DISPOSITIVOS HARDWARE E INTERFACES

Tema 2. El Arduino

Profesores de la asignatura

GTEC
Grupo de Tecnología Electrónica y Comunicaciones

UNIVERSIDADE DA CORUÑA

Índice

Tema 2. Arduino

Índice

Tema 2. Arduino

- Guía de programación
  - Variables y Constantes
  - Funciones de I/O digitales
  - Funciones de I/O analógicas
  - Funciones de tiempo
  - Funciones de números aleatorios
- Uso de Librerías
- Otros entornos de desarrollo

- Periféricos I/O
  - Puertos I/O
    - Entradas/salidas digitales
    - Entradas/salidas analógicas
  - Puerto serie USART
  - Interrupciones
  - Temporizadores/Contadores
  - Modos de bajo consumo

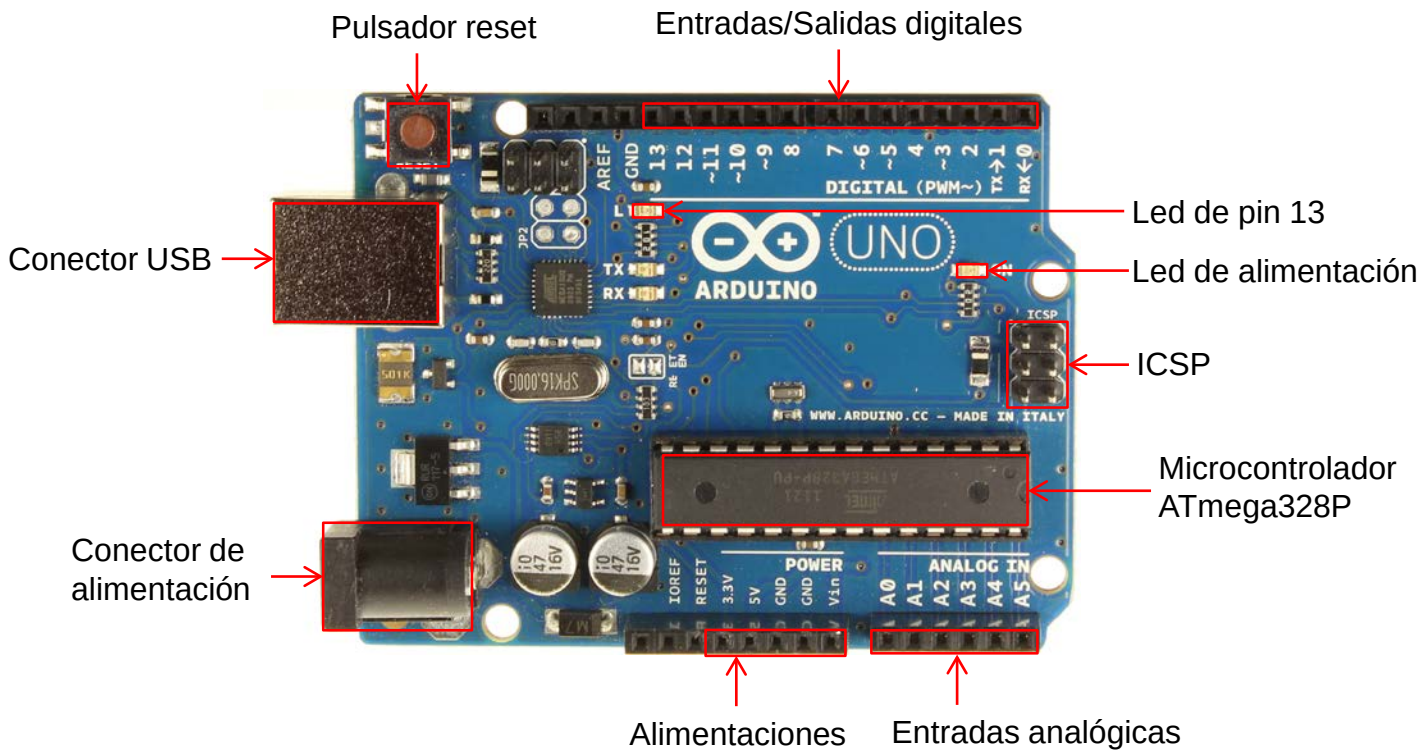
Hardware

¿Qué es Arduino?

- Arduino es una plataforma abierta hardware y software pensada para hacer más accesible el desarrollo de proyectos que usan electrónica.
- El hardware Arduino consiste en una tarjeta de hardware abierto que incluye un microcontrolador AVR o ARM de Atmel y un soporte de entrada/salida.
- El software Arduino consiste en un lenguaje de programación estándar y un programa cargador (*bootloader*) que se ejecutan en la misma tarjeta.
- El hardware Arduino se programa usando un lenguaje de programación similar a C/C++ con algunas simplificaciones y modificaciones, y un entorno de desarrollo IDE.
- Arduino es ideal para iniciarse en el mundo de la electrónica y programación, aunque también puede ser utilizado por programadores avanzados.

Hardware

Arduino Uno



UNIVERSIDADE DA CORUÑA GTEC

Hardware

Arduino Uno. Alimentación

- **Arduino Uno** es una tarjeta basada en el µC ATmega328 de Atmel. Dispone de 14 pines de entrada/salida digitales (de la cuales 6 pueden ser usados como salidas PWM), 6 entradas analógicas, oscilador a 16 MHz, conexión USB, conector de alimentación, bloque de terminales para ICSP y pulsador de reset. Además incorpora un µC ATMega16U2 programado como convertidor de USB a serie.

- El Arduino Uno puede ser alimentado vía la conexión USB (5 V) o con un alimentador externo (7 a 12 V no regulados). La fuente de alimentación la elige él automáticamente. Además dispone de varios terminales de alimentaciones:

Vin.- Entrada alternativa de alimentación externa (7 a 12 V no regulados).

5 V.- Tensión de salida regulada de +5 V.

3,3 V.- Tensión de salida regulada de +3,3 V. Máxima corriente suministrada 50 mA.

GND.- Masa común de circuitos.

- El ATmega328 tiene 32 kB de memoria flash (0,5 kB ocupados con el programa cargador), 2 kB de SRAM y 1 kB de EEPROM (Librería EEPROM).

Hardware

Arduino Uno. Entradas/Salidas digitales

- Cada uno de los 14 pines de entrada/salida digitales (0 a 13) puede ser usado como entrada o como salida [*pinMode*(), *digitalWrite*(), y *digitalRead*()].
- Las entradas/salidas digitales operan en 5 voltios, cada pin puede suministrar o drenar un máximo de 40 mA, y tienen una resistencia de elevación (*pull-up*) interna de 20 a 50 kΩ desconectada por defecto.
- Algunos pines poseen además funciones especiales:

0 (Rx) y 1 (Tx).- Recepción y transmisión de datos en serie (USART) en niveles TTL. Internamente están conectadas a las líneas correspondientes del ATMega16U2.

2 (INT0) y 3 (INT1).- Entradas de interrupción externa [*attachInterrupt*()].

3, 5, 6, 9, 10 y 11.- Salidas PWM de 8 bits [*analogWrite*()].

10 (SS), 11 (MOSI), 12 (MISO) y 13 (SCK).- Interfaz SPI (Librería SPI).

13.- Led que enciende con salida en alto (HIGH).

Hardware

Arduino Uno. Entradas analógicas

- El Arduino Uno tiene 6 entradas analógicas (0 a 5) cada una de las cuales proporciona un valor de 10 bits de resolución (0 a 1023). Por defecto el rango de medida va de 0 a 5 voltios, pero puede ser modificado usando la entrada AREF (0 a VAREF) o una tensión de referencia interna de 1,1V (0 a 1,1 V).
- Además algunos pines tienen funciones especiales:

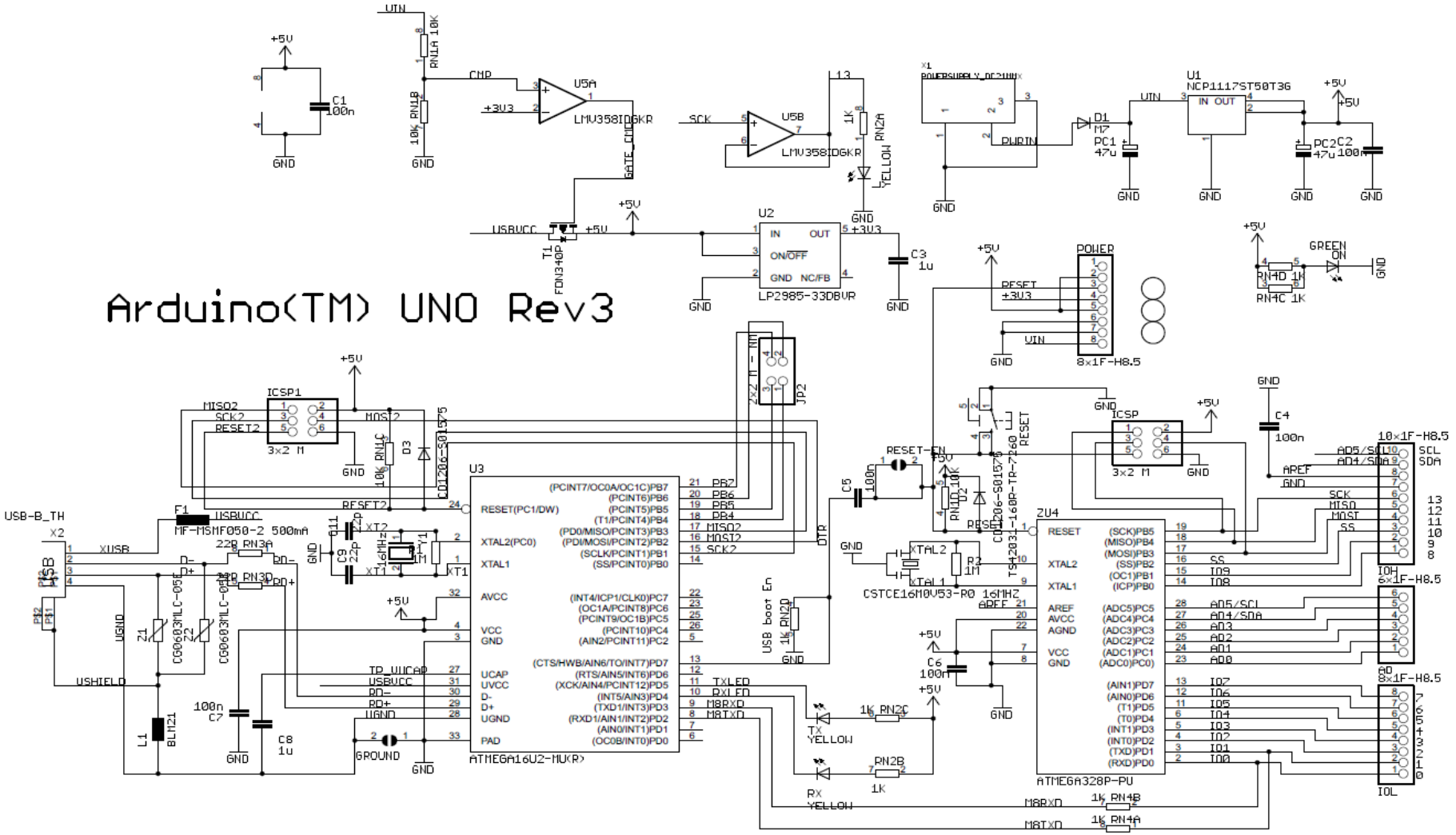
4 (SDA) y 5 (SCL).- Proporcionan comunicaciones serie I2C (TWI) usando la librería *Wire*. Además, en el Arduino UNO la línea SDA está repetida en el pin digital 16 y la línea SCL está repetida en el pin 17.

- Hay un par de pines adicionales en la placa:

AREF.- Entrada para aplicar una referencia de tensión externa entre 0 y 5 V usando [*analogReference*(EXTERNAL)].

Reset.- Un nivel bajo en esta entrada proporciona un reset del procesador.

Hardware (Esquema)



UNIVERSIDADE DA CORUÑA

GTEC

Hardware

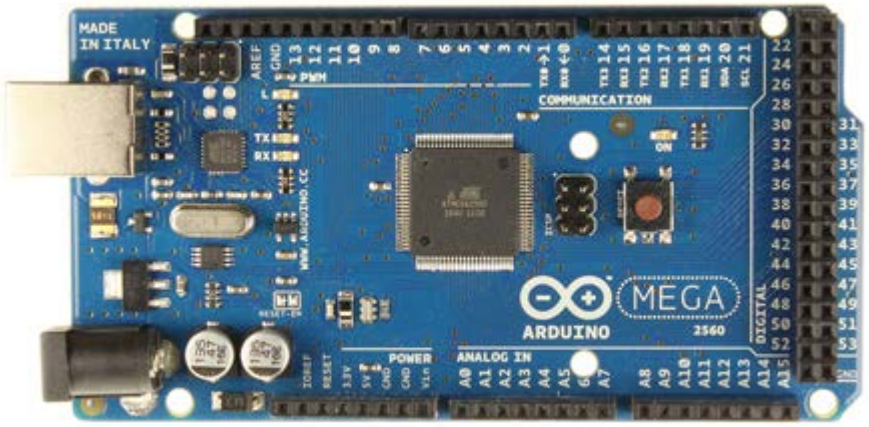
Arduino Ethernet

- La tarjeta de ampliación Arduino Shield Ethernet permite conectar una placa Arduino a una red LAN utilizando la librería *Ethernet*.
- Dispone de un conector Ethernet estándar tipo RJ-45 con un transformador integrado y capacidad de alimentación por Ethernet (PoE). También incluye un lector de tarjetas micro-SD accesibles a través de la librería *SD*.

Hardware

Arduino Mega 2560

- El Arduino Mega 2560 es una tarjeta basada en el µC Atmega2560 de Atmel. Tiene 54 pines de entrada/salida digital (14 salidas PWM), 16 entradas analógicas, 4 UARTs, oscilador a 16 MHz, conexión USB a través de convertidor realizado con ATmega16U2, conector de alimentación externa, ICSP y pulsador de reset.
- El ATmega2560 tiene 256 kB de memoria flash (8 kB ocupados con el programa cargador), 8 kB de SRAM y 4 kB de EEPROM (Librería EEPROM).



UNIVERSIDADE DA CORUÑA GTEC

Hardware

Arduino Zero Pro

- El Arduino Zero Pro es una extensión de 32 bits de la plataforma UNO. Está basado en el µC SAMD21G18 ARM Cortex-M0+ de Atmel. Tiene 20 pines de entrada/salida, 2 UARTs, oscilador a 48 MHz, 2 puertos USB (nativo + programación con chip EDBG que permite depuración usando Atmel Studio, 1 I2C (TWI), ICSP con SPI, JTAG, 12 canales DMA, RTC de 32 bit, generador de CRC, etc.
- El SAM3X8E es un ARM Cortex M0+ de 32 bits con 256 kB de memoria flash, 32 kB de SRAM y 32 kB de EEPROM por emulación.

UNIVERSIDADE DA CORUÑA GTEC

Hardware

Arduino Due

- El Arduino Due es una tarjeta basada en el µC SAM3X8E ARM Cortex-M3 de Atmel. Tiene 54 pines de entrada/salida digital (12 salidas PWM), 12 entradas analógicas, 4 UARTs, oscilador a 84 MHz, 1 puerto USB OTG, 2 I2C (TWI) ,conector de alimentación externa, ICSP, JTAG, pulsadores de reset y borrado, CAN, etc. Para el desarrollo del software puede usarse el IDE Arduino v1.5.
- El SAM3X8E es un ARM Cortex M3 de 32 bits con 512 kB de memoria flash, 96 kB de SRAM y controladora DMA.

Hardware

Arduino Yún

- El Arduino Yún es una combinación de un Arduino Leonardo basado en el ATmega32U4 con un sistema Wifi sobre una distribución GNU/Linux denominada OpenWrt-Yun. Integra la máquina Linux con el Arduino Leonardo de tal modo que permite ejecutar comandos de Linux para acceso a las interfaces Ethernet y Wifi.
- El sistema Wifi está basado en el procesador AR9331 e incorpora un conector USB adicional y tarjeta SD.

UNIVERSIDADE DA CORUÑA GTEC

Software

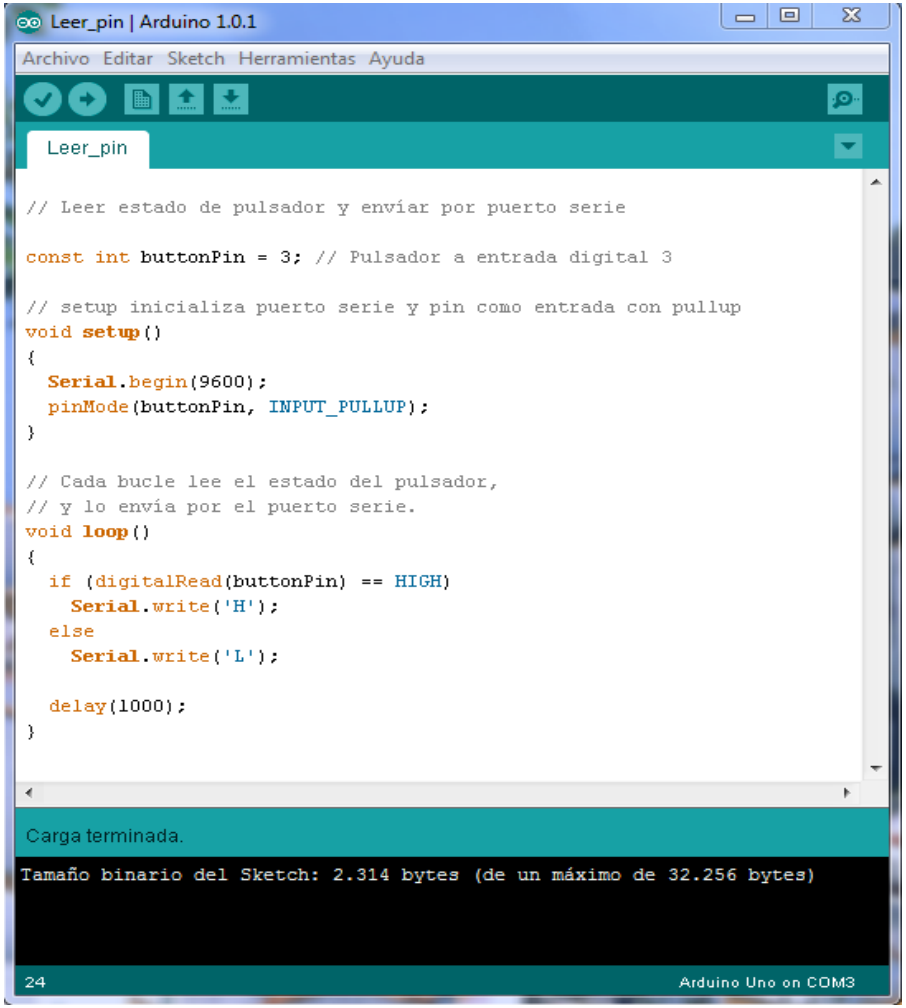
Introducción

- El lenguaje de programación del Arduino es el *Wiring* (basado en C/C++).
- La principal ventaja del lenguaje *Wiring* es proporcionar un conjunto de funciones que encapsulan el funcionamiento del hardware, empezando por la propia estructura del programa (*sketch*). Además permite utilizar las características de C/C++ como funciones, punteros, clases, objetos, etc., incluso utilizar lenguaje ensamblador, y características propias del compilador de C/C++ para microcontroladores AVR/ARM.
- *Wiring* también permite desarrollar librerías de funciones que pueden instalarse en el propio entorno de desarrollo, existiendo gran cantidad de ellas en Internet para control de motores, displays LCD, comunicaciones serie, GPS, etc.
- El entorno de desarrollo Arduino funciona bajo Linux, Windows y MacOS X, gracias a que fue desarrollado en Java. Permite la creación de programas llamados *sketchs* en lenguaje *Wiring*, compilarlos mediante el compilador de C/C++ de AVR/ARM y sus librerías, y descargarlos (*download*) a la tarjeta Arduino a través de un puerto USB.

Software

Entorno de desarrollo (IDE)

- El entorno de desarrollo (IDE) Arduino contiene un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas para funciones típicas, y un conjunto de menús. Se conecta con el hardware Arduino para cargar programas y comunicarse con ellos.



UNIVERSIDADE DA CORUÑA GTEC

Software

Lenguaje de programación *Wiring*

- El *sketch* equivale a un proyecto de un IDE, consiste en uno o más ficheros escritos en lenguaje *wiring*. La principal diferencia entre C/C++ y *wiring* es que en éste no hay que declarar funciones globales antes de ser usadas, puesto que todos los ficheros de un *sketch* comparten la misma unidad de traducción. Se pueden añadir ficheros .c y .cpp que son tratados de modo convencional.
- El lenguaje wiring no requiere definir el método **main()**, y en su lugar son necesarias dos funciones: **setup()** y **loop()**. Cada vez que se arranca un *sketch*, se ejecuta una sola vez setup() mientras que loop() se ejecuta repetitivamente.
- El compilador IDE realiza una traducción fuente a fuente desde los ficheros del *sketch* a C/C++. En primer lugar, por concatenación de los ficheros .ino, produce un fichero que lista todas las funciones definidas por el usuario. Después incluye la librerías de funciones básicas y declaraciones de funciones como setup() y loop(). Por último incorpora la función main() que proporciona el bloque definitivo que pasa al compilador de C/C++.

Software

Estructura de un programa

- La estructura básica de un programa Arduino es bastante simple y se compone de al menos dos funciones básicas y necesarias para que trabaje el programa:

1) La función de configuración **setup()** es la primera función a ejecutarse (una sola vez), y se utiliza fundamentalmente para las inicializaciones de los periféricos I/O. Se debe escribir a continuación de la declaración de las variables globales.

2) La función de bucle **loop()** contiene el código que se ejecuta continuamente, es el núcleo del programa Arduino y la que realiza la mayor parte del trabajo.

```
// Leer estado de pin y envíar por puerto serie

const int pinPin = 3; // Pin a entrada digital 3

// setup inicializa puerto serie y pin como entrada con pullup
void setup()
{
  Serial.begin(9600);
  pinMode(pinPin, INPUT_PULLUP);
}

// Cada bucle lee el estado del pin,
// y lo envía por el puerto serie.
void loop()
{
  if (digitalRead(pinPin) == HIGH)
    Serial.write('H');
  else
    Serial.write('L');

  delay(1000);
}
```

Guía de programación AVR: Variables

- **Tipos de datos básicos**:

boolean	Valor lógico true o false que ocupa un byte en memoria
byte	Valor entero sin signo de 8 bits (0 a 255)
int	Valor entero de 16 bits (-32768 a 32767)
long	Valor entero de 32 bits (-2147483648 a 2147483647)
float	Valor real en coma flotante en 32 bits (-3,402835E+38 a 3,402835E+38)

- **Alcance (Scope)**: El alcance de una variable queda determinado por el lugar donde se declara. Según su alcance las variables pueden ser:

Variables globales.- Suelen ser declaradas al inicio del programa antes de la función *setup*() y pueden ser usadas por cualquier función o instrucción del programa.

Variables locales.- Se declaran dentro de una función o de un bucle *for*, y sólo pueden ser usadas dentro de la función o bucle ya que al salir se pierden. Si son declaradas con el especificador de clase de almacenamiento **static**, su valor es retenido entre llamadas a la función.

Software

Guía de programación AVR: Constantes

HIGH/LOW	Definen los niveles alto (HIGH) o bajo (LOW) usados cuando se lee o se escribe en los pines digitales de entrada/salida (I/O). HIGH se define como el nivel lógico 1, estado ON, o +5 V, mientras que LOW es el nivel lógico 0, estado OFF, o 0 V.
INPUT/OUTPUT	Con la función *pinMode*() se usan constantes para definir el modo de funcionamiento del pin: INPUT para modo entrada, INPUT_PULLUP para modo entrada con resistencia de *pullup*, y OUTPUT para modo salida.
true/false	Constantes booleanas que definen valores lógicos: falso (false) es definido como 0, y cierto (true) cualquier valor entero no cero.

- El calificador de tipo **const** permite declarar e inicializar cualquier variable en memoria. Es un valor constante, no se puede cambiar. Es más general que #define.

const int buttonPin = 3;

Software

Guía de programación AVR: Funciones de I/O digitales

pinMode (pin, mode) Usada en setup() configura el 'pin' digital de I/O como entrada o como salida según la constante mode usada:

INPUT — Configura un pin como entrada de alta impedancia (100 MΩ). Es el estado por defecto después de un reset.
pinMode (buttonPin, **INPUT**);

INPUT_PULLUP — Configura un pin como entrada con una resistencia interna de elevación (*pullup*) de unos 20 kΩ.
pinMode (buttonPin, **INPUT_PULLUP**);

OUTPUT — Configura un pin como salida de baja impedancia con capacidad para suministrar o drenar una corriente de hasta 40 mA
pinMode (ledPin, **OUTPUT**);

- Los pines de entrada analógicos pueden ser también usados como pines digitales sin más que referirse a ellos como A0, A1, etc.
  const int buttonPin = **A3**;

Software

Guía de programación AVR: Funciones de I/O digitales

digitalRead (pin) Lee el estado HIGH o LOW del 'pin' digital de I/O, previamente configurado como INPUT o INPUT_PULLUP. 'pin' puede ser una variable o una constante.
if (**digitalRead** (buttonPin) == HIGH)

digitalWrite (pin, valor) Escribe el 'valor' HIGH o LOW a un pin digital I/O, previamente configurado como OUTPUT.
digitalWrite (ledPin, HIGH);

```
// Leer estado de pin y sacar a led
const int pinPin = 3; // Pin a entrada digital 3
const int ledPin= 13; // Led a salida digital 13

// setup inicializa entrada pin con pullup y salida led
void setup(){
  pinMode(ledPin, OUTPUT);
  pinMode(pinPin, INPUT_PULLUP);
}

// Cada bucle lee el estado del pin y saca a led
void loop() {
  digitalWrite(ledPin, digitalRead(pinPin));
}
```

Software

Guía de programación AVR: Funciones de I/O analógicas

analogRead (pin) Lee un valor de un pin de entrada analógico con 10 bits de resolución siendo el resultado un valor entero de 0 a 1023. Los pines analógicos no necesitan ser previamente configurados.
if (**analogRead** (sensorPin) == 0)

analogWrite (pin, valor) Escribe un valor 'analógico' mediante salida por modulación de anchura de impulsos (PWM).
analogWrite (ledPin, valor);

```
// Leer valor de pin y sacar a PWM
const int anaPin = 0; // Pin a entrada analógica 0
const int PWMPin= 10; // Salida PWM 10

void setup(){
}

// Cada bucle lee el valor de pin y saca a PWM
void loop() {
  analogWrite(PWMPin, analogRead(anaPin)/4);
}
```

Software

Guía de programación AVR: Funciones de tiempo

delay (ms)

Genera una espera en la ejecución del programa de un tiempo en milisegundos igual al valor especificado como *unsigned long*. Durante la espera los periféricos siguen operando y la CPU atiende las interrupciones normalmente. Usa T0.
delay (1000); // Espera de 1000 ms (1 s)

delayMicroseconds (µs)

Genera una espera en la ejecución del programa de un tiempo en microsegundos igual al valor especificado *unsigned int*. Durante la espera los periféricos siguen operando y la CPU atiende las interrupciones normalmente. Precisión sólo de 3 µs.
delayMicroseconds (50); // Espera de 50 µs

millis ()

Devuelve el número de milisegundos desde el inicio de ejecución del programa hasta el momento actual. Es un *unsigned long* que desbordará después de aproximadamente 50 días. Usa T0.
time = **millis**(); // Guarda en *time* el tiempo en *ms* desde inicio

micros ()

Devuelve los microsegundos, con resolución de 4 µs, desde el inicio de ejecución del programa hasta el momento actual. Es un *unsigned long* que desbordará después de 70 minutos. Usa T0.

Software

Guía de programación AVR: Funciones de números aleatorios

randomSeed (semilla) Establece un valor o **semilla** de inicio de la función *random*(). Se pueden utilizar como valores semilla los valores devueltos por las funciones *millis*() o *analogRead*(). **randomSeed**(analogRead(0));

random (max)
random (min,max) Devuelve un valor entero (*long*) **pseudo-aleatorio** dentro del intervalo comprendido entre *min* y *max*-1. Si no se especifica el valor *min* toma por defecto *min*= 0
random(10, 20); // Genera pseudo-aleatorio entre 10 y 20

```
// Número aleatorio a puerto serie USART

// Inicializa puerto serie y semilla random
void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
}
// Cada bucle genera valor aleatorio de 0 a 9
// y lo envía por el puerto serie.
void loop(){
  Serial.println(random(10));
  delay(100);
}
```

Software

Uso de Librerías

- Las librerías proporcionan funcionalidades extra para trabajar con hardware o manipular datos. Para usar una librería en un *sketch* se selecciona desde *Sketch*> *Importar Librería*, y el IDE la introduce automáticamente: #**include** <EEPROM.h>

- El IDE Arduino dispone de la siguiente lista de **Librerías estándar**:

EEPROM	Leer/Escribir en memoria no volátil (EEPROM).
Ethernet	Conexión a Internet usando la placa Arduino Shield Ethernet.
Firmdata	Conexión con aplicaciones en ordenador usando un protocolo estándar.
GSM	Para conectarse a una red GSM/GPRS.
LiquidCrystal	Control de display LCD.
SD	Acceso a tarjetas SD.
Servo	Control de servomotores.
SoftwareSerial	Comunicaciones serie en cualquier pin.
SPI	Comunicaciones serie con dispositivos de bus SPI.
Stepper	Control de motores paso a paso.
TFT	Permite presentar textos y gráficos en una pantalla TFT.
WiFi	Proporciona conexión a Internet usando Wifi.
Wire	Comunicaciones serie con dispositivos de bus I2C (TWI).

Software

Otros entornos de desarrollo

Scratch S4A y Minibloq son entorno de programación gráfica que incluyen bloques de sensores y actuadores conectados al Arduino, cuyo objetivo es llevar la computación física y la robótica a la escuela primaria y a los principiantes.

Labview es un entorno de programación gráfica muy extendido en el campo de la instrumentación. Se conecta a la tarjeta Arduino mediante la interfaz LIFA (*Labview Interface for Arduino*) para adquirir datos del µC, procesarlos en Labview y reenviar.

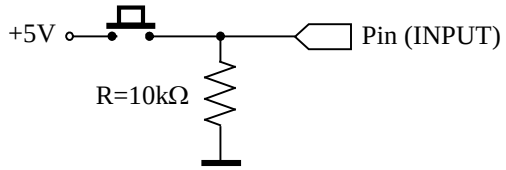
El conjunto Visual Studio + Visual Micro es una buena alternativa profesional al limitado IDE del Arduino, y además disponen de versiones gratuitas. Para uso más profesional **Atmel Studio** es el IDE gratuito más recomendable para las familias AVR y ARM de Atmel, y también permite tratar directamente los proyectos Arduino.

Fritzing es un software de automatización de diseño electrónico basado en una tarjeta de prototipado (*protoboard*). Tiene interés didáctico porque visualiza los componentes electrónicos y su interconexión. Permite obtener el esquema eléctrico y el trazado de las pistas de circuito impreso (PCB).

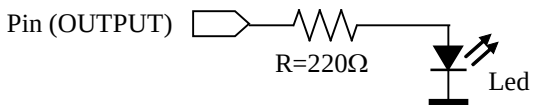
Puertos I/O. Entradas/Salidas digitales

- La entrada/salida digital es la más sencilla con solo dos posibles estados: HIGH o LOW. En el siguiente ejemplo se lee un pulsador conectado al PIN3; cuando está pulsado lee HIGH y encenderá un LED colocado en el PIN13 configurado salida.

- **Entrada digital de pulsador:**



- **Salida digital a led:**

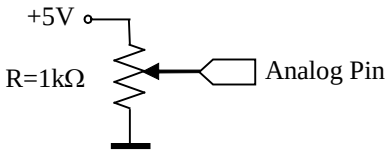



```
// Leer estado de pulsador y sacar a led

// Definición de entradas/salidas
const int pulsPin = 3; // Pulsador a entrada digital 3
const int ledPin= 13; // Led a salida digital 13

// setup inicializa entrada pulsador y salida led
void setup(){
  pinMode(ledPin, OUTPUT);
  pinMode(pulsPin, INPUT);
}

// Cada bucle lee el estado del pulsador y saca a led
void loop() {
  digitalWrite(ledPin, digitalRead(pulsPin));
}
```

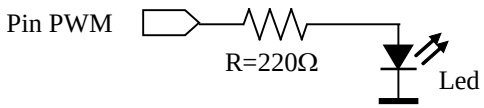
Puertos I/O. Entradas/Salidas analógicas

- Las entradas analógicas permiten la conversión de señales de tensión a valores digitales entre 0 y 1023. El siguiente ejemplo utiliza un potenciómetro en A0 para controlar el brillo de un led conectado en la salida analógica-PWM 10.

- **Entrada analógica de potenciómetro:**



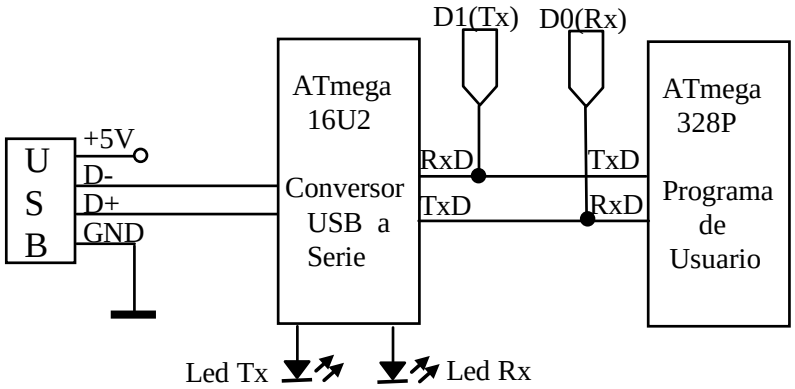
- **Salida analógica a led:**




```
// Leer valor de potenciómetro y sacar a PWM
// para controlar el brillo de un led

// Definiciones
const int potPin = 0; // Pot a entrada analógica 0
const int PWMPin= 10; // Salida PWM 10

// Las entradas/salidas analógicas no se configuran
void setup(){
}

// Cada bucle lee el valor del pot y saca a PWM
void loop() {
  analogWrite(PWMPin, analogRead(potPin)/4);
}
```

Puerto serie USART

- La unidad USART del ATMega328 permite la recepción y transmisión de datos en serie en niveles TTL. Internamente está conectada a la unidad correspondientes del ATMega16U2, que se encarga de realizar la conversión de USB a serie.
- Cuando se utiliza la comunicación serie los pines digitales 0 (RX) y 1 (TX) no pueden utilizarse al mismo tiempo para otra función.



Serial.begin (velocidad) Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos. El valor típico es de 9600, pero soporta otras velocidades desde 300 bps hasta 2 Mbps.

Serial.begin (9600);

Periféricos I/O

Puerto serie USART. Transmisión

Serial.print (dato, formato)
Serial.print (dato)

Envía al puerto serie un número o una cadena de caracteres en formato ASCII. Si el dato es numérico lo envía como valor en decimal, y si es un tipo *float* saca dos decimales. Si es un dato tipo cadena de caracteres lo envía tal cual. Para formato se pueden especificar: DEC (decimal), OCT (octal), BIN (binario), y HEX (hexadecimal). Un valor numérico en el campo formato especifica los decimales de un *float*.

Serial.print (1.23456, 4); // Saca "1.2346"
Serial.print (F("Hola")); // Saca "Hola" desde memoria flash

Serial.println (dato, formato)
Serial.println (dato)

Envía al puerto serie un número o una cadena de caracteres en formato ASCII, seguido de un carácter de retorno de carro (CR o '\r') y de un avance de línea (LF o '\n').

Serial.println (1.23456); // Saca "1.23" y avanza a siguiente

- Estas dos funciones devuelven el control antes de que se realice la transmisión.

Puerto serie USART. Recepción/Transmisión

Serial.available() Devuelve un valor entero con el número de bytes (caracteres) disponibles para leer del buffer de recepción del puerto serie. Este buffer puede almacenar un máximo de 64 bytes (defecto).

if (**Serial.available**() > 0)

Serial.read() Lee o captura un byte (un carácter) desde el buffer del puerto serie. Si no hay ningún carácter disponible devuelve -1.

Serial.write(dato) Envía un byte o una serie de bytes al puerto serie.

```
// Realiza un eco de cada carácter recibido

// Configura puerto serie
void setup() {
  Serial.begin(9600);
}

// Detecta carácter recibido y retransmite
void loop() {
  if (Serial.available()> 0) {
    Serial.write(Serial.read());
  }
}
```

Interrupciones. Definición

- Las interrupciones son útiles para automatizar funciones del programa y resolver problemas de tiempo. Las interrupciones externas se usan por ejemplo para leer impulsos procedentes de un encoder, monitorizar una entrada de usuario, etc.

attachInterrupt(nº_int, función, modo) Especifica la función a ser llamada cuando ocurre una determinada interrupción. En el Arduino Uno hay dos interrupciones externas asociadas a dos entradas digitales: la número 0 asociada al pin 2 y la número 1 asociada al pin 3.
nº_int.- Número de interrupción (0 o 1).
función.- Función a ser llamada que no recibe ni devuelve parámetros, y es la Rutina de Servicio de Interrupción (ISR).
modo.- Define el modo en que la entrada es sensible a la interrupción: LOW indica disparo por nivel bajo, CHANGE disparo por cambio de nivel, RISING disparo por flanco de subida, y FALLING disparo por flanco de bajada.

detachInterrupt(nº_int.) Deshabilita la interrupción especificada. El procesador deja de atender la interrupción.

Interrupciones. Ejemplo

- Es importante hacer notar que dentro de la función que atiende una interrupción la función *millis*() no se incrementa y, por otro lado, pueden perderse caracteres recibidos por el puerto serie. Además las variables globales que se modifican dentro de una ISR deben ser declaradas con el calificador de tipo **volatile**, para instruir al compilador para que acceda a ellas en RAM y no en registros temporales.
- El siguiente ejemplo muestra como se usan las interrupciones para atender el cambio de estado de un pulsador (pin 2) para control de un led (pin 13). Se observa que no afecta el retardo de 5 s introducido en el bucle principal (*loop*), dado que se atienden las interrupciones dentro de la función *delay*(5000).

```
// Pulsador a led usando interrupción
const int pulsad= 2;
const int pinled= 13;
void setup() {
  pinMode(pinled, OUTPUT);
  attachInterrupt(0, sacar_led, CHANGE);
}
void loop() {
  delay (5000);
// digitalWrite(pinled, digitalRead(pulsad));
}
void sacar_led() {
  digitalWrite(pinled, digitalRead(pulsad));
}
```

Temporizadores/Contadores

- Un temporizador/contador (*timer/counter*) es una unidad hardware interna del ATmega328 que permite la medida y control de tiempos y el conteo de eventos.
- El Arduino dispone de 3 temporizadores/contadores:

Timer 0 (8 bits).- Usado por las funciones de temporización *delay*(), *millis*(), *micros*(), y las salidas analógicas PWM 5 y 6.
Timer 1 (16 bits).- Usado por la librería Servo, y las salidas analógicas PWM 9 y 10.
Timer 2 (8 bits).- Usado por la función *tone*(), y las salidas analógicas PWM 11 y 3.

MsTimer2

La librería MsTimer2 permite usar el Timer 2 para generar interrupciones con un intervalo de tiempo de 1 ms de resolución.

MsTimer2::set (ms, función).- Especifica la función a ser llamada cuando desborda el valor ms.

MsTimer2::start ().- Habilita interrupción.

MsTimer2::stop ().- Deshabilita interrupción.

```
// Contador de segundos con Timer 2
#include <MsTimer2.h>
volatile int contador= 0;
void setup() {
  Serial.begin(9600);
  MsTimer2::set(1000, avanzar); // 1000ms
  MsTimer2::start();
}
void loop() {
}
void avanzar() {
  contador++;
  Serial.println(contador);
}
```

Periféricos I/O

Modos de bajo consumo

- Los modos de bajo consumo (*Sleep*) del ATmega328 permiten pasar a suspensión la CPU y los módulos no usados proporcionando importantes ahorros de energía.

Modos Sleep	CPU	T0/T1/ADC	T2	SPI/USART	INT0/1,WDT, I2C	Consumo
SLEEP_MODE_IDLE						15 mA
SLEEP_MODE_ADC						6,5 mA
SLEEP_MODE_PWR_SAVE						1,62 mA
SLEEP_MODE_EXT_STANDBY						1,62 mA
SLEEP_MODE_STANDBY						0,84 mA
SLEEP_MODE_PWR_DOWN						0,36 mA

- Si ocurre una interrupción habilitada mientras el µC está en modo *sleep*, se despierta y ejecuta la rutina de interrupción correspondiente, y reasume la ejecución en la siguiente instrucción a la del paso a modo *sleep*. En caso de un reset en modo *sleep*, el µC lo ejecuta desde el vector de reset.

Modos de bajo consumo. Gestión

• Para la gestión de los modos de bajo consumo hay que usar la librería del compilador de C/C++ para AVR (*avr-libc*), incluyendo al inicio del programa la directiva **#include <avr/sleep.h>**, que proporciona las siguientes funciones:

set_sleep_mode(modo)	Selecciona el modo *sleep* (ver tabla de modos).
sleep_enable()	Habilita el modo *sleep.*
sleep_disable()	Deshabilita el modo *sleep.*
sleep_cpu()	Pone en modo *sleep.*

• A continuación se muestra *SleepNow*() un ejemplo de una función que pasa el µC al modo ***sleep*** configurado con *set_sleep_mode()*. Observe que se suspende la ejecución de código justo con la ejecución de la función *sleep_cpu*().

```
void sleepNow() {
  sleep_enable();
  sei(); // habilita interrupcciones
  sleep_cpu();
  sleep_disable();
}
```

Watchdog Timer

- El watchdog genera un reset cuando el contador alcanza el valor de tiempo programado. En modo normal de funcionamiento el SW debe realizar un *watchdog* reset (WDR) para reiniciar el contador antes de que desborde.
- Para la gestión del *Watchdog Timer* hay que usar la librería del compilador de C/C++ para AVR (*avr-libc*), incluyendo al inicio del programa la directiva **#include <avr/wdt.h>**, que proporciona las siguientes funciones:

wdt_enable(valor)	Habilita el *watchdog* y carga valor de conteo en la constante: WDTO_15MS, WDTO_30MS, WDTO_60MS, WDTO_120MS, WDTO_250MS, WDTO_500MS, WDTO_1S, WDTO_2S, WDTO_4S, WDTO_8S.
wdt_disable()	Deshabilita el *watchdog.*
wdt_reset()	Reinicia el contador del *watchdog.*

Watchdog Timer. Ejemplo

- A continuación se muestra un test del uso del Watchdog Timer, forzando un reset de *watchdog* ocasionado por un bucle *while* de espera infinito.

```
#include <avr/wdt.h>

void setup ()
{
  Serial.begin (115200);
  Serial.println ("Restarted.");
  wdt_enable (WDTO_1S);  // reset after one second
 }  // end of setup

void loop ()
{
  Serial.println ("Entered loop ...");
  wdt_reset ();  // give me another second
  while (true) ;   // and wait for watchdog-reset
}  // end of loop
```