Practica 1:Comenzando con Arduino

1.1 Objetivos.

- Introducción al Arduino.
- Primeros pasos con Arduino.
- Funciones básicas para su programación.
- Ejemplo de aplicación.

1.2 Materiales empleados.

- Arduino UNO.
- Diodo Led.

1.3 Introducción al Arduino.

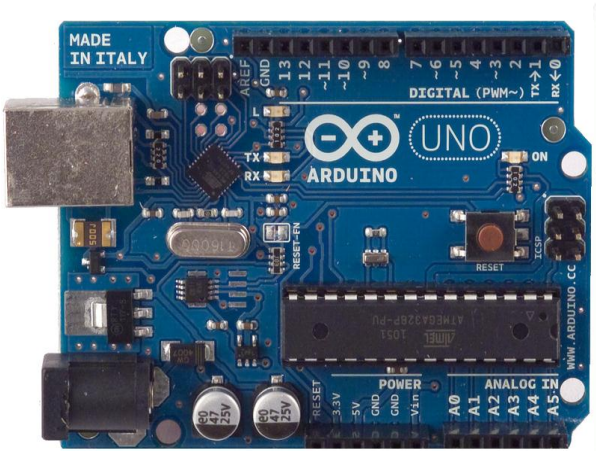
¿Qué es Arduino?

Arduino es una herramienta para hacer que los ordenadores puedan sentir y controlar el mundo físico a través de tu ordenador personal. Es una plataforma de desarrollo de computación física (physical computing) de código abierto, basada en una placa con un sencillo microcontrolador y un entorno de desarrollo para crear software (programas) para la placa.

Puedes usar Arduino para crear objetos interactivos, leyendo datos de una gran variedad de interruptores y sensores y controlar multitud de tipos de luces, motores y otros actuadores físicos. Los proyectos con Arduino pueden ser autónomos o comunicarse con un programa (software) que se ejecute en tu ordenador. La placa puedes montarla tú mismo o comprarla ya lista para usar, y el software de desarrollo es abierto y lo puedes descargar gratis desde la página www.arduino.cc/en/.

El Arduino puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente.

Hardware y cable USB

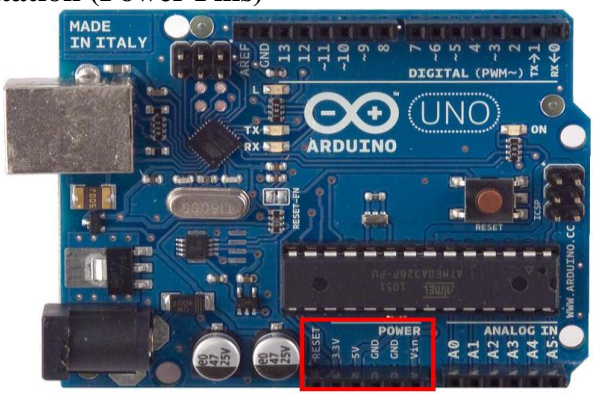


Especificaciones técnicas

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current for I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Power, Inputs and Outputs.

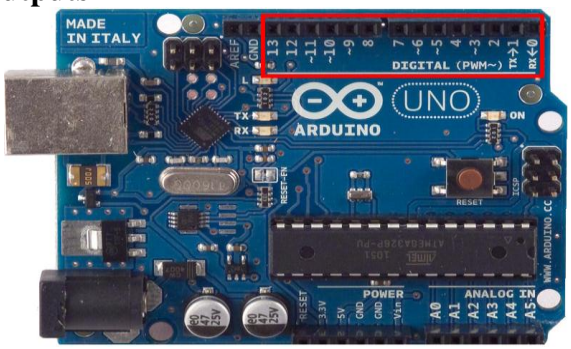
Pines de alimentation (Power Pins)



Bien alimentemos al arduino mediante la conexión USB o mediante una fuente externa (recomendada de 7-12V), vamos a tener unas salidas de tensión continua debido a unos reguladores de tensión y condensadores de estabilización. Estos pines son:

- VIN: se trata de la fuente tensión de entrada que contendrá la tensión a la que estamos alimentando al Arduino mediante la fuente externa.
- 5V: fuente de tensión regulada de 5V, esta tensión puede venir ya sea de pin VIN a través de un regulador interno, o se suministra a través de USB o de otra fuente de 5V regulada.
- 3.3V: fuente de 3.3 voltios generados por el regulador interno con un consumo máximo de corriente de 50mA.
- GND: pines de tierra.

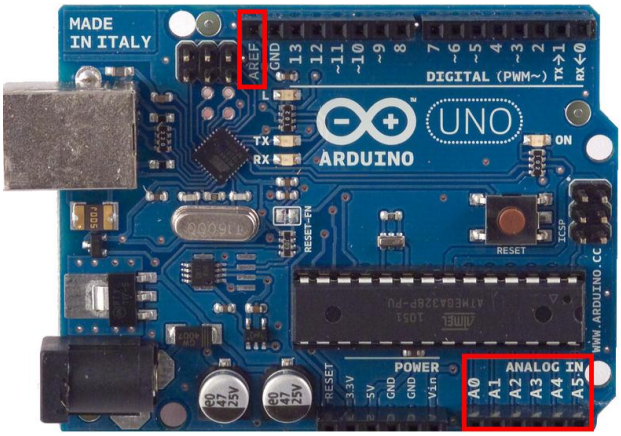
Digital Inputs/Outputs



Cada uno de los 14 pines digitales se puede utilizar como una entrada o salida. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia de pull-up (desconectado por defecto) de 20 a 50 kOhm. Además, algunos pines tienen funciones especializadas como:

- Pin 0 (RX) y 1 (TX). Se utiliza para recibir (RX) y la transmisión (TX) de datos serie TTL.
- Pin 2 y 3. Interrupciones externas. Se trata de pines encargados de interrumpir el programa secuencial establecido por el usuario.
- Pin 3, 5, 6, 9, 10 y 11. PWM (modulación por ancho de pulso). Constituyen 8 bits de salida PWM con la función analogWrite ().
- Pin 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines son de apoyo a la comunicación SPI.
- Pin 13. LED. Hay un LED conectado al pin digital 13. Cuando el pin es de alto valor, el LED está encendido, cuando el valor está bajo, es apagado.

Analog Inputs



El Arduino posee 6 entradas analógicas, etiquetadas desde la A0 a A5, cada una de las cuales ofrecen 10 bits de resolución (es decir, 1024 estados). Por defecto, tenemos una tensión de 5V, pero podemos cambiar este rango utilizando el pin de AREF y utilizando la función analogReference(), donde le introducimos una señal externa de continua que la utilizara como referencia.

1.4 Primeros pasos con Arduino

Descarga del IDE (Software) de Arduino

Una vez que conocemos todo los pines necesarios para nuestro manejo y control del Arduino, vamos a instalar el software para poder programarlo mediante el ordenador.

Contamos la placa Arduino

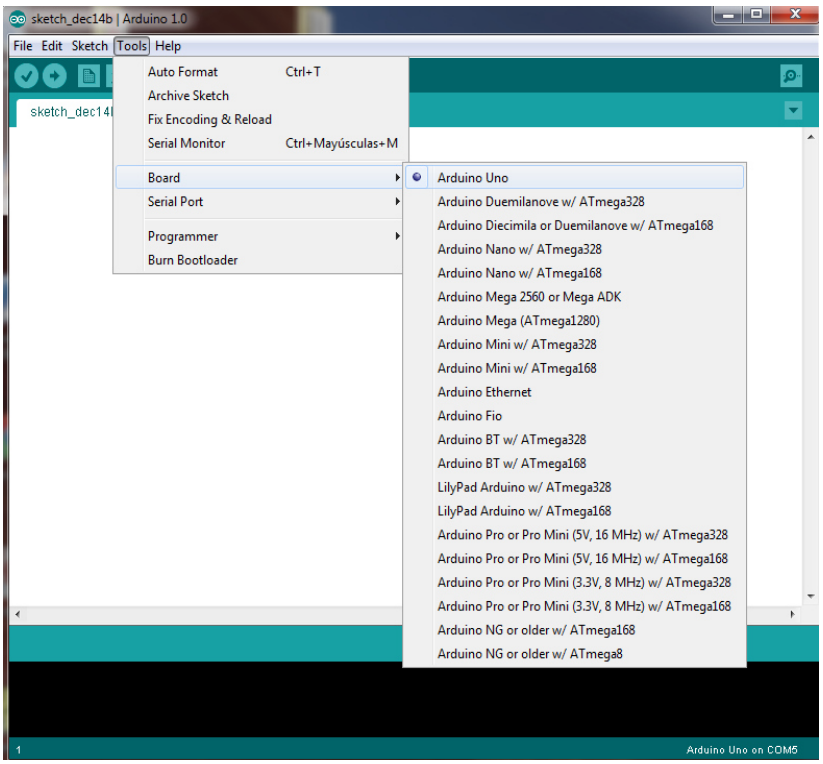
Conectamos la placa Arduino al ordenador usando el cable USB, una vez conectada el led de la placa PWR (led de alimentación) deberá permanecer encendido a partir de ahora.

Instalamos los drivers

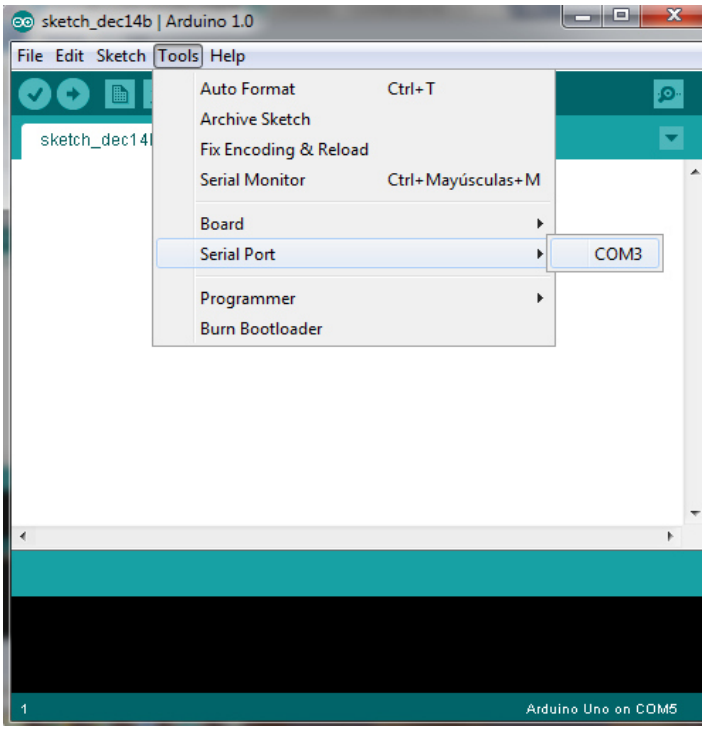
Al conectar el Arduino, Windows automáticamente deberá de inicializar la instalación de los drivers.

Ejecutamos la aplicación Arduino, seleccionamos la placa y el puerto serie.

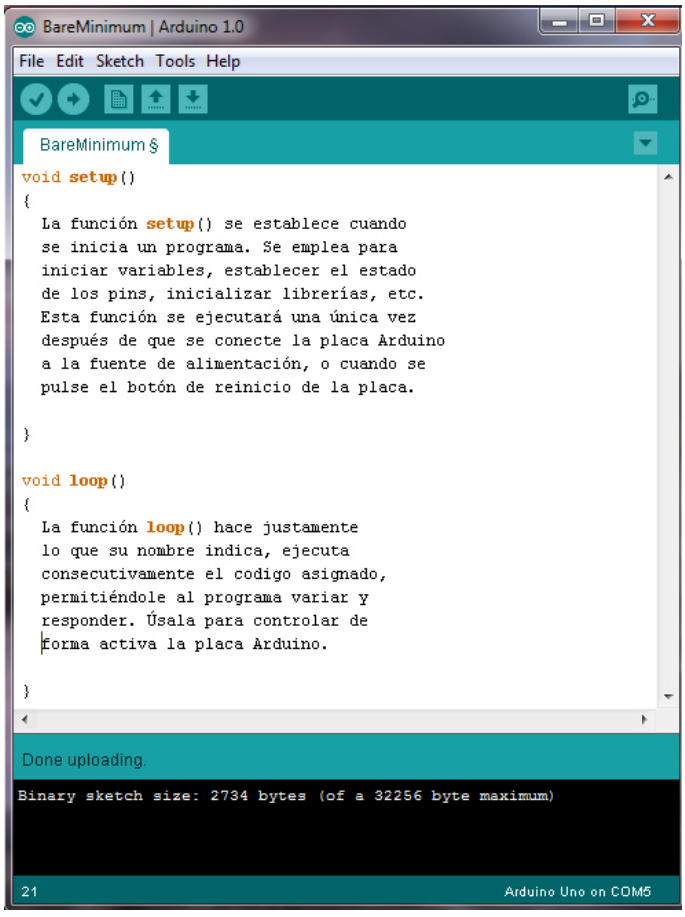
Una vez abierta la aplicación nos vamos a Tools→Board→Arduino UNO



Una vez seleccionado el modelo de nuestra placa tendremos que seleccionar el dispositivo serie de la placa:

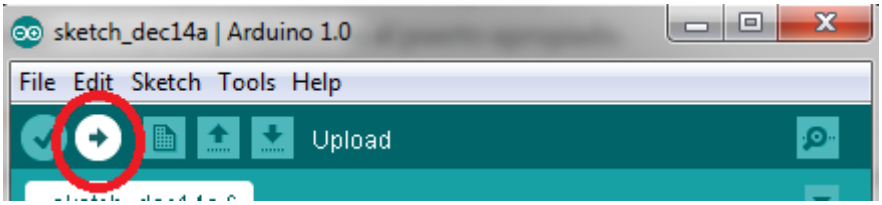


Una vez que tenemos configurada nuestra placa Arduino al ordenador, vamos a estudiar la estructura del lenguaje de programación de Arduino.



Cargar el programa a la placa.

Una vez que tenemos desarrollado el programa completo para cargarlo en el Arduino solo tenemos que:



1.5 Funciones básicas.

- ### E/S Digitales

pinMode(pin,modo)

Configura el pin especificado para comportarse como una entrada (INPUT) o una salida (OUTPUT).

```
Ejm:  pinMode(Pin13, OUTPUT)
```

digitalWrite(pin,valor)

Asigna el valor HIGH (5V) o LOW (0V) a un pin digital.

```
Ejm: digitalWrite(Pin13 , HIGH);
```

digitalRead(pin)

Lee el valor de un pin digital especificado, HIGH o LOW.

```
Ejm: val = digitalRead(Pin13);
```

- ### E/S Analógicas

analogRead(pin)

Lee el valor de tensión en el pin analógico especificado. La placa Arduino posee 6 canales conectados a un conversor analógico digital de 10 bits. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023. Esto proporciona una resolución en la lectura de: 5 voltios / 1024 unidades, es decir, 0.0049 voltios (4.9mV) por unidad. El rango de entrada puede ser cambiado usando la función analogReference().

```
Ejm:  val = analogRead(Pin3)
```

analogWrite(pin,valor)

Escribe un valor analógico (PWM) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor. Después de llamar a la función **analogWrite()**, el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función **analogWrite()** (o una llamada a las funciones **digitalRead()** o **digitalWrite()** en el mismo pin). La frecuencia de la señal PWM será de aproximadamente 490 Hz. los valores de analogRead van desde 0 a 1023 y los valores de analogWrite van desde 0 a 255

Parametros:

- pin: Es el pin en el cual se quiere generar la señal PWM.
- valor: El ciclo de trabajo deseado comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

```
Ejm:   val = analogRead(analogPin);
       analogWrite(ledPin, val / 4);
```

- **Comunicación Serie**

Se utiliza para la comunicación entre la placa Arduino y un ordenador u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie **Serial**. Se comunica a través de los pines digitales 0 (RX) y 1 (TX), así como con el ordenador mediante USB. Por lo tanto, si utilizas estas funciones, no puedes usar los pines 0 y 1 como entrada o salida digital. Puedes utilizar el monitor del puerto serie incorporado en el entorno Arduino para comunicarte con la placa Arduino. Haz clic en el botón del monitor de puerto serie en la barra de herramientas y selecciona la misma velocidad en baudios utilizada en la llamada a begin().

Serial.begin(speed)

Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para comunicarse con el ordenador, utilice una de estas velocidades: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200.

```
Ejm: Serial.begin(9600);
```

Serial.read()

Lee los datos entrantes del puerto serie.

```
Ejm; Byte = Serial.read();
```

Serial.print(val,[format])

Imprime los datos al puerto serie como texto ASCII.
val: el valor a imprimir - de cualquier tipo
format: especifica la base (formato) a usar; los valores permitidos son BYTE, BIN (binarios o base 2), OCT (octales o base 8), DEC (decimales o base 10), HEX (hexadecimales o base 16). Para números de coma flotante, este parámetro especifica el número de posiciones decimales a usar.

Ejm: Serial.print(78) imprime "78"
Serial.print('N') imprime "N"
Serial.print(78, BYTE) imprime "N"
Serial.print(78, DEC) imprime "78"
Serial.println(1.23456, 0) imprime "1.23"
Serial.println(1.23456, 2) imprime "1.23"

Serial.println(val,[format])

Imprime los datos al puerto serie como texto ASCII seguido de un retorno de carro (ASCII 13, o '\r') y un carácter de avance de línea (ASCII 10, o '\n').

```
Ejm:Serial.println(analogValue);     // imprime como ASCII
decimal
    Serial.println(analogValue, HEX);// imprime como ASCII
hexadecimal
```

Serial.available()

Devuelve el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Se refiere a datos ya recibidos y disponibles en el buffer de recepción del puerto (que tiene una capacidad de 128 bytes).

```
if (Serial.available() > 0)
       {
           //realiza la lectura del puerto serie
       }
```

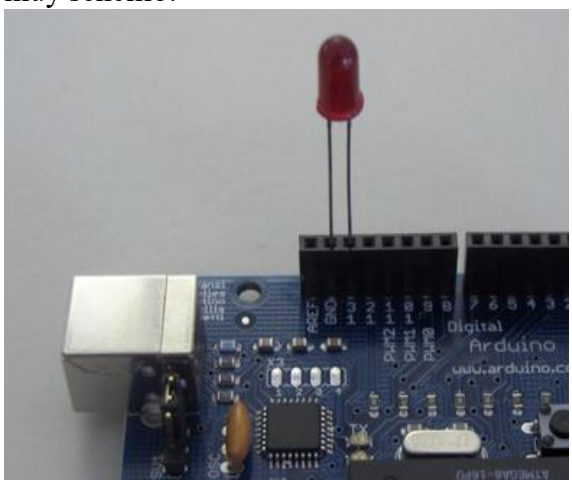
1.6 Ejemplo de Aplicación

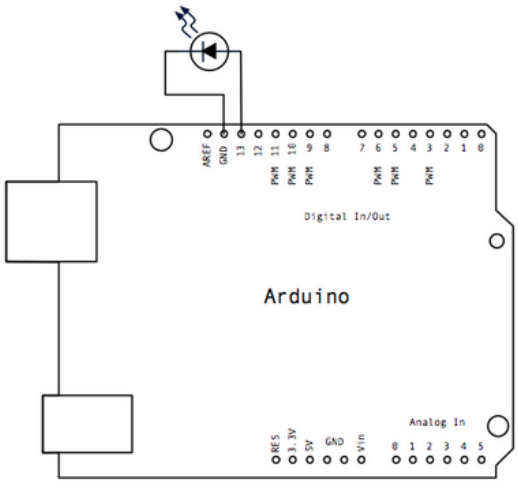
Crear un programa que controle el parpadeo de un led, con las funciones anteriormente explicadas, y que se pueda variar el tiempo de estado alto y el tiempo de estado bajo del mismo.

Utilizar la instrucción delay(ms), que hace que el programa se pare y espere el tiempo seleccionado en milisegundos.

Solución:

Podemos asignar la función que encienda y apague a cualquier pin digital, exceptuando el 0 y 1, como comentamos anteriormente, por lo tanto si asignamos el pin 13 como el responsable de esta función, el circuito a montar es muy sencillo:





Y el código de programación para el control del parpadeo del LED sería:

```
/*
PRACTICA 1:
Crear un programa que controle el parpadeo de un led, con
las funciones anteriormente explicadas, y que se pueda
variar el tiempo de estado alto y el tiempo de estado
bajo del mismo.

Utilizar la instrucción delay(ms), que hace que el programa
se pare y espere el tiempo seleccionado en milisegundos.
 */

void setup() {
  // Inicializamos el Pin 13 como una salida, donde va a
  //estar conectado el LED
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // Pontemos el LED en estado alto
  delay(1000);              // Tiempo de estado alto en mseg
  digitalWrite(13, LOW);    // Ponemos el LEd en estado bajo
  delay(1000);              // Tiempo de estado bajo en mseg
}
```

Practica 2: Estructuras de Control

2.1 Objetivos.

- **Conocimiento de las estructuras de control en el lenguaje de programación de Arduino.**
- **Manejo y utilización de las estructuras de control.**
- **Ejemplo de aplicación.**

2.2 Material empleado.

- Arduino UNO.
- Diodo Led.

2.3 Conocimientos Teóricos.

A continuación se muestran todas las estructuras de control más usuales en la programación de Arduino.

- **If(condicional)**

  Esta estructura puede ser usada en conjunto con uno o más operadores de comparación, comprueba si cierta condición se cumple, por ejemplo, si un input posee un valor mayor a cierto número. El formato para una comprobación *if* es el siguiente:

```
If(algunavariable>50)
{
 // Realiza una operación
}
```

  Veamos los distintos operadores de comparación:

  x == y (x es igual a y)
  x != y (x no es igual a y)
  x < y (x es menor a y)
  x > y (x es mayor a y)
  x <= y (x es menor o igual a y)
  x >= y (x es mayor o igual a y)

- **if… else.. o if…else if…**

  if/else permite mayor control sobre el flujo del código que la declaración *if* básica, por permitir agrupar múltiples comprobaciones. Por ejemplo, una salida analógica podría ser comprobada, y tomarse una acción si el valor de la salida es menor a 500, y, otra acción si el valor es igual o mayor a 500. Ejm:

```
if (pinCincoInput < 500)
{ // acción A }
else
{ // acción B }
```

If/else if puede proceder a una comprobación **if**, de esta forma, se pueden realizar múltiples comprobaciones en una misma estructura de condiciones. Cada comprobación procederá a la siguiente, sólo cuando su propio resultado sea *FALSE*. Cuando el resultado sea *TRUE*, su bloque de código contenido, será ejecutado, y el programa esquivará las siguientes comprobaciones hasta el final de la estructura de comprobaciones. Si ninguna comprobación devuelve valor *TRUE*, el **else** será ejecutado, y de no haber ninguno declarado, simplemente no sucede nada.

Entonces un bloque **else if** puede ser usado con o sin **else** al final. La cantidad de declaraciones **else if**, y sus ramificaciones son ilimitadas.
Ejm:

```
if (pinCincoInput < 500)
{
  // ejecutar A
}
else if (pinCincoInput >= 1000)
{
  // ejecutar B
}
else
{
  // ejecutar C
}
```

- **Switch/case**

Al igual que la sentencia **if**, **switch...case** controla el flujo de programas permitiendo a los programadores especificar diferentes códigos que deberían ser ejecutados en función de varias condiciones. En particular, una sentencia switch compara el valor de una variable con el valor especificado en las sentencias case. Cuando se encuentra una sentencia case cuyo valor coincide con dicha variable, el código de esa sentencia se ejecuta.
La palabra clave **break** sale de la sentencia switch, y es usada típicamente al final de cada case. Sin esta sentencia break, la estructura switch continuaría ejecutándose hasta encontrar un break o hasta llegar al final de la sentencia switch. Ejm:

```
switch (var) {
    case 1:
      //hacer algo cuando sea igual a 1
      break;
    case 2:
      //hacer algo cuando sea igual a 2
      break;
    default:
      // si nada coincide, ejecuta el "default"
      // el "default" es opcional
  }
```

- **While**

  Los bucles **while** se ejecutan continuamente, hasta que la expresión de dentro del paréntesis, (), pasa a ser falsa. Algo debe modificar la variable comprobada, si no el bucle **while** nunca terminará. Lo que modifique la variable puede estar en el código, como una variable que se incrementa, o ser una condición externa, como el valor que da un sensor.Ejm:

```
var = 0;
while(var < 200){
  // haz algo repetitivo 200 veces
  var++;
}
```

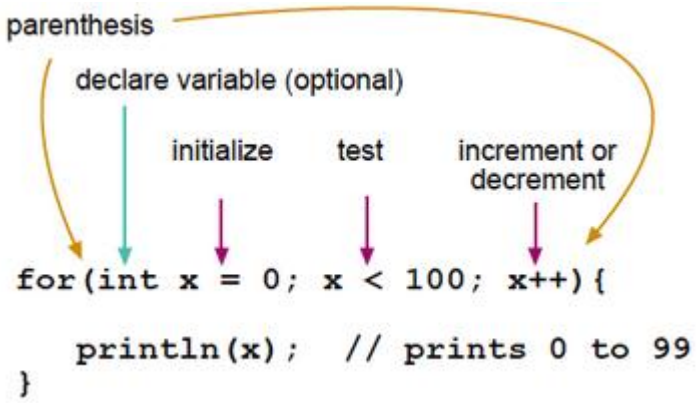
- **Do/While**

  El bucle "'do'" trabaja de la misma manera que el bucle "'while'", con la excepción de que la condición se comprueba al final del bucle, por lo que este bucle se ejecuta "siempre" al menos una vez.

```
do
{
  delay(50);  // espera a que los sensores se
estabilicen
  x = readSensors(); // comprueba los sensores

} while(x < 100); //si se cumple la condición se repite
el bucle
```

- **For**

  La declaración **for** es usada para repetir un bloque encerrado entre llaves. Un incremento de un contador es usado, normalmente, para aumentar y terminar con el bucle. La estructura **for** es muy útil para la mayoría de las operaciones repetitivas, y habitualmente se usa para operaciones con vectores, para operar sobre conjuntos de datos/pines



2.4 Ejemplo de Aplicación.

Se pretende realizar un programa que controle el tiempo de estado alto y bajo de un led, es decir que le introduzcamos por el puerto serie un numero de 0 a 9, que indican 0mseg y 900mseg de tiempo de estado alto y estado bajo, y podamos controlar el parpadeo del mismo. Por lo que vamos a programar con el fin de que nos pida por pantalla que introduzcamos el tiempo de estado alto y el tiempo de estado bajo y una vez introducido que se realice el control del parpadeo del led.

Notas: ASCII-48=DEC

Solución:

```
void setup() {
  Serial.begin(9600);   //Inicializamos el puerto serie
  pinMode(13, OUTPUT);  //Asignamos al pin 13 como salida donde conectaremos el LED
  //Escribimos por puerto serie la primera vez
  Serial.println("Introduce el tiempo de estado ALTO:");
}
 //Inicializacion de variables
  char datosbyte;
  byte f=0;
  int x=0;
  int y=0;
  void loop() {

  //Vamos a leer por puerto serie el valor de estado alto y bajo
    if (Serial.available() > 0)//Compruebo si el puerto serie esta accesible
       {
           if(f==0) //Untilizamos una variable llamada bandera que nos ralizara
                     //la funcion de conmutacion de tiempo alto y bajo
             {
               datosbyte=Serial.read(); //Leo los datos del puerto serie
               x=datosbyte-48;  // paso de codigo ASCII a decimal
               Serial.print("El tiempo de estado ALTO es: ");
               Serial.println(x,DEC); // Muetro el tiempo introducido

               Serial.println("Introduce el tiempo de estado BAJO:");
               f=1;
             }
       }

 '   if (Serial.available() > 0)
        {
              if(f==1)
              {
                datosbyte=Serial.read();  //Voi leyendo los datos del puerto serie
                y=datosbyte-48; //Convierto de codigo ASCII a decimal
                Serial.print("El tiempo de estado BAJO es: ");
                Serial.println(y,DEC);// Muestro el tiempo bajo introdcido

                Serial.println("Introduce el tiempo de estado ALTO:");
                f=0;
              }
        }


       digitalWrite(13, HIGH);   // Pongo el Led en estado alto
       int x1=1000*x;            // multiplico los segundos introducidos para
       delay(x1);                // pasarlos a milisegundos
       digitalWrite(13, LOW);    // Pongo el Led en estado bajo
       int y1=1000*y;            // Vuelvo a multiplicar
       delay(y1);                // Y mantengo en estado bajo
}
```

Practica 3: Modulación Por Ancho de Pulso.

3.1 Objetivos

- Conocer la finalidad de la modulación por ancho de pulso.
- Aplicación práctica de la modulación por ancho de pulso.

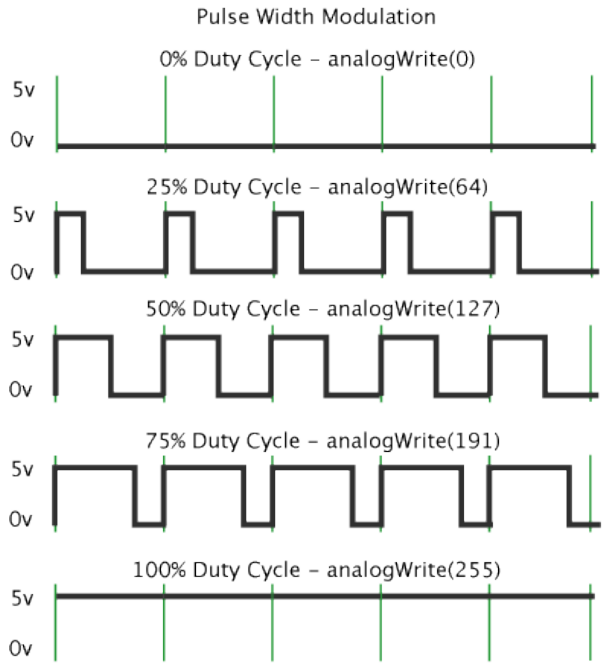
3.2 Material empleado.

- Arduino UNO.
- Diodo Led.

3.3 Conocimientos Teóricos.

La Modulación por Ancho de Pulso (PWM = Pulse Width Modulation) es una técnica para simular una salida analógica con una salida digital. El control digital se usa para crear una onda cuadrada, una señal que conmuta constantemente entre encendido y apagado. Este patrón de encendido-apagado puede simular voltajes entre 0 (siempre apagado) y 5 voltios (siempre encendido) simplemente variando la proporción de tiempo entre encendido y apagado. A la duración del tiempo de encendido (ON) se le llama Ancho de Pulso (pulse width). Para variar el valor analógico cambiamos, o modulamos, ese ancho de pulso. Si repetimos este patrón de encendido-apagado lo suficientemente rápido por ejemplo con un LED el resultado es como si la señal variara entre 0 y 5 voltios controlando el brillo del LED.

En el gráfico de abajo las líneas verticales representan un periodo regular. Esta duración o periodo es la inversa de la frecuencia del PWM. En otras palabras, con la Arduino la frecuencia PWM es bastante próxima a 500Hz lo que equivale a periodos de 2 milisegundos cada uno. La llamada a la función analogWrite() debe ser en la escala desde 0 a 255, siendo 255 el 100% de ciclo (siempre encendido), el valor 127 será el 50% del ciclo (la mitad del tiempo encendido), etc.



3.4 Conocimientos Prácticos: Ejemplo de aplicación.

Se pretende realizar mediante la modulación por ancho de pulso un programa que permita el control de la luminosidad de un led, es decir que empiece desde cero hasta el valor máximo de luminosidad y a continuación decremente. Además queremos visualizar el proceso de incremento y decremento a través del monitor del puerto serie.

Solución:

```
int ledPin1 = 9;    //Incializacion de las variables
int ledPin2 = 10;   //correspondiente a cada pin

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

void loop()  {
  // Mediante un for y una variable aleatoria hacemos el incremeto
  //para poder visualizar el apagado y encendido progesivo del led
  //Recordando que el valor de 0V corresponde con 0 y el de
  //5V con 255
  for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5)
  { //vamos escribiendo el valor de tension que lleva esta variable
    //en el pin correspondiente a cada led
    analogWrite(ledPin1, fadeValue);
    analogWrite(ledPin2, fadeValue);
    Serial.println(fadeValue,DEC);
    // Esperamso un tiempo para observar el efecto
    delay(30);
  }

   // Y de igual forma que incrementamos debemos de decrementar la
   //tension del led
   for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {

     analogWrite(ledPin1, fadeValue);
     analogWrite(ledPin2, fadeValue);
     Serial.println(fadeValue,DEC);

     delay(30);
   }
 }
```

Practica 4: Lectura de un sensor digital de temperatura.

4.1 Objetivos

- Conexión de un sensor con el Arduino

4.2 Material empleado.

- Arduino UNO
- Sensor temperatura
- Programación Labview.

4.3 Conocimientos Teóricos.

Una vez que tenemos conocimientos previos de programación con Arduino y sabemos cómo trabajar con el puerto serie, en esta práctica se pretende obtener la lectura de un sensor, bien sea, de temperatura, humedad o cualquier otro tipo. Vamos a conectar el sensor a la plataforma Arduino y mediante comunicación puerto serie vamos a enviar la información de la lectura al ordenador para un posterior procesado o con cualquier otro fin.

En esta práctica concretamente utilizaremos el sensor de temperatura mostrado en la Figura4.1 donde por un lado tenemos la caja del módulo de adecuación de señales y por otro lado el sensor o termopar.

El módulo de adecuación de señales posee un selector para variar el rango de temperaturas entorno al que nos movemos para tener una mayor precisión y por otro lado tenemos la salida de tensión bien de 0V a 1V o de -0.25V a 0.25V, por ejemplo si seleccionamos la escala de 0 a 100ºC, la salida que obtendremos será para 0ºC de 0V mientras que para los 100ºC obtendremos 1V y para temperaturas intermedios obtendremos su nivel de tensión correspondiente. El valor de tensión será la salida que recogeremos con el Arduino para su posterior estudio y mediante una simple regla de tres podremos obtener el valor de la temperatura en cada instante.

Figura4.1

Y por otro lado tenemos la sonda que será la encargada de captar la temperatura a la cual la estemos sometiendo y que ira conectada al módulo.

4.4 Aplicación Practica

Una vez conocido el funcionamiento del termómetro en esta práctica como bien comentamos anteriormente se pretende realizar un programa que permita recoger los datos obtenidos por el sensor para un posterior procesado.

Practica 5: Control de una Pantalla LCD mediante Arduino

5.1 Objetivos

- Control y conocimiento de la hoja de característica de una pantalla LCD.
- Conexionado de la pantalla LCD con el Arduino.
- Funciones de la librería LiquidCrystal.
- Aplicación práctica del control de la pantalla mediante Arduino.

5.2 Material Empleado.

- Arduino UNO.
- Pantalla LCD.
- Potenciómetro 10K.
- Pila AAA.

5.3 Hoja de características de la pantalla LCD.

Antes de meterle mano a la programación en el Arduino es necesario conocer algunos parámetros límites que posee la pantalla además de la localización de cada una de las patillas que posteriormente necesitaremos para conectar con el Arduino.

En la Tabla5.1 mostramos los límites básicos que posee la pantalla que utilizaremos en nuestra práctica.

Parameter	Symbol	Min	Max	Unit
Supply voltage for logic	VDD	-0.3	7.0	V
Supply voltage for LCD	VDD - VO	-0.3	VDD+0.3	V
Input voltage	VI	-0.3	VDD+0.3	V
Normal operating temperature	TOP	0	50	°C
Normal storage temperature	TST	-10	60	°C
Wide operating / storage temperature (except FSTN)	TOP / TST	-30	80	°C
Wide operating / storage temperature (FSTN)	TOP / TST	-30	70	°C

Tabla5.1- Limites operativos de la pantalla LCD 162B

En la Figura5.1 podemos observar la localización de las patillas que posee la pantalla, para saber cuál es la patilla 1 y cual la 16.

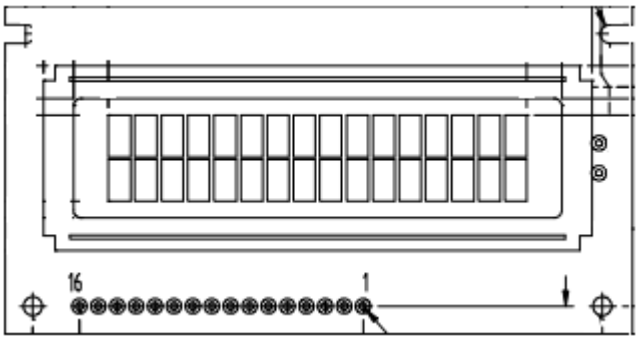


Figura5.1. Localización de las patillas de la pantalla.

Y por último en la Figura5.2 observamos la correspondencia de cada patilla y una explicación posterior de la funcionalidad de cada una de ellas.

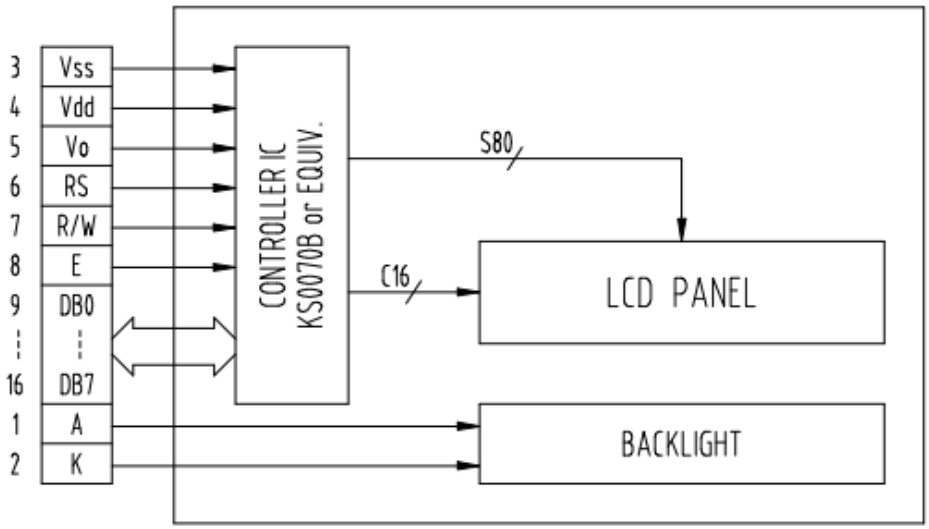


Figura5.2.

El LCD tiene un interfaz paralelo, significando esto que el microcontrolador tiene que manipular varios pines del interfaz a la vez para controlarlo. El interfaz consta de los siguientes pines:

- **Vss**: Tensión GND o de referencia (0V).
- **Vdd**: Tensión de alimentación a la pantalla (+5V).
- **Vo**: Pin de contraste de la pantalla.
- **RS**: Pin de selección de registro que controla en que parte de la memoria del LCD estas escribiendo datos.
- **R/W**: Modo lectura (H) o modo escritura (L).
- **E**: Pin de habilitación de los registros (Enable).
- **DB0…DB7**: se tratan de 8 pines de datos cuyos estados (H o L) son los bits que estas escribiendo a un registro cuando escribes, o los valores de lectura cuando están en el modo lectura.
- **A**: Ánodo del diodo de retro-iluminación (Bklt+).
- **K**: Cátodo del diodo de retro-iluminación (Bklt-).

5.4 Conexionado del LCD con el Arduino

Una vez que tenemos todos los pines del LCD identificados el siguiente paso es el conexionado entre el Arduino y la pantalla como mostramos en la Figura5.3.

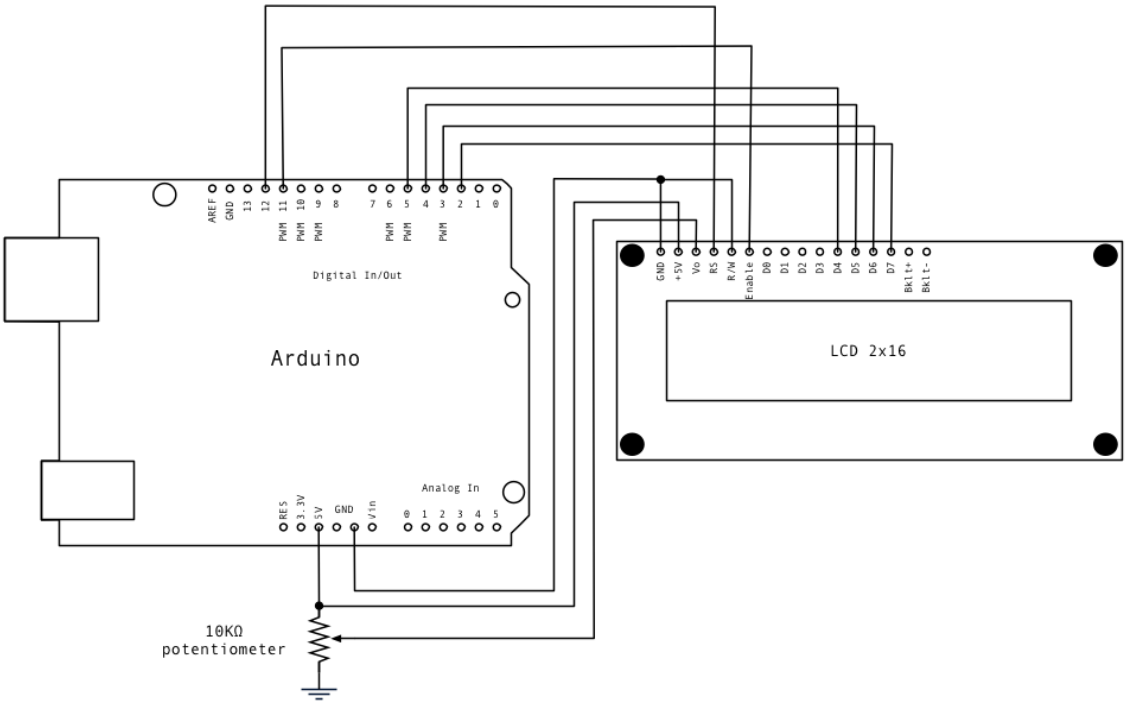


Figura5.3.Conexionado de la placa Arduino con la pantalla LCD.

Se destaca:

- El pin RS del LCD conectado a la E/S digital en el pin 12
- El pin enable del LCD conectado a la E/S digital en el pin 11.
- Los pines D4 - D7 conectado a las E/S digitales desde el pin 5 hasta el 2.
- Los pines de voltaje y tierra conectados a +5V y tierra.
- El pin Vo, que controla el contraste, conectado a un potenciómetro. Ajusta el potenciómetro para que el texto tenga el contraste que tú quieras.

5.5 Funciones de la librería LiquidCrystal

Esta biblioteca permite a la placa Arduino controlar displays LCD. La biblioteca trabaja en modo 4-bit o en 8-bit (es decir, por medio de 4 u 8 líneas de datos, además de RS, ENABLE, y, opcionalmente, las líneas de control RW). Entre las funciones básicas a destacar tenemos:

- **LiquidCrystal():**
  - Descripción:

    Crea una variable de tipo LiquidCrystal. La pantalla se puede controlar por medio de 4 u 8 líneas de datos. En el primer caso, omitir los números de pines para d0 hasta d3 y dejar esos pines no conectados. El pin RW pueden ser conectado a masa en lugar de conectarse a un pin de Arduino, si es así, omítelo de los parámetros de esta función.
  - Sintaxis:

    LiquidCrystal (rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
  - Parámetros:
    - rs: número del pin de Arduino que está conectado al pin RS del LCD
    - rw: número del pin de Arduino que está conectado al pin RW del LCD (opcional)
    - enable: número del pin de Arduino que está conectado al pin ENABLE del LCD
    - d0, d1, d2, d3, d4, d5, d6, d7: números de pines de Arduino que están conectados a los correspondientes pines de datos del LCD. d0, d1, d2, y d3 son opcionales; si se omiten, el LCD será controlado usando solamente cuatro líneas de datos (d4, d5, d6, d7).

- **Begin()**
  - Descripción:

    Especifica las dimensiones (ancho y alto) del display LCD.
  - Sintaxis:

    Lcd.begin(columnas,filas)
  - Parámetros:
    - lcd: una variable de tipo LiquidCrystal.
    - columnas: número de columnas que tiene el display.
    - filas: número de filas que tiene el display.

- **clear**
  - Descripción:

    Borra la pantalla del display LCD y posiciona el cursor en la esquina superior-izquierda.
  - Sintaxis:

    Lcd.clear()
  - Parámetros:
    - lcd: una variable de tipo LiquidCrystal.
- **home()**
  - Descripción:

    Posiciona el cursor en la esquina superior-izquierda del LCD. Se trata de la posición en la que aparecerá el siguiente texto escrito en el display. A diferencia con al anterior que esta no borra el contenido anterior del display.
  - Sintaxis:

    Lcd.home()
  - Parámetros:
    - lcd: una variable de tipo LiquidCrystal.
- **setCursor()**
  - Descripción:

    Establece la posición del cursor donde empezaran aparecer los siguientes caracteres escritos en el LCD.
  - Sintaxis:

    Lcd.setCursor(columna,fila)
  - Parámetros:
    - lcd: una variable de tipo LiquidCrystal.
    - columna: columna donde posicionar el cursor (0-primera columna).
    - fila: fila donde posicionar el cursor (0-primera fila).
- **write()**
  - Descripción:

    Escribe un carácter en el LCD.
  - Sintaxis:

    Lcd.write(data)
  - Parámetros:
    - lcd: una variable de tipo LiquidCrystal.
    - data: el carácter a escribir en el display.

- **print()**
  - Descripción:

    Imprime un texto en el LCD.
  - Sintaxis:

    Lcd.print(data,BASE)
  - Parámetros:
    - lcd: una variable de tipo LiquidCrystal.
    - data: los datos a imprimir (char, byte, int…).
    - BASE (opcional): la base en la que se van a imprimir los números: BIN, DEC, OCT, HEX.

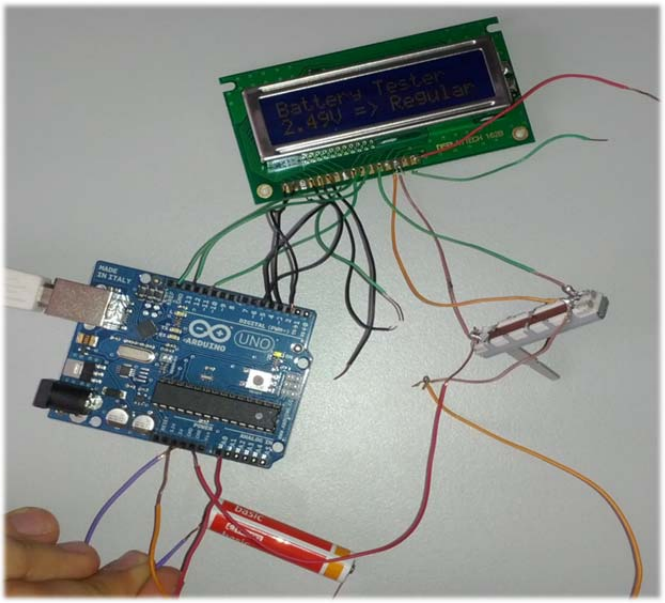
5.6 Ejercicios Prácticos

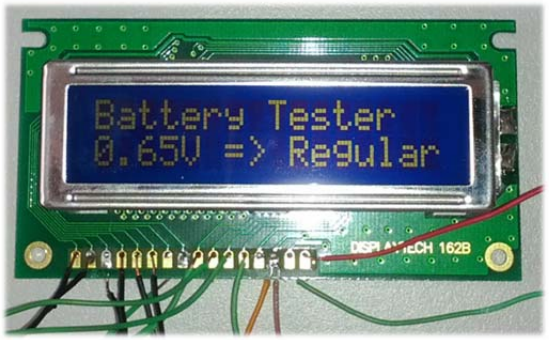
Ejercicio 1:

Una vez que tenemos conocimiento de cómo conectar nuestra pantalla LCD al Arduino y conocemos las funciones básicas de la librería LiquidCrystal que nos permiten realizar la conexión, como ejercicio se pretende realizar un programa que permita transmitir mediante puerto serie una palabra o texto corto que nosotros escribamos y mostrarlo en el Arduino además de que en una segunda línea de la pantalla nos muestre un contador con el número de caracteres introducidos anteriormente.

Ejercicio 2:

En este segundo ejercicio vamos a modificar el programa anteriormente creado para testear el nivel de tensión de una pila AA o AAA. Se pretenderá conectar una pila entre la entrada analógica 0 y tierra, leer el valor de tensión y mostrar en la pantalla LCD el estado de la pila si es Perfecto (>1.40V), Bueno ([1.20V, 1.40V]), Regular ([1.20V, 0.30V]), Malo ([<0.30V]). En la siguiente imagen se observa el resultado de este ejercicio:





Solución Ejercicio1:

```
// Habilitamos la libreria LiquidCrystal
#include <LiquidCrystal.h>

// Inicializacion de los pines a utilizar
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
    // Inicializamos el numero de filas y columnas de la pantalla
  lcd.begin(16, 2);
  // Inicializamos la comunicacion por puerto serie:
  Serial.begin(9600);
}

void loop()
{

  int cont=0; //Declaracion e inicializacion del contador de palabras
  if (Serial.available())
  {
    delay(100);
    // Lipiamos la pantalla
    lcd.clear();

    // Leemos los caracteres introducidos por puerto serie
    while (Serial.available() > 0)
    {
      // Escribimos en la pantalla LCD
      lcd.write(Serial.read());
      //Contador de caracteres;
      cont=cont+1;
    }
    //Bajamos el cursor a la segunda linea de la pantalla
    lcd.setCursor(0,1);
    //Escribimos el valor del contador en pantalla
    lcd.print(cont);
  }
}
```

Solución Ejercicio 2:

```
/*
   Battery Tester
  Programa que trata de mostral el voltaje de una pila en una pantalla LCD,
  estando la pila conectada entre la entrada analaogica 0 y tierra. La pantalla
  LCD dependiendo del nivel de tension de la Pila nos mostrara el estado de
  la misma(Perfecto, Bueno, Regular, Malo).
 */

// Habilitamos la libreria LiquidCrystal
#include <LiquidCrystal.h>

// Inicializacion de los pines a utilizar
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // Inicializamos el numero de filas y columnas de la pantalla
  lcd.begin(16, 2);
  // Mensaje instantaneao al iniciar el programa
  lcd.print("ARDUINO");
  delay(1000);
  lcd.clear(); //Limpiamos el texto anterior
}

void loop() {
  lcd.clear();
  lcd.setCursor(0, 0);          //Ponemos el contador en la posicion 0 de la pantalla:
  lcd.print("Battery Tester"); //Escribimos el texto
  lcd.setCursor(0, 1);         //bajamos el cursor a la segunda linea
  lcd.print(analogRead(0)*5.00/1024); //Conversion del valor leido entre 0 y 1024
                                      //lo pasamos a voltios
  lcd.print("V =>");
  lcd.setCursor(9,1); //Desplazamos el cursor sobre la segunda fila
  //Clasificamos segun el voltaje detectado
  if ((analogRead(0)*5.00/1023.00) > 1.40)
  {
    lcd.print("Perfecto");
  }
  else if ((analogRead(0)*5.00/1023.00) > 1.20 && (analogRead(0)*5.00/1023.00) < 1.40)
  {
    lcd.print("Bueno");
  }
  else if ((analogRead(0)*5.00/1023.00) < 1.20 && (analogRead(0)*5.00/1023.00) > 0.30)
  {
    lcd.print("Regular");
  }
  else if ((analogRead(0)*5.00/1023.00) < 0.30)
  {
    lcd.print("Malo");
  }
  else
  {
  lcd.print("");
  }
  delay(250);//Esperamos para ver el efecto
}
```

Practica 6: Coche Digital

6.1 Objetivos

- Conocer aplicaciones más avanzadas sobre Arduino.
- Poner en práctica conocimientos obtenidos en prácticas anteriores.

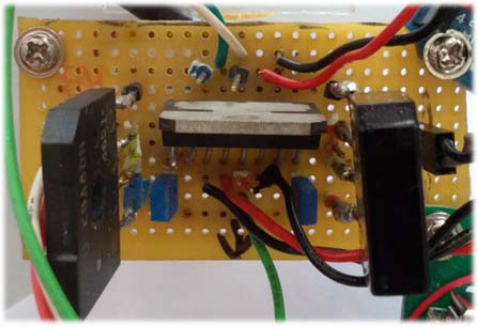
6.2 Material empleado

- Arduino UNO.
- Motores eléctricos 1,6W con reductora.
- Controlador dual de puente completo L298N.
- Pantalla LCD.
- Juego de Fotodiodos ópticos de rango visible.

6.3 Componentes del Coche Digital

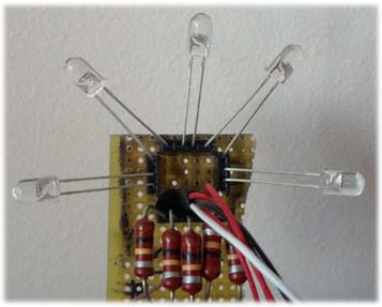
En este apartado vamos a describir cada uno de los elementos que constituye nuestro coche. Empezamos describiendo los dos motores eléctricos de 1,6W con reductora como los de la siguiente figura:

Estos serán los encargados de proporcionar la potencia necesaria a las ruedas para poder realizar el desplazamiento del coche. Estos motores al demandar una potencia que el Arduino es incapaz de dar debemos de conectar entre ambos una etapa de potencia constituida por un Controlador dual de puente completo (L298N) y dos puentes de diodos. La etapa de potencia quedaría como mostramos en la siguiente figura:



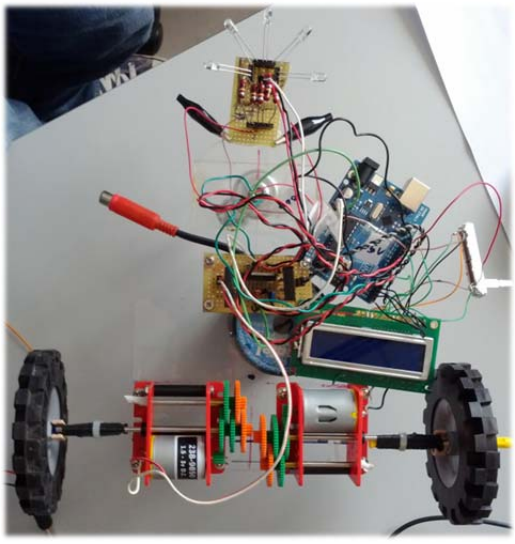
A continuación tenemos conectado el Arduino UNO al cual conectaremos la pantalla LCD que nos indicara la dirección que toma el coche en cada momento.

Y por último el juego de fotodiodos que constituirán el “ojo” del coche ya que son los encargados de captar la luz y girar a esta.



Una pequeña descripción de cómo funciona este juego de fotodiodos es que al incidir la luz más a uno que otro, evaluaremos el máximo y ese estará conectado a una entrada digital del Arduino que dará un nivel alto y se desplazara hacia donde ha captado esa luz.

A continuación mostramos una imagen de como ha quedado compuesto el coche con todo lo mencionado anteriormente:



6.4 Como crear una librería propia para Arduino.

Para comenzar esta práctica necesitamos crear una librería propia sobre las funciones que utilizaremos para los movimientos del coche. Para ello a continuación iremos explicando paso a paso cómo realizar esta librería.

Para crear una librería se necesita al menos dos archivos: un archivo de cabecera (w / con extensión. H) y el código fuente (w / extensión. cpp). El archivo de cabecera contiene definiciones para la librería: básicamente un listado de todo lo que hay dentro, mientras que el archivo del código fuente tiene el código real. Vamos a llamar a nuestra librería "CocheDigital", por lo que nuestro archivo de cabecera será CocheDigital.h. Echemos un vistazo a lo que contiene. Puede parecer un poco extraño al principio, pero tendrá más sentido una vez que se vea el código fuente que lo acompaña.

El archivo de cabecera consiste básicamente en una clase con una línea para cada función de la librería, junto con las variables que se van a usar:

```
class CocheDigital {

  public:
    CocheDigital(int izq_en,int izq_avan,int izq_retro,int der_en,int der_avan,int der_retro);
    void avanza();
    void retrocede();
    void avanza(int tiempo);
    void retrocede(int tiempo);
    void para();
    void frena();
    void derecha();
    void derecha(int tiempo);
    void izquierda();
    void izquierda(int tiempo);
    void derecha_avanza();
    void derecha_avanza(int tiempo);
    void izquierda_avanza();
    void izquierda_avanza(int tiempo);
    void derecha_retrocede();
    void derecha_retrocede(int tiempo);
    void izquierda_retrocede();
    void izquierda_retrocede(int tiempo);
  private:
    int _izq_avan;
    int _izq_retro;
    int _der_avan;
    int _der_retro;
    int _izq_en;
    int _der_en;

};
```

Una clase es simplemente una colección de funciones y variables agrupadas en un mismo lugar. Estas funciones y variables pueden ser públicas, lo que significa que las podrán usar las personas que están utilizando la librería, o privada, lo que significa que sólo se puede acceder a ellas desde la propia clase. Cada clase tiene una función especial conocida como constructor, que se utiliza para crear una instancia de la clase (o sea, un objeto). El constructor tiene el mismo nombre que la clase, y no devuelve nada.

Se necesitan un par cosas más en el archivo de encabezado. Una de ellas es una instrucción # include que da acceso a los tipos estándar y las constantes del lenguaje Arduino (esto se agrega automáticamente a los programas normales, pero no a las librerías). Se parece a esto (y se coloca antes de la definición de la clase mostrada anteriormente):

#include "Arduino.h"

Y por último para tener la cabecera completa necesitamos las dos siguientes líneas de comando, cuya función es preguntar si esta definida la librería y si no lo está la define:

```
#ifndef CocheDigital_h
 #define CocheDigital_h
            …
#endif
```

Con todo esto ya podemos completar nuestro archivo de cabecera de la librería como mostramos a continuación:

```
/*
 CocheDigital.h
 Libreria para el control del Coche desarrollado por el
 PAIDI-TIC_168 con control digital de los motores.
 Abril 2012.
 */


   #ifndef CocheDigital_h
   #define CocheDigital_h

   #include "Arduino.h"

class CocheDigital {

  public:
   CocheDigital(int izq_en,int izq_avan,int izq_retro,int der_en,int der_avan,int der_retro);
    void avanza();
    void retrocede();
    void avanza(int tiempo);
    void retrocede(int tiempo);
    void para();
    void frena();
    void derecha();
    void derecha(int tiempo);
    void izquierda();
    void izquierda(int tiempo);
    void derecha_avanza();
    void derecha_avanza(int tiempo);
    void izquierda_avanza();
    void izquierda_avanza(int tiempo);
    void derecha_retrocede();
    void derecha_retrocede(int tiempo);
    void izquierda_retrocede();
    void izquierda_retrocede(int tiempo);
  private:
    int _izq_avan;
    int _izq_retro;
    int _der_avan;
    int _der_retro;
    int _izq_en;
    int _der_en;

};
   #endif
```

A continuación lo que nos quedaría es explicar las partes que componen el código fuente de CocheDigital.cpp.

Lo primero son un par de # include. Con esto el resto del código tendrá acceso a las funciones estándar de Arduino, y a las definiciones definidas en la cabecera CocheDigital.h.:

```
#include "Arduino.h"
#include "CocheDigital.h"
```

A continuación viene el constructor. Una vez más, en el constructor se establece lo que debe ocurrir cuando alguien crea una instancia de la clase. En este caso, se especifica los pines que vamos a utilizar. Configuramos los pines como salida de las variables privadas para su uso en las otras funciones:

```
CocheDigital::CocheDigital(int izq_en,int izq_avan,int izq_retro,int der_en,int der_avan,int der_retro)
  {
    _izq_avan=izq_avan;
    _izq_retro=izq_retro;
    _der_avan=der_avan;
    _der_retro=der_retro;
    _izq_en=izq_en;
    _der_en=der_en;
  }
```

Se puede observar un par de cosas extrañas en este código. La primera es el *CocheDigital::* antes del nombre de la función. Esto indica que la función es parte de la clase CocheDigital. Verás esto en otras funciones de la clase. Lo segundo es el subrayado en el nombre de las variables privadas, *_variable*. Esta variable puede tener cualquier nombre, siempre y cuando coincida con la definición que figura en el archivo de encabezado. Se le agrega un subrayado al inicio del nombre dejar claro que las variables son privadas, y también para diferenciarlas del argumento de la función, esto es como norma estándar.

Una vez descrito la primera parte del código fuente vamos al cuerpo donde quedar definidas todas las funciones de nuestra propia librería.

Vamos a indicar dos tipos de funciones que podría realizar el coche y como las hemos introducido en la librería y el resto de ellas sería similares.

Por un lado tenemos la función *avanza()* que es la encargada de que el coche ande en dirección recta por lo cual le damos un valor alto a los pines de salidas digitales (*_izq_avan y _der_avan*) que son los que están conectados a ambos motores que moverán las ruedas del coche. El código de la función es:

```
void CocheDigital::avanza()
{
digitalWrite(_izq_avan,HIGH);
digitalWrite(_der_avan,HIGH);
digitalWrite(_izq_retro,LOW);
digitalWrite(_der_retro,LOW);
digitalWrite(_izq_en,HIGH);
digitalWrite(_der_en,HIGH);
}
```

En esta función observamos dos pines habilitados de ENABLE la función de este se ve recogida claramente en la siguiente tabla:

	Avanza	**Retrocede**	**Operación**
E=1	1	0	Avanza
	0	1	Retrocede
	1	1	X
E=0	x	x	X

Se observa que hasta que el enable no este activado no va a realizar la función ordenada por eso en la función anterior le damos la orden prevista de que avance y hasta que no se activa el enable no se va a ejecutar.

Otra posible función que hemos creado es la de *avanza(int tiempo)* la diferencia de esta con la anteriormente descrita es que a esta le introducimos el tiempo que queremos que realice la operación correspondiente.

```
void CocheDigital::avanza(int tiempo)
{
 digitalWrite(_izq_avan,HIGH);
 digitalWrite(_der_avan,HIGH);
 digitalWrite(_izq_retro,LOW);
 digitalWrite(_der_retro,LOW);
 digitalWrite(_izq_en,HIGH);
 digitalWrite(_der_en,HIGH);
 delay(tiempo);
 digitalWrite(_izq_en,LOW);
 digitalWrite(_der_en,LOW);
}
```

En esta se observa a diferencia con la anterior que le damos la instrucción correspondiente activamos el enable de ambos motores esperamos el tiempo introducido y desactivamos el enable para que deje de realizar al instrucción ordenada.

Una vez que tenemos creado tanto el archivo de cabecera como el código fuente el siguiente paso será crear el directorio CocheDigital dentro del subdirectorios de librerías de Arduino. Copiamos tanto al fichero *CocheDigital.h* como *CocheDigital.cpp* en ese directorio. Ahora ejecutamos el IDE de Arduino. Vamos al menú Sketch > Import Library y deberíamos ver la opción CocheDigital. Una vez seleccionado la librería ya será compilada con los programas que utiliza. Si no aparece la librería, debemos de asegurarnos de que los nombres de los archivos terminan realmente en .cpp y .h.

Por ultimo para terminar de completar la librería Arduino no es capaz de reconocer las funciones que hemos creado en el cuerpo y no las resalta, por lo que debemos de crear un archivo llamado keywords.txt en el directorio de CocheDigital. Este fichero tendrá este aspecto:

CocheDigitalKEYWORD1 avanzaKEYWORD2 retrocedeKEYWORD2

Cada línea tiene el nombre de la palabra clave, seguido de un tabulador (no espacios), seguido por el tipo de palabra clave. Las clases deben ser del tipo KEYWORD1 y se muestran de color naranja y las funciones deben ser del tipo KEYWORD2 y serán de color marrón.

6.5 Control del Coche basándose en la librería creada

Una vez que sabemos cómo está constituido el coche y hemos creado la librería con todas las funciones posibles, vamos a crear unos programas para su control.

Programa 1:

El primer programa consiste en mediante puerto serie introducir una letra y que el coche realice la operación correspondiente durante un tiempo establecido, utilizando las funciones que tenemos que introducir un tiempo. Las letras por ejemplo pueden establecerse de la siguiente forma:

a-avanza
p-para
d-derecha
i-izquierda
r-retrocede
b-derecha avanza
c-izquierda avanza
e-derecha retrocede
f-izquierda retrocede
0-frena

El código del programa junto con la explicación de cada línea seria:

```
//Incluimos la libreria correspondiente
#include <CocheDigital.h>

CocheDigital coche(4,5,6,8,9,10);
/*Pin 4 = izq_en
  Pin 5 = izq_avan
  Pin 6 = izq_retro
  Pin 8 = der_en
  Pin 9 = der_avan
  Pin 10= der_retro*/
/*Declaramos una variable tiempo que sera la
encargada de establecer el tiempo que queremos
que realiza la operacion correspondiente.*/
int tiempo=1000;

void setup()
{
  Serial.begin(9600);//Inicializamos el Puerto
  //Serie ya que es por donde introduciremos el
  //carecter para que realize una o otra operacion.
}
char inByte;
void loop()
{
  if (Serial.available() > 0) {//Comprobamos si el puerto esta disponible
    inByte = Serial.read();//Guardamos el valor introducido en una variable
    //Y ya ejecutamos la operacion correspondiente dependiendo del caracter
    //introducido, mostrandonos por puerto serie la operacion realizada
```

```
    if (inByte=='a'){
      Serial.println("Avanza");
      coche.avanza(tiempo);
    }
    else if (inByte=='p'){
      Serial.println("Para");
      coche.para();
    }
        else if (inByte=='d'){
          Serial.println("Derecha");
      coche.derecha(tiempo);
     }
        else if (inByte=='i'){
      Serial.println("Izquierda");
      coche.izquierda(tiempo);
     }
        else if (inByte=='r'){
      Serial.println("Retrocede");
      coche.retrocede(tiempo);
     }
        else if (inByte=='b'){
      Serial.println("Derecha avanza");
      coche.derecha_avanza(tiempo);
    }
        else if (inByte=='c'){
      Serial.println("Izquierda avanza");
      coche.izquierda_avanza(tiempo);
    }
        else if (inByte=='e'){
      Serial.println("Derecha retrodece");
      coche.derecha_retrocede(tiempo);
     }
        else if (inByte=='f'){
      Serial.println("Izquierda_retrocede");
      coche.izquierda_retrocede(tiempo);
     }
    else if (inByte=='0'){
      Serial.println("Frena");
      coche.frena();
    }
    else Serial.println("Comando de control incorrecto");
  }
}
```

Programa 2:

Ahora este segundo programa vamos a complicar un poco más la cosa y vamos hacer que el coche realice la operación correspondiente por sí solo, para ello como comentamos al principio de la práctica tenemos un juego de fotodiodos que serán los encargados de transmitirle una señal al Arduino que controlara el movimiento de coche dependiendo de la cantidad de luz y el mismo se orienta hacia ella hasta encontrarla y una vez que se enfrente avanza en dirección a la luz.

Como complemento a este problema hemos instalado la pantalla LCD que ya controlamos en prácticas anteriores, y en esta pantalla lo que nos va indicar es la dirección que está tomando el coche en cada momento.

A continuación se muestra el código del programa con la explicación en cada una de las líneas de comando:

```
/*Inicializacion de ambas librerias */
#include <LiquidCrystal.h>
#include <CocheDigital.h>

/*Le damos a ambas Librerias los pines que tienen habilitados */
CocheDigital coche(4,5,6,8,9,10);
/*Pin 4 = izq_en
  Pin 5 = izq_avan
  Pin 6 = izq_retro
  Pin 8 = der_en
  Pin 9 = der_avan
  Pin 10= der_retro*/
LiquidCrystal lcd(11, 7, 3, 2, 1, 0);

void setup()
{
/*Utilizamso los pines 13 y 12 como fuentes de
5V para alimentar a la pantalla y al juego de fotodiodos */
  pinMode(13,OUTPUT);
  digitalWrite(13,HIGH);
  pinMode(12,OUTPUT);
  digitalWrite(12,HIGH);

  //Inicializacion de la pantalla 16 filas y 2 columnas
  lcd.begin(16, 2);
  //Mesaje inicial en la pantalla
  lcd.print("ARDUINO");
  delay(1000);
  lcd.clear(); //Limpiamos la pantalla
}
/*Definicion de variables utilizadas */
int Sensor[5];
int maximo1,maximo2,maximo;
```

```
void loop()
{
  /*Cada lectura de los fotodiodos la
  introducimos en una posicion del vector Sensor */
  Sensor[0]=analogRead(A0);
  Sensor[1]=analogRead(A1);
  Sensor[2]=analogRead(A2);
  Sensor[3]=analogRead(A3);
  Sensor[4]=analogRead(A4);

  /* Buscamos el maximo entre la tension
  proporcionada por los sensores*/
  maximo1=max(Sensor[0],Sensor[1]);
  maximo2=max(Sensor[2],Sensor[3]);
  maximo=max(maximo1,maximo2);
  maximo=max(maximo,Sensor[4]);

  /*Y dependiendo de donde detecta el maximo de
  tension que corresponde con el maximo de haz de
  luz detectado realizara un movimiento u otro
  Y en la pantalla se escribira el desplazamiento
  correspondiente*/
 if (Sensor[0]==maximo||Sensor[1]==maximo)
 {
   coche.derecha(500);
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("Derecha");
   lcd.setCursor(8, 0);
   lcd.print("<--");
 }
 if (Sensor[3]==maximo||Sensor[4]==maximo)
 {
   coche.izquierda(500);
   lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("Izquierda");
   lcd.setCursor(10, 0);
   lcd.print("-->");
 }
 if (Sensor[2]==maximo)
 {
   if ((Sensor[1]-Sensor[3])>50)
   {
     coche.derecha(200);
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("Derecha");
     lcd.setCursor(8, 0);
     lcd.print("<--");
   }
```

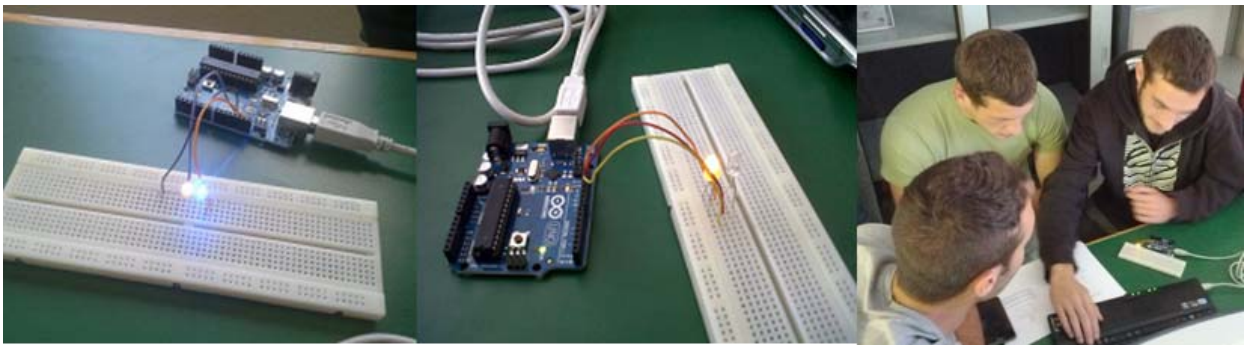
```
      else if ((Sensor[1]-Sensor[3])<-50)
      {
        coche.izquierda(200);
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Izquierda");
        lcd.setCursor(10, 0);
        lcd.print("<--");
      }
      else
      {
        coche.avanza(200);
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Avanza");
        lcd.setCursor(8, 0);
        lcd.print("|");
        lcd.setCursor(8, 1);
        lcd.print("V");
      }
  }

}
```

Objetivo 1: Programación de micro-controladores, dedicados a medidas de diversas magnitudes físicas. Con esta actividad potenciaremos el uso de instrumentos de medida dedicados a una tarea concreta, muy presentes en instrumentos de medida de mano (portátiles) y en otros dispositivos portables.

Actividades realizadas y resultados obtenidos:

Para la consecución de este objetivo ha sido necesaria la adquisición y uso de la placa micro-controlada programable ARDUINO como herramienta didáctica que, gracias a su versatilidad y fácil instalación, está experimentando una amplia difusión en la docencia electrónica. Concretamente se han comprado 8 placas básicas ARDUINO UNO (apropiadas para la iniciación en la programación de micro-controladores y con capacidad para el desarrollo de aplicaciones de nivel intermedio), y 2 placas de un nivel superior ARDUINO MEGA (que permiten abordar proyectos de mayor entidad para brindar posibilidades a la hora de realizar Proyectos Fin de Carrera y Trabajos Fin de Máster).

- Durante el curso 2011/2012, dentro de las prácticas de laboratorio de la asignatura Electrónica en el Grado de Ingeniería en Tecnologías Industriales, se ha desarrollado una práctica con microcontroladores ARDUINO pensada como una primera toma de contacto con dispositivos de electrónica avanzada –en este punto del grado los alumnos sólo han trabajado con elementos electrónicos básicos como resistencias, condensadores o diodos. La práctica consiste en el control de LEDs mediante la programación de la placa de manera que el alumno debe conseguir ciertas secuencias de iluminación problema. Para conseguir estas secuencias se deben usar tanto las salidas analógicas como digitales de la placa. Así, con esta práctica el alumno adquiere un conocimiento básico sobre la programación y conexión de la placa que será muy útil cuando se vuelva a reincidir sobre este dispositivo en cursos posteriores. Los resultados han sido muy satisfactorios teniendo en cuenta el interés que han mostrado los alumnos por ARDUINO. De hecho nos comentaban que era la primera vez que veían como un programa repercutía en un elemento físico externo al ordenador y pedían información sobre el precio y la forma de adquirir estas placas.

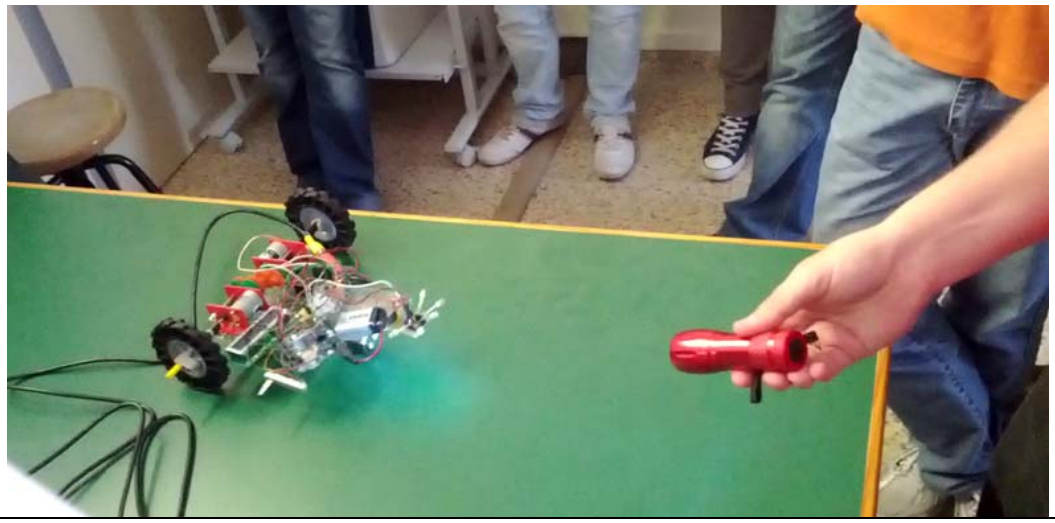


Imágenes extraídas de las memorias de prácticas de los alumnos de Electrónica de Grado.

- En la asignatura Instrumentación Electrónica de la titulación Ingeniería Técnica Industrial en Electrónica Industrial (curso 2011/2012) se ha realizado Cinco prácticas ARDUINO desde iniciación a nivel intermedio, en el control y gestión de entradas y salidas analógicas y digitales y aplicaciones de medida (Se incluyen como anexos). De nuevo los resultados han sido satisfactorios en función de la disposición de los alumnos a trabajar con este tipo de esquemas donde dispositivos físicos e informática interaccionan. Dado que una parte importante de la práctica consiste en el desarrollo de paneles con LabVIEW este punto será de nuevo comentado y ampliado en el Objetivo 2.

- Con el fin de mostrar las posibilidades de ARDUINO, y dado que las prácticas realizadas durante este curso pueden considerarse de iniciación, creímos oportuno desarrollar un dispositivo de nivel avanzado y hacer una demostración para concluir las prácticas anteriormente descritas. El dispositivo en cuestión era un autómata equipado con unos sensores lumínicos direccionales. Estos sensores se conectaban a las entradas de la placa de manera que ésta podía discernir qué sensor estaba más iluminado y por tanto la dirección de la provenía la mayor cantidad de luz. Procesando esta información la placa activaba unos motores acoplados a ruedas para girar y avanzar hacia la dirección en que provenía la luz desarrollando así un comportamiento "inteligente" (ver http://www.youtube.com/watch?v=4cedJHsK_Hc). Después de haber trabajado durante la práctica con ARDUINO, y tras una somera explicación de los componentes y la programación empleada en el autómata, los alumnos eran capaces de entender el funcionamiento e interacción de elementos y por tanto el modo en

que los sistemas electrónicos avanzados reciben, procesan y responden de manera compleja a la información física exterior gracias al uso de microprocesadores. De nuevo el nivel de interés de los alumnos fue muy elevado, mostrando algunos de ellos su predisposición a realizar proyectos futuros semejantes y si éstos justificarían un proyecto fin de carrera o fin de master.

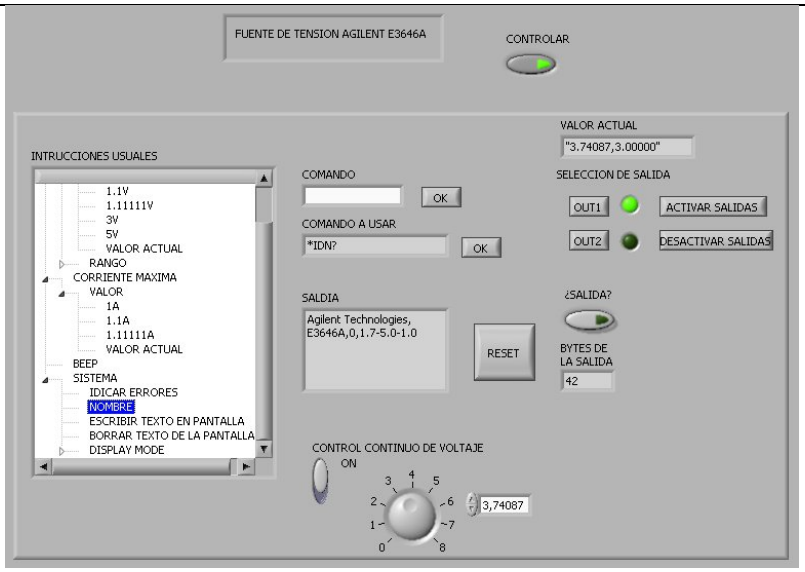


Objetivo 2: Diseño de paneles de instrumentación virtual mediante LabVIEW, que es la referencia mundial en esta materia. Con esta actividad se pretende potenciar la programación en LabVIEW para los títulos, y que hemos aprendido como investigadores en nuestro grupo PAIDI-TIC-168 en Instrumentación Computacional y Electrónica Industrial (ICEI).

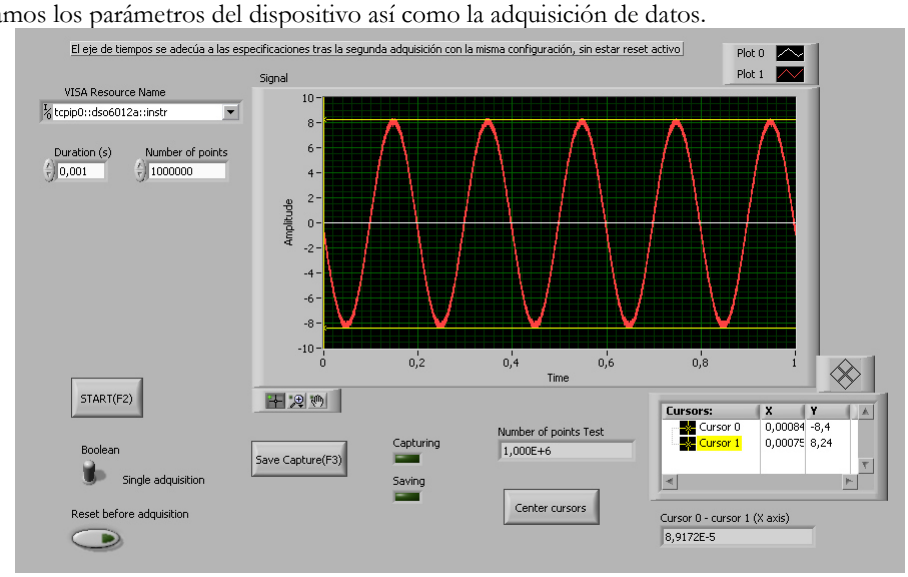
Actividades realizadas y resultados obtenidos:

Este objetivo ha tenido durante este curso su reflejo en una asignatura, mientras que se han sentado las bases para la utilización en otras asignaturas en los próximos cursos.

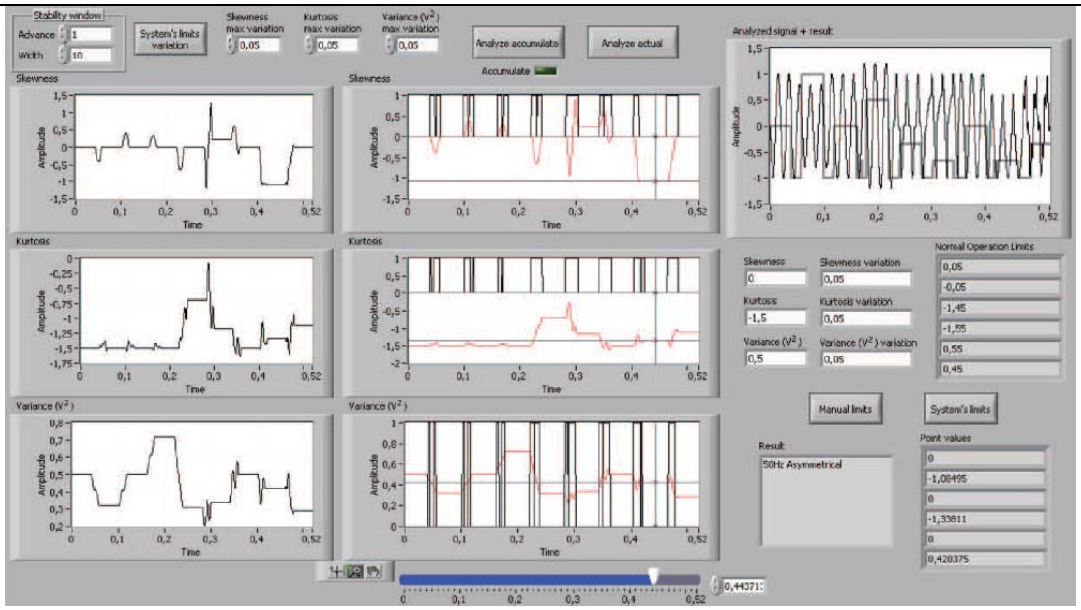
- **Práctica de monitorización de Temperaturas en** Instrumentación Electrónica (I.T.I. E.I.). consistente en la conexión de sensores de temperatura PhilipHarris E60170/1 a través de las entradas analógicas de la placa de manera que los registros son almacenados en la memoria de ésta y enviados a través de puerto serie al ordenador, donde una interfaz de LabVIEW procesa los datos y los representa en un panel diseñado para ello.
- **Control de fuente de CC Agilent E2646A**. Este desarrollo se usará en cursos próximos como ejemplo de control de sistemas a través de puertos GPIB. Para ello se ha desarrollado un módulo en LabVIEW con un panel que controla todos los parámetros del dispositivo (voltaje, intensidad máxima, display,…) cuyo resultado es la posibilidad de obtener señales variables en el tiempo a partir de una fuente continua. Así los alumnos trabajan con conexionado, programación LabVIEW, dispositivos electrónicos y pueden testear el resultado total de la implementación a través de la señal de salida.



- **Control del osciloscopio Agilent DSO6012A**. Este desarrollo se usará en cursos próximos como ejemplo de control de sistemas a través de tarjetas Ethernet. Esto permite el acceso vía LAN al osciloscopio desde cualquier punto Escuela Politécnica Superior de Algeciras. De nuevo, mediante una interfaz LabVIEW controlamos los parámetros del dispositivo así como la adquisición de datos.



- **Instrumento para la medición de la calidad de la energía eléctrica.** Nuestro grupo de Investigación PAIDI-TIC-168 en Instrumentación Computacional y Electrónica Industrial (ICEI) tiene concedido actualmente un proyecto nacional que trata sobre la monitorización de la calidad en el suministro de energía eléctrica TEC2010-19242-C03-03 (SIDER-HOSAPQ). Hemos creído interesante implementar los resultados de este proyecto en paneles LabVIEW. De este modo, y al igual que ocurría con el autómata en el Objetivo 1, los alumnos pueden observar/manejar/testear una aplicación de nivel avanzado en LabVIEW, completando y reforzando el conocimiento adquirido en las actividades planteadas con esta herramienta.



Objetivo 3: Integración de instrumentos de medida en internet y control de los mismos desde cualquier ordenador de la universidad, con extensibilidad hacia fuera de la universidad. Se pretende iniciar una vía formativa práctica en este ámbito.

Actividades realizadas y resultados obtenidos:

- **Control del osciloscopio Agilent DSO6012A**. Como se ha comentado en el Objetivo 2, el control de este osciloscopio se hace vía LAN. Gracias a una dirección IP propia suministrada por el CITI**,** el alumno puede comprobar como desde cualquier punto de la red local puede tener acceso al control básico del osciloscopio gracias a la interfaz Web que proporciona el fabricante; o bien, usando la interfaz LabVIEW comentada en el Objetivo 2 puede tener acceso al control y a la adquisición de datos.

- **Sistema de alarmas vía Internet:** Dada la compatibilidad que ofrece LabVIEW con .NET framework estamos desarrollando aplicaciones que envíen mensajes de correo electrónico ante ciertos eventos. Actualmente, hemos desarrollado un sistema de alarmas para el **Instrumento para la medición de la calidad de la energía eléctrica** (ver Objetivo 2) que se activa cuando el sistema se ha bloqueado. Se pretende, utilizando los mismos elementos, dotar de sistemas de alarma y de transmisión de datos a las aplicaciones LabVIEW propuestas a alumnos.

Objetivo 4: Diseño de aplicaciones informáticas para la docencia de circuitos electrónicos

Actividades realizadas y resultados obtenidos:

Se han realizado una serie de simulaciones sobre contenidos clave en la docencia de las asignaturas (Electrónica y Electrónica de potencia (G.I.T.I.)). Gracias a ellos el alumno podrá trabjar/estudiar/comprobar el comportamiento de elementos, circuitos o propiedades electrónicas cambiando parámetros característicos. Del mismo modo, podrá usar estas aplicaciones como generador de problemas resueltos adaptados a la nomenclatura y metodologías usados en las asignaturas concernidas. Se ha desarrollado en OpenOffice para facilitar su difusión.

- **Electrónica de Grado**

Bandas de Energía
Concentración – Cálculos de concentración de átomos para distintos elementos
Conducción – Se calculan parámetros de conducción en metales
Nivel de Fermi – De distintos metales

Semiconductores y unión PN
Semiconductores I – Concentraciones, velocidades de portadores mayoritarios y minoritarios y tipología
Semiconductores II – Cálculo de la distancia del Nivel de Fermi
Efecto Hall – Conductividad y movilidad de portadores
Diodo – Resolución de circuito típico Diodo y Resistencia en Serie (Gráficas: Recta de carga del circuito y V-I del diodo)

Transistor JBT
Polarización fija – Resuelve el circuito indicando la región de trabajo del transistor (Gráficas: Rectas de carga en activa)
Colector-Base –
Autopolarizado-

Transistor JFET
Análisis JFET – Definiendo el estado de los terminales, determina la región de trabajo y la corriente que circula por el drenador
Análisis MOSFET deplexión –
Análisis MOSFET acumulación –
Dos NMOS serie – Problema típico.

- **Electrónica de potencia**

Circuitos básicos de electrónica
RC carga
RLC paralelo
RLC serie

Convertidores CC-CC
Elevador
Reductor
Inverso

Rectificadores Monofásicos Controlados
Media onda con carga resistiva
Onda completa con carga resistiva
Onda completa con carga R-L

Objetivo 5: Diseño de actividades de evaluación a distancia que consistan en tomar medidas y programar los instrumentos vía internet, programar micro-controladores a partir de una batería de ejercicios con un marcado carácter práctico.

Actividades realizadas y resultados obtenidos:
Para la consecución de este objetivo es necesaria la completa funcionalidad de los elementos descritos en los Objetivos anteriores. Por ello el desarrollo total de este punto se plantea a más largo plazo. Avances necesarios en este sentido son:

- Programación de microcontroladores (Objetivo 1)
- Control del osciloscopio Agilent DSO6012A (Objetivo 2 y Objetivo 3)
- Sistema de alarmas vía Internet (Objetivo 3)