Vacas Martinez, José Antonio

# Tema 5 - Programación avanzada

En este tema vamos a empezar con un repaso sobre la programación con Arduino (y con todo lo relacionado con C y C++) y también aprenderemos cómo es el proceso de compilación.

A continuación veremos como medir tiempos lo que nos permitirá hacer programas sin esperas (delay)

Veremos el concepto de interrupción y cómo utilizar las interrupciones de tipo software (por tiempo). Terminaremos viendo cómo utilizar los puertos de arduino de un modo avanzado (Registrosº).

## Apéndice: Introducción a la programación en C++

Veamos una breve introducción a la programación, aplicada a Arduino pero que es común con muchos lenguajes en concreto C++

En este capítulo se presenta casi todo lo que vamos a necesitar para programar en el resto del curso. Es un capítulo como referencia, por lo que no te preocupes de aprender, sino que volveremos a él a medida que vayamos usando las distintas opciones.

### Comentarios

Un comentario es una línea que no influye en nuestro programa pero que nos sirve para documentar nuestro código. Es muy importante comentar nuestro código, puesto que nos ayudará a entender lo que estamos haciendo y si pensamos compartirlo con más gente, les facilitará enormemente la labor de comprender lo que pretendemos hacer.

Existen 2 tipos de comentarios:

* El comentario de bloque, todo el contenido entre los símbolos /\* y \*/ será tomado como contenido del comentario
* /\* Hola esto es un comentario tan largo y extenso que no cabe en  
  una sola línea. \*/
* Línea de comentario: a partir de las dos barras // se ignorará el contenido hasta la siguiente línea
* // Este es un comentario de línea

### Bloques

Un bloque es un conjunto de líneas de código que se encuentran encerradas entre dos llaves  
{ … }

Más adelante veremos que determinadas sentencias de control incluyen bloques de instrucciones, como por ejemplo las instrucciones de control o las de decisión  
### Variables

Una variables es una forma de etiquetar y guardar un valor. El valor que contiene puede ser de diferentes tipos, como puede ser un número entero, un número decimal, o un carácter.

Las variables conservan su valor hasta que lo modificamos. Podemos modificar su valor y a partir de ese momento contendrá ese nuevo valor.

El tipo de la variable será definido cuando declaramos la variable por primera vez y a partir de ahí no se modificará. Definiremos la asignación como el momento en que le damos valor a la variable. Podemos declarar y asignar un valor en la misma sentencia. Veamos ejemplos de declaraciones y asignaciones:

int a=10;  
float c=10.2;  
int b;  
b=15;

Las variables tiene un ámbito (scope en inglés) que define la zona del código donde existen. Fuera de esta zona no podemos acceder a su valor.

int a=10;  
void prueba() {  
 float b=10.2\*a;  
}

En este ejemplo la variable *b* sólo está definida dentro de la función prueba, es decir, no podremos usarla fuera de la misma. En cambio *a*, al estar definida fuera, también puede ser usada dentro.

Puedes encontrar más ejemplos en la [página de Arduino](http://www.Arduino.cc/en/Tutorial/Variables).

### Tipos de variables

Veamos algunos de los tipos de variables existentes. Cada tipo determinado, tiene un rango y una precisión:

byte: 8bits 0 a 255  
int: 16bits -32768 a 32767  
word: 16bits 0 a 65535  
long: 32bits -2x10\^6 a 2x10\^6  
float: guarda decimales -3.4x10\^34 a 3.4x10\^34

**¡¡¡Cuidado con los números negativos!!! ¡¡Cuidado con pasarnos!!**

Cuando una variable llega a su límite, tanto superior, como inferior se produce un desbordamiento y continua por el otro límite. Por ejemplo si tenemos una variable de tipo word con valor 0 y le restamos 1 ¡¡¡pasará a tener el valor 65535!!!

### Operadores aritméticos

Representa la operación que se realizará entre los valores de la expresión:

+, -, \\*, /, % (módulo, el resto de la división entre los valores)

Ejemplos:

a=a+3;  
b=a/3;  
c=b%3;

### Operadores

* Operadores Booleanos: && (AND / Y), || (OR / O ), !(NOT / NO) Representan las operaciones lógicas que podemos aplicar a las condiciones
* Operadores de acceso a punteros: \*, & Los usaremos para acceder a memoria
* Operadores de bits: &, |, ^, ~, <<, >> Permiten manipular los bits individuales de las variables representadas en [lenguaje binario](http://es.wikipedia.org/wiki/Binario)
* Operadores compuestos:
* Incremento/decremento de variables: ++, -- Incrementan/decrementan el valor de la variable
* Asignación y operación: +=, -=, \*=, /=, &=, |= Realizan la operación indicada y luego la asignación del resultado a la variable

### Array

Un array es una agrupación de variables con un tipo y un tamaño determinados. Podemos acceder a cada elemento individualmente por medio del operador [], indicando la posición. Empiezan en la posición 0 y la última será la que indica su longitud menos 1.

int miArray[5]; //Declaramos un array de 5 elementos enteros  
int miOtroArray[]={1,23,2}; // Declaramos un array de 3 elementos con sus valores asignados

### Constantes

Son variables a las que no podemos modificar al valor. Algunos ejemplos:

true/false  
HIGH/LOW  
INPUT/OUTPUT

Conviene usarlas para evitar confusiones.

### Funciones

Una función es un conjunto de instrucciones agrupadas para un nombre, al que le pasaremos unos argumentos y devolverá un valor.

¿Por qué usamos funciones?

* Cuando repetimos un fragmento de código
* Ahorro de espacio
* Estructuración

El formato siempre es:

tipo funcion(tipo argumento 1, tipo argumento 2) {  
 // Codigo  
}

### Librerías

Cuando vamos a reutilizar varias veces un código puede ser buena idea que creemos una librería, que no es más que una colección de código que empaquetamos de una manera concreta y cuyas funciones podemos utilizar sin más que incluirlas en nuestro código. En el ejemplo se incluye la librería serie que se usa para comunicar el pc con Arduino.

#include <Serial.h>

Existen muchas librerías cuyo uso iremos viendo a lo largo del curso

### Estructuras de control

Son aquellas estructuras que nos permiten modificar el flujo de ejecución de nuestro programa, es decir el orden en el que se ejecutan las sentencias.

## Sentencias condicionales

Permiten decidir si se ejecutan o no, unas sentencias en función de que se cumplan determinadas condiciones

### Sentencia if

Ejecutaremos un código si se cumple la condición y otro distinto si no se cumple

if (condición){  
// código si se cumple  
}  
else{  
// código si no se cumple  
}

Ejemplo

if ((valorEntrada < 500) && (valorEntrada>100)){  
 // código A  
}  
else{  
 // código B  
}

### Sentencia switch

Dependiendo del valor de una variable ejecutaremos un código distinto

switch (var) {  
 case 1:  
 //hacemos algo si var es 1  
 break;  
 case 2:  
 //hacemos algo si var es 2  
 break;  
 default:  
 // si nada concuerda, default  
 // default es opcional  
 }

### Bucles

Realizamos un bucle siempre que queremos que se repita un determinado fragmento de código. Todo bucle tiene una condición que determinará las veces que se repetirá el código.

Existen 3 tipos de bucles:

### Bucle for

Es el más natural para usar cuando la iteración tiene un número claro de repeticiones:

for (inicializacion; condicion; incremento) {  
 //sentencia(s);  
}

por ejemplo

for (int i=0;i\<20;i=i+1) {  
 //sentencia(s);  
}

que se realizará 20 veces

### Bucle while

El bucle while ejecutará las sentencias de su bloque mientras que la condición inicial se cumpla

while(expresion){  
 // sentencia(s);  
}

Si la condición no se cumple inicialmente no se realizará ninguna iteración.

### Bucle do

En este bucle la comprobación se hace después de la iteración, por lo que tenemos garantizado que al menos se ejecutará una vez la iteración

do {  
//sentencia(s)  
} while (condicion);

### ¿Como salir de los bucles?

De un bucle saldremos cuando no se verifique la condición, pero también podemos formar la salida usando las siguientes instrucciones

* [break](http://Arduino.cc/en/Reference/Break) // sale del bucle
* [continue](http://Arduino.cc/en/Reference/Continue) // salta el paso actual del bucle
* [return](http://Arduino.cc/en/Reference/Return) // sale de la función
* [goto](http://Arduino.cc/en/Reference/Goto) label //salta a la etiqueta *label*
* for(int i=0;i\<10;i++){  
   if (bsalto\>0)  
   continue; // Se salta el código del resto de la iteración  
   else  
   break; //sale del bucle  
  }

### String: o cadenas de caracteres

Para facilitar el trabajo con cadenas de caracteres, existe la clase [String](http://Arduino.cc/en/Reference/StringObject). Tiene (entre otros muchos) los siguientes métodos:

* [compareTo](http://Arduino.cc/en/Reference/StringCompareTo)() //compara dos cadenas
* [concat](http://Arduino.cc/en/Reference/StringConcat)() // concatena
* [endsWith](http://Arduino.cc/en/Reference/StringEndsWith)() // comprueba si termina por...
* [equals](http://Arduino.cc/en/Reference/StringEquals)() //comprueba si son iguales
* [equalsIgnoreCase](http://Arduino.cc/en/Reference/StringEqualsIgnoreCase)() // comprueba igualdad ignorando mayúsculas y minúsculas
* [indexOf](http://Arduino.cc/en/Reference/StringIndexOf)() // devuelve la primera posición del carácter que le pasamos
* [lastIndexOf](http://Arduino.cc/en/Reference/StringLastIndexOf)() // devuelve la última posición del carácter que le pasamos
* [length](http://Arduino.cc/en/Reference/StringLength)() // longitud de la cadena
* [replace](http://Arduino.cc/en/Reference/StringReplace)() // reemplaza
* [startsWith](http://Arduino.cc/en/Reference/StringStartsWith)() // comprueba si empieza por
* [substring](http://Arduino.cc/en/Reference/StringSubstring)() // devuelve un fragmento de la cadena original
* [toLowerCase](http://Arduino.cc/en/Reference/StringToLowerCase)() // convierte a minúsculas
* [toUpperCase](http://Arduino.cc/en/Reference/StringToUpperCase)() // convierte a mayúsculas
* [trim](http://Arduino.cc/en/Reference/StringTrim)() // elimina los espacios iniciales y finales

### Conversiones de tipos

Muchas veces es necesario convertir valores de variables de tipos diferentes. Para eso existen en Arduino un conjunto de funciones que nos permiten hacer estas transformaciones con garantías:

char() // Convierte a un carácterer  
byte() // Convierte a un byte  
int() // Convierte a un int  
word() // Convierte a un word  
long() // Convierte a un long  
float() // Convierte a un número decimal

Por ejemplo, veamos como convierte a entero. Obviamente perdemos la parte decimal.

float a=2.4;  
int b=int(a);

### Funciones matemáticas

Aunque el procesador de Arduino no excesivamente potente en lo que a cálculo matemático se refiere, existen librerías que nos permiten hacer la mayoría de las funciones matemáticas usuales:  
### Matemáticas

* [min](http://Arduino.cc/en/Reference/Min)(a,b) // mínimo de a y b
* [max](http://Arduino.cc/en/Reference/Max)(a,b) // máximo de a y b
* [abs](http://Arduino.cc/en/Reference/Abs)(a) // valor absoluto de a
* [pow](http://Arduino.cc/en/Reference/Pow)(a,b) // devuelve a elevado a b
* [sqrt](http://Arduino.cc/en/Reference/Sqrt)(a) // devuelve la raiz cuadrada de a

### Trigonometría

* [sin](http://Arduino.cc/en/Reference/Sin)(a) // calcula el seno de a
* [cos](http://Arduino.cc/en/Reference/Cos)(a) // calcula el coseno de a
* [tan](http://Arduino.cc/en/Reference/Tan)(a) // calcula la tangente de a

### Números aleatorios

* [randomSeed](http://Arduino.cc/en/Reference/RandomSeed)(a) // inicializa los número aleatorios con a
* [random](http://Arduino.cc/en/Reference/Random)() // devuelve un número aleatorio

Obviamente, dada la poca capacidad de cálculo de Arduino, el usar estas funciones matemáticas complejas es muy costoso computacionalmente, por lo que se debería de evitar u optimizar su uso.

### Introducción a la programación de objetos

Un objeto es un conjunto de funciones (métodos) y variables (atributos) que trataremos de una forma global.

El trabajar con objetos nos permite encapsular (esconder las interioridades), mostrando un interface (una vista) con lo que permitimos que otros vean (atributos públicos) y usen (métodos públicos) de nosotros.

A los que los usamos nos facilitan el trabajo al agrupar las funcionalidades.

Por ejemplo el objeto Serial, permite trabajar con todo lo relacionado con el manejo de los puertos serie:

* Serial.print enviará por el puerto serie
* Serial.read leerá por el puerto serie
* Serial.write escribirá...

La mayoría de las librerías de Arduino están formadas por objetos.

### ¿Donde encontrar más información?

* [Arduino](http://Arduino.cc/)
* [Arduino programing notebook (inglés)](http://Arduino.cc/playground/uploads/Main/Arduino_notebook_v1-1.pdf)
* [Guía de programación de Arduino (español)](http://books.openlibra.com/pdf/Manual-Programacion-Arduino.pdf)
* [Freeduino page](http://www.freeduino.org/)

## Funciones relacionadas con el tiempo

### Esperas ...

Vamos a repasar las funciones relacionadas con la medida del tiempo

delay(espera); // espera expresado en milisegundos

delayMicroseconds(espera); // espera expresado en microsegundos // El número máximo que soporta es 16383. Si se necesita esperar más tiempo usamos delay

**Ventaja**: son muy fáciles de usar

**Inconveniente**: no nos permiten hacer nada mientras esperamos

Por esto vamos a aprender a medir el tiempo y así no tener que esperar

### Medida del tiempo

Arduino nos permite medir el paso del tiempo con su reloj interno, pero no dispone de un reloj "real" que nos permita saber la fecha y hora con precisión

long millis() : número de milisegundos desde que se encendió la placa. Al usar un tipo long, su valor vuelve a cero cada 50 días aproximadamente.

Lo usaremos así:

long ini = millis(); //hacemos algo long millis\_pasados = millis()-ini; // milisegundos que han pasado long segundos = millis\_pasados / 1000; // Segundos que han pasado

### Parpadeo sin Esperas

Vamos a usar lo visto para hacer un programa que produzca un parpadeo pero sin usar delay y por tanto si hacer esperas (y mientras puedes por ejemplo vigilar el estado de un botón)

El algoritmo será:

* Anoto el tiempo actual en que enciendo el led
* Hago las otras tareas
* Vuelvo a medir el tiempo y comparo con el anterior
* Si ha pasado más del intervalo apago el led y anoto el tiempo actual
* Si no ha pasado sigo haciendo otras tareas

El código quedaría así:

#define PIN\_LED 13  
#define INTERVALO 1000 // intervalo de parpadeo  
  
int estadoLed = LOW; // Ultimo estado del led  
long anteriorMillis = 0; // almacena el momento del ultimo cambio de estado  
  
void setup() {  
 pinMode(PIN\_LED, OUTPUT);   
}  
  
void loop() {  
 unsigned long actualMillis = millis(); // Guardamos el tiempo actual  
 if(actualMillis - anteriorMillis > INTERVALO) {  
 anteriorMillis = actualMillis;   
 if (estadoLed == LOW) { // Invertimos el estado estado = !estado  
 estadoLed = HIGH;  
 } else {  
 estadoLed = LOW;  
 }  
 digitalWrite(PIN\_LED, estadoLed); // Ponemos el nuevo estado  
 }  
}

### Librería Time

Para usar el formato de horas y minutos podemos usar la librería Time que te permitirá usar arduino como un reloj (con poca precisión eso sí)

Un ejemplo sencillo de su uso sería:

#include <Time.h>  
  
void setup()  
 { Serial.begin(9600);  
 setTime(10,23,00,14,3,2018); // tienes que poner la hora actual!!  
 }  
void loop()  
 { time\_t t = now();  
  
 Serial.print(day(t));  
 Serial.print(+ "/") ;  
 Serial.print(month(t));  
 Serial.print(+ "/") ;  
 Serial.print(year(t));  
 Serial.print( " ") ;  
 Serial.print(hour(t));   
 Serial.print(+ ":") ;  
 Serial.print(minute(t));  
 Serial.print(":") ;  
 Serial.println(second(t));  
 delay(1000);  
 }

## Proceso de compilación en C++ y en Arduino

Es importante saber cómo es el proceso de compilación de nuestro código para así poder entender el funcionamiento de algunas cosas.

### Fases del proceso de compilación

Se realiza en 3 fases y es muy importante el orden

#### Preprocesado

* Includes: se includen todos los ficheros (de manera recursiva, el include que include un fichero...)
* Defines: se sustituyen las etiquetas por su valor
* Directivas de preprocesado: se validan y se decide qué código se usa y cual no

### Compilado

* Se valida la sintaxis, se revisan la declaración de todas las variables, se validan los tipos, expresiones, etc...
* Se genera un código objeto

### Enlazado ("Linkado")

* Se juntan todos los códigos objetos (de los diferentes ficheros) y de las librerías usadas
* Se verifica que no falten símbolos
* Se genera el ejecutable

En este vídeo explicamos el proceso con un [Ejemplo](./Preprocesador/Preprocesador.ino)

#define verde 7  
#define rojo 8  
  
#define enciende(x) digitalWrite(x,HIGH)  
#define apaga(x) digitalWrite(x,LOW)  
  
  
#define \_DEBUG 1  
  
void setup() {  
 // put your setup code here, to run once:  
  
 pinMode(verde,OUTPUT);  
 pinMode(rojo,OUTPUT);  
  
  
#ifdef \_DEBUG  
 Serial.begin(9600);  
#endif  
  
}  
  
void loop() {  
 // put your main code here, to run repeatedly:  
#ifdef \_DEBUG  
 Serial.println("Verde");  
#endif  
 enciende(verde); // digitalWrite(verde,HIGH);  
 apaga(verde); // digitalWrite(verde,LOW);  
#ifdef \_DEBUG  
 Serial.println("Rojo");  
#endif  
 enciende(rojo); // digitalWrite(rojo,HIGH);  
 apaga(rojo); // digitalWrite(rojo,LOW);  
  
}

[Vídeo](https://youtu.be/ODG07Y9SIHc)



1

## Interrupciones

Las interrupciones son un recurso de los modernos procesadores que permiten detener la ejecución del programa principal para atender un evento (suceso) determinado, realizar algunas acciones y luego retomar la ejecución del programa principal por el punto en el que se detuvo.

De esta manera podemos hacer nuestro programa más simple, programando de forma independiente las reacciones a determinados eventos.

Para ellos crearemos una función que se encargará de realizar el procesamiento de este evento y que tendrá el siguiente formato:

void funcionInterrupcion(){ // Código de procesamiento de la interrupcion }

El código de estas funciones debe ser lo más breve posible ya que mientras el se procesa una interrupción no se atienden otras tareas que el micro debe hacer (como controlar las señales pwm, de servos, medir el tiempo, etc) Por tanto dentro del código de estas funciones está desaconsejado usar métodos como delay y los relacionados con el envío de datos vía Serial

Estos eventos pueden estar originados por el hardware (cambios en el estado de algún pin) o por un temporizador (parecido a una alarma) que llamaremos de tipo Software

Más adelante hablaremos de las interrupciones hardware, aquellas que se producen cuando el estado de alguna de las patillas cambia.

## Interrupciones software

Las interupciones software se producen por tiempo, es parecido a cuando programamos una alarma en un despertador: \* Configuramos un temporizador (timer en lenguaje técnico) con un tiempo determinado \* Activamos el timer \* Asignamos una función que será llamada cuando llegue ese momento

¿Qué es un timer? : es un hardware independiente al microprocesador que va contando el tiempo de forma paralela al microprocesador

Todos los timers dependen del system clock de Arduino. Normalmente es 16MHz, salvo Arduino Pro 3,3V con 8Mhz. El ritmo al que cambian estos contadores viene dado por un registro del micro que se llama [prescaler](https://playground.arduino.cc/Code/Prescaler). Por ejemplo, un prescaler de 1000, hará que cada 1000 ciclos del microprocesador, se incremente e1 valor del temporizador.

Los Timers son usados internamente por el core de Arduino para sus funciones internas, como pueden ser millis, delay, analogWrite,..

El microcontrolador de las placas Arduino UNO dispone de 3 módulos Contador/Timers, dos de 8 bits y uno de 16 bits. El tamaño en bits nos da idea de hasta que valores pueden contar.

Las placas de Arduino Mega tienen además otros timer: timer3, timer4 y timer5 todos de 16bit

* Timer0: Timer0 es un timer de 8bit. Arduino UNO lo usa para delay(), millis() y micros().
* Timer1: Timer1 es un timer de 16bit. Arduino UNO lo usa para controlar los Servos (timer5 en Arduino Mega).
* Timer2: Timer2 is a 8bit timer like timer0. Se usa para tone()

Las interrupciones se pueden activar o desactivar, usando las llamadas [**interrupts**](https://www.arduino.cc/reference/en/language/functions/interrupts/interrupts/) y [**noInterrupts**](http://arduino.cc/en/Reference/NoInterrupts).

Para configurar un temporizador con tiempo determinado son muchos los registros que hay que configurar y el cálculo de sus valores no es algo sencillo. Podemos leerlo en detalle en el [siguiente enlace](http://arduino.cc/es/Tutorial/SecretsOfArduinoPWM)

Afortunadamente existen librerías que nos facilitan enormemente la tarea.

## Librería TimerOne

Es una librería que nos permite trabajar con interrupciones software de maner muy sencilla usando el Timer1

La instalamos desde el gestor de librería y podemos usar directamente sus ejemplos

La libería nos permite hacer varias cosas (a coste de perder el PWM de los pines 9 y 10)

void initialize(long microseconds=1000000); // Inicializamos el periodo de disparo en microsegundos  
void start(); // Arrancamos el temporizador  
void stop(); // Para el temporizador  
void restart(); // Lo rearranca por donde iba  
unsigned long read(); // tiempo hasta disparo  
void setPeriod(long microseconds); // cambia el periodo  
void pwm(char pin, int duty); // para usar pwm en pines 9 y 10  
void setPwmDuty(char pin, int duty);  
void disablePwm(char pin); // desactiva pwm en 9 y 10  
void attachInterrupt(funcionDisparo); // establece la función de disparo  
void detachInterrupt(); // elilmina la función de disparo

Veamos un ejemplo de uso en el que haremos que un led cambie de estado cada segundos

#include

#define PIN\_LED 13

void setup() {

pinMode(PIN\_LED , OUTPUT);  
  
Timer1.initialize(1000000); // 500000 microseconds o 0.5 segundo   
Timer1.attachInterrupt( funcionDisparo ); // establecemos la funcion de disparo  
Timer1.start(); // Arancamos la interrupción

void loop() { // Mientras podemos hacer en el loop otras cosas... }

void funcionDisparo() {

if(digitalRead(PIN\_LED) == HIGH ){ // Invertimos el estado del led  
 digitalWrite( PIN\_LED, LOW);  
} else {  
 digitalWrite( PIN\_LED, HIGH );  
}

}

Ahora puedes revisar el ejemplo "Interrupt" de la librería

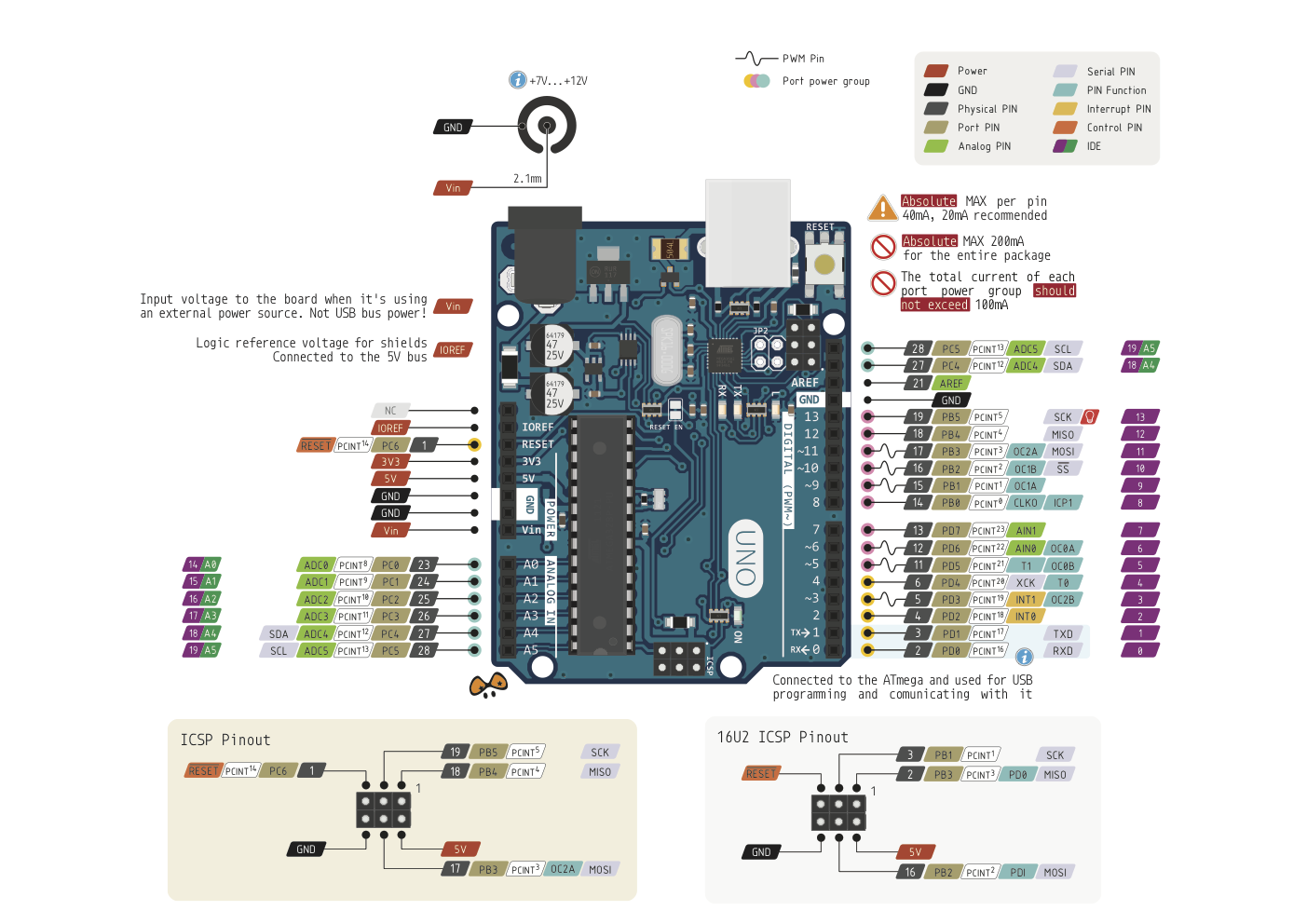
[Vídeo sobre interrupciones software](https://www.youtube.com/embed/oG73lT56m7A)

## Uso avanzado de puertos

Hasta ahora todo el acceso a los puertos que hemos hecho ha sido usando las instrucciones digitalWrite o digitalRead. Con ella sólo podíamos acceder a un pin al mismo tiempo, es decir para hacer varias acceso a diferentes pines teníamos que hacer diferentes llamadas.

Son muchas las ocasiones en las que es necesario el acceder simultáneamente a varios pines como por ejemplo cuando estamos usando muchos leds. O bien cuando accedemos a los bits y podríamos realizar operaciones matemáticas de una forma sencilla para obtener los resultados.

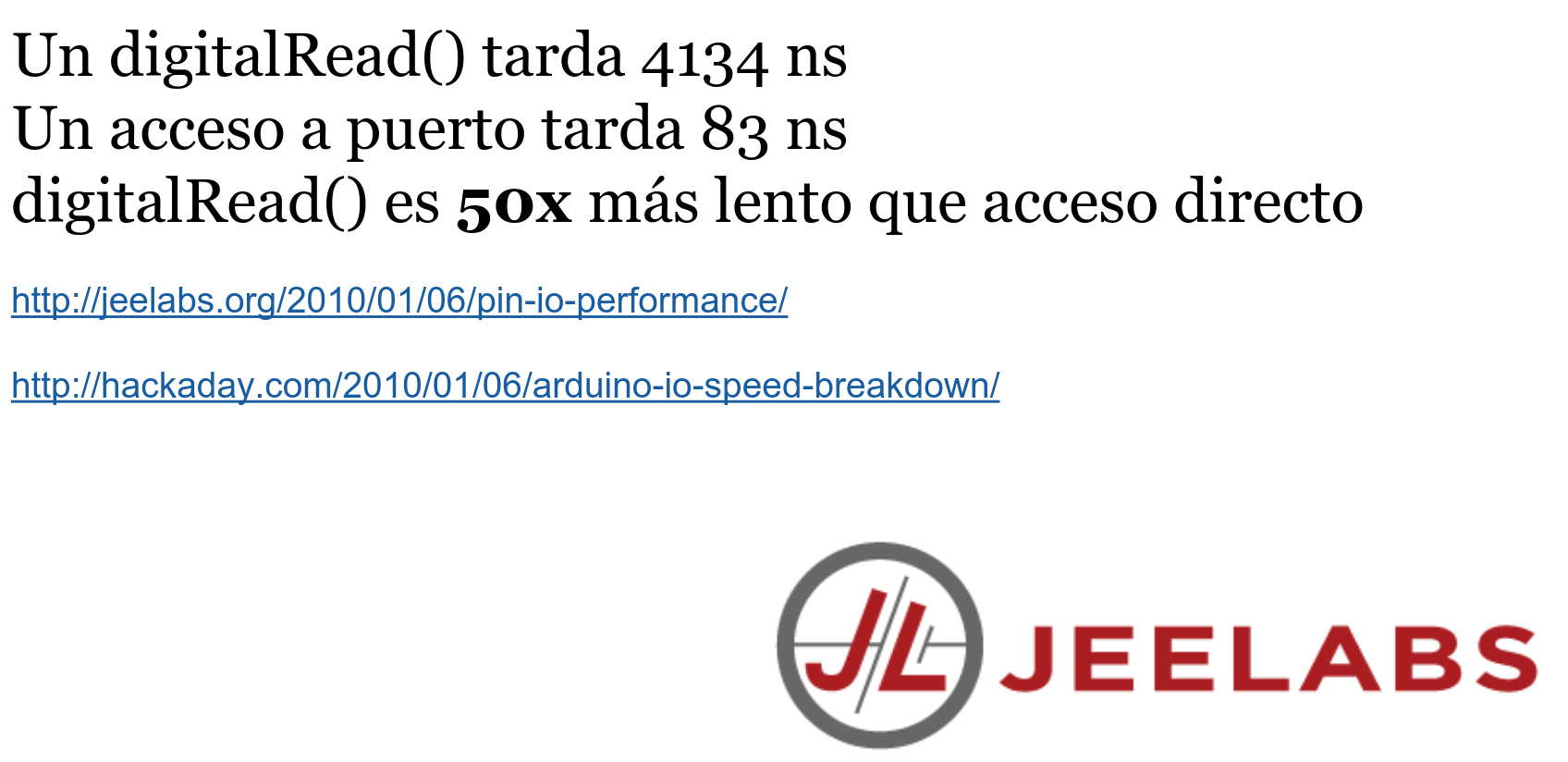
Por otro lado, internamente todos los pines de los microcontroladores están organizados en diferentes palabras o registros.



Por ejemplo, en Arduino UNO las E/S en los microcontroladores están agrupadas en puertos \* B (digital pin 8 to 13) \* C (analog input pins) \* D (digital pins 0 to 7)

Cada puerto está controlado por 3 registros donde cada bit representa a un pin \* DDR que define si el pin es INPUT o OUTPUT \* PORT que define el estado en caso de que sea pin de salida HIGH o LOW \* PIN que define el estado en caso de que sea entrada INPUT

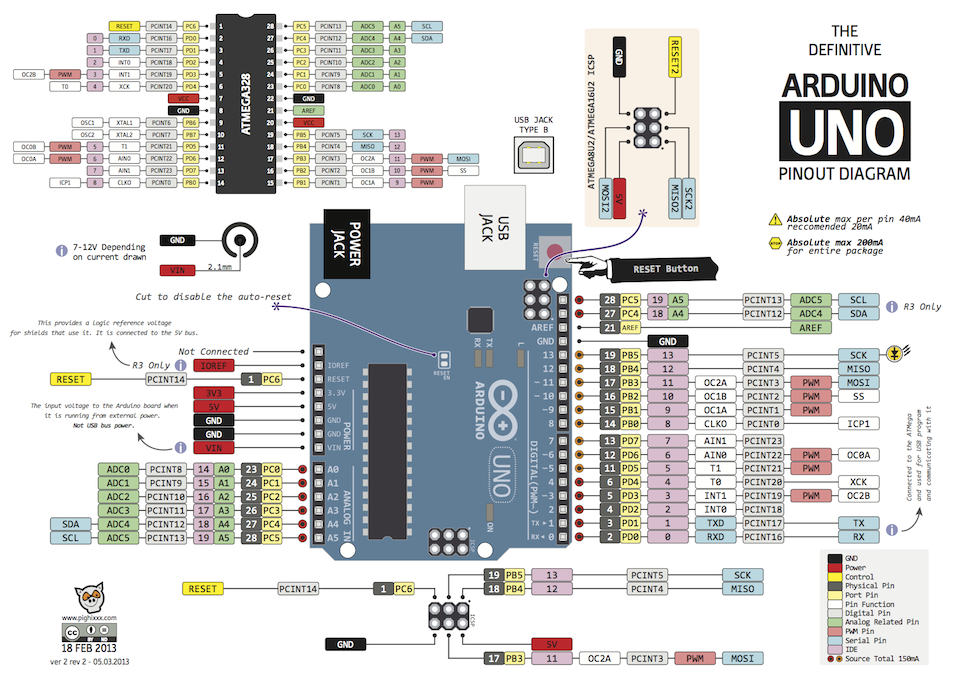
Otra ventaja que tiene acceder de esta manera que es que el acceso es mucho más rápido.



TiemposPuertos.png

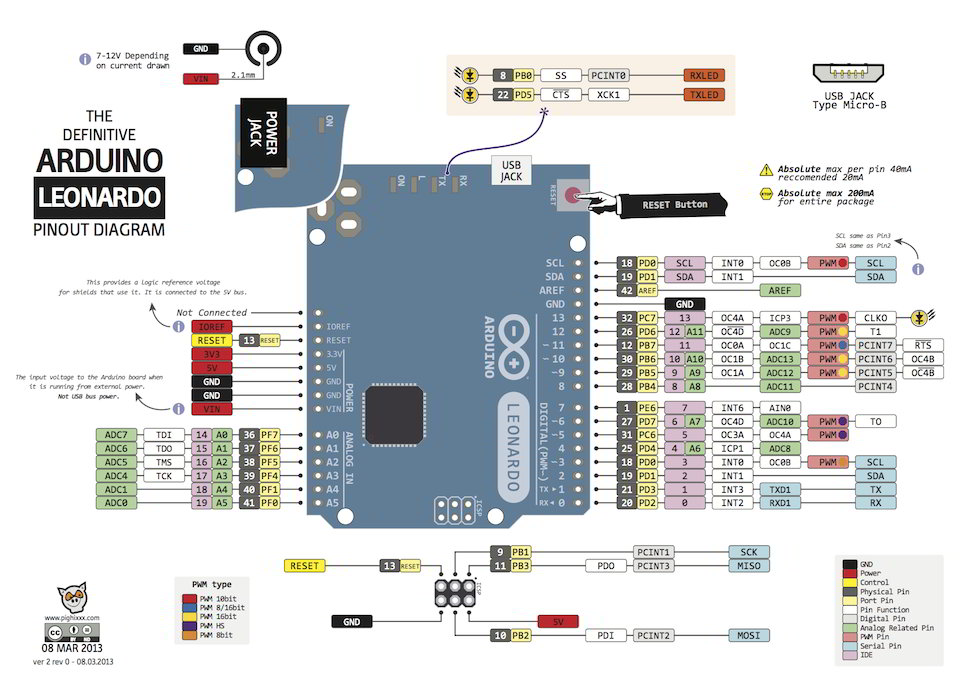
Se propone realizar como práctica: midiendo la velocidad de acceso directo a puertos y de digitalwrite

Veamos la estructura de los pines de un Arduino UNO



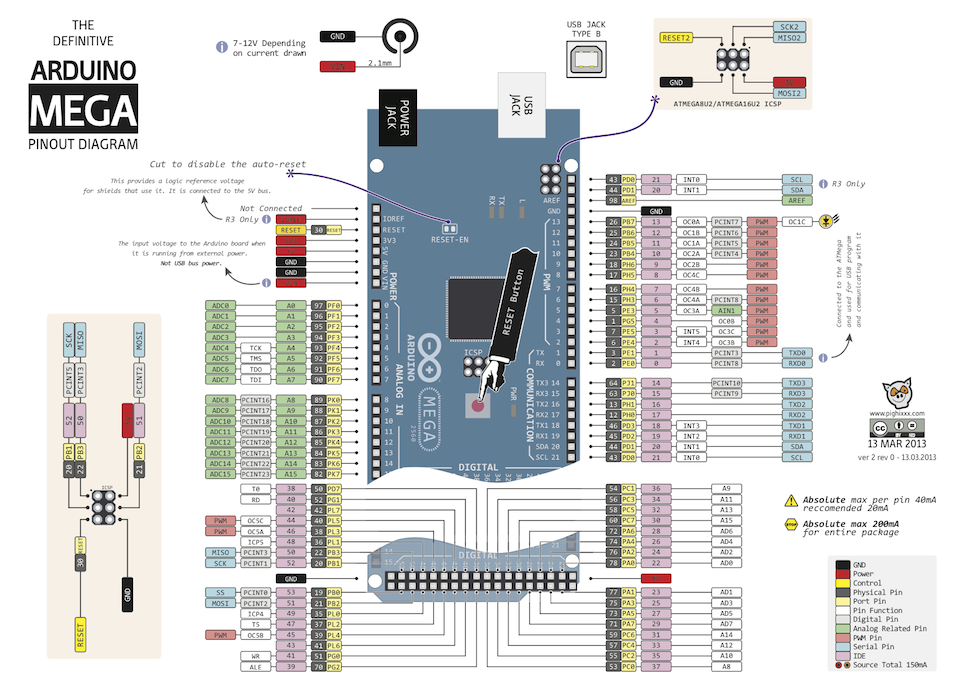
Pinout Arduino UNO

Para un arduino Leonardo el diagrama es parecido aunque con diferencias, como por ejemplo los pines SDA y SCL y sobre todo la disposición e los pines de puerto agrupados



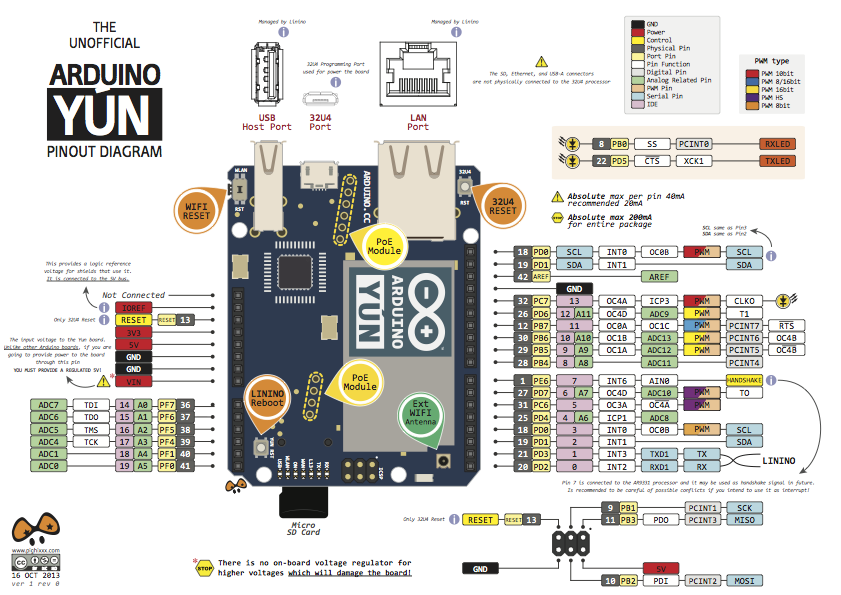
Pinout\_Leonardo.png

Y no podía faltar el esquema para el Arduino Mega



Pinout\_Mega.png

Arduino Yun



Pinout\_yun.png

(Esquemas del gran Alberto Piganti @pighixxx)

En el [siguiente vídeo](https://www.youtube.com/embed/xkSOt-tlCmM) vamos a ver cómo usar los pines por medio de los registros

### Más detalles

* PORTD Arduino digital pins 0 to 7
* DDRD - Port D Data Direction Register - read/write
* PORTD - Port D Data Register - read/write
* PIND - Port D Input Pins Register - read only
* PORTB Arduino digital pins 8 to 13
* DDRB - Port B Data Direction Register - read/write
* PORTB - Port B Data Register - read/write
* PINB - Port B Input Pins Register - read only
* PORTC Arduino analog pins 0 to 5. Pins 6 & 7 sólo en Arduino Mini
* DDRC - Port C Data Direction Register - read/write
* PORTC - Port C Data Register - read/write
* PINC - Port C Input Pins Register - read only

[Referencia](http://www.arduino.cc/en/Reference/PortManipulation)

### Ejemplo sencillo

Hemos dicho que PortD mapea los digital pins 0 to 7 (pins 0 & 1 son TX y RX)

DDRD es la dirección del Port D (Arduino digital pins 0-7) controla si PORTD se configura como entrada o salidas

Si en nuestro código hacemos

DDRD = B11111110; // 1 to 7 as outputs, pin 0 as input

Estaremos haciendo que los pines 1-7 sean salidas y el 0 entrada

Si ahora hacemos

PORTD = B10101000; // ponemos los pin 7,5,3 HIGH

Estamos poniendo los pines 7,5 y 3 en estado alto

Hay que tener cuidado con los pines 0 y 1 puesto que si los manipulamos con el registro PORTD podemos producir problemas de comunicación puesto que son los pines que se encargan de ello

Vamos a hacer ahora el tipico ejemplo del "coche fantástico o Kit", donde vamos desplazando el led encendido entre los 5 disponibles

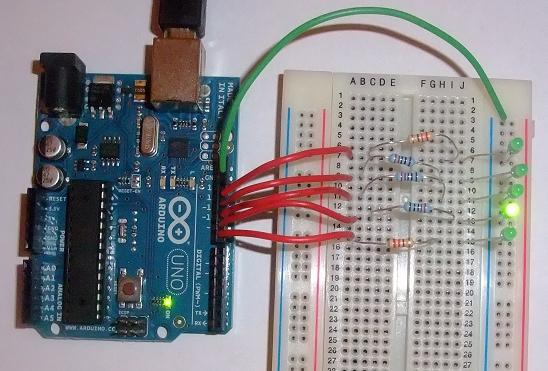
Antes vamos a un truco matemático que se usa mucho. El operador << nos permite rotar hacia la derecha el valor de un registro.

Si tenemos el valor 1 (B00000001 en binario) y lo rotamos 1 vez tendremos B00000010 (que en decimal equivale al valor 2)

Si queremos hacer que nuestro led "se deplace", sólo tenemos que rotar el valor 1 un cierto número de veces.

El resultado de este operador es multiplicar por 2 el valor orignal

Del mismo modo existe el operador >> que rota en sentido contrario y que equivale a dividir por 2.



LED\_kit\_paralelo.png

El código sería

unsigned char upDown=1; // Indica si vamos hacia arriba o hacia abajo  
  
unsigned char cylon=0; // será el led que encendemos de 0 a 4  
  
void setup() {  
 DDRB = B00011111; // Arduino port B pines 0 to 4 como salida  
}  
  
void loop() {  
 if ( upDown == 1 ) { // Vamos hacia arriba  
 cylon++; // Pasamos al siguiente  
 if(cylon>=4) { // Si llegamos a 4 tenemos que empezar a ir hacia abajo  
 upDown=0;  
 }  
 } else {  
 cylon--; // Vamos hacia abajo  
 if ( cylon == 0 ) { // Si llevamos abajo (0) empezamos a subir  
 upDown=1;  
 }  
  
 }  
 PORTB = 1 << cylon; // Rotamos a la derecha determinado numero de veces el led  
 delay(150); // Esperamos un poquito  
  
}