Periféricos y Dispositivos de Interfaz Humana
Grado en Ingeniería Informática

PRÁCTICA 2: EXPERIMENTACIÓN CON UN SISTEMA DE MICROCONTROLADOR: ARDUINO

Objetivos

I. Aprender a manejar sencillos componentes que actúan como periféricos: sensores de temperatura, pulsadores, LEDs, etc. Extender el concepto de periférico a este tipo de dispositivos.

II. Implementar proyectos autónomos de control de situaciones análogas a las presentes en el mundo real: control de temperatura, reproducción de sonido, control de tráfico, etc.

III. Aprender la forma de comunicación entre un PC y un microcontrolador a través de un puerto serie.

IV. Realizar prototipos con documentación (Processing) y diseños y esquemáticos (Fritzing) a partir de herramientas específicas.

Contenido

1. Introducción

En esta práctica se propone crear y verificar el funcionamiento de diversos sistemas de control basados en Arduino. La mayoría de los proyectos están implementados de forma equivalente o similar en múltiples recursos en Internet, por lo que se aconseja analizar esas implementaciones y adaptarlas a los requisitos que se piden.

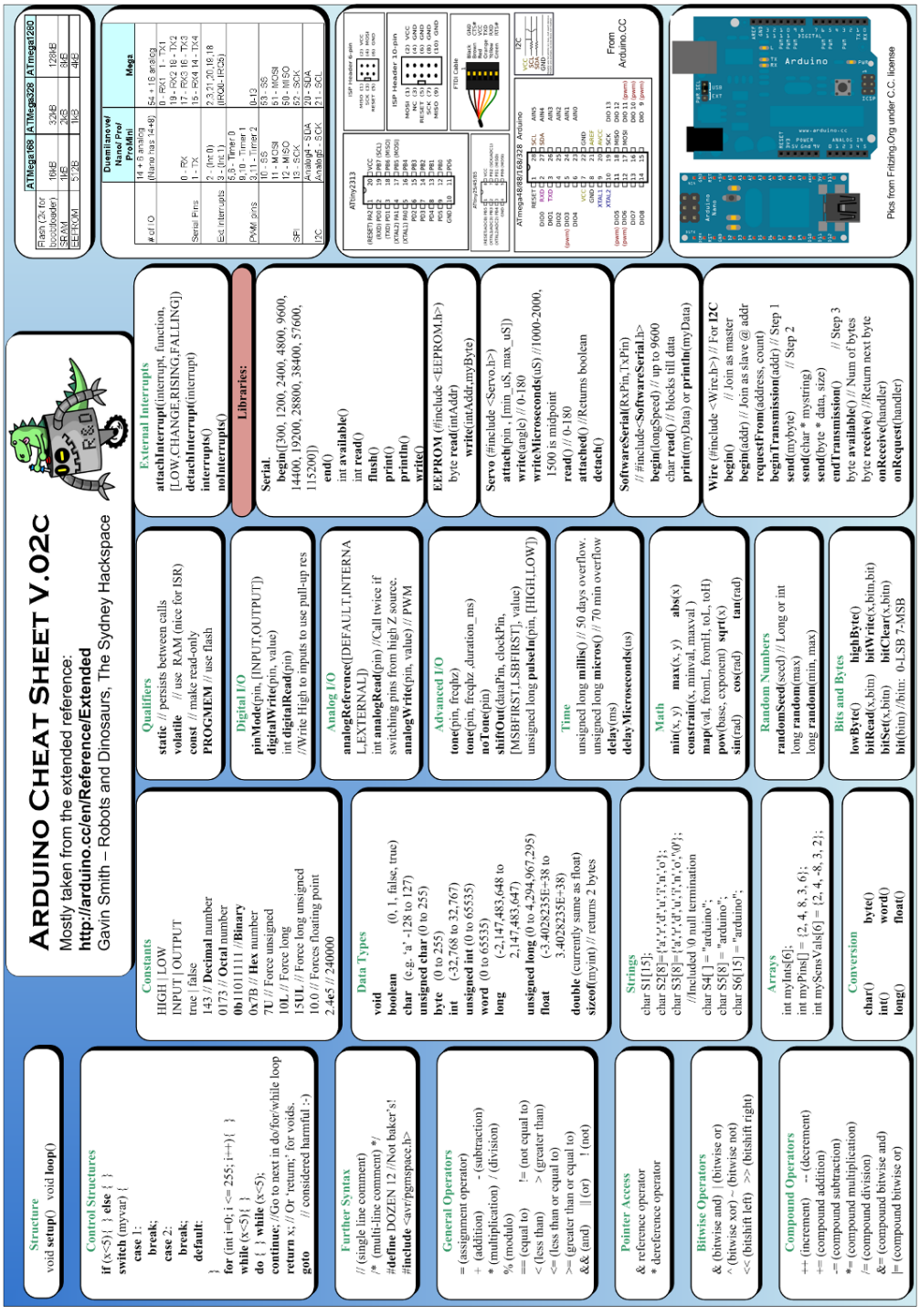


Figura 1. Hoja resumen de las principales órdenes y características de Arduino. (http://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/ArduinoCheatSheet.pdf)

2. Proyectos a implementar

Antes de implementar en una placa real de Arduino es recomendable simular el prototipo en un simulador software de Arduino. Se recomienda probar 123DCircuits (http://123d.circuits.io), que además permite añadir componentes a la placa de Arduino (otros simuladores sólo permiten trabajar con la propia placa).

2.1. Cruce de semáforos

Se trata de implementar un cruce de semáforos controlado por Arduino, para ello utilizaremos en el primer semáforo los pines 3 (led rojo), 4 (led ámbar), 5 (led verde), en el segundo semáforo utilizaremos los pines 6 (led rojo), 7 (led ámbar) y 8 (led verde). La secuencia de funcionamiento debe ser:

1. rojo 1 – verde 2 durante 3 segundos
2. rojo 1 – ambar 2 durante 500 ms
3. verde 1 – rojo 2 durante 3 segundos
4. ambar 1 – rojo 2 durante 500 ms

Adicionalmente, se agregarán dos botones para el control manual del semáforo:

- Botón 1: conmuta entre modo automático temporizado o modo manual.
- Botón 2: si el botón 1 se encuentra en modo manual, se produce un cambio del tráfico (el semáforo que estaba en verde pasa a ámbar [0.5s] y después a rojo, momento en que el que estaba en rojo pasa a verde).

Consejos:

1. Colocar los LEDs en la *protoboard* de forma que simulen un semáforo real. Separar físicamente ambos semáforos para saber a qué semáforo pertenece cada LED.
2. Usar una resistencia de *pull-up* para mantener el nivel de tensión en el pulsador (http://arduino.cc/es/tutorial/button).
3. Otra opción muy interesante y quizá más cómoda es usar interrupciones para detectar la pulsación de un botón: https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

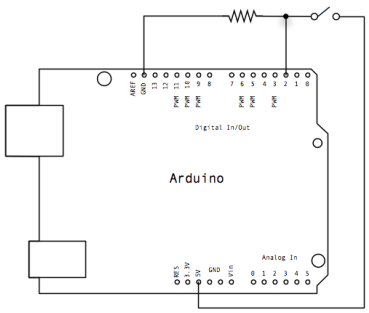


Figura 2. Conexión de una resistencia de pull-up/down a un botón de interrupción.

Nombre del Sketch	Descripción
cruceSemaforos	Implementa el cruce de semáforos

2.2. Dado de LEDs electrónico

Vamos a implementar un dado mediante 7 luces LEDs que genere una tirada aleatoria cada vez que se pulse un botón conectado al sistema. El dado "rueda" mostrando valores aleatorios durante 1.5 segundos, hasta que se detiene, mostrando el valor obtenido. La disposición de los LEDs debe ser análoga a la de un dado convencional, iluminándose los puntos que correspondan a cada número del 1 al 6 (Figura 3).

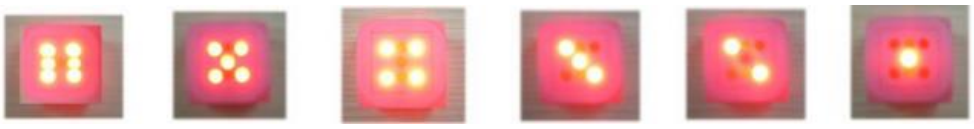


Figura 3. Configuración de los leds que deben encenderse para cada valor posible de la tirada.

Para generar números aleatorios, se puede utilizar la función "random" de Processing: http://arduino.cc/es/Reference/Random

Sintaxis:

random(min, max)

min - límite inferior del valor aleatorio, inclusive (opcional)

max - límite superior del valor aleatorio, exclusive (se devuelve hasta el anterior)

Además, cada vez que el dado se tira se debe enviar el valor a través del puerto serie. Para ello, usaremos la librería "SoftwareSerial" (http://arduino.cc/es/Reference/SoftwareSerial). Para comprobar que el dado no está trucado, realizar un *sketch* adicional que realice 200 lanzamientos del dado sin tener que pulsar el botón. Posteriormente, recoger los datos del monitor serie y realizar un histograma con una hoja de cálculo para comprobar la distribución de los valores.

Consejos:

1. Utilizar funciones en la implementación (por ejemplo "rodarElDado").
2. Conviene utilizar inicialmente la función "randomSeed" para que las secuencias no sean iguales cada vez que se reinicie la placa.

Nombre del Sketch	**Descripción**
dadoElectronico	Implementa el lanzamiento del dado al pulsar un botón
probadorDado	Realiza 200 lanzamientos del dado sin intervención del botón
Otros	**Descripción**
histogramaTiradas	Muestra una tabla con los valores de los 200 lanzamientos y el histograma que le corresponde

2.3. Reproductor de sonido

Generar sonido a través del altavoz piezoeléctrico (*buzzer*) incluido en el lab-kit de Cooking Hacks. Además, se debe controlar el volumen del altavoz a través de un potenciómetro y usar la función de "play-pause" a través de un botón. El *buzzer* OBO-27225 no está polarizado, por lo que es indiferente el pin que se conecte a tierra (Figura 4).

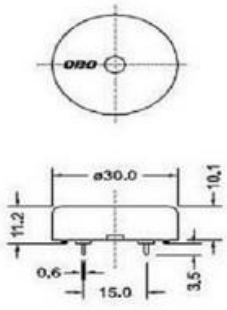
SOUND PRESSURE LEVEL	輸出音壓	85dB min. at 2.5KHz / 9Vp-p Square Ware / 10cm 75dB min. at 2.5KHz / 1Vrms Sine Ware / 10cm
CAPACITANCE	靜電容量	20, 000pF± 30% at 120Hz
ALLOWABLE INPUT VOLTAGE	承受電壓	30Vp-p max.
CURRENT CONSUMPTION	消耗電流	3mA max. at 2.5KHz / 9Vp-p Square Wave
CASE MATERIAL	塑膠規格	Top Case: PC(UL 94V-2) Bottom Case: PB T(UL94V-0)
LEAD PIN MATERIAL	導針規格	Phosphor Bronze (Sn Plated)
OPERATING TEMP. RANGE	操作溫度	-20°C to + 70°C
STORAGE TEMP. RANGE	儲存溫度	-40°C to + 85°C
WEIGHT	重　量	5.5gms
Dimensions (Unit: mm ± 0.5)		

Figura 4. Especificaciones del altavoz OBO-27225
(http://www.syntax.com.tw/products/products_c.php?itemid=8139)

El *sketch* de la batería de ejemplos de Arduino “toneMelody” puede servir de referencia para implementar este proyecto. Se aconseja emplear la cabecera "pitches.h". Este archivo contiene todos los valores de las frecuencias de las notas musicales en formato anglosajón. Por ejemplo, NOTE_C4 es una C media (do de la 4ª escala). NOTE_FS3 es F aguda (fa sostenido de la 3ª escala), y así sucesivamente. Un valor ‘0’ significa un silencio.

Por ejemplo, la canción de Mario Bros se podría almacenar en dos vectores, uno con las notas musicales (o silencios) y otro con la duración de cada una:

```
int melodia[] = {
  NOTE_E7, NOTE_E7, 0, NOTE_E7,
  0, NOTE_C7, NOTE_E7, 0,
  NOTE_G7, 0, 0,  0,
  NOTE_G6, 0, 0, 0,

  NOTE_C7, 0, 0, NOTE_G6,
  0, 0, NOTE_E6, 0,
  0, NOTE_A6, 0, NOTE_B6,
  0, NOTE_AS6, NOTE_A6, 0,

  NOTE_G6, NOTE_E7, NOTE_G7,
  NOTE_A7, 0, NOTE_F7, NOTE_G7,
  0, NOTE_E7, 0,NOTE_C7,
  NOTE_D7, NOTE_B6, 0, 0,

  NOTE_C7, 0, 0, NOTE_G6,
  0, 0, NOTE_E6, 0,
  0, NOTE_A6, 0, NOTE_B6,
  0, NOTE_AS6, NOTE_A6, 0,

  NOTE_G6, NOTE_E7, NOTE_G7,
  NOTE_A7, 0, NOTE_F7, NOTE_G7,
  0, NOTE_E7, 0,NOTE_C7,
  NOTE_D7, NOTE_B6, 0, 0
  };
```

```
// note durations: 4 = quarter note,
 eighth note, etc.:
int duraciones[] = {
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  9, 9, 9,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  9, 9, 9,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
};
```

Figura 5. Canción de Mario Bros (http://www.linuxcircle.com/2013/03/31/playing-mario-bros-tune-with-arduino-and-piezo-buzzer/)

Otra melodía de prueba mucho más sencilla es la que incluye el ejemplo “toneMelody”:

```
int melodia[] = {
      NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3,NOTE_C4};

int noteDurations[] = {
  4, 8, 8, 4,4,4,4,4 };
```

Nombre del Sketch	Descripción
reproductorMusica	Reproduce una melodía y para o reanuda cuando se pulsa un botón

2.4. Termómetro con registro de datos

En este proyecto se va a acceder a la memoria EEPROM para memorizar una serie de valores de temperatura registrados a ciertos intervalos de tiempo o cuando se solicita manualmente mediante el puerto serie. Para acceder a la memoria EEPROM utilizaremos la librería "EEPROM.H" (http://arduino.cc/es/Reference/EEPROM).

Para ello, utilizaremos el sensor de temperatura MCP9700 incluido en el LabKit. Este sensor va a entregar una variación de tensión lineal de 10 mV por cada grado Celsius de variación, con un rango de temperatura que va desde -40ºC hasta 125ºC. Podemos encontrar información del fabricante sobre este componente en http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf.

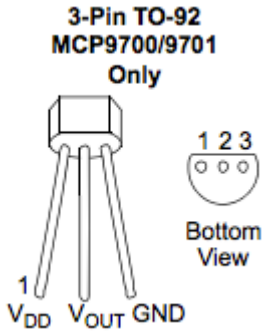


Figura 6. Esquema de conexiones del MCP9700

En la hoja de datos se puede leer que 500mV corresponde a 0°C, es decir que cuando la patilla Vout entregue 500mV, se están midiendo a 0°C. Teniendo en cuenta que cada 10mV de variación supone 1°C y que la lectura analógica en Arduino se discretiza en 1024 valores para un rango de 5V, se puede obtener la temperatura a partir de la lectura hecha por Arduino como:

$$Temperatura = \frac{\left(\frac{LecturaAnalógica \times 5}{1024} - 0.5\right)}{0.01}$$

El control del sistema se realizará a través del puerto serie cuando la placa Arduino está conectada al PC, admitiendo las siguientes opciones:

Tecla	Acción
R/r	Leer todos los datos de la EEPROM de temperatura y enviarlos como texto a través del puerto serie
W/w	Escribir la temperatura actual como un nuevo registro
B/b	Borrar los datos de temperatura de la EEPROM. Poner el contador de lecturas a 0

A partir de los datos obtenidos de un mínimo de 50 lecturas, crear una hoja de cálculo que dibuje una gráfica de evolución de la temperatura.

Consejos:

1. Se debe controlar si se llena la memoria EEPROM. Para ello, no se debe permitir almacenar más de 128 lecturas (512 bytes para 128 números reales de 4 bytes). Para escribir y leer datos de la EEPROM, se pueden usar las funciones put (https://www.arduino.cc/en/Reference/EEPROMPut) y get (https://www.arduino.cc/en/Reference/EEPROMGet).
2. Para detectar cuándo el PC está enviando datos a la placa Arduino mediante el puerto serie se puede usar la función "Serial.available" (http://arduino.cc/es/Serial/Available)
3. Definir el intervalo de tiempo (en segundos) para realizar las lecturas con "#define" o como variable global. Utilizar un contador que se compare con dicho valor en cada iteración. Para que cada iteración tarde aproximadamente 1 segundo, usar la función "delay".

Nombre del Sketch	**Descripción**
registroTemperatura	Registra la temperatura a intervalos regulares de tiempo
Otros	**Descripción**
graficaTemperaturas	Muestra una tabla con los valores de, al menos, 50 mediciones de temperatura y su gráfica de evolución

2.5. Termómetro con registro de datos y visor LCD

Este proyecto amplia al anterior de registro de temperaturas, permitiendo visualizar en el visor LCD la temperatura actual y el número de mediciones que se llevan guardadas en la EEPROM actualmente.

Figura 7. Ejemplo de salida esperada en el LCD

Para usar un LCD como el que incluye el LabKit, debemos realizar el cableado que se muestra en la Figura 8. Para un manejo sencillo del LCD dentro de la placa de prototipado, es recomendable que esté soldada con los pines de conexión (se pueden soldar en el aula de prácticas). Para comprobar que todo funciona bien, se puede utilizar el ejemplo "Scroll" incluido en el IDE de Arduino.

El manejo de este tipo de pantallas LCD es muy simple en Arduino si utilizamos la librería "LiquidCrystal". Para más información, visitar la página de referencia de dicha librería: http://arduino.cc/en/Reference/LiquidCrystal.

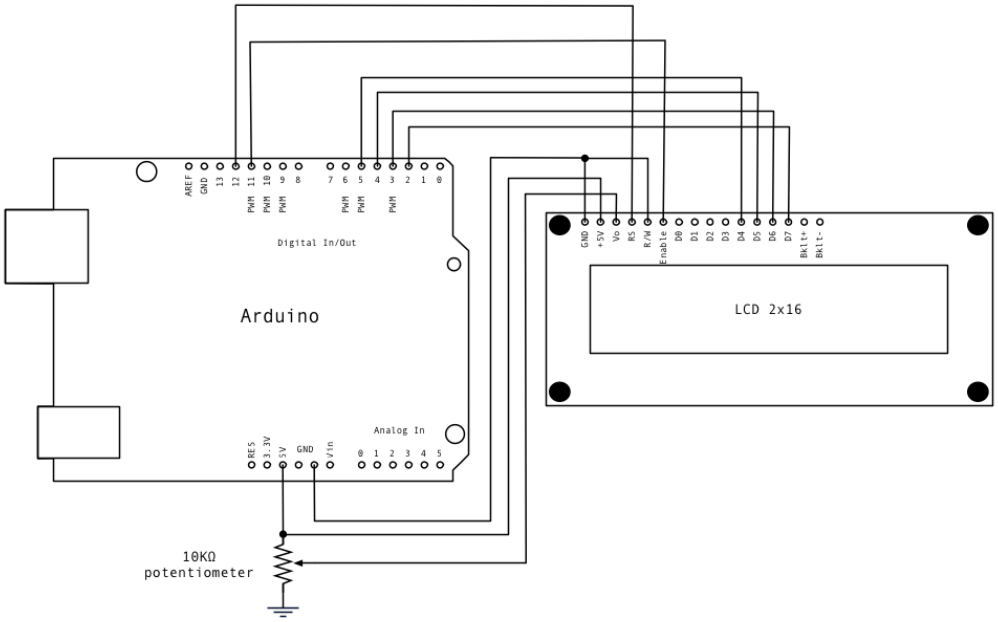


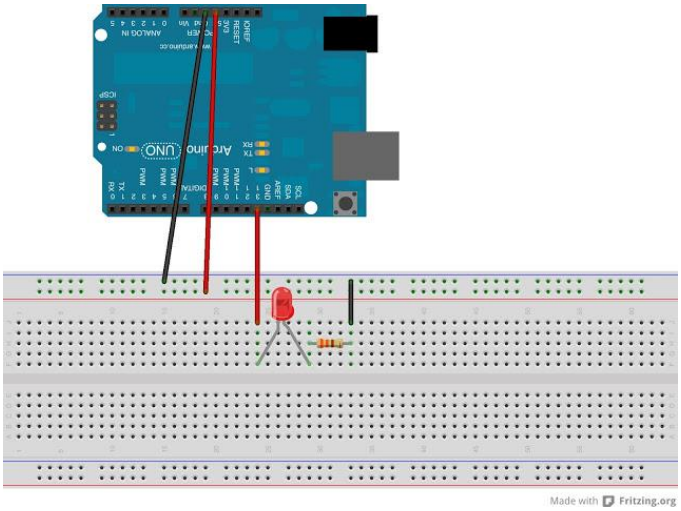
Figura 8. Conexión de un LCD 2x16 a Arduino

Nombre del Sketch	Descripción
registroTemperaturasLCD	Registra la temperatura a intervalos regulares de tiempo y muestra en la pantalla LCD la temperatura actual y el número de lecturas realizadas.

3. Evaluación y entrega

Para cada proyecto se debe entregar y evaluar:

1. Imagen con el esquema de conexiones realizado en Fritzing.



2. Código fuente debidamente documentado. Describir los pines de entrada y salida que se usan y su significado.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

3. Fotografías y/o vídeos demostrando el funcionamiento real del proyecto. Si los vídeos ocupan demasiado espacio, se deben recodificar para que se puedan subir a SWAD.
4. Demostración "en vivo" del proyecto funcionando en el laboratorio de prácticas. Si no es posible porque se ha tenido que desmontar el circuito para montar otro, tratar de que las imágenes y videos anteriores sean suficientemente descriptivos.

Los ficheros se **clasificarán por carpetas** (Proyecto 1, proyecto 2, etc.) y el tamaño máximo de cada carpeta será 20MB.

- **Número de sesiones para esta práctica:** 9 de abril, 16 de abril y 23 de abril de 2018.
- **Entrega individual o por parejas.**
- **Fecha límite de entrega y defensa:** 30 de abril de 2018.