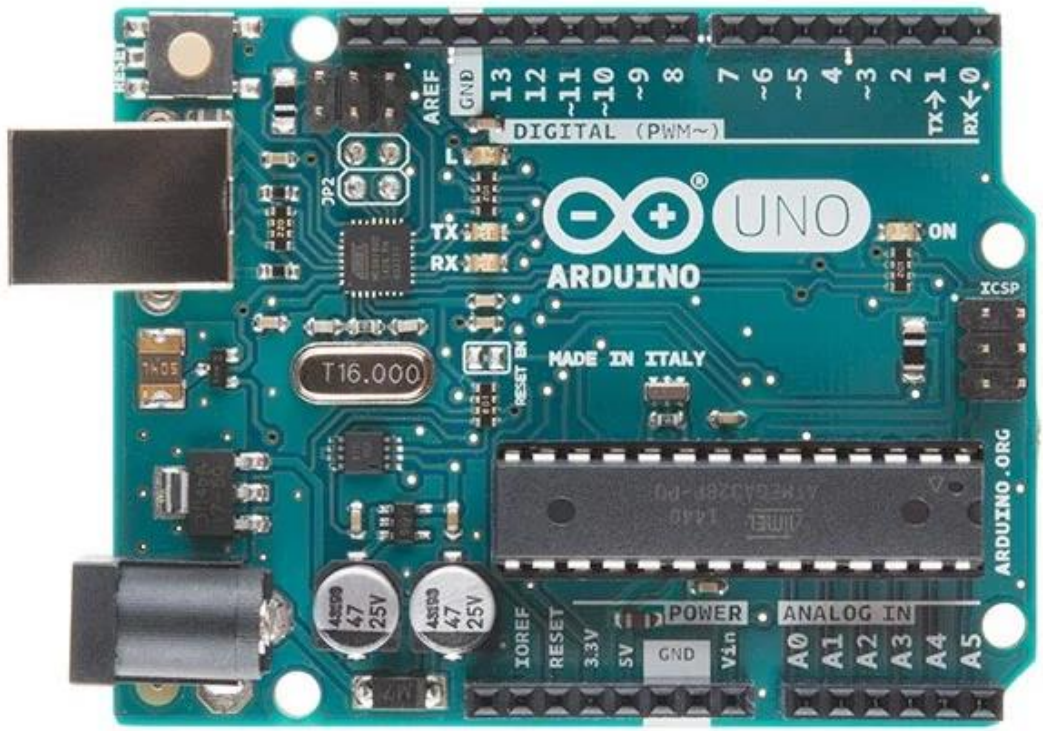
1. ARDUINO UNO

Esta es la placa más clásica de todas, el **Arduino UNO R3** es todo un veterano con características básicas interesantes para los amantes de la electrónica y la programación. Es ideal para utilizar en tus primeros para con la plataforma Arduino ya que es ampliamente utilizado y tiene una infinidad de ejemplos prácticos en Internet con casi cualquier sensor o actuador

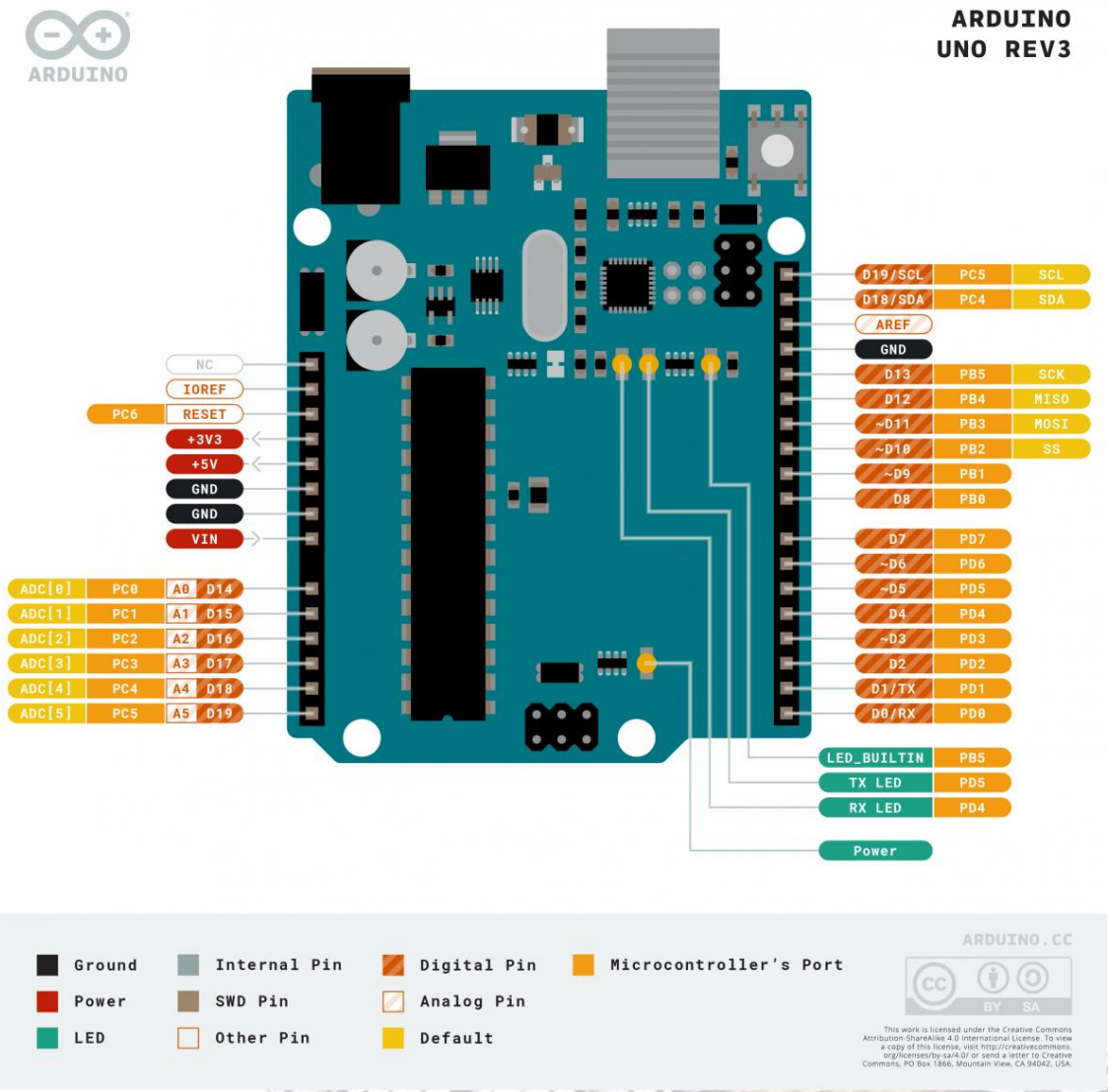


Características de Arduino UNO R3

Microcontrolador:	ATmega328P
Voltaje de operación:	5V
Entrada de alimentación:	7-12V
Pines digitales I/O:	14
Pines PWM:	6
Pines analógicos:	6
Corriente por pin:	20 mA
Corriente del pin 3.3.V:	50 mA max
Memoria Flash:	32 KB (0.5 KB usados por el bootloader)
SRAM:	2 KB
EEPROM:	1 KB
Velocidad de reloj:	16 MHz
LED programable integrado:	13
Dimensiones:	68.6 x 53.4 mm
Peso:	25 gramos

Definición de pines y esquema del Arduino UNO R3

Gracias a esta imagen, puedes ver de un tirón para qué vale cada uno de sus pines. Es un esquema-chuleta muy práctica que siempre está bien tener a mano. Más abajo tienes unos enlaces donde si quieres lo puedes descargar en formato PDF para poder imprimirlo.



Cómo programar el Arduino UNO

El Arduino Uno se puede programar con el Software Arduino IDE. Para eso, debes seleccionar **"Arduino Uno"** en el menú **Herramientas / Placa**. El ATmega328 en Arduino Uno viene preprogramado con un **bootloader** que le permite cargar programar sin el uso de un programador de hardware externo.

Se comunica usando el protocolo STK500 original. También puedes omitir el bootloader de arranque y programar el microcontrolador a través del conector ICSP (In Circuit Serial Programming) usando **Arduino ISP** o similar.

Ésta operación borrará el bootloader de la placa y no podrás volver a cargar programas mediante el puerto USB.

Protección del bus USB

El Arduino Uno tiene un polifusible integrado que protege los puertos USB de tu ordenador ante cortocircuitos y sobrecorriente. Aunque la mayoría de los ordenadores proporcionan su propia protección interna, el fusible proporciona una capa adicional de protección. Si se aplican más de 500 mA al puerto USB, el fusible cortará automáticamente la conexión hasta que se elimine el cortocircuito o la sobrecarga.

Comunicación USB

El Arduino UNO no utiliza un conversor serial como el FTDI232. En su lugar, tiene otro pequeño microcontrolador Atmega16U2 programado como conversor USB-Serial. También se puede programar mediante ICSP para usos avanzados.

Cómo alimentar el Arduino UNO

La placa Arduino Uno se puede alimentar 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 provenir de un adaptador de corriente de pared o de una batería.

El adaptador se puede conectar con un enchufe de **centro positivo de 2,1 mm de diámetro** en el conector Jack de alimentación de la placa. Los cables de una batería se pueden insertar en los cabezales de los pines **GND** y **VIN** del conector POWER.

La placa puede funcionar con una fuente externa de 6 a 20 voltios. Sin embargo, si se alimenta con menos de 7 V, el pin de 5 V puede suministrar menos de cinco voltios y la placa puede volverse inestable. Si usas más de 12V, el regulador de voltaje puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Desglose de pines de alimentación:

- **VIN:** El voltaje de entrada a la placa Arduino cuando está usando una fuente de alimentación externa (a diferencia de los 5 voltios de la conexión USB u otra fuente de alimentación regulada). Puedes suministrar voltaje a través de este pin o, si suministras voltaje a través del conector de alimentación, acceder a él a través de este pin.
- **5V:** Este pin proporciona 5V regulados desde el regulador interno de la placa. La placa puede recibir alimentación desde el conector de alimentación DC (7 a 12 V), el conector USB (5 V) o el pin VIN de la placa (7-12 V). El suministro de voltaje a través de los pines de 5 V o 3,3 V

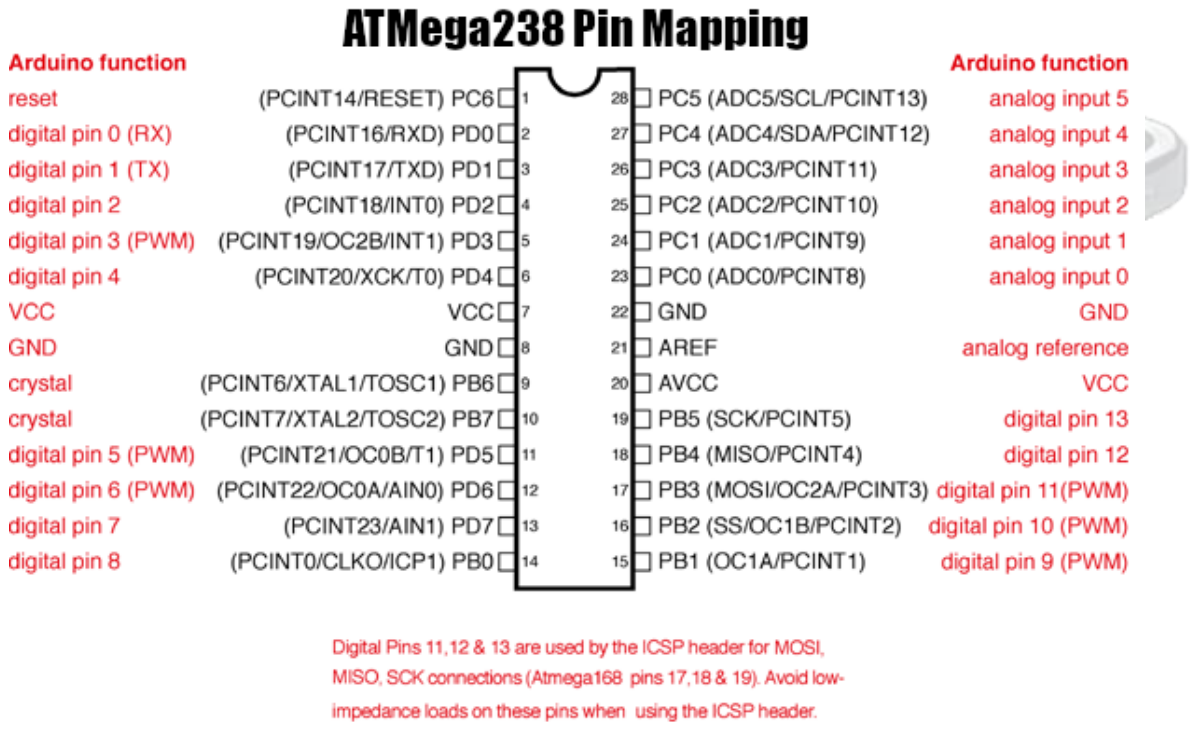
- puentea el regulador y puede dañar la placa si lo haces mal. Por lo tanto ese método alimentación no es recomendable
- **3V3:** Proporciona 3.3 Voltios generado por el regulador integrado. El consumo máximo de corriente es 50 mA.
- **GND:** Pines negativos (masa)
- **IOREF:** Este pin en la placa Arduino proporciona la referencia de voltaje con la que opera el microcontrolador. Una shield configurada correctamente puede leer el voltaje del pin IOREF y seleccionar la fuente de alimentación adecuada o permitir que los conversores de voltaje (ADC) en las salidas funcionen con 5V o 3.3V.

Memoria Flash, RAM y EEPROM

El ATmega328 tiene 32 KB (0,5 KB ocupados por el gestor de arranque, bootloader) disponibles para almacenar el programa. También tiene 2 KB de SRAM (volátil) y 1 KB de EEPROM (permanente), que se puede leer y escribir con la **librería EEPROM.**

Mapa de pines del microcontrolador ATMega328 en Arduino

El mapa de pines es la correlación entre el nombre de un pin en el entorno de Arduino y su conexión física con el microcontrolador ATMega328. El mapa es idénticos en todos los modelos que tengas un microcontroladro ATMega8, ATMega168 y ATMega328.



Mapa de pines del microcontrolador ATMega328 en Arduino

Cada uno de los 14 pines digitales del Arduino Uno se pueden utilizar como entrada o salida, utilizando las funciones **pinMode()**, **digitalWrite()** y **digitalRead()**. Todos los pines funcionan a 5 Voltios. Cada pin puede proporcionar o recibir 20 mA (como máximo) como condición de operación recomendada y tiene una resistencia de *pull-up* interna (desconectada por defecto) de 20 a 50k ohmios.

Un **máximo de 40 mA** es el valor que no debe excederse en ningún pin para evitar daños permanentes al microcontrolador. Además, algunos pines tienen funciones especializadas:

- **Serial: Pin 0 (RX) y 1 (TX):** Se utiliza para recibir (RX) y transmitir (TX) datos por el puerto serie TTL. Estos pines están conectados a los pines correspondientes del chip serie ATmega8U2 USB a TTL.
- **Interrupciones externas: Pin 2 y 3:** Estos pines se pueden configurar para activar una interrupción en un valor bajo, un flanco ascendente o descendente o un cambio de valor. Suele utilizarse con la **función attachInterrupt()**
- **PWM: Pines 3, 5, 6, 9, 10 y 11:** Proporcione una salida PWM con una resolución de 8 bits usando la función **analogWrite().**
- **Bus SPI: Pines 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK):** Estos pines permiten una comunicación SPI mediante la biblioteca SPI. Muchos dispositivos como pantallas LCD funcionan con éste tipo de bus. Además es un bus bastante rápido comparado con el I2C.
- **LED: Pin 13:** Hay un LED incorporado conectado al **pin digital 13**. Cuando el pin tiene un valor ALTO, el LED está encendido, cuando el pin está en BAJO, está apagado.
- **Bus I2C: pin A4 (SDA) y pin A5 (SCL):** También llamado TWI, permite una comunicación I2C utilizando la biblioteca Wire. Una gran cantidad de sensores utilizan éste tipo de bus I2C y funciona con hasta 128 dispositivos teóricos sobre el mismo bus.
- **AREF:** Voltaje de referencia para las entradas analógicas. Usado con *analogReference ()*.
- **RESET:** Si pones ese pin a nivel BAJO puedes reiniciar el microcontrolador. Normalmente se usa para agregar un botón de reinicio a los shield que bloquean el que está en la placa.

El Arduino Uno tiene 6 entradas analógicas, etiquetadas como **A0 a A5**, cada una de las cuales proporciona una resolución de 10 bits (es decir, 1024 valores diferentes). Por defecto miden desde tierra hasta 5 voltios, aunque es posible cambiar el extremo superior de su rango usando el pin **AREF** (IOREF) y usando la función **analogReference()**.

Puerto de comunicación Serie

El Arduino Uno tiene varias funciones para comunicarse con el ordenador, otras placa Arduino u otros microcontroladores. El ATmega328 proporciona comunicación serie UART TTL (5V), que está disponible en los pines digitales 0 (RX) y 1 (TX). El ATmega16U2 integrado en la placa se encarga de esta comunicación en serie a través de USB y aparece como un puerto de comunicación virtual para el software en el ordenador.

El firmware del ATmega16U2 utiliza los controladores COM USB estándar y no se necesita ningún controlador o driver externo. Sin embargo, en algunos sistemas Windows antiguos como el XP o el 7, se requiere un **archivo .inf**.

El software Arduino IDE incluye un monitor serie que permite enviar datos simples hacia y desde la placa. Los LED RX y TX de la placa parpadearán cuando los datos se transmitan a través del chip USB a serie y la conexión USB al ordenador (pero no para la comunicación en serie en los pines 0 y 1).

Una buena librería para comunicación serie es la **SoftSerial** que viene integrada en Arduino IDE.

Reinicio automático

En lugar de requerir una pulsación física del botón de RESET al cargar un programa, la placa Arduino Uno está diseñada de una manera que permite que se reinicie mediante un software que se ejecuta en el ordenador. Una de las líneas de control de flujo de hardware (DTR) del ATmega16U2 está conectada a la línea de RESET del ATmega328 a través de un condensador de 100 nanofaradios.

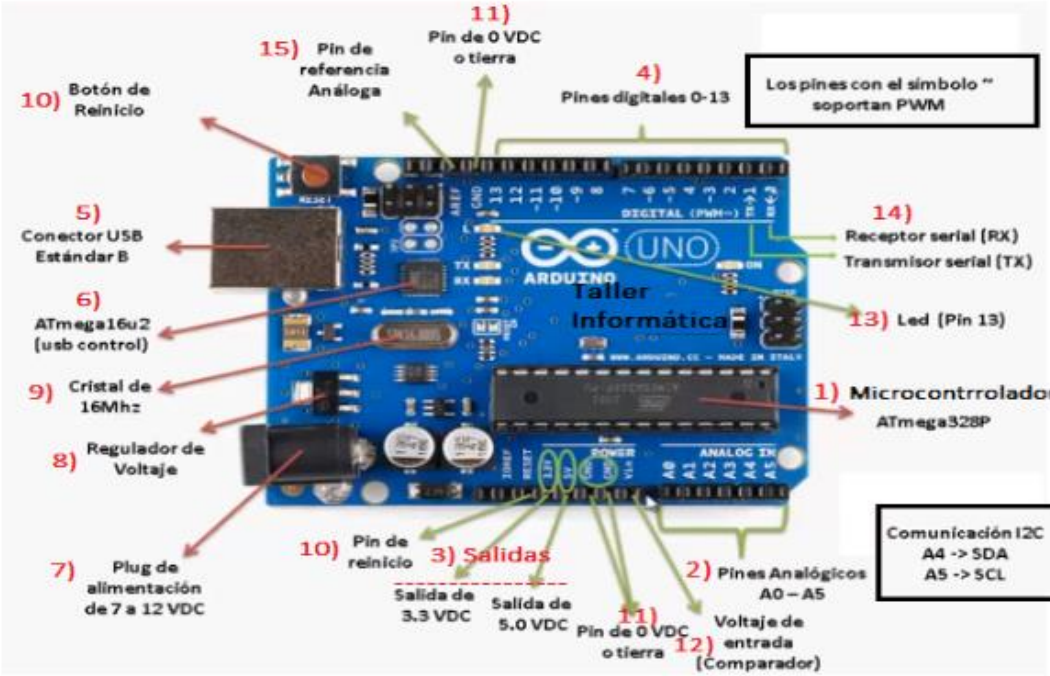
Cuando esta línea se pone a nivel bajo, la línea de reinicio cae lo suficiente para reiniciar el chip. El software Arduino utiliza esta capacidad para permitirle cargar código simplemente presionando el botón de carga en la barra de herramientas de la interfaz. Esto significa que el *bootloader* puede tener un tiempo de espera más corto, ya que la reducción de DTR puede coordinarse bien con el inicio de la carga.

Esta configuración tiene otras implicaciones. Cuando el Uno está conectado a una computadora con **Mac OS X** o **Linux**, se reinicia cada vez que se establece

una conexión desde el software (a través de USB). Durante el siguiente medio segundo más o menos, el gestor de arranque se ejecuta en el Arduino Uno. Si bien está programado para ignorar los datos con formato incorrecto (es decir, cualquier cosa además de la carga de un nuevo código), interceptará los primeros bytes de datos enviados a la placa después de que se abra una conexión. Si un *Sketch* que se ejecuta en la placa recibe una configuración única u otros datos cuando se inicia por primera vez, asegúrate de que el software con el que se comunica **espera al menos un segundo** después de abrir la conexión y antes de enviar estos datos.

La placa Uno contiene una pista que se puede cortar para desactivar el reinicio automático. También tiene unos pads a cada lado de la placa que se pueden soldar juntas para volver a habilitarlo. Tiene la etiqueta **"RESET-EN"**. También puede desactivar el reinicio automático conectando una resistencia de 110 ohmios de 5 V a la línea de RESET.

ARDUINO UNO: Características



1) El Microcontrolador: Es la parte que procesa toda la información, es donde se graba el código, en el software de Arduino se conoce como "Sketch". Los Microcontroladores que usa Arduino son económicos lo que abarata el costo de la Tarjeta en general.

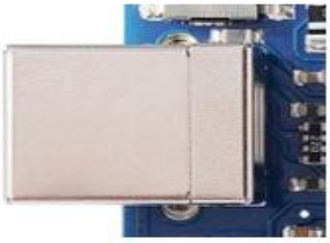
2) Pines analógicos: Estos pueden detectar señales análogas como por ejemplo la Luz o la Temperatura, estos sensores poseen un segmento de voltaje de funcionamiento desde cero (0) a Cinco (5) Voltios. A través de estos se pueden medir cosas del mundo real como por ejemplo Temperatura. Si el Pin de lectura analógica tiene una resolución de ocho (8) bits, te va permitir dividir los cinco (5) voltios en 256 segmentos, es decir la Temperatura máxima del sensor equivale a 256 y la mínima a cero (0).

3) Pines de Poder o de Salidas: A través de estos se pueden alimentar componentes que requiera de poca alimentación como 3, 3 o 5 voltios.

4) Pines Digitales: Estos detectan si hay un Cero (09 o un Uno (1) lógico. Se utilizan para pulsaciones de botones o dispositivos que mandan o reciben información digital.

Los Pines que contiene una línea ondulada (como en la Ñ) soportan PWM (Power o Pulse-Width Modulation – Modulación por ancho de Pulso), usado para el control de intensidad, por ejemplo de luz.

5) Puerto o conector USB: Este nos permite conectar nuestra Arduino a la PC, cargar nuestro código y alimentar la tarjeta.



6) Control de USB: Este circuito integrado es el moderador entre el Microcontrolador y el software o C, es decir, se encarga de convertir la información del Microcontrolador hacia la información que va a tu computadora.

7) Alimentación: Nos permite alimentar nuestra tarjeta con voltaje de Corriente Continua de Siete (7) a Doce (12) voltios.

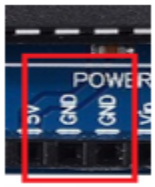
8) Regulador de Voltaje: Permite una salida estable de Cinco (5) voltios independientemente del voltaje de entrada.

9) Cristal: Da el Ciclo reloj, le marca el pulso o tiempo de trabajo al Microcontrolador para que este trabaje perfectamente.

10) Botón y Pin de Reinicio: El botón de reinicio está directamente conectado al Pin número uno (1) del Microcontrolador conocido como "Clear" ó "Master Clear", este necesita de sus cinco (5) voltios para hacer correr el programa, al presionar el botón interrumpe el voltaje a cero (09 voltios y detiene el programa que el Arduino está ejecutando y vuelve a ejecutar el programa desde su inicio.

El pin de reinicio o Reset permite hacer lo mismo que el botón pero a través de un circuito individual.

11) GND, Pin Cero Voltios o Tierra: Es la contraparte del positivo (5 voltios), es lo que permite cerrar el círculo de alimentación.



12) Voltaje de entrada Comparador: No es un voltaje de alimentación a la tarjeta, es una entrada analógica que utiliza para comparar la salida entre positivo y negativos devolviendo un Cero o un Uno.

13) El LED: El LED está conectado directamente al Pin 13 y contiene su respectiva resistencia, cada vez que el Pin 13 recibe un Uno (1) lógico ese LED se enciende.

14) Recepción (RX) y Transmisión (TX) Serial: Esta transmisión se da a través de los Pines Cero (0) y Uno (1).

PROGRAMACION DE ARDUINO

1. Estructura del Programa

Los programas de Arduino, llamados sketches, tienen dos funciones principales obligatorias:

1.1 setup()

- Se ejecuta una sola vez al iniciar el programa.
- Configura el hardware y las variables iniciales.

```
void setup() {
  pinMode(13, OUTPUT); // Configurar el pin 13 como salida
}
```

1.2 loop()

- Se ejecuta de forma repetitiva mientras la placa esté encendida.
- Contiene el comportamiento principal del programa.

Copiar código

```
void loop() {
  digitalWrite(13, HIGH); // Encender LED
  delay(1000);            // Esperar 1 segundo
  digitalWrite(13, LOW);  // Apagar LED
  delay(1000);            // Esperar 1 segundo
}
```

Manual de Programación Arduino

Este manual ofrece una guía completa sobre las estructuras de programación en Arduino, incluyendo tipos de datos, variables, aritmética, constantes, control de flujo, entradas/salidas, temporización y comunicación por puerto serie.

1. Estructura del Programa

Los programas de Arduino, llamados sketches, tienen dos funciones principales obligatorias:

1.1 setup()

- Se ejecuta una sola vez al iniciar el programa.
- Configura el hardware y las variables iniciales.

```
void setup() {
  pinMode(13, OUTPUT); // Configurar el pin 13 como salida
}
```

1.2 loop()

- Se ejecuta de forma repetitiva mientras la placa esté encendida.
- Contiene el comportamiento principal del programa.

cpp

Copiar código

```
void loop() {
  digitalWrite(13, HIGH); // Encender LED
  delay(1000);            // Esperar 1 segundo
  digitalWrite(13, LOW);  // Apagar LED
  delay(1000);            // Esperar 1 segundo
}
```

2. Variables y Tipos de Datos

2.1 Tipos de Datos

Tipo	Tamaño	Rango	Uso
`int`	2 bytes	-32,768 a 32,767	Números enteros
`unsigned int`	2 bytes	0 a 65,535	Números enteros positivos
`long`	4 bytes	-2,147,483,648 a 2,147,483,647	Números enteros grandes
`unsigned long`	4 bytes	0 a 4,294,967,295	Enteros positivos grandes
`float`	4 bytes	Decimales con 6-7 dígitos de precisión	Números decimales
`char`	1 byte	-128 a 127	Caracteres (letras, símbolos)
`boolean`	1 byte	`true` o `false`	Valores lógicos

2.2 Declaración de Variables

int ledPin = 13; // Variable entera

float temperatura = 23.5; // Variable decimal

boolean encendido = true; // Variable booleana

2.3 Constantes

Usa const para definir valores que no cambian durante la ejecución.

const int ledPin = 13; // Pin fijo para LED

3. Operadores Aritméticos

Arduino admite los operadores comunes de C/C++:

Operador	Uso	Ejemplo
`+`	Suma	`a + b`
`-`	Resta	`a - b`
`*`	Multiplicación	`a * b`
`/`	División	`a / b`
`%`	Módulo (residuo)	`a % b`

int resultado = (5 + 3) * 2; // resultado = 16

4. Control de Flujo

4.1 Condicionales

```
if (condición) {
  // Código si la condición es verdadera
} else {
  // Código si la condición es falsa
}
```

Ejemplo

```
if (digitalRead(2) == HIGH) {
  digitalWrite(13, HIGH); // Enciende el LED
} else {
  digitalWrite(13, LOW);  // Apaga el LED
}
```

4.2 Bucles

- **for**: Ejecuta un bloque de código un número determinado de veces.

```
for (int i = 0; i < 10; i++) {
  Serial.println(i);
}
```

while: Ejecuta mientras la condición sea verdadera.

```
while (digitalRead(2) == LOW) {
  delay(100);
}
```

5. Entradas y Salidas

5.1 Configuración de Pines

- pinMode(pin, mode): Configura un pin como entrada o salida.
    - mode: INPUT, OUTPUT, INPUT_PULLUP.

5.2 Control de Pines Digitales

- digitalWrite(pin, value): Establece el estado de un pin digital (HIGH o LOW).
- digitalRead(pin): Lee el estado de un pin digital (0 o 1).

5.3 Pines Analógicos

- analogRead(pin): Lee un valor entre 0 y 1023 (entrada analógica).
- analogWrite(pin, value): Envía una señal PWM entre 0 y 255 (pseudo salida analógica).

6. Temporización

Arduino incluye funciones para manejar tiempos:

- delay(ms): Pausa la ejecución durante ms milisegundos.
- millis(): Devuelve el tiempo en milisegundos desde que Arduino se encendió.

Ejemplo:

```
unsigned long tiempoAnterior = 0;
const int intervalo = 1000;


void loop() {
  unsigned long tiempoActual = millis();
  if (tiempoActual - tiempoAnterior >= intervalo) {
    digitalWrite(13, !digitalRead(13)); // Cambia el estado del LED
    tiempoAnterior = tiempoActual;
  }
}
```

7. Comunicación Serie

La comunicación serie permite enviar y recibir datos entre Arduino y un ordenador u otros dispositivos.

7.1 Inicializar el Puerto Serie

```
void setup() {
  Serial.begin(9600); // Configura la velocidad a 9600 baudios
}
```

7.2 Enviar Datos

- Serial.print(dato): Imprime datos sin salto de línea.
- Serial.println(dato): Imprime datos con salto de línea.

Ejemplo:

```
void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue); // Enviar valor al monitor serie
  delay(500);
}
```

7.3 Recibir Datos

- Serial.available(): Devuelve el número de bytes disponibles para leer.
- Serial.read(): Lee el primer byte disponible.

Ejemplo:

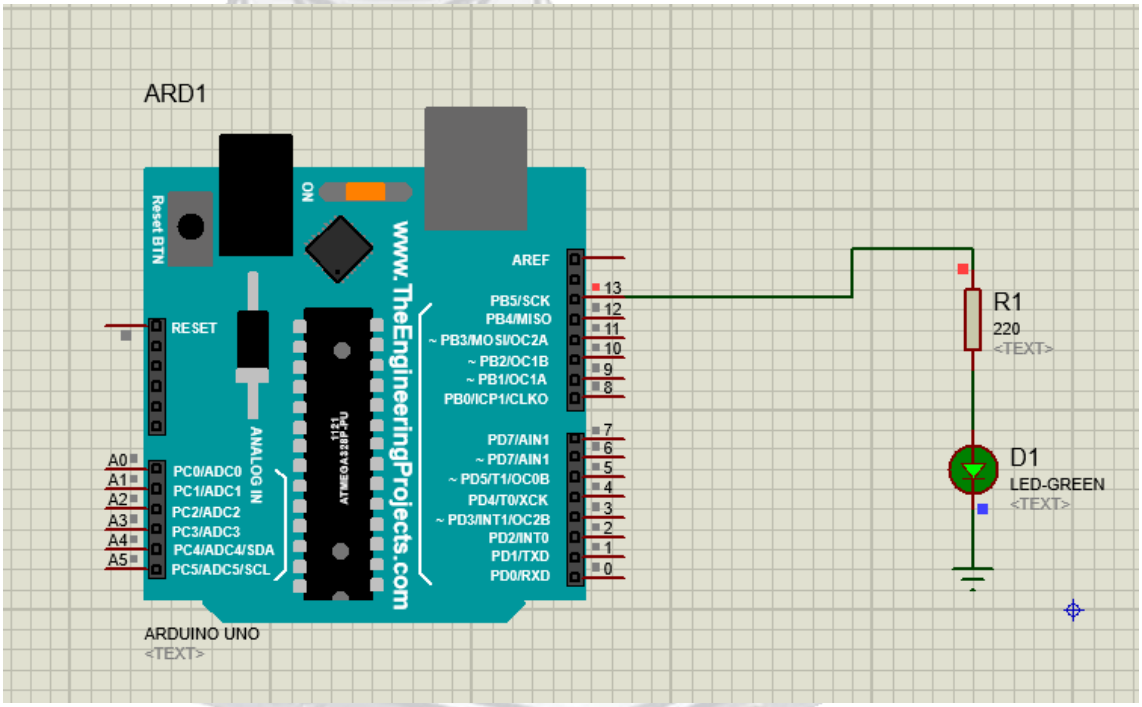
```
void loop() {
  if (Serial.available() > 0) {
    char dato = Serial.read(); // Lee un carácter del puerto serie
    Serial.print("Recibido: ");
    Serial.println(dato);
  }
}
```

```
void loop() {
  if (Serial.available() > 0) {
    char dato = Serial.read(); // Lee un carácter del puerto serie
    Serial.print("Recibido: ");
    Serial.println(dato);
  }
}
```

PROGRAMA EJEMPLO

TEMA 1 : INTRODUCCION

1. **Parpadeo de un Led**



Parpadeo de un LED (Blink).

```
// Definir el pin del LED
const int ledPin = 13; // Pin 13, donde está el LED interno o externo

void setup() {
  // Configurar el pin 13 como salida
  pinMode(ledPin, OUTPUT);
}
```

```
void loop() {
  digitalWrite(ledPin, HIGH); // Encender el LED
  delay(1000);              // Esperar 1 segundo
  digitalWrite(ledPin, LOW);  // Apagar el LED
  delay(1000);              // Esperar 1 segundo
}
```