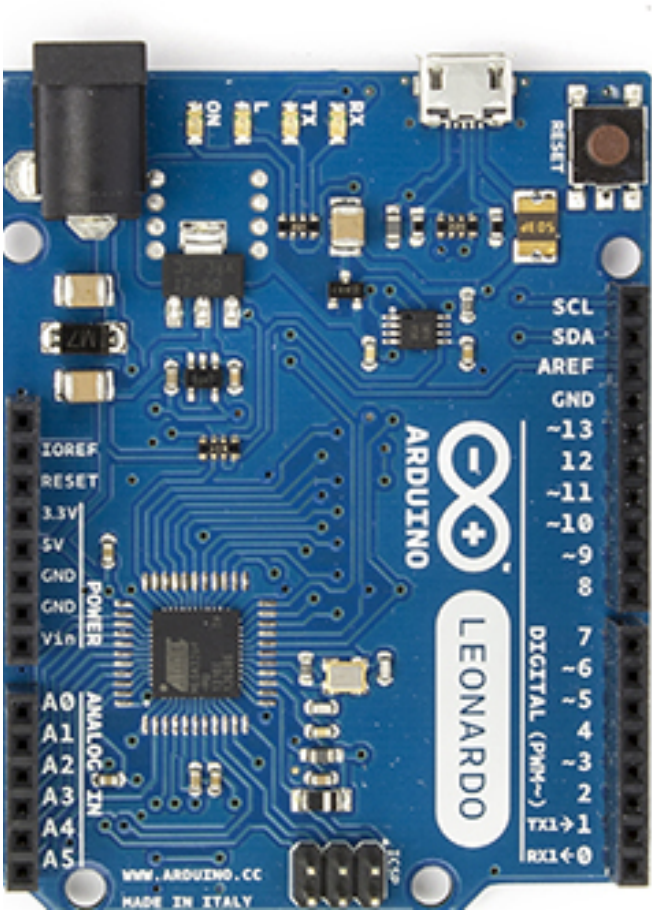
Arduino



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

PRESENTACIÓN

Arduino es una plataforma desarrollo electrónico open source basado en un software y hardware flexible y fácil de usar. Arduino puede percibir el entorno en el que se ejecuta mediante la lectura de una variedad de sensores y puede interactuar con el entorno utilizando luces, control de motores, y otros actuadores.

El curso se desarrollará íntegramente sobre Arduino para obtener sistemas autónomos o que se puedan comunicar con el software que se ejecute dentro en un ordenador (por ejemplo, Processing, Matlab, o LabVIEW) o dentro de un telefono. Se enseñará a utilizar las hojas de datos de todos los componentes para trabajar de forma óptima, y al mismo tiempo enseñará los conceptos fundamentales de electrónica para iniciarse en el desarrollo hardware e introducirse en el diseño de circuitos impresos utilizando la plataforma Fritzing.

Objetivos General:

Que los alumnos se introduzcan al desarrollo de software embebido sobre la plataforma Arduino, y una introducción básica de electrónica que les permita avanzar en nuevos desarrollos.

Objetivos específicos:

Que los participantes:

- Comprendan y aprendan a desarrollar sobre lenguaje Arduino.
- Comprendan y logren interactuar con el mundo externo utilizando sensores y actuadores.
- Comprendan y logren comunicarse con la computadora y otros Arduinos utilizando el puerto serial.
- Logren una comprensión mínima de electrónica fundamental: Ley de Ohm, Leyes de Kirchhoff y conozcan los instrumentos básicos como Multímetros y Osciloscopios.

Destinatarios:

Esta dirigido al público general iimrenteresado en introducirse en el desarrollo electrónico, automatización, adquisición de datos, domótica y sistemas de control, artistas, diseñadores, y hobbistas que quieran construir objetos interactivos utilizando esta plataforma. También para futuros estudiantes de carreras de grado como Ing. Electrónica.

Duración:

12 clases de 3 horas, totalizando 36 horas, con una carga horaria semanal de 3hs.

Docente:

- Matias De lellis: Estudiante de Ingeniería Electrónica, desarrollador de Huayra GNU/Linux - Conectar Igualdad.

Correo:

- mati86dl@gmail.com

Grupo de Discusión:

- https://groups.google.com/d/forum/curso-arduino-sceu-utn-frba-2016-2

TEMARIO

Módulo 1: Introducción a la plataforma

Clase 1: Introducción

- Presentación de la Plataforma.
- Descripción del hardware de Arduino UNO.
- Instalación del entorno Arduino IDE.
- Funciones, descripción de un Sketch.
- Primeros ejemplos:
  - Parpadear el led integrado sobre el pin 13
  - Desarrollo de un semáforo
- Variables integer.

Clase 2: Funciones y variables

- Funciones definidas por el usuario.
- Sentencia de control If()
- Tipo de variables
  - void, boolean, char, byte, int, word, long, float, y double.
- Alcance variable y modificadores.
- static, volatile, cont.
- Aritmética de variables.
  - =, +, *, / y %.
- Operadores compuestos
  - ++, --, +=, -=, *= y /=.

Clase 3: Sentencias de control

- If Statement
- For Loop
- Introducción del array.
- Formalización de la Ley de Ohm para cuidar los Leds.
- While Loop
- Switch Case
- Introducción del puerto serial.

Clase 4: Puertos analógicos

- Lectura de tensión sobre un potenciómetro y actuando obre un led.
- Introducción de la Segunda Ley de Kirchhoff.
- Lectura de de temperatura usando un LM35, y encender el led.
- Modificar la iluminación de un Led según el puerto serial.
- Controlar la velocidad de un motor.
- Ejemplo práctico:
  - Controlar la velocidad de un ventilador según la temperatura ambiente, y mostrar la temperatura en la computadora.

Módulo 2: Comunicación, sensores y actuadores:

Clase 5:

- Display LCD de 2x16.
- Teclado matricial.
- Tarjetas RFID

Clase 6: Comunicación inalámbrica

- Bluetooth HC 06
- Xbee.
- GSM

Clase 7: Actuadores

- Servo.
- Motores Brushless
- Acelerómetro y giroscopio.

Clase 8: Proyecto ejemplo (Alarma domiciliaria)

- Configuración vía USB.
- Activación usando teclado matricial.
- Sensores Magnéticos
- Alarma, luces, y envío de mensaje SMS en caso de intromisión.

Módulo 3: Desarrollo Avanzado

Clase 9: Arduino en Protoboard.

- Desarrollo de circuitos compatibles con Arduino en protoboard.
- Instalación del Bootloader utilizando el puerto ICSP.
- Carga de sketchs utilizando adaptadores USB SERIAL.

Clase 10: Introducción al diseño de PCB en Fritzing.

- Diseño de un PCB del proyecto: Alarma domiciliaria.
- Transferencia y revelado.
- Soldado de componentes.

Clase 11:

- Numeración binaria.
- Operaciones sobre bits.
- Memoria Flash.

Clase 12:

- Presentación de Hojas de datos Atmega328p
- AVR Studio.
  - Desarrollo ejemplo blink.c
- Carga de ejecutable usando ICSP.
- Comparación de los ejecutables entre código Arduino y C.
- Comparación del resultado utilizando un Osciloscopio.

INTRODUCCIÓN

¿Qué es Arduino?

Arduino es una plataforma de prototipos de electronica de código abierto basado en hardware y software fácil de usar. Las placas Arduino son capaces de leer entradas (la luz en un sensor, la pulsación de un botón, o un mensaje de Twitter) y convertirla en una salida (la activación de un motor, encender un LED, publicar algo en línea). Todo esto se define por un conjunto de instrucciones programadas a través del Software Arduino (IDE).

A través de los años Arduino ha sido el cerebro de miles de proyectos. Una comunidad mundial de los fabricantes (estudiantes, aficionados, artistas, programadores y profesionales) se ha reunido alrededor de esta plataforma de código abierto, y sus contribuciones han añadido una increíble cantidad de conocimiento accesible que puede ser de gran ayuda tanto para los principiantes como para expertos.

Arduino nació en Italia en instituto de diseño IVREA como una herramienta de prototipado fácil, rápida, dirigido a estudiantes sin experiencia en electrónica ni programación. Tan pronto como llegó a una comunidad más amplia, la placa Arduino comenzó a cambiar para adaptarse a las nuevas necesidades y desafíos, diferenciándose de otras placas de 8 bits mas simples con productos para aplicaciones de la IoT (Internet of Things), impresión 3D, wearable (vestibles) y entornos embebidos. Todas las placas Arduino son completamente de código abierto, permitiendo a los usuarios crear de forma independiente y, finalmente, adaptarlos a sus necesidades particulares. El software también es de código abierto, y está creciendo a través de las aportaciones de los usuarios en todo el mundo.

¿Porque Arduino?

Gracias a su sencilla y accesible experiencia de usuario, Arduino se ha utilizado en miles de diferentes proyectos y aplicaciones. El software de Arduino es fácil de usar para los principiantes, pero lo suficientemente flexible para los usuarios avanzados. Se ejecuta en Mac, Windows y Linux. Los profesores y los estudiantes lo utilizan para construir los instrumentos científicos de bajo coste, para demostrar los principios de química y física, o para iniciarse en la programación y la robótica. Diseñadores y arquitectos construyen prototipos interactivos, músicos y artistas lo utilizan para instalaciones y experimentar con nuevos instrumentos musicales. Los fabricantes, por supuesto, lo utilizan para construir muchos de los proyectos expuestos en la Maker Faire, por ejemplo. Arduino es una herramienta clave para aprender cosas nuevas. Cualquier persona - niños, aficionados, artistas, programadores - puede comenzar a juguetear simplemente siguiendo paso a paso las instrucciones de un kit, o compartir ideas en línea con otros miembros de la comunidad de Arduino.

Hay muchos otros microcontroladores y plataformas de microcontroladores disponibles para computación física. Parallax Basic Stamp, de Netmedia BX-24, Phidgets, Handyboard del MIT, y muchos otros ofrecen una funcionalidad similar. Todas estas herramientas toman los detalles sucios de programación de microcontroladores y lo envuelve en un paquete fácil de usar. Arduino de la misma forma simplifica el proceso de trabajar con los microcontroladores, pero ofrece algunas ventajas para los profesores, estudiantes y aficionados interesados sobre otros sistemas:

- Accesibles: Las placas Arduino son relativamente baratos en comparación con otras plataformas de microcontroladores. La versión menos costosa del módulo Arduino puede ser montado a mano, e incluso los módulos de Arduino premontados cuestan menos de $100
- Multiplataforma: El software de Arduino (IDE) se ejecuta en los sistemas operativos Windows, Macintosh OSX y GNU/Linux. La mayoría de los sistemas de microcontroladores se limitan a Windows.
- Ambiente de programación limpio y simple: El software de Arduino (IDE) es fácil de usar para los principiantes, pero lo suficientemente flexible para aprovecharlo también por los usuarios avanzados. Para los profesores, se basa convenientemente en el entorno de programación Processing (Lenguaje muy utilizado en diseño), por lo que los estudiantes que aprenden a

programar en ese entorno estarán familiarizados sobre cómo funciona el Arduino IDE.

- Software de código abierto y extensible: El software de Arduino está publicado como herramientas de código abierto, disponible para la extensión por programadores experimentados. El lenguaje se puede ampliar a través de bibliotecas C++, y la gente con ganas de entender los detalles técnicos pueden dar el salto de Arduino para el lenguaje de programación AVR C en el que se basa. Del mismo modo, puede agregar código AVR-C directamente en sus programas de Arduino si quieres.
- Hardware de código abierto y el extensible: Los planos de las placas Arduino se publican bajo una licencia de Creative Commons, por lo que los diseñadores de circuitos experimentados pueden hacer su propia versión del módulo, ampliándolo y mejorándolo. Incluso los usuarios con poca experiencia pueden construirla con el fin de entender cómo funciona y ahorrar dinero.

CONCEPTOS SOBRE ELECTRICIDAD

¿Qué es la electricidad?

Un electrón es una partícula subatómica que posee carga eléctrica negativa. Por lo tanto, debido a la ley física de atracción entre sí de cargas eléctricas de signo opuesto (y de repulsión entre sí de cargas eléctricas de mismo signo), cualquier electrón siempre es atraído por una carga positiva equivalente. Una consecuencia de este hecho es que si, por razones que no estudiaremos, en un extremo (también llamado “polo”) de un material conductor aparece un exceso de electrones y en el otro polo aparece una carencia de estos (equivalente a la existencia de “cargas positivas”), los electrones tenderán a desplazarse a través del conductor desde el polo negativo al positivo. A esta circulación de electrones por un material conductor se le llama “electricidad”.

¿Qué es el voltaje?

En el estudio del fenómeno de la electricidad existe un concepto fundamental que es el de voltaje entre dos puntos de un circuito eléctrico (también llamado “tensión”, “diferencia de potencial” o “caída de potencial”). Expliquémoslo con un ejemplo.

Si entre dos puntos de un conductor no existe diferencia de cargas eléctricas, el voltaje entre ambos puntos es cero. Si entre esos dos puntos aparece un desequilibrio de cargas (es decir, que en un punto hay un exceso de cargas negativas y en el otro una ausencia de ellas), aparecerá un voltaje entre ambos puntos, el cual será mayor a medida que la diferencia de cargas sea también mayor. Este voltaje es el responsable de la generación del flujo de electrones entre los dos puntos del conductor. No obstante, si los dos puntos tienen un desequilibrio de cargas entre sí pero están unidos mediante un material no conductor (lo que se llama un material “aislante”), existirá un voltaje entre ellos pero no habrá paso de electrones (es decir, no habrá electricidad).

Generalmente, se suele decir que el punto del circuito con mayor exceso de cargas positivas (o dicho de otra forma: con mayor carencia de cargas negativas) es el que tiene el “potencial” más elevado, y el punto con mayor exceso de cargas negativas es el que tiene el “potencial” más reducido. Pero no olvidemos nunca que el voltaje siempre se mide entre dos puntos: no tiene sentido decir “el voltaje en este punto”, sino “el voltaje en este punto respecto a este otro”; de ahí sus otros nombres de “diferencia de potencial” o “caída de potencial”.

También por convención (aunque físicamente sea en realidad justo al contrario) se suele decir que la corriente eléctrica va desde el punto con potencial mayor hacia otro punto con potencial menor (es decir, que la carga acumulada en el extremo positivo es la que se desplaza hacia el extremo negativo).

Para entender mejor el concepto de voltaje podemos utilizar la analogía de la altura de un edificio: si suponemos que el punto con el potencial más pequeño es el suelo y asumimos este como el punto de referencia con valor 0, a medida que un ascensor vaya subiendo por el edificio irá adquiriendo más y más potencial respecto el suelo: cuanta más altura tenga el ascensor, más diferencia de potencial habrá entre este y el suelo. Cuando estemos hablando de una “caída de potencial”, querremos decir entonces (en nuestro ejemplo) que el ascensor ha disminuido su altura respecto al suelo y por tanto tiene un voltaje menor.

La unidad de medida del voltaje es el voltio (V), pero también podemos hablar de milivoltios (1 mV = 0,001 V), o de kilovoltios (1 kV = 1000 V). Los valores típicos en proyectos de electrónica casera como los que abordaremos en este libro son de 1,5V, 3,3V, 5V... aunque cuando intervienen elementos mecánicos (como motores) u otros elementos complejos, se necesitará aportar algo más de energía al circuito, por lo que los valores suelen algo mayores: 9V, 12V o incluso 24V.

¿Qué es la intensidad de corriente?

La intensidad de corriente (comúnmente llamada “corriente” a secas) es una magnitud eléctrica que se define como la cantidad de carga eléctrica que pasa en un determinado tiempo a través de un punto concreto de un material conductor.

Podemos imaginar que la intensidad de corriente es similar en cierto sentido al caudal de agua que circula por una tubería: que pase más o menos cantidad de agua por la tubería en un determinado tiempo sería análogo a que pase más o menos cantidad de electrones por un cable eléctrico en ese mismo tiempo.

Su unidad de medida es el amperio (A), pero también podemos hablar de miliamperios (1 mA = 0,001 A), de microamperios (1 µA = 0,001 mA), o incluso de nanoamperios (1 nA = 0,001 µA).

¿Qué es la resistencia eléctrica?

Podemos definir la resistencia eléctrica interna de un objeto cualquiera (aunque normalmente nos referiremos a algún componente electrónico que forme parte de nuestros circuitos) como su capacidad para oponerse al paso de la corriente eléctrica a través de él. Es decir, cuanto mayor sea la resistencia de ese componente, más dificultad tendrán los electrones para atravesarlo, hasta incluso el extremo de imposibilitar la existencia de electricidad.

Esta característica depende entre otros factores del material con el que está construido ese objeto, por lo que podemos encontrarnos con materiales con poca o muy poca resistencia intrínseca (los llamados “conductores”, como el cobre o la plata) y materiales con bastante o mucha resistencia (los llamados “aislantes”, como la madera o determinados tipos de plástico, entre otros).

La unidad de medida de la resistencia de un objeto es el ohmio (Ω). También podemos hablar de kilohmios (1 kΩ = 1000 Ω), de megaohmios (1 MΩ = 1000 kΩ), etc.

¿Qué es la Ley de Ohm?

La Ley de Ohm dice que si un componente eléctrico con resistencia interna, R, es atravesado por una intensidad de corriente, I, entre ambos extremos de dicho componente existirá una diferencia de potencial, V, que puede ser conocida gracias a la relación $V = I \cdot R$.

De esta fórmula es fácil deducir relaciones de proporcionalidad interesantes entre estas tres magnitudes eléctricas. Por ejemplo: se puede ver que (suponiendo que la resistencia interna del componente no cambia) cuanto mayor es la intensidad de corriente que lo atraviesa, mayor es la diferencia de potencial entre sus extremos.

También se puede ver que (suponiendo en este caso que en todo momento circula la misma intensidad de corriente por el componente), cuanto mayor es su resistencia interna, mayor es la diferencia de potencial entre sus dos extremos.

Además, despejando la magnitud adecuada de la fórmula anterior, podemos obtener, a partir de dos datos conocidos cualesquiera, el tercero. Por ejemplo, si conocemos V y R, podremos encontrar I mediante $I = V/R$, y si conocemos V e I, podremos encontrar R mediante $R = V/I$.

A partir de las fórmulas anteriores debería ser fácil ver también por ejemplo que cuanto mayor es el voltaje aplicado entre los extremos de un componente (el cual suponemos que posee una resistencia de valor fijo), mayor es la intensidad de corriente que pasa por él. O que cuanto mayor es la resistencia del componente (manteniendo constante la diferencia de potencial entre sus extremos), menor es la intensidad de corriente que pasa a través de él.

¿Qué son las señales digitales y las señales analógicas?

Podemos clasificar las señales eléctricas (ya sean voltajes o intensidades) de varias maneras según sus características físicas. Una de las clasificaciones posibles es distinguir entre señales digitales y señales analógicas.

Señal digital es aquella que solo tiene un número finito de valores posibles (lo que se suele llamar “tener valores discretos”). Por ejemplo, si consideramos como señal el color emitido por un semáforo, es fácil ver que esta es de tipo digital, porque solo puede tener tres valores concretos, diferenciados y sin posibilidad de transición progresiva entre ellos: rojo, amarillo y verde.

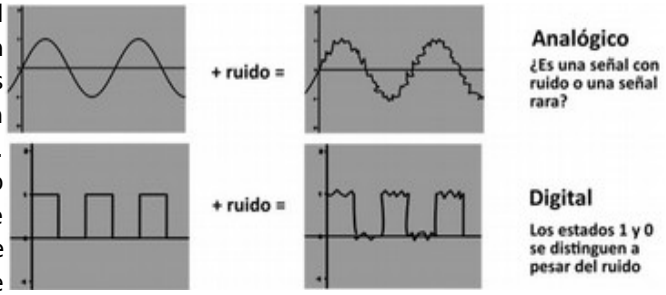
Un caso particular de señal digital es la señal binaria, donde el número de valores posibles solo es 2. Conocer este tipo de señales es importante porque en la electrónica es muy habitual trabajar con voltajes (o intensidades) con tan solo dos valores. En estos casos, uno de los valores del voltaje binario suele ser 0 –o un valor aproximado– para indicar precisamente la ausencia de voltaje, y el otro valor puede ser cualquiera, pero lo suficientemente distinguible del 0 como para indicar sin ambigüedades la presencia de señal. De esta forma, un valor del voltaje binario siempre identifica el estado “no pasa corriente” (también llamado estado “apagado” –“off” en inglés– , BAJO –LOW en inglés–, o “0”) y el otro valor siempre identifica el estado “pasa corriente” (también llamado “encendido” –“on” – , ALTO –HIGH – , o “1”).

El valor de voltaje concreto que se corresponda con el estado ALTO será diferente según los dispositivos electrónicos utilizados en cada momento. En los proyectos que utilizaremos en el curso, por ejemplo, será habitual utilizar valores de 5 V ó 3,3 V. Pero atención: es importante tener en cuenta que si sometemos un dispositivo electrónico a un voltaje demasiado elevado (por ejemplo, si aplicamos 5V como valor ALTO cuando el dispositivo solo admite 3,3V) corremos el riesgo de dañarlo irreversiblemente.

Señal analógica es aquella que tiene infinitos valores posibles dentro de un rango determinado (lo que se suele llamar “tener valores continuos”). La mayoría de magnitudes físicas (temperatura, sonido, luz...) son analógicas, así como también las más específicamente eléctricas (voltaje, intensidad, potencia...) porque todas ellas, de forma natural, pueden sufrir variaciones continuas sin saltos.

No obstante, muchos sistemas electrónicos (un computador, por ejemplo) no tienen la capacidad de trabajar con señales analógicas: solamente pueden manejar señales digitales (especialmente de tipo binario; de ahí su gran importancia). Por tanto, necesitan disponer de un conversor analógico-digital que “traduzca” (mejor dicho, “simule”) las señales analógicas del mundo exterior en señales digitales entendibles por dicho sistema electrónico. También se necesitará un conversor digital-analógico si se desea realizar el proceso inverso: transformar una señal digital interna del computador en una señal analógica para poderla así emitir al mundo físico. Un ejemplo del primer caso sería la grabación de un sonido mediante un micrófono, y uno del segundo caso sería la reproducción de un sonido pregrabado mediante un altavoz.

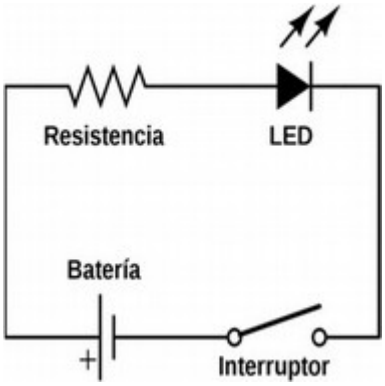
Sobre los métodos utilizados para realizar estas conversiones de señal analógica a digital, y viceversa, ya hablaremos extensamente más adelante, pero lo que debemos saber ya es que, sea cual sea el método utilizado, siempre existirá una pérdida de información (de “calidad”) durante el proceso de conversión de la señal.



A pesar de lo anterior, la razón por la cual la mayoría de sistemas electrónicos utilizan para funcionar señales digitales en vez de analógicas es porque las primeras tienen una gran ventaja respecto las segundas: son más inmunes al ruido. Por “ruido” se entiende cualquier variación no deseada de la señal, y es un fenómeno que ocurre constantemente debido a una gran multitud de factores. El ruido modifica la información que aporta una señal y afecta en gran medida al correcto funcionamiento y rendimiento de los dispositivos electrónicos. Si la señal es analógica, el ruido es mucho más difícil de tratar y la recuperación de la información original se complica.

REPRESENTACIÓN GRÁFICA DE CIRCUITOS

Un circuito básico



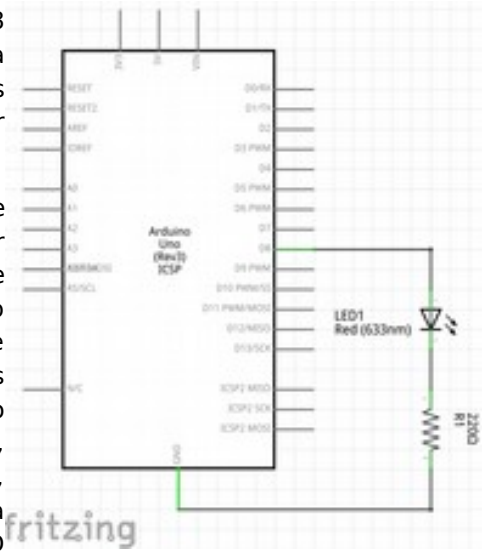
Para describir de una forma sencilla y clara la estructura y la composición de un circuito eléctrico se utilizan esquemas gráficos. En ellos se representa cada dispositivo del circuito mediante un símbolo estandarizado y se dibujan todas las conexiones existentes entre ellos. Por ejemplo un circuito muy simple sería el siguiente.

En este esquema podemos apreciar cuatro dispositivos (presentes prácticamente en cualquier circuito) representados por su símbolo convencional: una pila o batería (cuya tarea es alimentar eléctricamente al resto de componentes), una resistencia (componente específicamente diseñado para oponerse al paso de la corriente, de ahí su nombre), un LED (componente que se ilumina cuando recibe corriente) y un interruptor.

En este ejemplo, la batería creará la diferencia de potencial necesaria entre sus dos extremos –también llamados “bornes” o “polos”– para que se genere una corriente eléctrica, la cual surgirá desde su polo positivo (marcado con el signo “+”), pasará a través de la resistencia, pasará seguidamente a través del LED (Iluminándolo) y llegará a su destino final (el polo negativo de la batería) siempre y cuando el interruptor cierre el circuito.

Ejemplo Arduino

Así es que llegamos a nuestro primer ejemplo de la utilización de Arduino y donde redondearemos varios de los conceptos discutidos anteriormente.

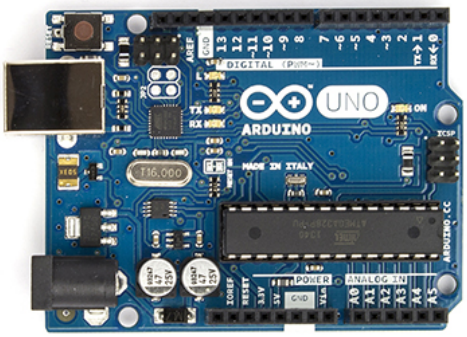


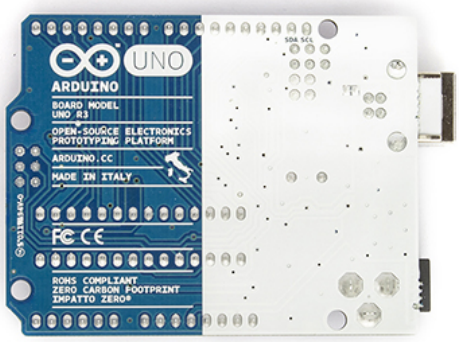
En este esquema podemos apreciar ahora 3 componentes, el LED que pretendemos iluminar, la resistencia para limitar la intensidad de corriente, y así proteger los componentes, y la placa Arduino que actuara como controlador del circuito.

Aunque no lo parezca, este circuito es esencialmente exactamente igual al anterior. La batería necesaria para alimentar eléctricamente el circuito se encuentra implícitamente dentro de la placa Arduino, y de la misma forma, el interruptor encargado de cerrar el circuito para encender el LED se encuentra dentro de la placa. Por supuesto, este no es un botón físico, y solo estamos haciendo una analogía. La forma de actuar sobre este supuesto interruptor, es darle las instrucciones correctas a la placa Arduino, para que sobre su salida D4 haya un potencial de 5V, promoviendo una diferencia de potencial, la cual implica una intendencia de corriente, y en consecuencia se iluminara el LED exactamente cuando nosotros queremos.

Sin entrar en detalles que si profundizamos mas adelante, así debemos pensar a Arduino, su utilidad más importante es interactuar con su entorno físico. Esta placa recibe información a través de sus entradas -Sensores, como un simple botón, o sensores de temperatura, humedad o otros-, analiza la información y toma ciertas decisiones -Según las instrucciones que tenga programada-, y finalmente realiza ciertas tareas a través sus actuadores como en este caso un led -Pueden ser motores, parlantes, display, etc-.

ARDUINO UNO: Resumen de la plataforma





Descripción

El Arduino Uno es una placa electrónica basada en el ATmega328p. Cuenta con 14 pines digitales de entrada/salida (de los cuales 6 pueden utilizarse para salidas PWM), 6 entradas analógicas, un resonador cerámico 16 MHz, una conexión USB, un conector de alimentación, un conector ICSP, y un botón de reinicio. Contiene todo lo necesario para apoyar el microcontrolador. Basta con conectarlo a un ordenador con un cable USB o a un adaptador de corriente alterna a corriente continua para empezar.

"Uno" valga la redundancia de hablar en castellano, significa “Uno” en italiano y se nombro así para conmemorar el lanzamiento de Arduino 1.0. El Uno y la versión 1.0 del entorno de desarrollo y del lenguaje, son las versiones de referencia de toda la plataforma Arduino.

Microcontrolador	ATmega328p
Voltaje de operación	5V
Voltaje de entrada (Recomendado)	7-12V
Voltaje de entrada (Limites)	6-20V
Pines Digitales I/O	14 (6 proveen salidas PWM)
Pines de entrada Analógicas	6
Corriente máxima de Pines I/O	40 mA
Corriente máxima del pin 3.3V	50 mA
Memoria Flash	32 KB (ATmega328) con 0.5 KB usados por el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad del Reloj.	16 MHz
Longitud	68.6 mm
Ancho	53.4 mm
Peso	25 g

Esquemáticos y Diseños de referencia

La placa Arduino es hardware libre porque sus ficheros esquemáticos están disponibles para descargar de la página web del proyecto con la licencia Creative Commons Attribution Share-Alike (http://es.creativecommons.org/licencia), la cual es una licencia libre que permite realizar trabajos derivados tanto personales como comerciales (siempre que estos den crédito a Arduino y publiquen sus diseños bajo la misma licencia). Así pues, uno mismo se puede construir su propia placa Arduino “a mano”, o comprarlas en un distribuidor preensambladas y listas para usar.

Alimentación

El Arduino Uno puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente.

La alimentación externa (no USB) puede venir con un adaptador de CA a CC o una batería. El adaptador se puede conectar con un enchufe de centro-positivo de 2.1mm sobre el conector de alimentación de la placa. Los cables de una batería también se pueden insertar en los cabezales de pin GND y Vin del conector de alimentación.

La placa puede funcionar con un suministro externo de 6 a 20 voltios. Si se suministra con menos de 7V, sin embargo, el pin de 5V puede suministrar menos de 5 V y la placa puede ser inestable. También si se utiliza más de 12 V, el regulador de voltaje puede sobrecalentarse y dañar la placa. Entonces el rango recomendado de alimentación es de 7 a 12 voltios.

Los pines de alimentación son como sigue:

- VIN: El voltaje de entrada a la placa Arduino cuando se utiliza una fuente de alimentación externa (en oposición a 5 voltios de la conexión USB u otra fuente de alimentación regulada). Usted puede suministrar tensión a través de este pin, o, si el suministro de tensión es a través de la toma de alimentación, puede acceder a él a través de este pin.
- 5V: Este pin provee los 5V regulados en la placa. La placa puede ser alimentada ya sea desde la toma de alimentación de CC (7 - 12 V), el conector USB (5V), o por este pin VIN (7-12V). El suministro de tensión a través de los pines de 5V o 3.3V no utiliza el regulador, y puede dañar la placa. No se aconseja hacer ello.
- 3V3: Un suministro de 3,3 voltios generada por el regulador de la placa. Provee una corriente máxima es de 50 mA.
- GND: Pines de tierra.
- IOREF: Este pin de la placa Arduino proporciona la referencia de tensión con la que opera el microcontrolador. Un shield conectado a la placa puede leer el voltaje del pin IOREF y seleccionar la fuente de alimentación adecuada para comunicarse correctamente con el microcontrolador.

Memoria

El ATmega328 tiene 32 KB (con 0,5 KB utilizado por el gestor de arranque). También cuenta con 2 KB de SRAM y 1 KB de EEPROM que se puede leer y escribir con la librería EEPROM.

Entradas y Salidas

Cada uno de los 14 pines digitales en el Uno se puede utilizar como una entrada o salida, utilizando pinMode (), digitalWrite (), y las funciones digitalRead (). Funcionan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia de pull-up (desconectada por defecto) de 20 a 50 kOhm. Además algunos pines tienen funciones especializadas:

- Serial: 0 (RX) y 1 (TX): Se utiliza para recibir (RX) y transmitir datos en serie (TX) TTL. Estos se encuentran conectadas a los pines correspondientes de la USB-to-TTL chips Serial ATmega8U2.
- Interrupciones externas: 2 y 3: Estos pines pueden configurarse para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor.
- PWM: 3, 5, 6, 9, 10, y 11: Pueden proporcionar una salida PWM de 8 bits con la función analogWrite ().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK): Estos pines soportan la comunicación SPI utilizando la librería SPI.
- LED: 13: Hay un LED incorporado conectado al pin digital 13. Cuando el pin se encuentra en un estado alto (5V), el LED se mantiene encendido, cuando el pin esta en estado bajo, el LED se

mantiene apagado.

- 6 entradas analógicas, etiquetadas como A0 a A5, cada uno de los cuales proporcionan 10 bits de resolución (es decir, 1.024 valores diferentes). Por defecto se miden desde 0 V (Tierra) a 5 V, aunque es posible cambiar el extremo superior de su rango usando el pin AREF y la función analogReference (). Además, algunos de estos pines tienen funciones especializadas:
- TWI: pin A4 o A5 o SDA y SCL pin: Comunicación TWI / I2C utilizando la librería Wire.
- Hay un par de pines extras en la placa:
- AREF: Es una entrada de para proveer el voltaje de referencia superior para las entradas analógicas. Se utiliza con analogReference ().
- Reset: Debe colocar este pin en un estado bajo (0 V o Tierra) para reiniciar el microcontrolador. Normalmente se utiliza para añadir un botón de reinicio para shields que tapan el botón de la placa original.

Comunicación

El Arduino Uno tiene varias facilidades para comunicarse con un ordenador, otro Arduino u otros microcontroladores. El ATmega328 ofrece UART TTL (5V) de comunicación en serie, que está disponible en los pines digitales 0 (RX) y 1 (TX). Un chip ATmega16U2 se encuentra conectado a estos pines, y provee una comunicación en serie a través de USB que aparece como un puerto COM virtual nuestra computadora. El "firmware" del chip 16U2 utiliza los controladores USB COM estándar, y no se necesita ningún controlador externo. Sin embargo, en Windows, es necesario un archivo .inf.

El entorno de Arduino incluye un monitor serial que nos permite enviar y recibir datos a la placa Arduino. Los LEDs RX y TX en la placa parpadean cuando se están transmitiendo datos a través del chip y el USB de la computadora, pero no para la comunicación en serie en los pines 0 y 1.

Una biblioteca SoftwareSerial permite la comunicación en serie en cualquiera de los pines digitales del Uno. Sin embargo al hacerse a nivel de software tiene algunas limitaciones.

El ATmega328 también también puede comunicarse utilizando I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso del bus I2C. Para la comunicación SPI, se utiliza la biblioteca de SPI.

Programación

El Arduino Uno se puede programar con el software de Arduino (descargar). Seleccione "Arduino Uno" desde el menú *Tools* > *Boards* (de acuerdo con el microcontrolador en su tablero).

Los ATmega328 en la Arduino Uno viene prequemado con un gestor de arranque que le permite cargar nuevo código a él sin el uso de un programador de hardware externo que se requiere normalmente.

También puede pasar por alto el gestor de arranque y programar el microcontrolador a través del ICSP (In-Circuit Serial Programming) header y un programador ICSP estándar, lo cual eliminaría el gestor de arranque que viene por defecto.

Protección USB

El Arduino Uno tiene una polyfusible reiniciable que protege a los puertos USB de su ordenador contra cortocircuitos o sobrecorriente. Aunque la mayoría de las computadoras ofrecen su propia protección interna, el fusible proporciona una capa adicional de protección. Si se aplican más de 500 mA al puerto USB, el fusible automáticamente rompe la conexión hasta que se elimina el cortocircuitos o se elimina la sobrecarga.

COMENZANDO CON ARDUINO

Nuestro primer ejemplo

A modo de introducción al lenguaje Arduino desarrollaremos nuestro primer ejemplo. Este es parpadear un led que todas las placas Arduino tienen conectados al pin 13 de forma integrado. De esta forma no requeriremos Hardware externo, ni grandes conocimientos de electrónica para introducirnos en el lenguaje.

Estructura de un sketch en Arduino

Un programa de Arduino se denomina “sketch” y son archivos que tienen la extensión .ino. La estructura básica del lenguaje de programación Arduino es muy sencilla y se organiza en al menos dos funciones que encierran bloques de declaraciones.

```
void setup ()
{
      statements;
}
void loop ()
{
      statements;
}
```

setup ()

Es la primer función que se ejecuta cuando se alimenta Arduino y lo hace por única vez hasta que se reinicie la placa. Dentro de ella se deben configurar cualquier elemento que sea necesario para el funcionamiento de nuestro programa.

Recordemos que dijimos que cualquiera de los pines digitales puede funcionar como una entrada o salida, y en nuestro caso, tenemos que indicarle a la placa que el pin donde esta conectado el LED que queremos iluminar sera utilizado como una salida.

```
void setup ()
{
      // Ajusta el modo del 'pin 13' como salida.
      pinMode (13, OUTPUT);
}
```

loop ()

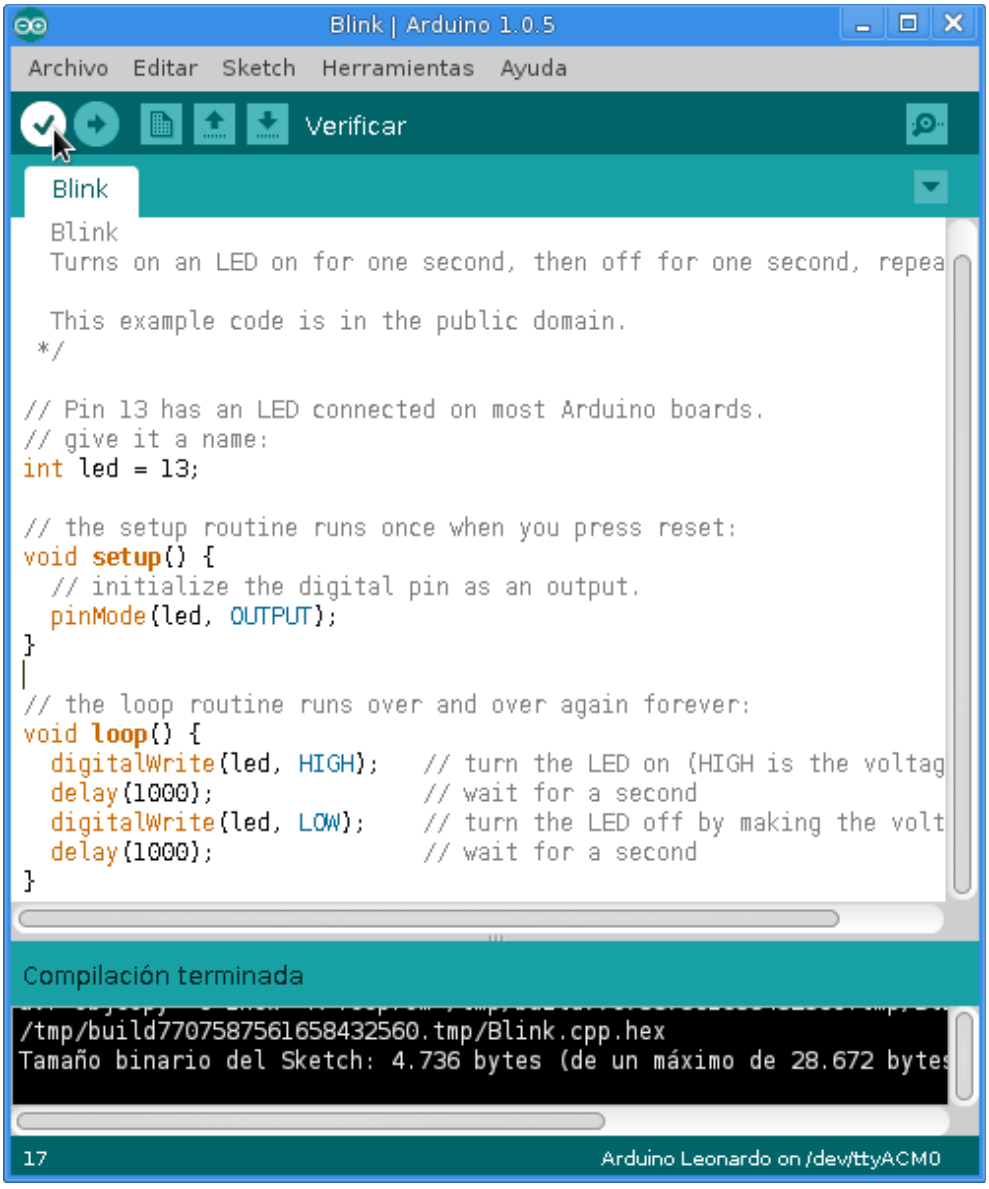
La función loop se ejecuta a continuación e incluye el código que se ejecuta continuamente. Por lo general, leyendo entradas, procesando datos y activando salidas. Así, esta función es el núcleo de todos los programas Arduino y la que hace la mayor parte del trabajo.

En nuestro caso, necesitamos activar la salida, (que como dijimos coloca una diferencia de potencial en el pin que seleccionamos, generando una intensidad de corriente que al atravesar el diodo led lo iluminara), esperar un segundo, desactivar la salida, y luego esperar otro segundo.

```
void loop ()
{
      digitalWrite (13, HIGH);  // Activa 'Pin'
      delay (1000)              // Es pera un segundo.
      digitalWrite (13,  LOW);  // Desactiva 'Pin'
      delay (1000)              // Es pera un segundo.
}
```

Esta función se ejecuta continuamente, y así luego de esperar el ultimo segundo, El microcontrolador sale de esta función, y vuelve a ejecutarla nuevamente, volviendo a titilar mientras se encuentre alimentado a la placa.

Este ejemplo, es considerado el mas básico para explicar el desarrollo sobre cualquier plataforma de microcontroladores, y se encuentra integrado como ejemplo en el entorno de Arduino. De esta forma, debemos ejecutar el entorno Arduino, vamos al menú *File > Sketchbook > Examples > Digital > Blink.*



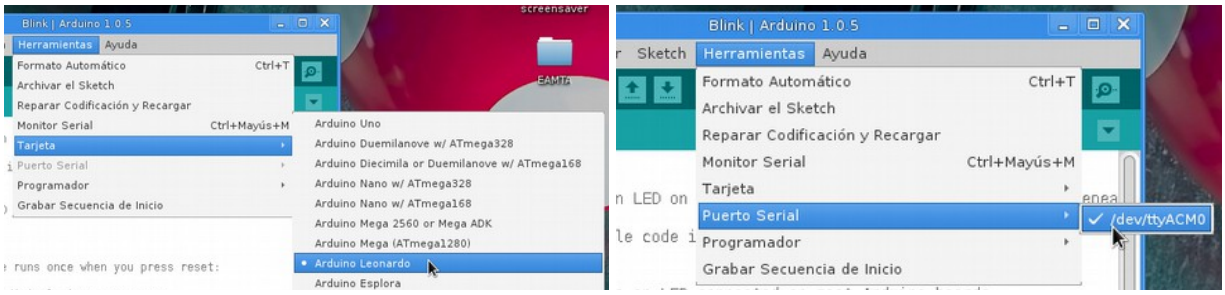
Subiendo el ejemplo a la placa

Para el ejemplo que desarrollamos estamos haciendo parpadear un Led que realmente se encuentra integrado a la placa Arduino, y por lo tanto no necesitamos agregar ningún Hardware externo.

Primero debemos comprobar que el código escrito es correcto, y para ello tenemos que hacer clic en el primer boton de la barra de herramientas. Si escribimos el código correctamente, veremos en la parte inferior de la aplicación un mensaje indicando que la compilación fue terminada como indica la imagen anterior. Si hemos cometido un error veremos un mensaje indicando primero el Error de compilación, y abajo una descripción del error que cometimos,

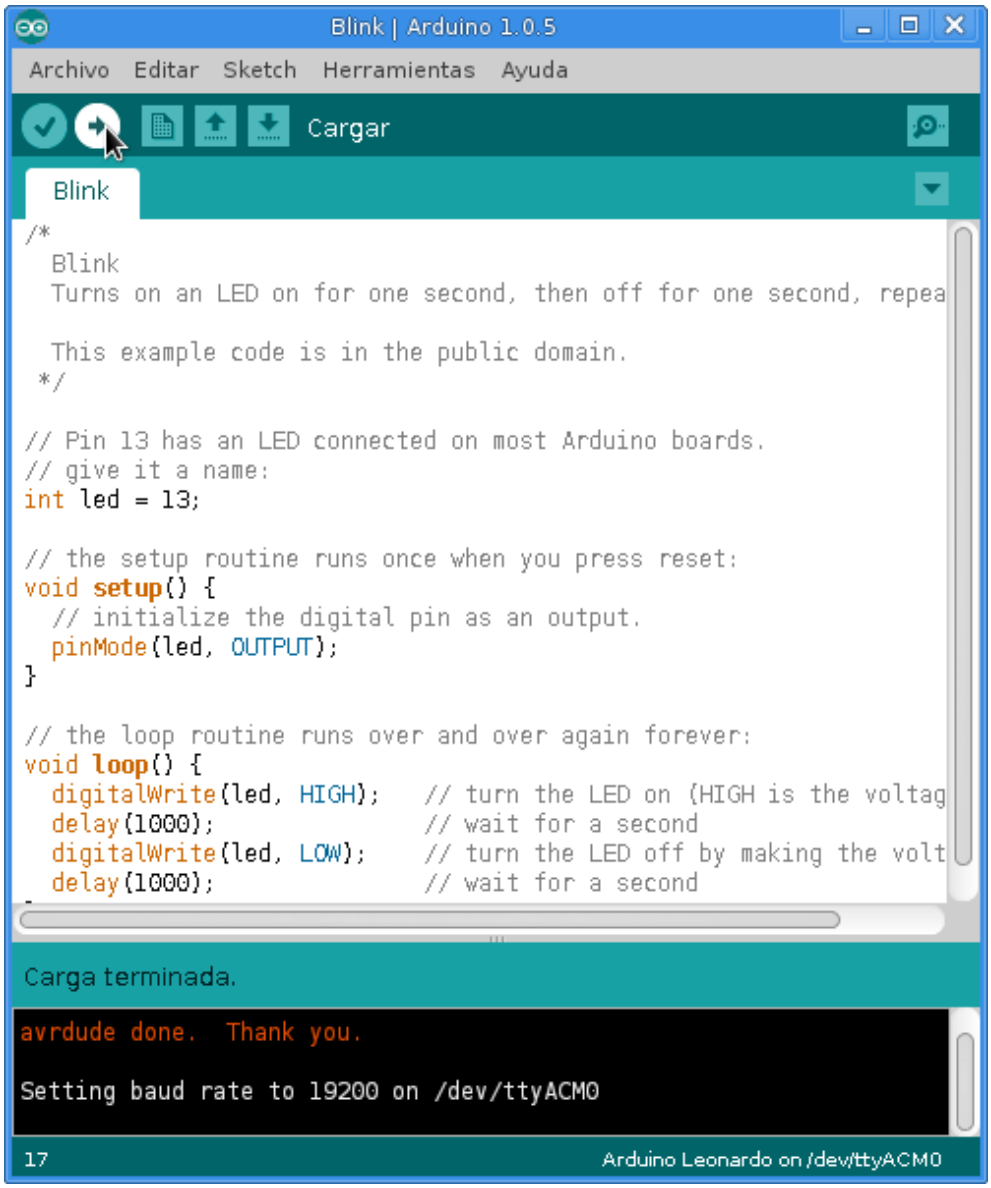
Si tuvimos suerte, podremos subir nuestro primer programa a la placa Arduino. Primero debemos conectar el cable USB a nuestra computadora, y el otro extremo a la placa Arduino para alimentarla.

Luego debemos indicarle al entorno Arduino la placa que utilizaremos, en este caso el menú *Herramientas > Tarjeta > Arduino Leonardo* y seleccionar el puesto Serial con el que se comunicara con la placa. *Herramientas > Puerto Serial > /dev/ttyACM0*. Generalmente aparecerá un único puerto COM# en Windows o /dev/tty# en GNU/Linux.



Una vez que el código fue verificado, hemos conectado la placa y configurado el entorno, procedemos a la carga de nuestro programa.

Debemos hacer Clic en el segundo botón de la barra de herramientas, y al finalizar nos aparecerá un mensaje en la parte inferior del entorno indicando que la carga fue terminada. Luego, podemos observar en la placa Arduino como el led integrado empezara a parpadear.



Esta es la mecánica de trabajo de la plataforma Arduino. A partir de ahora, podemos avanzar en proyectos mas complejos.

LENGUAJE ARDUINO

La sintaxis del lenguaje de programación Arduino es una versión simplificada de C/C+ y soporta todas las funciones del estándar C y algunas de C++. Analicemos el ejemplo anterior para comprender la estructura del código.

Comentarios

Las primeras líneas del sketch ejemplo Blink son un comentario.

```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */
```

Todo lo que se encuentre entre /* y */ es ignorado por el Arduino cuando se ejecuta el sketch (Los agregados * al comienzo de cada línea son solo para hacer el comentario más bonito, y no son necesarios). Generalmente se comenta para explicar lo que hace el programa, cómo funciona, o por qué está escrito de una forma en particular. Es una buena práctica comentar sus sketch, y usarlo para mantener documentado todos los cambios que realiza en el código. Esto ayuda a otras personas a aprender desde su código o modificarlo.

Hay otro estilo de comentarios cortos de una sola línea. Estos comienzan con // y continúan hasta el final de la línea. Por ejemplo, en la línea de:

```
int ledPin = 13;                // LED connected to digital pin 13
```

El mensaje “LED connected to digital pin 13”, es un comentario y sera ignorado al momento de compilar nuestro programa.

Variables

Una variable es un lugar para almacenar información. Tiene un nombre, un tipo y un valor. Por ejemplo, en el sketch Blink:

```
int ledPin = 13;                // LED connected to digital pin 13
```

se declara una variable con el nombre ledPin, se indica su tipo es int, y se le da un valor inicial de 13.

En este caso se utiliza para indicar en que pin de Arduino está conectado el LED. Cada vez que el nombre ledPin aparece en el código, se recuperará su valor. En este caso, la persona que escribe el programa podría haber optado por no crear la variable ledPin y en su lugar pudo haber escrito “13” en todas partes que necesitaba especificar el número de pin. La ventaja de utilizar una variable es que es más fácil cambiar el LED para utilizar un pin diferente: sólo tiene que editar la línea de código donde se asigna el valor inicial a la variable.

Punto y coma: ;

El punto y coma “;” se utiliza para separar instrucciones en el lenguaje de programación de Arduino.

```
int ledPin = 13;                // LED connected to digital pin 13
```

Si te olvidas de poner fin a una línea con un punto y coma al momento de verificar el codigo te devolverá un error de compilación. El texto de error puede ser obvio y se referirá a la falta de una coma,

o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

Funciones

Una función (también conocido como un procedimiento o subrutina) es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas cuando se llama a la función en otra parte del sketch.

```
type nombreFunción(parámetros)
{
      instrucción;
}
```

Las funciones se declaran asociadas a un tipo de valor “type”. Este valor será el que devolverá la función, por ejemplo 'int' se utilizará cuando la función devuelve un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”, que significa “función vacía”. Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

Por ejemplo, aquí está la definición de la función setup () del ejemplo Blink:

```
void setup()
{
      pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}
```

La primera línea es la definición de la función que no devuelve ningún valor por ser tipo void, se llama “setup” y para llamarla o invocarla no necesitamos pasarle ningún argumento. El código entre el { y } se llama el cuerpo de la función, y es lo que hace la función al momento de invocarla.

Usted puede llamar a una función que ya ha sido definida (ya sea en su sketch o como parte del lenguaje Arduino). Por ejemplo, la línea pinMode(ledPin, OUTPUT); llama a la función pinMode() que es propia del lenguaje Arduino, y se le pasa los parámetros: ledPin y OUTPUT. Estos parámetros son requeridos por la función pinMode () para decidir qué pin debe configurar y de que forma debe hacerlo.

Llaves: {}

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Por ejemplo las funciones setup (), loop (), o otras sentencias como if (), while () etc.

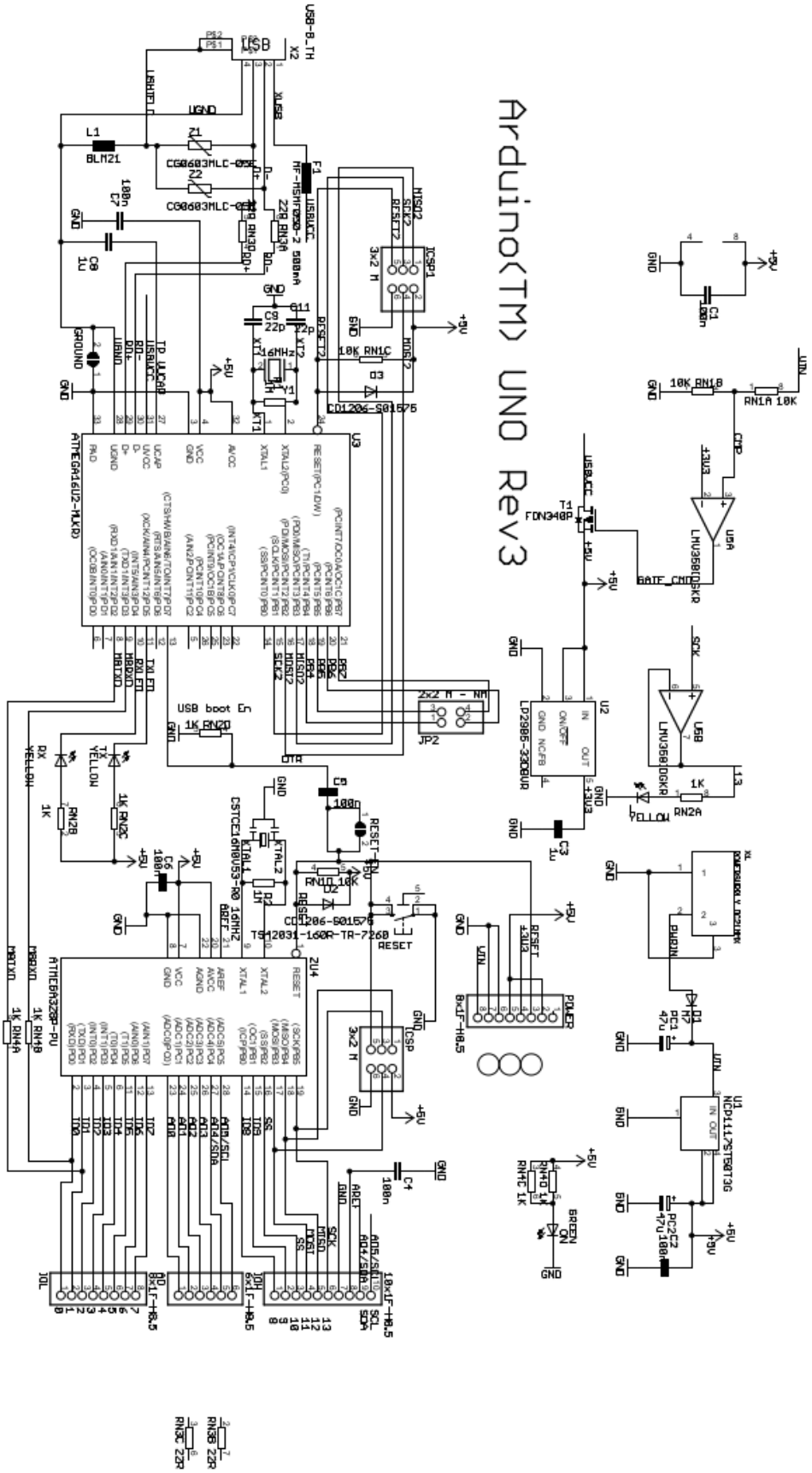
```
type funcion()
{
      instrucciones;
}
```

Una llave de apertura “{“ siempre debe ir seguida de una llave de cierre “}”, si no es así la verificación del programa dará errores.

Funciones pinMode(), digitalWrite(), y delay()

Las funciones pinMode(), digitalWrite(), y delay(), son funciones definidas por el lenguaje Arduino. Aunque parecen definidas solo de forma implícita, están implementadas en el core Arduino de la misma forma que vimos. Se encuentran escritas indicando su tipo, nombre, parámetros, e instrucciones, y cumplen las mismas reglas de corchetes, comentarios y punto y coma.

ANEXO I: Esquemático Arduino UNO



ANEXO II: Características Eléctricas

30. Electrical Characteristics (T_A = -40°C to 105°C)

30.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0mA
DC Current V_{CC} and GND Pins	200.0mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

30.2 DC Characteristics

Table 30-1. Common DC characteristics T_A = -40°C to 105°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

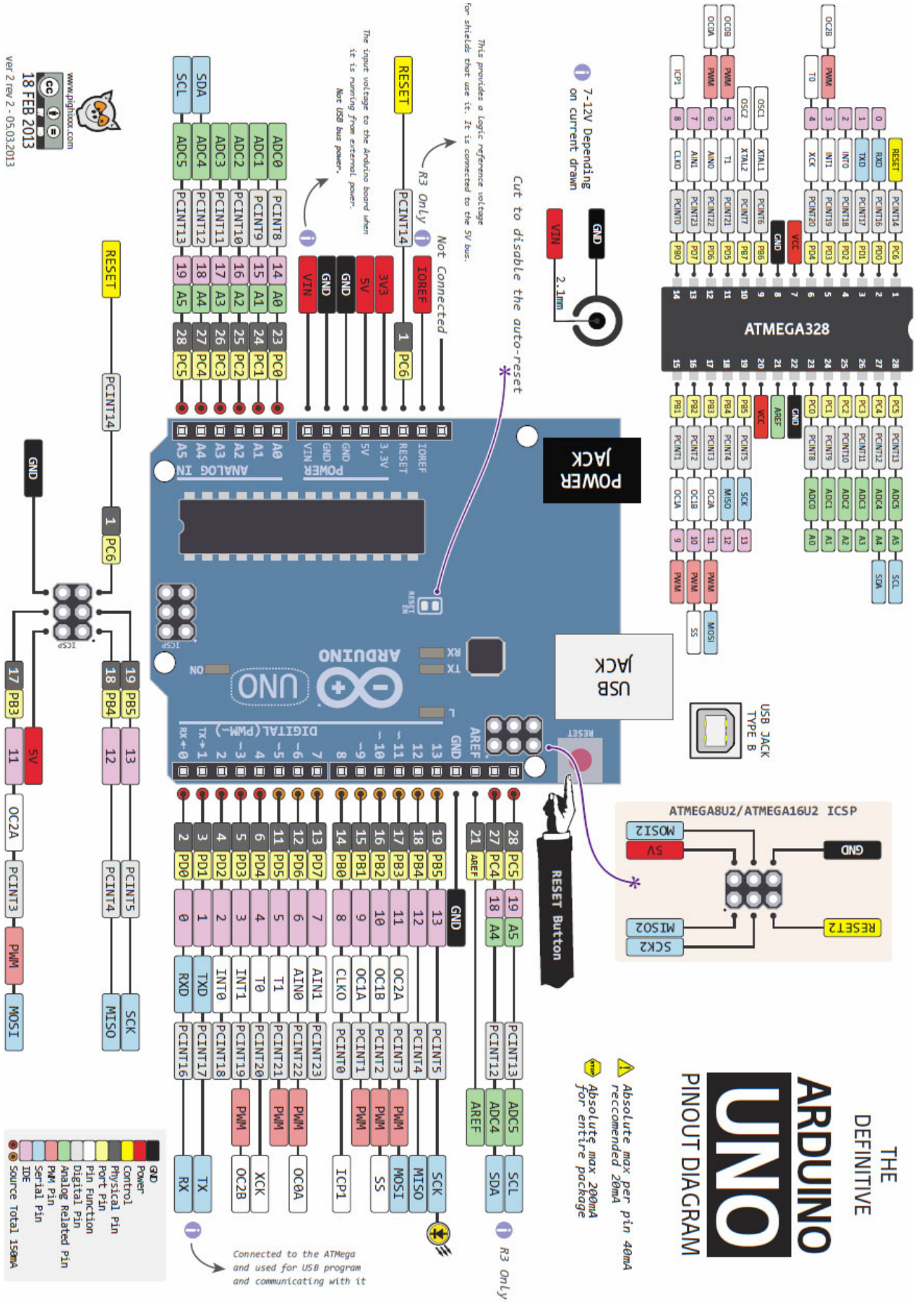
Symbol	Parameter	Condition		Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage, except XTAL1 and $\overline{\text{RESET}}$ pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V		-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH}	Input High Voltage, except XTAL1 and $\overline{\text{RESET}}$ pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V		$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 1.8V - 5.5V		-0.5		$0.1V_{CC}$(1)	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V		$0.8V_{CC}$(2) $0.7V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL2}	Input Low Voltage, $\overline{\text{RESET}}$ pin	V_{CC} = 1.8V - 5.5V		-0.5		$0.1V_{CC}$(1)	V
V_{IH2}	Input High Voltage, $\overline{\text{RESET}}$ pin	V_{CC} = 1.8V - 5.5V		$0.9V_{CC}$(2)		V_{CC} + 0.5	V
V_{IL3}	Input Low Voltage, $\overline{\text{RESET}}$ pin as I/O	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V		-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH3}	Input High Voltage, $\overline{\text{RESET}}$ pin as I/O	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V		$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(4) except RESET pin	I_{OL} = 20mA, V_{CC} = 5V	T_A=85°C			0.9	
			T_A=105°C			1.0	
		I_{OL} = 10mA, V_{CC} = 3V	T_A=85°C			0.6	
			T_A=105°C			0.7	V

Table 30-1. Common DC characteristics T_A = -40°C to 105°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted) (Continued)

Symbol	Parameter	Condition		Min.	Typ.	Max.	Units
V_{OH}	Output High Voltage[3] except Reset pin	I_{OH} = -20mA, V_{CC} = 5V	T_A=85°C	4.2			
			T_A=105°C	4.1			
		I_{OH} = -10mA, V_{CC} = 3V	T_A=85°C	2.3			
			T_A=105°C	2.1			V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)				1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)				1	µA
R_{RST}	Reset Pull-up Resistor			30		60	kΩ
R_{PU}	I/O Pin Pull-up Resistor			20		50	kΩ
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2			<10	40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2		-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V			750 500		ns

Notes:
1. "Max" means the highest value where the pin is guaranteed to be read as low
2. "Min." means the lowest value where the pin is guaranteed to be read as high
3. Although each I/O port can source more than the test conditions (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   ATmega48A/PA/88A/PA/168A/PA/328/P:
   1] The sum of all I_{OH}, for ports C0 - C5, D0- D4, ADC7, $\overline{\text{RESET}}$ should not exceed 150mA.
   2] The sum of all I_{OH}, for ports B0 - B5, D5 - D7, ADC6, XTAL1, XTAL2 should not exceed 150mA.
   If I_{OH} exceeds the test condition, V_{OH} may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.
4. Although each I/O port can sink more than the test conditions (20mA at V_{CC} = 5V, 10mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   ATmega48A/PA/88A/PA/168A/PA/328/P:
   1] The sum of all I_{OL}, for ports C0 - C5, ADC7, ADC6 should not exceed 100mA.
   2] The sum of all I_{OL}, for ports B0 - B5, D5 - D7, XTAL1, XTAL2 should not exceed 100mA.
   3] The sum of all I_{OL}, for ports D0 - D4, RESET should not exceed 100mA.
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.

ANEXO III: Resumen de pines



ANEXO IV: Resumen de funciones digitales

pinMode ()

Descripción

Esta función configura el pin especificado para actuar como entrada o salida digital. Los pines de entrada analógica también se pueden utilizar como pines digitales, refiriéndose como A0, A1, etc.

Sintaxis

```
pinMode (pin, modo);
```

Parámetros

- *pin:* El número del pin cuyo modo que desea ajustar.
- *modo:* INPUT, OUTPUT, o INPUT_PULLUP.

Devuelve

- *Nada.*

digitalWrite ()

Descripción

Activa (HIGH) o desactiva (LOW) un pin digital. Si el pin ha sido configurado como OUTPUT (salida) con la función pinMode(), su voltaje será activado a 5V (o 3.3V en las tarjetas que funcionen a 3.3V) si se activa (HIGH) o a 0V (tierra) si se desactiva (LOW).

Si el pin ha sido configurado como INPUT (entrada), digitalWrite() activará (si usamos el parámetro HIGH) o desactivará (con LOW) la resistencia "pullup" del pin de entrada especificado. Sin embargo se recomienda activar la resistencia interna "pullup" del pin utilizando la función *pinMode(pin, INPUT_PULLUP).*

Sintaxis

```
digitalWrite (pin, valor)
```

Parámetros

- *pin:* El número de pin a modificar.
- *valor:* HIGH o LOW

Devuelve

- *Nada.*

digitalRead ()

Descripción

Lee el valor de un pin digital especificado, ya sea alto o bajo. Tenga en cuenta que el pin siempre debe estar conectado a algo. Si no lo hace, la lectura es incierta y puede variar aleatoriamente.

Sintaxis

```
digitalRead(pin);
```

Parámetros

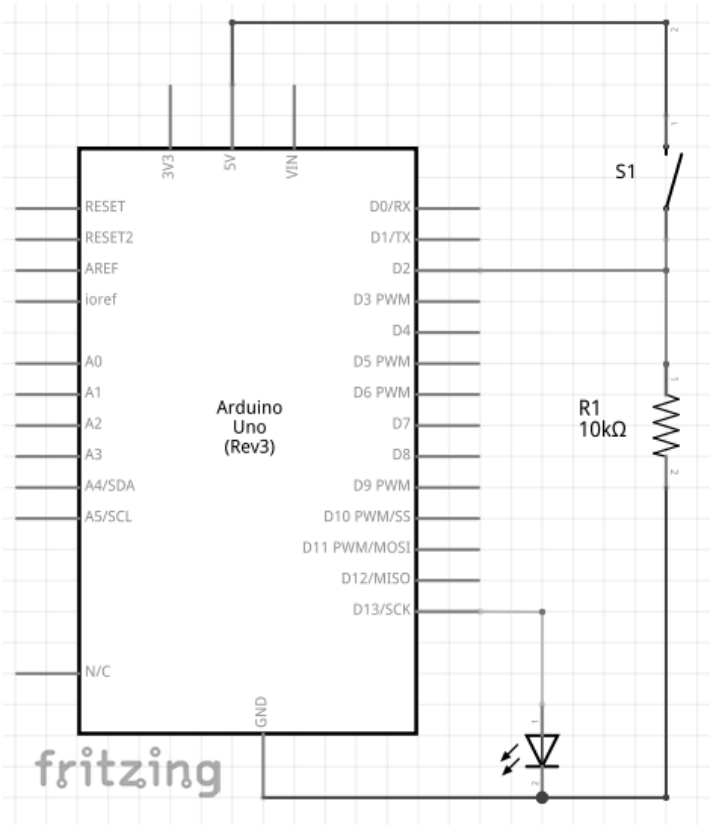
- Pin: El número del pin digital desea leer. (int)

Devuelve

- *HIGH o LOW:* El estado de la entrada digital.

Ejemplo

Se coloca un pulsador en el pin digital 2 (Junto con una resistencia de pull-down), que se configura como una entrada, y utilizamos el diodo LED interno de pin digital 13, configurado como una salida. Leemos el estado del pulsador, y lo reflejamos a la salida.




```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 2;   // pushbutton connected to digital pin 2
int val = 0;     // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
  pinMode(inPin, INPUT);        // sets the digital pin 2 as input
}

void loop()
{
  val = digitalRead(inPin);     // read the input pin
  digitalWrite(ledPin, val);    // sets the LED to the button's value
}
```

ANEXO V: Resumen de funciones analogicas

analogRead ()

Descripción

Esta función lee la tensión aplicada sobre un pin especificado, y hace una conversión analógica-digital con una resolución de 10 bits devolviendo un entero en el rango de 0 a 1023 proporcional a la tensión.

Sintaxis

```
analogRead(pin);
```

Parámetros

- *pin:* El número del pin donde se desea hacer una lectura.

Devuelve

- *Entero de 0 a 1023*

NOTA: A diferencia de los pines digitales, los pines analógicos no deben ser configurados para ser utilizados. Por defecto son entradas analógicas, sin embargo vale aclarar, que pueden ser configuradas como entradas o salidas digitales utilizando la función pinMode();

analogReference ()

Descripción

Configura la tensión de referencia utilizado para la comparación de las entradas analógicas. Este es utilizados como la parte superior del rango de entrada. La referencia analógica por defecto es DEFAULT, y es de 5 voltios (en las placas Arduino de 5V) o 3,3 voltios (en los placas Arduino de 3.3V)

Sintaxis

```
analogReference(tipo);
```

Parámetros

- *Tipo: DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, or EXTERNAL).*

Devuelve

- Nada.

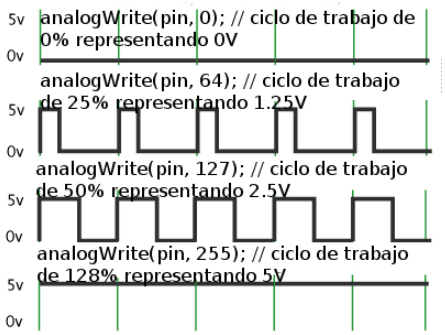
analogWrite()

Descripción

Esta función escribe un valor pseudoanalógico usando modulación por ancho de pulso (PWM en inglés) a un pin de salida marcado como PWM. Lo que hace este tipo de señal es emitir, en lugar de una señal continua, una señal cuadrada formada por pulsos de frecuencia constante, que en el promedio de tiempo en que la señal se encuentra encendio y apagado, representa un valor pseudoanalogio. En Arduino UNO esta función traba ja en los pines 3, 5, 6, 9, 10 y 11.

Sintaxis

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



Parámetros

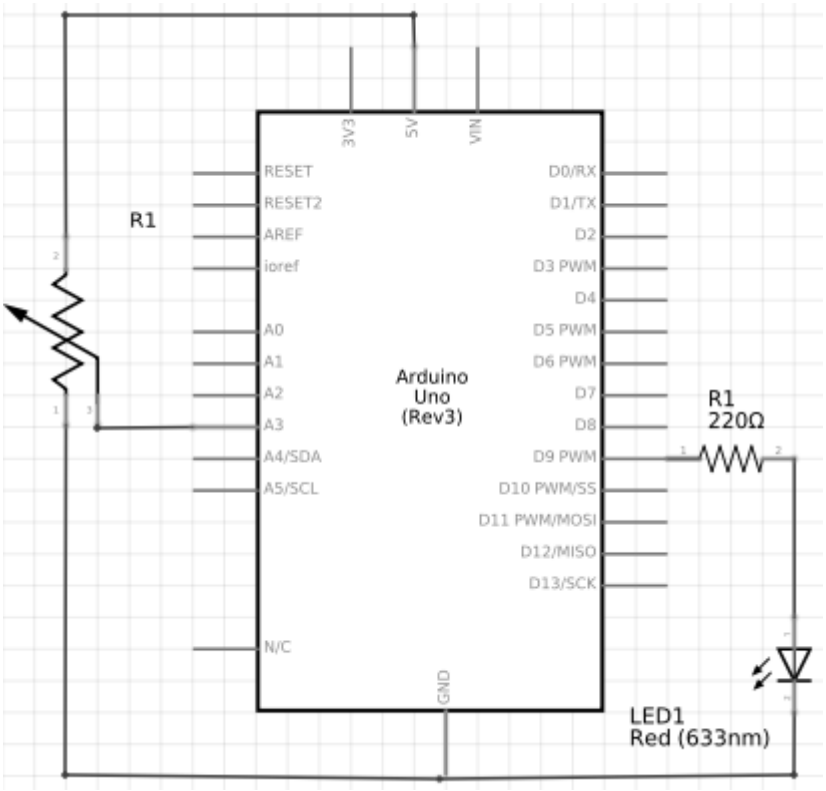
- *pin:* El número de pin a sacar una señal pseudoanalógico.
- *valor:* Un entero entre 0 y 255.

Devuelve

- *Nada.*

Ejemplo

Se coloca una resistencia variable que divide la tensión del circuito aplicando la tensión resultante en el pin analógico 3. También se coloca un led en el pin digital 9 con su correspondiente resistencia. Asi leeremos la tensión aplicada y la reflejamos de forma pseudoanalogica sobre el led.




```
const int ledPin = 9;      // LED connected to digital pin 9
const int analogPin = 3;   // potentiometer connected to analog pin 3

void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the pin as output
}

void loop()
{
  int val = 0;                    // variable to store the read value
  val = analogRead(analogPin);    // read the input pin
  analogWrite(ledPin, val / 4);  // analogRead values go from 0 to 1023,
                                  // analogWrite values from 0 to 255
}
```

ANEXO X: Códigos de resistencias

El código de colores se utiliza en electrónica para indicar los valores de los componentes electrónicos. Las dos primeras franjas desde la izquierda, indican las primeras cifras del valor del componente, mientras que una tercera indica por cuanto debe multiplicarse el valor de la cifra leída. La última franja, más separada del resto, y típicamente de color dorado o plata, indica la tolerancia, es decir, el margen de error que garantiza el fabricante. En el caso de las resistencias de precisión, se cuenta con seis bandas de colores: las tres primeras indican cifras, la cuarta el multiplicador, la quinta la tolerancia y la sexta, el coeficiente de temperatura.

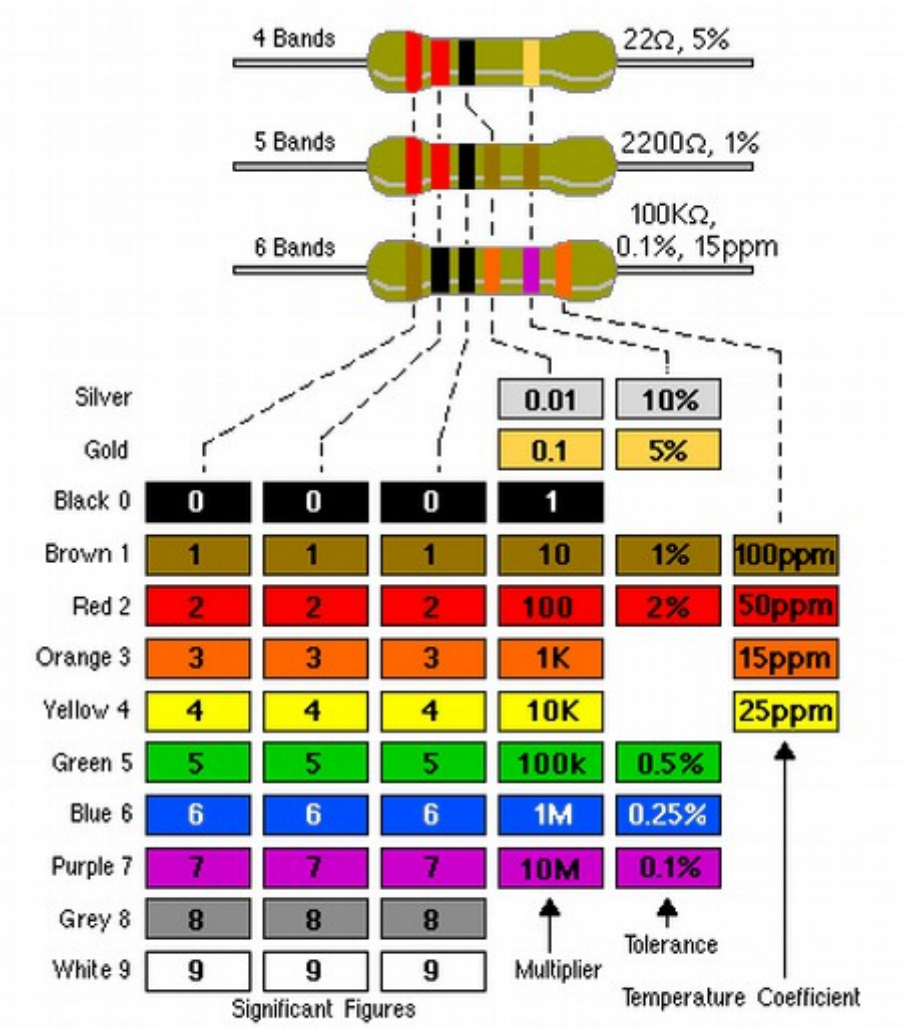


Tabla de valores comerciales de las Resistencias

Muchas veces, al realizar el calculo de la resistencia que necesitamos colocar en nuestro circuito eléctrico, nos da valores que difícilmente podamos conseguir en las tiendas de electrónica, en esos casos tendremos que comprar la resistencia de valor mas cercano a la que estemos necesitando, o en su defecto poner varias en una configuración particular que la resistencia equivalente nos de la que calculamos previamente. Obviamente hacer esto ultimo se traduce en encarecer muchísimo el valor de nuestro desarrollo y simplemente no tiene sentido hacerlo, ya que los valores comerciales de resistencias eléctricas tienen un rango de tolerancias, con lo cual no tiene sentido buscar la exactitud si el fabricante ya nos dice que su valor puede variar en un 5% un 10% o mas..

Lo primero que tenemos que hacer es conocer los valores de resistencias comerciales que hay en el mercado, en las siguiente tabla se muestra un listado de las que podemos conseguir en casi cualquier casa de electrónica.

X1	X10	X100	X1.000	X10.000	X100.000	X1.000.000
1Ω	10Ω	100Ω	1kΩ	10kΩ	100kΩ	1.0MΩ
1.2Ω	12Ω	120Ω	1.2kΩ	12kΩ	120.0kΩ	1.2MΩ
1.5Ω	15Ω	150Ω	1.5kΩ	15kΩ	150.0kΩ	1.5MΩ
1.8Ω	18Ω	180Ω	1.8kΩ	18kΩ	180.0kΩ	1.8MΩ
2.2Ω	22Ω	220Ω	2.2kΩ	22kΩ	220.0kΩ	2.2MΩ
2.7Ω	27Ω	270Ω	2.7kΩ	27kΩ	270.0kΩ	2.7MΩ
3.3Ω	33Ω	330Ω	3.3kΩ	33kΩ	330.0kΩ	3.3MΩ
3.9Ω	39Ω	390Ω	3.9kΩ	39kΩ	390.0kΩ	3.9MΩ
4.7Ω	47Ω	470Ω	4.7kΩ	47kΩ	470.0kΩ	4.7MΩ
5.1Ω	51Ω	510Ω	5.1kΩ	51kΩ	510.0kΩ	5.1MΩ
5.6Ω	56Ω	560Ω	5.6kΩ	56kΩ	560.0kΩ	5.6MΩ
6.8Ω	68Ω	680Ω	6.8kΩ	68kΩ	680.0kΩ	6.8MΩ
8.2Ω	82Ω	820Ω	8.2kΩ	82kΩ	820.0kΩ	8.2MΩ