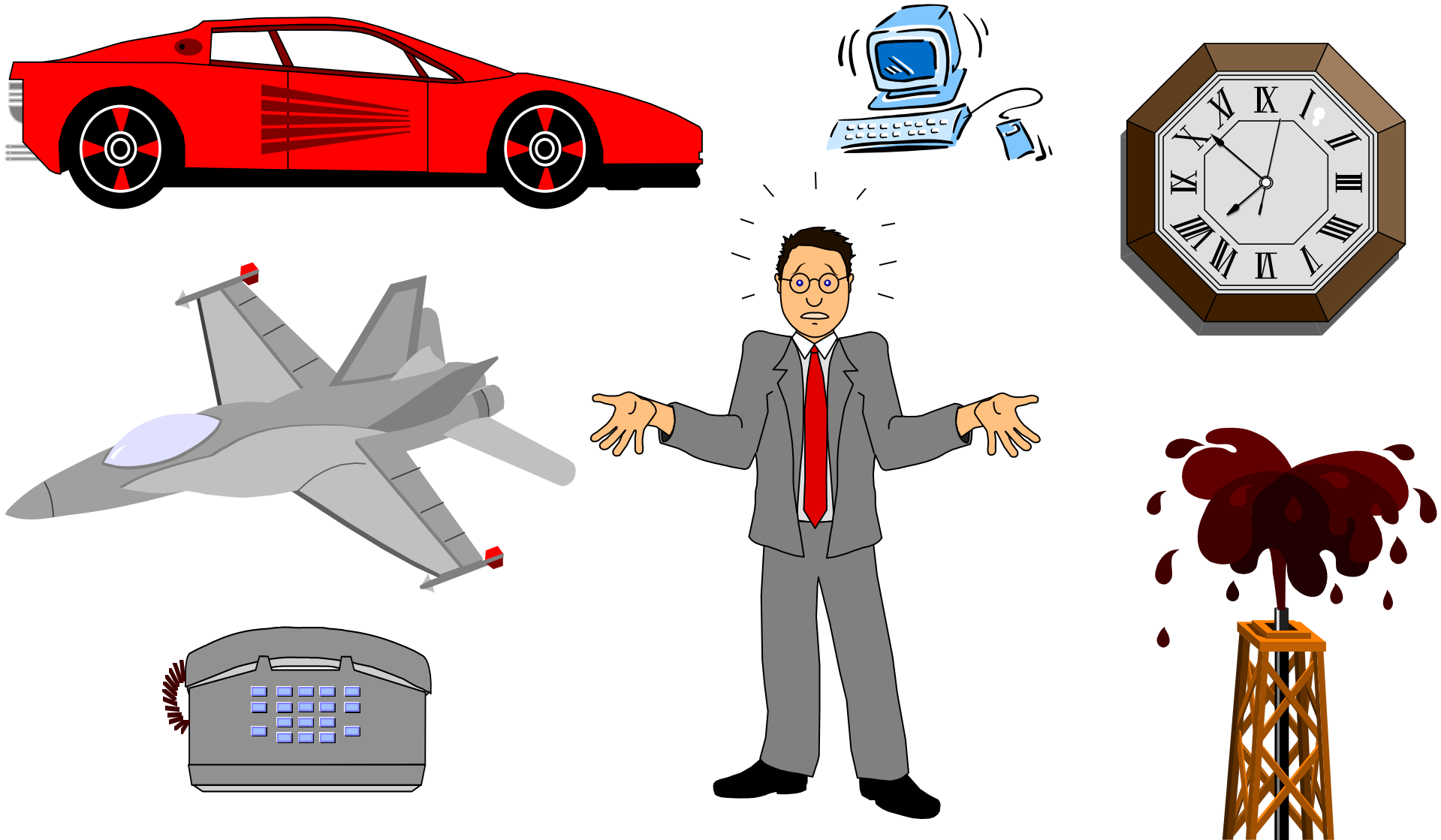


Tema 4. Sistemas digitales

Índice

1. Introducción.
2. Sistemas embebidos.
3. El microprocesador.
4. El microcontrolador.
5. Arduino.
6. El entorno de programación.
7. Programación.
8. Sensores y actuadores.

1. Introducción: ¿Qué es un sistema?



1. Introducción: ¿Qué es un sistema?

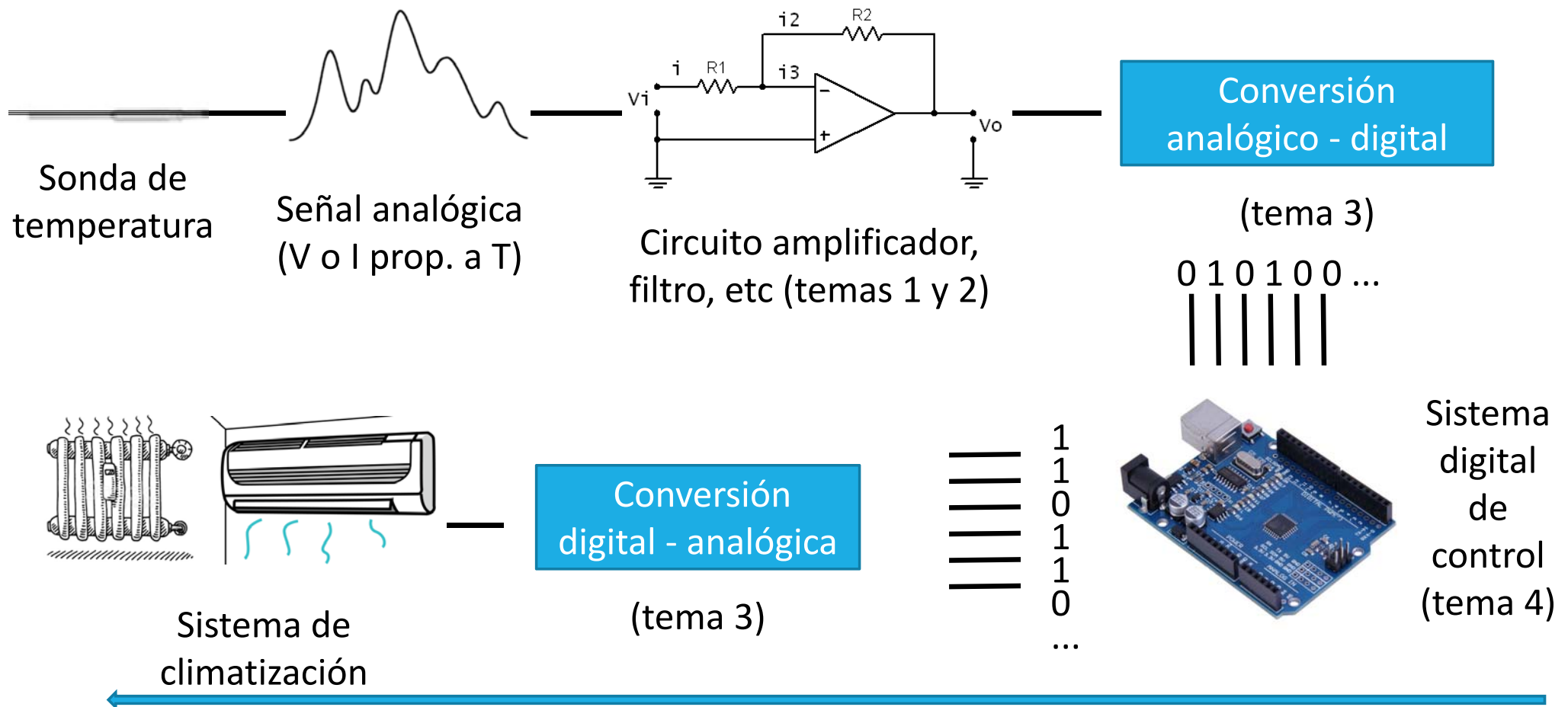
Un sistema puede ser definido como todo aquello que interactúa con su entorno.

Un esquema general podría ser:



1. Introducción: Sistemas Digitales

Recordando la introducción de la asignatura:



1. Introducción: Sistemas Digitales

Los sistemas digitales pueden implementarse mediante:

- Circuitos combinacionales / secuenciales: componentes discretos.
- Circuitos de lógica programable (PLDs, FPGAs...)
- Microprocesadores
- Microcontroladores
- Autómatas programables (PLC's)
- ...

1. Introducción

En la práctica, hoy en día, el diseño de un circuito digital para una determinada aplicación no se resuelve mediante el diseño desde cero de un sistema digital dedicado y específico para dicha aplicación, sino que se implementa mediante un **sistema empotrado o embebido**, un sistema digital programable basado en un microprocesador o un microcontrolador cuya programación permite su uso en muy diferentes aplicaciones.



2. Sistemas embebidos

Un **sistema embebido** (*embedded*) o **empotrado** es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas. Al contrario de lo que ocurre con un ordenadores de propósito general que están diseñados para cubrir un amplio rango de necesidades, los sistemas embebidos se diseñan para cubrir necesidades específicas.

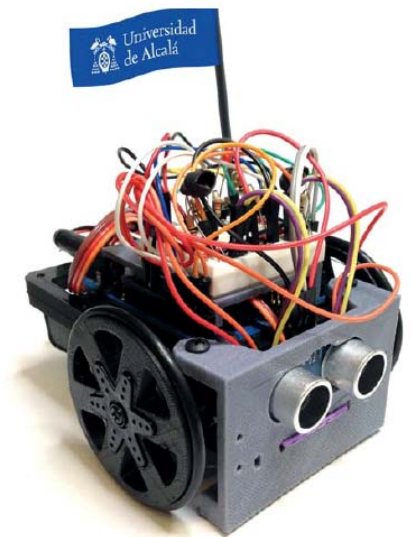


2. Sistemas embebidos

Son generalmente sistemas reactivos de tiempo real:

- “Reaccionan” a eventos externos.
- Mantienen interacción permanente.
- Están continuamente funcionando.
- Están sujetos a restricciones externas de tiempo.

Realizan varias tareas concurrentemente.



2. Sistemas embebidos: Ejemplos

Ejemplos de sistemas embebidos

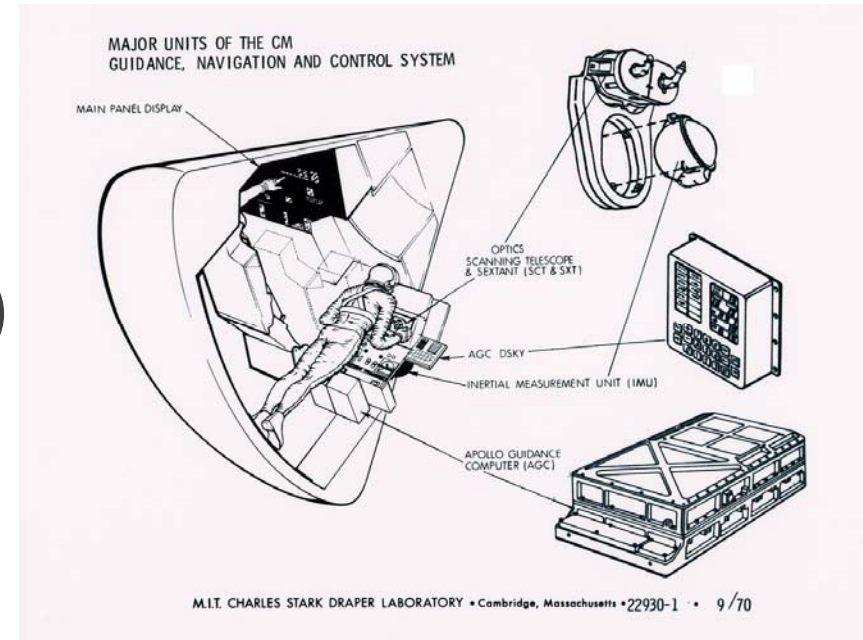
- Electrónica de consumo: TVs, lavadoras, frigoríficos, lavaplatos, ...
- Automóviles: control velocidad, climatización, visualización, ABS, ASR, inyección,...
- Telecomunicaciones: radios trunking, teléfonos móviles, ...
- Aviónica, espacial: computadores de vuelo, de misión, robots, ...
- Defensa: bombas y misiles inteligentes, vehículos, dirección de tiro, ...



2. Sistemas embebidos: Ejemplos

Sistema de guiado Apollo 11 (1969)

- Primera vez que se emplearon circuitos integrados.
- Datos : 16 bits (15 bits datos + bit paridad)
- Memoria tipo magnético:
 - ROM: 36.864 x 2 bytes.
 - RAM: 2.048 x 2 bytes.
- Número de instrucciones: 34.
- Frecuencia de reloj: 85 kHz.
- Número de puertas lógicas: 5.600 .
- Peso: 30 kg.
- Consumo: 70 W.



2. Sistemas embebidos: Ejemplos

Raspberry Pi 3 Model B (2016)

- A 1.2GHz 64-bit quad-core ARMv8 CPU.
- 802.11n Wireless LAN, Bluetooth 4.1, Bluetooth Low Energy (BLE).
- 1GB RAM.
- 40 GPIO pins.
- Slot para tarjeta Micro SD .
- 4 puertos USB, puerto Ethernet .
- Audio jack 3.5mm combinado con vídeo compuesto.
- Salida Full HDMI, Interfaz para Cámara, Interfaz para Display.
- ...



2. Sistemas embebidos: Características

Fiabilidad y seguridad

- Un fallo en un sistema de control puede hacer que el sistema controlado se comporte de forma peligrosa o antieconómica.
- Es importante asegurar que si el sistema de control falla lo haga de forma que el sistema controlado quede en un estado seguro.
 - Hay que prever los posibles fallos o excepciones del diseño

Eficiencia

- Gran parte de los sistemas de control deben responder con gran rapidez a los cambios en el sistema controlado.

2. Sistemas embebidos: Características

Bajo consumo

- Muchos de estos sistemas están alimentados con batería o pilas. Menor consumo => mayor autonomía.
- En muchos casos necesidades de bajo voltaje (3V).

Bajo peso

- Característica deseable en sistemas portátiles
- No depende únicamente del computador embarcado y su periferia sino también de la alimentación (baterías) o de los sensores y actuadores.

2. Sistemas embebidos: Características

Bajo precio

- Aplicable a electrónica de consumo y otros dispositivos con mercados muy competitivos (p.e. telefonía móvil).

Pequeñas dimensiones

- Las dimensiones de un sistema empotrado no dependen sólo de sí mismo sino también del espacio disponible en el sistema que controla y/o monitoriza.

2. Sistemas embebidos: Características

Concurrencia

- Las distintas tareas del sistema controlado o monitorizado funcionan simultáneamente.
- El sistema de control debe atenderlo y generar las acciones de control o visualización de forma simultánea.
- Si hay una sola unidad de proceso: se emplean técnicas de multiproceso, empleando sistemas operativos en tiempo real.
- Si hay varias unidades de proceso se habla de multiprocesadores.

3. El microprocesador

El **microprocesador** Circuito digital universal de propósito general muy versátil.

Funciona ejecutando una serie de “pasos” básicos:

- Cada uno de los pasos: instrucción.
- Cada elemento de información: dato.
- Programa: el conjunto ordenado de todas las instrucciones .

Para variar la funcionalidad del circuito se modifica el programa.



3. El microprocesador

El microprocesador es la unidad central de proceso que permite ejecutar un programa pero requiera para su utilización en la práctica de:

- Reloj,
- Memoria (ROM, RAM),
- Buses de comunicación,
- Periféricos de E/S

Diseñar sistema embebidos basados en microprocesadores implica diseñar para cada problema una placa con el micro, la memoria, etc.

4. El microcontrolador

Un **microcontrolador** es un circuito integrado de alta escala de integración que incorpora la mayor parte de los elementos que configuran un sistema informático: el microprocesador, la memoria, periféricos, E/S, etc.

Un microcontrolador permite la implementación rápida de una computadora con una circuitería externa mínima, lo que convierte estos dispositivos en una buena solución para el diseño de sistemas embebidos.



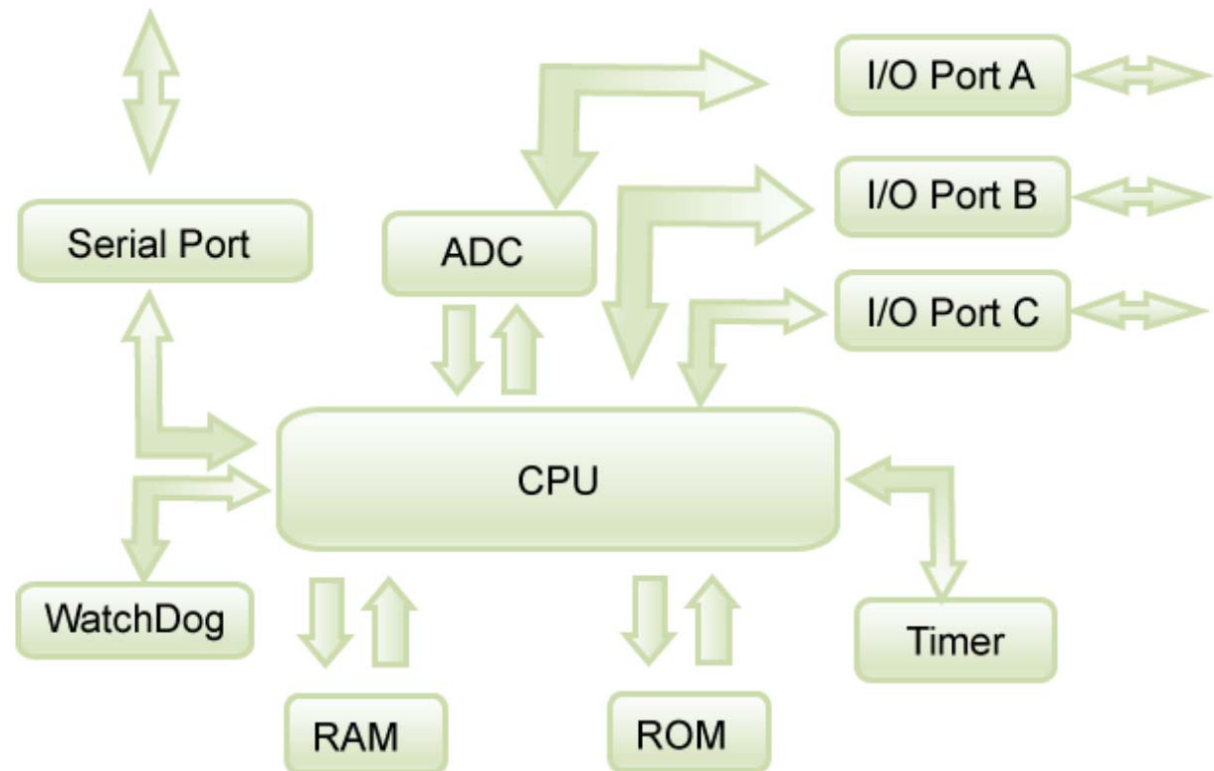
4. El microcontrolador

Existen diferentes tipos de microcontroladores en función de su arquitectura (Von Neumann, Harvard), las características del conjunto de instrucciones con las que se programan (RISC, CISC), la capacidad del procesador (16, 32 o 64 bits), cantidad de memoria, número de periféricos, etc.

Algunos de los microcontroladores más comunes son: SAM7 o SAM3 (Atmel), ColdFire (Motorola), Cortex-M3 o Cortex-M0 (NXP o Texas)

4. El microcontrolador

Arquitectura genérica



4. El microcontrolador

Arquitectura genérica

- Procesador: unidad de control, unidad aritmético-lógica, registros, buses, conjunto de instrucciones.
- Memoria: de programas y de datos. ROM, EEPROM, FLASH, RAM, SRAM, DRAM, etc.
- Periféricos: conversores analógico/digitales, puertos de comunicación serie (SPI, I2C, USB), Ethernet, modulador de ancho de pulsos, temporizadores, contadores, entradas y salidas de propósito general.

5. Arduino

En 2005, la empresa Arduino lanzó al mercado una primera placa que integraba un microcontrolador (AVR de Atmel) y algunos puertos digitales y analógicos de entrada/salida junto con un sencillo entorno de programación.

En la actualidad, hay comercializadas distintas alternativas (Uno, Pro, Mega, etc.) con placas de expansión (shields) que permiten desarrollar multitud de aplicaciones avanzadas.

<https://www.arduino.cc/en/Guide/HomePage>

5. Arduino UNO

- Microcontrolador AVR
- Circuito de reloj (16 MHz)
- Circuito de Reset
- Conversor USB-Serie
- Regulador de alimentación
- Pines de conexión

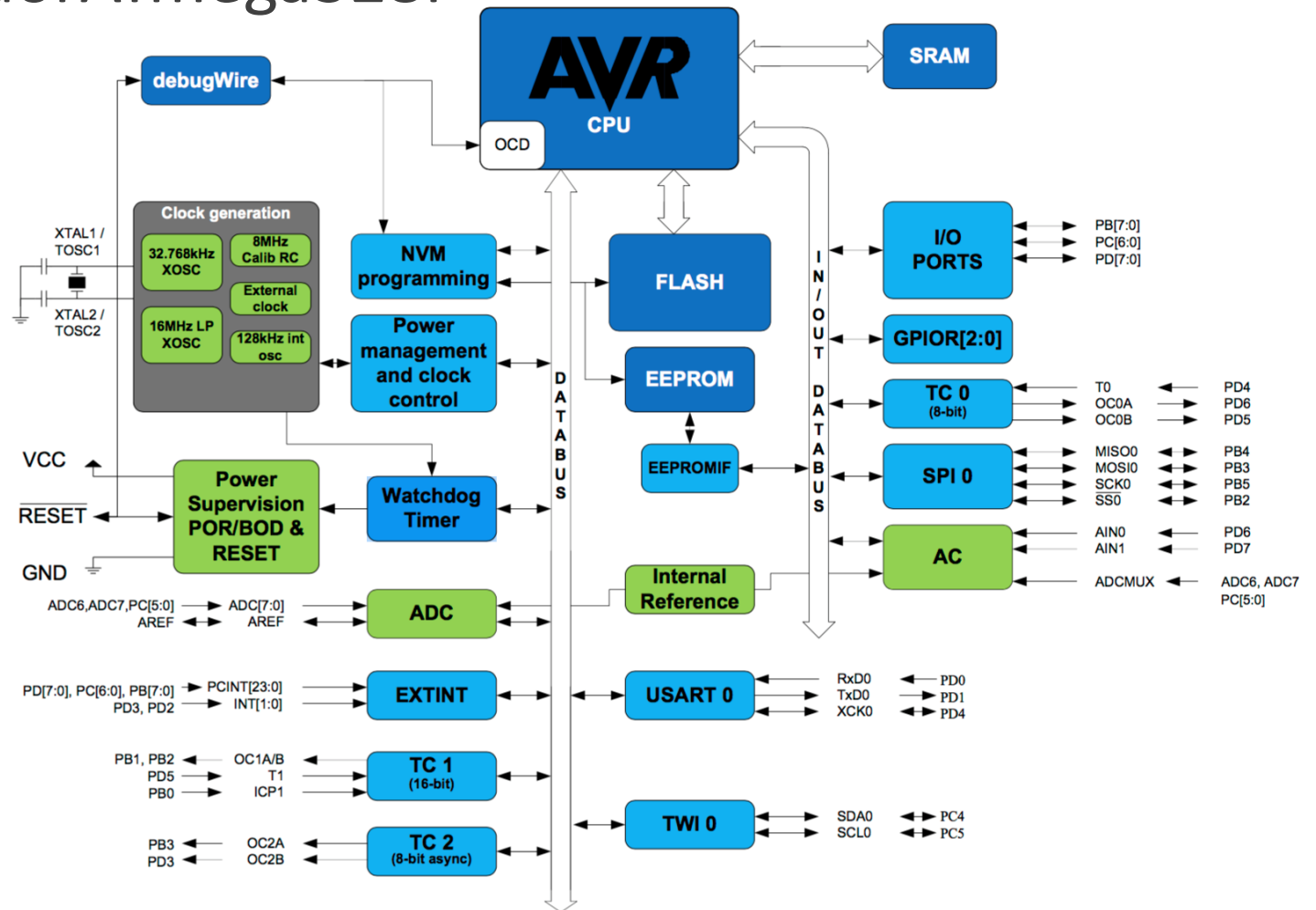


5. Arduino UNO

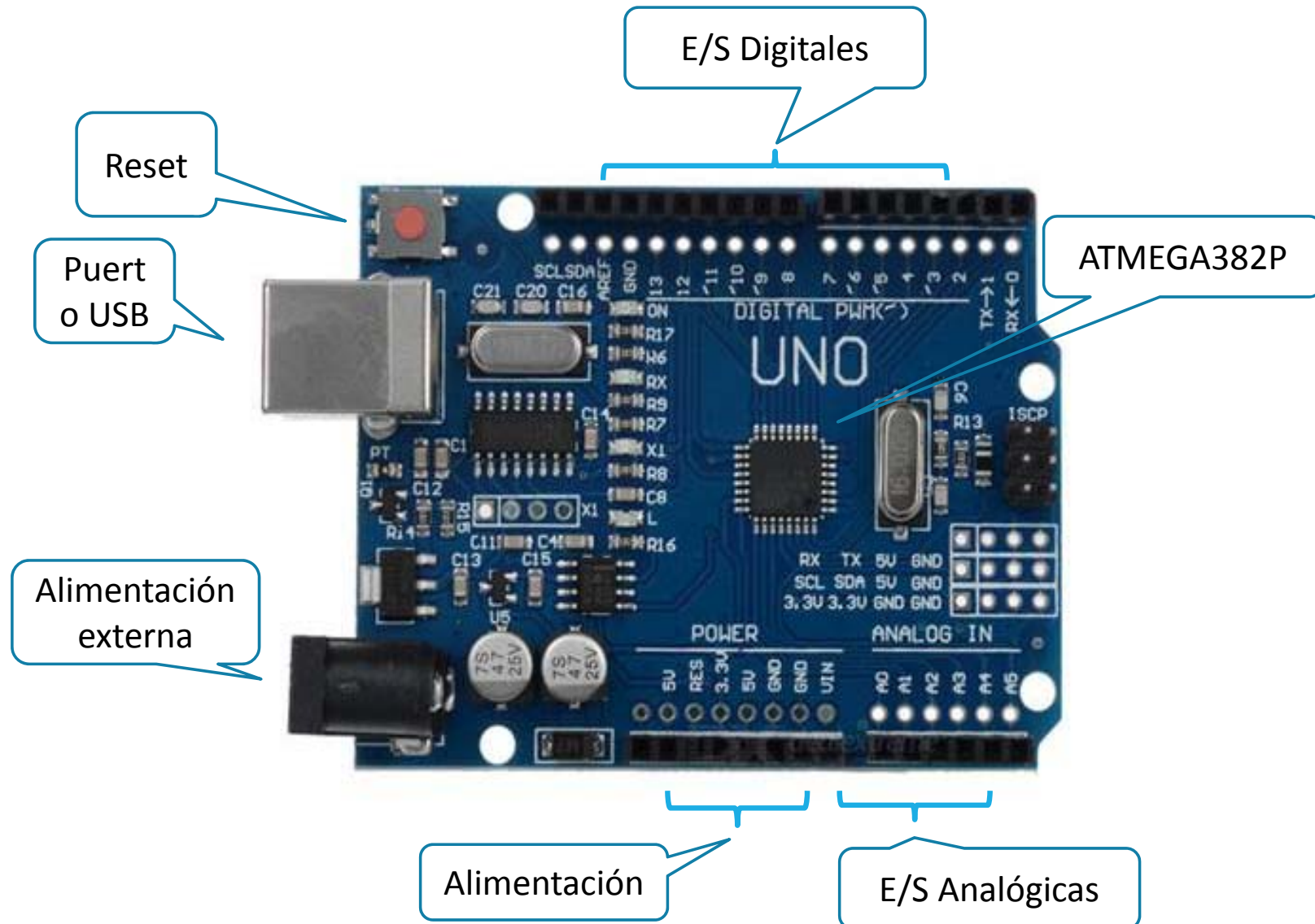
Microcontrolador	ATmega328P
Alimentación	5V
E/S Digitales	14
E Analógicas	6
Memoria Flash	32 KB (ATmega328P)
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
LED integrado	Pin 13
Longitud	68.6 mm
Anchura	53.4 mm
Peso	25 g

5. Arduino UNO

Microcontrolador ATmega328P



5. Arduino UNO



6. El entorno de programación



- Gratuito
- Interfaz gráfica en Java.
- Muy sencillo.
- Integra editor, compilador, descarga a placa y monitor serie.
- Archivos .ino, además soporta .c, .cpp, .h
- Organización del código por pestañas.
- Para descargar el entorno de programación: <https://www.arduino.cc/en/Main/Software>



6. El entorno de programación



Ventana de edición

Ventana de mensajes

MotorPasoPaso Arduino 1.6.8

```
#include <AccelStepper.h>
#define HALFSTEP 8

#define motorPin1 8 // IN1 en la placa ULN2003
#define motorPin2 9 // IN2 en la placa ULN2003
#define motorPin3 10 // IN3 en la placa ULN2003
#define motorPin4 11 // IN4 en la placa ULN2003

AccelStepper motor(HALFSTEP, motorPin1, motorPin3, motorPin2, motorPin4);

void setup() {
  motor.setMaxSpeed(1000.0);
  motor.setAcceleration(500.0);
}

void loop() {
  int val = analogRead(A0);
  val = map(val, 0, 1023, 0, 1000);

  motor.setSpeed(val);
  motor.run();
}
```

Compilado

El Sketch usa 6.006 bytes (18%) del espacio de almacenamiento de programación.
Las variables Globales usan 109 bytes (5%) de la memoria dinámica, de 2048 bytes.

22 Arduino/Genuino Uno en /dev/cu.wchusbserial1420

7. Programación

Un programa debe contener al menos las dos funciones siguientes:

- `void setup()`: función que se ejecuta una única vez al comenzar la ejecución del programa después de su descarga, después de encender la alimentación o tras la pulsación del botón de reset.
- `void loop()`: función que se ejecuta cíclicamente de manera interrumpida hasta que se desconecte la alimentación de la placa.

7. Programación

Un ejemplo que hace parpadear un LED de la propia placa:

```
void setup() {  
    // Antes de hacer uso de un terminal de E/S digital, deben especificarse si se utilizará como  
    // entrada (INPUT) o salida (OUTPUT). La instrucción siguiente indica que el pin 13, al que está  
    // conectado el LED integrado en la placa, se utilizará como salida.  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH);    // Escribe un nivel alto (HIGH) en la salida digital 13 -> Enciende el LED.  
    delay(1000);              // Espera un segundo  
    digitalWrite(13, LOW);    // Escribe un nivel bajo (LOW) en la salida digital 13 -> Apaga el LED  
    delay(1000);              // Espera un segundo  
}
```

7. Programación: Tipos de datos

- void: utilizado en la declaración de funciones que no devuelven ningún valor.
- boolean: true or false.
- char: un carácter ('A', 'f', ...).
- byte: almacena un valor numérico de 8 bits entre 0 y 255.
- int: número entero almacenado en 16 bits entre -32768 y 32767.
- unsigned int: número entero positivo entre 0 y 65535.
- long: numero entero almacenado en 32 bits (de -2,147,483,648 a 2,147,483,647).
- float: numero real.
- array: conjunto de variables que se acceden mediante un índice.
- string: array de caracteres.

7. Programación: Tipos de datos

- `void loop()`
- `boolean activado= false;`
- `char letra = 'A';`
- `byte num = B0101;` (número 5 en binario)
- `int i = 8;`
- `long l = 1534761;`
- `float f = 50.0;`
- `array:`
 - `int numeros[6];` //Crea un array vacío para almacenar 6 números enteros.
 - `int numeros2[] = {2, 4, 8, 3, 6};` //Array de enteros relleno.
 - `int n = numeros2[2];` // Se declara una variable entera 'n' que coge el valor de la tercera posición del array (la primera es la posición 0).

7. Programación: Estructuras de control

```
if (v > 50)
{
    // se hace algo
}
```

```
if (v > 50)
{
    // se hace algo
}
else
{
    // se hace otra cosa
}
```

```
int v = 0;
while (v < 200){
    // se hace algo 200 veces
    v++; // se incrementa la variable v
}
```

```
for(int v = 0; v < 6; v++){
    println(v);
    // imprime los números del 0 al 5
}
```

7. Programación: Estructuras de control

```
do  
{  
    // haz algo mientras v sea menor que 5  
} while (v < 5);
```

```
switch (v) {  
    case 1:  
        // haz algo si v es 1  
        break;  
    case 2:  
        // haz algo si v es 2  
        break;  
    default:  
        // haz algo en el resto de casos  
        break;  
}
```

7. Programación: Funciones disponibles

Para entradas y salidas digitales:

- `pinMode(pin, mode)`

- Configura un pin como entrada (INPUT) o salida (OUTPUT)

Ej.: `int pin = 12;`
 `pinMode (pin, INPUT); // configura el pin 12 como entrada.`

- `digitalWrite(pin, value)`

- Escribe un valor alto (HIGH) o bajo (LOW) en un pin **configurado** como **salida**

Ej.: `digitalWrite (10, HIGH); // escribe un 1 en el pin 10.`

- `digitalRead(pin, value)`

- Lee el valor de un pin **previamente** configurado como **entrada**

Ej.: `int val;`
 `val = digitalRead (11); // guarda el valor del pin 11 en val.`

7. Programación: Funciones disponibles

Para comunicación serie:

- `Serial.begin(velocidad)`

- Configura la velocidad de la comunicación serie.

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

- `Serial.print(dato)`

- Escribe un dato por el puerto serie

Ej.: `Serial.print("Hola");`

- `Serial.println(dato)`

- Funciona igual que `Serial.print` añadiendo un salto de línea al final

7. Programación: Funciones disponibles

Entradas analógicas:

- `analog.Read(pin)`
 - Devuelve el valor de una entrada analógica

Ej.: `int pinAnalog = 3;`
 `int valor;`
 `valor = analog.Read(pinAnalog);`

7. Programación: Funciones disponibles

Temporización:

- `delay(tiempo)`

- Detiene la ejecución del programa durante los milisegundos indicados en la variable tiempo.

Ej.: `delay(500);` // Detiene el programa durante 500 milisegundos.

- `delayMicroseconds(tiempo)`

- Detiene la ejecución del programa durante los microsegundos indicados en la variable tiempo.

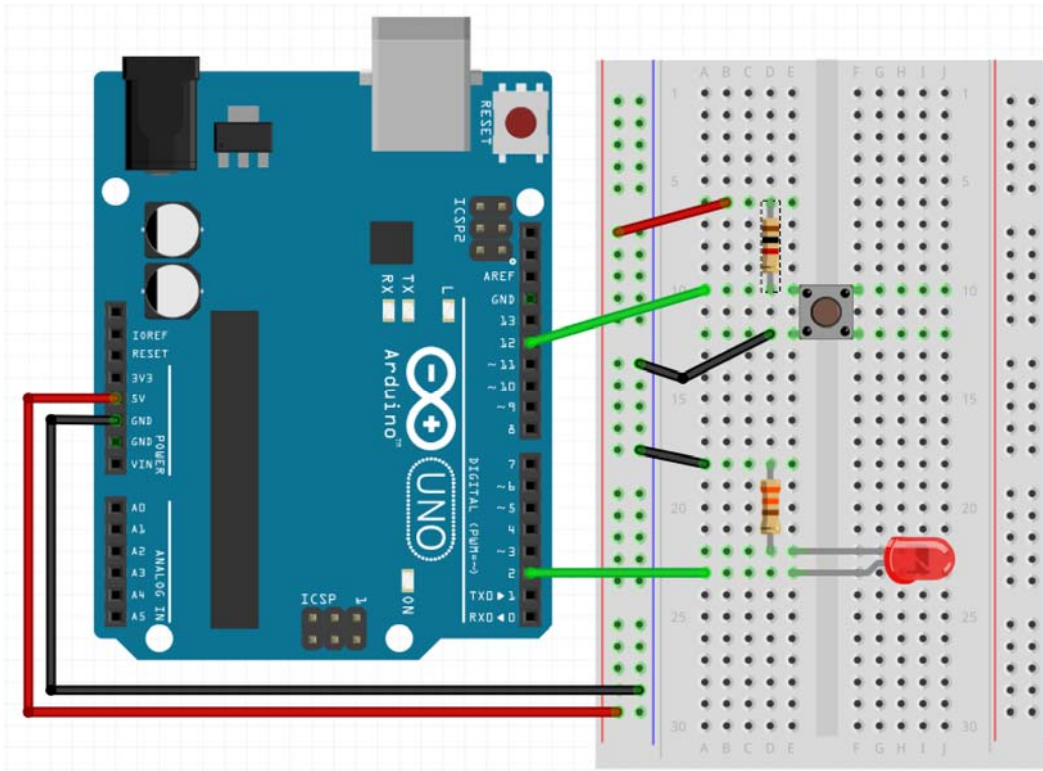
Ej.: `delay(500);` // Detiene el programa durante 500 microsegundos.

- `millis()`

- Devuelve, en milisegundos, el tiempo transcurrido desde que empezó a ejecutarse el programa hasta el momento en el que se ejecuta la instrucción.

Ej.: `tiempotranscurrido = millis();`

7. Programación



```
int pinPuls = 12;    //Se almacena en la variable pinPuls el número
                    //del pin al que se conecta el pulsador

int pinLed = 2;      //Se almacena en la variable pinLed el número
                    //del pin al que se conecta el LED

int puls = 0;        // Se define una variable 'puls' en la
                    // que se almacenará el estado del pulsador

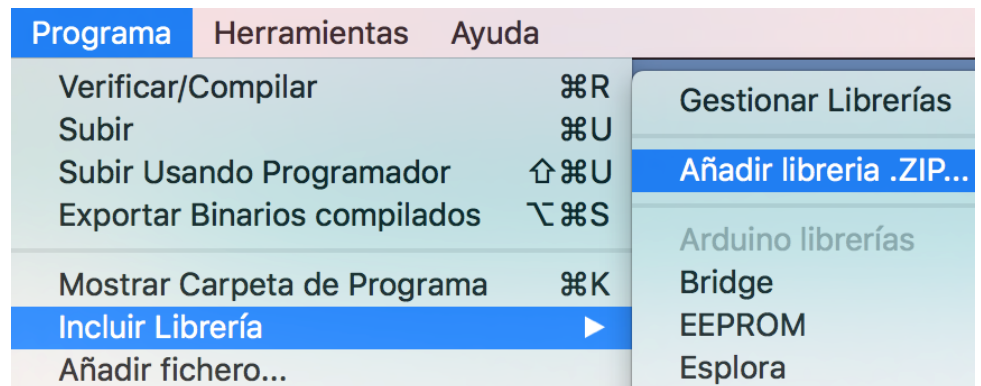
void setup(){
  pinMode(pinLed,OUTPUT); // El pin al que se conecta el LED
                        // se configura como salida
  pinMode(pinPuls,INPUT); // El pin al que se conecta el pulsador
                        // se configura como entrada
}

void loop(){
  puls = digitalRead(12);
  if (puls == HIGH){
    digitalWrite(pinLed,LOW);
  }else{
    digitalWrite(pinLed,HIGH);
  }
}
```

7. Programación

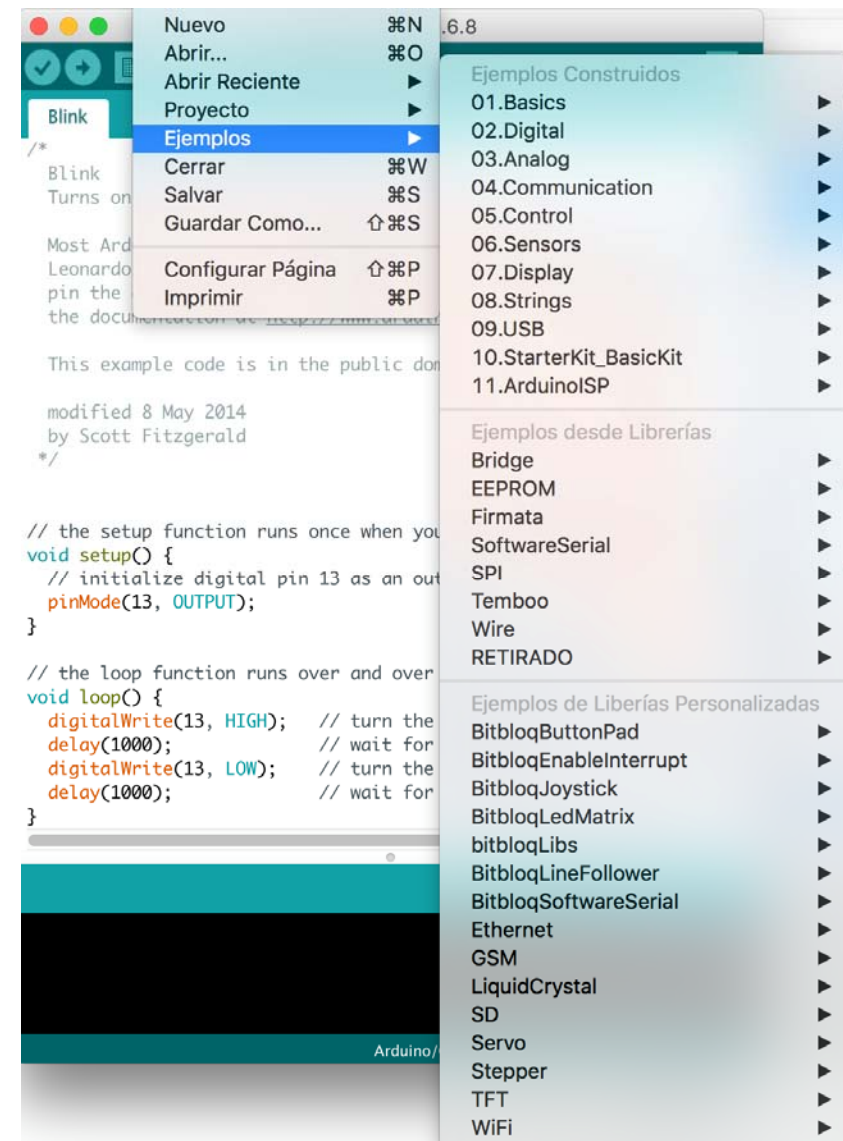
La comunidad de usuarios de Arduino es grande y facilita una amplia variedad de librerías para simplificar la programación de multitud de aplicaciones (*Open Source*).

Para utilizar una librería es necesario descargarse el archivo .zip correspondiente, incluirla en el entorno de programación e incluirla en nuestro programa con la instrucción `#include <NombreLibreria.h>`



7. Programación

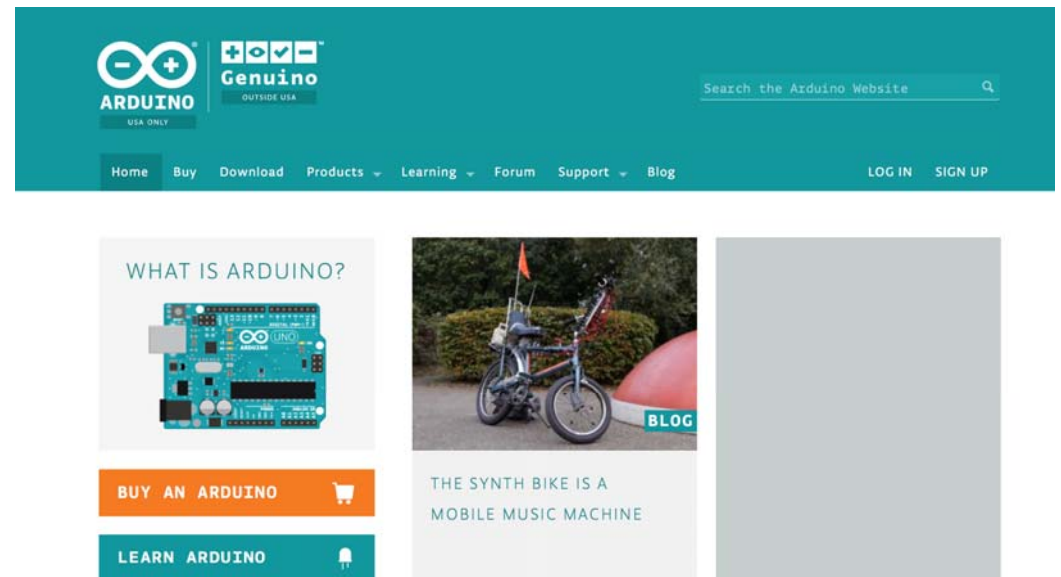
- La mejor manera de empezar son los ejemplos básicos que incluye el propio entorno de programación
- Existen multitud de ejemplos y librerías disponibles en Internet.
- Hardware y software abierto



7. Programación: otros recursos

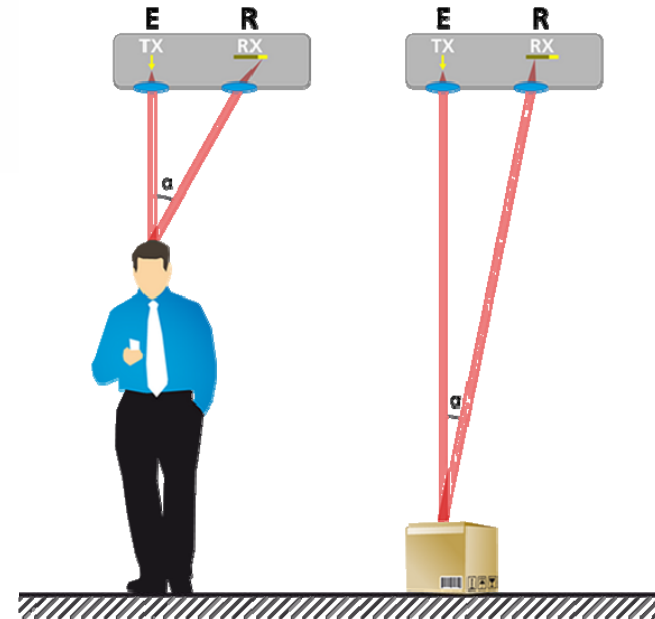
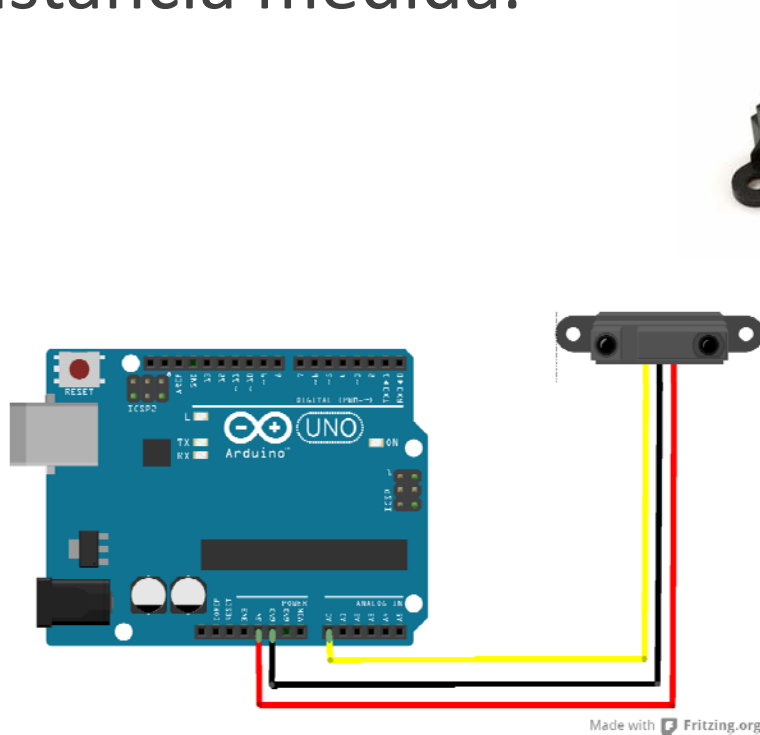
En la página oficial de Arduino, puede consultarse por ejemplo:

- Una referencia completa del lenguaje de programación <https://www.arduino.cc/en/Reference/HomePage>
- Multitud de librerías para todo tipo de aplicaciones y periféricos <http://playground.arduino.cc>



8. Sensores y actuadores

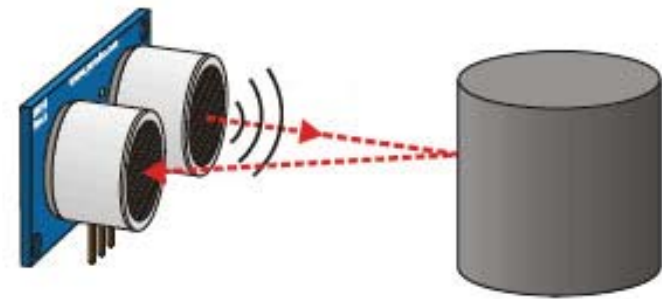
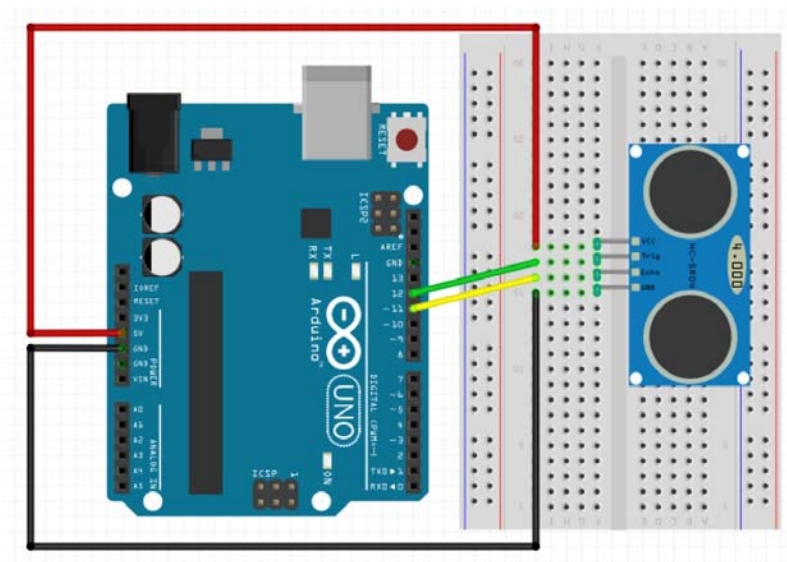
GP2D12: sensor que permite medir distancias mediante infrarrojos. Indica mediante una salida analógica la distancia medida.



<https://dartecne.wordpress.com/recursos/sensores/>

8. Sensores y actuadores

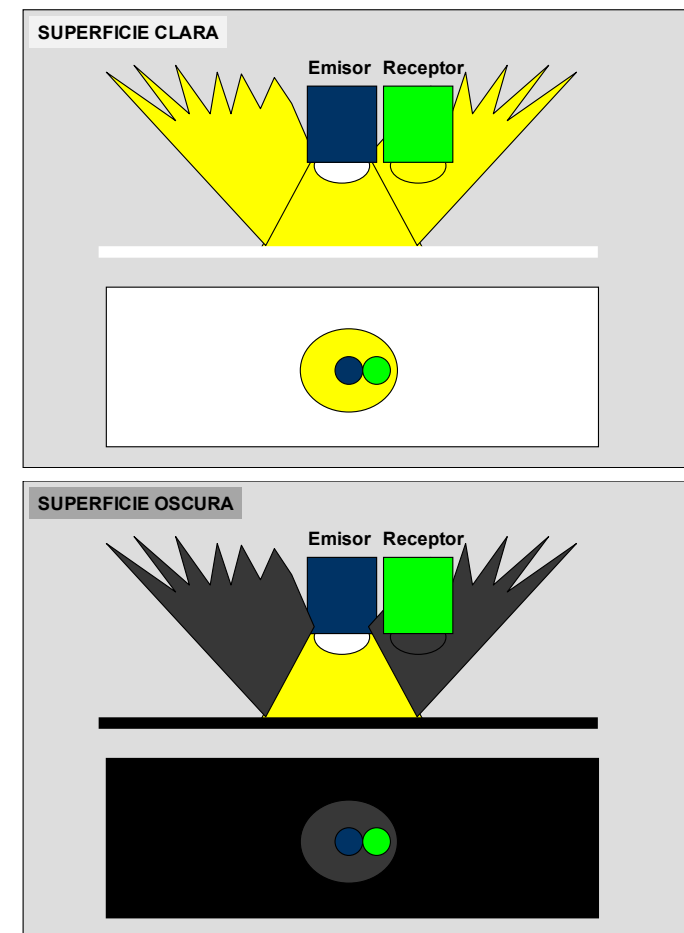
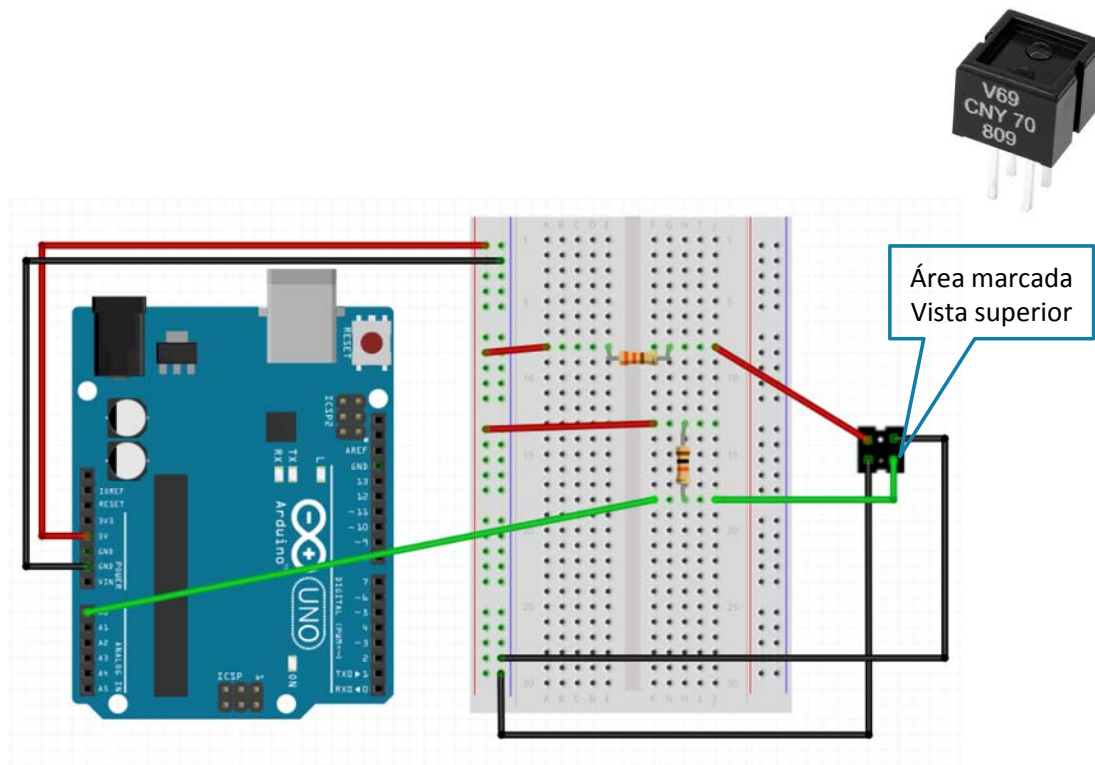
HC-SR04: sensor que permite medir distancias mediante ultrasonidos. Indica mediante una salida analógica la distancia medida.



$$\text{Tiempo} = 2 \cdot (\text{Distancia} / \text{Velocidad})$$
$$\text{Distancia} = \text{Tiempo} \cdot \text{Velocidad} / 2$$

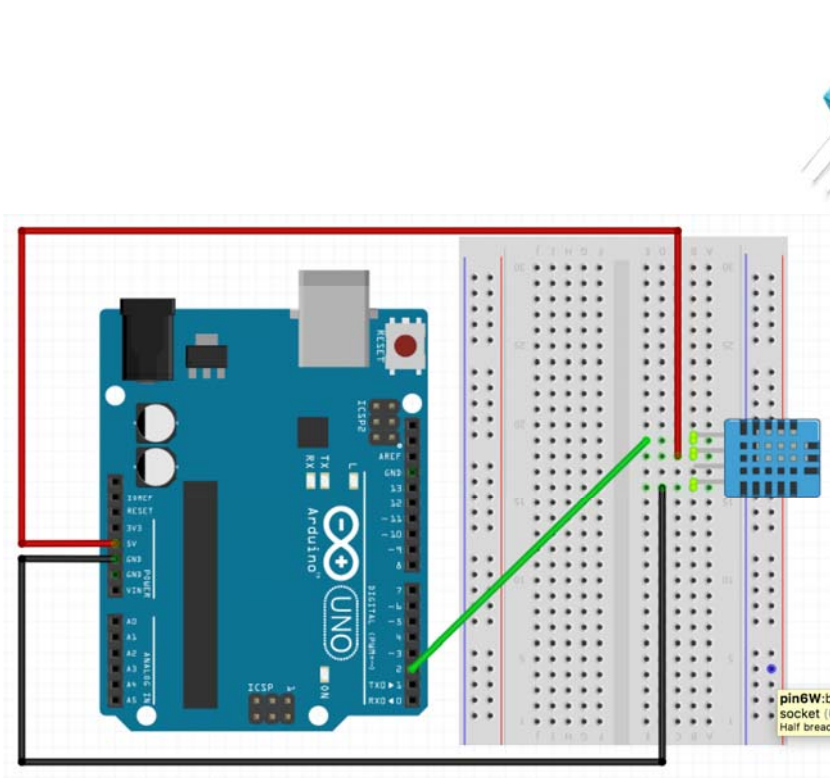
8. Sensores y actuadores

CNY70: sensor óptico infrarrojo, de un rango de corto alcance (menos de 5 cm) que se utiliza para detectar colores de objetos y superficies.



8. Sensores y actuadores

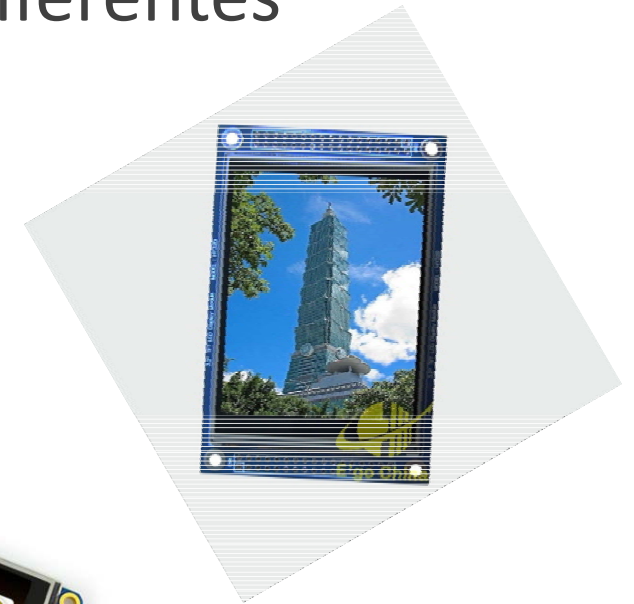
DHT11: sensor de temperatura y humedad que funciona con 3,3 o 5V de alimentación.



- Rango de temperatura: de 0º a 50º con 5% de precisión.
- Rango de humedad: de 20% al 80% con 5% de precisión.
- 1 muestra por segundo
- Bajo consumo.
- Devuelve la medida en ºC mediante una señal digital.

8. Sensores y actuadores

Sistema de visualización: es un dispositivo que nos permite mostrar información de manera gráfica. Existen muchos sistemas distintos de variada complejidad (número de caracteres, colores, etc.) y diferentes tecnologías.



8. Sensores y actuadores

Motor paso a paso: Un motor paso a paso (stepper en inglés) es un sistema electromecánico en el que impulsos eléctricos provocan desplazamientos en ángulos discretos o pasos. Los pasos pueden ser muy pequeños, por lo que son motores muy precisos con un posicionamiento muy exacto.

