

Taller de Desarrollo de Sistemas Domóticos Basados en Arduino

Unidad 2. Programación de Arduino

José L. Poza Luján
Sergio Sáez Barona

Escuela Técnica de Ingeniería Informática 



Unidad 2 Programación de Arduino

Objetivos

Contenido
Introducción
Hardware
Programación
Funciones
Interrupciones
Librerías
Conclusiones



2

Taller de Desarrollo de Sistemas Domóticos Basados en Arduino

 José L. Poza Luján
Sergio Sáez Barona



Objetivos

- Conocer los detalles de conexiones de la placa Arduino Uno para poder realizar los montajes de aprendizaje
- Tomar conciencia de las precauciones a tener en cuenta cuando se trabaja con la placa Arduino Uno
- Aprender las particularidades del lenguaje de programación de Arduino, especialmente los tipos de datos y las instrucciones del lenguaje
- Aprender a manejar las funciones básicas de trabajo de Arduino


Unidad 2
Programación de Arduino
 Objetivos
Contenido
 Introducción
 Hardware
 Programación
 Funciones
 Interrupciones
 Librerías
 Conclusiones
 3
 Taller de Desarrollo de
 Sistemas Domóticos
 Basados en Arduino
 José L. Poza Luján
 Sergio Sáez Barona

Contenido

- Introducción
- Hardware
 - Arduino uno
 - Alimentación
 - Conexión
 - Peligros y precauciones
- Programación
 - Introducción
 - Sintaxis
 - Variables
 - Tipos de datos
 - Constantes
 - Constantes definidas
 - Operadores
 - Estructuras de control
 - Bucles
 - Flujo de programa
- Funciones
 - Definición
 - Tipos de datos
 - Bucle de control
 - Entrada-Salida digital
 - Entrada-Salida analógica
 - Cálculo
 - Tiempo
 - Números aleatorios
 - Manejo de bits
- Interrupciones
- Librerías
 - Inclusión
 - Comunicación serie
- Conclusiones

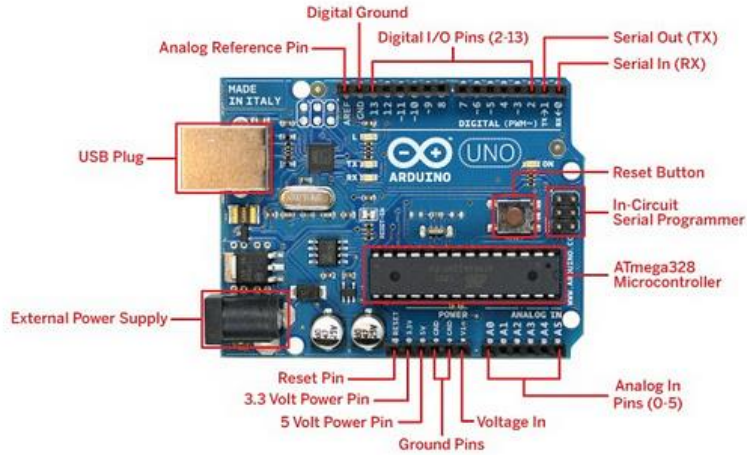

Unidad 2
Programación de Arduino
 Objetivos
Contenido
Introducción
 Hardware
 Programación
 Funciones
 Interrupciones
 Librerías
 Conclusiones
 4
 Taller de Desarrollo de
 Sistemas Domóticos
 Basados en Arduino
 José L. Poza Luján
 Sergio Sáez Barona

Introducción

- Sin hardware no hay software
 - Por tanto, es importante conocer el hardware sobre el que actuará el software
- Sistema basado en Arduino
 - Hardware
 - Placa Arduino: pins
 - Sensores y actuadores
 - Conectores
 - Software
 - Programación: lenguaje
 - Comunicación: protocolo

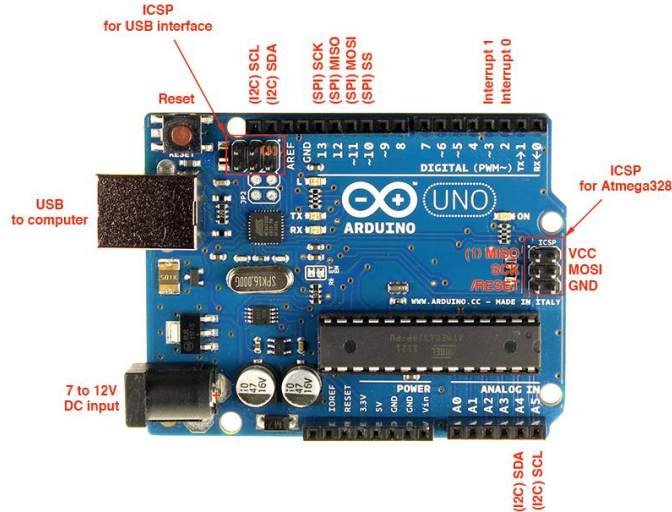
Arduino uno: conexiones


- Pinout, pin mapping



Arduino uno: comunicaciones

- Programación y comunicación (rev3)





Unidad 2
Programación de Arduino

Objetivos
 Contenido
 Introducción
Hardware
 Programación
 Funciones
 Interrupciones
 Librerías
 Conclusiones


7

Taller de Desarrollo de
 Sistemas Domóticos
 Basados en Arduino

 José L. Poza Luján
 Sergio Sáez Barona

Funciones de los pins


Pin	Funcionalidad
[0,13]	Entradas/Salidas digitales
0, 1	RX, TX. Serie TTL (ATmega8U2 USB-to-TTL chip)
2, 3	Interrupciones externas (por nivel alto o bajo, y por flancos de subida y bajada)
3,5,6,9,10,11	Salidas analógicas PWM (pulse-width modulation). Simulación de salidas analógicas por medio de una señal digital
10,11,12,13	Comunicaciones SPI (Serial Peripheral Interface)
GND	Tierra (Ground)
AREF	Referencia analógica
[A0,A5]	Entradas analógicas. 10 bits resolución = rango de 0 a 1023
A4,A5	Comunicaciones TWI/I2C (Two Wire Interface)
Vin	Alimentación externa (en combinación con GND)
5V	Salida de 5 V (en combinación con GND)
3V3	Salida de 3.3V, 50mA (en combinación con GND)
RESET	Reinicia la placa (activa a nivel bajo)
ICSP	In Chip/Circuit Serial Programmer (programador directo al ATmega)


Unidad 2
Programación de Arduino

Objetivos
 Contenido
 Introducción
Hardware
 Programación
 Funciones
 Interrupciones
 Librerías
 Conclusiones

8

Taller de Desarrollo de
 Sistemas Domóticos
 Basados en Arduino

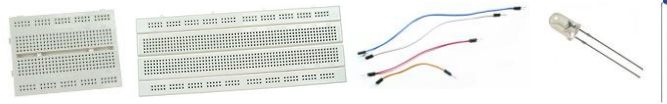
 José L. Poza Luján
 Sergio Sáez Barona

Alimentación

- USB
 - Tensión de alimentación: 5V
- Externa
 - Rangos
 - Trabajo: 7-12V
 - Límite: 6-20V
 - Conectores
 - 2.1mm (+ centro)s
 - Pins Vin, GND
- En el Arduino Uno, la selección de la fuente de alimentación es automática

Conexión

- Breadboard (tablero de circuitos)



- Buses

– Llevan en el mismo cable la alimentación y la señal del sensor/actuador



Peligros



- Algunas formas de destrozar un Arduino
 - #1 Cortocircuitar un pin de E/S con GND
 - #2 Cortocircuitar dos pines de E/S
 - #3 Aplicar sobretensiones en los pines de E/S
 - #4 Cambiar la polaridad de alimentación Vin y GND
 - #5 Cortocircuitar Vin y GND
 - #6 Aplicar tensión a los pines de salida de tensión (5V y 3V3)
 - #7 Aplicar más de 12 V al pin de reset
 - #8 Sobrecargar el Arduino (conectarlo todo sin alimentación adicional)

<http://ruggedcircuits.com/html/ancp01.html>



Y más, y más, y más

Precauciones



- **Hardware**
 - Montar los circuitos SIN alimentar el Arduino
 - Comprobar las conexiones antes de alimentar el Arduino
 - No forzar conexiones: al conectar y al desconectar
 - Comprobar que cada componente funciona correctamente individualmente
 - No desesperarse: es difícil que un montaje funcione a la primera
- **Software**
 - Guardar código frecuentemente
 - Comentar código
 - Un cambio, una prueba



- **Una vez conocido el hardware...**
 - ¿Qué características pueden ser convenientes para programar un Arduino?
 - a) Punteros
 - b) Funciones
 - c) Hilos
 - d) Interrupciones
 - ¿Qué lenguaje de programación parece más adecuado?
 - a) Ensamblador
 - b) Lenguaje de nivel medio (C)
 - c) Lenguaje de alto nivel (C++, JAVA)

Introducción a la programación

- Los programas de Arduino se conocen como “sketches” (bocetos)
- El lenguaje está basado en “[Processing](#)”
 - Origen en el [MIT](#) (2001)
 - Similar al C++
 - La adaptación a Arduino como plataforma de desarrollo se conoce como [Wiring](#)
 - Integra el lenguaje y la plataforma
- Referencias
 - <http://arduino.cc/es/Reference/HomePage>
 - <http://arduino.cc/es/Reference/Extended>



Programación. Sintaxis

- Declaraciones, instrucciones y sentencias
 - Entre llaves “{}”
 - Balanceadas (se cierran tantas como se han abierto)
 - Anidadas (se permiten grupos de llaves dentro de otros)
 - Terminadas en punto y coma ‘;’
- Comentarios
 - Línea: ‘//’
 - Bloque ‘/*’ ... ‘*/’

```
{
  declaraciones;
}
```

```
// Ejemplo de comentario de línea
/*
  Ejemplo de
  comentario de bloque
*/
```

Variables en Arduino

- Declaración
 - tipo_de_dato nombre = valor_por_defecto;
- Asignación
 - Mediante el signo '='
- Ámbito
 - Global
 - Declarada fuera de una función
 - Funcional
 - Declarada dentro de una función
 - Bloque
 - Declarada dentro de un bloque de llaves {}

```


int variable_global = 10;
void setup()
{
  int variable_local = 10;
  {
    int variable_bloque = 10;
  }
}
  
```

Ámbitos de las variables

- Estáticas
 - No se crean ni se destruyen cada vez que son llamadas las funciones en las que se declaran
 - Se declaran con la palabra "static" antes de la declaración
- Volátiles
 - Se emplea cuando una variable puede ser modificada por otro código fuera del ámbito en que se ha declarado (por ejemplo por una rutina de interrupción)
 - No realiza copias de la variable, la almacena en RAM


```
static int variable_estatica = 10;
```

```
volatile int estado = LOW;
```



Unidad 2
Programación de Arduino
 Objetivos
 Contenido
 Introducción
 Hardware
Programación
 Funciones
 Interrupciones
 Librerías
 Conclusiones

17

Taller de Desarrollo de Sistemas Domóticos Basados en Arduino


 José L. Poza Luján
 Sergio Sáez Barona

Tipos de datos

- **byte**
 - 1 byte (8 bits). Entero sin signo. [0,255]
- **int**
 - 2 bytes (almacenado en [complemento a dos](#))
 - Con signo [-32.768,+32.767]
 - Sin signo [0,65,535]
- **word** (es un “unsigned int”)
- **long**
 - 4 bytes (almacenado en [complemento a dos](#))
 - Con signo [-2.147.483.648, +2.147.483.647]
 - Sin signo [0, 4.294.967.295]


```

byte b = 10001001B;
int inicio = 20;
unsigned int ledPin = 13;
word ledPin = 13;
long speedOfLight = 300000L;
  
```


Unidad 2
Programación de Arduino
 Objetivos
 Contenido
 Introducción
 Hardware
Programación
 Funciones
 Interrupciones
 Librerías
 Conclusiones

18

Taller de Desarrollo de Sistemas Domóticos Basados en Arduino

 José L. Poza Luján
 Sergio Sáez Barona

Tipos de datos

- **float**
 - 4 bytes
 - [-3.4028235E+38, +3.4028235E+38]
 - 7 decimales de precisión
 - NO es aritmética exacta
- **double**
 - Es un “float”
- **void**
 - Se emplea sólo en la definición de funciones
- **boolean**
 - Solo puede tomar los valores: “true”/”false”

```
float sensorCalbrate = 1.117;
```

– Precaución al operar conjuntamente con enteros

```
boolean running = false;
```

Tipos de datos

- **Matrices (arrays)**
 - Conjunto de valores almacenado secuencialmente
 - Se acceden de forma similar a como se hace en C
 - “Zero indexed”: el primer elemento se encuentra en la posición 0.
 - Declaración
 - Tamaño []
 - Inicialización {}

```
int arrayUno[5];
int arrayDos[] = {1, 2, 3, 4, 5};
int arrayTres[5] = {1, 2, 3, 4, 5};
```

- Acceso de lectura y escritura

```
int variable = arrayUno[3];
arrayUno[3] = 10;
```

- Los multidimensionales deben tener definido el tamaño

```
int arrayUno[10][2];
int arrayDos[2][2] = {{2,2},{3,3}};
```

Tipos de datos

- **char**
 - Almacena caracteres [ASCII](#) (7bits)
 - Valores
 - Carácter entre comillas simples("), decimal, hexadecimal
 - Internamente es un entero con signo
 - Valores de -128 a +127
 - Permite operaciones
 - Se permite sin signo (pero se considera “byte”)

```
char charUno = 'A';
char charDos = 65;
char charTres = 0x41;
```

- **string**
 - Secuencia de caracteres almacenados como array
 - Terminados en carácter NULL
 - [Amplia variedad de definiciones](#)

```
char cadena[] = "Ejemplo";
```

Constantes

- Definición

- Etiqueta “#define”

- Sin ‘;’ ni ‘=’
 - Compilador: sustituye la definición por el dato

```
#define ledPin 3
```

- Palabra clave “const”

- Se recomienda usar en lugar de #define
 - Evita errores de compilación

```
const float pi = 3.14;
```

- Para definir matrices, sólo es válido “const”

```
const int myPins[3] = {2,4,8};
```

Constantes definidas

- Booleanas: true / false

- En minúsculas
 - “false” es 0 (cero), “true” cualquier otro valor

- Pines digitales

- INPUT: alta impedancia por lo que permiten leer sensores pero no proporcionarles corriente
 - OUTPUT: baja impedancia, proporcionan corriente de hasta 40mA

- Puede alimentar LEDs
 - No puede alimentar sensores ni motores
 - PELIGRO: un pin a output no debe ser conectado ni a 5V ni a GND



Constantes definidas

• Niveles de los pines: HIGH/LOW

	INPUT Se configura como entrada con pinMode()	OUTPUT Se configura como salida con pinMode()
HIGH	<ul style="list-style-type: none"> • Si se lee con digitalRead, el microcontrolador nos retornará HIGH si en el pin hay 3 voltios o más. • Si se establece a HIGH con digitalWrite, conectará el pin a 5 Voltios a través de una resistencia interna de 20K (resistencia pull-up) y establecerá el pin al estado de lectura HIGH a menos que la conectemos a una señal LOW a través de un circuito externo. 	<ul style="list-style-type: none"> • Si se establece a HIGH con digitalWrite, el pin tiene 5V. En este estado puede usarse como fuente de corriente, e.j. Luz y LED que se conectan a través de resistencias en serie a masa (tierra), o a otro pin configurado como salida y establecido a LOW.
LOW	<ul style="list-style-type: none"> • Si se lee con digitalRead, el microcontrolador retornará LOW si el voltaje presente en el pin es de 2V o menor. 	<ul style="list-style-type: none"> • Si se establecido LOW con digitalWrite, el pin tiene 0 voltios. En este estado puede aceptar corriente, e.j. Luz y LED que se conectan a través de resistencias en serie a +5 voltios, o a otro pin configurado como salida, y establecido a HIGH.

Constantes numéricas

• Constantes enteras

– Formateador de base

- Se incluyen al inicio del número
 - B: Binario (8bits), O: Octal, Ox: Hexadecimal

– Formateador de longitud

- Se incluyen al final del número
 - U: Unsigned (sin signo), L: Long (largo)
 - UL: Unsigned long (entero largo, sin signo)

```
const int n = B10011001;
const int n = 0xAB;
const int n = 1000ul;
```

• Constantes en coma flotante

– Formateador de notación científica: E, e.

```
const float n = 1.005;
const float n = 1.1e-02;
```

Operadores

- **Asignación (=)**
 - Evalúa el valor o el resultado de la expresión en el lado derecho del signo igual, y lo almacena en la variable a la izquierda del signo igual.
 - Si la variable es de distinto tipo del resultado, éste no será correcto.
- **Aritméticos**
 - Todos: Suma (+), Resta (-), Producto (*), División (/)
 - Sólo enteros: Módulo (%)
 - La operación (y por tanto el resultado) emplea el tipo de datos de los operandos
 - En cuanto hay una variable “float” o “double” la operación se realiza en coma flotante.
 - Pueden haber desbordamientos (comprobar límites)

```

resultado = valor1 + valor2;
resultado = valor1 - valor2;
resultado = valor1 * valor2;
resultado = valor1 / valor2;
resultado = valor1 % valor2;
  
```

Operadores

- **Comparación**
 - Igual (==), Distinto (!=), Menor (<) Mayor (>), Menor o igual (<=), Mayor o igual (>=)
 - El resultado de la expresión devuelve “true” o “false”

```

x == y // x es igual a y
x != y // x no es igual a y
x < y // x es menor a y
x > y // x es mayor a y
x <= y // x es menor o igual a y
x >= y // x es mayor o igual a y
  
```

- **Booleanos expresiones**
 - Permiten realizar expresiones complejas
 - Es posible anidar expresiones con paréntesis
 - And (&&), Or (||), Not (!)

```

(var1 == var2) && (var1 < var3) // var1 es igual a var2 y además var1
es menor que var3
(!var1) // La expresión es cierta si var1 es falso
  
```

Operadores

- **BITS**

- Permiten realizar operaciones “bit a bit”
- Booleanos
 - And (&) empleado para borrar bits a partir de una máscara
 - Or (|) usado para poner bits a uno a partir de una máscara
 - Not(~) ó ALT+126 Empleado para invertir bits
 - Xor (^) Empleado para invertir bits a partir de una máscara

```
byte var1 = B10011001;
byte var2 = var1 & B00001111; // var2 será el número binario 00001001
```

- Desplazamientos
 - Desplazamiento a derecha (>>). Extiende el bit de signo
 - Desplazamiento a izquierda (<<). Rellena con ceros

```
byte var1 = B10011001;
byte var2 = var1 << 3; // var2 será el número binario 11001000
```

Operadores

- **Compuestos**

- Realizan una operación y una asignación
- Incremento (++) y disminución (--)

```
int x = 1;
x++; // equivalente a la expresión x = x + 1;
x--; // equivalente a la expresión x = x - 1;
```

- Incremento (+=) y disminución (-=) en un intervalo
- Producto (*=) y división (/=) en un intervalo;

```
x += y; // equivalente a la expresión x = x + y;
x -= y; // equivalente a la expresión x = x - y;
x *= y; // equivalente a la expresión x = x * y;
x /= y; // equivalente a la expresión x = x / y;
```

- And (&=) y Or (|=)

```
x &= y; // es equivalente a x= x & y;
x |= y; // es equivalente a x= x | y;
```

- **Memoria**

- Referencia (&) devuelve la dirección de memoria
- Referencia (*) devuelve el contenido de la dirección de memoria

Estructuras de control

- Ramificación condicional (if)
 - Comprueba si cierta condición se da, en tal caso, ejecuta el bloque
 - Alternativa al condicional (else)

```

if (condicion) {
  // acción si condición es cierta
}
else {
  // acción si condición es falsa
}

```

- Ramificación múltiple condicional
 - Permite comprobar varias condiciones excluyentes

```

if (condicion1) {
  // acción si condición 1 es cierta
}
else if (condicion2)
  // acción si condición1 es falsa y condición2 es cierta
}
else {
  // acción si condición1 es falsa y condición2 es falsa
}

```

Estructuras de control

- Ramificación múltiple (switch, case, break, default)
 - Comprueba el valor de una variable y en función del valor, se ejecuta uno u otro bloque
 - Cuando se da la condición de un “case” se van ejecutando bloques hasta encontrar el “default”

```

int variable;
switch (variable){
  case 1:
  case 2:
    // acción si variable vale 1 o 2
    break;
  case 3:
    // acción si variable vale 3
  case 4:
    // acción si variable vale 3 o 4
    break;
  case 5:
    // acción si variable vale 5
    break;
  default:
    // si nada coincide, ejecuta el "default"
    // el "default" es opcional
};
}

```

Bucles

- **for(inicialización; condición; incremento)**
 - Se emplea cuando se conoce el número de iteraciones
 - Cualquier expresión es válida
 - Puede omitirse la inicialización, la condición y el incremento
 - Los ‘;’ son obligatorios.

```
for (int i=1; i<100; i++){
  // Bloque de instrucciones a ejecutar
}
```

- **while(condicion)**
 - Similares al bucle “for”
 - Es importante controlar la condición de salida del bucle mediante la modificación de las variables que intervienen en la condición

```
int i=1;
while(i<100){
  // Bloque de instrucciones a ejecutar
  i++;
}
```

Bucles

- **do..while(condición)**
 - Funciona igual que el bucle “while”
 - La condición se comprueba al final del bucle, por lo que este bucle se ejecuta como mínimo una vez.

```
int i=1;
do{
  // Bloque de instrucciones a ejecutar
  i++;
}
while(i<100);
```

- **Interrupción del bucle**
 - **Continue:** pasa a la siguiente iteración omitiendo el resto de instrucciones del bloque a ejecutar
 - **break:** sale del bucle sin terminar el bloque de instrucciones ni las iteraciones restantes

```
for (int i=0; i<20; i++){
  // Bloque de instrucciones a ejecutar siempre
  if (j<1) continue;
  if (j>10) break;
  // Bloque de instrucciones a ejecutar (si j >= 1 y j <= 10)
}
```


Flujo del programa

- goto
 - Cambia el flujo de programa, saltando la ejecución al punto etiquetado que se le indique
 - Etiqueta: nombre seguido de dos puntos (:)
 - Suele desaconsejarse, aunque puede ser útil para salir de bucles anidados

```
etiqueta;  
// Bloque de instrucciones a ejecutar  
goto etiqueta;
```

- return
 - Sale de la función sin ejecutar el resto de instrucciones
 - Puede devolver, o no, un valor (depende de la declaración de la función)

```
boolean funcion()  
{  
    boolean resultado;  
    // Bloque de instrucciones  
    if(resultado) return true;  
    else return false;  
}
```

Funciones en Arduino

- Agrupan código
 - Para resumir la funcionalidad
 - Facilitan la programación
- Similares a las de los principales lenguajes
 - Un dato de salida (retorno)
 - Varios datos de entrada (parámetros)
 - Bloque de código entre llaves

```
tipo_retorno nombre_funcion(tipo_parámetro parámetro,...)  
{  
    declaraciones;  
}
```

- Facilitan la programación
- Consumen menos memoria
- Ralentizan la ejecución

Manejo de tipos de datos

- **Conversión**

- Se emplean para convertir o forzar una variable a un tipo concreto
 - char(variable)
 - byte(variable)
 - int(variable)
 - word(variable)
 - long(variable)
 - float(variable)

```
int entero = 65;
char caracter = char(entero);
```

- Se debe tener especial precaución en la conversión de tipos
 - Posibles truncamientos, desbordamientos y comportamientos no previstos)

- **Tamaño**

- sizeof(variable). Devuelve el tamaño (en bytes) que ocupa una variable o un array

```
int tamanyo = sizeof(arrayUno);
```

Bucle de control

- **Inicialización (setup):**

- Declaración variables
- Es obligatoria, aunque no tenga contenido
- Se ejecuta una sola vez al inicio
 - Define los pins que serán de entrada o salida
 - Determina las comunicaciones

```
void setup()
{
  declaraciones;
}
```

- **Bucle de control (loop): ejecución continua**

- Núcleo de todos los programas

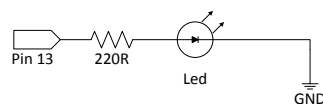
```
void loop()
{
  declaraciones;
}
```

Entrada-Salida digital

- `pinMode(pin, modo)`
 - Configura el “pin” para comportarse de entrada (modo=INPUT) o salida (modo=OUTPUT)
- `digitalRead(pin)`
 - Lee el valor del pin: HIGH o LOW en función del umbral de voltaje de entrada
 - Si no hay nada conectado al pin, puede devolver HIGH o LOW de forma aleatoria (estado desconocido)
- `digitalWrite(pin,valor)`
 - Escribe el valor (HIGH o LOW) en el “pin” seleccionado
 - Puede emplearse con el pin en modo entrada (INPUT) en tal caso se habilita una resistencia de “pullup” que permite circular tensión en el pin

Entrada-Salida digital

- Examples → Basics → Blink



```

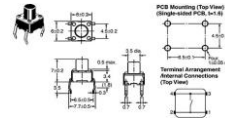
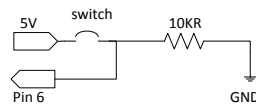
int pinLed = 13;

void setup()
{
  pinMode(pinLed, OUTPUT);
}

void loop()
{
  digitalWrite(pinLed, HIGH);
  delay(1000);
  digitalWrite(pinLed, LOW);
  delay(1000);
}
  
```

Entrada-Salida digital

- Examples → Basics → DigitalReadSerial



```

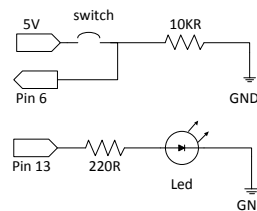
int pushButton = 2;

void setup()
{
  Serial.begin(9600);
  pinMode(pushButton, INPUT);
}

void loop()
{
  int buttonState = digitalRead(pushButton);
  Serial.println(buttonState);
  delay(1);
}
  
```

Entrada-Salida digital

- Control sencillo



```

int pinSwitch = 2;
int pinLed = 13;

void setup()
{
  pinMode(pinSwitch, INPUT);
  pinMode(pinLed, OUTPUT);
}

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



Unidad 2 Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación

Funciones

Interrupciones
Librerías
Conclusiones

41

Taller de Desarrollo de
Sistemas Domóticos
Basados en Arduino

José L. Poza Luján
Sergio Sáez Barona



- Ampliar el código anterior (Control sencillo) haciendo que ...
 - Al apretar el botón se apague el Led.
 - Al apretar el botón se encienda el Led y esté cinco segundos encendida (independientemente de si se vuelve a apretar el botón)
 - Al apretar el botón parpadee el Led tres veces (cada parpadeo debe durar un segundo encendido y un segundo apagado).



Unidad 2 Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación

Funciones

Interrupciones
Librerías
Conclusiones

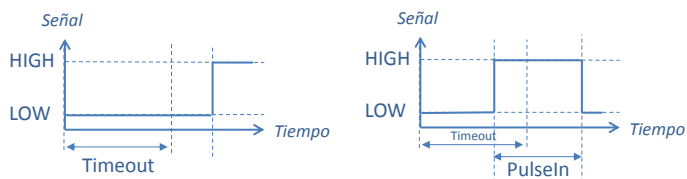
42

Taller de Desarrollo de
Sistemas Domóticos
Basados en Arduino

José L. Poza Luján
Sergio Sáez Barona

Entrada-Salida digital

- `pulseIn(pin, value, timeout)`
 - Mide un pulso del “pin”
 - “value” determina si se mide un pulso alto (HIGH) o bajo (LOW)
 - Devuelve la anchura del pulso medido en microsegundos
 - Funciona correctamente en pulsos con una anchura de 10 microsegundos a tres minutos, en mayores tiempos puede aparecer un error
 - “timeout” (opcional) en caso de especificarse devuelve 0 si el pulso no se inicia en el tiempo establecido



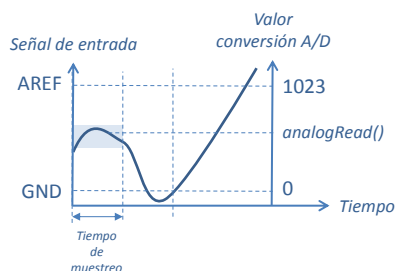
Entrada-Salida analógica

- `analogReference(tipo)`
 - Configura el voltaje de referencia usado por la entrada analógica.
 - Tipos
 - DEFAULT: 5V o 3.3V (en función de la placa)
 - INTERNAL: 1.1V en los ATMEGA 168 y 328, 2.56V en el ATMEGA8
 - EXTERNAL: se empleará la tensión del pin AREF
 - La función `analogRead()` devolverá un valor de 1023 para aquella tensión de entrada que sea igual a la tensión de referencia.
 - Es recomendable que cuando se use la referencia de tensión externa se conecte al pin AREF usando una resistencia de 5K. Lo que forma un divisor resistivo que deberá tenerse en cuenta en el cálculo de la lectura
 - La configuración por defecto del Arduino es la de no tener nada conectado de forma externa al pin AREF. En este caso la configuración de la tensión de referencia será DEFAULT lo cual conecta AVCC (Alimentación positiva +5V) de forma interna al pin AREF. Este pin es un pin de baja impedancia (mucho corriente) por lo que si usando la configuración DEFAULT de la tensión de referencia se conecta otra tensión a AREF que no sea la que posee AVCC, se dañará el chip ATmega.



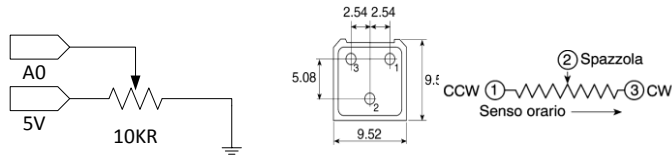
Entrada-Salida analógica

- `analogRead(pin)`
 - Lee la tensión del “pin” en función de la referencia establecida
 - El convertor es de 10 bits, lo que proporciona 1024 valores entre 0 y la tensión de referencia.
 - Si se emplean 5v de tensión de referencia, la precisión será de 4,9mV
 - El convertor tarda aproximadamente 100 microsegundos (0.0001 segundos) en leer una entrada analógica
 - Puede llevar una tasa de lectura máxima aproximada de 10.000 lecturas por segundo.



Entrada-Salida analógica

- Examples → Basics → AnalogReadSerial



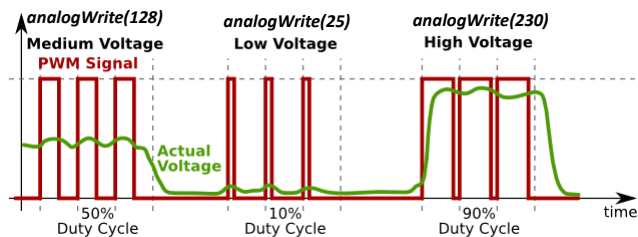
```

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(1);
}
  
```

Entrada-Salida analógica

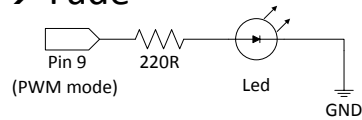
- analogWrite(pin,valor)
 - Escribe en el “pin” la tensión “valor” o ciclo de trabajo (tiempo del nivel superior de la onda, que va de 0 (siempre apagado) a 255 (siempre encendido))
 - El pin genera una onda cuadrada estable con el ciclo de trabajo especificado en “valor”
 - La frecuencia de la onda es de 490Hz
 - Los pines 5 y 6 poseen ciclos de trabajo alterados ya que usan el mismo temporizador que para las funciones de temporización



Analogical output & Analogical input

- Examples → Basics → Fade

```
int led = 9;
int brightness = 0;
int fadeAmount = 5;
```

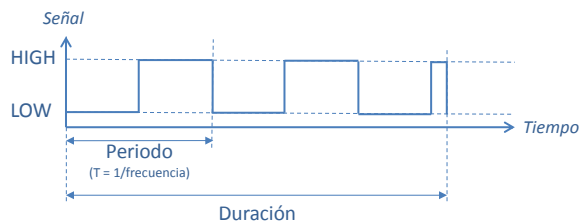


```
void setup()
{
  pinMode(led, OUTPUT);
}
```

```
void loop()
{
  analogWrite(led, brightness);
  brightness = brightness + fadeAmount;
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  delay(30);
}
```

Entrada-Salida analógica

- tone(pin,frecuencia,duracion)
 - Genera una onda cuadrada de la frecuencia especificada (y un 50% de ciclo de trabajo) en un pin. La duración (en milisegundos) puede ser especificada, en caso contrario la onda continua hasta que haya una llamada a noTone().
 - Sólo se puede generar un tono (de entre todos los pins)



- noTone(pin)
 - Detiene la señal cuadrada generada en el "pin" especificado

Cálculo

- **min(x,y), max(x,y)**
 - Calculan el mínimo y el máximo de dos números (x e y)
 - Se pueden combinar tipos de datos
 - El resultado se adapta al tipo asignado a la función
- **constrain(x,min,max)**
 - Restringe el rango de una variable
 - Devuelve x, si ésta se encuentra entre el mínimo y el máximo
 - Si $x < \text{min}$, devuelve “min”, si $x > \text{max}$, devuelve “max”

```
int x=1;
float y=10.2;
int minimo=min(x,y);
```

```
valSensor = constrain(valSensor, 10, 100);
```

- **abs(x)**
 - Devuelve el valor absoluto de x

```
int x=-1;
int resultado=abs(x); // resultado = 1 (no -1)
```

Cálculo

- **map(value, fromLow, fromHigh, toLow, toHigh)**
 - Escala (mapea) el “value” de la escala “from” a la escala “to”

```
val = map(val, 0, 1023, 0, 255); // de 10 bits a 8 bits
```

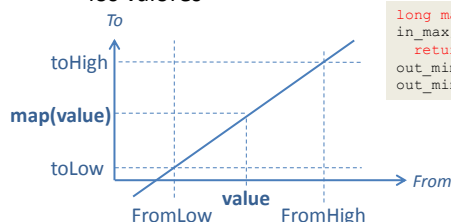
- No limita los valores fuera de rango, también los mapea
 - Para limitar se debe usar la función “constrain”
- Puede emplearse para invertir valores

```
y = map(x, 0, 50, 50, 0); // Aplica la función y = 50-x
```

- También trabaja con números negativos

```
y = map(x, 0, 100, -100, 0); // Aplica la función y = x-100
```

- Aplica la ecuación de la recta que pasa por los límites de los valores



```
long map(long x, long in_min, long in_max, long out_min, long out_max){
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;}

```

Cálculo

- Potencias
 - pow(base,exponente)
 - Eleva la “base” al “exponente”
 - Funciona con enteros y con reales
 - sq(x)
 - Eleva al cuadrado el número “x”
 - sqrt(x)
 - Calcula la raíz cuadrada de “x”
 - exp(x)
 - Devuelve “e” elevado a x
 - log(x)
 - Devuelve el logaritmo en base “e” de x
- Trigonométricas
 - sin(x)
 - cos(x)
 - tan(x)
 - Devuelven, respectivamente, el seno, el coseno y la tangente del parámetro x
 - El parámetro x deberá estar en radianes (float)

Gestión del tiempo

- millis()
 - Tiempo en milisegundos desde que el programa se inició
 - El parámetro de retorno es un “unsigned long”
 - Desborda en aproximadamente 50 días y vuelve a contar desde cero
- micros()
 - Tiempo en microsegundos desde que el programa se inició
 - Desborda en aproximadamente 70 minutos
 - La precisión depende de la frecuencia de reloj del procesador
- delay(x)
 - Pausa la ejecución del programa durante “x” milisegundos
 - El procesador no puede realizar (casi) ninguna otra acción mientras está en pausa
 - Sigue funcionando la comunicación serie de RX
 - No inhabilita las interrupciones
- delayMicroseconds(x)
 - Pausa la ejecución durante “x” microsegundos



Unidad 2 Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación
Funciones
Interrupciones
Librerías
Conclusiones

53

Taller de Desarrollo de
Sistemas Domóticos
Basados en Arduino

CC BY NC SA José L. Poza Luján
Sergio Sáez Barona

Números aleatorios

- **randomSeed(x)**
 - Inicia la “semilla” para la generación de números aleatorios
 - El valor “x” generará la secuencia, a un mismo valor, una misma secuencia
 - Para evitar la repetición de secuencias, conviene incluir una lectura aleatoria (de un puerto analógico, por ejemplo)

```
randomSeed(analogRead(0));
```

- **random(x)**
 - Genera números entre 0 y un valor máximo “x-1”
 - Si no se emplea “randomSeed()” la secuencia generada será siempre igual para todas las ejecuciones
- **random(x,y)**
 - Genera números entre los valores “x” e “y”
 - Si no se emplea “randomSeed()” la secuencia generada será siempre igual para todas las ejecuciones



Unidad 2 Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación
Funciones
Interrupciones
Librerías
Conclusiones


54

Taller de Desarrollo de
Sistemas Domóticos
Basados en Arduino

CC BY NC SA José L. Poza Luján
Sergio Sáez Barona

Manejo de bits

- En las funciones de bits, el primer bit es el de la posición cero, y se considera el bit de menor peso
- **bitClear(x,n)**
 - Pone a cero el bit de la posición “n” de la variable numérica “x”
- **bitSet(x,n)**
 - Pone a uno el bit de la posición “n” de la variable numérica “x”
- **bitRead(x,n)**
 - Lee el bit de la posición “n” en la variable numérica “x”
- **bitWrite(x,n,b)**
 - Escribe “b” (uno o cero) en la posición “n” de la variable numérica “x”
- **lowByte(x)/highByte(x)**
 - Extraen el byte menos/más significativo de la variable “x”
- **bit(n)**
 - Obtiene la potencia de dos de la posición del bit “n”


ARDUINO


Unidad 2
Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación
Funciones
Interrupciones
Librerías
Conclusiones

55


Taller de Desarrollo de Sistemas Domóticos Basados en Arduino

José L. Poza Luján
Sergio Sáez Barona



Interrupciones en Arduino

- Las interrupciones se emplean...
 - ...para atender automáticamente los eventos que se producen en el sistema
 - ...para resolver problemas de temporización
 - ...para monitorizan entradas externas y de usuario
- Permiten descargar al bucle de control
 - El sistema funciona por eventos en lugar de funcionar por muestreo
 - El resultado es más eficiencia
- Toda interrupción lleva aparejada una función que la atiende
 - Esto tiene efectos colaterales con las funciones de temporización (delay(), millis(), etc.)
- La mayoría de las placas Arduino tienen dos interrupciones externas:
 - la número 0 (en el pin digital 2)
 - la número 1 (en el pin digital 3).
 - La Arduino Mega tiene otras cuatro más


ARDUINO


Unidad 2
Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación
Funciones
Interrupciones
Librerías
Conclusiones

56

Taller de Desarrollo de Sistemas Domóticos Basados en Arduino

José L. Poza Luján
Sergio Sáez Barona



Manejo de interrupciones

- interrupts()
 - Habilita las interrupciones
- noInterrupts()
 - Deshabilita las interrupciones
 - Librerías y comunicaciones pueden no funcionar
- attachInterrupt(interrupt, function, mode)
 - Asocia la función “function” a la interrupción “interrupt”
 - Sustituye a la función que anteriormente estuviera asociada
 - La función asociada a la interrupción no puede tener parámetros ni devolver ningún valor
 - El “mode” define el evento que dispara la interrupción:
 - LOW. Siempre que el valor del pin se encuentre a nivel LOW
 - RISING. Cuando el pin pasa de valor HIGH a LOW
 - FALLING. Cuando el pin pasa de valor LOW a HIGH
 - CHANGE. Cuando el valor del pin cambia LOW a HIGH, o de HIGH a LOW
- detachInterrupt(interrupt)
 - Elimina la asociación de la función a la interrupción

Librerías en Arduino

- Inclusión de librerías

- Etiqueta `#include`
 - Sin `'` ni `=`
 - Compilador: emplea el código de la librería
- Sintaxis
 - `"ruta/librería"`: busca primero en el directorio y luego en la ruta
 - `<ruta/librería>`: busca en la ruta

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

- Librerías incluidas
 - Serie, EPROM, LCD, SD, y muchas más
- [Librerías compatibles](#)
- Las funciones se emplean referenciando tanto el nombre de la librería como el de la función
 - `librería.función`

Librerías en Arduino

- Estándares

- [EEPROM](#) - Leer y escribir en memorias "permanentes".
- [Ethernet](#) - Conectar a internet usando el Shield de Ethernet.
- [Firmata](#) - Comunicación con aplicaciones en el ordenador por COM.
- [LiquidCrystal](#) - Control de "displays" de cristal líquido (*LCD*)
- [Servo](#) - Control de *servomotores*
- [SoftwareSerial](#) - Comunicación serie por cualquier pin digital.
- [Stepper](#) - Control de motores paso a paso (*Stepper motors*)
- [Wire](#) - Interfaz de dos cables, ó *Two Wire Interface (TWI/I2C)*, envío y recepción de datos a través de una red de dispositivos y sensores.

- Comunicación (*networking* y protocolos):

- [Messenger](#) - Procesa mensajes de texto desde el ordenador.
- [NewSoftSerial](#) - Versión mejorada de la librería *SoftwareSerial*.
- [Simple Message System](#) - Envía mensajes entre Arduino y el ordenador.
- [SSerial2Mobile](#) - Envía mensajes de texto o emails usando un teléfono móvil (vía comandos AT serie)
- [Webduino](#) - Librería de web server extensible (*Ethernet Shield*)
- [X10](#) - Para enviar señales X10 a través de líneas de corriente AC.
- [XBee](#) - Para comunicaciones entre *XBee*s en modo *API*.
- [SerialControl](#) - Para controlar remotamente otras Arduino a través de una conexión serial.

Comunicación serie

- Todas las placas Arduino tienen al menos un puerto serie (también conocido como UART o Universal Asynchronous Receiver-Transmitter)
 - Se comunica a través de los pines digitales 0 (RX) y 1 (TX)
 - Si se emplea la comunicación serie, los pines 0 y 1 no pueden emplearse
 - Con el ordenador se comunica mediante USB
- Para la comunicación se emplea la librería “Serial”
 - Integrada en el IDE, no es necesario incluirla

Comunicación serie

- `Serial.begin(velocidad)`
 - Inicia las comunicaciones series a la velocidad en [baudios](#)
 - Baudio: cambios de estado por segundo (no bits por segundo)
 - Puede ponerse cualquier velocidad
 - Habitualmente se emplea 9600 baudios
- `Serial.end()`
 - Desactiva la comunicación serie
 - Los pines 0 y 1 vuelven a estar disponibles
- `Serial.available()`
 - Devuelve el número de bytes que ya se han recibido
 - Son bytes en buffer (el buffer de Arduino es de 128 bytes)
 - Se emplea para hacer comprobaciones de llegadas de datos

```
if (Serial.available() > 0)
{
    // Se pueden leer bytes de entrada
}
```

Comunicación serie

- **Serial.read()**
 - Devuelve el primer byte del buffer de recepción del puerto serie
 - Si no hay datos, devuelve -1
- **Serial.flush()**
 - Vacía el buffer de entrada
 - Cualquier llamada a `Serial.read ()` o `Serial.available ()` devolverá sólo los datos recibidos después la llamada más reciente a `Serial.flush ()`.
- **Serial.write(val)**
 - Envía un solo byte (`val`) por el puerto serie
- **Serial.write(str)**
 - Envía la cadena “`str`” (de tipo string) como datos binarios
- **Serial.write(buf,len)**
 - Envía un número “`len`” de bytes del array almacenado en “`buf`”

Comunicación serie

- **Serial.print(val)**
 - Imprime por el puerto serie, como texto ASCII, el valor “`val`”
 - Los números se convierten a ASCII (cada dígito un carácter)
 - Los “float” se envían con dos decimales
 - Las cadenas “string” se envían tal cual
- **Serial.print(val,format)**
 - Similar a la anterior, con la ampliación de que “`format`” especifica la base
 - BYTE: tal cual se almacena en memoria
 - BIN: Base 2, binarios
 - OCT: Base 8, octal
 - DEC: Base 10, decimal
 - HEX: Base 16, hexadecimal
 - “`número`”: posiciones decimales de los “float”
- **Serial.println(val)/Serial.println(val,format)**
 - Similar a las anteriores, pero al final de la impresión añade un carácter de retorno de carro (ASCII 13, o “`\r`”) y un carácter de avance de línea (ASCII 10, o “`\n`”)



Unidad 2 Programación de Arduino

Objetivos
Contenido
Introducción
Hardware
Programación
Funciones
Interrupciones
Librerías
Conclusiones

63

Taller de Desarrollo de
Sistemas Domóticos
Basados en Arduino

CC BY-NC-SA José L. Poza Luján
Sergio Sáez Barona

Conclusiones

- Arduino funciona como microcontrolador con entradas y salidas digitales y analógicas
- El lenguaje de programación es similar a C
- La programación se basa en
 - Iniciación: setup()
 - bucle de control: loop()
- Dispone de una gran cantidad de funciones
 - Genéricas
 - De Entrada/Salida
- Atiende interrupciones
- La potencia de Arduino se amplía con la inclusión de librerías

José L. Poza Luján
Sergio Sáez Barona

ETSINF
ETSINF



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA