiene la propiedad de convertir una señal analógica a digital, gracias a que posee en sus entradas un convertidor analógico-digital (CAD) y sus salidas pueden simular una señal analógica gracias a los puertos *PWM* (~ modulación por ancho de pulso), que es una técnica para transferir información o energía.

*¿Cómo interpreta las entradas analógicas el microcontrolador Arduino?*

Arduino trabaja solamente con señales digitales, pero es capaz de manejar señales analógicas, mediante la discretización de la señales. Esto es convierte una señal analógica a una señal digital. Por ejemplo, imaginemos una señal seno analógica, el modo en que Arduino puede trabajar con ella es convertirla a una señal digital. Las señales analógicas son continuas mientras las señales digitales cambian de un valor a otro sin tener que pasar por los valores intermedios.

Primero que nada, tenemos que identificar con cuantos puertos analógicos cuenta el microcontrolador Arduino UNO, los cuales en la placa están representados como entradas analógicas. Entradas analógicas, la placa Arduino tiene 6, las cuales van del A0 → A5, en la figura 5 se muestra un esquema de cómo están situados los puertos en la placa de Arduino:



Figura 5. Entradas analógicas en la placa Arduino

En Arduino para la programación de la lectura de los parámetros en el *pin* configurado, se trabaja con la función *AnalogRead (pin)*.

Y los pasos para programar son los siguientes:

1. Lo primero a realizar es leer la señal conectada al pin analógico de entrada.
2. El valor leído lo pasa a un convertidor analógico digital de una resolución de 10 bits ( $2^{10}=1024$ ).
3. Los puertos analógicos no se tienen que inicializar como de entrada.
4. La lectura de los datos se realiza con la función `int analogRead (pin)`.
5. Los parámetros del *pin* debe ser un número de 0 a 5 volts y nos va a indicar el número del *pin* de la entrada analógica que deseamos leer.



La función *analogWrite (pin, valor)*

1. El formato de lectura para la programación *analogWrite (pin, valor)*.
2. Se indica el *pin* 3, 5, 6, 9, 10, 11, en el cual se quiere generar la señal *PWM*.
3. El valor puede ser especificado como una variable o constante con valor entre 0 y 255; en donde 0 siempre es apagado y 255 siempre encendido.
4. El *pin* generará una onda cuadrada de unos 490 Hz, esta parte del periodo que está en nivel alto y el resto del tiempo en nivel bajo. Así parece como que se ha escrito una señal analógica (*PWM*).

Es muy importante tomar en cuenta que las terminales analógicas, al contrario que los digitales, no necesitan ser declarados al principio como *INPUT* u *OUTPUT*. En la figura 6 se muestra el circuito electrónico de cómo debe implementarse el prototipo.

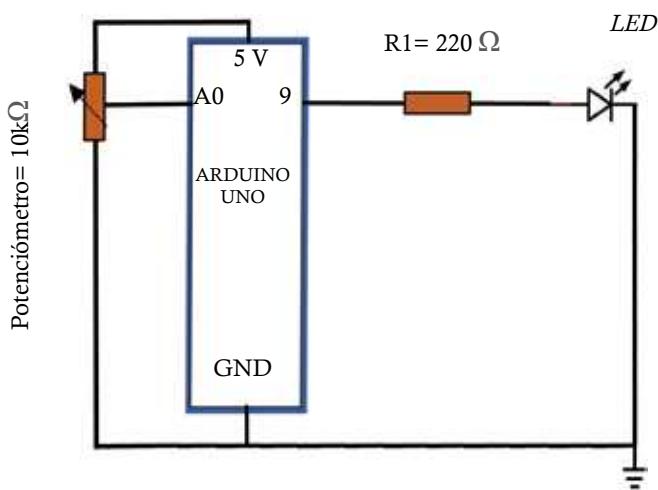


Figura 6. Circuito electrónico del prototipo

## Objetivo

- Diferenciará e identificará las terminales digitales y las terminales analógicos del microcontrolador Arduino.
- Hará uso y entenderá cómo funcionan las terminales digitales *PWM* del microcontrolador Arduino.
- Entenderá cómo funciona el convertidor analógico-digital del microcontrolador Arduino para las terminales analógicos.
- Aumentará su destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.



## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 *LED* (cualquier color).
- 1 resistencia de  $220\ \Omega$ .
- 1 potenciómetros de  $10\ k\Omega$ .

## Desarrollo

Conectar la placa Arduino a la *protoboard*, con la alimentación de 5 volts y *ground (GND)*, como lo indica la figura 7.

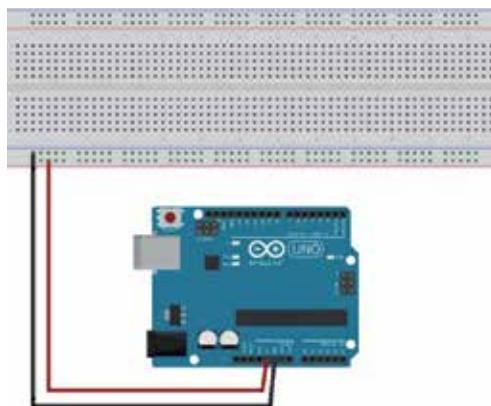


Figura 7. Esquema eléctrico del prototipo

Incorporamos el *LED* a la *protoboard* conectando sus terminales una resistencia de  $220\ \Omega$ , como muestra la figura 8.

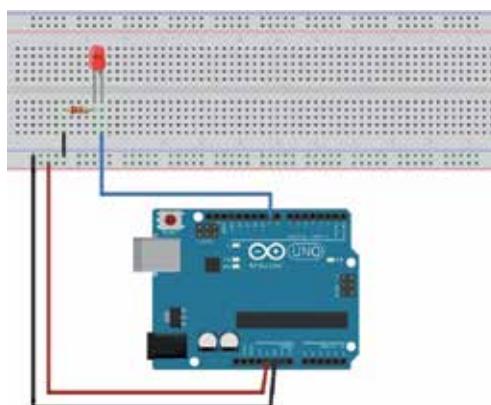


Figura 8. Esquema eléctrico del prototipo



Agregamos el potenciómetro de  $10\text{ k}\Omega$ , los extremos de estos van con su respectiva alimentación y el pin del centro que es para control se conecta a la entrada analógica A0.

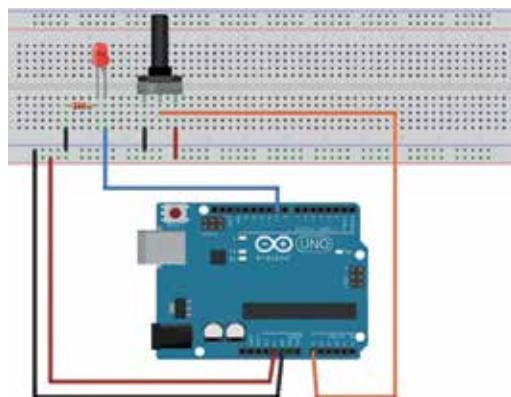


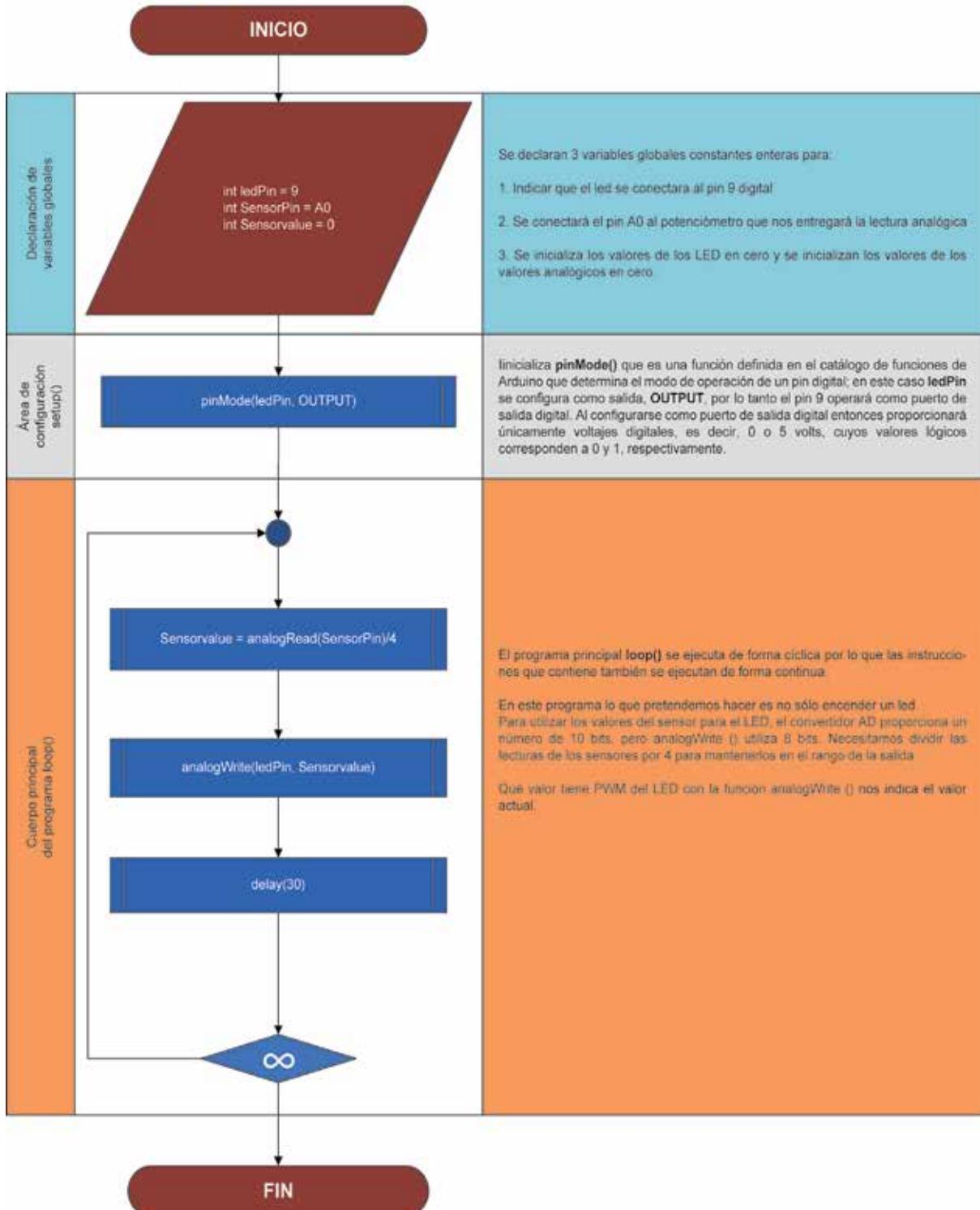
Figura 9. Esquema eléctrico del prototipo

Copiamos el código a nuestro *IDE* de Arduino para compilar y correr el programa.



## Diagrama de flujo

### Entrada Analógica y salida PWM



## Código

|                                      |   |
|--------------------------------------|---|
| Declaración de variables globales    | // Práctica entrada analógica y escritura PWM<br><br>int ledPin = 9; // Led conectado al pin digital 9<br>int SensorPin = A0; // Pin de lectura del potenciómetro<br>int Sensorvalue = 0; //Inicialización del valor del sensor analógico |
| Área de configuración setup()        | // Configuración setup()<br><br><b>void setup()</b> {<br>pinMode(ledPin, OUTPUT); // Configuramos el pin 9 como de salida<br>}<br><br><b>void loop()</b> {  |
| Cuerpo principal del programa loop() | Sensorvalue = analogRead(SensorPin)/4; //Se divide en 4 la lectura del convertidor A/D<br>analogWrite(ledPin, Sensorvalue); //Se envía la intensidad al led<br>delay(30);   }   |

## Resultados y conclusiones

En esta práctica abordamos dos aprendizajes fundamentales para entender el funcionamiento del microcontrolador Arduino, donde primeramente diferenciamos e identificamos las terminales digitales y las analógicas.

El primer aprendizaje que trabajamos fue el uso y el cómo funcionan las terminales digitales *PWM* del microcontrolador Arduino. Y el segundo, cómo funciona el convertidor analógico-digital del microcontrolador Arduino para las terminales analógicas.

Con estos aprendizajes podemos ir generando prototipos con las bases obtenidas en esta práctica, así como aumentar nuestra destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.

## Referencias

PWM (s.f.). Arduino LLC. Recuperado el 7 de noviembre de 2013, de <http://arduino.cc/en/pmwiki.php?n=Tutorial/PWM>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). Get to know your tools. *Arduino Projects Book* (pp. 20-30). Italia: Arduino LLC.

Resistores variables (s.f.). Universitat de les Illes Balears. Recuperado el 7 de agosto de 2013, de [http://www.uib.cat/depart/dfs/GTE/education/industrial/tec\\_electronica/teoria/resistores\\_variables.pdf](http://www.uib.cat/depart/dfs/GTE/education/industrial/tec_electronica/teoria/resistores_variables.pdf)



## Actividad 7. LED multicolor

Misael Aguilar Zamora / José Luis Aguilar Hinojosa

### Antecedentes

¿Cómo funciona un LED multicolor?

Es un *LED* que tiene cuatro patillas; en su encapsulado contiene tres ledes internos, que emite luz cada uno de ellos con los colores primarios de rojo, verde, azul (*RGB Red, Green, Blue*), existen en el mercado de ánodo y cátodo común; para que funcione éste, debe estar siempre alimentado. El brillo de cada color se determina por su voltaje de entrada. Mediante la combinación de los tres colores en diferentes cantidades, puedes convertir el *LED* a cualquier color que deseas.

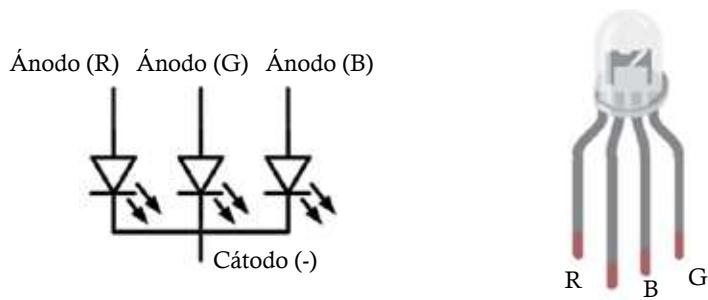


Figura 1. Símbolo eléctrico y esquema de un *LED* RGB. Imagen obtenida del software *Fritzing*

El *LED* es un componente sólido sin filamento ni gases, por lo que en la actualidad es muy conveniente en proyectos de iluminación gracias a su amplia vida útil y bajo consumo de energía, es antiexplosivo porque no utiliza chispa, ni la incandescencia de un filamento y prácticamente no genera temperatura en comparación con las demás tecnologías de iluminación.

El *LED* tiene polaridad, ya que funciona polarizándolo de una sola forma tiene un terminal que debe conectarse a negativo y el otro debe conectarse a positivo; la terminal negativo es el cátodo y el positivo el ánodo.

### Introducción

El microcontrolador Arduino cuenta con un convertidor analógico-digital, se encuentra en los pines A0, A1, A2, A3, A4 y A5. Estos pines de entrada son utilizados para leer los valores de múltiples sensores a utilizar, con señales analógicas. Este tipo de señales son las que comúnmente entregan los sensores para lectura de fenómenos físicos, como temperatura, distancia, presión, gas, calor, radiación, etcétera.



| Entrada analógica<br>(volts) | Salida digital |
|------------------------------|----------------|
| 0.000                        | 0              |
| 0.625                        | 127            |
| 1.250                        | 255            |
| 1.875                        | 383            |
| 2.500                        | 511            |
| 3.125                        | 639            |
| 3.750                        | 767            |
| 4.5                          | 920            |

Tabla 1. Mapeo de señal analógica a señal digital.

Las terminales de entrada analógicos A0 – A5, leen el voltaje cambiante y lo convierte a un número entre 0 y 1023, tabla 1. En otras palabras el Arduino trabaja con voltajes de 0 a 5 volts, si el valor de entrada es 0 volts el valor digital es 0 (cero), cuando tenemos todo el recorrido analógico hay 5 volts que nos muestran el valor de entrada de 1023. La función en Arduino es, *analogRead()* que devuelve un número entre 0 y 1023 que es proporcional a la cantidad de voltaje de 0 a 5 volts, El convertidor analógico-digital es de 10 bits por lo que  $2^{10} = 1024$ . Esto significa que va a asignar voltajes de entrada entre 0 y 5 volts en valores enteros entre 0 y 1023, en otras palabras no maneja cifras decimales el convertidor analógico-digital; si fuera el caso tomaría el entero superior o inferior. La relación para convertir valores analógicos a valores digitales en Arduino se calcula con la siguiente relación:

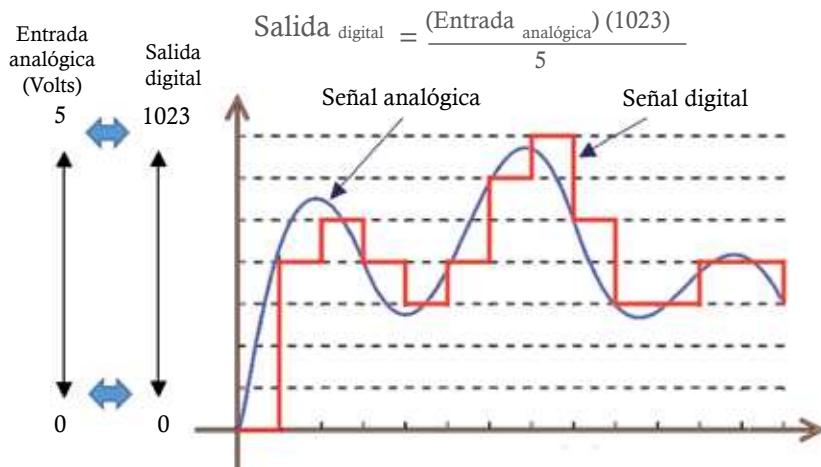


Figura 2. Señal de entrada analógica (azul) y su señal digital (rojo)

Lo que pretendemos hacer con esta práctica es no sólo encender un *LED RGB* mediante el microcontrolador Arduino, sino cambiar también el color del *LED* utilizando los diferentes valores que nos entrega un potenciómetro. Así, sólo tendremos que girar en un sentido o en otro los distintos potenciómetros para obtener las diferentes mezclas de colores que nos ofrece este componente electrónico (*LED RGB*), como lo muestra la figura 3.



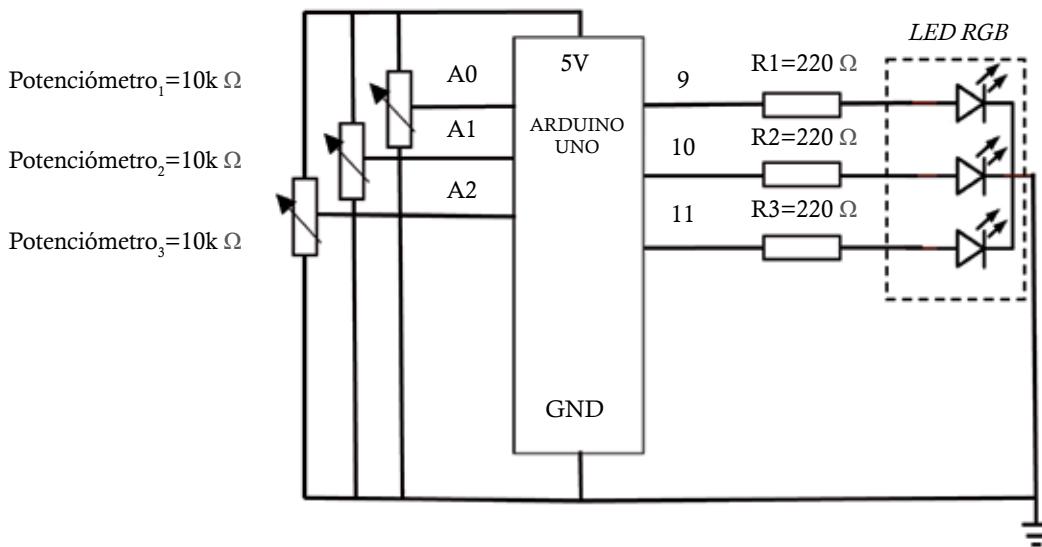


Figura 3. Circuito electrónico del proyecto *LED multicolor RGB*

El código de colores *RGB*, rojo-verde-azul, se basa en la mezcla de estos tres colores para conseguir toda la gama completa. Cada uno de los colores toma un valor entre 0 y 255, un total de 256, con los que se consigue un total de  $256 \times 256 \times 256 = 16\ 777\ 216$  colores distintos. Este valor se representa en hexadecimal, con lo que el rango va de 00 a FF por cada uno de ellos. El código se expresa así: #RRGGBB siendo cada uno de los valores de 2 cifras el rango de cada uno de los 3 colores, con lo que obtenemos el valor final que representa a cada color. Algunos ejemplos:

- Negro: representa la ausencia de color, por tanto su valor será: #000000
- Blanco: es la mezcla de todos los colores: #FFFFFF
- Rojo: todo el tono es del rojo, siendo los otros 2 nulos: #FF0000
- Verde: sólo aparece el verde, sin tener valor los otros 2: #00FF00
- Azul: idéntico razonamiento, pero sólo con el azul: #0000FF

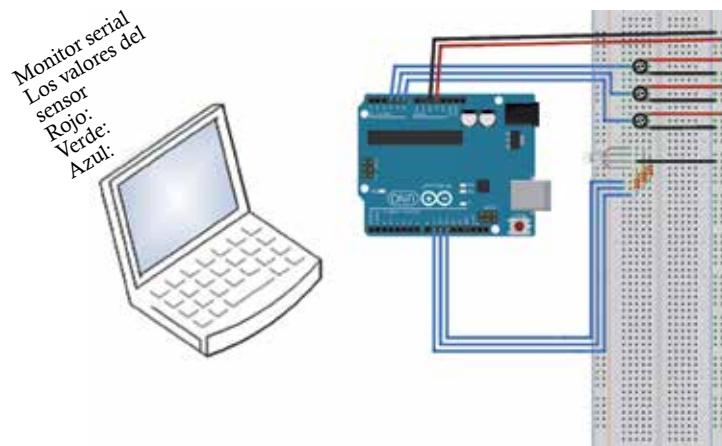


Figura 4. Circuito del proyecto *LED multicolor RGB* y su *monitor serial*. Imagen obtenida del software *Fritzing*



En la figura 4 observamos en la computadora el *monitor serial*, que se encuentra en el *IDE* de Arduino, con el que podemos observar cómo van cambiando los parámetros de las entradas analógicas que se obtienen del giro de los potenciómetros.

## Objetivos

Al finalizar la práctica el alumno:

- Diferenciará la forma en la cual se deben de conectar los componentes *LED* multicolor y potenciómetro para que se puedan usar como dispositivos de entrada analógica y de salida digital.
- Aprenderá mediante estructuras de control, a nivel de programación, la función *analogRead()* para operar un circuito electrónico.
- Aumentará su destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.
- Entenderá cómo funciona el convertidor analógico-digital del microcontrolador Arduino.
- Describirá las características generales del *LED* multicolor y como interconectarlo con el microcontrolador Arduino.

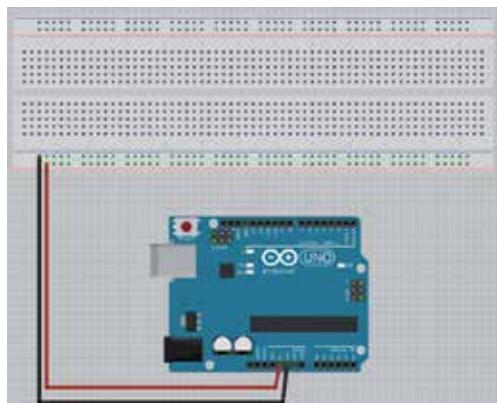
## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 *LED* multicolor.
- 3 resistencias de  $220\ \Omega$ .
- 3 potenciómetros de  $10\ k\Omega$ .

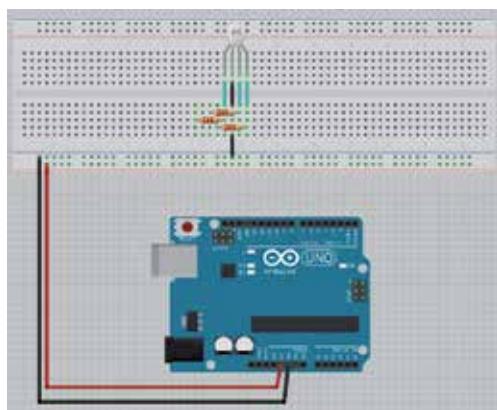
## Desarrollo

Conectar la placa Arduino a la *protoboard*, con la alimentación de 5 volts y el voltaje de referencia (*ground*).

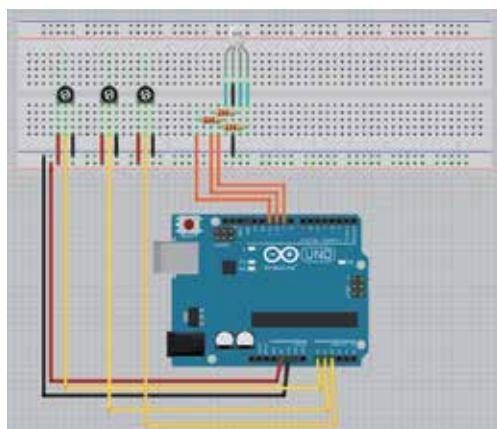




Incorporamos el *LED RGB* a la *protoboard* conectando en tres de sus pines una resistencia de  $220\ \Omega$ , y en la cuarta la mandamos a tierra, como se muestra en la figura:



Agregamos los tres potenciómetros de  $10\text{ k}\Omega$ ; los extremos de éstos van con su respectiva alimentación de 5 voltios y el pin del centro, que es para ajustar, se mandan a las entradas analógicas A3, A4 y A5, respectivamente, y las resistencias se conectan a las salidas digitales 9, 10 y 11.

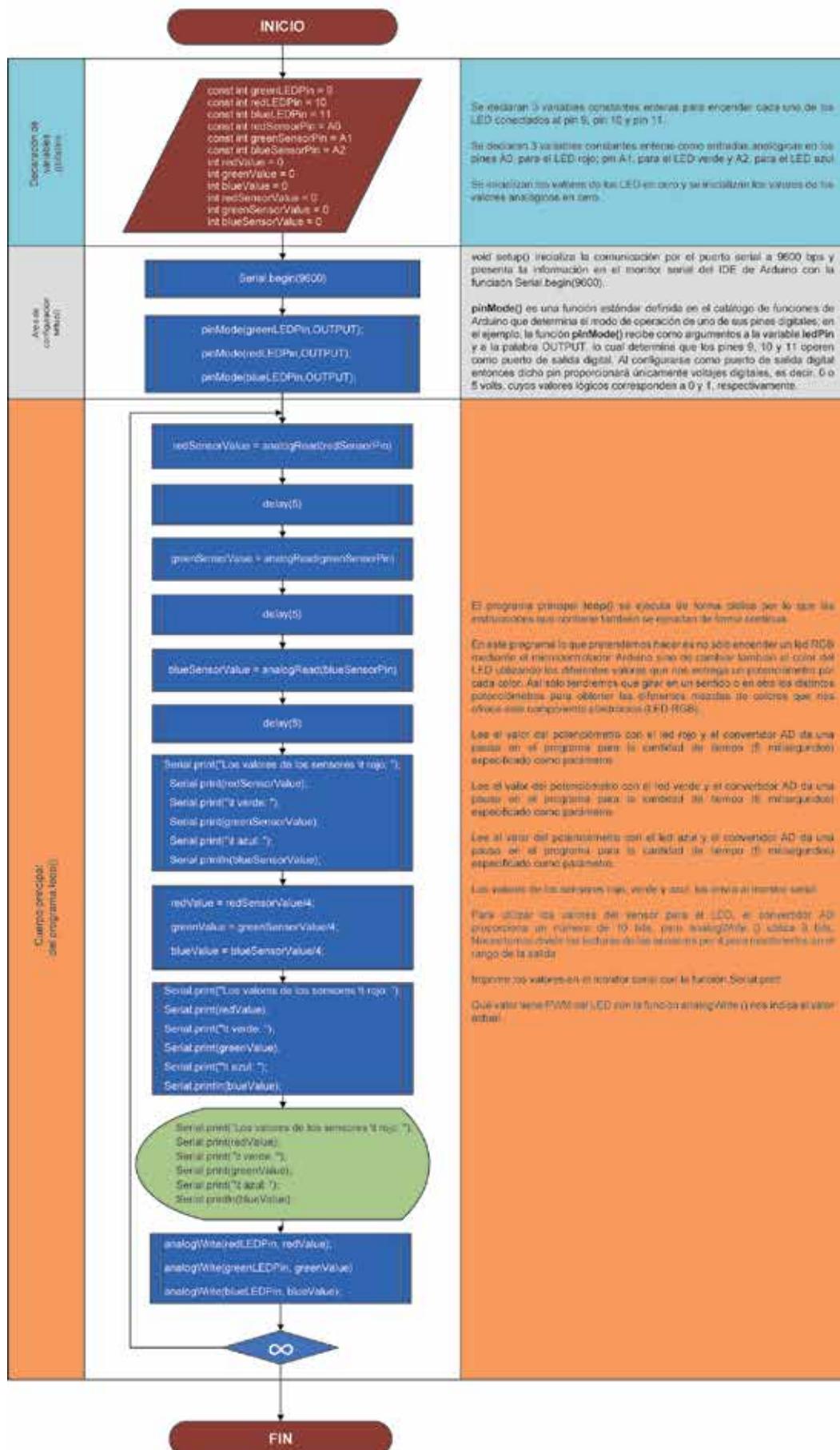


Copiamos el código a nuestro *IDE* de Arduino para compilar y correr el programa.



# Diagrama de flujo

## Led multicolor



## Código

|                                   |   |
|-----------------------------------|---|
| Declaración de variables globales | <pre>// Práctica LED multicolor  const int greenLEDPin = 9; // LED conectado al pin 9 const int redLEDPin = 10; // LED conectado al pin 10 const int blueLEDPin = 11; // LED conectado al pin 11 const int redSensorPin = A0; // Patilla de control del potenciómetro LED rojo const int greenSensorPin = A1; // Patilla de control del potenciómetro LED verde const int blueSensorPin = A2; // Patilla de control del potenciómetro LED azul  int redValue = 0; // valor a escribir en el LED rojo int greenValue = 0; // valor a escribir en el LED verde int blueValue = 0; // valor a escribir en el LED azul int redSensorValue = 0; // valor de la variable del sensor de color rojo int greenSensorValue = 0; // valor de la variable del sensor de color verde int blueSensorValue = 0; // valor de la variable del sensor de color azul</pre> |
| Área de configuración<br>setup()  | <pre>// Configuración setup()  void setup() {     // inicializar la comunicación por el puerto serial a 9600 bps:     Serial.begin(9600);     // Los pines digitales como salidas     pinMode(greenLEDPin,OUTPUT);     pinMode(redLEDPin,OUTPUT);     pinMode(blueLEDPin,OUTPUT); }</pre>   |



Cuerpo principal del programa  
loop()

```

void loop() {
    // Leer los sensores primero:
    // leer el valor del potenciómetro con el led rojo:
    redSensorValue = analogRead(redSensorPin);
    // Al convertidor AD dar un tiempo de espera para resolver
    delay(5);
    // leer el valor del potenciómetro con el led verde:
    greenSensorValue = analogRead(greenSensorPin);
    // Al convertidor AD dar un tiempo de espera para resolver
    delay(5);
    // leer el valor del potenciómetro con el led azul:
    blueSensorValue = analogRead(blueSensorPin);
    // Al convertidor AD dar un tiempo de espera para resolver
    delay(5);
    // imprimir los valores en el monitor serial
    Serial.print("Los valores de los sensores \t rojo: ");
    Serial.print(redSensorValue);
    Serial.print("\t verde: ");
    Serial.print(greenSensorValue);
    Serial.print("\t azul: ");
    Serial.println(blueSensorValue);
    /*
    Para utilizar los valores del sensor para el LED, el convertidor AD proporciona un número de 10 bits, pero
    analogWrite () utiliza 8 bits. Necesitamos dividir las lecturas de los sensores por 4 para mantenerlos en el
    rango de la salida
    */
    redValue = redSensorValue/4;
    greenValue = greenSensorValue/4;
    blueValue = blueSensorValue/4;
    // imprimir los valores en el monitor serial
    Serial.print("Los valores de los sensores \t rojo: ");
    Serial.print(redValue);
    Serial.print("\t verde: ");
    Serial.print(greenValue);
    Serial.print("\t azul: ");
    Serial.println(blueValue);
    // Qué valor tiene PWM del LED.
    analogWrite(redLEDPin, redValue);
    analogWrite(greenLEDPin, greenValue);
    analogWrite(blueLEDPin, blueValue);
}

```



## Resultados y conclusiones

En esta práctica se trabajó con el microprocesador Arduino y se desarrollaron habilidades en su implementación para la creación de un prototipo, en donde podemos diferenciar la forma en la cual se deben de conectar los componentes *LED* multicolor y un potenciómetro, para que se puedan usar como dispositivos de entrada analógica y de salida digital. Utilizamos la función *analogRead()* para operar un circuito electrónico con las características generales del *LED* multicolor y como interconectarlo con el microcontrolador Arduino.

Un aprendizaje muy valioso en el uso del microprocesador Arduino es entender cómo funciona el convertidor analógico-digital, donde aumentamos la destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.

## Referencias

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Color mixing lamp. Arduino Projects Book* (pp. 53-60). Italia: Arduino LLC.

Specification for Zhongzhou led lamp (s.f.). Zhongzhou Electronics Ltd. Recuperado el 23 de septiembre de 2013, de <http://www.casadelled.com.ar/ZZ-GY-S0001B,ZL-503RCA2.pdf>

RGB Color Codes Chart. (s. f.). RapidTables, Online Reference & Tools. Recuperado el 3 de octubre de 2014, de [http://www.rapidtables.com/web/color/RGB\\_Color.htm](http://www.rapidtables.com/web/color/RGB_Color.htm) consultada.



## Actividad 8. Control de luminosidad con fotorresistencia

Norberto Alejandro Pérez Colín

### Antecedentes

*¿Qué es una fotorresistencia?*

Una fotorresistencia es un sensor que detecta cambios de luz por lo que su resistencia eléctrica varía por la acción de la iluminación. Una fotorresistencia está compuesta de material semiconductor, por esto también se les llama resistencias dependientes de luz por sus siglas *LDR* (*light dependent resistor*), es decir, aquellos que responden al cambio en la intensidad de la luz. En la figura 1 podemos observar, a la izquierda, dos símbolos utilizados para este dispositivo y a la derecha, la fotorresistencia físicamente.

Características:

- Son ideales para proyectos de iluminación o proyectos que necesiten controlar la luz ambiente.
- Resistencia (con luz): ~50k Ohm.
- Resistencia (oscuridad): ~1000kOhm.
- Voltaje Vmax: 150V.



Figura 1. Izquierda dos símbolos eléctricos y a la derecha la fotorresistencia físicamente.

Imágenes obtenidas del software Fritzing

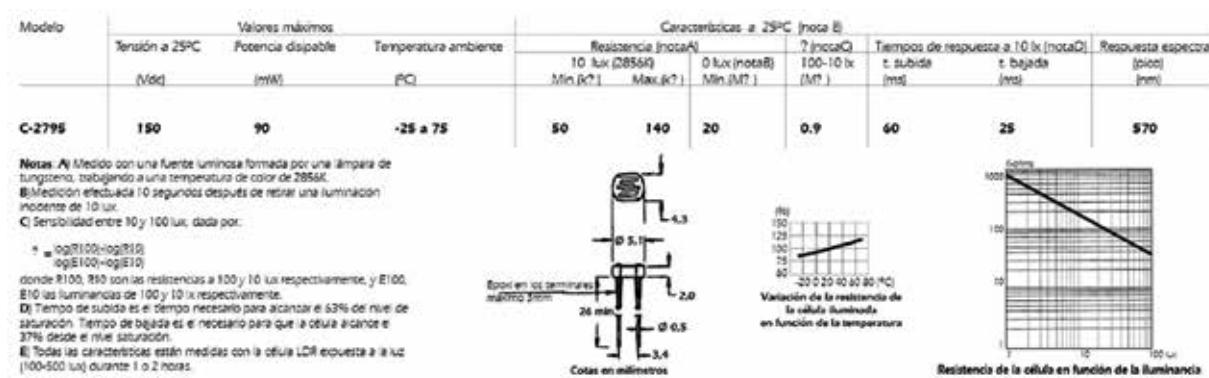


Figura 2. Características técnicas de la fotorresistencia. Imagen obtenida de datasheet

<http://www.electan.com/datasheets/cebek/CE-C2795.pdf>

## Introducción

*¿Qué es un convertidor analógico-digital?*

Un convertidor analógico-digital es un dispositivo electrónico que tiene la capacidad de convertir una señal analógica en un valor binario, se encarga de cambiar señales analógicas a digitales (0 y 1).



Figura 3. Esquema de un convertidor analógico-digital

El convertidor analógico-digital forma una relación entre su entrada (señal analógica) y su salida (señal digital) que depende de su resolución. La resolución establece la precisión con la que se interpreta a la señal original.

La resolución se puede saber cuando conocemos el valor máximo de la entrada analógica a convertir y la cantidad máxima de la salida en dígitos binarios.

$$\text{Resolución} = \frac{V_{ref}}{2^n} \text{ (donde } n \text{ son bits)}$$

Por ejemplo, un convertidor A/D de 8-bits puede convertir valores que van desde 0 V hasta el voltaje de referencia (Vref) y su resolución será de:

$$\text{Resolución} = \frac{V_{ref}}{2^8}$$

Lo que significa que mapeará los valores de voltaje de entrada, entre 0 y Vref volts, a valores enteros comprendidos entre 0 y 255 ( $2^n-1$ ).

La tarjeta Arduino uno utiliza un convertidor A/D de 10-bits, así que:

$$\text{Resolución} = \frac{V_{ref}}{2^{10}} = \frac{V_{ref}}{1024}$$

Por lo que mapeará los valores de voltaje de entrada entre 0 y Vref volts, a valores enteros comprendidos entre 0 y 1023 ( $2^n-1$ ). Con otras palabras, nuestros sensores analógicos están caracterizados con un valor comprendido entre 0 y 1023.



| Entrada analógica (volts) | Salida digital |
|---------------------------|----------------|
| 0.000                     | 0              |
| 0.625                     | 127            |
| 1.250                     | 255            |
| 1.875                     | 383            |
| 2.500                     | 511            |
| 3.125                     | 639            |
| 3.750                     | 767            |
| 4.5                       | 920            |
| 5.000                     | 1023           |

Tabla 1. Mapeo de señal analógica a señal digital

Las terminales A0 – A5 leen el voltaje cambiante y lo convierte a un número entre 0 y 1023, tabla 1. En otras palabras, el Arduino trabaja con voltajes de 0 a 5 volts, si el valor de entrada es 0 volts el valor digital es 0 (cero), cuando tenemos todo el recorrido analógico hay 5 volts que nos muestran el valor de entrada de 1023. La función en Arduino es *analogRead()* que devuelve un número entre 0 y 1023, el cual es proporcional a la cantidad de voltaje de 0 a 5 volts, El convertidor analógico-digital es de 10 bits por lo que  $2^{10} = 1024$ . Esto significa que va a asignar voltajes de entrada entre 0 y 5 volts en valores enteros entre 0 y 1023, no maneja cifras decimales el convertidor analógico-digital, si fuera el caso tomaría el entero superior o inferior, ya sea el caso.

En esta práctica utilizaremos un dispositivo electrónico llamado fotorresistencia, conoceremos cómo interpreta las entradas analógicas el microcontrolador Arduino, vamos a convertir una señal analógica que nos entrega la fotorresistencia a una señal digital, gracias a que posee en sus entradas analógicas un convertidor analógico-digital (A/D). La figura 4 muestra el esquema de conexión para lograr un control de luminosidad con una fotorresistencia.

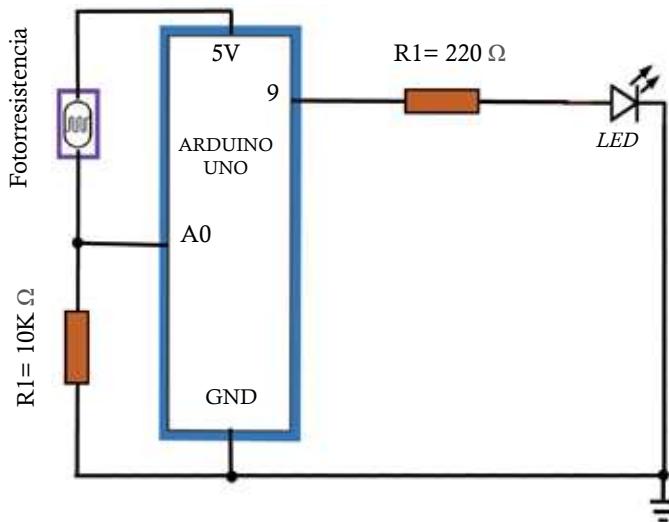


Figura 4. Esquema de conexiones del control de luminosidad con fotorresistencia



## Objetivo

- Hará uso y entenderá cómo funciona el elemento electrónico llamado fotorresistencia.
- Entenderá cómo funciona el convertidor analógico-digital del microcontrolador Arduino para las terminales analógicos.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar USB (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 LED (cualquier color).
- 1 resistencia de  $220\ \Omega$ .
- 1 resistencia de  $10K\ \Omega$ .
- 1 fotorresistencia.

## Desarrollo

Conectar la placa arduino a la *protoboard*, con la alimentación de 5 volts y *ground*, ver figura 5.

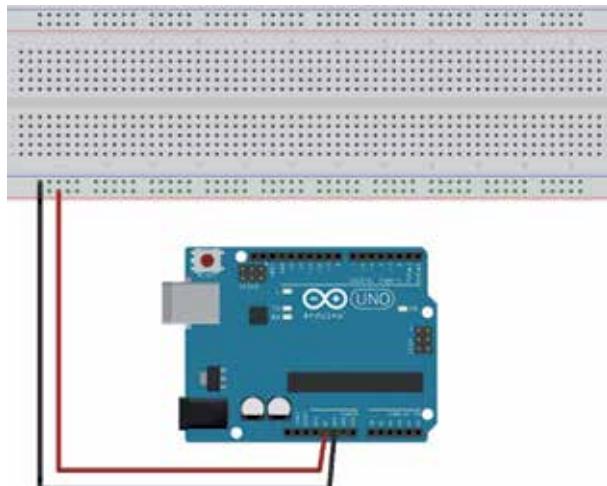


Figura 5. Esquema de conexiones del prototipo. Imagen obtenida del software Fritzing

Incorporamos el LED a la *protoboard* conectando a sus terminales a una resistencia de  $220\ \Omega$  y conectamos la fotorresistencia, acoplada con una resistencia de  $10K\Omega$ , como lo muestra la figura 6.



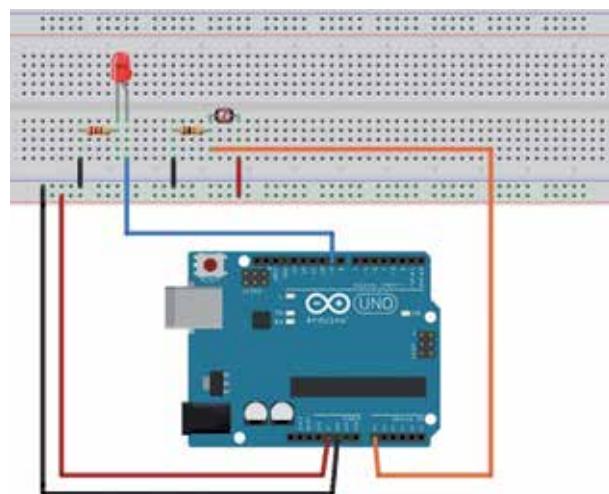


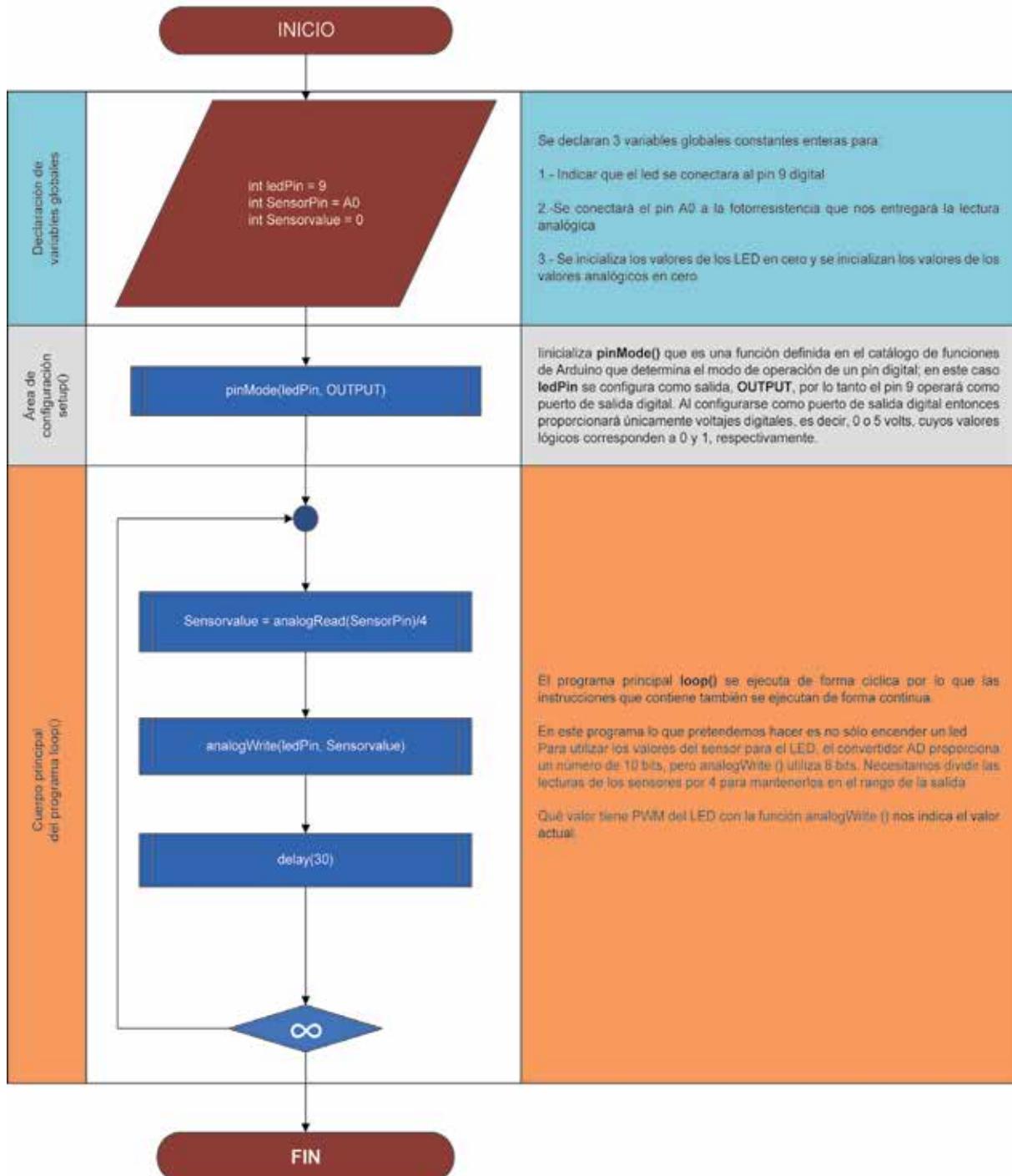
Figura 6. Esquema de conexiones del prototipo. Imagen obtenida del software Fritzing

Copiamos el código a nuestro *IDE* de Arduino para compilar y correr el programa.



## Diagrama de flujo

### Control de luminosidad con fotorresistencia



## Código

|                                      |  |
|--------------------------------------|--|
| Declaración de variables globales    | // Práctica entrada analógica y escritura PWM<br><br>int ledPin = 9; // LED connected to digital pin 9<br><br>int SensorPin = A0; // Pin de lectura de la fotorresistencia<br><br>int Sensorvalue = 0; //Inicialización del valor del sensor analógico |
| Área de configuración setup()        | // Configuración setup()<br><br>void setup() {<br><br>pinMode(ledPin, OUTPUT); // Configuramos el pin 9 como de salida<br><br>}  |
| Cuerpo principal del programa loop() | void loop() {<br><br>Sensorvalue = analogRead(SensorPin)/4; //Se divide en 4 la lectura del<br>//convertidor analógico-digital<br><br>analogWrite(ledPin, Sensorvalue); //Se envía la intensidad al led<br><br>delay(30);<br>}                         |



## Resultados y conclusiones

En esta práctica se trabajó con el microprocesador Arduino para desarrollar habilidades en la implementación de un prototipo y controlar la luminosidad de un *LED*, con una fotorresistencia, en donde el aprendizaje se basa en entender cómo funciona el convertidor analógico-digital que tiene el microcontrolador Arduino para las terminales analógicos.

Otro aprendizaje que se desarrolló en la práctica es el uso y entender cómo funciona el elemento electrónico llamado fotorresistencia, donde podemos diferenciar la forma en la cual se deben de conectar los componentes a la entrada analógica y la salida digital. Utilizamos instrucciones para desarrollar algoritmos a nivel de programación, la función *analogRead()* y *analogWrite()*, para operar un circuito electrónico y aprendimos cómo interconectarlo con el microcontrolador Arduino

## Referencias

Conversor Analógico-Digital (A/D) (s.f.). Arduino LLC. Recuperado el 7 de noviembre de 2013, de <http://playground.arduino.cc/ArduinoNotebookTraducion/Appendix6>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 52-60). Italia: Arduino LLC.

Fotoconductores de una pieza (fotorresistencias) (s.f.). Instituto Tecnológico de la Laguna. Recuperado el 7 de noviembre de 2013, de [http://www.uib.cat/depart/dfs/GTE/education/industrial/tec\\_electronica/teoria/resistores\\_variables.pdf](http://www.uib.cat/depart/dfs/GTE/education/industrial/tec_electronica/teoria/resistores_variables.pdf)



## Actividad 9. Detector de blanco

Norberto Alejandro Pérez Colín

### Antecedentes

*¿Qué es un diodo y cómo funciona?*

Es el dispositivo electrónico más sencillo, se puede encontrar en casi cualquier circuito electrónico. El diodo es un componente electrónico que permite el flujo de la corriente eléctrica en un solo sentido. La flecha de su símbolo en un esquema eléctrico nos indica la dirección en la corriente eléctrica, como se muestra en la figura 1.



Figura 1. La figura izquierda símbolo eléctrico y a la derecha el diodo físicamente.

Imágenes obtenidas del software Fritzing

Los diodos se fabrican en material de silicio y germanio. El diodo puede funcionar de dos formas diferentes:

**Polarización directa.** Cuando la corriente eléctrica ( $i_d$ ) circula por el diodo en sentido directo, como se muestra en la figura 2, es decir, del ánodo al cátodo, siguiendo la ruta de la flecha (la del diodo). En este caso la corriente atraviesa el diodo comportándose prácticamente como un corto circuito, entonces el diodo conduce.

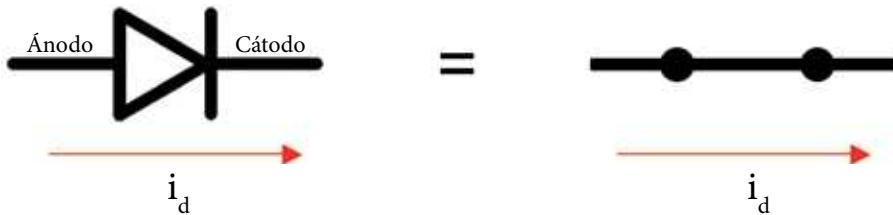


Figura 2. Diodo en polarización directa. Imágenes obtenidas del software Fritzing

**Polarización inversa.** Es cuando la corriente en el diodo desea circular en sentido opuesto a la flecha (la flecha del diodo), del cátodo al ánodo. En este caso, la corriente eléctrica no atraviesa el diodo y su funcionamiento es como un circuito abierto, ver figura 3:

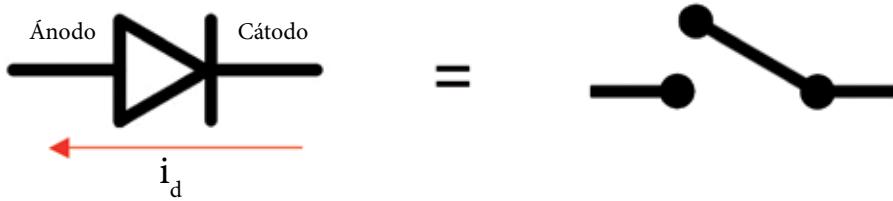


Figura 3. Diodo en polarización inversa. Imágenes obtenidas del software Fritzing



## Introducción

En esta práctica realizaremos el encendido de un *LED* por medio de una detector del color blanco, para esto nos apoyaremos esencialmente en un sensor de infrarrojos CNY70, como entrada digital. El CNY70 es un sensor óptico infrarrojo, que se muestra en la figura 4, con rango de corto alcance con menos de 5 mm que se utiliza para detectar colores de alguna superficie. Su uso más común es en la construcción de pequeños robots sigue líneas. Contiene un emisor de radiación infrarroja (fotodiodo) y un receptor (fototransistor). El fotodiodo emite un haz de radiación infrarroja, el fototransistor recibe ese haz de luz cuando se refleja sobre alguna superficie de un objeto. Para esta práctica buscaremos superficies de color “blanco”, las cuales dependiendo de la cantidad de luz recibida por el fototransistor, éste envía una señal de retorno como entrada digital al microcontrolador Arduino.

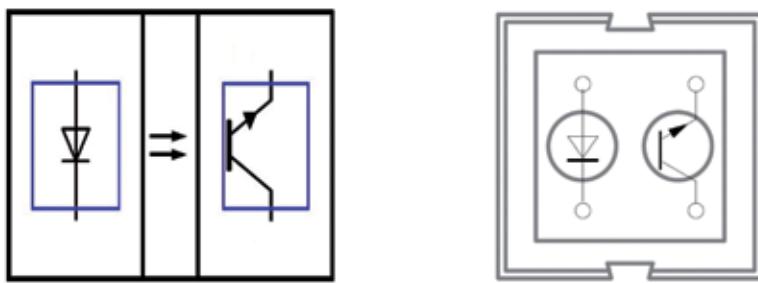


Figura 4. La figura izquierda símbolo eléctrico y a la derecha el encapsulado físicamente.

Imagen izquierda obtenida del software *Fritzing* y derecha <http://html.alldatasheet.com/html-pdf/26332/VISHAY/CNY70/179/1/CNY70.html> del 12/nov/2014

Tanto en el emisor como en el receptor están en el mismo encapsulado, colocadas dos pequeñas ventanas ópticas la que permiten ver el haz de luz, de modo que cuando un objeto pasa el haz de luz con color “blanco”, el receptor la detecta. El circuito electrónico de la práctica del “detector de blanco” se muestra en la figura 5.

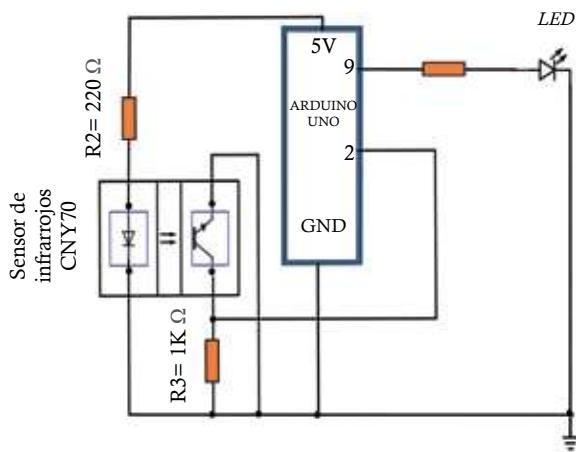


Figura 5. Esquema de conexiones de la práctica “detector de blanco”



Las características técnicas del sensor óptico infrarrojo se muestran a continuación:

## CNY70

Vishay Telefunken



### Absolute Maximum Ratings

#### Input (Emitter)

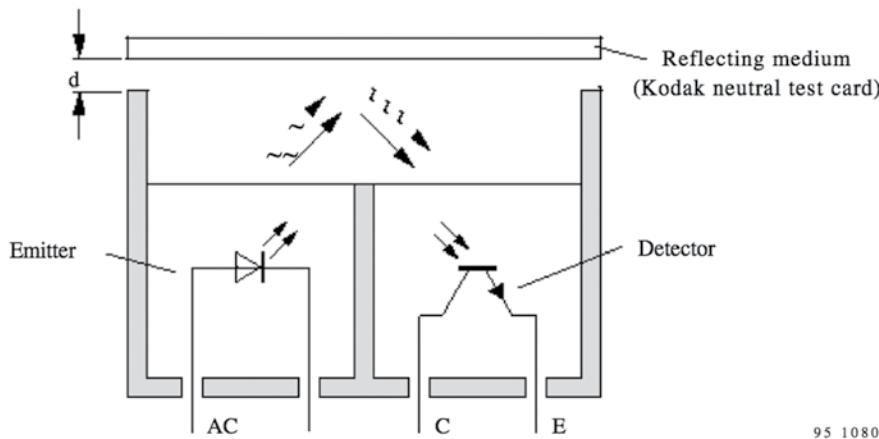
| Parameter             | Test Conditions                 | Symbol    | Value | Unit |
|-----------------------|---------------------------------|-----------|-------|------|
| Reverse voltage       |                                 | $V_R$     | 5     | V    |
| Forward current       |                                 | $I_F$     | 50    | mA   |
| Forward surge current | $t_p \leq 10 \text{ ms}$        | $I_{FSM}$ | 3     | A    |
| Power dissipation     | $T_{amb} \leq 25^\circ\text{C}$ | $P_V$     | 100   | mW   |
| Junction temperature  |                                 | $T_j$     | 100   | °C   |

#### Output (Detector)

| Parameter                 | Test Conditions                 | Symbol    | Value | Unit |
|---------------------------|---------------------------------|-----------|-------|------|
| Collector emitter voltage |                                 | $V_{CEO}$ | 32    | V    |
| Emitter collector voltage |                                 | $V_{ECO}$ | 7     | V    |
| Collector current         |                                 | $I_C$     | 50    | mA   |
| Power dissipation         | $T_{amb} \leq 25^\circ\text{C}$ | $P_V$     | 100   | mW   |
| Junction temperature      |                                 | $T_j$     | 100   | °C   |

#### Coupler

| Parameter                 | Test Conditions                      | Symbol    | Value       | Unit |
|---------------------------|--------------------------------------|-----------|-------------|------|
| Total power dissipation   | $T_{amb} \leq 25^\circ\text{C}$      | $P_{tot}$ | 200         | mW   |
| Ambient temperature range |                                      | $T_{amb}$ | -55 to +85  | °C   |
| Storage temperature range |                                      | $T_{stg}$ | -55 to +100 | °C   |
| Soldering temperature     | 2 mm from case, $t \leq 5 \text{ s}$ | $T_{sd}$  | 260         | °C   |



95 10808

Figura 6. Características técnicas del sensor CNY70 <http://html.alldatasheet.com/html-pdf/26332/VISHAY/CNY70/179/1/CNY70.html> consultado el 12/nov/2014.



## Objetivos

- Hará uso y entenderá cómo funciona el elemento electrónico llamado sensor de infrarrojos
- Aumentará su destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.
- Entenderá como el *hardware* y *software* del microcontrolador Arduino interactúan a través de las terminales digitales.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar USB (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 LED (cualquier color).
- 2 resistencias de  $220\ \Omega$ .
- 1 resistencia de  $1K\ \Omega$ .
- 1 sensor de infrarrojos CNY70.

## Desarrollo

Conectar la placa arduino a la *protoboard*, con la alimentación de 5 volts y *ground*, como en la figura 7.

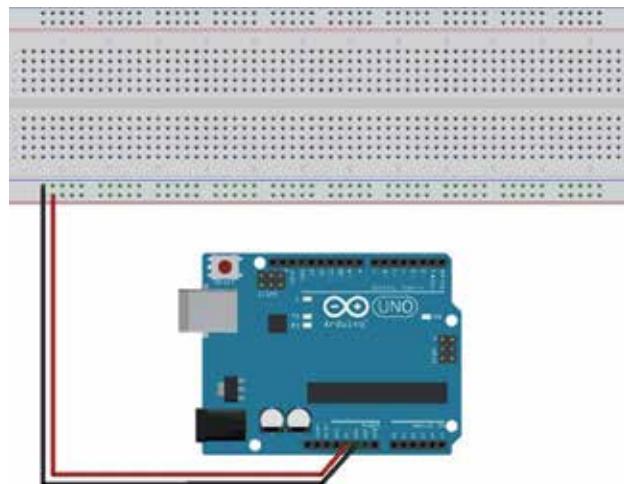


Figura 7. Circuito de la práctica “detector de blanco”. Imagen obtenida del software Fritzing



Incorporamos el *LED* a la *protoboard* conectando a sus pines una resistencia de  $220\Omega$  y conectamos el sensor de infrarrojos acoplado con una resistencia de  $220\Omega$  y una de  $1K\Omega$ , como se observa en la figura 8.

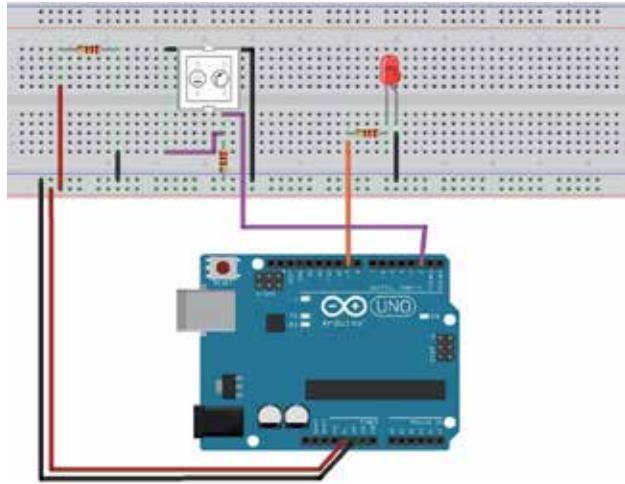


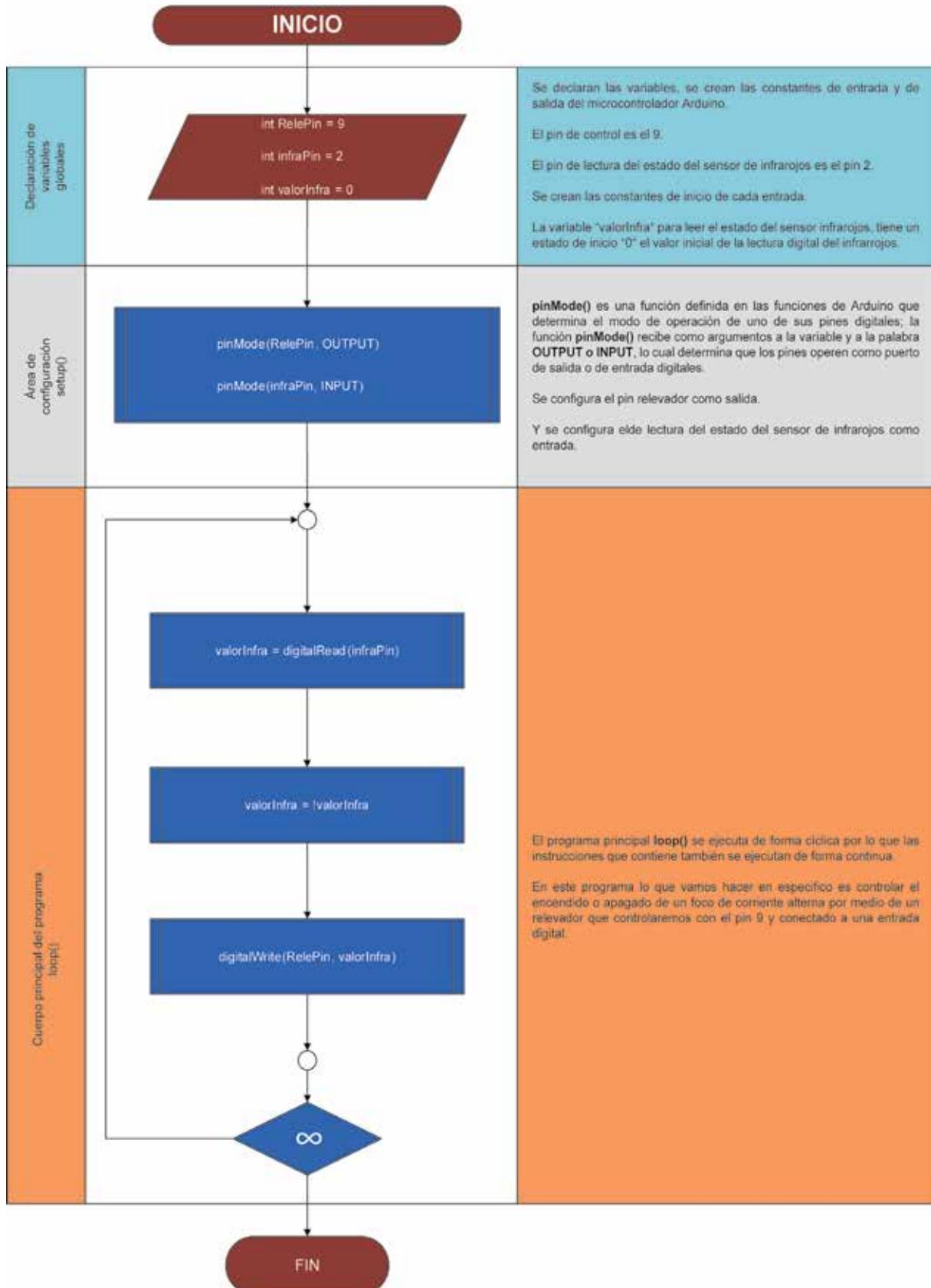
Figura 8. Circuito completo de la práctica “detector de blanco”. Imagen obtenida del software *Fritzing*

Copiamos el código a nuestro *IDE* de Arduino para compilar y correr el programa.



## Diagrama de flujo

### “ Detector de blanco ”



## Código

|                          |  |
|--------------------------|--|
| Declaración de Variables | <pre>///"Detector de blanco " // Se crean las constantes de entrada y de salida en el microcontrolador Arduino int ledPin = 9; // pin de LED int infraPin = 2; // pin del infrarrojos utilizado como entrada digital int valorInfra = 0; // Valor inicial de la lectura digital del infrarrojos.</pre>   |
| Setup()                  | <pre>void setup() { pinMode(ledPin, OUTPUT); // Inicializa el pin del LED2 como salida digital pinMode(infraPin, INPUT); // Inicializa el pin 2 como entrada digital }</pre>   |
| Loop()                   | <pre>//la función loop() se ejecuta cíclicamente y de manera ininterrumpida void loop() { valorInfra = digitalRead(infraPin); // Lee el valor de la entrada 2, el valor del infrarrojo valorInfra = !valorInfra; // Se asigna a valorInfra el valorInfra negado digitalWrite(ledPin, valorInfra); // Escribe en el pin 9 el valor negado }</pre> |

## Resultados y conclusiones

En esta práctica se trabajó con el microprocesador Arduino y se desarrollaron habilidades en su implementación para la creación de un prototipo, en donde se puede decir que se cumplieron los objetivos completamente donde se trabajó en de diferenciar e identificar los pines digitales de entrada y salida del microcontrolador Arduino. Asimismo, se utilizó y entendió cómo funciona el elemento electrónico llamado sensor de infrarrojos.

El desarrollo de nuestras habilidades aumento en la destreza y armado de circuitos electrónicos, además de entender como el *hardware* y *software* del microcontrolador Arduino interactúan a través de los pines digitales.

## Referencias

digitalWrite() (s.f.). Arduino LLC. Recuperado el 30 de agosto de 2014, de <http://arduino.cc/en/pmwiki.php?n=Reference/DigitalWrite>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). Get to know your tools. *Arduino Projects Book* (pp. 24-41). Italia: Arduino LLC.

EL DIODO (s.f.). Departamento de Tecnología Electrónica, Universidad de Vigo. Recuperado el 30 de agosto de 2014, de <http://www.dte.uvigo.es/recursos/potencia/ac-dc/archivos/diodo.htm>

CNY70 Datasheet (s.f.). Alldatasheet.com. Recuperado el 30 de agosto de 2014, de <http://html.alldatasheet.com/html-pdf/26332/VISHAY/CNY70/179/1/CNY70.html>



## Actividad 10. Detector de blanco con acoplamiento de corriente alterna

Norberto Alejandro Pérez Colín

### Antecedentes

*¿Qué es un relevador o relé?*

Es un dispositivo electromecánico cuyo funcionamiento es realmente como interruptor controlado por un circuito electrónico. Por medio de una bobina y un electroimán, se acciona un contacto que permiten abrir o cerrar el circuito eléctrico. En la figura 1 se puede observar el esquema de un relevador en sus dos estados.

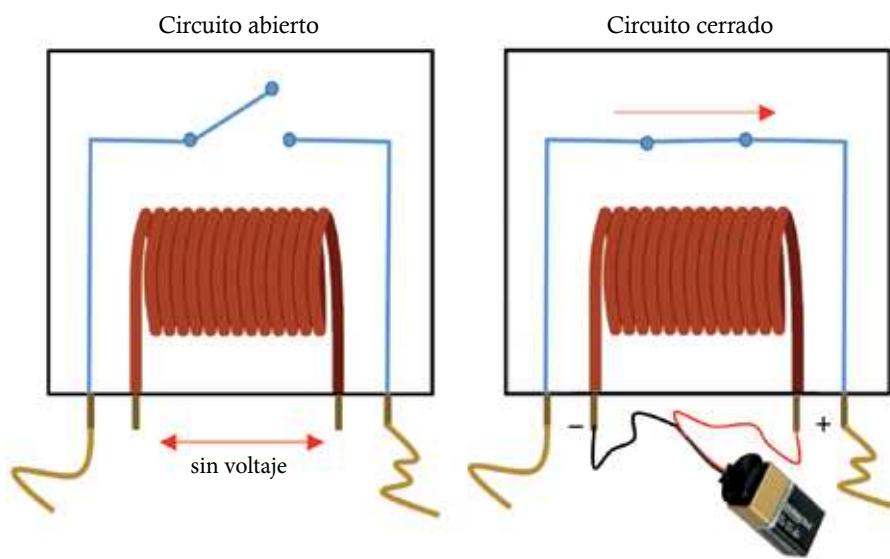


Figura 1. La figura izquierda relevador abierto y derecha relevador cerrado. Imagen obtenida del software Fritzing

Los relevadores se manejan en circuitos eléctricos esencialmente para conmutar a distancia, para la conmutación de alta tensión o de alta corriente. Son particularmente valiosos porque pueden controlar estas altas tensiones y corrientes con sólo un pequeño voltaje. Los relevadores funcionan como interruptores de alimentación de corriente alterna y mantienen las señales de control con aislamiento.

### Introducción

En esta práctica realizaremos el encendido de un foco incandescente por medio de un detector del color blanco. Para esto nos apoyaremos esencialmente en un sensor de infrarrojos CNY70 como entrada digital (como se trabajó en la práctica “Detector de blanco”). El CNY70 emite un haz de radiación infrarroja, el fototransistor recibe ese haz de luz cuando se refleja sobre alguna superficie de un objeto. Para esta práctica buscaremos superficies de color “blanco”, las cuales, dependiendo



de la cantidad de luz recibida por el fototransistor, envían una señal de retorno como entrada digital al microcontrolador Arduino. Una salida digital del microcontrolador Arduino la utilizaremos para activar el relevador que acoplado a la corriente eléctrica alterna que controlaremos para encender un foco incandescente. El esquema de conexiones de la práctica “Detector de blanco con acoplamiento de corriente alterna” se muestra en la figura 2.

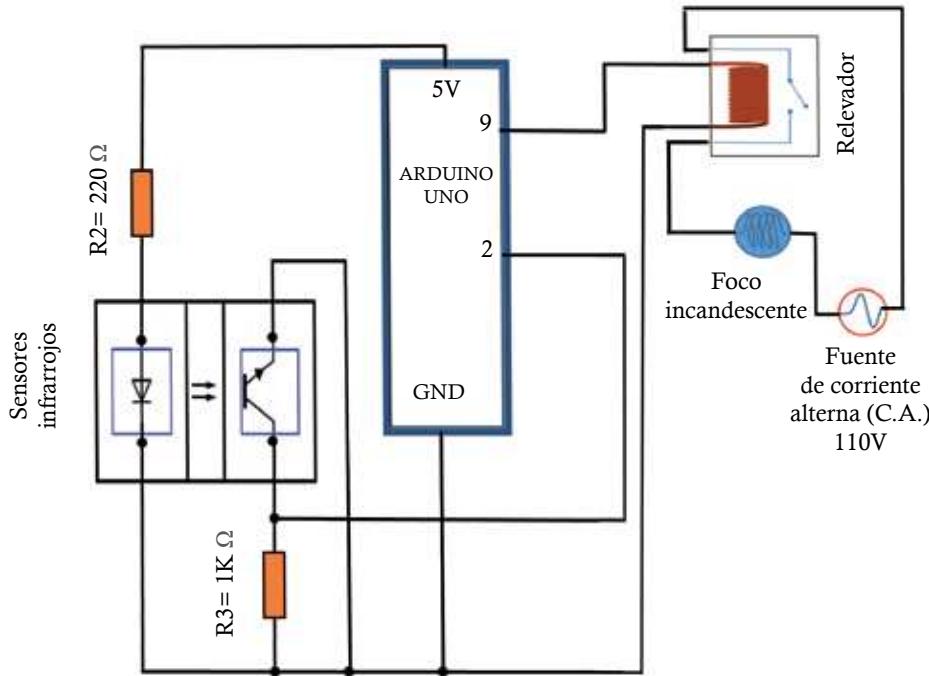


Figura 2. Esquema de conexiones de la práctica “Detector de blanco con acoplamiento de corriente alterna”

Cuando un objeto pasa el haz de luz con color blanco, el receptor lo detecta y enciende el foco incandescente. El relevador que vamos a utilizar para los fines de esta práctica es el MD-5 y sus características técnicas se observan en la figura 3.



| COIL RATING (at 20°C) |                               |                       |                                |                       |                        |                              |
|-----------------------|-------------------------------|-----------------------|--------------------------------|-----------------------|------------------------|------------------------------|
| NOMINAL VOLTAGE (VDC) | COIL RESISTANCE (Ω) (+/- 10%) | POWER CONSUMPTION (W) | NOMINAL CURRENT (mA) (+/- 10%) | PULL IN VOLTAGE (VDC) | DROP OUT VOLTAGE (VDC) | MAX. ALLOWABLE VOLTAGE (VDC) |
| 5V                    | 500 Ω                         | 0.5W                  | 10 mA                          | 3.75V                 | 0.8V                   | 15V                          |
| 12V                   | 1000 Ω                        | 0.144W                | 12 mA                          | 8.6V                  | 0.6V                   | 30V                          |

| PERFORMANCE (at initial value)            |   |
|---|---|
| Item                                      | Type  |
| Contact Resistance                        | 150mΩ Max. (initial value)  |
| Operate Time                              | 0.5msec Max.  |
| Release Time                              | 0.1msec Max.  |
| Dielectric Strength                       | AC 1500V (1min)<br>AC 250V (1min)   |
| between coil & contact<br>between contact |   |
| Insulation Resistance                     | 100MΩ Min.(DC 500V)   |
| Operating Ambient Temperature             | -40°C ~ +85°C   |
| Humidity                                  | 35 to 85% RH  |
| Vibration Resistance                      | 10G (10~55Hz) (Dual amplitude:1.5mm)  |
| Shock Resistance                          | 10G   |
| Life Expectancy                           | 100,000,000 ops. Min. (1800 ops./h)<br>100,000 ops. Min. ( 0.5A 20VDC , 0.1A 100VDC, 0.5A 20VAC, 0.1A 100VAC,<br>resistive load 1200 ops./h)<br>50,000,000 ops. Min. (5VDC, 10mA, resistive load 1200 ops./h) |
| Mechanically<br>Electrically              |   |
| Weight                                    | 2g (about)  |

Figura 3. Características técnicas del relevador MD-5 [http://us.100y.com.tw/pdf\\_file/29-SUN%20HOLD\\_MD-X.pdf](http://us.100y.com.tw/pdf_file/29-SUN%20HOLD_MD-X.pdf)  
consultado el 14/enero/2015.

## Objetivos

- Hará uso y entenderá cómo funcionan los dispositivos llamados sensor de infrarrojos y relevador.
- Entenderá cómo se interconectan y acoplan el *hardware* utilizado en esta práctica con la corriente alterna.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar USB (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 foco incandescente.
- 1 *socket*.
- 1 resistencia de 220 Ω.
- 1 resistencia de 1K Ω.
- 1 cable de corriente alterna.
- 1 sensor de infrarrojos CNY70.



## Desarrollo

Conectar la placa arduino a la *protoboard*, con la alimentación de 5 volts y *ground*, como en la figura 4.

Incorporamos el foco, el *socket* y el cable de CA (corriente alterna) a la *protoboard* conectando a sus terminales una resistencia de  $220\ \Omega$  y conectamos el sensor de infrarrojos acoplado con una resistencia de  $220\Omega$  y una de  $1K\ \Omega$ .

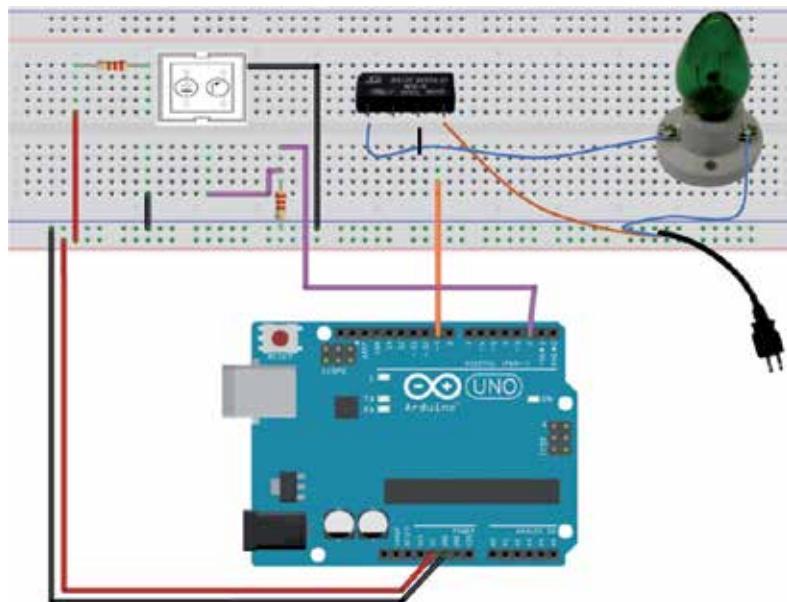


Figura 4. Circuito completo de la práctica “Detector de blanco con acoplamiento de corriente alterna“.

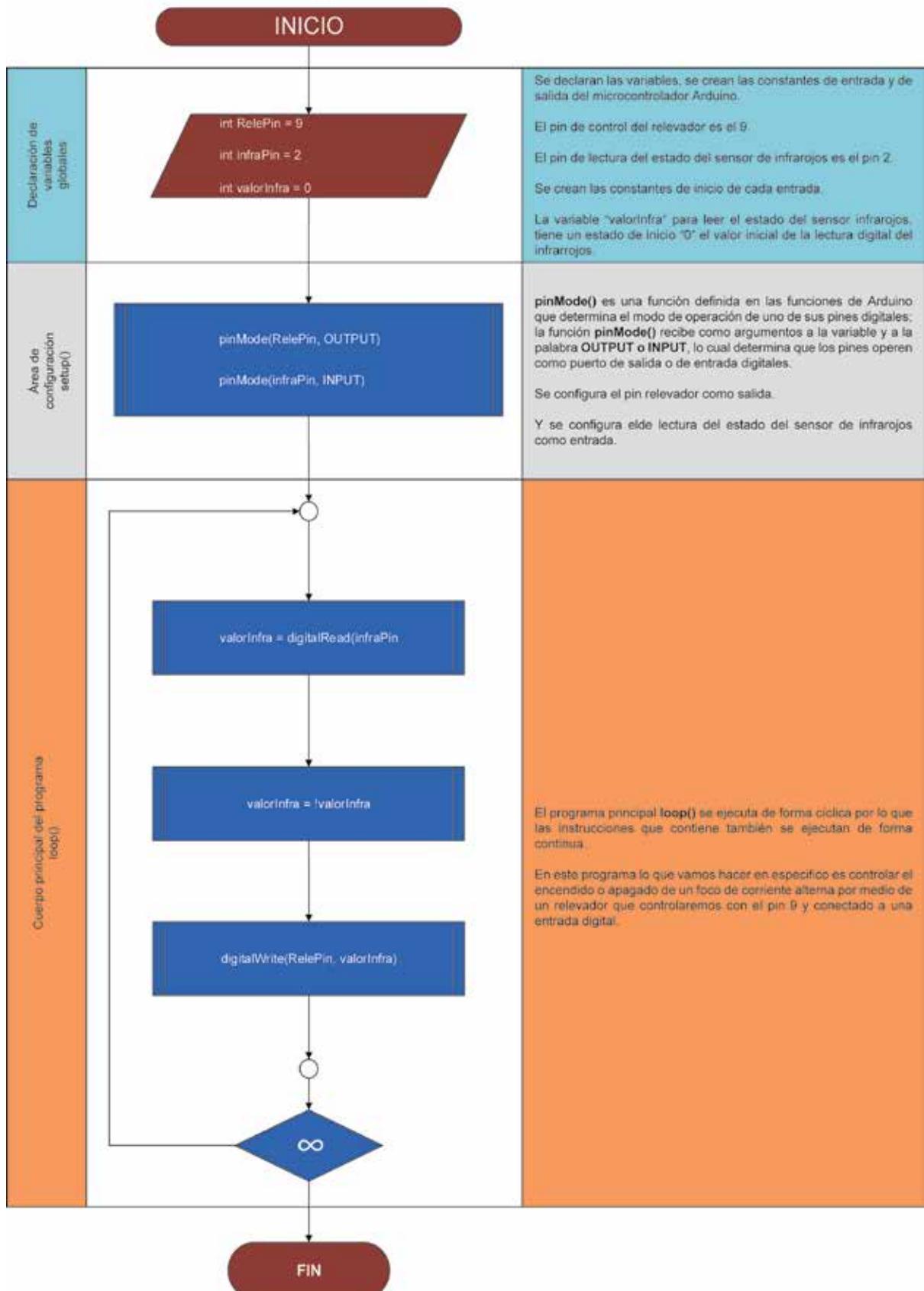
Imagen obtenida del software *Fritzing*

Copiamos el código a nuestro *IDE* de Arduino para compilar y correr el programa.



## Diagrama de flujo

### “Detector de blanco con acoplamiento de corriente alterna”



## Código

|                          |  |
|--------------------------|--|
| Declaración de Variables | //Detector de blanco con acoplamiento de corriente alterna<br>// Se crean las constantes de entrada y de salida en el microcontrolador Arduino<br><br><b>int</b> RelePin = 9; // pin del relevador<br><b>int</b> infraPin = 2; // pin del infrarrojos utilizado como entrada digital<br><b>int</b> valorInfra = 0; // Valor inicial de la lectura digital del infrarrojos. |
| Setup ()                 | <b>void</b> setup() {<br>pinMode(RelePin, OUTPUT); // Inicializa el pin del LED como salida digital<br>pinMode(infraPin, INPUT); // Inicializa el pin 2 como entrada digital<br>}<br><br>//la función loop () se ejecuta cíclicamente y de manera ininterrumpida   |
| Loop ()                  | <b>void</b> loop() {<br>valorInfra = digitalRead(infraPin); //Lee el valor de la entrada 2, el valor del infrarrojo<br>valorInfra = !valorInfra; // Se asigna a valorInfra el valorInfra negado<br>digitalWrite(RelePin, valorInfra); // Escribe en el pin 9 el valor negado<br>}  |

## Resultados y conclusiones

En esta práctica se trabajó con el microprocesador Arduino para la creación de un prototipo electrónico, que controlará un foco incandescente de corriente alterna.

Se entendió cómo funciona el dispositivo electrónico llamado sensor de infrarrojos y el dispositivo electromecánico relevador. Asimismo, se logró el aprendizaje de cómo se interconectan y acoplan el *hardware* utilizado en esta práctica con la corriente alterna.

Donde se puede decir que se cumplieron los objetivos completamente en el desarrollo de nuestras habilidades y se aumento la destreza y armado de circuitos electrónicos, además de entender cómo el *hardware* y *software* del microcontrolador Arduino interactúan.

## Referencias

Conversor Analógico-Digital (A/D) (s.f.). Arduino LLC. Recuperado el 10 de noviembre de 2014, de <http://playground.arduino.cc/ArduinoNotebookTraducion/Appendix6>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 24-41). Italia: Arduino LLC.

SUN HOLD MD-5 datasheet (s.f.). Centenary Materials Co., Ltd Recuperado el 10 de noviembre de 2014, de [http://us.100y.com.tw/pdf\\_file/29-SUN%20HOLD\\_MD-X.pdf](http://us.100y.com.tw/pdf_file/29-SUN%20HOLD_MD-X.pdf)

Contactores y relevadores (2004). Universidad Tecnológica de Puebla. Recuperado de <http://electricidad.utpuebla.edu.mx/Manuales%20de%20asignatura/4to%20cuatrimestre/Contactores%20y%20relevadores.pdf>



## Actividad 11. Fading

Norberto Alejandro Pérez Colín / Juventino Ávila Ramos

### Antecedentes

*¿Qué es la estructura de control “FOR”?*

Las estructuras de control son estructuras en programación, formas de trabajo que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos llevan a la solución de problemas.

Las estructuras de control nos permiten controlar el flujo del programa, tomar alguna decisión y realizar acciones repetitivas dependiendo de las condiciones que establezcamos.

La figura 1 nos muestra la estructura de control *for*, que se utiliza en situaciones en que conocemos la cantidad de veces que queremos que se ejecute el bloque de instrucciones.

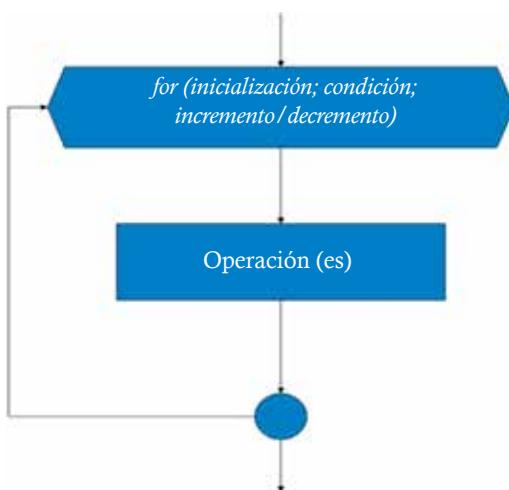


Figura 1. Estructura de control “FOR”

En su forma básica esta estructura de control requiere de una variable entera que cumpla la función de un contador de vueltas. En la sección indicada como inicialización, se coloca casi siempre el nombre de la variable que será el contador, asignándole a dicha variable un valor inicial. En la sección de condición se coloca una condición que deberá ser verdadera para que el ciclo prosiga (en caso de un falso, el ciclo se detendrá). Finalmente, en la sección de incremento/decremento, se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa).

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de inicialización. Después se verifica, automáticamente, si la condición es verdadera. En este caso, se ejecuta el bloque de operaciones y al finalizar el mismo, se ejecuta la instrucción que se haya colocado. Inmediatamente, se regresa al valor de la condición, y así prosigue hasta que dicha condición entregue un falso.



- Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un *for*
- Casos particulares del ciclo *for*;
1. El ciclo comienza en uno y se va incrementando de uno en uno, éste es el caso más general.
  2. Pero el valor inicial puede ser diferente de uno, ejemplo;  
$$\text{for}(x=5; x \leq 15; x=x+1).$$
  3. El valor inicial puede ser negativo, ejemplo;  
$$\text{for}(x = -3; x \leq 8; x=x+1);$$
  4. Los incrementos también pueden ser diferentes al de uno en uno, ej.  
$$\text{for}(x=1; x \leq 20; x=x+3).$$
  5. Incluso pueden ser decrementos, sólo que en este caso, recordar;
    - El valor inicial de la variable debe ser mayor que el valor final.
    - Cambiar el sentido de la condición.
  6. Solo para los casos de incrementos y decrementos de una en una, la unidad sustituir en el *for*:

El  $x = x + 1$  por  $x++$

El  $x = x - 1$  por  $x-$

## Introducción

En esta práctica realizaremos el encendido y apagado gradual de un *LED*, para esto nos apoyaremos esencialmente en la programación, con la estructura de control “*for*”. Y el circuito electrónico se muestra en la figura 2.

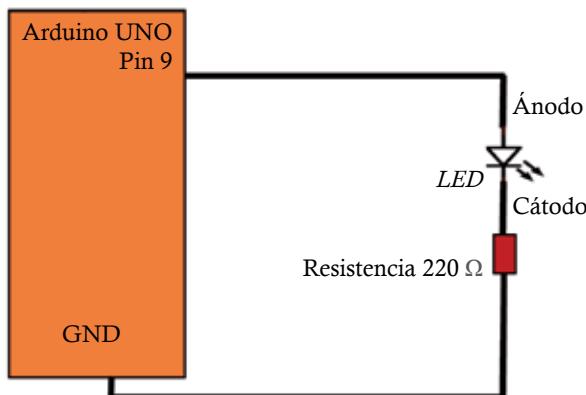


Figura 2. Esquema de conexión de un *LED* al puerto digital *Pin 9*

¿Qué es la estructura de control “*FOR*”, en *Arduino*?

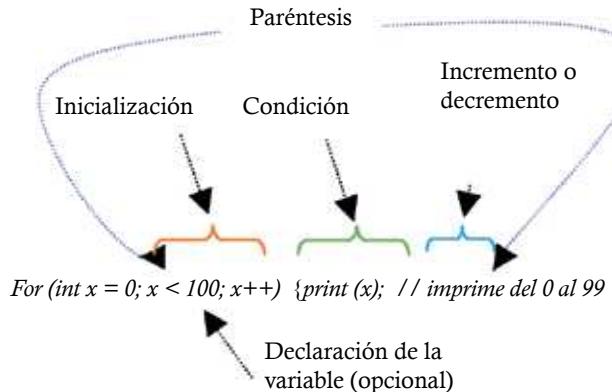
La sentencia *for* se utiliza para realizar repeticiones en bloque de sentencias encerradas entre llaves. Un contador se utiliza para incrementar y terminar el bucle o ciclo. La sentencia *for* es útil para cualquier operación que se repita y se utiliza a menudo en combinación con arreglos para operar las colecciones de datos.



Su sintaxis es como se indica a continuación, hay tres partes en la cabecera de bucle:

```
for (inicialización; condición; incremento/decremento) { //declaración(s);  
}
```

Ejemplo:



*¿Qué es la función analogWrite de Arduino?*

La función `analogWrite` nos permite escribir un valor analógico (onda PWM) para un *pin* digital. Se puede utilizar para encender un *LED* a diferentes brillos o accionar un motor a distintas velocidades. Después de llamar a `analogWrite()`, el *pin* generará una onda cuadrada estable del ciclo de trabajo especificado hasta la siguiente llamada a `analogWrite()` en el mismo *pin*.

El formato de lectura para la programación, su sintaxis es:

```
analogWrite(pin,valor)
```

Esta función sólo puede utilizarse en los pines 3, 5, 6, 9, 10, 11, el cual se quiere generar la señal PWM (localizados en Arduino como ~)

El valor puede ser especificado como una variable o constante con valor entre 0 y 255; en donde 0 siempre es apagado y 255 siempre encendido.

## Objetivo

- Describirá e identificará las características generales de la estructura de control *FOR*.
- Diferenciará e identificará los pines digitales y cómo interactúan con la función `analogWrite` del microcontrolador Arduino.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar USB (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 *LED* (cualquier color).
- 1 resistencia de 220 Ω.



## Desarrollo

De acuerdo con la figura 3, comenzaremos por energizar la *protoboard* y cómo se deben conectar el ánodo del *LED* a la terminal 9 del Arduino, el cátodo debe conectarse al voltaje de referencia (*GND*) a través de una resistencia de  $220\ \Omega$ , la cual tiene el propósito de limitar la corriente que pasa por el *LED* y así evitar posibles daños a este dispositivo.

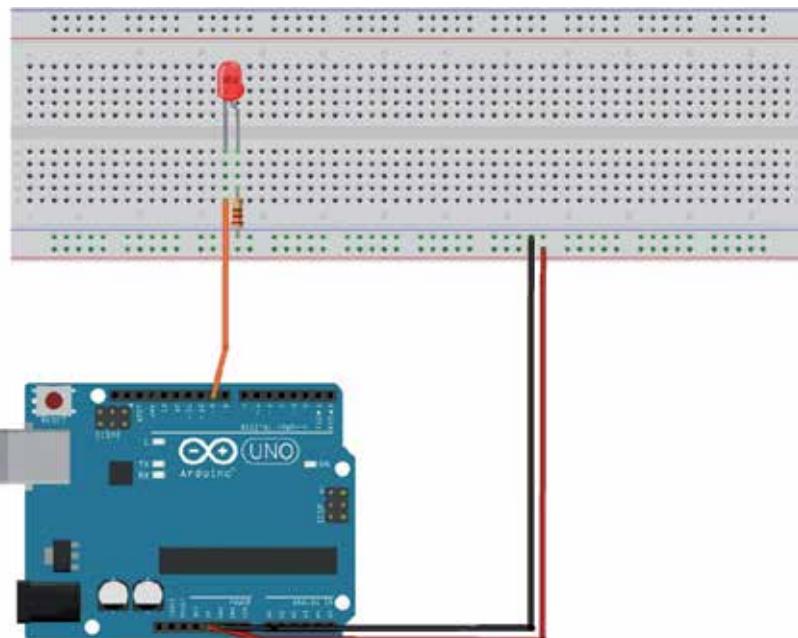


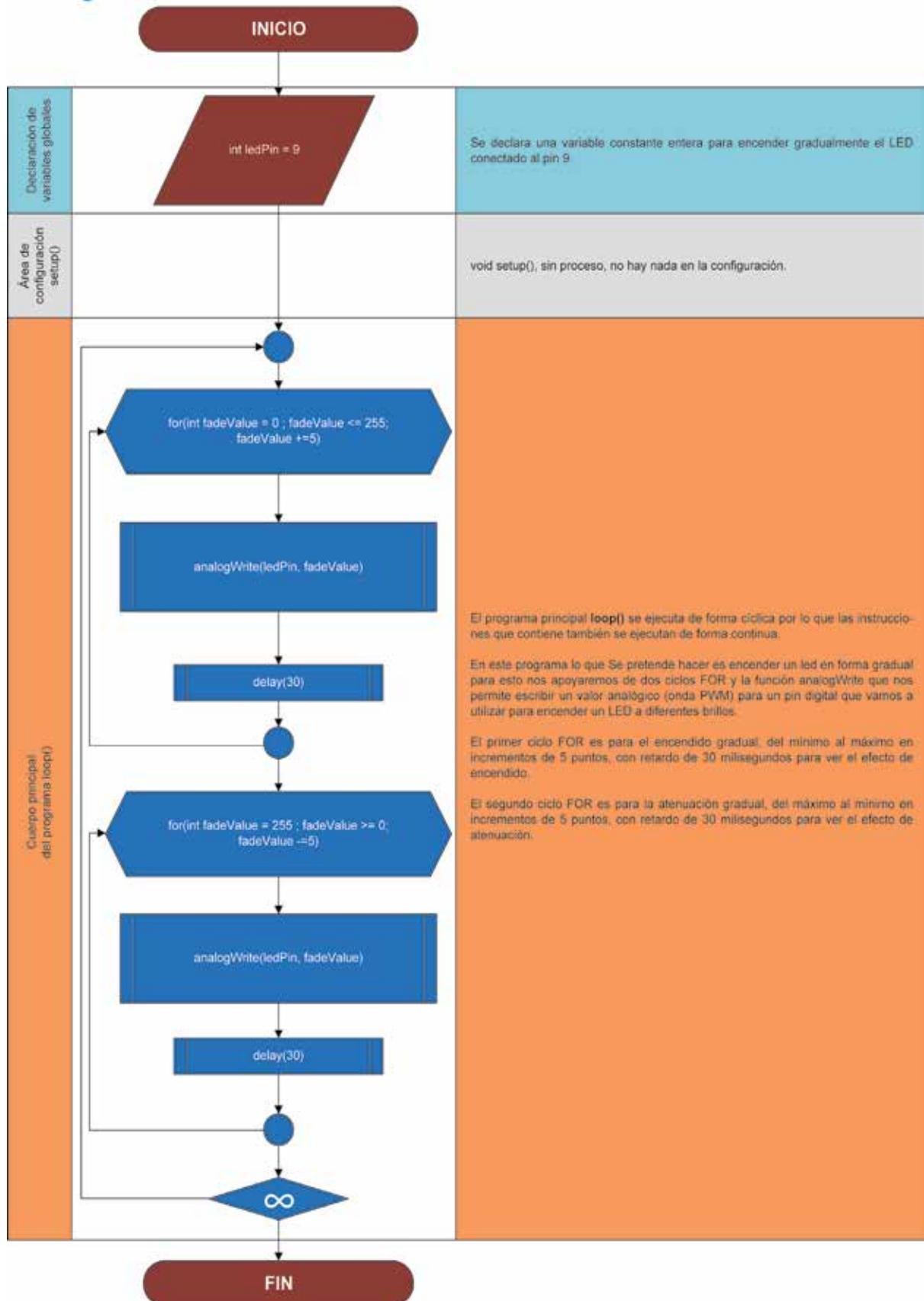
Figura 3. Esquema del prototipo “*Fading*”. . Imagen obtenida del software *Fritzing*

Conectamos la tarjeta Arduino a la computadora mediante el cable *USB*; ejecutamos el entorno de programación de Arduino (*IDE*); abrimos el *sketch* de los ejemplos en analógicos “*Fading*” y procedemos a cargar en el Arduino. El código se explica en las dos siguientes secciones.



## Diagrama de flujo

“ Fading ”



## Código

|   |  |
|---|--|
| Declaración de variables globales       | // Fading<br><b>int ledPin = 9;</b> // LED conectado al pin digital 9 con PWM  |
| Configuración                           | // Configuración setup()<br><b>void setup()</b> {<br>// no hay nada en la configuración<br>}<br><br><b>void loop()</b> {   |
| Cuerpo principal del programa<br>loop() | // El encendido gradual, del mínimo al máximo en incrementos de 5 puntos<br><b>for(int fadeValue = 0 ; fadeValue &lt;= 255; fadeValue +=5) {</b><br>//Se establece el rango de 0 a 255<br><b>analogWrite(ledPin, fadeValue);</b><br>// esperar 30 milisegundos para ver el efecto de encendido<br><b>delay(30);</b><br>}<br>// La atenuación gradual, del máximo al mínimo en incrementos de 5 puntos<br><b>for(int fadeValue = 255 ; fadeValue &gt;= 0; fadeValue -=5) {</b><br>//Se establece el rango de 0 a 255<br><b>analogWrite(ledPin, fadeValue);</b><br>// esperar 30 milisegundos para ver el efecto de atenuación<br><b>delay(30);</b><br>} |

## Resultados y conclusiones

En esta práctica sus aprendizajes fueron de gran utilidad en el manejo del microprocesador Arduino, ya que el trabajo con las estructuras de control en la programación nos da herramientas para continuar con el manejo del *IDE* de Arduino y aprendimos, describimos e identificamos las características generales de un ciclo “*for*”.

El segundo aprendizaje nos ayudó a diferenciar e identificar las terminales digitales y cómo interactúan con la función *analogWrite* del microcontrolador Arduino, ya que gracias a esta práctica podemos observar que esta función es sólo aplicable a puertos digitales que tengan *PWM*, identificados en la tarjeta Arduino con el símbolo (~).

## Referencias

For statements (s.f.). Arduino LLC. Recuperado el 22 de octubre de 2013, de <http://arduino.cc/en/Reference/For>

Fading (s.f.). Arduino LLC. Recuperado el 22 de octubre de 2013, de <http://arduino.cc/en/pmwiki.php?n=Tutorial/Fading>



## Actividad 12. Molino activado con transistor

Faustino Jerónimo Albino / Norberto Alejandro Pérez Colín

### Antecedentes

#### ¿Qué es el transistor?

El transistor es un dispositivo electrónico semiconductor que se utiliza para entregar una señal de salida en respuesta a una señal de entrada. El transistor se utiliza por su función como amplificador, oscilador, conmutador o rectificador. La palabra “transistor” viene de la contracción en inglés de *transfer* y de *resistor* (resistencia de transferencia). Hoy en día se encuentran en todos los aparatos electrónicos. El transistor fue inventado en 1947 y fue el sustituto de la válvula termoiónica, creando una nueva era dentro de la electrónica. La figura 1 nos muestra algunos de sus encapsulados de fabricación para ser interconectados.

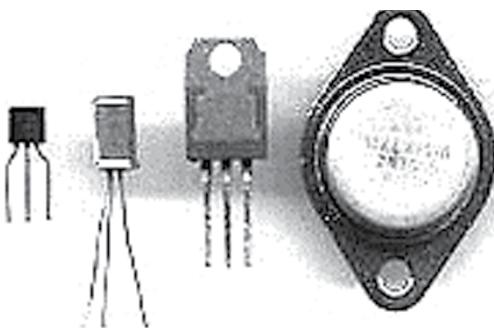


Figura 1. Diferentes encapsulados del transistor

Las razones por las que el transistor reemplazó a la válvula termoiónica son varias:

- Las válvulas necesitan voltajes muy altos.
- Las válvulas consumen mucha energía, poco útiles para el uso con baterías.
- El peso que iba desde algunos kilogramos a decenas de kilogramos.
- El tiempo medio entre fallas de las válvulas termoiónicas es muy corto.
- Las válvulas tardan en funcionar, ya que necesitan estar calientes.
- Los transistores son más pequeños que las válvulas.
- Los transistores trabajan con impedancias bajas, es decir, con tensiones reducidas y corrientes altas.
- El costo de los transistores está muy por debajo.

Los transistores son elementos que han facilitado nuestra vida actual, ya que existen en todos los equipos electrónicos; su diseño en circuitos electrónicos se ha reducido en tamaño, su gran versatilidad y facilidad de control, han creado una gran revolución en la electrónica desde su aparición.

Los transistores están hechos por la unión de tres cristales semiconductores, dos del tipo P uno del tipo N (PNP), o bien dos del tipo N y uno del P (NPN).



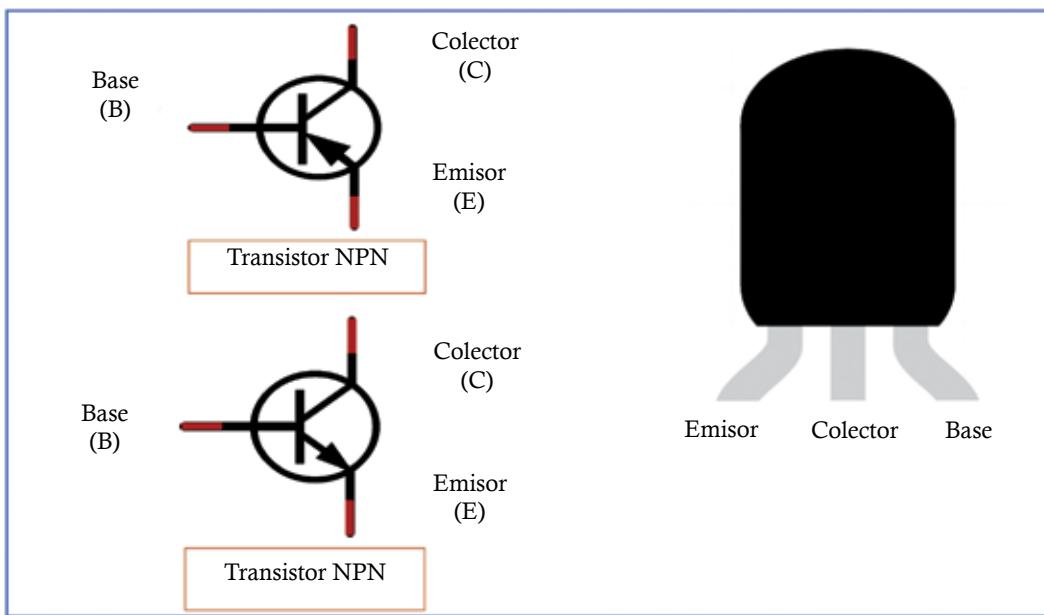


Figura 2. Símbolo electrónico y esquema de un transistor. Imagen obtenida del software Fritzing

El transistor tiene básicamente dos formas de trabajo:

1. Deja pasar o corta señales eléctricas a partir de una señal de control.
2. Funciona como un elemento amplificador de señales.

El funcionamiento del transistor puede tener tres estados en su trabajo dentro de un circuito:

1. Estado en activa: deja pasar más o menos corriente.
2. Estado en corte: no deja pasar la corriente.
3. Estado en saturación: deja pasar toda la corriente.

Para comprender los tres estados del transistor, haremos una analogía mediante una llave hidráulica, que es fácil de entender.

Primero vamos a imaginar que el transistor es una llave de agua como las que tenemos en lavamanos, fregaderos y otros lugares donde tomamos agua, hablemos de agua para entender cómo es el flujo de la corriente eléctrica.

Veamos la figura 3, nos muestra la llave de agua en sus tres estados diferentes. Para que la llave suba y pueda pasar agua desde el tubo E hacia el tubo en C, es necesario que entre un poco de agua por el tubo B y empuje la llave hacia arriba.



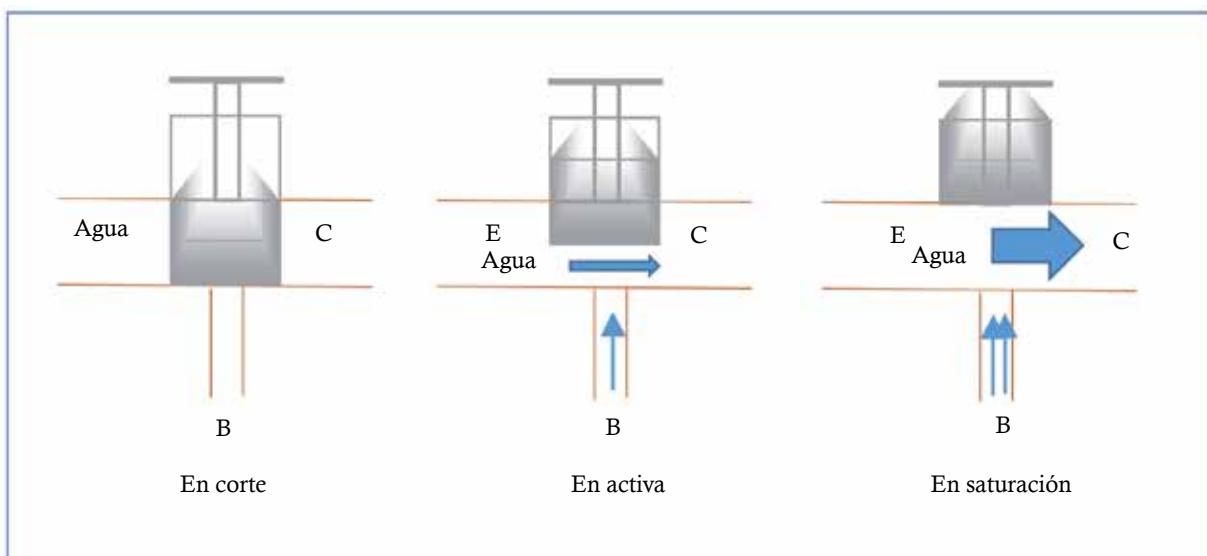


Figura 3. Analogía hidráulica (llave de agua) con un transistor

- *Funcionamiento en corte:* si no hay presión de agua en B (no pasa agua por su tubería), la válvula está cerrada, no se produce un flujo de agua desde E (emisor) hacia C (colector).
- *Funcionamiento en activa:* si llega algo de presión de agua a la base B, se abrirá la válvula en función de la presión que llegue, iniciando un flujo de agua desde E (emisor) hacia C (colector).
- *Funcionamiento en saturación:* si llega la presión suficiente por B se abrirá totalmente la válvula y toda el agua podrá pasar desde el emisor E hasta el colector C.

En el transistor, cuando no le llega nada de corriente a la base, no hay paso de corriente entre el emisor y el colector (en corte), funciona como un interruptor abierto entre el emisor y el colector y cuando llega una corriente máxima en la base (en saturación) su funcionamiento es como un interruptor cerrado, entre el emisor y el colector hay flujo de corriente y además, pasa la máxima corriente permitida por el transistor entre E y C, como se muestra en la figura 4.

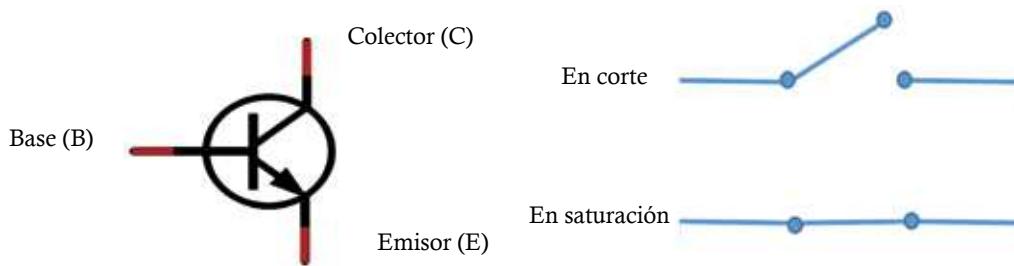


Figura 4. Transistor en corte y en saturación

El funcionamiento del transistor se puede imaginar como un *interruptor* que se acciona eléctricamente, por medio de la corriente en la base. Pero también se puede considerar un *amplificador*



de corriente porque con una pequeña corriente en la base conseguimos una corriente mayor entre emisor y colector. Esto es tercer caso (en activa) cuando a la base del transistor le llega una corriente más pequeña para que se abra el transistor, entonces entre Emisor y Colector pasará una corriente intermedia que no llegará a la máxima.

La diferencias entre el transistor PNP y el NPN es que en el PNP la corriente de salida entra por el emisor y sale por el colector. La flecha del símbolo eléctrico apunta a la base. En el NPN la corriente entra por el colector y sale por el emisor, al revés. Y la flecha apunta hacia afuera del símbolo eléctrico. En la figura 5 se puede observar el flujo de corrientes de los dos tipos de transistores NPN y PNP.

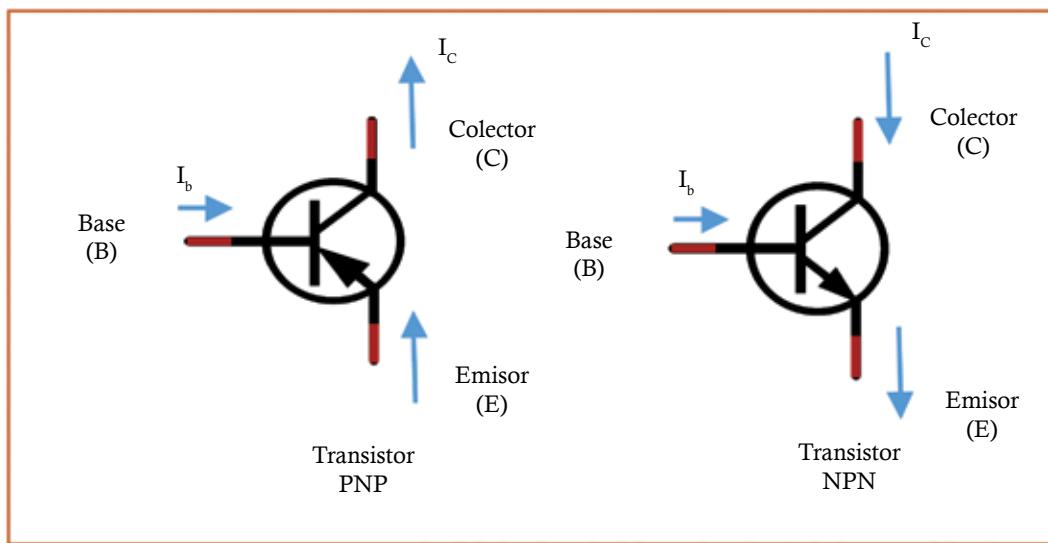


Figura 5. Esquema del flujo de corrientes eléctricas en el transistor

## Introducción

En esta práctica el centro del aprendizaje es el transistor; vamos a trabajar con él en estado de corte y en saturación, esto es, su funcionamiento será equivalente a un interruptor (*switch*), por lo que pretendemos mover un motor con una fuente externa, donde el control lo tendremos desde el *pin 9* del microcontrolador Arduino, que, a su vez, activará la base del transistor, permitiendo que entre en un estado de saturación dejando pasar la corriente entre el colector y el emisor; lo cual permitirá el movimiento rotatorio del motor. El transistor que utilizaremos es el BC547B con las siguientes especificaciones técnicas:





|        |         |
|--------|---------|
| Type   | Marking |
| BC547B | BC547B  |
| BC547C | BC547C  |

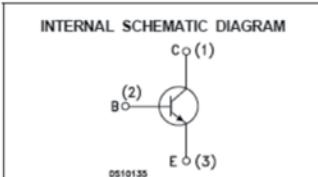
- SILICON EPITAXIAL PLANAR NPN TRANSISTORS
- TO-92 PACKAGE SUITABLE FOR THROUGH-HOLE PCB ASSEMBLY
- BC547B - THE PNP COMPLEMENTARY TYPE IS BC557B
- 
- APPLICATIONS
- WELL SUITABLE FOR TV AND HOME APPLIANCE EQUIPMENT
- SMALL LOAD SWITCH TRANSISTORS WITH HIGH GAIN AND LOW SATURATION VOLTAGE

**BC547B  
BC547C**

SMALL SIGNAL NPN TRANSISTORS



TO-92



INTERNAL SCHEMATIC DIAGRAM

0510135

ABSOLUTE MAXIMUM RATINGS

| Symbol           | Parameter                                     | Value      | Unit |
|------------------|---|------------|------|
| $V_{CB}$         | Collector-Base Voltage ( $I_E = 0$ )          | 50         | V    |
| $V_{CE}$         | Collector-Emitter Voltage ( $I_E = 0$ )       | 45         | V    |
| $V_{BE}$         | Emitter-Base Voltage ( $I_C = 0$ )            | 6          | V    |
| $I_C$            | Collector Current                             | 100        | mA   |
| $I_{CM}$         | Collector Peak Current                        | 200        | mA   |
| $P_{\text{tot}}$ | Total Dissipation at $T_c = 25^\circ\text{C}$ | 500        | mW   |
| $T_{stg}$        | Storage Temperature                           | -65 to 150 | °C   |
| $T_j$            | Max. Operating Junction Temperature           | 150        | °C   |

ELECTRICAL CHARACTERISTICS ( $T_{\text{case}} = 25^\circ\text{C}$  unless otherwise specified)

| Symbol        | Parameter   | Test Conditions   | Min.       | Typ.        | Max.        | Unit     |
|---------------|---|---|------------|-------------|-------------|----------|
| $I_{CBO}$     | Collector Cut-off Current ( $I_E = 0$ )           | $V_{CB} = 30 \text{ V}$<br>$V_{CB} = 30 \text{ V}$ $T_c = 150^\circ\text{C}$  |            |             | 15<br>5     | nA<br>μA |
| $I_{EBO}$     | Emitter Cut-off Current ( $I_C = 0$ )             | $V_{EB} = 5 \text{ V}$  |            |             | 100         | nA       |
| $V_{(BR)CEO}$ | Collector-Emitter Breakdown Voltage ( $I_S = 0$ ) | $I_C = 10 \text{ mA}$   | 45         |             |             | V        |
| $V_{CE(sat)}$ | Collector-Emitter Saturation Voltage              | $I_C = 10 \text{ mA}$ $I_B = 0.5 \text{ mA}$<br>$I_C = 100 \text{ mA}$ $I_B = 5 \text{ mA}$                                 |            | 0.09<br>0.2 | 0.25<br>0.6 | V<br>V   |
| $V_{BE(sat)}$ | Base-Emitter Saturation Voltage                   | $I_C = 10 \text{ mA}$ $I_B = 0.5 \text{ mA}$<br>$I_C = 100 \text{ mA}$ $I_B = 5 \text{ mA}$                                 |            | 0.7<br>0.9  |             | V<br>V   |
| $V_{BE(on)}$  | Base-Emitter On Voltage                           | $I_C = 2 \text{ mA}$ $V_{CE} = 5 \text{ V}$<br>$I_C = 10 \text{ mA}$ $V_{CE} = 5 \text{ V}$                                 | 0.58       | 0.66        | 0.7<br>0.77 | V<br>V   |
| $h_{FE}$      | DC Current Gain                                   | $I_C = 2 \text{ mA}$ $V_{CE} = 5 \text{ V}$<br>for BC547B<br>for BC547C   | 200<br>420 |             | 450<br>800  |          |
| $f_T$         | Transition Frequency                              | $I_C = 10 \text{ mA}$ $V_{CE} = 5 \text{ V}$ $f = 100 \text{ MHz}$  | 100        |             |             | MHz      |
| $C_{CBO}$     | Collector-Base Capacitance                        | $I_E = 0$ $V_{CB} = 10 \text{ V}$ $f = 1 \text{ MHz}$   |            | 1.5         |             | pF       |
| $C_{EBO}$     | Emitter-Base Capacitance                          | $I_C = 0$ $V_{EB} = 0.5 \text{ V}$ $f = 1 \text{ MHz}$  |            | 11          |             | pF       |
| NF            | Noise Figure                                      | $V_{CE} = 5 \text{ V}$ $I_C = 200 \mu\text{A}$ $f = 1 \text{ kHz}$<br>$\Delta f = 200 \text{ Hz}$ $R_g = 2 \text{ k}\Omega$ |            | 2           | 10          | dB       |

\* Pulsed: Pulse duration = 300 μs, duty cycle ≤ 2 %

Figura 6. Especificaciones técnicas del transistor BC547B



El circuito electrónico que vamos a emplear para crear nuestro prototipo se puede observar en el siguiente esquema (figura7):

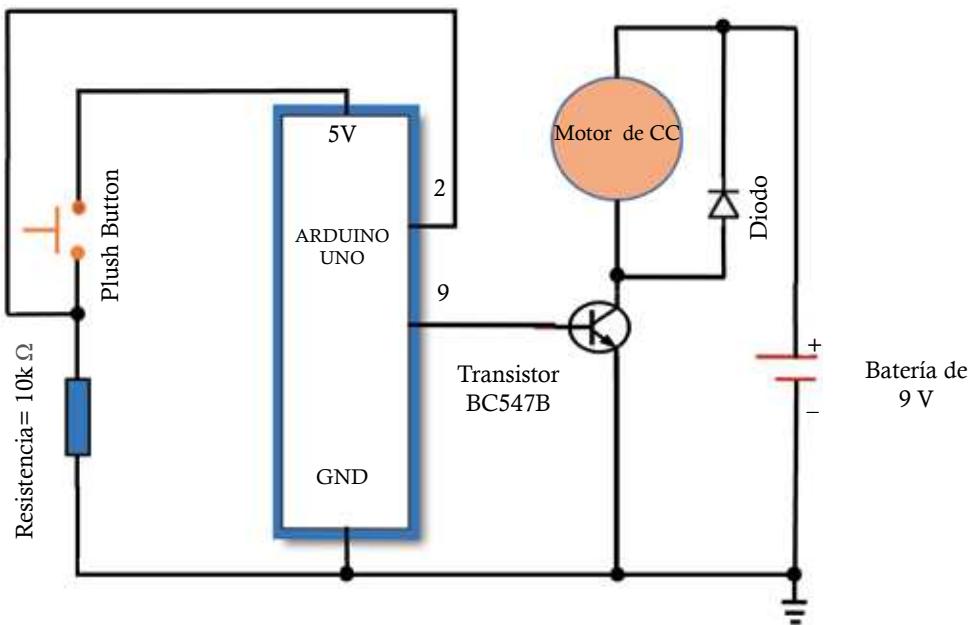


Figura 7. Circuito electrónico del prototipo “Molino activado con transistor”

En el circuito de la figura 7 se puede observar un diodo conectado en paralelo con el motor de corriente continua, a esta forma de acoplamiento se le llama “diodo de fly-back”, su función tiene que ver con las corrientes de retorno.

Todos los motores crean un campo magnético gracias al flujo de corriente eléctrica, al estar energizados, pero cuando lo apagamos existen algunos giros por la inercia del paro; esto genera un campo magnético durante un tiempo muy breve, que a su vez genera una corriente eléctrica en sentido contrario al de la corriente utilizada para mover el motor. Esta corriente se llama corriente de retorno, puede ser muy intensa y puede dañar la electrónica, por lo que se utiliza un diodo en paralelo y en inversa, para no dejar pasar estas corrientes.

## Objetivo

Al finalizar la práctica, el alumno:

1. Diferenciará la forma en la cual se deben de conectar los componentes transistor, diodo y motor.
2. Aumentará su destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.
3. Entenderá cómo funciona el transistor acoplado al microcontrolador Arduino.



## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 transistor BC547B.
- 1 resistencia de  $10K\Omega$ .
- 1 diodo 1N4007.
- 1 motor de corriente continua.
- 1 batería de 9V.
- 1 conector para la batería.
- 1 pulsador.

## Desarrollo

Conectar la alimentación del Arduino a la *protoboard*.

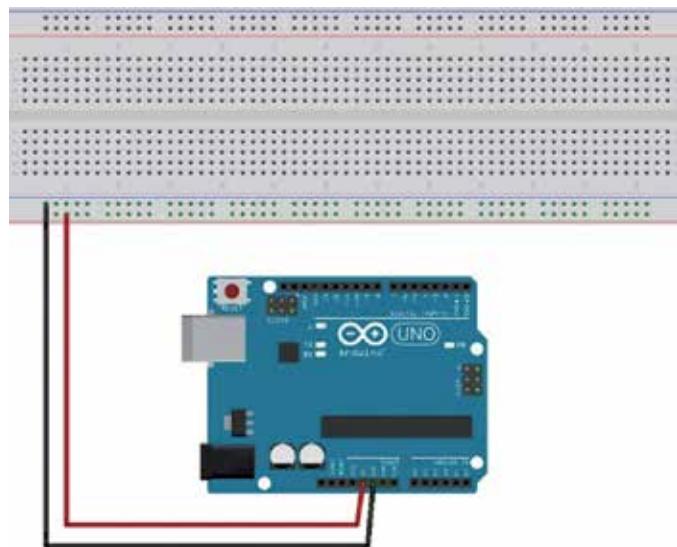


Figura 8. Circuito del proyecto. Imagen obtenida del software *Fritzing*

Añadimos un pulsador, conectando la patilla de entrada a la alimentación y la de salida al pin digital 2 del Arduino. Añadiremos también la resistencia de  $10K\Omega$  entre la patilla de salida del pulsador y tierra.



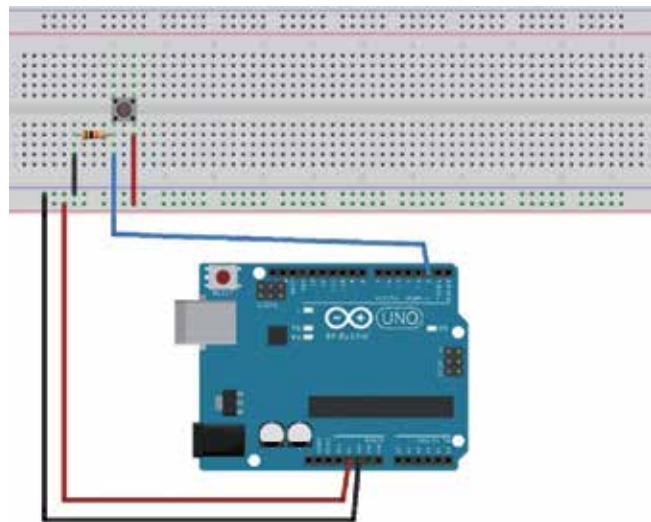


Figura 9. Circuito del proyecto. Imagen obtenida del software Fritzing

Cuando utilizemos circuitos con diferentes voltajes, debemos conectar sus tierras de forma conjunta, para conseguir una tierra común. Enseguida conectamos la batería de 9V a la *protoboard*. Unimos mediante un cable, la tierra de la batería con la del Arduino en la *protoboard*. Finalmente, conectamos la otra patilla del motor a la alimentación de 9V.

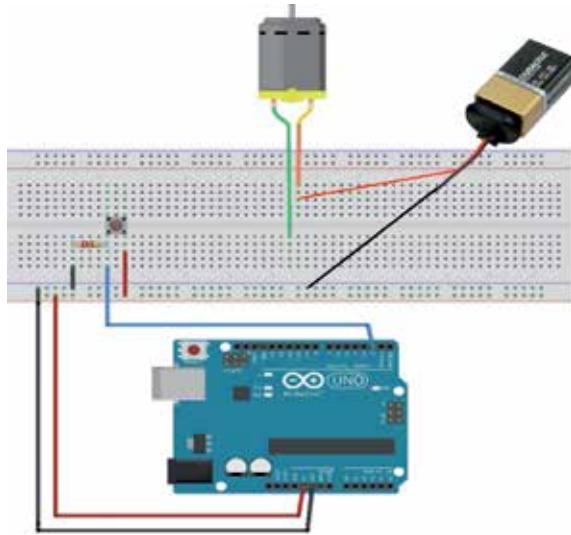


Figura 10. Circuito del proyecto. Imagen obtenida del software Fritzing

A continuación, colocaremos el transistor en la *protoboard*. El siguiente paso será conectar el *pin* digital 9 al *pin* izquierdo del transistor. Cuando Arduino activa el transistor suministrándole tensión a la base, el colector se conectarán con la tercera patilla, llamada emisor. Finalmente, conectamos el emisor a tierra. Recordemos que la tensión de retorno fluye en dirección opuesta a la de la tensión que le suministramos al motor, por esto conectaremos un diodo. Terminado de montar el circuito, nos quedaría de la siguiente manera.



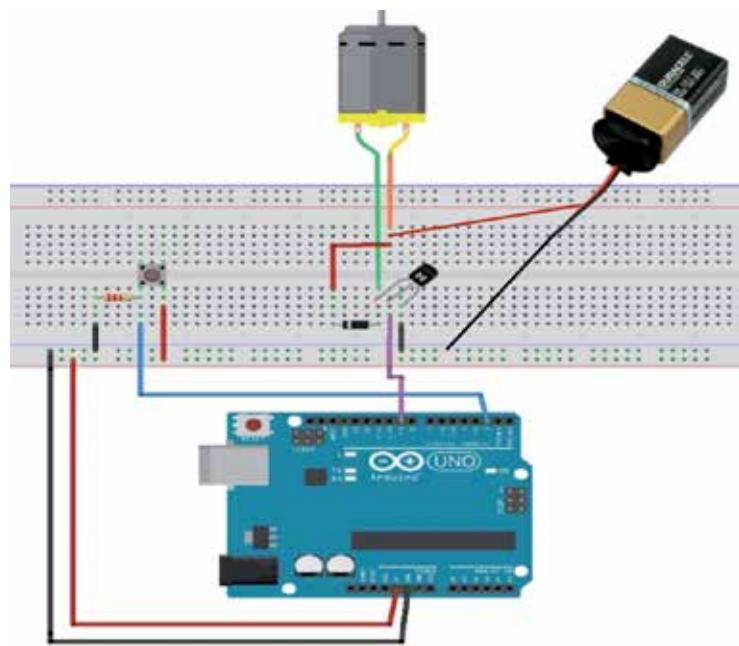
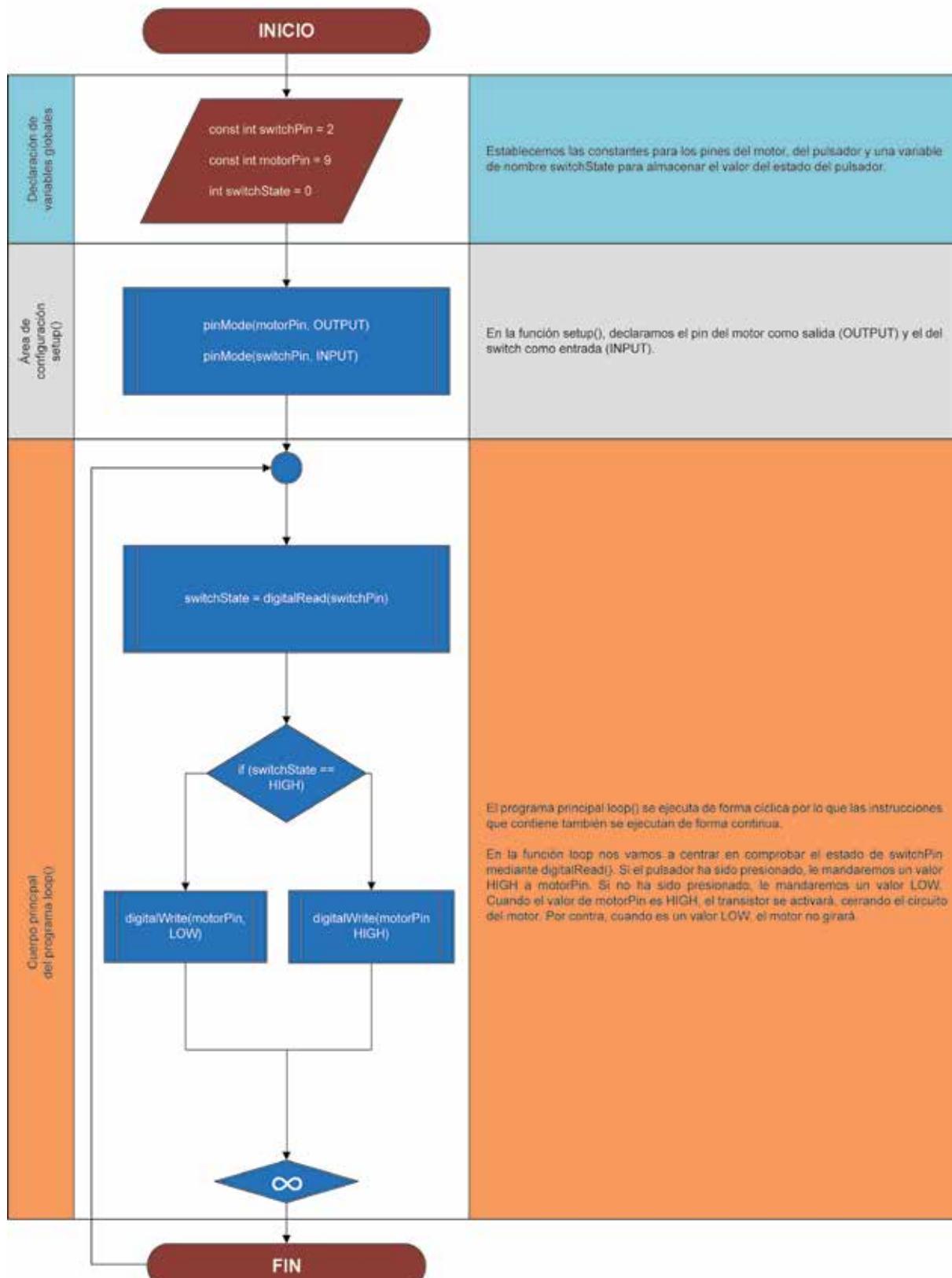


Figura 11. Circuito del proyecto. Imagen obtenida del software *Fritzing*



# Diagrama de Flujo

## " Molino activado con transistor "



## Código

|                          |   |  |
|--------------------------|---|--|
| Declaración de Variables | <pre>//Molino activado con transistor<br/>const int switchPin = 2;<br/>const int motorPin = 9;<br/>int switchState = 0;</pre>   | Lo primero de todo es establecer unas constantes para los pines del motor y del pulsador y una variable de nombre <i>switchState</i> para almacenar el valor del pulsador. |
| Setup ()                 | <pre>void setup() {<br/>    pinMode(motorPin, OUTPUT);<br/>    pinMode(switchPin, INPUT);<br/>} //Fin de la función setup.</pre>  | En la función <i>setup()</i> , declararemos el pin del motor como salida ( <i>OUTPUT</i> ) y el del switch como entrada ( <i>INPUT</i> ).                                  |
| Loop ()                  | <pre>//la función loop() se ejecuta cíclicamente y de manera ininterrumpida<br/>void loop() {<br/>    switchState = digitalRead(switchPin);<br/><br/>    if (switchState == HIGH) {<br/>        digitalWrite(motorPin, HIGH);<br/>    }<br/>    else {<br/>        digitalWrite(motorPin, LOW);<br/>    }<br/>} //Fin de la función loop.</pre> | En la función <i>loop ()</i> , nos centraremos en comprobar el estado de <i>switchPin</i> mediante <i>digitalRead()</i> .  |



## Resultados y conclusiones

En esta práctica se trabajó con el microprocesador Arduino y se desarrollaron habilidades en su implementación para la creación de un prototipo, en donde pudimos diferenciar la forma, en la cual se deben de conectar los componentes transistor, diodo y motor, para que se puedan usar como dispositivos y desarrollar algoritmos de control, para operar un circuito electrónico con las características generales de un motor y cómo interconectarlo con el microcontrolador Arduino

Un aprendizaje muy valioso en el uso del microprocesador Arduino es entender cómo funciona el transistor combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.

## Referencias

El transistor (s.f.). Recursos y Conocimientos Sobre Tecnología, Tecnología Industrial y Electrotécnica. Recuperado el 24 de noviembre de 2014, de <http://www.areatecnologia.com/TUTORIALES/EL%20TRANSISTOR.htm>

Torrente Artero, O. (2013). *Arduino : curso práctico de formación* (pp. 274-276). Madrid: RC Libros

El transistor (s.f.). Recursos y Conocimientos Sobre Tecnología, Tecnología Industrial y Electrotécnica. Recuperado el 24 de noviembre de 2014, de <http://www.areatecnologia.com/TUTORIALES/EL%20TRANSISTOR.htm>

BC547B (s.f.). DatasheetCatalog.com. Recuperado el 24 de noviembre de 2014, de <http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/8854.pdf>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 94-101). Italia: Arduino LLC.



## Actividad 13. Semáforo basado en un potenciómetro

Víctor Arturo Morales Díaz

### Antecedentes

*¿Qué es un potenciómetro?*

Es una resistencia cuyo valor es variable. Con el potenciómetro podemos controlar la intensidad de corriente que fluye por un circuito.

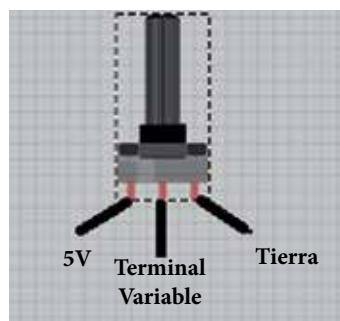


Figura 1. Partes de un potenciómetro, imagen generada en el software Fritzing

Como podemos ver en la figura 1, una de las patillas del potenciómetro va a nuestra fuente (5 V), la otra va a tierra (0 V) y la otra es la terminal variable, es decir, que es la que va a estar controlando la intensidad de la corriente en nuestro circuito, en nuestro caso es con la que vamos a indicar que LED tiene que prender, de acuerdo con la intensidad.

### Introducción

En esta práctica controlaremos la secuencia de las señales de un semáforo; lo haremos con un potenciómetro, con esto vamos a poder hacer un semáforo mucho más realista.

Al hacer variar la resistencia del potenciómetro vamos a cambiar la secuencia de las luces.

¿Cómo funciona un semáforo? Primero que nada, tenemos que un semáforo cuenta con tres luces:

- Rojo: Alto, no puedes pasar.
- Amarillo: Precaución, tienes que bajar la velocidad de tu automóvil para detenerte.
- Verde: Siga, puedes avanzar.

Ahora el funcionamiento del semáforo es el siguiente:

1. Se prende la luz roja.
2. Se prende la luz verde, antes de que pase a la siguiente luz, ésta parpadea unos segundos.
3. Se prende la luz amarilla.
4. Se vuelve a prender la luz roja.



## Objetivos

- Controlará la secuencia de las señales a través de un potenciómetro.
- Simulará el funcionamiento de un semáforo.
- Describirá las características generales del *hardware* y *software* del microcontrolador Arduino.
- Describirá e identificará las características generales del potenciómetro.
- Diferenciará e identificará las terminales analógicas del microcontrolador Arduino.
- Hará uso de las terminales analógicas del microcontrolador Arduino.
- Hará uso de la programación para controlar las terminales analógicas del Arduino.

## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- 3 ledes (rojo, amarillo, verde).
- Resistencia 270 Ω.
- Resistencia 100 Ω.
- Potenciómetro 10 K Ω.

## Desarrollo

Como se muestra en la figura 2 y con ayuda de una *protoboard*, conectemos el ánodo del *LED* rojo al *pin* 7 del Arduino, el ánodo del *LED* amarillo al *pin* 4 del Arduino y del ánodo del *LED* verde al *pin* 2 del Arduino. El cátodo de los tres ledes a tierra (*GND*) a través de las resistencias de 270 Ω.

El potenciómetro lo conectaremos de la siguiente manera, la patita de en medio irá conectada al *pin* A0 del Arduino, que es una entrada analógica, la patita izquierda del potenciómetro, viéndolo de frente, va a tierra y la derecha a corriente.

El *pin* de 5 volts y de tierra (*GND*) del Arduino se conecta en la *protoboard*.

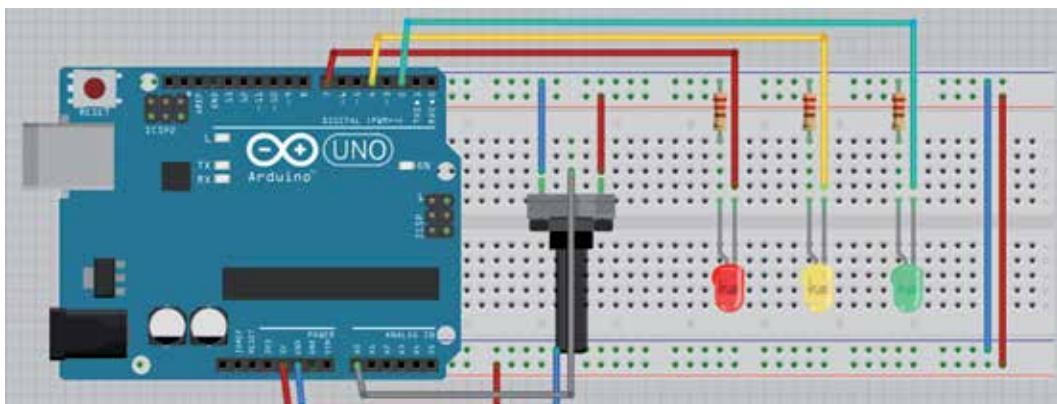


Figura 2. Prototipo de un semáforo, imagen generada en *Fritzing*



El *pin* A0 se configurará, a través del código de programación (ver código en la sección correspondiente), como un *pin* analógico que se alimentará con una señal analógica variable, con ayuda de un potenciómetro, de 0 a 5 volts, de acuerdo con el valor del voltaje prenderemos los ledes.

El rango que utilizaremos para prender los ledes será el siguiente:

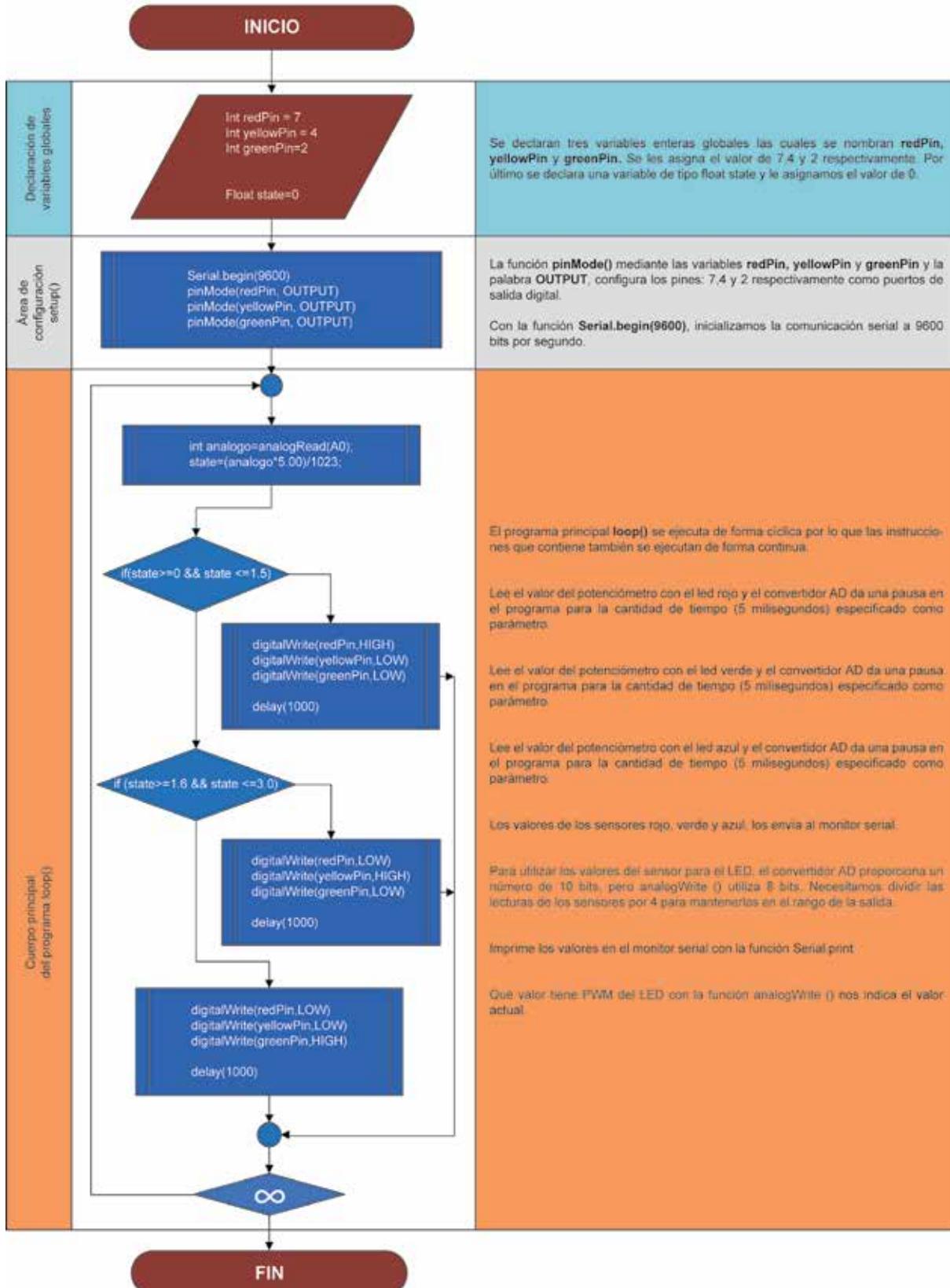
- De 0 V a 1.5 V: se enciende el *LED* rojo.
- De 1.6 V a 3.0 V: se enciende el *LED* amarillo.
- De 3.1 V a 5.0 V: se enciende el *LED* verde.

1. Conecta tu tarjeta Arduino a la computadora.
2. Ejecuta el *software* de programación de tu Arduino; carga el programa: P11. El código se explica en la siguiente sección.
3. Observarás que el *LED* rojo se prende, si tu potenciómetro tiene en 0 V.
4. Varía la resistencia del potenciómetro y verás cómo los ledes se prenden y se apagan conforme vas variando la resistencia.



## Diagrama de flujo

### “ Semáforo basado en un potenciómetro ”



## Código

|                                      |   |
|--------------------------------------|---|
| Declaración de variables globales    | // Semáforo basado en un potenciómetro<br><br><pre>int redPin = 7; // LED rojo conectado al pin 7 int yellowPin = 4; // LED amarillo conectado al pin 4 int greenPin=2; // LED verde conectado al pin 2 float state=0; // Se declara una variable de estado en forma decimal</pre>  |
| Configuración setup()                | <pre>void setup() {   Serial.begin(9600); // Se inicializa el monitor serial a 9600 bps   pinMode(redPin, OUTPUT); // Se declaran los pines digitales como salidas   pinMode(yellowPin, OUTPUT); // Se declaran los pines digitales como salidas   pinMode(greenPin, OUTPUT); // Se declaran los pines digitales como salidas }</pre>   |
| Cuerpo principal del programa loop() | <pre>//la función loop() se ejecuta cíclicamente y de manera ininterrumpida void loop() {   int analogo=analogRead(A0); // Se declara una variable local para leer la patilla de control                                 //del potenciómetro en la entrada analógica A0    state=(analogo*5.00)/1023; // la valor de la variable analogo la convertimos de voltajes                             // de entrada entre 0 y 5 volts en valores enteros entre 0 y 1023   if(state&gt;=0 &amp;&amp; state &lt;=1.5) //Si el valor de state es menor que 1.5 prende solo el rojo   {     digitalWrite(redPin,HIGH);     digitalWrite(yellowPin,LOW);     digitalWrite(greenPin,LOW);     delay(1000);   }   else if (state&gt;=1.6 &amp;&amp; state &lt;=3.0) //Si el valor de state es mayor que 1.6 y menor a 3  //prende solo el amarillo   {     digitalWrite(redPin,LOW);     digitalWrite(yellowPin,HIGH);     digitalWrite(greenPin,LOW);     delay(1000);   }   else                               //Si el valor de state es mayor a 3 prende solo el verde   {     digitalWrite(redPin,LOW);     digitalWrite(yellowPin,LOW);     digitalWrite(greenPin,HIGH);     delay(1000);   } }</pre> |



## Resultados y conclusiones

Al realizar esta práctica se desarrolló la simulación de un semáforo, para poder apagar o prender las luces se tuvo que controlar la resistencia del circuito.

Se utilizó un potenciómetro para conmutación para controlar la secuencia de las señales de un semáforo.

Pudimos conocer otras características generales del *hardware* y *software* del microcontrolador Arduino, como lo son las terminales analógicas, ya que a través de ellos pudimos controlar la secuencia de las señales del semáforo. Obviamente todo esto se logró por medio de la programación de dichas terminales.

## Referencias

Monk, S. (2012). *Modelo del semáforo. 30 proyectos con Arduino* (pp. 41-44). Madrid: Editorial Esterior, S.L.



## Actividad 14. Mood Cue

Flor Clara Cubillas Hernández

### Antecedentes

#### ¿Qué es un servo motor?

Un servo es un dispositivo pequeño, similar a un motor eléctrico, cuya particularidad es que no da vueltas sin parar, puede ser controlado tanto en velocidad como en una posición determinada y permanece en esa posición hasta que se indique la instrucción de volverse a mover. Como puede realizar giros de 180 grados, son comúnmente usados en modelismo como aviones, barcos, helicópteros, trenes para controlar de manera eficaz los sistemas de motores y los de direcciones; en robótica se pueden tener pequeñas aplicaciones didácticas hasta el más complejo diseño robótico. Veamos figura 1.

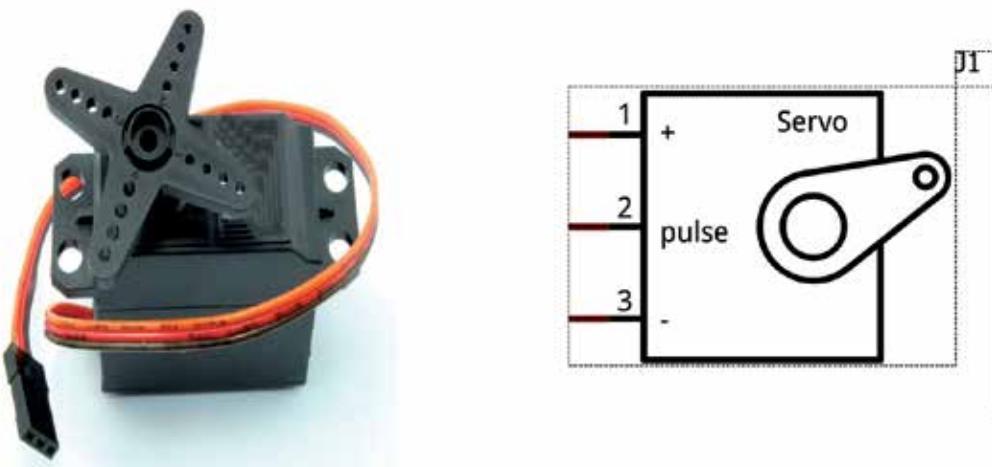


Figura 1. Imagen de servo motor (izquierda) y símbolo del servo motor (derecha). Imagen (izquierda) editada con el *software de photoshop* y símbolo (derecha) realizado con el *software de Fritzing*

#### ¿Qué es la función map?

Es una técnica donde un servo motor espera recibir una orden de pulsos que le indiquen en qué ángulo ha de posicionarse. Estos pulsos siempre tienen el mismo intervalo de tiempo, pero su amplitud varía entre 1 y 2 milisegundos, el *software* de Arduino tiene una librería (conjunto de *software* que expande las funciones del entorno de desarrollo), que va a permitir controlar fácilmente dicho servo motor. Los servo motores sólo giran 180 grados y la entrada analógica está comprendida entre 0 y 1023, en este caso se utiliza una función llamada *map()* para cambiar la escala de los valores que toma del potenciómetro. En nuestro programa, se tendrá que importar dicha librería para disponer de toda su funcionalidad.



## Introducción

En esta práctica veremos cómo es el funcionamiento del servo motor, cómo se desplaza de una posición a otra nueva, cómo son sus entradas analógicas de nuestro microcontrolador, las librerías de Arduino que se utilizan para facilitar el control del motor. Para esto se utiliza una función llamada *map*, con la que se pueden cambiar los valores de nuestro potenciómetro.

Para la conexión de nuestro servo-motor, nos apoyaremos en la figura 2.

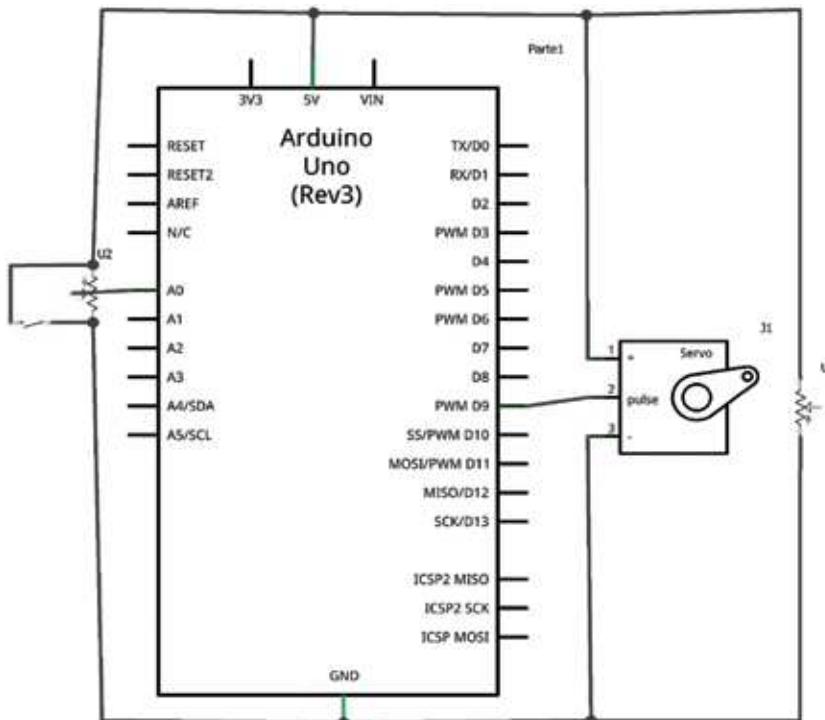


Figura 2. Esquema de conexión del servo motor. Realizada con el software de *fritzing*

## Objetivo

- Aprenderá a usar la función *map* con sus diferentes parámetros.
- Entenderá cómo se utilizan las librerías del servo motor.
- Continuará identificando las entradas digitales y analógicas, así como los dispositivos que se integran a la placa de Arduino.
- Entenderá cómo es el funcionamiento un servo motor.
- Controlará los cambios del servo motor usando un potenciómetro.
- Entenderá cómo el *hardware* y *software* del microcontrolador Arduino interactúan a través de las terminales digitales.
- Aprenderá a utilizar el puerto de comunicaciones de Arduino para recibir y transmitir valores.



## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A - conexión B).
- Microcontrolador Arduino UNO.
- 1 servo motor.
- 1 potenciómetro.
- 1 condensadores de 100 uF.

## Desarrollo

Como se ilustra en la figura 3, lo primero que haremos es montar en la *protoboard* nuestro circuito, poner el cable de tierra y la alimentación de 5V; se procede a conectar el potenciómetro, figura 4, el cual tiene 3 patillas que están juntas, se conecta una a la alimentación y la otra a tierra. De hecho, un potenciómetro no es más que un divisor de tensión. Al hacerlo girar, cambiamos el valor de la tensión que cae entre el *pin* del medio (el que se encuentra solo) y el *pin* que se encuentra conectado a la alimentación. Podemos leer este valor en una entrada analógica de nuestro Arduino, es por ello que conectaremos el *pin* del medio al *pin* analógico A0. Con esto podremos controlar la posición de nuestro servo motor.

Nuestro servo motor tiene tres cables: uno es la alimentación (color rojo), otro es la conexión a tierra (color negro) y el tercer cable es el que va a recibir la información que proviene del Arduino. Como el conector del servo motor no nos sirve para conectarlo a la *protoboard*, tendremos que utilizar un adaptador de tres pins que nos permita hacerlo. Una vez que se tiene el adaptador puesto, conectaremos cada *pin* a una hilera diferente de la *protoboard*. El cable rojo irá conectado a la alimentación, el negro a tierra y el cable blanco al *pin* 9 del Arduino.

Se tendrán que conectar los condensadores, cuando un servo motor se empieza a mover, necesita que tenga más corriente que cuando ya está en marcha, lo que provoca una caída de tensión en la placa. Para poder evitar este problema de la variación de la tensión, se colocara un condensador de 110 uF conectado al positivo y al negativo del servo motor; del mismo modo, conectaremos un condensador a la alimentación y a la tierra del potenciómetro.

A los condensadores conectados de esta forma se les denomina condensadores separadores, porque reducen o separan los cambios producidos por ciertos componentes del resto del circuito, se debe conectar el cátodo a tierra y el ánodo a la alimentación. De lo contrario, al estar conectados inversamente, podrían explotar. Tener en cuenta que un condensador tiene polaridad y viene indicada en la carcasa del mismo. Normalmente el cátodo se muestra por una franja de color negro, o si el condensador ya es de por sí de color negro, por una franja blanca, figura 5.



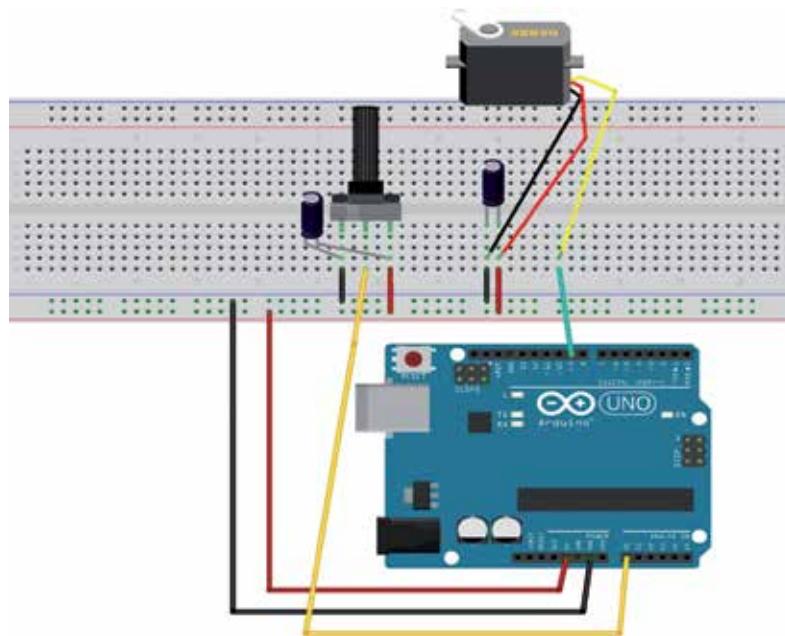


Figura 3. Prototipo de un servo motor. Realizada con el *software de Fritzing*

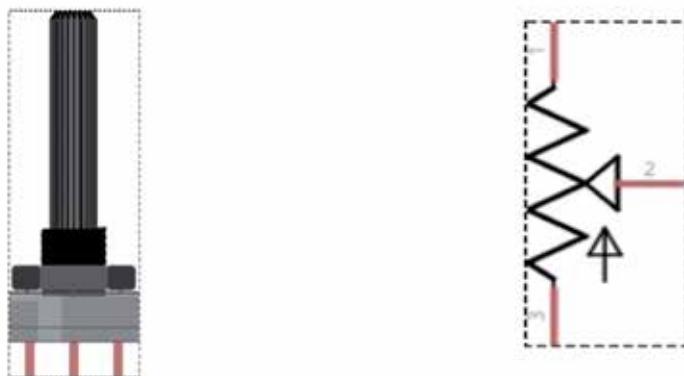


Figura 4. Imagen de potenciómetro (izquierda) y símbolo del potenciómetro (derecha).

Imagen realizada con el *software de Fritzing*

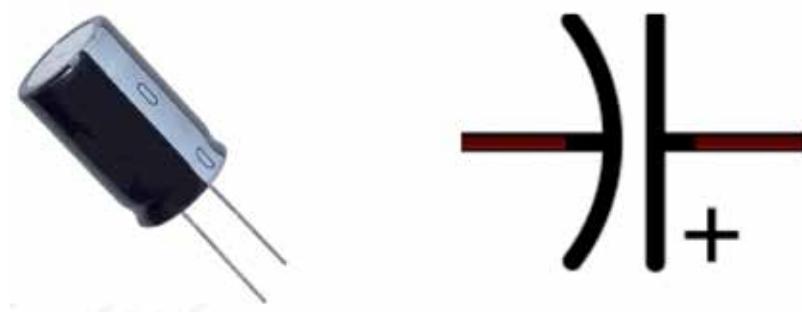


Figura 5. Imagen de condensador (izquierda) y símbolo del condensador (derecha).

Imagen (derecha) realizado con el *software de Fritzing*



1. Conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.
2. Ejecuta el *software* de programación de tu Arduino, carga el código del programa para compilar y verificar que funcione correctamente. El código se explica en la siguiente sección.

Una vez que esté alimentado y programado nuestro Arduino, abriremos el monitor serie. Podremos ver una serie de valores parecidos a estos:

```
potVal: 1023, angle: 179  
potVal: 1023, angle: 179
```

Cuando se gire el potenciómetro, se verá cómo estos números cambian y el servo se desplaza a su nueva posición.

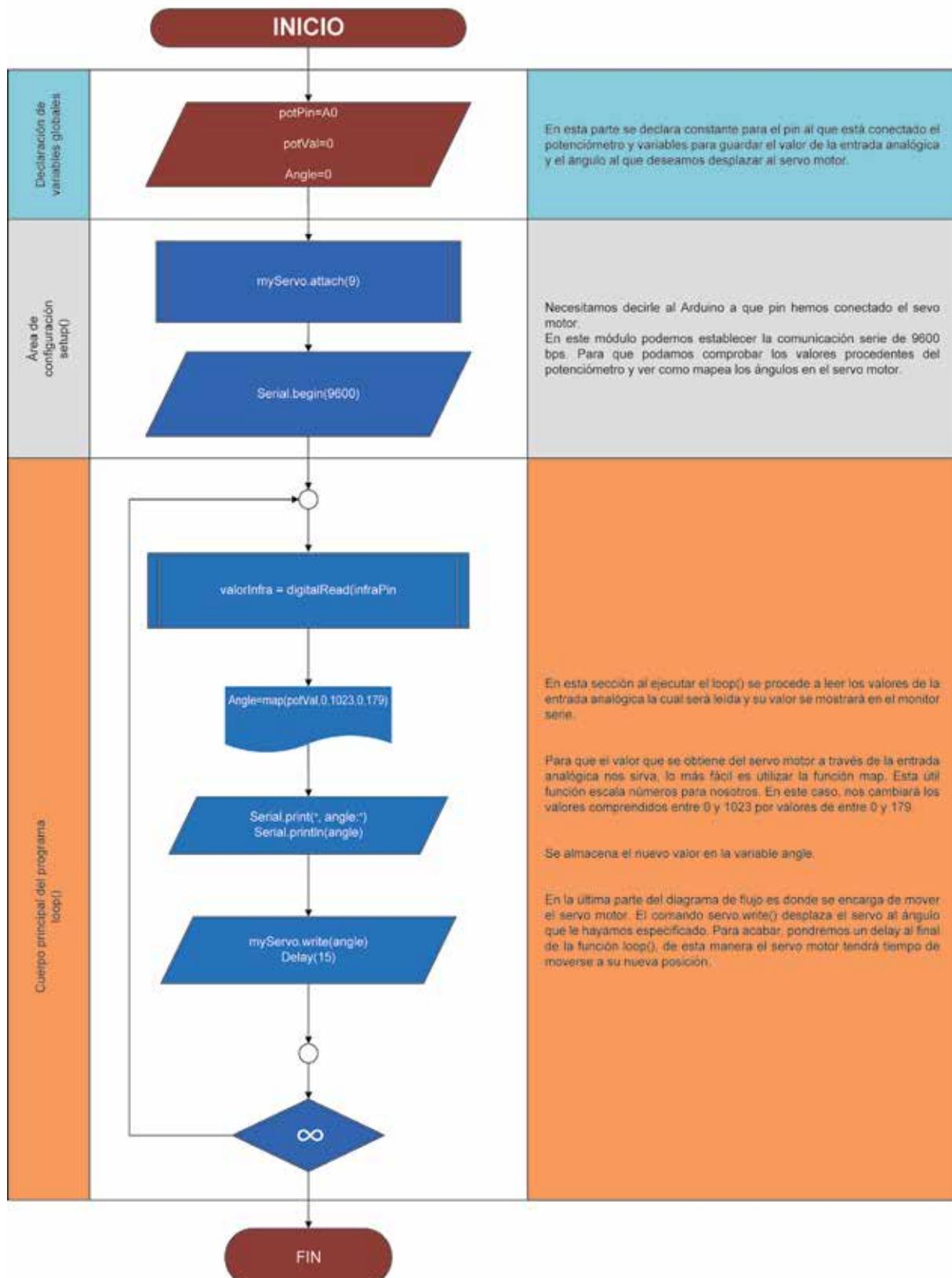
Prestemos atención a la relación entre el valor de *potVal*, el ángulo en el monitor serie y la posición del servo motor. Deberíamos ver valores congruentes como resultado de hacer girar el potenciómetro.

Un hecho positivo derivado de usar potenciómetros como entradas analógicas, es que nos proporcionan un amplio abanico de valores comprendidos entre 0 y 1023. Esto puede ser de gran ayuda a la hora de hacer pruebas en proyectos que utilicen la entrada analógica.



## Diagrama de flujo

“Mood Cue”



## Código

|                                   |  |
|-----------------------------------|--|
| Declaración de variables globales | <pre>// include the servo library #include &lt;Servo.h&gt;  Servo myServo; // create a servo object  int const potPin = A0; // analog pin used to connect the potentiometer int potVal; // variable to read the value from the analog pin int angle; // variable to hold the angle for the servo motor</pre>   |
| setup()                           | <pre>void setup() {     myServo.attach(9); // attaches the servo on pin 9 to the servo object     Serial.begin(9600); // open a serial connection to your computer }</pre>   |
| loop()                            | <pre>void loop() {     potVal = analogRead(potPin); // read the value of the potentiometer     // print out the value to the serial monitor     Serial.print("potVal: ");     Serial.print(potVal);      // scale the numbers from the pot     angle = map(potVal, 0, 1023, 0, 179);      // print out the angle for the servo motor     Serial.print(", angle: ");     Serial.println(angle);      // set the servo position     myServo.write(angle);      // wait for the servo to get there     delay(15); }</pre> |



## Resultados y conclusiones

Se demostró en esta práctica el funcionamiento del servo motor el usar el potenciómetro para el desplazamiento a diferentes posiciones. Se aprendió a utilizar la función *map*, a implementar las librerías del servo para poder comprobar los valores del potenciómetro y ver cómo mapea los ángulos en el servo motor, además se pudieron ver los resultados en el monitor serie.

Al realizar esta práctica se desarrolló el manejo del microcontrolador Arduino, el funcionamiento del *pin* digital específicamente del *pin 9* y de la entrada analógica A0 de la tarjeta Arduino UNO. Pudimos observar el funcionamiento del servo motor, las diferentes formas de desplazamiento a su nueva posición. También el hecho de usar potenciómetros como entradas analógicas, es que nos proporcionan un amplio abanico de valores comprendidos entre 0 y 1023. Esto nos puede ser de gran ayuda a la hora de hacer pruebas en proyectos que utilicen la entrada analógica. Aprendimos a importar las librerías del servo motor, a utilizar el puerto serie configurando la velocidad en 9600 bps, a través de este tipo de comunicación podemos enviar y recibir datos desde nuestro Arduino a la computadora.

## Bibliografía

Map(value, fromLow, fromHigh, toLow, toHigh) (s.f.). *Arduino LLC*. Recuperado el 30 de agosto de 2014, de <http://arduino.cc/en/pmwiki.php?n=Reference/Map>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 62-69). Italia: Arduino LLC.



## Actividad 15. MEDIDOR VU (vúmetro)

Edith López Martínez

### Antecedentes

Para la ejecución de esta práctica se debe tener claro el concepto de vúmetro. El vúmetro nos permite medir la intensidad de una señal acústica de manera visual en unidades de volumen, ya que la entrada del sonido es captada y medida dependiendo del nivel que ésta tenga.

Hoy en día existen vúmetros construidos de diferentes formas: analógicos, en forma de barras o de manera digital mediante el uso de pantallas *LCD*.

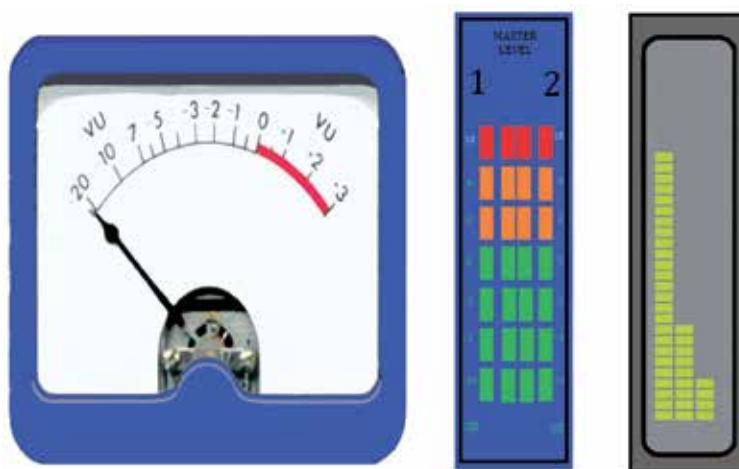


Figura 1. Vúmetros analógicos, de *LED* y *LCD*

En esta práctica se muestra el funcionamiento de un vúmetro mediante Arduino UNO, el cual indica el nivel de señal de audio al tomarlo como una señal de entrada y representándolo en una barra de *LED*. Se presenta el esquema electrónico para saber cómo se conectará en el *protoboard* y la comunicación que habrá con el microcontrolador.

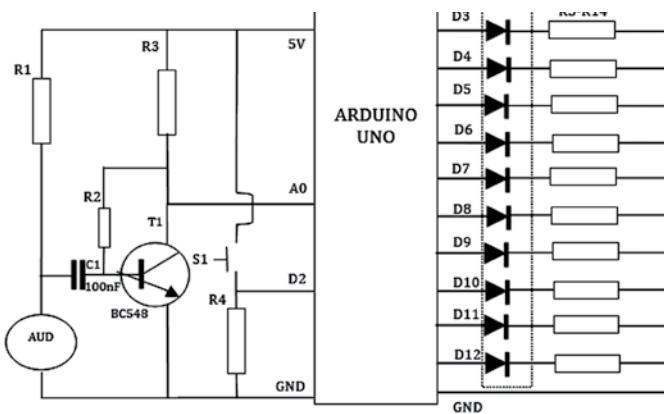


Figura 2. Esquema electrónico de un vúmetro.



## Objetivos

- Describirá las características generales del *hardware* y *software* del microcontrolador Arduino UNO utilizado en esta práctica.
- Describirá e identificará las características generales del vómetro, su función y algunas aplicaciones.
- Hará uso de la programación para controlar las terminales del Arduino.
- Medirá la señal de audio de entrada mediante la visualización de los ledes o diodos emisores de luz.

## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- Bocina.
- Barra de *LED* de 10 segmentos o 10 ledes (diodos emisores de luz).
- 14 resistencias de  $270\text{ M}\Omega$ .
- 1 entrada de Audio.

## Desarrollo

En primer lugar, se colocan los ledes con sus respectivas resistencias.

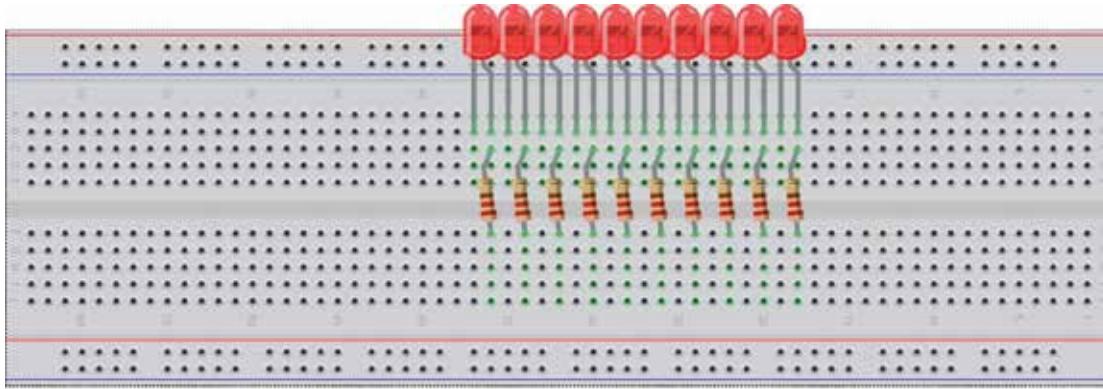


Figura 3. Prototipo de un vómetro. Realizada con el *software* de *Fritzing*

Posteriormente, armaremos el cableado para establecer la comunicación con el microcontrolador, así como la entrada del audio.



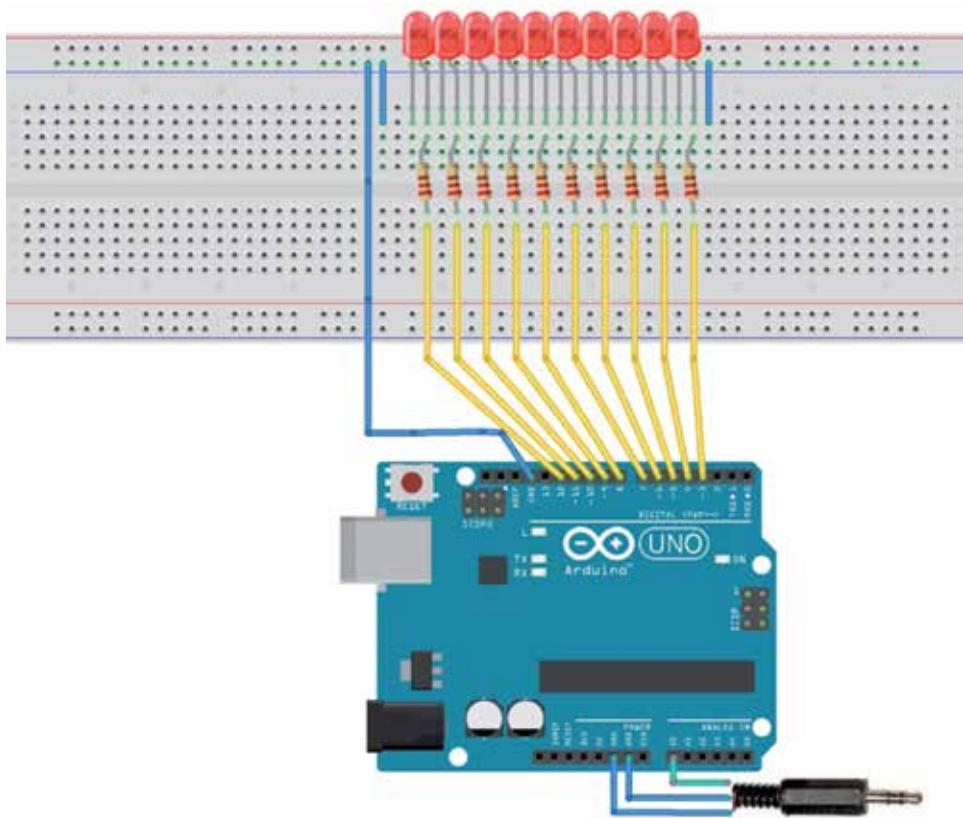
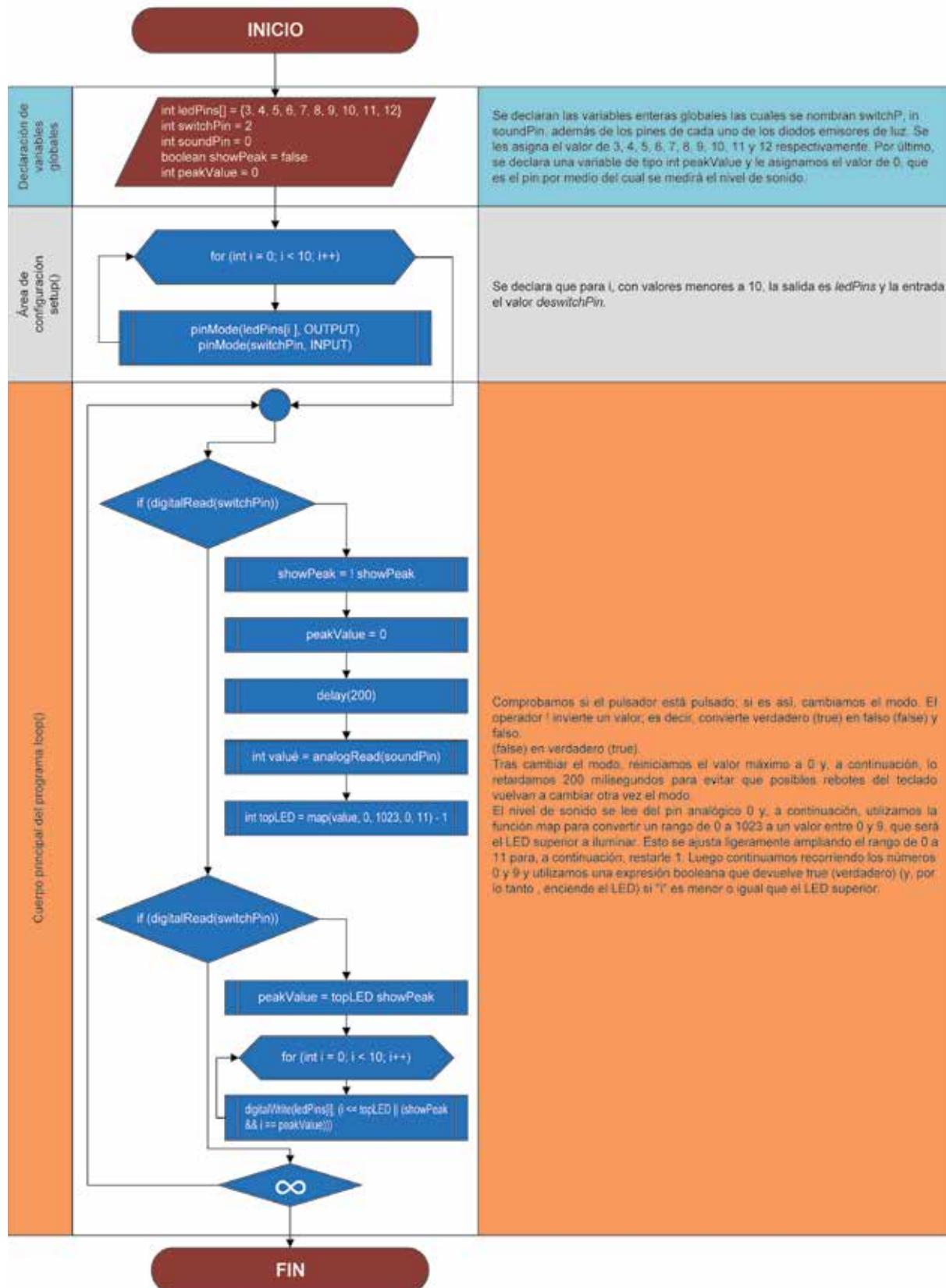


Figura 4. Prototipo de un vúmetro. Realizada con el software de *Fritzing*



## Diagrama de flujo

“Medidor VU (vúmetro) ”



## Código

|                                      |  |
|--------------------------------------|--|
| Declaración de variables globales    | <pre>int ledPins[] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 12};<br/>int switchPin = 2;<br/>int soundPin = 0;<br/>boolean showPeak = false;<br/>int peakValue = 0;</pre>  |
| Configuración setup()                | <pre>void setup()<br/>{<br/>    for (int i = 0; i &lt; 10; i++)<br/>    {<br/>        pinMode(ledPins[i], OUTPUT);<br/>    }<br/>    pinMode(switchPin, INPUT);<br/>}</pre>  |
| Cuerpo principal del programa loop() | <pre>void loop()<br/>{<br/>    if (digitalRead(switchPin))<br/>    {<br/>        showPeak = ! showPeak;<br/>        peakValue = 0;<br/>        delay(200);<br/>        int value = analogRead(soundPin);<br/>        int topLED = map(value, 0, 1023, 0, 11) - 1;<br/>        if (topLED &gt; peakValue)<br/>        {<br/>            peakValue = topLED;<br/>        }<br/>        for (int i = 0; i &lt; 10; i++)<br/>        {<br/>            digitalWrite(ledPins[i], (i &lt;= topLED    (showPeak &amp;&amp; i == peakValue)));<br/>        }<br/>    }<br/>}</pre> |



## Resultados y conclusiones

Como resultado de la elaboración de esta práctica con ayuda del microcontrolador Arduino UNO, obtenemos un medidor visual de una señal sonora en la escala de los ledes, el cual es muy utilizado en la vida cotidiana, como en los estéreos de los autos, en amplificadores de sonido, etcétera. Los resultados fueron favorables ya que las luces de los diodos emisores de luz prenden y apagan debido a que la señal de entrada de sonido no es uniforme, por lo tanto, cumplen la función de representación del nivel en los ledes.

Al término de la práctica, se conocieron las características generales del *hardware* y *software* del microcontrolador Arduino UNO utilizado, además de identificar y conocer las características y usos generales de un vúmetro, no sin antes hacer uso de la programación para el control de las terminales en dicho microcontrolador. Finalmente, se cumplieron los objetivos, ya que se midió la señal de audio de entrada mediante la visualización de los ledes o diodos emisores de luz indicando el nivel de ésta.

## Referencias

Monk, S. (2012). *Medidor VU. 30 proyectos con Arduino* (pp. 120-124). Madrid: Editorial Esteribor, S.L.



## Actividad 16. Reloj de arena digital

Flor Clara Cubillas Hernández

### Antecedentes

*¿Qué es la función millis() de Arduino?*

La función *millis* permite contabilizar el tiempo que transcurre en algún evento. Cuando se quería utilizar al cabo de un intervalo específico de tiempo se recurría a la función *delay*, pero era un poco limitante, así que el uso de este tipo de retraso no es muy útil para realizar un seguimiento del tiempo transcurrido. La función *millis* nos ayuda a contabilizar en milisegundos el tiempo que nuestro Arduino está funcionando; la variable que se utiliza para almacenar el tiempo se denomina *unsigned long* (*long* sin signo). Cuando un tipo de variable es llamada *unsigned*, se le considera solamente positiva.

*¿Qué es un sensor de inclinación?*

Sensores de inclinación hay de varias clases, el más usado es el de mercurio. Cuando se habla de inclinación, significa que adentro del mismo sensor tiene mercurio, lo que hace es unir dos puntos de metales y une un paso de energía o cortarlo cuando se le inclina, lo une al inclinarse porque es metal líquido. Se pueden utilizar estos sensores en robótica para hacer que el robot se ponga de pie automáticamente si se cae por cualquier motivo; en aviación se usa para una precisión real del horizonte, en los giros para que de un ángulo exacto y preciso de inclinación del avión.

### Introducción

En esta práctica vamos a construir un reloj que enciende un *LED* cada diez minutos, después de esto cada diez minutos se activará otro *LED*. Después de una hora, los seis ledes estarán encendidos, en cuanto se oprima el interruptor la placa cambiará de estado, los ledes se apagarán y el temporizador empezará a contar en cero.

Para la conexión de nuestro reloj, nos apoyaremos en la figura 1 en donde conectamos los cables de alimentación en el *protoboard*, después se unirá el ánodo de cada uno de los seis ledes a las salidas digitales (*Pin 2*, al *Pin 7*) de Arduino; el cátodo de los ledes se conecta a la tierra mediante las resistencias de  $220\ \Omega$ , se conecta el interruptor en un extremo al voltaje y el otro extremo se conectaría a la tierra a través de la resistencia de  $10\text{ k}\ \Omega$ , se procede a conectar un cable que vaya desde el *pin 8* a la unión entre el interruptor y la resistencia de  $10\text{ k}\ \Omega$ .



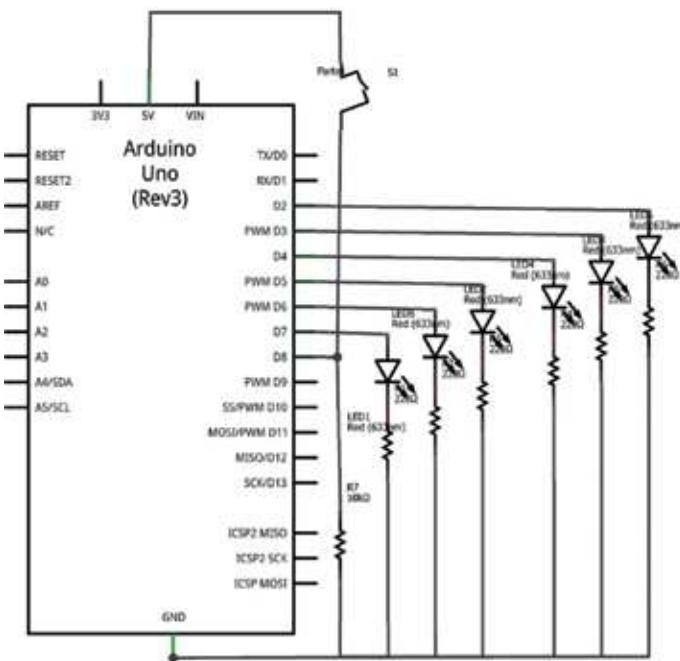


Figura 1. Esquema de conexión de reloj de arena digital. Realizada con el software de *Fritzing*

## Objetivos

- Aplicará los diferentes usos que tiene en las diferentes áreas del saber.
- Continuará identificando las entradas digitales, así como los dispositivos que se integran a la placa de Arduino.
- Entenderá cómo el *hardware* y *software* del microcontrolador Arduino interactúan a través de las terminales digitales.
- Entenderá para nos sirve la función *millis* y la diferencia de utilizar la función *delay*.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A - conexión B).
- 1 microcontrolador Arduino UNO.
- 6 ledes rojos.
- 1 interruptor.
- 1 resistencia 10 kΩ.
- 6 resistencias 220 Ω.



## Desarrollo

Como se ilustra en la figura 2 y con ayuda de una *protoboard*, se conectan los cables de alimentación, continuamos uniendo el ánodo de cada uno de los seis ledes a las terminales de Arduino del 2 al 7. El cátodo de los ledes lo conectaremos a la tierra con su respectiva resistencia de  $220\Omega$ .

Se conecta una de las patillas del interruptor o sensor a la alimentación y la otra va a la tierra conectada con una resistencia de  $10\text{k}\Omega$ .

Ahora se conecta un cable que vaya del pin 8 de nuestro Arduino a la unión entre el sensor y la resistencia  $10\text{k}\Omega$ .

Conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.

Ejecuta el *software* de programación y carga el programa en tu Arduino.

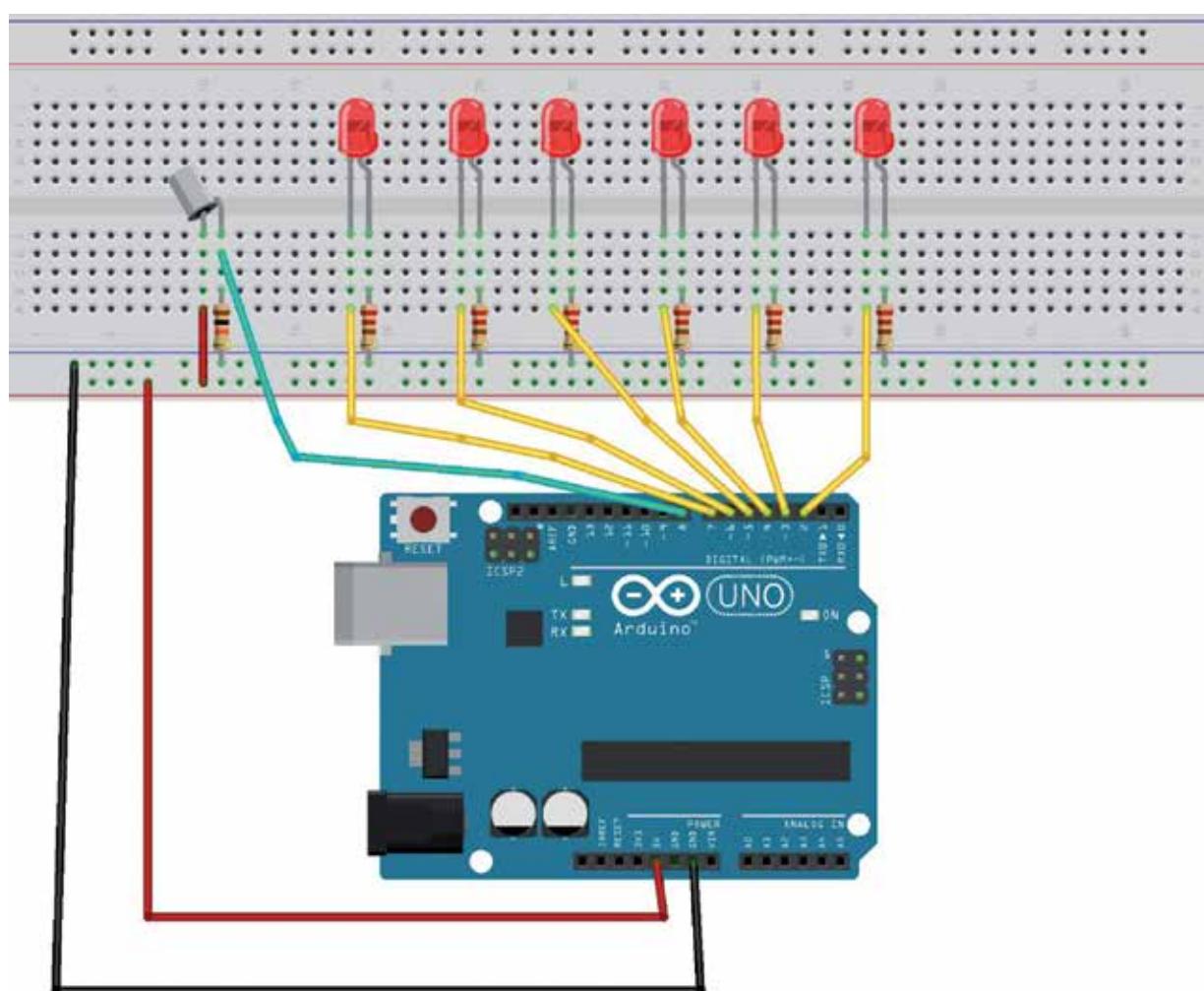
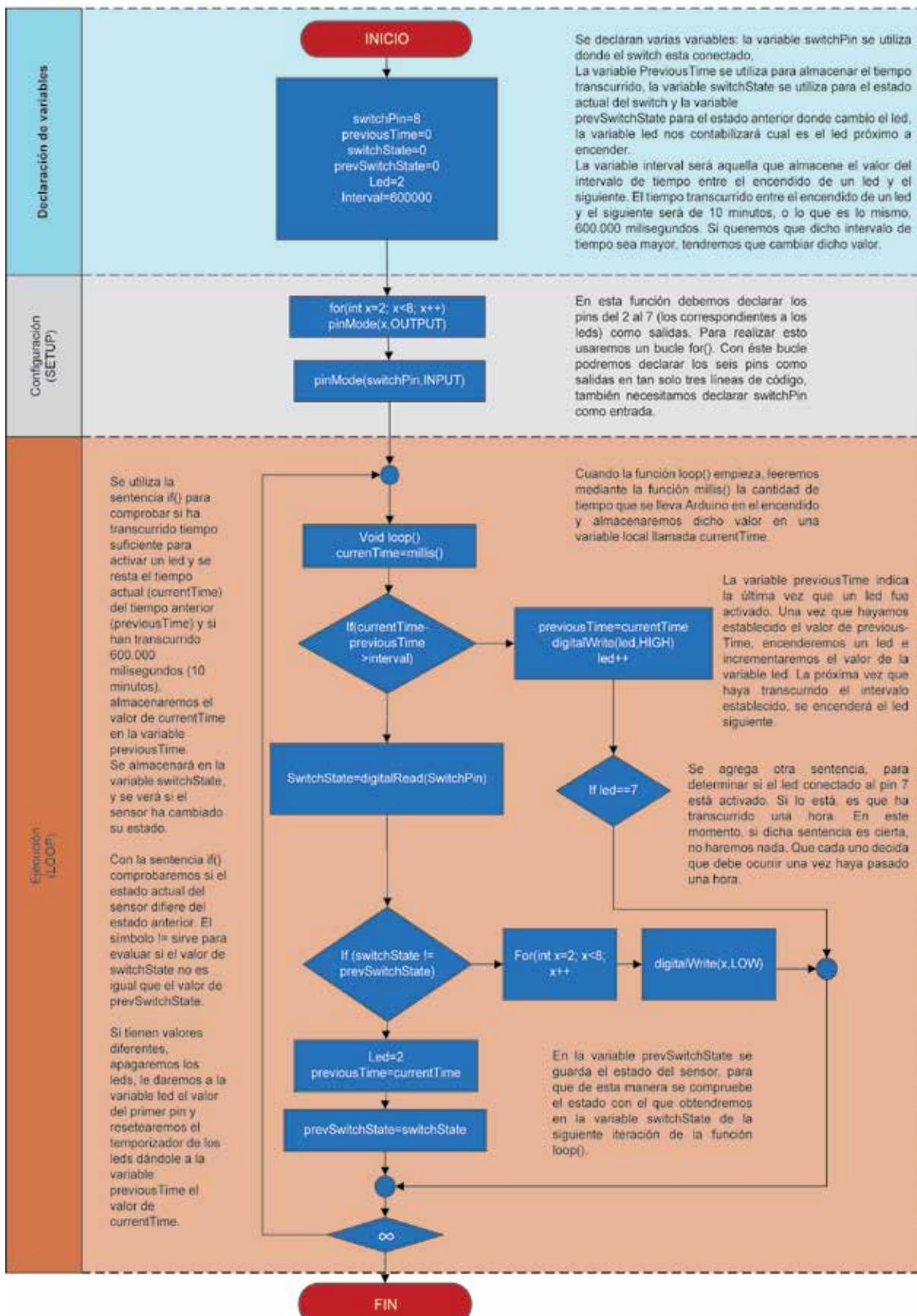


Figura 2. Prototipo de reloj de arena digital. Realizada con el *software* de Fritzing



# Diagrama de flujo

## Reloj de arena digital



## Código

|                                   |   |
|-----------------------------------|---|
| Declaración de variables globales | <pre>// Reloj de arena digital<br/>// Declaramos la constante para el interruptor<br/>const int switchPin = 8;<br/><br/>unsigned long previousTime = 0; // El estado del LED, iniciamos en cero<br/>int switchState = 0; // el estado de conmutación actual<br/>int prevSwitchState = 0; // el estado del interruptor anterior<br/>int led = 2; // Variable para hacer referencia a los LEDs<br/><br/>// 600000 = 10 minutos milisegundos<br/>long interval = 600000; // Intervalo en el que a la luz llega al próximo LED</pre>  |
| Configuración<br>setup()          | <pre>void setup() {<br/>    // Establecer las patillas del LED como salidas<br/>    for(int x = 2;x&lt;8;x++){<br/>        pinMode(x, OUTPUT);<br/>    }<br/>    // Configurar el interruptor al pin como entrada<br/>    pinMode(switchPin, INPUT);<br/>}<br/>*</pre>  |
| loop()                            | <pre>void loop(){<br/>    // store the time since the Arduino started running in a variable<br/>    unsigned long currentTime = millis();<br/><br/>    // compare the current time to the previous time an LED turned on<br/>    // if it is greater than your interval, run the if statement<br/>    if(currentTime - previousTime &gt; interval) {<br/>        // save the current time as the last time you changed an LED<br/>        previousTime = currentTime;<br/><br/>        // Turn the LED on<br/>        digitalWrite(led, HIGH);<br/>        // increment the led variable<br/>        // in 10 minutes the next LED will light up<br/>        led++;</pre> |



```
if(led == 7){  
    // the hour is up  }  
}  
  
// read the switch value  
switchState = digitalRead(switchPin);  
  
// if the switch has changed  
if(switchState != prevSwitchState){  
    // turn all the LEDs low  
    for(int x = 2;x<8;x++){  
        digitalWrite(x, LOW);  
    }  
    // reset the LED variable to the first one  
    led = 2;  
    //reset the timer  
    previousTime = currentTime;  
}  
// set the previous switch state to the current state  
prevSwitchState = switchState;  
}
```

## Resultados y conclusiones

Se demostró en esta práctica el funcionamiento de los sensores de inclinación, el uso de la función *millis()*; utilizamos sentencias *if* y almacenar diferentes valores que se relacionan unos con otros para comprobar la posición del *LED* cuando está prendido.

Al realizar esta práctica hemos aprendido a utilizar la función *millis* para medir el tiempo, el tipo de variables para almacenar el transcurso de encendido, también aprendimos a utilizar un sensor de inclinación y para qué se puede utilizar.

## Referencias

millis() (s.f.). Arduino LLC. Recuperado el 30 de agosto de 2014, de <http://arduino.cc/en/pmwiki.php?n=Reference/Millis>

unsigned long (s.f.). Arduino LLC. Recuperado el 30 de agosto de 2014, de <http://arduino.cc/en/Reference/UnsignedLong>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 86-93). Italia: Arduino LLC.



## Actividad 17. Teclado musical

Flor Clara Cubillas Hernández

### Antecedentes

*¿Qué es un zumbador?*

Es un componente que puede transformar la electricidad en sonido continuo o intermitente de un mismo tono mini parlante (mini-bocina) de bajo costo; cuando se conecta con un circuito integrado puede producir diferentes tonos y las aplicaciones que puede tener son el diseño de alarmas, para aparatos domésticos sirven como mecanismo de señalización o aviso.



Símbolo eléctrico

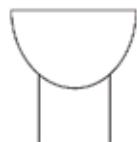


Figura 1. Imagen de un zumbador (izquierda, obtenida del software Fritzing) y símbolo eléctrico (derecha, realizado con el software de Visio)

*¿Qué es un interruptor?*

Es un dispositivo que tiene dos posiciones (encendido y apagado). Es muy común que los usemos en la vida cotidiana como el encender la luz, cuando se conecta dentro del interruptor, dos cables son unidos lo que permite fluir a la corriente para que se encienda, cuando oprimimos nuevamente se corta el fluido de corriente. Estos dispositivos mecánicos tienen cuatro patillas, dos de ellas están unidas cada una entre sí, ver figura 2.

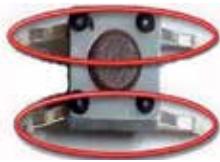


Figura 2. Imagen de un interruptor (izquierda, obtenida del software Fritzing) y símbolo eléctrico (derecha, realizado con el software de Visio)

*¿Qué es una escalera de resistencias?*

La escalera de resistencias nos va a permitir leer cierto número de interrupciones utilizando la entrada analógica de Arduino. En donde debemos tener varios interruptores encadenados en paralelo a la entrada analógica cero, conectados a la alimentación mediante una resistencia y cuando se



interrumpa cada uno de ellos llegará un nivel diferente de tensión a la entrada del *pin*. Si pulsamos dos al mismo tiempo, obtendremos una combinación única de las dos resistencias en paralelo.

*¿Qué es la función tone() en Arduino?*

Los tonos son una serie numérica, en las que se pueden reproducir determinadas señales, que dependerán de su frecuencia y el tiempo de duración. La función *tone()* de Arduino genera una onda cuadrada *PWM* (ciclo de trabajo del 50%) y la frecuencia se especifica con un *pin*.

Su sintaxis es:

*tone (pin, frecuencia)*

*tone (pin, frecuencia, duración)*

Con esta función podemos generar dos parámetros de las notas musicales.

## Introducción

En esta práctica conoceremos cómo es el funcionamiento del teclado musical con interruptores por medio de nuestro microcontrolador Arduino UNO. Para esto, utilizaremos interruptores conectados en paralelo con sus respectivas resistencias cada uno, como se muestra en la figura 3.

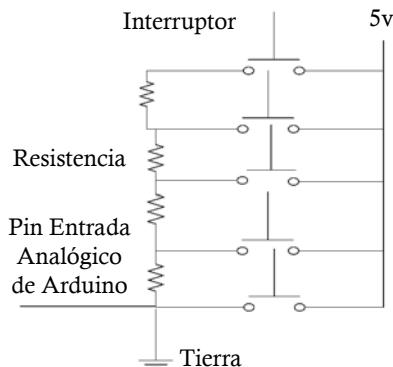


Figura 3. Esquema de escalera de resistencias, imagen realizada con *software* de Visio

## Objetivos

- Aplicará los diferentes usos que tiene en las diferentes áreas del saber.
- Aprenderá a utilizar la función *tone()* de Arduino para generar notas musicales y cómo utilizar los arreglos.
- Continuará identificando las entradas digitales y analógicas, así como los dispositivos que se integran a la placa de Arduino.
- Entenderá cómo el *hardware* y *software* del microcontrolador Arduino interactúan a través de las terminales digitales.



## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A - conexión B).
- Microcontrolador Arduino UNO.
- *Switch*.
- Zumbador.
- Resistencia 10 kilohm.
- Resistencia 1 megohm.
- Resistencia 220 ohm.

## Desarrollo

Como se ilustra en la figura 4 y con ayuda de una *protoboard*, se conecta la bocina un extremo al *pin* de tierra (*GND*) y el otro extremo al *pin* 8 de Arduino.

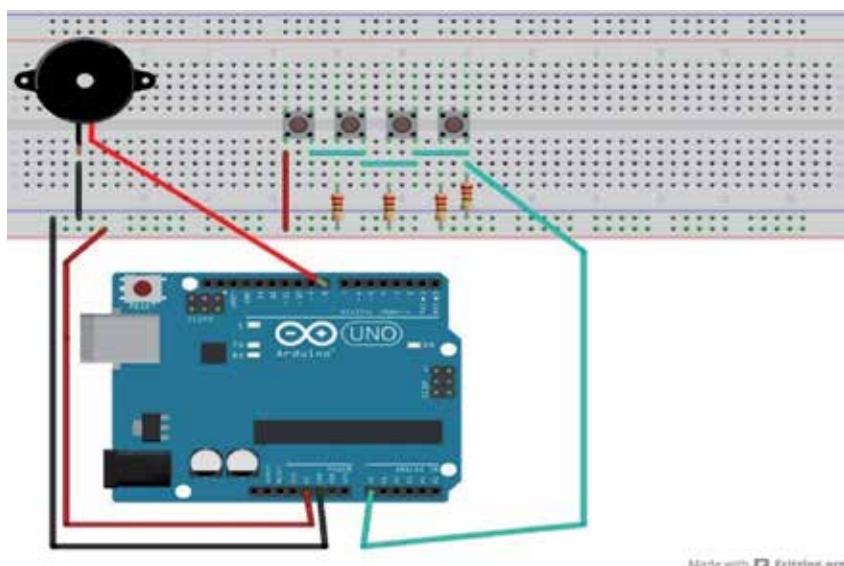


Figura 4. Prototipo de un teclado musical

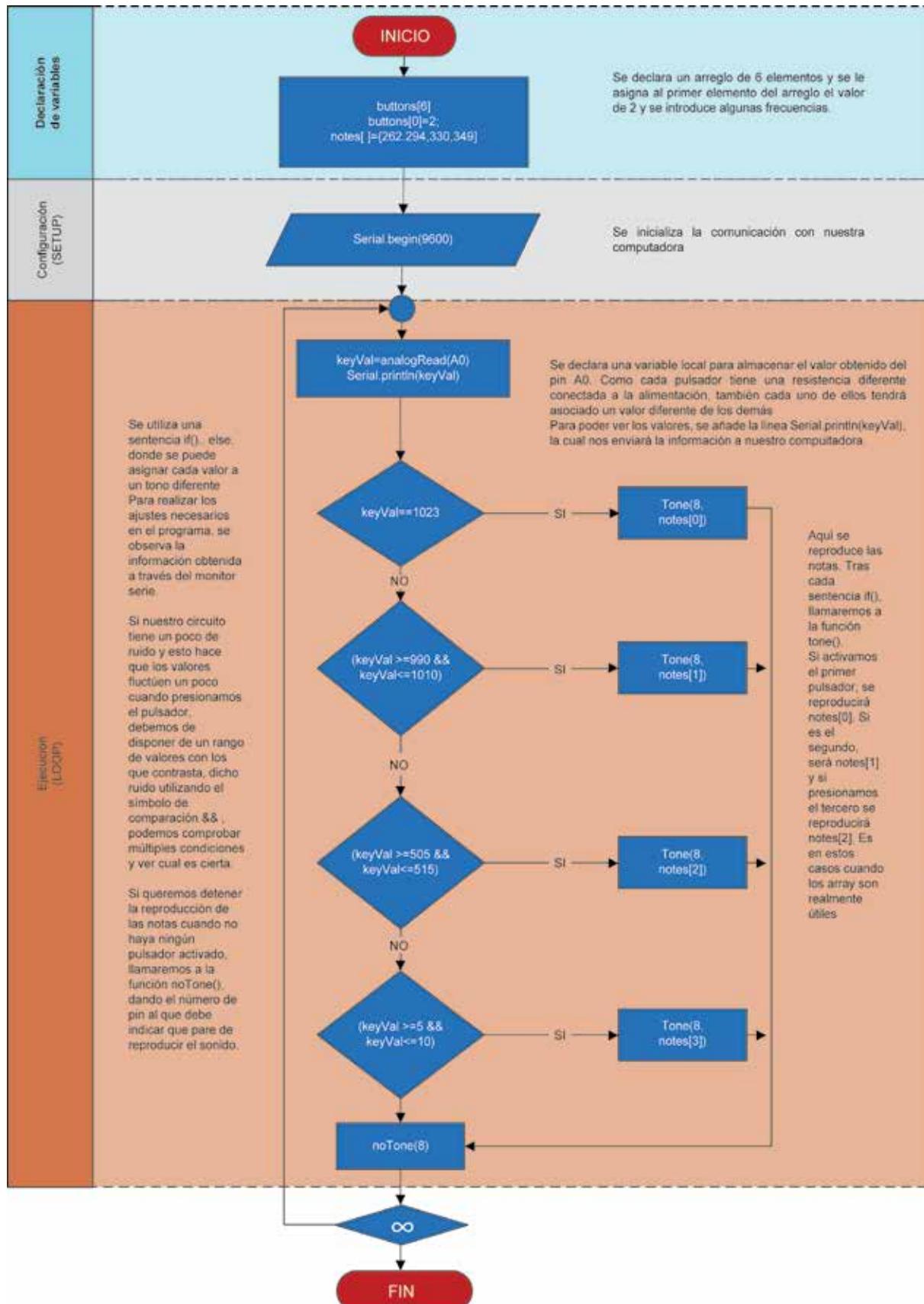
Después se conectan los interruptores en el *protoboard* con sus respectivas resistencias, el primer interruptor se conecta directo a la corriente 5V de Arduino, el segundo, tercero y cuarto interruptor con sus resistencias de  $220\Omega$ ,  $10k\Omega$ ,  $1M\Omega$ , respectivamente, después se conectan todos los interruptores unidos continuamente directo a la entrada analógica A0 y cada uno de éstos actúa como divisor de voltaje.

Se conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.

Ejecuta el *software* de programación de tu Arduino, carga el código del programa para compilar y verificar que funcione correctamente. El código se explica en la siguiente sección.



## Diagrama de flujo



## Código

|                                   |   |
|-----------------------------------|---|
| Declaración de variables globales | <pre>// Teclado musical: creación de una matriz de notas musicales<br/>// los números a continuación corresponden a<br/>// las frecuencias de middle C, D, E, y F<br/><br/>int notes[] = {262, 294, 330, 349};</pre>  |
| Área de configuración setup()     | <pre>void setup() {<br/>    // iniciar la comunicación serial<br/>    Serial.begin(9600);<br/>}</pre>   |
| Programa principal loop()         | <pre>void loop() {<br/>    // Crear una variable local para la entrada en el pin analógico A0<br/>    int keyVal = analogRead(A0);<br/>    // Enviar el valor de A0 para el Monitor Serial<br/>    Serial.println(keyVal);<br/><br/>    // Tocar la nota correspondiente a cada valor de A0<br/>    if(keyVal == 1023){<br/>        // Tocar la primera frecuencia de la matriz en el pin 8<br/>        tone(8, notes[0]);<br/>    }<br/>    else if(keyVal &gt;= 990 &amp;&amp; keyVal &lt;= 1010){<br/>        // Tocar la segunda frecuencia de la matriz en el pin 8<br/>        tone(8, notes[1]);<br/>    }<br/>    else if(keyVal &gt;= 505 &amp;&amp; keyVal &lt;= 515){<br/>        // Tocar la tercera frecuencia de la matriz en el pin 8<br/>        tone(8, notes[2]);<br/>    }<br/>    else if(keyVal &gt;= 5 &amp;&amp; keyVal &lt;= 10){<br/>        // Tocar la cuarta frecuencia en la matriz en el pin 8<br/>        tone(8, notes[3]);<br/>    }<br/>    else{<br/>        // si el valor está fuera de rango, tocar sin tono<br/>        noTone(8);<br/>    }<br/>}</pre> |



## Resultados y conclusiones

Se demostró en esta práctica; el funcionamiento de los interruptores, el uso de la escalera de resistencias, la función `tone()`, se aprendió a utilizar arreglos, almacenar diferentes valores que se relacionan unos con otros para guardar la frecuencia de una escala musical y se pudo apreciar que si se dan diferentes frecuencias podemos ampliar el rango de sonidos.

Al realizar esta práctica hemos aprendido qué es una escalera de resistencias y cómo poder utilizarla para crear nuestro teclado musical. Sería conveniente agregar más interruptores con sus respectivas resistencias para poder crear un teclado mucho más completo, ya que sólo se puede reproducir una nota al mismo tiempo. En esta práctica hemos aprendido qué son y cómo se utilizan los arreglos a la hora de realizar y compilar nuestro programa.

## Referencias

- García, V. (2014). Pulsadores sin rebotes. Experiencias con Arduino. Recuperado el 17 de agosto de 2014, de <http://hispavila.com/3ds/atmega/pulsadores.html>.
- Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Keyboard instrument. Arduino Projects Book* (pp. 79-85). Italia: Arduino LLC.



## Actividad 18. Color mixing lamp

Flor Clara Cubillas Hernández

### Antecedentes

¿Qué es una fotorresistencia y en qué se usa?

Es un componente electrónico que usa una resistencia, cuyo valor depende de la energía lumínosa y va a disminuir cuando aumenta la intensidad de la luz. Se utiliza para medir la cantidad de luz recibida en cualquier instante; también se usa como sensor de luz para los robots, en cámaras fotográficas, las alarmas de seguridad, los sistemas de encendido y apagado del alumbrado de las avenidas en función de la luz ambiente. Veámos la figura 1.



Figura 1. Imagen de la fotorresistencia (izquierda) y símbolo eléctrico (derecha) obtenida de: imagen izquierda con el software Fritzing y símbolo derecha realizado con el software de Visio

¿Qué es la función serial begin?

El cable que se utiliza del microprocesador de Arduino a la computadora es básicamente la comunicación *serial* donde se reciben y transmiten valores. Para que su función se extienda a la comunicación durante el tiempo que se está ejecutando el proceso, se tiene que abrir el puerto *serial* en el programa que descargamos o estemos ejecutando, necesitamos llamar a la función *begin(9600)* en donde 9600 viene siendo la velocidad de transmisión y es importante que todos los dispositivos que se vayan a comunicar tengan la misma velocidad (9600), ya que es un valor estándar y es el que se tiene por defecto Arduino para poder iniciar el proceso de comunicación. Una vez abierto el puerto, los valores que vayan leyendo los sensores los ejecuta en la función *Serial.print(data)*. Este comando manda a imprimir los datos al puerto *serial* y los podemos visualizar en la pantalla.

¿Qué es modulación por ancho de pulso (pwm)?

La modulación por ancho de pulso sirve principalmente para poder controlar los tiempos y movimientos, variar la intensidad de un *LED*, controlar la velocidad de motores y mover servomotores. Generando una onda cuadrada dependiendo del tiempo o variación que se le dé para cambiar el ciclo de trabajo o periodo de encendido y apagado de nuestra señal, pueden simularse voltajes entre 0 y 5



volts. Si repetimos este patrón de encendido-apagado lo suficientemente rápido, por ejemplo con un *LED*. El resultado es como si la señal cambiara entre 0 y 5 volts controlando el brillo del *LED*. Si observamos la figura 2, podemos ver las diferentes señales que existen dependiendo del ciclo de trabajo que se le dé; este tipo de señales es muy utilizado en circuitos digitales que necesitan emular una señal analógica, en la placa Arduino UNO sólo puede llevarse a cabo en las terminales 3,5,6,9, 10 y 11.

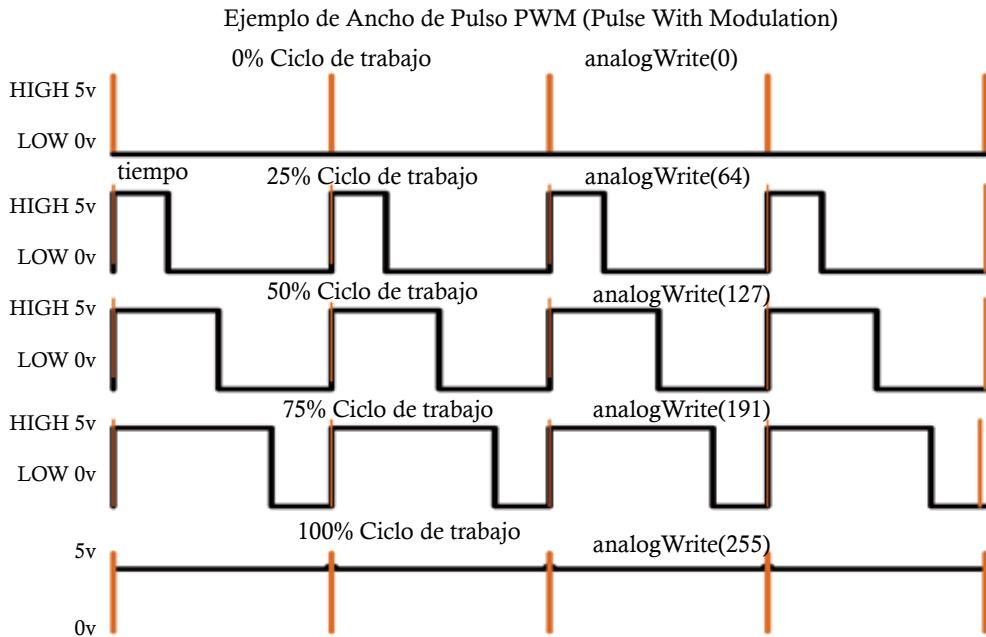


Figura 2. Ejemplo de ancho de pulso *PWM*. Realizado con el *software* de Visio

Podemos ver que Arduino maneja una frecuencia de 500Hz para *PWM*, en donde cada ciclo (espacio entre líneas anaranjadas) dura 2 milisegundos. Este valor lo podemos hacer con la función analógica *analogWrite()*, la cual recibe como parámetro un entero entre 0 y 255, donde 255 es 100% del ciclo *HIGH*, 127% es el 50% *HIGH* y 0 es 100% del ciclo en *LOW*.

## Introducción

En esta práctica vamos a crear una lámpara que cambia suavemente de color dependiendo de las condiciones de iluminación externa. Se conocerá cómo es el funcionamiento de las salidas digitales y analógicas de nuestro microcontrolador Arduino uno. Para esto utilizaremos una fotorresistencia (sensores que cambian su valor de resistencia en función de la cantidad de luz que reciben) y una técnica para simular la variación de voltaje por medio de modulación por ancho de pulso (*PWM*) para apagar y encender nuestro *LED*. También se usa la técnica *MAP*, la cual cambia rápidamente el valor del *pin* de salida entre *HIGH* y *LOW* en un lapso determinado. Este cambio se produce más rápido de lo que el ojo humano puede detectar, como si fuera una serie de imágenes fijas de forma tan rápida que da la sensación de que está cambiando o hay movimiento. Cuando se varía el valor del *pin HIGH* y *LOW* se está variando la tensión.



Para la conexión de nuestra lámpara nos apoyaremos en la figura 3, en donde nuestras salidas digitales (*Pin 9*, *Pin 10* y *Pin 11*) estarán conectadas a nuestro *LED RGB*, conectado cada *pin* con sus respectiva resistencia eléctrica como protección y el común a (*GND*) de la tarjeta Arduino para su buen funcionamiento. En las entradas digitales (*A0*, *A1* y *A2*) se conectarán las fotorresistencia con sus respectivas resistencias eléctricas y el común a (*GND*) de la tarjeta Arduino.

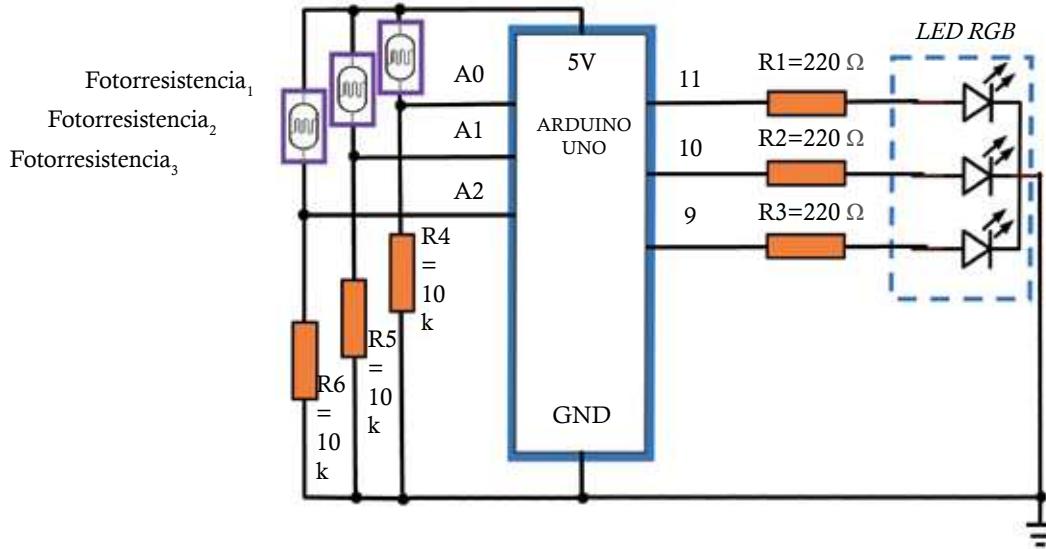


Figura 3. Esquema de conexión de lámpara que cambia su color. Realizada con el software de *fritzing*

## Objetivo

- Aprenderá a usar la función *analogWrite* con sus diferentes parámetros.
- Entenderá cómo es la técnica “modulación por ancho de pulso” y la utilización de la fotoresistencia.
- Controlará el tiempo de encendido y apagado del *LED* mediante el uso de una variable global.
- Aprenderá a utilizar el puerto de comunicaciones de Arduino para recibir y transmitir valores.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A - conexión B).
- 1 microcontrolador Arduino UNO.
- 1 *LED RGB*.
- 3 resistencias  $220\ \Omega$ .
- 3 resistencias  $10\ k\Omega$ .
- 3 fotorresistencia.



## Desarrollo

Tal como se ilustra en la figura 4, lo primero que tenemos que hacer es montar en la *protoboard* nuestro circuito: poner el cable de tierra y la alimentación de 5V; luego se procede a conectar el *LED RGB*, el cual tiene 4 patillas una para el rojo, una para el color verde, una para el color azul y otra que sirve de tierra base común (el cátodo), ver figura 5, donde cada una de las patillas deberá estar conectada a una resistencia de  $220\Omega$ , excepto la patilla más larga que viene siendo el cátodo que va conectada a tierra (*GND*). Enseguida se procede a conectar la patilla de color azul a la salida digital 9, la patilla de color verde se conecta al *pin 10* y la patilla de color rojo se conecta al *pin 11* de Arduino. Cada uno de estos filtros permite sólo la luz de una determinada longitud de onda. El filtro rojo sólo permite el paso de luz roja, el filtro verde pasa sólo la luz verde, y el filtro azul pasa sólo la luz azul. Esto le permite detectar los niveles de color con respecto a la luz que recibe sus sensores y evitar que se queme.

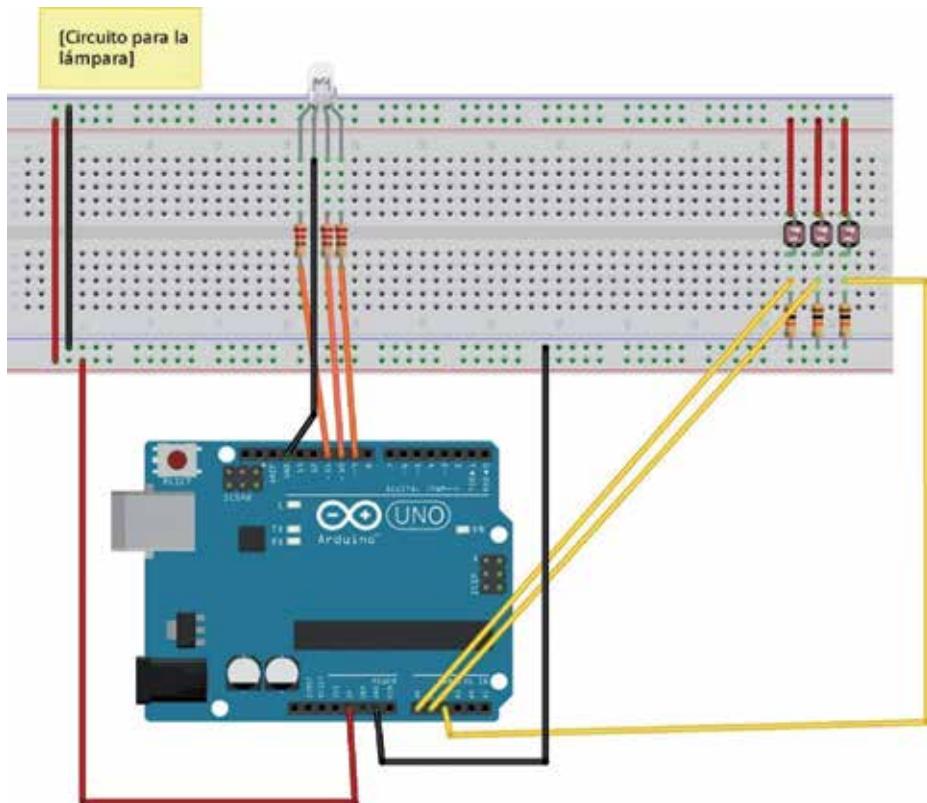


Figura 4. Prototipo de lámpara que cambia de color. Realizada con el software de *fritzing*





Figura 5. LED RGB. Editada en software Photoshop

Después colocamos tres fotorresistencias, cada una conectada en cada extremo en la corriente y el otro extremo conectado a una resistencia de  $10\text{ K}\Omega$  conectadas a tierra (*GND*); las resistencias están en serie con las fotorresistencias. Juntos forman un divisor de voltaje y la tensión en el punto donde se reúnen es proporcional a la relación de sus resistencias. Se conecta la fotorresistencia a la salida analógica en los pines 0, 1 y 2.

Se procede a cargar el código de programación (ver código en la sección correspondiente) en el *IDE* de Arduino. Observar que el *pin* de salida digital proveerá una señal digital. Dicha señal proporcionará cíclicamente un voltaje de 5 volts (nivel ALTO o *HIGH*) durante cierto tiempo o, en otras palabras, durante medio ciclo de la señal; y después proveerá de un voltaje 0 volts (nivel BAJO o *LOW*) durante el restante medio ciclo. Esto significa que en el ánodo del *LED* estará a niveles BAJOS y ALTOS de voltaje, es decir, estará sometido a 0 volts y a 5 volts. El tiempo que el *LED* se mantenga encendido o apagado estará determinado en el código del programa a través de la función *AnalogWrite()*. Esta función recibe como argumento un valor entero que corresponde a una pausa que hará el programa en milisegundos, es decir, nos indica cuánto tiempo permanecerá el *LED* encendido o apagado.

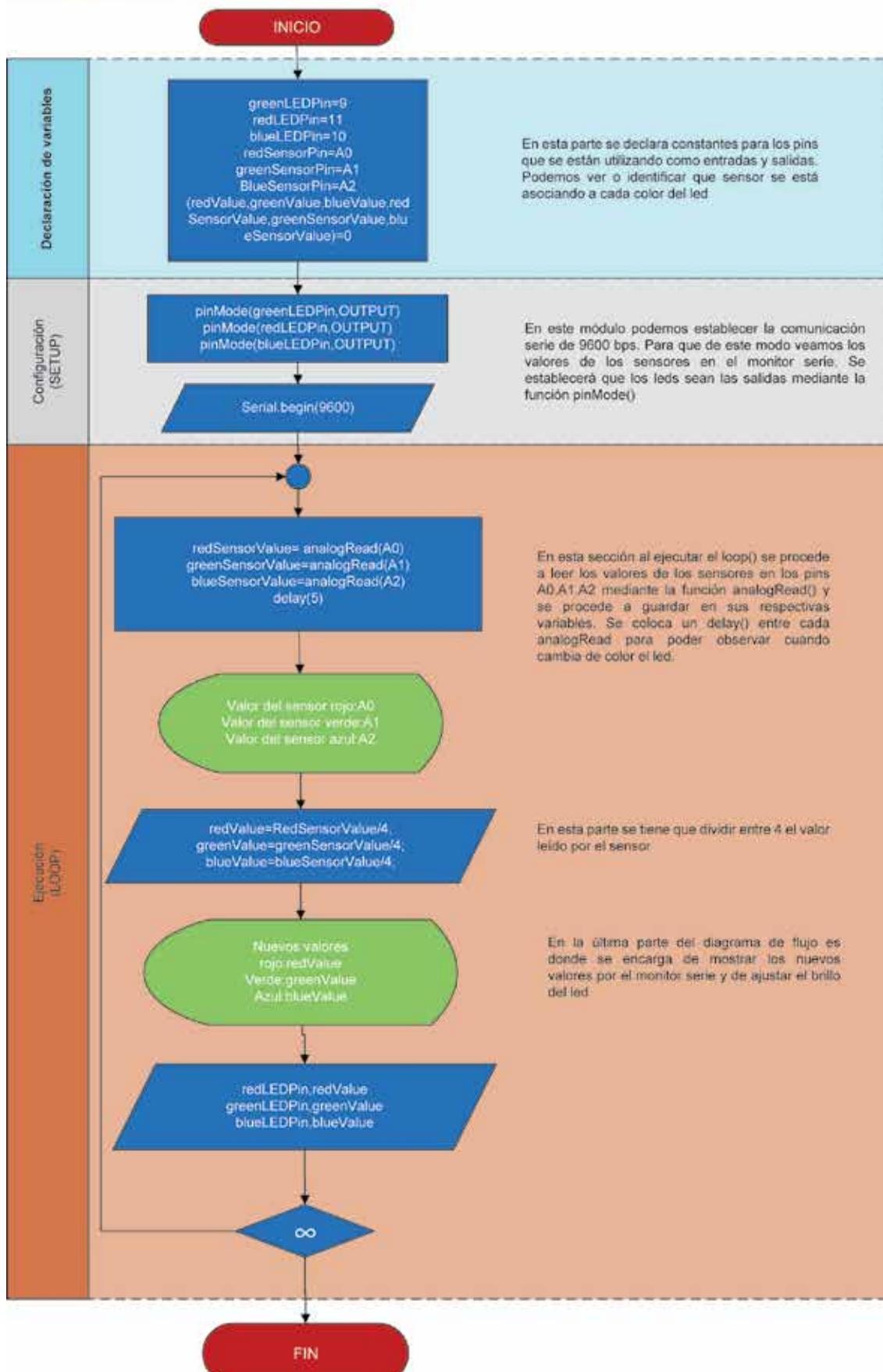
Conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.

Ejecuta el *software* de programación de tu Arduino, carga el código del programa para compilar y verificar que funcione correctamente. El código se explica en la siguiente sección.



## Diagrama de flujo

### Color Mixing Lamp



## Código

|                                   |   |
|-----------------------------------|---|
| Declaración de variables globales | <pre>// Actividad No. 17 "Color Mixing Lamp<br/>const int greenLEDPin = 9; // LED conectado al Pin digital 9<br/>const int redLEDPin = 10; // LED conectado al Pin digital 10<br/>const int blueLEDPin = 11; LED conectado al Pin digital 11<br/>const int redSensorPin = A0; // pin con la fotorresistencia con el rojo<br/>const int greenSensorPin = A1; // pin con la fotorresistencia con el verde<br/>const int blueSensorPin = A2; // pin con la fotorresistencia con el azul<br/>int redValue = 0; // valor a escribir en el LED rojo<br/>int greenValue = 0; // valor a escribir en el LED verde<br/>int blueValue = 0; // valor a escribir en el LED azul<br/>int redSensorValue = 0; // variable para el valor del sensor rojo<br/>int greenSensorValue = 0; // variable para el valor del sensor verde<br/>int blueSensorValue = 0; // variable para el valor del sensor azul</pre> |
| setup()                           | <pre>void setup() {<br/>    //inicializar la comunicación serial a 9600 bps<br/>    Serial.begin(9600);<br/>    // configurar los pines digitales como salidas<br/>    pinMode(greenLEDPin,OUTPUT);<br/>    pinMode(redLEDPin,OUTPUT);<br/>    pinMode(blueLEDPin,OUTPUT);<br/>}</pre>  |



Configuración principal del programa  
loop()

```
void loop() {  
    // Primera lectura de los sensores , leer el valor de la fotorresistencia-rojo  
    redSensorValue = analogRead(redSensorPin);  
    // dar a el convertidor analógico digital un momento de retardo de 5 ms  
    delay(5);  
    //// leer el valor de la fotorresistencia-verde  
    greenSensorValue = analogRead(greenSensorPin);  
    // dar a el convertidor analógico digital un momento de retardo de 5 ms  
    delay(5);  
    // leer el valor de la fotorresistencia-azul  
    blueSensorValue = analogRead(blueSensorPin);  
    // Enviar los valores de salida al monitor serial  
    Serial.print("raw sensor Values \t red: ");  
    Serial.print(redSensorValue);  
    Serial.print("\t green: ");  
    Serial.print(greenSensorValue);  
    Serial.print("\t Blue: ");  
    Serial.println(blueSensorValue);  
  
    /* Para utilizar los valores del sensor para el LED, tenemos que hacer un poco de matemáticas. El ADC  
    proporciona un número de 10 bits, pero analogWrite () utiliza 8 bits. Podemos dividir las lecturas de los  
    sensores por 4 para mantenerlos en el rango de la salida. */  
    redValue = redSensorValue/4;  
    greenValue = greenSensorValue/4;  
    blueValue = blueSensorValue/4;  
    // imprimir los valores asignados  
    Serial.print("Mapped sensor Values \t red: ");  
    Serial.print(redValue);  
    Serial.print("\t green: ");  
    Serial.print(greenValue);  
    Serial.print("\t Blue: ");  
    Serial.println(blueValue);  
  
    /* El valor de PWM de cada LED */  
    analogWrite(redLEDPin, redValue);  
    analogWrite(greenLEDPin, greenValue);  
    analogWrite(blueLEDPin, blueValue);  
}
```



## Resultados y conclusiones

Se demostró en esta práctica, el funcionamiento de las terminales digitales y específicamente las terminales 9, 10 y 11 de nuestra tarjeta Arduino UNO; además hemos aprendido a utilizar una fotoresistencia como sensor que nos proporciona datos de entrada cuando se cambiaron los valores en el monitor *serial*. Al cubrir cada sensor con un plástico, cada uno de estos reaccionó a la longitud de onda perteneciente al color de dicho plástico; independientemente, se utilizó la función *analogWrite* y la técnica *MAP* para hacer que el *LED* nos muestre la información obtenida por los sensores, donde se aprendió a utilizar el cable serial para tener comunicación con el microprocesador Arduino hacia la computadora.

Al realizar esta práctica se desarrolló el manejo del microcontrolador Arduino, el funcionamiento de las terminales digitales específicamente de las terminales 9, 10 y 11 de la tarjeta Arduino UNO. Se describieron las características generales del *hardware* y *software* utilizado del microcontrolador Arduino, así como las de una fotoresistencia, su simbología y la identificación física; se aprendió a utilizar la modulación de ancho de pulsos, la función *analogWrite* y la técnica *MAP*; se aprendió a utilizar el puerto serie configurando la velocidad en 9600 bps, a través de este tipo de comunicación podemos enviar y recibir datos desde nuestro Arduino a la computadora.

## Referencias

Begin () (s.f.). Arduino LLC. Recuperado el 30 de Agosto de 2014, de <http://arduino.cc/en/Serial/Begin>

PWM () (s.f.). Arduino LLC. Recuperado el 30 de Agosto de 2014, de <http://arduino.cc/en/pmwiki.php?n=Tutorial/PWM>

AnalogWrite() (s.f.). Arduino LLC. Recuperado el 30 de Agosto de 2014, de <http://arduino.cc/en/pmwiki.php?n=Reference/analogWrite>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 52-60). Italia: Arduino LLC.

Fotoconductores de una pieza (fotoresistencias) (s.f.). Instituto Tecnológico de la Laguna. Recuperado el 7 de noviembre de 2013, de [http://www.uib.cat/depart/dfs/GTE/education/industrial/tec\\_electronica/teoria/resistores\\_variables.pdf](http://www.uib.cat/depart/dfs/GTE/education/industrial/tec_electronica/teoria/resistores_variables.pdf)



## Actividad 19. Control de bloqueo

Edgar Ruiz Rojas

### Antecedentes

*¿Por qué se necesitan construir circuitos que sirvan para controlar bloqueos?*

Muchos dispositivos elaborados hoy en día nos ayudan a controlar aspectos de nuestra vida cotidiana de forma mucho más sencilla, sin necesidad de revisar continuamente dichas situaciones o aspectos de forma manual.

Una función importante de estos dispositivos, es la capacidad de bloquear la ejecución de ciertas acciones, bajo determinadas circunstancias que se presenten, así como el desbloquear y continuar con dichas acciones, al presentarse otras situaciones también.

Algunos ejemplos de dispositivos con control integrado para bloquear y desbloquear acciones son, entre otros:

- Una bomba de agua: tiene programado en qué momento se debe activar para subir agua a un tinaco y en qué momento se debe desactivar para dejar de hacerlo.
- Un horno eléctrico o un horno de microondas: ambos tienen un mecanismo para medir el tiempo en el cual deben dejar de emitir calor internamente.
- Un refrigerador: tiene un mecanismo para activar la emisión de frío y otro para desactivar el mismo, etcétera.

*¿Cómo se puede armar un circuito con controles de bloqueo en Arduino?*

Para armar un circuito que nos permita bloquear y desbloquear, primero se le deben conectar componentes que reciban señales del exterior. Después, se deben programar acciones que deben de realizar cuando el Arduino esté en un estado de bloqueado y cuando esté en un estado de desbloqueado también.

Los componentes que pueden recibir señales del exterior son, entre otros: un potenciómetro (con el cual regulamos la resistencia del paso de electricidad), un fotorresistor (para medir la intensidad de la luz), un sensor de temperatura, un zumbador o *piezo* (para medir la vibración de golpes externos), un botón de pulsación o *push button*, etcétera.

Una vez recibida una señal de entrada debemos poner el Arduino en un estado de bloqueo o de desbloqueo interno. En programación esto se puede hacer creando una variable en donde se almacene dicho estado. Por ejemplo, podemos crear una variable de tipo entero, llamada *estado*, a la que le asignemos sólo dos posibles valores numéricos: 1, para indicar un estado de bloqueo y 0, para indicar un estado de desbloqueo.

Después se deben utilizar las estructuras de control, como la estructura *if* (condicionante “si...”), para llevar a cabo diversas acciones que nos permitan resolver una necesidad específica. Las acciones que se realicen dependerán del valor que tenga la variable de estado, justo en un determinado momento.



## Introducción

En esta práctica diseñaremos un circuito con el que podamos controlar el cierre de una caja, con una tapa en su parte superior y que tenga con un cerrojo. En este circuito vamos a implementar un mecanismo que nos ayude a bloquear y a desbloquear dicho cerrojo de la caja. El circuito estará programado para que, cuando la caja esté cerrada, con un cierto número de golpes que se le den y con intensidad, abra el cerrojo, entonces se podrá abrir la caja y el circuito cambiará a un estado de desbloqueado.

Cabe aclarar que los circuitos creados para controlar sistemas de seguridad y de accesos implementados en la realidad son mucho más complejos que la práctica que vamos a realizar. Sin embargo, este circuito se elaborará de la forma más sencilla posible para fines didácticos y de enseñanza, principalmente.

La acción para bloquear el cerrojo la vamos a hacer oprimiendo un botón de pulsación o *push button*. El cerrojo de la caja será una hélice que estará conectada a un servo motor. Los golpes para abrir la caja se recibirán como pulsaciones por un componente *piezo*. Asimismo, se usarán tres ledes especiales, visibles desde el exterior de la caja para indicar si el circuito está bloqueado, desbloqueado o para observar los golpeteos válidos que se den, que nos permitirán abrir la caja con posterioridad.

Se ha visto en prácticas anteriores que un componente *piezo* se puede usar como generador o emisor de sonidos al suministrarle energía eléctrica de forma diferida, es decir, en forma de frecuencias. Por el contrario, si se le suministra energía eléctrica constante al *piezo* y se le conecta un cable de lectura en el extremo opuesto, justo antes de su conexión a tierra, ahora el *piezo* servirá como un dispositivo medidor de golpes o de vibraciones del exterior.

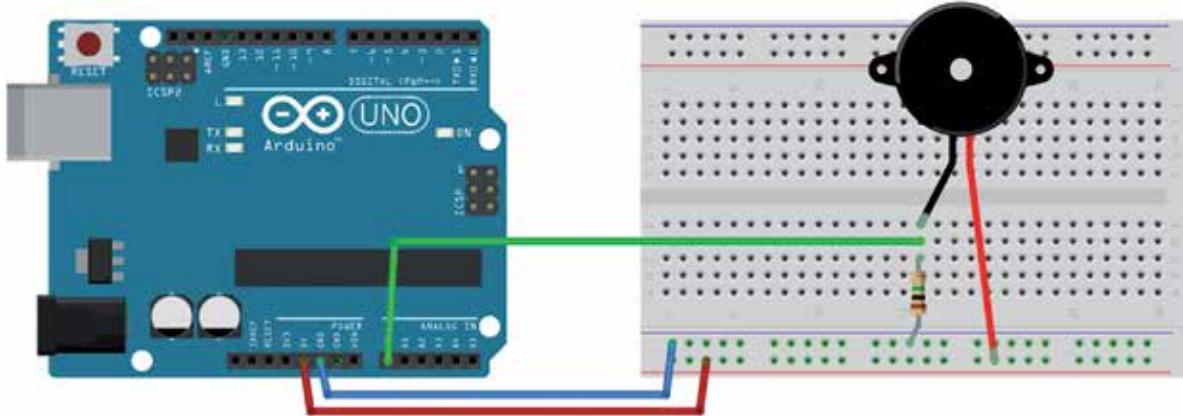


Figura 1. Conexión del *Piezo* para recibir vibraciones del exterior. Imagen obtenida del software Fritzing

En la figura 1 podemos apreciar que el cable lector de vibraciones del *piezo* está conectado a la entrada analógica A0 del Arduino. La lectura obtenida de vibraciones con el *piezo* siempre será de tipo analógica, y ésta se realizará usando la función de programación *analogRead()*. Vamos a aprovechar la ventaja del *piezo* para recibir vibraciones del exterior y, con ello, activar un mecanismo de seguridad que nos ayude a bloquear y desbloquear una puerta de acceso de forma automatizada con nuestro circuito.



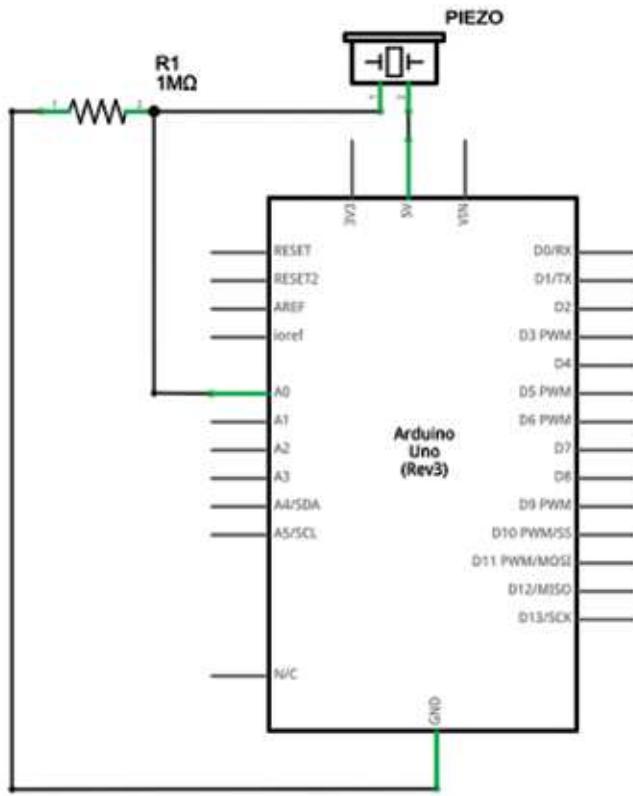


Figura 2. Circuito básico de conexión de un *piezo* en el Arduino, para recibir vibraciones del exterior.

Imagen obtenida del software *Fritzing*

Aunque la elaboración de esta práctica implica montar el circuito dentro de una caja, por cuestiones de prácticas vamos a omitir este último paso de montaje para concentrarnos principalmente en el armado del circuito, así como en los aspectos teóricos y de programación que todo esto conlleva.

## Objetivos

Al finalizar la práctica, el alumno:

- Diferenciará la forma en la cual se debe de conectar un componente *piezo* para que se pueda usar como un dispositivo de entrada (un lector de vibraciones), en lugar de un dispositivo de salida (un emisor de sonido).
- Aprenderá mediante estructuras de control, en programación, a bloquear y a desbloquear sistemas de control operados por medio de circuitos electrónicos.
- Aumentará su destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y el funcionamiento de cada uno de ellos.



## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB*.
- Microcontrolador Arduino UNO.
- Tarjeta de pruebas de circuitos o *protoboard*.
- Botón de pulsación, *switch* o *push button*.
- 3 ledes de colores: rojo, verde y amarillo.
- 1 *piezo zumbador*.
- 1 servo motor.
- 1 capacitor de  $100 \mu\text{f}$ .
- 1 resistor de  $10 \text{k}\Omega$ .
- 3 resistores de  $220 \Omega$ .
- 1 resistor de  $1 \text{ M}\Omega$ .

## Desarrollo

En un extremo de la *protoboard* conecta el ánodo (su parte positiva) del *LED* rojo al pin número 5 del Arduino. De igual forma, conecta el ánodo del *LED* verde al pin número 4, y el del *LED* amarillo al pin número 3. Conecta los cátodos de los tres *LED* a Tierra (*GND*). Conecta las resistencias de  $220 \Omega$  entre los ánodos de los ledes, como en la figura 3:

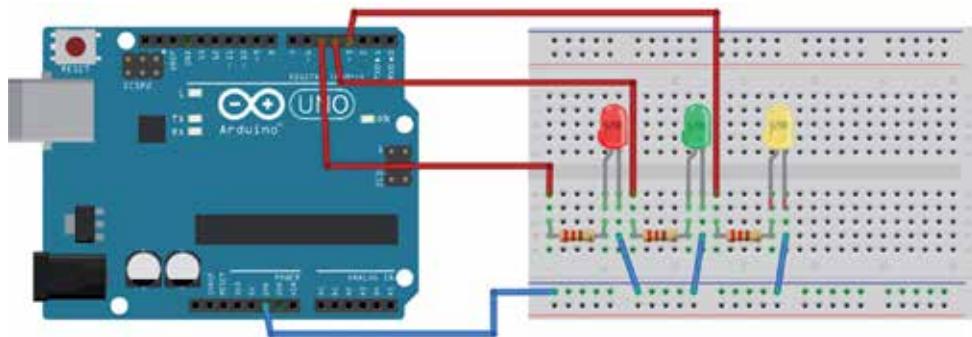


Figura 3. Conexión inicial de los ledes. Imagen obtenida del software *Fritzing*

En el otro extremo de la *protoboard* conecta el botón de pulsación o *push button*. Conecta un cable para recibir la entrada de pulsación del botón, con destino al pin número 2 del Arduino. Utiliza el resistor de  $10 \text{k}\Omega$  para conectar el botón de pulsación a tierra, como en la figura 4:



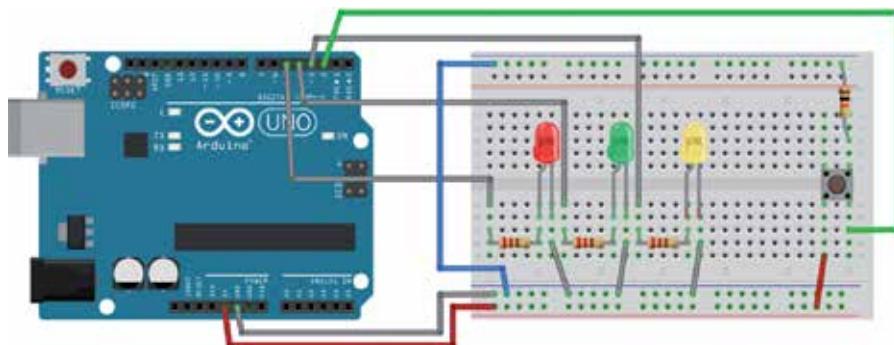


Figura 4. Conexión del botón de pulsación en el circuito. Imagen obtenida del software Fritzing

Conecta en la parte central de la *protoboard* el servo motor, conectando también el capacitor de 100  $\mu\text{F}$  sobre el mismo. Conecta un cable que vaya del *pin* analógico número 9 del Arduino, a la conexión de entrada del servo motor, como en la figura 5:

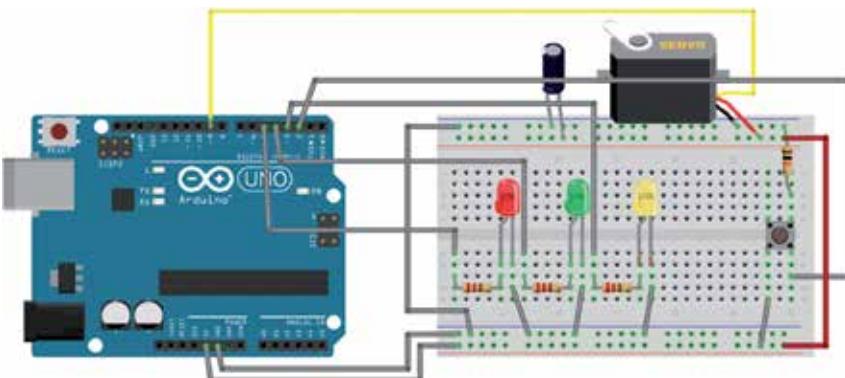


Figura 5. Conexión del servo motor y el capacitor en el circuito. Imagen obtenida del software Fritzing

Conecta el componente *piezo*, en el espacio de la *protoboard* que queda entre los ledes y el botón de pulsación o *push button*. Utiliza el resistor de  $1\text{ M}\Omega$  para conectar la salida del *piezo* a Tierra (*GND*). Entre el cátodo del *piezo* y el resistor conecta un cable que se dirija al *pin* analógico A0 del Arduino. El circuito debe quedar armado en su totalidad, como en la figura 6:

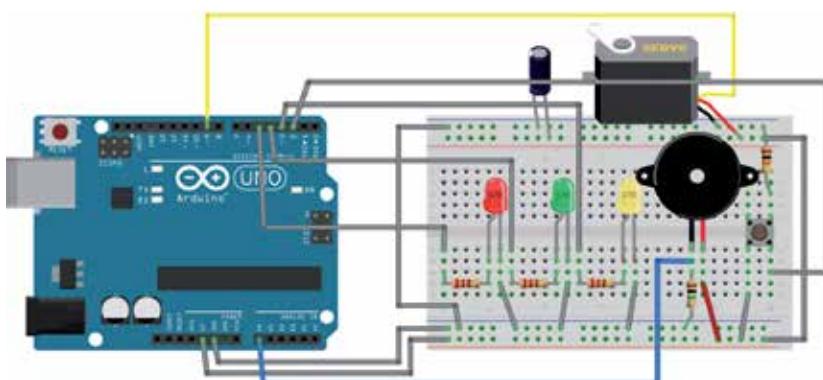


Figura 6. Conexión del componente piezo en el circuito. Imagen obtenida del software Fritzing



Conecta la tarjeta Arduino al puerto *USB* de tu computadora.

En tu computadora abre el programa Arduino.

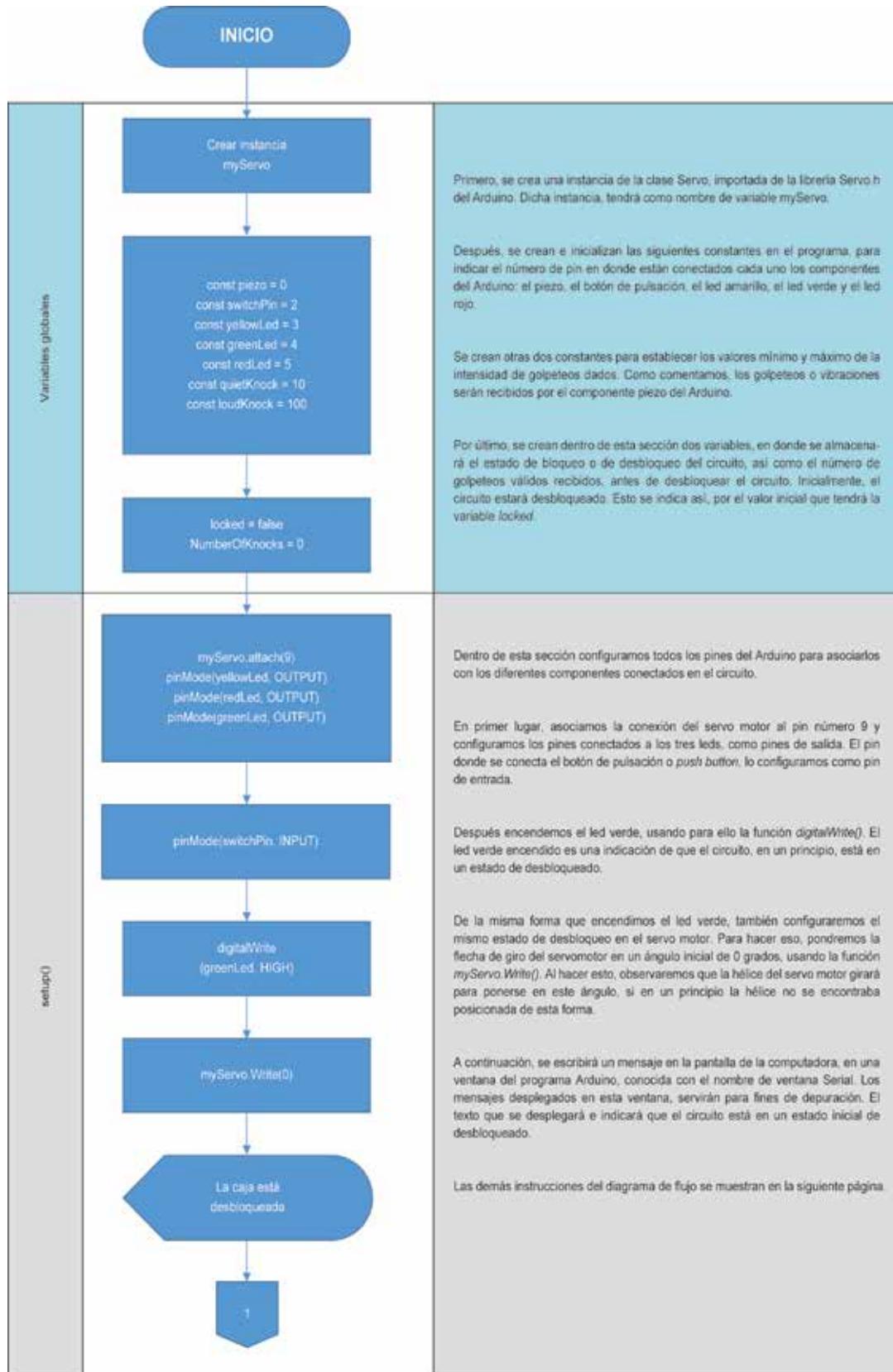
Carga el programa llamado “*KnockLock*”. Para hacerlo selecciona el conjunto de opciones del menú de Arduino Archivo>Ejemplos>10.StarterKit>p12\_KnockLock. El código de este programa se explicará en la siguiente sección.

Carga el programa *KnockLock* en el microprocesador Arduino. Para ello, selecciona la opción del menú Archivo>Cargar, o bien, oprime el botón de flecha a la derecha, en la barra de botones de la ventana de Arduino.

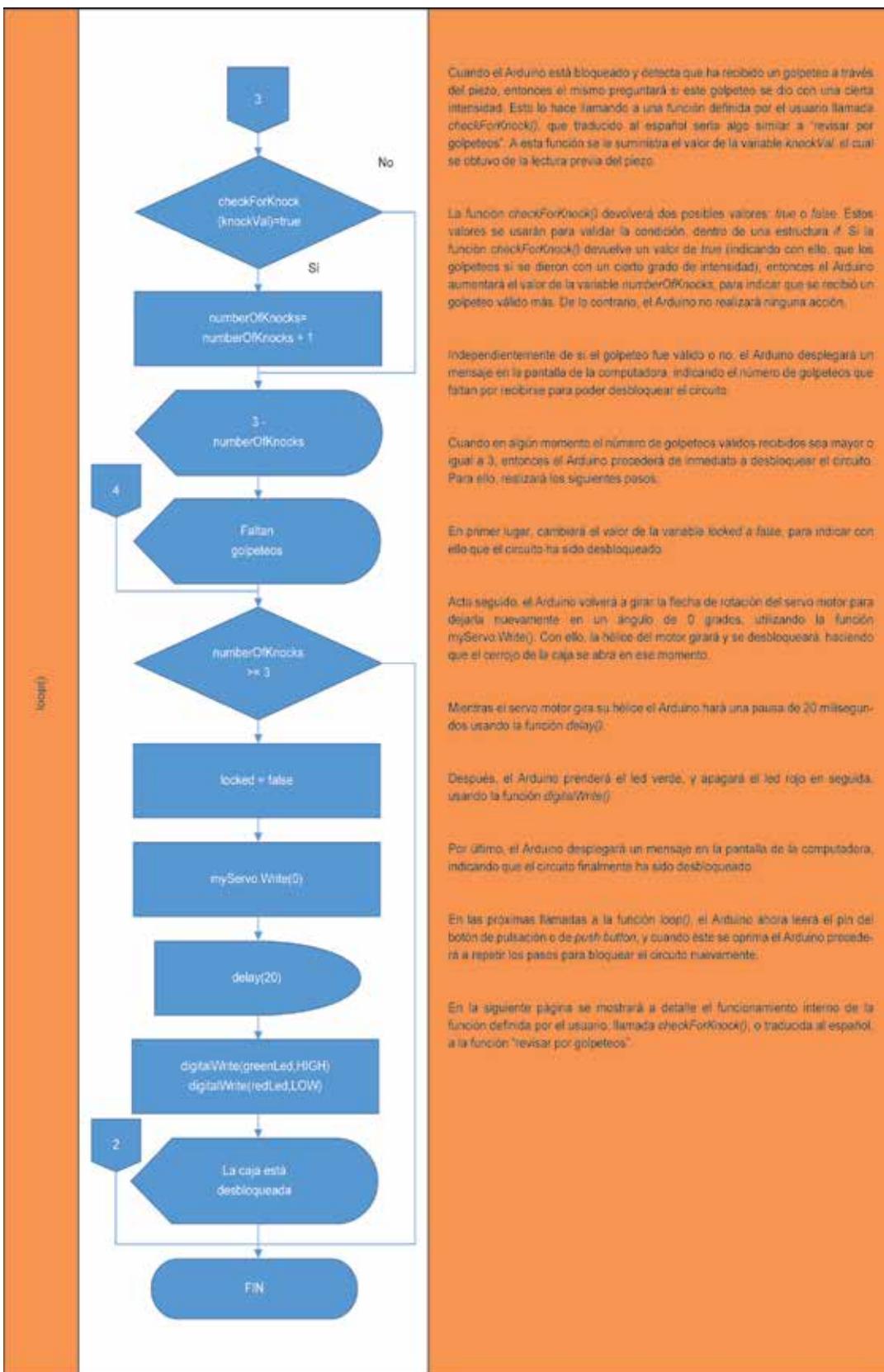
1. Una vez cargado el programa, oprime el botón de pulsación o *push button* en el circuito. Notarás que el *LED* verde se apaga, el *LED* rojo se prende, y el servo motor se pone en un estado de bloqueo. Golpea ligeramente con los nudillos de tus dedos la superficie cercana a la *protoboard* tres veces seguidas. Observarás que, cada vez que golpeas la superficie con una cierta intensidad el *LED* amarillo se enciende por unos instantes; esto significa un golpe válido para desbloquear el circuito. Repite el golpeteo tres veces, y observarás que el circuito se desbloqueará, cambiando el estado del servo motor y prendiéndose ahora el foco verde.

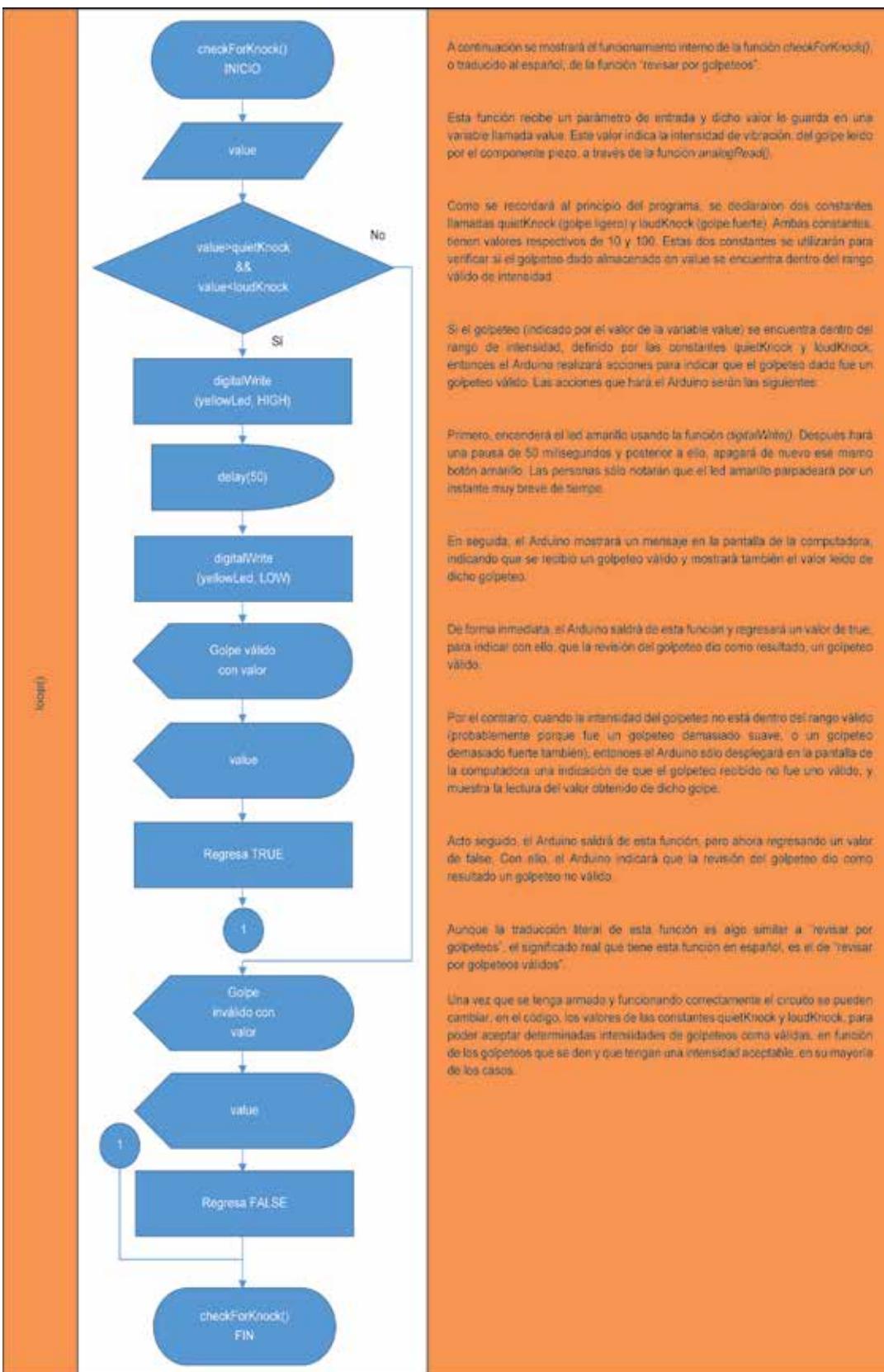


## Diagrama de flujo









## Código

Declaración de variables globales

```
/*
Ejemplo del Arduino Starter Kit
Proyecto 12 - Knock Lock
Este programa está escrito, para acompañar al Proyecto 12 en el
Arduino Starter Kit
Componentes requeridos:
1 resistencia de 1 Mega ohm
10 resistencias de 10 kilo ohm
tres resistencias de 220 ohm
piezo
servo motor
botón de pulsación
un LED rojo
un LED amarillo
un LED verde
capacitor de 100 uF
Creado el 18 de Septiembre de 2012
por Scott Fitzgerald
Gracias a Federico Vanzati por las mejoras
http://arduino.cc/starterKit
Este código de ejemplo es parte del dominio público
*/
// importamos la librería
#include <Servo.h>
// creamos una instancia de la librería servo
Servo myServo;

const int piezo = A0;    // pin en donde se conecta el piezo
const int switchPin = 2; // pin en donde se conecta el botón de pulsación
const int yellowLed = 3; // pin en donde se conecta el LED amarillo
const int greenLed = 4; // pin en donde se conecta el LED verde
const int redLed = 5; // pin en donde se conecta el LED rojo

// variable para guardar el valor del piezo
int knockVal;
// variable para guardar el valor del botón de pulsación
int switchVal;

// constantes para indicar los límites superior e inferior de un golpeteo válido
const int quietKnock = 10;
const int loudKnock = 100;

// variable para indicar si el circuito está bloqueado o no
boolean locked = false;
// cuántos golpeteos válidos se han recibido
int number_of_knocks = 0;
```



```
void setup(){
    // Conectamos el servo al pin 9
    myServo.attach(9);

    // configuramos los pines de LEDs como pines de salida
    pinMode(yellowLed, OUTPUT);
    pinMode(redLed, OUTPUT);
    pinMode(greenLed, OUTPUT);

    // configuramos el pin del botón de pulsación como pin de entrada
    pinMode(switchPin, INPUT);

    // iniciamos la comunicación serial para depuración
    Serial.begin(9600);

    // encendemos el LED verde
    digitalWrite(greenLed, HIGH);

    // movemos el servo a la posición de desbloqueado
    myServo.write(0);

    // imprimimos el estado inicial, en el monitor serial
    Serial.println("the box is unlocked!");
}

void loop(){
    // si la caja no está bloqueada
    if(locked == false){

        // leemos el valor del pin conectado al botón de pulsación
        switchVal = digitalRead(switchPin);

        // si el botón ha sido presionado, entonces bloqueamos la caja
        if(switchVal == HIGH){
            // cambiamos la variable locked (bloqueado) a "true"
            locked = true;

            // cambiamos el estado de los LEDs
            digitalWrite(greenLed,LOW);
            digitalWrite(redLed,HIGH);

            // movemos el servo a la posición de bloqueado
            myServo.write(90);

            // imprimimos el estado
            Serial.println("the box is locked!");
        }
    }
}
```



```
// esperamos mientras el servo mueve su posición
delay(1000);
}

// si la caja está bloqueada
if(locked == true){

    // revisamos el valor del piezo
    knockVal = analogRead(piezo);
    // si no hay golpeteos válidos suficientes
    if(numberOfKnocks < 3 && knockVal > 0){

        // revisamos para ver si el golpeteo, está dentro del rango válido
        if(checkForKnock(knockVal) == true){

            // aumentamos el número de golpeteos válidos
            numberOfKnocks++;
        }

        // imprimimos el estado de los golpeteos
        Serial.print(3 - numberOfKnocks);
        Serial.println(" more knocks to go");
    }

    // si hay tres golpeteos válidos acumulados
    if(numberOfKnocks >= 3){
        // desbloqueamos la caja
        locked = false;

        // movemos el servo a la posición de desbloqueo
        myServo.write(0);

        // esperamos mientras el servo se mueve
        delay(20);

        // cambiamos el estado de los LEDs
        digitalWrite(greenLed,HIGH);
        digitalWrite(redLed,LOW);
        Serial.println("the box is unlocked!");
    }
}
}

// esta función revisa, si un
// golpeteo leído está dentro del rango mínimo y máximo válidos
```



```
boolean checkForKnock(int value){  
    // si el valor del golpeteo es mayor que  
    // el mínimo, y menor que el máximo aceptado  
    if(value > quietKnock && value < loudKnock){  
        // encendemos el LED de estado  
        digitalWrite(yellowLed, HIGH);  
        delay(50);  
        digitalWrite(yellowLed, LOW);  
        // imprimimos el estado  
        Serial.print("Valid knock of value ");  
        Serial.println(value);  
        // la función regresa true  
        return true;  
    }  
    // si el golpeteo no está dentro del rango aceptado  
    else {  
        // imprimimos el estado  
        Serial.print("Bad knock value ");  
        Serial.println(value);  
        // la función regresa false  
        return false;  
    }  
}
```



## Resultados y conclusiones

Se demostró en esta práctica que el componente *piezo* se puede utilizar para que, al conectarlo de una cierta forma, pueda servir como un dispositivo de entrada para leer vibraciones de golpes dados, cercanos a un circuito.

Se demostró también que, con el armado de circuitos electrónicos, en donde se combinan varios componentes a la vez (en lo que cada uno de ellos cumple una determinada función), se pueden implementar diversos mecanismos de control que pueden permitir bloquear o desbloquear diversas acciones y, con ello, pueden resolverse muchas necesidades que se lleguen a presentar.

A diferencia de las prácticas anteriores, en donde se elaboraron circuitos sencillos para probar la funcionalidad de cada componente por separado, el desarrollo de esta práctica requirió combinar varios componentes a la vez e integrarlos todos ellos de forma conjunta, para elaborar un proyecto en donde se lograra resolver una determinada necesidad. Esto está un poco más apegado a la realidad.

El nivel de complejidad de esta práctica puede considerarse entre intermedio y avanzado. Para que este circuito se pueda completar con éxito, se requerirá que sus creadores cuenten con bases sólidas, respecto al funcionamiento y la correcta conexión de cada uno de los componentes utilizados, así como a muchos aspectos relacionados con programación, referente al *software*.

La elaboración de este circuito ayudará a detectar las fortalezas y debilidades de cada uno de los desarrolladores. Una vez detectadas éstas, se podrán tomar las decisiones didácticas que se consideren necesarias.

## Referencias

Knock Sensor (s.f.). Arduino LLC. Recuperado el 20 de junio de 2014, de <http://www.arduino.cc/en/pmwiki.php?n=Tutorial/KnockSensor>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 124-135). Italia: Arduino LLC.



## Actividad 20. Dado de ledes

Edgar Ruiz Rojas

### Antecedentes

¿Se puede simular con el Arduino el lanzamiento de un dado al aire para obtener un número al azar?

Sí es posible simular el lanzamiento de un dado al aire con Arduino. Se puede armar un circuito que contenga un botón de pulsación o *push button*. Al oprimir el botón, simularíamos el lanzamiento del dado al aire. Se pueden instalar unos ledes para simular las vueltas del dado. Estos ledes van a ir gradualmente cambiando de valores hasta llegar a un valor aleatorio final, cuando se simule que el dado ha dejado de rodar.

¿El resultado obtenido al simular el lanzamiento del dado es un número fijo ya establecido por la computadora?

No. Se diseñará el circuito para que, en cada nueva simulación de lanzamiento, el Arduino siempre nos entregue resultados diferentes. Inclusive, se diseñará el mismo para que el Arduino siempre nos entregue resultados, que ni siquiera nosotros como diseñadores hayamos definido de manera previa.

¿Cómo podemos hacer que la computadora o los circuitos electrónicos nos entreguen datos, que no sepamos con anticipación?

Para integrar esta funcionalidad se pueden obtener con el Arduino números aleatorios, a través de una función interna de programación llamada *random()*. Esta función tiene instrucciones integradas que generan una secuencia de números que se pueden considerar como aleatorios hasta cierto grado y que actualmente se utilizan en una amplia variedad de aplicaciones.

¿Los números aleatorios generados por la computadora son números aleatorios en verdad?

Estrictamente hablando, no. Lo deseable es que la computadora siempre genere números aleatorios de forma real, pero ésta sólo contendrá instrucciones que siempre se ejecutarán de una misma forma, para intentar generar números, que sólo simulen ser aleatorios.

En la introducción de este apartado se verá más a detalle el funcionamiento interno de la computadora para generar números aleatorios.

¿Se necesitarán crear muchas variables en el programa de Arduino, para que cada una de ellas maneje el valor de cada uno de los dispositivos conectados?

No necesariamente. Para que sea más sencilla la programación y el diseño de circuitos se han desarrollado herramientas que nos ayudan a concentrar y a manipular datos de un mismo tipo, concentrados todos ellos en una única variable. A esta estructura de datos se le conoce con el nombre de matrices o, más comúnmente, con el nombre de arreglos. En la introducción también se explicará con más detalle la estructura de organización de arreglos.



## Introducción

En esta práctica diseñaremos un circuito que simule el lanzamiento de un dado y que regrese un número como resultado de dicho lanzamiento. El número obtenido será generado de forma aleatoria por la computadora.

Para poder generar este circuito se tendrán que incluir en la programación las funciones especiales para generar números aleatorios, llamadas *random()* y *randomSeed()*. A continuación, se explicará la forma en la cual se generan números aleatorios en la computadora, así como los aspectos que se tienen que considerar sobre la misma.

Para generar números aleatorios la computadora procesa instrucciones que combinan resultados de diferentes operaciones, como son: residuos de divisiones, funciones trigonométricas, seno, coseno, tangente, entre otras. Combinando todos estos resultados, la función *random()* siempre entregará un número, que a simple vista parecerá aleatorio, pero que en realidad no lo es. En realidad, una vez obtenido este número la computadora lo guardará en un lugar interno, para generar a partir del mismo otro número pseudo-aleatorio, al volverse a invocar a la misma función *random()*, en una ocasión posterior. El número guardado lo reutilizará suministrándolo como una nueva entrada para las siguientes operaciones de residuo, seno, coseno, etcétera.

De inicio, esto se podrá ver como un fallo o una vulnerabilidad importante. Sin embargo, los números generados de esta forma a través de la computadora han ayudado a resolver una amplia variedad de necesidades en el mundo real. En especial, han ayudado a resolver problemas en donde se necesita probar la funcionalidad de determinados sistemas. A estos sistemas se les suministra una cantidad numerosa de entradas aleatorias para garantizar la efectividad de los mismos.

Para su funcionamiento interno la función *random()* siempre dependerá del suministro de un número de entrada. Este número podrá ser un número de entrada inicial (que tiene definido la computadora y que tendrá siempre un mismo valor, al llamar a la función *random()* por primera vez) o bien, será un número generado y devuelto por la misma función *random()*. En este último caso, la computadora volverá a reutilizar este mismo número para generar otro siguiente número pseudo-aleatorio, al llamar a la función *random()* una próxima vez.

El hecho de que la computadora utilice un mismo número de entrada, al llamar a la función *random()* por primera vez, siempre será un inconveniente. Así, la secuencia de números aleatorios generados será siempre la misma, y eso le quitará a la función *random()* el carácter de generar números que intenten ser lo más aleatorios posibles.

Para resolver este problema se necesitará invocar una función de manera previa, llamada *randomSeed()*. Esta función deberá recibir algún dato de entrada aleatorio, llamado semilla, justo al momento en el cual se comienza a ejecutar el programa. Esta semilla puede ser, por ejemplo, la fecha y hora de la computadora, representada como un número que indique el total de segundos transcurridos a partir de una fecha específica, o bien, puede ser el dato variable de algún puerto de entrada del Arduino.

De esta forma, la función *randomSeed()* generará una semilla de entrada inicial, siempre distinta para la función *random()*. También se podrá garantizar que los números aleatorios generados, en todo momento, siempre tendrán una secuencia distinta en cada ejecución.



El circuito estará conformado por un botón de pulsación o *push button*, así como por siete ledes, los cuales mostrarán un resultado aleatorio, después de haber simulado el lanzamiento del dado. Al oprimir el botón *push button*, los ledes se prenderán y apagarán simultáneamente durante un periodo de tiempo, simulando que son los puntos del dado dando vueltas al aire.

Todos los valores que tengan los ledes los almacenaremos en una sola variable, la cual será un arreglo de datos de tipo entero, de siete elementos. Todo lo que necesitamos saber sobre arreglos y sus usos se describe a continuación.

Un arreglo es una estructura de datos en la cual podemos almacenar muchos datos o muchos valores, todos de un solo tipo, y que todos ellos los podemos identificar con el nombre de una sola variable. Los arreglos están organizados en forma de elementos y dentro de cada elemento se puede almacenar un solo valor. Cada elemento de un arreglo se podrá diferenciar del resto a través de un número consecutivo de identificación, llamado subíndice.

Cuando definimos un arreglo en un lenguaje de programación, generalmente indicamos las siguientes cosas, respecto a ese nuevo arreglo:

- El nombre de la variable que tendrá el arreglo.
- El tipo de datos que va a contener el arreglo.
- El número de elementos que va a contener el arreglo.
- Las dimensiones que va a tener dicho arreglo.

Un ejemplo de definición de arreglo es el siguiente:

```
int ArregloDatos[4];
```

En este caso estamos definiendo con esta instrucción lo siguiente:

- Estamos creando una variable para manejar un arreglo, cuyo nombre es ArregloDatos.
- Los datos que va a contener esta variable son de tipo entero.
- El número de elementos que va a contener este arreglo son cuatro.
- Este arreglo sólo contendrá datos, organizados en una sola dimensión.

Una vez creado el arreglo, podemos almacenar valores dentro de sus cuatro elementos generados, de la siguiente forma:

```
ArregloDatos[0] = 26;  
ArregloDatos[1] = 32;  
ArregloDatos[2] = 48;  
ArregloDatos[3] = 59;
```

Como se mencionó arriba, este arreglo tiene cuatro elementos, pero cada elemento está identificado por la siguiente secuencia de subíndices numéricos: 0, 1, 2 y 3. El primer elemento del arreglo siempre estará identificado con el subíndice 0. Los demás se identificarán por los siguientes subíndices consecutivos (1, 2 y 3).



Con Arduino se puede crear una variable de arreglo y asignar valores a todos sus elementos al mismo tiempo. A la acción de asignar valores a variables por primera vez comúnmente se le denomina inicializar variables. La forma de crear un arreglo e inicializarlo a la vez, en Arduino, es como sigue:

```
int ArregloDatos[4] = { 26, 32, 48, 59 };
```

Para hacer uso de alguno de los valores almacenados en el arreglo tenemos que indicar dos cosas principales:

1. El nombre de la variable que contiene el arreglo.
2. El subíndice que tiene el valor que deseamos utilizar.

Por ejemplo, si queremos obtener el valor del segundo elemento del arreglo ArregloDatos para asignarlo dentro de otra variable, podemos acceder a este valor de la siguiente forma:

```
int otra_variable = ArregloDatos[1];
```

En este caso, el valor que se almacenará dentro de la variable otra\_variable será el número 32. Debemos recordar que el segundo elemento está identificado con el subíndice número 1. El primer elemento de todo arreglo en Arduino, siempre va a estar identificado con el subíndice número 0.

Si deseamos crear un arreglo de dos dimensiones, la forma de hacerlo será como la siguiente:

```
int Coordenadas[3][3];
```

Este arreglo (Coordenadas), contendrá nueve elementos. En él se podrán almacenar hasta nueve valores, un valor dentro de cada elemento. Los elementos estarán organizados, en forma de 3 filas y 3 columnas.

Para asignar valores a los elementos de dicho arreglo se tendrán que especificar los subíndices de cada una de las dimensiones, como se muestra a continuación:

```
Coordenadas[0][0] = 36;  
Coordenadas[0][1] = 25;  
Coordenadas[0][2] = 48;  
Coordenadas[1][0] = 54;  
Coordenadas[1][1] = 15;  
Coordenadas[1][2] = 8;  
Coordenadas[2][0] = 20;  
Coordenadas[2][1] = 86;  
Coordenadas[2][2] = 37;
```



De igual forma que los arreglos de una sola dimensión, se puede crear un arreglo de dos dimensiones e inicializarlo al mismo tiempo. Esta es la forma de crear e inicializar el arreglo Coordenadas, con los valores mostrados arriba:

```
int Coordenadas[3][3] = {  
    { 36, 25, 48 },  
    { 54, 15, 8 },  
    { 20, 86, 37 }  
};
```

Se puede acceder al valor contenido en un elemento de este arreglo, indicando el nombre de la variable de arreglo, seguido de los subíndices de cada una de las dimensiones escritos entre corchetes cuadrados. Ejemplo:

```
int otra_variable = Coordenadas[2][1];
```

En este caso, el valor que se almacenará dentro de la variable con nombre otra\_variable, será el número 86.

Una ventaja que se tiene al almacenar valores en un solo arreglo es que se pueden realizar operaciones de forma masiva sobre todos sus elementos, de forma muy rápida utilizando estructuras de control. Tomemos como ejemplo la creación del arreglo de una sola dimensión:

```
int ArregloDatos[4] = { 26, 32, 48, 59 };
```

Si por alguna razón, necesitamos sumarles el número 3 a todos los elementos que conforman este arreglo, podríamos realizar la operación de suma para todos sus elementos, a través de esta forma:

```
ArregloDatos[0] = ArregloDatos[0] + 3;  
ArregloDatos[1] = ArregloDatos[1] + 3;  
ArregloDatos[2] = ArregloDatos[2] + 3;  
ArregloDatos[3] = ArregloDatos[3] + 3;
```

Este bloque de instrucciones repetidas para cada elemento, en principio, podría funcionar sin contratiempos. Sin embargo, si el número de elementos en el arreglo es muy grande, entonces será muy difícil repetir esta misma instrucción, para cada elemento por separado.

A través de una estructura de control *for* (ciclo de repetición para...), se puede simplificar el aplicar la suma del número 3 a todos los elementos del arreglo, escribiendo la instrucción de suma sólo una vez y manipulando cada elemento a través de su subíndice. Para ello, debemos incrementar el valor del subíndice en cada nueva repetición que se haga, de la siguiente forma:



```
for (x = 0; x < 4; x++) {
    ArregloDatos[x] = ArregloDatos[x] + 3;
}
```

Para un arreglo de dos dimensiones se tendrán que utilizar dos ciclos *for* anidados (es decir, un ciclo dentro del otro), para poder realizar una determinada operación sobre todos sus elementos. Consideremos el arreglo de dos dimensiones del ejemplo anterior:

```
int Coordenadas[3][3] = {
    { 36, 25, 48 },
    { 54, 15, 8 },
    { 20, 86, 37 }
};
```

Para sumar el número 3 a todos sus elementos, tenemos que usar dos ciclos *for* para manipular las dos dimensiones, como se muestra a continuación:

```
for (x = 0; x < 3; x++) {
    for (y = 0; y < 3; y++) {
        Coordenadas[x][y] = Coordenadas[x][y] + 3;
    }
}
```

Si queremos realizar esta acción, sólo sobre los elementos que conforman el subíndice 1 de la primera dimensión del arreglo (es decir, sobre los elementos con valores 54, 15 y 8), entonces lo podremos hacer utilizando un solo ciclo *for* y manteniendo constante el valor del primer subíndice del arreglo, como se muestra en el siguiente código:

```
for (y = 0; y < 3; y++) {
    Coordenadas[1][y] = Coordenadas[1][y] + 3;
}
```

El valor fijo del subíndice puede estar guardado también en una variable de manera previa. De ser el caso, podemos utilizar el valor fijo de esta variable como subíndice de alguna de las dimensiones del arreglo e ir recorriendo todos los demás elementos del otro subíndice, como se muestra a continuación:

```
int subindice_fijo = 1;
for (y = 0; y < 3; y++) {
    Coordenadas[subindice_fijo][y] = Coordenadas[subindice_fijo][y] + 3;
}
```



## Objetivos

Al finalizar la práctica, el alumno:

- Aprenderá a aplicar las funciones que sirven para generar números aleatorios y, con ello, podrá resolver diversas necesidades que se presenten.
- Dominará el uso de arreglos de una y dos dimensiones, y los aplicará para la generación de diversos circuitos electrónicos.

## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB*.
- Microcontrolador Arduino UNO.
- Tarjeta de pruebas de circuitos o *protoboard*.
- Botón de pulsación o *push button*.
- 7 ledes.
- 1 resistencia de 10 Kilo Ohms.
- 7 resistencia de 220 Ohms.

## Desarrollo

En un extremo de la *protoboard*, conecta los ánodos (las partes positivas) de los 7 ledes a las terminales número 2 a 8 del Arduino. Conecta también los cátodos de los 7 ledes a Tierra (*GND*). Conecta las resistencias de 220 Ohms entre los ánodos de los ledes y las terminales de Arduino, como se muestra en la figura 1:

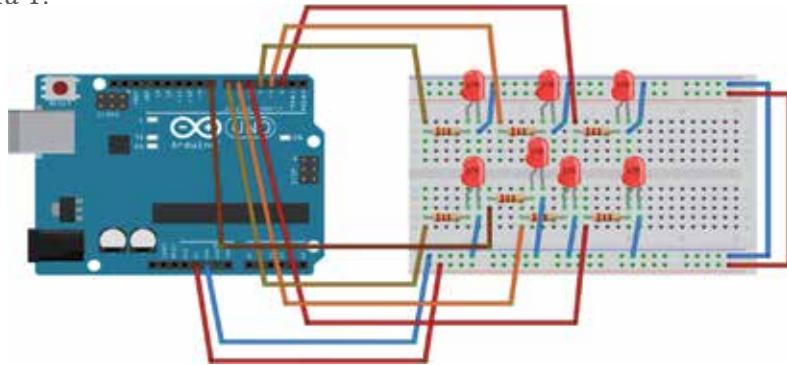


Figura 1. Conexión inicial de los ledes. Imagen obtenida del software Fritzing

En el otro extremo de la *protoboard*, conecta el botón de pulsación o *push button*. Conecta un cable para recibir la entrada de pulsación del botón con destino al pin número 9 del Arduino. Utiliza la resistencia de 10 KOhms para conectar el botón de pulsación a tierra. El circuito armado final debe quedar como el que se muestra en la figura 2:



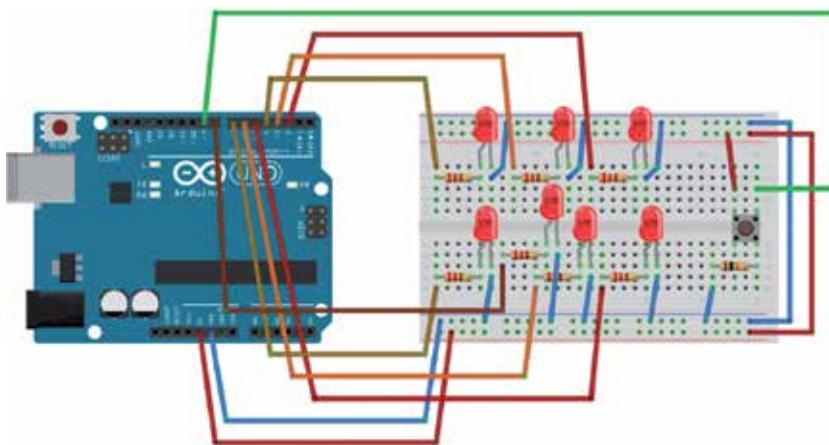


Figura 2. Conexión del botón de pulsación en el circuito. Imagen obtenida del software Fritzing

Conecta la tarjeta Arduino al puerto *USB* de tu computadora.

En tu computadora abre el programa Arduino.

En el editor de Arduino captura el programa que se muestra en la sección “Código” de este documento. Procura capturar el código correctamente para evitar errores de sintaxis o de ejecución del mismo.

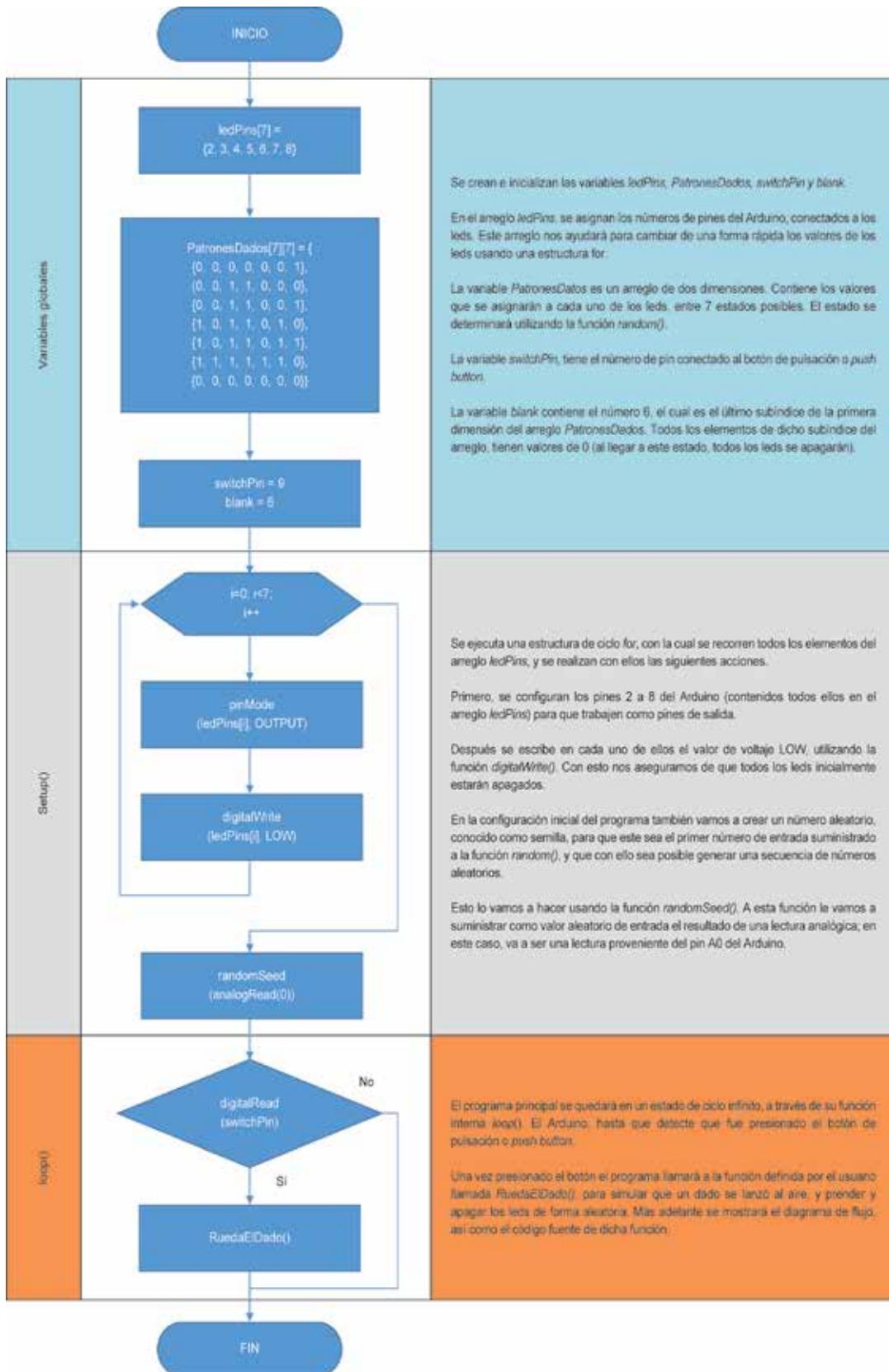
Guarda el programa capturado en un archivo de tu computadora. Para ello selecciona la opción del menú Archivo>Guardar, elige una carpeta y escribe un nombre de archivo con el cual lo quieras guardar.

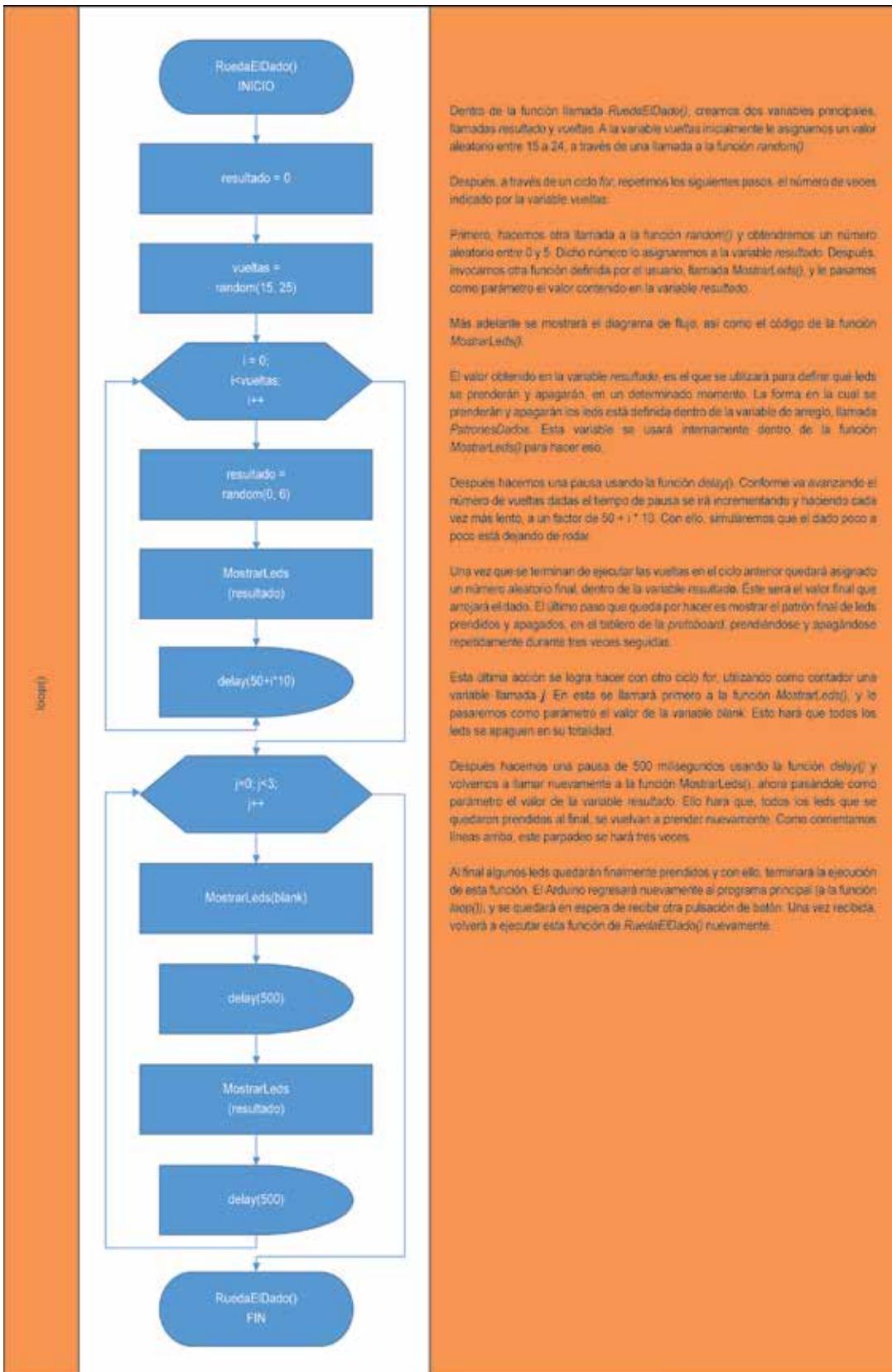
Carga el programa capturado en el microprocesador Arduino. Para ello, selecciona la opción del menú Archivo>Cargar, o bien, oprime el botón de flecha a la derecha, en la barra de botones de la ventana de Arduino.

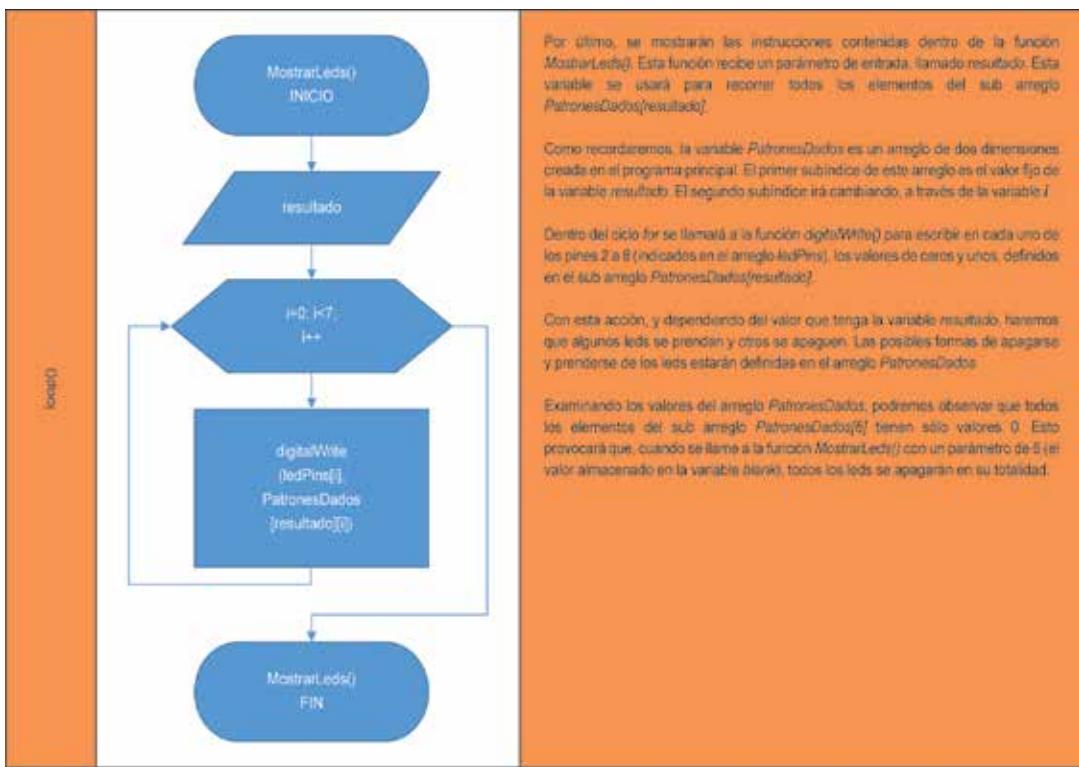
Una vez cargado el programa, oprime el botón de pulsación o *push button* en el circuito. Observarás cómo los ledes cambian constantemente de valor, simulando que un dado es lanzado al aire y que su parte frontal cambia de la misma forma. Conforme pasa el tiempo los cambios se van haciendo más lentos (simulando que el dado está dejando de rodar), hasta llegar a detenerse por completo. Los ledes finalmente se quedarán en un solo estado, determinado por los valores obtenidos con la función *random()* del programa.



## Diagrama de flujo







Por ultimo, se mostrarán las instrucciones contenidas dentro de la función `MostrarLed()`. Esta función recibe un parámetro de entrada, llamado resultado. Esta variable se usará para recorrer todos los elementos del sub-arrreglo `PatronesDatos[resultado]`.

Como recordaremos, la variable `PatronesDatos` es un arreglo de dos dimensiones, creado en el programa principal. El primer subíndice de este arreglo es el valor fijo de la variable `resultado`. El segundo subíndice irá cambiando, a través de la variable `i`.

Dentro del ciclo for se llamará a la función `digitalWrite()` para escribir en cada uno de los pines 2 a 8 (indicados en el arreglo `ledPins`), los valores de ceros y unos, definidos en el sub-arreglo `PatronesDatos[resultado]`.

Con esta acción, y dependiendo del valor que tenga la variable `resultado`, haremos que algunos led's se prendan y otros se apaguen. Las posibles formas de apagarse y prendese de los led's estarán definidas en el arreglo `PatronesDatos`.

Examinando los valores del arreglo `PatronesDatos`, podrás observar que todos los elementos del sub-arreglo `PatronesDatos[6]` tienen sólo valores 0. Esto provocará que, cuando se llame a la función `MostrarLed()` con un parámetro de 6 (el valor almacenado en la variable `blank`), todos los led's se apagaran en su totalidad.

## Código

|                          |   |
|--------------------------|---|
| Declaración de variables | <pre> /* Dado de leds.  Programa con el que se simula el lanzamiento de un dado, y se regresa un número de forma aleatoria.  */  // Pines del Arduino conectados a los leds int ledPins[7] = {2, 3, 4, 5, 6, 7, 8};  // Patrones de encendido y apagado para los dados int PatronesDados[7][7] = {     {0, 0, 0, 0, 0, 0, 1},     {0, 0, 1, 1, 0, 0, 0},     {0, 0, 1, 1, 0, 0, 1},     {1, 0, 1, 1, 0, 1, 0},     {1, 0, 1, 1, 0, 1, 1},     {1, 1, 1, 1, 1, 1, 0},     {0, 0, 0, 0, 0, 0, 0} };  // Pin en donde está conectado el botón de pulsación o push button int switchPin = 9;  // Subíndice del arreglo PatronesDados, donde están todos los valores en blanco int blank = 6; // La función de configuración setup() se ejecuta sólo una vez void setup() {     for (int i = 0; i &lt; 7; i++)     {         // Se configuran todos los pines de los leds como pines de salida         pinMode(ledPins[i], OUTPUT);         // Se apagan temporalmente todos los leds         digitalWrite(ledPins[i], LOW);     }     // Se inicializa la semilla de entrada, para generar números aleatorios     randomSeed(analogRead(0)); } </pre> |
|--------------------------|---|



```
// La función loop() se ejecuta cíclicamente y de forma ininterrumpida
void loop()
{
    // Si se oprimió el botón de pulsación, entonces
    if (digitalRead(switchPin))
    {
        RuedaElDado();
    }
    delay(100);
}

void RuedaElDado()
{
    int resultado = 0;
    // Se obtiene el número de vueltas que simulará dar el dado
    int vueltas = random(15, 25);
    for (int i = 0; i < vueltas; i++)
    {
        // Se obtiene un resultado aleatorio del 0 al 5 (y no del 1 al 6)
        resultado = random(0, 6);
        MostrarLeds(resultado);
        // Se hace una pausa para simular las vueltas del dado, y conforme
        // avanza el ciclo, las vueltas se hacen cada vez más lentas
        delay(50 + i * 10);
    }
    // Una vez obtenido el número entregado por el dado, el Arduino
    // hará parpadear tres veces los leds que quedaron prendidos al final
    for (int j = 0; j < 3; j++)
    {
        MostrarLeds(blank);
        delay(500);
        MostrarLeds(resultado);
        delay(500);
    }
}

void MostrarLeds(int resultado)
{
    for (int i = 0; i < 7; i++)
    {
        // Escribe en los leds conectados a los pines 2 a 8, los valores
        // almacenados dentro del sub arreglo PatronesDado[resultado]
        digitalWrite(ledPins[i], PatronesDado[resultado][i]);
    }
}
```



## Resultados y conclusiones

Se pudo demostrar en esta práctica la aplicación correcta de la función *random()* para la generación de circuitos electrónicos construidos con Arduino. Asimismo, se logró demostrar que el procesamiento de múltiples datos se pudo llevar a cabo de forma muy eficiente, utilizando arreglos de una y dos dimensiones.

El desarrollo de esta práctica ayudará a reforzar el conocimiento de los principios básicos necesarios para generar números aleatorios en la computadora. Servirá para conocer también los aspectos que se deben de tener en cuenta para generar dichos números.

La práctica va a ayudar, de igual forma, a reforzar el dominio de la manipulación de arreglos de una y dos dimensiones para poder usarlos de un modo práctico en la implementación de circuitos electrónicos, cuyo diseño y armado se simplificarán notablemente al usar los mismos.

## Referencias

- Monk, S. (2012). Dado de LEDs. *30 proyectos con Arduino* (pp. 55-59). Madrid: Editorial Esteribor, S.L.
- Número pseudoaleatorio. (2014). En *Wikipedia*. Recuperado el 21 de junio de 2014, de [http://es.wikipedia.org/wiki/N%C3%BAmero\\_pseudoaleatorio](http://es.wikipedia.org/wiki/N%C3%BAmero_pseudoaleatorio).
- Arreglos. (s.f.). Departamento de Computación del CINVESTAV. Recuperado el 21 de Junio de 2014, de <http://computacion.cs.cinvestav.mx/~acaceres/courses/estDatosCPP/node3.html>, página consultada el día.
- Vector (informática). (2015). En *Wikipedia*. Recuperado el 21 de junio de 2014, de [http://es.wikipedia.org/wiki/Vector\\_%28inform%C3%A1tica%29](http://es.wikipedia.org/wiki/Vector_%28inform%C3%A1tica%29), página consultada el día.



## Actividad 21. Manejo de pantalla alfanumérica LCD 16x2

Erika Díaz García / Víctor Hugo Leyva García

### Antecedentes

El mercado de la electrónica doméstica y profesional ofrece diversos dispositivos de características variadas (portátil, fijo, pequeño formato, gran formato, etcétera) que tienen una pantalla de cristal líquido (*LCD*, por sus siglas en inglés) para la presentación de información ya sea en formato de texto, gráficas, imágenes o video. Teléfonos celulares, televisores, relojes, calculadoras, juguetes electrónicos, hornos de microondas y computadoras portátiles son ejemplos de algunos dispositivos digitales que incluyen una pantalla *LCD*.

Los *LCD* más comunes son el alfanumérico y el gráfico. El primero sólo permite el despliegue de caracteres (letras, números, símbolos, etcétera); mientras que el segundo es capaz de desplegar, además de texto, elementos gráficos como puntos, líneas, círculos, rectángulos, etcétera.

En esta práctica aprenderemos a configurar una pantalla *LCD* alfanumérica de matriz de puntos (*dot-matrix*), basada en el *chipset Hitachi HD44780*, la cual recibirá datos desde el microcontrolador Arduino.

### Introducción

#### Pantalla alfanumérica LCD 16x2

Una pantalla alfanumérica es un dispositivo formado por una pantalla de cristal líquido o *LCD* sobre la que se pueden desplegar mensajes formados por caracteres: letras, números y símbolos.

Existen diferentes formatos de pantallas *LCD* alfanuméricas determinados por el número de líneas y el número de caracteres por línea que pueden visualizar. En la figura 1 se muestra un *LCD* de 2 líneas y de 16 caracteres ( ). Otros formatos comunes son 8x1, 16x1, 20x2, 20x4, 40x2 y 40x4.

En la figura 1 se muestra la distribución de las terminales de un *LCD* 16x2, y en la tabla 1 se indica la función que cumple cada *pin*.

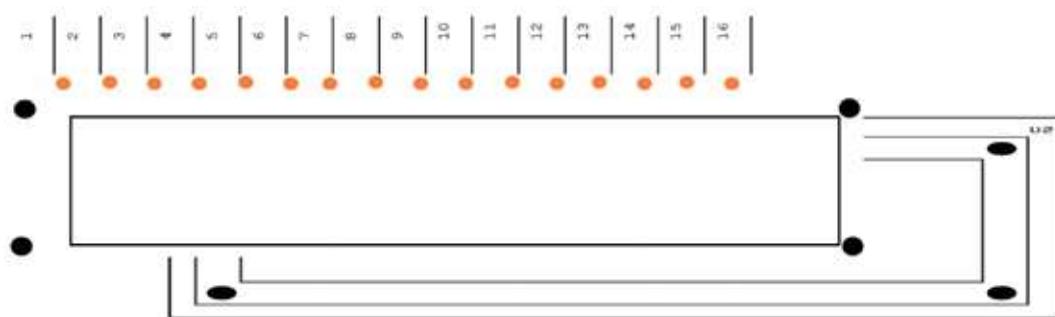


Figura 1 Distribución de pines en una pantalla *LCD*



| Pin  | Símbolo  | Descripción  |
|------|----------|--|
| 1    | $V_{ss}$ | Tierra de alimentación <i>GND</i>  |
| 2    | $V_{cc}$ | Alimentación de +5V  |
| 3    | $V_0$    | Ajuste del contraste de cristal líquido (o a +5 V)   |
| 4    | RS       | Selección del registro (Register Select)<br>RS = 1 => Registro de datos<br>RS = 0 => Registro de instrucciones |
| 5    | R/W      | Selección del modo de operación Lectura/Escritura (Read/Write)<br>R/W = 1 => Lectura<br>R/W = 0 => Escritura   |
| 6    | E        | Habilitación del LCD ( <i>Enable</i> )<br>E = 1 => Habilitado<br>E = 0 => Deshabilitado                        |
| 7-14 | D[0-7]   | Pines de datos bidireccionales.  |
| 15   | LED +    | Ánodo de LED <i>backlight</i>  |
| 16   | LED-     | Cátodo de LED <i>backlight</i>   |

Tabla 1. Descripción de pines de pantalla LCD basada en el *chipset Hitachi HD44780*

Las pantallas LCD tienen integrado un controlador que se encarga de gestionar los puntos de la pantalla, generar y desplazar los caracteres, mostrar el cursor, etcétera; el controlador más común es el HD44780 de *Hitachi*, que provee una serie de instrucciones de alto nivel que facilitan su manejo.

Arduino provee de la biblioteca de funciones *LiquidCrystal.h* para el control de las pantallas LCD basadas en el *chipset Hitachi HD44780* (o compatibles).

#### Biblioteca de funciones *LiquidCrystal.h*

La biblioteca *LiquidCrystal.h* está integrada por un conjunto de funciones que nos permiten el manejo de las pantallas LCD compatibles con el *chipset Hitachi HD44780* de cristal líquido y compatible. En la tabla 2 se enlistan las funciones integradas en la biblioteca *LiquidCrystal.h*



| <i>Liquid Crystal()</i> | <i>no Blink ()</i>           |
|-------------------------|------------------------------|
| <i>begin ()</i>         | <i>display ()</i>            |
| <i>setCursor()</i>      | <i>noDisplay ()</i>          |
| <i>print()</i>          | <i>scrollDisplayLeft ()</i>  |
| <i>clear ()</i>         | <i>scrollDisplayRight ()</i> |
| <i>home ()</i>          | <i>autoscroll ()</i>         |
| <i>write ()</i>         | <i>noAutoscroll ()</i>       |
| <i>cursor ()</i>        | <i>LeftToRight ()</i>        |
| <i>noCursor ()</i>      | <i>rightToLeft ()</i>        |
| <i>blink()</i>          | <i>createChar ()</i>         |

Tabla 2. Funciones de la biblioteca *LiquidCrystal.h*

A continuación se describen las funciones *Liquid Crystal.h ()*, *begin ()*, *print ()* y *set Cursor ()* las cuales se emplearán en la práctica actual.

#### Función *Liquid Crystal ()*

Crea una variable o instancia de tipo *Liquid Crystal* mediante la cual se controlará la pantalla *LCD*. La pantalla se puede controlar mediante 4 u 8 líneas de datos (modos de 4 u 8 bits, respectivamente); en el modo de 4 bits se deshabilitan las líneas de datos *d0*, *d1*, *d2* y *d3*, mientras que en el modo de 8 bits se encuentran habilitadas las 8 líneas de datos (*d0* a *d7*).

La sintaxis para declarar una variable de tipo *Liquid Crystal* en modo de 4 bits es:

*LiquidCrystal nombreDeLaVariable (rs, enable, d4, d5, d6, d7);*

*LiquidCrystal nombreDeLaVariable (rs, rw, enable, d4, d5, d6, d7);*

La sintaxis para declarar una variable de tipo *Liquid Crystal* en modo de 8 bits es:

*LiquidCrystal nombreDeLaVariable (rs, enable, d0, d1, d2, d3, d4, d5, d6, d7);*

*LiquidCrystal nombreDeLaVariable (rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7);*

Donde:

*nombre DeLaVariable* es el nombre de la instancia de tipo *Liquid Crystal*.

*rs* es el número de pin de Arduino que se conecta al *pin RS del LCD*;

*rw* es el número de *pin* de Arduino que se conecta al *pin R/W del LCD*. *rw* es un parámetro opcional que no es necesario especificar cuando el *pin R/W del LCD* se ha conectado al voltaje de referencia (*GND*) del circuito.

*enable* es el número de *pin* de Arduino que se conecta al *pin enable del LCD*;

*d0* a *d7* son los números de las terminales de Arduino que se conectan a los correspondientes *pin* de datos del *LCD*.



### *Función begin()*

Especifica las dimensiones (ancho y alto) de la pantalla *LCD*.

La sintaxis de la función es: *nombreDeLaVariable.begin (columnas, filas)*

Donde:

*nombreDeLaVariable* es una instancia de tipo *Liquid Crystal*;

*columnas, filas* es el número de columnas que tiene el *LCD*; y

*columnas, filas* es el número de filas que tiene el *LCD*.

### *Función print()*

Escribe en el *LCD* una cadena de caracteres, un número o una variable, según se indique.

La sintaxis de la función es: *nombreDeLaVariable.print (cadena\_de\_control)*

Donde:

*nombreDeLaVariable* es una instancia de tipo *Liquid Crystal*;

*cadena\_de\_control* puede ser una cadena de caracteres, una variable, un número o una operación matemática.

### *Función setCursor*

Se establece la posición del cursor en el *LCD*. Esta posición determina la fila y columna en que aparecerán los siguientes caracteres escritos en el *LCD*.

La sintaxis de la función es:

*nombreDeLaVariable.setCursor(columna, fila)*

Donde:

*nombreDeLaVariable* es una instancia de tipo *Liquid Crystal*;

*columna* es la columna en donde se posiciona el cursor (con 0 se selecciona la primera columna).

*fila* es la fila en donde se posiciona el cursor (con 0 se selecciona la primera fila, con la segunda fila).

## Objetivos

- Aprenderá el procedimiento de conexión de una pantalla *LCD 16x2* con el microcontrolador Arduino.
- Utilizará las funciones básicas para el envío de datos a la pantalla *LCD* desde el microcontrolador Arduino.



## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- Pantalla de cristal líquido (*LCD 2x16*).
- Potenciómetro de  $10\text{ k}\Omega$  o mayor.
- Resistencia de  $220\ \Omega$ .

## Desarrollo

Conecta la pantalla de cristal líquido al Arduino, como se indica en la figura 2.

En la tabla adjunta se indica una relación de las conexiones para la pantalla *LCD* con el microcontrolador. De tal manera que la terminal 1 del *LCD* se conecta a la terminal de referencia del Arduino (*GND*); la terminal 2 del *LCD* con la terminal de  $+5\text{ V}$  del Arduino; el resto de las terminales del *LCD* se conectan según se indica en la tabla.

El lector debe notar que las terminales 4 y 15 del *LCD* no se conectan directamente al microcontrolador. La terminal 4 se conecta al cursor central del potenciómetro, mientras que la terminal 5 se conecta a un extremo de la resistencia de  $220\ \Omega$ . Asimismo, una de las terminales fijas del potenciómetro se conecta a la terminal de  $+5\text{ V}$ , y la otra terminal fija se conecta a la referencia (*GND*); mientras que la resistencia de  $220\ \Omega$  también se conecta, por uno de sus extremos, a  $+5\text{ V}$ .

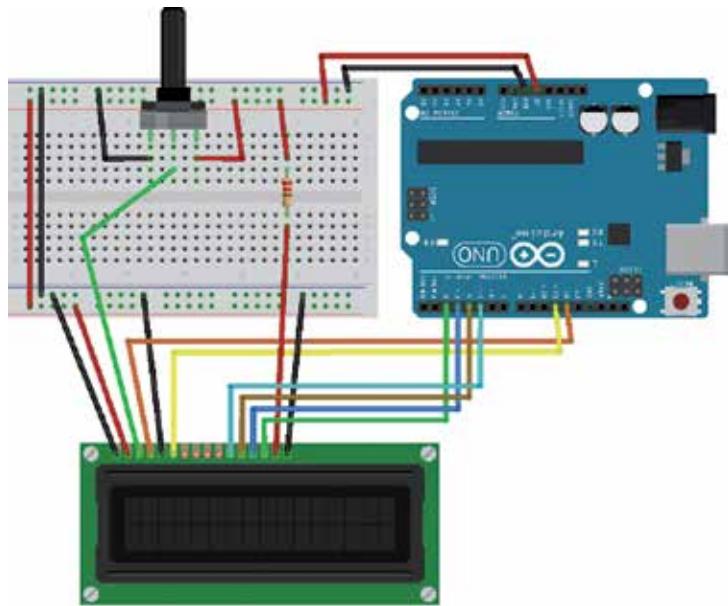


Figura 2. Esquema de conexión de componentes



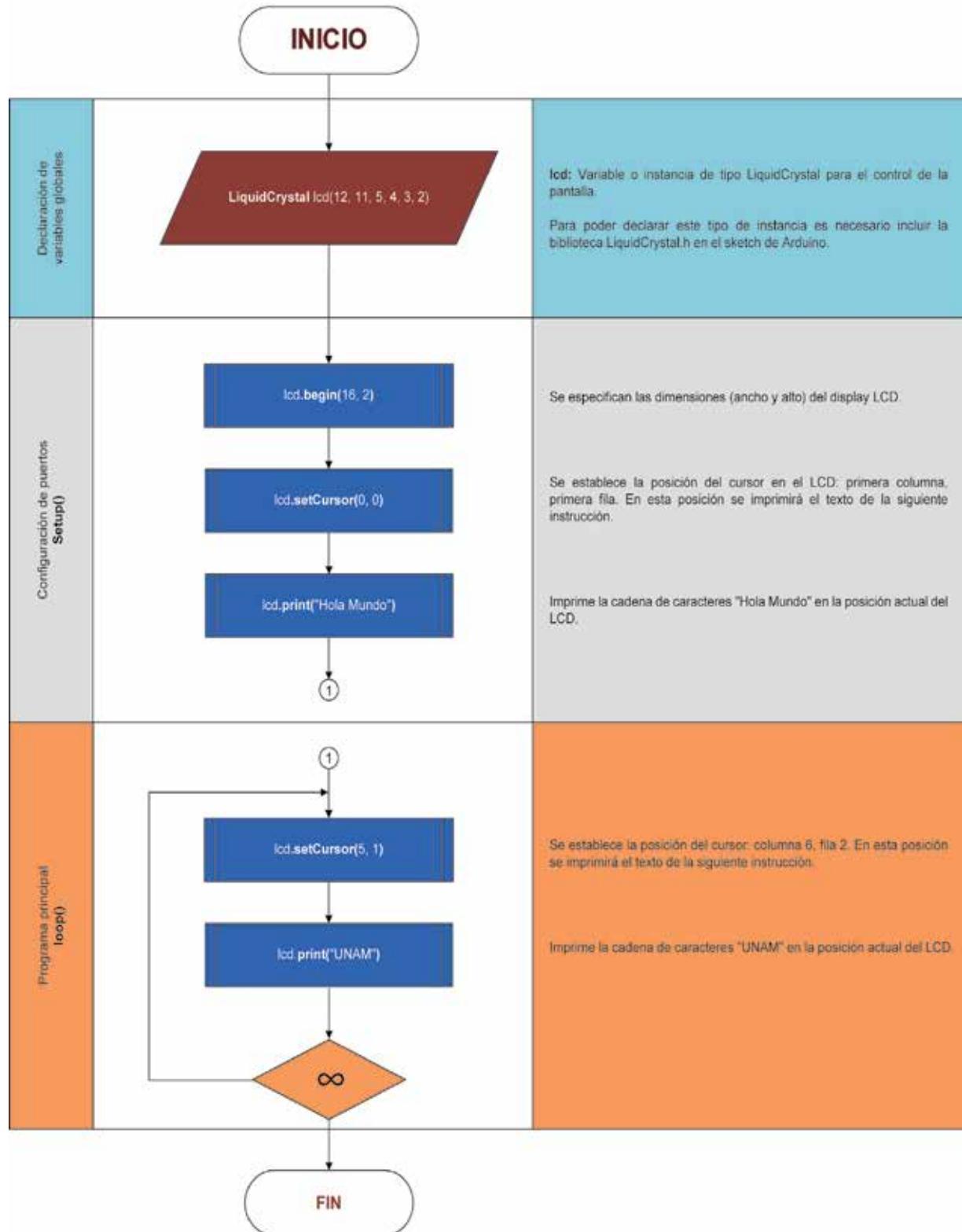
| LCD |          | Arduino | Componente   |
|-----|----------|---------|--------------|
| Pin | Símbolo  | Pin     |              |
| 1   | $V_{ss}$ | GND     |              |
| 2   | $V_{cc}$ | +5V     |              |
| 3   | $V_0$    |         | $R_{cursor}$ |
| 4   | $RS$     | 12      |              |
| 5   | $R/W$    | GND     |              |
| 6   | $E$      | 11      |              |
| 7   | $D0$     |         |              |
| 8   | $D1$     |         |              |

| LCD |         | Arduino | Componente      |
|-----|---------|---------|-----------------|
| Pin | Símbolo | Pin     |                 |
| 9   | $D2$    |         |                 |
| 10  | $D3$    |         |                 |
| 11  | $D4$    | 5       |                 |
| 12  | $D5$    | 4       |                 |
| 13  | $D6$    | 3       |                 |
| 14  | $D7$    | 2       |                 |
| 15  | $LED+$  |         | $R_{220\Omega}$ |
| 16  | $LED-$  | GND     |                 |

Tabla 3. Conexión de componentes entre la pantalla *LCD* con el Arduino y otros componentes



## Diagrama de flujo



## Código

|                                      |   |
|--------------------------------------|---|
| Declaración de variables globales    | <pre> /* Se explica el uso de una pantalla LCD 16X2 compatible con el chipset Hitachi HD44780 Este sketch imprime en el LCD "Hola Mundo" y "UNAM" */ </pre>   |
| Área de configuración setup()        | <pre> // Se incluye la biblioteca LiquidCrystal, que permite que el Arduino controle pantallas LCD #include &lt;LiquidCrystal.h&gt;  /* Se crea una instancia de tipo LiquidCrystal y con nombre lcd, indicando un modo de operación de 4 bits para el LCD */ LiquidCrystal lcd(12, 11, 5, 4, 3, 2); </pre>   |
| Cuerpo principal del programa loop() | <pre> void setup() {   // Se especifican las dimensiones (ancho y alto) del display LCD   lcd.begin(16, 2);   // Se imprime una cadena de caracteres en el LCD.   lcd.print("Hola Mundo"); }  // la función loop() se ejecuta cíclicamente y de manera ininterrumpida void loop() {   // Se establece la posición del cursor: sexta columna, segunda fila.   lcd.setCursor(5, 1);   // Se imprime una cadena de caracteres en el LCD.   lcd.print("UNAM"); } </pre> |

## Resultados y conclusiones

En esta práctica se aprendieron los procesos de conexión de una pantalla alfanumérica *LCD 16x2* con el microcontrolador Arduino. También se hizo uso de las funciones *LiquidCrystal()*, *begin()*, *print()* y *setCursor()* integradas en la biblioteca *LiquidCrystal.h* de Arduino, para el envío de cadenas de caracteres desde al microcontrolador hacia la pantalla *LCD*.

## Referencias

LiquidCrystal - “Hello World!”. (s.f.) Arduino LLC. Recuperado el 21 de noviembre de 2014, de <http://www.arduino.cc/en/Tutorial/LiquidCrystal/>.



## Actividad 22. Arreglos unidimensionales

Víctor Hugo Leyva García

### Antecedentes

En esta práctica introduciremos al lector en el manejo de *arreglos unidimensionales* (también conocidos como vectores, *arrays* o *matrices*), un tipo estructurado de datos que nos permite el manejo de una gran cantidad de datos del mismo tipo organizados mediante un nombre o identificador común.

A manera de introducción reflexiona en el siguiente ejemplo: se pretende llevar un registro de las calificaciones de los alumnos inscritos en cierto curso, para después conocer el promedio de ellas, así como las calificaciones mayor y menor. Una forma de hacerlo es mediante la creación de una estructura de datos, a la cual llamaremos *arreglo*, cuyo nombre o identificador será *evaluación*, el cual almacenará todas las calificaciones y proveerá de mecanismos simples para acceder a la información de interés. Este *arreglo* funcionará a manera de un contenedor dividido en varias celdas, en donde cada celda almacenará una calificación y se le asignará un identificador único llamado *índice*. De tal manera que cada celda se identificará tanto por el nombre del arreglo al que pertenece, como por su *índice* particular.

Para ejemplificar la situación anterior, el arreglo *evaluación* se ilustra en color azul en la figura 1. Dicho arreglo está compuesto de cuatro celdas o elementos (cuadros de color gris), las cuales almacenan el mismo número de calificaciones, es decir, una calificación en cada celda. Las celdas se identifican por el nombre del arreglo al que pertenecen, que en este caso es *evaluación* y por un índice particular; el primer índice es el número cero y los siguientes índices son números enteros consecutivos. En amarillo se indica el nombre de las celdas; el lector debe observar que las cuatro celdas tienen el mismo nombre pero difieren en el índice.

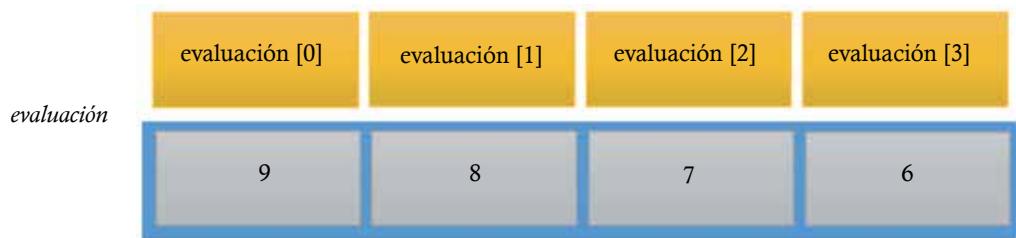


Figura 1. Ejemplo de arreglo unidimensional. Arreglo de nombre *evaluación*

La calificación menor corresponde al elemento *evaluación[1]*, mientras que la calificación mayor corresponde al elemento *evaluación[0]*.

Los arreglos nos permiten almacenar un conjunto finito de datos en una única estructura, lo cual representa dos ventajas importantes:

- Se posibilita un uso eficiente de la memoria del microcontrolador.
- Se disponen de operaciones que operan sobre todos los datos a la vez, y no uno por uno.



## Introducción

Los arreglos son tipos estructurados de datos que ocupan un grupo de celdas en memoria, y se identifican por un nombre y uno o más índices. Pueden ser de una dimensión, de dos y de tres o más dimensiones.

En esta práctica estudiaremos los arreglos unidimensionales.

### Arreglo unidimensional

Un arreglo unidimensional es una colección finita, homogénea y ordenada de datos en la que se hace referencia a cada elemento del arreglo por medio de un índice.

Se considera que es una estructura de datos finita porque el número máximo de elementos está bien determinado. Homogénea, porque todos los elementos del arreglo son del mismo tipo simple de datos; es decir, todos los elementos son exclusivamente enteros, caracteres, etcétera, pero no una combinación de ellos. Ordenada, porque la posición de cada elemento está bien determinada (primer elemento, segundo elemento, etcétera).

La figura 2 muestra la representación gráfica de un arreglo unidimensional.

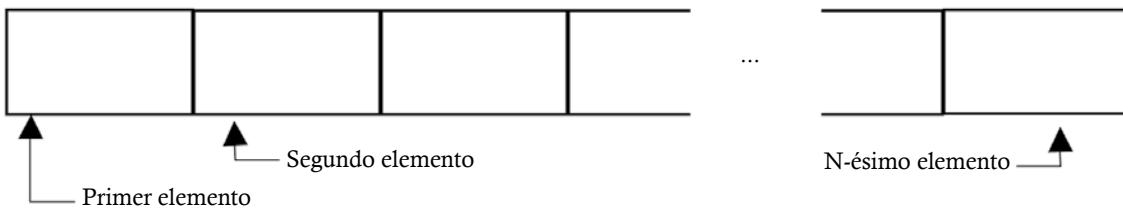


Figura 2. Representación de un arreglo unidimensional

Los arreglos tienen dos partes fundamentales: *a)* los elementos o valores almacenados en cada celda; *b)* los índices, los cuales determinan la posición de cierto elemento dentro del arreglo.

En la figura 1 se ilustró un arreglo de nombre evaluación. Entonces *evaluación[0]* es el nombre del elemento que está en la posición 0, el cual contiene 9; *evaluación[1]* es el nombre del elemento que está en la posición 1, que contiene el valor 8; y así sucesivamente.

En general, el elemento *i*-ésimo está en la posición *i*-1. De tal manera que si el arreglo tienen *n* elementos, sus nombres son: evaluación [0], evaluación [1],..., evaluación [*n*].

El lector debe observar que los índices de un arreglo tienen como límite inferior 0, y como límite superior el tamaño del arreglo menos 1.

Existen tres operaciones básicas para el manejo de arreglos: declaración del arreglo y asignación y lectura de datos. También existen otras operaciones avanzadas, como actualización (inserción, eliminación y modificación), ordenación y búsqueda. Sin embargo, en esta práctica solamente haremos uso de las operaciones básicas. Para ilustrarlas se hará uso de los ejemplos presentados anteriormente.



## Declaración de arreglos y asignación de datos

Un arreglo, al igual que cualquier otro tipo de variable, se debe declarar antes de poder ser utilizado.

La declaración de un arreglo se hace de forma similar a como se declaran otros tipos de datos. Sin embargo, se debe indicar el tamaño o longitud del arreglo.

La sintaxis para declarar un arreglo unidimensional es:

*tipo nombreArray [númeroDeElementos]*

Con tipo se declara el tipo de datos para todos los elementos del arreglo. El tipo de los elementos puede ser entero, real, cadena de caracteres, registro, arreglo, o cualquier otro tipo de datos.

*númeroDeElementos* debe ser un valor ordinal; puede ser una variable, una constante o una expresión que dé como resultado un valor ordinal, *númeroDeElementos* se escribe entre corchetes “[ ]”.

A continuación se describe un ejemplo de declaración de arreglos:

*int myInts[5];*

Se está declarando un arreglo de seis elementos vacíos; es decir, no se asigna valor a los elementos de cada celda (figura 3). En realidad, dichas celdas no están vacías pues, guardan datos ininteligibles, considerados “basura”.

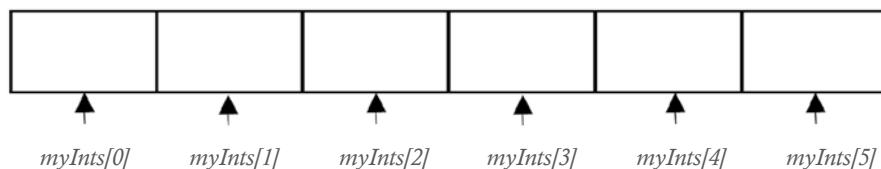


Figura 3. Representación del arreglo *myInts*

*int int mySenVals[5] = { 2, 4, -8, 3, 2};*

Se está declarando el arreglo *my Sen Vals* de cinco elementos, por ello se pone el número entre 5 corchetes [ ]. Además se inicializan los cinco elementos del arreglo mediante los valores indicados entre las llaves {} (figura 4).

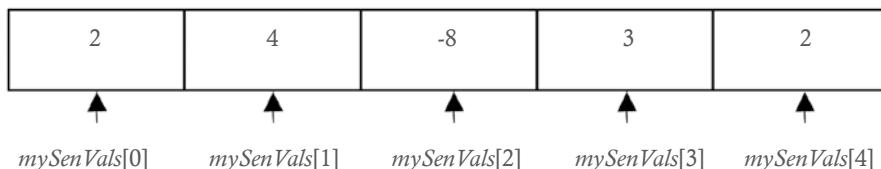


Figura 4. Representación del arreglo *mySenVals*

*int int myArray[] = { 2, 4, 8, 3, 6};*

En este ejemplo se está declarando un arreglo de cinco elementos enteros; aunque no se indica explícitamente el tamaño del arreglo, al no determinar el tamaño entre los corchetes, el compilador cuenta los elementos y crea un arreglo de un tamaño adecuado para almacenar los elementos ya indicados (figura 5).



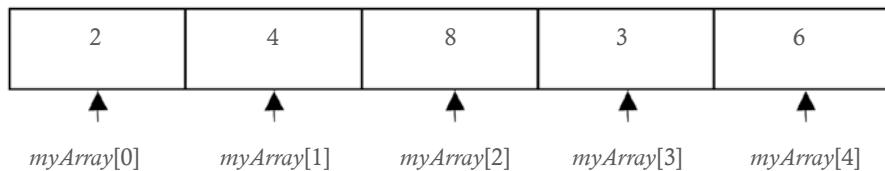


Figura 5. Representación del arreglo *myArray*

`char message[6] = "hello";`

Se declara un arreglo donde cada elemento será un dato de tipo carácter. La cadena de caracteres “hello” está compuesta de cinco caracteres o letras, cada una de las cuales ocupará una celda del arreglo. Sin embargo, el lector debe notar que en la declaración del arreglo de caracteres se estableció que almacenaría seis elementos: un elemento por cada uno de los cinco caracteres de la cadena y un sexto elemento correspondiente al carácter nulo `\n`, mismo que el compilador agrega al final de toda cadena (figura 6). En este punto se debe tener en claro que una cadena de caracteres contiene un carácter nulo al final, el cual debe considerarse al declarar el arreglo de caracteres que la contendrá.

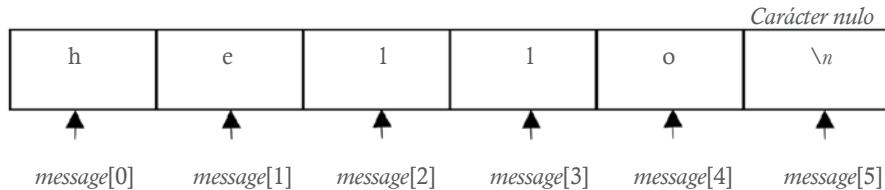


Figura 6. Representación del arreglo *message*

### Lectura de datos

Para acceder a cierto elemento del arreglo se utiliza el nombre del arreglo y el índice entre corchetes. Por ejemplo:

`lcd.print(myArray[2]);`

Esta instrucción visualiza en una pantalla de un cristal líquido (*LCD*) el elemento 3 del arreglo *myArray*, cuyo valor es 8, como se ilustra en la figura 7. El lector debe recordar que el primer índice de un arreglo unidimensional es el número 0. De esta manera, el arreglo *myArray* contiene los siguientes elementos individuales:

*myArray [0]=2, myArray [1]=4, myArray [2]=8, myArray [3]=3 y myArray [4]=6 .*





Figura 7. Pantalla LCD desplegando el elemento `myArray[2]`

## Objetivos

- Aprenderá el manejo de arreglos unidimensionales.
  - Hará uso de una pantalla de cristal líquido (*LCD 16x2*) para visualizar los elementos de un arreglo unidimensional.

## Material utilizado

- Equipo de cómputo.
  - Cable estándar *USB* (conexión A a conexión B).
  - Microcontrolador Arduino UNO.
  - Pantalla de cristal líquido (*LCD 16x2*).
  - Potenciómetro de  $1\text{ k}\Omega$ .
  - Resistencia de  $220\ \Omega$ .

## Desarrollo

Conecta la pantalla de cristal líquido al Arduino, como en la figura 8.

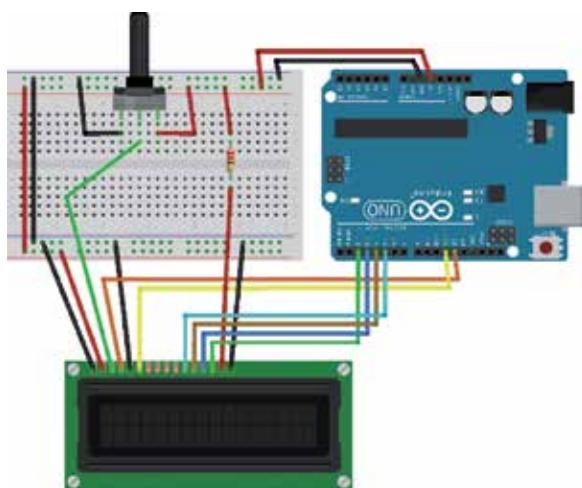


Figura 8. Diagrama de conexión de componentes



Conecta tu tarjeta Arduino a la computadora, como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.

Ejecuta el entorno de programación de Arduino; abre el programa de ejemplo “Arreglos” y cárgalo en tu Arduino. El código se explica en la siguiente sección.

El programa “Arreglos” lo encontrarás en la carpeta de ejemplos que acompañan a las prácticas.

Observarás que la pantalla *LCD* muestra, de manera consecutiva, los índices del arreglo *myArray* así como el valor de sus elementos (figura 9).

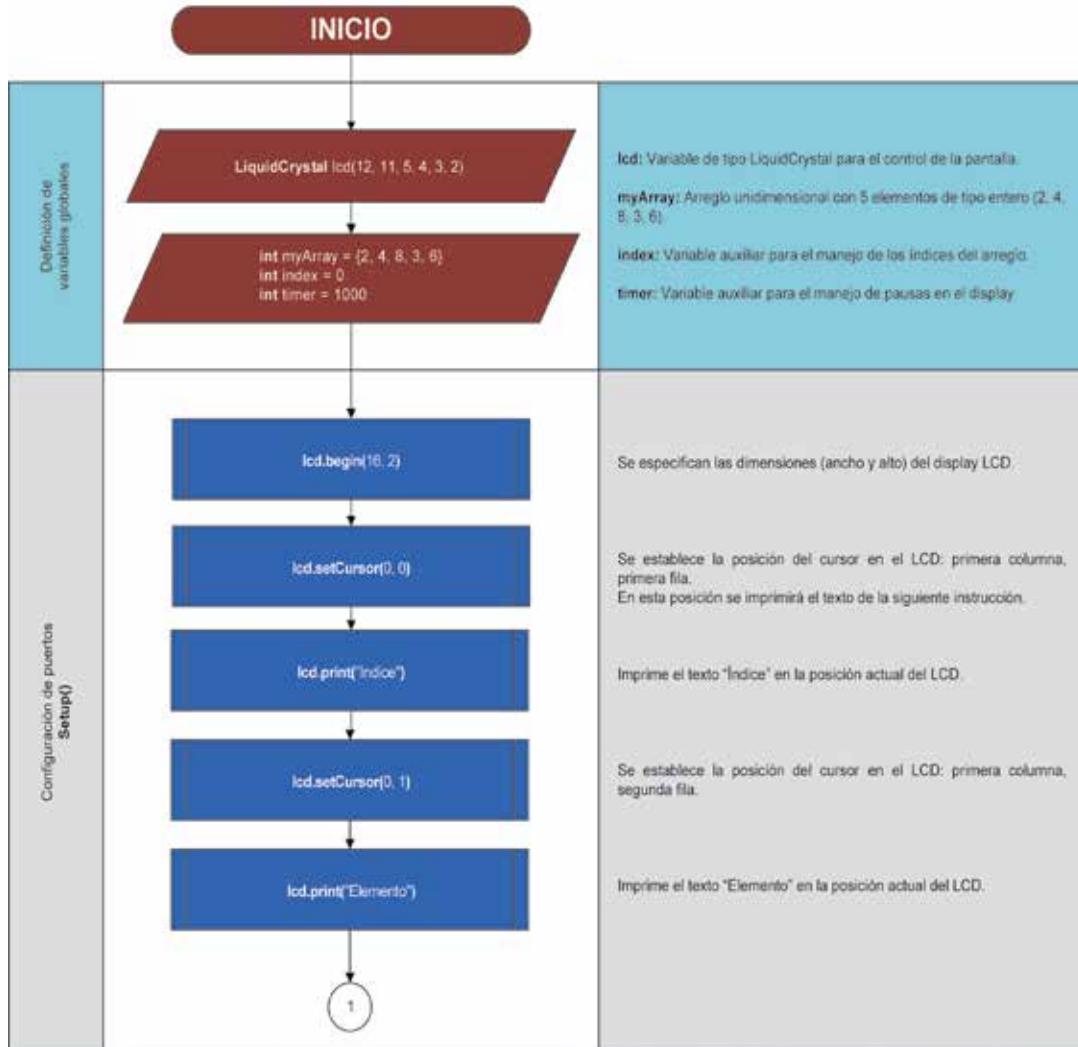


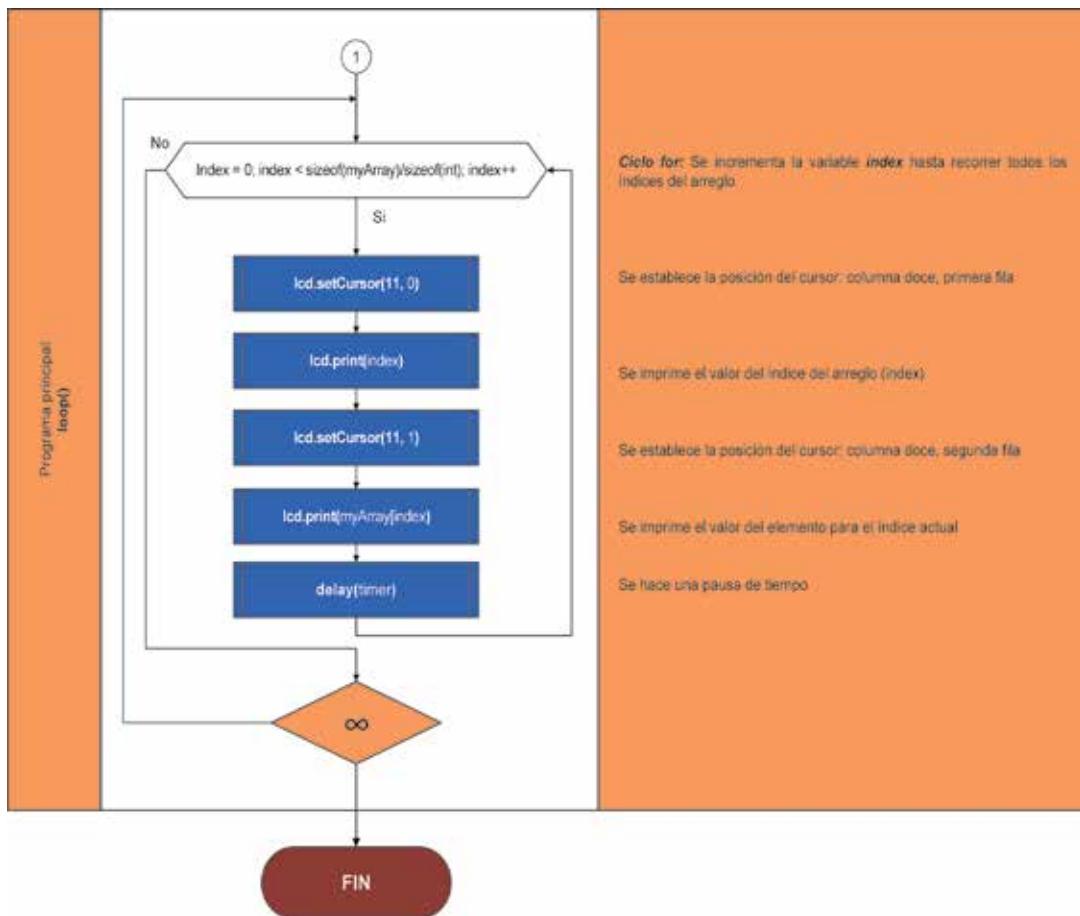
Figura 9. Pantalla *LCD* desplegando los elementos del arreglo *myArray*.

Cambia los elementos del arreglo *myArray*, así como la cantidad y tipo de éstos. Observarás que al cargar nuevamente el código modificado el display mostrará los nuevos elementos.



## Diagrama de flujo





## Código

|                                      |   |
|--------------------------------------|---|
| Declaración de variables globales    | <pre>// Se incluye la biblioteca LiquidCrystal, que permite que el Arduino controle pantallas LCD #include &lt;LiquidCrystal.h&gt;  // Creación de la variable lcd, de tipo LiquidCrystal, para el control de la pantalla LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  /* Creación de un arreglo unidimensional de tipo entero, de 5 elementos y asignación de valores*/ int myArray[ ] = {2,4,8,3,6};  // Variable auxiliar para el manejo de los índices del arreglo int index = 0;  // Variable auxiliar para el manejo de pausas en el display int timer = 1000;</pre>                 |
| Área de configuración setup()        | <pre>void setup() { // Se especifican las dimensiones (ancho y alto) del display LCD lcd.begin(16, 2); // Se establece la posición del cursor: primera columna, primera fila lcd.setCursor(0, 0); // Imprime texto en la posición actual del LCD lcd.print("Índice:"); // Se establece la posición del cursor: primera columna, segunda fila lcd.setCursor(0, 1); // Imprime texto en la posición actual del LCD lcd.print("Elemento:"); }  // la función loop() se ejecuta cíclicamente y de manera ininterrumpida</pre>   |
| Cuerpo principal del programa loop() | <pre>void loop() { // Ciclo for. Se incrementa la variable index hasta recorrer todos los índices del arreglo. for (index=0; index &lt; sizeof(myArray)/sizeof(int); index++) { // Se establece la posición del cursor: columna doce, primera fila lcd.setCursor(11, 0); // Se imprime el valor del índice del arreglo (index) lcd.print(index); // Se establece la posición del cursor: columna doce, segunda fila lcd.setCursor(11, 1); // Se imprime el valor del elemento para el índice actual lcd.print(myArray[index]); // Se hace una pausa de tiempo delay(timer); } }</pre> |

En este *sketch* se ha introducido el operador *sizeof*, el cual toma un argumento ya sea un tipo de dato o el nombre de una variable (entera, carácter, arreglo, etcétera), y regresa el tamaño en *bytes* del argumento.



La sintaxis del operador es:

*sizeof*(nombre\_variable)  
*sizeof*(tipo\_variable)  
*sizeof*(expresión)

A continuación se describen algunos ejemplos del operador *sizeof*.

```
lcd.print (sizeof (myArray));
```

En el ejemplo se imprime en la pantalla *LCD* el tamaño en bytes del arreglo *myArray*.

Al ser un arreglo de cinco números enteros (datos de tipo *int*), y considerando que el tipo de datos *int* ocupa 2 *bytes* en memoria, entonces la instrucción anterior desplegaría en pantalla el número (figura 10).



Figura 10. Pantalla *LCD* desplegando el tamaño del arreglo *myArray*

En el ejemplo se imprime en la pantalla *LCD* el tamaño en *bytes* del tipo de datos *int*. Es decir, se imprimiría el valor 2 (figura 11).



Figura 11. Pantalla *LCD* desplegando el tamaño en *bytes* del tipo de datos *int*

```
lcd.print (sizeof (myArray)) / sizeof (int));
```

En el ejemplo se imprime en la pantalla *LCD* la razón entre el tamaño en *bytes* del arreglo *myArray* y el tamaño en *bytes* del tipo de datos *int*. Es decir, se imprimiría el valor 5 (figura 12).

Es importante que el lector observe que esta razón corresponde al tamaño del arreglo.



Figura 12. Pantalla *LCD* desplegando el tamaño del arreglo *myArray*



## Resultados y conclusiones

En esta práctica se aprendió el manejo de arreglos unidimensionales, en especial se tuvo un acercamiento a los arreglos unidimensionales de tipo entero y de tipo carácter.

Se aprendió que un arreglo unidimensional es una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de un índice, cuyo valor es un número ordinal que tiene como límite inferior **0**, y como límite superior el tamaño del arreglo menos 1.

También se aprendió que una forma eficiente de recorrer a todos los elementos de un arreglo es mediante el uso del ciclo *for*. Y que el uso del operador *sizeof* nos permite determinar el número de elementos de un arreglo, además del tamaño en *bytes* de los diferentes tipos de datos.

## Referencias

LiquidCrystal - “Hello World!”. (s.f.). *Arduino LLC*. Recuperado el 21 de noviembre de 2014, de <http://www.arduino.cc/en/Tutorial/LiquidCrystal/>.

Arrays. (s.f.). *Arduino LLC*. Recuperado el 21 de noviembre de 2014, de <http://arduino.cc/en/reference/array>.



## Actividad 23. Comunicación serial

Víctor Hugo Leyva García

### Antecedentes

Una característica importante de los microcontroladores es su capacidad de comunicarse con otros dispositivos (computadoras, sensores y otros microcontroladores).

El microcontrolador Arduino UNO posee un puerto serie para comunicaciones denominado *UART* (*Universal Asynchronous Receiver/Transmitter*, por sus siglas del inglés), el cual permite la comunicación con la computadora a través del puerto *USB*.

El envío y recepción de datos entre la computadora y el Arduino se realiza *bit a bit*.

### Introducción

En la práctica 3. *LED* intermitente y resistencia hicimos uso de las salidas digitales del microcontrolador Arduino UNO, con ellas controlamos el encendido y apagado de un *LED*.

En esta práctica continuaremos con el manejo de las salidas digitales para controlar el encendido y apagado de un *LED* mediante el teclado de la computadora.

El lector se ha de preguntar sobre la manera en que lograremos la comunicación entre el Arduino y la computadora. Por el momento basta comentar que la comunicación entre la computadora y el microcontrolador se realizará mediante los puertos seriales de ambos dispositivos, es decir, a través de sus respectivos puertos *USB* (del inglés, *Universal Serial Bus*), y que en el código del *sketch* se debe habilitar el canal de comunicación *serial* y, asimismo, establecer la velocidad en que la comunicación se realizará (velocidad de transferencia de datos).

Vamos a reutilizar el circuito empleado en la práctica anterior (conectando un diodo emisor de luz al puerto digital 12 de nuestra tarjeta Arduino). En la figura 2 se muestra el esquema de conexión de componentes que utilizaremos. En dicho esquema se indica el *monitor serial*, una herramienta del entorno de programación de Arduino que permite *visualizar* el flujo de datos entre la computadora y el microcontrolador; esta misma herramienta permite el envío de datos desde la computadora hacia el Arduino y viceversa. El envío o transmisión de datos se indica la con la etiqueta *Tx*; mientras que la recepción de datos con la etiqueta *Rx*. El lector debe notar que en el esquema se indica que la comunicación entre los dispositivos es bidireccional, es decir, ambos dispositivos transmiten y reciben información.



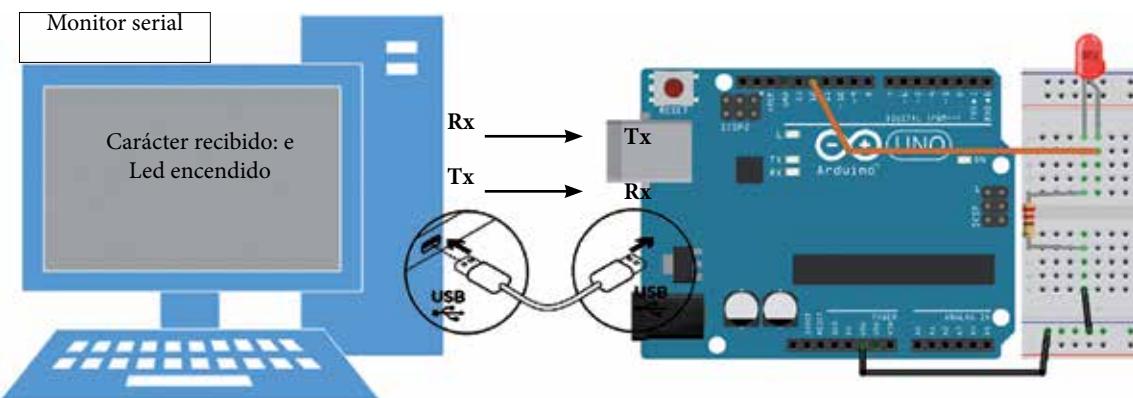


Figura 1. Esquema de conexión de componentes.

En este proyecto actual programaremos un *sketch* que permita el control del encendido y apagado del *LED* mediante el teclado; las letras introducidas en el teclado serán transmitidas (*Tx*) por la computadora y recibidas (*Rx*) por el microcontrolador, el cual apagará o encenderá el *LED* en función del carácter recibido. Para ello haremos uso de las funciones de comunicación *serial* incluidas en la biblioteca de funciones de Arduino.

#### Funciones de comunicación serial

Se utiliza para establecer la comunicación entre el microcontrolador Arduino y otro dispositivo, como computadoras, sensores u otros microcontroladores.

El microcontrolador tiene dos vías de comunicación *serial*: a través del puerto *USB*; y a través de los pines digitales 0 (*Rx*) y 1 (*TX*). En esta práctica utilizaremos el puerto *USB*.

Cuando se habilita la comunicación *serial*, entonces las terminales 0 y 1 quedan reservados para dicho objetivo, por lo que no pueden utilizarse como entradas o salidas digitales.

A continuación se describen algunas funciones de comunicación *serial* integradas en la biblioteca de funciones de Arduino.

#### Función *Serial.begin()*

La función *Serial.begin()* habilita la comunicación *serial* a través del puerto *USB* o mediante los puertos digitales 0 (*RX*) y 1 (*TX*) del Arduino. La forma general de la función es:

*Serial.begin(velocidad\_de\_transmisión)*

donde *velocidad\_de\_transmisión* es un número entero que establece la velocidad de transmisión de datos en bits por segundo (baudios). Suelen utilizarse 9600 baudios para comunicarse con la computadora, aunque también es posible utilizar otras velocidades de transferencia de datos.

A continuación se incluye un ejemplo de uso de la función *Serial.begin()*:



*Serial.begin(9600)*

Cuando la comunicación *serial* se ha establecido, entonces las terminales 0 y 1 no pueden utilizarse como entradas o salidas digitales.

*Función Serial.end()*

Esta función desactiva la comunicación *serial*. La forma general de la función es:

*Serial.end()*

y no lleva ningún parámetro, o valor, entre los paréntesis.

Al desactivarse la comunicación *serial*, entonces las terminales 0 (RX) y 1 (TX) pueden ser utilizados como entradas o salidas digitales.

*Función Serial.read()*

Esta función lee un carácter almacenado en el *buffer* de comunicación *serial*. La forma general de la función es:

*Serial.end()*

y no lleva ningún parámetro, o valor, entre los paréntesis.

*Función Serial.print()*

La función *Serial.print()* imprime en el puerto serie (en el monitor *serial*, por defecto) una cadenas de caracteres, un número o una variable, según se indique. La forma general de esta función es:

*Serial.print(cadena\_de\_control[, formato])*

La función *Serial.print()* imprime el contenido en la *cadena\_de\_control*, de acuerdo con el formato definido; el formato es un argumento opcional que aplica sólo a cadenas de control numéricas, por ello se ha puesto entre corchetes ([, formato]). En caso de no indicar el formato, los datos numéricos se imprimen en texto legible *ASCII*, por defecto.

La *cadena\_de\_control* puede ser una cadena de caracteres, una variable, un número o una operación matemática.

El *formato* determina la forma en que se imprimirá la *cadena\_de\_control*, cuando ésta es un valor numérico, puede representarse como un número decimal (DEC), octal (OCT) o hexadecimal (HEX).



Considere el ejemplo siguiente:

```
int i=12;  
Serial.print("\nEl valor de la variable i es: ");  
Serial.print(i);  
Serial.print("\nEl doble de la variable i es: ");  
Serial.print(2*i);  
Serial.print("\nEl valor de i expresado en hexadecimal es: ");  
Serial.print(i, HEX);
```

En la primera línea de código se declara una variable entera, la cual se nombra “*i*” y se le asigna el valor “12”. En las siguientes líneas se imprimen seis cadenas de control. En la segunda, cuarta y sexta líneas se imprimen *cadenas de caracteres*, las cuales están encerradas entre comillas dobles (“ ”); en la tercera se imprime el valor numérico de la variable “*i*”; en la quinta línea se imprime el resultado de multiplicar el número 2 con el valor numérico de la variable entera *i*; y en la última línea se imprime el valor numérico de la variable *i* expresado en hexadecimal.

De esta manera, al ejecutar dicho código obtendríamos:

*El valor de la variable i es: 12  
El doble de la variable i es: 24  
El valor de i expresado en hexadecimal es: c*

Las funciones *print()* imprimen las cadenas de control, una detrás de otra; no agrega un salto de línea automáticamente entre ellas. Para agregar el salto de línea se escribió \n, también conocido como “carácter salto de línea”.

### Función Serial.println()

La función *Serial.println()* imprime en el puerto serie (en el monitor *serial*, por defecto) una cadenas de caracteres, un número o una variable, según se indique, seguido de un salto de línea (\n). La forma general de esta función es:

*Serial.println(cadena\_de\_control[, formato])*

Esta función tiene los mismos argumentos que *Serial.print()*, es decir, requiere del argumento *cadena\_de\_control* y de un segundo argumento opcional (*formato*).



## Objetivos

El alumno:

- Aprenderá a programar al microcontrolador Arduino para establecer un canal de comunicación *serial* con la computadora.
- Controlará el encendido y apagado de un *LED* mediante el teclado de la computadora: la letra “a” apagará al *LED* y la letra “e” lo encenderá.

## Material utilizado

- Equipo de cómputo.
- Cable estándar *USB* (conexión A a conexión B).
- Microcontrolador Arduino UNO.
- *LED* de cualquier color.
- Resistencia de  $220\ \Omega$ .

## Desarrollo

Como se ilustra en la figura 1, se conecta el ánodo del *LED* a la terminal 12 del Arduino. Pudo haberse seleccionado cualquier otra salida digital; pero, cuando se establece un canal de comunicación *serial* entonces las terminales 0 (*Rx*) y 1 (*Tx*) quedan reservados para tal fin, por lo que no pueden utilizarse como salidas o entradas digitales.

El cátodo debe conectarse al voltaje de referencia (*GND*) a través de una resistencia de  $220\ \Omega$ , la cual tiene el propósito de limitar la corriente que atravesará el *LED* y así evitar posibles daños en éste.

En el *sketch* se configurará al *pin* 12 como un puerto de salida digital, el cual proveerá voltajes de 5 o 0 volts –también conocidos como niveles *HIGH* (ALTO) y *LOW* (BAJO), respectivamente –durante el tiempo especificado en el *sketch* a través de la función *delay()*. En consecuencia, el ánodo del *LED* estará sometido a 0 volts y a 5 volts proporcionados por el *pin* 12; por lo tanto, cuando en el *pin* 12 se tenga un nivel ALTO, entonces el *LED* encenderá; y cuando en dicho *pin* se tenga un nivel BAJO, entonces el *LED* se apagará.

Conecta tu tarjeta Arduino a la computadora mediante el cable *USB*.

Ejecuta el entorno de programación de Arduino; abre el *sketch* de ejemplo “Comunicación *serial* para el control de un *LED*” y cárgalo en tu Arduino. El código se explica en las dos siguientes secciones.

El *sketch* “Comunicación *serial* para el control de un *LED*” lo encontrarás en el *CD* que acompaña a estas prácticas.

En la parte superior de la ventana del entorno de programación de Arduino encontrarás el ícono . Al hacer *clic* en dicho ícono se abre el *monitor serial* (figura 2).

En la ventana del monitor serial se introducen los caracteres de control para el encendido y apagado de un *LED*. El carácter “e” encenderá el *LED*, mientras que el carácter “a” lo apagará; cualquier otro carácter será ignorado por el programa.



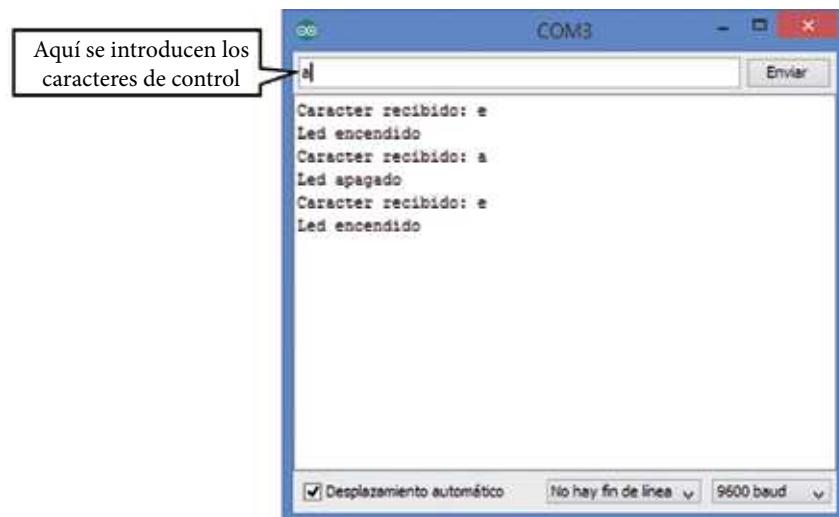
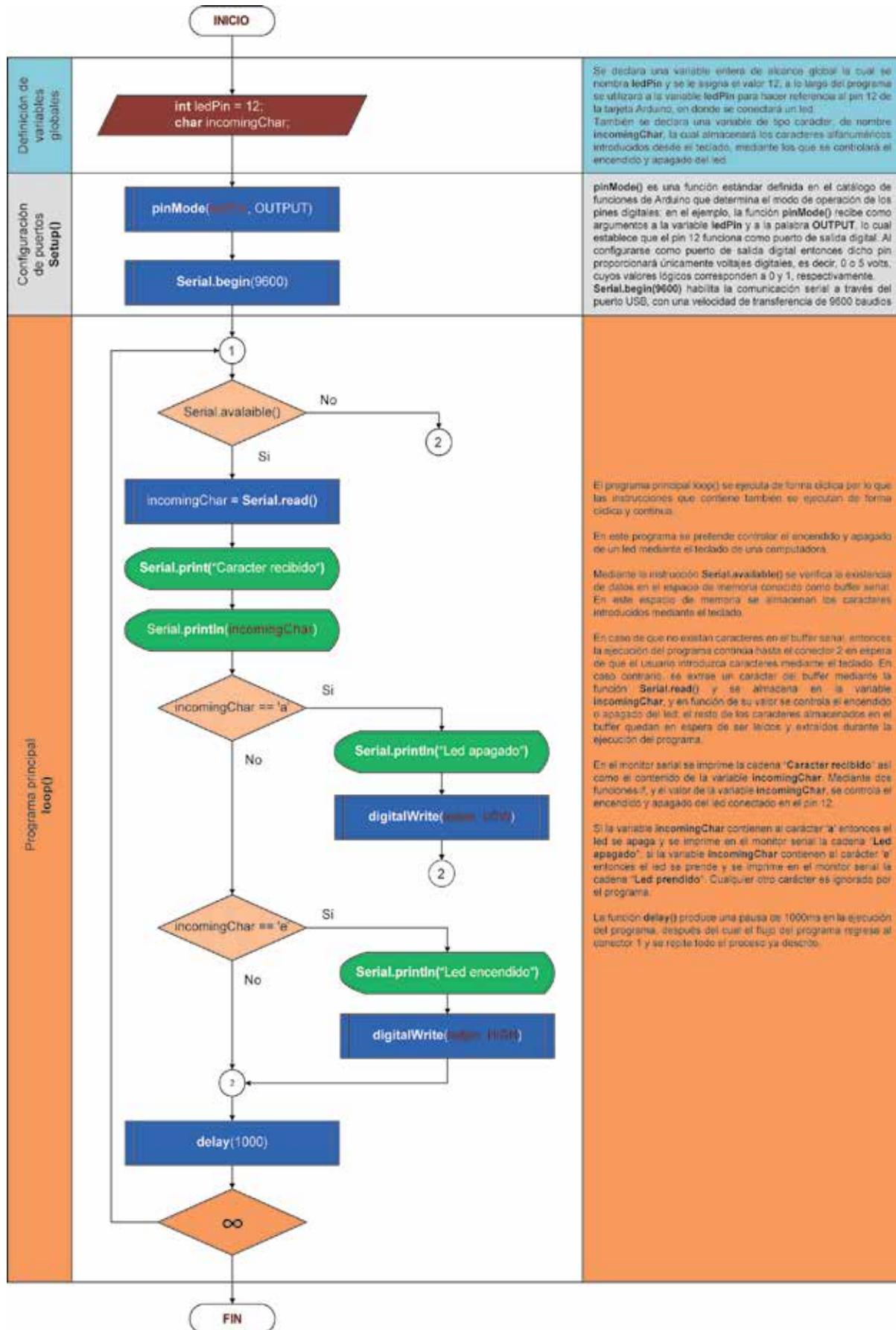


Figura 2. Monitor serial.



## Diagrama de flujo



## Código

|                                      |  |
|--------------------------------------|--|
| Declaración de variables globales    | <pre>char incomingChar; // Carácter de entrada int ledPin = 12; // Se conectará el led al pin 12 del Arduino</pre>   |
| Área de configuración setup()        | <pre>void setup() {     // Configura ledPin (pin 12) como pin de salida digital     pinMode(ledPin, OUTPUT);     // Habilita el puerto serial, velocidad de transferencia de 9600 bps     Serial.begin(9600); }</pre>  |
| Cuerpo principal del programa loop() | <pre>void loop() {     // Verifica en la existencia de datos en el buffer serial.     if (Serial.available() &gt; 0) {         // Lee los caracteres almacenados en el buffer serial.         incomingChar = Serial.read();         // Imprime texto en el monitor serial         Serial.print("Carácter recibido: ");         // Imprime el valor de la cadena incomingChar en el monitor serial'         Serial.println(incomingChar);</pre>   |
| Subprogramas                         | <pre>// Bloque de código que se ejecuta al introducir el carácter 'a' if (incomingChar == 'a') {     // Imprime estado del led en el monitor serial     Serial.println("Led apagado");      // Apaga el led     digitalWrite(ledPin, LOW); }  // Bloque de código que se ejecuta al introducir el carácter 'e' else if(incomingChar == 'e') {     // Imprime estado del led en el monitor serial     Serial.println("Led encendido");     // Enciende el led     digitalWrite(ledPin, HIGH); }  // No se ejecuta código al introducir un carácter diferente de 'a' o 'e' else ; delay(1000); }</pre> |



## Resultados y conclusiones

En esta práctica se introdujo el concepto de comunicación serial para controlar el encendido y apagado de un *LED* mediante el teclado de la computadora. Se hizo uso del “*monitor serial*” para el envío de datos desde la computadora hacia el Arduino, así como de las funciones de comunicación serial incluidas en la biblioteca de funciones del microcontrolador.

## Referencias

Serial. (s.f.). Arduino LLC. Recuperado el 30 de Agosto de 2014, de  
<http://arduino.cc/en/pmwiki.php?n=Reference/Serial>



## Actividad 24. La estructura selectiva *switch*

Víctor Hugo Leyva García

### Antecedentes

En el diseño de circuitos electrónicos son frecuentes las situaciones en las que debemos tomar decisiones. Para dar solución a este tipo de situaciones se emplean diferentes estructuras lógicas en la programación de Arduino, como las estructuras *if*, *if-else* y *switch*. La toma de decisión se basa en la evaluación de una o más condiciones que determinan los procedimientos a seguir.

En prácticas anteriores estudiamos las estructuras selectivas *if* y *if-else*, en esta práctica abordaremos la estructura selectiva *switch*.

### Introducción

#### *Sentencia de control switch*

La sentencia *switch* se utiliza para seleccionar una de entre múltiples alternativas. Tal elección depende del valor de una variable o de una expresión simple denominadas expresión de control o selector. El selector puede tomar valores de tipo ordinal (por ejemplo, *int* o *char*, pero no *float* o *string*), de un conjunto previamente establecido.

Considérese un ejemplo en el que el selector puede tomar valores enteros entre 1 y N. Así, se ejecutará la acción 1, cuando el selector tome el valor de 1; si el selector toma el valor 2, se ejecutará la acción 2; y si toma el valor N, se ejecutará la acción N. Si el selector toma un valor distinto de los comprendidos entre 1 y N, entonces se ejecuta una acción por defecto (*default*); esta instrucción por defecto puede omitirse. En la figura 1 se ilustra esta estructura selectiva.

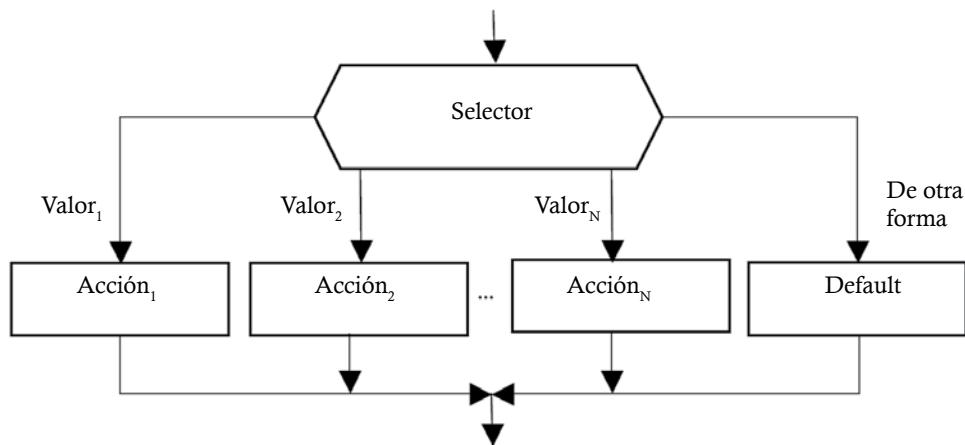


Figura 1. Estructura selectiva múltiple *switch*



La estructura de control *switch* tiene la siguiente sintaxis:

```
Switch (selector)
{
    case etiqueta1:
        sentencias1;
        break;
    case etiqueta2:
        sentencias2;
        break;
    default:
        sentencias;
}
```

El flujo del código de la estructura *switch* depende del valor de la expresión de control o selector. Para ello, se evalúa la expresión de control o selector y el valor se compara con cada una de las etiquetas de *case*. Cada etiqueta debe tener un valor único y diferente. Si el valor de la expresión selector es igual a una de las etiquetas de *case*, entonces se ejecutan las sentencias que están a continuación de dicha etiqueta y antes de una expresión *break*. El objetivo de la expresión *break* es concluir la ejecución de la sentencia *switch*. Si el selector no coincide con ninguna etiqueta, entonces se ejecutan las sentencias que están a continuación de *default*.

En el siguiente ejemplo se desea controlar el encendido y apagado de dos *LED* (*pinLed1* y *pinLed2*) mediante el teclado de la computadora. En caso de introducir el carácter “a” se encenderá el primer *LED* (*pinLed1*); con el carácter “b” se encenderá el segundo *LED* (*pinLed2*); el carácter “c” prenderá ambos ledes; mientras que el cualquier otro carácter apagará a los dos ledes. Esta última situación corresponde a las instrucciones por defecto (*default*). El código para resolver dicha situación se indica a continuación:

```
Switch (opción)
{
    case a:
        //Enciende el LED 1 y apaga el LED 2;
        digitalWrite(ledPin 1, HIGH);
        digitalWrite(ledPin 2, LOW);
        break;
    case b:
        //Apaga el LED 1 y enciende el LED 2;
        digitalWrite(ledPin 1,LOW);
        digitalWrite(ledPin 2, HIGH);
        break;
    case c:
```



```
//Enciende ambos leds;  
digitalWrite(ledPin 1, HIGH);  
digitalWrite(ledPin 2, HIGH);  
break;  
default:  
    //Apaga ambos leds;  
    digitalWrite(ledPin 1, LOW);  
    digitalWrite(ledPin 2, LOW);  
}
```

## Objetivos

El alumno:

- Aprenderá el uso de la función de selección múltiple *switch*.
- Establecerá un canal de comunicación serial entre el microcontrolador Arduino y la computadora.
- Controlará el encendido y apagado de dos ledes mediante el teclado de la computadora: el carácter “a” encenderá el primer *LED* (*pinLed1*); con el carácter “b” se encenderá el segundo *LED* (*pinLed2*); el carácter “c” prenderá ambos ledes; mientras que el cualquier otro carácter apagará a los dos ledes.

## Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 2 ledes de cualquier color.
- 2 resistencias de  $220\ \Omega$ .

## Desarrollo

Tal como se ilustra en la figura 2, se conecta el ánodo del primer *LED* a la terminal 11 del Arduino (*ledPin1*), y el ánodo del segundo *LED* a la terminal 12 (*ledPin2*). Dichos *pin* serán ocupados como salidas digitales.

Pudieron ocuparse otros *pin* digitales diferentes a los considerados en esta práctica, a excepción de los *pin* 0 (*Rx*) y 1 (*Tx*), ya que éstos están reservados para la comunicación *serial*, por lo que no pueden utilizarse como salidas o entradas digitales.

Para limitar la corriente que circulará a través de los ledes, sus cátodos deben conectarse al voltaje de referencia (*GND*) a través de una resistencia de  $220\ \Omega$ .



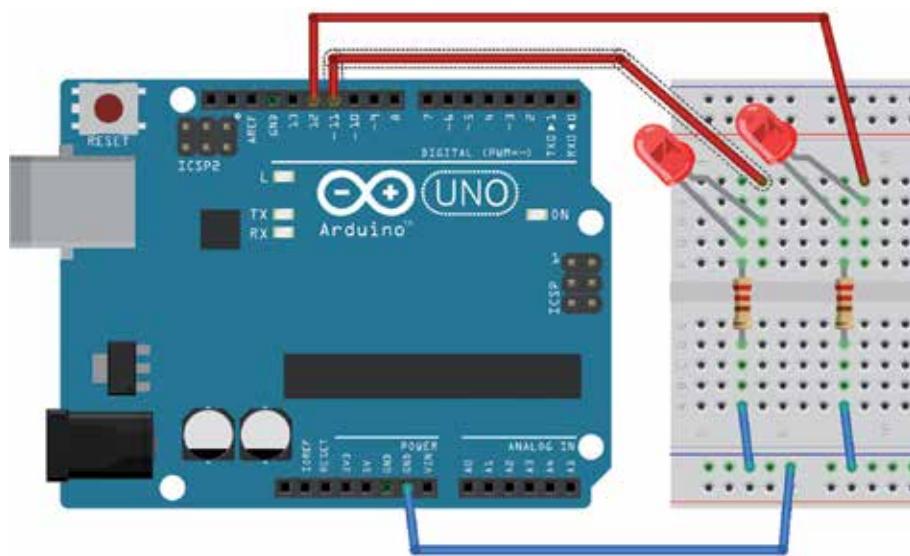


Figura 2. Esquema de conexión de componentes

Conecta tu tarjeta Arduino a la computadora mediante el cable *USB*.

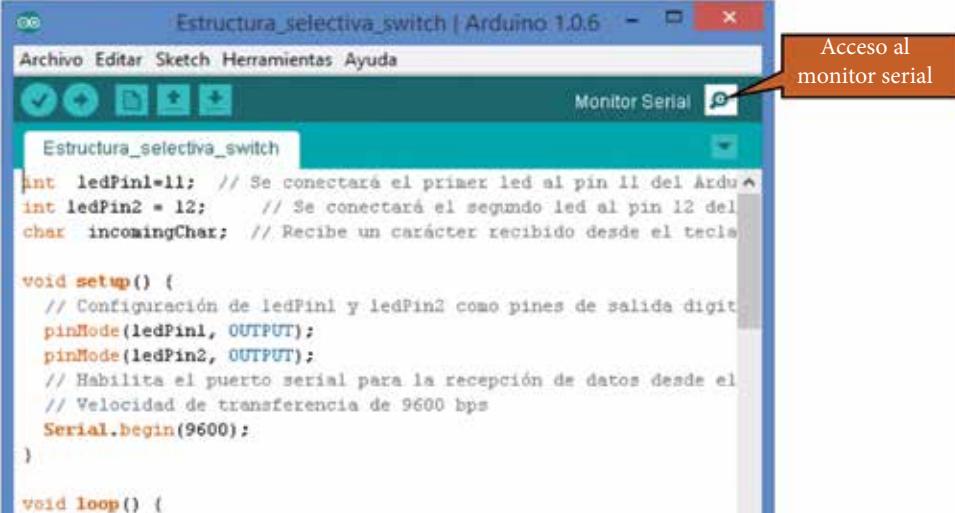
Ejecuta el entorno de programación de Arduino; abre el *sketch* de ejemplo “Estructura selectiva *switch*” y cárgalo en tu Arduino. El código se explica en las dos siguientes secciones.

El *sketch* “Estructura selectiva *switch*” se encuentra en el *CD* que acompaña a estas prácticas.

En la parte superior de la ventana del entorno de programación de Arduino encontrarás el ícono del *monitor serial*  (figura 3). Al hacer clic en dicho ícono se abre el *monitor serial* (figura 4).

En la ventana del *monitor serial* se introducen los caracteres de control para el encendido y apagado de los ledes. El carácter “a” encenderá al *LED* conectado en *ledPin1* y apagará al *LED* conectado en *ledPin2*; el carácter “b” apagará al led *ledPin1* y encenderá al led *ledPin2*; el carácter “c” encenderá a ambos ledes y cualquier otro carácter los apagará.





```
Estructura_selectiva_switch | Arduino 1.0.6
Archivo Editar Sketch Herramientas Ayuda
Monitor Serial

Estructura_selectiva_switch
int ledPin1=11; // Se conectará el primer led al pin 11 del Arduino
int ledPin2 = 12; // Se conectará el segundo led al pin 12 del Arduino
char incomingChar; // Recibe un carácter recibido desde el puerto serial

void setup() {
    // Configuración de ledPin1 y ledPin2 como pines de salida digitales
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    // Habilita el puerto serial para la recepción de datos desde el teclado
    // Velocidad de transferencia de 9600 bps
    Serial.begin(9600);
}

void loop() {
```

Figura 3. Acceso al *monitor serial*.

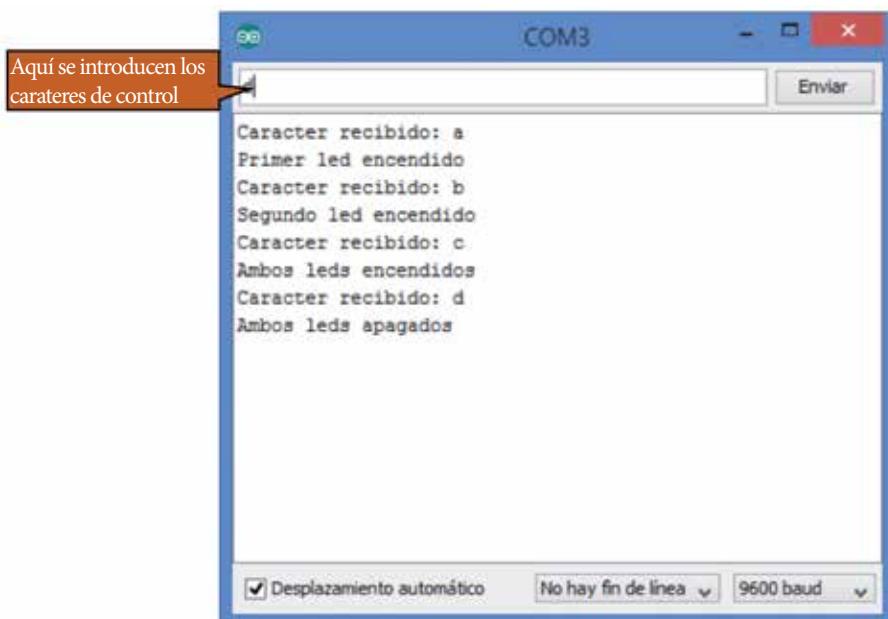
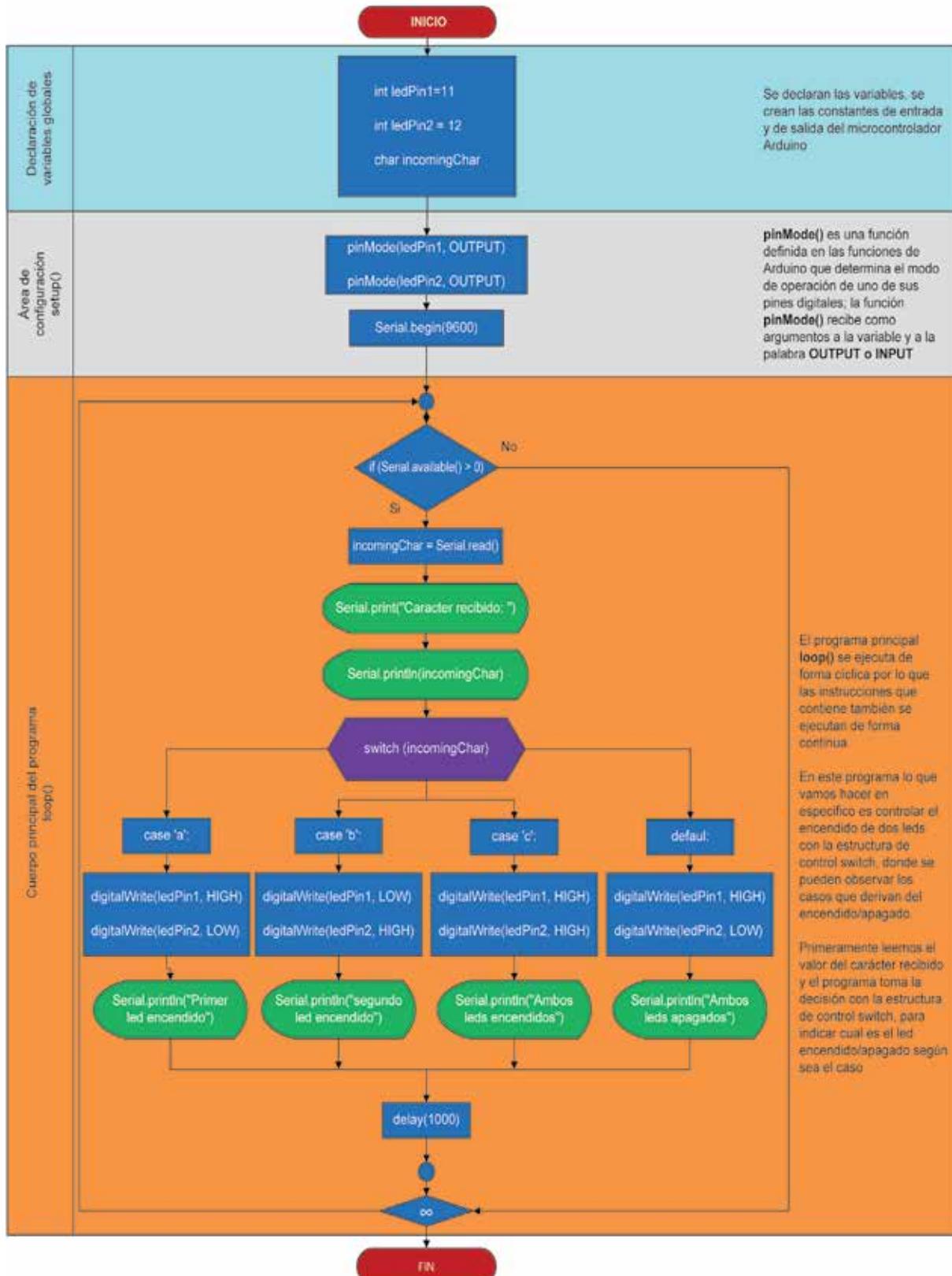


Figura 4. *Monitor serial*



# Diagrama de flujo

## La estructura selectiva switch



## Código

|                                   |  |
|-----------------------------------|--|
| Declaración de variables globales | <pre>int ledPin1=11; // Se conectará el primer led al pin 11 del Arduino int ledPin2 = 12; // Se conectará el segundo led al pin 12 del Arduino char incomingChar; // Recibe un carácter recibido desde el teclado</pre>   |
| setup()                           | <pre>void setup() {     // Configuración de ledPin1 y ledPin2 como pines de salida digital     pinMode(ledPin1, OUTPUT);     pinMode(ledPin2, OUTPUT);     // Habilita el puerto serial para la recepción de datos desde el teclado     // Velocidad de transferencia de 9600 bps     Serial.begin(9600); }  void loop() {</pre>   |
| loop()                            | <pre>// Verifica en la existencia de datos en el buffer serial. if (Serial.available() &gt; 0) {     // Lee los caracteres almacenados en el buffer serial.     incomingChar = Serial.read();     // Imprime texto en el monitor serial     Serial.print("Carácter recibido: ");     // Imprime el valor de la cadena incomingChar en el monitor serial'     Serial.println(incomingChar);     // Bloque de código que se ejecuta al introducir el carácter 'a'     switch (incomingChar)     {         case 'a':             // Prende ledPin1             digitalWrite(ledPin1, HIGH);             // Apaga ledPin2             digitalWrite(ledPin2, LOW);             // Imprime texto en el monitor serial             Serial.println("Primer led encendido");             break;         case 'b':             // Apaga ledPin1             digitalWrite(ledPin1, LOW);             // Prende ledPin2             digitalWrite(ledPin2, HIGH);             Serial.println("Segundo led encendido");             break;     } }</pre> |



```
loop()
{
    case 'C':
        // Prende ledPin1
        digitalWrite(ledPin1, HIGH);
        // Prende ledPin2
        digitalWrite(ledPin2, HIGH);
        Serial.println("Ambos leds encendidos");
        break;
    default:
        // Apaga ledPin1
        digitalWrite(ledPin1, LOW);
        // Apaga ledPin2
        digitalWrite(ledPin2, LOW);
        Serial.println("Ambos leds apagados");
    }
    // Pausa de un segundo (1000 ms)
    delay(1000);
}
```

## Resultados y conclusiones

En esta práctica se hizo uso de las funciones de comunicación serial del Arduino para controlar el encendido y apagado de un *LED* mediante el teclado de la computadora.

También se hizo uso de la estructura selectiva *switch* para seleccionar una de entre múltiples alternativas.

## Referencias

- switch / case statements (s.f.). Arduino LLC. Recuperado el 18 de noviembre de 2014, de <http://arduino.cc/en/pmwiki.php?n=Reference/SwitchCase>
- Serial (s.f.). Arduino LLC. Recuperado el 18 de noviembre de 2014, de <http://arduino.cc/en/Reference/Serial>
- Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 24-25). Italia: Arduino LLC.



## Actividad 25. Secuencia S.O.S. en código Morse

Víctor Hugo Leyva García

### Antecedentes

#### *El código Morse y la señal S.O.S.*

En la actualidad el correo electrónico es una herramienta de comunicación telemática rápida y barata. Pero durante el siglo XIX el método de comunicación más popular se basaba en el código Morse, el cual es un sistema de transmisión y recepción de mensajes que emplea impulsos acústicos o luminosos, y cuyo alfabeto está determinado por la duración e intermitencia de dichos impulsos. En este sentido, la codificación del mensaje consiste de “puntos” y “rayas”.



Figura 1. Caricatura. Mensaje en código Morse (Cyanide & Happiness, 2010)

En la tabla 1 se indica el alfabeto Morse; observa que cada letra del alfabeto tiene un código único compuesto de rayas y de puntos.

| Letra | Código | Letra | Código | Letra | Código |
|-------|--------|-------|--------|-------|--------|
| A     | .-     | J     | .---   | R     | .-.    |
| B     | -....  | K     | -.-    | S     | ...    |
| C     | -.-.   | L     | .-..   | T     | -      |
| D     | -..    | M     | --     | U     | ..-    |
| E     | .      | N     | -.     | V     | ...-   |
| F     | .-.-   | Ñ     | --,-   | W     | .--    |
| G     | --.    | O     | ---    | X     | -..-   |
| H     | ....   | P     | .--.   | Y     | -,-    |
| I     | ..     | Q     | --,-   | Z     | --..   |

Tabla 1. Alfabeto en código Morse (Pinilla Morán, 2011)



La raya se interpreta acústicamente con un sonido largo y visualmente con un destello luminoso largo; y el punto se interpreta con un sonido breve o con un destello también corto. Por convención, los puntos son de corta duración (una [1] unidad de tiempo) mientras que la duración de las rayas es 3 veces la duración del punto (3 unidades de tiempo); entre los puntos y rayas de una misma letra debe haber una pausa de una unidad de tiempo; entre letra y letra debe haber una pausa de 3 unidades de tiempo y entre palabras debe haber una pausa de 7 unidades de tiempo.

S.O.S. (salvar nuestras almas, del inglés) es una señal de socorro reconocida internacionalmente y aprobada durante una conferencia internacional en Berlín en 1906 para ser utilizada en las transmisiones telegráficas.

En este proyecto haremos que un *LED* parpadee de forma continua una secuencia S.O.S. codificada en Morse. En la figura 2 se muestra la codificación S.O.S.

... - ...  
--  
S O S

Figura 2. Señal S.O.S. codificada en Morse

## Introducción

En la práctica anterior aprendimos el manejo de las salidas digitales del microcontrolador Arduino UNO y con ello controlamos el encendido y apagado de un *LED*. En esta práctica continuaremos con el manejo de las salidas digitales para generar una señal S.O.S. codificada en Morse; para ello vamos a reutilizar el circuito empleado en la práctica anterior: conectaremos un diodo emisor de luz a alguna de las 14 salidas digitales de nuestra tarjeta Arduino.

En la figura 3 se muestra el esquema de conexión de componentes que utilizaremos

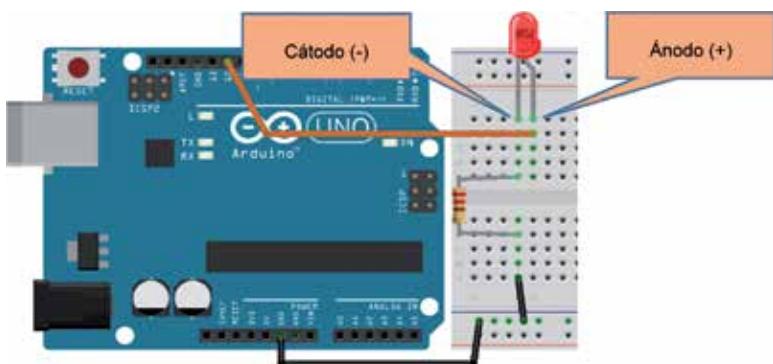


Figura 3. Esquema de conexión de componentes para generar una señal S.O.S. codificada en Morse

El sketch que utilizaremos incluye todas las funciones e instrucciones básicas abordadas en la práctica anterior, en donde aprendiste que un *sketch* de Arduino está constituido de dos funciones fundamentales: *setup()* y *loop()*. En la primera se inicializan variables, modos de operación de los pines y otros parámetros de configuración. *loop()* es la función principal y se ejecuta cíclicamente; está compuesta de las instrucciones y subprogramas que el microcontrolador ejecuta.



Ahora aprenderemos a programar mediante el diseño de subprogramas o subrutinas, lo cual nos permitirá desarrollar un código más claro y eficiente.

### Subprogramas o subrutinas

Un subprograma es un programa contenido dentro de otro. Los subprogramas pueden realizar tareas específicas— como imprimir un valor en pantalla, prender un *LED* o mover un motor—, incluso, pueden devolver uno o más valores (enteros, decimales, caracteres, etcétera) para ser reutilizados por el programa principal u otros subprogramas. Asimismo, los subprogramas pueden o no recibir valores de entrada (parámetros) procedentes del programa que los llama, y el subprograma los utiliza en cálculos intermedios o para la toma de decisiones.

La figura 4 muestra el bloque que representa a un subprograma en un diagrama de flujo.



Figura 4. Subprograma o subrutina

### Objetivos

El alumno:

- Hará uso de alguno de los 14 terminales digitales del Arduino para controlar el encendido y apagado de un *LED* para generar una señal *S.O.S.* codificada en Morse.
- Controlará el tiempo de encendido y apagado del *LED* mediante un subprograma o subrutina.

### Material utilizado

- 1 equipo de cómputo.
- 1 cable estándar *USB* (conexión A a conexión B).
- 1 microcontrolador Arduino UNO.
- 1 *LED* de cualquier color.
- 1 resistencia de  $220\ \Omega$ .

### Desarrollo

Como se ilustra en la figura 3, se conecta el ánodo del *LED* a la terminal 12 del Arduino (aunque pudo haberse seleccionado cualquier otra salida digital). El cátodo debe conectarse al voltaje de referencia (*GND*) a través de una resistencia de  $220\ \Omega$ , la cual tiene el propósito de limitar la corriente que atravesará el *LED* para evitar posibles daños en éste. Ahora bien, debes recordar que, a diferencia de los primeros trece *pin* digitales (*pin* 0 a 12), el puerto ya tiene configurada una resistencia interna, por lo que un *LED* conectado en dicho puerto no requiere de la resistencia de  $220\ \Omega$  para su protección.



Mediante el *sketch* o programa, el *pin* 12 se configurará como un puerto de salida digital y proveerá, en consecuencia, una señal digital. Los puertos de salida digitales proveen un voltaje de 5 o 0 volts –también conocidos como niveles *HIGH* (ALTO) y *LOW* (BAJO), respectivamente– durante el tiempo especificado en el programa a través de la función *delay()*. Esto significa que el ánodo del *LED* estará sometido a 0 volts y a 5 volts proporcionados por el *pin* 12 de manera constante e intermitente, lo cual dará a lugar a la secuencia *S.O.S*. Por lo tanto, cuando en el *pin* 12 se tenga un nivel ALTO, entonces el *LED* encenderá; y cuando en dicho *pin* se tenga un nivel BAJO, entonces el *LED* se apagará.

La función *delay()* recibe como argumento un valor entero que corresponde al tiempo en milisegundos que permanecerá encendido o apagado el *LED*.

Para producir la secuencia *S.O.S.* en código Morse es necesario establecer la unidad de tiempo que se utilizará en la generación de los puntos y rayas. Una unidad de tiempo adecuada es de 200 milisegundos; sin embargo, es posible establecer una unidad de tiempo diferente. De tal manera que los tiempos para general la señal *S.O.S* quedarían según se indica en la tabla 2.

| Elemento  | Duración en unidades de tiempo | Duración en milisegundos (ms) |
|---|--------------------------------|-------------------------------|
| Punto   | 1                              | 200 ms                        |
| Raya  | 3                              | 600 ms                        |
| Separación entre puntos y rayas de una sola letra | 1                              | 200 ms                        |
| Separación entre letras                           | 3                              | 600 ms                        |
| Separación entre palabras                         | 7                              | 1400 ms                       |

Tabla 2. Definición de unidades de tiempo para la generación del código Morse

En la tabla 3 se indica la codificación Morse de la señal *S.O.S.* y se indican las escalas de tiempo que se utilizarán en el *sketch*.

| Código Morse        | ...   |                    | ---   |                    | ...   |                      |
|---------------------|---|--------------------|---|--------------------|---|----------------------|
| Letra               | S   |                    | O   |                    | S   |                      |
|                     | 6   | 3                  | 12  | 3                  | 6   | 7                    |
| Unidades de tiempo: | 1 unidad por punto, una unidad entre puntos | Pausa entre letras | 3 unidades por raya, una unidad entre rayas | Pausa entre letras | 1 unidad por punto, una unidad entre puntos | Pausa entre palabras |

Tabla 3. Unidades de tiempo para la codificación de la señal *S.O.S.*



Conecta tu tarjeta Arduino a la computadora mediante el cable *USB*; realiza las configuraciones tal como se te indicó en la práctica “Primeros pasos en el manejo y comprobación de funcionamiento del microcontrolador Arduino”.

Ejecuta el entorno de programación de Arduino; abre el *sketch* de ejemplo “*SOS* en código Morse” y cárgalo en tu Arduino. El código se explica en las dos siguientes secciones.

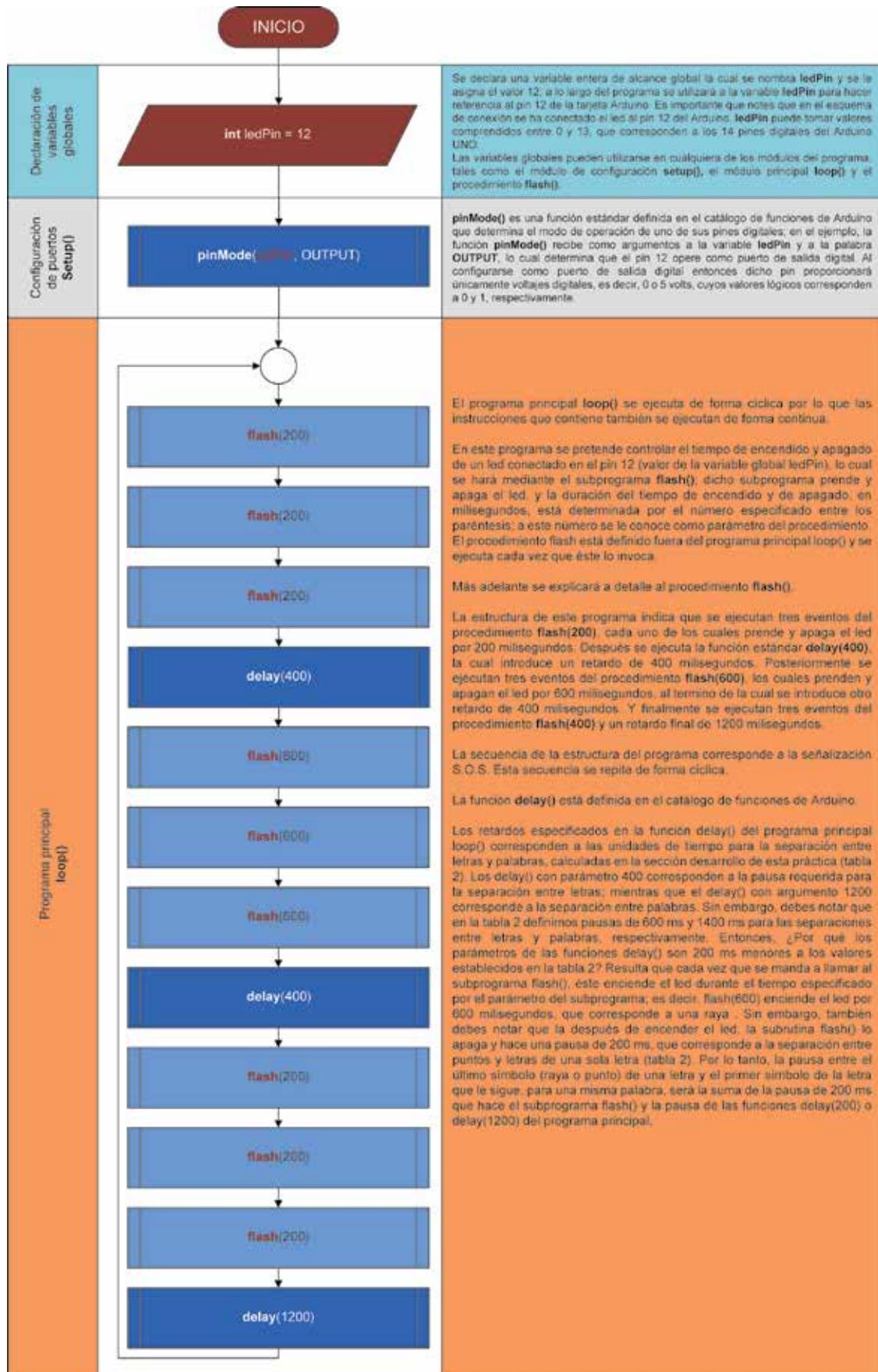
El *sketch* “*SOS* en código Morse” lo encontrarás en el *CD* que acompaña a estas prácticas.

Observarás que el *LED* prende y apaga de manera intermitente siguiendo la secuencia *S.O.S* que diseñamos en el punto 1 de esta sección, siguiendo cíclicamente la codificación que proponemos en la tabla 3.

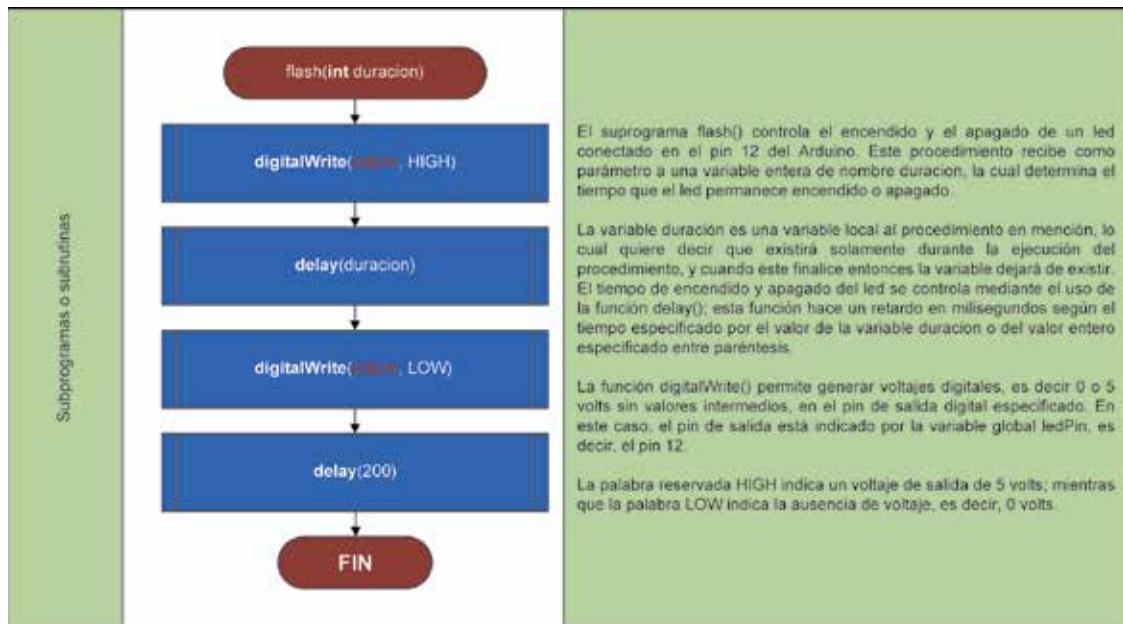
Cambia la unidad de tiempo que te proponemos, de tal manera que se modifique la velocidad de la secuencia *S.O.S*.



## Diagrama de flujo



### Subprogramas o subrutinas



## Código

|                                   |  |
|-----------------------------------|--|
| Declaración de variables globales | <pre>// Código Morse intermitente</pre>  |
|                                   | <pre>// Se conectara el led al pin 12 del Arduino.</pre>   |
| setup()                           | <pre>int ledPin = 12;</pre>  |
|                                   | <pre>// la función de configuración setup() se ejecuta una sola vez</pre>  |
| loop()                            | <pre>void setup() {     // se configura el pin 12 (ledPin) como pin de salida     pinMode(ledPin, OUTPUT); }</pre>   |
|                                   | <pre>// la función loop() se ejecuta cíclicamente y de manera ininterrumpida</pre>   |
| Subprogramas                      | <pre>void loop() {     // Letra S     flash(200); flash(200); flash(200);     // Pausa entre letras     delay(400);     // Letra O     flash(600); flash(600); flash(600);     // Pausa entre letras     delay(400);     // Letra S     flash(200); flash(200); flash(200);     // Pausa entre palabras     delay(1200); }  // Definición del subprograma flash()</pre>  |
|                                   | <pre>void flash(int duration) {     // enciende el LED (HIGH indica 5 volts de salida en el pin 12)     digitalWrite(ledPin, HIGH);     delay(duration); // Mantiene encendido el led el tiempo indicado por el parámetro duration     // se apaga el LED (LOW indica la ausencia de voltaje)     digitalWrite(ledPin, LOW);     delay(200); // Mantiene apagado el led por 200 ms (separación entre letras) }</pre> |



## Resultados y conclusiones

En esta práctica se introdujo el concepto de subprogramas y se generó una señal *S.O.S.* codificada en clave Morse, utilizando el microcontrolador Arduino y un *LED* intermitente conectado a uno de las terminales digitales del microcontrolador. La duración del destello luminoso se controló mediante el parámetro de entrada del subprograma *flash()*, el cual determinaba el tiempo de encendido del *LED* para dar lugar a las “rayas” y “puntos” de la codificación Morse.

## Referencias

Cyanide & Happiness. (2010). *Code morse* [imagen en línea]. Recuperada de <http://explosm.net/comics/1954/>

Pinilla Morán, V. D., et. al. (2011). *Conceptos Básicos de la Radioafición* (pp. 34-37). México: Facultad de Ingeniería, UNAM. Recuperado de <http://www.areunam.unam.mx/papime/Práctica01.pdf>



## Actividad 26. ZOETROPE (Máquina estroboscópica)

Jaime Martínez Santiago / Norberto Alejandro Pérez Colín

### Antecedentes

*¿Qué es una señal digital?*

Es una señal cuyo valor puede ser analizado con lógica de dos estados representados por dos niveles de tensión eléctrica, uno alto, H y otro bajo, L (de *High* y *Low*, respectivamente, en inglés). Dichos estados se sustituyen por ceros y unos, lo que facilita la aplicación de la lógica y la aritmética binaria. Si el nivel alto se representa por 1 y el bajo por 0.

Una señal digital es algo de naturaleza incremental, en tanto, una señal analógica es algo que varía de forma continua.

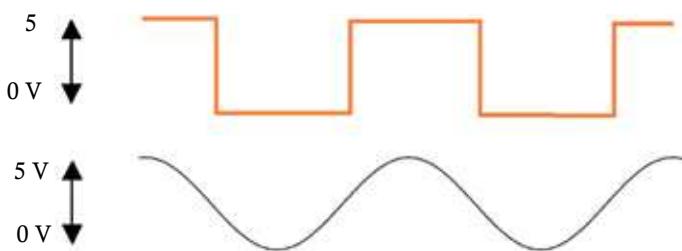
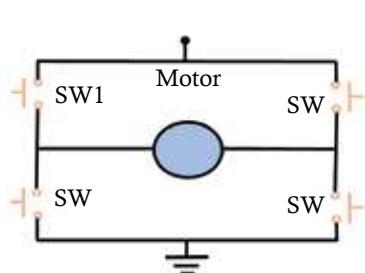


Figura 1. Señal digital (arriba) y señal analógica (abajo)

En la señal digital la amplitud varía de forma rápida, no existiendo estados o fases intermedias entre sus amplitudes como lo muestra la figura 1.

*¿Qué es un Puente H?*

Un puente H es un circuito construido para cambiar el sentido de giro de un motor eléctrico, se puede lograr cambiar su sentido de giro al cambiar básicamente la polaridad de la alimentación eléctrica, pero esa conmutación se puede hacer efectiva conectando un puente-H que permita activar el motor eléctrico, ser activados, en un sentido u otro y, al mismo tiempo, permite controlar su velocidad de los mismos. La figura 2 muestra el diagrama esquemático del puente H, estos circuitos son de uso frecuente en robótica.



| Motor  | SW1         | SW2         | SW3         | SW4         |
|--|-------------|-------------|-------------|-------------|
| Parado                                       | abierto(0)  | abierto(0)  | abierto(0)  | abierto(0)  |
| Gira a la derecha                            | abierto(0)  | cerrado (1) | cerrado (1) | abierto(0)  |
| Gira a la izquierda                          | cerrado (1) | abierto(0)  | abierto(0)  | cerrado (1) |
|  |             |             |             |             |
| El resto de los estados no están permitidos. |             |             |             |             |

Figura 2. Diagrama esquemático y su tabla de estados del puente H



El concepto “Puente-H” se deriva de la representación esquemática del circuito eléctrico, básicamente se construye con interruptores (mecánicos o de estado sólido). En la figura 3 se puede observar el diagrama esquemático del puente H, mostrando el cambio de giro del motor.

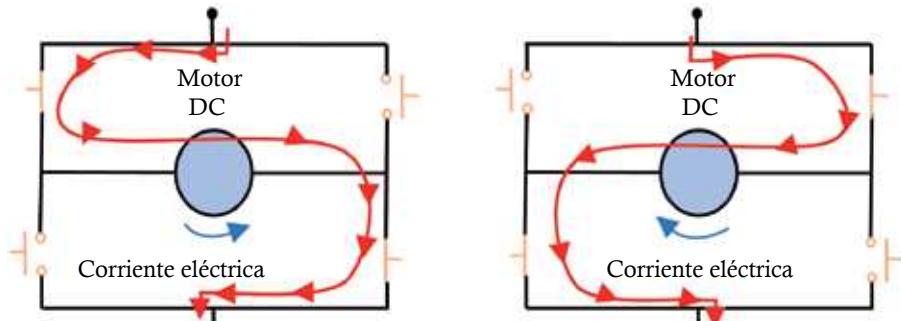


Figura 3. Diagrama esquemático del puente H, mostrando el cambio de giro del motor

## Introducción

En los inicios de la industria cinematográfica, uno de los intentos por hacer imágenes en movimiento motivó la creación de un instrumento llamado *Zoetrope*. Este invento crea la ilusión óptica de un conjunto de imágenes que se encuentran realizando un movimiento, las cuales se colocaban en el interior de un cilindro, y se observaban por las ranuras de el mismo, cuando éste se encontraba girando en cualquiera de sus sentidos, en un principio se movían manualmente o por un mecanismo rotatorio.

Para realizar esta práctica utilizaremos un interruptor que nos permita controlar la dirección del giro de un motor de corriente directa y un potenciómetro que dará el control de su velocidad, como se muestra en las figuras 4 y 5, donde se muestran los esquemas del prototipo de una máquina estroboscópica.

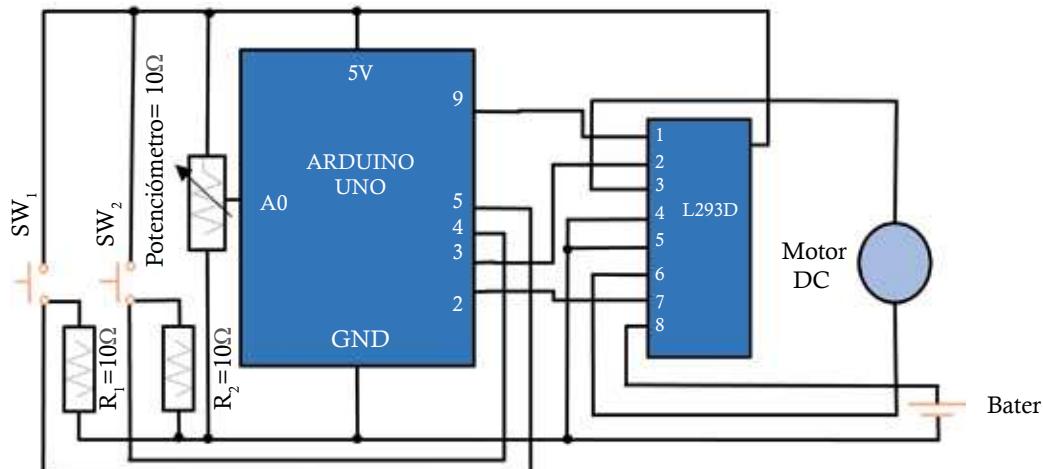


Figura 4. Esquema eléctrico del prototipo de una máquina estroboscópica



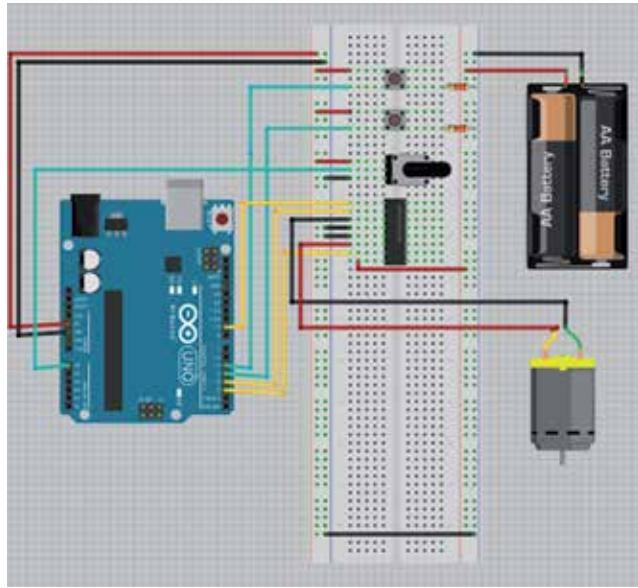


Figura 5. Prototipo de una máquina estroboscópica

El motor de corriente directa o continua, cuyos símbolo e imagen podemos observar en la figura 6, basa su funcionamiento en la repulsión de los polos magnéticos de un imán, ubicados en la parte fija del chasis, contra los polos magnéticos de un electroimán que está montado en un eje (rotor), cuando circula la corriente eléctrica por la bobina del electroimán, el campo electromagnético que se genera interactúa con el campo magnético del imán permanente; cuando los polos magnéticos coinciden se produce un giro de fuerzas que hace que el rotor rompa su inercia y gire sobre su eje en sentido de las manecillas del reloj o en sentido contrario de acuerdo con la forma en que recibe el voltaje.

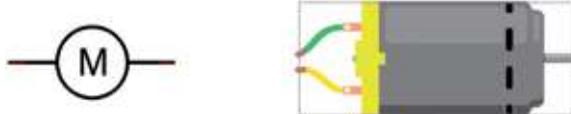


Figura 6. Símbolo e imagen del motor de corriente directa (CD). Imágenes obtenidas con el software Fritzing

El puente H que utilizaremos en nuestra práctica es el L293D un circuito integrado, con características de conexión que se muestran en la figura 7.

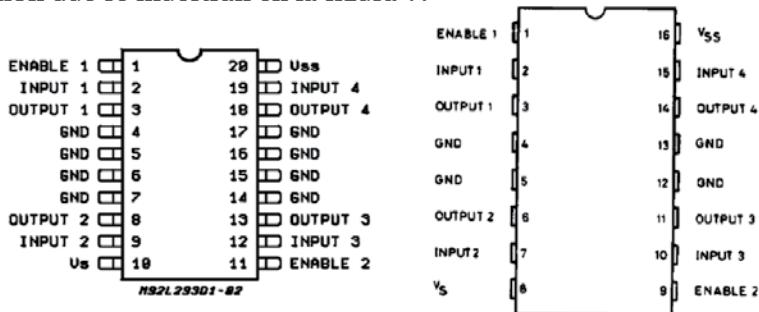


Figura 7. Esquema de conexiones del L293D



## Objetivo

- Diferenciará la forma en la cual se debe de conectar un motor para que se pueda usar como un dispositivo para controlar.
- Describirá las características generales del *hardware* y *software* del Puente H.
- Aumentará su destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos.
- Verificará las imágenes en movimiento, tanto en reversa como en sentido normal, mediante un Arduino conectado a un motor y un circuito integrado.

## Material utilizado

- 1 placa Arduino.
- 2 interruptores (*push button*).
- 1 conector para pila.
- 1 potenciómetro  $10k\Omega$ .
- 1 circuito integrado L293D (Puente H).
- 2 resistencias de  $10k$  Omhs.
- 1 motor de DC.
- 1 batería de 9 volts.

## Desarrollo

Conectamos al Arduino la *protoboard* en las terminales de 5 volts y tierra (figura 8).

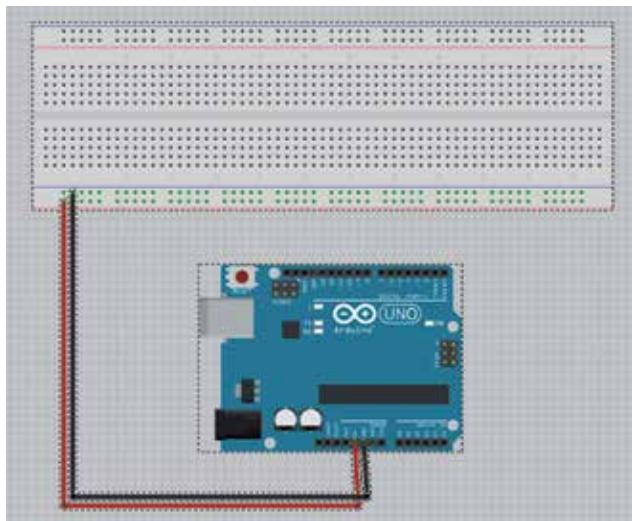


Figura 8. Imagen obtenida del software Fritzing



Agregamos los 2 interruptores a la *protoboard* conectando a sus terminales de alimentación de 5 volts, las 2 resistencias de 10 komhs se agregan al otro extremo en su *pin* respectivo, en serie y hacia tierra de la alimentación de la batería.

El pin de uno de los interruptores se conectarán al *pin* 4 del Arduino y controlará la dirección, el otro interruptor se conectarán al *pin* 5 del Arduino y apagará o encenderá al motor de CD (figura 9).

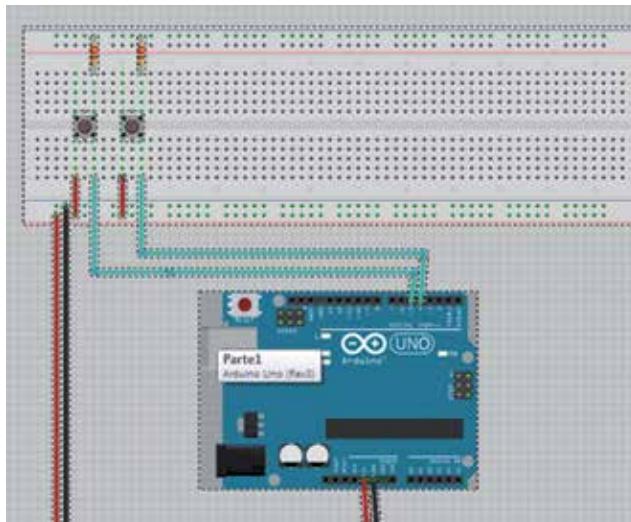


Figura 9. Imagen obtenida del software Fritzing

Conectar el potenciómetro de 10k Omhs en la *protoboard*, 5 volts a uno de los extremos del mismo y el otro extremo a la tierra, el *pin* de en medio se conectara a la entrada analógica del Arduino A0, esto controlará la velocidad del motor de CD (figura 10).

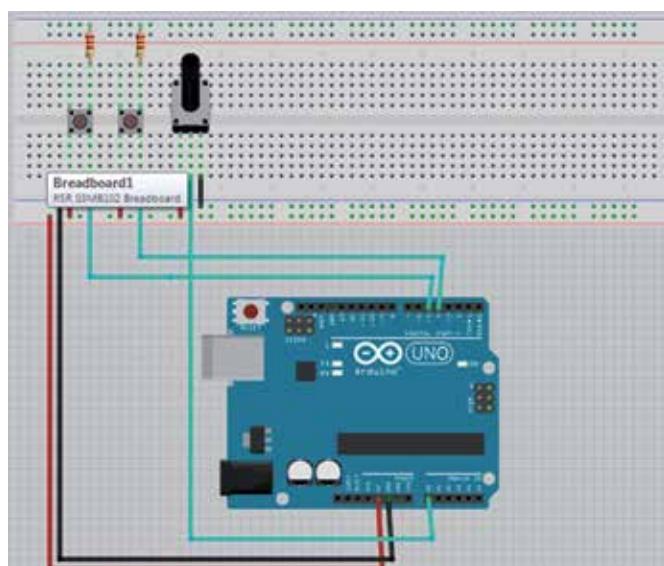


Figura 10. Imagen obtenida del software Fritzing



Agregamos el circuito integrado a la *protoboard*, centrándola entre la división de la *protoboard*, conectamos el *pin 1* hacia el *pin 9* del Arduino, esto es para habilitar al integrado cuando recibe 5 volts, enciende al motor de CD, cuando cambia a 0 volts, apaga al motor. Este *pin* se usará para ajustar la potencia y velocidad del motor.

Conectar el *pin 2* del circuito integrado al *pin digital 3* del Arduino, así como el *pin 7* al *pin 2*, esto servirá de comunicación para indicarnos el giro de la dirección. Si el *pin 3* está en *Low*, y el *pin 2* está en *High*, el motor girará en un sentido; si el *pin 2* está en *low* y el 3 está en *high*, el motor girará en el sentido opuesto. Si las terminales están en *high* o *low* al mismo tiempo, el motor se detendrá.

El circuito integrado se alimenta del *pin 16*, así que lo conectamos a los 5 volts, de la *protoboard* las terminales 5 y 4 se conectan a tierra (figura 11).

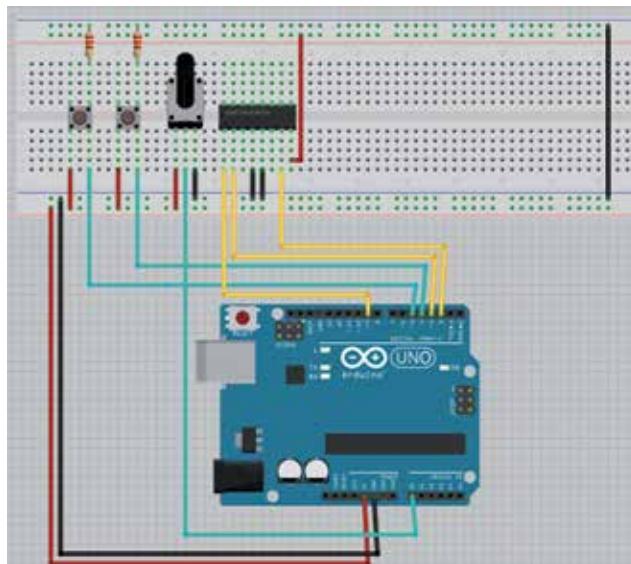


Figura 11. Imagen obtenida del software Fritzing

Conectamos el motor de DC a las terminales del circuito integrado en 3 y 6, éstos encienden o apagan según las señales que se envían a las terminales 2 y 7(figura 12).



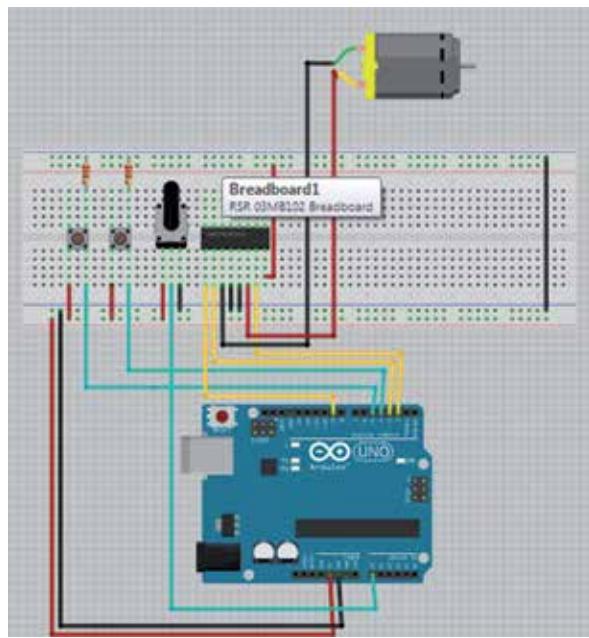


Figura 12. Imagen obtenida del software *Fritzing*

Conectamos la batería a su respectiva vía de alimentación en la *protoboard*, rojo al positivo y negro a tierra, de aquí conectamos al pin 8 del circuito integrado a la vía de la batería en positivo. Este pin es el que alimenta al motor, asegúrenos de que no se junten los 9 y 5 volts, sólo se comparte la tierra (figura 13).

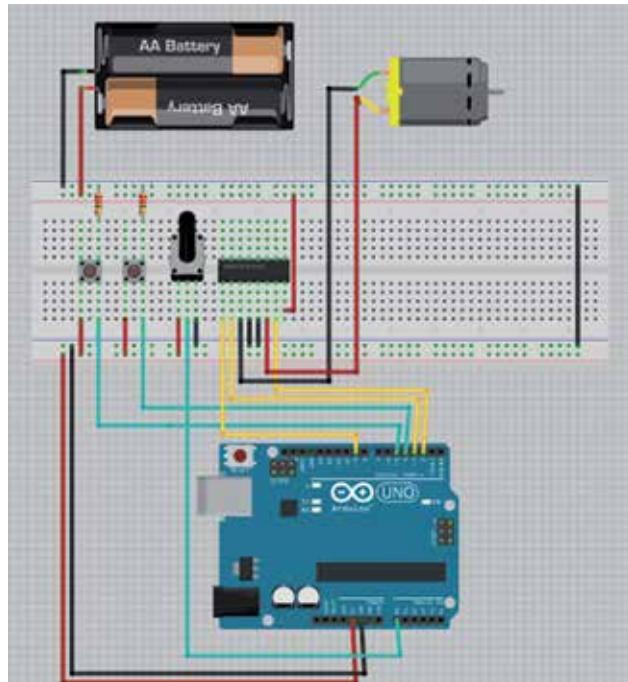
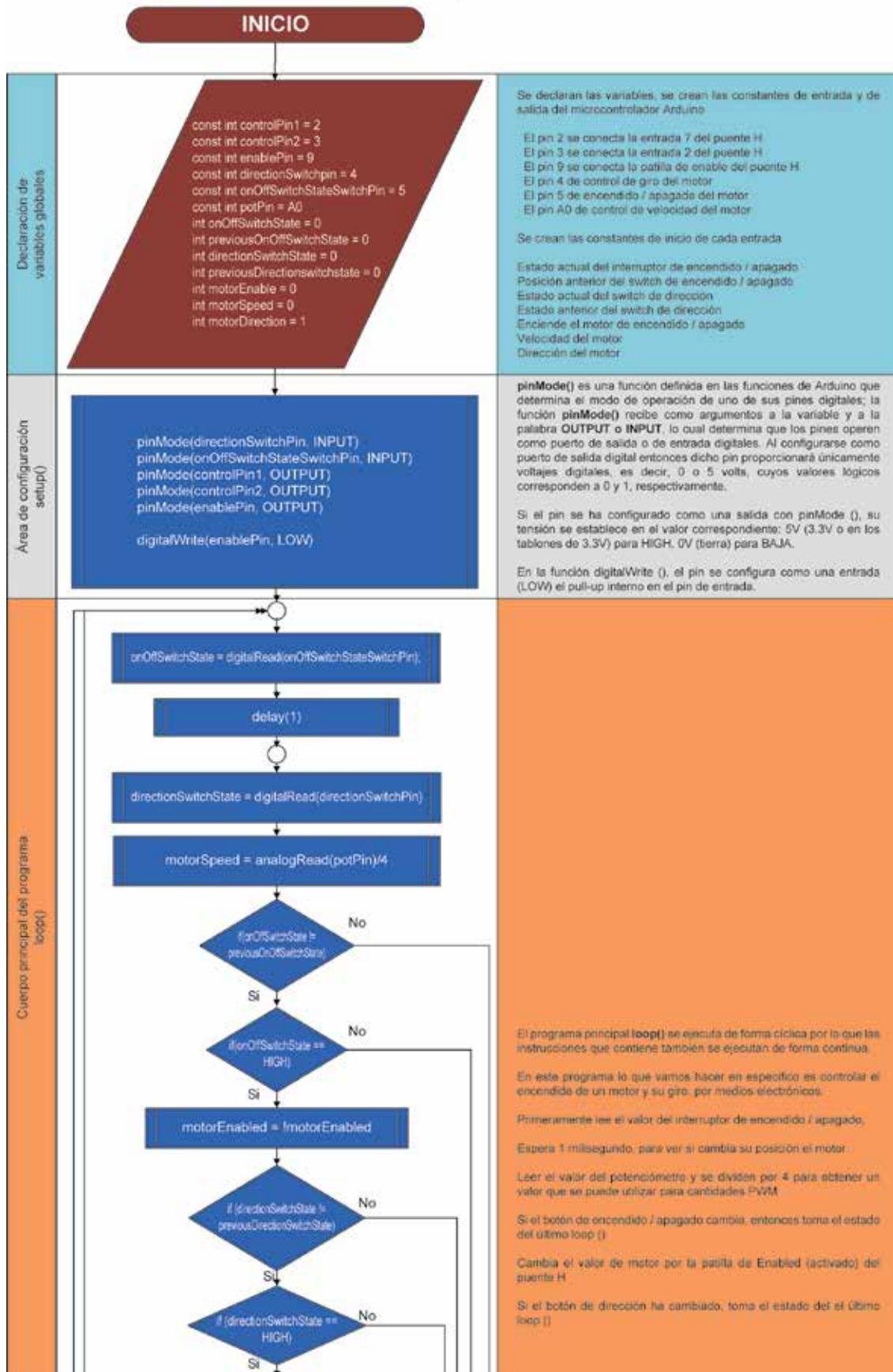


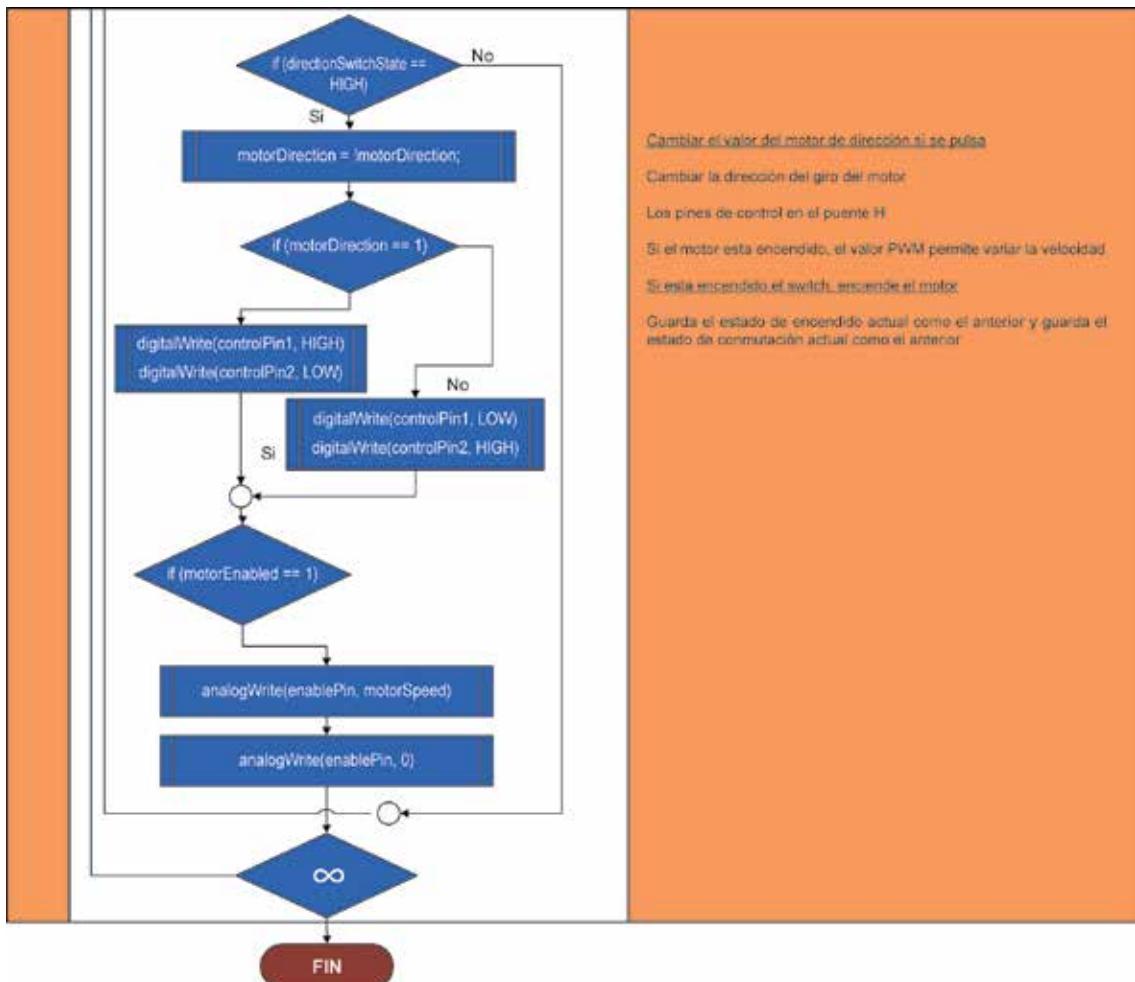
Figura 13. Imagen obtenida del software *Fritzing*



## Diagrama de flujo

## ZOETROPE (Máquina estroboscópica)





## Código

|   |  |
|---|--|
| Declaración de variables globales       | <pre>// Se crean las constantes de entrada y de salida de microcontrolador Arduino const int controlPin1 = 2; // pin en donde se conecta la entrada 7, control del puente H const int controlPin2 = 3; // pin en donde se conecta la entrada 2, control del puente H const int enablePin = 9; // pin en donde se conecta la patilla de enable,del puente H const int directionSwitchpin = 4; // pin de control de giro del motor const int onOffSwitchStateSwitchPin = 5; // pin de encendido / apagado del motor const int potPin = A0; // pin de control de velocidad del motor  // Se crean las constantes de inicio de cada entrada int onOffSwitchState = 0; // Estado actual del interruptor de encendido / apagado int previousOnOffSwitchState = 0; // Posición anterior del switch de encendido / apagado int directionSwitchState = 0; // Estado actual del switch de dirección int previousDirectionswitchstate = 0; // Estado anterior del switch de dirección int motorEnable = 0; //Enciende el motor de encendido / apagado int motorSpeed = 0; // Velocidad del motor int motorDirection = 1; // Dirección del motor</pre> |
| Área de configuración<br>setup()        | <pre>void setup(){     pinMode(directionSwitchPin, INPUT);     // configuramos el interruptor de dirección de giro del motor como salida     pinMode(onOffSwitchStateSwitchPin, INPUT);     // configuramos el interruptor de on/off del motor como entrada     pinMode(controlPin1, OUTPUT);     // configuramos el pin en donde se conecta la entrada 1, control del puente H como salida     pinMode(controlPin2, OUTPUT);     // configuramos el pin en donde se conecta la entrada 1, control del puente H como salida     pinMode(enablePin, OUTPUT);     // Configuramos el pin en donde se conecta la patilla de enable,del puente H, como salida     digitalWrite(enablePin, LOW);     // Configuramos el pin en donde se conecta la patilla de enable,del puente H, como bajo (cero digital) }</pre>   |
| Cuerpo principal del programa<br>loop() | <pre>void loop(){     // Leer el valor del interruptor de encendido / apagado     OnOffSwitchState = DigitalRead(onOffSwitchStateSwitchPin);     delay(1);     // esperamos 1 milisegundo, para ver si cambia su posición el motor     directionSwitchState = digitalRead(directionSwitchPin);     // Leer el valor del potenciómetro y se dividen por 4 para obtener     // Un valor que se puede utilizar para cantidades PWM     motor Speed = analogRead(potpin)/4 ;</pre>   |



Cuerpo principal del programa  
loop()

```
// Si el botón de encendido / apagado cambia, entonces toma el estado del último loop ()  
if(onOffSwitchState != previousOnOffSwitchState) {  
    // Cambiar el valor de motor por la patilla de Enabled (activado) del puente H  
    if(onOffSwitchstate == HIGH) {  
        motorEnabled = !motorEnabled;  
    }  
}  
  
// Si el botón de dirección ha cambiado, toma el estado del el último loop ()  
if (directionSwitchState != previousDirectionSwitchstate) {  
    // Cambiar el valor del motor de dirección si se pulsa  
    if (directionSwitchState == HIGH) {  
        motorDirection = !motorDirection;  
    }  
}  
  
// Cambiar la dirección del giro del motor  
// Los pines de control en el puente H  
if (motorDirection == 1) {  
    digitalWrite(controlPin1, HIGH);  
    digitalWrite(controlPin2, LOW);  
}  
else {  
    digitalWrite(controlPin1, LOW);  
    digitalWrite(controlPin2, HIGH);  
}  
  
// Si el motor esta encendido  
if (motorEnabled == 1) {  
    // PWM permite variar la velocidad  
    analogWrite(enablePin, motorSpeed);  
}  
else {  
    // Si esta encendido el interruptor, enciende el motor  
    analogWrite(enablePin, 0);  
}  
  
// Guardar el estado de encendido actual como el anterior  
previousDirectionSwitchState = directionSwitchstate;  
// Guardar el estado de conmutación actual como el anterior  
previousOnOffSwitchState = onOffSwitchState;  
}
```



## Resultados y conclusiones

Se demuestra con esta práctica que nuestra tarjeta Arduino UNO puede controlar de una forma eficiente y fácil a través de un puente H el giro y velocidad de un motor de corriente directa.

El abordar temas de gran interés nos dejan aprendizajes: en esta práctica se tratarán varios temas, cómo se debe de conectar un motor para que se pueda usar como un dispositivo para controlar. Así mismo, después de lograr el armado del prototipo, podemos describir las características generales del *hardware* y *software* del Puente H.

Con esta práctica aumentamos nuestra destreza en la elaboración y armado de circuitos electrónicos, combinando varios componentes previamente estudiados, garantizando la correcta interacción y funcionamiento de cada uno de ellos, observando el movimiento tanto en reversa como en sentido normal, mediante un Arduino conectado a un motor y un circuito integrado.

## Referencias

Conversor Analógico-Digital (A/D) (s.f.). Arduino LLC. Recuperado el 22 de octubre de 2014, de <http://playground.arduino.cc/ArduinoNotebookTraducion/Appendix6>

Fitzgerald, S., & Shiloh, M. (Eds.). (2012). *Get to know your tools. Arduino Projects Book* (pp. 102-112). Italia: Arduino LLC.

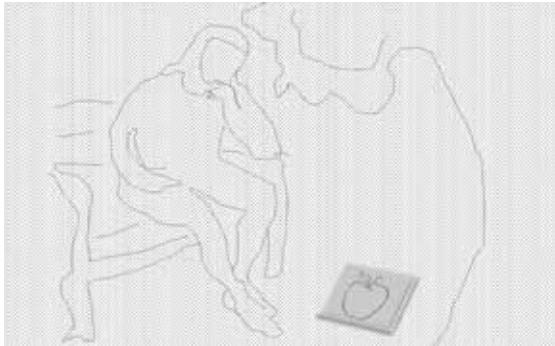
L293D Datasheet (s.f.). Alldatasheet.com. Recuperado el 22 de octubre de 2014, de <http://www.alldatasheet.com/datasheetpdf/pdf/22432/STMICROELECTRONICS/L293D.html>

El Puente-H (o H-bridge) (s.f.). Electrónica Hispavila. Recuperado el 22 de octubre de 2014, de <http://www.hispavila.com/3ds/atmega/hpuente.html>



## Actividad 27. Medición de la gravedad terrestre mediante un péndulo simple con Arduino

Víctor Hugo Leyva García / Norberto Alejandro Pérez Colín



*Cuando Isaac Newton vio caer la manzana del árbol, entró en profunda meditación acerca de la causa que arrastra a todos cuerpos siguiendo una línea que, si se prolongara, pasaría muy cerca del centro de la Tierra.*

Voltaire, "Philosophie de Newton", 1738.

### Resumen

En este trabajo se propone un sistema de medición del valor de la gravedad terrestre a partir de un péndulo simple y un sistema de censado para la medición del periodo del péndulo. Las mediciones se realizaron en Ciudad Universitaria y se encontró el valor de la gravedad de  $g_{CU} = 9.53927726 \pm 0.255698752 \text{ m/s}^2$ . Se considera que el error encontrado en nuestras mediciones es del 2.27%, respecto al valor reportado por el CENAM.

### Introducción

La gravedad es una de las fuerzas fundamentales del universo. Las otras tres fuerzas son el electromagnetismo, la interacción nuclear fuerte y la interacción nuclear débil.

La gravedad es la responsable que los cuerpos, dado su peso, caigan con aceleración constante. La fuerza de gravedad es responsable del movimiento de los planetas y las estrellas en la galaxia, pero también del movimiento de ella.

El valor de la gravedad presenta su valor máximo en la superficie terrestre y disminuye con la profundidad (hacia el interior de la Tierra) y con la altura (hacia el espacio exterior). Sin embargo, debido a la forma irregular de la Tierra (denominada geoide), el valor de la gravedad en la superficie terrestre varía con la latitud, de tal forma que el valor de  $g$  es menor en el ecuador que en los polos, es decir,  $g_e = 9.78049 \text{ m/s}^2$  y  $g_p = 9.83221 \text{ m/s}^2$ , respectivamente.



La aceleración de la gravedad terrestre puede ser medida de las siguientes maneras:

- a) Mediante la caída libre de un cuerpo testigo.
- b) Mediante la oscilación de un péndulo en oscilación libre.
- c) Mediante la oscilación de una masa sujetada a un muelle.

Uno de los experimentos más sencillos para determinar la aceleración de la gravedad, consiste en medir el periodo de un péndulo constituido por una masa puntual suspendida del extremo de un hilo delgado de cierta longitud. Galileo Galilei encontró que el tiempo de oscilación del péndulo (periodo) es independiente de la masa o del ángulo de lanzamiento del mismo. De tal manera que el periodo depende sólo del largo de la cuerda del péndulo y de la aceleración de la gravedad. En este sentido, se deduce que si se conoce la longitud de la cuerda de un péndulo y su periodo de oscilación, entonces es posible determinar el valor de la aceleración local; ésta es la experiencia que se describe en este proyecto.

La oscilación de un péndulo es un movimiento vibratorio que también se conoce como “movimiento armónico simple”. El movimiento armónico simple es un movimiento periódico en el que la posición varía según una ecuación de tipo senoidal.

## Marco teórico

Todo cuerpo suspendido por un punto que puede oscilar alrededor de un eje que pasa por él y que no contenga al centro de gravedad, es un péndulo.

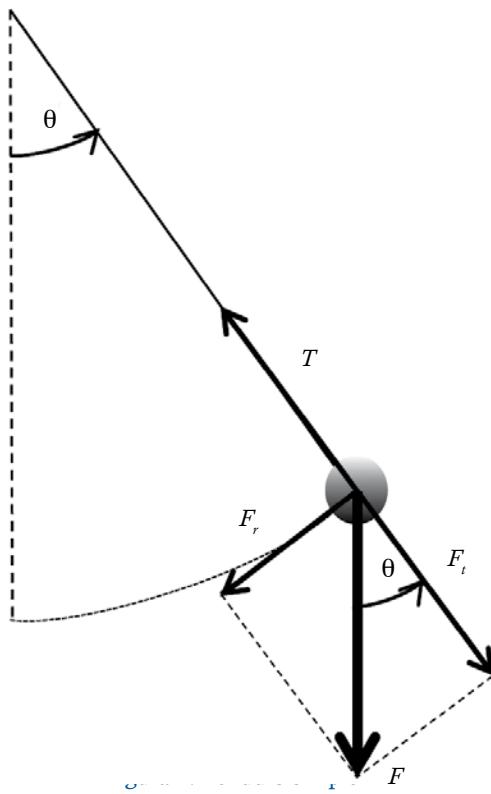
El movimiento oscilatorio del péndulo está caracterizado por los siguientes parámetros:

- *Oscilación completa o ciclo.* Es el desplazamiento de la masa desde uno de sus extremos más alejados de la posición de equilibrio hasta su punto simétrico (pasando por la posición de equilibrio) y desde este punto de nuevo hasta la posición inicial, es decir, dos oscilaciones sencillas.
- *Periodo.* Es el tiempo empleado por la masa en realizar un ciclo u oscilación completa.
- *Frecuencia.* Es el número de ciclos realizados en la unidad de tiempo.
- *Amplitud.* Es el máximo valor de la elongación o distancia hasta el punto de equilibrio, que depende del ángulo entre la vertical y la cuerda.

Un *péndulo simple* es un sistema ideal formado por un cuerpo de masa  $m$  suspendido de una cuerda indeformable de masa despreciable y longitud  $l$ . Si se empuja la masa fuera de su posición de equilibrio y luego se suelta, el péndulo oscilará en un plano vertical bajo la acción de la gravedad.

Fuera de su posición de equilibrio, la cuerda forma un ángulo  $\theta$  con la vertical. Las fuerzas que actúan sobre la masa son el peso ( $F$ ) de ésta y la tensión ( $T$ ) de la cuerda.





El peso ( $F$ ) del cuerpo está determinado por:

$$F = mg \quad (\text{Ec. 1})$$

donde  $g$  es el valor de la aceleración de la gravedad y  $m$  es la masa de dicho cuerpo.

Para su análisis, el peso de la masa puede descomponerse en sus componentes radial (de magnitud  $F_r = F \cdot \cos\theta$ ) y tangencial (de magnitud  $F_t = F \cdot \sin\theta$ ). La componente radial tiene igual magnitud, pero con sentido opuesto, que la tensión que la cuerda ejerce sobre la masa, por lo que no interviene en el movimiento del cuerpo. La componente tangencial, perpendicular a la cuerda es la *fuerza restauradora* que actúa sobre la masa obligándola a regresar a su posición de equilibrio ( $\theta = 0$ ).

Por estos motivos se emplea la fuerza restauradora  $F_r$  en el análisis del movimiento del péndulo. Dicha fuerza está determinada por la siguiente ecuación:

$$F_r = -mg \cdot \sin\theta \quad (\text{Ec. 2})$$

donde el signo menos indica que la fuerza se opone a la dirección en que se incrementa  $\theta$ .

En la ecuación anterior se observa que la fuerza restauradora  $F_r$  es proporcional a  $\sin\theta$ , y no al desplazamiento angular  $\theta$ . Esta situación implica que el movimiento resultante no es estrictamente armónico simple. Sin embargo, si el ángulo  $\theta$  es pequeño, el seno de este ángulo es aproximadamente igual a  $\theta$  (medido en radianes), es decir:



$$\operatorname{sen}\theta \approx \theta \quad (\text{Ec. 3})$$

De tal forma que para ángulos pequeños el movimiento del péndulo es prácticamente una línea recta, y el desplazamiento a lo largo del arco es:

$$x = l \cdot \theta \quad (\text{Ec. 4})$$

Sustituyendo la ecuación 4 en la ecuación 2 tenemos que:

$$F_r = -mg\theta \quad (\text{Ec. 5})$$

Y a partir de las ecuaciones 4 y 5 se obtiene que:

$$F_r = -\left(\frac{mg}{l}\right)x \quad (\text{Ec. 6})$$

De la ecuación 6 se observa que para pequeños desplazamientos , la fuerza restauradora es proporcional al desplazamiento y actúa en dirección opuesta al desplazamiento, lo cual constituye el criterio básico del movimiento armónico simple.

En este sentido, el periodo de oscilación del péndulo simple para pequeños desplazamientos está dado por:

$$T = 2\pi\sqrt{\frac{l}{g}} \quad (\text{Ec. 7})$$

De la ecuación anterior se deduce que la oscilación no depende ni de la masa  $m$  ni de la amplitud inicial  $\theta$  , por lo que puede calcularse  $g$  a partir de mediciones del periodo de oscilación ( $T$ ) y de la longitud de la cuerda ( $l$ ), es decir:

$$g = 4\pi^2 \frac{l}{T^2} \quad (\text{Ec. 8})$$

## Objetivo de la investigación

El objetivo de este trabajo es determinar de manera automática el valor local de la aceleración de la gravedad utilizando un péndulo simple y un sensor infrarrojo conectado a un Arduino UNO.

La medición del valor de la gravedad a partir de la oscilación de un péndulo es un método clásico desarrollado en los cursos de física del Colegio de Ciencias y Humanidades, que permite calcular el valor de la gravedad con una precisión considerable. Sin embargo, durante desarrollo de estas mediciones se incurren a diversos errores durante la toma de lecturas.

El proyecto presentado en este trabajo pretende introducir mejoras creativas y significativas en un sistema del péndulo simple, que permitan determinar con una mayor precisión y exactitud las mediciones del valor de la gravedad.

Asimismo, se pretende que los alumnos se familiaricen con el uso y programación de arduino.



## Problema

El valor de la aceleración de la gravedad se puede determinar con bastante precisión, a través de la medición del periodo de oscilación de un péndulo simple de longitud determinada. Sin embargo, la precisión en el cálculo del valor de la gravedad depende de la precisión y de la exactitud con la que se toman las mediciones del periodo de oscilación, así como de los errores humanos inherentes durante las mediciones.

En este proyecto se pretenden disminuir los errores en el cálculo del valor de la gravedad mediante el uso de un sensor infrarrojo, controlado con el microcontrolador arduino, y con ayuda del *software audacity*, para el procesamiento de las señales enviadas por el microcontrolador a la computadora. El sensor infrarrojo nos genera una señal analógica que permite determinar el periodo de oscilación del péndulo y, posteriormente, el valor de la gravedad.

## Hipótesis

Se espera que el valor de la gravedad medido en la Ciudad Universitaria de la UNAM sea menor al valor que se tiene para el nivel del mar ( $9.81 \text{ m/s}^2$ ).

## Materiales

- *LED IR.*
- Receptor IR.
- Arduino uno.
- 1 resistencia de  $500 \Omega$ .
- 1 resistencia de  $100 \Omega$ .
- 1 *push button*.
- Péndulo.
- Computadora con el siguiente *software* instalado: *Audacity y Excel*.

## Desarrollo

### *Diseño del péndulo.*

En la figura 2 se muestra el péndulo diseñado para el desarrollo del proyecto. La longitud ( $l$ ) del péndulo es de  $l = 84 \text{ cm}$ . Esta longitud corresponde a los 82 cm de longitud de la cuerda más 2 cm del centro de masa de la pelota.





Figura 2. Diseño del péndulo

*Sistema de medición del periodo de oscilación del péndulo.*

En la figura 3 se muestra el péndulo y el sistema de medición del periodo de oscilación del péndulo compuesto por: sensor y transmisor IR controlados por el microcontrolador Arduino y computadora con el software Audacity.

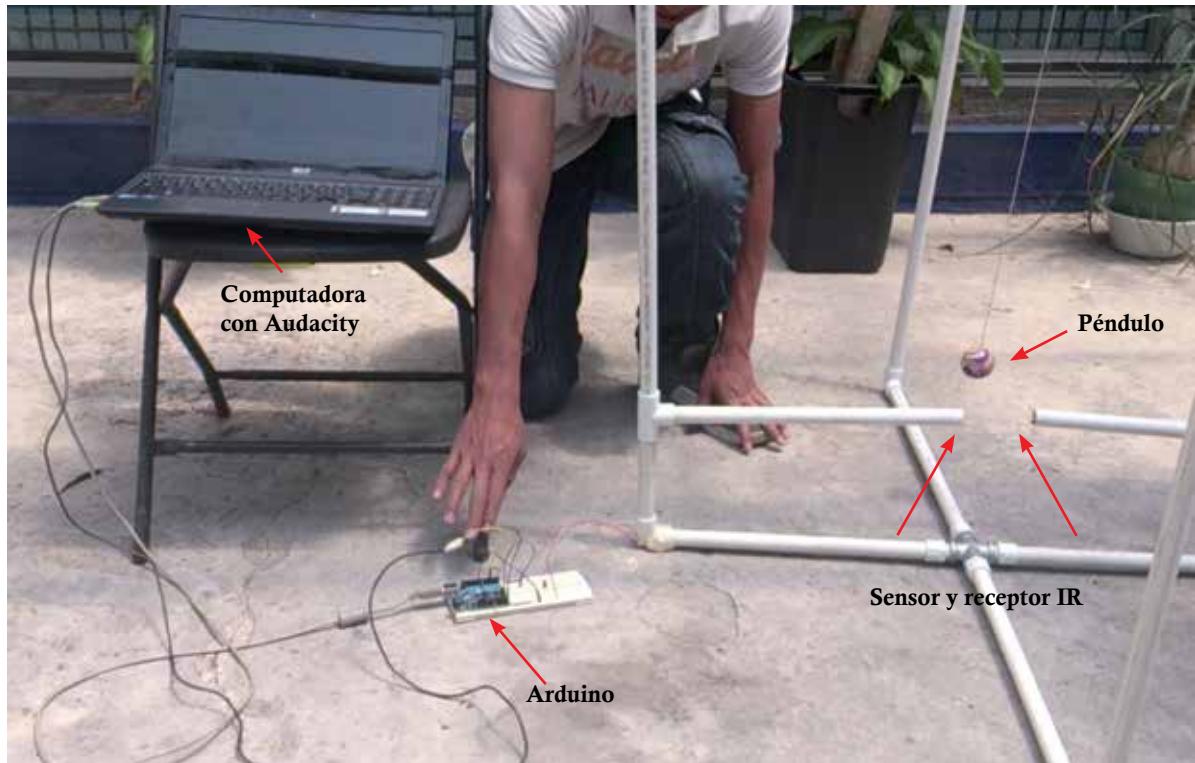


Figura 3. Sistema de medición del periodo de oscilación del péndulo



## Diagrama electrónico

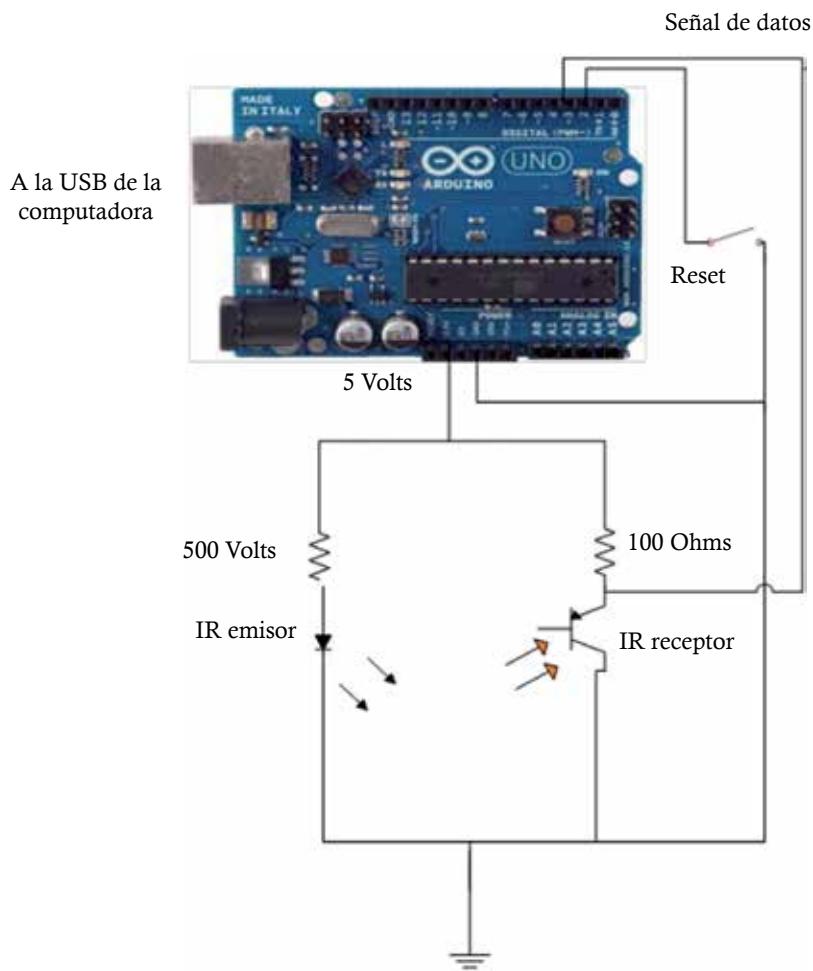


Figura 3. Esquema electrónico del prototipo



## Código

|                                   |   |
|-----------------------------------|---|
| Declaración de variables globales | <pre>// Medición de la gravedad terrestre mediante un péndulo simple con Arduino int cicles=20; int debounce=200; double pi = 3.1415926535; double longitud = 0.920; double period = 0; double time; double gravity =0;</pre>   |
| setup()                           | <pre>void setup() {     Serial.begin(9600);     pinMode(3, INPUT); // Sensor infrarrojo     pinMode(2, INPUT);     cicles = 2 *cicles; }</pre>  |
| loop()                            | <pre>void loop() {     // Lectura del sensor infrarrojo     unsigned long timeCounter[cicles];     int i =0;     if (digitalRead(2)==LOW){         Serial.println("Captura de datos:");         while (i&lt;=cicles){             if (digitalRead(3)==HIGH){ // 5 Volts infrarrojo                 timeCounter[i]=millis();                 Serial.print(i);                 Serial.print(": ");                 Serial.print(timeCounter[i]);                 Serial.println();                 delay(debounce); // for debounce                 i++;             }         }         // Cálculo del valor de la gravedad terrestre         time=0;         for (int i=cicles; i &gt; 1; i=i-2){             time = time + (timeCounter[i]-timeCounter[i-2]);         }         time = time/1000;         period = time / (cicles/2);         Serial.print("T: ");         Serial.print(time);         Serial.println();         gravity = ((4*longitud*pi*pi))/(period*period);         Serial.print("Valor de la gravedad: ");         Serial.print(gravity);         Serial.print(" (m/s^2)");         Serial.println();         delay(1000);     } }</pre> |



## Resultados

En la siguiente tabla se muestran las mediciones del periodo del péndulo obtenidas con el sistema de medición propuesto. Asimismo, se muestran los valores para la gravedad obtenidas con dichas mediciones y a partir de la ecuación 8.

| Experimento 1  |                                 | Experimento 2  |                                 | Experimento 3  |                                 |
|--|---------------------------------|--|---------------------------------|--|---------------------------------|
| Periodo<br>[s]   | Gravedad<br>[m/s <sup>2</sup> ] | Periodo<br>[s]   | Gravedad<br>[m/s <sup>2</sup> ] | Periodo<br>[s]   | Gravedad<br>[m/s <sup>2</sup> ] |
| 1.87   | 9.483219648                     | 1.88   | 9.382602645                     | 1.87   | 9.483219648                     |
| 1.88   | 9.382602645                     | 1.84   | 9.794976012                     | 1.88   | 9.382602645                     |
| 1.88   | 9.382602645                     | 1.89   | 9.283578508                     | 1.88   | 9.382602645                     |
| 1.84   | 9.794976012                     | 1.85   | 9.689370573                     | 1.85   | 9.689370573                     |
| 1.88   | 9.382602645                     | 1.88   | 9.382602645                     | 1.85   | 9.689370573                     |
| 1.86   | 9.585463865                     | 1.87   | 9.483219648                     | 1.85   | 9.689370573                     |
| 1.89   | 9.283578508                     | 1.84   | 9.794976012                     | 1.87   | 9.483219648                     |
| 1.89   | 9.283578508                     | 1.86   | 9.585463865                     | 1.85   | 9.689370573                     |
| 1.85   | 9.689370573                     | 1.89   | 9.283578508                     | 1.85   | 9.689370573                     |
| 1.89   | 9.283578508                     | 1.89   | 9.283578508                     | 1.86   | 9.585463865                     |
| 1.88   | 9.382602645                     | 1.86   | 9.585463865                     | 1.89   | 9.283578508                     |
| 1.89   | 9.283578508                     | 1.86   | 9.585463865                     | 1.89   | 9.283578508                     |
| 1.87   | 9.483219648                     | 1.88   | 9.382602645                     | 1.86   | 9.585463865                     |
| 1.89   | 9.283578508                     | 1.85   | 9.689370573                     | 1.89   | 9.283578508                     |
| 1.86   | 9.585463865                     | 1.89   | 9.283578508                     | 1.88   | 9.382602645                     |
| 1.86   | 9.585463865                     | 1.86   | 9.585463865                     | 1.87   | 9.483219648                     |
| 1.84   | 9.794976012                     | 1.86   | 9.585463865                     | 1.89   | 9.283578508                     |
| 1.84   | 9.794976012                     | 1.86   | 9.585463865                     | 1.87   | 9.483219648                     |
| <b>Valor promedio de la gravedad [m/s<sup>2</sup>]</b> | <b>9.485857368</b>              | <b>Valor promedio de la gravedad [m/s<sup>2</sup>]</b> | <b>9.51371211</b>               | <b>Valor promedio de la gravedad [m/s<sup>2</sup>]</b> | <b>9.490710064</b>              |

Mediante el análisis de las mediciones experimentales realizadas con el instrumento de medición de la gravedad que proponemos, hemos encontrado que en Ciudad Universitaria, lugar donde se desarrollaron las mediciones, el valor de la gravedad es:

$$g_{CU} = 9.53927726 \pm 0.255698752 \text{ m/s}^2$$



## Análisis e interpretación de resultados

En los resultados presentados en el inciso anterior se muestra el promedio de los valores calculados para la gravedad en Ciudad Universitaria, lugar donde se realizaron las mediciones. Ya que nuestras mediciones indican que el valor de las estás mediciones difieren del valor reportado por el Centro Nacional de Metrología (CENAM), que es de  $g = 9.780845 \text{ m/s}^2$ , valor de la gravedad para el Distrito Federal.

Si comparamos el valor reportado por el CENAM contra el valor que nosotros hemos encontrado, se observa un error del 2.27% , es decir:

$$\% \text{ error} = \frac{|9.780845 - 9.53927726|}{9.780845} * 100 = 2.27\%$$

Consideramos que si bien tenemos un error en el valor encontrado, es posible mejorar la exactitud de nuestras mediciones mediante la selección de un sensor y un transmisor de IR con mejores características técnicas a los seleccionados en el diseño del sistema.

## Referencias

Sotelo Fajardo, J. C. (2012). “El Concepto de gravedad desde las concepciones de Newton y Einstein: Una propuesta didáctica dirigida a estudiantes de Ciclo V.” Tesis de Maestría. Universidad Nacional de Colombia.

Cálculo de la Atracción Local de la Gravedad (2006). Centro Nacional de Metrología. Recuperado el 16 de marzo de 2013, de <http://www.cenam.mx/fyp/gravedad.html>

PÉNDULO SIMPLE. MEDIDA DE  $g$  (s.f.). Universidad Complutense de Madrid. Recuperado el 10 de marzo de 2013, de <http://pendientedemigracion.ucm.es/info/Geofis/Prácticas/prac05.pdf>.



## Evaluación. Modelo de semáforo

Edith López Martínez

Se utiliza la metodología del Aprendizaje Basado en Problemas (ABP) enfocado a profesores y alumnos del bachillerato, por las ventajas que tiene, ya que estimula e involucra al interactuar con la realidad de los prototipos electrónicos, además de observar resultados inmediatos. Y generar aprendizajes significativos con una dinámica hacia un pensamiento crítico y creativo.

Durante el desarrollo en la creación de los prototipos de la “Guía didáctica de Robótica educativa con Arduino. Aplicaciones para el bachillerato” los aprendizajes que se pretenden tienen el objetivo de comprender y no sólo de memorizar, conceptos y procedimientos. Al estimular habilidades con este método de estudio autodirigido con el microcontrolador Arduino, se pretende mejorar la capacidad para estudiar e investigar sin ayuda, con el fin de afrontar cualquier obstáculo futuro, de orden teórico o práctico, y poder realizar sus propios prototipos electrónicos. Causa por la cual se presenta la siguiente evaluación con el fin de apropiarse de los conocimientos dentro de una planeación por el método de proyectos.

### Planeación de una Práctica con el método de proyectos

#### *Objetivo*

Creación y elaboración de un proyecto con el microcontrolador Arduino en forma personal aplicado a Robótica educativa para el bachillerato.

#### *Criterios de aplicación*

- Describir la estructura del microcontrolador Arduino y los dispositivos que comparten en el prototipo electrónico.
- Describir la función de las entradas y salidas digitales, analógicas y de potencia, en *software* y *hardware*.
- Describir cómo se almacena la información, cómo se compila y cómo se corren los programas en Arduino.
- Interpretar y comprender los textos para enlistar las funciones de las entradas y salidas digitales, analógicas y de potencia, en *software* y *hardware* del microcontrolador Arduino.

#### *Planeación de la práctica*

El alumno, apoyándose del esquema electrónico de conexión de la figura 1, desarrollará el prototipo de nombre “Modelo de semáforo”, siguiendo los objetivos señalados. Asimismo, deberá investigar y desarrollar las diferentes secciones que conforman a una práctica: antecedentes, introducción, material utilizado, desarrollo, diagrama de flujo, código, resultados y conclusiones y referencias.

En esta práctica se busca conocer el funcionamiento del pulsador al mandar la señal hacia los ledes o diodos emisores de luz, uno a la vez, para simular un semáforo, así como la configuración



de las entradas y salidas digitales que se van a manejar en el microcontrolador Arduino UNO. Para esto se utilizarán tres diodos emisores de luz conectados a las salidas digitales de la tarjeta Arduino, así como un pulsador que nos permitirá el control de los diodos. El esquema electrónico nos muestra cómo los ledes están conectados cada uno con su resistencia con el fin de protegerlos y que cada uno de ellos funcione de manera óptima.

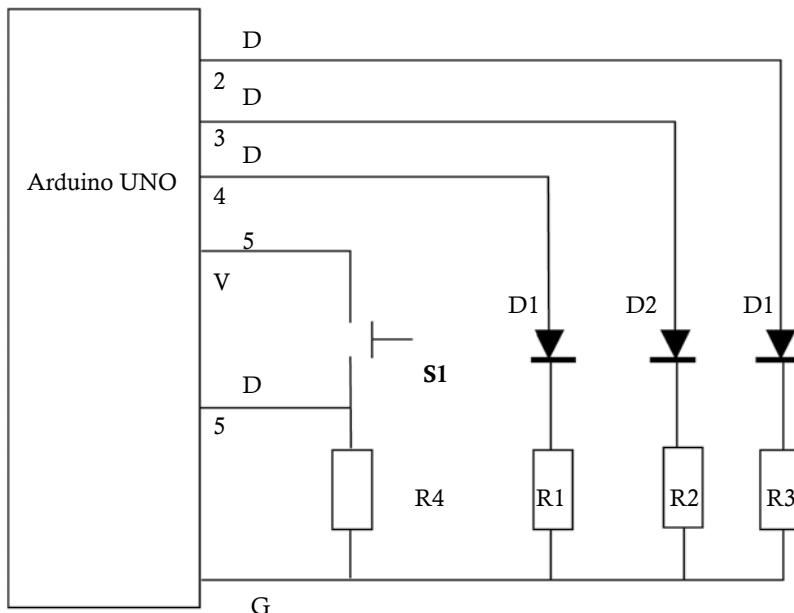


Figura 1. Esquema electrónico de conexión

## Objetivo

- Utilizará los *pines* de alimentación del microcontrolador Arduino UNO.
- Describirá e identificará las características generales del pulsador.
- Diferenciará e identificará las terminales analógicas y digitales del microcontrolador Arduino.
- Hará uso de la programación para controlar las terminales digitales del Arduino.
- Desarrollará la práctica para conocer la configuración y el funcionamiento de las terminales de Arduino como entradas o salidas.
- Identificará las terminales digitales del microcontrolador Arduino.
- Hará uso de resistencias y conocerá su principal función, la cual limita la corriente como método de protección.
- Controlará el encendido y apagado de los ledes mediante un pulsador.

Las secciones restantes deben ser desarrolladas por los alumnos como parte de la evaluación.



## Solución de la evaluación. Modelo de semáforo

Edith López Martínez

### Antecedentes

Para comenzar con esta práctica es necesario recordar que en la placa de Arduino UNO vienen integrados 14 terminales, numerados del 0 al 13 y que éstos pueden ser utilizados como entrada y salida digitales.

Pero, ¿por qué los llamamos entradas y salidas digitales? porque las terminales de entrada sirven para escuchar información del exterior y los de salida para enviar la información de la placa hacia el exterior y, además, sólo pueden manejar los valores 0 y 1, es decir, 0 volts o 5 volts.

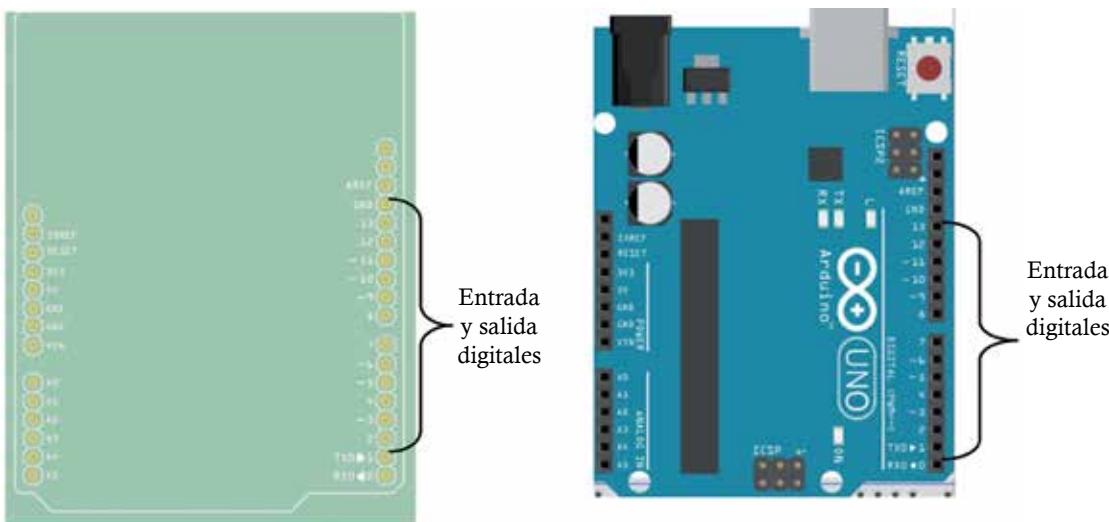


Figura 1. Entradas y salidas digitales de Arduino. Imágenes generadas con *Fritzing*

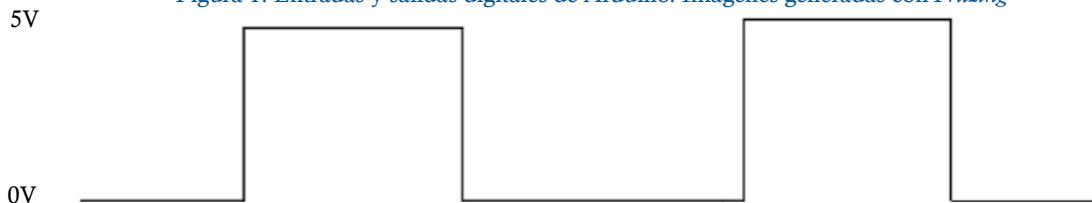


Figura 2. Señal cuadrada de voltaje

### Introducción

En esta práctica se da a conocer el funcionamiento del pulsador al mandar la señal hacia los ledes o diodos emisores de luz, así como la configuración de las entradas y salidas digitales que se van a manejar en el microcontrolador Arduino UNO. Para esto se utilizarán tres diodos emisores de luz conectados a las salidas digitales de la tarjeta Arduino, así como un pulsador que nos permitirá el control de los diodos. El esquema electrónico nos muestra cómo los ledes están conectados cada uno con su resistencia con el fin de protegerlos y que cada uno de ellos funcione de manera óptima.



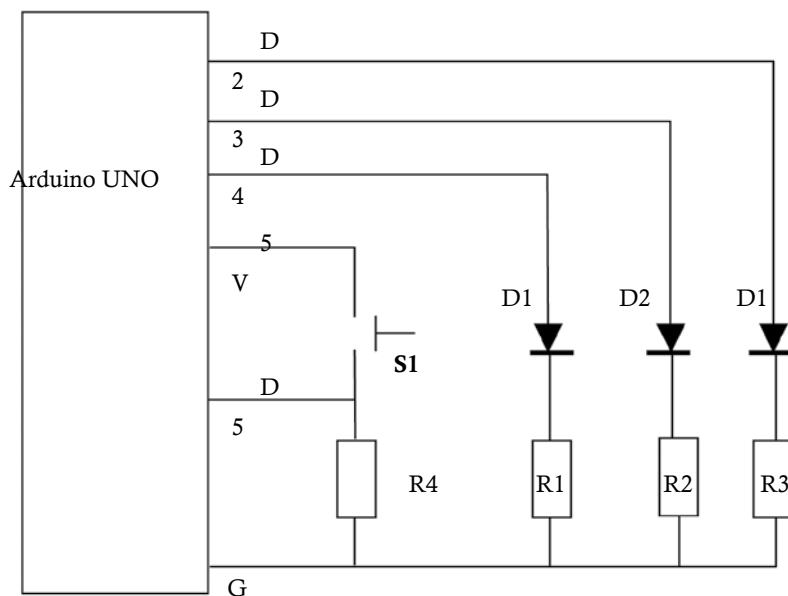


Figura 3. Esquema electrónico de conexión

## Objetivos

- Utilizará las terminales de alimentación del microcontrolador Arduino UNO.
- Describirá e identificará las características generales del pulsador.
- Diferenciará e identificará las terminales analógicas y digitales del microcontrolador Arduino.
- Hará uso de la programación para controlar las terminales digitales del Arduino.
- Desarrollará la práctica para conocer la configuración y el funcionamiento de las terminales de Arduino como entradas o salidas.
- Identificará las terminales digitales del microcontrolador Arduino.
- Hará uso de resistencias y conocerá su principal función, la cual limita la corriente como método de protección.
- Controlará el encendido y apagado de los ledes mediante un pulsador.

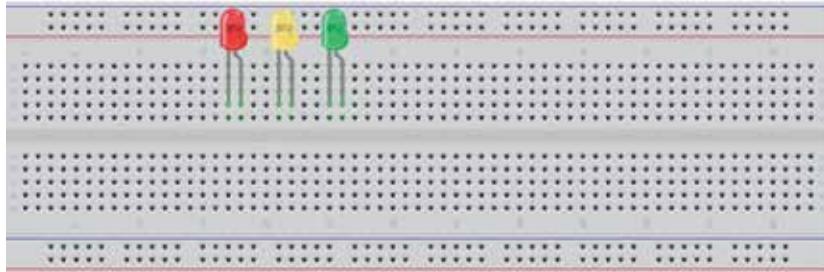
## Material utilizado

- Equipo de cómputo.
- Cable estándar USB (conexión A a conexión B).
- Microcontrolador Arduino uno.
- 3 resistencias de  $270\ \Omega$ .
- 1 resistencia de  $100\ K\Omega$ .
- 1 LED color rojo.
- 1 LED color Amarillo.
- 1 LED color verde.
- 1 push button

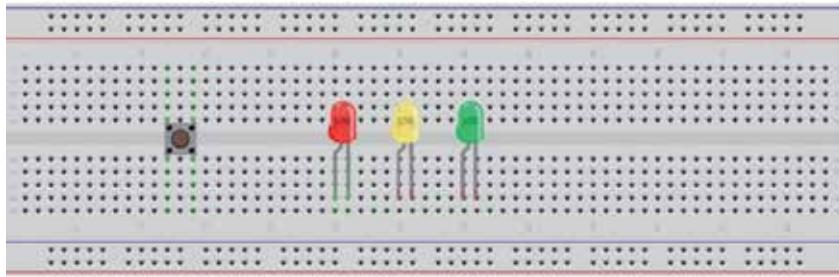


## Desarrollo

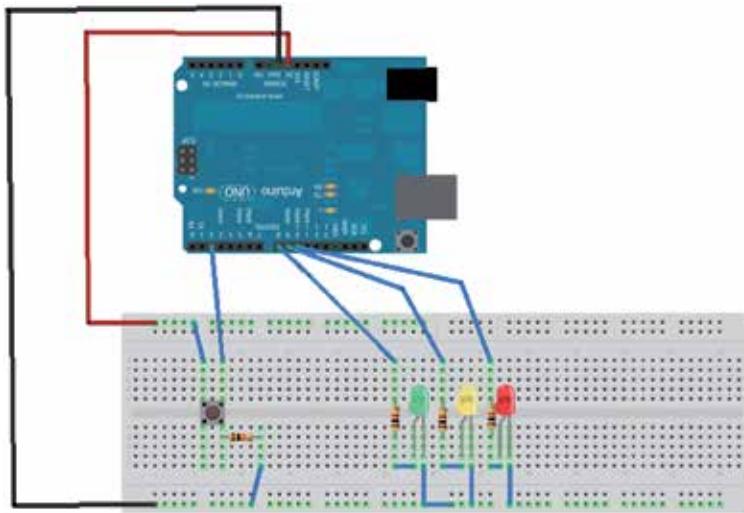
En el *protoboard* conectamos un *LED* rojo, uno verde y un amarillo.



Colocamos el *push button*.



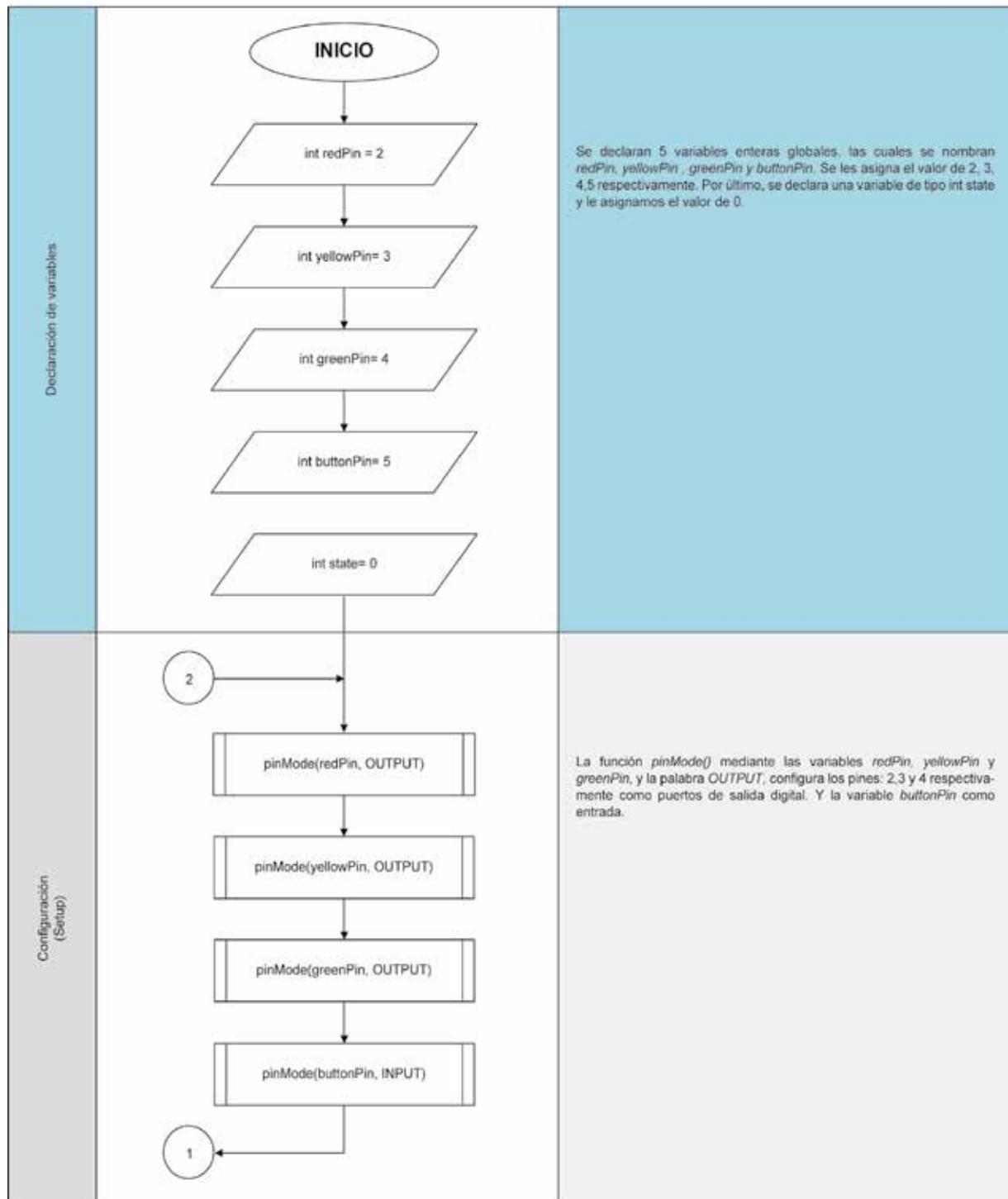
Y procedemos con el cableado de la placa Arduino con el *protoboard*, de acuerdo con el esquema electrónico hasta quedar de la siguiente manera:



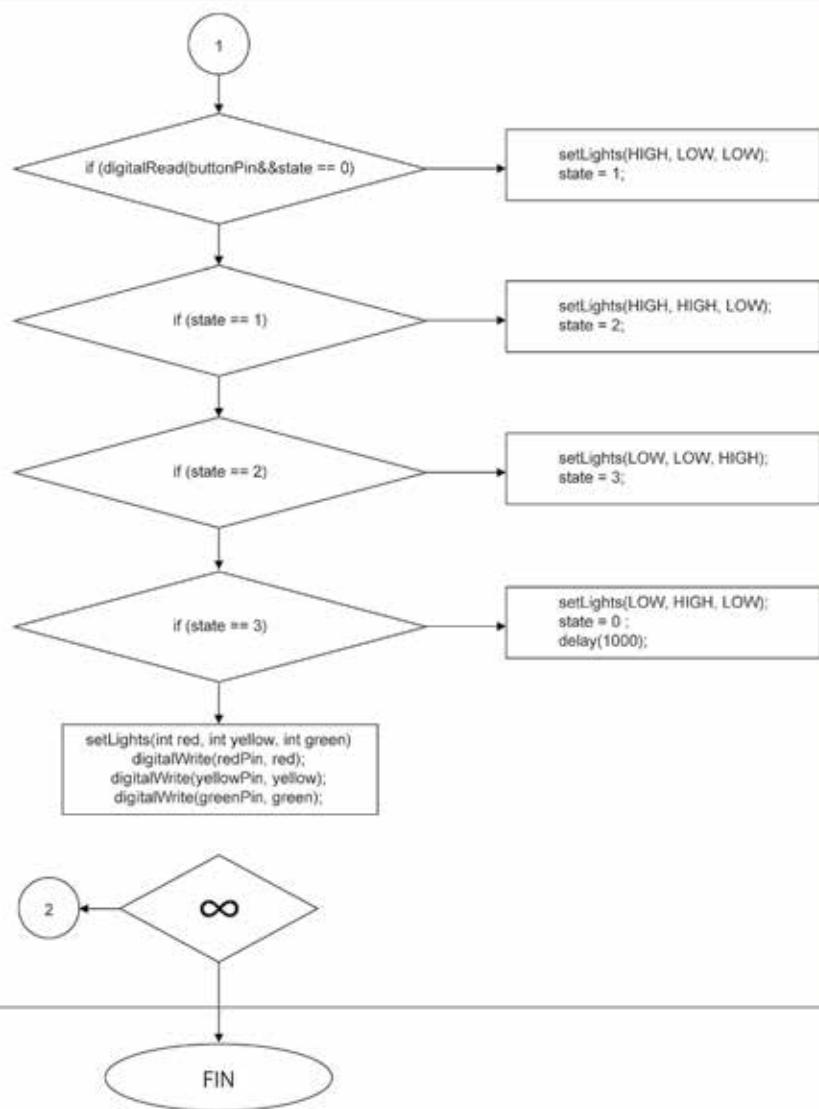
Cada vez que pulsemos el botón, la luz del semáforo cambiará al color siguiente de la secuencia, es decir, de verde a amarillo y de amarillo a rojo. Además, si dejamos presionado el botón, los ledes cambian solos de acuerdo a la secuencia mencionada.



## Diagrama de flujo



Cuerpo principal del programa (loop())



De acuerdo al valor que state tenga y la función digitalRead() del pulsador vamos a comparar el valor de state y según su valor los leds van a ir prendiendo apagando

## Código

|                                   |  |
|-----------------------------------|--|
| Declaración de variables globales | <pre>int redPin = 7; int yellowPin = 4; int greenPin=2; float 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); }</pre> |
| Área de configuración setup()     |  |

## Resultados y conclusiones

Durante el desarrollo de esta práctica se demostró que mediante el uso del microcontrolador Arduino y la programación del mismo, se puede tomar control de las señales de un semáforo mediante un pulsador. Los resultados obtenidos fueron satisfactorios ya que se pudo realizar la simulación del funcionamiento del semáforo y también, se comprobó que si mantenemos el botón pulsado, los ledes prenden y apagan según la secuencia de manera continua.

Se utilizaron las terminales de alimentación del microcontrolador ARDUINO para hacer funcionar a los elementos del *protoboard*, en esta práctica también se conoció la función de un pulsador y las características del mismo.

Gracias a la realización de esta práctica, se identificaron cuáles eran las terminales digitales y los analógicos; al realizar esto, se hizo uso de las terminales digitales mediante la programación de la placa de Arduino UNO y se utilizaron como entradas y salidas. Asimismo, se dio a conocer el uso y la importancia de las resistencias que evitan que se dañen los dispositivos por el paso de la corriente, en este caso, se utilizaron para la protección de los diodos emisores de luz y del pulsador.

## Referencias

- Monk, S. (2012). Modelo del semáforo. *30 proyectos con Arduino* (pp. 41-44). Madrid: Editorial Esteribor, S.L.





*Guía didáctica de robótica educativa con Arduino.  
Aplicaciones para el bachillerato*

Editado por el Colegio de Ciencias y Humanidades de la UNAM,  
se terminó de imprimir en abril de 2016.

Se usaron en la composición los tipos  
Calisto Mt 11/15 pts.



\* D - 7 0 1 7 0 0 \*

