PRÁCTICA 11: ENTRADA/SALIDA II

ENTRADA/SALIDA EN MICROCONTROLADOR

ARQUITECTURA DE COMPUTADORES. 2º CURSO

1. INTRODUCCIÓN y OBJETIVOS

Las sesiones prácticas que se han sucedido hasta ahora se han focalizado, en su mayoría, a trabajar con simuladores; ya que, a nivel de usuario, es difícil poder observar el funcionamiento a bajo nivel desde un computador de propósito general.

Únicamente en aquellas sesiones en las que se ha trabajado con códigos de alto nivel no se ha hecho uso de simulador, sino directamente con un entorno de programación sobre el sistema operativo.

Cuando trabajamos con los dispositivos de entrada/salida es más difícil esta apreciación a bajo nivel, entre otras cosas porque los sistemas operativos tienden a enmascarar los accesos a los periféricos (por motivos de seguridad, transparencia al usuario, etc.); es por ello que la primera práctica de E/S se llevó a cabo bajo un sistema operativo tipo Linux, ya que éstos permiten el acceso a las funciones del kernel del sistema y, por lo tanto, acceder al espacio de E/S.

En dicha sesión se trabajó con un periférico ubicado en el mapa de E/S (conexión de E/S separada), y accediendo a él mediante E/S programada (polling o encuesta).

En cuanto al tema de interrupciones, está más ligado a los drivers de los dispositivos externos. Es por ello que, para trabajar con interrupciones en un computador de propósito general, habría que desarrollar un driver al completo e integrarlo en el kernel del sistema. El tiempo y los conocimientos requeridos para ello suponen un lastre enorme si lo que pretendemos es únicamente mostrar un abanico con los diversos mecanismos de acceso a E/S.

Por todo lo anterior, en esta práctica se hará uso de una placa de circuito impreso (PCB) con un microcontrolador básico de 8 bits (de la familia ATMEL). El sistema utilizado forma parte de la comunidad Arduino, con lo que posee un conjunto de librerías con las que manejar los pines del micro de forma muy simple para el usuario. Sin embargo, nosotros **NO HAREMOS USO** de dichas funciones de alto nivel, sino que trabajaremos directamente con las direcciones físicas de los distintos registros que controlan los pines. El principal motivo de ello es ejercitar el manejo de estos dispositivos, ya que no todos los sistemas proveen al usuario de librerías de alto nivel.

A bajo nivel, este sistema posee la **E/S mapeada en memoria** y todas las direcciones definidas de antemano en constantes. Además, mantiene una estructura similar a la vista en teoría en cuanto a registros de acceso a los dispositivos externos. En este caso, los registros de E/S controlan un conjunto de pines, así que habrá que hacer uso en gran medida de máscaras y operadores de desplazamiento para controlar cada pin por separado.

La finalidad de esta práctica no será únicamente mostrar la E/S por Polling, sino que nos servirá para repasar el acceso a registros de E/S, así como trabajar con máscaras y operadores de desplazamiento. Tras ello, nos introduciremos de forma sencilla en las interrupciones para comparar el funcionamiento en ambos casos.

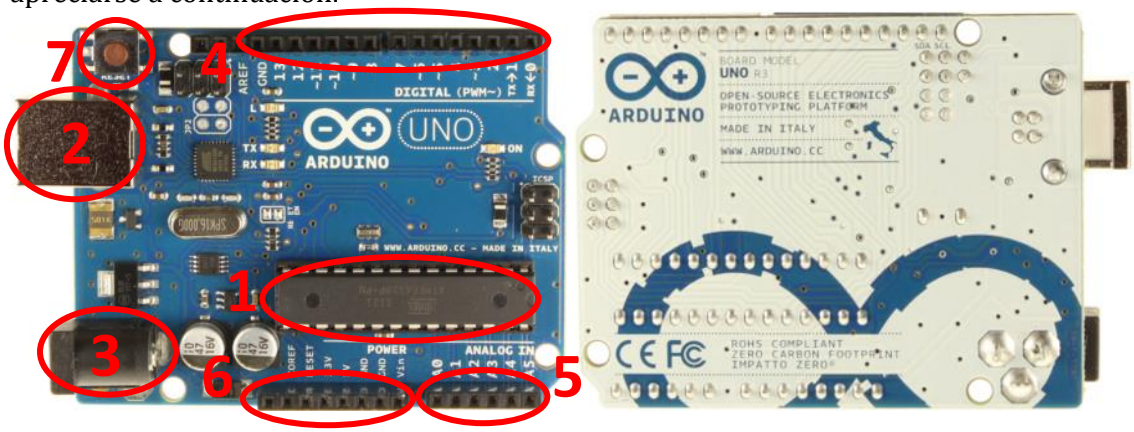
ESTUDIO PREVIO:

Lea detenidamente el siguiente boletín y **realice el estudio previo** del mismo. Para ello tendrá que comprender lo explicado a continuación, así como buscar información adicional no incluida aquí. Entregue el cuestionario impreso al comienzo de la sesión práctica.

2. DESARROLLO DE LA PRÁCTICA

Parte 1 – La PCB de Arduino

La PCB (Printed Circuit Board – Placa de Circuito Impreso) utilizada en esta práctica puede apreciarse a continuación:



En ella pueden apreciarse los siguientes elementos:

- (1) Microcontrolador ATMega328:

Voltaje operativo	5V
Corriente por GPIO	40 mA
Memoria Flash	32 KB. 0.5 KB utilizados para elbootloader. No volátil. Almacenamiento para el programa y variables con valores constantes.
SRAM	2 KB. Almacenamiento común para variables y pilas.
EEPROM	1 KB. No volátil. Almacenamiento de variables cuyo contenido se mantiene entre apagado y encendido pero que, a su vez, pueden ser modificadas.
Frecuencia de reloj	16 MHz

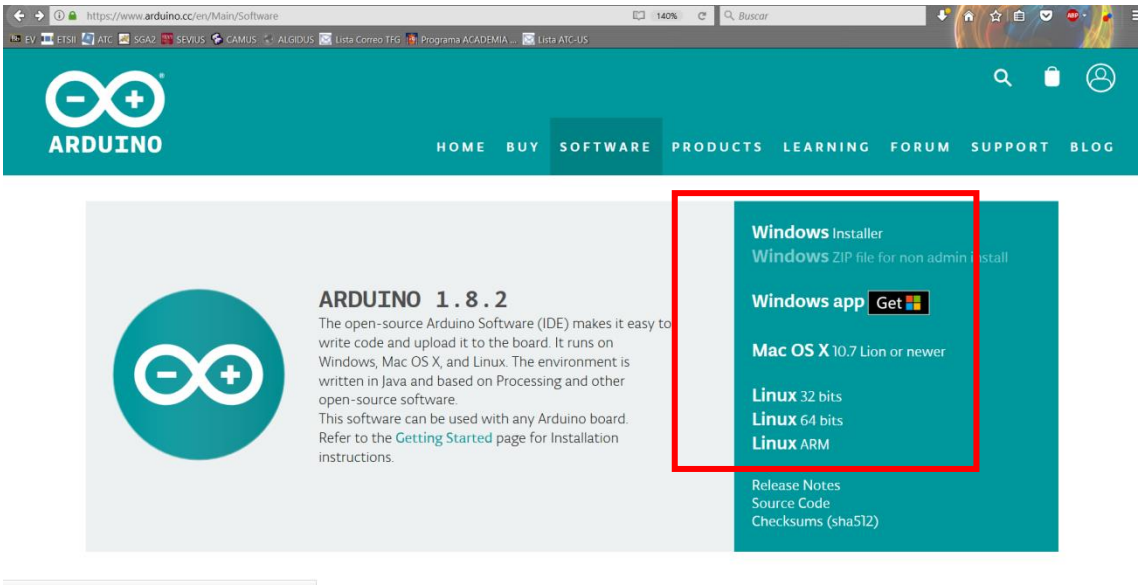
- (2) Conector USB tipo B: alimentación, programación y comunicación serie.
- (3) Alimentación externa: 5V.
- (4) 14 pines digitales numerados del 0 al 13: 6 de ellos están preparados para ser utilizados para señales PWM (pines 3, 5, 6, 9, 10 y 11) de 8 bits de resolución, 2 de ellos pueden utilizarse para comunicación serie (0: Rx, 1: Tx) anulando la comunicación serie USB.
- (5) 6 pines analógicos: cada uno conectado a un ADC de 10 bits de resolución.
- (6) Salida de alimentación (3.3 y 5V), tierras y reset externo.
- (7) Botón de reset.

Parte 2 – El entorno de programación

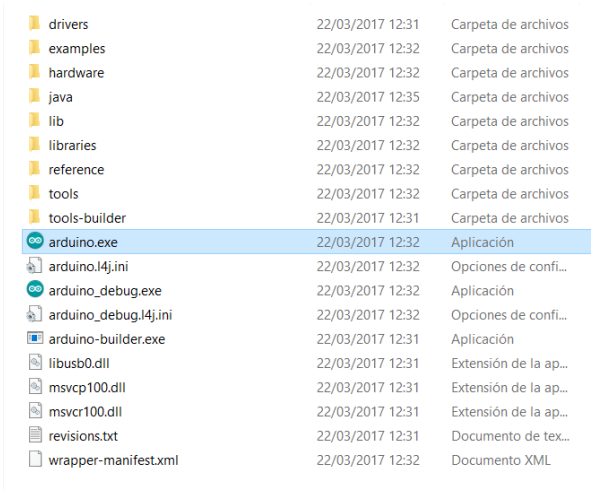
Existen varias posibilidades para desarrollar firmware para una placa Arduino. Por un lado, se puede hacer uso del entorno proporcionado por Arduino (http://arduino.cc/en/Main/Software), que comúnmente es el más utilizado. Sin embargo, la comunidad de Arduino permite otras alternativas como compilación web, plugins para diversos entornos de escritorio, etc.

El IDE de Arduino es un entorno de una simplicidad tan patente que obvia algunas herramientas y opciones muy útiles a la hora programar, como puede ser la división del programa en diversos ficheros fuente, el autocompletado o la depuración. Aun así, los códigos a utilizar en esta sesión son de una complejidad tal que no se precisarán utilidades avanzadas.

Para instalar el software de la comunidad Arduino, basta con acceder a su web y seguir las indicaciones: nos permiten tanto descargar el entorno para instalar (para cualquier sistema operativo), como hacer uso de una descarga comprimida y, de esta forma, evitamos la instalación (opción solo disponible para Windows).



Haciendo uso de la descarga sin instalación, obtendremos un archivo comprimido con el entorno correspondiente. De esta forma, podremos visualizar los siguientes ficheros en su interior:



Si ejecutamos el archivo sombreado (“arduino.exe”) arrancará el entorno de programación.

En el caso de la descarga con instalación, al completarse la misma aparecerá un acceso directo en el escritorio.

Parte 3 – Simulador Online

Para que sirva de utilidad a la hora de realizar pruebas y/o ejercitarse en la materia, son muchos los simuladores que se han desarrollado en base a las especificaciones de las placas de la comunidad de Arduino; integrando en ellos el montaje del circuito, el conexionado con la placa, la programación y la simulación de su ejecución.

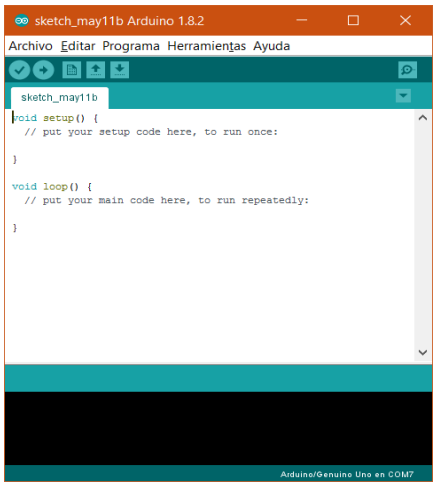
No es motivo de esta práctica hacer uso de simuladores, sino trabajar directamente con la placa. Aun así, para aquellos interesados, existe un simulador online gratuito (únicamente se precisa el registro en la plataforma) denominado TINKERCAD.

Es una herramienta gratuita de prototipado y simulación (no solo a nivel de microcontrolador, sino también de diseño 3D); es por ello que, para simular su circuito, debe introducirse en la sección “circuits” de la plataforma.

Parte 4 – “Proyectos” en Arduino IDE

Un proyecto en el IDE de Arduino consta únicamente de un fichero con extensión “.ino” ubicado en una carpeta con el mismo nombre que el fichero.

En el arranque del entorno, éste ya aportará un proyecto genérico (con una plantilla de programación) con un nombre dependiente de la fecha actual. Se puede trabajar con ese proyecto directamente, aunque tenga presente que la ubicación del mismo por defecto será “Documentos” → ”Arduino”.

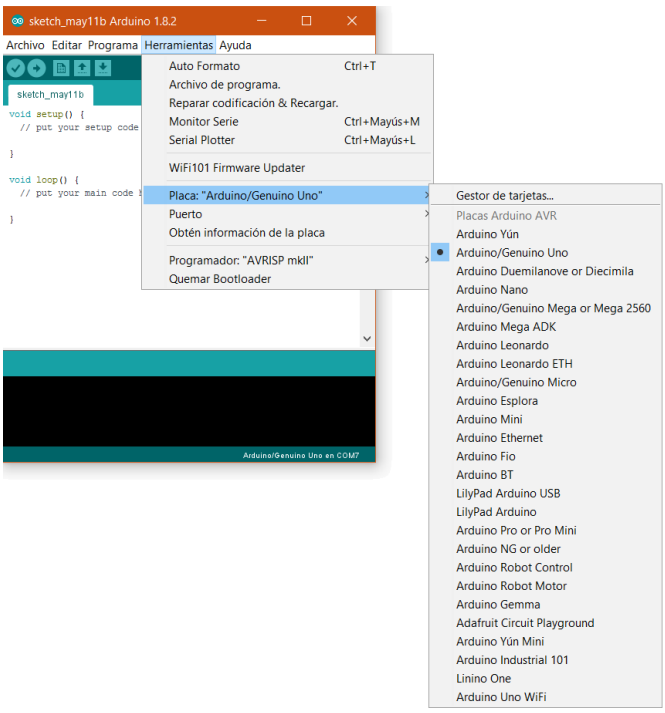


Se recomienda antes de comenzar, guardar el proyecto con un nombre diferente y en una ubicación conocida (“Archivo” → “Guardar Como”).

Parte 4 – Conexión con la PCB

Para poder comunicarse correctamente con su placa, hay que realizar una serie de configuraciones en el entorno de programación. Para una primera conexión de la placa al equipo, lleve a cabo los siguientes pasos:

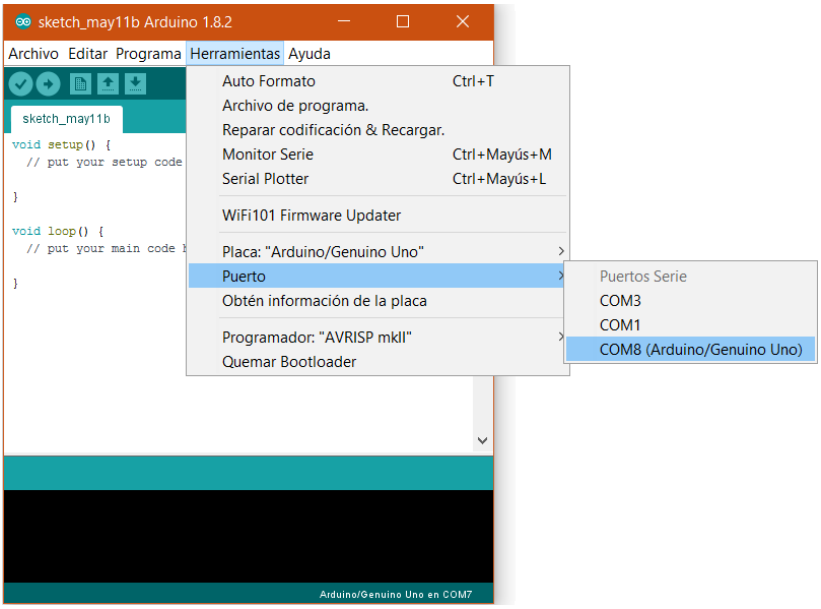
2.1. Al conectar la placa al equipo, necesitará instalarse el driver del dispositivo: si el entorno de arduino se ha instalado en el equipo, el sistema operativo encontrará automáticamente el driver apropiado y lo instalará. Si se encuentra en la situación en la que ha descargado la carpeta del entorno (y no se ha instalado en el equipo), no podrá encontrar el driver automáticamente y habrá que indicarle la ruta; en ese caso, el paquete que ha descargado posee una carpeta llamada “drivers” y, dentro de ella “FTDI USB Drivers”, donde se encontrará el driver requerido.



2.2. Al arrancar el entorno debe seleccionar el tipo de placa que tiene conectada. Para ello acceda a la opción “Herramientas”→”Placa”. En el

desplegable seleccione el tipo de placa que tiene conectada; en nuestro caso será "Arduino/Genuino Uno".

2.3. Debe seleccionar el puerto virtual (COM) en el que tiene conectada la placa. Para ello, acceda a la opción "Herramientas"→"Puerto". Aparecerá un listado de puertos de comunicación virtuales, todos identificados con las siglas "COM" seguidas de un número del puerto. En el caso de esta versión del entorno, aparecerá entre paréntesis dónde se encuentra conectada la placa.



Si no es el caso (por algún error de conexión o por la utilización de una versión anterior del entorno de programación), tendrá que acceder al "Administrador de Dispositivos" del sistema operativo y buscar, de entre los puertos de comunicación, cuál se corresponde con la placa.

Parte 5 – El lenguaje de programación

En la programación de la mayoría de los microcotroladores se hace uso de un lenguaje de programación imperativo (normalmente C/C++). La programación de este entorno supone una extensión de C++ con algunas peculiaridades, pero es prácticamente idéntico. Un programa básico para una placa Arduino está dividido en 5 partes principales. La estructura básica de un programa es la siguiente (se recomienda tener presente este orden):

Elemento	Ejemplo
1. Inclusión de librerías	#include <LiquidCristal.h>
2. Definiciones de constantes	#define PI 3.141592
3. Variables globales	Int LedState = 0;
4. Función setup	void setup() {...}
5. Función loop	void loop(){...}
6. Otras funciones propias	int miFun(int val){...}

1. Las inclusiones de librerías hacen referencia comúnmente a periféricos estándar, ante los cuales se aporta la interfaz de comunicación para facilitar la tarea de controlar estos dispositivos externos. Las librerías incluidas al instalar el entorno pueden consultarse en la ruta "C:\Program Files (x86)\Arduino\libraries".
2. Las constantes son elementos cuyo valor no se puede modificar durante la ejecución del código. De cara al microcontrolador, no ocupan memoria pues su valor se sustituye en tiempo de compilación.
3. Las variables globales suelen utilizarse para representar los pines a los que se conectan los periféricos, ya que dichos valores deberán ser utilizados, al menos, en las dos funciones siguientes (setup y loop). Igualmente se pueden definir variables globales para otros propósitos.

Tanto en variables globales como locales, Arduino proporciona los siguientes tipos:

Tipo	Descripción	Valores
boolean	Valor lógico.	true, false.
char	Carácter (ASCII). Entero 8 bits.	'a', 'A', '.', …
byte	Entero de 8 bits sin signo.	0 – 255
int	Entero de 16 bits.	$[-2^{15}, 2^{15}-1]$
long	Entero de 32 bits.	$[-2^{31}, 2^{31}-1]$
float	Real de 16 bits.	3.15, -1.3, …
double	Real de 32 bits.	3.15, -1.3, …

Puede trabajarse igualmente con tipos compuestos, tal y como serían los arrays y las cadenas de caracteres: Ejemplos:

Array: intv[10] = {1,2,3,4,5,6,7,8,9,10};|| Cadena: charcad[50] = "Hola Mundo!";

Además, pueden aplicarse diversos modificadores sobre las variables para obtener diferentes resultados:

Modificador	Resultado
unsigned	Aplicable a enteros. Número sin signo.
const	Significa que la variable puede ser usada como cualquiera de su tipo, pero su valor no puede ser modificado.
volatile	Dirige al compilador para cargar la variable desde memoria RAM y no desde un registro de almacenamiento. Una variable debe ser declarada volatile siempre que su valor pueda ser modificado por algo más allá de la sección del código en el que aparece, como por ejemplo un hilo de ejecución concurrente.
static	No se crean y se destruyen cada vez que se llama al bloque de código en el que está definidas, sino que su valor se guarda para las sucesivas llamadas.

4. Función setup: Esta función se ejecuta únicamente una vez en el programa y se corresponde con la inicialización del dispositivo. En esta función será necesario establecer los modos de funcionamiento de cada uno de los periféricos, así como inicializar los pines que se van a utilizar.
5. Función loop: Esta función se ejecuta repetidamente en nuestro programa; por lo tanto, será el lugar indicado para realizar las lecturas y/o escrituras sobre los periféricos.
6. Otras funciones: es recomendable que encapsule determinadas funcionalidades aparte para dejar el código más legible y reutilizable.

De forma transparente al usuario, el código en C++ que se ejecuta en el microcontrolador finalmente se corresponde con algo similar a lo siguiente:

```
void main()
{
	setup();
	while(true)
	{
		loop();
	}
}
```

Puede observar que se corresponde claramente con un código estándar para controlar por Polling una serie de periféricos.

Parte 6 – Acceso a pines externos

6.1. Descripción Hardware

Tal como se ha visto en teoría, el acceso a los dispositivos externos se lleva a cabo mediante los controladores de E/S; éstos, a su vez, están integrados por una serie de registros internos utilizados para configurar, consultar o compartir información con estos dispositivos.

Al trabajar con un microcontrolador, normalmente se accede a estos registros mediante E/S mapeada en memoria (sin hacer uso de funciones específicas como "inb" u "outb") pero, para ello, es necesario conocer dónde se encuentran estos registros y qué contienen.

Como ayuda, los microcontroladores suelen traer definidas como constantes las direcciones de estos registros, que serán las que veremos a continuación:

Controlador PUERTO B→ Compuesto por 3 registros de 8 bits cada uno (todos con la misma distribución de pines, únicamente varía su utilidad):

7	6	5	4	3	2	1	0
OSC2	OSC1	Pin 13	Pin 12	Pin 11	Pin 10	Pin 9	Pin 8

- Registro **DDRB** (Data Direction Register B): utilizado para configurar cada pin como entrada (0) o salida (1). Es un registro de lectura y escritura.
- Registro **PORTB** (Data Register B): utilizado para indicar el valor correspondiente en los pines de salida. Es un registro de escritura.
- Registro **PINB** (Input Pins Register B): utilizado para consultar el valor de los pines de entrada. Es un registro de lectura.

Controlador PUERTO C→ Compuesto por 3 registros de 8 bits cada uno (todos con la misma distribución de pines, únicamente varía su utilidad):

7	6	5	4	3	2	1	0
X	Reset	Pin A5	Pin A4	Pin A3	Pin A2	Pin A1	Pin A0

- Registro **DDRC** (Data Direction Register C): utilizado para configurar cada pin como entrada (0) o salida (1). Es un registro de lectura y escritura.
- Registro **PORTC** (Data Register C): utilizado para indicar el valor correspondiente en los pines de salida. Es un registro de escritura.
- Registro **PINC** (Input Pins Register C): utilizado para consultar el valor de los pines de entrada. Es un registro de lectura.

Controlador PUERTO D→ Compuesto por 3 registros de 8 bits cada uno (todos con la misma distribución de pines, únicamente varía su utilidad):

7	6	5	4	3	2	1	0
Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0

- Registro **DDRD** (Data Direction Register D): utilizado para configurar cada pin como entrada (0) o salida (1). Es un registro de lectura y escritura.
- Registro **PORTD** (Data Register D): utilizado para indicar el valor correspondiente en los pines de salida. Es un registro de escritura.
- Registro **PIND** (Input Pins Register D): utilizado para consultar el valor de los pines de entrada. Es un registro de lectura.

6.2. Acceso por Firmware

En esta sesión práctica haremos uso únicamente del PUERTO D.

6.2.1. Escritura en un pin

1. (En el "setup") Configurar el pin como escritura (output). Esto es, acceder a su registro "DDRx" y escribir un '1' en el bit correspondiente.
   Por ejemplo: configurar el pin 3 a modo escritura (valor 1) → DDRD = DDRD | 0x08;

2. (En el "loop") Escribir el valor del pin en el puerto "PORTx".
   Por ejemplo: Escribir un '1' en el pin 3 → PORTD = PORTD | 0x08;
   Escribir un '0' en el pin 3 → PORTD = PORTD & 0xF7;

6.2.2. Lectura de un pin

1. (En el "setup") Configurar el pin como lectura (input). Esto es, acceder a su registro "DDRx" y escribir un '0' en el bit correspondiente.
   Por ejemplo: configurar el pin 3 a modo lectura (valor 0) → DDRD = DDRD & 0xF7;
2. (En el "loop") Leer el valor del pin a través del puerto "PINx".
   Por ejemplo: Leer el valor del pin 3 → unsigned char c = (PIND & 0x08)>>3;

Parte 7 – Interrupciones en Arduino

El Arduino UNO posee 2 entradas externas de interrupción (pines digitales 2 y 3, que se les conoce como línea de interrupción 0 y 1, respectivamente).

En el programa del Arduino se deberán tener en cuenta estos apartados:

- Asignar interrupción (en la función ***setup***): haciendo uso de la función de alto nivel "**void *attachInterrupt(numInt, rutinaInt, evento);***", donde en "*numInt*" se indica 0 o 1 dependiendo de la interrupción que se vaya a utilizar (pines digitales 2 o 3, respectivamente); en "*rutinaInt*" se indica el puntero a la rutina de tratamiento de dicha interrupción (se ejecutará cada vez que se dé la condición de interrupción); y en "*evento*" se indicará el tipo de circunstancia que será motivo de la activación de la interrupción (al ser información recibida únicamente por un pin, se puede dar el caso de **RISING** si queremos que se active en los flancos de subida, **FALLING** para los flancos de bajada, **CHANGE** para ambos).
- Habilitar interrupciones: función "***sei()***", que habilita las interrupciones del sistema de forma general. A ejecutar tras las asignaciones de las interrupciones.
- Implementar la rutina de tratamiento de la interrupción: función aparte que se ejecutará cuando se den las condiciones de la interrupción. Recuerde que, dentro de la rutina de tratamiento de la interrupción, deberá deshabilitar las interrupciones globales del sistema al comienzo ("**void noInterrupts();**") y volverlas a habilitar al finalizar ("**void interrupts();**").

<u>Un ejemplo general de uso sería:</u>

```
void setup()
{
     attachInterrupt(0, rutina_int, CHANGE);
     sei();
}
void loop()
{
}
void rutina_int()
{
     noInterrupts();
     //ejecutar el tratamiento de la interrupción
     interrupts();
}
```