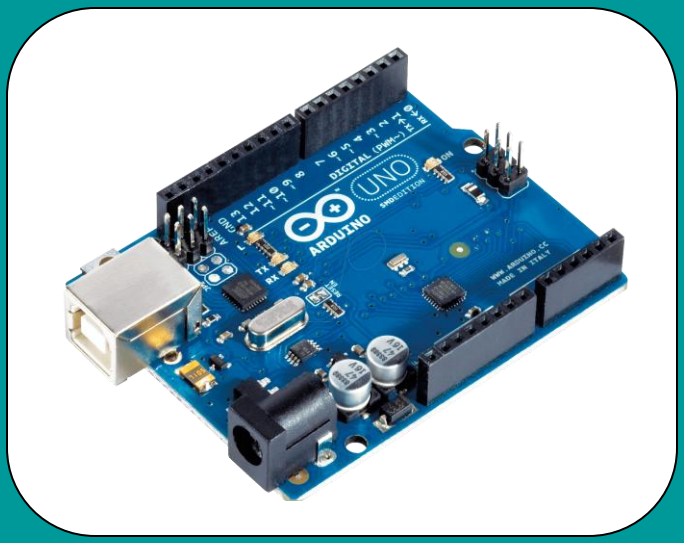
MANUAL DE PRACTICAS

CON ARDUINO UNO R3



ING. ROBERTO PATIÑO RUIZ

RESUMEN

Arduino es una plataforma de hardaware abierto para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

INTRODUCCIÓN

Arduino puede tomar información del entorno a través de sus pines de entrada, de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El Microcontrolador en la placa arduino se programa mediante el lenguaje de programación arduino (basado en wiring) y el entorno de desarrollo arduino (basado en processing). Los proyectos hechos con arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, processing, maxmsp).

Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, con la libertad de adaptarse a las necesidades de los estudiantes.

El presente manual de prácticas basado en la placa Arduino Uno R3, se elaboró con la finalidad de apoyar a los estudiantes Tecnólogos y Universitarios del CETI y UdeG respectivamente, en la construcción gradual de sistemas basados en Microprocesador, especialmente pensados para su uso en un amplio rango de aplicaciones industriales y de gran consumo, como dispositivos embebidos. Las prácticas descritas en este material se derivan de fuentes bibliográficas, así como de documentos libres de internet.

¿Qué es lo primero que debe hacer el estudiante para empezar a programar y compilar sus aplicaciones?

En realidad es bastante sencillo, el primer paso es descargar el software de arduino del siguiente enlace:

http://arduino.cc/en/Main/Software

"El verdadero progreso es el que pone la tecnología al alcance de todos."
H. Ford

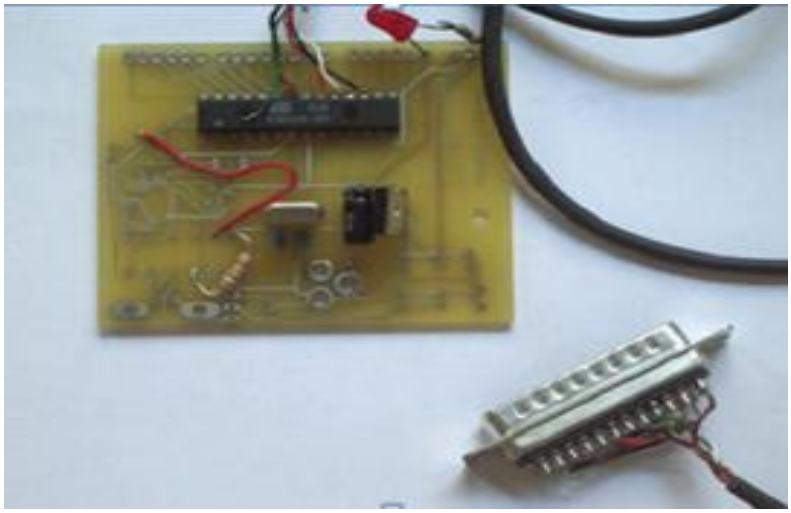
INDICE

MANUAL DE PRACTICAS CON ARDUINO R3

ANTECENDENTES DEL SISTEMA ARDUINO

En el año 2003 en Italia específicamente en el instituto IVREA, el docente **Massimo Banzi** enseñaba el uso de Microcontroladores PIC´s a estudiantes de Diseño Interactivo, los cuales no tenían el conocimiento técnico para utilizar las herramientas de bajo nivel para la programación que existían en esos momentos, por tal motivo desarrollo una herramienta de programación de PIC´s bajo la plataforma MAC (ya que la mayoría de los estudiantes utilizaban esta plataforma), esta herramienta y el ambiente de desarrollo Processing sirvieron como ejemplo a el colombiano Hernando Barragán que en ese momento era estudiante del instituto, para desarrollar la tarjeta Wiring, el lenguaje de programación y su ambiente de desarrollo.

Poco tiempo después Massimo, David Cuartilles investigador en el instituto y Gianluca Martino desarrollador local contratado para desarrollar hardware para los proyectos de los estudiantes, desarrollaron en 2005 una tarjeta basada en el trabajo de Hernando Barragán, la cual era más pequeña y económica que la Wiring a la cual llamaron **Arduino.** Más tarde se unieron a este grupo los estudiantes Mellis y Zambetti que mejoraron el modelo de Wiring, logrando construir una tarjeta básica y un ambiente de desarrollo completo.



En el mismo año se une a este equipo de trabajo Tom Igoe quien es conocido por sus trabajos en Computación Física (Construcción de sistemas físicos a través de hardware y software que pueden sentir y responder al mundo análogo), y se encarga de las pruebas del sistema con estudiantes del ITP en Estados Unidos, así como de realizar los contactos para la distribución de la tarjeta en territorio americano.

¿Qué es el Arduino?

Arduino es una plataforma de hardware de código abierto, basada en una sencilla placa con entradas y salidas, analógicas y digitales, en un entorno de desarrollo que está basado en el lenguaje de programación Processing. Es un dispositivo que conecta el mundo físico con el mundo virtual, o el mundo analógico con el digital.
Arduino es una plataforma de hardware en código abierto para gente que quiere hacer una introducción a la electrónica sin necesidad de tener conocimientos previos en ciencias. Se puede decir que todo el sistema de desarrollo, el software, los circuitos y la documentación son abiertos.

Las plataformas Arduino están basadas en los Microcontroladores Atmega168, Atmega328, Atmega1280, ATmega8 y otros similares, chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños. Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data).

La filosofía **open source** -código abierto- que lo sustenta, tanto el modo en que se ensambla la placa -hardware- como el código fuente del programa Arduino -software-, son de acceso público. Esto quiere decir que cualquiera puede usarlo y/o mejorarlo. Al ser **open hardware**, tanto su diseño como su distribución es libre. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haberse adquirido ninguna licencia.

En la feria Maker Fair del 2011, se presentó la primera placa Arduino de 32 bits para trabajar tareas más pesadas. Entre ellas se presentó la impresora en 3D de MakerBot capaz de imprimir en resina cualquier modelo en 3D.

ESTRUCTURA DE UN PROGRAMA EN LENGUAJE C

La estructura básica del lenguaje de programación del Arduino es muy simple, y corre en al menos 2 partes. Estas 2 partes requeridas, o funciones, encierran bloques de sentencias.

Código

```
void setup()

{

sentencias;

}

void loop()

{

sentencias;

}
```

Donde setup() es la preparación, loop() es la ejecución. Ambas funciones son requeridas para que el programa trabaje a la función “setup” debe seguir la declaración de las variables al principio del programa. Es la primer función para correr en el programa, se corre solo una vez, y es usada para fijar el modo de los pins (pinMode), o inicializar la comunicación serial.

La función “loop” es la que le sigue e incluye el código para ser ejecutada continuamente leyendo las entradas, disparando las salidas, etc. Esta función es el núcleo de todos los programas arduino, y hace la mayor parte del trabajo.

setup()

La función “setup()” es llamada una vez cuando arranca tu programa. Se usa para inicializar el modo de los pins (pinMode), o inicializar el modo serial. Se debe incluir en un programa aun cuando no haya sentencias para correrlas.

Código

```
void setup()

{

pinMode(pin, OUTPUT); // programa un 'pin' como salida

}
```

loop()

Después de llamar a la función setup(), la función loop(), hace precisamente lo que su nombre sugiere, y hace ciclos consecutivamente, permitiendo al programa cambiar, responder, y controlar a la tarjeta Arduino.

Código

```
void loop()

{

digitalWrite(pin, HIGH);  //pone el 'pin' en uno lógico

delay(1000);                    //se espera un segundo

digitalWrite(pin, LOW);         //pone el 'pin' en cero lógico

delay(1000);                    //se espera un segundo

}
```

Funciones

Una función es un block de código que tiene un nombre y un block de sentencias que son ejecutadas cuando la función es llamada. Las funciones void setup(), void loop() ya han sido revisadas y otras serán discutidas después.

Las funciones comunes pueden ser escritas para llevar a cabo tareas repetitivas y reducir el desorden en un programa. Las funciones se declaran, primero declarando el tipo de la función. Esto es el tipo de valor que va a regresar la función, tales como “int” para una función de tipo entero. Si no hay valor para ser regresado, el tipo de la función debe ser evitado (void). Después del tipo de función, se declara el nombre dado a la función, y entre paréntesis, cualquier parámetro que se pase a la función.

Código

```
type functionName(parametros)

{

sentencias;

}
```

La siguiente función de tipo entero “delayVal()”, se usa para fijar un valor de retardo en un programa por medio de leer un valor de un potenciómetro. Primero declara una variable local “v”, fija el valor del potenciómetro en “v”, lo cual da un valor entre 0-1023, luego divide ese valor entre 4 para un valor final entre 0-255, y finalmente regresa ese valor al programa principal.

Código

```
int delayVal()

{

int v;                          // crea una variable temporal 'v'

v = analogRead(pot);            // lee el valor del potenciómetro

v /= 4;                         // convierte 0-1023 a 0-255

return v;                       // regresa el valor final

}
```

{ } Llaves (curly braces)

Las llaves, definen el principio y el fin de una función bloque y el final de un bloque de sentencias tales como la función void loop(), y las sentencias “for” e “if”

Código

```
type function()

{

sentencias;

}
```

Una llave de apertura “{“ debe siempre ser seguido de una llave de cierre “}”, de lo contrario puede haber errores de compilación. El ambiente arduino incluye una característica conveniente para revisar el balance de las llaves. Solo selecciona una llave y se destaca a su compañera de cierre.

";" punto y coma (semicolon)

El punto y coma se debe utilizar para terminar una sentencia y separar elementos de un programa. También es usado para separar elementos de un ciclo “for”.

Código

```
int x = 13;    // declara la variable  'x'  como el entero 13
```

Nota: El olvidar terminar una línea con punto y coma puede resultar en un error de compilación. El error de texto puede ser obvio, e indicar el punto y coma faltante, o puede que no.

/*… */ bloque de comentarios (block comments)

Son áreas de texto ignoradas por el programa y son utilizadas para descripciones largas del código o comentarios que puedan ayudar a otros a entender partes del programa. Se inicia con /* y termina con */ y puede agrupar múltiples líneas.

Código

```
/* Este es un bloque multilinea de comentarios, no olvide cerrar el comentario, tienen que  estar balanceados los signos de apertura y cierre */
```

// línea de comentarios

Los comentarios de línea simple comienzan con // y terminan con la siguiente línea de código. Estos comentarios son ignorados por el programa y no ocupan espacio de memoria, al igual que el block de comentarios.

Código

```
// Este es un comentario de una sola linea
```

Variables

Una variable es un modo de nombrar y almacenar un valor numérico para uso posterior del programa. Una variable necesita ser declarada y opcionalmente se le asigna un valor que necesita ser almacenado. El siguiente código declara una variable llamada “inputVariable”, y luego se le asigna el valor obtenido en una entrada analógica (pin 2):

Código

```
int inputVariable = 0;                  // declara una variable de tipo entero y se le asigna el //valor “0”

inputVariable = analogRead(2);          // se le asigna a la variable, el valor análogo de entrada //en el pin 2
```

“inputVariable” es la variable misma. La primera línea declara que la variable es de tipo entero. La segunda línea asigna a la variable el valor análogo del pin 2. Esto hace que el valor del pin 2 sea accesible en cualquier lugar del código.

Una vez que la variable ha sido asignada, o reasignada, puedes verificar su valor para ver si esta encuentra ciertas condiciones, o puedes usar directamente su valor. Como un ejemplo para ilustrar 3 operaciones con variables, el siguiente código verifica si “inputVariable” es menor que 100, si es verdadero le asigna el valor de 100 a “inputVariable”, y después fija un retardo basado en “inputVariable” el cual ahora es un mínimo de 100:

Código

```
if (inputVariable < 100)            // verifica si la variable < 100

{

inputVariable = 100;                // si es verdadero le asigna 100

}

delay(inputVariable);               // usa la variable como retardo
```

Nota: *Las variables deben ser dadas con nombres descriptivos, para hacer el código más comprensible. Los nombres de variables como tiltSensor o pushButton ayuda al programador para entender que representa la variable. Los nombres de variables como "var" o "value", de otro modo, hacen poco para que el código sea más comprensible y aquí son usados solo como ejemplos. Una variable puede ser nombrada de cualquier forma que no sea una palabra reservada para el lenguaje Arduino.*

Declaración de una variable

Todas las variables tienen que ser declaradas antes de que ser usadas. Declarar una variable significa definir su tipo de valor como en int, long, float, etc. fijando un nombre específico y opcionalmente asignando un valor inicial. Esto solo necesita ser hecho una vez en un programa pero el valor puede ser cambiado en cualquier tiempo usando aritmética y varias asignaciones.

El siguiente ejemplo declara que la variable de entrada es un entero (int), o un tipo integrado, y que su valor inicial equivale a "0"

Código

```
int inputVariable = 0;
```

Una variable puede ser declarada en un número de ubicaciones a través de programa y donde esta definición toma lugar, determina que parte del programa puede usar la variable.

Ámbito de la variable

Una variable puede ser declarada al principio del programa antes del "void setup()", localmente dentro de funciones, y algunas veces dentro de un block de sentencias, tales como los lazos "for". Donde la variable es declarada determina el ámbito de la variable, o la disponibilidad de ciertas partes de un programa para hacer uso de la variable.

Una variable global: es aquella que puede ser vista y usada por todas las funciones y sentencias de un programa. Esta variable se declara al principio del programa antes de la función "setup()".

Una variable local: es aquella que es definida dentro de una función, o como parte de un lazo “for”. Solo es visible y solo puede ser usada dentro de la función en la cual fue declarada. Entonces es posible tener 2 o más variables del mismo nombre en diferentes partes del mismo programa que contiene diferentes valores. Asegurando que solo una función tiene acceso a sus variables, simplifica el programa y reduce el potencial para programar errores.

El siguiente ejemplo muestra como declarar unos pocos tipos diferentes de variables, y demuestra la visibilidad de cada variable.

Código

```
int value;                  // 'value' es visible para cualquier función


void setup() {}             // no se necesita la función setup
}
void loop()
{
for (int i=0; i<20;)        // 'i' es solo visible dentro de este for-loop
  {
    i++;
  }
float f;                    // 'f' es solo visible dentro de este for-loop
}
```

Byte

El byte almacena un valor numérico de 8 bits, sin punto decimal. Tiene un rango de 0-255

Código

```
byte someVariable = 180;                // declares 'someVariable' como de tipo byte
```

Int

Los enteros son los primeros tipos de datos para el almacenamiento de números sin puntos decimales y almacena valores de 16 bits, con un rango de 32,767 a -32,768

Código

```
int someVariable = 1500;           // declara 'someVariable'  como un tipo entero
```

Nota: *Las variables de tipo entero tendrán sobreflujo si se exceden sus valores máximo o mínimo, por una asignación o comparación. Por ejemplo, si x=32767 y una sentencia siguiente suma 1 a x, x=x+1, o x++, entonces "x" tendrá un sobreflujo y será igual a -32768*

Long

El tipo de datos extendidos para enteros grandes, sin puntos decimales, almacenados en un valor de 32 bits con un rango de 2,147,483,647 a -2,147,483,648.

Código

```
long someVariable = 90000;       // declares 'someVariable' como de tipo grande (long)
```

Float

Un tipo de datos para números de punto flotante, o números que tienen punto decimal. Los números de punto flotante tienen una resolución más grande que los enteros y son almacenados como valores de 32 bits con un rango de 3.4028235E+38 a -3.4028235E+38

Código

```
float someVariable = 3.14;       // declara 'someVariable' como de tipo de punto flotante
```

Nota: *Los números de punto flotante no son exactos, y pueden entregar resultados raros. Así que evítelos de ser posible.*

Arreglos (arrays)

Un arreglo es una colección de valores que se pueden acceder con un número indexado. Cualquier valor en el arreglo puede ser llamado, llamando el nombre del arreglo y el número de índice del valor. Los arreglos son indexados a cero, con el primer valor en el arreglo empezando en el numero de índice 0. Un arreglo necesita ser declarado y opcionalmente asignarle valores antes de que se puedan usar.

Código

```
int miArreglo[]= {valor0, valor1, valor2...}
```

Asimismo es posible declarar un arreglo, declarando el tipo de arreglo y el tamaño, y después asignar valores a una posición del índice:

Código

```
int miArreglo[5]; //Declara un arreglo de tipo entero con 6 posiciones

miArreglo[3] = 10;          //Asigna el valor 10 a la 4ta posición
```

Para recuperar el valor de un arreglo, se le asigna a una variable el arreglo y la posición del índice:

Código

```
x = miArreglo[3];          //x ahora es igual a 10
```

Los arreglos son frecuentemente usados en ciclos, donde el contador de incremento también es usado como la posición del índice para cada valor del arreglo. El siguiente ejemplo usa un arreglo para hacer parpadear un LED. Usando un ciclo “for”, el contador comienza en 0, escribe el valor contenido en la posición del índice 0 en el arreglo “flicker[]” in este caso 180, a el pin 10 PWM, pausa por 200ms, y entonces continúa a la siguiente posición del índice.

Código

```
int ledPin = 10;                                    //LED en el pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};      //Arreglo de 8 elementos diferentes //valores


Void setup()
{
pinMode(ledPin, OUTPUT);                            //Se establece como pin de salida
}
void ciclo()
{
For(int i=0; i<7; i++)                  //El ciclo es igual al número
 {                                      //de valores en arreglo
  analogWrite(ledPIn, flicker[i]);
                                        //escribe el valor del índice
  delay(200);                           //Pausa por 200ms
}

}
```

Aritmética

Los operadores aritméticos incluyen adición, substracción, multiplicación y división. Estos devuelven la suma, diferencia, producto o el cociente (respectivamente) de dos operandos.

Código

```
y = y + 3;

x = x – 7;

i = j * 6;

r = r / 5;
```

La operación es realizada usando el tipo de datos de los operandos, entonces, por ejemplo, 9/4 resulta 2 en lugar de 2.25 ya que 9 y 4 son enteros y son incapaces de usar puntos decimales. Esto también significa que la operación puede desbordarse si el resultado es mas largo que lo que se puede alojar en el tipo de dato.

Si los operandos son de diferente tipo, el tipo mas largo es usado para el cálculo. Por ejemplo, si uno de los números (operando) son del tipo flotante y el otro de tipo entero, la aritmética de tipo flotante será usada para el cálculo.

Escoja el tamaño de variables que son los suficientemente grandes para contener el resultado más grande de sus cálculos. Saber a qué punto su variable se volcara y también lo que sucede en la otra dirección p.e. (0 – 1) o (0 – 32768). Para la aritmética que requiera fracciones, use variables flotantes, pero sea consciente de sus inconvenientes: gran tamaño y baja velocidad de computación.

Nota: *(int)myFloat para convertir un tipo de variable en otro al vuelo. Por ejemplo,*

i = (int) 3.6 establecerá i=3

Asignaciones compuestas (compound assignments)

Las asignaciones compuestas combinan una operación aritmética con una asignación de variable. Estas son comúnmente encontradas en ciclos como se describirá después. Las asignaciones compuestas más comunes incluyen:

Código

```
x ++          //igual que x = x + 1, o incremento de x en +1

x --          //igual que x = x – 1, o decremento de x en -1

x += y        //igual que x = x + y, o incremento de x en +y

x -= y        //igual que x = x – y, o decremento de x en –y

x *= y        //igual que x = x * y, o multiplicar x por y

x /= y        //igual que x = x / y, o dividir x por y
```

Nota: *Por ejemplo, x *= 3 haría triple el valor anterior de x y le reasignaría el valor resultante a x.*

Operadores de Comparación

Comparaciones entre variables o constantes son frecuentemente usadas en sentencias “if” para validar una condición específica. En los ejemplos encontrados en paginas posteriores, “??” es usado para indicar cualquiera de las siguientes condiciones:

```
Código
x == y          //x es igual a y
x != y          //x es diferente de y
x < y           //x es menor que y
x > y           //x es mayor que y
x <= y          //x es menor o igual que y
x >= y          //x es mayor o igual que y
```

Operadores Logicos

Los operadores lógicos son usualmente un camino para comparar dos expresiones y devuelve un “TRUE” o “FALSE” dependiendo del operador. Hay tres operadores lógicos, “AND”, “OR” y “NOT”, que son frecuentemente usados en sentencias “ if ”

AND lógico:

```
Código
If (x > 0 && x < 5)          //verdadero solo si ambas expresiones son verdaderas
```

OR lógico:

```
Código
If (x > 0 || y > 0) //verdadero si una o ambas expresiones son verdaderas
```

NOT lógico:

```
Código
If (!x > 0)                  //verdadero solo si la expresión es //falsa
```

Constantes

El lenguaje Arduino tiene algunos valores predefinidos, qué son llamados constantes. Son usados para hacer programas fáciles de leer. Las constantes son clasificadas en grupos.

True/False

Estas son constantes Boolean que definen niveles lógicos. FALSE es fácilmente definido como 0 (cero) mientras que TRUE es frecuentemente definido como 1, pero puede también ser cualquiera exepto cero. Entonces en el sentido Boolean, -1, 2 y -200 tambien son definidos como TRUE.

Código

```
If (b == TRUE);

{

 doSomething;

}
```

High/low

Estas constantes definen niveles de pin como HIGH (alto) o LOW (bajo) y son usadas cuando se leen o se escriben pines digitales. HIGH es definido como nivel 1 lógico, ON o 5 volts mientras que LOW es el nivel lógico 0, OFF o 0 volts.

Código

```
digitalWrite(13, HIGH);
```

Input/output

Constantes usadas con la función pinMode() para definir el modo de un pin digital ya sea como INPUT (entrada) o OUTPUT (salida).

Código

```
pinMode(13, OUTPUT);
```

If

La sentencia “if” prueba si una cierta condición ha sido alcanzada, tal como un valor analógico estando encima de un cierto número, y ejecuta cualquier sentencia dentro de las llaves si la sentencia es verdadera. Si es falsa el programa pasa por alto las sentencias. El formato para una prueba “if” es:

Código

```
If (algunaVariable ?? valor)
{
 haceAlgo;
}
```

El ejemplo de arriba compara algunaVariable con otro valor, que puede ser una variable o constante. Si la comparación, o condición en paréntesis es verdadera, la sentencia dentro de las llaves correrá. Si no, el programa las pasara por alto y continuara después de las llaves.

Nota: *tener cuidado de usar accidentalmente '=', como en if (x=10), aunque técnicamente valido, define la variable x al valor de 10 y es como resultado siempre verdadero. En lugar de utilizar '==', como en if (x==10),que solo prueba si x es igual al valor 10 o no. Pensar en '=' como "igual" opuesto a '==' siendo "es igual a".*

If... else

If... else permite para 'si no' decisiones para ser tomadas. Por ejemplo, si quiere probar una entrada digital y hacer algo si la entrada fue HIGH o en lugar de eso hacer otra cosa si la entrada fue LOW, se deberá escribir de este modo:

Código

```
If (inputPin == HIGH)

{

 doThingA;

}

else

{

 doThingB;

}
```

Si no también puede proceder otra prueba "if",de modo que multiples, pruebas mutuamente exclusivas pueden correr al mismo tiempo. Es incluso posible tener un ilimitado número de ramas "else". Recordando que solo un grupo de sentencias correrán dependiendo en las condiciones de prueba:

Código

```
if (inputPin < 500)
{
 doThingA;
}
else if (inputPin >= 1000)
{
 doThingB;
}
else
{
 doThingC;
}
```

Nota: *una sentencia "if" simplemente prueba si la condición dentro del paréntesis es verdadera o falsa. Esta sentencia puede ser cualquier sentencia valida en C como en el primer ejemplo. If(inputPin == HIGH). En este ejemplo, la sentencia "if" solo verifica si en verdad la entrada especificada está en un nivel lógico alto, o +5v.*

For

La sentencia "for" es usada para repetir un bloque de sentencias encerradas en llaves un número especificado de veces. Un contador de incremento es frecuentemente usado para incrementar y terminar el ciclo. Hay tres partes, separadas por punto y coma (;), para el encabezado del ciclo "for":

Código

```
For (inicialización; condición; expresión)

{

 doSomething;

}
```

La inicialización de una variable local, o contador de incremento, pasan primero y solo una vez. Cada vez a través del ciclo, la siguiente condición es probada. Si la condición sigue siendo verdadera, los siguientes sentencias y expresión son ejecutadas y la condición es probada de nuevo. Cuando la condición se vuelve falsa, el ciclo termina.

El siguiente ejemplo comienza el entero i en 0, prueba si i es aun menos que 20 y si es verdadero incrementa i en 1 y ejecuta las sentencias encerradas:

Código

```
For (int i=0; i<20; i++)          //declara i, prueba si es menor
{                                 //que 20, incrementa i en 1
 digitalWrite(13, HIGH);          //activa el pin 13 en pausas de
 delay(250);                      //1/4 de segundo
 digitalWrite(13, LOW);           //desactiva el pin 13 en pausas de
 delay(250);                      //1/4 de segundo
}
```

Nota: *El ciclo "C for" es mucho más flexible que ciclos "for" en otros lenguajes de computación, incluidos BASIC. Cualquiera o todos los tres elementos del encabezado pueden ser omitidos, aunque el punto y comas son requeridos. También las sentencias para la inicialización, condición y expresión pueden ser cualquier sentencia valida en C. Estos tipos de sentencias "for" inusuales pueden proveer soluciones a algunos problemas de programación.*

While

Los ciclos "while" se estarán ciclando continua e infinitamente, hasta que la expresión dentro del paréntesis se vuelva falsa. Algo debe cambiar la variable de prueba, o el ciclo nunca terminara. Esto podría ser en el código, tal como una variable de incremento, o una condición externa, tal como un sensor de prueba.

Código

```
While (someVariable ?? value)
{
  doSomething;
}
```

El siguiente ejemplo prueba si 'someVariable' es menor que 200 y si es verdadero ejecuta las sentencias dentro de las llaves y se continuara ciclando hasta que 'someVariable' ya no sea menor que 200.

Código

```
While (someVariable < 200)       //Prueba si es menor que 200
{
  doSomething;                   //Ejecuta las sentencias encerradas
  someVariable++;                //Incrementa la variable en 1;
}
```

Do... while

El ciclo “do-while”, en la misma manera como el ciclo while con la excepción que la condición es probada al final del ciclo, entonces el ciclo “do-while” siempre correrá por lo menos una vez.

Código

```
Do
{
  doSomething;
}while (someVariable ?? value);
```

El siguiente ejemplo asigna readSensors() a la variable ‘x’, pausa por 50 milisegundos, entonces se cicla indefinidamente hasta que ‘x’ ya no sea menor que 100:

Código

```
Do
{
   x = readSensors();              //asigan el valor de readSensros()  a  x.
   delay (50);                     //pausa 50 milisegundos.
}while (x < 100);          //cicla si x es menor que 100.
```

pinMode(pin, mode)

Usado en void setup() para configurar un pin especifico para que se comporte ya sea como INPUT o como OUTPUT.

Código

```
pinMode(pin, OUTPUT);//Establece 'pin' como salida
```

Los pins digitales del arduino por default están como INPUTS (entradas), así que no es necesario declararlos explícitamente como entradas con pinMode(). Los pins configurados como INPUT se dice que están en un estado de alta impedancia.

Existen también resistores pullup de 20kOhms construidos dentro del chip Atmega que pueda ser accesados desde el software. Esta construcción de resistores pullup pueden ser accesados de la siguiente manera:

Código

```
pinMode(pin, INPUT);            //Establece 'pin' como entrada

digitalWrite(pin, HIGH);        //Activa los resistores pullup
```

Los resistores pullup pueden ser normalmente utilizados para conectar entradas como interruptores. Nótese que en el ejemplo de arriba no se convierte 'pin' a una salida, es simplemente un método para activar los pull-ups internos.

Pins configurados como OUTPUT se dice que están es un estado de baja impedancia y pueden proveer 40mA (mili ampers)de corriente a otros dispositivos/circuitos. Esta corriente es suficiente para encender un LED (sin olvidar la resistencia en serie), pero no la suficiente para energizar la mayoría de relevadores, seloides o motores.

Corto circuitos en los pines del arduino y corriente excesiva pueden dañar o destruir los pines de salida, o dañar por completo el chip Atmega. Es una buena idea conectar un pin de salida a un dispositivo externo en serie con una resistencia de 470 Ohms o 1 KOhm.

digitalRead(pin)

Lee el valor de un pin digital específico con el resultado ya sea HIGH o LOW. El pin puede ser especificado ya sea como variable o como constante (0-13).

Código

```
Value = digitalRead(Pin);       //Establece 'value' igual a

                                //el pin de entrada.
```

digitalWrite(pin, value)

Fija las salidas a un nivel lógico “HIGH o LOW” (enciende o apaga) en un pin digital especifico. El pin puede ser especificado como variable o como constante (0-13).

Código

```
digitalWrite(pin, HIGH);          // Establece ‘pin’ como HIGH
```

El siguiente ejemplo lee un pushbutton conectado a una entrada digital y enciende un LED conectado a una salida digital, cuando el botón ha sido presionado:

Código

```
int led   = 13;          //establece el pin 13 como  la variable entera  LED
int pin   =  7;          //establece el pin 7 como la variable entera pin
int value =  0;          //variable para almacenar el valor leido=0

void setup()
{
  pinMode(led, OUTPUT);       //establece el pin 13 como salida
  pinMode(pin, INPUT);        //establece el pin 7 como entrada
}
Void loop()
{
value =  digitalRead(pin)     // asigna a ‘value’ el valor digital leido en la entrada pin
                              //pin de entrada
digitalWrite(led, value);     //establece ‘led’ igual al valor de value
```

analogRead(pin)

Lee el valor de un pin analógico específico con una resolución de 10-bit. Esta función solo trabaja en los pines analógicos de entrada(0-5). El rango de valores enteros resultante va de 0 a 1023.

Código

```
value = analogRead(pin);          //establece ‘value’ igual a ‘pin’
```

Nota: *A diferencia de los pines digitales, los pines analógicos no necesitan ser declarados como INPUT u OUTPUT*

analogWrite(pin, value)

Escribe un valor pseudo análogo usando habilitación de hardware “pulse width modulation” (PWM) a un pin de salida marcado como PWM.

En los arduinos más nuevos con el chip ATmega168 , esta función trabaja en los pines 3, 5, 6, 9, 10 and 11. Los arduinos más viejos con el chip ATmega8 solo soporta los pines 9, 10 y 11.

El valor puede ser especificado como una variable o constante con el valor de 0 a 255.

Código

```
analogWrite(pin, value); //escribe ‘value’ a ‘pin’ analógico.
```

Un valor de 0 genera 0 volts estables de salida en el pin especificado; un valor genera 5 volts estables de salida en el pin especificado. Para valores entre 0 y 255, el pin alterna rápidamente entre 0 y 5 volts – entre mas grande sea el valor, más frecuentemente el pin estará en HIGH (5 volts). Por ejemplo, un valor de 64 será 0 tres cuartos del tiempo y 5 volts un cuarto de tiempo; un valor de 128 será 0 volts la mitad del tiempo y 5 volts la otra mitad de tiempo; y un valor de 192 será 0 volts una cuarta parte del tiempo y 5 volts tres cuartas partes del tiempo.

Ya que esto es una función de hardware, el pin generara un onda estable después de una llamada a analogWrite en segundo plano hasta la próxima llamada a analogWrite (o una llamada a digitalRead o digitalWrite en el mismo pin).

Nota: *A diferencia de los pines digitales, los pines analógicos no necesitan ser declarados como INPUT u OUTPUT.*

El siguiente ejemplo lee un valor analógico del pin de entrada análogo, convierte el valor dividiendo por 4, y emite una señal PWM en un pin PWM.

Código

```
int led   = 10;                 //led con resistencia de 220ohms en el pin 10
int pin   =  0;                 //potenciómetro en el pin analógico 0
int value         ;             //valor para lectura

void setup() { }                //no se necesita

void loop()
{
  value = analogRead(pin);      //establece ‘value’ igual a ‘pin’
  value /= 4;                   //convierte 0-1023 a 0-255
  analogWrite(led, value);      //emite la señal PWM al led
}
```

Delay(ms)
Pausa un programa por la cantidad de tiempo que sea especificado en milisegundos, donde 1000 equivale a 1 segundo.

Código

```
delay(1000);      //espera por un segundo
```

millis( )
Regresa el número de milisegundos como un valor de tipo long sin signo, desde que la tarjeta Arduino comenzó a correr el programa actual.

Código

```
value = millis();                    //establece ´value´ igual a lo que regresa la función millis()
```

Nota: *Este número se desbordara (restablece a cero), después de aproximadamente 9 horas.*

min(x, y)
Calcula el mínimo de dos números de cualquier tipo de dato y regresa el número más pequeño.

Código

```
value =  min(value, 100);          // establece ´value´ al más pequeño de ´value´ o 100,

                                   // asegurando que nunca sea mayor que 100
```

max(x, y)
Calcula el máximo de dos números de cualquier tipo de dato y regresa el número más grande.

Código

```
Value = max(value, 100);           // establece ´value´a la mayor de ´value´ o 100,

                                                        // Asegurando que al menos sea 100
```

.

randomSeed(seed)
Establece un valor, o semilla, como el punto de partida para la función “random( )”.

Código

```
randomSeed(value);                 // establece ´value´como la semilla aleatoria
```

Ya que el Arduino es incapaz de crear un numero real aleatorio, “randomSeed” te permite colocar una variable, constante u otra función dentro de la función “random”, que ayuda a generar números aleatorios “random”. Hay una variedad de diferentes semillas, o funciones, que pueden ser usadas en esta función incluyendo “millis()” o incluso “analogRead()” para leer un ruido electrónico a través de un pin análogo.

random(max)
random(min, max)
La función “random” permite regresar números pseudo-aleatorios con un rango especificado por valores mínimos y máximos.

Código

```
Value = random(100, 200);       //  establece ´value´ a un numero aleatorio entre 100-200
```

Nota: *Usa esto después de usar la función “randomSeed()”.*

El siguiente ejemplo crea un valor aleatorio entre 0 – 255 y emite una señal PWM por un pin PWM igual al valor aleatorio:

Código

```
Int randNumber;                     // variable para guardar el valor aleatorio
Int led = 10;                       // diodo LED en serie con un resistor de 220 ohms en el pin 10


void setup() {}                     //  no requerida
void loop()
{
   randomSeed(millis () );          // establece millis() como semilla
   randNumber = random(255);        // numero aleatorio de 0 – 255
  analogWrite(led, randNumber);     // emite señal PWM
  delay(500);                       // pausa medio Segundo
}
```

Serial.begin(rate)
Abre el puerto serial y establece la velocidad para la transmisión de datos en serie. La velocidad típica de transmisión para la comunicación con la computadora es de 9600bps aunque también soporta otras velocidades.

Código

```
Void setup()
{
  Serial.begin(9600);          // abre el puerto serial
}                              // establece la velocidad de transmisión a 9600 bps
```

Nota: *Cuando se usa comunicación serial, los pines digitales 0 (RX) y 1 (TX) no pueden ser usados al mismo tiempo.*

Serial.println(data)
Envia "data" al puerto serial, seguido por un acarreo de retorno automatico y una línea de alimentación. Este comando toma la misma forma que "Serial.print()", pero es más sencillo para leer "data" del monitor serial.

Código

```
Serial.println(analogValue);          // envía el valor de
                                      // 'analogValue'
```

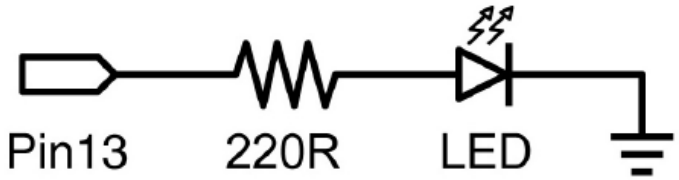
Nota: *Para más información de las diferentes permutaciones de las funciones "Serial.println()" y "Serial.print()" por favor recurre a la página web del Arduino.*

El siguiente ejemplo toma la lectura del pin análogo "pin0" y envía este dato a la computadora cada segundo.

Código

```
Void setup()
{
  Serial.begin(9600);                 // Establece el serial a 9600bps
}

Void loop()
{
  Serial.println(analogRead(0));      // envía el valor análogo
  Delay(1000);                        // pausa por un segundo
}
```

SALIDA DIGITAL

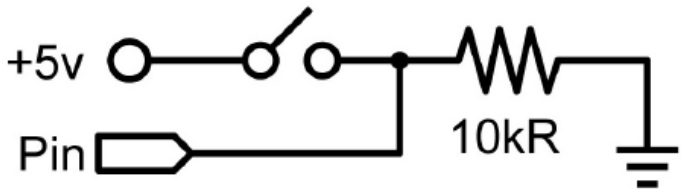


Este es el programa basico “hola mundo” usado para simplemente prender o apagar algo. En este ejemplo, un LED es conectado al pin13, y parpadea cada segundo. El resistor puede ser omitido en este pin ya que el Arduino tiene uno en su estructura.

Código

```
int ledpin = 13;                  // LED en el pin 13
void setup()                      // corre una vez
{
  pinMode(ledPin, OUTPUT);        // establece el pin 13 como salida
}
void loop()                       // corre una y otra vez
{
  digitalWrite(ledPin, HIGH);     // prende el LED
  delay(1000);                    // pausa por un segundo
  digitalWrite(ledPin, LOW);      // apaga el LED
  delay(1000);                    // pausa por un segundo
}
```

ENTRADA DIGITAL

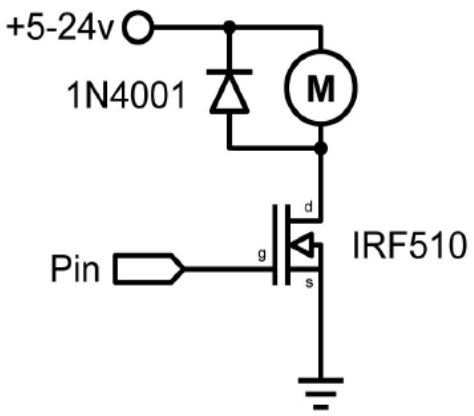


Este es la forma mas simple de entrada con solo 2 posibles estados: prendido o apagado. Este ejemplo lee un switch simpl o “pushbuttton” conectado al pin 2. Cuando el switch se cierra el pin de entrada leera HIGH y prendera un LED.

Código

```
int ledPin = 13;                    // pin de salida para el LED
int inPin = 2;                      // pin de entrada para el switch
void setup()
{
   pinMode(ledPin, OUTPUT);         // Se declara el LED como salida
   pinMode(inPin, INPUT);           // Se declara el Switch como entrada
}
void loop()
{
   If (digitalRead(inPin) == HIGH)  // Comprueba si la entrada esta en HIGH
   {
     digitalWrite(ledPin, HIGH);    // prende el LED
     delay(1000);                   // pausa por un segundo
     digitalWrite(ledPin, LOW);     // apaga el LED
     delay(1000);                   // pausa por un segundo
    }
}
```

SALIDA DE CORRIENTE ALTA



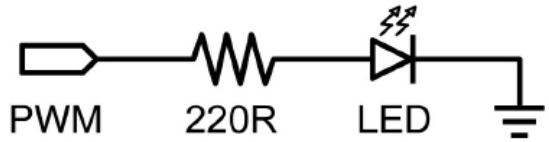
A veces es necesario controlar mas de 40ma para el arduino. En este caso un MOSFET o transistor puede ser usado para cambiar cargas de corriente. El siguiente ejemplo prende y apaga rapidamente el MOSFET 5 veces por segundo.

Nota: *El esquema muestra un motor y un diodo de protección pero pueden ser usados otras cargas no inductivas sin el diodo.*

Código

```
Int outPut = 5;                 // pin de slaida para el MOSFET

void setup()
{
   pinMode(outPin, OUTPUT);     // establece el pin 5 como salida
}

void loop()
{
   for  ( int i=0; i<=5; i++)   // repite 5 veces
   {
     digitalWrite(outPin, HIGH);// prende el MOSFET
     delay(250);                // pausa por un cuarto de segundo
     digitalWrite(outPin, LOW); // apaga el MOSFET
     delay(250);                // pausa por un cuarto de segundo
    }
   Delay(1000);                 // pausa por un segundo
}
```

SALIDA PWM

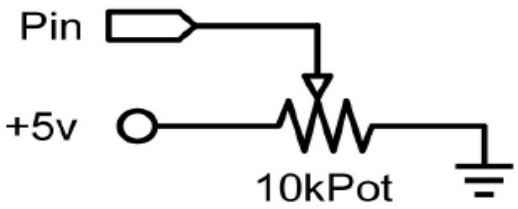


Modulacion por ancho de pulsos (Pulsewidth Modulation) es una manera de simular una salida analógica por la pulsación de la salida. Esto puede ser usado para atenuar o aumentar el brillo de un LED o incluso controlar un servomotor. El siguiente ejemplo atenua lentamente un LED usando ciclos for.

Código

```
int ledPin = 9;                 // pin PWM para el LED

void setup(){ ]                 // no se requiere

Void loop()
{
   for (int i=0; i<=255; i++)   // incremento de i
   {
     analogWrite(ledPin, i);    // establece el nivel de brillo a i
     delay(100);                // pausa por 100ms
   }
   for (int i=255; i>=0; i--)   // decremento de i
   {
     analogWrite(ledPin, i);    // establece el nivel de brillo a i
     delay(100);                //pausa por 100ms
   }
}
```

ENTRADA DE POTENCIOMETRO



Usando un potenciometro y un pin conversor analogico-a-digital (ADC) del arduino es posible leer valores de 0 a 1024. El siguiente ejemplo usa un potenciometro para controlar el valor de parpadeo de un LED.

Código

```
int potPin = 0;                 // pin de entrada para el potenciometro
int ledPin = 13;                // pin de salida para el LED


void setup()
{
   pinMode(ledPin, OUTPUT);     // Se declara el ledPin como salida
}


void loop()
  {
    digitalWrite(ledPin, HIGH);  // prende el LED
    delay(analogRead(potPin));   // pausa el programa
    digitalWrite(ledPin, LOW);   // apaga el LED
    delay(analogRead(potPin));   // pausa el programa
   }
```

ENTRADA DE RESISTOR VARIABLE

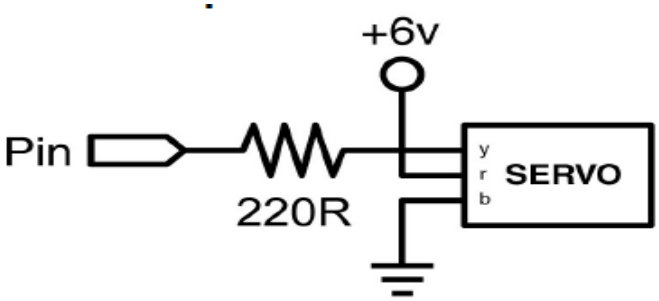
Resistores variables incluyendo fotoresistores, termistores, sensores de flexibilidad, etc.

Este ejemplo hace uso de una funcion para leer un valor analogo y establecer un tiempo de retraso. Esto controla la velocidad con la que un LED atenua y aumenta su luminosidad.

Código

```
int ledPin      =  9;                // pin PWM para el LED
int analogPin   =  0;                // resistor variable en el pin 0

void setup(){}                       // no se requiere

void loop()
{
   for  ( int i=0; i<=5; i++)        // incremento para i
   {
     analogWrite(ledPin, i);         // establece el nivel de brillo a i
     delay(delayVal());              // toma el valor de tiempo y pausa
   }
   For  ( int i=255; i>=0; i--)      // decremento para i
   {
     analogWrite(ledPin, i);         // establece el nivel de brillo a i
     delay(delayVal());              // toma el valor de tiempo y pausa
    }
}

int delayVal()
{
  int v;                             // crea una variable temporal
  v    = analogoRead(analogPin);     // lee el valor analogo
  v  /= 8;                           // convierte 0 – 1024 a 0 -128
  return v;                          // regresa el valor final
}
```

SALIDA SERVO

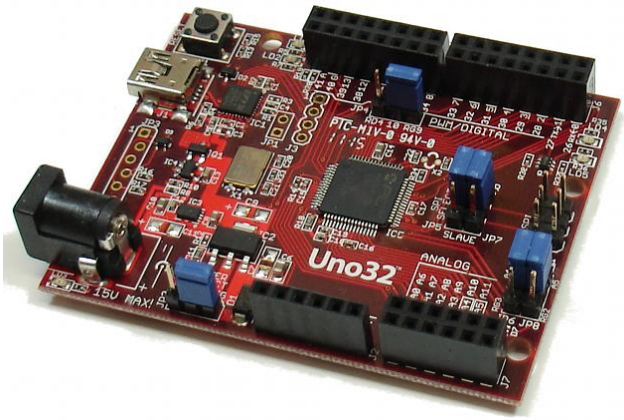


Los “hobby servos” son un tipo de motores autonomos que pueden moverse en un angulo de 180º. Todo lo que se necesita es enviar un pulso cada 20ms. Este ejemplo usa una función “servoPulse” para mover el servo de 10º a 170º y de regreso.

Código

```
int servoPin = 2;                    // servo conectado a pin digital 2
int myAngle;                         // angulo de el servo 0-180 aproximadamente
int pulseWidht;                      // funcion variable servoPulse

void setup()
{
   pinMode(servoPin, OUTPUT);                  // establece pn 2 como salida
}

Void servoPulse(int servoPin, int myAngle)
{
   pulseWidth = (myAngle * 10) + 600;          // determina el retraso
   digitalWrite(servoPin, HIGH);               // establece servo en HIGH
   delayMicroseconds(pulseWidth);              // pausa en microsegundo
   digitalWrite(servoPin, LOW);                //establece servo en LOW
}

void loop()
{
  // servo comienza en 10 grados y rota a 170 grados
  for (myAngle=10; myAngle<=170; myAngle++)
  {
     servoPulse(servoPin, myAngle);            // envia pin y angulo
     delay(20);                                // actualiza el ciclo
   }

// servo comienza en 170 grados y rota a 10 grados
   for (myAngle=170; myAngle>=10; myAngle--)
  {
     servoPulse(servoPin, myAngle);            // envia pin y angulo
     delay(20);                                // actualiza el ciclo
   }
}
```

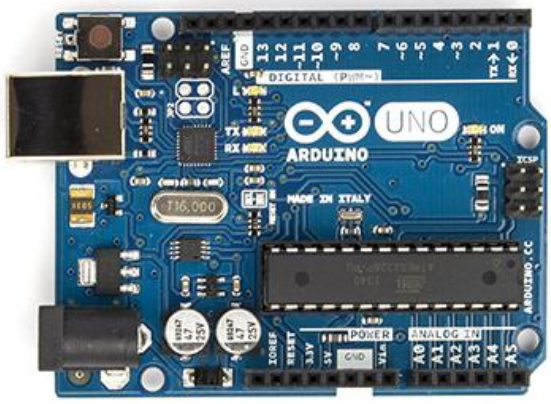
PLATAFORMAS DE HARDWARE ABIERTO



PIC32MX320F128H

(Familia: PIC32MX3XX/4XX)

"chipKIT Uno32"



ATMEGA328P

(Familia: AVR´S)

"Arduino Uno R3"

MICROCONTROLADOR PIC32MX320F128H

Features

MCU Core

- 80MHz/105DMIPS, 32-bit MIPS M4K Core
- 5 Stage pipeline, Harvard architecture
- MIPS16e mode for up to 40% smaller code size
- Single cycle multiply and hardware divide unit
- 32 x 32-bit Core Registers
- 32 x 32-bit Shadow Registers
- Fast context switch and interrupt response

MCU System Features

- 128K Flash (plus 12K boot Flash)
- 16K RAM (can execute from RAM)
- Flash prefetch module with 256 Byte cache
- Lock instructions or data in cache for fast access
- Programmable vector interrupt controller

Analog Features

- Fast and Accurate 16 channel 10-bit ADC,
- Max 1 Mega samples per second at +/- 1LSB, conversion available during SLEEP & IDLE

Power Management Modes

- RUN, IDLE, and SLEEP modes
- Multiple switchable clock modes for each power mode, enables optimum power settings

Other MCU Features

- Fail-Safe Clock Monitor – allows safe shutdown if clock fails
- Hardware RTCC (Real-Time Clock and Calendar with Alarms)
- 2 Internal oscillators (8MHz & 31KHz)
- Watchdog Timer with separate RC oscillator

Parameter Name	Value
Family	PIC32MX3XX
Max Speed MHz	80
Program Memory Size (KB)	128
RAM (KB)	16
Auxiliary Flash (KB)	12
Temperature Range (C)	-40 to 105
Operating Voltage Range (V)	2.3 to 3.6
SPI™	2
I²C™ Compatible	2
A/D channels	16
Max A/D Resolution	10
Output Compare/Std. PWM	5
16-bit Digital Timers	5
Parallel Port	PMP
Comparators	2
Internal Oscillator	8 MHz, 32 kHz
I/O Pins	53
Pin Count	64

El chipKIT Uno32 es basado en la popular plataforma de prototipado del Arduino de hardware libre y agrega el desempeño del micro controlador PIC32. Uno32 tiene la misma forma que la tarjeta del Arduino Uno y es compatible con los shield del Arduino. Presenta un puerto serial USB para conexión con el IDE y puede ser alimentado vía USB o una fuente de poder externa. Cuenta con un procesador a una velocidad de 80Mhz, 128Kb de memoria flash programable y 16Kb de memoria de datos SRAM.
El Arduino Uno32 puede ser programado por medio de la interfaz MPIDE el cual soporta PIC32. Provee de 42 pines de I/O que soportan funciones periféricas como es UART, SPI y puertos I^2C, y puertos de salida PWM (Pulso de Amplitud Modulada). Doce de los pines de entradas y salidas pueden ser usadas como entradas análogas o como entradas y salidas digitales.

Características:

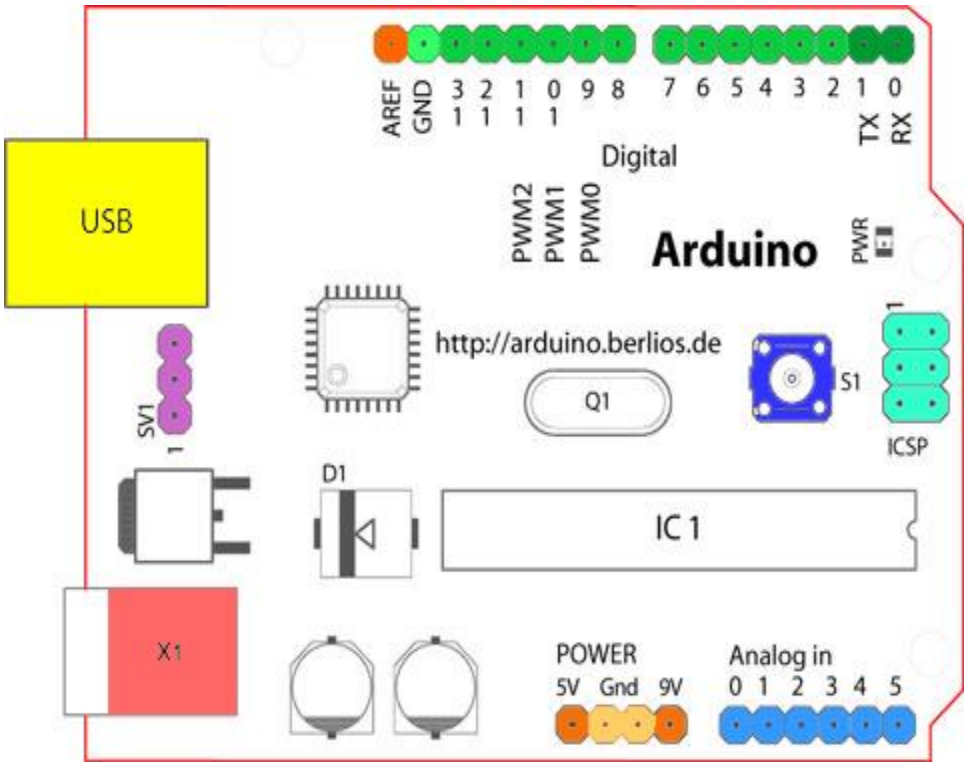
- Microcontrolador PIC32MX320F128H (80 MHz 32-bit MIPS, 128K Flash, 16K SRAM)
- Compatible con muchos ejemplos
- Compatible con muchos Arduino Shields
- 42 pines de entradas y salidas disponibles
- 2 LED de usuario
- 12 entradas análogas
- Voltaje de operación: 3.3V
- Frecuencia de operación: 80Mhz
- Corriente típica de operación: 75mA
- Voltaje de entrada recomendado: 7v a 15v
- Voltaje de entrada máximo: 20v
- Rango de voltaje análogo de entrada: 0v a 3.3v
- Corriente directa por pin: +/-18mA

MICROCONTROLADOR ATMEGA328P

	ATmega168	ATmega328
Voltaje operativo	5 V	5 V
Voltaje de entrada recomendado	7-12 V	7-12 V
Voltaje de entrada límite	6-20 V	6-20 V
Contactos de entrada y salida digital	14 (6 proporcionan PWM)	14 (6 proporcionan PWM)
Contactos de entrada analógica	6	6
Intensidad de corriente	40 mA	40 mA
Memoria Flash	16KB (2KB reservados para el bootloader)	32KB (2KB reservados para el bootloader)
SRAM	1 KB	2 KB
EEPROM	512 bytes	1 KB
Frecuencia de reloj	16 MHz	16 MHz

DESCRIPCIÓN DE LOS COMPONENTES DEL ARDUINO UNO R3

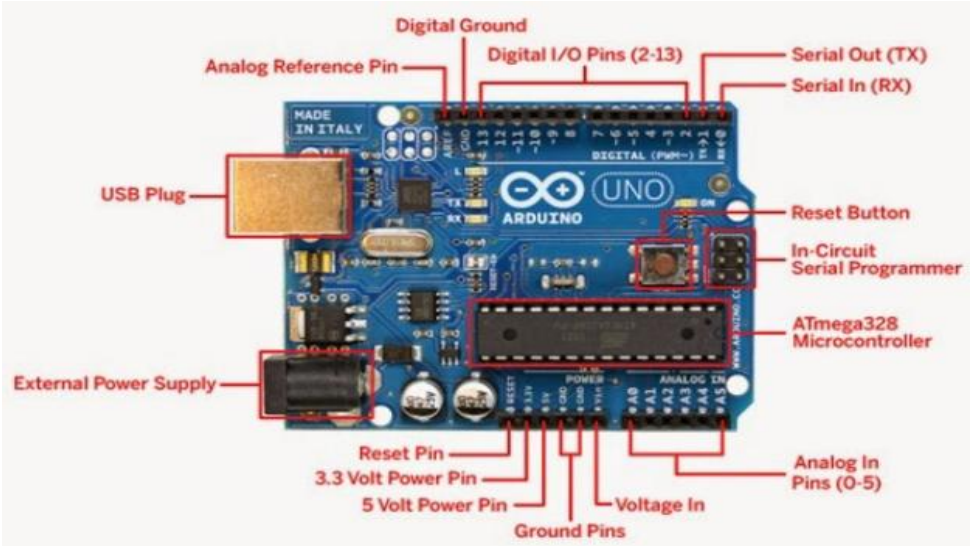
Mirando a la placa desde la parte de arriba, este es el esquema de lo que puedes ver (los componentes de la placa con los que puedes interactuar en su uso normal están resaltados)



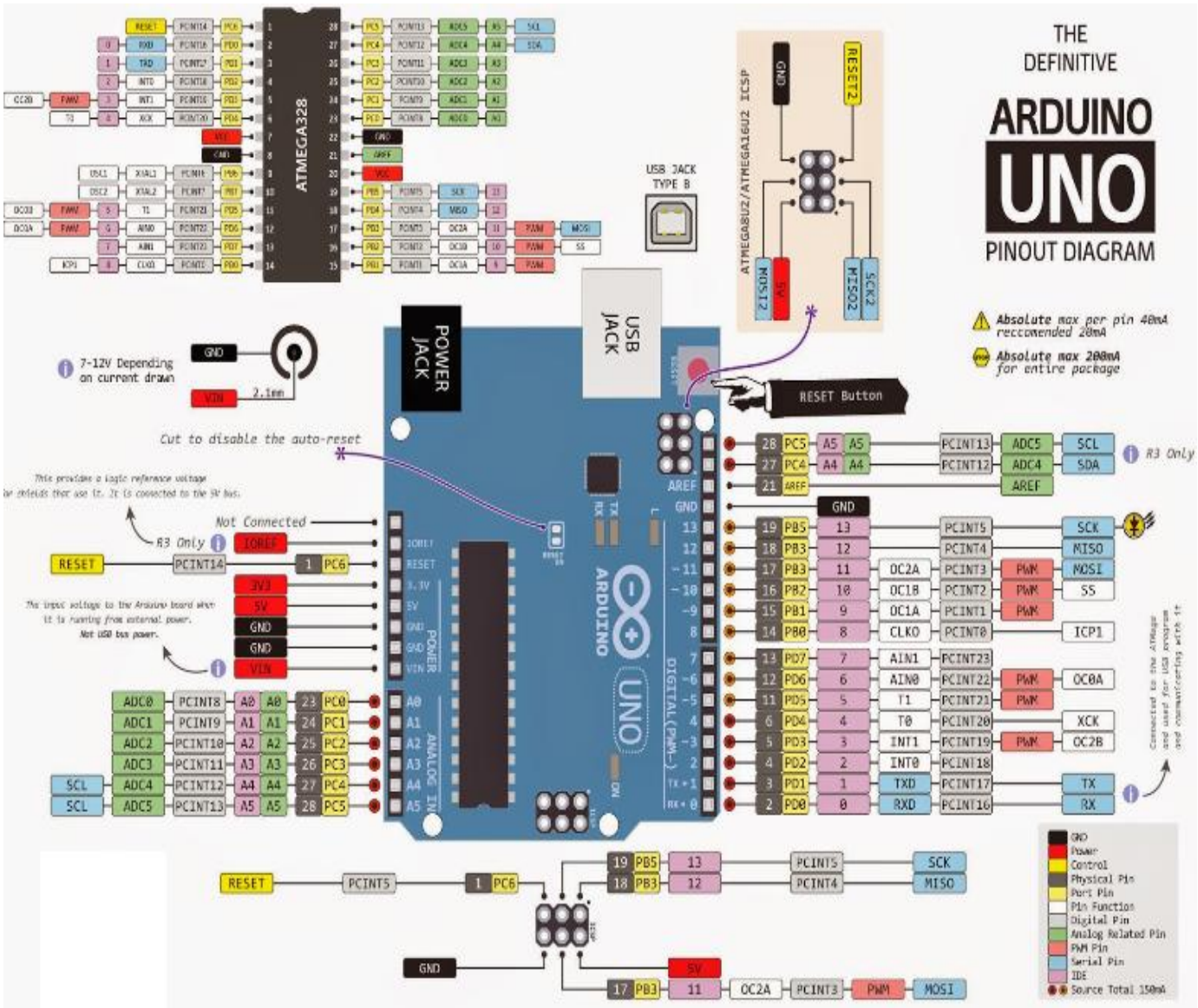
Empezando según las agujas del reloj:

- Terminal de referencia analógica (naranja)
- Tierra digital (verde claro)
- Terminales digitales 2-13 (verde)
- Terminales digitales 0-1/ E/S serie - TX/RX (verde oscuro) - *Estos pines no se pueden utilizar como e/s digitales (digitalRead() y digitalWrite()) si estás utilizando comunicación serie (por ejemplo **Serial.begin**)*.
- Botón de reinicio - S1 (azul oscuro)
- Programador serie en circuito "In-circuit Serial Programmer" o "ICSP" (azul celeste)
- Terminales de entrada analógica 0-5 (azul claro)
- Terminales de alimentación y tierra (alimentación: naranja, tierras: naranja claro)
- Entrada de alimentación externa (9-12VDC) - X1 (rosa)
- Selector de alimentación externa o por USB (coloca un jumper en los dos pines más cercanos de la alimentación que quieras) - SV1 (púrpura). En las versiones nuevas de Arduino la selección de alimentación es automática por lo que puede que no tengas este selector.
- USB (utilizado para subir programas a la placa y para comunicaciones serie entre la placa y el
- ordenador; puede utilizarse como alimentación de la placa) (amarillo)

DIAGRAMA DE PINES DEL ARDUINO UNO



URL: http://elektrikelektronikprojeleri.blogspot.mx/2014/11/arduino-uno.html



Terminales Digitales

- En adición a las funciones específicas listadas abajo, las terminales digitales de una placa Arduino pueden ser utilizados para entradas o salidas de propósito general a través de los comandos pinMode(), digitalRead(), y digitalWrite().

- Cada terminal tiene una resistencia pull-up que puede activarse o desactivarse utilizando DigitalWrite() (con un valor de HIGH o LOW, respectivamente) cuando el pin está configurado como entrada. La corriente máxima por salida es 40 mA.

- Serial: 0 (RX) y 1 (TX). Utilizado para recibir (RX) y transmitir (TX) datos serie TTL. En el Arduino Diacemila, estas terminales están conectadas a las correspondientes patas del circuito integrado conversor FTDI USB a TTL serie. En el Arduino BT, están conectados a las terminales correspondientes del modulo Bluetooth WT11. En el Arduino Mini y el Arduino LilyPad, están destinados para el uso de un módulo serie TTL externo (por ejemplo el adaptador Mini-USB).

- Interrupciones externas: 2 y 3. Estas terminales pueden ser configuradas para disparar una interrupción con un valor bajo, un pulso de subida o bajada, o un cambio de valor. Mira la función attachInterrupt() para más detalles.

- PWM: 3, 5, 6, 9, 10, y 11. Proporcionan salidas PWM de 8 bit con la función analogWrite(). En placas con ATmega8, las salidas PWM solo están disponibles en los pines 9, 10, y 11.

- Reset BT: 7. (solo en Arduino BT) Conectado a la línea de reset del módulo bluetooth.

- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estas terminales soportan comunicación SPI. Aunque esta funcionalidad esta proporcionada por el hardware, no está incluida actualmente el lenguaje Arduino.

- LED: 13. En el Diacemila y el LilyPad hay un LED en placa conectado al pin digital 13. cuando el pin tiene valor HIGH, el LED está encendido, cuando el pin está en LOW, está apagado

Pines Analógicos

En adición a las funciones específicas listadas abajo, los pines de entrada analógicos soportan conversiones analógico-digital (ADC) de 10 bit utilizando la función analogRead(). Las entradas analógicas pueden ser también usadas como pines digitales: entrada analógica 0 como pin digital 14 hasta la entrada analógica 5 como pin digital 19. Las entradas analógicas 6 y 7 (presentes en el Mini y el BT) no pueden ser utilizadas como pines digitales.

- I2C: 4 (SDA) y 5 (SCL). Soportan comunicaciones I2C (TWI) utilizando la librería Wire (documentación en la página web de Wiring).

Pines de alimentación

- **VIN :** (a veces marcada como "9V"). Es el voltaje de entrada cuando se está utilizando una fuente de alimentación externa (en comparación con los 5 voltios de la conexión USB o de otra fuente de alimentación regulada). Puedes proporcionar voltaje a través de este pin.

- **5V :** La alimentación regulada utilizada para alimentar el Microcontrolador y otros componentes de la placa. Esta puede venir de VIN a través de un regulador en placa o ser proporcionada por USB u otra fuente regulada de 5V.

- **3V3** : (solo en el Diacemila) Una fuente de 3.3 voltios generada por el chip FTDI de la placa.

- **GND** : Pin de tierra.

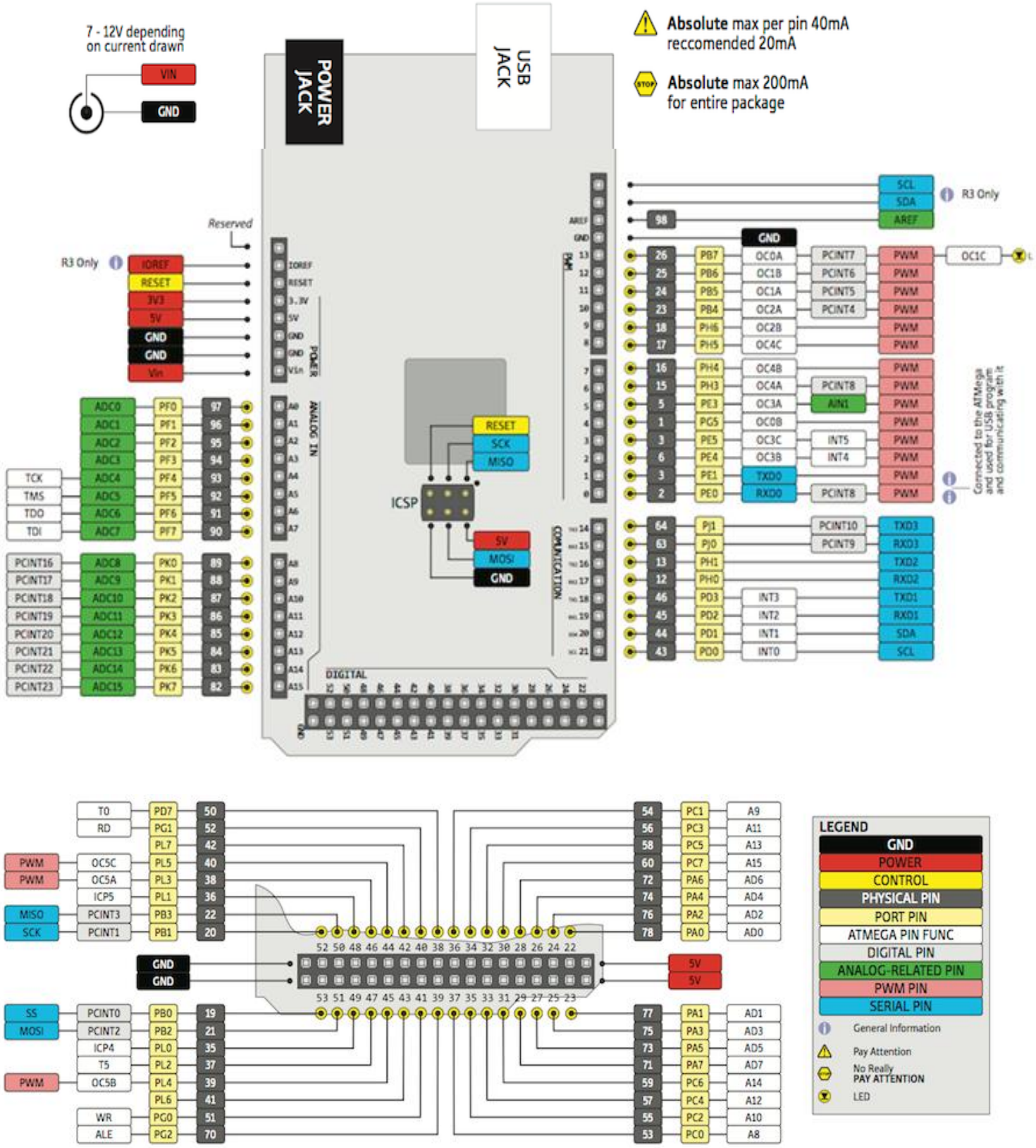
Otros Pines

- **AREF** : Referencia de voltaje para las entradas analógicas. Utilizada con la función analogReference().
- **Reset** : (Solo en el Diacemila). Pon esta línea a LOW para resetear el Microcontrolador. Utilizada típicamente para añadir un botón de reset a shields que bloquean el de la placa principal.

Arduino Uno Vs. Mega

Modelo	Arduino UNO	Arduino Mega 2560
Microcontroller	ATmega328	ATmega2560
Operating Voltage	5V	5V
Input Voltage	7-12V	7-12V
Input Voltage (limits)	6-20V	6-20V
Digital I/O Pins	14	54
Digital I/O Pins PWM output	6	15
Analog Input Pins	6	16
Analog Outputs Pins		
Total DC Output Current on all I/O lines	40 mA	40 mA
DC Current for 3.3V Pin	50 mA	50 mA
DC Current for 5V Pin		
Flash Memory	32 KB 0.5 KB used	256 KB 8 KB used by
SRAM	2 KB (ATmega328)	8 KB
EEPROM	1 KB (ATmega328)	4 KB
Clock Speed	16 MHz	16 MHz
Tipo de USB	Estandar	Estándar

DIAGRAMA DE PINES DEL ARDUINO MEGA



URL: http://www.openbuilds.com/builds/mox-cnc.1666/

PRÁCTICA 01 - LED INTERMITENTE (BLINK)

Se trata de realizar un ejercicio básico que consiste en encender y apagar un led que conectamos en el PIN 13 de Arduino que lo configuramos como salida. El tiempo de encendido y apagado es de 1 segundo.

Material:

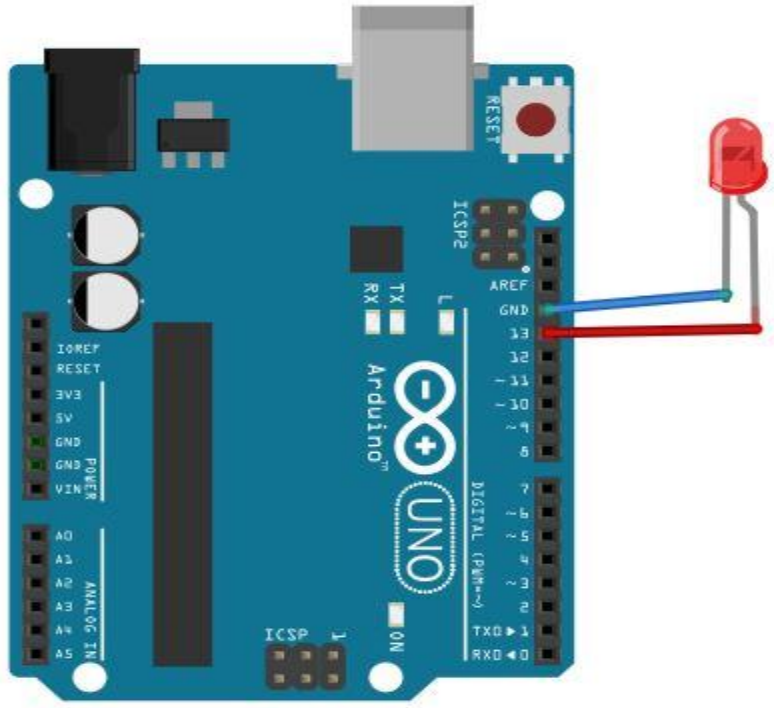
- **1 Protoboard**
- **1 Diodo LED**
- **Arduino Uno R3**

Colocar el diodo led sin resistencia en serie dado que el PIN13 de Arduino ya lleva incorporada una resistencia interior, en el caso de colocar el diodo LED en otra salida deberíamos colocar una resistencia de al entre 220 y 500 ohmios dependiendo del consumo de corriente del diodo.

Código

```
/*
* Intermitente
*
* Ejemplo básico con Arduino. Encendido y apagado de un led
* con una cadencia de 1 sg. usando el PIN 13 como salida
* no es necesario usar una resistencia para el led
* la salida 13 de Arduino la lleva incorporada.
*
*/

int ledPin = 13;                // Definición de la salida en el PIN 13
void setup()                    //Configuración
{
  pinMode(ledPin, OUTPUT);      // designa la salida digital al PIN 13
}

void loop()                       // bucle de funcionamiento
{
  digitalWrite(ledPin, HIGH);   // activa el LED
  delay(1000);                  // espera 1 seg. (tiempo encendido)
  digitalWrite(ledPin, LOW);    // desactiva el LED
  delay(1000);                   // espera 1 seg. (tiempo apagado)
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 1

Agregue aquí .....

PRÁCTICA 02 – FADE

Se trata de encender y apagar un LED colocado en la salida 9 usando la función analogWrite(), de tal manera que haga el encendido y apagado de forma lenta.

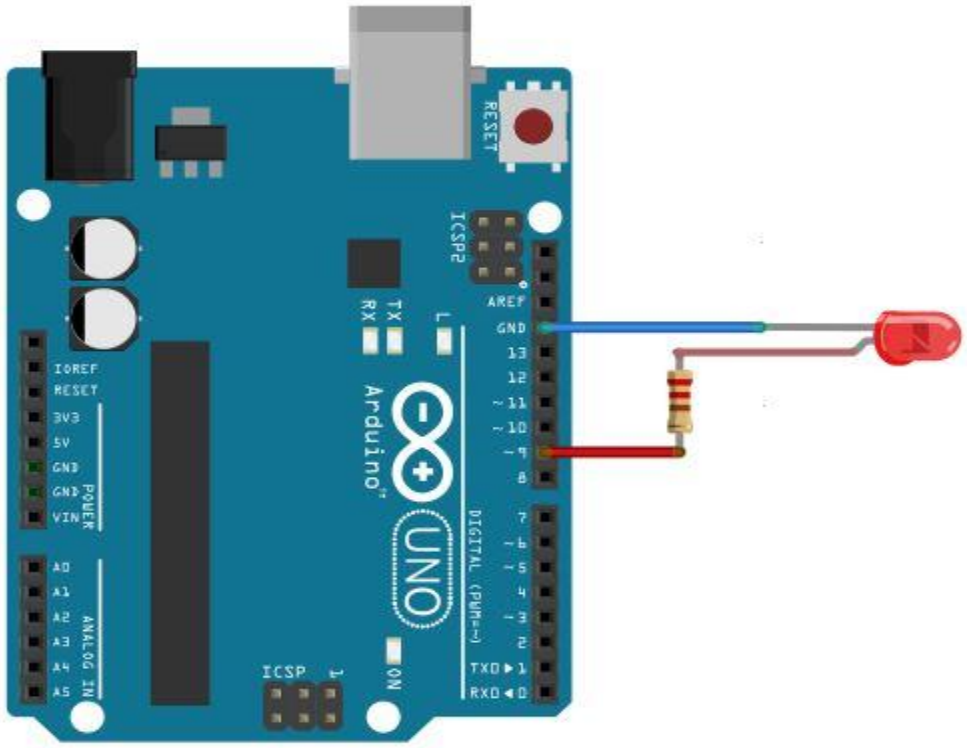
Material:

- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 220 ohm**
- **Arduino Uno R3**

Código

```
int brightness = 0;
int fadeAmount = 5;                    // puntos de desvanecimiento para el brillo


void setup()  {
   pinMode(9, OUTPUT);                          // Se declara el pin 9 como salida
}


void loop()  {
   analogWrite(9, brightness);              // establece el brillo para el pin 9
   brightness = brightness + fadeAmount;     // cambia el brillo para el siguiente ciclo


 if (brightness == 0 || brightness == 255) {
   fadeAmount = -fadeAmount ;
 }
 delay(30);                                          // espera 30 milisegundos para apreciar el
}                                                 // efecto
```

Conexiones

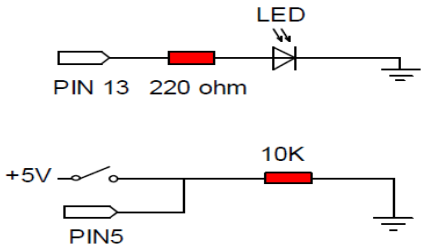


CONCLUSIONES PERSONALES DE LA PRACTICA N° 2

Agregue aquí .....

PRÁCTICA 03 – ALARMA

Cuando se pulsa el pulsador (entrada5 a “0”) se enciende y se apaga de forma intermitente la salida 13 .



Funcionamiento:

Cuando la E5 = 1 Entonces S13 = 0

Cuando la E5 = 0 Entonces S13 = 0-1 (Intermitente 200,200 ms)

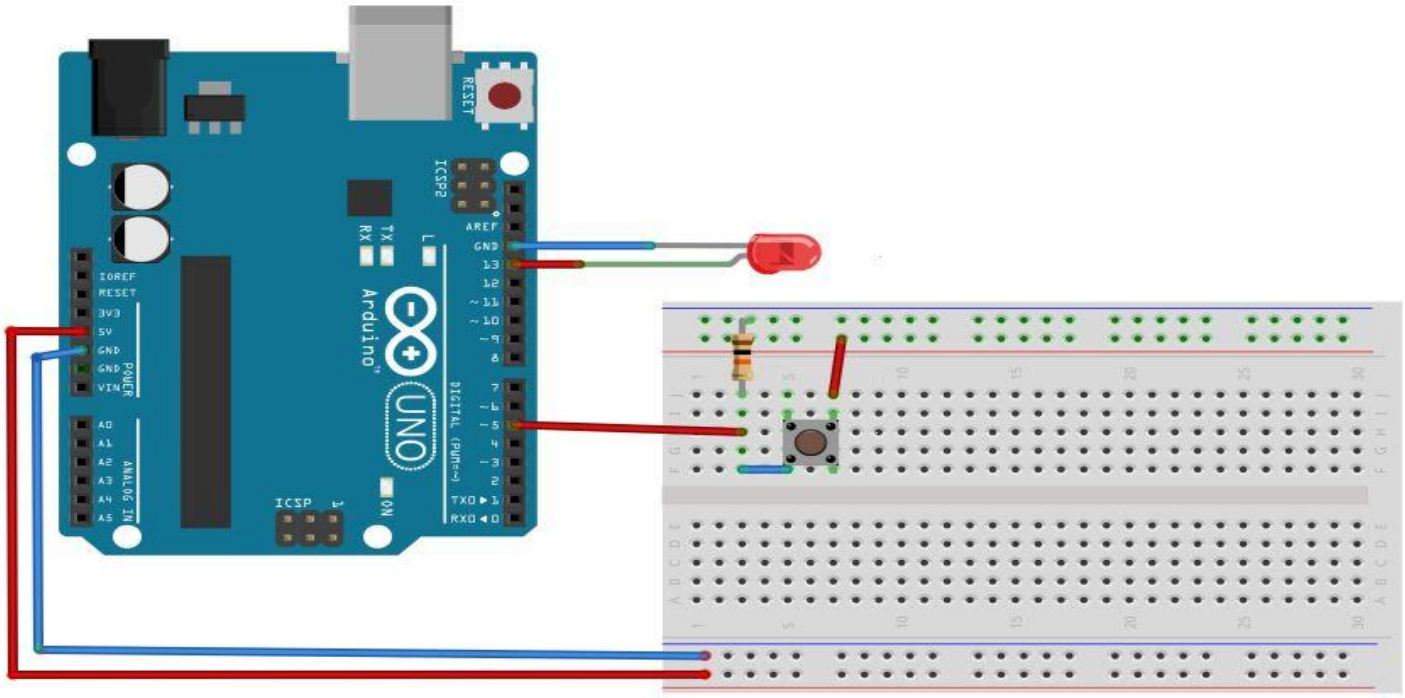
Material:

- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 10 K**
- **1 Push Bottom**
- **Arduino Uno R3**

Código

```
int ledPin= 13;                 // choose the pin for the LED
int inPin= 5;                   // choose the input pin (for a pushbutton)
int val= 0;                     // variable for reading the pin status
void setup() {
   pinMode(ledPin, OUTPUT);   // declare LED as output
   pinMode(inPin, INPUT);     // declare pushbutton as input
}
void loop(){
   val= digitalRead(inPin);   // lee valor de entrada
   if(val== HIGH) {           // chequea si el valor leído es “1” (botón presionado)
      digitalWrite(ledPin, LOW);  // pone el LED en OFF
} else{
   digitalWrite(ledPin, LOW);     // parpadea el LED
   delay(200);
   digitalWrite(ledPin, HIGH);
   delay(200);
   }
}
```

Conexiones

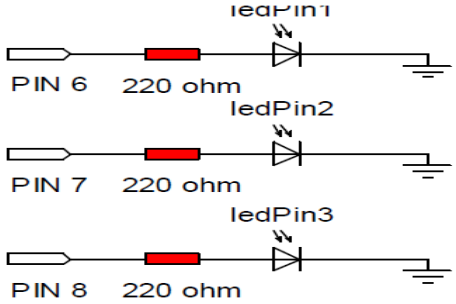


CONCLUSIONES PERSONALES DE LA PRACTICA N° 3

Agregue aquí .....

PRÁCTICA 04 – SECUENCIA BÁSICA DE 3 LEDS

Se trata de encender y apagar 3 LED colocados en las salidas 6, 7 y 8 (PIN6, PIN7 y PIN8) con una cadencia de 200 ms. Las variables asignadas a cada LED son **LEDPIN1, LEDPIN2 Y LEDPIN3.**



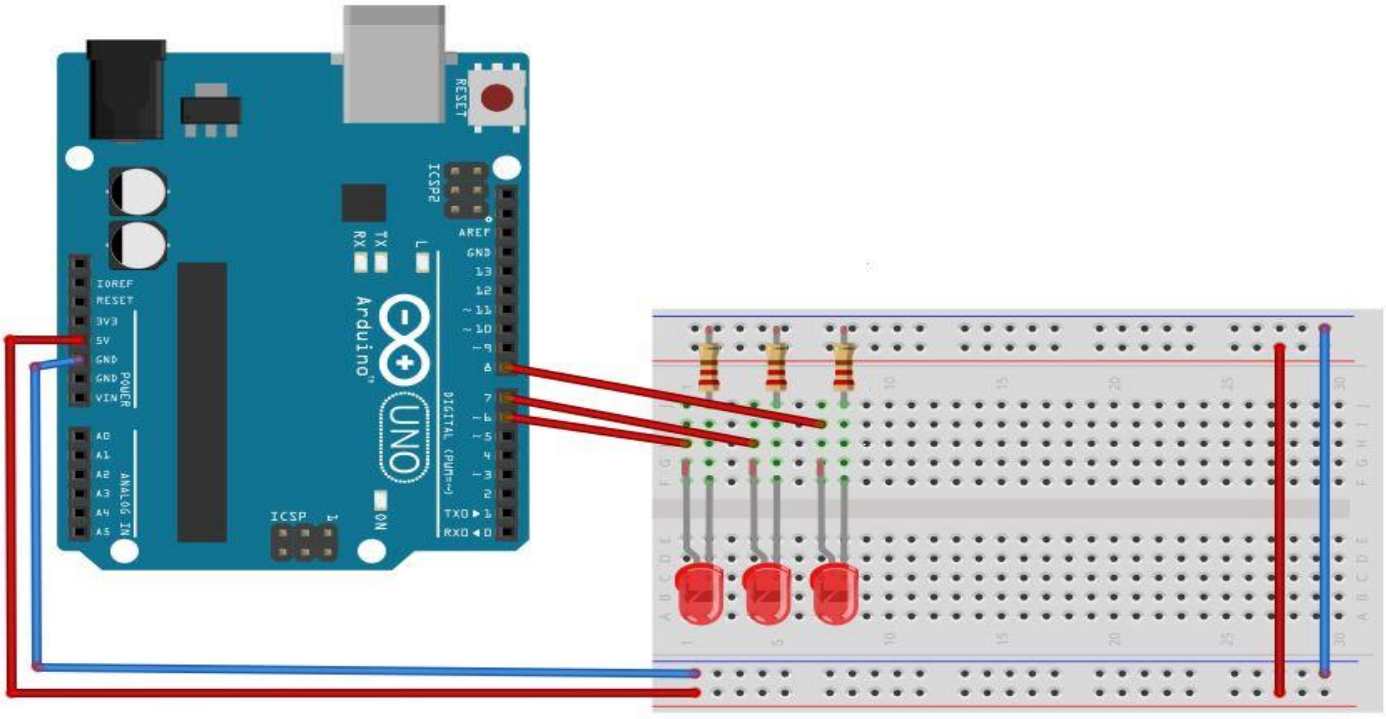
Material:

- **1 Protoboard**
- **3 Diodos LED**
- **3 Resistencias 220 ohm**
- **Arduino Uno R3**

Código

```
int ledPin1 = 6;                    // Define las salidas de los LED´s
int ledPin2 = 7;
int ledPin3 = 8;
void setup() {                              // Configura las SALIDAS
   pinMode(ledPin1, OUTPUT);  // declarar LEDs como SALIDAS
   pinMode(ledPin2, OUTPUT);
   pinMode(ledPin3, OUTPUT);
   digitalWrite(ledPin1, LOW);     // Apaga los LEDs
   digitalWrite(ledPin2, LOW);
   digitalWrite(ledPin3, LOW);
}
void loop(){                                //Bucle de Funcionamiento
   digitalWrite(ledPin1, HIGH);    // Apaga y enciende los leds cada 200 ms
   delay(200);
   digitalWrite(ledPin1, LOW);
   digitalWrite(ledPin2, HIGH);
   delay(200);
   digitalWrite(ledPin2, LOW);
   digitalWrite(ledPin3, HIGH);
   delay(200);
   digitalWrite(ledPin3, LOW);
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 4

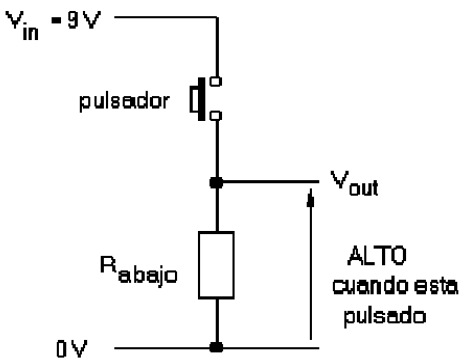
Agregue aquí .....

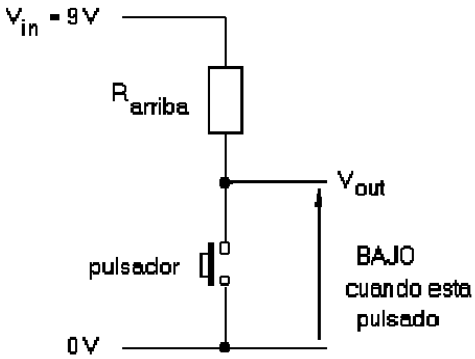
PRÁCTICA 05 – LECTURA DE UN PULSADOR

El pulsador es un componente que conecta dos puntos de un circuito cuando es presionado.

Para generar una señal de tensión con el pulsador, se necesita un divisor de tensión.

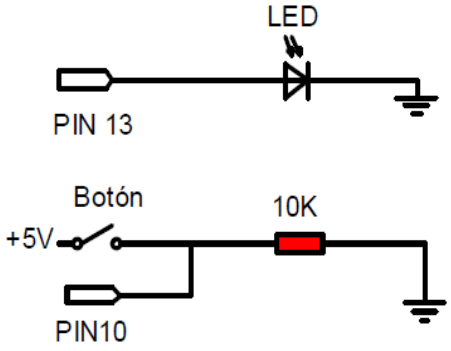
Ejemplo:





La resistencia trabajo (pull-down) en el primer circuito fuerza a Vout, llegando a nivel CERO, hasta que se acciona el pulsador. Este circuito entrega una tensión alta, cuando se presiona el pulsador. Un valor para la resistencia de 10 k es adecuada. En el segundo circuito, la resistencia Rarriba (pull-up) fuerza a nivel ALTO a Vout, mientras no se actúe sobre el pulsador. Al presionar el pulsador, se conecta Vout directamente con 0 V. Es decir, este circuito entrega un nivel BAJO cuando se presiona el pulsador.

Se utiliza una resistencia pull-down, junto con un pulsador, para conectarla a un pin de entrada digital, y de esta forma, poder saber cuándo el pulsador es presionado. Si el pulsador está presionado, el valor del pin 10 será de 0 voltios (LOW) en caso contrario será de + 5 voltios (HIGH). En una placa protoboard debe haber una resistencia de 10K conectada entre el pin de entrada y tierra como se ve el esquema y foto inferiores.



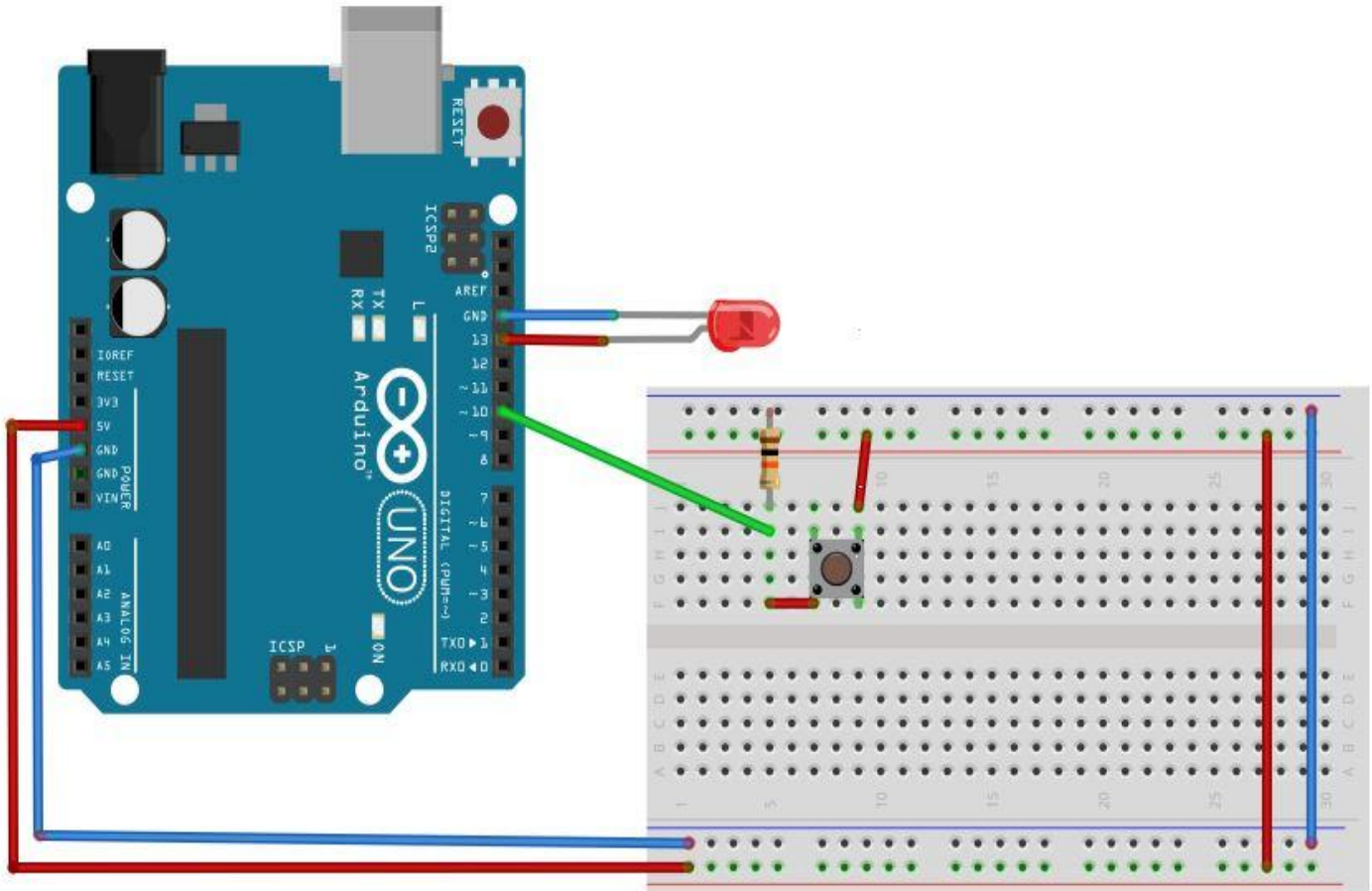
Material:

- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 10 K**
- **1 Push Bottom**
- **Arduino Uno R3**

Código

```
int ledPin = 13;                  // PIN del LED
int inPin = 10;                   // PIN del pulsador
int value = 0;                            // Valor del pulsador

void setup() {
   pinMode(ledPin, OUTPUT);   // Inicializa el pin 13 como salida digital
   pinMode(inPin, INPUT);        // Inicializa el pin 10 como entrada digital
}

void loop() {
   value = digitalRead(inPin);     // Lee el valor de la entrada digital
   digitalWrite(ledPin, value);
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 5

Agregue aquí .....

PRÁCTICA 06 - LECTURA DE SEÑAL ANALÓGICA CON UN POTENCIÓMETRO

El potenciómetro es un dispositivo electromecánico que consta de una resistencia de valor fijo sobre la que se desplaza un contacto deslizante, el cursor, que la divide eléctricamente.
Un potenciómetro es especificado por su resistencia total, R, entre los terminales externos 1 y 3; El movimiento del cursor origina un cambio en la resistencia medida entre el terminal central, 2, y uno cualquiera de los extremos. Este cambio de resistencia puede utilizarse para medir desplazamientos lineales o angulares de una pieza acoplada al cursor.
Se conectan en paralelo al circuito y se comporta como un divisor de tensión.

Circuito:
Se conectan tres cables a la tarjeta Arduino. El primero va a tierra desde el terminal 1 del potenciómetro. El terminal 3 va a la salida de 5 voltios. El terminal 2 va desde la entrada analógica #2 hasta el terminal interno del potenciómetro.
Girando el dial o ajustando el potenciómetro, cambiamos el valor de la resistencia variable. Esto produce oscilaciones dentro del rango de 5 y 0 voltios, que son capturados por la entrada analógica.

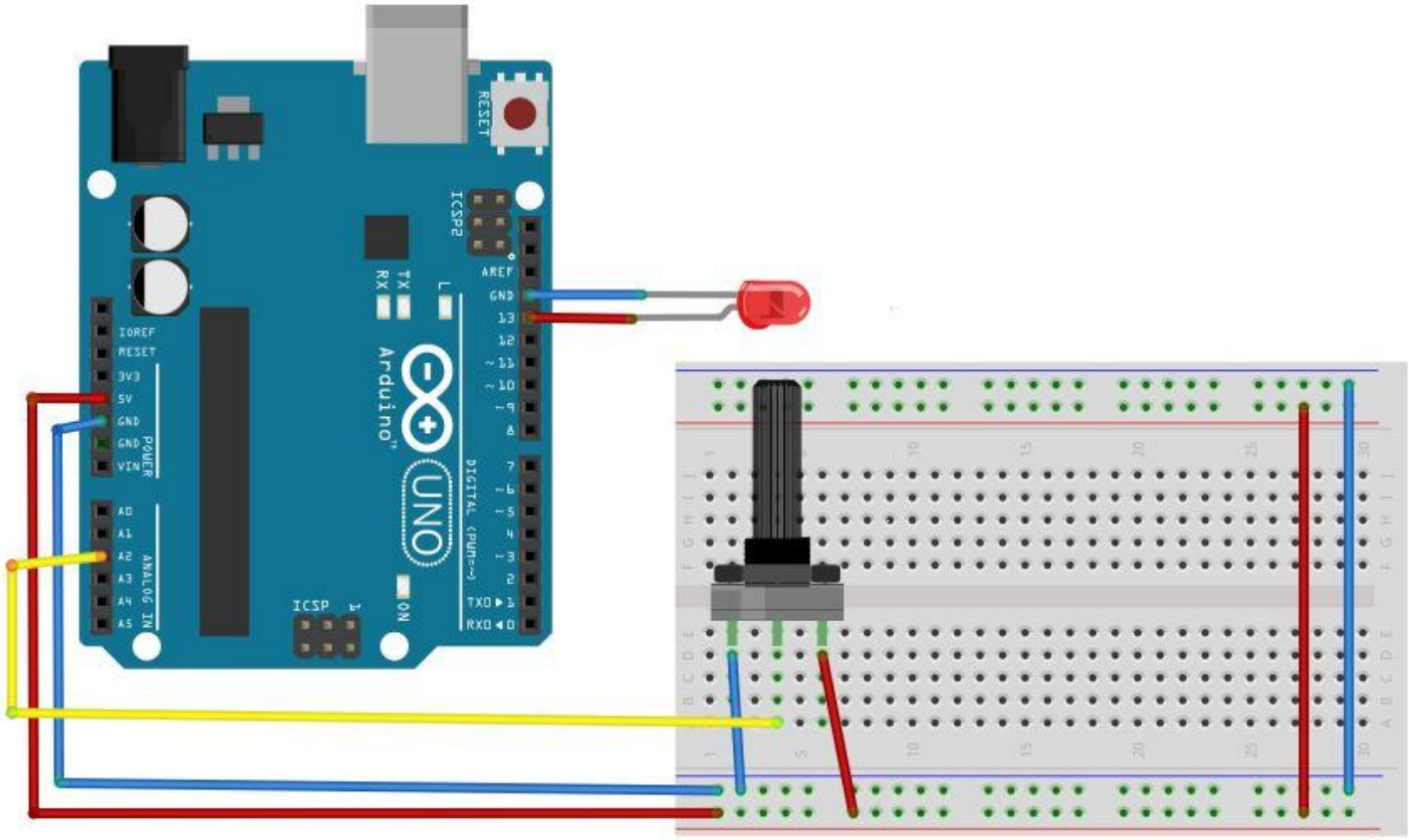
Material:

- **1 Protoboard**
- **1 Diodo LED**
- **1 Potenciómetro**
- **Arduino Uno R3**

Código

```
int potPin = 2;     // seleccionar el pin de entrada analógico para el potenciómetro
int ledPin = 13;  // seleccionar el pin de salida digital para el LED
int val = 0;                  // variable para almacenar el valor capturado desde el sensor
void setup() {
    pinMode(ledPin, OUTPUT);    // declara el ledPin en modo salida
}
void loop() {
   val = analogRead(potPin);       // lee el valor del sensor
   digitalWrite(ledPin, HIGH);       // enciende LED
   delay(val);                                  // detiene el programa por un tiempo “val”
   digitalWrite(ledPin, LOW);       // apaga el LED
   delay(val);                                  // detiene el programa por un tiempo “val”
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 6

Agregue aquí .....

PRÁCTICA 07 – EL AUTO FANTÁSTICO

Se denomina así el ejemplo por los efectos parpadeantes que realiza en los LED´s, de tal manera que he decidido, con el objetivo de aprender programación secuencial y buenas técnicas para programar la placa E/S, que sería interesante usar el término coche fantástico como una metáfora.

Este ejemplo hace uso de 6 LED-s conectados a los PIN 2 a 7 de la placa mediante resistencias de 220 Ohmios. El primer código de ejemplo hace parpadear a los LED en secuencia de uno en uno, utilizando sólo las funciones digitalWrite(pinNum,HIGH/LOW) y delay(time). El segundo ejemplo muestra cómo usar una secuencia de control for(;;;) para hacer lo mismo, pero en menos líneas de código. El tercer y último ejemplo se centra en el efecto visual de apagar y encender los LED-s de una forma más suave.

Material:

- **1 Protoboard**
- **6 Diodos LED**
- **6 Resistencias 220 ohm**
- **Arduino Uno R3**

Código

```
int pin2 = 2; // PIN-es de los LED
int pin3 = 3;
int pin4 = 4;
int pin5 = 5;
int pin6 = 6;
int pin7 = 7;
int timer = 100; // Temporizador
void setup(){
pinMode(pin2, OUTPUT); // Configuración de los PIN-es como salida
pinMode(pin3, OUTPUT);
pinMode(pin4, OUTPUT);
pinMode(pin5, OUTPUT);
pinMode(pin6, OUTPUT);
pinMode(pin7, OUTPUT);
}
void loop() {
digitalWrite(pin2, HIGH); // Enciende y apaga secuencialmente LED-s
delay(timer);
digitalWrite(pin2, LOW);
delay(timer);
digitalWrite(pin3, HIGH);
delay(timer);
digitalWrite(pin3, LOW);
delay(timer);
digitalWrite(pin4, HIGH);
delay(timer);
digitalWrite(pin4, LOW);
delay(timer);
digitalWrite(pin5, HIGH);
delay(timer);
digitalWrite(pin5, LOW);
delay(timer);
digitalWrite(pin6, HIGH);
delay(timer);
digitalWrite(pin6, LOW);
delay(timer);
digitalWrite(pin7, HIGH);
delay(timer);
```

```
digitalWrite(pin7, LOW);
delay(timer);
digitalWrite(pin6, HIGH);
delay(timer);
digitalWrite(pin6, LOW);
delay(timer);
digitalWrite(pin5, HIGH);
delay(timer);
digitalWrite(pin5, LOW);
delay(timer);
digitalWrite(pin4, HIGH);
delay(timer);
digitalWrite(pin4, LOW);
delay(timer);
digitalWrite(pin3, HIGH);
delay(timer);
digitalWrite(pin3, LOW);
delay(timer);
}
```

En el código siguiente las luces se encenderán y apagaran todas en un sentido y luego, acabada la secuencia en sentido contrario:

Código

```
int pinArray[] = {2, 3, 4, 5, 6, 7}; // Define el array de pines

int count = 0; // Contador

int timer = 100; // Temporizador

void setup(){

for (count=0;count<6;count++){ // Configuramos todos los PIN-es

pinMode(pinArray[count], OUTPUT);

}

}

void loop() { // Enciende y apaga secuencialmente los LED-s

for (count=0;count<6;count++) { // utilizando la secuencia de control for(;;)

digitalWrite(pinArray[count], HIGH); // Recorrido de ida
```

```
delay(timer);
digitalWrite(pinArray[count], LOW);
delay(timer);
}
for (count=5;count>=0;count--) {
digitalWrite(pinArray[count], HIGH); // Recorrido de vuelta
delay(timer);
digitalWrite(pinArray[count], LOW);
delay(timer);
}
}
```

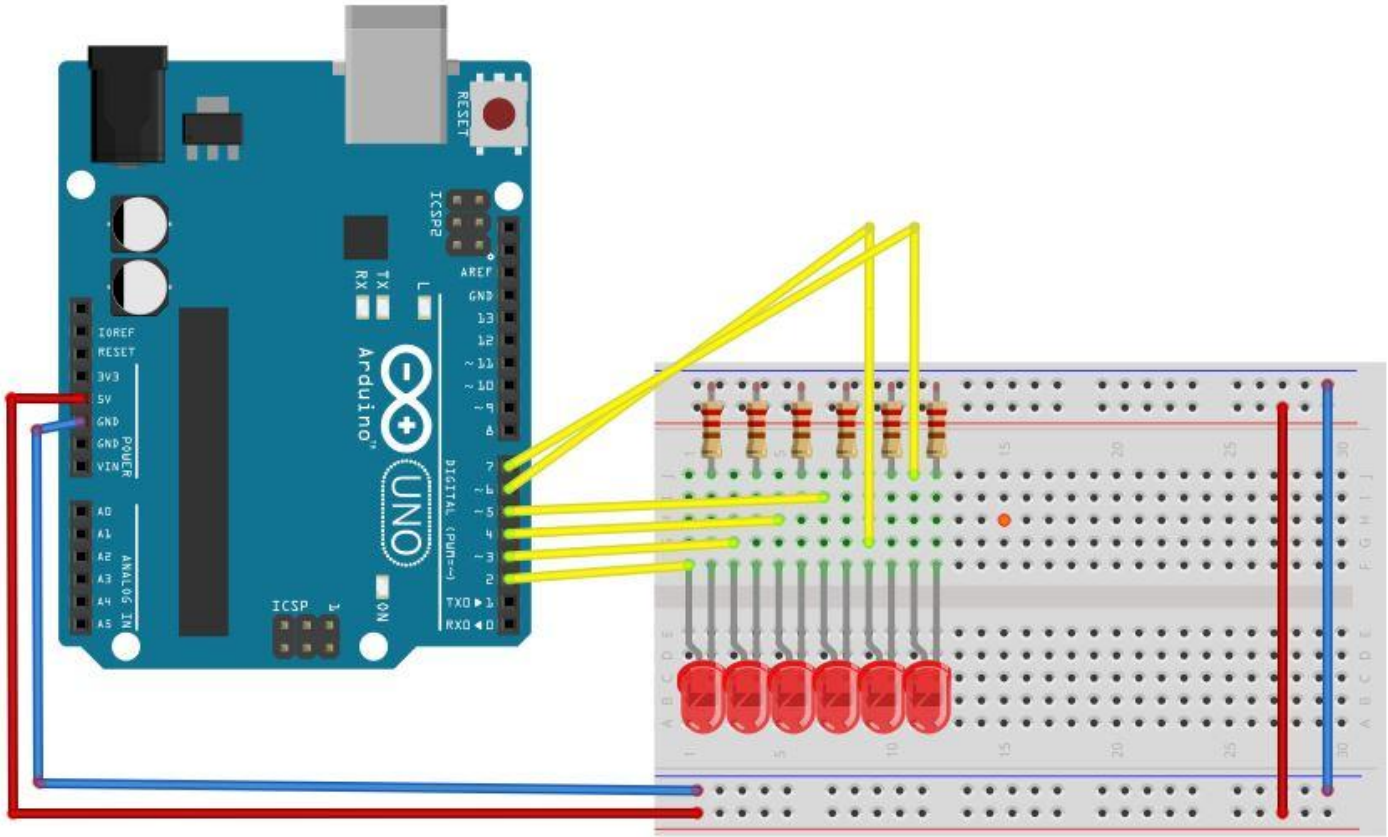
En este caso el efecto que se crea es una estela visual muy vistosa:

Código

```
int pinArray[] = {2, 3, 4, 5, 6, 7}; // PIN-es
int count = 0; // Contador
int timer = 30; // Temporizador
void setup(){
for (count=0;count<6;count++) { // Configuramos todas los PIN-es de golpe
pinMode(pinArray[count], OUTPUT);
}
}
void loop() {
for (count=0;count<5;count++) { // Enciende los LED creando una estela visual
digitalWrite(pinArray[count], HIGH);
delay(timer);
digitalWrite(pinArray[count + 1], HIGH);
delay(timer);
digitalWrite(pinArray[count], LOW);
delay(timer*2);
}
for (count=5;count>0;count--) {
digitalWrite(pinArray[count], HIGH);
```

```
delay(timer);
digitalWrite(pinArray[count - 1], HIGH);
delay(timer);
digitalWrite(pinArray[count], LOW);
delay(timer*2);
}
delay(timer);
for (count=5;count>=0;count--) {
digitalWrite(pinArray[count], HIGH); // Recorrido de vuelta
delay(timer);
digitalWrite(pinArray[count], LOW);
delay(timer);
}
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 7

Agregue aquí .....

PRÁCTICA 08 – CONTADOR

Se trata de contar las veces que se pulsa un botón conectado en la entrada 7 de Arduino a la vez que cada vez que contamos encendemos el led conectado en la salida 13. El valor de la variable que almacena el número de impulsos generados se envía al PC para que se visualice en la pantalla.

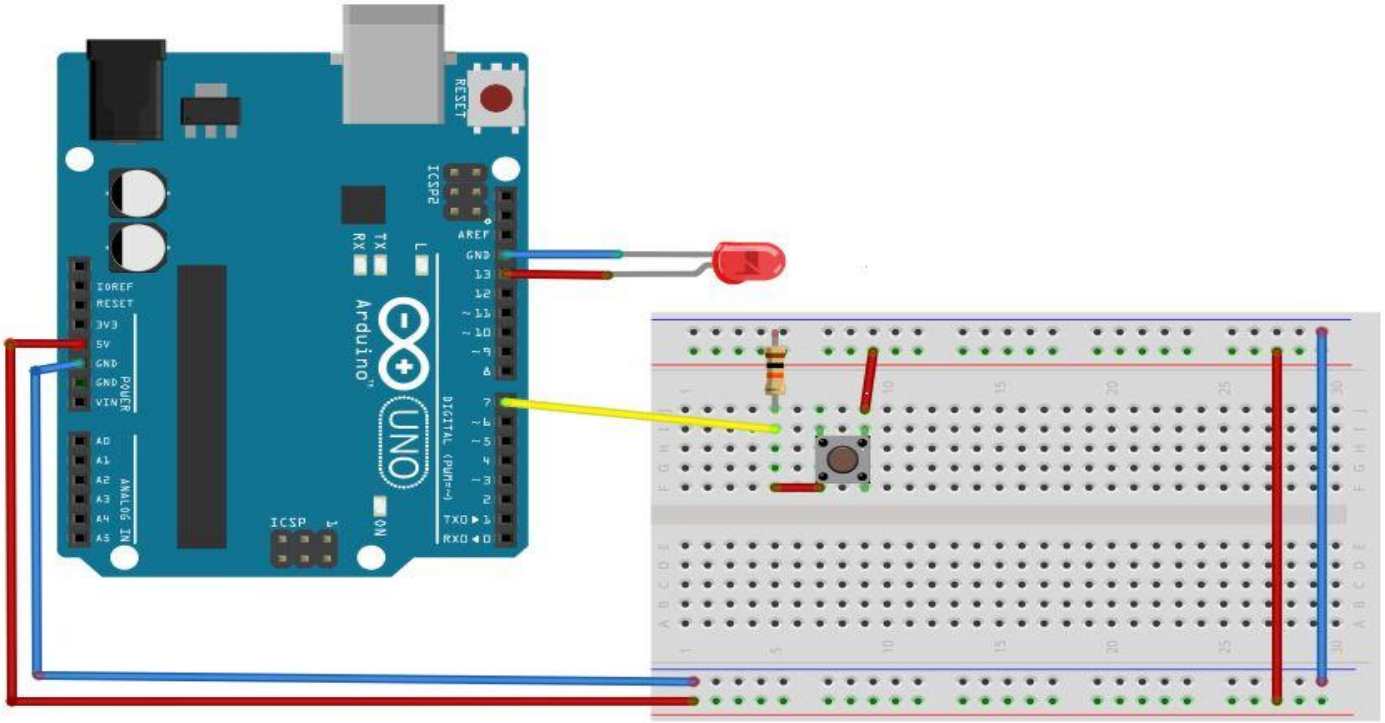
Material:

- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 10 K**
- **1 Push Bottom**
- **Arduino Uno R3**

Código

```
int LED = 13;

int Boton = 7;

int valor = 0;

int contador = 0;

int estadoanteriorboton = 0;

void setup()

{

Serial.begin(9600); // Configura velocidad de transmisión a 9600

pinMode(LED, OUTPUT); // inicializa como salida digital el pin 13

pinMode(Boton, INPUT); // inicializa como entrada digital el 7

}

void loop()

{

valor = digitalRead(Boton); // lee el valor de la entrada digital pin 7

digitalWrite(LED, valor);

if(valor != estadoanteriorboton){

if(valor == 1){

contador++;

Serial.print(contador);

Serial.write(10);

Serial.write(13);

}

}

estadoanteriorboton = valor;

}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 8

Agregue aquí .....

PRÁCTICA 09 - CONTROL DE ILUMINACIÓN DE UNA LÁMPARA

Con esta aplicación se pretende controlar el grado de iluminación de una lámpara (simulada con un LED) mediante un pulsador. Si no pulsamos el pulsador (entrada 0) la lámpara incrementar y decrementar su brillo o nivel de iluminación. Si pulsamos (entrada 1) la lámpara se encenderá y apagará con una cadencia de 50 mseg.

Material:

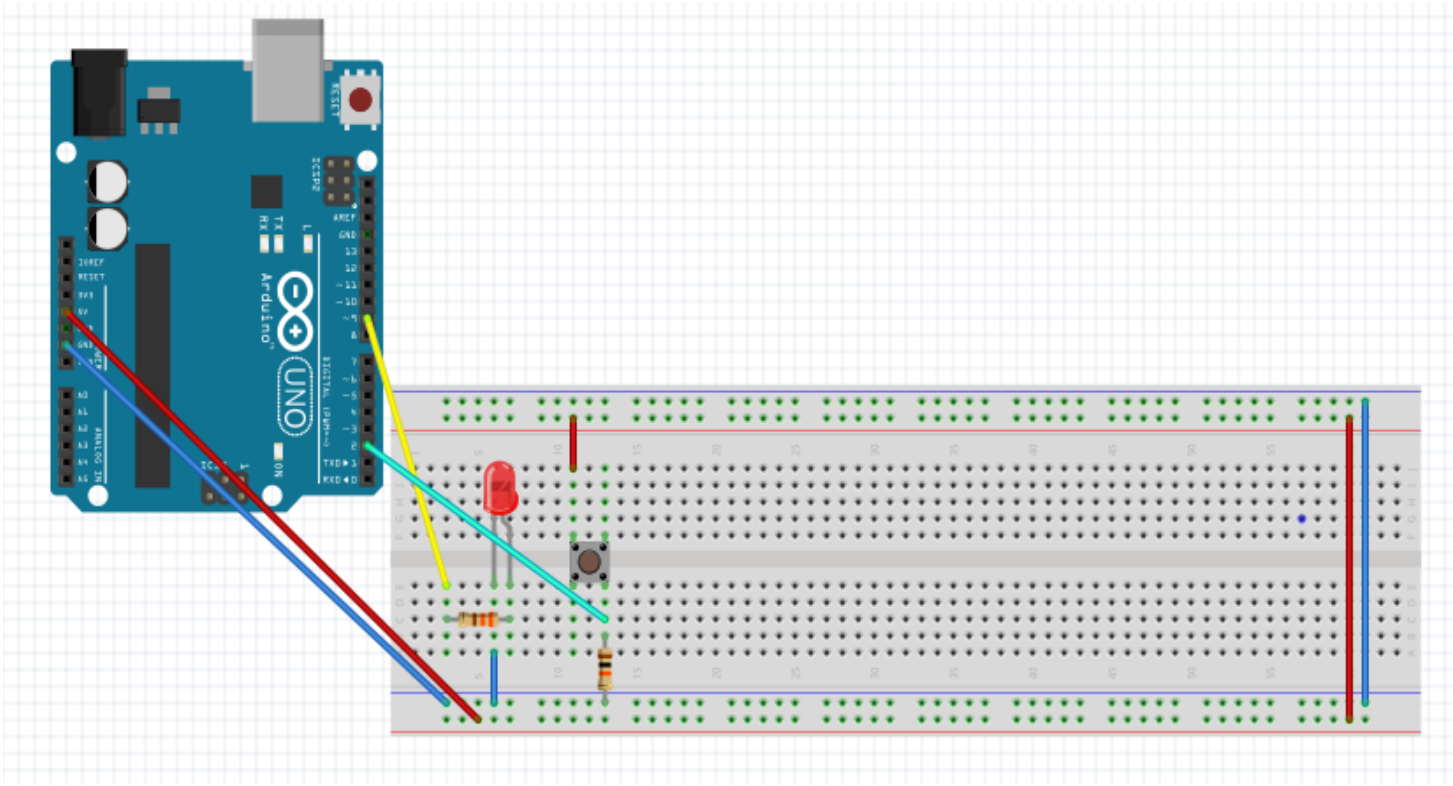
- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 10 K**
- **1 Push Bottom**
- **Arduino Uno R3**

Código

```
int ledPin = 9; // Selección del PIN de salida Analógica

int inputPin = 2; // Selección del PIN para la entrada de pulsador

int val = 0; // variable para leer el estado del pulsador

int fadeval = 0;

void setup() {

pinMode(ledPin, OUTPUT); // designación de salida Analógica

pinMode(inputPin, INPUT); // designación de pulsador de entrada

}

void loop(){

val = digitalRead(inputPin); // leer valor de entrada

if (val == HIGH) { // Botón pulsado

digitalWrite(ledPin, LOW); // puesta a "0" de la salida

delay(500);

digitalWrite(ledPin, HIGH); // puesta a "1" de la salida

delay(500);

}

else { // Si se presiona el boton
```

```
for(fadeval = 0 ; fadeval <= 255; fadeval+=5) { // valor de salida analógica

//asciende de min a max)

analogWrite(ledPin, fadeval); // fija el valor en la salida ( desde 0-255)

delay(100);

}

for(fadeval = 255; fadeval >=0; fadeval-=5) { // valor de salida analógica desciende

//(desde max to min)

analogWrite(ledPin, fadeval);

delay(100);

}

}

}
```

Conexiones

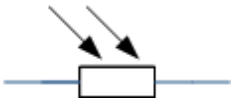


CONCLUSIONES PERSONALES DE LA PRACTICA N° 9

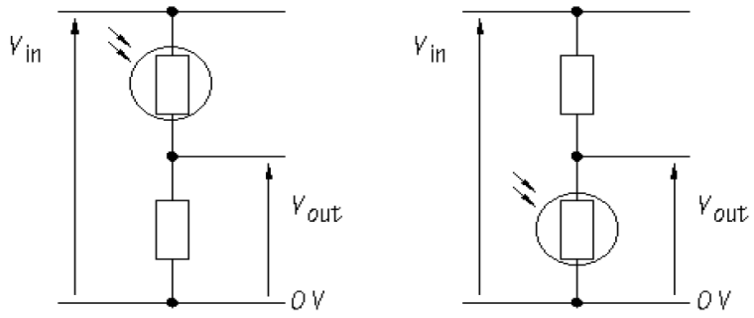
Agregue aquí .....

PRÁCTICA 10 – SENSOR DE LUZ (LDR)

Un LDR es una resistencia variable, que varía su valor dependiendo de la cantidad de luz que incide sobre su superficie. Cuanta más intensidad de luz incide en la superficie de la LDR menor será su resistencia y cuanto menos luz incide mayor será la resistencia. Suelen ser utilizados como sensores de luz ambiental o como una fotocélula que activa un determinado proceso en ausencia o presencia de luz.



Un sensor de luz se compone de una LDR como parte de un divisor de tensión resistivo.
Ejemplo:



Vout=((Rbotton/(Rbottom+Rtop))*Vin

Si la LDR es usada como Rtop, como en el primer circuito, da tensión alta (HIGH) en la salida cuando la LDR está en la luz, y una tensión baja (LOW) en la salida cuando la LDR está en la sombra.

La acción del divisor de tensión es inversa cuando la LDR es usada como Rbottom en lugar de Rtop, como en el segundo circuito. El circuito da tensión Baja (LOW) en la salida cuando la LDR está en la luz, y una tensión alta (HIGH) en la salida cuando la LDR está en la sombra. El circuito divisor de tensión dará una tensión de la salida que cambia con la iluminación, de forma inversamente proporcional a la cantidad de luz que reciba (sensor de oscuridad).

Material:

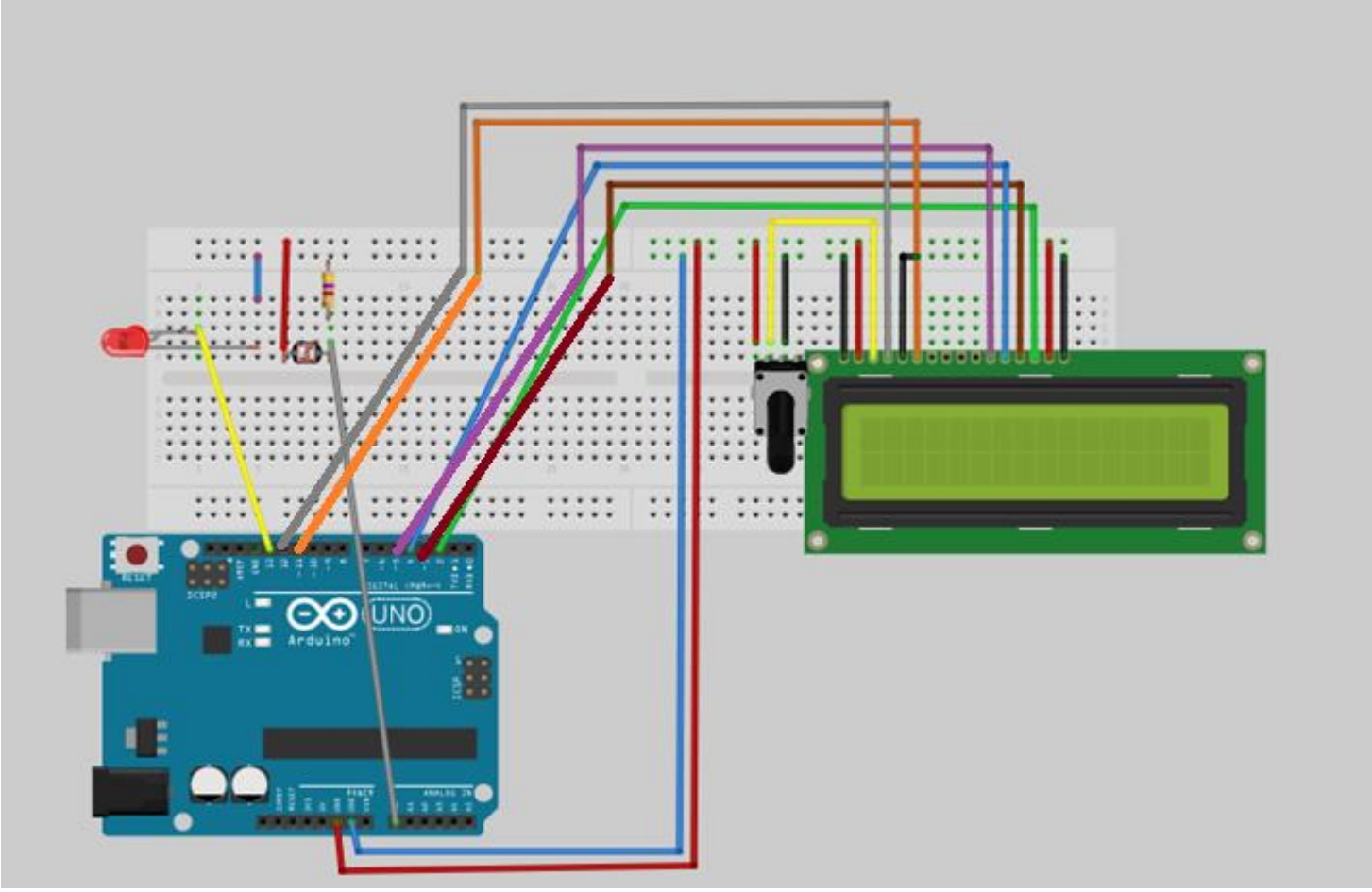
- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 5 K**
- **1 LDR**
- **Arduino Uno R3**

Código

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
        const int pinSensor = A0;     // pin del sensor de luz, va a la pata central del sensor
            const int pinLed    = 13;      // pin para el led
            const int umbral    = 20;     // umbral de la luz, ya en grados centigrados
            long valorSensor = 0;          // variable para guardar el valor leido del sensor
            float temperatura = 0;        // variable para guardar la medición de iluminosidad
            void setup() {
               // declaramos el pin del Led de salida
               pinMode(pinLed, OUTPUT);
               //inicializamos la comunicacion serial
               Serial.begin(9600);
            lcd.begin(16, 2);
              }
            void loop() {
               // leemos el valor del sensor
               valorSensor = analogRead(pinSensor);
          valorSensor = valorSensor * 100 / 1023;
           if(valorSensor < 15){
            valorSensor = 0;
           }
           Serial.print("La luminosidad es de: ");
```

```
            Serial.print(valorSensor);
        Serial.println(" %");
        if (valorSensor != 0){
            digitalWrite(pinLed, HIGH);
          } else {

            digitalWrite(pinLed, LOW);
          }
        lcd.print(valorSensor);
        lcd.print("% Luminosidad");
        delay(500);
        lcd.clear();
}
```

Conexiones

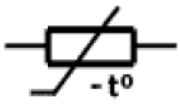


CONCLUSIONES PERSONALES DE LA PRACTICA N° 10

Agregue aquí .....

PRÁCTICA 11 – SENSOR DE TEMPERATURA

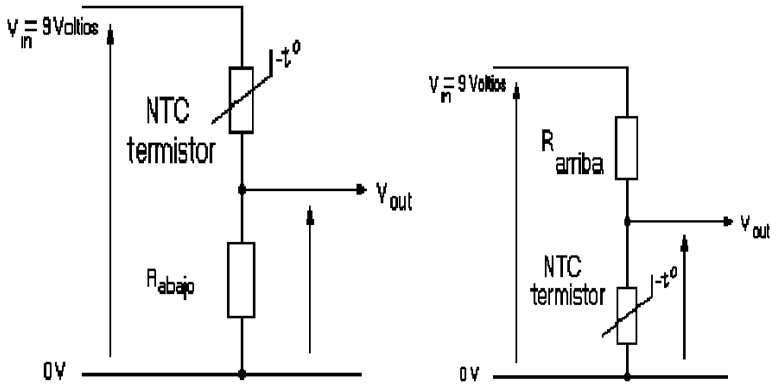
En este ejemplo se trata de medir la temperatura desde el PIN3 de entrada analógica y ver si este valor supera un valor dado de 500 (medida absoluta) si supera este valor activará la salida digital PIN13 y si no la apagara. Además queremos que se muestre en el monitor de salida del IDE Arduino el valor leído. D sensor utilizaremos un sensor del tipo NTC. Un NTC o termistor NTC es una resistencia variable, que varía su valor dependiendo de la temperatura ambiente. Cuanta más temperatura menor será su resistencia y cuanta menos temperatura mayor será la resistencia. Suelen ser utilizados en alarmas.



Un sensor de temperatura se compone de un NTC como parte de un divisor de tensión resistivo.

Ejemplo:

Como alarma de incendio o sensor de calor, utilizaremos un circuito que entregue una tensión alta cuando se detecten las condiciones de temperatura caliente. Necesitamos poner un divisor de tensión con un termistor NTC en la posición que ocupa **R**arriba:



Como alarma de frío o sensor de frío, usaremos un circuito que dé una tensión alta en condiciones frías. Necesitamos un divisor de voltaje con el termistor NTC en lugar de **R**bajo.

Material:

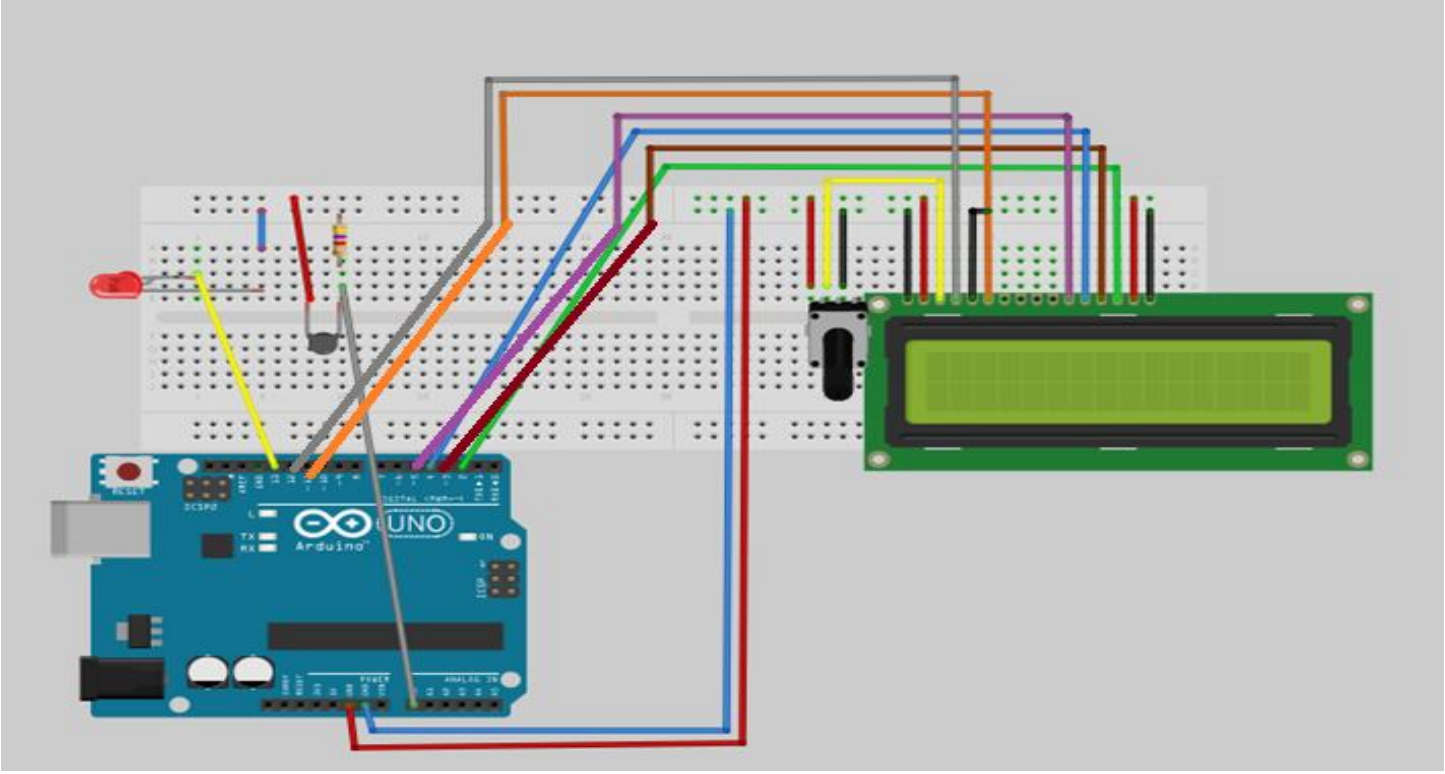
- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 1 K**
- **1 NTC**
- **Arduino Uno R3**

Código

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int pinSensor = A0;     // pin del sensor de temperatura, va a la pata central del sensor
const int pinLed    = 13;      // pin para el led
int valorSensor = 0;          // variable para guardar el valor leido del sensor
float temperatura = 0;        // variable para guardar la temperatura

void setup() {
                // declaramos el pin del Led de salida
                pinMode(pinLed, OUTPUT);
                //inicializamos la comunicacion serial
                Serial.begin(9600);
            lcd.begin(16, 2);
  }

void loop() {
                // leemos el valor del sensor
                valorSensor = analogRead(pinSensor);
                // este valor lo convertimos a milivolts
```

```
        float milivolts = (valorSensor / 1023.0) * 5000;
        // y lo convertimos a grados centigrados
        //  1 grado centigrado = 10 milivolts
        temperatura = milivolts/100;
        Serial.print("La temperatura es de: ");
        Serial.print(temperatura);
      Serial.println(" grados centigrados");
      lcd.print(temperatura);
      lcd.print(" oC");
      if(temperatura > 30){
        digitalWrite(pinLed, HIGH);
        } else {
          digitalWrite(pinLed, LOW);
        }
      delay(500);
      lcd.clear();
}
```

Conexiones



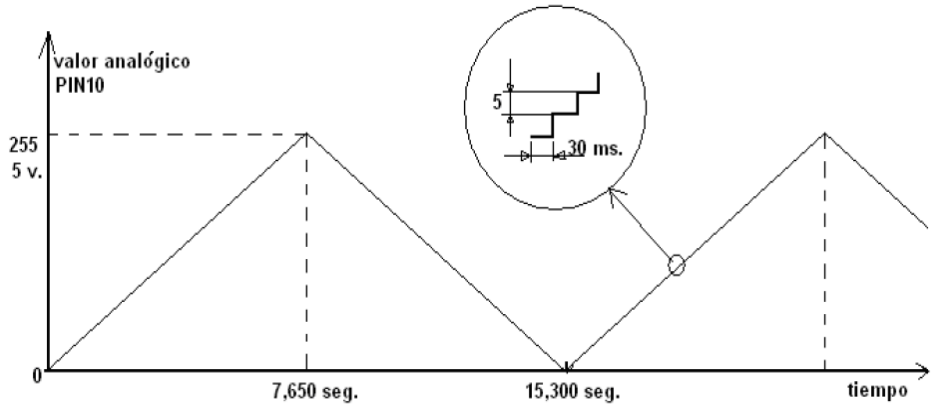
CONCLUSIONES PERSONALES DE LA PRACTICA N° 11

Agregue aquí .....

PRÁCTICA 12 - CONTROL DE UN MOTOR DE DC CON UN TRANSISTOR

Con este ejemplo vamos a controlar la velocidad de un motor de cc mediante la utilización de un transistor BD137. Se trata de utilizar la posibilidad de enviar una señal de PWM a una de las salidas configurables como salidas analógicas. Téngase en cuenta que el motor debe ser de bajo consumo por dos motivos: primero porque si alimentamos en las pruebas desde el conector USB no debemos sacar demasiada corriente del ordenador y segundo porque el transistor es de una corriente limitada.

El diodo 1N4001 se coloca como protección para evitar que las corrientes inversas creadas en el bobinado del motor puedan dañar el transistor.



30 ms. x 255 = 7,650 seg.

La tensión que sacaremos a la salida 10 (analógica tipo PWM) variara en forma de rampa ascendente y descendente de manera cíclica tal como vemos en la figura. Este efecto lo conseguimos con una estructura del tipo for:

```
for (valor = 0 ; valor <= 255; valor +=5)    (ascendente)
for (valor = 255; valor >=0; valor -=5)      (descendente)
```

Obsérvese que los incrementos del valor de la tensión van de 5 en 5 y tenemos que considerar que 0v equivale a 0 y 5 v. equivale a 255.

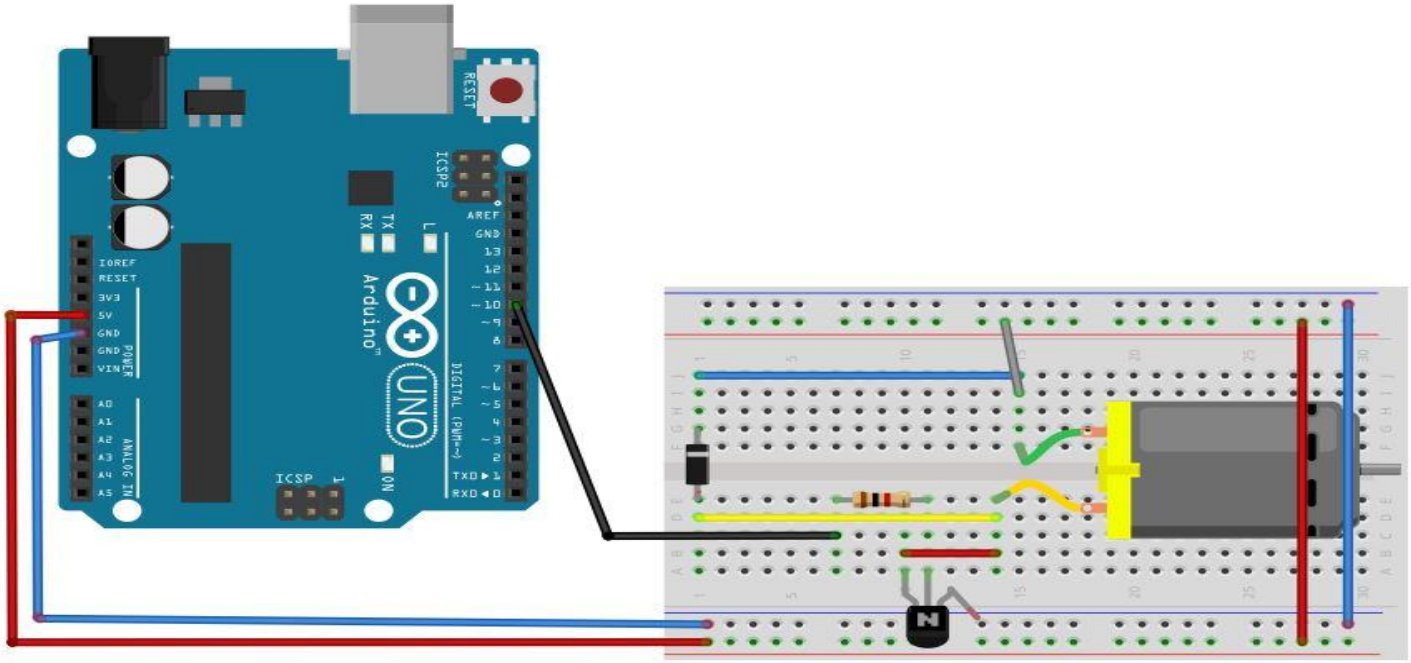
Material:

- **1 Protoboard**
- **1 Diodo LED**
- **1 Resistencia 1 K**
- **1 Transistor 2N2222**
- **1 Diodo 1N4001**
- **1 Motor DC**
- **Arduino Uno R3**

Código

```
int valor = 0; // variable que contiene el valor a sacar por el terminal analógico
int motor = 10; // motor conectado al PIN 10
void setup() { } // No es necesario
void loop() {
for(valor = 0 ; valor <= 255; valor +=5) {
// se genera una rampa de subida de tensión de 0 a 255 es decir de 0 a 5v
analogWrite(motor, valor);
delay(30); // espera 30 ms para que el efecto sea visible
}
for(valor = 255; valor >=0; valor -=5) {
// se genera una rampa de bajada de tensión de 255 a 0 es decir de 5 a 0v
analogWrite(motor, valor);
delay(30);
}
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 12

Agregue aquí .....

PRÁCTICA 13 - CONTROL DE UN MOTOR A VELOCIDAD Y SENTIDO DE GIRO VARIABLES

Vamos a demostrar el control simultáneo de la velocidad y del sentido de un motor de corriente continua. **!OJO probar en Arduino con un sólo motor¡**

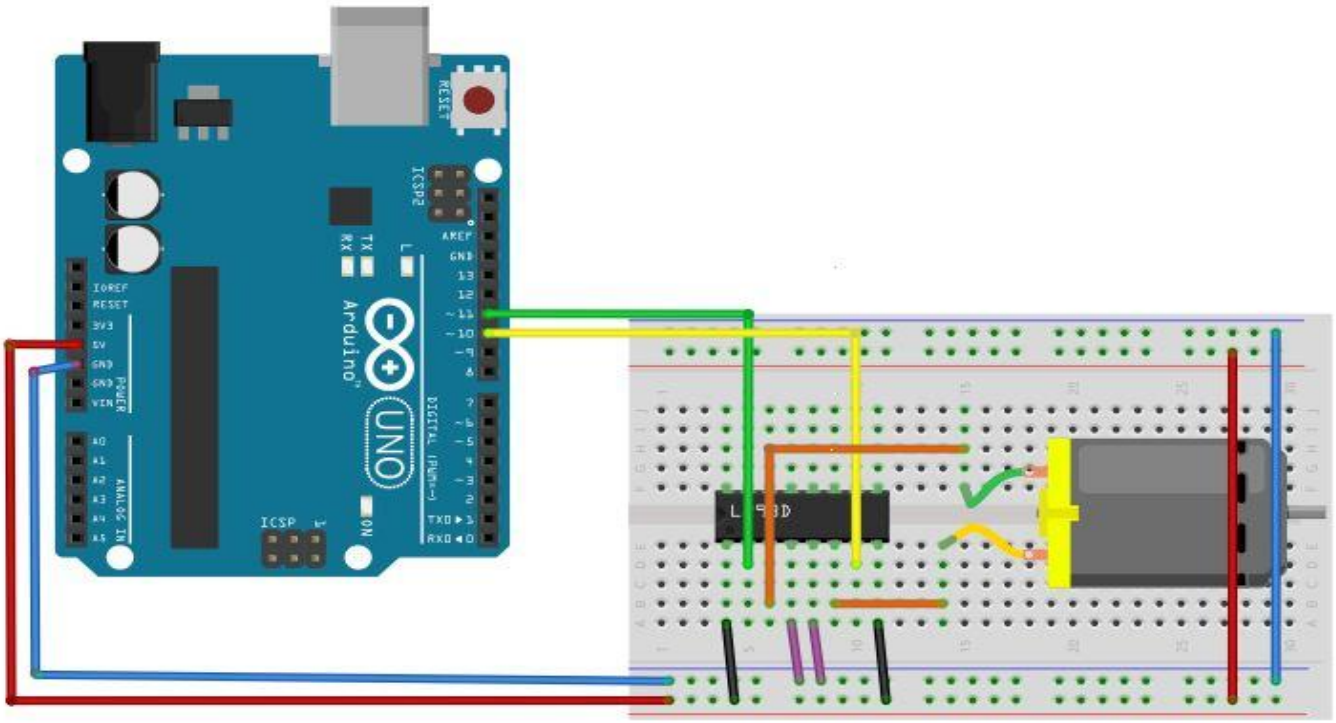
Material:

- **1 Protoboard**
- **1 Motor DC**
- **1 L293D**
- **Arduino Uno R3**

Código

```
// Control de Motor con driver L293D
int valor = 0; // variable que contiene el valor
int motorAvance = 10; // Avance motor --> PIN 10
int motorRetroceso = 11; // Retroceso motor --> PIN 11
void setup() { } // No es necesario
void loop() {
analogWrite(motorRetroceso, 0); // Motor hacia delante ... sube la velocidad
for(valor = 0 ; valor <= 255; valor+=5) {
analogWrite(motorAvance, valor);
delay(30);
}
for(valor = 255; valor >=0; valor-=5) { // Motor hacia delante ... baja la velocidad
analogWrite(motorAvance, valor);
delay(30);
}
analogWrite(motorAvance, 0); // Motor hacia detrás ... sube la velocidad
for(valor = 0 ; valor <= 255; valor+=5) {
analogWrite(motorRetroceso, valor);
delay(30);
}
for(valor = 255; valor >=0; valor-=5) { // Motor hacia detrás ... baja la velocidad
analogWrite(motorRetroceso, valor);
delay(30);
}
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 13

Agregue aquí .....

PRÁCTICA 14 – RELEVADOR

Este sencillo ejemplo enseña como encender una bombilla de 220V de corriente alterna (AC) mediante un circuito de 5V de corriente continua (DC) gobernado por Arduino. Se puede utilizar con cualquier otro circuito de 220V con un máximo de 10A (con el relevador del ejemplo).

¿Qué es un relevador?

El relé es un dispositivo electromecánico, que funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes. De aquí extraemos una información muy importante: Podemos separar dos circuitos de forma que funcionen con voltajes diferentes. Uno a 5V (Arduino) y otro a 220V (la bombilla).

Como se ve en el esquema inferior hay dos circuitos. El del cableado NEGRO funciona a 5V de DC y el del cableado ROJO a 220V de AC.

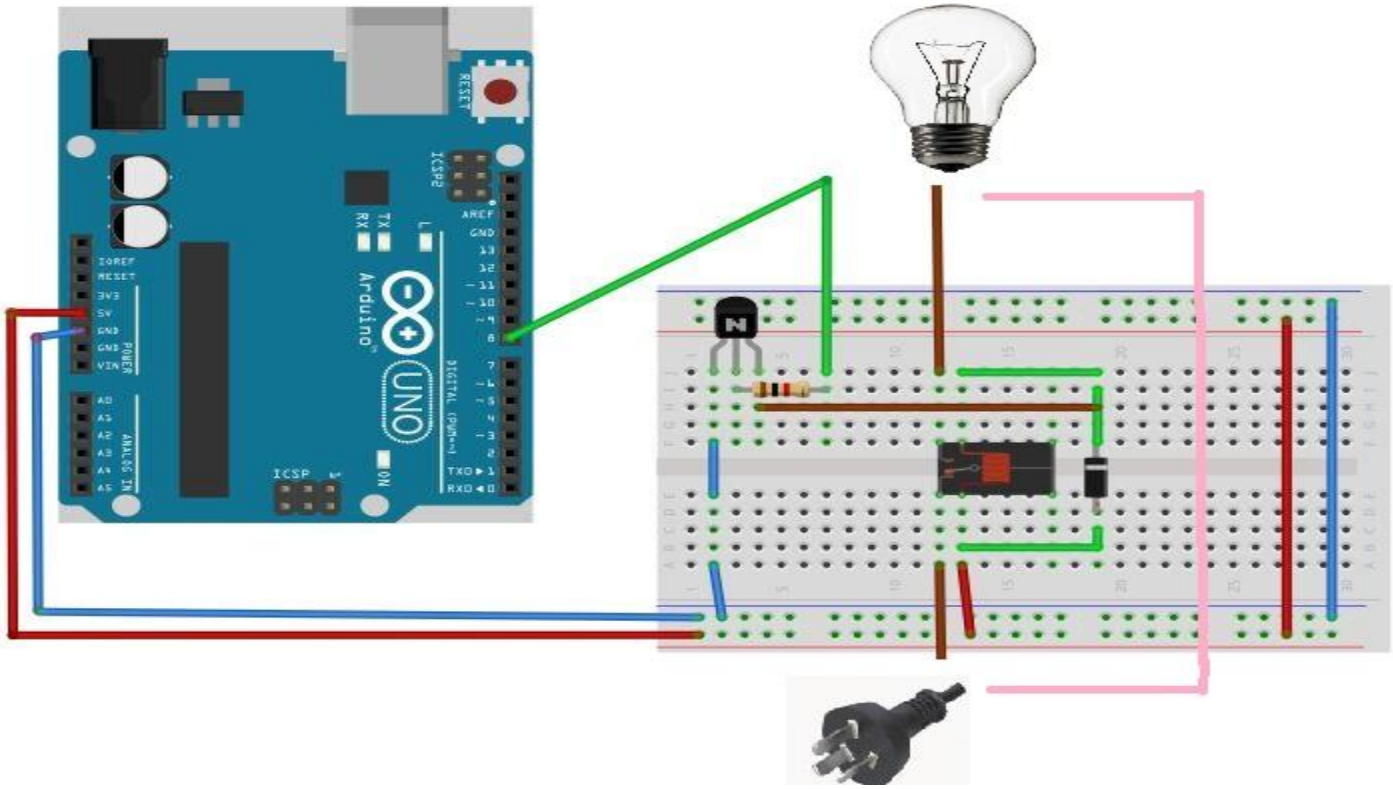
Material:

- **1 Protoboard**
- **1 Motor DC**
- **1 Relevador 220v**
- **1 Foco**
- **Cable para corriente**
- **1 Transistor BD137**
- **1 Diodo 1N4001**
- **1 Resistencia 10k**
- **Arduino Uno R3**

Código

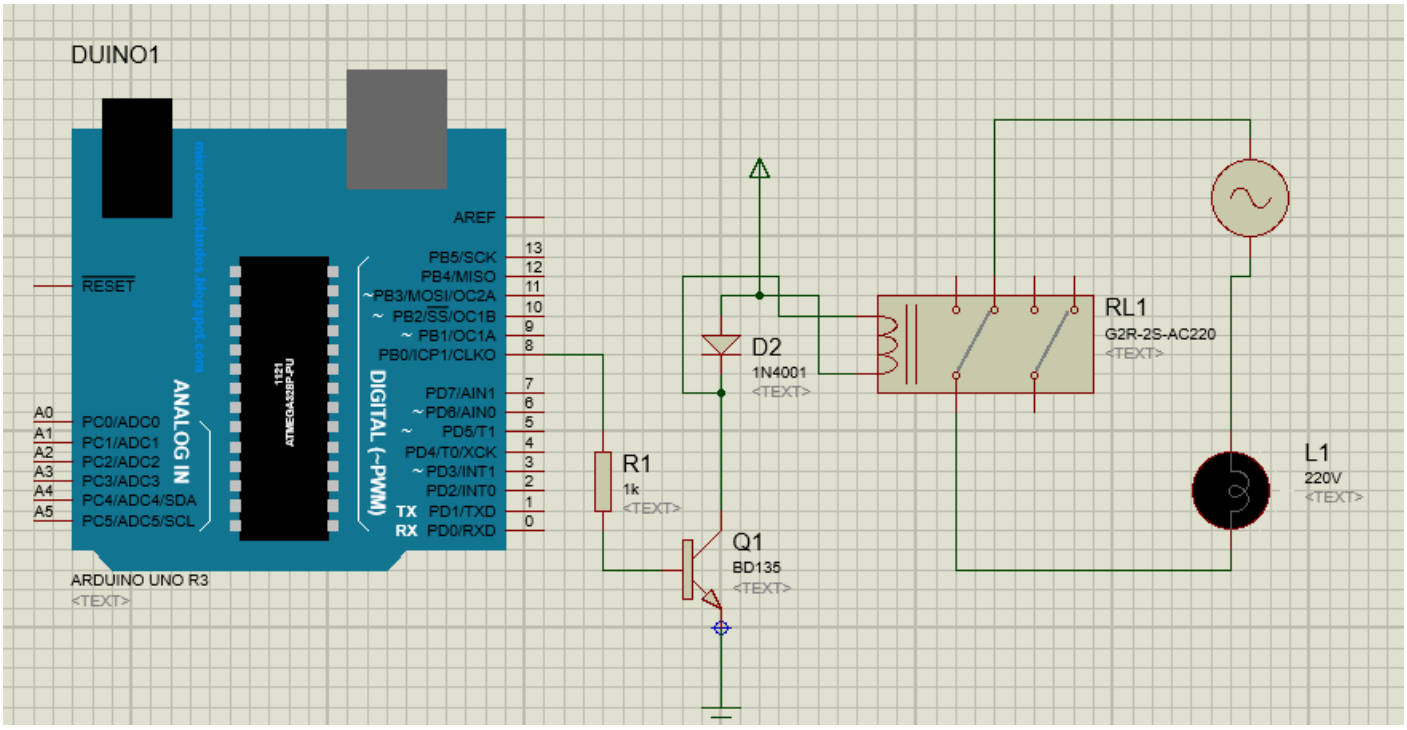
```
/*
Enciende y apaga una bombilla de 220V, cada 2 segundos, mediante
un relé conectado al PIN 8 de Arduino
*/
int relayPin = 8; // PIN al que va conectado el relé
void setup(){
pinMode(relayPin, OUTPUT);
}
void loop() {
digitalWrite(relayPin, HIGH); // ENCENDIDO
delay(2000);
digitalWrite(relayPin, LOW); // APAGADO
delay(2000);
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 14

Agregue aquí .....



PRÁCTICA 15 – LCD

Se trata del control de un Visualizador de Cristal Líquido (LCD), el cual cuenta con un controlador en C.I. denominado HD44780.

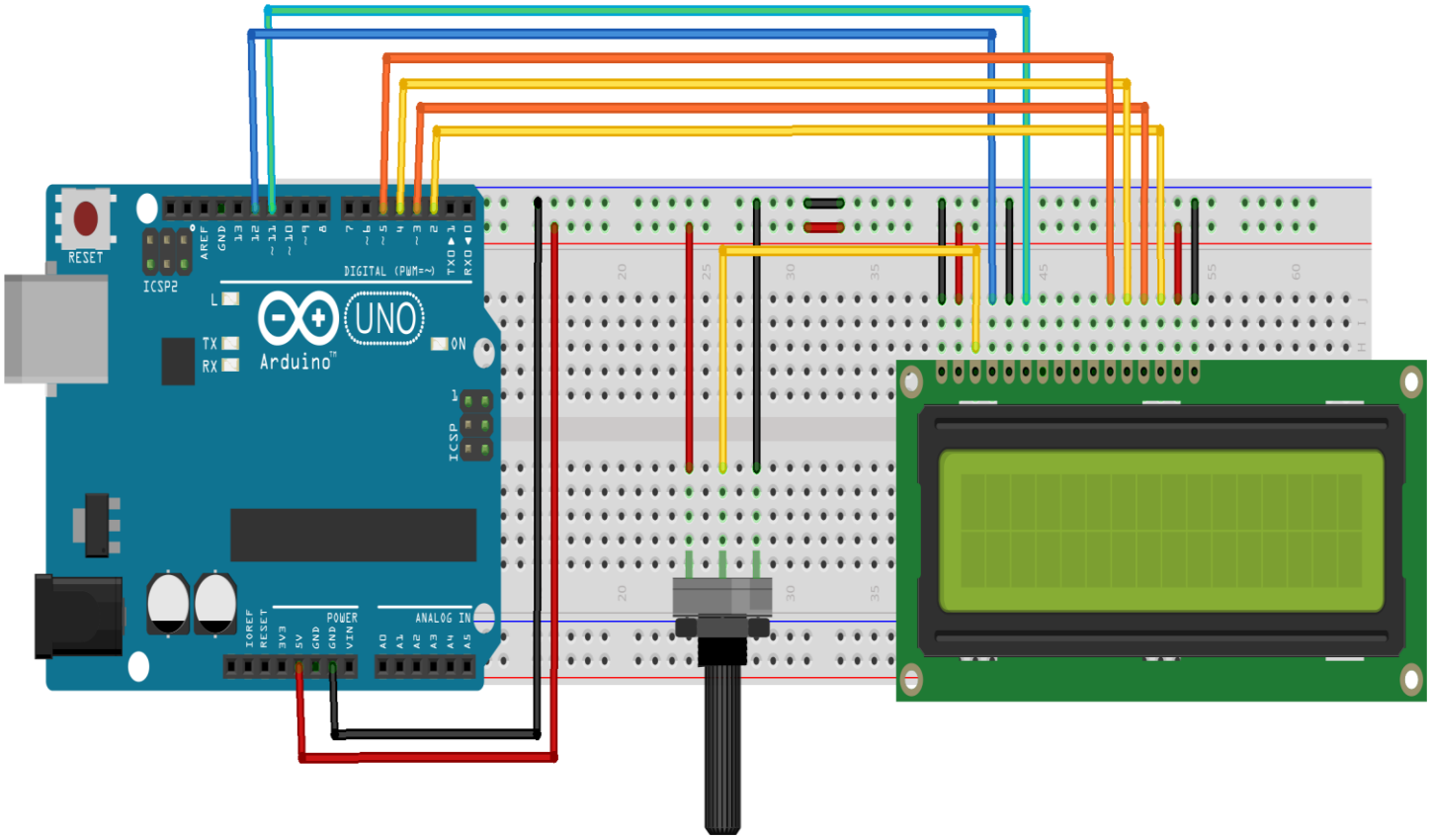
Material:

- **1 Protoboard**
- **1 LCD 16X2**
- **1 Potenciómetro 10k**

Código

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
  lcd.print("La ultima para el 100");

 delay(1000);
}
void loop() {
  for (int positionCounter = 0; positionCounter < 13; positionCounter++) {
    lcd.scrollDisplayLeft();
    delay(150);
  }

  for (int positionCounter = 0; positionCounter < 29; positionCounter++) {
     lcd.scrollDisplayRight();
    delay(150);
  }

  for (int positionCounter = 0; positionCounter < 16; positionCounter++) {
    lcd.scrollDisplayLeft();
    delay(150);
  }
  delay(1000);
}
```

Conexiones



CONCLUSIONES PERSONALES DE LA PRACTICA N° 15

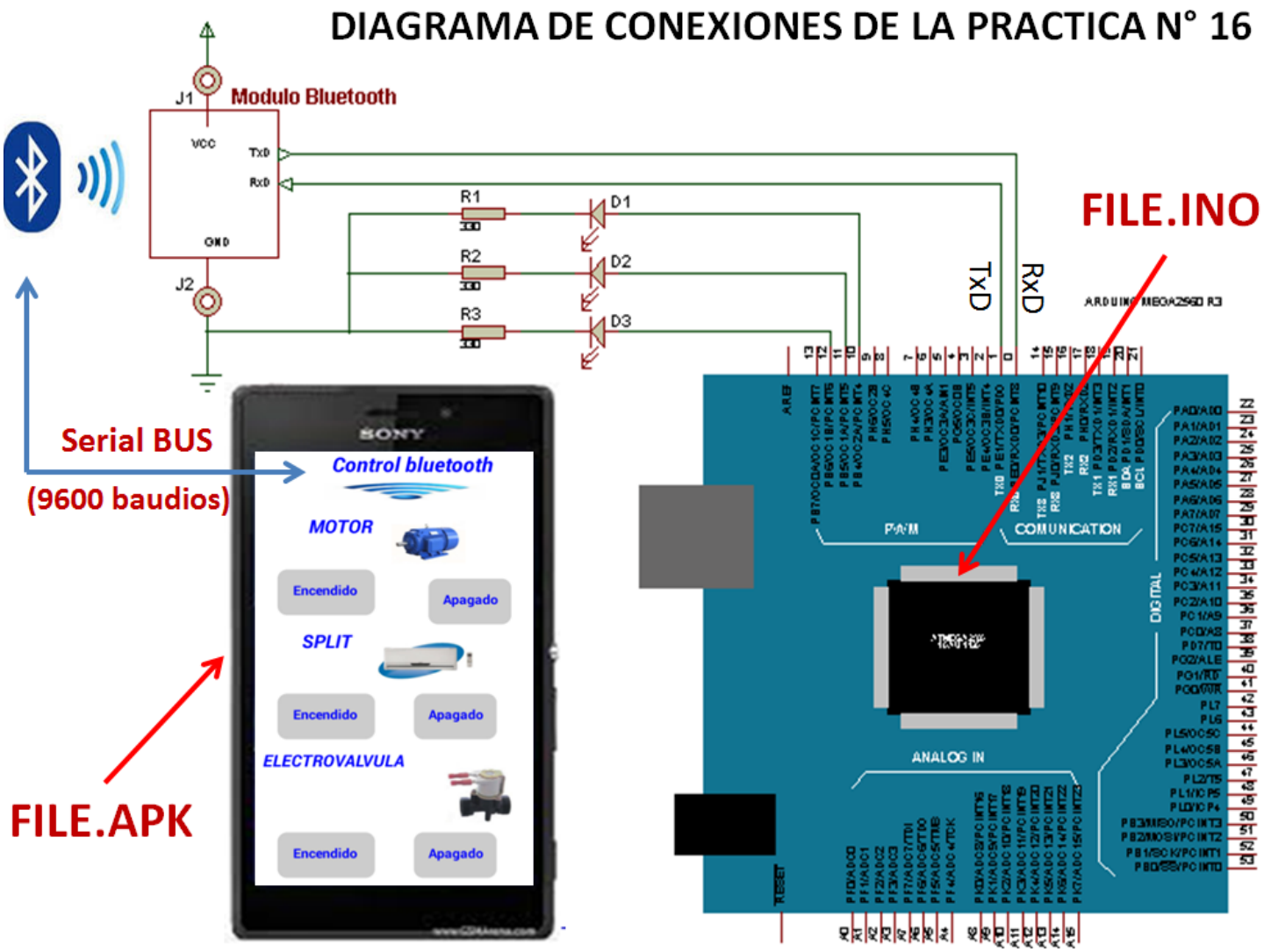
Agregue aquí .....

PRÁCTICA 16 – CONTROL POR BLUETOOTH

Esta práctica consiste en lograr la comunicación del Arduino vía Bluetooth con un Dispositivo Android y controlar el encendido y apagado de tres LED´s a distancia a través de una aplicación instalada en el Dispositivo Android (ej. un celular/tableta).

Material:

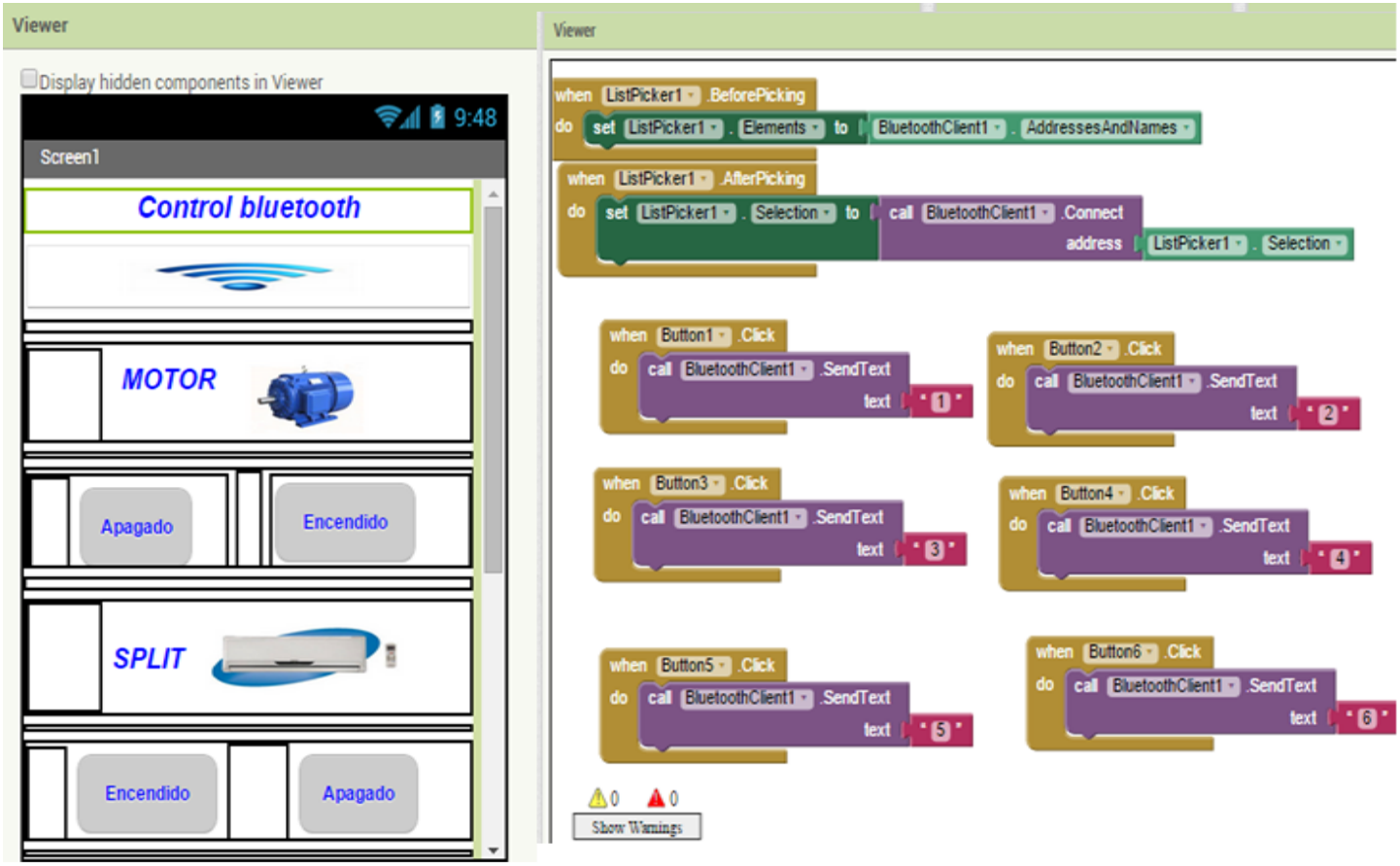
- **1 Protoboard**
- **1 modulo Bluetooth (HC06)**
- **1 celular**
- **3 resistencias de 330 ohms**
- **3 LED´s**
- **fuente de poder de 5 volts**
- **Cables**



FILE.INO de la Aplicación ON/OFF

```
int led12=12;
int led11=11;
int led10=10;
int estado=0;

void setup(){
  Serial.begin(9600);
  pinMode(led12,OUTPUT);
  pinMode(led11,OUTPUT);
  pinMode(led10,OUTPUT);
}

void loop(){
 if(Serial.available()>0){
 estado = Serial.read();
 }
 if (estado =='1'){
   digitalWrite(led12,HIGH);
  }
if(estado=='2'){
   digitalWrite(led12,LOW);
  }
 if (estado =='3'){
   digitalWrite(led11,HIGH);
  }
if(estado=='4'){
   digitalWrite(led11,LOW);
  }
   if (estado =='5'){
   digitalWrite(led10,HIGH);
  }
if(estado=='6'){
   digitalWrite(led10,LOW);
  }
}
```

FILE.APK de la Aplicación ON/OFF



CONCLUSIONES PERSONALES DE LA PRACTICA N° 16

Agregue aquí .....

ANEXO 1

¿Cuál Arduino comprar?

Si te has dado cuenta, existe una gran variedad de tarjetas Arduino para adquirir y tomar la decisión de cuál de ellas emplear en tus proyectos es sumamente importante, razón por la que se ha incorporado este anexo, el cual ofrece una breve explicación de las capacidades de cada modelo de Arduino para que puedas tomar una mejor decisión según tus necesidades.

IMPORTANTE: Todas las tarjetas Arduino son programables con el Arduino IDE (Integrated Development Environment), el cual es un software gratuito que simplifica en gran medida la tarea de programar el Microcontrolador.

1. Modelos de Arduino

Actualmente existen en el mercado una gran variedad de modelos así que primero que nada los abordaremos individualmente mostrando sus características principales:

1.1 Uno

Podríamos llamarlo el caballito de batalla, tiene 14 Entradas/Salidas digitales (6 de las cuáles pueden ser usadas como salidas PWM), 6 entradas analógicas, velocidad de reloj de 16MHz, conexión USB, Jack de alimentación y un botón de reset.

Contiene todos los componentes necesarios para que el Microcontrolador funcione correctamente; Simplemente conéctalo a una computadora con un cable USB o aliméntalo con el adaptador AC-DC o una bacteria para comenzar. El Arduino Uno es el modelo de referencia para la plataforma Arduino y es compatible con la gran mayoría de los shields (placas) existentes.

1.2 Pro

El Pro viene tanto en 3.3V/8MHz y 5V/16MHz. Tiene la misma cantidad de pines y periféricos que el Arduino UNO, pues está basado en el mismo Microcontrolador (ATmega328) pero carece de convertidor Serial-USB por lo que requiere de un cable FTDI para programarse, lo que lo hace una opción más barata que el Arduino UNO si piensas utilizar varias tarjetas a la vez (Un solo cable FTDI podrá programar todas las tarjetas).

Está diseñado para instalaciones semi-permanentes en objetos o exhibiciones. La tarjeta viene sin headers pre-soldados para permitir el uso de varios tipos de conectores o directamente cables/alambres soldados.

1.3 Pro Mini

Esta tarjeta es la versión miniatura del Arduino Pro, pensado para aplicaciones en las que el espacio sea muy reducido. Requiere un cable FTDI para programarse y puede encajar en un protoboard si se le soldan pines de espaciamiento de 0.1in.

1.4. Leonardo

Esta tarjeta tiene una forma muy similar a la tarjeta Uno, así que es compatible con todos los Shields (a excepción de algunos por incompatibilidad de ciertos pines).

La gran ventaja de esta tarjeta es que está basada en el procesador ATmega32u4, el cual tiene comunicación USB integrada, por lo que no requiere de un convertidor Serial-USB ni de un cable FTDI para programarse, además de ser un poco más económico que el Arduino Uno por requerir menos componentes. Gracias a sus capacidades USB puede emular las funciones de un mouse y un teclado, permitiéndote dar clic, doble clic, scroll o escribir texto de una manera muy sencilla.

1.5 Micro

Esta tarjeta es la versión en miniatura del Arduino Leonardo, por lo que cuenta con sus mismas capacidades, tales como comunicación USB nativa y emulación de Mouse/Teclado, además de permitirte utilizarlo junto a un protoboard y reducir en gran medida el tamaño de tu circuito por su diminuto tamaño.

1.6 MEGA

Si necesitas poder, pero a su vez compatibilidad con shields y código para Arduino Uno, el Arduino Mega es para ti. Esta tarjeta cuenta con una cantidad mucho mayor de I/O que el Arduino Uno (54 vs 14 del Uno), además de tener 14 salidas PWM, 4 puertos UART, I2C y 16 entradas analógicas.

Además, tiene una memoria de mayor capacidad lo que te permitirá utilizarlo para códigos muy extensos o que requieran de una gran cantidad de variables.

1.7 MEGA ADK

Esta tarjeta es compatible pin con pin con el Arduino Mega, con la gran ventaja de incluir una interfaz Host USB, que te permite conectar tu Arduino a un teléfono con SO Android y comunicarte con él para acceder a la información de sus sensores o recibir órdenes de tu celular para controlar motores, LEDs, etc.

Con ésta tarjeta puedes incluso recargar tu teléfono celular si requiere menos de 750mA y tienes una fuente de suficiente capacidad de corriente.

1.8 Due

Esta tarjeta está basada en un procesador ARM-cortex de 32-bits a 84MHz, es la tarjeta Arduino con mayor capacidad y velocidad de procesamiento de ésta lista. Tiene un footprint similar al del Arduino Mega, pero tiene periféricos adicionales, como 2 convertidores DA y 2 pines de comunicación CAN.

Nota: A diferencia de otras tarjetas Arduino, esta tarjeta trabaja a 3.3V por lo que un voltaje mayor a 3.3v en sus pines de I/O puede dañar la tarjeta.

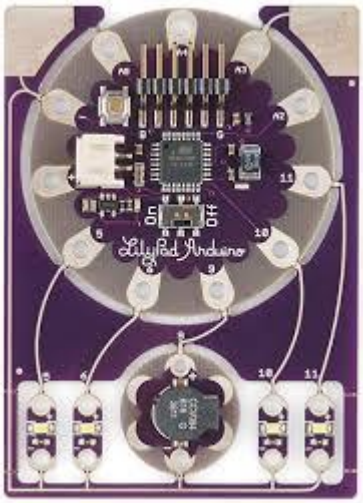
1.9 Ethernet

Ésta tarjeta está basada en el Microcontrolador ATmega328, al igual que el Arduino Uno, pero cuenta además con capacidad de conectarse a una red vía su puerto Ethernet. Cuenta además con un slot para tarjetas µSD por lo que podrás guardar una gran cantidad de información y utilizarla cuando lo requieras, aún después de haber reseteado la tarjeta.

Nota: Se requieren de los pines 10 – 13 para la comunicación Ethernet, por lo que no deberán usarse para otros propósitos y el número de pines digitales disponibles se reduce a 9, contando con 4 salidas PWM.

1.10 Lilypad

La tarjeta Lilypad está diseñada para ser utilizado en la ropa para crear e-textiles. Tiene pines especiales para coser sensores y actuadores utilizando un hilo conductivo.



1.11 Esplora

El Arduino Esplora está basado en el Leonardo y difiere de las anteriores tarjetas en que cuenta con un número de sensores integrados (de luz, temperatura, acelerómetro, joystick etc.) listos para usarse para interacción. Está diseñado para gente que quiere tomarlo y comenzar a programar su Arduino sin tener que aprender electrónica previamente.

Tabla comparativa de las Tarjetas Arduino

	Voltaje de Operación	Voltaje de Alimentación	Flash [KB]	SRAM [KB]	I/O digitales /PWM	Pines Analógicos I/O	UART	Compatibilidad Con Shields	Notas
Uno	5v	7-12v	32	2	14/6	6/0	1	Excelente	
Pro	5v	5-12v	32	2	14/6	6/0	1	Excelente	Requiere FTDI para programar
Pro Mini	5v	3.35-12v	32	2	14/6	6/0	1	N/A	
Leonardo	5v	7-12v	32	2.5	20/7	12/0	1	Decente (diferencias de pines)	USB nativo
Micro	5v	5v	32	2.5	20/7	12/0	1	N/A	Compatible con protoboards
Mega	5v	7-12v	256	8	54/15	16/0	4	Buena (Algunas diferencias de pines)	
Mega ADK	5v	7-12v	256	8	54/15	16/0	4	Buena (Algunas diferencias de pines)	Funciona con ADK (Android)
Due	3.3v	7-12v	512	96	54/12	12/2	4	Mala (Diferencias de pines y voltaje)	El procesador mas rápido
Ethernet	5v	7-12v	32	2	14/4	6/0	-	Muy buena (pocos conflictos con pines)	Requiere FTDI para programar
Lilypad	3.3v	2.7-5.5v	32	2	9/4	4/0	-	N/A	Pads para coserse
Esplora	5v	5v	32	2.5	N/A	N/A	-	N/A	Integración nativa con sensores

Fuente de la información: http://5hertz.com/tutoriales/?p=571#1.6QAC