CePETel
Sindicato de los Profesionales de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Agosto 2021

Introducción a Sistemas Embebidos con Arduino

Coordinador CePETel: Ing. Daniel Herrero
Coordinadores Curso: Mg.Ing. Mayer, Roberto Osvaldo.
Ing. Paradiso, Juan Carlos.
Ing. Crivelli, Marcelo
Docente: Ing. D'Angiolo, Federico Gabriel fgdangiolo@gmail.com

Planificación

- **Introducción a los Sistemas Embebidos**
- Introducción a la Programación mediante Arduino
- Herramientas de Programación para Arduino.
- Hardware básico para interacción con Arduino.
- Interacción entre Sensores y Arduino.
- Interacción entre Actuadores y Arduino.
- Visualización mediante salidas de Arduino.
- Módulos de Comunicaciones.
- Concepto de Interrupción.
- Integración de Conceptos.

Introducción a los Sistemas Embebidos

- **¿Qué es un Sistema Embebido? Relación entre Sistemas Embebidos y Arduino**
- Conceptos de Software y Hardware Libre.
- ¿Qué es Arduino?, descripción de placas: Arduino UNO, pines de I/O digitales y analógicos.
- Instalación de I.D.E. Diferentes modelos de placas: Arduino Uno, Mini, Nano, Mega, Due, STM32, ESP8266/ESP32.

¿Qué es un Sistema Embebido?

Se conoce como **sistema embebido** a un sistema está diseñado **específicamente para una función en particular pero**, desarrollado bajo **Hardware, Firmware y Software**.

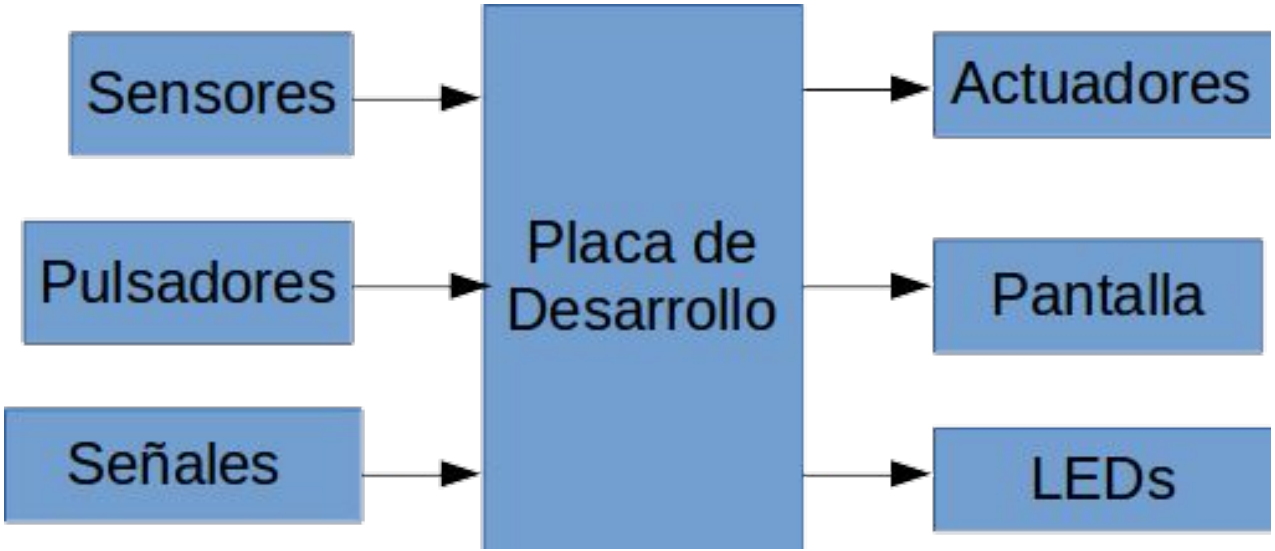
A diferencia de los sistemas computacionales de oficina y laptops, estos sistemas solucionan un problema específico y están dispersos en todos los ambientes posibles de la vida cotidiana. **[1]**

Un usuario puede tomar decisiones sobre la **funcionalidad** de un equipo, pero **no la puede cambiar simplemente modificando el Hardware y/o el Software**.

¿Qué es un Sistema Embebido?

[57]

¿Qué es un Sistema Embebido?



¿Qué es un Sistema Embebido?

Otro ejemplo, un **termostato digital** tiene un una **función dedicada al monitoreo y control** de la **temperatura circundante**. El usuario tiene **opciones** para **configurar las temperaturas** bajas y altas deseadas, **pero no puede cambiar su funcionalidad para que funcione de otra manera.**

https://www.youtube.com/watch?v=YWmAx3lwsZ4

Una de las ventajas de los **Sistemas Embebidos** es que, para actualizarlos, no es necesario modificar todo el sistema, simplemente **actualizando el Firmware**, se logra el objetivo. Esto se da actualmente en los **TVs, osciloscopios y modems**, dado que necesitan actualizar la información que intercambian, entre otra cosas.

¿Qué es un Sistema Embebido?

Entonces.....**¿qué diferencias existe entre una PC y un Sistema Embebido?**

¿Qué es un Sistema Embebido?

Podemos empezar pensando en las **características de una PC**:

- Tiene Microprocesador
- Una memoria primaria que contiene RAM, ROM y Cache.
- Una gran memoria secundaria como disco rígido o disco de estado sólido
- Unidades de E / S como pantalla, teclado, mouse y otros.
- Sistemas Operativos: Linux, Mac, Windows, etc.
- Software de Aplicación

[1]

¿Qué es un Sistema Embebido?

Características de un Sistema Embebido:

- Incorpora hardware que incluye el núcleo y las E / S necesarias para una función específica
- Incorpora el software de la aplicación principal en Flash incorporado.
- Incorpora un **Sistema Operativo en Tiempo Real** (RTOS) que **supervisa las tareas del software de la aplicación** que se ejecutan en hardware. **[1]**

Además hay que tener en cuenta que los sistemas de **memoria y de procesamiento son mucho más limitados que los de una PC.**

¿Qué es un Sistema Embebido?

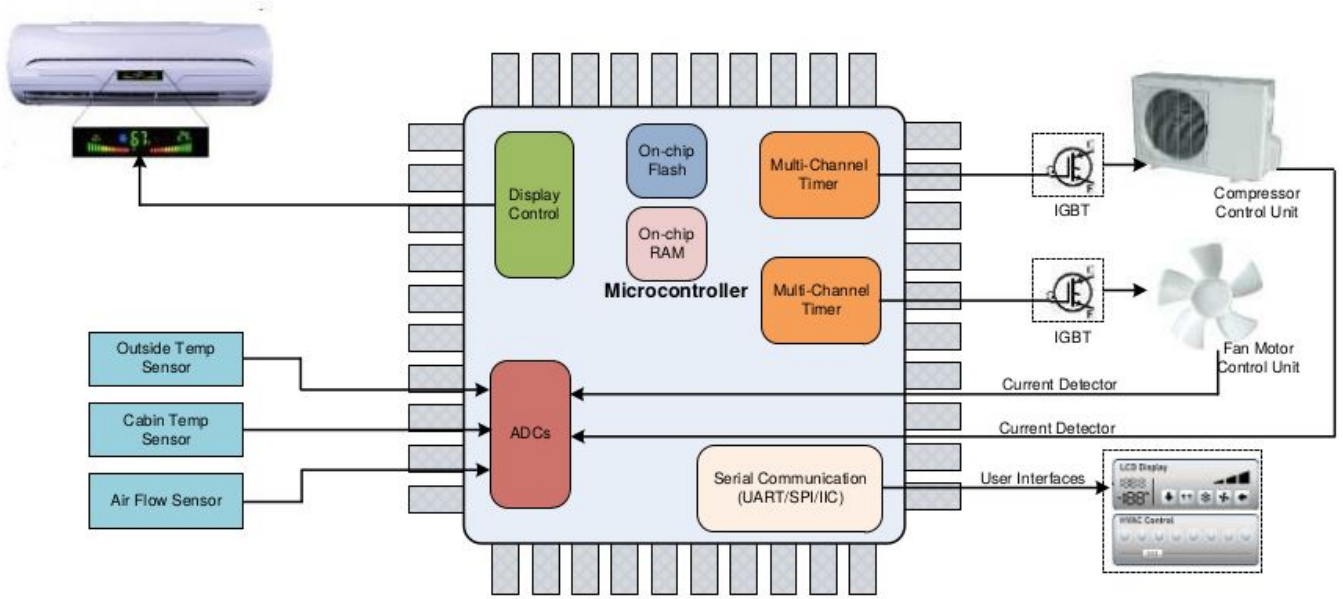
1) Ejemplo: Aire Acondicionado

El trabajo principal es **enfriar el aire interior**. Los acondicionadores de aire **controlan y regulan la temperatura del aire mediante un termostato.**

Para llevar a cabo esta tarea, estos sistemas incluyen en su interior, un **microcontrolador**, el cual actúa como “cerebro”, en el sentido de que puede sensar la temperatura del sistema (algunos aires acondicionados) y enfriar el aire interior. Además, permite mostrar la información en algún display y recibir información del algún control remoto. En la siguiente figura se observa esto:

¿Qué es un Sistema Embebido?

Ejemplo: Aire Acondicionado



[1]

¿Qué es un Sistema Embebido?

De la imagen anterior se pueden ver varias cosas:

1) El microcontrolador puede sensar la temperatura y el flujo de aire.
2) Envían información a un display.
3) Reciben información de un control Remoto.
4) Activan interfaces de potencia (transistores), para activar la ventilación.

Para todo esto, es esencial usar módulos que ya vienen integrados en el microcontrolador, tales como: **ADC** (toma la información de los sensores), **PWM** (Varía la velocidad de los motores), **UART/I^{2}C** (permite la comunicación con displays), etc.

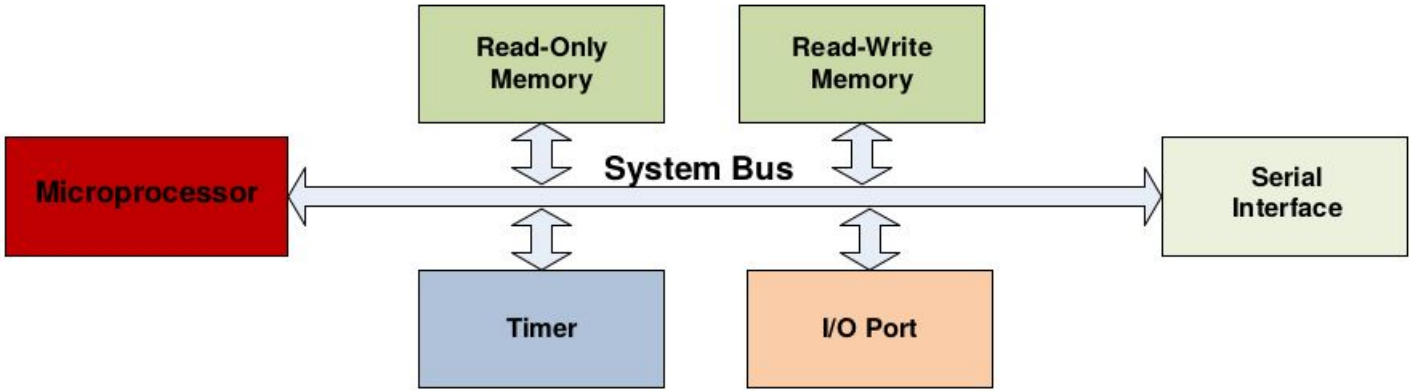
En un **microcontrolador**, aparte de los módulos mencionados, generalmente se encuentra un **microprocesador** o un **procesador embebido**.

¿Qué es un Sistema Embebido?

Entonces.....**¿qué diferencias existe entre un Microcontrolador y un Microprocesador?**

¿Qué es un Sistema Embebido?

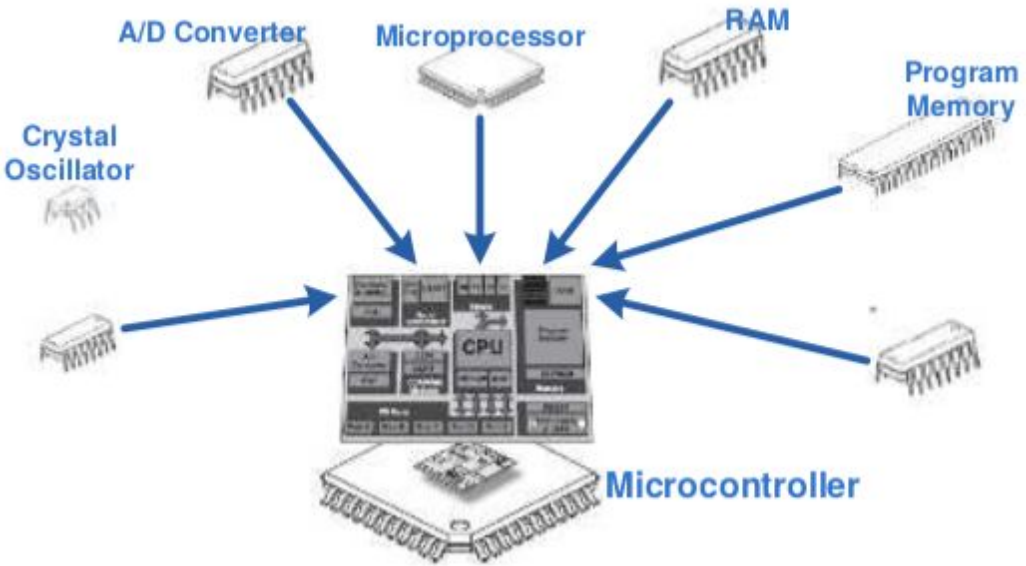
Un **microprocesador** es una **Unidad Central de Procesamiento** de computadora digital de propósito general. Lo que muchas veces se llama, **CPU**. Para que el sistema sea una ***microcomputadora***, además se le debe agregar memoria, como se muestra a continuación:



[1]

¿Qué es un Sistema Embebido?

A un **microcontrolador** lo podemos pensar de la siguiente forma:



[1]

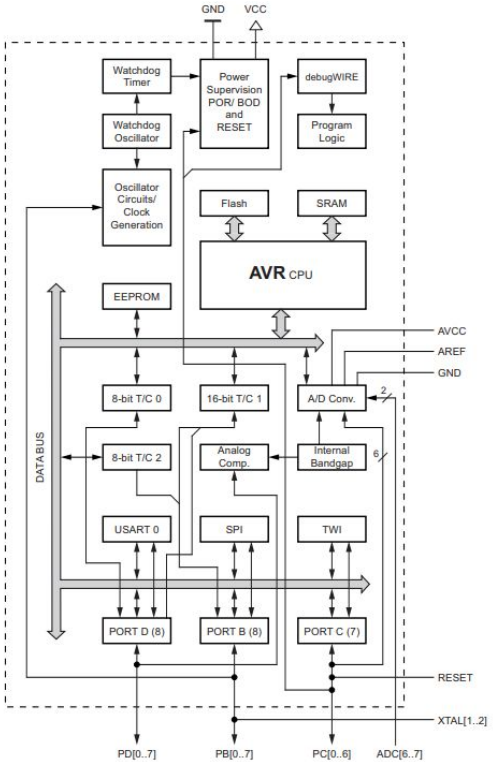
¿Qué es un Sistema Embebido?

Algunos **microcontroladores** actuales, pueden ser: PIC, AVR (8 bits), LPC4337, STM32F103 (32 bits). Estos últimos contienen un procesador **ARM**, siendo uno de los más importantes actualmente. Dentro de **ARM** se pueden encontrar modelos Cortex M3, Cortex M4, etc.

A continuación veremos la arquitectura de un **microcontrolador ATMEGA328**, que es el que utiliza **Arduino**.

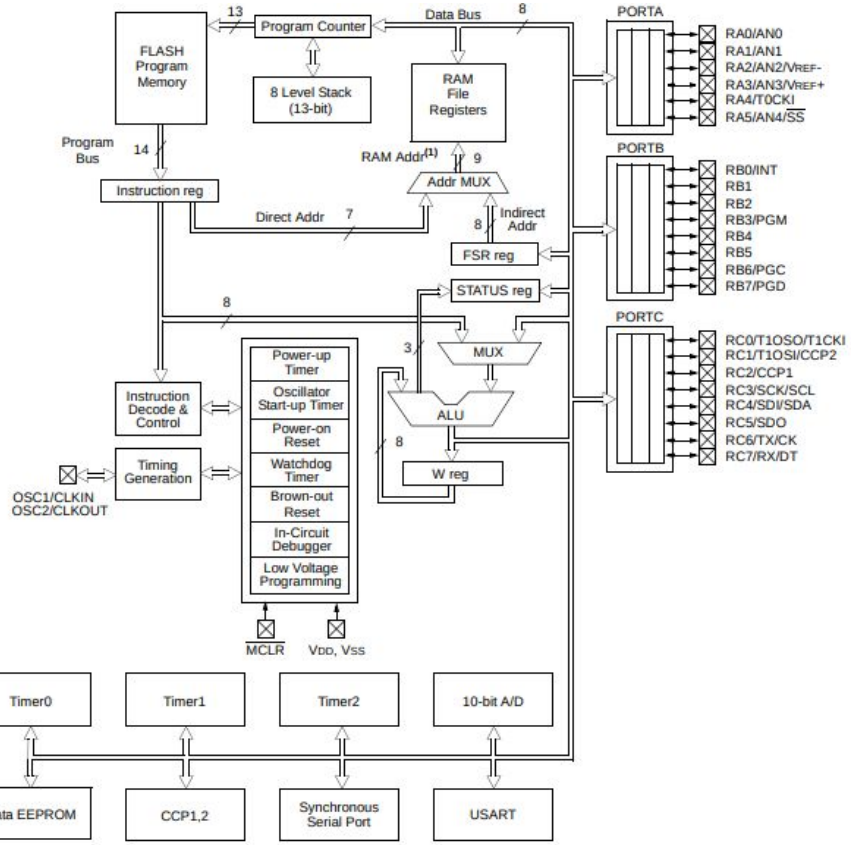
Luego veremos la arquitectura de un **microcontrolador PIC16F877** para compararlo con el anterior.

¿Qué es un Sistema Embebido?



[2]

¿Qué es un Sistema Embebido?

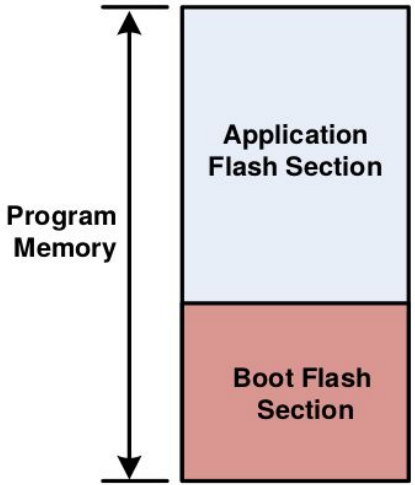


[3]

¿Qué es un Sistema Embebido?

Memoria de Programa y Memoria de Datos.

Como vimos, los microcontroladores poseen periféricos y memorias. En particular, existe la ***memoria de programa***, que es donde queda alojado el ***programa que el microcontrolador debe ejecutar***.



[1]

¿Qué es un Sistema Embebido?

En la figura anterior se puede ver que la Memoria de Programa se divide en dos partes, siendo una de ellas (Boot), el lugar físico donde reside el ***código de arranque.*** Esto resulta importante en ***Arduino***, donde se usa el ***Bootloader***.

Por otro lado tenemos la ***memoria de datos***, que es donde se pueden ***alojar datos*** durante la ***ejecución del programa***. Estos ***datos se pierden*** cuando se ***quita la alimentación*** del microcontrolador.

Introducción a los Sistemas Embebidos

- ¿Qué es un Sistema Embebido? Relación entre Sistemas Embebidos y Arduino
- **Conceptos de Software y Hardware Libre.**
- ¿Qué es Arduino?, descripción de placas: Arduino UNO, pines de I/O digitales y analógicos.
- Instalación de I.D.E. Diferentes modelos de placas: Arduino Uno, Mini, Nano, Mega, Due, STM32, ESP8266/ESP32.

Conceptos de Software y Hardware Libre.

Hoy en día, dado el gran avance en el Software y las grandes comunidades que se van formando, el Software libre se ve en ascenso.

El **software libre** ofrece al usuario cuatro libertades: **libertad de uso**, **de estudio y modificación**, **de distribución**, **y de redistribución de las versiones modificadas**.

Existen **licencias** que las garantizan y que dan una cobertura legal, como por ejemplo la licencia **GNU GPL**. El **hardware libre** toma estas mismas ideas del software libre para aplicarlas en su campo. **[43]**

Algunos ejemplos de Hardware Libre son: Arduino, Raspberry PI, e-puck, Reprap

Introducción a los Sistemas Embebidos

- ¿Qué es un Sistema Embebido? Relación entre Sistemas Embebidos y Arduino
- Conceptos de Software y Hardware Libre.
- **¿Qué es Arduino?, descripción de placas: Arduino UNO, pines de I/O digitales y analógicos.**
- Instalación de I.D.E. Diferentes modelos de placas: Arduino Uno, Mini, Nano, Mega, Due, STM32, ESP8266/ESP32.

Introducción a los Sistemas Embebidos

¿Qué es Arduino?

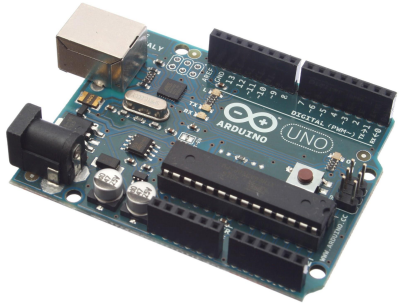
Introducción a los Sistemas Embebidos

Arduino es una **plataforma electrónica** de **código abierto** (open source), basada en **hardware y software** que son sencillos de usar.

Las placas Arduino pueden leer entradas (sensores de luz, teclas, teclados táctiles), procesar estas entradas para luego, mediante sus salidas, activar algunos actuadores tales como motores u otro tipo de salidas como por ejemplo LEDs, balizas, pantallas LC

Para realizar esta lectura, procesamiento y activación, resulta importante hacerlo mediante el **lenguaje de programación Arduino** y el **Software Arduino (IDE)**.

https://www.arduino.cc/en/Guide/Introduction



Introducción a los Sistemas Embebidos

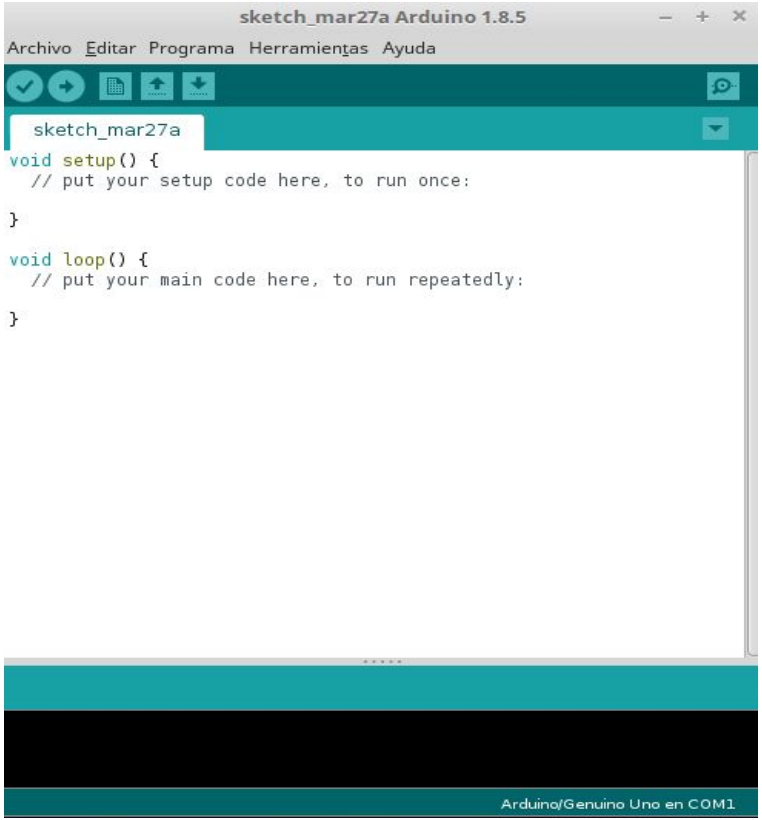
Software de Arduino:

Para poder desarrollar código, generalmente se usa un **IDE** (Entorno de Desarrollo Integrado), el cual **contiene un editor de texto para escribir código**, **un área de mensajes**, **una consola de texto**, **una barra de herramientas con botones para funciones comunes y una serie de menús**.

Una vez escrito el código, se compila y se lo envía al **hardware Arduino** para **cargar programas y comunicarse con ellos.**

https://www.arduino.cc/en/Guide/Environment

Introducción a los Sistemas Embebidos



Introducción a los Sistemas Embebidos

Algunas cosas importantes a la hora de trabajar con Arduino:

Los programas escritos con el ***software Arduino*** (IDE) se los llama ***sketch***.

Estos ***sketch*** se escriben en el editor de texto y se los guarda con la extensión **.ino**.

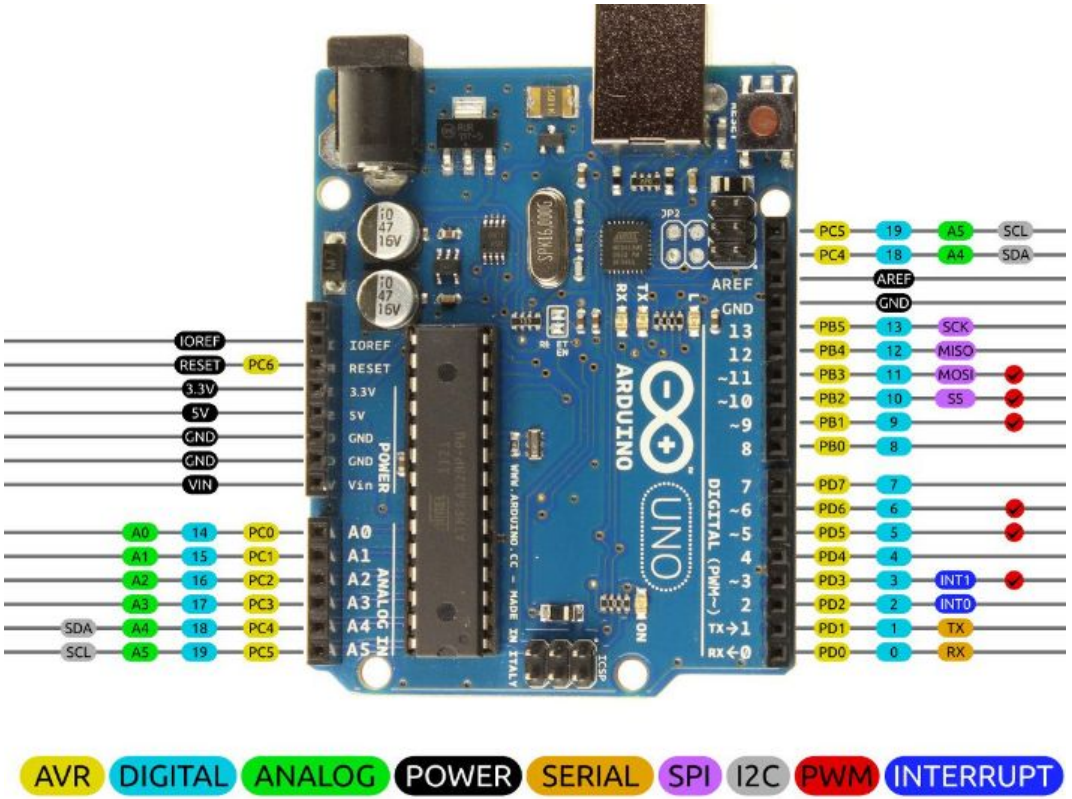
La consola muestra la salida de texto del software Arduino (IDE), incluidos mensajes de error completos y otra información.

La esquina inferior derecha de la ventana muestra la placa configurada y el puerto serie. Los botones de la barra de herramientas le permiten verificar y cargar programas, crear, abrir y guardar ***sketch*** y abrir el monitor en serie.

https://www.arduino.cc/en/Guide/Environment

Pinout Arduino UNO

Distribución de los pines



Introducción a los Sistemas Embebidos

- ¿Qué es un Sistema Embebido? Relación entre Sistemas Embebidos y Arduino
- Conceptos de Software y Hardware Libre.
- ¿Qué es Arduino?, descripción de placas: Arduino UNO, pines de I/O digitales y analógicos.
- **Instalación de IDE. Diferentes modelos de placas: Arduino Uno, Mini, Nano, Mega, Due, STM32, ESP8266/ESP32.**

Instalación IDE

Para poder instalar el IDE de Arduino conviene bajarlo de la página oficial:

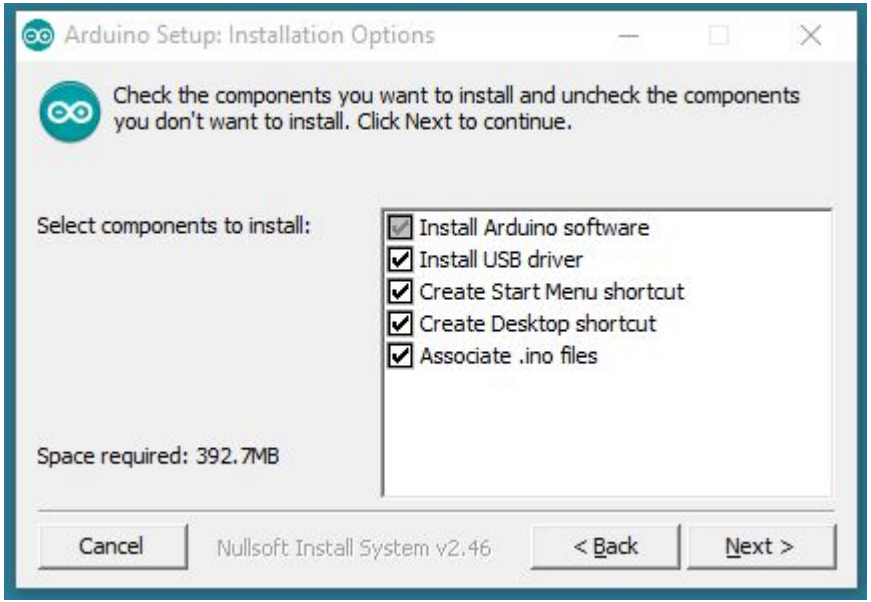
https://www.arduino.cc/en/software

Como se puede ver de la página oficial, se puede usar tanto en **Windows** como en **Linux** como en **MAC OS.**

Luego de haberlo bajado, se puede comenzar con la instalación:

Instalación IDE

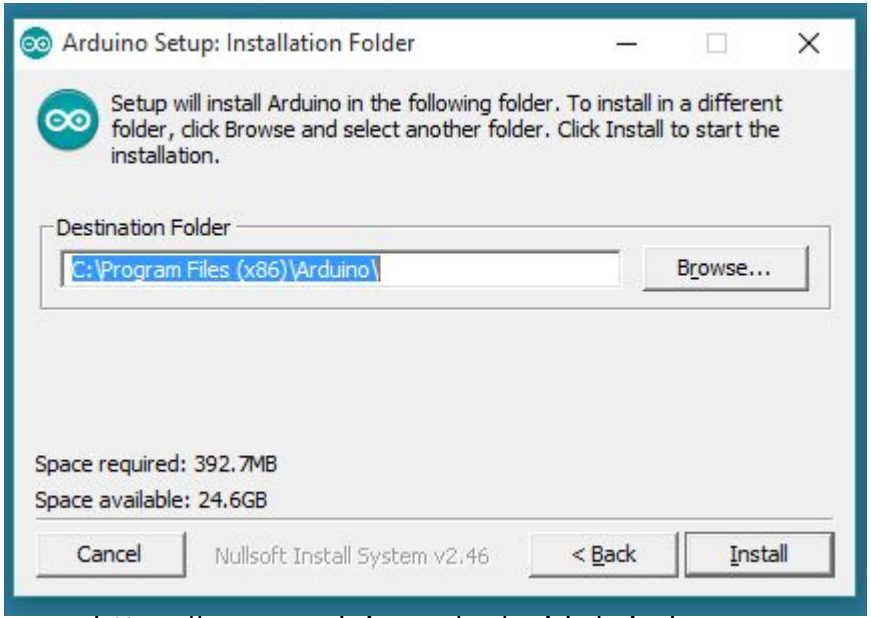
Windows



https://www.arduino.cc/en/guide/windows

Instalación IDE

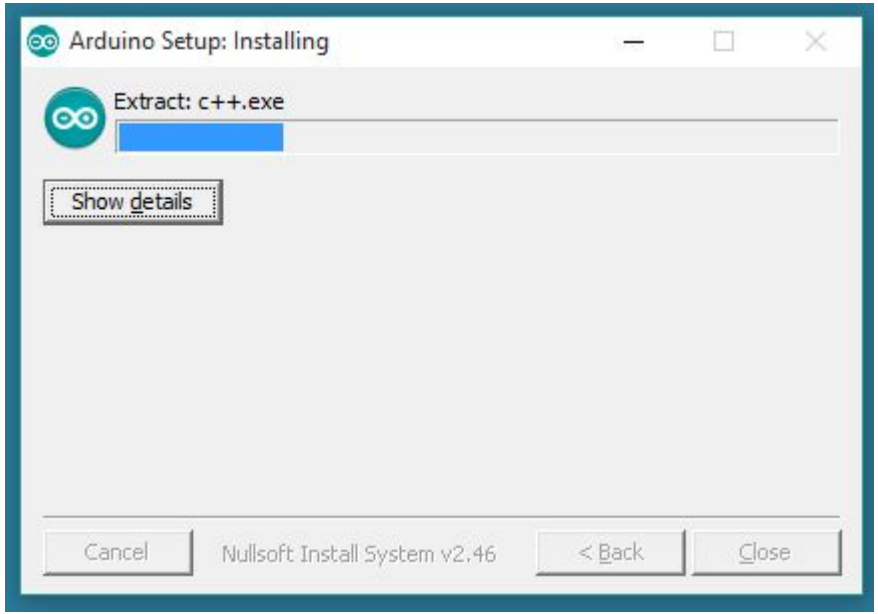
Windows



https://www.arduino.cc/en/guide/windows

Instalación IDE

Windows



https://www.arduino.cc/en/guide/windows

Instalación IDE

Importante:

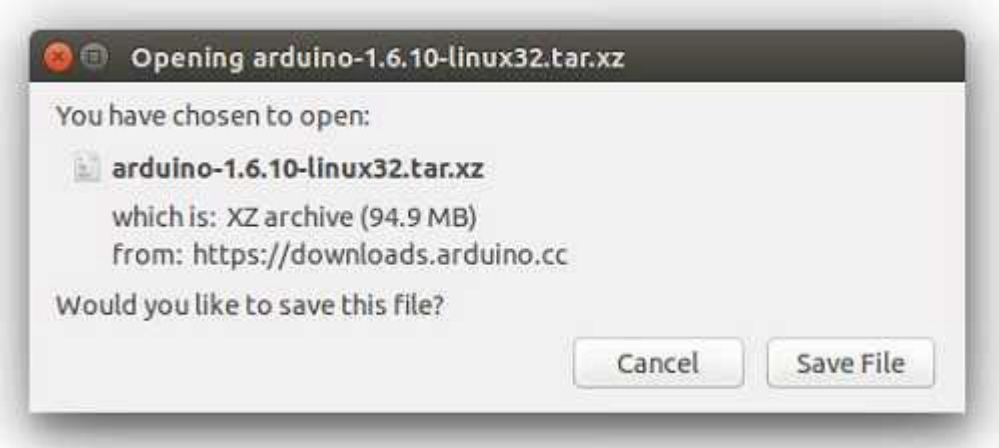
Bajando la versión **.exe del IDE, se instalan todos los drivers.**

Si se baja la versión **.zip del IDE**, hay que **instalar manualmente a los drivers.**

https://www.arduino.cc/en/guide/windows

Instalación IDE

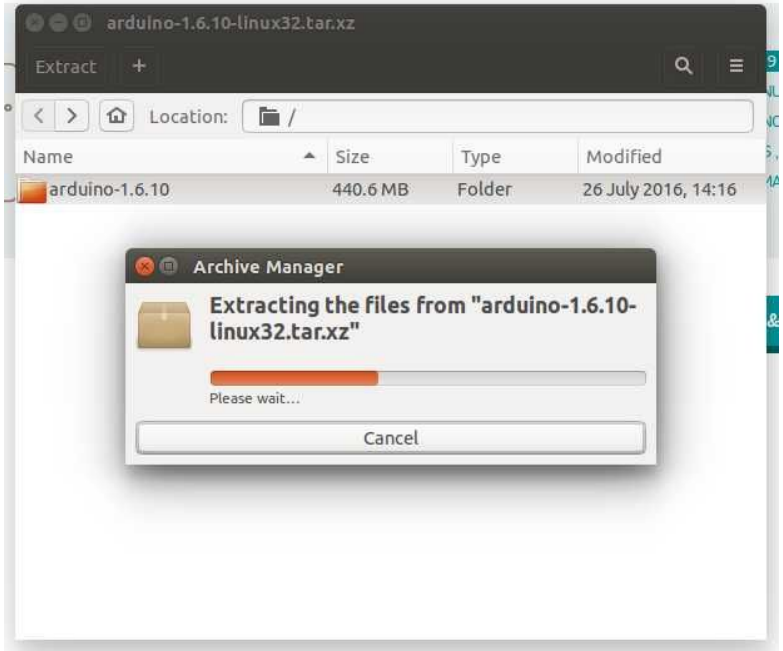
Linux



https://www.arduino.cc/en/Guide/Linux

Instalación IDE

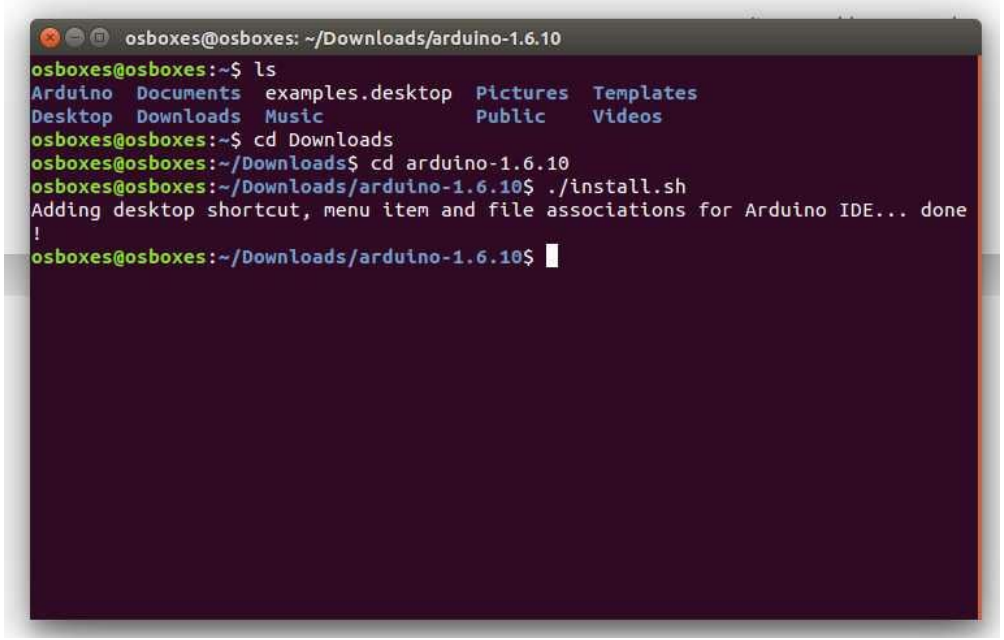
Linux



https://www.arduino.cc/en/Guide/Linux

Instalación IDE

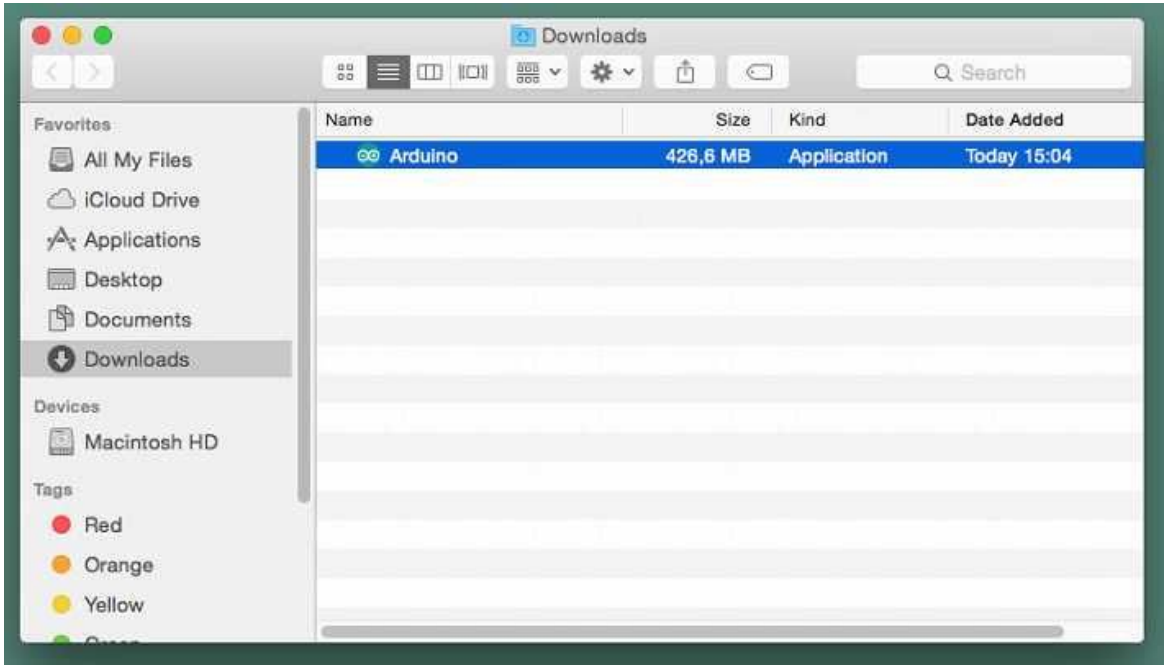
Linux



https://www.arduino.cc/en/Guide/Linux

Instalación IDE

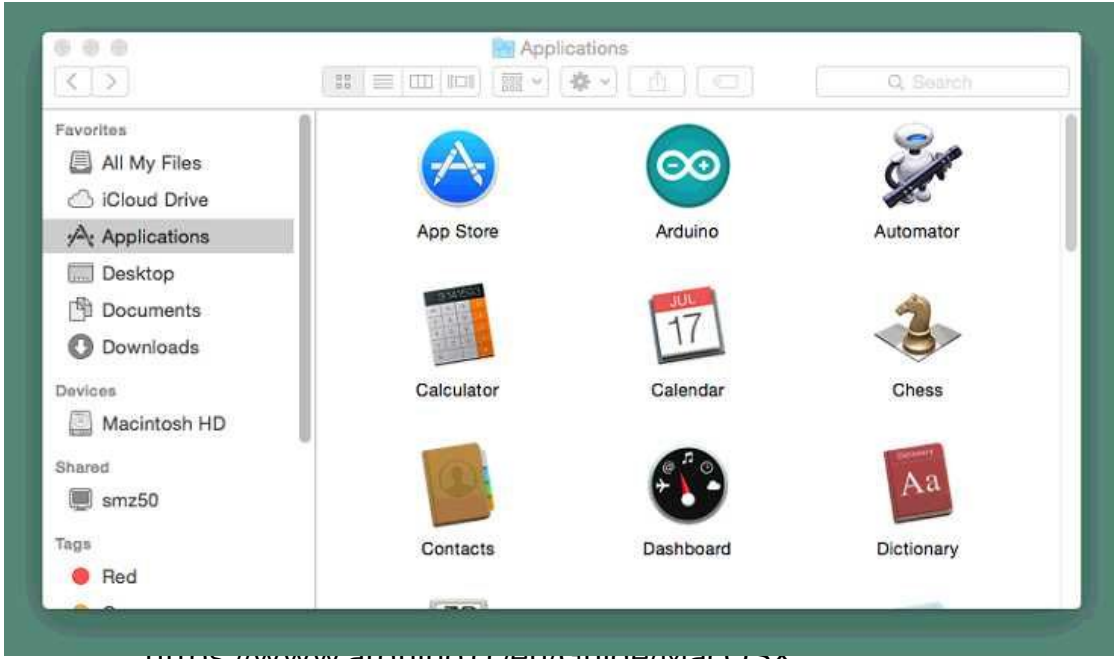
MAC



https://www.arduino.cc/en/Guide/MacOSX

Instalación IDE

MAC



https://www.arduino.cc/en/Guide/MacOSX

Introducción a los Sistemas Embebidos

Luego de instalar el **IDE**, ir a **herramientas** y seleccionar la **Placa** (Arduino UNO) y el **puerto**:

Com 1, 2, 3,..., en **Windows**.

/dev/ttyACM0 en **Linux.**

Introducción a los Sistemas Embebidos

A modo de prueba, podemos hacer lo siguiente:

Archivo -> **Ejemplos** -> **01.Basics** -> **Blink**

Este ejemplo produce un **parpadeo del LED** que trae la placa. Más adelante se lo estudiará.

Si se ve que el LED parpadea cada un segundo, tanto la placa como la comunicación funcionan correctamente.

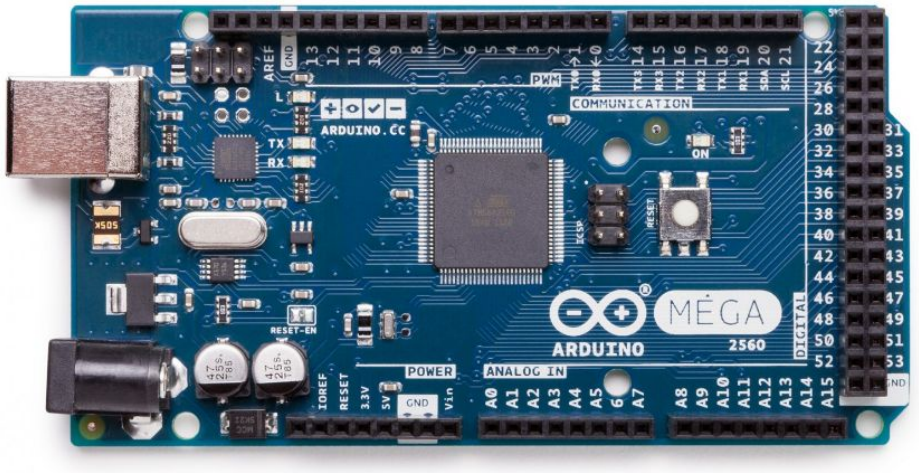
Introducción a los Sistemas Embebidos

¿Existen otras Placas aparte de Arduino UNO?

Plataformas de Desarrollo
8 bits

Arduino UNO

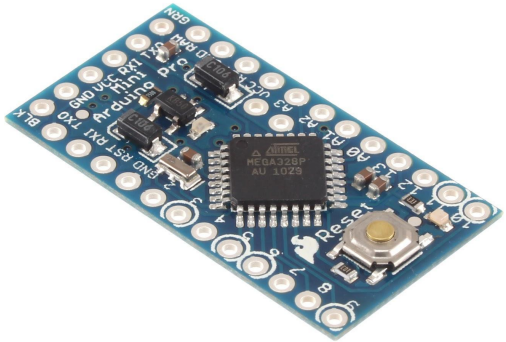
Arduino MEGA



Plataformas de Desarrollo
8 bits

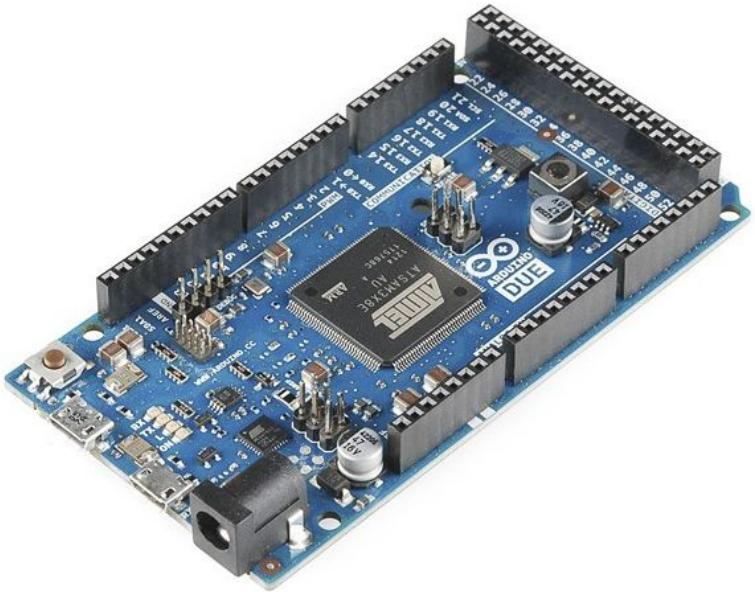
Arduino NANO

Arduino Pro Mini



Plataformas de Desarrollo 32 bits

Arduino DUE



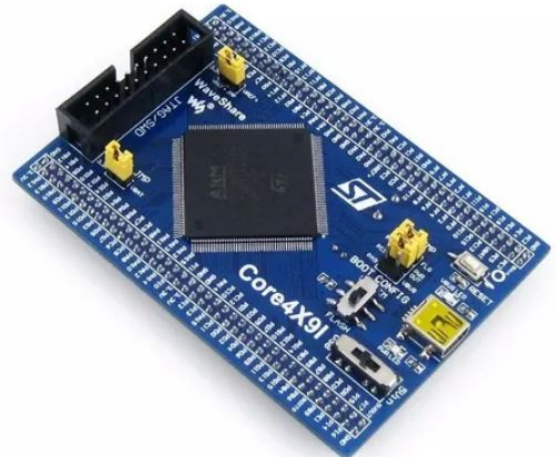
Arduino Galileo

Plataformas de Desarrollo 32 bits

STM32

STM32F103 - Cortex-M3

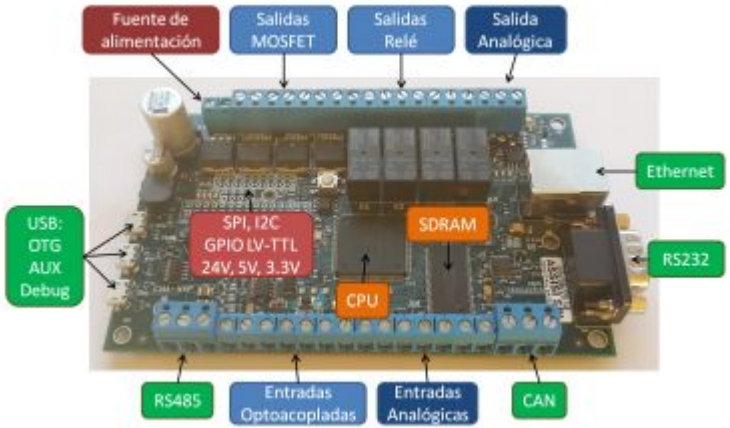
STM32



STM32f429 - Cortex M4

Plataformas de Desarrollo 32 bits

CIAA



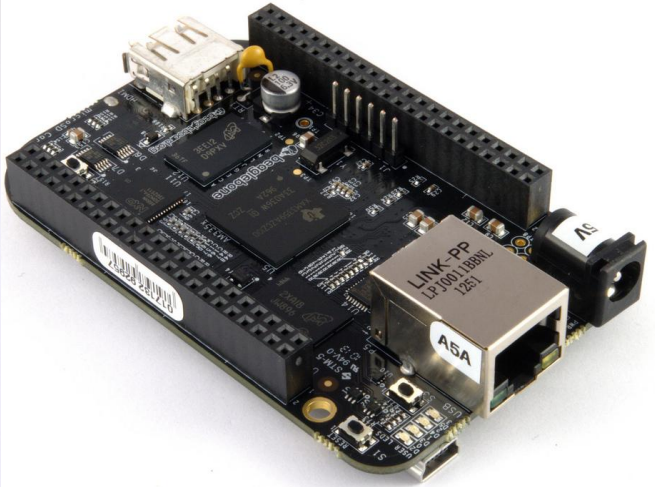
[2]

EDU-CIAA

[2]

Plataformas de Desarrollo 32 bits

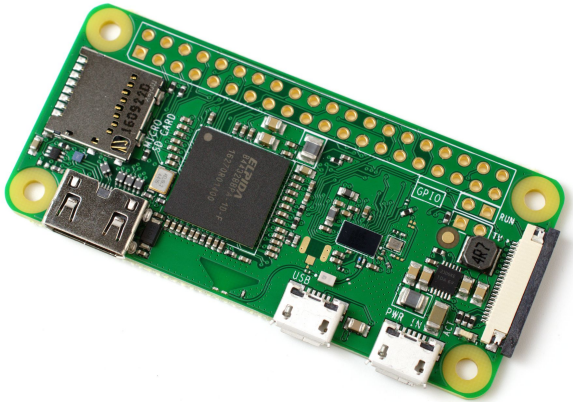
Raspberry Pi 3



Beaglebone Black

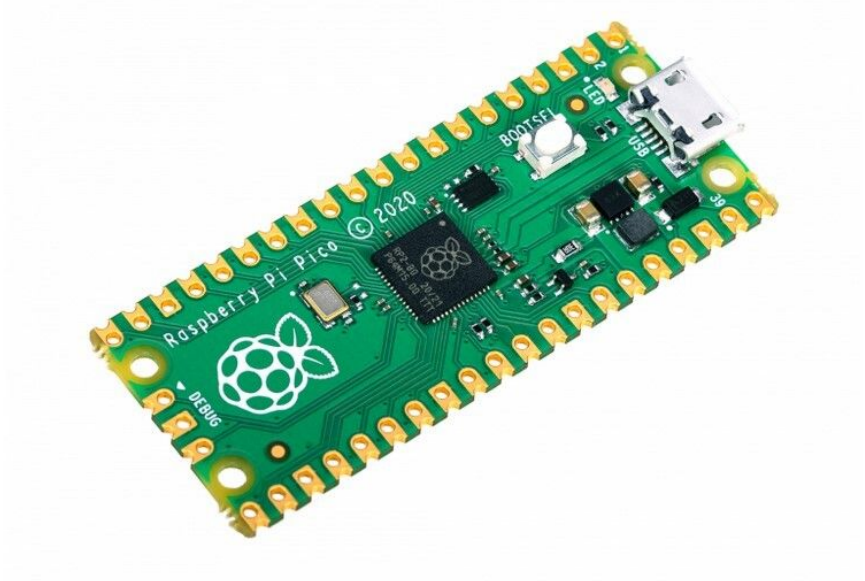
Plataformas de Desarrollo 32 bits

Raspberry Pi 4



Raspberry PI Zero W

Plataformas de Desarrollo 32 bits



Raspberry PI Pico

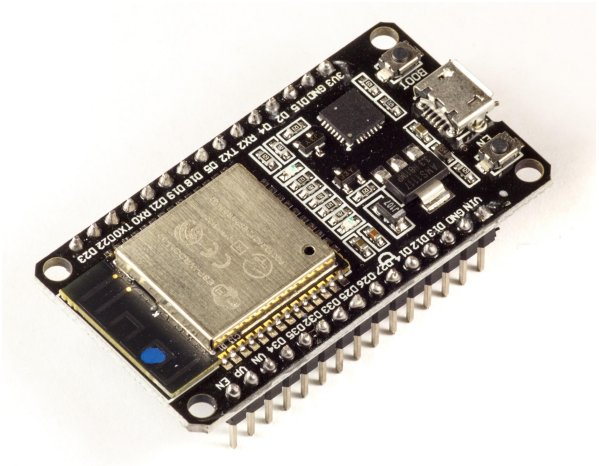
Plataformas de Desarrollo con acceso WiFi

Mini wemos. ESP-8266

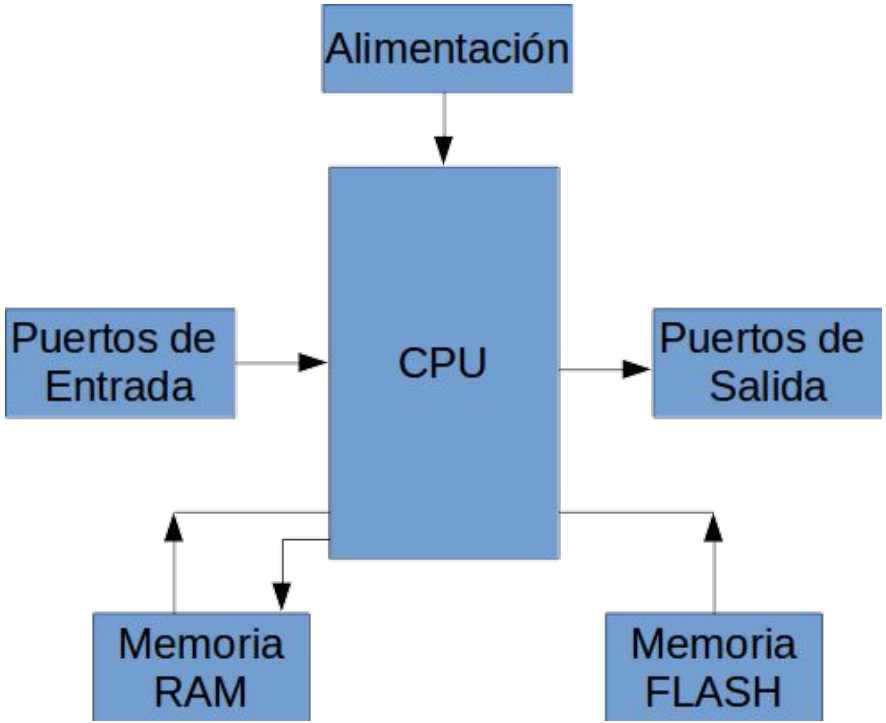
NodeMCU. ESP-8266

Plataformas de Desarrollo con acceso WiFi

ESP32



Arquitectura básica de una Placa de Desarrollo



Planificación

- Introducción a los Sistemas Embebidos
- **Introducción a la Programación mediante Arduino**
- Herramientas de Programación para Arduino.
- Hardware básico para interacción con Arduino.
- Interacción entre Sensores y Arduino.
- Interacción entre Actuadores y Arduino.
- Visualización mediante salidas de Arduino.
- Módulos de Comunicaciones.
- Concepto de Interrupción.
- Integración de Conceptos.

Planificación

- **Configuración y puesta en marcha. Primer programa en Arduino, funciones setup() y loop()**
- Instrucciones pinMode() y digitalWrite(). Manejo del led on-board. Función delay().
- Variables: tipos de variables, rango, signo, definición, asignación. Uso de variables en el sketch. Ámbito de las variables: variables locales y globales. Desbordamiento de variables. Operaciones matemáticas.
- Limitaciones de la función delay(). Funciones millis() y micros().
- Técnicas para optimización de memoria y tiempos de ejecución.

Algunos conceptos previos...

Previamente a la puesta en marcha con **Arduino UNO**, veremos algunos conceptos previos que nos servirán para comprender cómo trabajar con **Sistemas Embebidos.**

- **Concepto de numeración**
- **Posiciones de memoria**
- **¿Qué es un LED?**

Algunos conceptos previos...

Las computadoras trabajan en base a ***números binarios***, esto es, utilizan ***base dos.***

Por ejemplo, un número binario se puede representar como: **0101**

Cada ***dígito*** representa un ***bit.***

Cada número binario se puede representar en otra base, por ejemplo, ***en base diez o en base hexadecimal.***

Ejemplo: 10 (decimal) -> 1010 (binario) -> A (hexadecimal)

A los efectos de la programación que utilizaremos, podremos escribir valores en decimal ya que la computadora se podrá encargar de realizar la conversión.

Algunos conceptos previos...

Las computadoras trabajan en base a ***bits***. Muchas veces se utiliza la palabra ***byte*** la cual se refiere a que: ***1 byte = 8 bits***

Ejemplo: 1 byte = 01101010 1 byte = 10100010

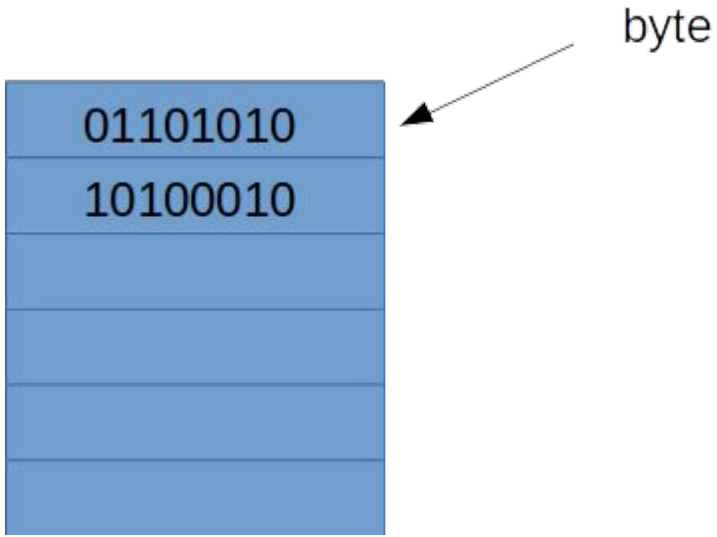
Las posiciones de memorias ocupan ***1 byte.***

Arduino UNO tiene dos memorias:

Flash: ***32K bytes*** (donde se almacena código)

SRAM: 2K bytes (donde se gestionan las variables)

EEPROM: 1K byte (se almacenan variables que deberán permanecer por un largo tiempo)



Algunos conceptos previos...

Para hacer conversiones entre números, usaremos la siguiente tabla

Decimal	Binario	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Algunos conceptos previos...

Ejemplos:

1) Se tiene el número **B2D** en Hexadecimal. ¿Cuál es su representación en binario?

B = 1011; **2** = 0010; **D** = 1101

En binario tenemos entonces: **1011 0010 1101**

2) Se tiene el número **1010 0110 1101** en Binario. ¿Cuál es su representación en Hexadecimal?

1010 = **A**; 0110 = **6**; 1101 = **D**

En hexadecimal tenemos entonces: **A6D**

Algunos conceptos previos...

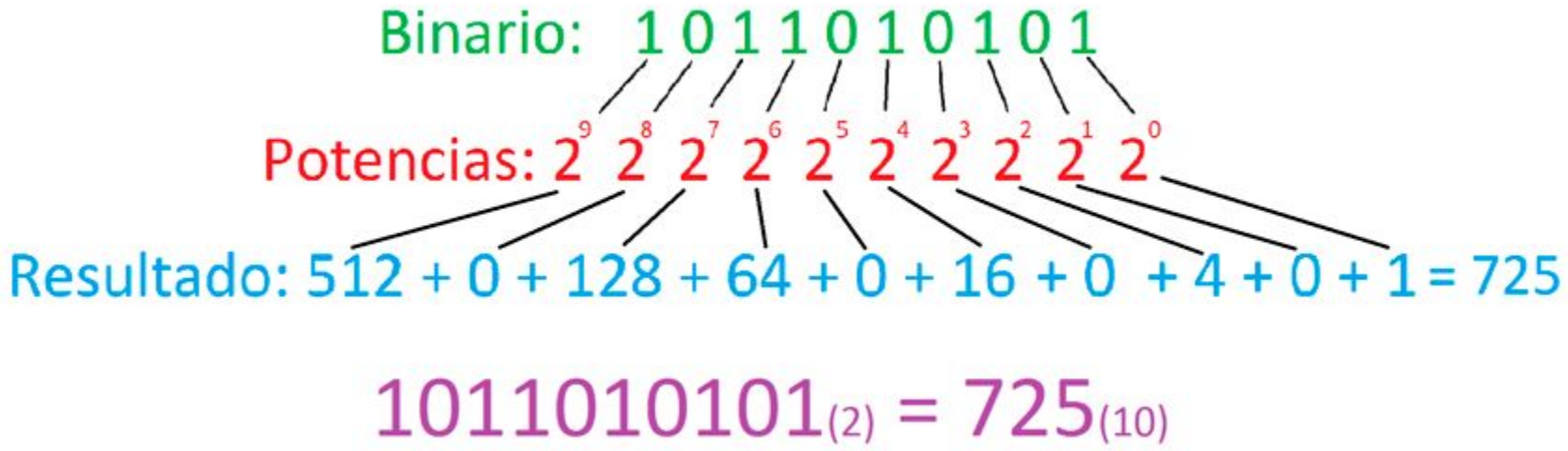
Ejercicios:

1) Se tiene el número **C4F** en Hexadecimal. ¿Cuál es su representación en binario?

2) Se tiene el número **1011 0010 1101** en Binario. ¿Cuál es su representación en Hexadecimal?

Algunos conceptos previos...

Si queremos convertir de **binario a decimal**, podemos proceder de la siguiente forma:



https://cual-es-mi-ip.online/herramientas/conversores-numericos/conversor-binario-a-decimal/

Algunos conceptos previos...

La cuenta anterior la podemos verificar de la siguiente forma:

$$Nro\ decimal = 2^9.1 + 2^8.0 + 2^7.1 + 2^6.1 + 2^5.0 + 2^4.1 + 2^3.0 + 2^2.1 + 2^1.0 + 2^0.1$$

$$Nro\ decimal = 512 + 0 + 128 + 64 + 0 + 16 + 0 + 4 + 0 + 1 = 725$$

Algunos conceptos previos...

Ejercicios:

1) Se tiene el número **0010 0001** y se quiere conocer su equivalencia en decimal.

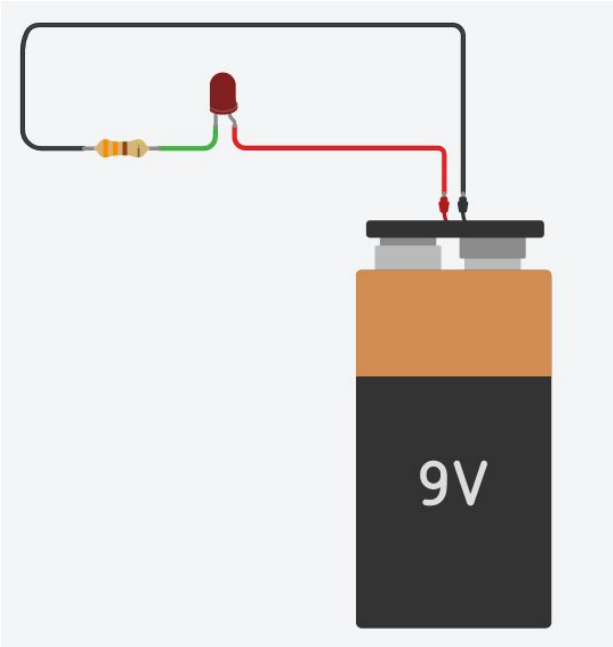
   (Sensor de temperatura DHT 11)

2) Se tiene el número **0001 1010** y se quiere conocer su equivalencia en decimal.

   (Sensor de humedad DHT 11)

Algunos conceptos previos...

Conceptos de circuitos

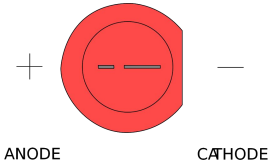


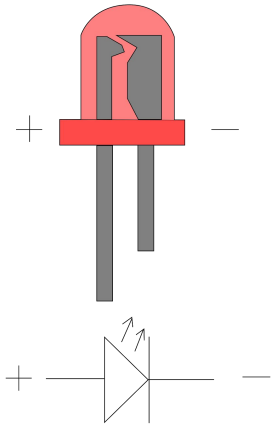
Algunos conceptos previos...

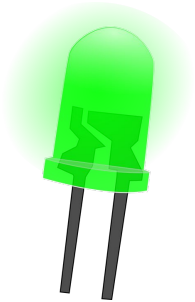
LED: Un LED (Diodo Emisor de Luz), es una fuente de luz constituida por un material semiconductor dotado de dos terminales. El mismo, tiene polaridad. **[11]**

En general, se encienden con una **tensión de 2V** (esto depende del color), y circula por ellos, una **corriente de 20mA.**

Los **LEDs de alto brillo** tienen otra caída de tensión (3V aprox), y consumen más corriente (30 mA aprox).







Algunos conceptos previos...

¿Cómo podemos encender un LED?

Algunos conceptos previos...

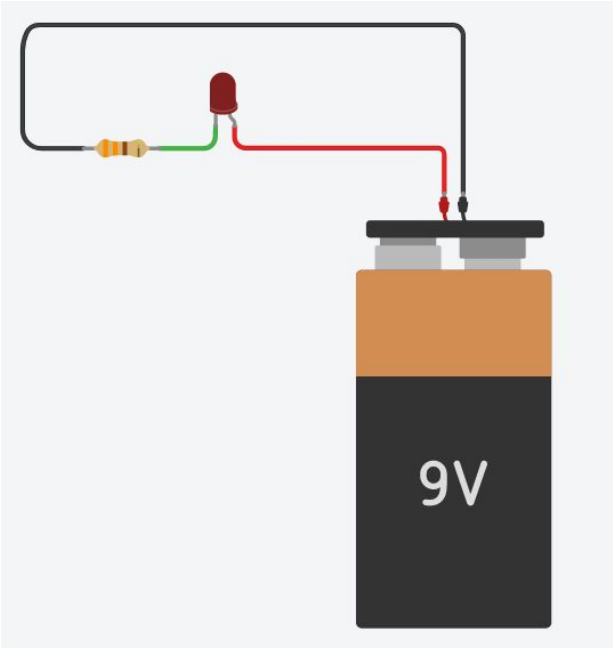
La forma más sencilla es mediante una pila o batería y una resistencia, como se muestra a continuación:

Componentes:

Batería 9V

LED (rojo)

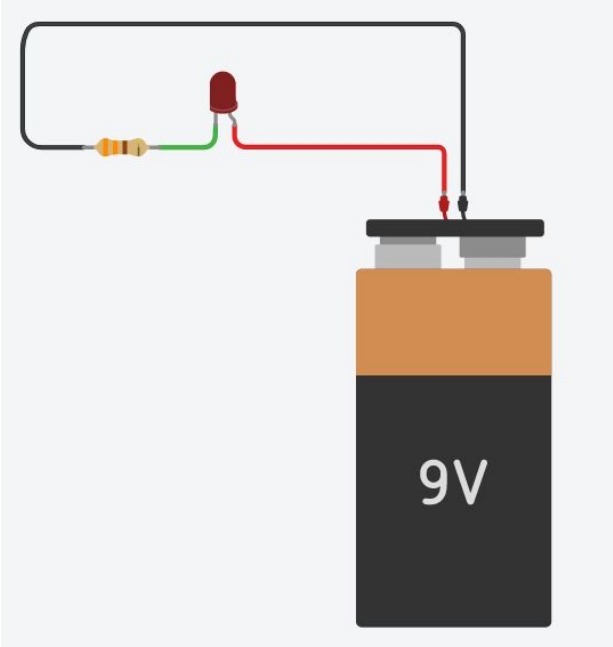
Resistencia: 330 ohms



Algunos conceptos previos...

En este circuito aparece una **resistencia**, el cual es un componente que **limita la corriente que circula por circuito**.

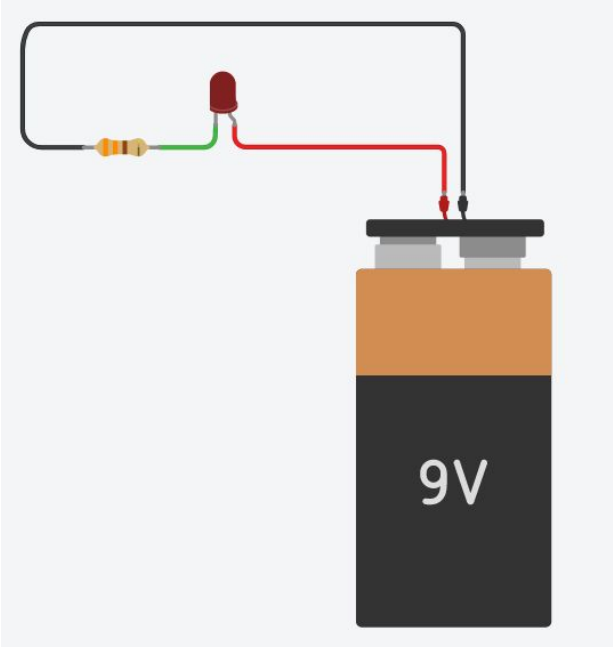
En nuestro caso, la **corriente** que queremos que circule es la del **LED**, o sea, **20 mA** (aprox). Dada esta corriente, **se debe calcular el valor que debe tener la resistencia.**



Algunos conceptos previos...

Para **calcular el valor de la resistencia,** se debe **recorrer el circuito**, planteando las **caídas de tensión en cada componente.** Al hacer esto, se llega a una ecuación como la siguiente:

$$R = \frac{V_p - V_{LED}}{I}$$



Siendo:

R: Resistencia; Vp = Tensión de la pila

V_{LED} = Tensión en el LED; I = Corriente en el LED

Algunos conceptos previos...

La ecuación anterior surge de plantear **Teoría de Circuitos Eléctricos.**

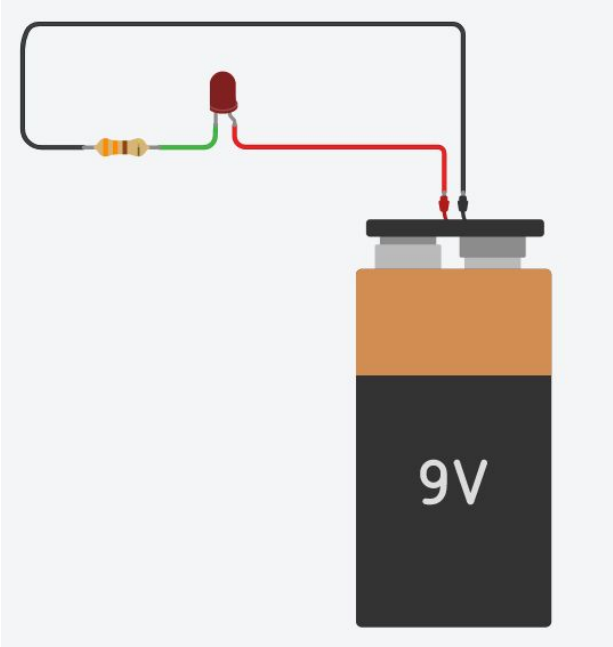
Si reemplazamos los valores en la ecuación anterior,

tenemos:

$$R = \frac{V_p - V_{LED}}{I}$$

$$R = \frac{9V - 2V}{20mA}$$

R = 350 ohms



Algunos conceptos previos...

El valor obtenido a veces puede ser dificil de conseguir comercialmente, lo que se hace es aproximar al valor más cercano,

el cual puede ser de 330 ohms.

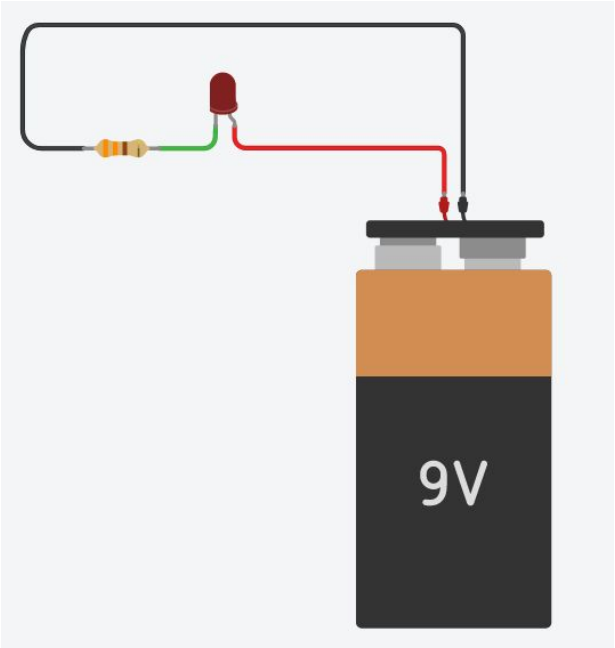
Las resistencias, físicamente, contienen **barras de colores**.

Estas barras **establecen el valor de la resistencia** y su

tolerancia.

Para el caso de los 330 ohms, las barras de colores son:

Naranja - Naranja - Marrón - Dorado

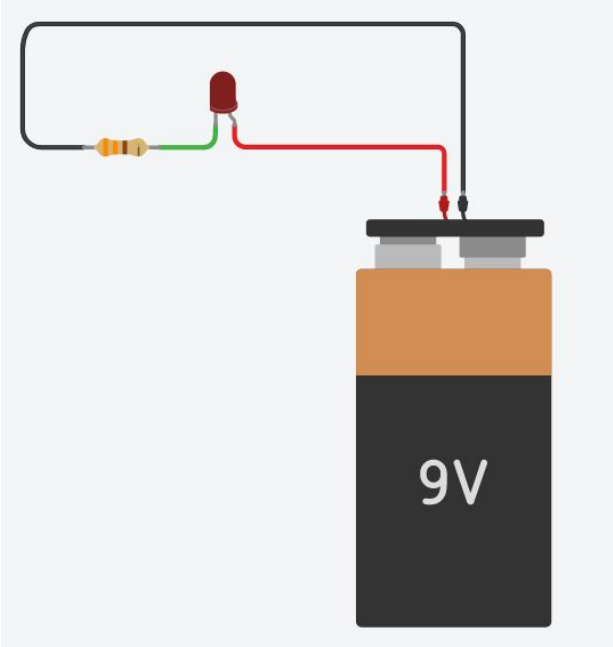


Algunos conceptos previos...

De los colores vistos:

Naranja - Naranja - Marrón - Dorado

Los tres primeros establecen el valor y el **último**, la **tolerancia (5%)**. Esto quiere decir que la resistencia puede no ser exactamente de 330 ohms sino que su valor se encuentra en un rango entre **313, 5 y 346,5 ohms**

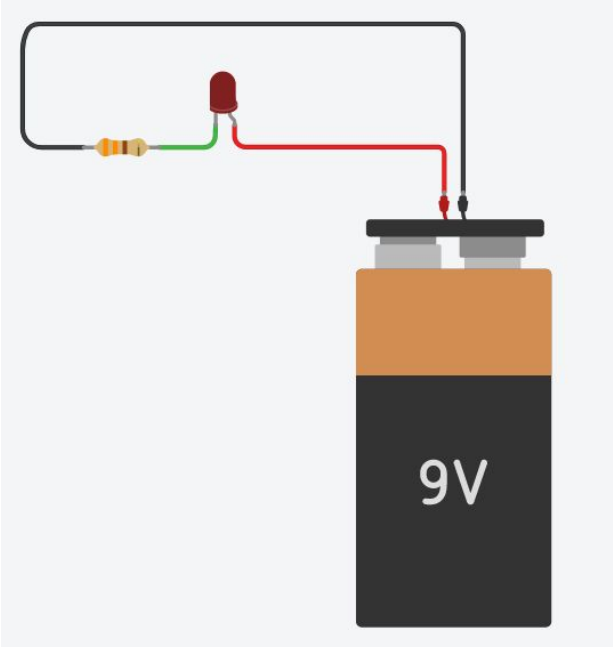


Algunos conceptos previos...

De los colores vistos:

Naranja - Naranja - Marrón - Dorado

Los tres primeros establecen el valor y el **último**, la **tolerancia (5%)**. Esto quiere decir que la resistencia puede no ser exactamente de 330 ohms sino que su valor se encuentra en un rango entre **313, 5 y 346,5 ohms.**



Algunos conceptos previos...

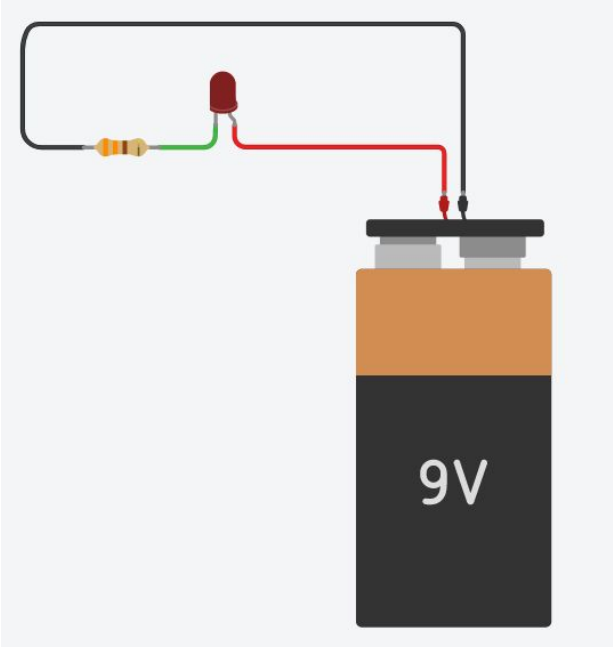
Si la resistencia **no es de 330 ohms exactamente,** al conectarla en el circuito, **la corriente cambiará**. Habría que ver que esa corriente no dañe

al LED.

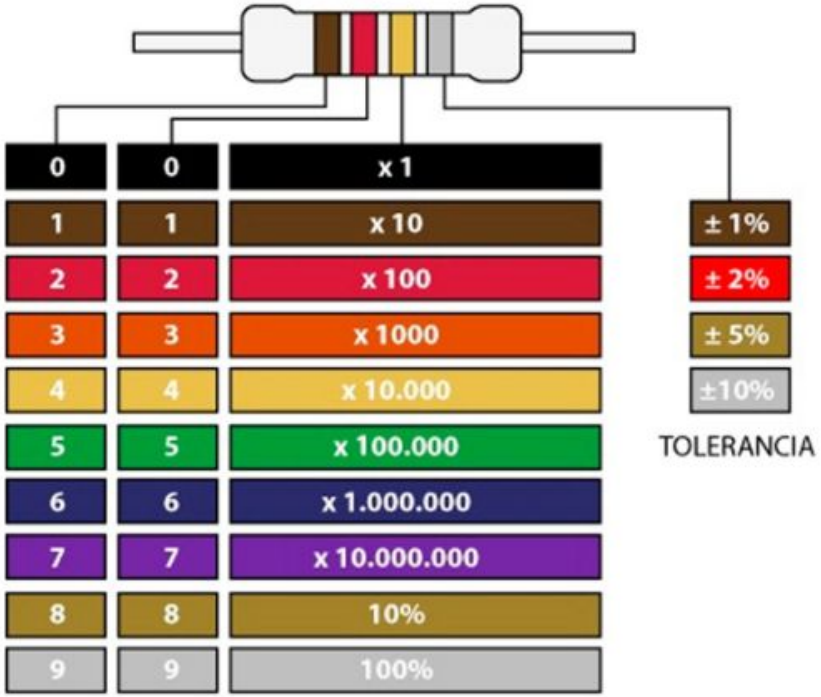
Para saber el **valor de las resistencias** se puede acudir

a una **tabla con el código de colores**, como se comenta

a continuación.



Algunos conceptos previos...



https://www.pinterest.es/pin/659214464175868883/

Algunos conceptos previos...

Calculadora de resistencias según los colores:

https://www.digikey.com/es/resources/conversion-calculators/conversion-calculator-resistor-color-code

Algunos conceptos previos...

Para el caso donde vamos a trabajar, **la pila o batería será reemplazada por la salida del microcontrolador** (ATMEGA328). Las salidas de estos es de **5V** y pueden entregar una **corriente máxima entre 20 y 25 mA.**

Por esta razón, cuando se conecta un LED u otro dispositivo, es **importante chequear la capacidad de corriente que puede entregar el microcontrolador y la tensión.**

Algunos conceptos previos...

Si usamos 5V como salida de un microcontrolador y queremos encender un LED cuya corriente sea de 20 mA, tendremos entonces:

$$R = \frac{V_p - V_{LED}}{I}$$

$$R = \frac{5V - 2V}{20mA}$$

R = 150 ohms.

En la práctica conviene elegir una r**esistencia un poco más alta** (220 ohms o 330 ohms, por ejemplo), **para no exigir la corriente del microcontrolador**. De igual forma, el LED se podrá ver encendido.

Algunos conceptos previos...

Para hacer mediciones de **corrientes** y **tensiones**, se usan algunos instrumentos

1) Voltímetros (miden tensión)
2) Osciloscopios (miden tensión)
3) Amperímetros (miden corriente)

Algunos conceptos previos...

Los Testers generalmente permiten

medir tensiones y corrientes.

[59]

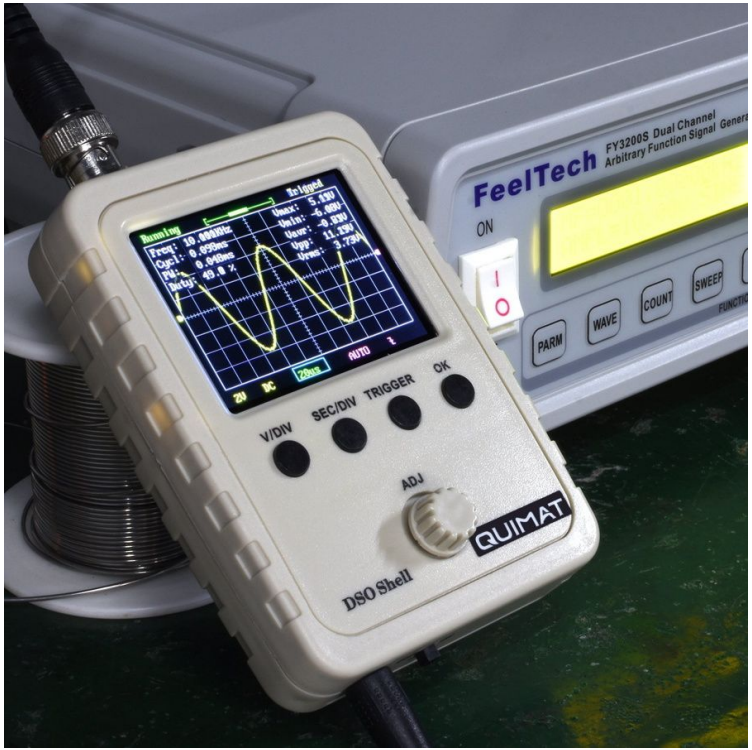
Algunos conceptos previos...

Algunos osciloscopios

[60]

Algunos conceptos previos...

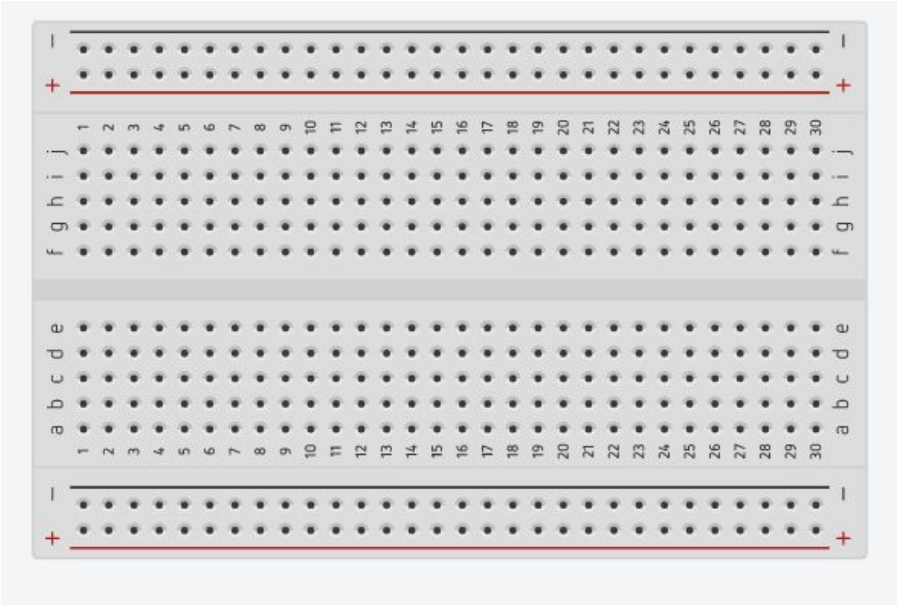
Algunos osciloscopios



[61]

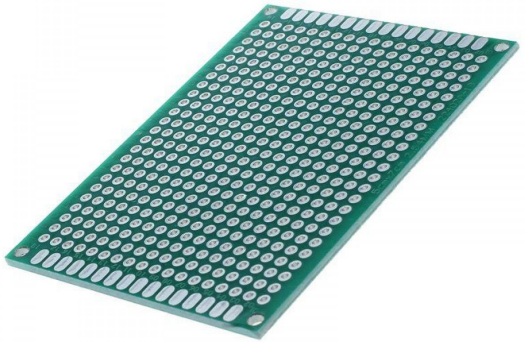
Algunos conceptos previos...

Para nuestros proyectos usaremos **Protoboard**



Algunos conceptos previos...

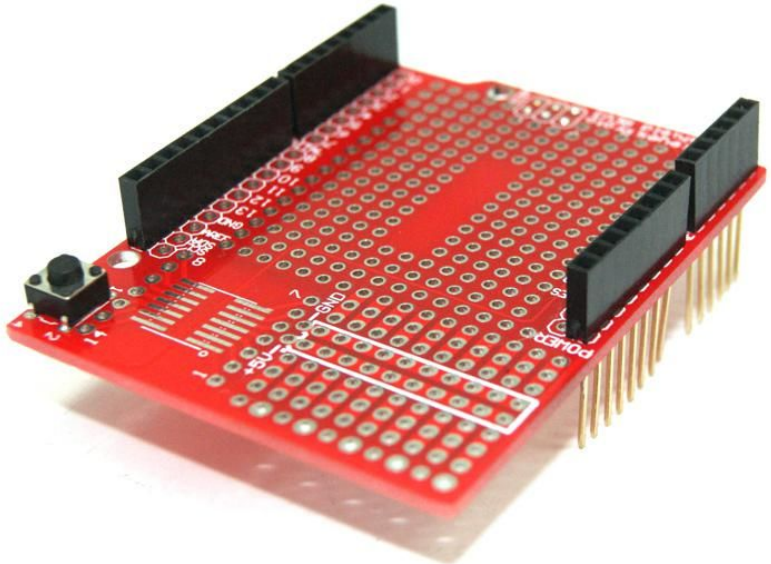
Sin embargo, se pueden usar ***placas perforadas y soldador***, para evitar conexiones con cables.



https://laelectronica.com.gt/placa-perforada-de-doble-cara-con-mascara-para-soldar-9x15-cm

Algunos conceptos previos...

Sin embargo, se pueden usar ***placas perforadas y soldador***, para evitar conexiones con cables.



https://hetpro-store.com/proto-shield-keyes-rev3-para-arduino/

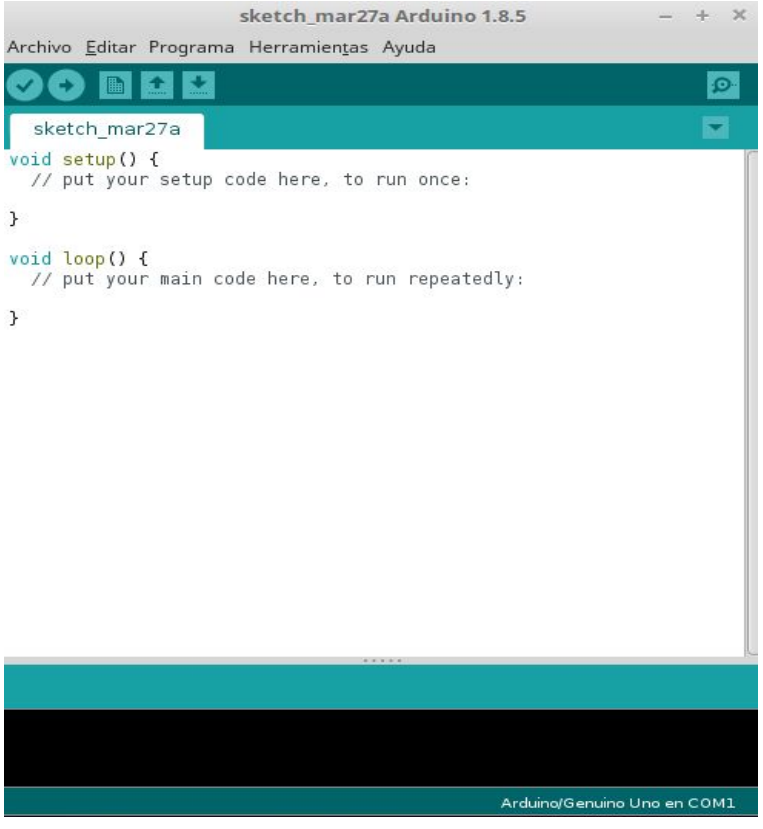
Algunos conceptos previos...

Sin embargo, se pueden usar ***placas perforadas y soldador***, para evitar conexiones con cables.

Primer Ejemplo en Arduino: Blinky

Para comenzar a programar,
es necesario utilizar un
IDE (Entorno de Desarrollo Integrado).
Aquí podremos escribir el código que
se ejecutará sobre la placa Arduino.

El código está basado en C/C++



Planificación

- Configuración y puesta en marcha. Primer programa en Arduino, funciones setup() y loop().
- **Instrucciones pinMode() y digitalWrite(). Manejo del led on-board. Función delay().**
- Variables: tipos de variables, rango, signo, definición, asignación. Uso de variables en el sketch. Ámbito de las variables: variables locales y globales. Desbordamiento de variables. Operaciones matemáticas.
- Limitaciones de la función delay(). Funciones millis() y micros().
- Técnicas para optimización de memoria y tiempos de ejecución.

Ejemplo: Blinky LED de Arduino

Hardware

El primer Ejemplo consistirá en utilizar

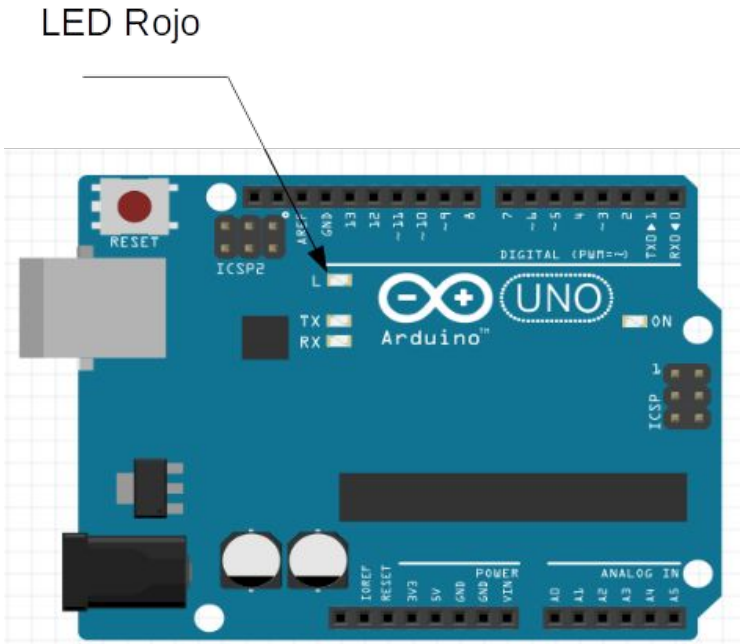
el LED que está incluido en la placa Arduino UNO.

El objetivo será que el LED pueda parpadear,

siendo que durante 1 segundo se prende

y durante el siguiente segundo, se apague.

Ver: Blinky.ino



Ejemplo: Blinky LED de Arduino

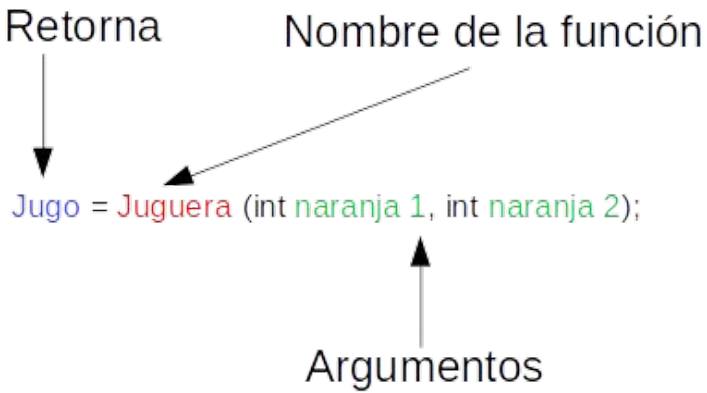
Código

```
void setup(){

 pinMode(13, OUTPUT);                    // Se configura al pin 13 como salida

}
void loop() {

  digitalWrite(13, HIGH);                // Se enciende el LED

   delay(500);                           // demora de 500ms

    digitalWrite(13, LOW);               // Se apaga el LED

    delay(500);                          // demora de 500ms
}
```

Descripción de las funciones

¿Qué son las Funciones?

Son un conjunto de instrucciones que realizan una tarea específica. En general toman ciertos valores de entrada, llamados ***parámetros*** y proporcionan un valor de salida o ***valor de retorno***. **[13]**



Descripción de las funciones

pinMode():

Descripción: Configura el pin especificado para que se comporte como una entrada o una salida.

Sintaxis: pinMode(pin, mode)

Parámetros:

pin: Número de pin con el que se desea trabajar

mode: INPUT o OUTPUT

Devuelve: Nada

Descripción de las funciones

digitalWrite():

Descripción: Escribe un valor ALTO o BAJO en la salida especificada

Sintaxis: digitalWrite(pin, value)

Parámetros:

pin: Número de pin con el que se desea trabajar

value: HIGH o LOW

Devuelve: Nada

Descripción de las funciones

delay():

Descripción: Pausa el programa por la cantidad de tiempo (en milisegundos) especificado como parámetro

Sintaxis: `delay(ms)`

Parámetros:

ms: Número milisegundos que se quiere pausar

Devuelve: Nada

Planificación

- Configuración y puesta en marcha. Primer programa en Arduino, funciones setup() y loop().
- Instrucciones pinMode() y digitalWrite(). Manejo del led on-board. Función delay().
- **Variables: tipos de variables, rango, signo, definición, asignación. Uso de variables en el sketch. Ámbito de las variables: variables locales y globales. Desbordamiento de variables. Operaciones matemáticas.**
- Limitaciones de la función delay(). Funciones millis() y micros().
- Técnicas para optimización de memoria y tiempos de ejecución.

Variables

Variable: es una forma de nombrar y almacenar un valor para su uso posterior por el programa. Se puede guardar el número de pin a usar, datos de un sensor o el valor de un cálculo **[10]**.

Antes de que se utilicen, todas las variables ***deben declararse***. Declarar una variable significa ***definir su tipo*** y, opcionalmente, establecer un valor inicial (inicializar la variable). Las variables no tienen que inicializarse (asignar un valor) cuando se declaran, ***pero a menudo es útil hacer esto.*** **[10]**.

Ejemplo:

```
int Variable1;
int Variable2 = 0;     // Ambas son correctas
```

Variables

Algunos tipos de variables:

1) ***char:*** Es un tipo de datos que ***ocupa 1 byte*** de memoria que almacena un valor de caracter. ***Con signo***
2) ***int:*** Se utiliza principalmente para el almacenamiento de números enteros. En el Arduino Uno, un ***int*** almacena un valor de 16 bits (***2 bytes***). ***Con signo***
3) ***unsigned int:*** Almacena números ***sin signo*** y ***ocupa 2 bytes.***
4) ***float:*** Es un tipo de datos para números de coma flotante, un número que tiene un punto decimal. ***Ocupa 4 bytes.***

Descripción de las funciones

digitalRead():

Descripción: Lee el valor de un pin digital especificado, ya sea ALTO o BAJO.

Sintaxis: digitalRead(pin)

Parámetros:

pin: el número del pin digital que quieres leer

Devuelve: HIGH o LOW

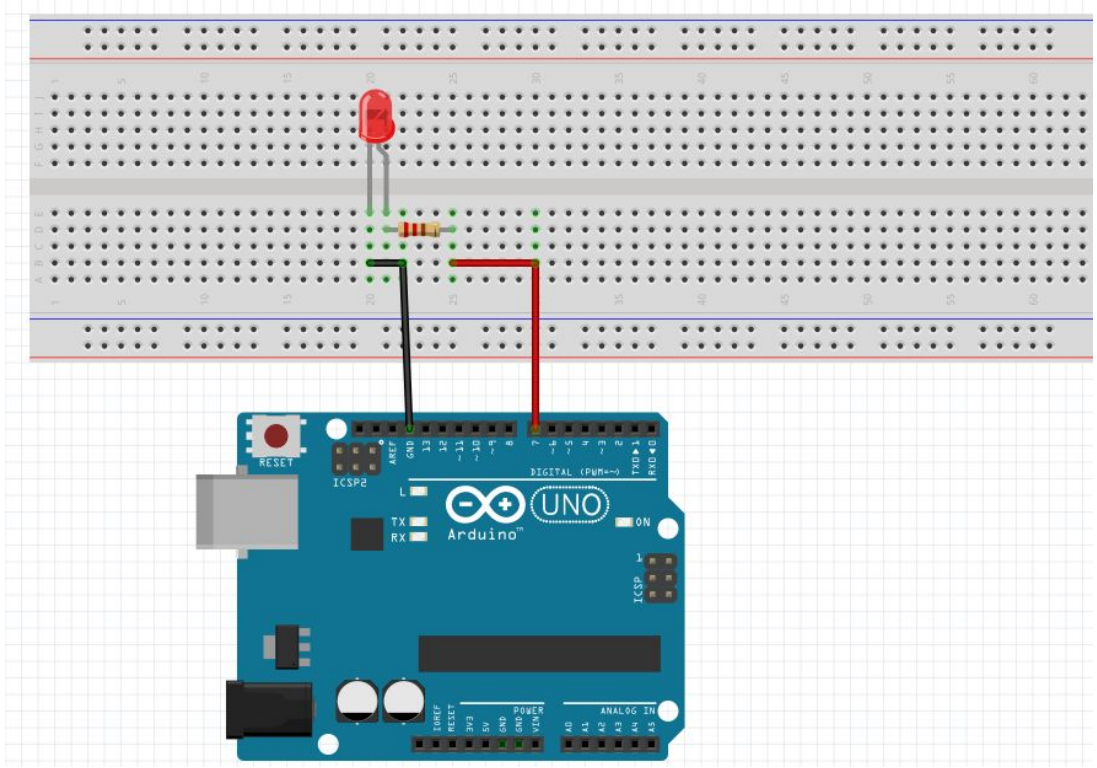
Blinky con LED externo

Desarrollar un sistema para prender un LED externo (no el que incluye la placa ARDUINO UNO), por el pin número 7 (digital).

Blinky con LED externo

Hardware

1) Arduino UNO
2) L.E.D (Cualquier color)
3) Resistor de 220 ohms
4) Cables
5) Protoboard



220 ohms = Rojo - Rojo - Marrón

Primer Ejercicio en Arduino

1) Realizar el mismo programa, pero con dos LEDs conectados a los pines 4 y 7, respectivamente.

2) Para el ejercicio anterior, modificar el tiempo de encendido de cada LED, de manera que no puedan parpadear los dos LEDs a la vez.

Planificación

- Configuración y puesta en marcha. Primer programa en Arduino, funciones setup() y loop().
- Instrucciones pinMode() y digitalWrite(). Manejo del led on-board. Función delay().
- Variables: tipos de variables, rango, signo, definición, asignación. Uso de variables en el sketch. Ámbito de las variables: variables locales y globales. Desbordamiento de variables. Operaciones matemáticas.
- **Limitaciones de la función delay(). Funciones millis() y micros().**
- Técnicas para optimización de memoria y tiempos de ejecución.

Limitaciones Función Delay

Las funciones como delay(), son muy importantes para generar retardos pero tienen el problema de que, **mientras se están ejecutando**, **el microcontrolador no hace otra tarea** Es decir, si hubiera un cambio en uno de los pines de entrada o llegara información a través del puerto serie, ***el microcontrolador no se estaría "enterando" de esta acción***, sólo lo podría hacer luego de terminar dicho delay.

Para esto, se pueden usar otro tipo de funciones como **millis()**

Limitaciones
Función Delay

La función millis() **devuelve la cantidad de milisegundos** desde que el microcontrolador ***empezó a ejecutar***, luego de un ***reinicio o el encendido***. Este número ***volverá a cero, después de 50 días.***

Ver: millis.ino

Sin embargo, ¿por qué utilizar millis() en vez de delay()?

Limitaciones Función Delay

La ventaja principal es que ***no impide ejecutar otro código mientras se está «esperando».***

Por ejemplo, si quisiéramos imprimir «Hola Mundo» en el puerto serie una vez por segundo mientras el micro prende/apaga un LED. Esto no es posible con delay(), ya que entrar a la función delay() ***se detiene todo el código***. Esto se puede hacer mejor con millis

Actividad

Modificar el código estudiado anteriormente

millis.ino

Para que, ***mientras se envía el dato por el puerto serie***, **pueda parpadear un LED mediante delays de 100 ms**

Limitaciones
Función Delay

La Función **delay()** se toma en ms, sin embargo existe otra función similar pero en microsegundos, llamada **delayMicroseconds().**

La función **millis()** también tiene su contraparte en microsegundos llamada **micros()**, con lo cual, también es una forma de obtener un retardo que no pueda bloquear el código. Lo único que hay que tener en cuenta es que el **desbordamiento *no es a los 50 días sino a los 70 minutos.***

Planificación

- Configuración y puesta en marcha. Primer programa en Arduino, funciones setup() y loop().
- Instrucciones pinMode() y digitalWrite(). Manejo del led on-board. Función delay().
- Variables: tipos de variables, rango, signo, definición, asignación. Uso de variables en el sketch. Ámbito de las variables: variables locales y globales. Desbordamiento de variables. Operaciones matemáticas.
- Limitaciones de la función delay(). Funciones millis() y micros().
- **Técnicas para optimización de memoria y tiempos de ejecución.**

Técnicas optimización de memoria

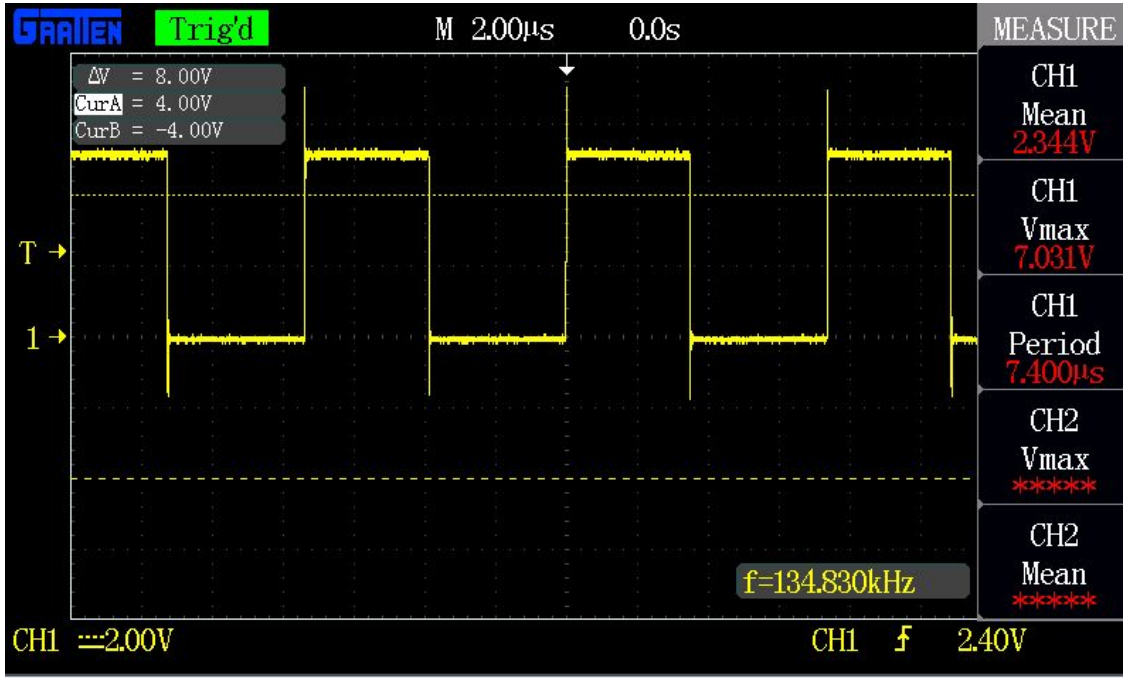
En Arduino, muchas funciones se encuentran escritas en alto nivel, con lo cual, algunas como **digitalWrite** y **digitalRead**, tienen una gran cantidad de código por "atrás", lo cual las vuelve más lentas para su ejecución. Por ejemplo, podemos **acelerar los pines de E/S (I/O)**

Si hacemos un código que apague y prenda un led, **sin delays**, podremos ver entonces que la **frecuencia** es de aproximadamente 134 KHz. **Ver:** Blinky_sin_optimizar.ino

Por ejemplo:

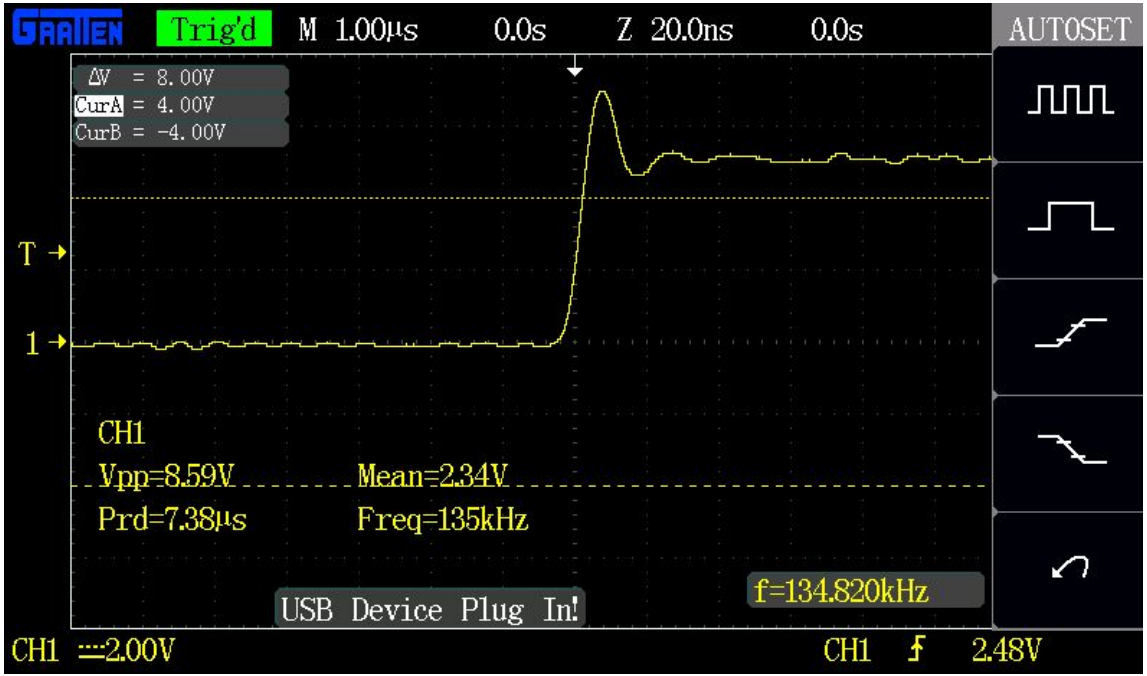
```
void loop(){

digitalWrite(13, LOW);

digitalWrite(13, HIGH);

 }
```

Técnicas optimización de memoria



f = 134,83 KHz

Técnicas optimización de memoria



Haciendo zoom

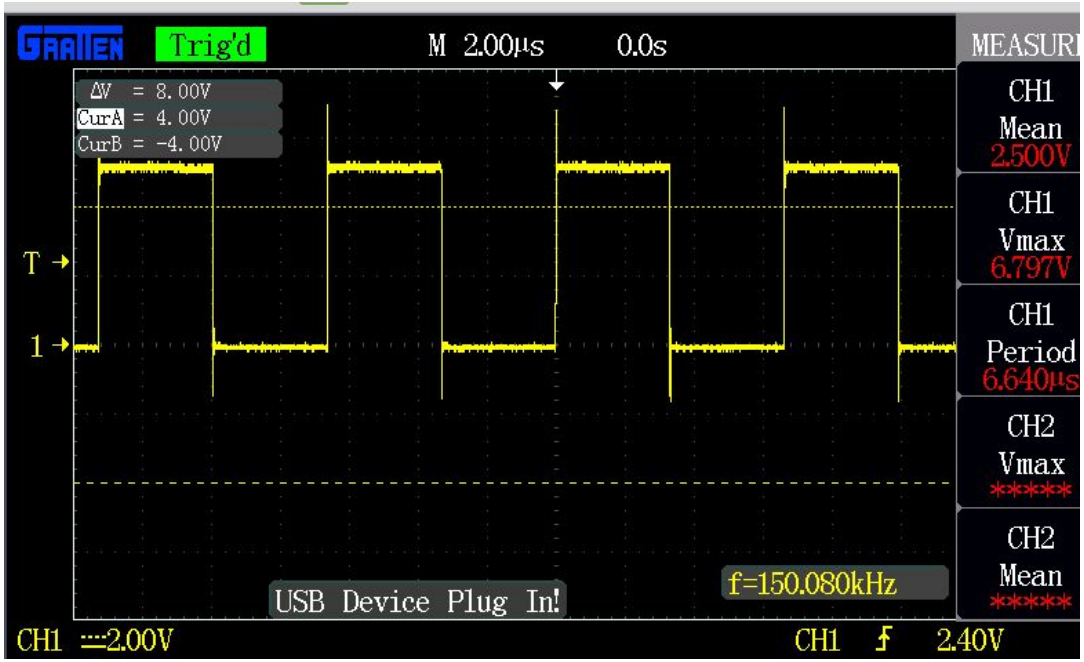
Técnicas optimización de memoria

El código anterior se puede mejorar si usamos un **loop infinito**, dentro de loop principal void loop(){

```
while (true) {

digitalWrite(13, LOW);

digitalWrite(13, HIGH);

}

}
```

Esto **acelera los tiempos pues hay menos llamadas al loop principal.** En este caso se llega a 150KHz. **Ver**: Blinky_optimizado.ino

Técnicas optimización de memoria



f = 150 KHz

Técnicas optimización de memoria

Otra técnica para gestionar la memoria es usar **Constantes**, **PROGMEM** y **Variables globales.**

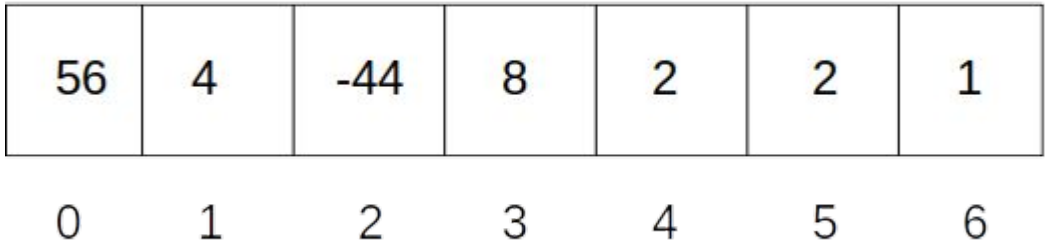
A veces hay que almacenar grandes cantidades de datos en el código para lo cual conviene usar la **variable global const** e **incluir la declaración PROGMEM.**

La declaración **PROGMEM** le indica al compilador que agregue esta información en la **memoria flash,** en lugar de en la **SRAM**. **Recordar** que **PROGMEM** es parte de la **biblioteca pgmspace.h**

Técnicas optimización de memoria

Esta técnica suele ser importante cuando trabajamos con **vectores**.

Un **vector o arreglo** es una **variable** que contiene varios valores pero **cada uno con una posición diferente.** Estos vectores tienen **nombre** y **tipo de dato.**



Técnicas optimización de memoria

A los vectores los podemos declarar como:

```
tipo_dato  nombre[largo];
```

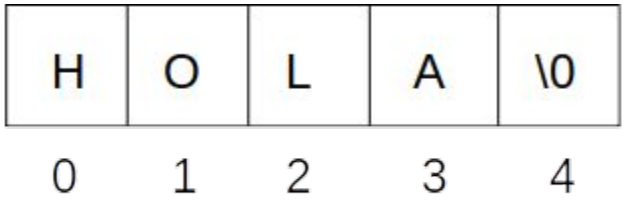
```
int vector_1[] = {5,1,0};
```

También podemos declarar un vector de **tamaños fijos**:

```
int vector_1[50] = {};
```

Técnicas optimización de memoria

Muchas veces resulta necesario utilizar **Strings.**



En **C/C++** se puede implementar como:

```
char saludos[] = "HOLA";
```

http://www.w3big.com/es/cprogramming/c-strings.html

Técnicas optimización de memoria

La sintaxis puede ser

const dataType variableName[] **PROGMEM** = {dato 0, dato 1, dato 2...}

Siendo:

dataType: tipo de variable

variableName: nombre del vector o matriz

Técnicas optimización de memoria

Ejemplo:

const int numeros[1000] **PROGMEM** = {9,7,4,3,0};

En los siguientes ejemplos se mostrará el funcionamiento de **vectores** y cómo se pone en juego este tipo de técnica.

Técnicas optimización de memoria

Otra técnica importante es analizar el tipo de variable. Por ejemplo, si usamos constantes para determinar los pines, ¿por qué usar **int**?, **¿se puede usar alguna otra que ocupe menos memoria y que no use valores negativos?**

Por ejemplo, ¿qué puede pasar con unsigned int, unsigned char?

Planificación

- Introducción a los Sistemas Embebidos
- Introducción a la Programación mediante Arduino
- **Herramientas de Programación para Arduino.**
- Hardware básico para interacción con Arduino.
- Interacción entre Sensores y Arduino.
- Interacción entre Actuadores y Arduino.
- Visualización mediante salidas de Arduino.
- Módulos de Comunicaciones.
- Concepto de Interrupción.
- Integración de Conceptos.

Planificación

- **Directiva de pre-procesamiento #define, ventajas de su uso.**
- Iteración, sentencia for(), autoincremento y autodecremento.
- Control de flujo: sentencia if, else. Toma de decisiones.
- Manejo de leds a través de un pulsador.
- Eliminación por software del efecto rebote (debounce) en pulsadores. Sentencia while()

Directiva de pre-procesamiento

#define es un **componente** útil de **C/C ++** que permite ***dar un nombre a un valor constante***. Este componente no ocupa espacio de memoria del programa en el microcontrolador. Por ejemplo:

```
#define LED 4
```

¿Cómo funciona?

El **compilador reemplazará** a esta constante con su valor, en las partes del código donde aparezca la palabra **LED**. Por ejemplo:

```
digitalWrite(LED,OUTPUT)
```

https://www.arduino.cc/reference/en/language/structure/further-syntax/define/

Directiva de pre-procesamiento

Para ver el efecto de **#define**, veamos el ejemplo:

Blinky_define.ino

Planificación

- Directiva de pre-procesamiento #define, ventajas de su uso.
- **Iteración, sentencia for(), autoincremento y autodecremento.**
- Control de flujo: sentencia if, else. Toma de decisiones.
- Manejo de leds a través de un pulsador.
- Eliminación por software del efecto rebote (debounce) en pulsadores. Sentencia while()

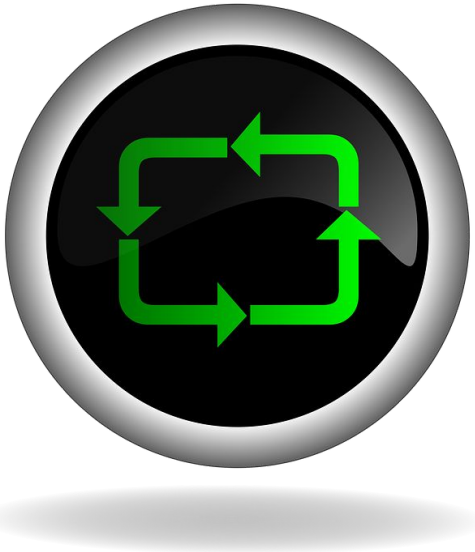
Iteración, ciclos for

¿Qué sucedería si tuviéramos que programar 4 pines de Arduino como salidas?

¿Podríamos hacer cada una de forma secuencial?

Iteración, ciclos for

No sería conveniente hacer una por una pues resulta ineficiente. Lo que se puede hacer es una ***iteración*** (loop) para ir programando ***cada pin del microcontrolador***. Esto se puede hacer mediante bucles. Uno de ellos es el ciclo **for**

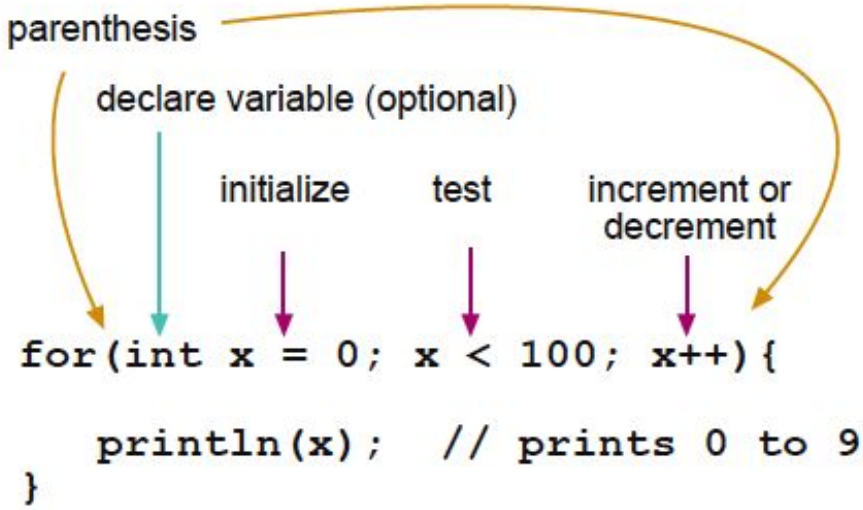


Iteración, ciclos for

Los ciclos **for** son ideales para **repetir ciertas partes de un código** o bloques específicos. Estas instrucciones son útiles para **operaciones repetitivas**. Para operar con estas instrucciones se hace de la siguiente forma:

```
for (inicialización; condición; incremento) {

    //código que queremos que se repita

}
```

Iteración, ciclos for



https://www.arduino.cc/en/pmwiki.php?n=reference/for

Iteración, ciclos for

Del ejemplo anterior, lo primero que se hace es inicializar a la variable.

Cada vez que se pasa por el bucle, se comprueba la condición:

a) Si es verdadera, se ejecuta el bloque de instrucciones y el incremento.
b) Si es falsa, el ciclo termina.

https://www.arduino.cc/en/pmwiki.php?n=reference/for

Iteración, ciclos for

En el caso de que se produzca un **incremento** dentro del for, el mismo se da mediante: **x++**,

En el caso de que se produzca un **decremento** dentro del for, el mismo se da mediante: **x--**,

Para cada caso hay que tener en cuenta la **inicialización** y la **condición** pues puede ser que ***el bucle se vuelva "infinito",*** es decir, ***que no se cumpla la condición para salir de él***.

Algo interesante es que este bucle, nos puede servir para **recorrer vectores (o strings).**

Iteración, ciclos for

Para ver el uso de **for**, veremos cómo programar el encendido y apagado de 4 LEDs, de forma secuencial.

Con el siguiente Hardware.

Ver: Leds_for.ino

Iteración, ciclos for

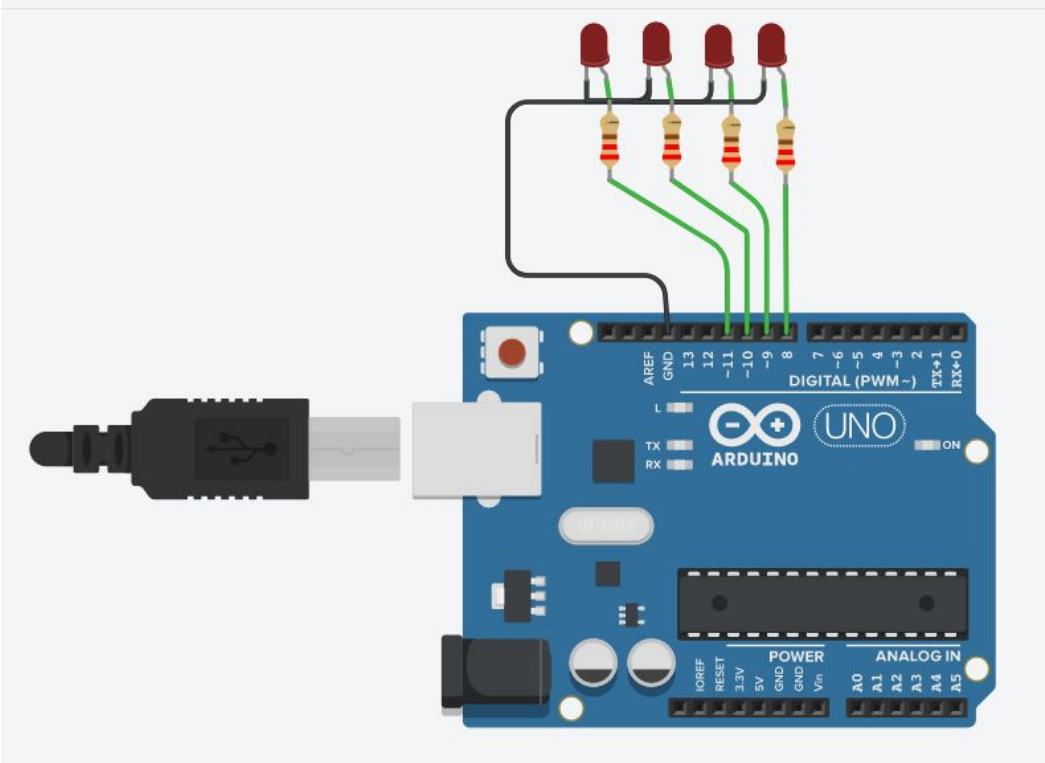
Los terminales largos

de los LEDs se conectan a los pines:

8, 9, 10 y 11.

Resistencias: 220 ohms (4)

LEDs (4)



Iteración, ciclos for

En base a los conceptos estudiados, **analizar el ejemplo: ForLoopIterations incluído** en los **ejemplos del IDE de Arduino.**

Archivos -> Ejemplos -> 05.Control -> ForLoopIterations

Iteración, ciclos for

En base a los conceptos estudiados, **analizar el ejemplo: Arrays incluído** en los **ejemplos del IDE de Arduino.**

Archivos -> Ejemplos -> 05.Control -> Arrays

Planificación

- Directiva de pre-procesamiento #define, ventajas de su uso.
- Iteración, sentencia for(), autoincremento y autodecremento.
- **Control de flujo: sentencia if, else. Toma de decisiones.**
- Manejo de leds a través de un pulsador.
- Eliminación por software del efecto rebote (debounce) en pulsadores. Sentencia while()

Estructuras condicionales

Utilización estructuras condicionales: **"if" y "else"**

La estructura condicional **if** ... **else** es la que nos permite tomar ese tipo de decisiones. Traducida literalmente del inglés, se la podría llamar la estructura "si...si no", es decir, "si se cumple la condición, haz esto, y sino, haz esto otro".

Un ejemplo sencillo sería el siguiente (no se trata de un programa completo, sino tan sólo una porción de código):

```
if (condición) {
        sentencias_si_verdadero;
} else {
        sentencias_si_falso;
}
```

[12]

Planificación

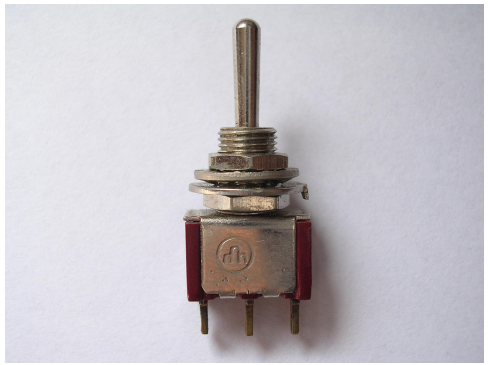
- Directiva de pre-procesamiento #define, ventajas de su uso.
- Iteración, sentencia for(), autoincremento y autodecremento.
- Control de flujo: sentencia if, else. Toma de decisiones.
- **Manejo de leds a través de un pulsador.**
- Eliminación por software del efecto rebote (debounce) en pulsadores. Sentencia while()

Pulsadores

Un **pulsador** es un componente con un botón que permite **pasar o interrumpir** una señal. Generalmente se usan en proyectos para encender/detener un motor, encender/apagar un LED, en teclados, etc.

Lo importante es que ***no se trata de una llave o interruptor,*** el pulsador ***cuando es oprimido***, ***cierra el circuito y cuando se lo suelta, vuelve a su estado anterior.***

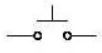
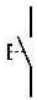
[44]



Pulsadores

Los interruptores pueden ser NC (normalmente cerrado) o NA (Normalmente abierto)

Pulsador NA/NC

	Símbolo americano	Símbolo europeo
NA		
NC		

Entradas y Salidas en Arduino

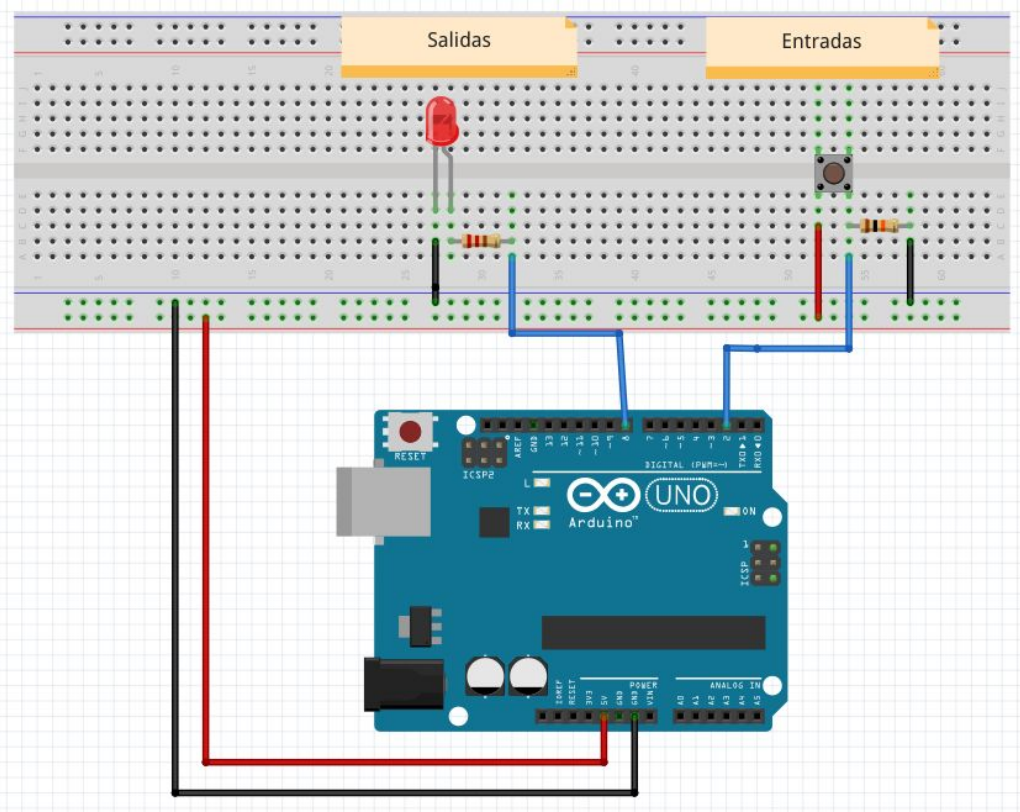
El siguiente ejemplo muestra cómo se relacionan las entradas con las salidas.

Ejercicio: Desarrollar un sistema para que, al accionar un pulsador, se prenda en un LED

Pulsadores o Switch

Hardware

1) Arduino UNO
2) L.E.D (cualquier color)
3) Resistor de 220 ohms
4) Resistor de 10K ohms
5) Switch o pulsador
6) Cables
7) Protoboard



220 ohm = Rojo - Rojo - Marrón

10K ohm = Marrón - Negro - Naranja

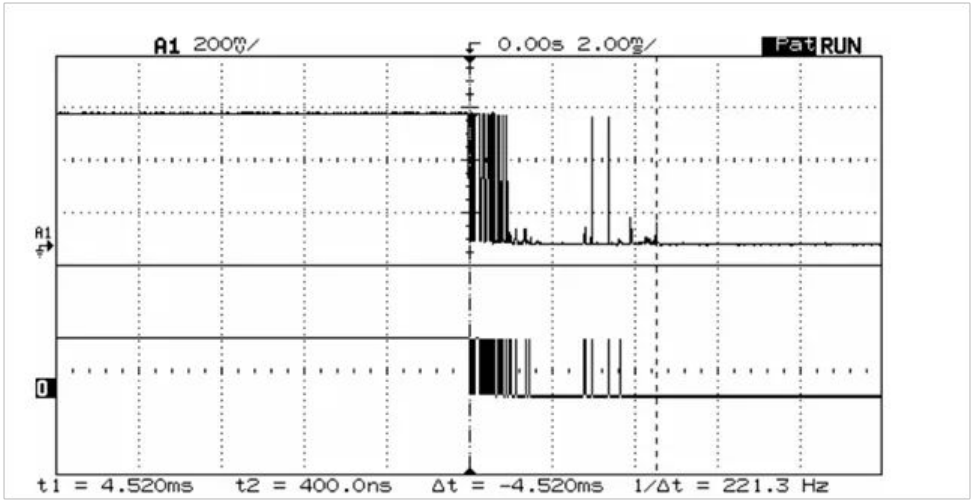
Ver: Switch.ino

Planificación

- Directiva de pre-procesamiento #define, ventajas de su uso.
- Iteración, sentencia for(), autoincremento y autodecremento.
- Control de flujo: sentencia if, else. Toma de decisiones.
- Manejo de leds a través de un pulsador.
- **Eliminación por software del efecto rebote (debounce) en pulsadores. Sentencia while()**

Debounce

De la práctica anterior vimos que al accionar un pulsador, se puede encender un LED pero, en ocasiones, suele costar accionar sobre el pulsador. Esto es porque el interruptor es un componente mecánico, con lo cual al accionarlo o soltarlo, el mismo produce **un efecto de rebote**. Esto se puede ver a continuación:



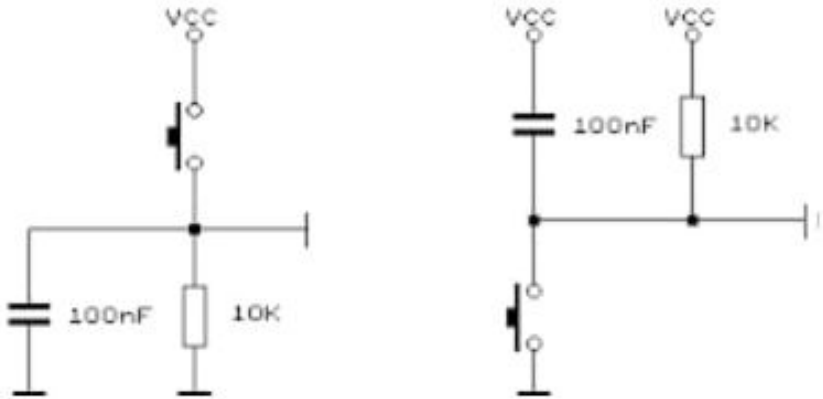
Debounce

¿Cómo lo solucionamos?

Debounce

Para solucionarlo, hay dos métodos:

1) **Por Hardware**



Debounce

2) Por Software: Este método no requiere de agregar ningún componente de Hardware, simplemente se soluciona desde el sketch.

Esta solución significa ***verificar dos veces en un corto período de tiempo*** para asegurarse de que el pulsador esté definitivamente **presionado**. Este ejemplo usa la función **millis ()** para realizar un seguimiento del tiempo transcurrido desde que se presionó el botón.

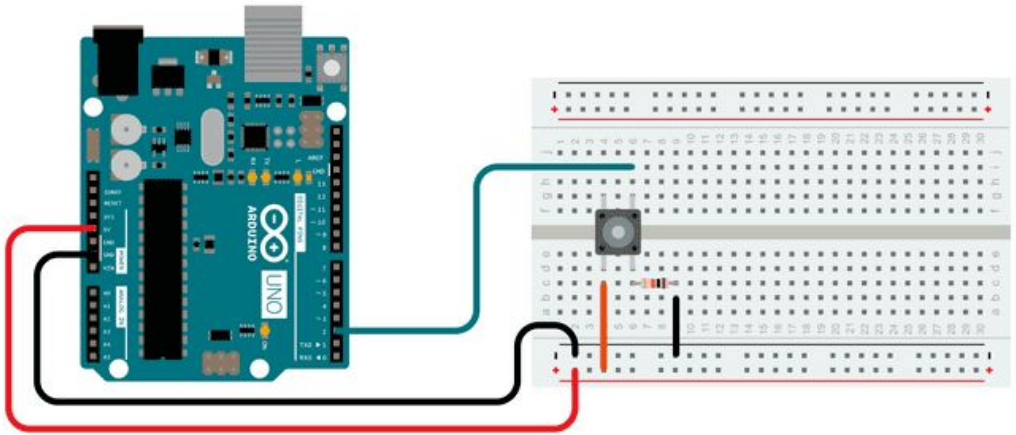
Ver el ejemplo: **antirrebote.ino**

https://www.arduino.cc/en/Tutorial/BuiltInExamples/Debounce

Debounce

1) Pulsador
2) Resistencia 10K ohms:

Marrón - Negro - Naranja



https://www.arduino.cc/en/Tutorial/BuiltInExamples/Debounce