



Universidad de Granada

[decsai.ugr.es](http://decsai.ugr.es)

# **Teoría de la Información y la Codificación**

Grado en Ingeniería Informática

**Seminario 1.- Introducción a Arduino.**



**DECSAI**

**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**



UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

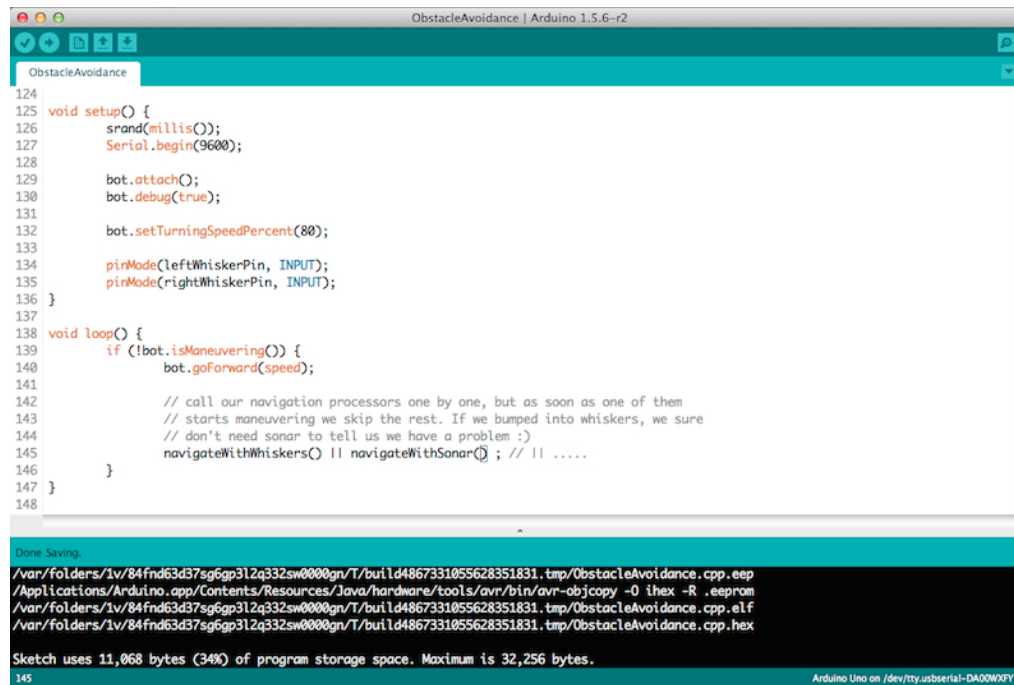
Grado en Ingeniería Informática

- » 1. Instalación para las prácticas
- 2. ¿Qué es Arduino?
- 3. Compilando para AVR
- 4. La biblioteca ArduTIC
- 5. Comunicaciones en serie
- 6. Dispositivos GPIO
- 7. Descripción de la práctica 1



DECSAI

- Arduino tiene su propio IDE de programación, con código C. Es el utilizado principalmente por aquellos que se quieren iniciar en Arduino como hobby:



```

124
125 void setup() {
126     srand(millis());
127     Serial.begin(9600);
128
129     bot.attach();
130     bot.debug(true);
131
132     bot.setTurningSpeedPercent(80);
133
134     pinMode(leftWhiskerPin, INPUT);
135     pinMode(rightWhiskerPin, INPUT);
136 }
137
138 void loop() {
139     if (!bot.isManeuvering()) {
140         bot.goForward(speed);
141
142         // call our navigation processors one by one, but as soon as one of them
143         // starts maneuvering we skip the rest. If we bumped into whiskers, we sure
144         // don't need sonar to tell us we have a problem :)
145         navigateWithWhiskers() || navigateWithSonar(); // || .....
146     }
147 }
148
Done Saving.
/var/folders/1v/84fnd63d37sg6gp312q332sw0000gn/T/build4867331055628351831.tmp/ObstacleAvoidance.cpp.eep
/Applications/Arduino.app/Contents/Resources/Java/hardware/tools/avr/bin/avr-objcopy -O ihex -R .eeprom
/var/folders/1v/84fnd63d37sg6gp312q332sw0000gn/T/build4867331055628351831.tmp/ObstacleAvoidance.cpp.elf
/var/folders/1v/84fnd63d37sg6gp312q332sw0000gn/T/build4867331055628351831.tmp/ObstacleAvoidance.cpp.hex

Sketch uses 11,068 bytes (34%) of program storage space. Maximum is 32,256 bytes.
145
Arduino Uno on /dev/tty.usbserial-DA00WXPY

```


- En las prácticas, nosotros no haremos uso de este IDE, sino del compilador GCC para procesadores AVR.

- Lo primero que haremos será descargar el IDE de Arduino desde la web:

**<https://www.arduino.cc/en/main/software>**



### Download the Arduino IDE



**ARDUINO 1.8.5**

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation Instructions.

**Windows** Installer, for Windows XP and up  
**Windows** ZIP file for non admin install

**Windows app** Requires Win 8.1 or 10  
[Get](#)

**Mac OS X** 10.7 Lion or newer

**Linux** 32 bits  
**Linux** 64 bits  
**Linux** ARM

[Release Notes](#)  
[Source Code](#)  
[Checksums \(sha512\)](#)

#### HOURLY BUILDS

Download a **preview of the incoming release** with the most updated features and bugfixes.

#### BETA BUILDS

Download the **Beta Version** of the Arduino IDE with experimental features. This version should NOT be used in production.

- Al instalar el IDE se abren automáticamente los puertos y permisos necesarios.

- Necesitaremos además lo siguiente:
  - Compilador **AVR-GCC**
  - Biblioteca **AVR-LIBC**
  - Utilidades AVR (en especial **AVR-DUDE**)
  - Un **IDE de desarrollo** (Por ejemplo, Code::Blocks, Atom, etc.)
- En **Ubuntu**, la instalación es simple:

**sudo apt-get install avrdude binutils-avr gcc-avr avr-libc**

- En **Windows**: Añadir la carpeta de instalación al path.
- En **MAC** y **Linux**, además también necesitaremos permisos para el grupo **dialout** (comunicaciones por USB), en caso de que la instalación del IDE de Arduino no lo haya hecho por nosotros:

**sudo adduser ElUsuario dialout**

### – Ejemplo de instalación por línea de comandos:

```
manupc@manupcws: ~
manupc@manupcws:~$ sudo apt-get install gcc-avr
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
gcc-avr ya está en su versión más reciente (1:4.9.2+Atmel3.5.0-1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 177 no actualizados.
manupc@manupcws:~$
manupc@manupcws:~$ sudo apt-get install avrdude
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
avrdude ya está en su versión más reciente (6.2-5).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 177 no actualizados.
manupc@manupcws:~$
manupc@manupcws:~$ sudo apt-get install avr-libc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
avr-libc ya está en su versión más reciente (1:1.8.0+Atmel3.5.0-1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 177 no actualizados.
manupc@manupcws:~$
```

### — En las aulas de la ETSIT:

- Entrar con vuestro usuario y clave.
- Código: **stymfd**
- Se puede usar el editor Code::Blocks para programar
- Para compilar para AVR:
  - En C:\ hay una carpeta con el compilador AVR-GCC
  - Se debe incorporar al path antes de compilar:

**set PATH=%PATH%; carpeta; carpeta; ...; carpeta**

- Hay que incorporar al PATH las carpetas: **bin, include, lib**
- Los ficheros makefile deberán compilar correctamente y enviar los programas a Arduino con esta configuración.



UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

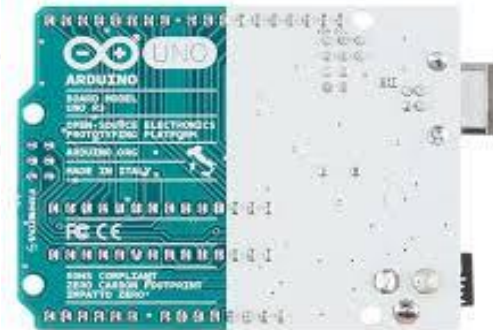
1. Instalación para las prácticas
- » 2. ¿Qué es Arduino?
3. Compilando para AVR
4. La biblioteca ArduTIC
5. Comunicaciones en serie
6. Dispositivos GPIO
7. Descripción de la práctica 1



DECSAI



- **Arduino** es un conjunto de placas controladoras y un entorno de programación, Open Hardware y Open Software.
- Facilitan la elaboración de proyectos de electrónica, automatismo, control, domótica, etc.
- Existen varios modelos de Arduino como son Uno, Leonardo, Mega...
- En el laboratorio utilizaremos el modelo **Arduino Uno**, por ser el más económico, el de mayor flexibilidad y posibilidades en proporción a su precio, y porque tiene la capacidad suficiente para la construcción de los prototipos de las prácticas.



- El desarrollo de proyectos con Arduino conlleva el uso de uno o varios elementos que se integran en la placa: entradas, salidas, alimentación, comunicación y shields de extensión.
  - **Entradas:** son pines incrustados en la placa. Se utilizan para adquirir datos desde sensores u otros dispositivos externos. En la placa Arduino Uno son los pines digitales (del 0 al 13) y los analógicos (del A0 al A5).
  - **Salidas:** los pines de salidas se utilizan para el envío de datos a dispositivos externos o a actuadores. En este caso los pines de salida son los pines digitales (0 a 13), que pueden configurarse como entrada o como salida.
  - **Otros pines de interés:** TX (transmisión) y RX (lectura) también usados para comunicación en serie, RESET para resetear el sistema, Vin para alimentar la placa con fuentes de alimentación externa, y los pines ICSP para comunicación por puerto SPI.

- El desarrollo de proyectos con Arduino conlleva el uso de uno o varios elementos que se integran en la placa: entradas, salidas, alimentación, comunicación y shields de extensión.
- **Alimentación:** Considerados para la alimentación de sensores y actuadores, tales como los pines GND (del inglés GROUND, tierra), pines 5V que proporcionan 5 Voltios, pines 3.3V que proporciona 3.3 Voltios, los pines REF de referencia de voltaje. El pin Vin sirve para alimentar la placa de forma externa, aunque lo normal es alimentarlo por USB o por el conector de alimentación usando un voltaje de 5 a 12 Voltios.
- **Comunicación:** Lo más normal es utilizar comunicación serie por USB para cargar los programas en la placa o para enviar/recibir datos. No obstante, existen shields que permiten extender la capacidad de comunicación de la placa utilizando los pines ICSP (comunicación ISP), los pines 10 a 13 (también usados para comunicación ISP), los pines TX/RX o cualquiera de los digitales.

- El desarrollo de proyectos con Arduino conlleva el uso de uno o varios elementos que se integran en la placa: entradas, salidas, alimentación, comunicación y shields de extensión.
- **Shields:** Son otras placas externas que se insertan sobre Arduino para extender sus capacidades. Algunas de las más comunes son las de Wi-Fi, sensores, actuadores (motores), Pantallas LCD, relés, matrices LED's, GPS, etc.



### – Especificaciones técnicas:



Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

### – Especificaciones técnicas:

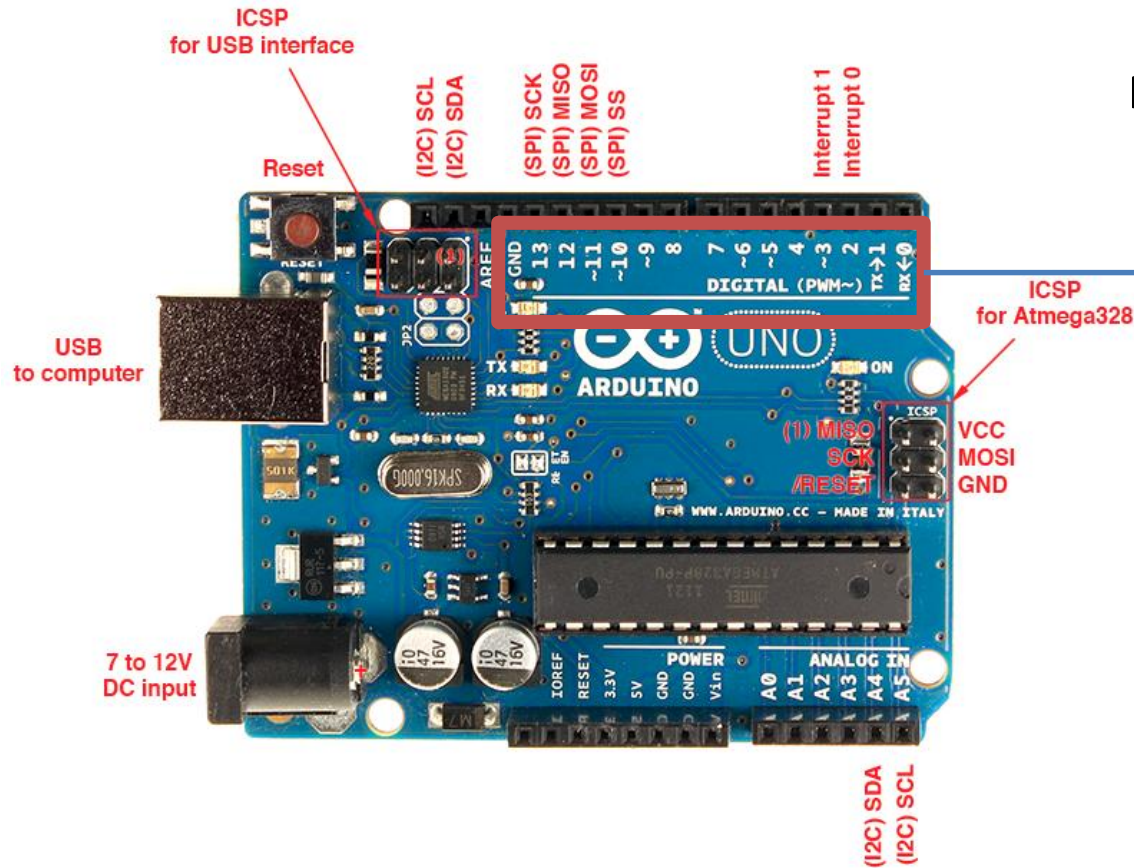


- **Sólo 2KB de memoria RAM** (menos lo que quiten bibliotecas específicas)
- **Sólo 32KB de tamaño de programa** (memoria de sólo lectura)
- **A 16 MHz**

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g



- Mapa de pines de la placa Arduino Uno (estándar):



Pines de entrada/salida (I/O) digitales a usar en las prácticas

- Arduino es seguro, pero también puede romperse. Aquí enunciamos **las 8 formas más comunes de destruir una placa Arduino**:
1. Conectar dos pines entre es arriesgado si no se sabe qué se hace. sí Especialmente si uno es tierra y el otro voltaje, la placa quedará dañada.
  2. Aplicar un voltaje superior a 5.5V a cualquier pin de entrada/salida.
  3. Al usar alimentación por Vin, invertir la corriente (conectar el positivo al negativo y viceversa).
  4. Conectar un voltaje superior al requerido en los pines de voltaje (por ejemplo, conectar 7V al pin de 5V o conectar 5V al pin de 3.3V).
  5. Conectar el voltaje directamente a tierra.
  6. Aplicar dos entradas de voltaje diferente para alimentar la placa (entrada externa Vin y entrada externa por el conector de alimentación).
  7. Aplicar un voltaje superior a 13V al pin RESET.
  8. Incluir en la placa una corriente superior a la soportada (la suma total de toda la corriente incluida en todos los pines de entrada/salida no puede superar los 200mA).





UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. Instalación para las prácticas
2. ¿Qué es Arduino?
- » 3. **Compilando para AVR**
4. La biblioteca ArduTIC
5. Comunicaciones en serie
6. Dispositivos GPIO
7. Descripción de la práctica 1



DECSAI

- Arduino Uno utiliza un microprocesador **AVR Atmega328p** de Atmel.



ATmega328 AVR  
Microcontroller

- Son procesadores de bajo consumo.
- Son de bajo precio.
- Tienen unas capacidades suficientes para el desarrollo de prototipos básicos.

- El mapa de puertos del procesador es el siguiente:

### Atmega168 Pin Mapping

Arduino function						Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)		analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)		analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)		analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)		analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)		analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)		analog input 0
VCC	VCC	7	22	GND		GND
GND	GND	8	21	AREF		analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC		VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)		digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)		digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)		digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)		digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)		digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

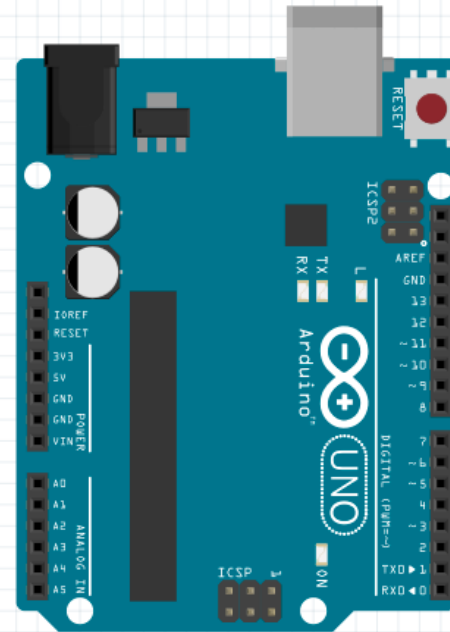
- La correspondencia de los puertos con los pines de la placa Arduino Uno es la siguiente:

MCU : Atmega 328  
 Input voltage : 7V-12V  
 Operating voltage : 5V  
 CPU Speed : 16MHZ  
 Analog In/Out : 6/0  
 Digital IO/PWM : 14/6  
 EEPROM : 1KB  
 SRAM : 2KB  
 Flash : 32KB  
 UART : 1

**Atmega168 Pin Mapping**

Arduino function	Atmega168 Pin	Atmega168 Pin	Arduino function
reset	(PCINT14/RESET) PC6	28	PC5 (ADC5/SCL/PCINT13)
digital pin 0 (RX)	(PCINT16/RXD) PD0	27	PC4 (ADC4/SDA/PCINT12)
digital pin 1 (TX)	(PCINT17/TXD) PD1	26	PC3 (ADC3/PCINT11)
digital pin 2	(PCINT18/INT0) PD2	25	PC2 (ADC2/PCINT10)
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	24	PC1 (ADC1/PCINT9)
digital pin 4	(PCINT20/XCK/T0) PD4	23	PC0 (ADC0/PCINT8)
VCC	VCC	22	GND
GND	GND	21	AREF
crystal	(PCINT6/XTAL1/TOSC1) PB6	20	AVCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	19	PB5 (SCK/PCINT5)
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	18	PB4 (MISO/PCINT4)
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	17	PB3 (MOSI/OC2A/PCINT3)
digital pin 7	(PCINT23/AIN1) PD7	16	PB2 (SS/OC1B/PCINT2)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	15	PB1 (OC1A/PCINT1)

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.



ARDUINO PIN	MICROCONTROLLER PIN
0	- PD0(RXD)
1	- PD1(TXD)
2	- PD2(INT0)
3	- PD3(INT1)
4	- PD4
5	- PD5
6	- PD6
7	- PD7
8	- PB0
9	- PB1
10	- PB2(SS')
11	- PB3(MOSI)
12	- PB4(MISO)
13	- PB5(SCK)
A0	- PC0
A1	- PC1
A2	- PC2
A3	- PC3
A4	- PC4(SDA)
A5	- PC5(SCL)

Programar para AVR en C no es diferente de programar para Windows o para Linux. Sólo hacen falta las bibliotecas necesarias:

### – Bibliotecas y macros útiles:

- **Biblioteca `<avr/io.h>`**: Contiene las definiciones de variables para direccionar puertos.
- **Biblioteca `<util/delay.h>`**: Contiene las definiciones de funciones para para la ejecución del programa por los instantes de tiempo requeridos.
- Macro **`F_CPU`**: Indica cada cuánto tiempo se refresca el tick del procesador.
- Macro **`#define _BV(bit) (1 << (bit))`**. Definida en `<avr/io.h>`: Se utiliza para transformar un bit a byte.

**Un programa tipo en C para Arduino UNO comenzaría así:**

```
// Utilizado para que el procesador pueda calcular el delay a partir del número de ticks del procesador
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>
```

- Escribiremos el siguiente programa en un fichero **main.cpp**:

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

### Qué vemos nuevo:

- Operadores a nivel de bits de C/C++:
  - $a \mid b$ : Realiza el or a nivel de bits de  $a$  y  $b$ . Ejemplo:

**unsigned char a= 1, b=4, c=a | b; // c vale 5: 00000101**

– **00000001 | 00000100 = 00000101**

- $a \& b$ : Realiza el and a nivel de bits de  $a$  y  $b$ . Ejemplo:

**unsigned char a= 3, b=1, c=a & b; // c vale 1: 00000001**

– **00000011 & 00000001 = 00000001**

Los operadores tienen sus correspondientes  $|=$  ,  $\&=$ .

- Operadores a nivel de bits de C/C++:
  - `~b`: Realiza el not a nivel de bits de `b`. Ejemplo:

`unsigned char a= 1, c=~a; // c vale 254: 11111110`

– `~00000001 = 11111110`

- `A<<n`: Desplaza todos los bits de `A`, `n` posiciones hacia la izquierda, introduciendo `n` 0's por la derecha.

– `A= 1; // 0b00000001      A<<=2; → // 0b00000100`

- `A>>n`: Desplaza todos los bits de `A`, `n` posiciones hacia la izquierda, introduciendo `n` 0's por la derecha.

– `A= 3; // 0b00000011      A>>=1; → // 0b00000001`



- Operadores a nivel de bits de C/C++:
  - ¿Cómo poner un bit  $n$  a 1? → Haciendo un OR a nivel de bits del dato con ese bit

**unsigned char a= 8; a |= (1<<4); // a vale 00001000,**  
**1<<4=00010000**

**Resultado: a=00011000**

- ¿Cómo poner un bit  $n$  a 0? → Haciendo un AND a nivel de bits del dato con el complementario de ese bit

**unsigned char a= 12; a &= ~(1<<2); // a vale 00001100,**  
**1<<2=00000100**

**~(1<<2)=11111011**

**Resultado: a=00001000**

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Arduino function	Pin	PC Function	Pin	PC Function	Arduino function
reset	1	(PCINT14/RESET) PC6	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	2	(PCINT16/RXD) PD0	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	3	(PCINT17/TXD) PD1	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	4	(PCINT18/INT0) PD2	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	5	(PCINT19/OC2B/INT1) PD3	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	6	(PCINT20/XCK/T0) PD4	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	7	VCC	22	GND	GND
GND	8	GND	21	AREF	analog reference
crystal	9	(PCINT6/XTAL1/TOSC1) PB6	20	AVCC	VCC
crystal	10	(PCINT7/XTAL2/TOSC2) PB7	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	11	(PCINT21/OC0B/T1) PD5	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	12	(PCINT22/OC0A/AIN0) PD6	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	13	(PCINT23/AIN1) PD7	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	14	(PCINT0/CLKO/CP1) PB0	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

**DDRB** es el registro de direccionamiento de datos del puerto B del microprocesador (PB0-PB7).

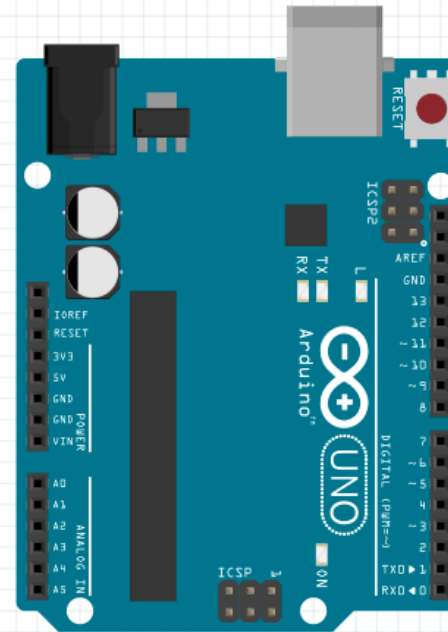
- Los pines PB0 a PB4 del microprocesador son los pines 8 a 12 de Arduino

MCU : Atmega 328  
 Input voltage : 7V-12V  
 Operating voltage : 5V  
 CPU Speed : 16MHZ  
 Analog In/Out : 6/0  
 Digital IO/PWM : 14/6  
 EEPROM : 1KB  
 SRAM : 2KB  
 Flash : 32KB  
 UART : 1

### Atmega168 Pin Mapping

Arduino function	MCU Pin	MCU Pin	MCU Pin	Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	VCC	7	22	GND
GND	GND	8	21	AREF
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

Digital Pins 11, 12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.



ARDUINO PIN		MICROCONTROLLER PIN
0	-	PD0(RXD)
1	-	PD1(TXD)
2	-	PD2(INT0)
3	-	PD3(INT1)
4	-	PD4
5	-	PD5
6	-	PD6
7	-	PD7
8	-	PB0
9	-	PB1
10	-	PB2(SS')
11	-	PB3(MOSI)
12	-	PB4(MISO)
13	-	PB5(SCK)
A0	-	PC0
A1	-	PC1
A2	-	PC2
A3	-	PC3
A4	-	PC4(SDA)
A5	-	PC5(SCL)

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function				Arduino function	
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLK0/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCS1, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

**DDB0** es la dirección del PIN 0 del puerto B del **microprocesador** (el pin PB0)

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function				Arduino function	
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCS1, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Haciendo esto estamos indicando que en el puerto B (microprocesador), el PIN 0 será de salida. Es decir, estamos haciendo  $DDRB = DDRB | 0x01$ .

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~ BV(PORTB0); // ~ es el NOT lógico a nivel de bits
    }
}
```

Atmega168 Pin Mapping

Arduino function				Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13) analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12) analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11) analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10) analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9) analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8) analog input 0
VCC	VCC	7	22	GND
GND	GND	8	21	AREF analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5) digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4) digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3) digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2) digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1) digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCS1, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

**Conclusión parcial:** El puerto DDRB del microprocesador controla los pines PB0 a PB7 del microprocesador, de los cuales los pines PB0 a PB5 se corresponden con los pines digitales de I/O de Arduino Uno, desde el pin 8 hasta el pin 13.



### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLK0/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCS1, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

**PORTB** es un byte que contiene los datos de salida existentes en los pines PB0-PB7 del **microprocesador**.

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function				Arduino function	
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLK0/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCSL, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

**PORTB0** es el bit 0: Corresponde al pin PB0 del **microprocesador**.



### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLK0/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCS1, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Haciendo esto estamos indicando al microprocesador que envíe voltaje alto por el pin 0 del puerto B. Es decir, estamos haciendo que los datos del puerto B sean  $PORTB = PORTB \mid 0x01$ . El Pin 8 de Arduino enviará voltaje.

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function				Arduino function	
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLK0/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCS1, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Aquí estamos esperando a que pase cierto tiempo

### — ¿¿ Qué hemos hecho ??

```
// Utilizado para el cálculo de ms en _delay_ms
#define F_CPU 1000000UL

#include <avr/io.h>
#include <util/delay.h>

#define BLINK_DELAY_MS 1000

int main (void)
{
    /* Pin 0 del puerto B del micro puesto como salida */
    DDRB |= _BV(DDB0);

    while(1) {
        /* Mandamos señal de voltaje alto al pin 0 del puerto B */
        PORTB |= _BV(PORTB0);
        _delay_ms(BLINK_DELAY_MS);

        /* Mandamos señal de voltaje bajo al pin 0 del puerto B */
        PORTB &= ~_BV(PORTB0); // ~ es el NOT lógico a nivel de bits
        _delay_ms(BLINK_DELAY_MS);
    }
}
```

Atmega168 Pin Mapping

Arduino function	Atmega168 Pin Mapping	Arduino function
reset	(PCINT14/RESET) PC6	PC5 (ADC5/SCL/PCINT13) analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	PC4 (ADC4/SDA/PCINT12) analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	PC3 (ADC3/PCINT11) analog input 3
digital pin 2	(PCINT18/INT0) PD2	PC2 (ADC2/PCINT10) analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	PC1 (ADC1/PCINT9) analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	PC0 (ADC0/PCINT8) analog input 0
VCC	VCC	GND
GND	GND	AREF analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	AVCC VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	PB5 (SCK/PCINT5) digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	PB4 (MISO/PCINT4) digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	PB3 (MOSI/OC2A/PCINT3) digital pin 11 (PWM)
digital pin 7	(PCINT23/AIN1) PD7	PB2 (SS/OC1B/PCINT2) digital pin 10 (PWM)
digital pin 8	(PCINT0/CLK0/ICP1) PB0	PB1 (OC1A/PCINT1) digital pin 9 (PWM)

Digital Pins 11, 12 & 13 are used by the ICSP header for MCSL, MISO, SCK connections (Atmega168 pins 17, 18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

**Aquí estamos haciendo la operación inversa: Poner a 0 la salida por el pin PB0 del microprocesador. El Pin 8 de Arduino enviará voltaje bajo (0V).**

### — Pasos para la compilación y envío del programa a la placa:

#### 1. Compilación:

```
avr-gcc -Os -mmcu=atmega328p -c -o main.o main.cpp
avr-gcc -mmcu=atmega328p main.o -o main
avr-objcopy -O ihex -R .eeprom main main.hex
```

#### 2. Búsqueda del puerto de conexión de Arduino Uno *(nota: Arduino Uno tiene que estar conectado al PC por el puerto USB):*

```
lsusb
ls /dev/serial/by-id -l
```

#### 3. Envío del programa a Arduino Uno:

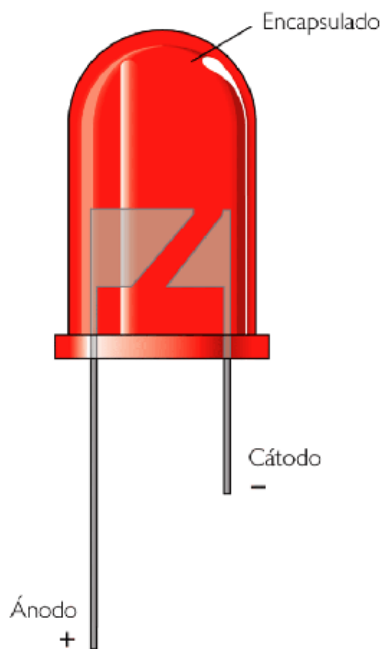
```
sudo avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:main.hex
```

#### 4. Envío para Windows: En lugar de /dev/ttyACM0, será COM2, COM3...

#### 5. Envío para MAC: En lugar de /dev/ttyACM0, será /dev/tty.usbmodel1411, /dev/cu.usbmodel1411, ...

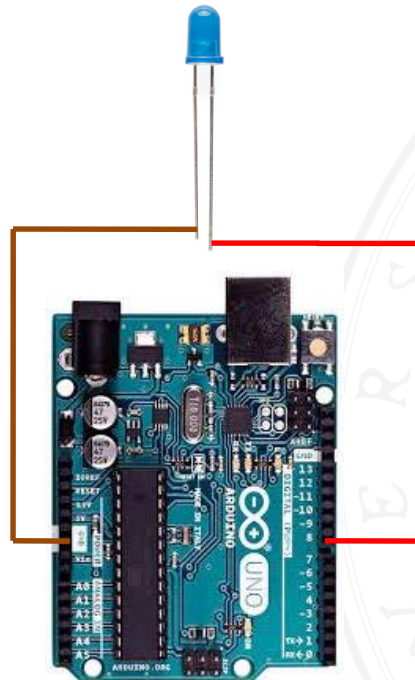
### – Prueba de funcionamiento:

1. **Seleccione un diodo LED en el laboratorio, y 2 cables de conexión macho-hembra.**
2. **Localice el cátodo (polo negativo) y el ánodo (polo positivo) del LED.**



### – Prueba de funcionamiento:

3. Conecte el extremo hembra de un cable al cátodo del LED, y el extremo macho a un pin GND de la placa Arduino.
4. Conecte el extremo hembra de otro cable al ánodo del LED, y el extremo macho al **PIN DIGITAL 8**.
5. Respuesta esperada: El LED debe parpadear.





UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. Instalación para las prácticas
2. ¿Qué es Arduino?
3. Compilando para AVR
- » 4. La biblioteca ArduTIC
5. Comunicaciones en serie
6. Dispositivos GPIO
7. Descripción de la práctica 1



DECSAI



Para simplificar las prácticas, se ha creado una biblioteca intermedia entre el código C y las bibliotecas de programación de los microchips AVR, específica para el modelo **AVR AtMega328p**.

Por tanto, no tendremos que lidiar de la misma manera a bajo nivel. La biblioteca está basada en las bibliotecas de **Arduino**. Por tanto, el código será prácticamente portable a la IDE de Arduino, con pocos cambios.

Se utiliza con **#include<ArduTIC.h>** en el fichero .cpp

**Habrá que enlazar también con el fichero libArduTIC.a**



Algunas funciones útiles de ArduTIC obtenidas de la API de Arduino:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```



Ejemplos:

**pinMode(8, OUTPUT)** configura el pin 8 de Arduino como SALIDA.  
**pinMode(9, INPUT)** configura el pin 9 de Arduino como ENTRADA.

**Siempre hay que configurar los pines como entrada o como salida al principio de nuestro programa.**

Algunas funciones útiles de ArduTIC obtenidas de la API de Arduino:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```



Ejemplos:

**digitalWrite(8, HIGH)** activa la salida por el pin 8 de Arduino (3.3v).  
**digitalWrite(8, LOW)** desactiva la salida por el pin 8 de Arduino (0v).

Sólo se debe aplicar **digitalWrite** sobre pines definidos como **SALIDA** con **pinMode**

**¡PELIGRO DE DAÑAR ARDUINO SI SE HACE DE OTRA FORMA!**

Algunas funciones útiles de ArduTIC obtenidas de la API de Arduino:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```



Ejemplos:

**int a= digitalRead(9)** Lee valor de entrada (HIGH/LOW) que haya en el pin 9 (por ejemplo, desde un sensor), y lo devuelve.

Sólo se puede aplicar **digitalRead** sobre **pines definidos como ENTRADA con pinMode**

Algunas funciones útiles de ArduTIC obtenidas de la API de Arduino:

```
#define HIGH 0x1
#define LOW 0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2
#define delay(ms) (_delay_ms(ms))

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
```

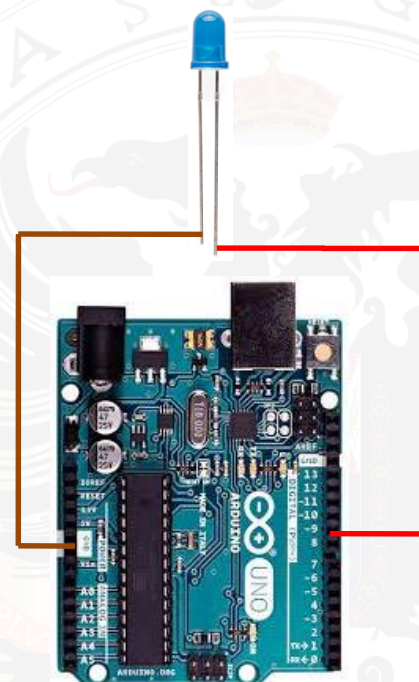


Ejemplos:

**delay(1000)** Provoca el bloque del programa por 1000 ms.

## – Prueba de funcionamiento:

1. Conecte el extremo del cátodo de un LED al pin 9, y el extremo ánodo a un pin GND de la placa Arduino.
2. Escriba un programa que:
  1. Defina el pin 9 como salida
  2. En un bucle infinito:
    1. Encienda el LED
    2. Espere 1 segundo
    3. Apague el LED
    4. Espere 1 segundo





## – Cómo compilar el programa:

### 1. Crear objeto:

```
avr-gcc -Os -mmcu=atmega328p -c -o ficheroObjeto.o fuente.cpp -I carpetasde.h
```

### 2. Crear binario:

```
avr-gcc -Os -mmcu=atmega328p ficheroObjeto.o -I carpetasde.h -lm -lArduTIC  
-Lcarpetade.a -o fichero.bin
```

### 3. Crear ejecutable:

```
avr-objcopy -O ihex -R .eeprom fichero.bin fichero.hex
```

### 4. Enviarlo a Arduino:

```
avrdude -F -V -c arduino -p ATMEGA328P -P elPuerto -b 115200 -U  
flash:w:fichero.hex
```

(OJO:Sustituir *elPuerto* por el puerto (ejemplo: /dev/ttyACM0) )





UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. Instalación para las prácticas
2. ¿Qué es Arduino?
3. Compilando para AVR
4. La biblioteca ArduTIC
- » 5. Comunicaciones en serie
6. Dispositivos GPIO
7. Descripción de la práctica 1



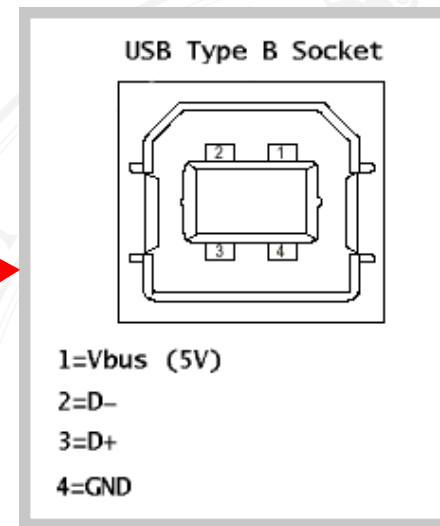
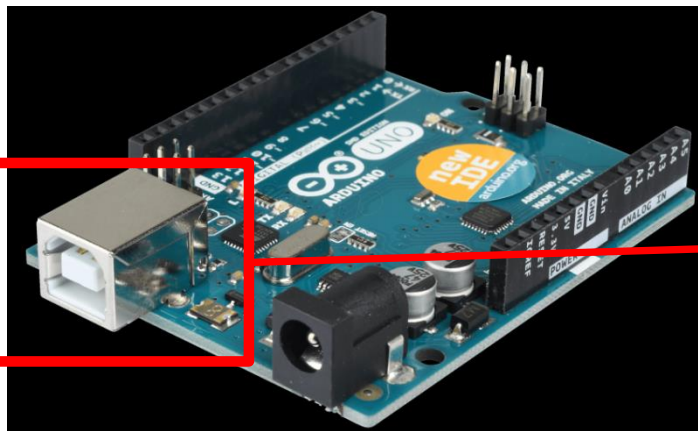
DECSAI

- **IMPORTANTE:** Siempre que entremos en prácticas, ejecutar y enviar a Arduino el programa vacío (programaVacio.zip), para evitar dañar cualquier componente de Arduino, sensores o actuadores.



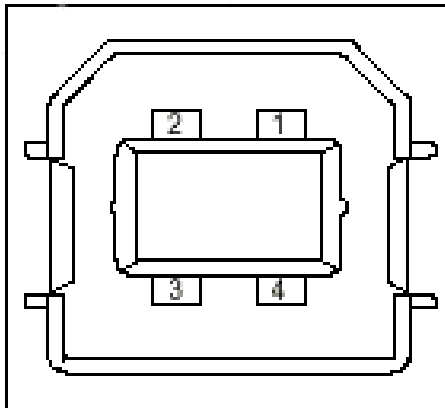
- Podemos comunicarnos con la placa Arduino de múltiples formas, aunque la mayoría de ellas requieren de shields adicionales:
  - Shield Wi-Fi
  - Shield Bluetooth
  - Pantallas táctiles
  - Etc.
- Todas las placas Arduino tienen posibilidad de comunicaciones por puerto serie (como mínimo un puerto serie). Así es como grabamos los programas en la memoria de la placa Arduino.
- Las **comunicaciones en serie** se realizan a través de **puertos UART**.

- El “**Bus Universal en Serie**” (**USB**) es un bus estándar que define las conexiones y protocolos para conectar, comunicar y alimentar periféricos y dispositivos electrónicos.
- En Arduino, **la comunicación por USB se realiza a través del puerto serie UART**. El conector utilizado en la placa es un **USB Tipo B**.



### – Descripción del conector USB Tipo B:

USB Type B Socket



1=Vbus (5V)

2=D-

3=D+

4=GND

- **Pin 1:** Alimentación con un voltaje de 5V DC
- **Pines 2 y 3:** Transmisión de datos
- **Pin 4:** Toma de tierra

- La gestión del puerto UART en procesadores AVR (y en cualquier procesador, por regla general), requiere de una inicialización.
- Esta inicialización implica también indicar cuál es la velocidad (en baudios) a la que se va a trabajar con el puerto (tradicionalmente, la velocidad del puerto serie más estándar es 9600, aunque un puerto USB puede trabajar cómodamente a 115200. Dependiendo de qué plataforma se use, a mayores velocidades se puede obtener errores en las comunicaciones.
  - Además, como el puerto UART es gestionado a través de interrupciones hardware, es necesario activar el módulo de gestión de interrupciones SEI.
- **Deberemos realizar estas operaciones sólo una vez al principio del programa, e incluirlas en todos los programas que implementemos y que hagan uso de comunicaciones en serie por USB.**

- La biblioteca ArduTIC proporciona el objeto global **Serial**, para realizar comunicaciones en serie.

```
#ifndef __cplusplus
extern "C" {
#endif

extern SerialPort Serial;

#ifdef __cplusplus
}
#endif
```

- Inicialización: **Serial.begin(baudios)**
- Ejemplo: **Serial.begin(9600);**

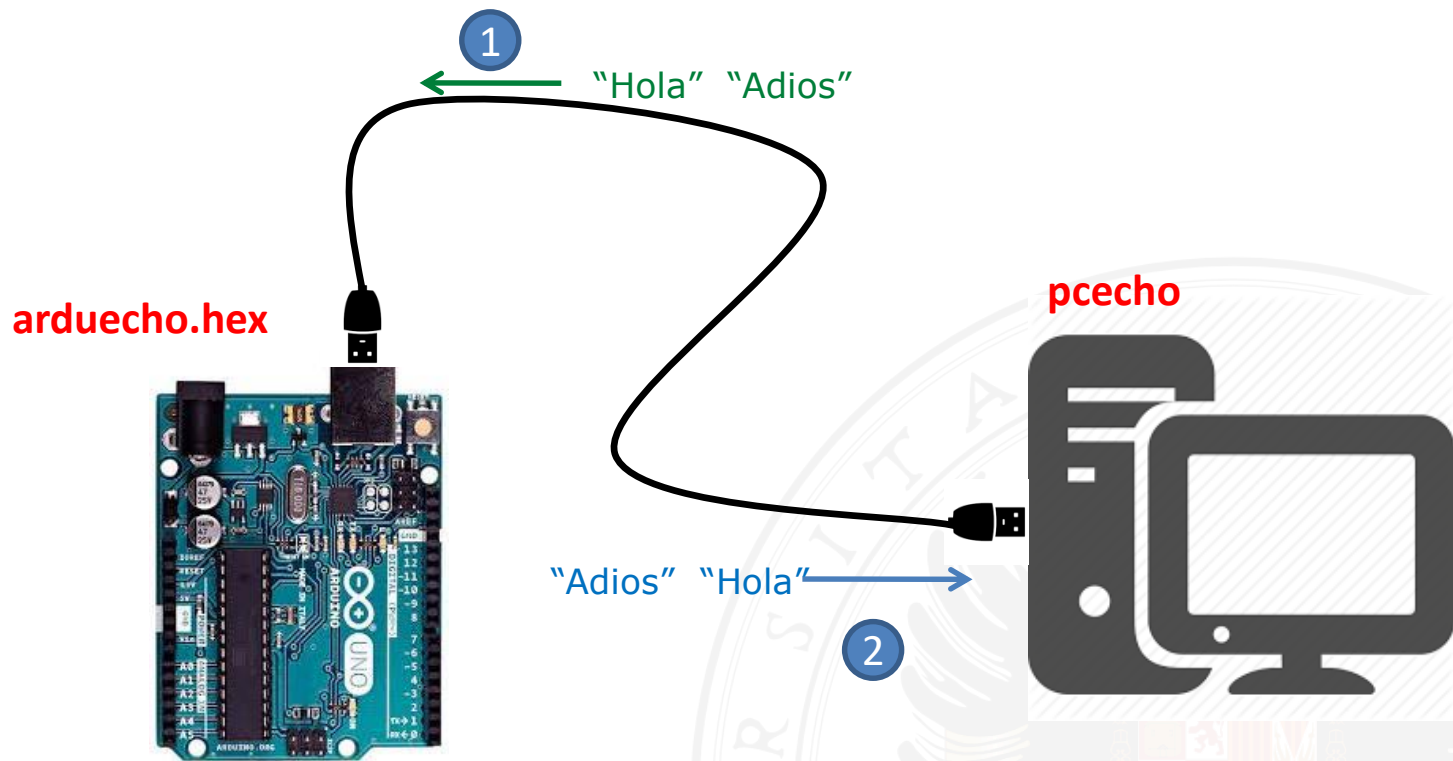


### – Métodos útiles de Serial:

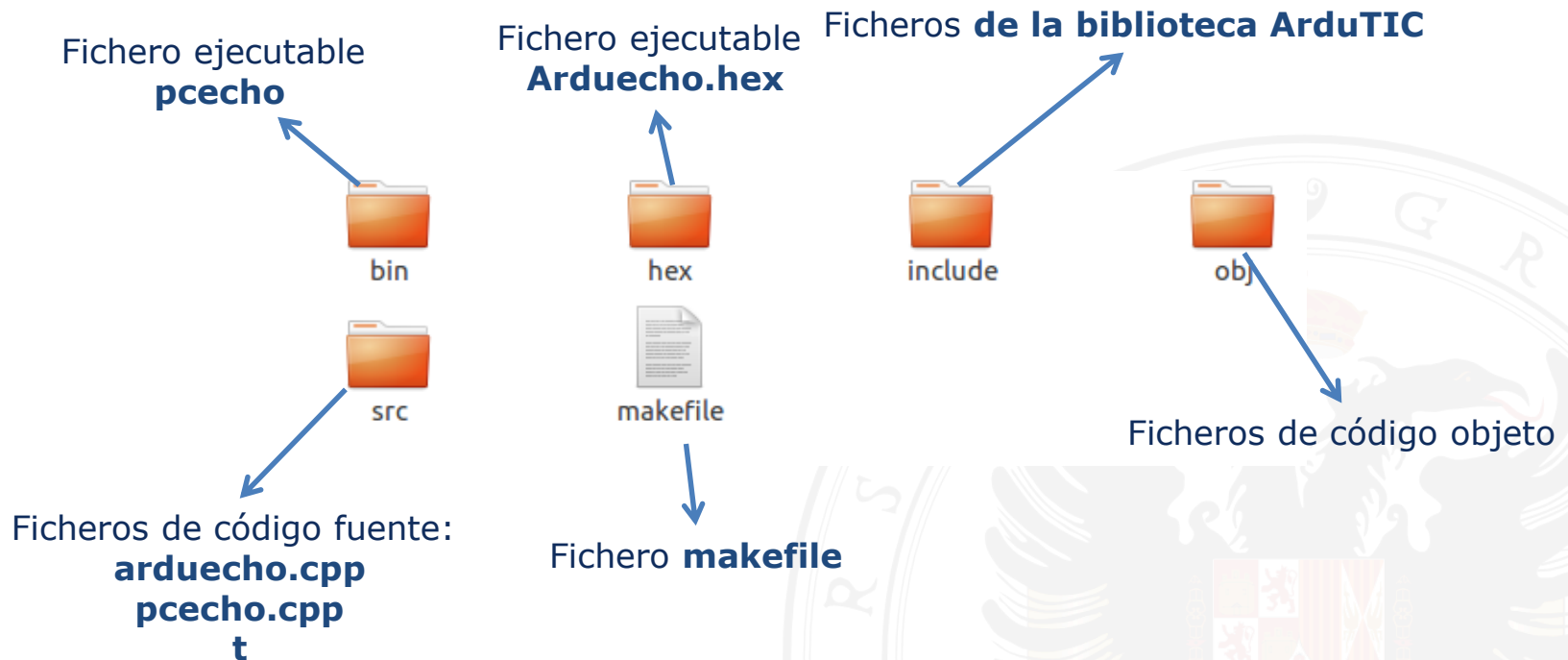
- **Serial.begin(baudios):** Inicializa las comunicaciones por el puerto serie (UART), a la velocidad dada en baudios.
- **Serial.print("Cadena"):** Envía la cadena por comunicaciones serie.
- **Serial.println("Cadena"):** Envía la cadena por comunicaciones serie, y la termina con un '\n'.
- **uint8\_t a= Serial.read():** Lee un byte desde comunicaciones serie.
- **Serial.readString(cadena):** Lee una cadena de caracteres desde puerto serie, y la guarda en el array **cadena** (**OJO: cadena** debe tener espacio suficiente (Ejemplo: char cadena[1000]).
- **bool a= Serial.available():** Devuelve true si hay datos disponibles para lectura por las comunicaciones serie. False en otro caso.

- Ejemplo de sistema para comunicación bidireccional PC-Arduino por puerto USB:
  - Construiremos un sistema cliente-servidor.
  - **Arduino** será el servidor (programa **arduecho.cpp**):
    - Estará escuchando el puerto USB para recibir datos.
    - **Enviaré por el puerto USB** los mismos datos recibidos (implementación de un servicio ECHO).
    - Utilizaremos una biblioteca existente **<uart.h>** que nos facilite la tarea de enviar y recibir datos por **UART**.
  - El **PC** será el cliente (programa **pcecho.cpp**):
    - Se conectará al puerto USB para enviar datos recibidos desde la línea de comandos.
    - Esperará a que Arduino responda por el puerto serie USB, con datos de respuesta, y los mostrará por pantalla.

### – Arquitectura general del sistema:



### – Estructura de carpetas del proyecto:



### — Programa arduEcho.cpp:

```
#include <ArduTIC.h>
```

```
int main() {
```

```
    // Inicializar puerto serie a 9600 baudios
    while (1) {
```

```
        // Si hay datos disponibles:
```

```
            // Leer los datos en una cadena
```

```
            // Enviar los datos por serie
```

```
        // En otro caso, esperar 500 ms
```

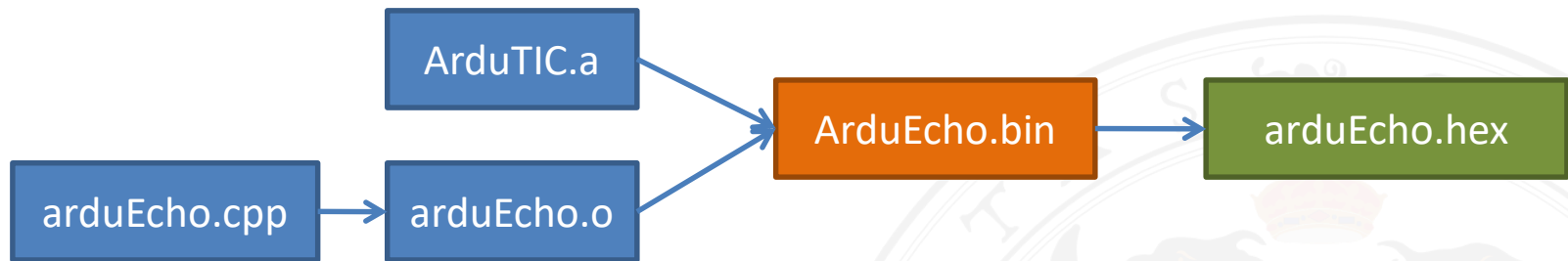
```
    }
```

```
    return 0;
```

```
}
```

### – El programa arduecho: Modelo de compilación

- Compilaremos según el siguiente esquema:



- Crearemos un fichero makefile para evitar repetir estas operaciones varias veces en el futuro.

### — El programa arduecho: Sección del fichero makefile

```

escriptorPORT=/dev/ttyACM0
lectorPORT=/dev/ttyACM0
INCLUDES=-I./include -I./lib
INCLUDEDIR=./include
LIBDIR=./lib
OBJDIR=./obj
CPPDIR=./cpp
BINDIR=./bin

# -----
# escriptor ARDUINO
# -----
escriptorArduino:
    avr-gcc -Os -mmcu=atmega328p -c -o $(OBJDIR)/arduEcho.o $(CPPDIR)/arduEcho.cpp $(INCLUDES)
    avr-gcc -Os -mmcu=atmega328p $(OBJDIR)/arduEcho.o $(INCLUDES) -lm -lArduTIC -L$(LIBDIR) -o $(OBJDIR)/arduEcho.bin
    avr-objcopy -O ihex -R .eeprom $(OBJDIR)/arduEcho.bin $(BINDIR)/arduEcho.hex

sendescriptorArduino:
    avrdude -F -V -c arduino -p ATMEGA328P -P $(escriptorPORT) -b 115200 -U flash:w:$(BINDIR)/arduEcho.hex

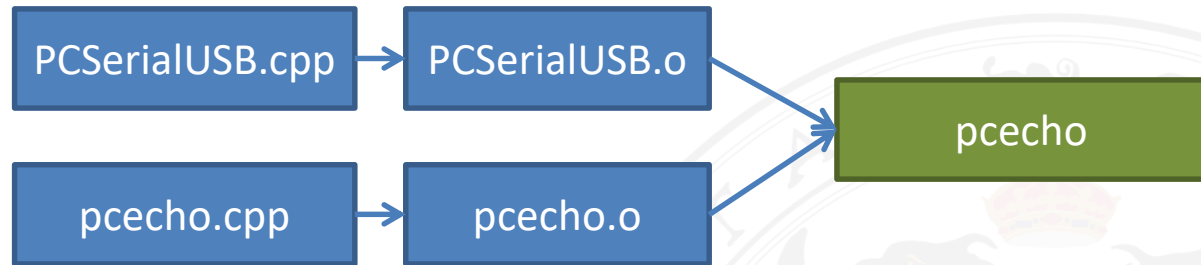
# -----
# escriptor PC
# -----
escriptorPC:
    g++ -o$(BINDIR)/pcEcho $(CPPDIR)/pcEcho.cpp $(LIBDIR)/PCSerialUSB.cpp $(INCLUDES)

```



### – El programa pcecho: Modelo de compilación

- Compilaremos según el siguiente esquema:



- Enlazaremos con el fichero **PCSerialUSB.cpp**, para ayudarnos en el resto de las prácticas.
- La biblioteca **PCSerialUSB** contendrá funciones para gestión de comunicaciones serie entre el PC y Arduino.
- Habrá que hacer **#include<PCSerialUSB.h>** en nuestro programa.

### – La biblioteca PCSerialUSB.h

- Desde el PC se deben seguir los mismos pasos que en Arduino:
  - Abrir el puerto de comunicaciones a una velocidad.
  - Leer o escribir.

```
/**
 * Función para inicializar el puerto USB
 *
 * Entradas: Una cadena de caracteres con el puerto USB (ejemplo: "/dev/ttyACM0", "/dev/ttyUSB0", "COM3", etc.)
 *
 * Salidas: Un descriptor de puerto correspondiente al puerto
 * de comunicaciones si hubo éxito (valor 0 o mayor). Un código de error (menor que 0) en caso contrario:
 * - Valor >=0 si todo se inicializó correctamente. Se corresponde con el código del puerto abierto
 * - Valor -1 si hubo error al abrir el puerto.
 * - Valor -2 si no se conoce el puerto de entrada dado en "portname"
 */
int InicializarUSB(const char *portname);
```

```
/**
 * Cierra las comunicaciones por el puerto USB para el descriptor dado por argumento
 */
void CerrarUSB(int &pd);
```

### – La biblioteca PCSerialUSB.h

- Desde el PC se deben seguir los mismos pasos que en Arduino:
  - Abrir el puerto de comunicaciones a una velocidad.
  - Leer o escribir.

```
/**
 * Función para enviar una cadena de caracteres por un descriptor de puerto USB
 * @param pd Descriptor del puerto USB a utilizar
 * @param Buffer a enviar
 */
bool sendUSB(int &pd, char *data);

/**
 * Función para recibir datos desde USB. La función bloquea la ejecución del programa hasta que
 * se ha recibido al menos 1 byte por USB.
 * @param pd Descriptor del puerto USB a utilizar para recibir datos.
 * @param data Buffer de salida. Debe contener memoria para almacenar los datos que se recibirán.
 */
bool receiveUSB (int &fd, char *data);
```

- El programa **pcecho**: Fichero **src/pcecho.cpp**
- El programa principal realizará las siguientes operaciones:
  1. Abrir y configurar el puerto de comunicaciones USB.
  2. Pedir datos por entrada estándar y enviarlos a Arduino.
  3. Esperar a que Arduino responda con datos, y mostrarlos por pantalla.
  4. **Cerrar el puerto de comunicaciones USB** y salir.

### — El programa pcecho:

```
#include <PCSerialUSB.h>
```

```
int main() {
```

```
    int fd; // Descriptor de puerto
```

```
    char cadena[1000];
```

```
    fd= InicializarUSB("/dev/ttyACM0");
```

```
    if (fd<0) return 0;
```

```
    cin>>cadena;
```

```
    if (sendUSB(fd, cadena)) cout <<"Enviado\n";
```

```
    if (receiveUSB(fd, cadena))
```

```
        cout<<"He recibido: "<<cadena<<endl;
```

```
    CerrarUSB(fd);
```

```
    return 0;
```

```
}
```



UNIVERSIDAD  
DE GRANADA

# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

1. Instalación para las prácticas
2. ¿Qué es Arduino?
3. Compilando para AVR
4. La biblioteca ArduTIC
5. Comunicaciones en serie
- » 6. Dispositivos GPIO
7. Descripción de la práctica 1

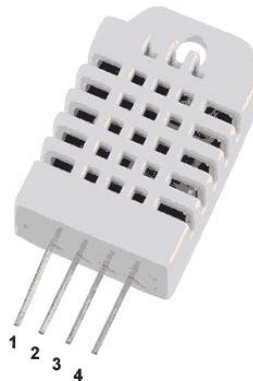


- **GPIO** → General Purpose Input/Output

Los pines de Arduino son GPIO, pines de entrada/salida de propósito general.

**Se utilizan por sensores y actuadores para comunicarse con la placa Arduino**

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND

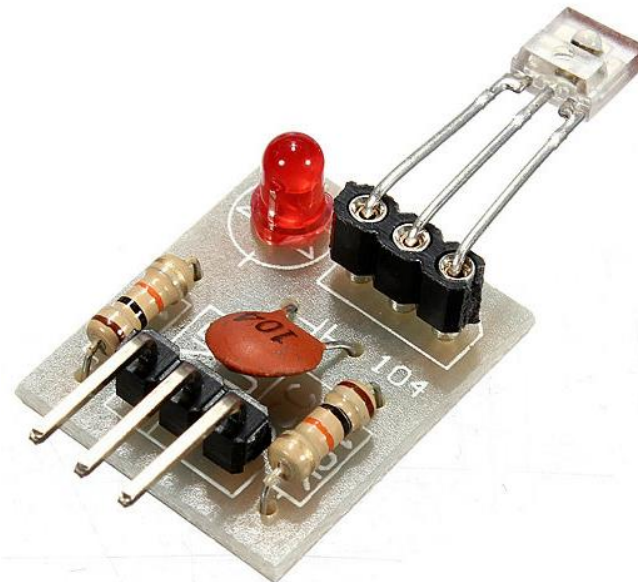


**Sensores:** Se configuran como INPUT con `pinMode(pin, INPUT)`

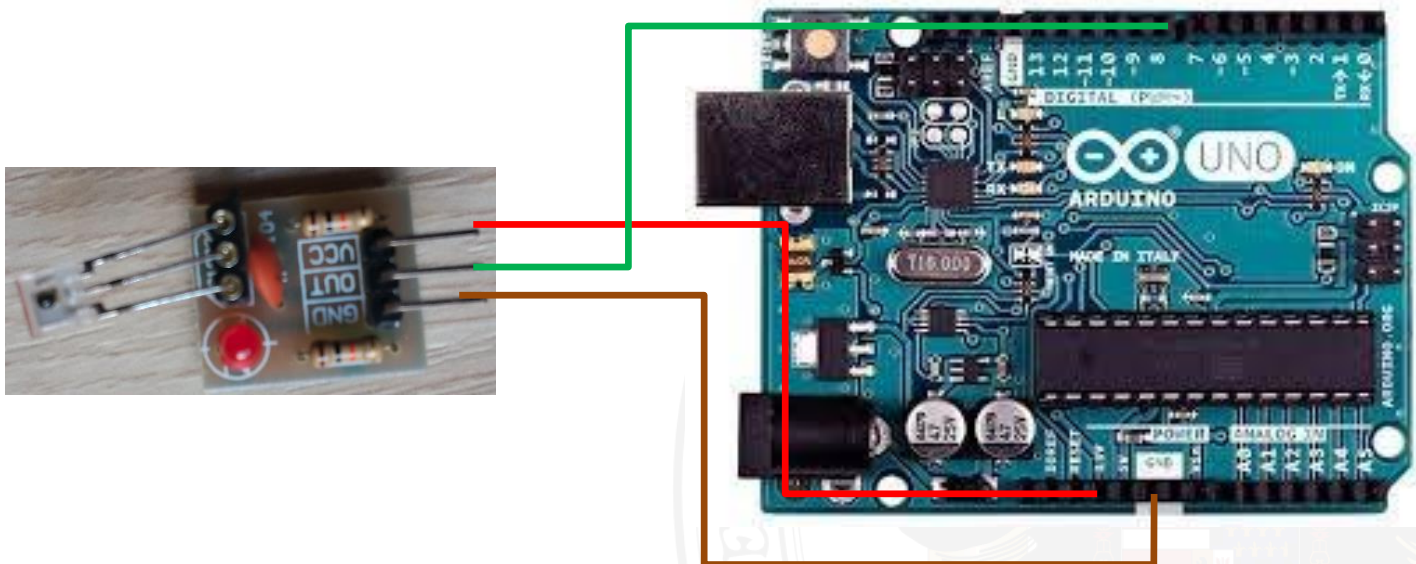
**Actuadores:** Se configuran como OUTPUT con `pinMode(pin, OUTPUT)`



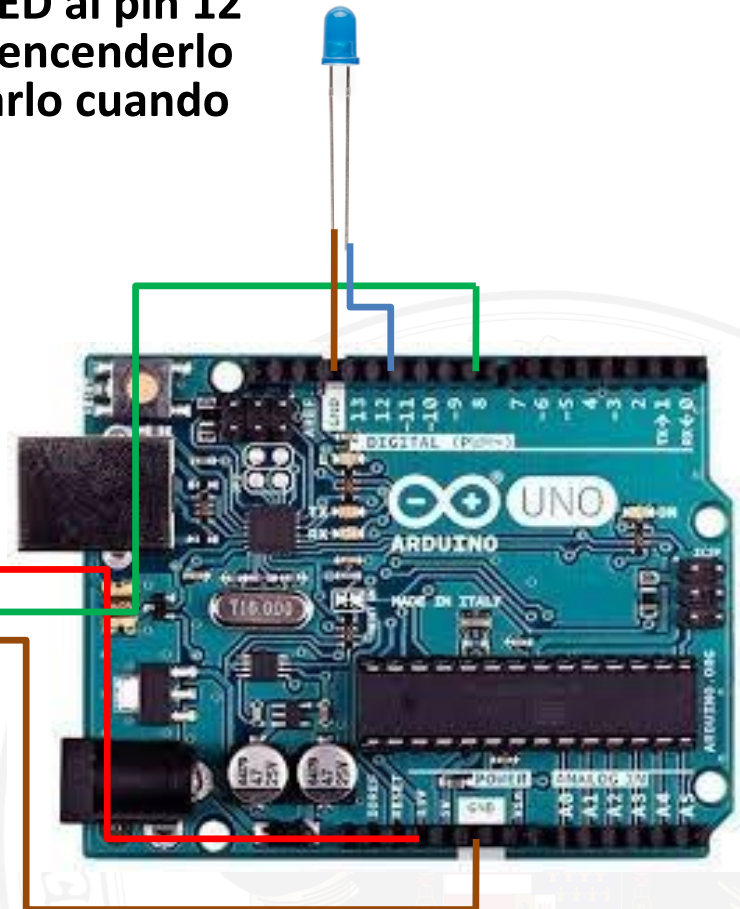
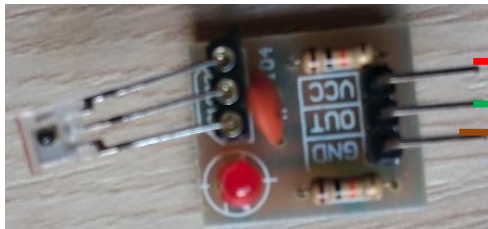
- Para construir los prototipos de emisión/recepción de información mediante láser, utilizaremos **fotorreceptores** capaces de captar la luz emitida mediante láser.
- El **Módulo receptor 2PCS** láser no modulador sensor Tubo es un fotorresistor que responde a la luz ambiental existente.
- Es un sensor que se conecta a un puerto digital, a un pin GPIO.
- Funciona a 3.3V.



- **Módulo receptor 2PCS** láser no modulador sensor Tubo. **Conexión con la placa:**
  - Utilizando cables de conexión de pines Macho-Hembra, conectaremos la patilla GND del sensor a tierra en la placa Arduino, la patilla VCC a la alimentación de 3.3V de la placa, y la salida del sensor VOUT al pin 8 de E/S digital:



- También conectaremos un LED al pin 12 de E/S digital Arduino, para encenderlo cuando se reciba luz y apagarlo cuando no se reciba.



- El programa **fotorreceptor** de Arduino Uno:

```
#include <ArduTIC.h>
```

```
int main() {
```

```
    // Configurar como entrada el Pin 8 de la placa.
```

```
    // Configurar como salida el Pin 12 de la placa.
```

```
    while (1) {
```

```
        // DigitalRead del pin 8
```

```
        // Si se devolvió HIGH
```

```
            // DigitalWrite HIGH al pin 12
```

```
        // En otro caso
```

```
            // DigitalWrite LOW al pin 12
```

```
    }
```

```
    Return 0;
```

```
}
```

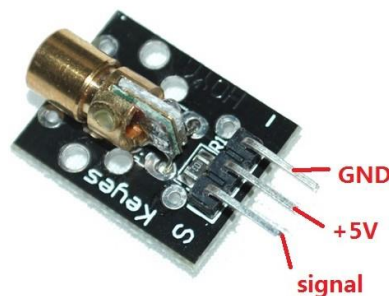
- El makefile simple del programa **fotorreceptor**:
- Ejercicio: Terminar el makefile para incluir la biblioteca ArduTIC

```

2
3 fotorreceptor:
4     @echo Compilando fotorreceptor...
5     @avr-gcc -Os -mmcu=atmega328p -c -o fotorreceptor.o fotorreceptor.cpp
6     @avr-gcc -mmcu=atmega328p fotorreceptor.o -o fotorreceptor.bin
7     @avr-objcopy -O ihex -R .eeprom fotorreceptor.bin fotorreceptor.hex
8     @echo Programa fotorreceptor compilado. Ejecute make sendfotorreceptor para enviarlo a Arduino.
9
10 sendfotorreceptor:
11     sudo avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:fotorreceptor.hex
12
13 clean:
14     rm -f -r *~
15     rm -f -r *.o
16     rm -f -r *.hex
17     rm -f -r *.bin|

```

- Para construir los prototipos de emisión/recepción de información mediante láser, utilizaremos **emisores láser** capaces de emitir luz en el rango visible con una longitud de onda de 650nm.
- El **Módulo emisor láser KY-008** es un emisor adecuado a nuestras necesidades.
- Es un emisor que se conecta a un GPIO de un puerto digital.
- Funciona a 5V.





- **IMPORTANTE:** El emisor láser que se utiliza en las prácticas tiene una longitud de onda relativamente alta (650nm), por lo que no produce daños erosivos ni quemaduras. Sin embargo, **el láser no se debe orientar directamente a los ojos, pues estos sí pueden ser dañados** ante un estímulo de estas características.







UNIVERSIDAD  
DE GRANADA

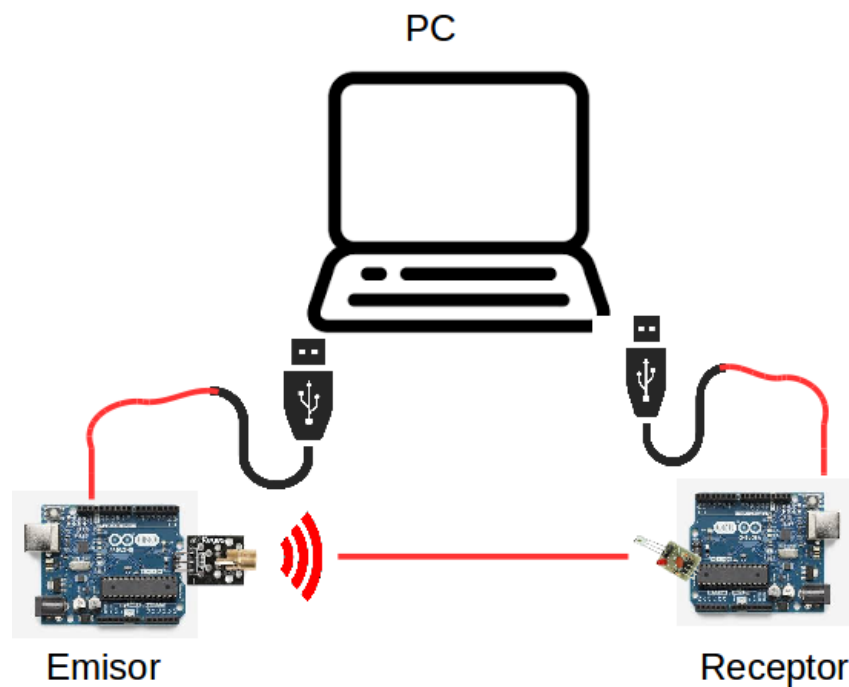
# Teoría de la Información y la Codificación

Grado en Ingeniería Informática

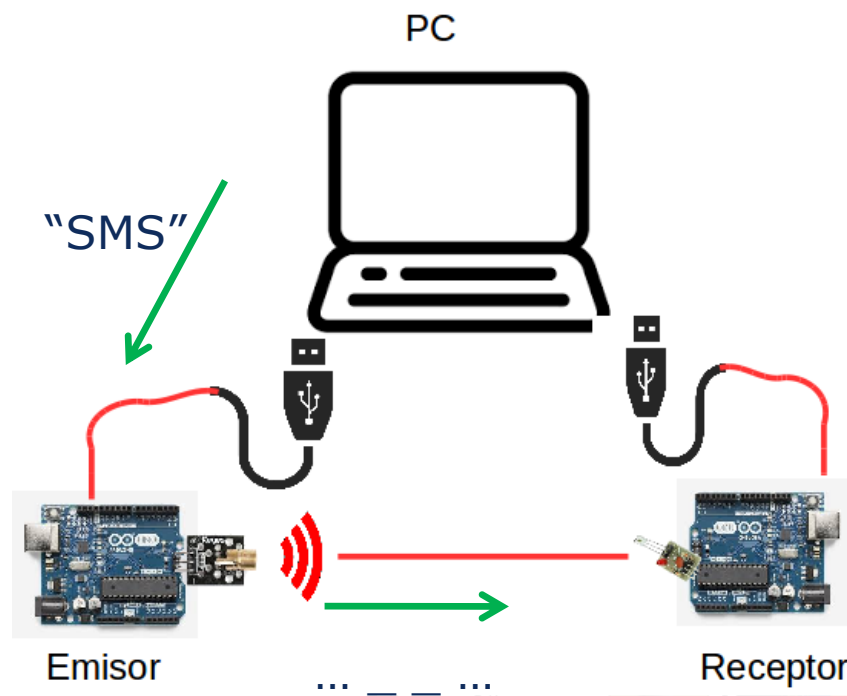
1. Instalación para las prácticas
2. ¿Qué es Arduino?
3. Compilando para AVR
4. La biblioteca ArduTIC
5. Comunicaciones en serie
6. Dispositivos GPIO
- » 7. Descripción de la práctica 1



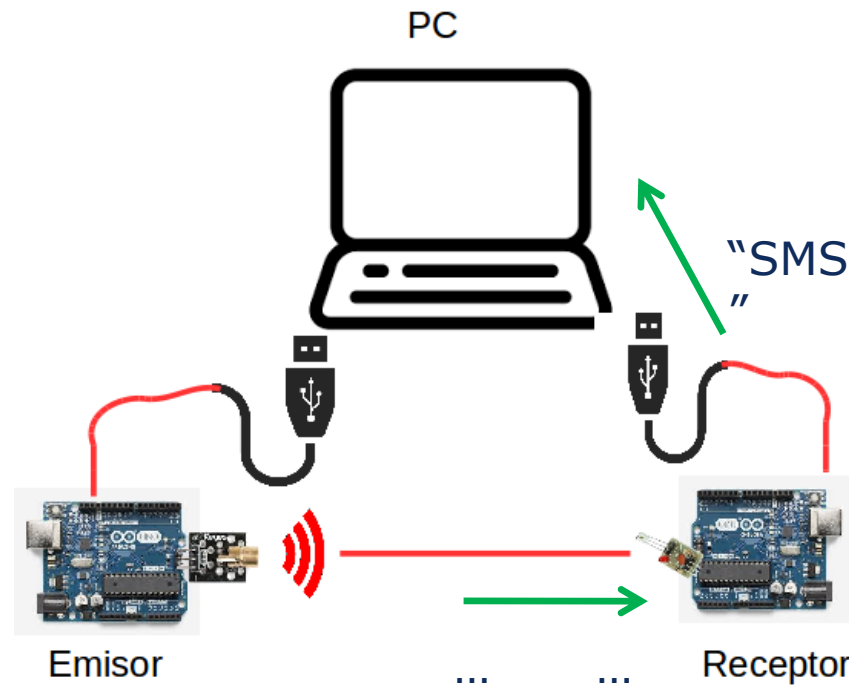
- **Utilizaremos 2 placas Arduino Uno:** Una como emisor, otra como receptor.
- Tanto el emisor como el receptor tendrán comunicaciones por USB con un PC (puede ser el mismo o diferente).



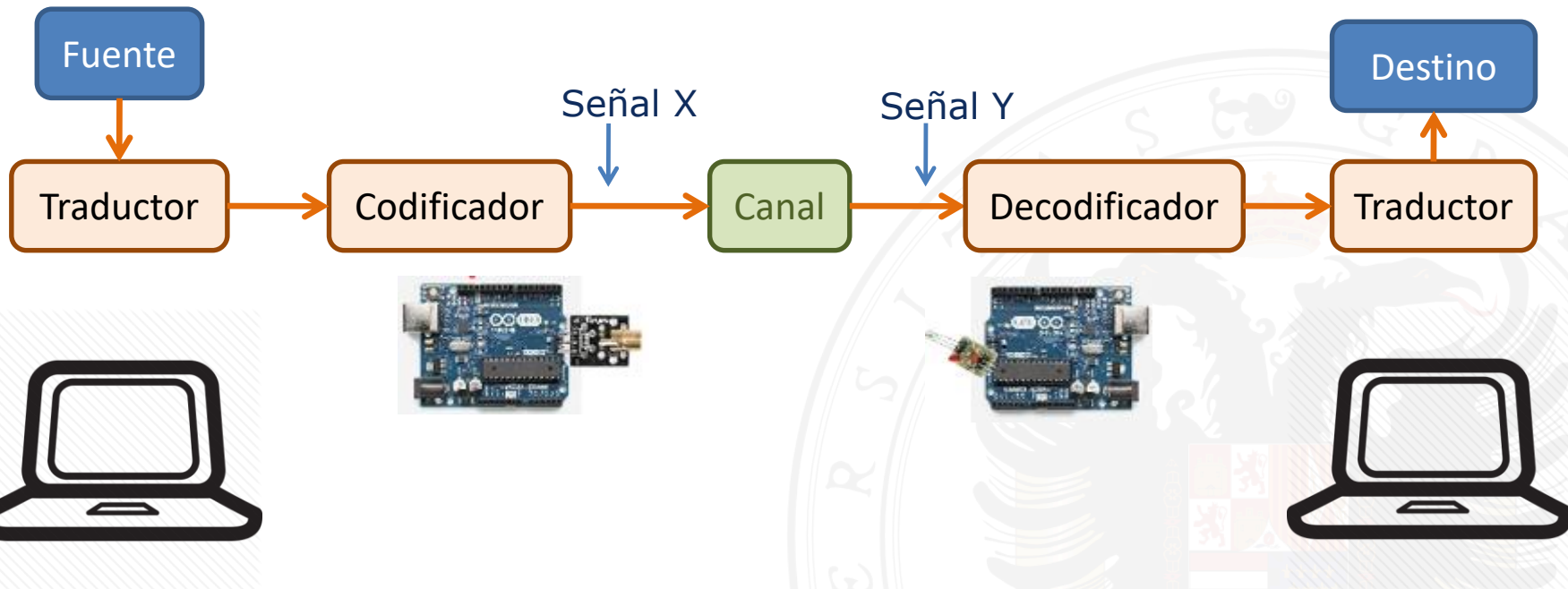
- **Mediante el puerto USB**, el PC del emisor enviará cadenas de datos al **Arduino emisor**.
- Este las codificará en señales luminosas, que serán transmitidas por el láser.



- **El Arduino receptor**, a través del fotorresistor, recibirá ráfagas luminosas desde el emisor láser, que interpretará y decodificará, transformándolas a símbolos ASCII.
- **Mediante el puerto USB**, el **Arduino receptor** enviará cadenas de caracteres con los símbolos ASCII decodificados del mensaje recibido.



### – Modelo de comunicaciones:



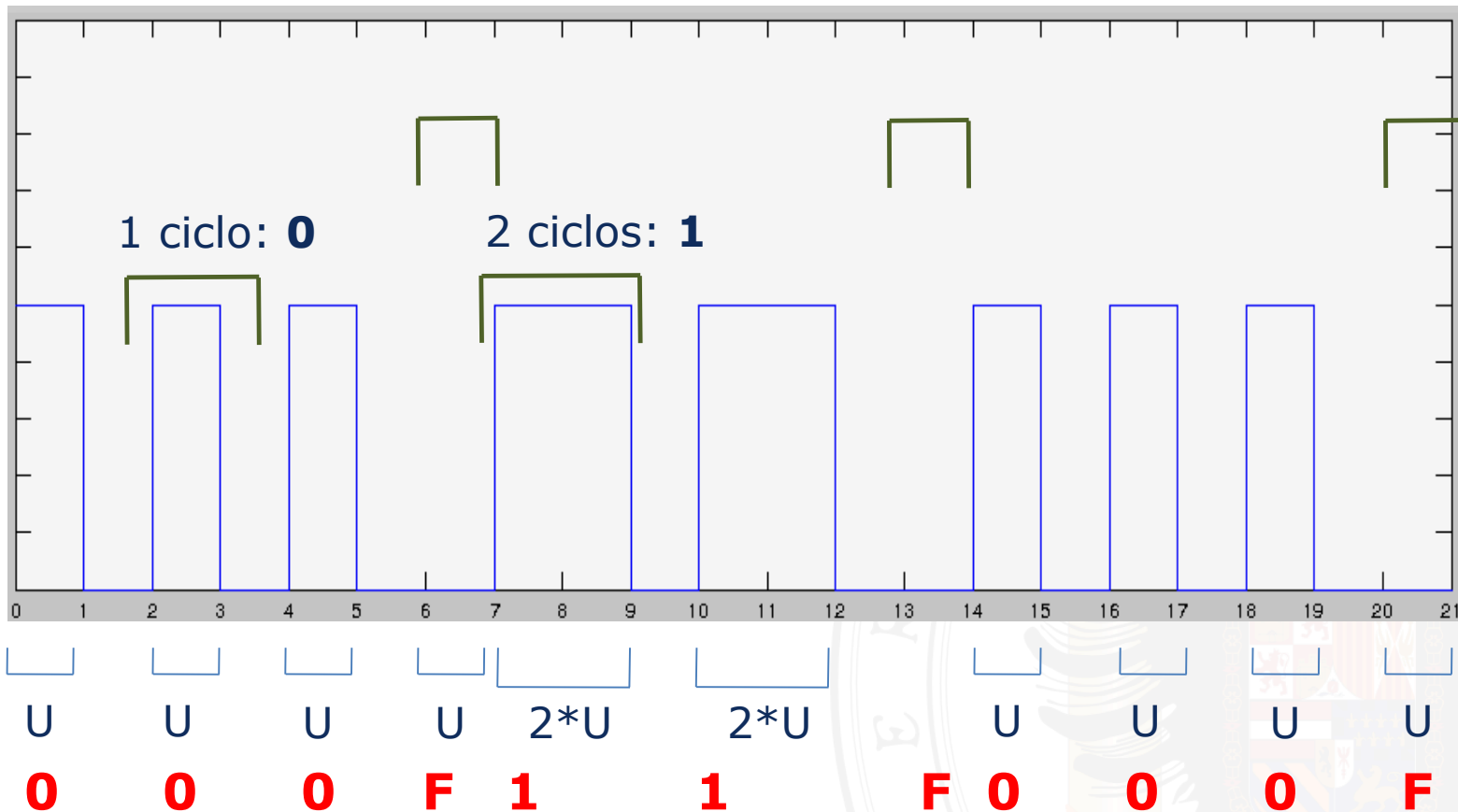
- **La fuente: cin**, dentro del programa del PC del emisor
- **El destino: cout**, dentro del programa del PC del receptor
- **Traductor (fuente)**: El programa del PC del emisor. Se encargará de codificar los caracteres ASCII dados por la fuente en un código inventado (secuencias de bits), y de enviarlos al codificador (Arduino).
- **Traductor (destino)**: El programa del PC del receptor. Se encargará de decodificar el código inventado (secuencias de bits) en caracteres ASCII.
- **Codificador (fuente)**: El programa del Arduino del emisor. Se encargará de recibir secuencias de bits por USB, transformarlas al **código del canal**, y emitir las por el canal (láser).
- **Decodificador (fuente)**: El programa del Arduino del receptor. Se encargará de **muestrear el canal** para recibir secuencias de datos (fotorreceptor), transformarlas del código del canal al código usado por el receptor, y enviarlas por USB

- En el canal tendremos un alfabeto formado por:
  - **Ráfagas cortas**: Asociadas con el símbolo binario 0.
  - **Ráfagas largas**: Asociadas con el símbolo binario 1.
  - **Sin emisión**: Fin de transmisión del símbolo.
  
- **Al codificar un “0”, se enviará HIGH LOW en 2 ciclos de reloj**
- **Al codificar un “1”, se enviará HIGH HIGH LOW en 3 ciclos de reloj**
  
- **Es necesario conocer a priori la duración de los ciclos de reloj (Umbral U).**



- **Limitaciones de esta alternativa:**
  - Todas las ráfagas deben obligatoriamente finalizar con voltaje bajo. Todos los ciclos duran un tiempo  $U$ .
- **Ráfaga corta (2 ciclos):**
  - En el primer ciclo de reloj, se envía señal luminosa.
  - En el segundo, no se envía señal luminosa.
- **Ráfaga larga (3 ciclos):**
  - En el primer ciclo de reloj, se envía señal luminosa.
  - En el segundo, también.
  - En el tercero, no se envía señal luminosa.
- **Fin de símbolo (1 ciclo):**
  - Ciclo único: No se envía señal luminosa.

Un ejemplo de una transmisión de varios mensajes (F=Fin del mensaje):



### – Parada estratégica (I):

- Llegados a este punto, conviene que diferenciamos correctamente entre dos términos:
  - Los **bits usados para codificar el símbolo que se desea enviar.**
  - Los **bits que realmente se transmiten por el medio físico.**
- En el ejemplo anterior, para transmitir un “0” del código, se han necesitado transmitir 2 ciclos por láser: Uno en voltaje alto y otro en voltaje bajo.
- Para transmitir un “1” del código, se han necesitado transmitir 3 ciclos por láser: Dos en voltaje alto y uno en voltaje bajo.
- Para evitar confusiones, en el resto de las prácticas de la asignatura nos referiremos a los bits de cada nivel de abstracción de la siguiente forma:
  - **codeBit:** Un bit de un código a transmitir. Los notaremos como 0 y 1
  - **laserBit:** Un dato transmitido en un ciclo por el medio físico. Los notaremos como H y L

### – Parada estratégica (II):

- Por tanto, para transmitir en el ejemplo los ***codeBits de 3 mensajes 000.11.000.***, se han necesitado un total de **21 *laserBits***:

**HLHLHLLHHLHLLHLHLHLL**

### – Estos ***laserBits*** se han utilizado para:

- Enviar 000 y el fin del mensaje (7 *laserBits* para 3 *codeBits*)
- Enviar 11 y el fin del mensaje (7 *laserBits* para 2 *codeBits*)
- Enviar 000 y el fin del mensaje (7 *laserBits* para 3 *codeBits*)

**Usaremos esta nomenclatura durante el resto de las prácticas, para conocer si nos estamos refiriendo a bits de códigos o a bits de transmisión de datos.**

### – La práctica. Paso 1. El programa emisorPC.cpp

1. Inventarse un código uniforme que permita codificar las letras del alfabeto en MAYÚSCULA, junto con el espacio y unos símbolos de puntuación más. **Máximo: 31 caracteres.**  
     'A' → "01101"  
     'B' → "10110"  
     ... etc.
2. Implementar el código en un programa en C.
3. Ampliar el programa pidiendo una cadena válida por teclado, de longitud máxima 100 (**NOTA: Es más cómodo que el programa transforme de minúsculas a mayúsculas automáticamente con toupper o similar**).
4. Transformar la cadena anterior a una cadena de caracteres compuesta por "0"s y "1"s de longitud máxima 500.
5. Enviarla por USB a Arduino (NOTA: Se puede probar con el programa ArduEcho para ver que Arduino la recibe correctamente).

### – La práctica. Paso 2. El programa receptorPC.cpp

1. Implementar el código diseñado para el emisor en otro programa en C, receptorPC.cpp.
2. Recibir una cadena de caracteres de “0”s y “1”s desde USB.
3. Decodificar cada secuencia de “0”s y “1”s según el código diseñado, e ir guardando los caracteres decodificados en una cadena.

“01101” → ‘A’

“10110” → ‘B’

... etc.

4. Mostrar el código por consola.

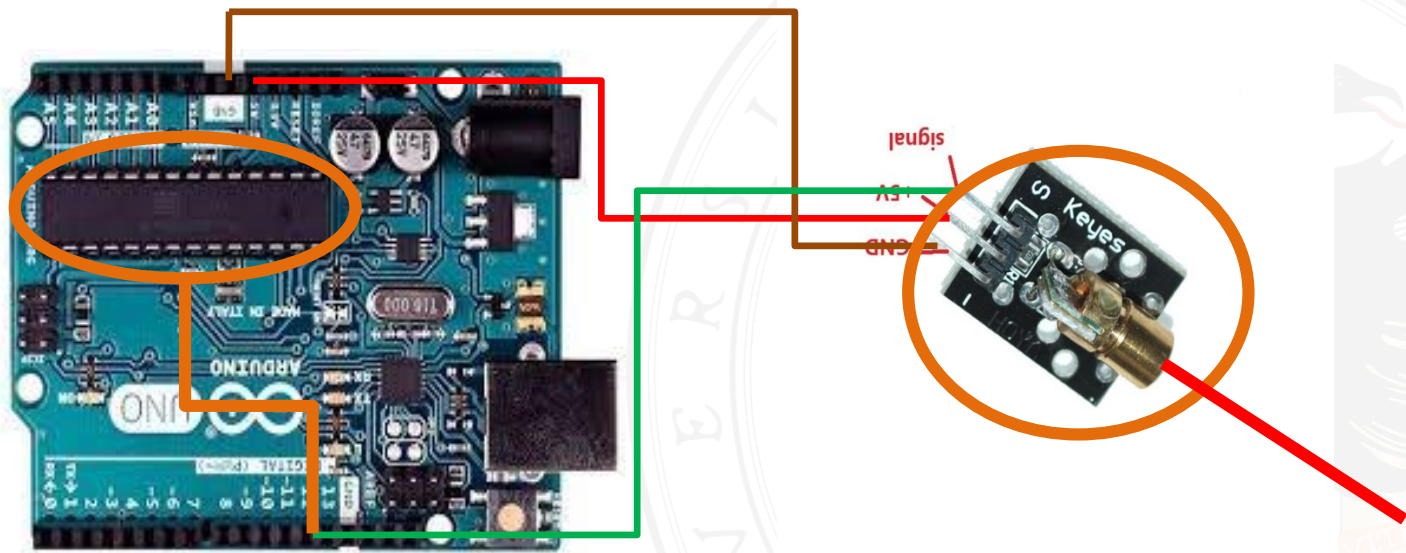
**(NOTA: Es más cómodo implementar un programa para Arduino que nos envíe constantemente cadenas de “0”s y “1”s prediseñadas, que el programa receptor PC pueda recoger).**

### – La práctica. Paso 3. El programa emisorArdu.cpp

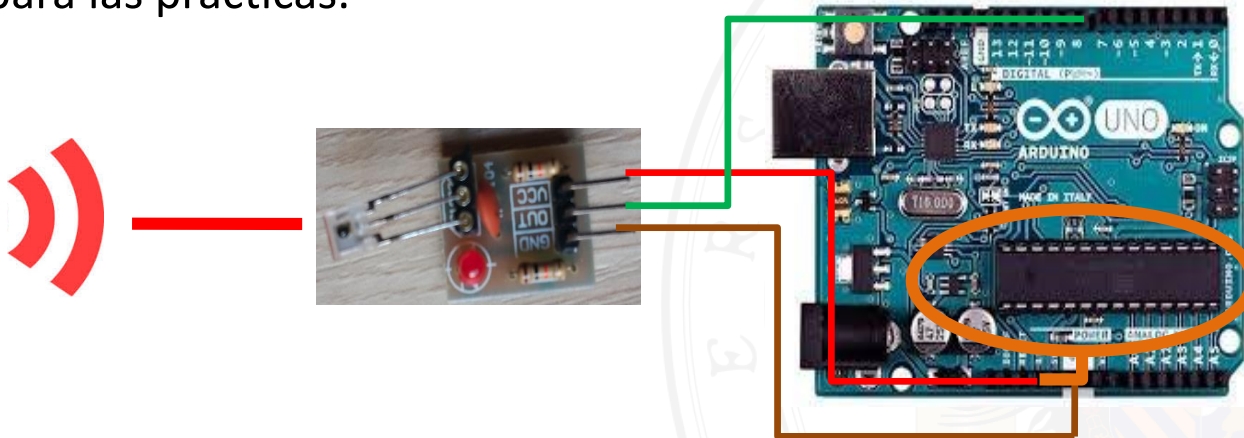
1. Seleccionar un umbral  $U$  (se recomienda mínimo 20 ms).
2. Implementar un programa en C para Arduino que, en un bucle infinito:
  1. Recoja cadenas de caracteres por USB (si hay)
  2. Recorra la cadena (si hay). Para cada componente:
    1. Si es un '0': Poner el láser a H durante  $U$  ms
    2. Si es un '1': Poner el láser a H durante  $2*U$  ms
    3. Finalmente, Poner el láser a L durante  $U$  ms



- El valor apropiado para  $U$  conlleva considerar:
  - El tiempo que tarda el microprocesador en procesar la orden
  - + el tiempo que se tarda en activar el voltaje en el pin
  - + el tiempo que el láser tarda en realizar su carga y encenderse
  - + el tiempo de transmisión por el canal
  - + la desincronización existente con el receptor.
  - + la ejecución de otras sentencias en el programa.
  - + el error en la función delay para efectuar la espera del envío.



- El valor apropiado para  $U$  conlleva considerar:
  - Además, es necesario tener en cuenta cuánto tiempo tarda el fotorreceptor en detectar la luz, y cuánto tiempo se requiere para procesarla en el Arduino receptor.
  - Teorema de Muestreo de Nyquist: Se debe muestrear a un periodo  $T$  de, como mucho  $U/2$  instantes de tiempo. Valores más seguros serían  $U/3$ ,  $U/4$ , ...
- A menor valor de  $U$ , más velocidad de transmisión, pero mayor posibilidad de errores (sobre todo si el código es ineficiente).
- Utilizaremos un valor no inferior a  **$U=15\text{ms}$ , y múltiplo de 3**, suficiente para las prácticas.

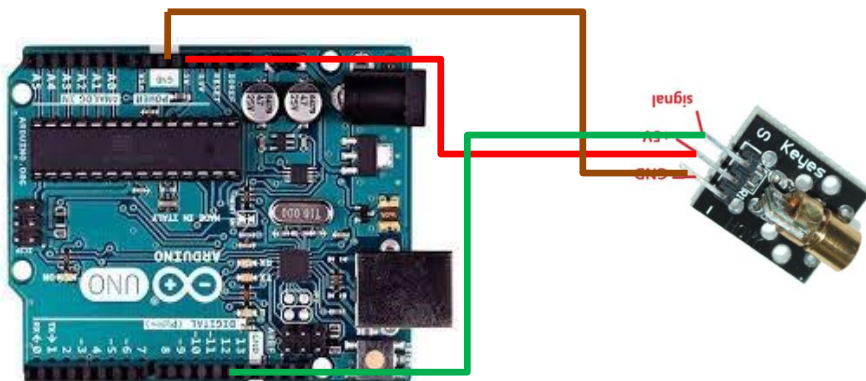


### – La práctica. Paso 4. El programa receptorArdu.cpp

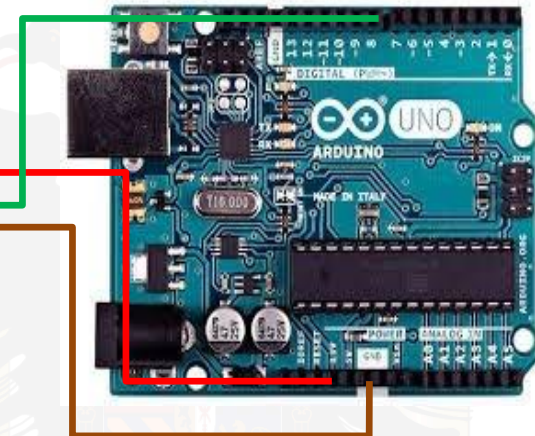
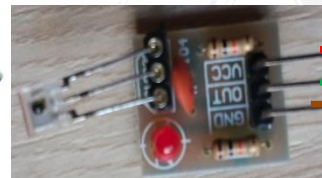
1. Seleccionar el mismo umbral  $U$  que en el emisor.
2. Seleccionar un periodo de muestreo  $T$  tal que  $U\%T \approx 0$
3. Implementar un programa en C para Arduino que, en un bucle infinito:
  1. Inicializar cadena  $\leftarrow ""$
  2. Mientras no se reciba nada por láser, no hacer nada.
  3. Mientras no acabe la transmisión
    1. Contador= 0
    2. Mientras el láser esté en alta
      1. Muestrear la señal del láser cada  $T$
      2. Contador++
    3. Si contador  $\geq U/T \rightarrow$  Se recibe un '1'
    4. En otro caso, se recibe un '0'
    5. Guardar el carácter '0' ó '1' recibido al final de cadena
    6. Comprobar si se acaba la transmisión
4. Enviar cadena por USB al receptor

### – El esquema de montaje:

- Necesitaremos 2 placas Arduino con las conexiones como se muestran en la figura (**fotorreceptor: Pin 8; emisor láser: Pin 12**):



**Placa emisora**



**Placa receptora**

- Debemos compilar:
  - El receptor de Arduino: **make receptorArdu**
  - El emisor de Arduino: **make emisorArdu**
  - El receptor en el PC: **make receptorPC**
  - El emisor en el PC: **make emisorPC**
  - Enviar el receptor: **make sendReceptor**
  - Enviar el emisor: **make sendEmisor**
  - Hacerlo todo: **make all**
- También debemos asegurarnos de que:
  - El receptor se encuentra en una zona sin iluminación (o muy poca)
  - El emisor láser está alineado con el fotorreceptor, para que este pueda recibir la información

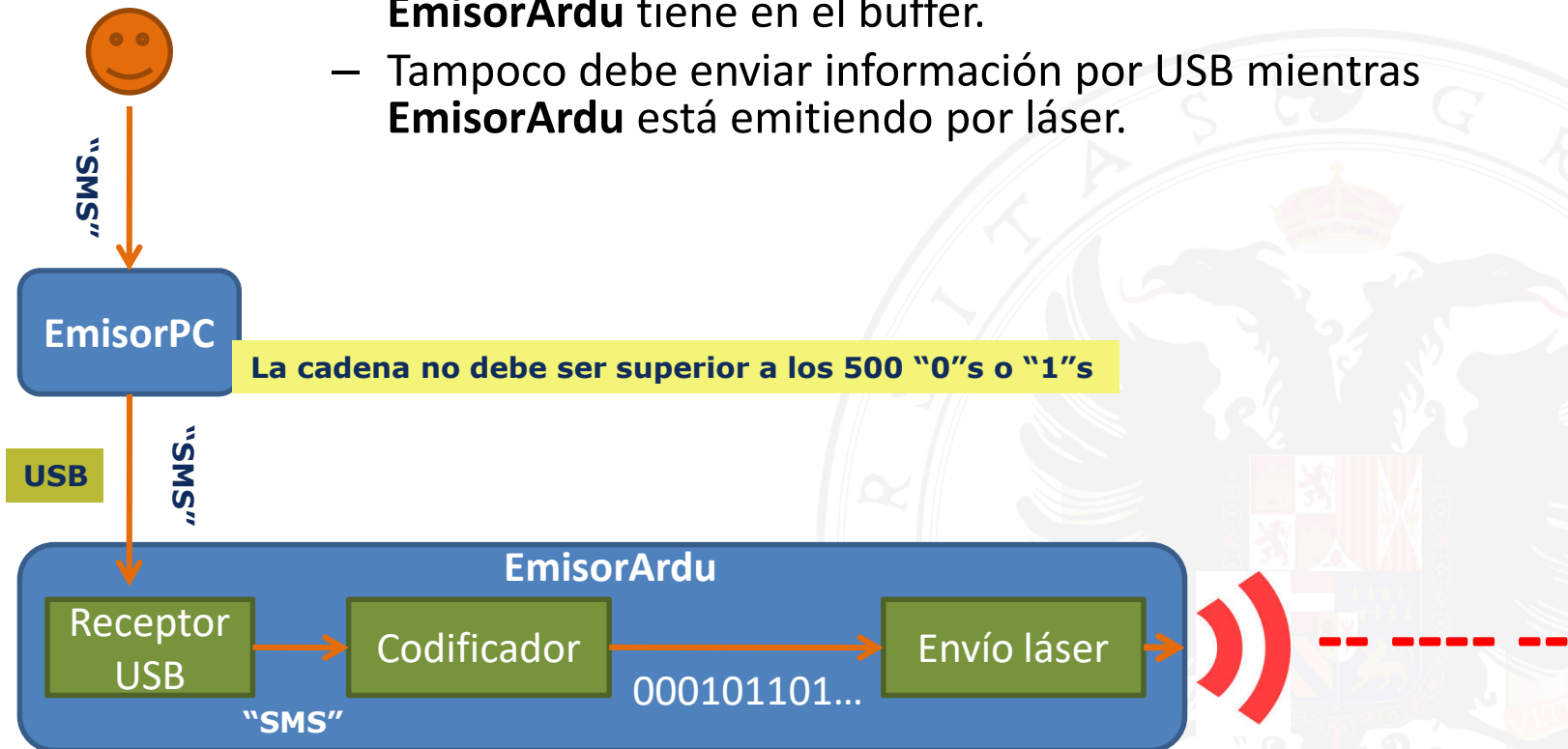
- El **proyecto final de la práctica** consiste en: Construir un sistema compuesto por 4 programas o aplicaciones:
  - **EmisorPC:** programa que envía un texto a una placa emisora Arduino Uno.
  - **EmisorArdu:** programa que recibe un texto por USB, lo codifica y lo envía a través de láser.
  - **ReceptorArdu:** programa de una placa receptora Arduino Uno que recibe un mensaje a través del fotorreceptor láser, lo decodifica en texto y lo envía por USB.
  - **ReceptorPC:** programa que recibe un mensaje de texto por USB y lo muestra por pantalla.



- Para realizar el proyecto final, es necesario considerar que todas las aplicaciones deben estar bien coordinadas para obtener los resultados esperados:

### Entrada estándar

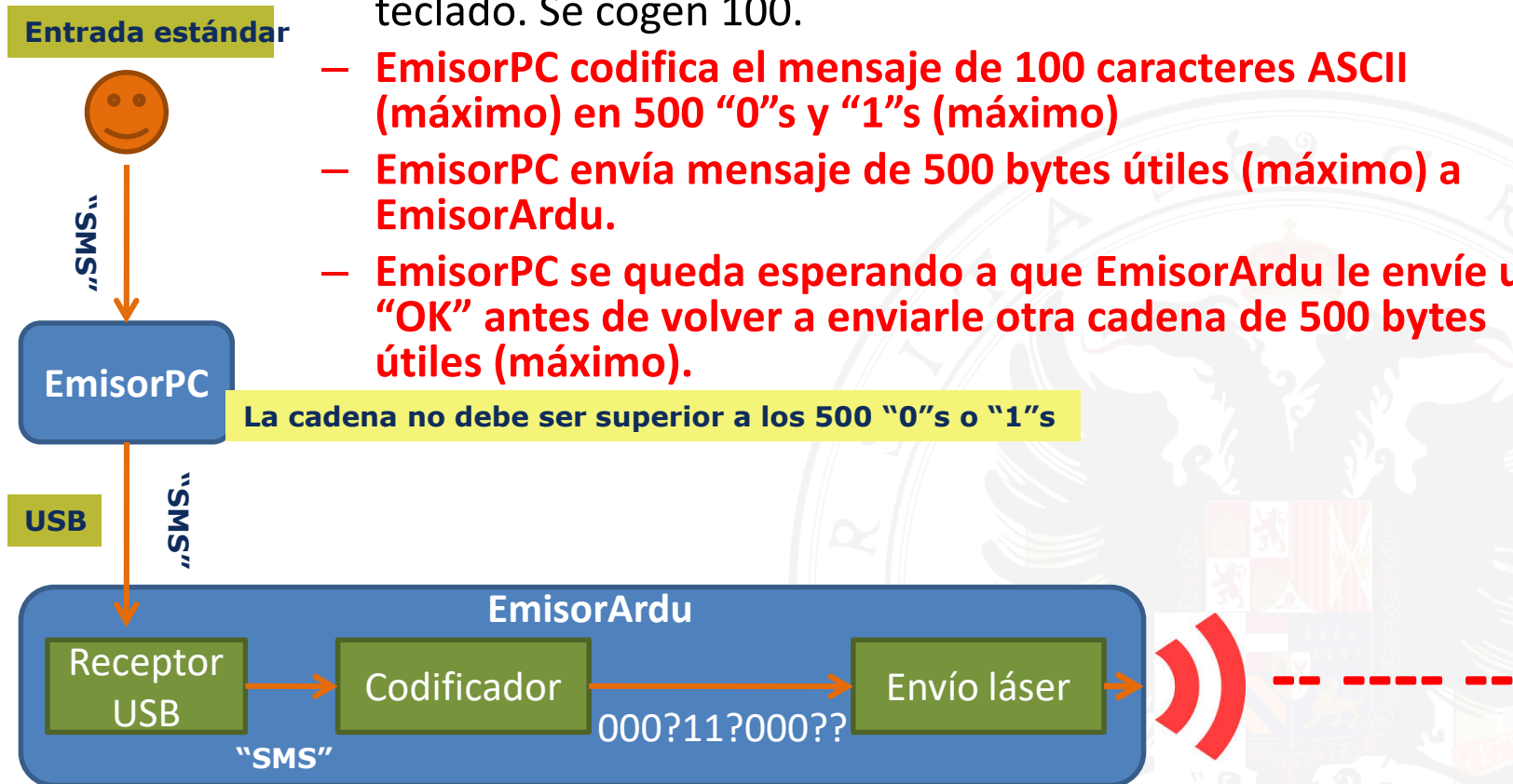
- El **EmisorPC** no debe enviar más información de la que el **EmisorArdu** tiene en el buffer.
- Tampoco debe enviar información por USB mientras **EmisorArdu** está emitiendo por láser.





### – Cómo coordinar el EmisorPC con el EmisorArdu:

- Inicialmente, **EmisorArdu** se encuentra esperando un mensaje por USB. El buffer será de 500 caracteres **(+ otro por '\0')**
- **EmisorPC**, por su parte, se encuentra esperando datos por teclado. Se cogen 100.
- **EmisorPC codifica el mensaje de 100 caracteres ASCII (máximo) en 500 "0"s y "1"s (máximo)**
- **EmisorPC envía mensaje de 500 bytes útiles (máximo) a EmisorArdu.**
- **EmisorPC se queda esperando a que EmisorArdu le envíe un "OK" antes de volver a enviarle otra cadena de 500 bytes útiles (máximo).**



- Cómo coordinar el EmisorArdu con el ReceptorArdu:
- Vamos a ver dos ejemplos.
- Supongamos que tenemos un alfabeto de 4 símbolos “A”, “B”, “C”, y “D”.
- Asumiremos que el codificador realiza las siguientes codificaciones sobre el alfabeto:

Símbolo	Codificación
“A”	00
“B”	01
“C”	10
“D”	11

- En el ejemplo 1, enviaremos sólo el mensaje de la cadena “C”.
- En el ejemplo 2, enviaremos el mensaje de la cadena “AB”.

- Ejemplo de coordinación para el envío de “C”
- Envío de datos:
  - Cadena enviarUSB ← “”
  - Se lee “C” desde teclado
  - Se codifica “C” como “10”, y se añade a enviarUSB
  - Se envía enviarUSB a Arduino
    - Arduino emisor está esperando a que llegue algo por USB
    - Arduino emisor recibe cadena ← “10” por USB
    - Envía HHL por láser, en ciclos de tiempo U, para el “1”
    - Envía HL por láser, en ciclos de tiempo U, para el “0”
    - Envía “OK” por USB al emisor
    - Vuelta a empezar (código de Arduino)
  - Se recibe “OK” desde Arduino
  - Vuelta a empezar (código del emisorPC)

– **Mientras tanto...**



- Ejemplo de coordinación para el envío de “C”
- Recepción de datos:
  - Arduino receptor hace cadena ← “”
  - Arduino receptor está esperando mientras digitalRead=LOW
  - Contador=0
  - Mientras no acabe el mensaje:
    - Muestrea cada T, mientras sea digitalRead=HIGH
    - Contador++
  - Si contador ≥ U/T, añadir “1” a cadena, en otro caso añadir “0”
  - Contador=0
  - Mientras digitalRead=LOW y contador < U/T
    - Contador++
    - Esperar T ms
  - Si Contador ≥ U/T, entonces fin del mensaje:
    - Enviar cadena por USB
    - Volver al paso 1
  - En otro caso, Volver al paso 3

**Recibe HHL HL L**

**Codifica “10”**

– Mientras tanto...



- Ejemplo de coordinación para el envío de “C”
- Recepción de datos (2):
  - Receptor PC está esperando a recibir algo por USB
  - Recibe “10”
  - Busca el código al que corresponde “10” → ‘C’, y lo mete en un buffer
  - Ya se ha acabado la cadena, la muestra por consola
  - Vuelta a empezar

**Decodifica “10” → ‘C’**

**Muestra “C”**



- Otro ejemplo de coordinación para el envío de “AB”
- Envío de datos:
  - Cadena enviarUSB ← “”
  - Se lee “AB” desde teclado
  - Se codifica “A” como “00”, y se añade a enviarUSB
  - Se codifica “B” como “01”, y se añade a enviarUSB
  - Se envía enviarUSB a Arduino
    - Arduino emisor está esperando a que llegue algo por USB
    - Arduino emisor recibe cadena ← “0001” por USB
    - Envía HL por láser, en ciclos de tiempo U, para el “0”
    - Envía HL por láser, en ciclos de tiempo U, para el “0”
    - Envía HL por láser, en ciclos de tiempo U, para el “0”
    - Envía HHL por láser, en ciclos de tiempo U, para el “1”
    - Envía “OK” por USB al emisor
    - Vuelta a empezar (código de Arduino)
  - Se recibe “OK” desde Arduino
  - Vuelta a empezar (código del emisorPC)

– Mientras tanto...



- Ejemplo de coordinación para el envío de **“AB”**
- **Recepción de datos:**
  - Arduino receptor hace cadena ← “”
  - Arduino receptor está esperando mientras digitalRead=LOW
  - Contador=0
  - Mientras no acabe el mensaje:
    - Muestrea cada T, mientras sea digitalRead=HIGH
    - Contador++
  - Si contador ≥ U/T, añadir “1” a cadena, en otro caso añadir “0”
  - Contador=0
  - Mientras digitalRead=LOW y contador < U/T
    - Contador++
    - Esperar T ms
  - Si Contador ≥ U/T, entonces fin del mensaje:
    - Enviar cadena por USB
    - Volver al paso 1
  - En otro caso, Volver al paso 3

**Recibe HL HL HL HHL L**

**Codifica “0001”**

- Ejemplo de coordinación para el envío de **“AB”**
- Recepción de datos (2):
  - Receptor PC está esperando a recibir algo por USB
  - Recibe **“0001”**

**Decodifica “00”→‘A’**
  - Busca el código al que corresponde **“00”→‘A’**, y lo mete en un buffer
 

**Decodifica “01”→‘B’**
  - Busca el código al que corresponde **“01”→‘B’**, y lo mete al final del buffer
  - Ya se ha acabado la cadena, **la muestra por consola**
  - **Vuelta a empezar**

**Muestra “AB”**



Universidad de Granada

[decsai.ugr.es](http://decsai.ugr.es)

# **Teoría de la Información y la Codificación**

Grado en Ingeniería Informática

**Seminario 1.- Introducción a Arduino.**



**DECSAI**

**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**