Control por Computador

Manual de arduino

Jorge Pomares Baeza
Grupo de **I**nnovación **E**ducativa en **A**utomática

Universitat d'Alacant
Universidad de Alicante

Metadata	No editar manualmente esta tabla.
Título	Manual de programación de Arduino
Estado	revision
Author	Jorge Pomares
Organisation	Universitat d'Alacant

MANUAL DE PROGRAMACIÓN DE ARDUINO

En esta manual se describen las características básicas de la placa Arduino Duemilanove y se muestran las principales consideraciones para realizar su programación.

Características generales de la placa

Se trata de una placa open hardware por lo que su diseño es de libre distribución y utilización, que incluso podemos construirnos nosotros mismos (En la Figura 1 se observa el aspecto de la placa). En la siguiente web puede encontrarse mucha información relativa a la placa:

http://arduino.cc/

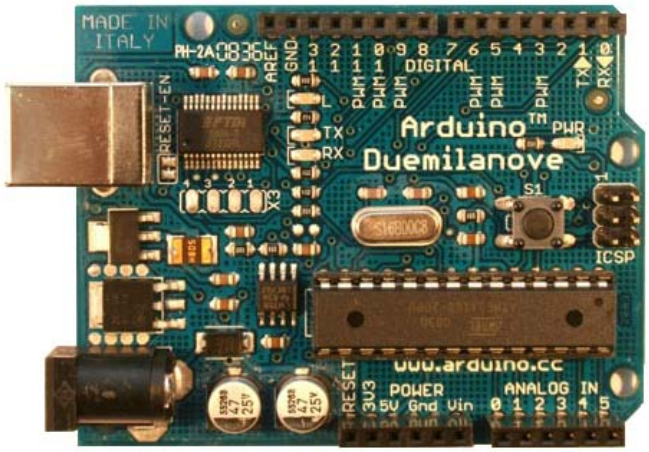

***Figura 1**. Aspecto de la placa Arduino Duemilanove.*

El programa se implementará haciendo uso del entorno de programación propio de arduino y se transferirá empleando un cable USB. Si bien en el caso de la placa USB no es preciso utilizar una fuente de alimentación externa, ya que el propio cable USB la proporciona, para la realización de algunos de los experimentos prácticos sí que será necesario disponer de una fuente de alimentación externa ya que la alimentación proporcionada por el USB puede no ser suficiente. El voltaje de la fuente puede estar entre 6 y 25 Voltios.

Entorno de desarrollo

Para programar la placa es necesario descargarse de la página web de Arduino el entorno de desarrollo (IDE). Se dispone de versiones para Windows y para MAC, así como las fuentes para compilarlas en LINUX. En la Figura 2 se muestra el aspecto del entorno de programación. En el caso de disponer de una placa USB es necesario instalar los drivers FTDI. Estos drivers vienen

incluidos en el paquete de Arduino mencionado anteriormente. Existen en la web versiones para distintos sistemas operativos.

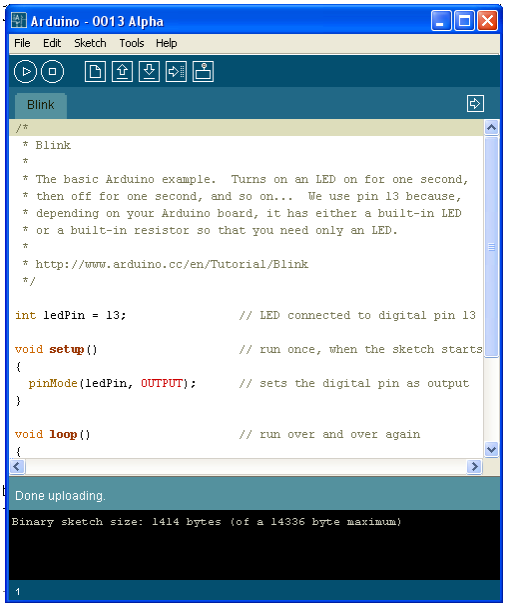

***Figura 2**. Entorno de desarrollo.*

Lo primero que tenemos que hacer para comenzar a trabajar con el entorno de desarrollo de arduino es configurar las comunicaciones entre la placa Arduino y el PC. Para ello deberemos abrir en el menú *"Tools"* la opción *"Serial Port"*. En esta opción deberemos seleccionar el puerto serie al que está conectada nuestra placa. En Windows, si desconocemos el puerto al que está conectado nuestra placa podemos descubrirlo a través del Administrador de dispositivos (Puertos COM & LPT/ USB Serial Port).

El primer paso para comprobar que todo lo que hemos hecho hasta ahora está bien y familiarizarnos con el interfaz de desarrollo, es abrir uno de los ejemplos. Se recomienda abrir el ejemplo "Blink". Para ello debemos acceder a través del menú File → Sketchbook → Examples → Digital → Blink.

El ejemplo "Blink" lo único que hace es parpadear un LED que está colocado en el pin número 13 de la placa. Vamos a ver qué hay que hacer para subir el programa a la placa Arduino. Primero comprobamos que el código fuente es el correcto. Para ello pulsamos el botón de verificación de código que tiene forma de triángulo inclinado 90 grados (Figura 3.a). Si todo va bien deberá aparecer un mensaje en la parte inferior de la interfaz indicando "Done compiling". Una vez que el código ha sido verificado procederemos a cargarlo en la placa. Para ello tenemos que pulsar el botón de reset de la placa (Figura 3.b) e inmediatamente después pulsar el botón que comienza la carga (Figura 3.c).

Durante la carga del programa, en la placa USB, se encenderán los LED que indican que se están enviando y recibiendo información por el puerto serie: TX/RX. Si todo se ha realizado correctamente debe aparecer el mensaje "Done uploading". Ahora tan sólo queda esperar unos 8

segundos aproximadamente para comprobar que todo ha salido bien. Si el led colocado en el pin 13 de la placa se enciende y se apaga cada segundo entonces todo ha ido bien. Por fin tenemos todo listo para empezar a trabajar con la placa Arduino.

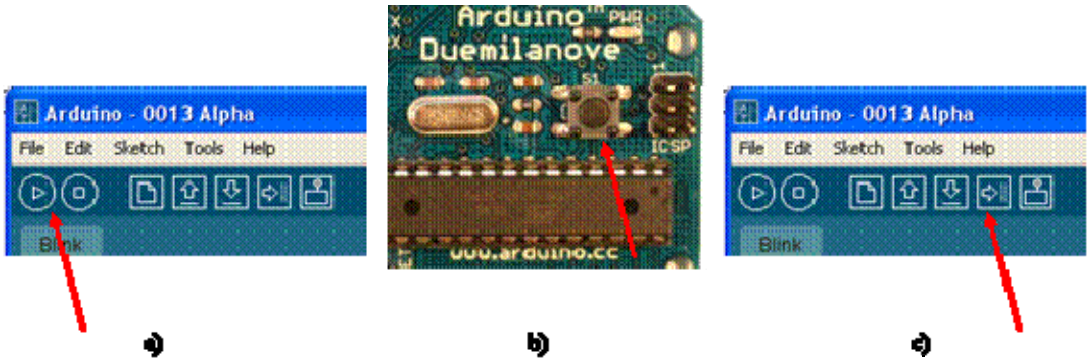

***Figura 3**. a) Compilar programa. b) Botón de reset. c) Transferir programa a la placa.*

Estructura básica de un programa

La estructura básica de programación de Arduino es bastante simple y divide la ejecución en dos partes: setup y loop. Setup() constituye la preparación del programa y loop() es la ejecución. En la función Setup() se incluye la declaración de variables y se trata de la primera función que se ejecuta en el programa. Esta función se ejecuta una única vez y es empleada para configurar el pinMode (p. ej. si un determinado pin digital es de entrada o salida) e inicializar la comunicación serie. La función loop() incluye el código a ser ejecutado continuamente (leyendo las entradas de la placa, salidas, etc.).

```
void setup() {
   inMode(pin, OUTPUT);      // Establece 'pin' como salida
}
void loop() {
   digitalWrite(pin, HIGH);  // Activa 'pin'
   delay(1000);              // Pausa un segundo
   digitalWrite(pin, LOW);   // Desactiva 'pin'
   delay(1000);
}
```

Como se observa en este bloque de código cada instrucción acaba con ; y los comentarios se indican con //. Al igual que en C se pueden introducir bloques de comentarios con /* … */.

Funciones

Una función es un bloque de código identificado por un nombre y que es ejecutado cuando la función es llamada. La declaración de una función incluye en primer lugar el tipo de datos que devuelve la función (e.j. int si lo que devuelve es un valor entero). Después del tipo de datos se especifica el nombre de la función y los parámetros de la misma. La siguiente función es empleada para realizar un retardo en el programa leyendo el valor de un potenciómetro:

```
int delayVal() {
   int v;                    // crea una variable temporal 'v'
   v = analogRead(pot);      // lee el valor del potenciómetro
   v /= 4;                   // convierte los valores 0-1023 a 0-255
```

```
   return v;                  // devuelve el valor final de la variable
}
```

Variables

Una variable debe ser declarada y opcionalmente asignada a un determinado valor. En la declaración de la variable se indica el tipo de datos que almacenará (int, float, long)

```
int inputVariable = 0;
```

Una variable puede ser declarada en el inicio del programa antes de setup(), localmente a una determinada función e incluso dentro de un bloque como pueda ser un bucle. El sitio en el que la variable es declarada determina el ámbito de la misma. Una variable global es aquella que puede ser empleada en cualquier función del programa. Estas variables deben ser declaradas al inicio del programa (antes de la función setup()).

```
int v;                        // 'v' es visible en todo el programa
void setup() {
                              // no se requiere setup
}
void loop() {
   for (int i=0; i<20;)       // 'i' es visible solo en el bucle
   i++;
   float f;        // 'f' es visible únicamente en la función loop()
}
```

Tipos de datos

Arduino permite manejar los siguientes tipos de datos:

- Byte. Almacena un valor numérico de 8 bits. Tienen un rango de 0-255.
- Int. Almacena un valor entero de 16 bits con un rango de 32,767 a -32,768.
- Long. Valor entero almacenado en 32 bits con un rango de 2,147,483,647 a -2,147,483,648.
- Float. Tipo coma flotante almacenado en 32 bits con un rango de 3.4028235E+38 a -3.4028235E+38.
- Arrays Se trata de una colección de valores que pueden ser accedidos con un número de índice (el primer valor del índice es 0). Ejemplos de utilización:
    - Definición y asignación. int myArray[] = {value0, value1, value2...}
    - Definición. int myArray[5]; // declara un array de 6 enteros
    - Asignación del cuarto componente. myArray[3] = 10;
    - Recuperar el cuarto componente y asignarlo a x. x = myArray[3];

Operadores aritméticos

Empleando variables, valores constantes o componentes de un array pueden realizarse operaciones aritméticas y se puede utilizar el operador cast para conversión de tipos. Ej. int a = (int)3.5; Además pueden hacerse las siguientes asignaciones:

x ++. Lo mismo que x = x + 1.
x --. Lo mismo que x = x - 1, or decrements x by -1.
x += y. Lo mismo que x = x + y, or increments x by +y.

x -= y. Lo mismo que x = x - y .
x *= y. Lo mismo que x = x * y.
x /= y. Lo mismo que x = x / y.

Para su utilización en sentencias condicionales u otras funciones Arduino permite utilizar los siguientes operadores de comparación:

x == y. x es igual a y.
x != y. x no es igual a y.
x < y, x > y, x <= y, x >= y.

Y los siguientes operadores lógicos:

Y lógico: if (x > 0 && x < 5). Cierto si las dos expresiones lo son.
O lógico: if (x > 0 || y > 0). Cierto si alguna expresión lo es.
NO lógico: if (!x > 0). Cierto si la expresión es falsa.

El lenguaje de Arduino presenta las siguientes constantes predefinidas:

TRUE / FALSE.
HIGH/LOW. Estas constantes definen los niveles de los pines como HIGH o LOW y son empleados cuando se leen o escriben en las entradas o salidas digitales. HIGH se define como el nivel lógico 1 (ON) o 5 V. LOW es el nivel lógico 0, OFF, o 0 V.
INPUT/OUTPUT. Constantes empleadas con la función pinMode() para definir el tipo de un pin digital usado como entrada INPUT o salida OUTPUT. Ej. pinMode(13, OUTPUT);

Sentencias condicionales

El lenguaje de arduino permite realizar sentencias condicionales if, if… else, for, while, do… while. Su utilización es similar a las funciones correspondientes en C.

Entradas y salidas digitales y analógicas

9.1. Función pinMode(pin, mode)

Función usada en la function setup() para configurar un pin dado para comportarse como INPUT o OUTPUT. Ej. pinMode(pin, OUTPUT); configura el pin número ‘pin’ como de salida. Los pines de Arduino funcionan por defecto como entradas, de forma que no necesitan declararse explícitamente como entradas empleando pinMode().

9.2. Función digitalRead(pin)

Lee el valor desde un pin digital específico. Devuelve un valor HIGH o LOW. El pin puede ser especificado con una variable o una constante (0-13). Ej. v = digitalRead(Pin);

9.3. Funcion digitalWrite(pin, value)

Introduce un nivel alto (HIGH) o bajo (LOW) en el pin digital especificado. De nuevo, el pin puede ser especificado con una variable o una constante 0-13. Ej. digitalWrite(pin, HIGH);

9.4. Función analogRead(pin)

Lee el valor desde el pin analógico especificado con una resolución de 10 bits. Esta función solo funciona en los pines analógicos (0-5). El valor resultante es un entero de 0 a 1023. Los pines analógicos, a diferencia de los digitales no necesitan declararse previamente como INPUT o OUTPUT.

9.5. Función analogWrite(pin, value)

Escribe un valor pseudo-analógico usando modulación por ancho de pulso (PWM) en un pin de salida marcado como PWM. Esta función está activa para los pines 3, 5, 6, 9, 10, 11. Ej analogWrite(pin, v); // escribe 'v' en el 'pin' analógico. Puede especificarse un valor de 0 – 255. Un valor 0 genera 0 V en el pin especificado y 255 genera 5 V. Para valores de 0 a 255, el pin alterna rápidamente entre 0 V y 5 V, cuanto mayor sea el valor, más a menudo el pin se encuentra en HIGH (5 V). Por ejemplo, un valor de 64 será 0 V tres cuartas partes del tiempo y 5 V una cuarta parte. Un valor de 128 será 0 V la mitad del tiempo y 5 V la otra mitad. Un valor de 192 será 0 V una cuarta parte del tiempo y 5 V tres cuartas partes.

Funciones de tiempo y matemáticas

- delay(ms). Realiza una pausa en el programa la cantidad de tiempo en milisegundos especificada en el parámetro (máximo 1000, mínimo 1).

- millis(). Devuelve la cantidad de milisegundos que lleva la placa Arduino ejecutando el programa actual como un valor long unsigned. Después de de 9 horas el contador vuelve a 0.

- min(x,y). max(x,y). Devuelve el mínimo y el máximo respectivamente de entre sus parámetros.

Funciones de generación aleatoria

- randomSeed(seed). Especifica un valor o semilla como el punto de inicio para la función random(). Este parámetro debe ser realmente aleatorio y para ello puede emplearse la función millis() o incluso analogRead() para leer ruido eléctrico desde una entrada analógica.

- random(max), random(min, max). Esta función devuelve un valor aleatorio entre el rango especificado.

Puerto serie

- Serial.begin(rate). Abre un Puerto serie y especifica la velocidad de transmisión. La velocidad típica para comunicación con el ordenador es de 9600 aunque se pueden soportar otras velocidades.

- Serial.println(data). Imprime datos al puerto serie seguido por un retorno de línea automático. Este comando tiene la misma forma que Serial.print() pero este último sin el salto de línea al final. Este comando puede emplearse para realizar la depuración de programas. Para ello puede mandarse mensajes de depuración y valores de variables por el puerto serie. Posteriormente, desde el entorno de programación de Arduino, activando el "Serial Monitor" se puede observar el contenido del puerto serie, y, por lo tanto, los mensajes de depuración. Para observar correctamente el contenido del puerto serie se debe tener en cuenta que el "Serial Monitor" y el puerto serie han de estar configurados a la misma velocidad (Para configurar la velocidad del puerto serie se hará con el comando Serial.begin(rate)).

- Serial.read().Lee o captura un byte (un caracter) desde el puerto serie. Devuelve -1 si no hay ningún carácter en el puerto serie.

- Serial.available(). Devuelve el número de caracteres disponibles para leer desde el puerto serie.

Ejemplos de código

Salida digital

En este ejemplo el LED conectado al pin 13 parpadea cada segundo.

```
int ledPin = 13;            // LED que se encuentra en el pin 13
   void setup(){
   pinMode(ledPin, OUTPUT); // El p1n 13 será una salida digital
}
void loop(){
   digitalWrite(ledPin, HIGH);     // Enciende el LED
   delay(1000);                    // Pausa de 1 segundo
   digitalWrite(ledPin, LOW);      // Apaga el LED
   delay(1000);                    // Pausa de 1 segundo
}
```

Salida digital II

En este ejemplo el LED conectado al pin 13 parpadea en un intervalo de tiempo variable que depende del número de veces que se ejecuta el programa (función *loop*)

```
int ledPin = 13;   // LED que se encuentra en el pin 13
int n = 0;         //Entero que contará el paso por la función loop
void setup(){
   pinMode(ledPin, OUTPUT); // El p1n 13 será una salida digital
}
void loop(){
   digitalWrite(ledPin, HIGH);     // Enciende el LED
   delay(1000);                    // Pausa de 1 segundo
   digitalWrite(ledPin, LOW);      // Apaga el LED
   n++;                            //Incrementamos n
   delay(delayVal(n));             //Pausa de un tiempo variable
}
//Función que devuelve un valor tipo entero según el parámetro pasado

int delayVal(int f){
   return f*100;
}
```

Entrada digital

Este ejemplo lee el valor de un interruptor conectado en el pin 2. Cuando el interruptor está cerrado en el pin de entrada habrá un estado alto (HIGH) y se encenderá el LED.

```
int ledPin = 13;   // Pin de salida para el LED
int inPin = 2; // Pin de entrada (donde está conectado el interruptor)
void setup() {
   pinMode(ledPin, OUTPUT);
   pinMode(inPin, INPUT);
}
void loop() {
   if (digitalRead(inPin) == HIGH){// Si se activa interruptor
      digitalWrite(ledPin, HIGH);  // Enciende el LED
```

```
      delay(1000);                  // Pause de 1 segundo
      digitalWrite(ledPin, LOW);    // Apaga el LED
      delay(1000);                  // Pausa de 1 segundo
   }
}
```

Salida PWM

Modulación por ancho de pulso (PWM) puede emplearse, por ejemplo, para establecer el brillo de un led o controlar un servomotor. En el siguiente ejemplo se va aumentando y decrementando el brillo del pin 9 mediante PWM.

```
int ledPin = 9;                     // Pin controlado por PWM
void setup(){}
void loop() {
   for (int i=0; i<=255; i++){
      analogWrite(ledPin, i);       // Establece el brillo a i
      delay(100);                   // Pausa de 100 ms
   }
   for (int i=255; i>=0; i--) {
      analogWrite(ledPin, i);
      delay(100);
   }
}
```

Entrada a partir de un potenciómetro

En el siguiente código se emplea arduino para controlar la frecuencia de parpadeo de un LED.

```
int potPin = 0;             // Pin de entrada para el potenciómetro
int ledPin = 13;            // Pin de salida para el LED
void setup() {
   pinMode(ledPin, OUTPUT); // Declara el pin del LED como de salida
}
void loop() {
   digitalWrite(ledPin, HIGH);      // Enciende el LED
   delay(analogRead(potPin));       // Lee el valor del potenciómetro
   digitalWrite(ledPin, LOW);       // Apaga el LED
   delay(analogRead(potPin));
}
```