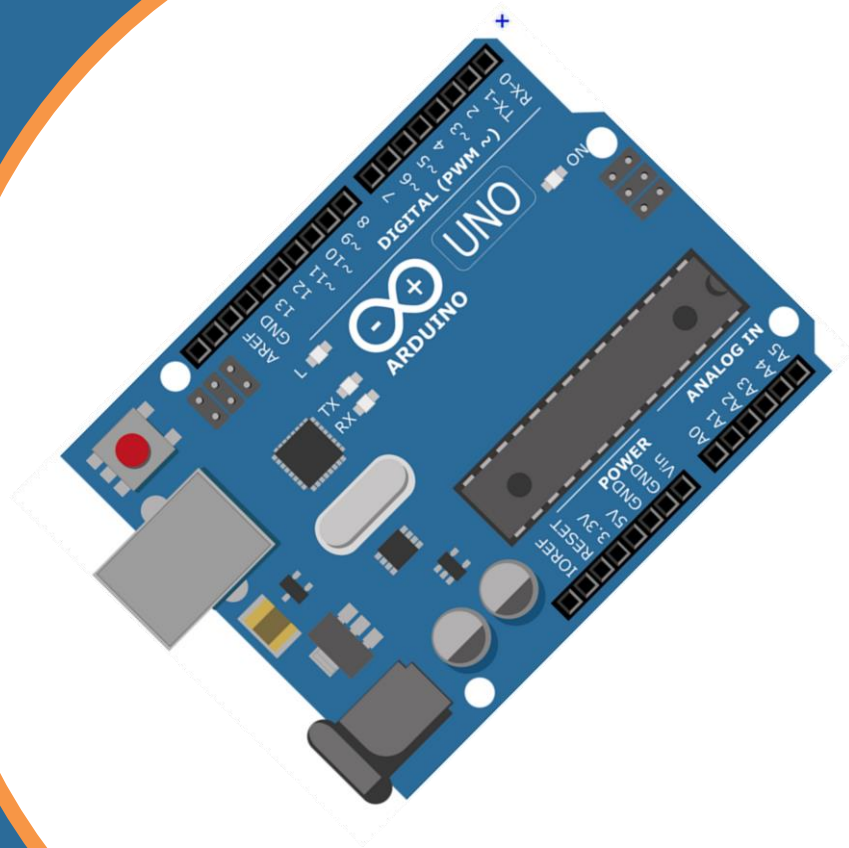


Mis proyectos con Arduino

José Miguel Castillo Castillo



Mis proyectos con Arduino

**Arduino Duemilanove Atmega 328P-PU.
2ª Versión. Actualizada y ampliada**

20/03/2024

No hay una forma mejor de aprender electrónica y a la misma vez aprender a programar un microcontrolador para desarrollar infinidad de aplicaciones electrónica con la placa de Arduino.

José Miguel Castillo Castillo

*A mi hijo Jaime y su madre Charo,
por todo lo que hacen por mi
y tenerme siempre feliz.*

Índice de contenido

1. INTRODUCCIÓN	(6)
1.1 Sistemas microprogramables	
1.2 Iniciación a un proyecto con microcontrolador	
2. CONOCIENDO ARDUINO.....	(9)
2.1 Placa Hardware PCB	
2.2 Software de Desarrollo	
2.3 Elementos de la Placa Arduino	
3. CARACTERÍSTICAS DE LA PLACA ARDUINO DUEMILANOVE.....	(14)
3.1. Análisis de la placa Arduino Duemilanove	
3.2. Alimentación	
3.3. Protección de sobrecarga de USB	
3.4. Memoria	
3.5. Entrada y Salida	
3.6. Comunicación	
3.7. Programación	
3.8. Reseteo por Hardware y Software	
4. GRABAR EL GESTOR DE ARRANQUE (BOOTLOADER).....	(25)
5. TRABAJAR DIRECTAMENTE CON LA PLACA PROTOBOARD.....	(30)
6. COMPONENTES HARDWARE PARA CONECTAR EN ARDUINO.....	(32)
7. ENTORNO DE DESARROLLO DE ARDUINO.....	(46)
7.1. Menú Archivo	
7.2. Menú Editar	
7.3. Menú Sketch	
7.4. Menú Herramientas	
8. LENGUAJE DE PROGRAMACIÓN DE ARDUINO.....	(54)
8.1. Estructura básica de un programa	
8.2. Funciones	
8.3. Variables	
8.4. Constantes	
8.5. Tipos de Datos	
8.6. Funciones de E/S Digitales	
8.7. Funciones de E/S Analógicas	
8.8. Función map()	
8.9. Funciones de Tiempo y Matemáticas	
8.10. Sentencias condicionales (Control de Flujo)	
8.11. Crear nuestras propias funciones	
8.12. Operadores Aritméticos	
8.13. Operadores Aleatorios	
8.14. Función de Interrupción	
8.15. Las Bibliotecas	
8.16. Comunicación Serial	
9. COMENZAR A TRABAJAR CON ARDUINO.....	(84)
9.1. Una programación simbólica: Blink	
9.2. Programaciones y circuitos eléctricos	

Índice de contenido

10. SALIDA Y ENTRADA DIGITAL.....	(91)
11. ENTRADA ANALOGICA.....	(93)
12. SALIDA ANALOGICA DEL TIPO PWM.....	(95)
13. ENTRADA ANALOGICA CON POTENCIOMETRO.....	(97)
14. AUMENTAR Y DISMINUIR LA LUMINOSIDAD DE UN LED.....	(100)
15. AUMENTAR LA LUMINOSIDAD DE UN LED CON PULSADOR.....	(101)
16. SECUENCIA DE LEDS.....	(102)
17. SECUENCIA DE LEDS CON PULSADOR.....	(105)
18. LUMINOSIDAD DE UN LED EN FUNCION DE LA LUZ EXTERIOR.....	(108)
19. CONTROL DE LUMINOSIDAD MEDIANTE FOTOCELULA LDR.....	(112)
20. CONTROL DE LAS LUCES DEL JARDIN MEDIANTE SENSOR LDR.....	(115)
21. ENCENDIDO ALEATORIO DE UN LED BIPOLAR.....	(117)
22. TERMOSTATO.....	(118)
23. TERMOSTATO CON VELOCIDAD DEL MOTOR VARIABLE.....	(120)
24. MEDIDOR Y CONTROL DE TEMPERATURA MEDIANTE NTC.....	(122)
25. DETECTOR DE ALARMA MEDIANTE ULTRASONIDOS HC-SR04.....	(126)
26. AVISO ACUSTICO PARA APARCAMIENTO.....	(130)
27. CONTROL DE AFORO DE UN LOCAL.....	(132)
28. CONTROL DE SERVOMOTORES.....	(135)
29. SALIDA DE ALTA CORRIENTE DE CONSUMO.....	(140)
30. PROGRAMACION DE UN CONTADOR.....	(142)
31. CRUCE REGULADO POR SEMAFOROS.....	(144)
32. SEMAFORO PARA PASO DE PEATONES.....	(147)
33. DADO ELECTRONICO.....	(151)
34. DETECTOR DE MOVIMIENTOS CON ACTIVACION DE LUZ.....	(155)
35. INTERRUPTOR POR CONTROL REMOTO POR INFRARROJO.....	(157)

1. Introducción

Con la aparición de las placas de Arduino, los usuarios expertos y no tan expertos, han visto una oportunidad para explotar su creatividad y realizar proyectos que antes no podían afrontar de una forma tan sencilla, pues se necesitaban de muchos recursos que solamente se lo podían permitir los fabricantes de equipos Hardware en sus departamentos de I+D y Producción.

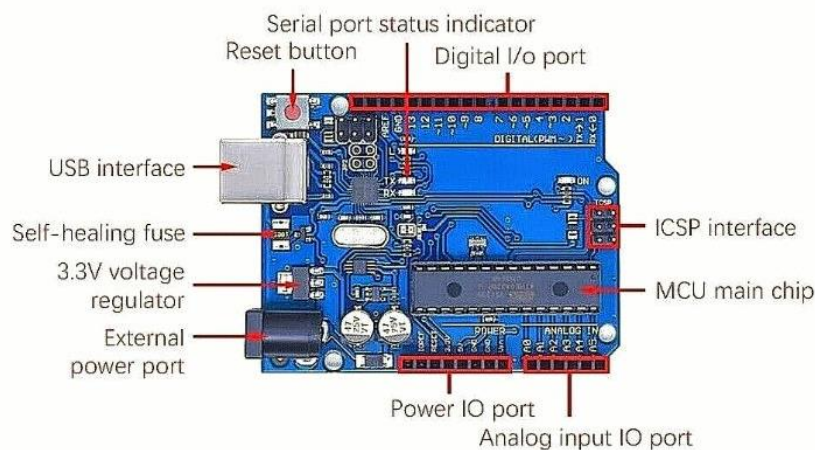
Arduino se ha proclamado como una de las plataformas escogidas para llevar a cabo proyectos tecnológicos. Esto es así debido a su versatilidad como instrumento para este cometido, por la gran cantidad de sensores que puede incorporar y por los precios que se están alcanzando, o que permite hoy en día a los usuarios tenerlo al alcance de la mano y del bolsillo, destacando también, la facilidad con la que se puede programar e interactuar con el medio que nos rodea.

En el mercado podemos encontrar una gran variedad de placas de Arduino; todas ellas ofrecen características diferentes unas de otras. Estas características se pueden apreciar visualmente, es decir, encontramos diferencias de tamaños y formas, pero también se diferencian por potencia eléctrica, de cálculo o soluciones software como, por ejemplo, diferencias en la resolución de lecturas analógicas o número de interrupciones.

Pero las diferencias vienen por parte del microcontrolador en concreto, más que por el diseño de la placa en sí. Ya que el hardware de Arduino es un elemento que incorpora los componentes necesarios que nos permiten trabajar con un determinado microcontrolador e incorporarlo a nuestros proyectos. Puesto que también, es idóneo para minimizar el número de componentes y, que a su vez pueda ser modificable su funcionamiento por software en futuras versiones.

Un microcontrolador, como veremos más adelante, es un circuito integrado constituido, internamente, por la CPU, Memoria, Entrada/Salida y Oscilador, todo esto encapsulado en un mismo chip de material semiconductor. Principalmente la memoria es del tipo flash que nos permita reprogramar, cargar una o varias veces el programa software, para que se ejecute las salidas y entradas de nuestro microcontrolador según el programa establecido y se mantengan los datos de programación incluso desconectando la tensión de alimentación.

Pues bien, en este caso nos vamos a centrar en el empleo y desarrollo de la placa de **Arduino Duemilanove** y especialmente en el microcontrolador Atmega328P-PU. Podemos acceder al siguiente enlace para obtener más información sobre la estructura y características del microcontrolador. [ATMEL-ATMEGA-328P](#)



Placa Arduino para proyectos

1. Introducción

1.1. Sistemas Microprogramables

Desde el primer momento en que se desea crear proyectos en los que se pretende que la solución a un problema sea algo automatizado y controlado electrónicamente estamos haciendo referencia a lo que se denomina **sistemas microprogramables**.

Un sistema microprogramable será todo sistema mediante una electrónica digital encapsulada en uno o varios circuitos integrados, con un generador de pulsos de alta velocidad que sea totalmente capaz de seguir una secuencia de instrucciones contenidas en un programa de forma rápida y eficaz.

Un sistema microprogramable consta de 5 subsistemas:

1. **Oscilador o generador de pulsos** (conocido normalmente por reloj). Genera los pulsos necesarios para que el sistema vaya perfectamente sincronizado. Por cada pulso de reloj se ejecutan una o varias instrucciones (según las características de la CPU) en el bloque CPU.
2. **Unidad Central de Proceso (CPU)**. Se encarga de ejecutar las instrucciones de los programas, así como realizar las operaciones aritmético-lógicas que se requiera durante la ejecución de dichos programas.
3. **Unidad de memoria**. Esta memoria almacenará los programas que se van a ejecutar y los resultados derivados de dicha ejecución.
4. **Bloque de Entrada y Salida**. Este bloque se encarga de gobernar el flujo de datos que existe entre el exterior y el interior del sistema. En el exterior contamos con los periféricos, que son dispositivos que introducen información al sistema.
5. **Periféricos**. Pueden ser otros dispositivos microprogramables o simplemente circuitos digitales que permiten al usuario interactuar con el sistema.

1.2. Iniciación a un proyecto con microcontrolador

Introduciéndonos un poco sobre los inicio de un proyecto, hay que añadir que, lo primero que debemos tener en cuenta a la hora de hacer un proyecto con un microcontrolador es de recopilar toda la información que podamos conseguir y anotándola con esquemas, organigramas, croquis, etc., para que se nos sea más fácil en el transcurso y desarrollo del proyecto.

Es primordial saber algunos aspectos de lo que queremos hacer, es decir, tener claro de qué información y datos necesitamos y cual disponemos, dónde la podemos conseguir, etc.

Al principio, estos puntos nos serán útiles para tener una base con la información general de nuestro proyecto. Por ejemplo, tenemos que recopilar datos de los componentes electrónicos que vayamos a utilizar en nuestro proyecto, entre ellos, sus características técnicas: alimentación, corriente de consumo, compatibilidad TTL/CMOS, polarización, etc.

Toda la información y datos que recopilemos la utilizaremos posteriormente para integrarlo en nuestro diseño y confeccionar la programación del microcontrolador y así obtener los resultados deseados.

1. Introducción

Podemos recopilar información sobre:

1. Qué componentes nos hace falta... un relé, una fotocélula, un display, una lámpara, un potenciómetro, un optoacoplador, un A.O., etc.
2. Obtener información de los componentes electrónicos que vamos a utilizar mediante sus hojas de datos, por ejemplo, en <https://www.alldatasheet.es>
3. Diferenciar los elementos de entrada digital/analógica con los de salida digital/analógica.
4. Dar nombres a cada uno de los elementos de E/S digital y analógica: LDR_01, L_01, PO_01...
5. Dar nombre a las diferentes variables... var01, var02, value...
6. Especificar por escrito la relación de pines utilizados por Software y Hardware.
7. Hacer un borrador con los datos recopilados para tenerlos presentes en el transcurso del proyecto.
8. Hacer un esquema u ordinograma que nos pueda servir de guía.
9. Etc...

Según los datos que estemos manejando y los sensores o componentes que proporcionan dichos datos, deberemos saber cuándo utilizar una entrada analógica y cuándo utilizar una entrada digital. Por ejemplo, para controlar un pequeño motor, nuestras entradas y salidas deberán ser analógicas; en cambio, para controlar el encendido y apagado de un diodo Led, deberemos utilizar las E/S digitales, y las E/S PWM para controlar la intensidad luminosa de un diodo Led.

Como se ha comentado al principio de esta introducción, no hay mejor forma de aprender electrónica y programación que desarrollar proyectos con microcontrolador donde se intervienen dos partes fundamentales:

1. El **Hardware**. Que son los componentes físicos que se utilizan en el proyecto: Microcontrolador, resistencias, Leds, condensadores, interruptores, sensores, potenciómetros, relés, tiristores, reguladores, pulsadores, triacs, optoacopladores, etc.
2. El **Software**. Es la parte de programación del microcontrolador. Se basa en instrucciones y códigos en lenguaje C. Se programa las salidas y entradas del microcontrolador para obtener el funcionamiento de los componentes electrónicos exteriormente conectados al microcontrolador (Hardware).

La depuración de un proyecto conlleva unas series de pruebas. No siempre se obtiene en la primera compilación los resultados deseados y, para ello, habrá que ir modificando o cambiando algún que otro código del programa o algún componente electrónico que no polariza adecuadamente el circuito hasta conseguir su valor y finalmente los resultados sean los deseados.

Es más, cuando tengamos ya logrado nuestro proyecto y funcionando 100% debemos volver a montarlo en otra placa con nuevos componentes, pero... ¡ojo!, con los mismos valores y características que hemos obtenidos en nuestro diseño, esto nos verificará que si funciona correctamente, nuestro proyecto está bien conseguido.

2. Conociendo Arduino

Fundamentalmente Arduino es una plataforma electrónica abierta para la creación de prototipos y que gira entorno a un microcontrolador. Esta plataforma posee una arquitectura hardware guiada por un programa o software que le va a permitir ejecutar programas previamente diseñados. Este recurso abierto significa que puede ser usado, distribuido, modificado, copiado, etc. gratuitamente.

La placa hardware de Arduino incorpora un microcontrolador reprogramable y una serie de pines-hembra, los cuales están unidos internamente a las patillas de E/S del microcontrolador que permiten conectar allí de forma muy sencilla y cómoda diferentes sensores y actuadores.



2.1. Placa Hardware PCB

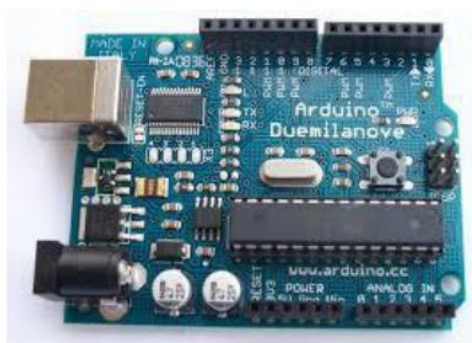
Cuando hablamos de la placa hardware nos estamos refiriendo en concreto a una PCB (del inglés *Printed Circuit Board*, o sea, placa de circuito impreso). Las PCBs son superficies fabricadas de un material no conductor (normalmente resinas de fibra de vidrio reforzada, cerámica o plástico) sobre las cuales aparecen laminadas (“pegadas”) pistas de material conductor (normalmente cobre). Las **PCBs** se utilizan para conectar eléctricamente, a través de pistas conductoras, diferentes componentes electrónicos soldados a ella. Una PCB es la forma más compacta y estable de construir un circuito electrónico (en contraposición a una breadboard, perfboard o similar) pero, al contrario que estas, una vez fabricada, su diseño es bastante difícil de modificar. Así pues, la placa Arduino no es más que una PCB que implementa un determinado diseño de circuitería interna.

No obstante, cuando hablamos de placa Arduino, deberíamos especificar el modelo concreto, ya que existen varios tipos de placas Arduino oficiales, cada una con diferentes características (como el tamaño físico, el número de pines-hembra ofrecidos, el modelo de microcontrolador incorporado y, como consecuencia, entre otras cosas, la cantidad de memoria utilizable, etc.). Conviene conocer estas características para identificar qué placa Arduino es la que nos convendrá más en cada proyecto.



2. Conociendo Arduino

Por ejemplo, en nuestro caso estudiaremos la placa Arduino **Duemilanove** Atmega-328P-PU que veremos en el siguiente capítulo con más detalle.



De todas formas, aunque puedan ser modelos específicos diferentes, tal como acabamos de comentar, los microcontroladores incorporados en las diferentes placas Arduino pertenecen todos a la misma “familia tecnológica”, por lo que su funcionamiento en realidad es bastante parecido entre sí. En concreto, todos los microcontroladores son de tipo **AVR**, una arquitectura de microcontroladores desarrollada y fabricada por la marca Atmel: <https://www.microchip.com/>



Podemos encontrar el microcontrolador Atmega328P en dos formatos: en formato **DIP**, que viene introducido en un zócalo y se puede extraer con relativa facilidad si utilizamos un extractor de circuitos integrados, o en formato **SMD**, que viene soldado a la placa de Arduino.

El diseño hardware de la placa Arduino está inspirado originalmente en otra placa de hardware libre preexistente, la placa **Wiring**.



Wiring es una plataforma de prototipado electrónico de fuente abierta compuesta de un lenguaje de programación, un entorno de desarrollo integrado (IDE), y un microcontrolador. Ha sido desarrollado desde 2003 como proyecto personal de Hernando Barragán, estudiante por aquel entonces del Instituto de Diseño de la Interacción Ivrea (lugar donde surgió en 2005 precisamente la placa Arduino).

2. Conociendo Arduino

2.2. Software de desarrollo

Un software (más en concreto, un “entorno de desarrollo”) gratis, libre y multiplataforma (ya que funciona en Linux, MacOS y Windows) que debemos instalar en nuestro ordenador y que nos permite escribir, verificar y guardar (“cargar”) en la memoria del microcontrolador de la placa Arduino el conjunto de instrucciones que deseamos que este empiece a ejecutar. Es decir: nos permite programarlo.

La manera estándar de conectar nuestro ordenador PC con la placa Arduino para poder enviarle y grabarle dichas instrucciones es mediante un simple cable USB, gracias a que la mayoría de placas Arduino incorporan un conector de este tipo. Los proyectos Arduino pueden ser autónomos o no.

En el primer caso, una vez programado el microcontrolador, la placa no necesita estar conectada a ningún ordenador y puede funcionar autónomamente si dispone de alguna fuente de alimentación externa.

En el segundo caso, la placa debe estar conectada de alguna forma permanente (por cable USB, por cable de red Ethernet, etc.) a un ordenador ejecutando algún software específico que permita la comunicación entre este y la placa y el intercambio de datos entre ambos dispositivos. Este software específico lo deberemos programar generalmente nosotros mismos mediante algún lenguaje de programación estándar como Python, C, Java, Php, etc., y será independiente completamente del entorno de desarrollo Arduino, el cual no se necesitará más, una vez que la placa ya haya sido programada y esté en funcionamiento.



USB tipo B USB tipo A (conexión PC).

Un lenguaje de programación libre. Por “lenguaje de programación” se entiende cualquier idioma artificial diseñado para expresar instrucciones (siguiendo unas determinadas reglas sintácticas) que pueden ser llevadas a cabo por máquinas. Concretamente dentro del lenguaje Arduino, encontramos elementos parecidos a muchos otros lenguajes de programaciones existentes (como los bloques condicionales, los bloques repetitivos, las variables, etc.), así como también diferentes comandos –asimismo llamados “órdenes” o “funciones” – que nos permiten especificar de una forma coherente y sin errores las instrucciones exactas que queremos programar en el microcontrolador de la placa. Estos comandos los escribimos mediante el entorno de desarrollo Arduino.

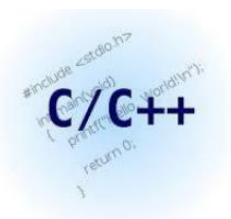
El programa se implementará haciendo uso del entorno de programación propio de Arduino y se transfiere empleando un cable USB. Si bien en el caso de la placa USB no es preciso utilizar una fuente de alimentación externa, ya que el propio cable USB la proporciona, para la realización de algunos de los experimentos un poco más complejo sí será necesario disponer de una fuente de alimentación externa de mayor corriente ya que la alimentación proporcionada por el USB puede no ser suficiente. El voltaje de la fuente puede estar comprendido entre 7 y 12 voltios.

2. Conociendo Arduino

Tanto el entorno de desarrollo como el lenguaje de programación Arduino están inspirado en otro entorno y lenguaje libre preexistente: Processing <https://www.processing.org>, desarrollado inicialmente por Ben Fry y Casey Reas.

Que el software Arduino se parezca tanto a **Processing** no es casualidad, ya que este está especializado en facilitar la generación de imágenes en tiempo real, de animaciones y de interacciones visuales, por lo que muchos profesores del Instituto de Diseño de Ivrea lo utilizaban en sus clases. Como fue en ese centro donde precisamente se inventó Arduino es natural que ambos entornos y lenguajes guarden bastante similitud.

No obstante, hay que aclarar que el lenguaje Processing está construido internamente con código escrito en lenguaje Java, mientras que el lenguaje Arduino se basa internamente en código **C/C++**



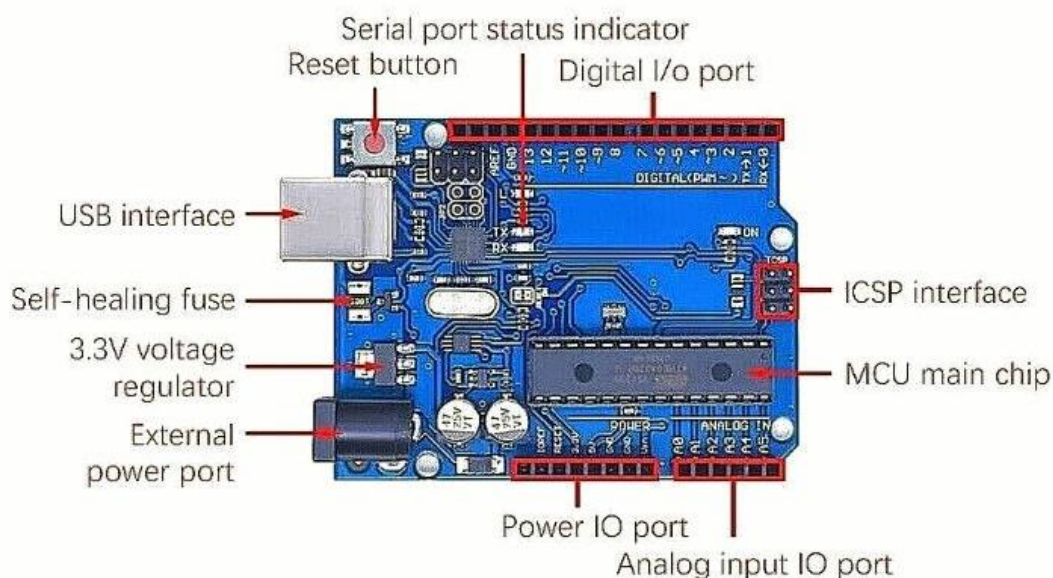
2.3. Elementos de la placa Arduino

La placa hardware de Arduino integra unas series de elementos específicos que la hace exclusiva y que a continuación se describen:

- ❖ **Puerto USB:** A través de este puerto podremos conectar el ordenador con la placa Arduino. Necesitaremos para ello un cable USB tipo AB (el de las impresoras) y a través de éste, podremos cargar en la placa los programas diseñados y también proporcionar alimentación eléctrica al dispositivo.
- ❖ **Chip integrador:** Actúa como puente y comunica el ordenador y el microcontrolador que veremos más adelante.
- ❖ **Power:** Podemos conectar una fuente de alimentación para que Arduino trabaje de manera autónoma sin necesidad de que esta placa esté conectada al ordenador mediante USB.
- ❖ **Chip - Cristal de cuarzo:** Actuaría como un reloj interno de Arduino. Da las pulsaciones de frecuencia necesarias para que la placa actúe de manera sincronizada y repetitiva. Cada vez que reiniciamos arduino, su contador comienza desde cero y esto puede ser útil en ciertos programas.
- ❖ **LEDs:** Hay tres: El "L" está vinculado al puerto 13 de Arduino y nos va a permitir hacer pruebas de funcionamiento de arduino sin necesidad de conectar otros dispositivos u otros LED. Los otros dos Leds son los LED Tx y Rx y éstos nos permiten conocer el estado de comunicación entre el ordenador y la placa.
- ❖ **Conexiones digitales:** Son pines de conexión rápida que funcionan como entrada y salida de datos digitales. Con ellos podremos remitir información del entorno a la placa (con el empleo de un sensor por ejemplo) o bien extraer respuestas desde la propia placa hacia otros periféricos conectados a Arduino (como luces LED, etc.).
- ❖ **LED ON:** Es una luz LED de color verde que nos permite conocer cuándo nuestro Arduino está conectado a la corriente (a través de baterías o a través del cable USB) y por lo tanto está encendido y en funcionamiento.

2. Conociendo Arduino

- ❖ **Microcontrolador:** Modelo Atmega328P. Va a ser el cerebro de la placa de Arduino. Tiene por defecto cargado un programa que es el gestor de arranque que le permite reiniciar el programa que tenga almacenado en memoria cada vez que lo conectamos a la red eléctrica.
- ❖ **Botón de reset:** Nos permite reiniciar la placa Arduino y por lo tanto también el programa que el microcontrolador tenga cargado y se encuentre ejecutando un bucle.
- ❖ **Entradas analógicas:** Hay seis entradas analógicas a través de estos pines de conexión rápida. Se emplean para la conexión a la placa Arduino de componentes que entreguen señal analógica como la de un potenciómetro.
- ❖ **Pines Power:** Esta barra de energía proporciona la energía suficiente para alimentar dispositivos externos y ajenos a la placa de Arduino pero que están conectados a ella (como una luz LED por ejemplo). Hay diferentes voltajes, tomas de tierra (GND) e incluso un pin con el que podremos resetear la placa Arduino a través de una señal eléctrica.
- ❖ **Reguladores:** La fuente de alimentación ofrece unos voltajes que se encontrarán entre 7 y 12 voltios (según la batería que estemos empleando). Los reguladores nos van a permitir reducir estos voltajes a 5 V que es con los que habitualmente trabaja la placa de Arduino.



Elementos que contiene la placa Arduino

Con Arduino se pueden realizar multitud de proyectos de rango muy variado: desde robótica hasta domótica, pasando por monitorización de sensores ambientales, sistemas de navegación, telemática, etc. Realmente, las posibilidades de esta plataforma para el desarrollo de productos electrónicos son prácticamente infinitas y tan solo están limitadas por nuestra imaginación.

<https://myelectronic.ueuo.com/arduino.html>

3. Características de la placa Arduino Duemilanove

Como se ha comentado anteriormente, el hardware de Arduino consiste en una placa con un microcontrolador Atmel **AVR** y puertos de entrada/salida. Posee un entorno de desarrollo **IDE** que implementa el lenguaje de procesamiento/cableado de código abierto que se puede descargar de forma gratuita (actualmente) para Mac OS X, Windows y Linux.

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

Aunque existen diferentes versiones de la placa de Arduino, en nuestro caso, nos centraremos en la placa Arduino **Duemilanove** con el microcontrolador **Atmega328P-PU**.

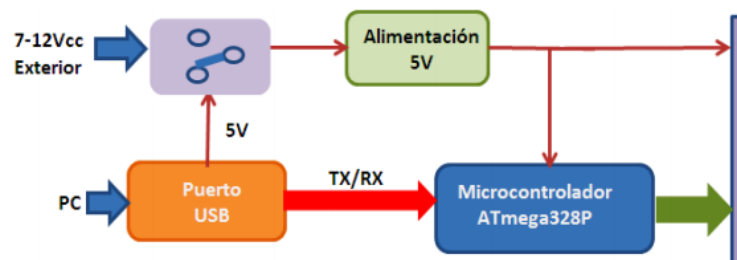


Esta placa Arduino Duemilanove es una placa de desarrollo que trabaja con el microcontrolador Atmega328P-PU de 8 bits; esto quiere decir que el microcontrolador puede gestionar instrucciones de una longitud de 8 bits, o lo que es lo mismo, de 1 byte.

La placa Arduino Duemilanove comprende tres partes principales:

1. **Alimentación.** La placa Arduino se puede seleccionar entre la tensión de 5 V que ofrece el puerto de comunicación USB o una alimentación externa comprendida entre 7 a 12 voltios, que se regula y estabiliza a 5 voltios. Con la tensión ofrecida por el puerto USB se podrá controlar dispositivos de poca potencia como Leds. Con protección de fusible USB y restablecimiento de software.
2. **Comunicación.** Un circuito integrado FT232RL con puerto USB que nos sirve de puente en la comunicación de la placa Arduino con el ordenador PC para el envío y recepción de datos Tx/Rx y ofreciendo 3,3 voltios y 5 voltios de alimentación.
3. **Microcontrolador.** ATmega-328P, de 8 bits. Que se compone de 28 pines DIP, 14 pines de entrada y salida digitales, 6 de ellos con salidas PWM y 6 pines de entradas analógicas.

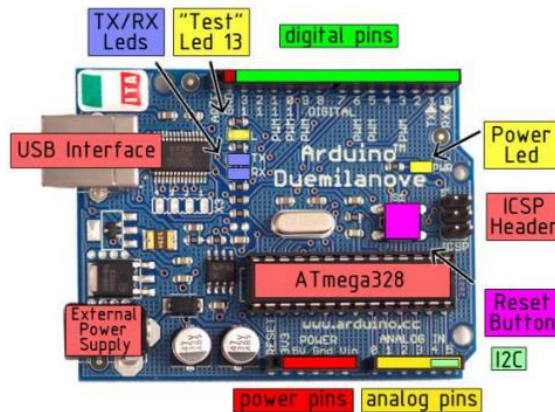
Todos estos pines están conectados a una regleta de pines, para la conexión de los componentes y circuitos prototipos.



3. Características de la placa Arduino Duemilanove

3.1. Análisis de la placa Arduino Duemilanove

La placa de Arduino Duemilanove Atmega328P se compone de unas series de elementos que a continuación describiremos.



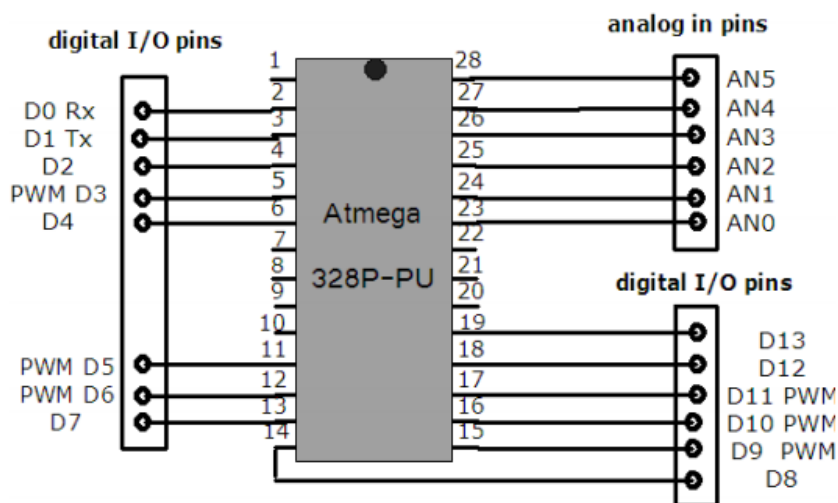
Pines de conexiones de E/S

La placa de Arduino Duemilanove posee unos pines para conexiones externas para conectar hilos rígidos de 0,5 mm de diámetro de diversos colores para que nuestro montaje sea fácilmente identificado con los componentes y circuitos exteriores: **digital pins**, **analog pins** y **power pins**:

- **Pines digitales** (*digital pins*). Son terminales que se emplean para comunicar la placa Arduino con el exterior, conectando sensores que proporcionan información digital (5 V o 0 V, 1 o 0). Se podrán configurar como entrada o salida. Arduino Duemilanove dispone de 14 pines de este tipo, de los cuales seis pines poseen salida analógica PWM.
- **Pines analógicos** (*analog pins*). Son terminales que se emplean para comunicar la placa Arduino con el exterior, conectando sensores que les proporcionan información analógica. Posee 6 pines de entradas analógicas de la **AN0** a **AN5** (**ANALOG IN**).
- **Pines de alimentación** (*power pins*). Estos terminales contiene los pines de salida de alimentación que permite dar servicio de tensión a nuestros circuitos y componentes conectados exteriormente. Posee cinco pines:
 1. **RESET**. Este pin tiene la misma función que el botón reset. Aquí lo encontramos en formato pin para poder resetear la placa mediante un pulsador externo y una resistencia de 10K a 5V.
 2. **3,3 V**. Este pin proporciona 3,3 voltios. Es posible que algún sensor o componente requiera de esa tensión para poder funcionar correctamente.
 3. **5 V**. Este pin proporciona 5 voltios para alimentar los dispositivos, sensores y/o componentes electrónicos conectados a Arduino.
 4. **GND**. Aquí se conectarán los terminales de masa de los componentes electrónicos que se hayan podido conectar al terminal de 5 o 3,3 voltios de Arduino.
 5. **Vin**. Este terminal permite alimentar a Arduino de la misma forma en que se realiza en el caso del conector de alimentación mediante un conector Jack de 9mm. Esta tensión debe estar comprendida entre 7 y 12 voltios máximo.

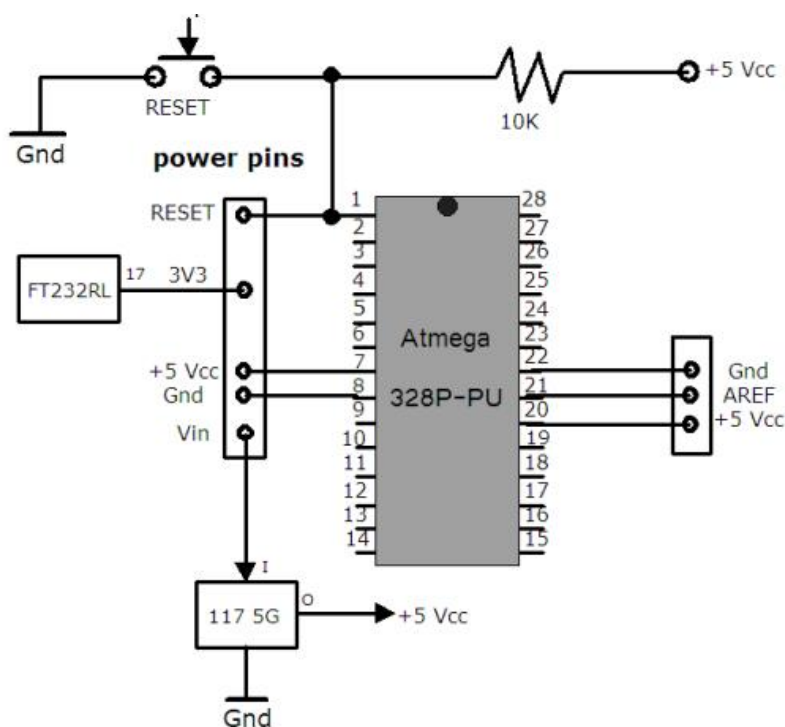
3. Características de la placa Arduino Duemilanove

El siguiente esquema muestra la correspondencia de conexión de los pines digitales y analógicos de la placa Arduino con el patillaje del microcontrolador Atmega328P.



NOTA: Es muy importante conocer la diferencia entre las señales analógicas y digitales. La señal analógica es aquella que presenta una variación continua con el tiempo, es decir, la información o la señal, para pasar desde un valor a otro pasa necesariamente por todos los valores intermedios. Es continua y puede tomar infinitos valores. Estas señales predominan en nuestro entorno (variaciones de temperatura, presión, velocidad, distancia, sonido etc.) y éstas pueden ser transformadas en señales eléctricas mediante un dispositivo denominado transductor. La señal digital es aquella que presenta una variación discontinua con el tiempo y sólo puede tomar ciertos valores discretos. Es decir, va a saltos entre uno y otro valor. La utilización de señales digitales para transmitir información puede ser de dos modos: En función del número de estados distintos que pueda tener: Binario, ternario... Y en función de su naturaleza eléctrica. Una señal binaria se puede representar como la variación de una amplitud respecto al tiempo.

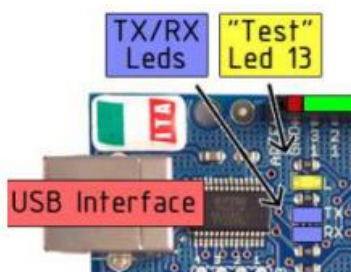
Esquema de conexionado de los pines de alimentación al microcontrolador Atmega328P.



3. Características de la placa Arduino Duemilanove

Indicador TX/RX Leds

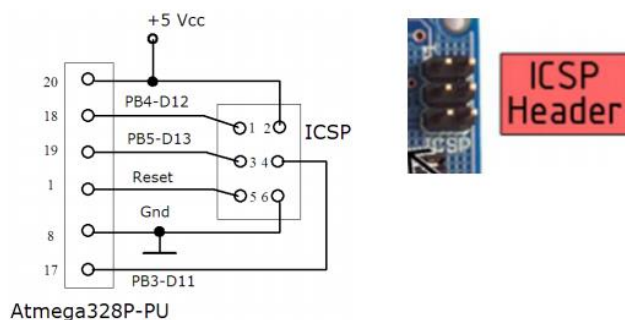
Indica que Arduino se está comunicando vía serie con el PC. Cuando esto ocurre, los indicadores parpadean, alertando de la transmisión y recepción de la información transmitida entre Arduino y el ordenador.



Los pines 2 y 3, **D0 (Rx)** y **D1 (Tx)**. Se encargan de recibir y transmitir los datos que le llegan desde el PC mediante el puerto serie USB señalizando las transmisiones mediante dos Leds TX/RX. Estos pines se encuentran disponibles en el conector digital pins con la idea de utilizarlo en la programación de un microcontrolador que se encuentra montado en otra placa.

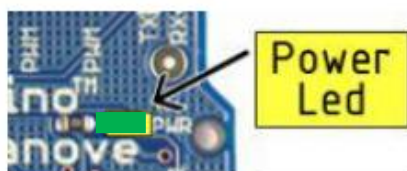
Conector ICSP

Se utilizan cuando se desea programar Arduino desde un entorno diferente del IDE y de la conexión típica por USB. Para realizar esta operación se requiere de un programador externo que irá conectado a los conectores mencionados. Si se desea programar Arduino de este modo, se deberá hacer en lenguaje Ensamblador o en lenguaje de alto nivel C.



Indicador de encendido

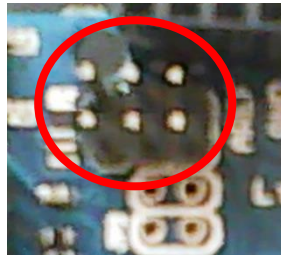
Mediante un pequeño Led verde indica que Arduino está alimentado correctamente y listo para programar. Cuando conectamos nuestra placa Arduino al puerto USB de nuestro ordenador, inmediatamente se enciende el **Power Led**, de color verde, que nos indica que la placa está conectada a la alimentación y es correcta. Al mismo tiempo el ordenador detecta un nuevo dispositivo y lo instala.



3. Características de la placa Arduino Duemilanove

El pin AREF en Arduino.

Es un conector dedicado a proporcionar el voltaje de referencia para los pines analógicos. Generalmente, esta referencia es de 0 a 5 voltios, pero podemos encontrarnos con componentes para Arduino que funcionan con otro rango de voltajes, por los que voltajes de referencia deberían ser ajustados.



Conexión alimentador externo 7v-12v

Mediante un Jack de 2,1, mm alimentaremos a la placa Arduino con un rango de tensión comprendido entre los 7 y los 12 voltios en continua.



Microcontrolador Atmega 328P -PU

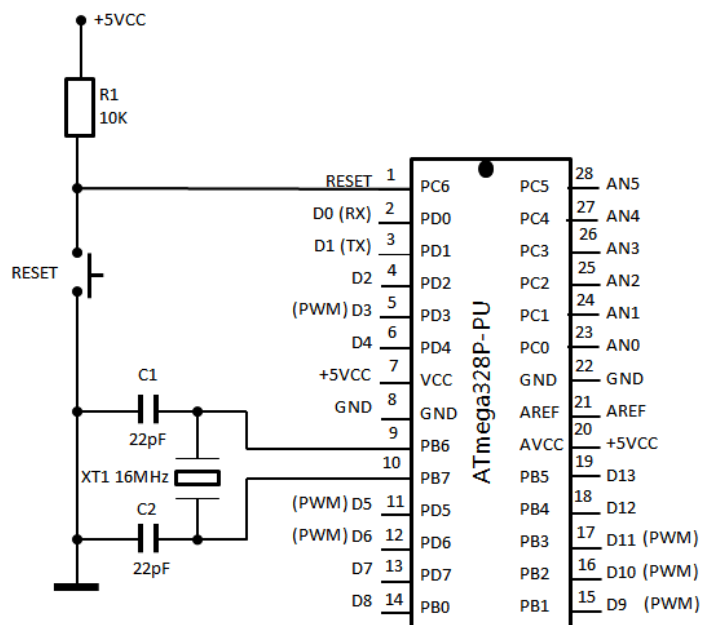
Este circuito integrado es el cerebro de la placa. Es el encargado de ejecutar las instrucciones de los programas creados por el usuario. En la siguiente tabla se detallan las características de este circuito integrado y su correspondiente esquema eléctrico.

Características principales:

- Microcontrolador ATmega328P-PU
- Voltaje de operación 5V
- Tensión de entrada(recomendada) 7-12 V
- Tensión de entrada (límite) 6-20 V
- Pines Digitales de E/S 14 (0-13, de los cuales 6 con salidas PWM)
- Pines de entrada analógicos 6 (0-5)
- Corriente DC por pin E/S 40 mA.
- Corriente DC para pin 3,3V 50 mA.
- Memoria Flash 32 KB (de los cuales 2 KB usados para bootloader)
- SRAM 2 KB
- EEPROM 1 KB
- Frecuencia de reloj 16 MHz.

[Hoja de datos del microcontrolador ATmega.](#)

3. Características de la placa Arduino Duemilanove



Microcontrolador ATmega328P-PU. Pines y componentes base.

Botón de reset

Permite realizar un reinicio a la placa. Una vez reseteada la placa, ésta vuelve a ejecutar el programa que tiene cargado.



Conector hembra USB

Se emplea para comunicar y alimentar la placa Arduino con el PC y la comunicación con Arduino a través del Monitor Serie.



Indicador de carga

Este indicador Led1 L, situado en el pin D13, parpadea cuando se carga un programa en la placa de Arduino.



3. Características de la placa Arduino Duemilanove

Este pin se encuentra en el conector de alimentación (*powerpins*) de la placa Arduino y además contiene los siguientes pines:

- **Vin.** Este pin permite alimentar a la placa Arduino de la misma forma en que se realiza en el caso del conector de alimentación mediante un conector Jack de 2,1mm.
- **5V.** Este pin proporciona 5 voltios para alimentar los dispositivos, sensores y/o componentes electrónicos conectados a Arduino. Este puede venir o desde **Vin** a través de un regulador en la placa, o ser suministrado por USB u otro suministro regulador de 5 voltios.
- **3V3.** Este pin proporciona 3,3 voltios. Es posible que algún sensor o componentes electrónicos requiera de esta tensión para poder funcionar correctamente. Es generado por el chip FTDI (FT232RL) en el pin 17 de la placa. La corriente máxima es de 50 mA.
- **GND.** Aquí se conectarán los terminales de masa de los componentes electrónicos que se hayan podido conectar al terminal de Vin, 5 o 3,3 voltios de Arduino.

3.3. Protección de sobrecarga del USB

El Arduino Duemilanove tiene un fusible reseteable que protege los puertos USB del ordenador de cortes y sobrecargas. Aunque la mayoría de los ordenadores proporcionan su propia protección interna, el fusible proporciona una capa de protección extra. Si más de 500 mA se aplican al puerto USB, el fusible automáticamente romperá la conexión hasta que el corte o la sobrecarga sean eliminados.

3.4. Memoria

El Atmega328P tiene 32KB de memoria Flash para almacenar códigos (de los cuales 2 KB se usa para el “**bootloader**”) y, vienen precargado con el gestor de arranque [bootloader](#). Tiene 2 K de SRAM y 1 KB de EEPROM (que puede ser leída y escrita con la librería EEPROM).

Los tipos de memorias que están integrado en el microcontrolador Atmega 328P son:

- **SRAM:** Variables locales, datos parciales. Es la encargada de almacenar los datos resultantes de la ejecución de las instrucciones de un programa. Usualmente se trata como banco de registros (PIC), Memoria volátil. Donde el sketch crea y manipula las variables cuando se ejecuta. Es un recurso limitado y debemos supervisar su uso para evitar agotarlo.
- **EEPROM:** Se puede grabar desde el programa del microcontrolador. Usualmente, constantes de programa. Memoria no volátil para mantener datos después de un reset. En ella van grabadas las librerías necesarias para interpretar los programas de Arduino. Las EEPROMs tienen un número limitado de lecturas/escrituras, tener en cuenta a la hora de usarla.
- **FLASH:** Memoria de programa. Usualmente desde 1 Kb a 4 Mb (controladores de familias grandes). Es donde se almacena los programas que los usuarios cargamos mediante el IDE de Arduino. Esta memoria está compartida con un gestor de arranque, que incorpora las instrucciones necesarias para que Arduino esté listo para poder trabajar con él. La cantidad compartida es de 0,5KB.

[Guía de Memoria de Arduino](#)

3. Características de la placa Arduino Duemilanove

3.5. Entrada y Salida

Cada uno de los 14 pines digitales del Arduino Duemilanove puede ser usado como entrada o salida, usando funciones **pinMode()**, **digitalWrite()** y **digitalRead()**. Opera a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia interna “pull-up” (desconectada por defecto) de 20 – 50K Ω .

Algunos pines tienen funciones principales:

- **Serial: 0 (Rx) y 1 (Tx).** Usados para recibir (Rx) y transmitir (Tx) datos TTL, en serie. Estos pines están conectados a los pines correspondientes del chip FTDI USB-a-TTL Serie.
- **Interruptores externos:** Los pines D2 y D3, 0 y 1, pueden ser configurados para disparar un interruptor en un valor bajo, un margen creciente o decreciente, o un cambio de valor. Su función es **attachInterrupt()**.
- **PWM:** 3, 5, 6, 9, 10 y 11. Proporcionan salida PWM de 8 bits con la función **analogWrite()**
- **LED: 13.** Hay un LED instalado en la placa Arduino conectado al pin D13. Cuando el pin 13 está a valor HIGH, el LED está encendido, cuando el pin está a LOW, está apagado.
- **Reset.** Pone esta línea a **LOW** para resetear el microcontrolador. Típicamente usada para añadir un botón de reset a dispositivo que bloquean a la placa principal.
- **Entradas analógicas.** Arduino Duemilanove tiene 6 entradas analógicas **AN0** a **AN5 (analogpins)**.

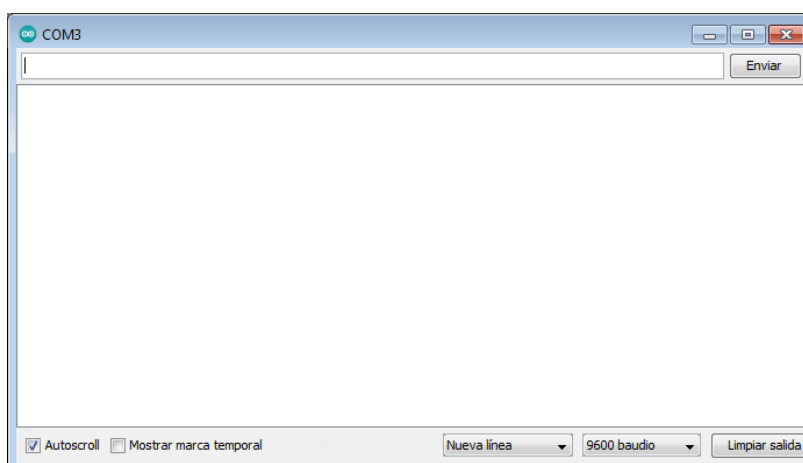
[Instrucciones de Lenguaje de Arduino](#)

3.6. Comunicación

La comunicación del microcontrolador con el PC se realiza mediante comunicación serie; esto quiere decir que los bits llegan a Arduino de uno en uno, y el microcontrolador trabaja con grupos de 8 bits, por lo que se dispone de otro circuito integrado soldado a la placa llamado UART (*Transmisor – Receptor Asíncrono Universal*), y es el encargado de gestionar los bits y adecuarlos según se necesitan en serie o en grupos de 8 bits para adaptarlo al microcontrolador.

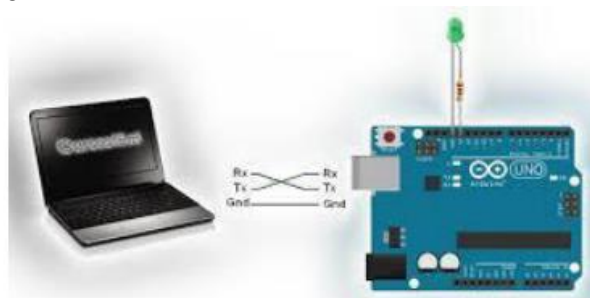
El **UART TTL (5V)**, un FTDI FT232RL, soldado en la placa de Arduino, canaliza esta comunicación serie al USB y los drivers **FTDI** (incluidos con el software Arduino) proporcionan un puerto de comunicación virtual al software del ordenador.

El software Arduino incluye un monitor serie que permite enviar y recibir datos de texto desde la placa Arduino al PC y viceversa.



3. Características de la placa Arduino Duemilanove

Una librería [Software Serial](#) permite comunicación serie en cualquiera de los pines digitales del Duemilanove.



3.7. Programación

El Arduino Duemilanove puede ser programado con el software **Arduino IDE**. Es un interfaz para programar de una forma sencilla y dinámica la plataforma de Arduino. Se escribe dentro del entorno de desarrollo de Arduino, se verifica, compila y lo carga en la memoria flash del microcontrolador **Atmega328P-PU**. Este entorno (IDE) se descarga de la página oficial de Arduino según el sistema operativo.

<https://www.arduino.cc/en/Main/Software>

```
contador Arduino 1.8.16
Archivo Editar Programa Herramientas Ayuda
contador
/* Programa Contador */

int LED=8;           //asignamos el pin 13 digital a LED
int boton=7;        //asignamos el pin 7 digital a boton
int valor=0;        //asignamos la variable valor a 0
int contador=0;     //asignamos la variable contador a 0
int estadoboton=0;  //asignamos la variable estadoboton a 0

void setup() {
  Serial.begin(9600); // configure velocidad de transmisión a 9600
  pinMode(LED, OUTPUT); // pone de salida digital el LED verde el pin 8
  pinMode(boton, INPUT); // pone de entrada digital el botón pin 7
}
void loop()
{
  valor=digitalRead(boton); // lee el valor de la entrada digital p
  digitalWrite(LED, valor); // enciende el LED si el nivel es alto
  if (valor!=estadoboton) // condiciona igualdad estado variable
  {
    //valor = 1; // enciende el LED si el nivel es alto
  }
}
```

El ATmega328P-PU de Arduino Duemilanove viene con un **bootloader** (*gestor de arranque*) pregrabado que nos permite subir nuevo código sin usar un programador hardware externo.

Los microcontroladores se programan generalmente a través de un programador a menos que, en nuestro caso, tengamos un gestor de arranque o zona de memoria de firmware en nuestro microcontrolador que permita instalar o desinstalar un programa sin la necesidad de un programador externo. Esto se llama un **gestor de arranque**. Se comunica usando el protocolo original STK500.

<https://www.arduino.cc/en/Tutorial/Bootloader>

3. Características de la placa Arduino Duemilanove

También puedes saltar el **bootloader** y programar el ATmega328P a través de la cabecera **ICSP** (*In-Circuit Serial Programming*). <https://www.arduino.cc/en/Hacking/Programmer>

3.8. Reseteo por Hardware y Software

En la placa Arduino posee un botón de reset que nos permite en cualquier momento, que se esté ejecutando el programa, de detenerlo y comenzar de nuevo la ejecución del programa. Esto es así, porque estamos aplicando, cuando pulsamos el botón de reset, una señal negativa GND a la patilla 1 del microcontrolador Atmega328P. El microcontrolador inmediatamente detiene el programa y comienza desde el principio, quedando la patilla 1 del microcontrolador a nivel alto mediante una resistencia de 10K a VCC.

En lugar de requerir una pulsación física del botón de reset antes de una subida, el Arduino Duemilanove está diseñado de forma que permite ser reseteado por software en ejecución en un ordenador conectado.

Una de las líneas de control de flujo de hardware (**DTR**) del **FT232RL** está conectada a la línea de reset del ATmega328P a través de un condensador de 100 nF. Cuando esta línea toma el valor LOW, la línea reset se mantiene el tiempo suficiente para resetear el chip. La versión 0009 del software Arduino usa esta capacidad para permitir cargar código simplemente presionando el botón de **upLoad** (*Cargar*) en el entorno Arduino. Esto significa que el **bootloader** (*gestor de arranque*) puede tener un tiempo de espera más corto, mientras la bajada del DTR puede ser coordinada correctamente con el comienzo de la subida.

Esta configuración tiene otras repercusiones. Cuando el Duemilanove está conectado a un ordenador que ejecuta Mac OS X o Linux, se resetea cada vez que hace una conexión a él por software (a través de USB). Durante el siguiente medio segundo aproximadamente, el **bootloader** se ejecutará en el Duemilanove. Mientras esté programado, para ignorar datos "malformados" (por ejemplo, cualquiera excepto una subida de código nuevo), interceptará los primeros bytes de datos enviados a la placa después de abrir la conexión. Si una rutina que se ejecuta en la placa recibe una configuración una vez u otros datos cuando empieza, asegurarse de que el software con el que se comunica espera un segundo después de abrir la conexión y antes de enviar estos datos.

Para resetear por software desde el mismo programa que está ejecutando el microcontrolador de Arduino se puede utilizar la siguiente función:

```
void(* resetFunc) (void) = 0; // esta es la funcion
resetFunc();                 // llamada
```

Con las siguientes instrucciones lo que hace es saltar por software otra vez a dicha dirección, pero no pasa por el estado en que todas las entradas I/O del micro pasan por estado inicial del micro.

```
unsigned int tstart;
void setup() {
  Serial.begin(9600);
  Serial.println("EMPEZANDO....");
  tstart=millis();
}
```


4. Grabar el Gestor de Arranque (Bootloader)

La **bootloader** de Arduino es un software alojado en la memoria flash del ATmega 328P que nos permite reprogramar Arduino a través del puerto serie sin necesidad de usar un programador externo.

Cuando adquirimos un nuevo microcontrolador ATmega que viene de fábrica y queremos incorporarlo al hardware del Arduino es necesario cargar la **bootloader**. Para ello es necesario el **IDE** y una placa de Arduino con un microcontrolador que funcione correctamente.

La **bootloader** de Arduino es una de las partes esenciales en las que reside la comodidad y sencillez de uso de Arduino. En general lo normal es que no necesitemos lidiar con él. Sin embargo, hay varias circunstancias en las que necesitaremos ser capaces de modificar el bootloader de Arduino.

Por ejemplo:

- Los usuarios avanzados pueden desear modificar y personalizar el proceso de arranque.
- En proyectos grandes, podemos querer aprovechar el espacio ocupado por el bootloader.
- Algunos fabricantes envían sus placas sin el bootloader precargado.
- En alguna circunstancia el bootloader puede corromperse.

En cualquiera de los casos, no necesitamos disponer de un programador externo sino que podemos grabar el gestor de arranque **bootloader** en un Arduino utilizando otro Arduino como programador. El **Arduino programador** es el que está perfectamente programado y en buenas condiciones de funcionamiento.

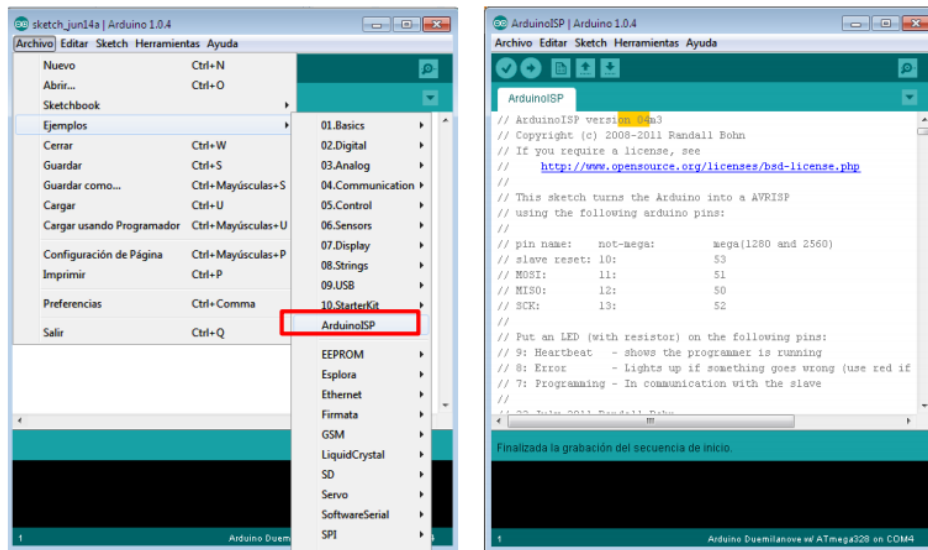
Al Arduino que actúa como programador lo llamaremos **MASTER**, y al que vamos a programar **SLAVE**. La comunicación entre PC y MASTER se realiza a través de puerto serie, USB, mientras que el MASTER se comunicará con el SLAVE a través de las conexiones **SPI**.

Este proceso consiste en la grabación del **gestor de arranque**, para ello, vamos a construir una estructura de programación en paralelo con una placa Arduino **Duemilanove** Atmega328P-PU que funcione correctamente y nos sirve como **Master** y, una placa de prototipo protoboard con un microcontrolador Atmega328P-PU, sin gestor de arranque o con problemas de ellos, será la que está configurado como **Slave**.

Para grabar el gestor de arranque, usando una placa de Arduino, hay que seguir estos pasos para el Arduino programador (Master):

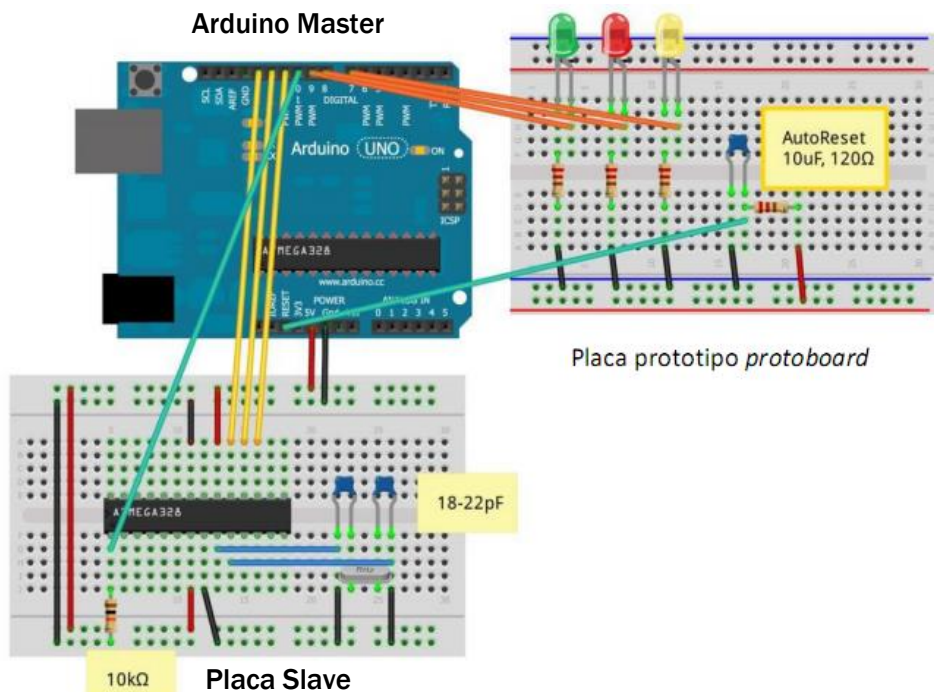
1. Utilizar un Arduino como programador y cargamos primeramente el programa **Archivo/Ejemplos/11. ArduinoISP** en la placa de **Arduino Master**: deberá seleccionar el tipo de tarjeta y el puerto serie del menú Herramientas que correspondan a su placa.

4. Grabar el Gestor de Arranque (Bootloader)



Una vez cargado el programa en la placa **Arduino Master** y sin errores, debe observarse que la placa Arduino no hace nada, no lucen ningún Led ni tampoco parpadean, esta situación es la correcta.

2. Desconectar la placa de Arduino **Master** del PC y la **alimentación**
3. Conectar la placa Arduino **Master** con la placa prototipo protoboard **Slave** y el microcontrolador ATmega328P, que no tenga el gestor de arranque, tal como se muestra en la siguiente imagen.



Existen diversos montajes para cargar el gestor de arranque; nosotros nos centraremos en uno de los más sencillos, ya que los componentes “extra” empleados son los que posteriormente necesitaremos para ensamblar nuestro Arduino autónomo.

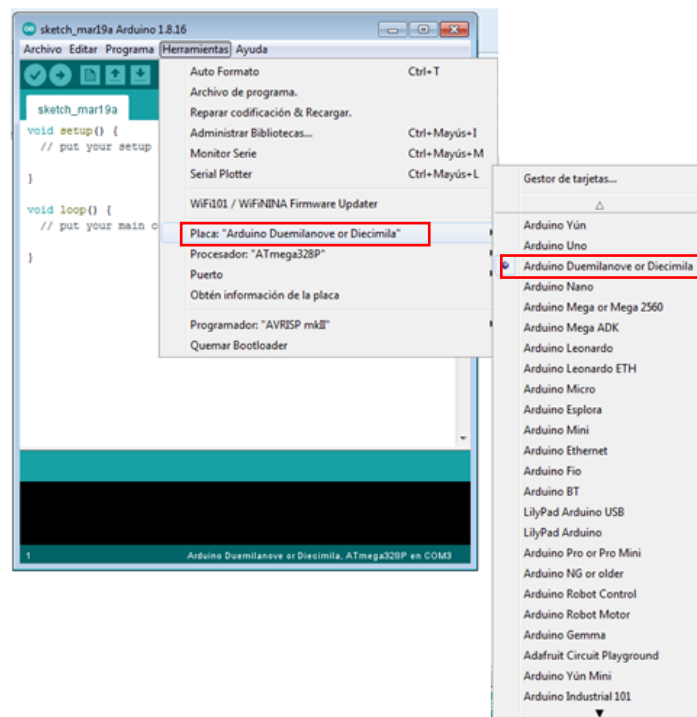
4. Grabar el Gestor de Arranque (Bootloader)

Componentes necesarios:

- 1. Placa Arduino Atmega328P-PU Duemilanove (Master)
- 1. Placa de prototipo protoboard (Slave)
- 1. Microcontrolador ATmega328P-PU (Sin Bootloader o defectuoso)
- 1. Oscilador Cristal de cuarzo a 16Mhz
- 2. Condensadores 22pF
- 3. Diodos Leds (Verde, Rojo y Amarillo)
- 1. Resistencia 10k Ω
- 3. Resistencias de 330 Ω (para los LEDs)
- 1. Resistencia 120 Ω
- 1. Condensador 10uF

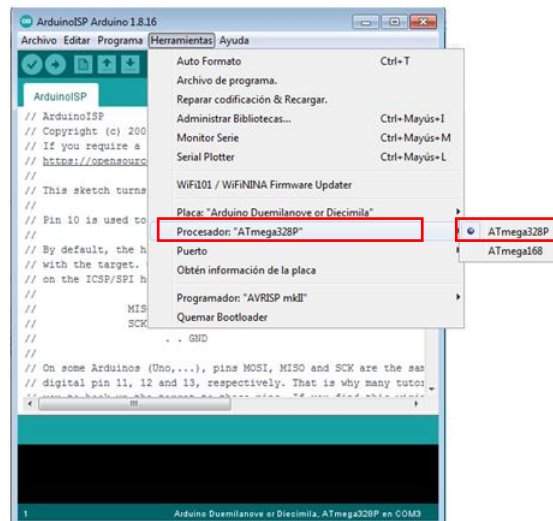
NOTA: La placa prototipo o protoboard se trata de una placa con conexiones internas en la que podemos pinchar nuestros componentes para realizar nuestros prototipos, sin tener que realizar un solo punto de soldadura de estaño, tantas veces como queramos. Debemos tener en cuenta como se distribuyen las conexiones internas de nuestra placa protoboard.

4. Conectar la placa Arduino **Master** al **PC**, si todo ha ido bien, al alimentar con USB en la placa de prueba protoboard **Slave** el LED verde comenzará a "latir"
5. Seleccionar en el menú **Herramientas/Placa/Arduino Duemilanove or Diecimila**

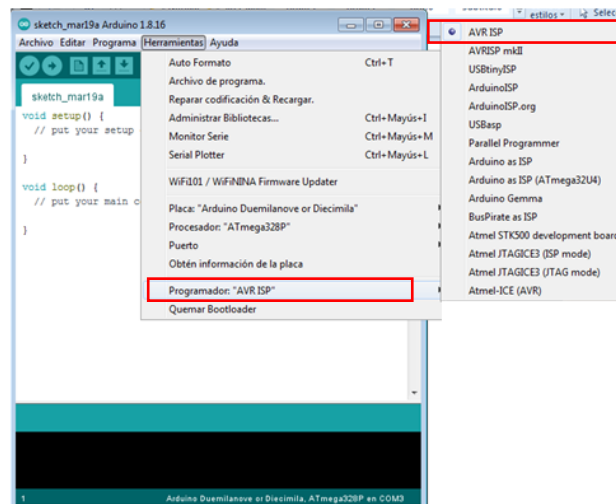


4. Grabar el Gestor de Arranque (Bootloader)

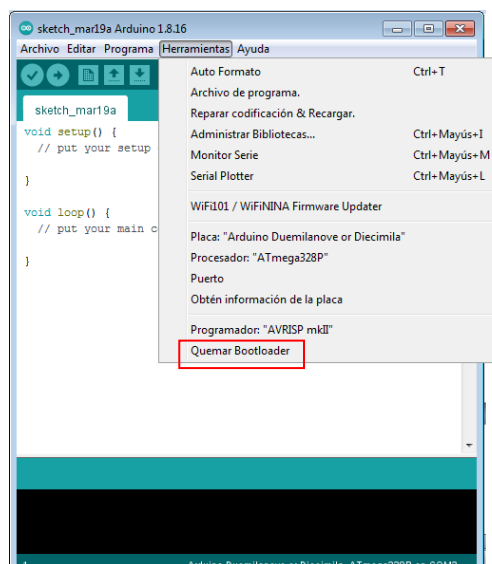
6. Seleccionar **Herramientas/Procesador/ATmega328P**



7. Seleccionar **Herramientas/Programador AVR ISP**



8. Ya estamos preparados para cargar el **bootloader** a nuestro microcontrolador que se encuentra en la placa de prototipo protoboard **Slave**. Algo tan sencillo como seleccionar a continuación **Herramientas/Quemar Bootloader**

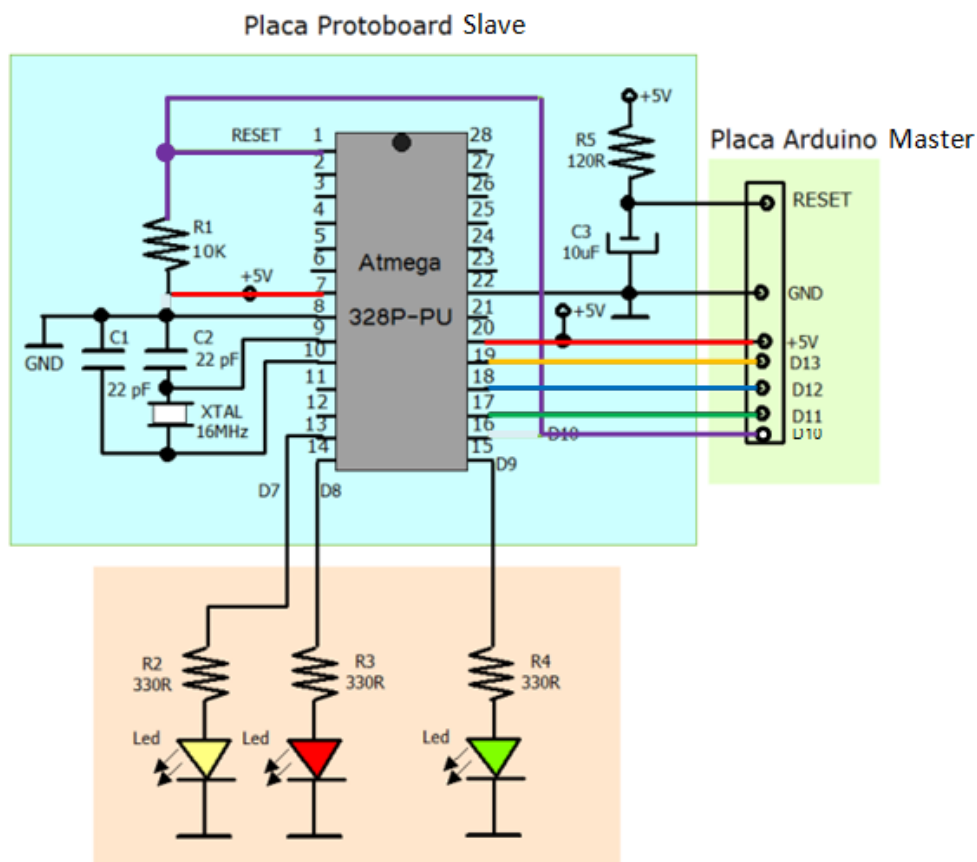


4. Grabar el Gestor de Arranque (Bootloader)

El proceso encenderá los Leds de la placa Arduino, **Lamp**, **Tx** y **Rx** y los Leds de la placa protoboard durante aproximadamente unos 15 segundos. Si todo va bien, se encenderá el LED amarillo de la placa Arduino y al cabo de un minuto tendremos nuestro Atmega328p listo para ser programado.

Seguidamente se puede quitar el microcontrolador ATmega328P de la placa Protoboard **Slave** y sustituirlo por el microcontrolador de la placa Arduino **Master** y cargar un programa sencillo como **Blink**. En este proceso debe parpadear los led **TX** y **RX** y posteriormente el led **L** de la placa Arduino, si es así, es buena señal y el microcontrolador está perfectamente grabado con su gestor de arranque.

Correspondencia de las conexiones	
Placa Arduino MASTER	Placa Protoboard SLAVE
Conector Hembra	Pines del Microcontrolador
GND	8 y 22
+5V	7 y 20
D13	19
D12	18
D11	17
D10	1 (RESET)



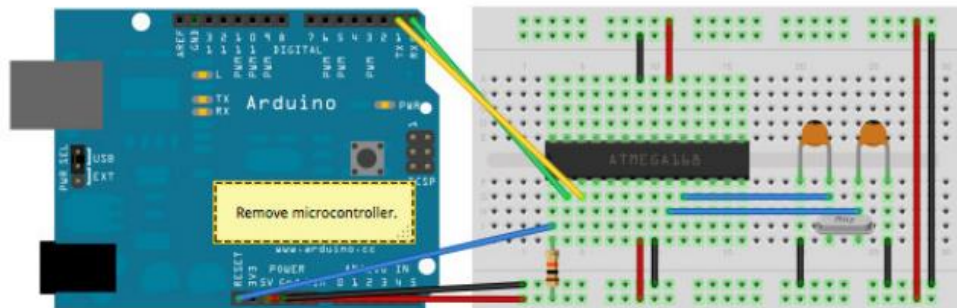
Esquema de conexiones de la placa Arduino (MASTER) a la placa protoboard (SLAVE)

5. Trabajar directamente con la placa Protoboard

Una vez que el microcontrolador Atmega328P tiene cargado el gestor de arranque y comprobado que funciona correctamente, se puede instalar en una placa prototipo de las denominadas **protoboard** con todos sus componentes adicionales y poder de esta forma cargar los programas utilizando los pines D0 y D1 (RX y TX) para llevar la transmisión y recepción desde la placa Arduino que se encuentra sin microcontrolador.

Para ello, hay que hacer unas pequeñas modificaciones. Primeramente debemos quitar el microcontrolador que viene en la placa de Arduino e insertarlo en la placa prototipo *protoboard* para que el chip de **FTDI** pueda comunicarse con el microcontrolador desde la placa Arduino.

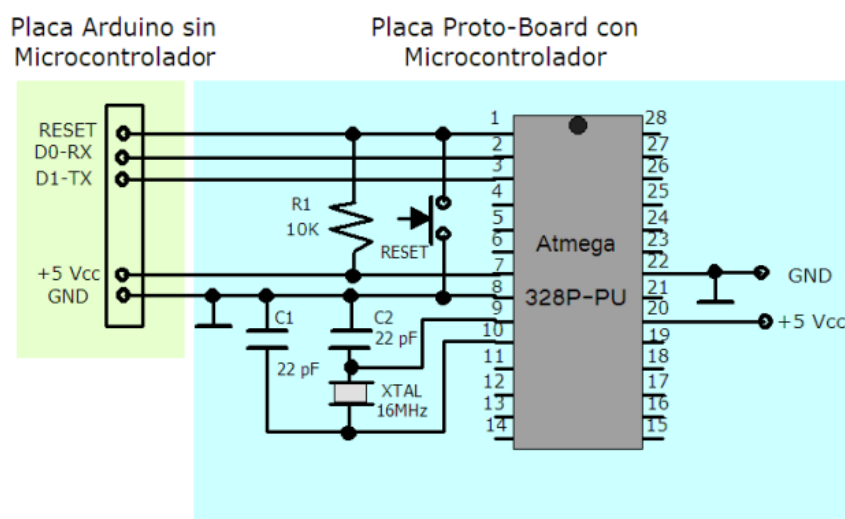
La imagen de abajo nos muestra cómo conectar las líneas **Reset, alimentación** y **Rx/Tx** de la placa Arduino a la placa de prototipo *protoboard*. Para programar el microcontrolador, seleccione en el menú: **Herramientas/Placa Arduino Duemilanove/ ATmega328**. A continuación, sube un programa y trabaja sobre la placa *protoboard*.



<https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard>

En esta configuración conectaremos los pines de comunicación **D0-Rx** y **D1-Tx** de la placa Arduino a los **pinos 2 y 3**, respectivamente, del microcontrolador Atmega328P montado en la placa *protoboard*.

También conectaremos el pin de **RESET** de la placa Arduino al **pin 1** del microcontrolador con una resistencia de 10K a +5V, la correspondiente alimentación de **+5 voltios** y **GND**, y la señal de reloj formado por un cristal de cuarzo de 16MHz y dos condensadores cerámicos de disco de 22pF.



5. Trabajar directamente con la placa Protoboard

Cuando se carga una programación veremos que los diodos leds **L**, **Tx** y **Rx** de la placa de Arduino comienza a encenderse, esto es buena señal, lo mismo ocurre cuando trabajamos con el monitor serie y se transmite y recibe los datos al PC. Como prueba inicial podemos cargar el programa **Blink** que nos indicará si todo ha ido correctamente.

A partir de ahora podemos ir trabajando con todos los componentes sobre la misma placa de prueba *protoboard* que contiene el microcontrolador.

NOTA: En esta fase es muy importante conocer y manejar el patillaje del microcontrolador ATmega328P para las conexiones de componentes para los diferentes proyectos. También es recomendable tener cuidado en la manipulación del microcontrolador: en su desmontaje y montaje ya que se pueden doblar fácilmente algunas patillas.

6. Componentes Hardware para conectar en Arduino

En los proyectos electrónicos intervienen unas series de componentes electrónicos que nos sirven para polarizar, señalar, amplificar, adaptar, detectar, ajustar o medir cualquier magnitud que necesitemos: velocidad, temperatura, movimiento, etc., para tratarla como datos para nuestro sistema de control.

Tanto los datos de entrada original como los de realimentación (los datos de salida) de los sistemas de control, son captados y se introducen en ellos mediante unos dispositivos que se denominan sensores. Los sensores serán dispositivos capaces de detectar las condiciones del entorno (temperatura, luz, humedad, movimiento...) y traducir esta información que le llega del exterior en un impulso eléctrico, normalmente digital (pasa o no pasa corriente), que puede ser analizado y procesado por la unidad de control del sistema.

Los sensores son componentes importantes para poder dotar a nuestros sistemas de sentidos.

Estos sensores son los encargados de captar la realidad que nos rodea en nuestro medio ambiente. Por ejemplo, sensores que nos los encontramos en las farolas de nuestras calles cuando se encienden de noche, detectores en las puertas automáticas cuando entramos en un supermercado, sensores que detectan la temperatura y humedad, detectores de aparcamiento del vehículo, etc...

SENSOR DE CONTACTO O BUMPER

Se emplean para detectar el final del recorrido o la posición límite de componentes mecánicos en movimiento.

El **bumper** es un microinterruptor que posee una lámina de metal u otro material que está en contacto con un pequeño interruptor. Así, cuando la lámina impacta con un objeto, ésta acciona el interruptor y se genera el cambio de posición en el microinterruptor. Por ejemplo: saber cuándo una puerta o una ventana que se abren automáticamente están ya completamente abiertas y por lo tanto el motor que las acciona debe pararse.

Los principales son los llamados **finés de carrera**. Se trata de un interruptor que consta de una pequeña pieza móvil y de una pieza fija que se llama NA, normalmente abierto, o NC, normalmente cerrado.



Diferentes tipos de microinterruptores-conmutados

6. Componentes Hardware para conectar en Arduino

SENSOR ÓPTICO CNY70

Este sensor óptico combina un diodo emisor de infrarrojos y un fototransistor receptor de infrarrojos que detecta objetos reflectantes y que sean blancos, pues los oscuros o negros no se reflejan. Su detección es a una distancia máxima de unos 8 mm. Estos sensores están indicados en máquinas de fabricación: detección de piezas, posicionamiento, contadores de piezas, etc.



Estructura interna y composición del sensor óptico CNY70

SENSOR DETECTOR DE OBSTACULOS IR INFRARROJOS

Este sensor de infrarrojos detecta obstáculos a una distancia de 2 a 30 cm. Con un ángulo de detección de 35°. Está constituido por un diodo emisor de infrarrojos y un fotodiodo receptor de infrarrojos que en el caso de detectar un objeto envía una señal, pudiendo activar un relé.



Sensor detector de obstáculo por infrarrojos

SENSOR MAGNÉTICO

Detecta los campos magnéticos que provocan los imanes o las propias corrientes eléctricas. Consiste en un par de láminas metálicas de materiales ferromagnéticos metidas en el interior de una cápsula que se atraen en presencia de un campo magnético (imán), cerrando con ello el circuito. El principal es el llamado interruptor **Reed**. Éste puede sustituir a los finales de carrera para detectar la posición final de un elemento móvil con la ventaja de que no necesita ser empujado físicamente por dicho elemento sino que puede detectar la proximidad sin contacto directo.

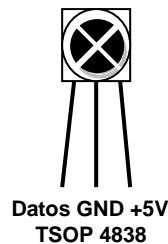


Sensores Reed magnéticos con diferentes formatos y simbología.

6. Componentes Hardware para conectar en Arduino

SENSOR RECEPTOR DE INFRARROJOS

En el espectro electromagnético, consiste en una franja de ondas electromagnéticas cuya frecuencia es muy baja para que nuestros ojos la detectaran. Esta franja son los **infrarrojos**. Pues bien, existen diodos capaces de emitir luz infrarroja y transistores sensibles a este tipo de ondas y que por lo tanto detectan las emisiones de los diodos. Esta es la base del funcionamiento de los mandos a distancia de nuestros televisores y detectores de posicionamiento de objetos.



Receptores de infrarrojos: TSOP4838, VS1838 y diodo emisor de infrarrojo

Estos sensores de infrarrojos para controlar adecuadamente su funcionamiento en la programación de Arduino necesitan utilizar librerías. El sensor infrarrojo **TSOP 4838** necesita de una librería especial la **IRremote**.

SENSOR DE MOVIMIENTO POR INFRARROJO PIR (HC-SR501/5)

Un sensor detector de movimientos por infrarrojos **PIR**, es un dispositivo pasivo que está formado por dos infrarrojos y una lente de tipo fresnel que es una capsula de espejos que amplía como un abanico los haces de infrarrojos que se propagan al exterior, adaptándose a la temperatura de los cuerpos y objetos que hay en el recinto. Estos sensores mide constantemente la diferencia de calor que puede haber entre un cuerpo y la estancia en la que está ubicado. Cuando un cuerpo pasa por su campo de acción, una de las partes detecta el calor del cuerpo y detecta una diferencia entre el calor que tenía como referencia y el detectado en ese mismo instante. La detección pueden activar un relé, una lámpara, un timbre, una electroválvula, etc.



Sensor de movimiento PIR HC-SR501

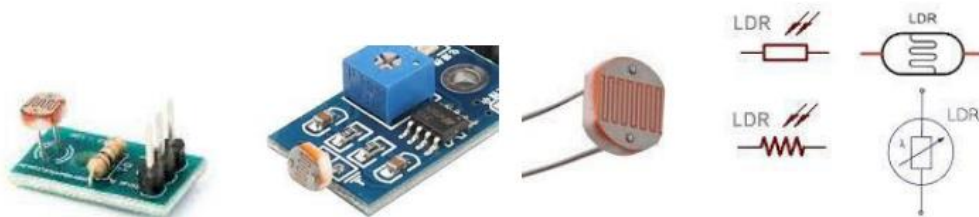
El sensor de movimientos **HC-SR501** nos permite detectar la presencia de un cuerpo en lugares como una habitación, un despacho o cualquier otro tipo de estancia. No requiere de ninguna librería extra por parte de Arduino. Este sensor da un 1 si detecta movimientos o un 0 si no lo detecta, por lo tanto, es un sensor que podemos conectar en uno de los pines digitales de Arduino.

6. Componentes Hardware para conectar en Arduino

SENSORES DE LUZ LDR

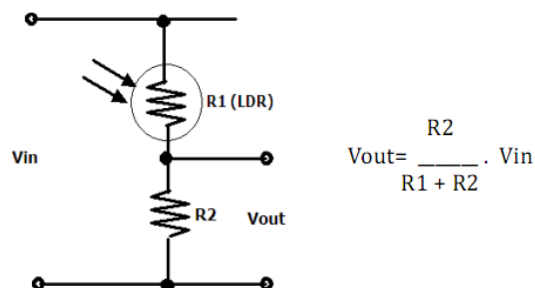
Las siglas **LDR** provienen del inglés *Light Dependent Resistor* (resistencia que depende de la luz). El valor que nos proporciona una **LDR** variará dependiendo de la cantidad de luz que incida sobre ella. El valor de la resistencia será bajo cuando la luz incida sobre ella, y será alto cuando no incida luz sobre ésta. Los valores de una LDR pueden ir de unos 50 Ohmios a varios Mega Ohmios.

Es decir, la **LDR** es un componente electrónico pasivo cuyo valor de la resistencia varía en función de la luz que recibe. Cuanta más luz reciba, el valor de su resistencia será menor. Su variación no es lineal, sino exponencial. Lleva una resistencia de polarización y ajuste.



Sensores de luz LDR y simbología

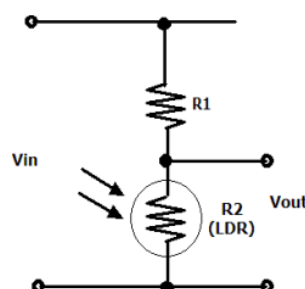
A todo esto, podemos comprender el funcionamiento de una LDR como parte de un divisor de tensión resistivo. Observemos el siguiente esquema:



Como se ha explicado anteriormente, tal como está configurado este divisor, la tensión resultante o de salida será la R_2 entre la suma de la **LDR** y la resistencia R_2 , y todo multiplicado por la tensión de entrada V_{in} .

Si la R_1 es la LDR, obtendremos una tensión alta en la salida si incide luz en la LDR, mientras que obtendremos una tensión baja o aproximadamente cero cuando no incide luz en la LDR.

Estas condiciones se pueden cambiar de una forma bien simple. Si, por ejemplo, deseamos que la **LDR** actúe al revés de como se ha explicado anteriormente, solo debemos intercambiar la LDR (R_1) con la R_2 , quedando de esta manera, la **LDR** abajo y la R_2 arriba, con lo que cuando la luz incide sobre la LDR, la tensión es baja, y cuando la luz no incide en la LDR, la tensión es alta.



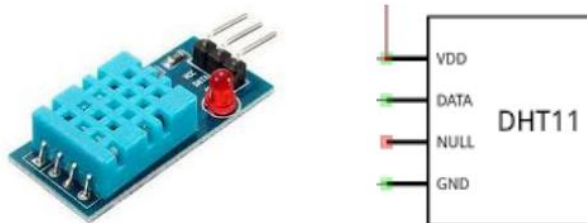
6. Componentes Hardware para conectar en Arduino

Debido a la naturaleza de la información resultante (en milivoltios), el sensor **LM35** se deberá conectar en las entradas analógicas de Arduino y se deberá tratar matemáticamente para obtener unos valores en grados acorde con las temperaturas que normalmente utilizamos.

```
valor=analogRead(temp);           //guardamos los datos leído temp en valor
grados=(5.0*valor*100.0)/1024;   //obtenemos los datos de grados
```

SENSORES DE TEMPERATURA Y HUMEDAD DHT11/22

El sensor **DHT11** o **DHT22** son dispositivos sensores que permitirá tomar lecturas de la humedad y la temperatura. Su precisión no es muy alta pero nos puede servir para un gran número de propósitos en los que no nos encontremos en condiciones extremas.

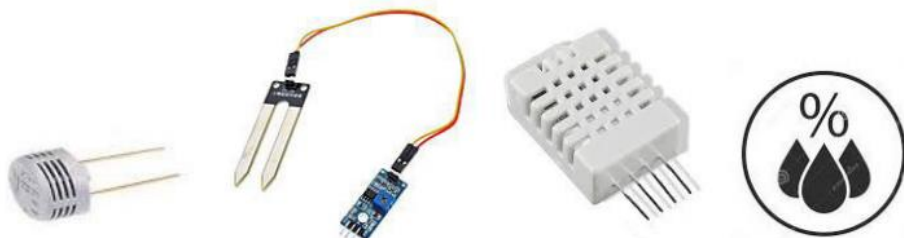


Estructura física y simbología del sensor de temperatura y humedad DHT11

SENSOR DE HUMEDAD

Se basan en la propiedad del agua como elemento que posee conductividad eléctrica (a diferencia del aire). Por lo tanto, un par de cables eléctricos desnudos (sin cinta aislante o plástico recubriéndolos) van a conducir una pequeña cantidad de corriente si el ambiente es suficientemente húmedo.

Si colocamos un transistor que amplifique esta corriente tendremos un detector de humedad. Los sensores de humedad se aplican para detectar el nivel de líquido en un depósito, en sistemas de riego de jardines para detectar cuándo las plantas necesitan riego, etc.

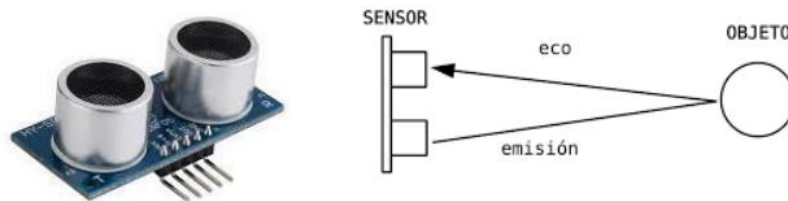


Diferentes sensores de humedad

6. Componentes Hardware para conectar en Arduino

SENSOR DE ULTRASONIDOS HC- SR04

El sensor de ultrasonidos **HC-SR04** es un dispositivo que genera sonidos en un rango de frecuencias de los 20KHz hasta los 400 KHz, aproximadamente. Está compuesto por dos elementos, un **emisor** que emite ondas de ultrasonidos en forma de eco y un **receptor** que captura el eco y es capaz de escuchar esas frecuencias que captaría las ondas sonoras que rebotan contra los objetos que se interpongan emitidas por el emisor. De esta manera sabrá que tenemos un objeto delante y la distancia a la que éste se encuentra, calculando lo que la onda ha tardado en volver, ya que conocemos la velocidad del sonido en el aire.



Sensores de ultrasonidos HC-SR04

Dispone de cuatro terminales: los dos terminales de los extremos es para alimentar el dispositivo (+5V y GND). Los dos terminales interiores están encargados de emitir ultrasonidos y el terminal de eco de recibir y capturar las frecuencias emitidas por el emisor.

SENSOR DETECTOR DE POLUCION CONTAMINACIÓN MQ-135

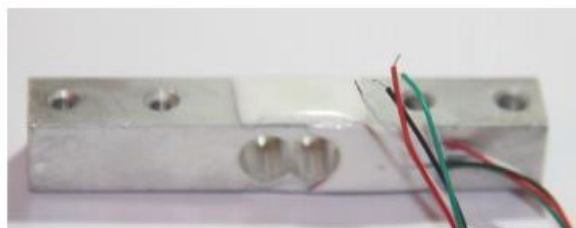
El **MQ-135** es un sensor de aire que detecta niveles de polución contaminante. Es ideal para colocarlo en cualquier parte del recinto que estemos interesados en saber el nivel de polución y contaminación que existe.



Detector de polución MQ-135

SENSOR DE CARGA EL0418

El **EL0418** es un sensor de carga cuyo peso máximo permitible es de 20Kg que trabaja de 3 a 12 voltios en corriente continua. Este sensor detecta los diferentes valores de peso señalizándolo al sistema de control.



Sensor de carga EL0418

6. Componentes Hardware para conectar en Arduino

SENSORES ACELERÓMETROS

Estos tipos de sensores nos permiten medir el grado de inclinación y la aceleración de un determinado objeto en uno, en dos o incluso en los tres ejes.



Sensores acelerómetros

SENSORES MICRÓFONOS

Estos sensores captan los sonidos (variaciones de presión del aire) y los transforman en señales electrónicas basándose en el efecto piezoeléctrico. Necesitan un filtrado y una amplificación para una determinada detección y obtener una señal adecuada.



Sensores microfónicos y simbología

SENSOR DE VELOCIDAD LM393

Este sensor está ideado para la detección de velocidad y contador de pulsos. Está constituido por el **LM393** que es un doble comparador A.O. Un optoacoplador óptico formado por un diodo Led y un fotodiodo, enfrentados, y esto nos permite detectar cuando se coloca un objeto en medio, se produce una señal que nos puede servir como contador de pulsos, en el control de velocidad de las impresoras, por ejemplo.



Sensores de velocidad LM393

6. Componentes Hardware para conectar en Arduino

ACTUADORES LED

Diodo emisor de luz. Podemos encontrarlos de múltiples colores. Los electrones que lo atraviesan producen una liberación de energía en forma de radiación lumínica. Para lograr que luzca, debemos hacer que por él circule una potencia superior a la tensión umbral del led (el mínimo que necesita para lucir), pero inferior a la máxima que puede soportar, o se quemará, por lo que es importante interponer resistencias.

El LED tiene polaridad. La pata más larga debe conectarse al ánodo (+). El lado achatado de la capsula corresponde al cátodo del diodo Led (-).



Estructura física del diodo Led y simbología

ACTUADORES DIODO LASER

El **diodo laser** (*Light Amplification by Stimulated Emission of Radiation*) se conoce como un diodo que realiza una amplificación de luz por emisión estimulada de radiación.

La luz que emitimos por un diodo Led, o una bombilla convencional, por ejemplo, es una luz dispersa, mientras que la luz que emitimos con un diodo laser es una luz concentrada en un punto.



Diodo laser concentra toda la luz en un punto

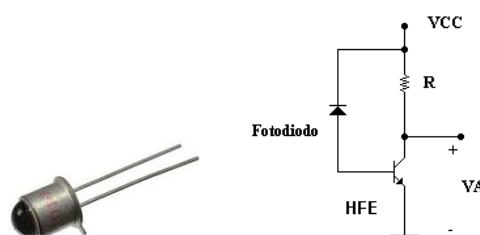


Diodo laser para Arduino

Por tanto, el diodo led laser nos aportan ventajas, como fiabilidad, eficacia, manejabilidad en cuanto al peso, distancias largas del haz, pero como inconveniente, es su capacidad para alinear un punto de luz a mucha distancia sobre un elemento receptor, por ejemplo, una LDR.

FOTODIODO

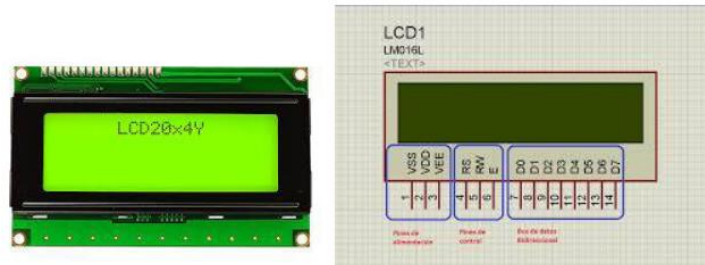
Un **fotodiodo** es un semiconductor construido con una unión PN, sensible a la incidencia de la luz visible o infrarroja. Para que su funcionamiento sea correcto se polariza inversamente, con lo que se producirá una cierta circulación de corriente cuando sea excitado por la luz.



6. Componentes Hardware para conectar en Arduino

ACTUADORES PANTALLA LCD

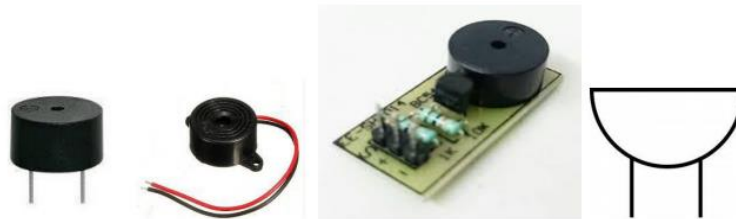
Pantalla de cristal líquido formada por píxeles. Para iluminar alguno de ellos lo que se hace es aplicar un campo eléctrico en la zona deseada, con lo que el cristal líquido se polariza y lo percibimos de otro color.



Actuadores de pantalla LCD

ACTUADORES ZUMBADORES

Estos dispositivos producen una vibración cuando se ve sometido a una corriente eléctrica basándose en el efecto piezoeléctrico. Esta vibración genera un sonido y puede emplearse como sistema de aviso.



Estructura física de los zumbadores y simbología

PULSADORES

Actúan cerrando o abriendo el circuito eléctrico mientras tengamos oprimido el botón para permitir o interrumpir el paso de la corriente eléctrica hacia otros componentes. Una vez que dejamos de oprimir el pulsador vuelve a su estado inicial. Los hay de dos tipos: uno que son **normalmente abierto NA** que cuando se pulsa, cierra el circuito, con lo que la corriente circula por todo el circuito y, los hay que son **normalmente cerrados NC**, que cuando se pulsa, abre el circuito, con lo que la corriente deja de circular por el circuito eléctrico.



Diferentes tipos de pulsadores y simbología

6. Componentes Hardware para conectar en Arduino

INTERRUPTORES

Actúan cerrando o abriendo el circuito eléctrico, estado ON-OFF, encendido o apagado, de los circuitos y equipos electrónicos. Permaneciendo enclavado en un estado cerrado o abierto hasta que no volvamos a actuar sobre la palanca del mando del interruptor.



Diferentes tipos de interruptores y su simbología

MOTORES DE CC

El motor de corriente continua CC es un dispositivo actuador que convierte la energía eléctrica en energía electromotriz. Esta energía electromotriz se destina, mediante engranajes externos, a producir movimiento. Funciona sobre la base de unos imanes que crean campos magnéticos opuestos y esto hace que su eje interno gire, produciendo movimiento.

Los motores de corriente continua tienen polaridad (+VCC y GND) por lo que pueden tener dos sentidos de rotación, según cambiemos la polaridad de la tensión en sus bornes de conexión, el sentido puede girar a la derecha o a la izquierda. Estos motores necesitan de un regulador de velocidad con Arduino para poder ser utilizados en proyectos como robot, seguidores de línea o robots.



Motor de Corriente Continua

SERVOMOTORES

Un servomotor básicamente es un motor de corriente continua con un potenciómetro que le permite saber la posición en la que se encuentra y así poder controlarla, es decir, que podemos posicionarlo a nuestro antojo, siempre dentro de su rango de actuación. Por lo general los servomotores suelen tener un rango de 180°.

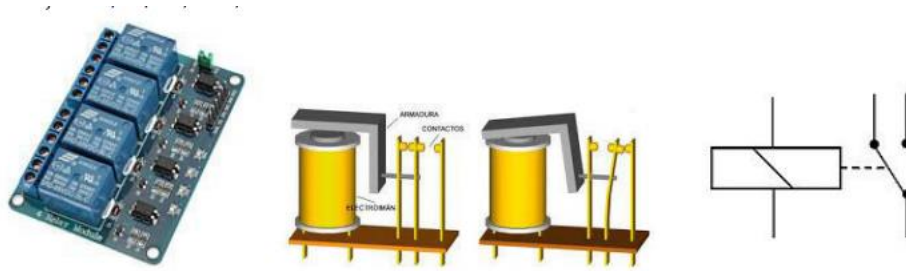


Servomotores de corriente continua y simbología

6. Componentes Hardware para conectar en Arduino

ACTUADORES DE RELES

El **relé** es un dispositivo electromecánico. Funciona como un interruptor-conmutador controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos conmutados, NC-C-NA, de gran potencia, que permiten abrir o cerrar otros circuitos eléctricos independientes que requieran de mayor corriente de trabajo. Según el tipo de relé, la tensión de la bobina puede trabajar a 3,3V, 5V, 12V, o 24 Voltios.



Relé en reposo. Relé activado

Estructura física de un relé y simbología

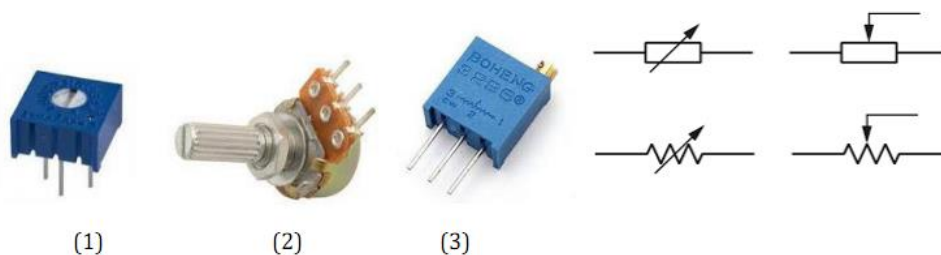
POTENCIÓMETROS

Un **potenciómetro** es una resistencia variable con el que podremos elegir el valor que éste puede tomar. Es una resistencia en la que se puede ajustar su valor girando una ruedecita o manecilla de derecha a izquierda o viceversa, aumentando la resistencia o disminuyéndola a nuestro gusto. De esta forma, controlamos la intensidad de corriente que fluye por un circuito si éste está conectado en paralelo, así como la diferencia de potencial si está conectado en serie.

La resistencia máxima que ofrece el potenciómetro entre sus dos extremos (que es constante) no es más que la suma de las resistencias entre los dos extremos y la patilla central.

El potenciómetro posee tres terminales o patillas, el que está en medio se utiliza para ajustar el valor de la resistencia a un determinado valor, por ejemplo, si la resistencia entre sus dos extremos tiene un valor de 10K, si colocamos el cursor en medio, tendremos un valor de 5K. A la misma vez que vayamos girando el terminal central hacia un extremo u otro del potenciómetros se observa que en un extremo la resistencia es mayor y en el otro extremo la resistencia es menor y viceversa.

(1) Resistencia ajustable (2) Potenciómetro manual (3) Resistencia ajustable de precisión.



Diferentes tipos de potenciómetros y simbología

6. Componentes Hardware para conectar en Arduino

RESISTENCIAS

Los resistores o resistencias fijas, son unos componentes utilizados para añadir una resistencia eléctrica entre dos puntos de un circuito de manera que nos permite distribuir adecuadamente tensiones y corrientes a lo largo de nuestro circuito, polarizando circuitos, divisores de tensión, cargas, protección, etc.

La unidad de medida es el **Ohmio** (Ω) y existe un abanico muy amplio de valores de resistencia y, de pequeña, mediana y gran potencia.



Diferentes tipos y tamaños de resistencias y su simbología

CONDENSADORES

Un condensador es un dispositivo empleado en electrónica para almacenar energía, que será proporcional a la diferencia de potencial que exista entre las dos placas que lo conforman. Está formado por un par de superficies conductoras, generalmente en forma de láminas o placas, en situación de influencia total (esto es, que todas las líneas de campo eléctrico que parten de una van a parar a la otra) separadas por un material dieléctrico por el vacío. Las placas, sometidas a una diferencia de potencial, adquieren una determinada carga eléctrica, positiva en una de ellas y negativa en la otra, siendo nula la variación de carga total.



Diferentes tipos de condensadores y su simbología

OPTOACOPLADORES

Un optoacoplador es un componente que contiene un LED infrarrojo y un fotodetector, generalmente un fototransistor, montados en un mismo encapsulado de modo que están acoplados óptimamente. Si no hay corriente en los terminales 1 y 2 del diodo Led, el fototransistor permanece cortado y por lo tanto en los terminales 3 y 4 no conduce y la carga está sin alimentar. Al llegarle corriente de 5V al LED emite la luz que saturará el fototransistor y éste comienza a conducir y la carga de 24V se alimentará.

6. Componentes Hardware para conectar en Arduino

La característica principal en todo circuito con optoacoplador es el aislamiento eléctrico entre los circuitos de control y el de la carga, que están separados de la diferencia de potencial entre uno y otro.

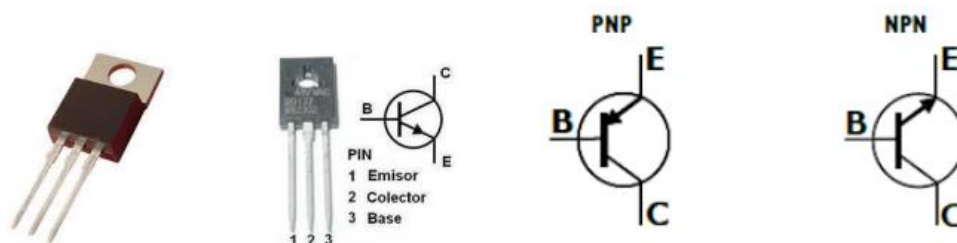


Diferentes tipos de optoacopladores y su simbología

TRANSISTORES

Un transistor es un dispositivo semiconductor usado para amplificar e interrumpir señales electrónicas o potencia eléctrica. Está compuesto de materiales semiconductores y de tres terminales, Emisor, Base y Colector para conexión externa al circuito. Gracias a que la potencia de salida puede ser más grande que la potencia de control un transistor puede amplificar una señal.

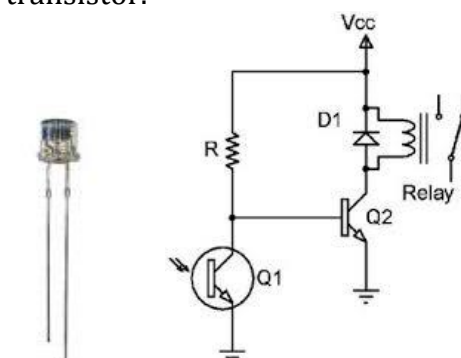
Los hay del tipo NPN y PNP. Algunos transistores aún son construidos en encapsulados individuales, pero la mayoría son construidos como parte de circuitos integrados.



Diferentes tipos de transistores, de pequeña y mediana potencia, NPN y PNP, y su simbología.

FOTOTRANSISTORES

Un **fototransistor** es un transistor sensible a varias longitudes de onda de la luz. La luz incide sobre la región de base, generando portadores en ella. Esta carga de base lleva el transistor al estado de conducción. El fototransistor es más sensible que el fotodiodo por el efecto de ganancia propio del transistor.



Fototransistor y circuito aplicativo

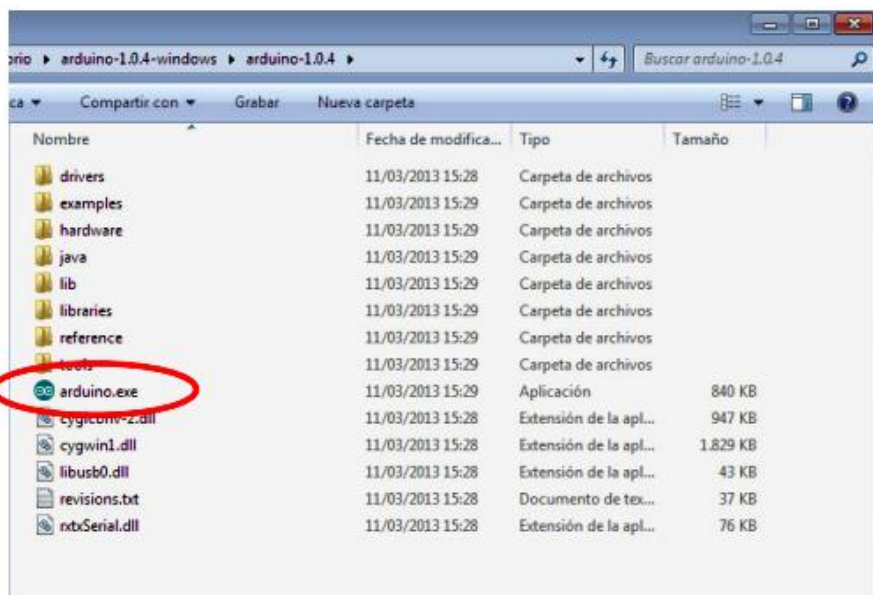
7. Entorno de desarrollo de Arduino

Hemos visto que para poder trabajar con Arduino necesitamos un hardware, un software y un programa. Ya conocemos al detalle el hardware, que es la propia placa Arduino y todos los componentes que podemos conectar a ella. Ahora nos adentraremos en la instalación del software. Puesto que Arduino, a diferencia del ordenador que usas normalmente, no tiene pantalla ni teclado, se necesita un programa externo ejecutado en otro ordenador para poder escribir programas para la placa Arduino. Éste software es lo que llamamos Arduino IDE. IDE significa “*Integrated Development Environment*” (Entorno de Desarrollo Integrado).

Para programar la placa es necesario descargarse de la página web de Arduino el entorno de desarrollo (IDE). Se dispone de versiones para Windows y para MAC, así como las fuentes para compilarlas en LINUX. <http://www.arduino.cc/en/Main/Software>

La interfaz del entorno de desarrollo de Arduino cuenta con un entorno nativo creado en JAVA, por lo que es multiplataforma y el lenguaje que utiliza es propio de Arduino y está basado en el lenguaje C/C++.

NOTA: Cuando termine la descarga, descomprime el archivo descargado. Asegúrate de conservar la estructura de carpetas. Haz doble clic en la carpeta para abrirla. Debería haber archivos y subcarpetas en su interior.



Una vez conectados PC y Arduino, el sistema operativo Windows detecta el dispositivo conectado al puerto USB. Normalmente, en los IDE actuales (1.5, 1.6, 1.8 y en adelante) los drivers USB se instalan en el momento en que se está instalando el IDE; de esta forma, se detecta a Arduino automáticamente cada vez que conectamos la placa al PC. Por eso se aconseja instalar la IDE más actual posible si es la primera vez que lo hace.

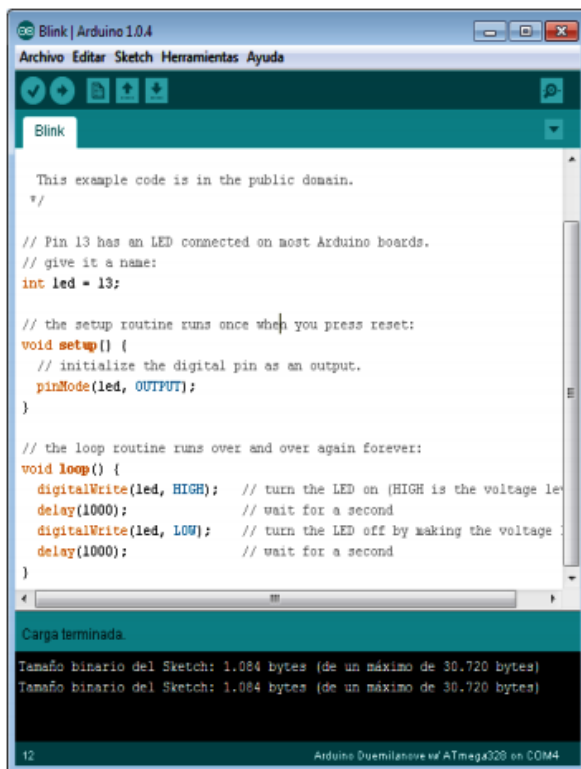
De todas formas si necesitamos instalar o reinstalar los drivers para el chip FTDI de la placa. Estos pueden encontrarse en el directorio drivers /FTDI USB Drivers de la distribuidora Arduino: <http://www.ftdichip.com/Drivers/VCP.htm>

7. Entorno de desarrollo de Arduino

Una vez que ejecutamos el entorno de Arduino nos aparece la ventana de inicio de la aplicación.

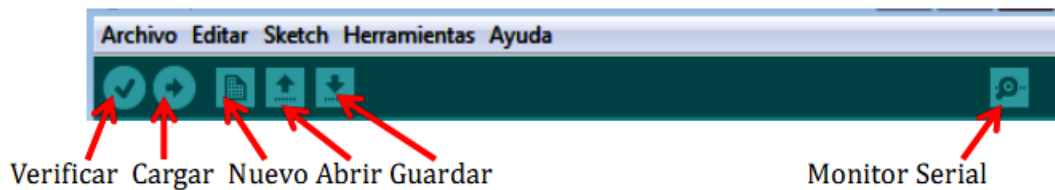


En las siguientes imágenes se muestran el aspecto del entorno de programación de Arduino IDE, con el programa **Blink** cargado y compilado correctamente y otro con error de compilación por faltarle un “punto y coma” en una de las instrucciones del programa.



7. Entorno de desarrollo de Arduino

La barra de botones rápidos, son las funciones más importantes que solemos utilizar con más frecuencia para poder empezar y trabajar con el programa en el entorno de Arduino.



Botón Verificar

Después de escribir un programa es conveniente revisar los posibles errores sintéticos que se hayan podido cometer, por eso es aconsejable clicar el botón de verificación para que el compilador determine si todo el código que se ha escrito está libre de errores sintéticos.

En el caso de que el código tuviera errores, el IDE de Arduino lo muestra en la ventana inferior con fondo negro y las letras de color naranja.

Botón Cargar

Permite cargar el programa, ya escrito y corregido sintácticamente, al microcontrolador de Arduino. Si optamos por cargar un programa que no ha sido verificado anteriormente, el proceso de verificación sintáctica se realiza también antes de que tenga lugar la carga en el microcontrolador.

Botón Nuevo

Genera un nuevo sketch. Al clicar sobre él se abre una nueva ventana, y un nuevo sketch está listo para ser programado.

Botón Abrir

Abre una ventana de diálogo, mostrando la carpeta por defecto donde el programa guarda los **sketchs**. Clicando en este botón podemos recuperar los sketchs de proyectos anteriores.

Botón Guardar

Como su nombre indica, permite guardar el sketch actual en el directorio que el usuario escoja. Por defecto, guardará el **sketch** en el directorio Mis documentos/arduino.

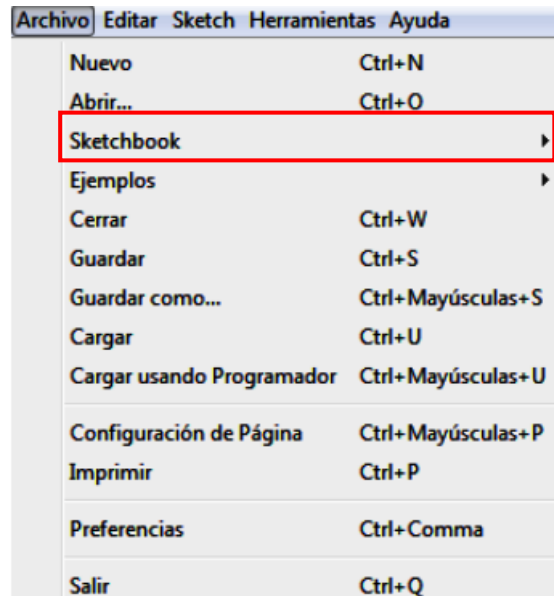
Botón Monitor Serie

Abre una ventana en la que podemos observar el valor que va adquiriendo las variables o para interaccionar con Arduino. Para poder realizar cualquiera de estas dos acciones, se deben introducir las órdenes necesarias en el programa.

7. Entorno de desarrollo de Arduino

7.1. Menú Archivo

Este menú permite gestionar los archivos Arduino con extensión *.ino. Se podrá abrir, cerrar, cargar, guardar, imprimir, etc. En la opción **Sketchbook** nos aparece nuestros archivos abierto recientemente. Algunas preferencias pueden ser ajustadas en la opción Preferencias.



7.2. Menú Editar

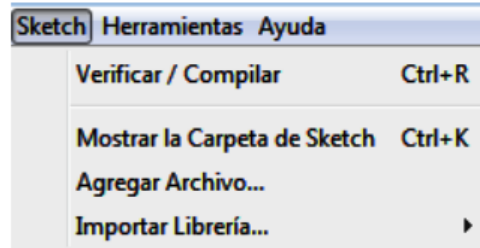
Este menú permite realizar la gestión del contenido de los archivos de Arduino: copiar, cortar, pegar, seleccionar, etc. En **Copiar para el Foro**, nos permite copiar el código de nuestra rutina al portapapeles de forma conveniente para postear en un foro.



7. Entorno de desarrollo de Arduino

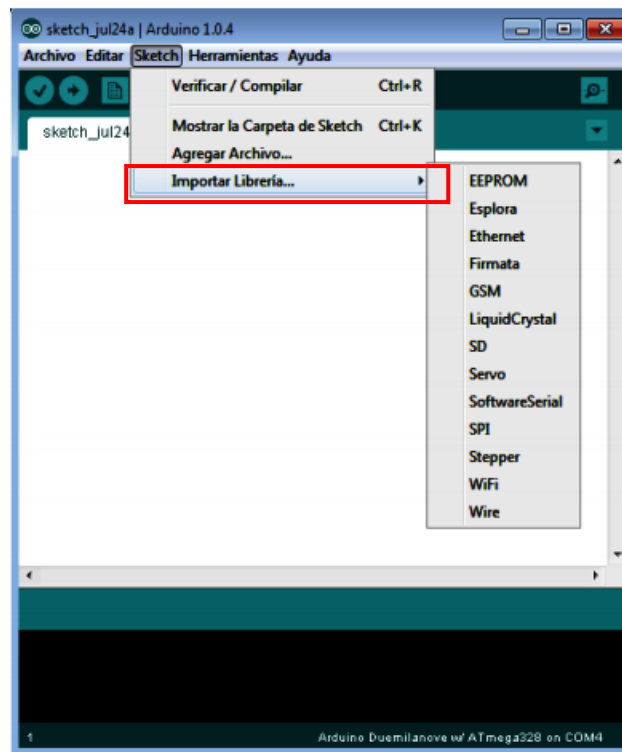
7.3. Menú Sketch

Este menú permite Verificar/Compilar, comprobando las rutinas para errores. Mostrar la carpeta de Sketch, mostrando la carpeta de rutina en el escritorio, Agregar Archivo fuente a la rutina e Importar Librería.



Las **librerías** son un conjunto de funciones e instrucciones que hacen que un dispositivo se pueda vincular a Arduino de una forma más sencilla, como por ejemplo, el número de sensores, con lo que en ocasiones será de gran ayuda introducir las librerías para poder manipular de manera más sencilla el sensor en el código del proyecto.

Tenemos dos tipos de librerías: las que incluye el **IDE de Arduino** y las que son de **contribución**. Estas últimas son librerías desarrolladas por usuarios de Arduino, que las comparten con los demás usuarios para facilitar la programación.



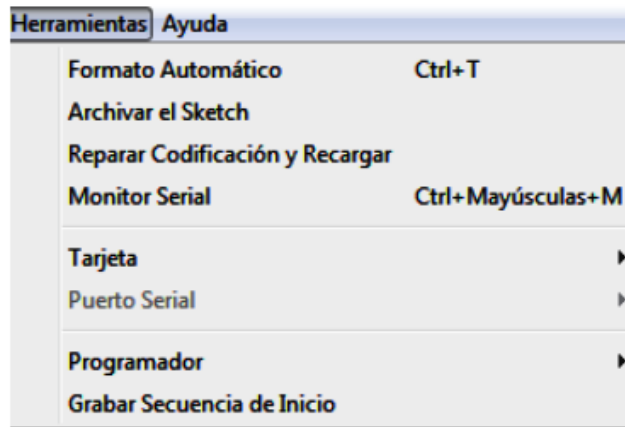
Las librerías que incorpora por defecto Arduino y que se deben invocar o cargar manualmente se pueden observar si hacemos clic en **Sketch / Importar Librería...** en el IDE de Arduino.

Podemos ver algunos, como SD, EEPROM, GSM, Servo, etc. Si seguimos mirando la lista, llegamos a un punto donde nos aparece la palabra **contribución**, a partir de ahí, las librerías que encontremos son externas, lo que quiere decir que tenemos que descargarla e instalarla nosotros mismo.

7. Entorno de desarrollo de Arduino

7.4. Menú Herramientas

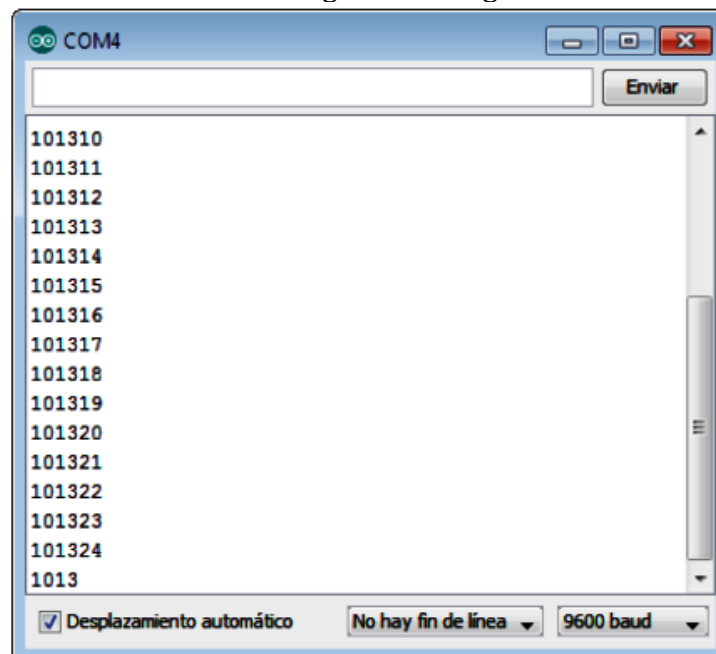
Este menú permite utilizar diversas opciones para mejorar la usabilidad de las instrucciones y códigos de nuestro programa.



En **Formato Automático** nos formatea el código amigablemente. Además podremos archivar el Sketch, Reparar Codificación y Recargar.

La opción del **Monitor Serial** nos permite a través de una ventana, enviar y recibir datos a nuestro programa que se está ejecutando. Se conecta por medio del puerto serie que es la forma principal de comunicar una placa Arduino con un ordenador.

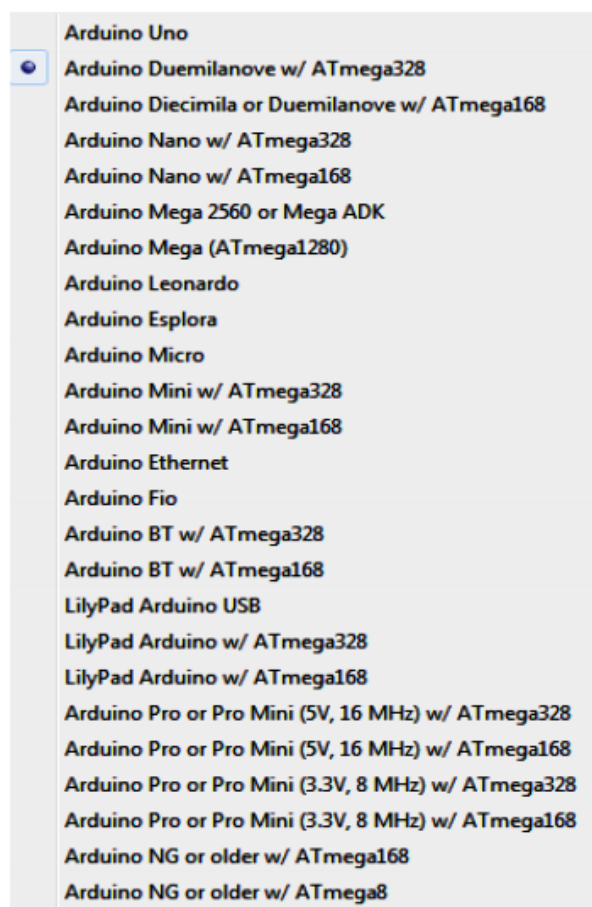
El monitor de puerto serie es una pequeña utilidad integrada dentro de **IDE Standard** que nos permite enviar y recibir fácilmente información a través del puerto serie. Su uso es muy sencillo, y dispone de dos zonas, una que muestra los datos recibidos, y otra para enviarlos. Estas zonas se muestran en la siguiente imagen.



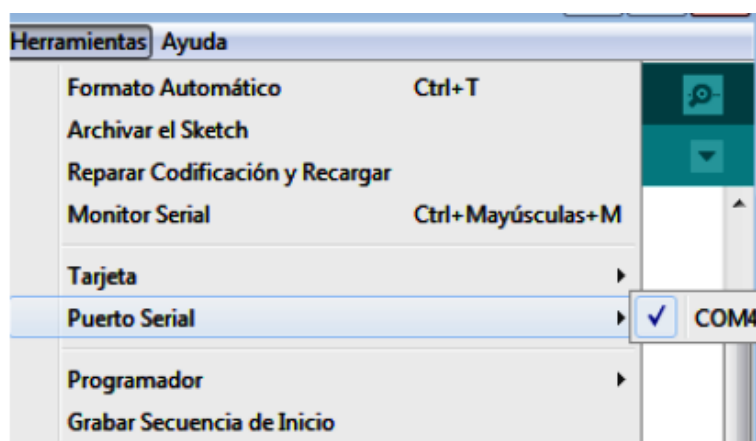
NOTA: Para realizar la conexión mediante puerto serie únicamente es necesario conectar nuestra placa Arduino empleando el mismo puerto que empleamos para programarlo. A continuación abrimos el IDE Standard de Arduino y hacemos click en el "Monitor Serial".

7. Entorno de desarrollo de Arduino

La opción **Tarjeta**, nos permite seleccionar, entre todas ellas, la placa de Arduino que corresponde con la que tenemos disponible. Por ejemplo, **Arduino Duemilanove w/ATmega328**.



El **Puerto Serial** contiene todos los dispositivos series (reales o virtuales) de nuestro PC. Antes de subir el programa, necesitamos seleccionar el elemento de este menú que representa a nuestra placa Arduino. En el Mac, esto es probablemente algo como `/dev/tty-usbserial-1B1` (para la placa USB), o `/dev/tty.USA19QW1b1P1.1` (para una placa serie conectada con un adaptador USB-a-Serie Keyspan). En Windows, es probablemente COM1 o COM2 (para una placa serie) o COM4, COM5, COM7 o superior (para una placa USB).



7. Entorno de desarrollo de Arduino

La opción de **Programador** permite grabar un **bootloader** (Grabar Secuencia de Inicio) en tu placa con una variedad de programadores. Esto no es necesario para uso normal de una placa Arduino, pero puede ser útil si encargas **ATmega** adicionales o estás construyendo una placa por tu cuenta. Asegúrate que has seleccionado la placa correcta del menú **Tarjeta**. Para grabar un bootloader con el **AVR ISP**, necesitas seleccionar el elemento que corresponde a tu Programador y seguidamente pulsar **Grabar Secuencia de Inicio**.



Recuerda estos 10 pasos para instalar y comprobar el entorno de desarrollo de Arduino:

1. Obtener una placa Arduino con microcontrolador ATmega
2. Descargar de la página web oficial el entorno de desarrollo Arduino
3. Instalar el IDE y los Drivers USB
4. Conectar la placa Arduino al PC mediante el puerto USB
5. Observar que se enciende el LED verde de POWER, cuando conectamos la placa Arduino
6. Ejecutar la aplicación del entorno de programación de Arduino IDE
7. Cargar como ejemplo básico el programa "Blink".
8. Verificar y ejecutar el programa
9. Ver que se transmiten los datos encendiéndose los Leds L, Rx y Tx
10. Ver que parpadea el LED naranja L del pin D13 cada 1 segundo.

8. Lenguaje de programación de Arduino

El entorno de desarrollo de Arduino emplea un lenguaje de programación especial, donde podemos decir que se basa en la sintaxis de otros programas, como pueden ser C y C++. Aún y así, este lenguaje que utiliza Arduino, posee sus características propias, orientado a una fácil programación de los sensores y dispositivos externos. Por lo tanto, no es un C++ puro sino que es una adaptación proveniente de avr-libc que provee de una librería de C de alta calidad para usar con GCC en los microcontroladores AVR de Atmel.

8.1. Estructura básica de programación

La estructura de lenguaje de programación de Arduino es simple y se compone de dos partes o funciones que encierran bloques de declaraciones: **void setup()** y **void loop()**

```
void setup()
{
Declaraciones;
}
void loop()
{
Declaraciones;
}
/* Ambas funciones son requeridas para que el programa funcione*/
```

void setup()

Constituye la preparación del programa. Contiene la declaración de cualquier variable al comienzo del programa. Es la primera función, se ejecuta una vez y es usada para asignarles los pines de entrada y salida **pinMode()** o inicializar las comunicaciones en serie.

```
void setup()
{
pinMode(pin, OUTPUT; // ajusta "pin" como salida
}
```

void loop()

Contiene el código que se ejecuta continuamente y de forma cíclica leyendo entradas, activando salidas de la placa, etc. Esta función es el núcleo de todos los programas Arduino y hace la mayor parte de trabajo.

```
void loop()
{
digitalWrite(pin, HIGH);           // Activa "pin"
delay(1000);                       // espera un segundo
digitalWrite (pin, LOW);           // desactiva "pin"
delay(1000);                       // espera un segundo
}
```

Las llaves { }

Definen el comienzo y el final de bloques de función y bloques de declaraciones como voidloop() y sentencias for o if. Las llaves deben estar balanceadas. Una llave de apertura "{" siempre debe ir seguida de una llave de cierre "}". Las llaves no balanceadas provocan errores de compilación.

```
void loop()
{
Declaraciones;
}
```

8. Lenguaje de programación de Arduino

El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo tienes que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

El punto y coma ;

Debe usarse al final de cada declaración y separa los elementos del programa. También se usa para separar los elementos en un bucle for.

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

NOTA: Olvidar un punto y coma al final de una declaración producirá un error de compilación.

Bloques de comentarios /*...*/

Se usan para escribir comentarios multilinea o área de texto ignorados por el programa y se usan para grandes descripciones de códigos o comentarios que ayudan a otros programadores a entender partes del programa. Empieza con /* y termina con */ y puede abarcar múltiples líneas, e incluso nos pueden servir para localizar errores de programación troceando partes del programa y dejarlo en comentarios, para de esta forma detectar donde se produce los errores de compilación y resolver el problema.

```
/*Este es un bloque de comentario  
Tiene que estar balanceado.*/
```

Los comentarios de línea //

Sirve para escribir un comentario en la misma línea y termina con la siguiente línea de código. Se utiliza para proporcionar información acerca de lo que hace esa línea de instrucción o para recordarla más adelante y para otros programadores. No olvides que son dos barras //, si dejas una sola te da error de compilación.

```
{  
declaración; // Esto es un comentario de línea  
}
```

NOTA: En programación no utilizar las variables y nombres acentuados. Respetar los códigos que deben estar escritos en minúsculas y los que van en mayúsculas, pues el compilador tiene en cuenta estas diferencias y da error de compilación. El entorno de programación de Arduino facilita la interpretación del código que está bien escrito cambiando el Font de color naranja. Cualquier omisión de un punto y coma, llaves no balanceadas, falta de paréntesis, un código incorrectamente escrito o variables no declaradas, estos producen errores de compilación.

8.2. Funciones

Una función es un bloque de código que tiene un nombre y un grupo de declaraciones que se ejecutan cuando se llama a la función. Podemos hacer uso de funciones integradas como **void setup()** y **void loop()** o escribir nuevas.

8. Lenguaje de programación de Arduino

Las funciones se escriben para ejecutar tareas repetitivas y reducir el desorden en un programa.

En primer lugar se declara el tipo de la función, que será el valor retornado por la función (int, void...). A continuación del tipo, se declara el nombre de la función y, entre paréntesis, los parámetros que se pasan a la función.

```
Tipo funcionNombre (parámetros)
{
Declaraciones;
}
```

La siguiente función `int delayVal()`, asigna un valor de retardo en un programa por lectura del valor de un potenciómetro.

```
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;        //devuelve el valor final de v
}
```

8.3. Variables

Una variable es un pequeño contenedor de memoria que se emplea para almacenar datos, ya sean letras, números o una combinación de ambos. Es una forma de llamar y almacenar un valor alfanumérico para usarse después por el programa. Como su nombre indica, las variables son valores que pueden cambiarse continuamente, al contrario que las constantes, cuyo valor nunca cambia. Una variable necesita ser declarada y, opcionalmente, asignada al valor que necesita para ser almacenada.

```
int inputVariable=0; //declara una variable y asigna el valor a 0
inputVariable=analogRead(2); // ajusta la variable al valor del pin
analógico 2
```

Una vez que una variable ha sido asignada, o reasignada puedes testear su valor para ver si cumple ciertas condiciones o puedes usarlo directamente.

```
if(inputVariable) < 100) { // comprueba si la variable es menor que 100
inputVariable = 100;      // si es cierto asigna el valor 100
}
delay(inputVariable);     // usa la variable como retardo
```

Declaración de variables

Todas las variables tienen que ser declaradas antes de que puedan ser usadas por el programa y opcionalmente le podemos asignar un valor inicial. Si no la declaramos el programa nos dará un error. Declarar una variable significa definir su tipo de valor, como int, long, float, etc., definir un nombre específico y, opcionalmente, asignar un valor inicial. Esto sólo necesita hacerse una vez en un programa pero el valor puede cambiarse en cualquier momento usando aritmética y varias asignaciones.

8. Lenguaje de programación de Arduino

```
int inputVariable = 0;
```

Una variable puede ser declarada en un número de posiciones en todo el programa y donde esta definición tiene lugar determina que partes del programa pueden usar la variable.

Utilización de una variable

La variable puede ser declarada al comienzo del programa antes de **void setup()**, localmente dentro de funciones, y algunas veces en un bloque de declaraciones, por ejemplo, bucles **for**. Donde la variable es declarada determina el ámbito de la variable, o la habilidad de ciertas partes de un programa de hacer uso de la variable.

Una variable local es una que se define dentro de una función o como parte de un bucle **for**. Sólo es visible y sólo puede ser usada dentro de la función en la cual fue declarada. Además, es posible tener dos o más variables del mismo nombre en diferentes partes del programa que contienen diferentes valores.

```
int value;           // "value" es visible por cualquier función
void setup() { }    // no hay configuraciones setup

void loop() {
for (int i=0; i<20) { // "i" es solo visible dentro del bucle for
i++;
}
float f;           // "f" es solo visible dentro de loop
}
```

NOTA: En el entorno de desarrollo de Arduino, cuando introduces una instrucción la misma aplicación te indica si es incorrecto, cuando no enciende el color de la fuente del texto.

8.4. Constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados **constantas**. A diferencia de las variables, que pueden cambiar a cada momento según se determine en el programa, un dato constante no cambiará jamás su valor. La constante va a adquirir un único valor que no va a poder ser modificado durante la evolución del programa y se clasifican en grupos.

TRUE/FALSE (Cierto/Falso)

Estas son constantes booleanas que definen los niveles **HIGH** (alto) y **LOW** (bajo) cuando estos se refieren al estado de las salidas digitales. **FALSE** se asocia con 0 (cero) mientras **TRUE** se asocia con 1 (uno). Pero **TRUE** también puede ser cualquier otra cosa excepto cero. Por lo tanto en sentido booleano -1, 2 y -200 son todos también y se define como **TRUE** (esto es importante tenerlo en cuenta).

```
if (b == TRUE) { //si b es igual a verdadero
instrucciones; //ejecuta las instrucciones
}
```

8. Lenguaje de programación de Arduino

HIGH/LOW (Alto/Bajo)

Estas constantes definen los niveles de los pines con HIGH o LOW y son empleados cuando se leen o escriben en las entradas o salidas digitales. **HIGH** se define como el nivel lógico 1 (ON) o 5 V. **LOW** es el nivel lógico 0, OFF, o 0V.

```
digitalWrite(13, HIGH); // activa la salida 13 con un nivel alto 5V
```

INPUT/OUTPUT (Entrada/salida)

Estas constantes son empleadas para definir al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción pinMode() de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT. Se definen en el void setup().

```
pinMode(13, OUTPUT); // asignamos el pin 13 de salida
```

8.5. Tipos de datos

Arduino permite manejar los siguientes tipos de datos:

byte

Almacena un valor numérico de 8 bits sin decimales. Tienen un rango de 0 a 255.

```
byte unaVariable = 180; // declara "unaVariable" como tipo byte
```

NOTA: Bit es el acrónimo de Binary Digit (Dígito Binario). Como ya hemos mencionado, en un sistema binario únicamente se usan dos dígitos (0,1) para representar un número y en digital suele asociarse a dos estados, encendido y apagado.

Cuando decimos que una entrada de Arduino es de 8 o 10 bits a lo que nos estamos refiriendo es al número de dígitos de los que disponemos para representar un determinado valor. Es decir, por ejemplo, 10 bits serían 10 posibles dígitos: Y Y Y Y Y Y Y Y Y Y en binario, cada uno de estos dígitos puede tener dos posibles valores o estados (0,1).

int

Almacena un valor entero de 16 bits sin decimales con un rango comprendido entre 32,767 a -32,768.

```
int unaVariable = 1500;
```

long

Valor entero almacenado en 32 bits sin decimales que se encuentra dentro del rango -2147483648 a 2147483647.

```
long unaVariable = 90000; // declara "unaVariable" como tipo long
```

El formato de variable numérica de tipo "long" se refiere a números enteros.

8. Lenguaje de programación de Arduino

float

El formato de dato del tipo coma flotante “float” se aplica a los números con decimales. Los números con coma flotante tienen una mayor resolución almacenado en 32 bits con un rango de 3.4028235E+38 a 3.40282235+38.

```
float unaVariable = 3.14; // declara “unaVariable como tipo flotante
```

NOTA: Los números con coma flotante no son exactos y pueden producir resultados extraños en las comparaciones. Los cálculos matemáticos de coma flotante son también mucho más lentos que los del tipo de números enteros, por lo que debe evitarse su uso si es posible.

Array

Se trata de un conjunto de valores a los que se accede con un número índice (el primer valor del índice es 0). Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número de índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir, el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado.

```
int miArray() = (valor0, valor1, valor2 ...)
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición específica.

```
int miArray(5); //declara una array de enteros de 6 posiciones  
miArray(3) = 10; // asigna 1 valor 10 a la posición 4
```

Para leer un array basta con escribir el nombre y la posición a leer:

```
x = miArray(3); // x ahora es igual a 10 que está en la posición 3 del array
```

Las matrices se utilizan a menudo para instrucciones de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un LED.

Utilizando un bucle tipo **for**, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del **array parpadeo()** en este caso 180, que se envía a la salida analógica tipo **PWM** configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice “i”.

```
int ledPin = 10; // led en el pin 10  
byte parpadeo() = (180, 30, 255, 200, 10, 90, 150, 60); // array de 8  
valores  
void setup() {  
  pinMode(ledpin, OUTPUT); //configura la salida  
}  
void loop() {  
  for (int; i=7; i<7; i++) {  
    analogWrite(ledPin, parpadeo(i)); // escribe los intervalos en el ledPin  
    delay(200); // espera 200 milisegundos } }
```

8. Lenguaje de programación de Arduino

8.6. Funciones de E/S Digitales

Ya hemos visto que la plataforma Arduino posee una serie de entradas y salidas para comunicarse con el exterior bien de forma analógica o bien de forma digital. Antes de comenzar a describir nuestro programa, es necesario configurar estos pines en la manera en la que vayan a ser usados, ya que ningún pin puede usarse al mismo tiempo como entrada y salida. En el ATmega328P de Arduino posee 14 pines digitales de entradas y salidas: D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12 y D13. Para llevar a cabo la configuración usamos la instrucción **pinMode()**, señalando a continuación el número de pin seguido de cómo queremos que éste actúe: entrada (**INPUT**) o salida (**OUTPUT**) y lo haríamos en el apartado **void setup()**.

Estos pines trabajarán de forma binaria a través de dos estados: **HIGH** (alto) – Cuando toma un valor de 5 voltios, o **LOW** (bajo), asociado al valor de voltaje 0 voltios. Con la sentencia **digitalRead()** leeremos el estado de un pin almacenado como **HIGH** o como **LOW**. Con la sentencia **digitalWrite()** ponemos un pin de salida en **HIGH** o en **LOW**.

Función pinMode(pin,mode)

En las funciones de **void setup()** se configuran los pines específicos para la programación, estos pines se pueden configurar en el modo de entrada **INPUT** o como de salida **OUTPUT**.

```
pinMode(pin,OUTPUT); //declara "pin" como salida
```

Los pines configurados como **OUTPUT** se dicen que están en un estado de baja impedancia y pueden proporcionar 40 mA a otros dispositivos o circuitos para la activación y polarización de estos.

Los pines digitales de Arduino están ajustados a **INPUT** por defecto, por lo que no se necesitan ser declarados explícitamente como entradas con **pinMode()**. Los pines configurados como **INPUT** se dice que están en un estado de alta impedancia.

```
pinMode(pin, INPUT); //declara "pin" como entrada
digitalWrite (pin, HIGH); // activa la resistencia de pull-up
```

NOTA: Los cortocircuitos en los pines del microcontrolador o corriente excesiva pueden dañar o destruir el pin de salida e incluso dañar el chip ATmega328P. A menudo es buena idea conectar un pin **OUTPUT** a un dispositivo externo en serie con una resistencia de 470 Ohmios.

Las instrucciones **digitalRead(pin)** y **digital Write(pin)** leen y escriben el valor desde un pin.

Función digitalRead(pin)

Lee el valor desde un pin digital especificado con el resultado **HIGH** o **LOW**. El pin puede ser especificado o como una variable o como una constante (0 – 13).

```
value = digitalRead(pin); // declara "value" igual al pin leído
```

8. Lenguaje de programación de Arduino

Función `digitalWrite(pin, value)`

Introduce un nivel lógico alto HIGH o bajo LOW (activa o desactiva) en el pin digital especificado. El pin puede ser especificado como una variable o constante (0 -13).

```
digitalWrite(pin, HIGH); // declara "pin" a HIGH
```

```
/* ejemplo de programa */

int led = 13; // conecta " led" al pin 13
int pin = 7; //conecta "pushbutton" al pin 7
int value = 0; //variable para almacenar el valor leído
void setup()
{
pinMode(led, OUTPUT); // ajusta el pin 13 como salida
pinMode(pin, INPUT); // ajusta el pin 7 como entrada
}
void loop()
{
value=digitalRead(pin); // ajusta "value" igual al pin de entrada
digitalWrite(led, value); // ajusta "led" al valor del botón
}
}
```

8.7. Funciones de E/S Analógicas

A veces no es suficiente con captar o no una señal para poder activar o desactivar las cosas, sino que necesitamos cuantificar magnitudes reales para que Arduino responda en proporción. Esto nos lo facilitarán las entradas y salidas analógicas. En el microcontrolador Arduino Duemilanove ATmega328P, se disponen de 6 pines de entradas y salidas analógicas, estas son: AN0, AN1, AN2, AN3, AN4 y AN5.

Por defecto, todos los pines analógicos son de entrada INPUT y no necesitan ser declarados como tales. La lectura con la función **analogRead()** leeremos un determinado pin analógico almacenando un valor de 10 bits. Es decir, vamos a tener un rango de 1024 valores distintos en los que se van a poder obtener lecturas analógicas.

Función `analogRead(pin)`

Lee el valor desde el pin analógico especificado con una resolución de 10 bits. Esta función solo funciona en los pines analógicos (**AN0-AN1-AN2-AN3-AN4-AN5**). El valor resultante es un entero de 0 a 1023. Los pines analógicos, a diferencia de los digitales no necesitan declararse previamente como **INPUT** o **OUTPUT**.

```
value=analogRead(pin); // ajusta "value" igual a "pin"
```

8. Lenguaje de programación de Arduino

Función `analogWrite(pin, value)`

Escribe un valor pseudo analógico, usando modulación por ancho de pulso (“**PWM**”) a un pin de salida marcado como PWM. En los Arduinos más nuevos con el chip Atmega328P, esta función trabaja en los pines D3, D5, D6, D9, D10 y D11. Los Arduinos más antiguos con un ATmega8 sólo soportan los pines D9, D10 y D11. El valor puede ser especificado como una variable o constante con un valor de 0 a 255.

```
analogWrite(pin, value); //escribe “value” al “pin” analógico
```

`AnalogReference()`

Configura el voltaje de referencia usado por la entrada analógica. La función `analogRead()` devolverá un valor de 1023 para aquella tensión de entrada que sea igual a la tensión de referencia. Las opciones son:

- **DEFAULT**. Generalmente entre 5V y 3,3V.
- **INTERNAL**. 1,1 V o 2,56V
- **EXTERNAL**. Se usará una tensión de referencia externa que tendrá que ser conectada al pin **AREF**. Importante: este voltaje debe estar comprendido entre 0 y 5 Volts DC.

`AnalogReference(EXTERNAL)`

Es necesario llamar esta función antes que se lean los valores de voltaje utilizando `analogReference()` ya que de lo contrario podrías dañar la placa Arduino.

Para capturar la medida analógica, el Arduino posee seis entradas analógicas donde cada una está conectada a un canal del conversor. Así, se podría utilizar un pin de entrada analógica del Arduino para leer el voltaje que proporciona el sensor de temperatura LM35.

Las salidas analógicas están asociadas a los pines **PWM**. En realidad son pines digitales y no se hace entrega de una señal analógica pura, sino que se entrega un determinado valor de tensión a través de complicados procesos de modulación que no vamos a entrar en describir. Las salidas digitales trabajan con 8 bits, es decir, 256 valores diferentes (del 0 al 255). Por ello hay que tener en cuenta que la lectura puede realizarse en 10 bits, pero la escritura va a tener que traducirse a 8 bits.

Para leer este voltaje se utiliza una función que hemos visto anteriormente `analogRead()` que proporciona un rango de valores comprendidos entre 0 y 1023. Como ya sabemos, este rango proviene de la resolución de bits en el conversor del microcontrolador. Pero estos valores están tomados en referencia a 5 volts.

8.8. Función `map()`

Podemos encontrarnos que, en ocasiones, deseemos acotar los valores analógicos que leemos de un sensor analógico que comprende de 0 a 1.023, por ejemplo, quizás nos interesa que el valor máximo en vez de ser 1.023 fuese 500, y que el valor mínimo en vez de ser 0 fuese 100.

Arduino cuenta entre sus instrucciones de programación con la función `map()`, que permite mapear valores y adecuarlos a nuestras necesidades.

8. Lenguaje de programación de Arduino

La función **map()** permite adecuar los valores que proporcionan algunos sensores a otro rango de valores más apropiados para nuestros intereses. Por ejemplo, si diseñamos un circuito con una fotorresistencia LDR y deseamos observar los valores que obtenemos en función de la luz que capta, comprobaremos que estos valores están situados entre un rango de 0 a 1.023, al ser un sensor analógico.

Por el contrario, deseamos que estos valores, por motivo que sea, estén en un rango comprendido entre 0 y 255, que son los valores que obtenemos para valores digitales de 8 bits. Esto es posible con la función `map()`. La sintaxis de esta función es la siguiente:

`map (valor, origen_menor, origen_mayor, destino_menor, destino_mayor)`

Ejemplo de la función `map()`:

```
/* mapeando el valor de una LDR */
int ldr=A0; // pin de conexión LDR analogico
int valor=0; // almacena valor LDR
void setup() {
  Serial.begin (9600); // iniciamos la comunicación serie
  pinMode(ldr, INPUT); // declaramos pin LDR entrada
}
void loop() {
  valor=analogRead(ldr); // lee valor LDR y alacena en valor
  map(valor,0,1023,0,255); //establecemos valores origen-destino minimo-maximo
  if (valor==255){// si el valor de la ldr es igual a 255
    Serial.println("LUZ MAXIMA"); // mensaje de aviso
  }
}
```

8.9. Funciones de tiempo y matemáticas

Gestión del tiempo en Arduino

En el lenguaje de programación de Arduino existen una series de funciones relacionadas con el tiempo que nos permiten aplicar tiempos en la ejecución de instrucciones.

Estas son:

- **delay(ms)**. Esta función detiene la ejecución del programa durante un tiempo determinado. Durante este tiempo Arduino no detectará eventos como presionar un interruptor, activar un sensor, etc. Otro aspecto importante a tener en cuenta es que Arduino mide el tiempo en milisegundos. 1 segundo es igual a 1000 milisegundos.

```
delay(milisegundos); // realiza una pausa en el programa de milisegundos
```

dónde:

Milisegundos: es el tiempo que va a esperar hasta pasar a la siguiente instrucción de nuestro programa.

Por ejemplo:

```
delay(1000); // realiza una pausa en el programa de 1 segundo
```

8. Lenguaje de programación de Arduino

La función **delay** hará esperar 1 segundo al microcontrolador antes de pasar a la siguiente instrucción. Estas instrucciones, al estar en el bloque **void loop()**, se repetirán continuamente.

NOTA: El inconveniente que aparece con esta función es que el microcontrolador se detiene durante el tiempo, en milisegundos, que introducimos. Esto hará que se pierda un tiempo excesivo, durante el cual el microcontrolador no atiende a ninguna otra orden mientras se esté ejecutando esta instrucción: un sensor que cambia, un botón que se active, etc.

- **millis()**. La función `millis()` actúa de contador en el momento que es activada, por lo que contará en milisegundos desde el momento que es activada hasta aproximadamente 50 días.

Arduino tiene un reloj interno que va a ir contando los milisegundos desde que la placa se conecte a la corriente eléctrica y el programa se inicie. Arduino puede contar hasta casi 50 días, cuando el tiempo volvería a contar desde cero. Es decir, devuelve la cantidad de milisegundos que lleva la placa Arduino ejecutando el programa actual como un valor **long unsigned**. Después de las 9 horas el contador vuelve a cero.

Ejemplo:

```
unsigned long tiempo; // variable unsigned long
void setup() {}
void loop() {
tiempo = millis();//la función se activa y se guarda en la variable tiempo
}
```

- **micros ()**. La función `micros ()` también actúa de contador en el momento que es activada, por lo que contará en microsegundos desde el momento que es activada hasta unos 70 minutos.

Para que la función **micros()** devuelva el tiempo contabilizado y los almacene en una variable, ésta también debe ser del tipo `unsigned long`.

El ejemplo anterior para la **función millis()** también es aplicable para la **función micros()**.

```
unsigned long tiempo; // variable unsigned long
void setup() {}
void loop() {
tiempo = micros(); // la función se activa y se guarda en la variable tiempo
}
```

- **delaymicroseconds()**. Esta función responde de igual forma que la función `delay()`, sólo que entre paréntesis introduciremos el tiempo en microsegundos y no en milisegundos.

Funciones de matemáticas

MIN / MAX Son funciones que comparan dos valores devolviendo el menor o el mayor de ellos:

8. Lenguaje de programación de Arduino

- **min(x,y).** Calcula el mínimo de dos números para cualquier tipo de datos devolviendo el número más pequeño.

```
valor = min(valor, 100); //asigna a valor el más pequeño de los dos números
```

Si valor es menor que 100 valor recogerá su propio valor, si valor es mayor de 100 valor pasara a valer 100.

- **max(x,y).** Calcula el máximo de dos números para cualquier tipo de datos devolviendo el número mayor de los dos.

```
valor = max(valor,100); //asigna a valor el mayor de los dos números valor100
```

De esta manera nos aseguramos de que **valor** será como mínimo **100**.

8.10. Sentencias condicionales (Control de flujo)

El lenguaje de Arduino permite realizar sentencias condicionales: **if**, **if... else**, **for**, **while**, **do... while**. Su utilización es similar a las funciones correspondientes en el lenguaje C. Sirven para guiar el programa en una u otra dirección en función de si se cumple o no una serie de condiciones que establecemos en el programa, están las **condicionales**, los **bucles** y **control de flujo**.

Las **condicionales** chequean una condición y si se cumple, se ejecutan las instrucciones englobadas dentro de la condición. Estas sentencias son:

- **if... :** Si se cumple se ejecutan las sentencias del bloque que se encuentra entre llaves. Si no se cumple el programa salta este bloque sin ejecutar instrucción alguna.
- **if...else:** Funciona prácticamente igual que la anterior, pero si no se cumple la condición, no se salta el bloque, sino que ejecuta las instrucciones del bloque “**else**”.

Los **bucles** son elementos que hacen que el programa entre en un ciclo de repetición mientras se cumplan una serie de condiciones:

- **for:** repite un bloque de sentencias mientras se cumpla una condición.
- **while:** repite las instrucciones entre llaves mientras se esté cumpliendo la expresión incluida en el bucle.
- **do...while:** funciona igual que “**while**” pero ejecuta las instrucciones al menos una vez, ya que comprueba si se cumplen las condiciones al final.

Los elementos de **control de flujo** se encuentran las siguientes sentencias:

- **goto:** Para realizar un salto a una parte del programa que esté marcada con la etiqueta correspondiente.
- **return:** En el momento que el programa lee esta sentencia, éste vuelve a la posición desde la que se realizó el último salto “**goto**”.
- **break:** Se rompe el bucle y el programa sale de el sin tener en cuenta si se cumplen o no las condiciones.

NOTA: Goto y Return no se suelen utilizar mucho en la programación de Arduino. Para ello, se puede utilizar creando un bloque de instrucciones y llamarla desde cualquier parte del programa.

8. Lenguaje de programación de Arduino

Cuando se llama a una función ésta se ejecuta y una vez que termina la función vuelve el flujo del programa a donde se llamó a la función. Ver el apartado 7.10 “Crear nuestras propias funciones”.

if (si condicional)

if es una declaración que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves.

El formato para **if** es el siguiente:

```
if (A==3){ // si A es igual a 3, haz lo siguiente...
A=A+5; // A almacena el resultado de 3+5
C=5+4; // C almacena el resultado de 5+4
B=A+C; // B almacena el resultado de sumar A+C
}
C=4; // si la condición no se cumple (A=3), C almacena un 4
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de las llaves se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

NOTA: Tenga en cuenta el uso especial del símbolo “=”, poner dentro de **if** ($x=10$), podría parecer que es válido pero sin embargo no lo es, ya que esa expresión asigna el valor 10 a la variable x , por eso dentro de la estructura **if** se utilizaría $X==10$ que en este caso lo que hace el programa es comprobar si el valor de x es 10. Ambas cosas son distintas por lo tanto dentro de las estructuras **if**, cuando se pregunte por un valor se debe poner el signo doble de igual “==”.

if ... else (si ... entonces)

El bucle **if ...else, si/entonces**, ejecuta unas instrucciones de manera condicional. La función tiene tres enganches de piezas. La primera es la condición, la segunda es la instrucción que se ejecuta si la condición se cumple y la tercera es la instrucción que se ejecutará si la condición no se cumple.

Viene a ser una estructura que se ejecuta en respuesta a la idea “si esto no se cumple haz esto otro”. Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alta o hacer otra cosa si la entrada es baja, se escribirá de la siguiente forma:

```
if (inputPin == HIGH){
Instrucciones;
}
else {
Instrucciones;
}
```

8. Lenguaje de programación de Arduino

else puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez. Es incluso posible tener un número ilimitado de estos condicionales. Recuerde sin embargo que solo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada.

Ejemplo:

```
if (inputPin < 500) { //si input Pin es menor de 500
Instrucciones; //ejecuta la siguientes instrucciones
}
else if (inputPin>= 1000) { //de lo contrario inputPin es mayor o igual a 500
Instrucciones;//ejecuta las siguientes instrucciones
}
else { //de lo contrario
Instrucciones; //ejecuta las siguientes instrucciones
}
```

NOTA: Una declaración del tipo if prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración válida. En el anterior ejemplo, si cambiamos y ponemos (inputPin == HIGH). En este caso, la declaración if solo chequearía si la entrada especificado está en nivel alto (HIGH), o +5V.

for

La declaración **for** se usa **para** repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración for tiene tres partes separadas por (;).

```
for (inicialización; condición; expresión) {
Instrucciones;
}
```

La inicialización de una variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

El siguiente ejemplo inicia el entero "a" en el 0, y la condición es probar que el valor es inferior a 20 y si es cierta "a" se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves hasta que se cumpla que "a" es mayor de 20 que finaliza el bucle:

```
for (int a=0; a<20; a++)//declara a y prueba si es menor que 20, incrementaa
{
digitalWrite(13, HIGH); // enciende el pin 13
delay(250); // espera un cuarto de segundo
digitalWrite(13, LOW); // apaga el pin 13
delay(250); // espera un cuarto de segundo
}
```

8. Lenguaje de programación de Arduino

El código anterior es equivalente al siguiente:

```
int a=0;
while(a<20){
Códigos
a++
}
```

NOTA: El bucle en el lenguaje C es mucho más flexible que otros bucles encontrados en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier declaración válida en lenguaje C sin relación con las variables declaradas. Estos tipos de estados son raros pero permiten disponer de soluciones a algunos problemas de programación raras.

while

Un bucle del tipo **while** es un bucle de ejecución continua mientras se cumpla la expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro del código del bucle o también por el cambio de un valor en una entrada de un sensor.

```
while (unaVariable ? valor)
{
Ejecutarsentencias;
}
```

El siguiente ejemplo comprueba si la variable “una Variable” es inferior a 200 y, si es verdad, ejecuta las declaraciones dentro de la llaves y continuará ejecutando el bucle hasta que “unaVariable” no se inferior a 200.

```
while (unaVariable<200) // comprueba si es menor que 200
{
Instrucciones; // ejecuta las instrucciones entre llaves
unaVariable++; // incrementa la variable en 1
}
```

do ... while

El bucle **dowhile** funciona de la misma manera que el bucle **while**, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutará al menos una vez.

```
do {
Instrucciones; //ejecuta las instrucciones
}
While (unaVariable ? valor); //mientras unaVariable sea ¿? valor
```

El siguiente ejemplo asigna el valor leído **leeSensor()** a la variable “x”, espera 50 milisegundos y luego continua mientras que el valor de la “x” sea inferior a 100.

8. Lenguaje de programación de Arduino

```
do {  
x=leeSensor(); //guarda en x el valor leído de leeSensor  
delay(50); //espera 50 milisegundos  
}  
while (x<100); //mientras x sea menor de 100
```

switch/case y break

Estas dos sentencias (**SWITCH/CASE** y **BREAK**) son un complemento la una de la otra, al igual que ocurre con los mandatos **if** y **else**.

Compara el valor de una variable con el valor específico en las sentencias cuyo valor coincide con dicha variable, el código de esa sentencia se ejecuta.

```
switch(variable){  
Case 1:  
Código usado "variable" es igual 1  
break;  
Case 2:  
Código cuando variable es igual 2  
break;  
default;  
Código ejecutado cuando ninguno de la sentencias se cumple}
```

break

Esta instrucción es usada para salir de los bucles de **for** o **while**, pasando por alto la condición normal del bucle. Es usado también para salir de una estructura de control **switch**, como hemos visto en el anterior apartado.

```
while (a>b){  
Código;  
if(a==5){  
break  
}  
}
```

8.11. Crear nuestras propias funciones

Hasta ahora se han estado analizando funciones predefinidas para la programación de la placa Arduino. Nosotros mismo podemos desarrollar nuestras propias funciones para mejorar el código de un sketch.

En muchas ocasiones, después de plasmar las ideas de un proyecto y transcribir el código en el IDE de Arduino, y tras realizar toda una serie de cambios, pruebas y modificaciones dentro del propio sketch, podría ser que el código se acabe asemejando más a un maremágnum de instrucciones y números que a un programa bien estructurado, ocasionando que al cabo de un tiempo pueda resultarnos complicado volver a comprender el código que se había escrito.

8. Lenguaje de programación de Arduino

Un modo de evitarlo es establecer el código en bloques o partes perfectamente diferenciadas. Esto reposta unas ventajas a la hora de programar:

- Mejor compresión del código.
- El código se convierte en escalable. Esto quiere decir que si se necesita añadir algunas líneas más, podremos hacerlo sin apenas complicaciones al estar debidamente estructurado.
- El código se ve más claro y limpio, permitiendo deshacernos de posibles variables o procesos que no necesita nuestro programa.

Para realizar esto podemos crear nuestras propias funciones, que serán llamadas por el programa principal en el mismo sketch, pero que nos dará una visión más limpia del código, más clarificadora y apta para futuras modificaciones. Veamos cómo implementarlo. Para crear una función en un sketch de Arduino escribimos:

```
void nombre_de_la_funcion(){
Instrucciones que deseamos que realice la función;
}
```

Para invocar la función simplemente escribimos, allí donde deseamos que se ejecute ese bloque de código:

```
Nombre_función(); //nombre de la nueva función
```

Veamos un ejemplo para esclarecer los conceptos:

```
int nivel=300;          //ponemos la variable nivel a 300

void setup(){
Serial.begin(9600); //establece visualizacion por puerto serial
}
void monitoriza(){//Creacion de la funcion monitoriza para configurar valor
dia/noche
Serial.print("El valor es:"); // texto indicativo de linea serial
Serial.println(medida); // pone en linea el valor de medida
delay(1000); // para evitar saturar el puerto
}
void loop(){
monitoriza(); // llama y ejecuta el bloque de instrucciones monitoriza
medida=analogRead(ldr); // tomamos lectura de la ldr
if (medida<nivel){// condiciona que si la medida es menor que 300
digitalWrite(led,HIGH);
}
}
```

Este es un fragmento del código que realiza la medición del nivel de luz de una fotocélula LDR y se visualiza por medio del Monitor Serial.

En este fragmento podemos ver las invocaciones de las diferentes instrucciones de Serial llamado por la función creada **monitoriza()**.

8. Lenguaje de programación de Arduino

Hay que recordar que la invocación de la función se puede realizar dentro del **void loop()**, con lo que repetirá dicha invocación dentro del **loop** o dentro del **void setup()**, la cual se invocará una sola vez, es decir, cuando se lean las instrucciones del bloque **setup**.

Las funciones se pueden situar al final del programa, después del claudator que cierra el **voidloop()**.

En resumen las llamadas a las funciones las podemos ubicar tanto en el **setup** (sólo se las llamará una vez) como en el **loop** que serán llamadas repetidamente.

8.12. Operadores aritméticos

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente (respectivamente) de dos operandos.

$$y = y + 3; \quad x = x - 7; \quad i = j * 6; \quad r = r / 5;$$

La operación se efectúa teniendo en cuenta el tipo de datos que hemos definido para los operandos (int, dbl, float, etc...) por lo que, por ejemplo, si definimos 9 y 4 como enteros "int", 9/4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 son valores de tipo entero "int" (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos.

Si los operandos son de diferentes tipos, para el cálculo se utilizará el tipo más grande de los operandos en juego. Por ejemplo, si uno de los números (operandos) es del tipo **float** y otra de tipo **integer**, para el cálculo se utilizará el método de **float** es decir el método de coma flotante.

Escoja el tamaño de las variables de tal forma que sea lo suficientemente grande como para que los resultados sean los precisos que se desea conseguir. Para las operaciones que requieran decimales utilice variables tipo float, pero tenga en cuenta de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el computo.

NOTA: Utilice el operador "int" para convertir un tipo de variable a otra sobre la marcha. Por ejemplo, `i=(int) 3,6` establecerá `i` igual a 3.

Asignaciones compuestas

Empleando variables, valores constantes o componentes de un array pueden utilizarse operaciones aritméticas y se pueden utilizar el operador cast para conversión de tipos. Ej. **int a =(int)3.5;** Además pueden hacerse las siguientes asignaciones compuestas:

- `x++` // Lo mismo que `x=x+1`.
- `x--` // Lo mismo que `x=x-1`.
- `x+=y` // Lo mismo que `x=x+y`
- `x-=y` // Lo mismo que `x=x-y`.
- `x*=y` // Lo mismo que `x=x*y`.
- `x/=y` // Lo mismo que `x=x/y`.

8. Lenguaje de programación de Arduino

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles.

Por ejemplo, `x *= 3` hace que `x` se convierta en el triple del antiguo valor `x...` y por lo tanto `x` es reasignada a nuevo valor.

Operadores de comparación

Para su utilización en sentencias condicionales u otras funciones Arduino permite utilizar los siguientes operadores de comparación:

- `x==y` // `x` es igual a `y`.
- `x!=y` // `x` no es igual a `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`

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo **if** para testear si una condición es verdadera.

Operadores lógicos

En el lenguaje de programación de Arduino, así como en cualquier otro lenguaje de programación (como Java) existen las llamadas estructuras de control.

La principal estructura de control, de la cual se derivan las demás es la estructura secuencial, donde todas las instrucciones se ejecutan una tras otra en orden descendente, de principio a fin.

Luego de la estructura secuencial, creo que la más utilizada es la estructura selectiva, el llamado `if`. Su funcionamiento es sencillo y su uso permite utilizar la lógica en la creación de algoritmos.

En ocasiones es necesaria que se cumpla más de una condición para lograr determinados resultados. Es aquí donde entran en juego los Operadores Lógicos.

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un VERDADERO o FALSO dependiendo del operador. Existen tres operadores lógicos:

1. **AND** (`&&`),
2. **OR** (`||`) y
3. **NOT** (`!`),

...que a menudo se utilizan en instrucciones de tipo **if**.

En programación se usa básicamente las estructuras AND y OR y a partir de ellas se derivan las demás.

8. Lenguaje de programación de Arduino

Lógica AND

Este operador permite validar 2 o más condiciones, las cuales se deben cumplir todas a la misma vez para que se ejecute el código que se escribe dentro de las llaves.

```
if (x > 0 && x < 5){...} //cierto solo si las dos expresiones son ciertas
```

Lógica OR

Este operador permite que cuando cualquiera de las condiciones establecidas se cumpla, entonces se ejecuta el código entre las llaves. Solamente se necesita que al menos 1 sea verdadera para que se devuelva un valor true y se ejecute las instrucciones establecidas.

```
if(x> 0 || y>0) {...} //cierto si una cualquiera de las expresiones es cierta
```

Lógica NAND

Este operador permite no validar cuando todas las condiciones establecidas se cumpla. Supongamos que hay 5 personas en una casa. Si se les pregunta si poseen un Arduino y NINGUNO de ellos tiene, entonces se ha cumplido una condición y el NAND devolverá un valor true.





Lógica NOR

De la misma forma como el NAND se deriva del operador AND, el NOR se deriva del OR y posee dos casos en los que puede decir que se utiliza. Se cumplirá cuando en algunas de las condiciones se devuelva un valor false.

Lógica NOT

```
if (!x > 0){...} // cierto solo si la expresión es falsa
```

En el siguiente cuadro se muestran los operadores más utilizados a la hora de programar.

Operador	Nomenclatura en Arduino	Significado	Símbolo
AND	&&	Devuelve true cuando la primera Y la segunda condición se cumplen	
OR		Devuelve true cuando la primera O la segunda condición se cumple	
NAND	(i=) && (i=)	Devuelve true cuando NI la primera NI la segunda condición se cumplen	
NOR	(i=) (i=)	Devuelve true cuando NO se cumple alguna de las condiciones	
NOT	!=	Devuelve true cuando NO se cumple una condición	

8. Lenguaje de programación de Arduino

Tabla resumen de los operadores:

OPERADOR	SÍMBOLO	OPERACIÓN
Paréntesis	()	Paréntesis
Aritméticos	**, * / DIV,\n%,mod + -	Potencia Producto División División entera Módulo(resto de la división entera) Signo positivo o suma Signo negativo o resta
Alfanuméricos	+ -	Concatenación Concatenación eliminando espacio
Relacionales	==,= !=,<> < <= > >=	Igual a Distinto a Menor que Menor o igual que Mayor que Mayor o igual que
Lógicos	!, NOT, no &&, AND, y , OR, o	Negación Conjunción Disyunción

8.13 Operadores aleatorios

Función randomSeed(seed)

Esta función nos permite inicializar, a partir de una variable o de otra función, una semilla para generar números aleatorios y como punto de partida para la función **random()**.

```
randomSeed(valor); // hace que valor sea la semilla del random
```

Por ejemplo:

randomSeed(millis) generará números aleatorios a partir del valor de la función **millis**.

Recordemos que esta función devuelve en **milisegundos** el tiempo desde que Arduino está ejecutando el programa actual.

Debido a que Arduino es incapaz de crear un verdadero número aleatorio, **randomSeed** le permite colocar una variable, constante u otra función de control dentro de la función **random**, lo que permite generar números aleatorios “al azar”. Hay una variedad de semillas o funciones, que pueden ser utilizados en esta función, incluido **millis()** o incluso **analogRead()** que permite leer ruido a través de un pin analógico.

8. Lenguaje de programación de Arduino

Función random: random(max) y random(min,max)

Para utilizar la función random, primero debemos emplear la función **randomSeed()**.

La función **random()** (aleatorio) genera números aleatorios en un rango de 0 a un máximo, o un rango preestablecido por el usuario con las variables **max** y **min**:

- **random (max)** devuelve un valor aleatorio entre 0 y max.
- **random (min, max)** devuelve un valor aleatorio entre min y max.

```
valor=random(100,200); //asigna a la variable valor un número aleatorio comprendido entre 100-200
```

El siguiente ejemplo genera un valor aleatorio entre 0-255 y lo envía a una salida analógica PWM:

```
int randomNumber; // variable que almacena el valor aleatorio
int led=10; // define led como 10
void setup(){ // No es necesario configurar nada
void loop() {
randomSeed(millis()); //genera una semilla para aleatorio a partir función millis
ranNumber = random(255); //genera numero aleatorio entre 0-255
analogWrite(led, ranNumber); //envía a la salida led de tipo PWM
delay(500); // espera 0,5 segundos
}
```

8.14 Función de Interrupción

Una interrupción la podemos definir como una llamada al microcontrolador, el cual, deja lo que está ejecutando y atiende dicha llamada. Esta llamada o interrupción, normalmente lleva al microcontrolador a otra parte del código que debe ejecutarse con mayor prioridad.

Una vez ejecutado ese bloque de código, el microcontrolador vuelve al punto anterior, es decir, a la línea de la instrucción donde lo había dejado antes de recibir la interrupción. Este tipo de interrupciones también las podemos llamar interrupciones hardware, porque es mediante componentes exteriores quienes ejecuta esta interrupción.

Las interrupciones son similares a una función; la diferencia estriba en que las funciones que hemos programado tenían que ser “llamadas” desde una parte del programa. Una interrupción no es más que una función que se ejecutará, no por ser “llamada”, sino porque se configura para que, ante un evento se ejecute.

Como se ha comentado, las interrupciones son funciones que se ejecutan por haber sido previamente configuradas; por lo tanto, es por donde tenemos que empezar. Este tipo de interrupciones necesitan configurar tres parámetros:

1. El pin en donde se puede producir el evento.
2. El nombre de la función que se ha de ejecutar si ocurre el evento.
3. El tipo de evento que ha de ocurrir.

8. Lenguaje de programación de Arduino

Antes de continuar, hay que revisar ciertas peculiaridades de este tipo de **interrupciones**:

1. No se pueden atender eventos en todos los pines de Arduino. Arduino posee dos pines para crear interrupciones. En este caso serán interrupciones externas, ya que las vamos a generar nosotros desde fuera del microcontrolador. Estas interrupciones son **INT0**, **INT1**, las cuales están vinculadas con los pines **D2** y **D3** respectivamente.
2. El nombre de la función que programamos tiene que respetar las mismas reglas que cualquier otra función. Como lo que, al fin y al cabo, lo que se va a ejecutar es una función, será necesario indicar el nombre de la función que se ha de ejecutar.
3. Tenemos una serie de palabras reservadas para indicar el tipo de interrupción:
 - **LOW**: si configuramos la interrupción con este parámetro, se ejecutará la función cuando el pin reciba una señal a nivel bajo.
 - **CHANGE**: si configuramos la interrupción con este parámetro, se ejecutará la función cuando el pin cambie de estado, tanto si es de 0 a 5 voltios (flanco de subida) como de 5 a 0 voltios (flanco de bajada).
 - **RISING**: si configuramos la interrupción con este parámetro, se ejecutará la función cuando el pin cambie de 0 a 5 voltios (flanco de subida).
 - **FALLING**: si configuramos la interrupción con este parámetro, se ejecutará la función cuando el pin cambie de 5 a 0 voltios (flanco de bajada).
 - **HIGH**: si configuramos la interrupción con este parámetro, se ejecutará la función cuando el pin reciba una señal a nivel alto. No obstante, el Arduino UNO no dispone de este tipo de evento.

El último parámetro es del tipo de evento, cuando recibimos una señal tenemos que saber que comportamiento de esta nos indica que hay que atender a un evento: señal a nivel alto, a nivel bajo, flanco de subidas, etc.

Por ejemplo, un evento crítico puede ser una parada de emergencia. Para que se ejecute una interrupción por evento externo, tenemos que tener una señal que nos indique que ha ocurrido algo, esa señal se ha de conectar a uno de los pines del Arduino, con lo que tenemos que indicar por programación en qué pin está conectada esa señal.

Esta interrupción se puede utilizar, por ejemplo, cuando hemos utilizado un **delay()**, durante el cual el microcontrolador queda exclusivamente ejecutando el tiempo sin atender ninguna otra orden programada. Con la interrupción rompería esa rutina, pues tiene prioridad.

Función **attachInterrupt(,,)**

Para desarrollar la **interrupción** con Arduino, tenemos una función que genera la interrupción deseada, la instrucción "**attachInterrupt(,,)**", que permite programar interrupciones por evento externo; necesita los tres parámetros que se han mencionado anteriormente, siguiendo el mismo orden y tiene la siguiente sintaxis:

attachInterrupt (nº int, nombre función, modo)

- **Nº int**: número de la interrupción. Si utilizamos la interrupción 0, el pin a utilizar será el D2. Si utilizamos la interrupción 1, el pin será el D3.
- **Nombre función**: nombre que le damos a la función interrupción que deseamos llamar.

8. Lenguaje de programación de Arduino

- **Modo:** Define cuando la interrupción deberá ser activada. Observamos 4 formas de activación:
 1. **CHANGE:** se activa cuando el valor en el pin pasa de HIGH a LOW o de LOW a HIGH. Es el modo más empleado.
 2. **LOW:** Se activa cuando el valor en el pin es LOW.
 3. **RISING:** Se activa cuando el valor del pin pasa de LOW a HIGH.
 4. **FALLING:** Se activa cuando el valor del pin pasa de HIGH a LOW.

Ejemplo de interrupción con la función `attachInterrupt`

```
/* encendido de un led mediante una interrupción */
int botón=2;
int led = 13;
int vb=0; // variable que almacena el valor del botón
void setup() {
  pinMode (botón, INPUT);
  pinMode (led, OUTPUT);
  Serial.begin (9600);
  attachInterrupt(0,parpadeo,HIGH);
}
void loop() {
  for (int t=0; t<500; t++)
  {
    Serial.println (t);
  }
}
```

Lo que tenemos que tener presente es que, si se ejecuta una interrupción, el programa saltará desde la línea de código en donde se encuentre, ejecutará las líneas de código que haya dentro de la función de la interrupción y volverá a la línea de programa desde donde ha saltado y continuará con la ejecución normal del programa.

Veamos el siguiente ejemplo de dos led que se encienden alternativamente sin detenerse, y aplicamos una interrupción hardware mediante un pulsador exterior para saltar a una línea de desconexión de los diodos Leds (apagados) durante un determinado tiempo.

```
/* interrupciones en Arduino */

int led1=7;//asignamos el pin 7 digital a led1
int led2=8;//asignamos el pin 8 digital a led2
voidsetup() {
  pinMode(2,INPUT);//configuramos el pin 2(0) de entrada
  pinMode(led1, OUTPUT);//configuramos el pin led1 de salida
  pinMode(led2, OUTPUT);//configuramos el pin led2 de salida
  attachInterrupt(0, desconectar, HIGH); //configuramos parametros
interrupcion
}
voidloop(){//creamos secuencia infinita del encendido de dos Leds
  digitalWrite(led1, HIGH);//encendemos led1
  delay(500);
  digitalWrite(led1, LOW);//apagamos led1
  digitalWrite(led2, HIGH);//encendemos led2
  delay(500);
  digitalWrite(led2, LOW);//apagamos led2
}
void desconectar(){//creamos función desconectar
```

8. Lenguaje de programación de Arduino

```
digitalWrite(led1, LOW); //apagamos led1
digitalWrite(led2, LOW); //apagamos led2
delay(1000000); //establecemos el tiempo de desconexion
}
```

8.15. Las bibliotecas

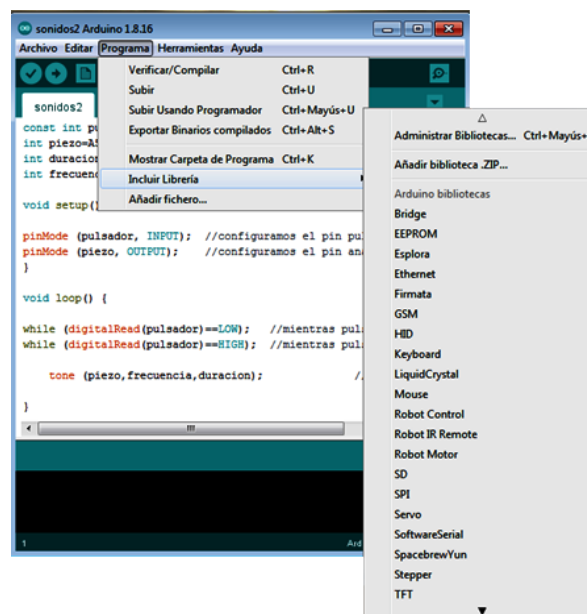
Las bibliotecas son una herramienta muy poderosa cuando se trabaja con Arduino, especialmente para principiantes. Una biblioteca es un archivo (un conjunto de archivos) que contiene exactamente el mismo código C++ en el que escribimos un sketch (a veces también hay inserciones de ensamblador). Podemos conectar la biblioteca a nuestro código y utilizar las capacidades que proporciona, y hay muchas opciones: “herramientas” listas para usar y para trabajar con sensores y módulos externos, para trabajar con los periféricos internos del microcontrolador (temporizadores, ADC, memoria), bibliotecas de diversas herramientas, matemáticas y es mucho más.

Lo bueno de trabajar con una biblioteca que no necesitamos saber cómo funciona el código que contiene, utilizamos herramientas listas para usar proporcionadas por el desarrollador de la biblioteca. Muy a menudo hay descripciones / documentación y ejemplos de uso de bibliotecas.

Asistente de bibliotecas

Como se ha comentado anteriormente una biblioteca es una colección de archivos de texto con código. La biblioteca se puede instalar de dos formas: desde el repositorio oficial o manualmente. Arduino tiene varias bibliotecas que se pueden obtener directamente del programa **Arduino IDE** utilizando el administrador de bibliotecas incorporado, que le permite instalar, eliminar y actualizar bibliotecas.

Esto es para iniciarse, porque esta lista no incluye todas las bibliotecas existentes y el administrador no proporciona una descripción normal. Para instalar una biblioteca del repositorio oficial de Arduino, vaya a **Programa/Incluir librería ...** Se abrirá el administrador de la biblioteca, en el que puede encontrar e instalar una biblioteca de la lista con un solo clic.



Incluir librería en el IDE de Arduino

8. Lenguaje de programación de Arduino

Dentro de la Biblioteca

La biblioteca, dependiendo de la cantidad de código y el estado de ánimo del programador, se puede diseñar de forma muy compacta y detallada, con un montón de archivos y carpetas adicionales. Consideremos la composición clásica de la biblioteca. Para mayor comodidad, recomiendo habilitar la visibilidad de las extensiones de archivo. Todos los siguientes archivos de muestra son archivos de texto ordinarios, puede abrirlos con un bloc de notas normal. También recomiendo usar el «bloc de notas de programador» – Notepad ++ (enlace al sitio oficial), que resalta la sintaxis y, en general, es una herramienta muy adecuada para el desarrollador.

El **<nombre de biblioteca>.h**: es un archivo de encabezado, el archivo de biblioteca más importante. Es tan importante que la biblioteca solo puede estar formada por él. Por lo general, se encuentra en la raíz de la biblioteca o en la carpeta **src**, **source** (fuente). Este archivo generalmente enumera todas las clases / métodos / funciones / tipos de datos, contiene información sobre la biblioteca, a menudo hay una descripción extendida para cada método o función. Muy a menudo, el archivo de encabezado principal es la mini documentación de la biblioteca.

Una biblioteca puede tener una estructura de varios archivos con una gran cantidad de archivos de encabezado, pero el archivo de encabezado principal siempre es uno, tiene el mismo nombre que la carpeta con la biblioteca.

Un archivo con la extensión **.cpp** es un archivo de implementación que contiene el código ejecutable principal del programa. Por lo general, va en parejo a su archivo **.h** de encabezado, es decir, **<nombre de la biblioteca>.cpp** .

keywords.txt: un archivo que enumera los nombres de funciones, métodos y otros nombres de trabajo de la biblioteca resaltados en el IDE de Arduino (resaltados en un color diferente) en el código.

Archivo **library.properties**: archivo que contiene información sobre la biblioteca para los desarrolladores y el administrador de la biblioteca (nombre, versión, autor, categoría, etc.).
Carpeta **Src**: esta carpeta puede contener los archivos de la biblioteca principal (**.h** , **.cpp** , **.c**).

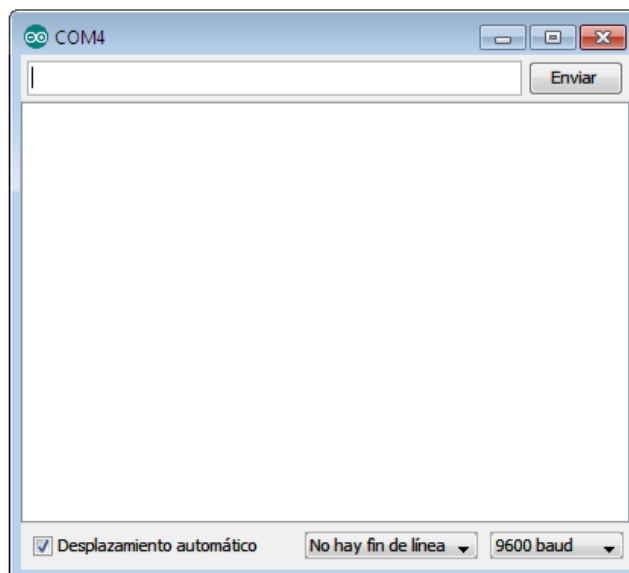
8.16. Comunicación Serial

Hay ocasiones en las que deseamos que Arduino no sólo lea o devuelva un valor determinado, sino que además queremos que nos lo muestre en pantalla para saber si la plataforma está aportando valores coherentes. Porejemplo: Si estamos leyendo la temperatura de la clase y nos devuelve 80°C... Es que algo está fallando, ¿No? Para ello usamos el denominado **Puerto Serie**.

Asignaríamos un valor al puerto serie y a continuación indicaríamos al programa que queremos que éste valor sea impreso.

8. Lenguaje de programación de Arduino

Arduino posee como principal característica la habilidad de comunicarse con nuestro ordenador a través del **Puerto Serie**. Esto se conoce como comunicación serial. Debido a que el uso de este puerto ha quedado un poco en desuso a favor de la tecnología USB. Para ello, Arduino cuenta con un convertidor de Serial a USB que permite a nuestra placa ser reconocida por nuestro ordenador como un dispositivo conectado a un puerto COM aun cuando la conexión física sea mediante USB.



A través de esta ventana se puede enviar o recibir información utilizando el puerto serie. Nótese que para poder abrir esta ventana es necesario que tengamos nuestra placa Arduino conectada a nuestro ordenador mediante el puerto USB.

Para iniciar la comunicación serial con Arduino utilizando el Monitor Serial debemos establecer algunos comandos en el Arduino IDE y luego subirlos al microcontrolador.

```
void setup(){
  Serial.begin(9600); //establece la configuracion del monitor serie a 9600
}
void loop(){
  Serial.println('1'); //visualiza por el monitor serie pulsacion del 1
  delay(1000);
}
```

En la función `setup` inicializamos la comunicación serial con la sentencia **`Serial.begin(9600);`**

El **9600** indica el rango en baudios, o la cantidad de baudios que manejará el puerto serie. Se define baudio como una unidad de medida, usada en telecomunicaciones, que representa el número de símbolos por segundo en un medio de transmisión ya sea analógico o digital. Para nuestros propósitos utilizaremos comúnmente una velocidad de símbolo de 9600.

Siempre que vayamos a comunicarnos con Arduino vía puerto serie se necesita invocar la sentencia `Serial.begin(9600)`.

Ahora, en la función `loop` nos encontramos con una sentencia interesante: **`Serial.println('1')`**.

8. Lenguaje de programación de Arduino

Cuando usamos **Serial.println()** le estamos diciendo al microcontrolador que tendrá que imprimir un carácter a través del puerto serie. Hay otros métodos como `Serial.print` o `Serial.write` que nos sirven para imprimir o escribir en el puerto serie, sin embargo el método `Serial.println` agrega un salto de línea cada vez que se envía un dato, lo que es provechoso para nosotros en algunos casos, especialmente cuando utilizamos el Monitor Serial. Cuando utilizamos el **Serial.println** se debe establecer lo que se quiere imprimir entre paréntesis y con comillas.

Serial.begin(rate)

Esta instrucción abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. Para ver los valores que van apareciendo en el programa tenemos que abrir el Monitor Serial, en el entorno de Arduino.

El valor típico de velocidad para comunicarse con el ordenador es de 9600, aunque otras velocidades pueden ser soportadas.

```
void setup() {  
  Serial.begin(9600); //abre el puerto serie configurando la velocidad en  
  9600bps }  
}
```

NOTA: Cuando se utiliza la comunicación serie los pins digital 0 (Rx) y el 1 (Tx) no pueden utilizarse al mismo tiempo.

Serial.println(data)

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que `Serial.print()`, pero es más fácil para la lectura de los datos en el Monitor del Software.

```
Serial.println(analogValue); // envía el valor analogValue al puerto
```

El siguiente ejemplo toma de una lectura analógica `pin0` y envía estos datos al ordenador cada 1 segundo.

```
void setup() {  
  Serial.begin(9600); //configura el puerto serie a 9600 bps  
}  
void loop() {  
  Serial.println(analogRead(0)); // envía valor analógico  
  delay(1000); // espera 1 segundo  
}
```

Serial.print (data, data type)

Vuelca o envía un número o una cadena de caracteres, al Puerto serie. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que utilizamos para definir el formato de volcado de los números.

Parámetros:

- **data:** el número o la cadena de caracteres a volcar o enviar.
- **data type:** determina el formato de salida de los valores numéricos (decimal, octal, binario, etc.) DEC, OCT, BIN, HEX, BYTE, si no se pone nada vuelve ASCII.

8. Lenguaje de programación de Arduino

Ejemplo:

```
Serial.print(b); //envía el valor de b como un numero decimal en caracteres
ASCII
int b = 79;
Serial.print(b); // imprime la cadena 79
Serial.print(b, HEX); //envía el valor de b como un numero hexadecimal en
caracteres ASCII "4F"
Serial.print(b, BIN); //envía el valor de b como un numero binario en
caracteres ASCII "1001111"
Serial.print(b, BYTE); //devuelve el carácter "0" el cual representa
el carácter ASCII del valor 79
Serial.print(str); //envía la cadena de caracteres como una cadena ASCII
Serial.print("Hello World!"); // Vuelca "Hello World!"
```

Serial.available()

Devuelve un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, Serial.available() será mayor que cero. El buffer serie puede almacenar como máximo 64 bytes.

```
int Serial.avaliabile() /* Obtiene un numero entero con el numero de bytes
(caracteres) disponibles para leer o capturar desde el puerto serie*/
```

Ejemplo:

```
int incomingByte = 0; // almacena el dato serie
void setup() {
Serial.begin(9600); // abre el puerto serie, y le asigna la velocidad de 9600
bps
}
void loop() {
if (Serial.available() > 0) // envía datos solo si los recibe
{
incomingByte=Serial.read(); //lee el byte de entrada: lo vuelca a pantalla
Serial.print("I received: "); //muestra por monitor el texto
Serial.println(incomingByte, DEC); //muestra por monitor valor incomingByte
}
}
```

Serial.Read()

Lee o captura un byte (un carácter) desde el puerto serie. Devuelve el siguiente byte (carácter) desde el puerto serie o - 1 si no hay ninguno.

Ejemplo:

```
int incomingByte = 0; // almacena el dato serie
void setup() {
Serial.begin(9600); //abre el puerto serie y le asigna la velocidad de 9600
bps
}
void loop() {
if (Serial.available() > 0) //envía datos solo si los recibe
{
```

8. Lenguaje de programación de Arduino

```
incomingByte=Serial.read();//lee el byte de entrada y lo vuelca a la pantalla
Serial.print("I received: ");
Serial.println(incomingByte, DEC);
}
}
```

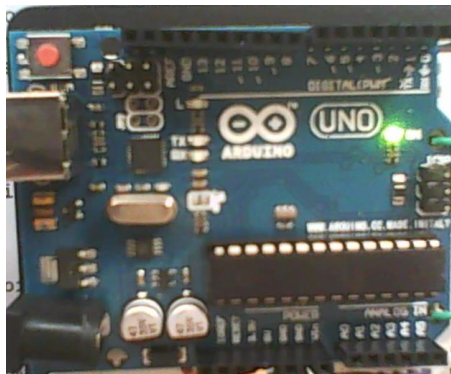
Veamos a continuación un ejemplo de comunicación Serial para controlar el encendido de luces desde el teclado de nuestro ordenador.

Nos centraremos en este ejemplo en encender un LED con las teclas de nuestro ordenador, utilizando el Monitor Serial para la comunicación con la placa de Arduino. Para ello introducimos en la línea de Enviar un 1 y clic en enviar, el LED se enciende. Cualquier otra tecla o numeración que no esté el 1 se apagará el LED.

```
int input; //creamos una variable input
void setup() {
  pinMode(13, OUTPUT); //configuramos que utilizaremos el pin 13 como salida
  Serial.begin(9600); //establecemos puerto serial con rango de 9600
}
void loop() {
  if (Serial.available()>0){
    input=Serial.read(); //almacena el valor leído en input
    if (input=='1'){ //si el valor de input es igual a 1
      digitalWrite(13, HIGH); //enciende el led
    }
    else //de lo contrario es diferente a 1
    {
      digitalWrite(13, LOW); // apaga el LED
    }
  }
}
```

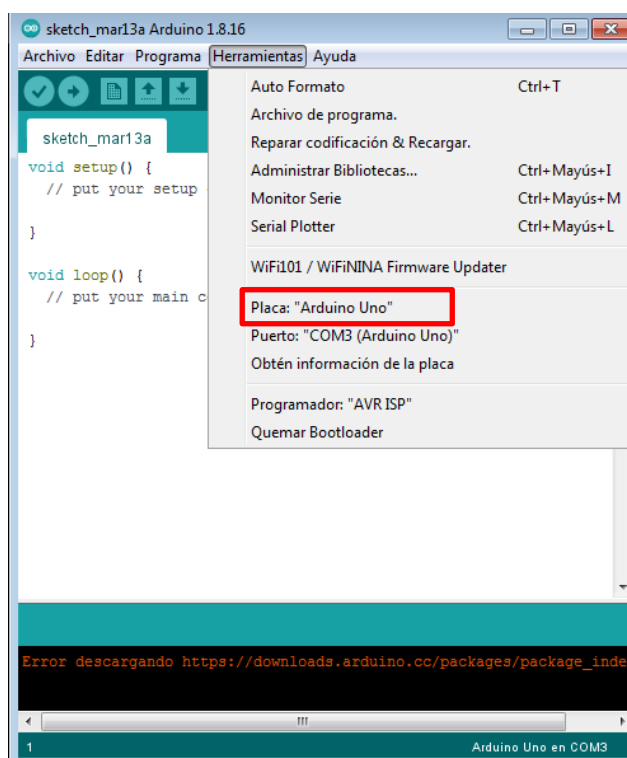
9. Comenzar a trabajar con Arduino

Lo primero que tenemos que hacer para empezar a trabajar con Arduino es instalar el entorno de desarrollo de Arduino **IDE** y configurar las comunicaciones entre la placa Arduino y el PC. Para ello deberemos conectar nuestro cable USB a la placa de Arduino y a un puerto USB del PC. Se encenderá un Led verde "Power Led" que nos indicará que la placa está alimentada.



Según el sistema operativo que dispongamos en nuestro PC, detectará o no, nuestra placa Arduino, en Windows 7, por ejemplo, detectará automáticamente la placa Arduino e instalará los drivers automáticamente para que se dé de alta en el sistema, de lo contrario tendremos que instalarlo manualmente.

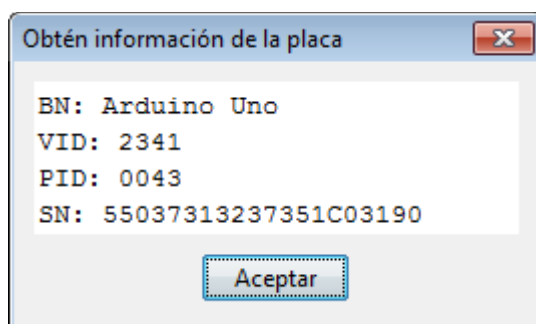
Posteriormente vamos al menú "Herramientas" y en la opción "Placa" seleccionamos el tipo de placa que coincida con la que tenemos conectada, en este caso sería "Arduino UNO".



La siguiente opción deberemos seleccionar el **Puerto Serial** al que está conectada nuestra placa. En Windows si desconocemos el puerto al que está conectado nuestra placa podemos descubrirlo a través del administrador de dispositivos (Puertos COM&LPT/USB Serial Port). En este caso nos configura el puerto serie COM3 (Arduino UNO).

9. Comenzar a trabajar con Arduino

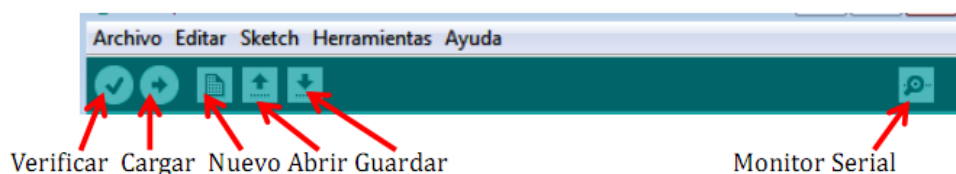
El siguiente paso es comprobar que nuestra placa está correctamente configurada y para ello accedemos a la opción de “Obtén información de la placa”



El siguiente paso es comprobar que todo lo que hemos hecho hasta ahora está bien y familiarizarnos con el interfaz de desarrollo, es abrir uno de los ejemplos. Se recomienda abrir el ejemplo “Blink”. Para ello debemos acceder a través del menú Archivo/Ejemplos/01.Basics/Blink. El ejemplo “**Blink**” lo único que hace es parpadear un LED naranja L que está instalado en la misma placa de Arduino y que corresponde con el pin número D13 de la placa.

Vamos a ver qué hay que hacer para subir el programa a la placa Arduino:

1. Primero comprobamos que el código de programación esté correcto para su compilación. Para ello pulsamos el botón de **verificación** en la barra de herramientas del interfaz de programación de Arduino que tiene forma de triángulo inclinado 90 grados. Si todo está correcto deberá aparecer un mensaje en la parte inferior de la interfaz indicando “Compilación Terminada. Tamaño binario del Sketch: 1084 (de un máximo de 30.720 bytes)”.
2. Una vez que el código ha sido verificado procederemos a cargarlo en la memoria flash del microcontrolador de la placa Arduino. Para ello tenemos que pulsar el botón de **cargar** (símbolo en forma de flecha hacia la derecha) si todo está correcto aparece el mensaje Carga Terminada. Tamaño binario del Sketch: 1084 (de un máximo de 30.720 bytes).
3. Al finalizar la carga se espera unos milisegundos para ejecutarse el programa y observar que se ejecuta correctamente, en este caso el **Led naranja L** de carga de la placa arduino empieza a parpadear con un intervalo de un segundo.

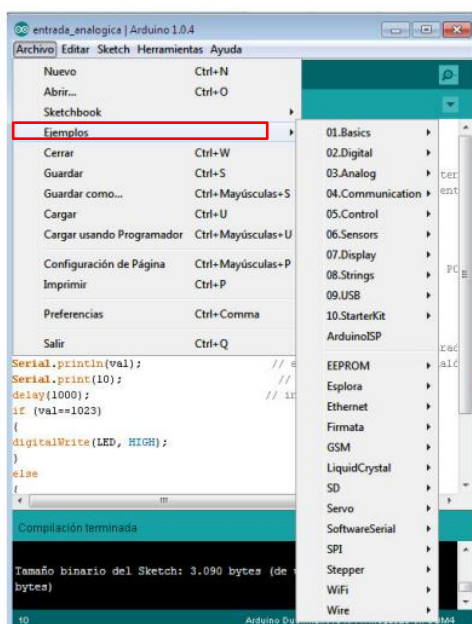


Durante la carga del programa, en la placa de Arduino, se encenderán los LEDs de color naranja de comunicación Tx/Rx que nos indican que se está produciendo transmisión y recepción de datos por el Puerto Serial.

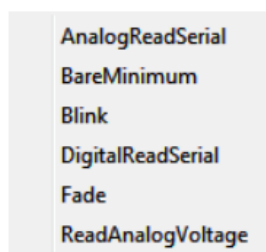
NOTA: Para mayor rapidez, se puede obviar la verificación y pulsar directamente “cargar”, puesto que también el programa antes de cargarlo en la memoria flash lo verifica.

9. Comenzar a trabajar con Arduino

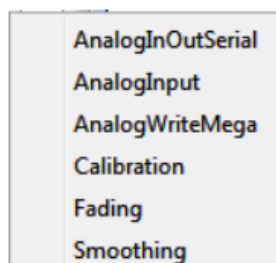
En el entorno de desarrollo de Arduino nos permite cargar diversidad de ejemplos de Sketch de diferentes autores y utilidades específicas: Digital, Analógico, Control, Sensores, Display, comunicación, etc., accediendo al menú **Archivo/Ejemplos**:



Cada una de las opciones posee varios ejemplos de Sketch, por ejemplo, en la opción **Basic**, tenemos:



En la opción **Analog**, tenemos:



En cada uno de los ejemplos de programación, en su encabezamiento, se muestran los autores, los comentarios relacionados con el Sketch: tipos de variables, funciones, constantes, etc. Estos programitas son de libre utilización y nos aporta una gran ayuda para tener una referencia y añadirlo a nuestro programa que estemos desarrollando.

9. Comenzar a trabajar con Arduino

9.1. Una programación simbólica: Blink

Este es un ejemplo de programación simbólica del microcontrolador Arduino "Blink" (Parpadeo).

Este programita nos sirve de prueba para comprobar que nuestro Arduino funciona correctamente, haciendo la siguiente secuencia: carga el programa Blink en el entorno de programación de Arduino, lo verifica y lo compila, y si no hay fallos de compilación lo carga en la memoria flash del microcontrolador y lo ejecuta.



En este ejemplo el Led está conectado en el pin 13, salida digital, y se enciende y apaga cada segundo. La resistencia que se debe colocar en serie con el Led es de 220-330 Ohmios.

Si nos fijamos en la programación el inicio comienza declarando el pin D13(int led=13) donde corresponde al pin 19 de Arduino. Seguidamente en void setup() se declara pin Led como de salida (OUTPUT). En void loop se introducen valores de alto y bajo para que el Led se apague y encienda a intervalo de 1 segundo entre apagado y encendido. La secuencia continuaría así indefinidamente.

NOTA: Este ejemplo está sacado del entorno de desarrollo de Arduino desde Archivo/Ejemplos/01Basics/Blink y es de dominio público como todos los que aparecen en esta opción de ejemplos. En las líneas de comentarios se especifica en que consiste el Sketch. La programación de Blink lo único que hace es parpadear indefinidamente un LED que está colocado en el pin número 13 de la placa cada un segundo.

```
/* Blink Turns on an LED on for one second, then off for one second,
repeatedly. This example code is in the public domain. */
int led = 13; // Pin 13 has an LED connected on most Arduino boards

void setup() { // the setup routine runs once when you press reset
  pinMode(led, OUTPUT); // initialize the digital pin as an output
}
void loop() { // the loop routine runs over and over again forever
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

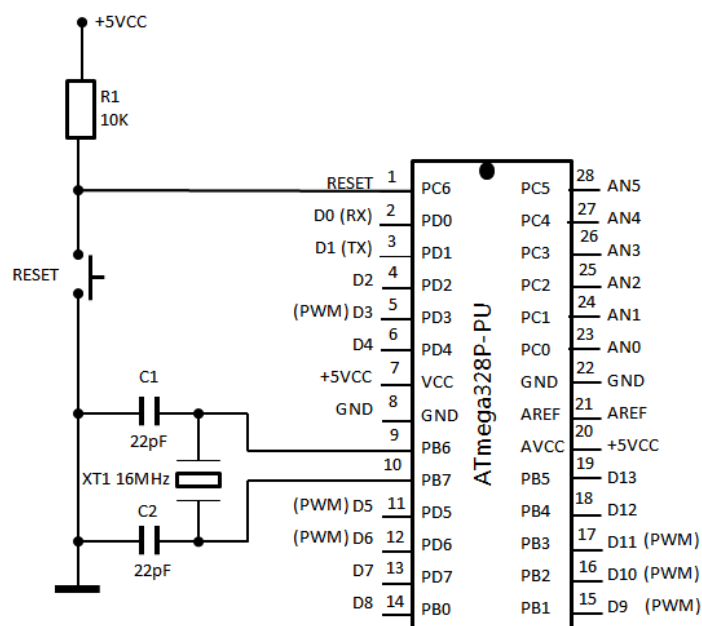
9. Comenzar a trabajar con Arduino

9.2. Programaciones y circuitos eléctricos

El hardware de la placa Arduino es un elemento que incorpora los componentes necesarios que nos permite trabajar con un determinado microcontrolador, el ATmega 328P-PU e incorporarlo a nuestros proyectos. Entre los componentes que nos permiten el uso del microcontrolador, podemos resaltar:

- El programador Arduino, cuya finalidad principal es reprogramar el microcontrolador tantas veces como haga falta, pero también permite alimentarlo, con lo cual ya podría ejecutar las tareas programadas y dotar de un conector para la comunicación exterior.
- El sistema de alimentación externo superior a 5 voltios regulado. Este elemento nos ofrece alimentación para el microcontrolador, con una fuente de alimentación superior a 5 voltios. El microcontrolador necesita alimentación como cualquier sistema electrónico para poder operar; en concreto necesita una alimentación de 5V por lo que esta alimentación externa regulada aumenta el rango de alimentación, pero fija una salida de 5V.
- Redistribución de los pines. Este componente, si lo consideramos como tal, es útil de cara a un uso sencillo de la plataforma de Arduino, puesto que ordena las patillas del microcontrolador de una forma lógica, con lo cual nos agiliza su uso sin tener que consultar la hoja características del microcontrolador.
- Señal de reloj. Algunos microcontroladores incorporan un oscilador interno. Esta señal de reloj es la que marca los tiempos de ejecución de las instrucciones, la que permite contabilizar tiempo, generar señales PWM, etc.
- Conector hembra, para que sea más sencillo conectar elementos externos al microcontrolador y al resto de la electrónica del hardware.

El ATmega328P-PU es un microcontrolador que trabaja con niveles de tensión de 3,3, voltios y 5 voltios y tan solo puede entregar una corriente máxima de 40mA a las salidas de sus pines, por lo que, tan solo se va a poder manejar señales que estén dentro de éste rango. Por lo tanto es recomendable utilizar en los circuitos electrónicos una fuente de 5 voltios en continua y de 12 voltios en continua que ofrezca una corriente de unos 500 mA.



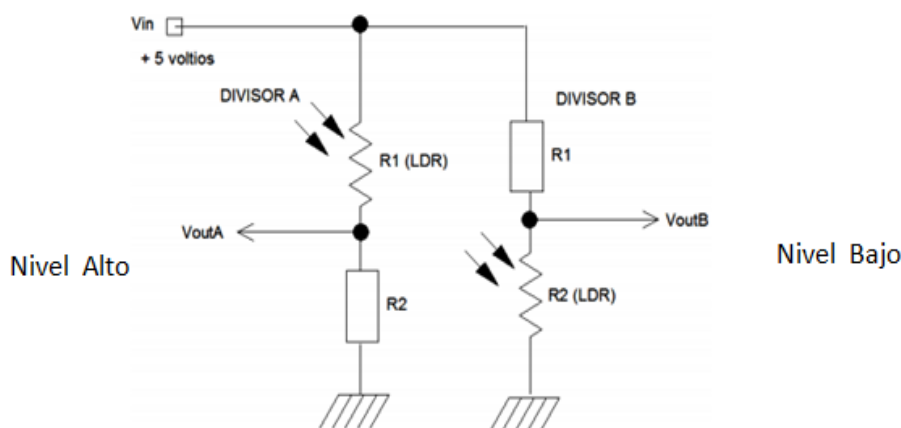
Microcontrolador ATmega328P-PU. Pines y componentes base.

9. Comenzar a trabajar con Arduino

Si la señal con la que estamos trabajando está dentro del rango que puede soportar el microcontrolador ATmega, no vamos a tener ningún problema a cualquier tipo de señal, ya sea digital o analógica.

Pero no todos los casos son favorables y lo podemos desconocer, para ello se recomienda, en los casos desconocidos, utilizar y aislar mediante circuitos optoacopladores las señales y tensión de los circuitos exteriores para una protección del microcontrolador.

Otro de los aspectos importantes que se debe tener en cuenta es la utilización de resistencias del tipo PULL-UP y PULL-DOWN para polarizar la entrada de los puertos digitales: pulsadores o interruptores, sensores, etc. Para conmutar la entrada de los pines entre un nivel ALTO (señal) y nivel BAJO (sin señal). Para conmutar los niveles ALTOS se utilizan los PULL-DOWN, una resistencia R2 se conecta a GND con el pin de entrada y un y un conmutador o sensor en serie conectado a +5Vcc. Y para conmutar los niveles BAJOS, se usan los PULL-UP, resistencia R1 conectada a +VCC y al pin de entrada y el sensor o conmutador conectado a GND.



Los esquemas eléctricos que se muestran en los ejemplos siguientes comprende la conexión de dos placas: una corresponde a la placa Arduino, sin microcontrolador, utilizando las conexiones de los puertos TX/RX, Reset y alimentación (+5V y GND), para conectarlos con la placa de pruebas protoboard, que incluye el microcontrolador y, las conexiones de Reset, Reloj y alimentación y sus componentes de polarización.

En la placa protoboard irán los siguientes elementos:

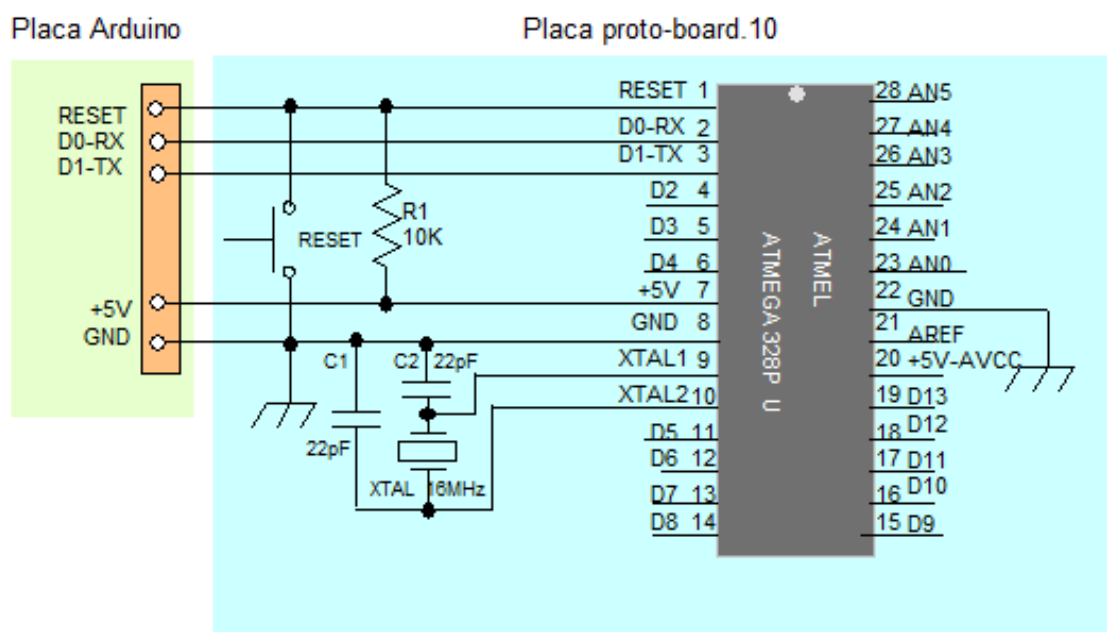
- La alimentación. El microcontrolador para que funcione correctamente necesita una tensión de +5V y GND que necesariamente se tiene que conectar el positivo en los pines 7(VCC) y 20 (AVCC) y el negativo en los pines 8 (GND) y 22(GND). Todos los demás elementos de entradas y salidas deben trabajar con +5V y GND.
- Comunicación TX-RX. Para enviar y recibir datos del microcontrolador o cuando se carga la programación desde el PC al microcontrolador se hace mediante las líneas de transmisión TX y recepción RX, de los pines 2 y 3 del microcontrolador.
- La señal de reloj. Para el correcto funcionamiento del microcontrolador, se utiliza un circuito que genera una señal de reloj: que marca el ritmo de ejecución de instrucciones, permite contabilizar tiempo y generar señales PWM. Está formado por dos condensadores cerámicos de disco de 22pF y un cristal de cuarzo de 16Mhz, conectados a los pines 9 y 10 del microcontrolador.

9. Comenzar a trabajar con Arduino

- La señal de reset, es cuando se resetea Arduino se activa el gestor de arranque, el cual comprueba si hay algún programa pendiente de carga. Si lo hay, lo que hace el gestor de arranque es sobrescribir en la memoria FLASH el nuevo programa. Si no hay ningún programa el gestor de arranque inicia la ejecución normal del programa.

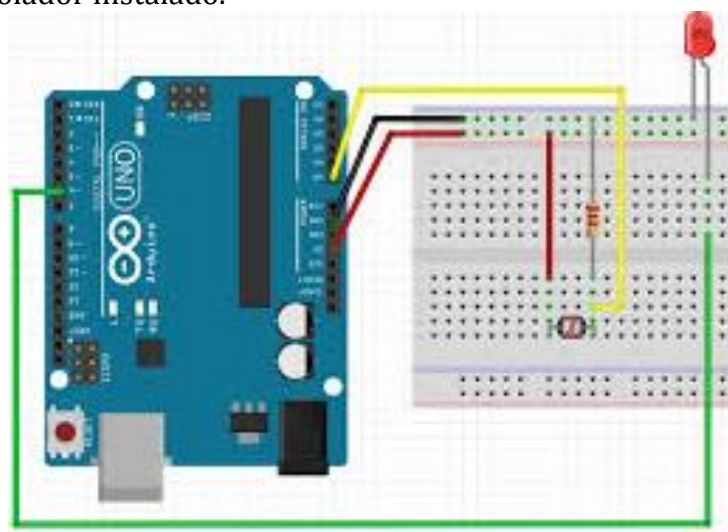
El reseteo del hardware se puede hacer por software o de forma manual, el poder hacerlo por software nos resultará más cómodo ya que no tenemos que preocuparnos de eso, puesto que el sistema lo hace automáticamente. El reseteo por hardware está formado por la conexión del pin 1 del microcontrolador conectado a una resistencia de 10 K y ésta a +5V y de un botón pulsador, Normalmente Abierto, conectado a GND, puesto que la entrada de RESET al microcontrolador se activa a nivel bajo (0).

En los capítulos posteriores se describen diferentes programaciones con sus circuitos de conexión de la placa Arduino a la placa Protoboard con los componentes auxiliares.



Esquema conexión de la placa Arduino a la placa de pruebas protoboard

También se puede conectar la placa protoboard, que requiere únicamente la alimentación de +5V y GND y las conexión de los componentes a la regleta hembra de la placa Arduino con su microcontrolador instalado.

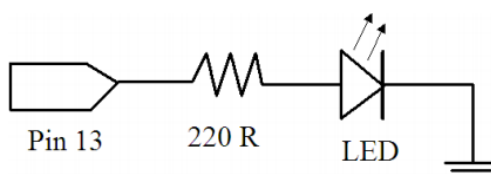


10. Salida y entrada digital

Salida digital

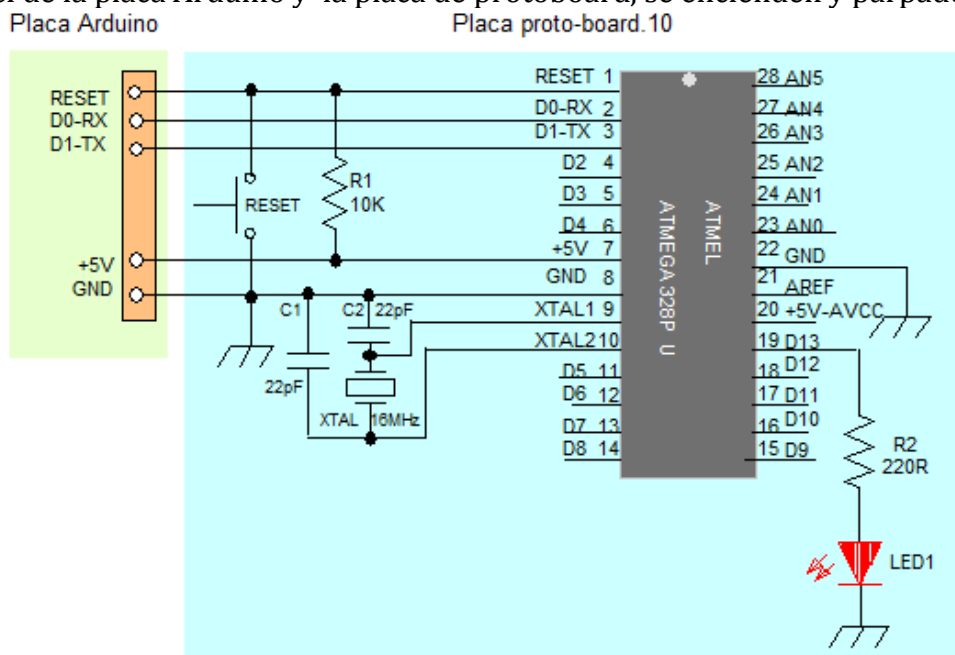
Éste es el ejemplo básico que se toma de referencia en la programación de Arduino y que algunos programas se fundamenta en ello, como el programa “Blink” que se suele utilizar para verificar si la placa Arduino está bien configurada, entra bien la programación y el microcontrolador funciona correctamente.

Simplemente lo que hace este sketch es encender y apagar un Led. En este ejemplo el LED está conectado en el pin 13 y se enciende y apaga “parpadeando” cada segundo. La resistencia que se debe colocar en serie con el LED en este caso puede omitirse ya que el pin 13 de la placa Arduino ya incluye en la tarjeta esta resistencia.



```
/* parpadeo de un led "Blink"*/  
int ledPin = 13;      // LED en el pin digital 13  
void setup()          // configura el pin de salida  
{  
  pinMode(ledPin, OUTPUT); // configura el pin 13 como salida  
}  
void loop()           // inicia el bucle del programa  
{  
  digitalWrite(ledPin, HIGH); // enciende el LED  
  delay(1000); // espera 1 segundo  
  digitalWrite(ledPin, LOW); // apaga el LED  
  delay(1000); // espera 1 segundo  
}
```

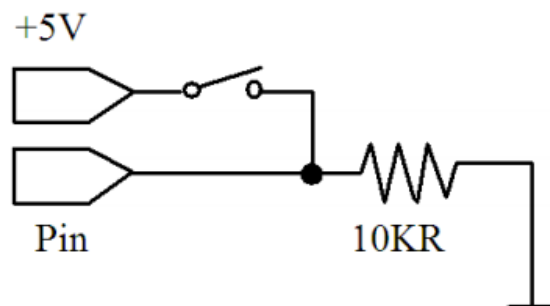
En el caso de que deseemos montarlo en nuestra placa protoboard, observaremos que los dos leds, el de la placa Arduino y la placa de protoboard, se encienden y parpadean a la vez.



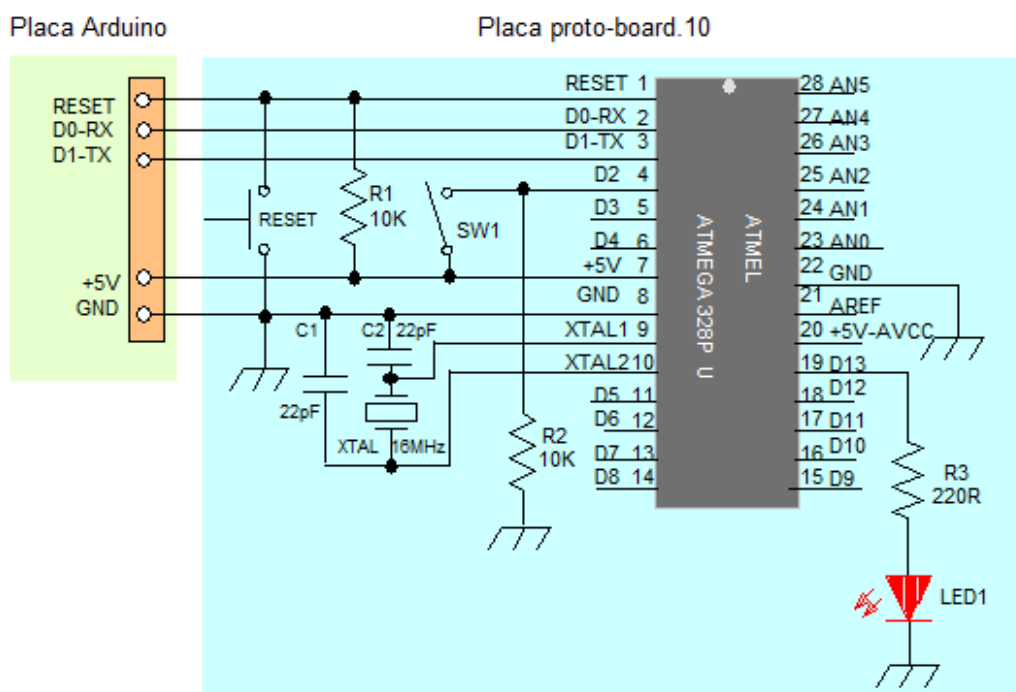
10. Salida y entrada digital

Entrada digital

Esta es la forma más sencilla de entrada con solo dos posibles estados encendido o apagado. En este ejemplo se lee un simple switch o pulsador conectado a pin2. Cuando el interruptor está cerrado el pin de entrada se lee ALTO y encenderá un LED colocado en el pin 13. Cuando el interruptor se abra el LED se apagará a los dos segundos.



```
int ledPin = 13; // pin 13 asignado para el LED de salida
int pulsador = 2; // pin 2 asignado para el pulsador
void setup() // configura entradas y salidas
{
  pinMode(ledPin, OUTPUT); // declara LED como salida
  pinMode(pulsador, INPUT); // declara pulsador como entrada
}
void loop()
{
  if (digitalRead (pulsador) == HIGH) //si el pulsador esta pulsado
  {
    digitalWrite(ledPin, HIGH); // enciende el LED
    delay(2000); //espera 2 segundo encendido
    digitalWrite(ledPin, LOW); // apaga el LED
  }
}
```



11. Entrada analógica

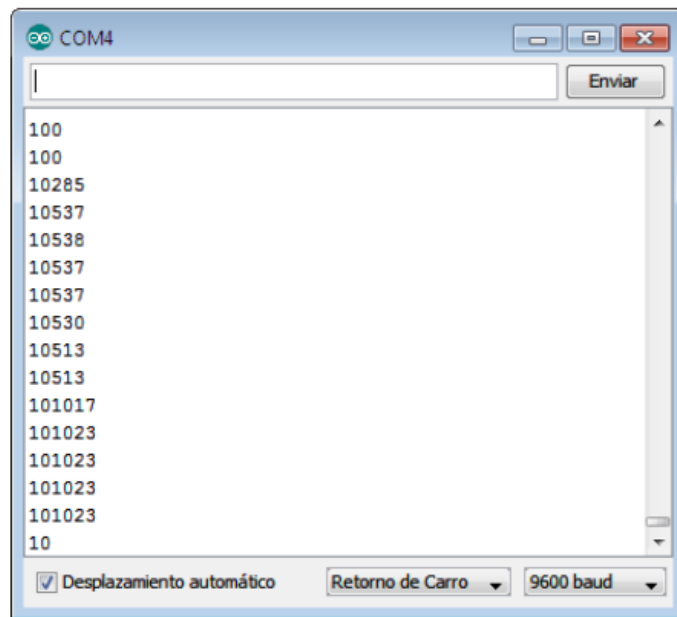
Esta vez, se trata de configurar un canal de entrada analógico pin 5 y enviar el valor leído al PC para visualizarlo mediante el Monitor Serial.

En esta práctica vamos a utilizar un potenciómetro de 22 K Ohmios donde el punto medio o central del potenciómetro se conectará a la entrada pin 5 analógica del microcontrolador Atmega328P-PU, y los extremos a + 5Vcc y Gnd. Configuraremos un LED de salida para cuando llegue el potenciómetro a marcar al valor 1023 se active el LED.

```
/* Entrada Analógica */
int potPin=5; // selecciona el pin de entrada para colocar el potenciómetro
int val=0; // variable para almacenar el valor leído por la entrada
analógica
int LED=13; //selecciona el pin 13 para LED
void setup(){
pinMode(LED, OUTPUT); //configuramos el pin LED de salida
Serial.begin(9600); // pone a 9600 la velocidad de transmisión al PC
}
void loop(){
val=analogRead(potPin); // lee el valor del canal de entrada analógica
Serial.println(val); // envía al PC el valor analógico leído y lo
muestra en pantalla
Serial.print(10); // indice 10
delay(1000); // intervalo de respuesta
if (val==1023){ //si val es igual 1023
digitalWrite(LED, HIGH); //enciende el LED
}
else { //de lo contrario
digitalWrite(LED, LOW); //apaga el LED
}
}
```

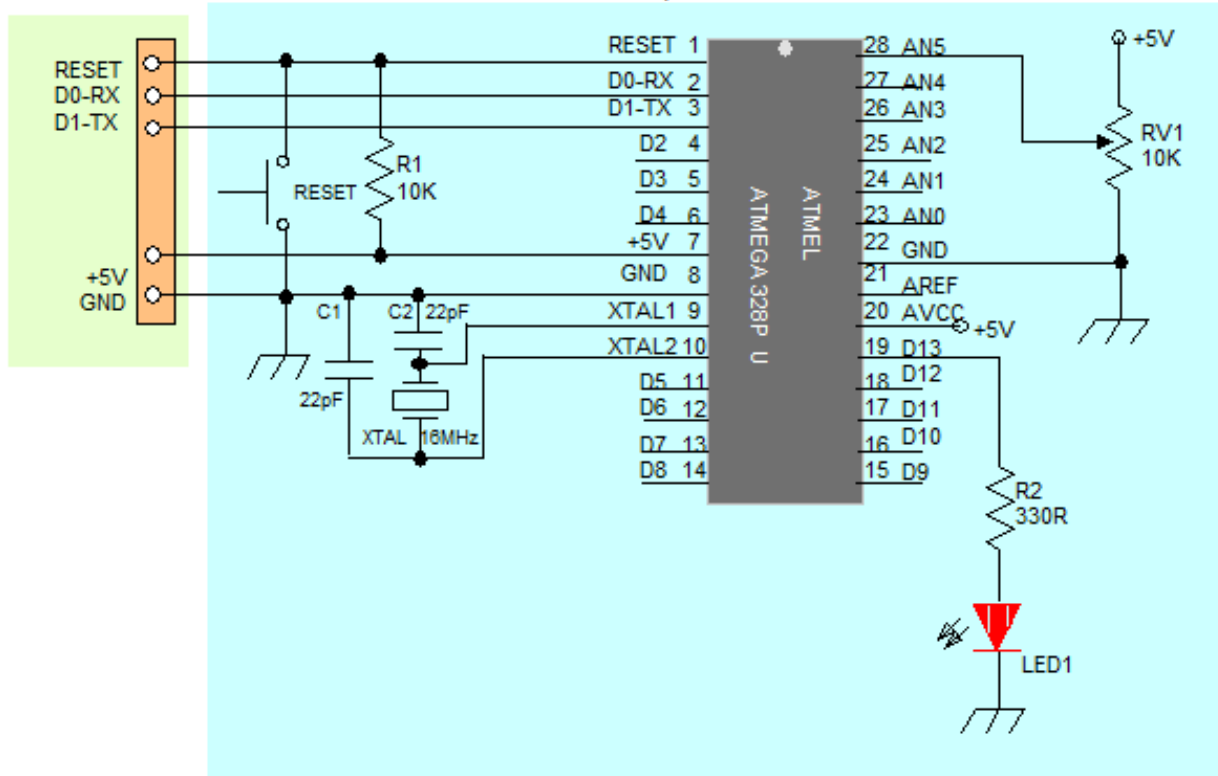
NOTA: Cuando se termina la compilación de un Sketch sin ningún error, esto no quiere decir que nuestro diseño hardware vaya a funcionar correctamente, pues si no se enciende un LED que se debería encender según lo programado, es que ha habido algo que se nos ha escapado a la vista en las instrucciones del programa, para ello, revisamos los códigos del programa y ya puesto comprobamos que el Led está bien polarizado y no está fundido...por si acaso.

11. Entrada analógica



Placa Arduino

Placa proto-board.10



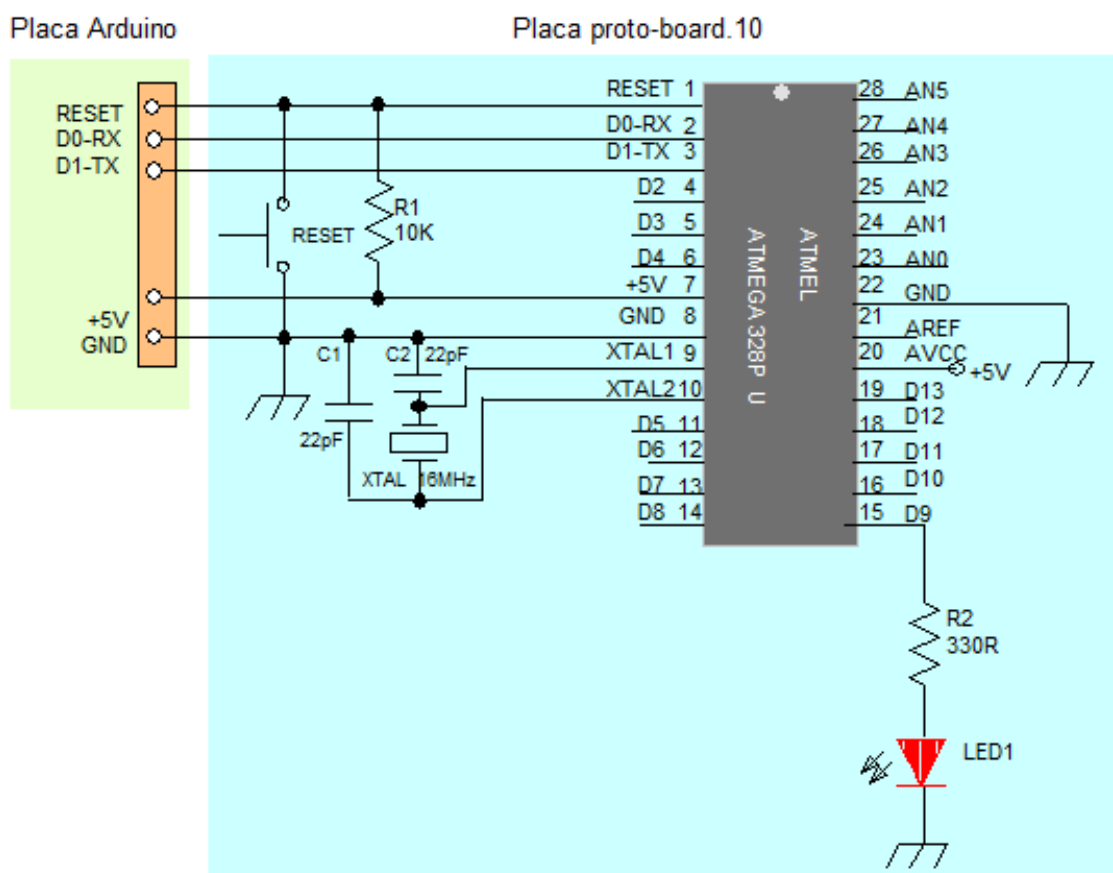
12. Salida analógica del tipo PWM

La Modulación de impulsos en Frecuencia **PWM** es una forma de conseguir una “falsa” salida analógica. Esto podría ser utilizado para modificar el brillo de un LED o controlar un servo motor. El siguiente ejemplo lentamente hace que el LED se ilumine y se apague haciendo uso de dos bucles.

La modulación por ancho de pulsos (también conocida como **PWM**, siglas en inglés de *pulse-width modulation*) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

Arduino Duemilanove tiene entradas analógicas que gracias a los conversores analógico digital puede entender ese valor el microcontrolador, pero no tiene salidas analógicas puras y para solucionar esto, usa la técnica de **PWM**.

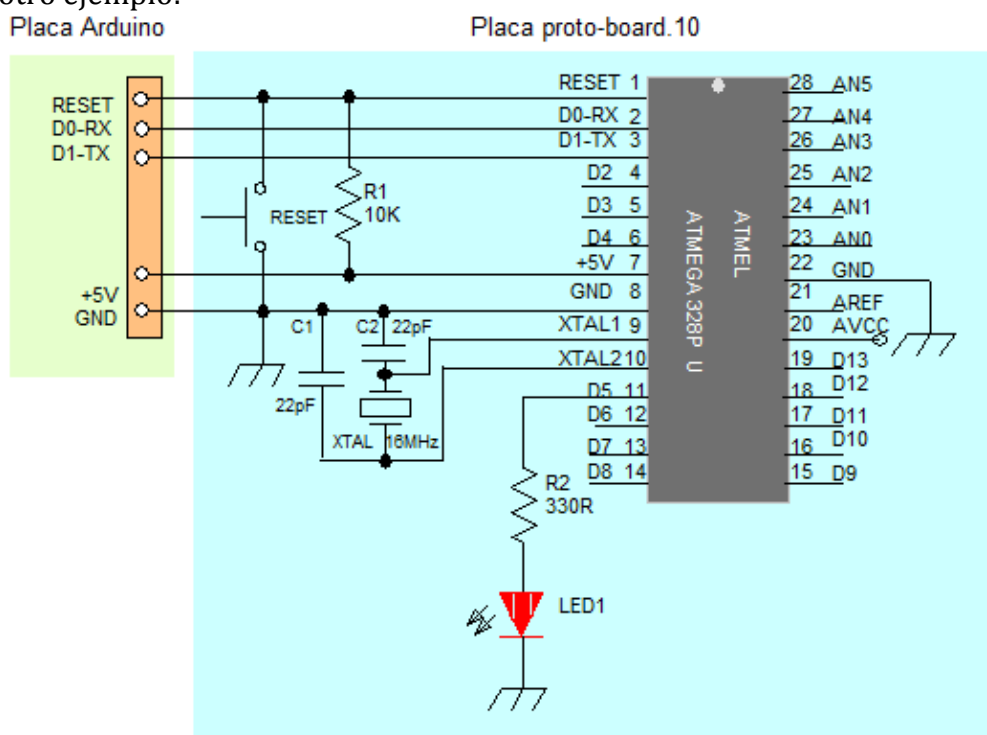
Las Salidas **PWM** (*Pulse Width Modulation*) permiten generar salidas analógicas desde pines digitales desde el microcontrolador Atmega328P-PU son D3, D5, D6, D9, D10 y D11.



12. Salida analógica del tipo PWM

```
/* luminosidad variable mediante pulsos PWM en un diodo LED*/
int ledPin=9;           // pin PWM para el LED
void setup(){          // no es necesario configurar nada
void loop() {
for (int i=10; i<=255; i++) { // el valor de I asciende
analogWrite(ledPin, i);      //se escribe el valor de i en el PIN de salida
del led
delay(100);                  // pausa durante 100ms
}
for (int i=255; i>=0; i--){ // el valor de i desciende
analogWrite(ledPin, i);      // se escribe el valor de i
delay(100);                  // pausa durante 100ms
}
}
}
```

Veamos otro ejemplo:

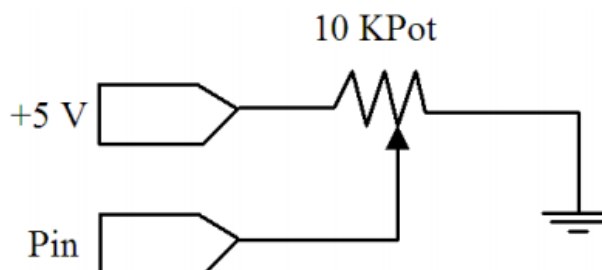


```
/*Luminosidad variable de un led mediante pulsos PWM */
void setup(){
pinMode(5,OUTPUT);      //asignamos y configuramos el pin 5 PWM de salida
}
void loop(){            // configuramos la rutina
analogWrite (5, LOW);  // comienza con el led apagado
brillo();              // llamada a la función brillo
}
void brillo(){          //bloque instrucciones de la función brillo
for (int i=0; i<=255; i=i+10){ // bucle for incrementa de 10 en 10 variable
i
analogWrite (5,i);     //led empieza a encenderse según la variable
delay(200);           //pausa de 200 milisegundos
}
}
}
```


13. Entrada analógica con potenciómetro

Uno de los dispositivos que requieren un tratamiento analógico son los llamados **potenciómetros** o resistencias variables.

Los potenciómetros están perfectamente diseñados para maniobrar con ellos a través de su eje que permite, cuando se actúa manualmente sobre ello, de variar el valor de la resistencia a diferentes rangos, del valor de la resistencia más bajo al más alto.

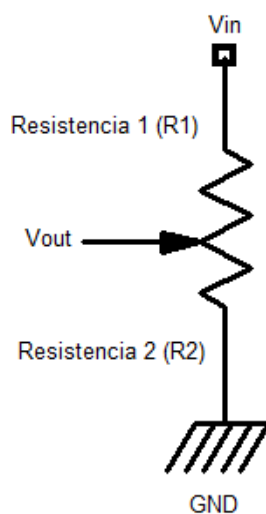


Posee tres terminales. El valor de la resistencia en los terminales extremos es fija, por ejemplo 10K, en el terminal central será el que permita, moviendo el eje de derecha a izquierda, variar y obtener un valor de resistencia desde $0\ \Omega$ a $10000\ \Omega$, aumentando o disminuyendo la resistencia a nuestro gusto.

Cuando esto ocurre, si tenemos un led conectado a éste, la luminosidad de este led se verá afectada con más o menos intensidad. Esto es debido a que creamos un divisor de tensión y se puede calcular mediante la siguiente fórmula:

$$V_{out} = \frac{R2}{R1 + R2} \cdot V_{in}$$

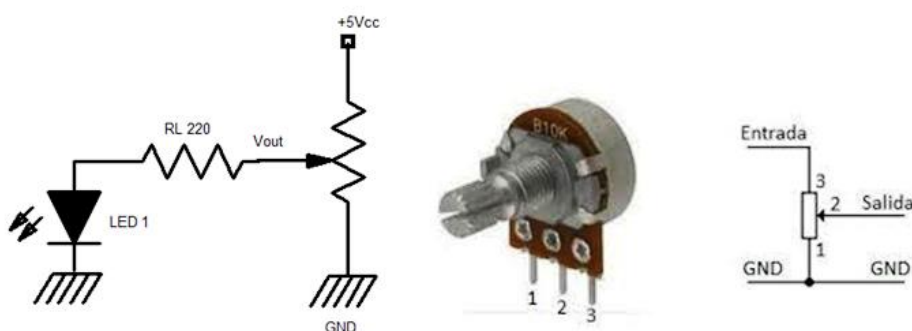
La **V_{out}** será la tensión de salida que obtendremos, dependiendo del valor de las resistencias **R₁** y **R₂** y la tensión de entrada de nuestro circuito **V_{in}**.



13. Entrada analógica con potenciómetro

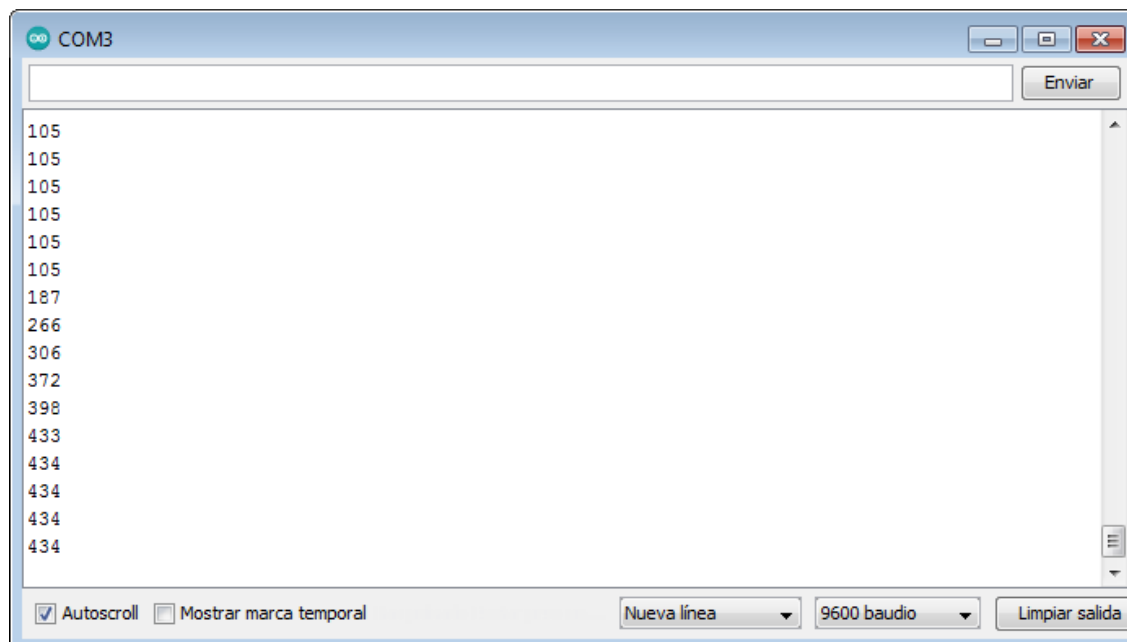
Si hacemos girar el eje del potenciómetro, hacia el potencial negativo o GND, la tensión en el diodo Led será muy pequeña y, en consecuencia, el Led no recibirá tensión y no se encenderá, ya que éste no recibirá los 1,7 voltios que necesita para lucir.

Pero en cambio, si giramos el eje hacia el lado contrario, potencial positivo o +5V, la tensión será aproximadamente la tensión de entrada V_{in} y tendremos una tensión máxima, con la que nuestro diodo Led lucirá completamente sin miedo a que se funda por tener una resistencia de protección de 220Ω .

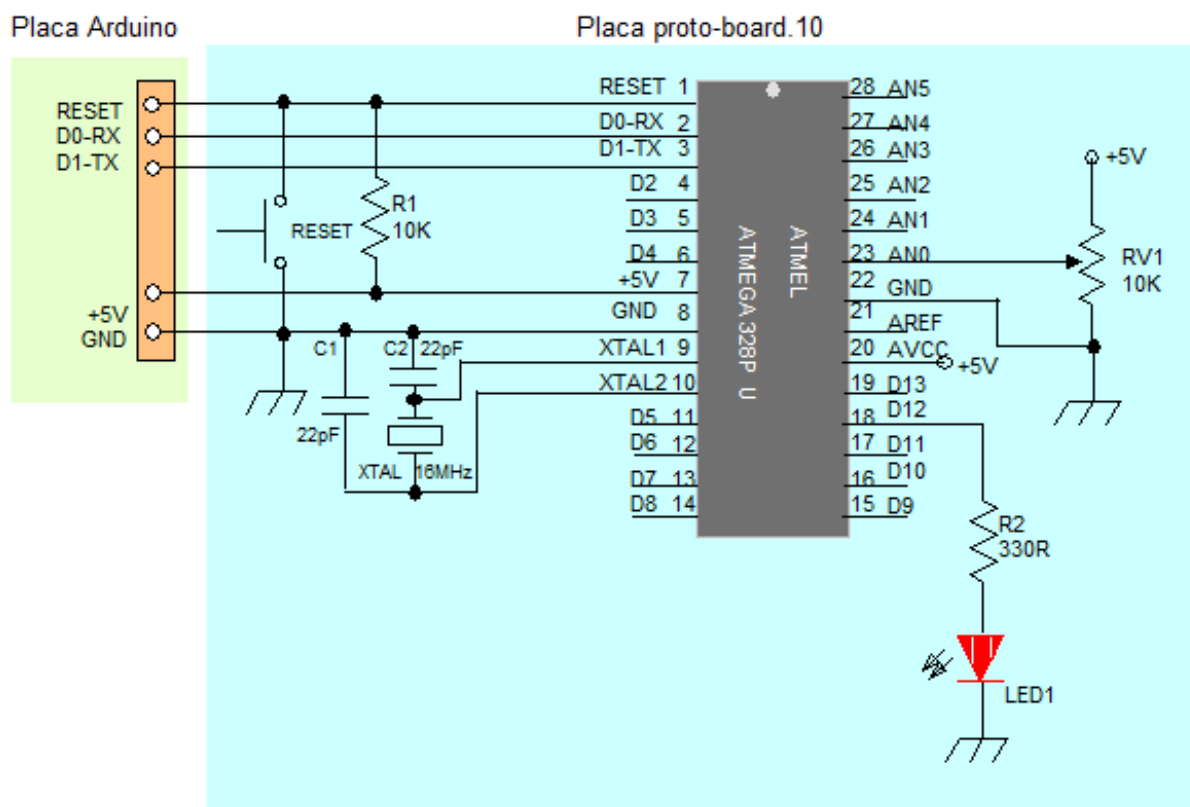


La placa Arduino dispone de 6 pines de entradas analógicas (ANALOG IN) desde la AN0 a AN5. El uso de un potenciómetro y uno de los pines de entrada analógica-digital de Arduino (ADC) permite leer valores analógicos que se convertirán en valores dentro del rango de 0-1024.

El siguiente ejemplo utiliza un potenciómetro RV1 para controlar el encendido y apagado de un diodo LED1, mediante los valores que aparecen por el monitor serie.



13. Entrada analógica con potenciómetro



Se programa tres rangos de valores para el encendido y apagado del diodo LED1, cuyos rangos se especifican en la programación mediante condicionales.

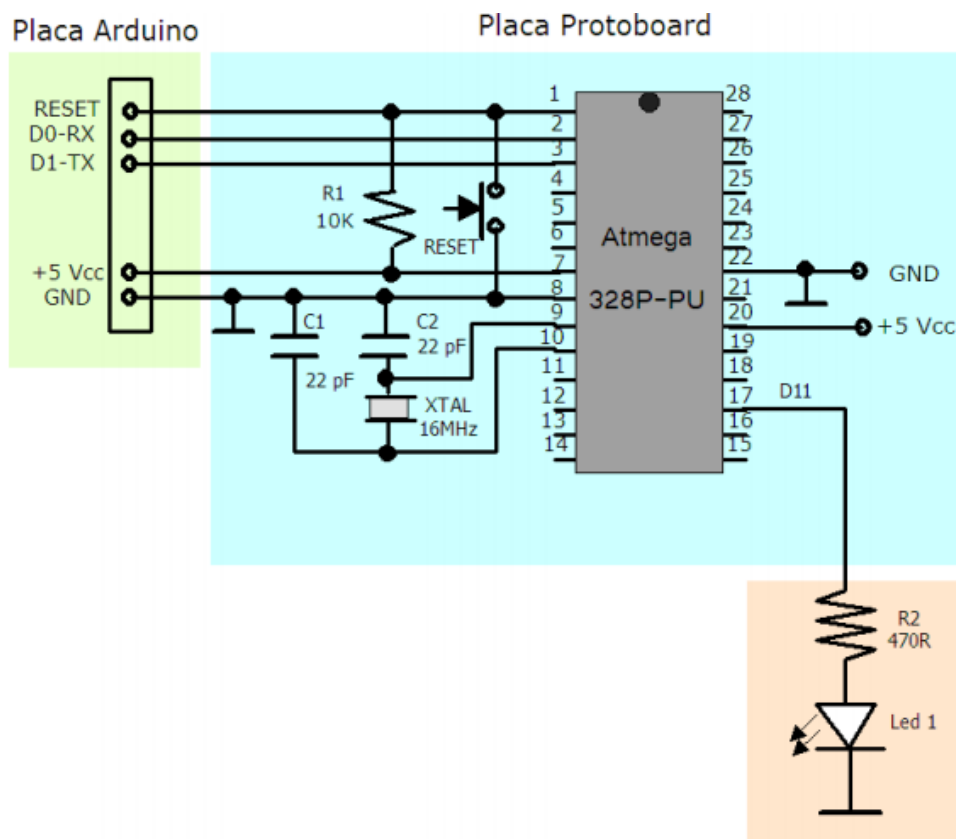
```
/*codigo de utilizacion de un potenciómetro*/
int potPin=A0; //asignamos pin analogico A0 terminal central potenciómetro
int valor; //variable que almacena los valores obtenidos del potenciómetro
int ledPin=12; // asignamos pin digital 12 a ledPin
void setup() {
  Serial.begin(9600); //habilitamos el monitor serie a 9600 bps
  pinMode (potPin, INPUT); //configuramos el potenciómetro de entrada
  pinMode (ledPin, OUTPUT); // configuramos el led de salida
}
void loop() {
  valor=analogRead(potPin); //almacena los valores en la variable valor
  Serial.println(valor); // muestra por pantalla los valores cuando giramos
  potenciómetro
  delay(500); //establece medio segundo para ver los valores
  if (valor<= 100) { // si valor es menor o igual a 100
    digitalWrite (ledPin, HIGH); // enciende el led
  }
  if (valor>300 && valor <500) { //si valor esta entre 300 y 500
    digitalWrite (ledPin, LOW); // apaga el led
  }
  if (valor>900){ //si valor es mayor de 900
    digitalWrite(ledPin, HIGH); // enciende el led
  }
}
```

14. Aumentar y disminuir la luminosidad de un Led

En esta práctica programaremos un Led que irá aumentando y disminuyendo la luminosidad usando la capacidad de ofrecer una tensión variables que da una salida analógica. Para ello se conecta un Led al pin D11 y se provoca que su luminosidad pase de mínima a máxima, para luego ir de máxima a mínima. Los valores de salidas analógicas van del mínimo 0 al máximo 255.

Podemos utilizar esta práctica como iluminación autónoma del día y la noche de un Belén, modificando el valor de delay, para propagar la duración del tiempo de día a la noche y viceversa. `int luminosidad=0;`

```
/* variable para asignar la luminosidad al Led */
int Led=11;           // pin de Led
int cantidadfundido=5; // puntos de fundido del Led
void setup() {
  pinMode(Led,OUTPUT); // declara pin Led de salida
}
void loop() {        // Comienza a correr rutina
  analogWrite(Led,luminosidad); // cambia led al valor de la luminosidad
  luminosidad = luminosidad + cantidadfundido; // sube la luminosidad
  if (luminosidad == 0 || luminosidad == 255){ // condiciona igualdad 0 y 255
    cantidadfundido = -cantidadfundido; // baja la luminosidad
  }
  delay(50); // pausa de 50 milisegundos para ver el efecto
}
```

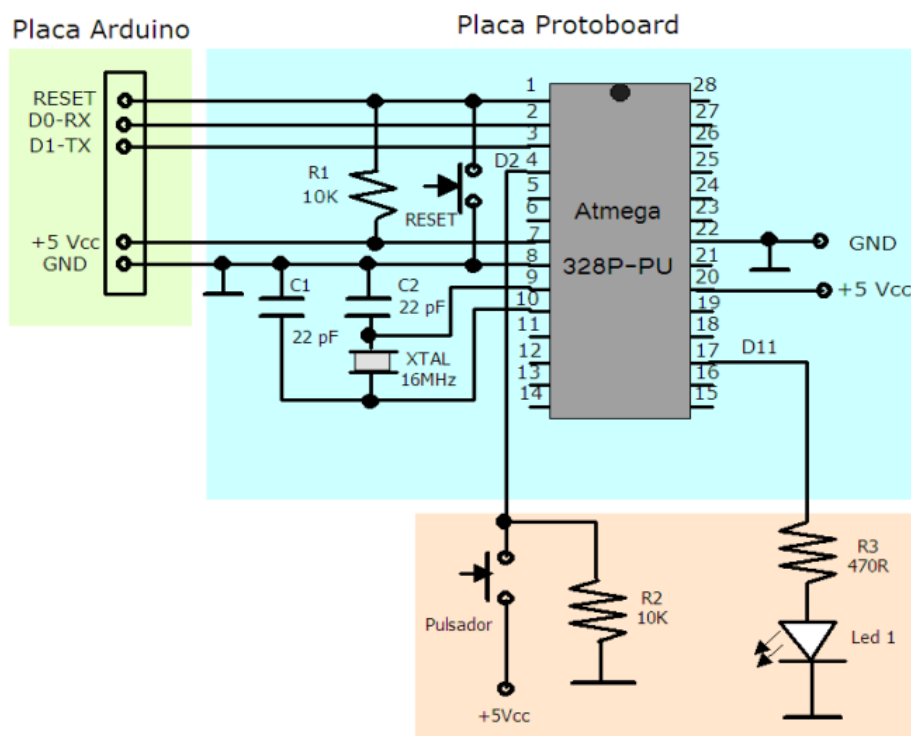


15. Aumentar la luminosidad de un Led con pulsador

En esta otra práctica muy semejante a la anterior, aumenta y disminuye la luminosidad de un Led pero en este caso controlada mediante un pulsador. El pulsador debe estar conectado a un pin digital D2 como entrada y como salida tendremos el diodo Led en el pin D11.

Mientras el pulsador está conectado (pulsado) aumenta la luminosidad del Led hasta llegar a su valor máximo (255) una vez que llegue a este umbral comienza a disminuir hasta su valor (0). Si el pulsador se desactiva (deja de pulsar) se mantendrá la intensidad de luminosidad en ese punto, hasta que se vuelva a pulsar y el valor de luminosidad llegue a su máximo (255) y pulsando más veces la luminosidad llegará a un valor nulo (0).

```
/* aumento de la intensidad de luz con un pulsador */
int led = 11;      // elegimos el pin del diodo led
int pulsador = 2; // elegimos el pin del pulsador
int x=0;          // configuramos la variable para incrementar el valor de luz
void setup(){
  pinMode(led,OUTPUT);    // declaramos led como salida
  pinMode(pulsador,INPUT); // declaramos pulsador como entrada
}
void loop(){
  // corre la rutina
  while (digitalRead(pulsador) == HIGH && x<=255){ /* chequea si el pulsador
  está pulsado y x es menor de 255*/
  analogWrite(led,x); /* aumenta la luminosidad del led en función del tiempo
  de activación de pulsador */
  delay(40);        // para ver el efecto de encendido
  x=x+3;           // suma mas 3 al valor de x
  }
  if (x>255) {     //si x es mayor de 255
  x=0;             // asigna el valor 0 a x
  analogWrite(led,0); // apaga el Led
  }
}
```



16. Secuencia de Leds

En esta práctica veremos como se enciende y apagan 4 Leds secuencialmente, es decir, se visualiza el desplazamiento de un solo Led, mientras que los otros tres están apagados.

Los Leds deben estar conectados a los pines D5, D6, D7 y D8, y se deben encender y posteriormente apagar los Leds desde el pin D5 al D8, con un tiempo de duración de encendido y apagado de 200 milisegundos.

Con esta práctica se consigue aprender a declarar variables tipo lista de valores, declarar una función y llamarla cuando sea necesario.

El código de programación puede tener varias formas de programarlo, veamos el primero:

```
int tiempo=100; // declara una variable para tiempo de desplazamiento
/* según demos menos tiempo irá más rápido el desplazamiento y al revés */
void setup() {           // comienza la configuración
pinMode(5,OUTPUT);      // pone el pin 5 de salida
pinMode(6,OUTPUT);      // pone el pin 6 de salida
pinMode(7,OUTPUT);      // pone el pin 7 de salida
pinMode(8,OUTPUT);      // pone el pin 8 de salida
}
void loop(){            // comienza el bucle de rutina
digitalWrite(5,HIGH);   // pone nivel alto pin 5
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(5,LOW);    // pone nivel bajo pin 5
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(6,HIGH);   // pone nivel alto pin 6
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(6,LOW);    // pone nivel bajo pin 6
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(7,HIGH);   // pone nivel alto pin 7
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(7,LOW);    // pone nivel bajo pin 7
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(8,HIGH);   // pone nivel alto pin 8
delay(tiempo);          // pausa de 100 milisegundos
digitalWrite(8,LOW);    // pone nivel bajo pin 8
delay(tiempo);          // pausa de 100 milisegundos
}
```

16. Secuencia de Leds

Segunda forma y observar que se reduce el código y se hace por medio de un bucle **for**

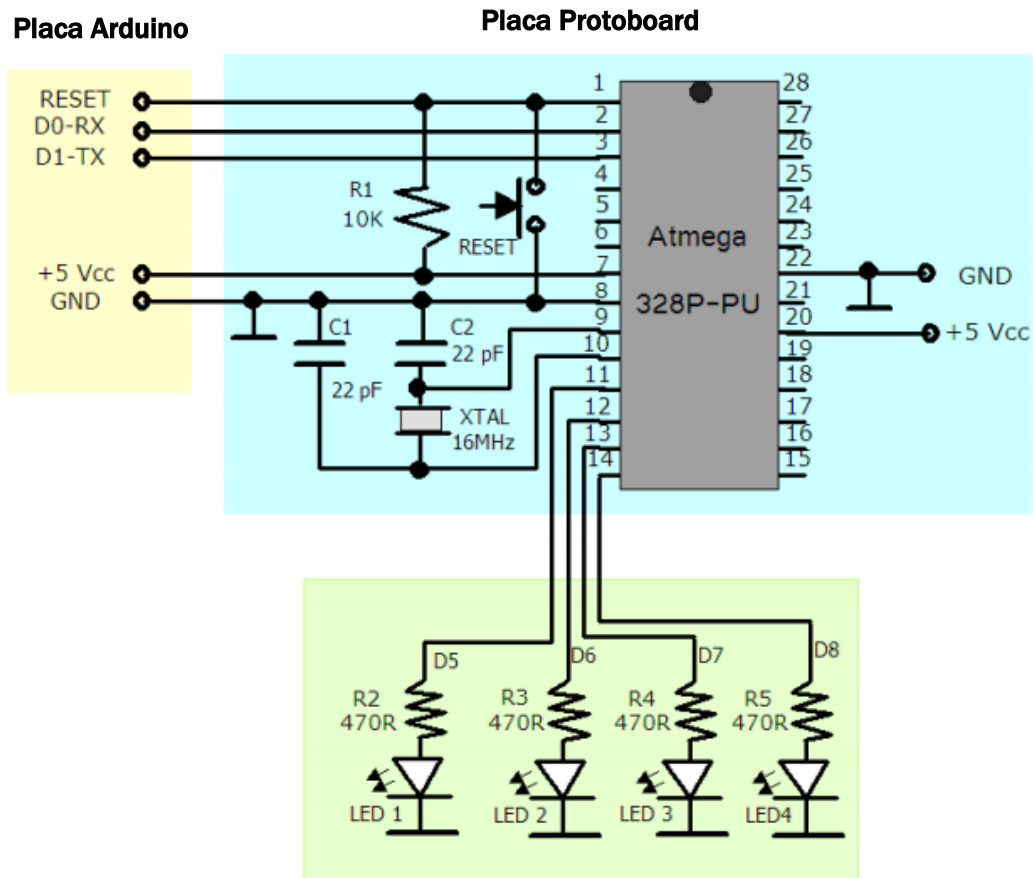
```
int tiempo=200;    // variable tiempo toma el valor de 200 milisegundos
int n=0;          // creamos la variable n
void setup() {    // comienza la configuración
for (n=5; n<9; n++) {    // contador de n del 10 al 14
pinMode(n,OUTPUT);    // configuramos el pin n de salida
}
}
void secuencia() {    // bloque de rutinas secuencia
for(n=5;n<9; n++) {    // damos valores al bucle for
digitalWrite(n,HIGH);    // pone a nivel alto el valor de n
delay(tiempo);    // pausa de 200 milisegundos
digitalWrite(n,LOW);    // pone a nivel bajo el valor de n
delay(tiempo);    // pausa de 200 milisegundos
}
}
void loop(){    // comienza el bucle de rutina
secuencia();    // llama al bloque secuencia
}
```

Otra tercera forma sería la siguiente:

```
int Leds[]= {5,6,7,8}; // declaramos variables tipo lista de valores
int tiempo=200;    // declaramos tiempo con valor 200 milisegundos
int n=0;          // declaramos variable n
void setup() {    // comienza la configuracion
for(n=0; n<4; n++) {    // contador de n de 1 a 4
pinMode(Leds[n],OUTPUT); // toma valores Leds con salida
}
}
void secuencia() {    // bloque de instrucciones repetitiva
for (n=0; n<4; n++){    // bucle for valores de n hasta 4
digitalWrite (Leds[n], HIGH); // enciende leds
delay(tiempo);    // pausa de 200 milisegundos
digitalWrite (Leds[n], LOW); // apaga leds
delay(tiempo);    // pausa de 200 milisegundos
}
}
void loop(){    // configuración bucle
secuencia();    // llama al grupo secuencia
}
```

16. Secuencia de Leds

Esquema de conexionado de la placa Arduino a la placa de prototipo con los cuatro Leds conectados del D5 al D8.



17. Secuencia de Leds con pulsador

Esta práctica es similar a la anterior pero en este caso se le añade un pulsador que establece la secuencia de Leds. Se trata, pues de, encender y apagar 4 Leds secuencialmente al accionar un pulsador. El pulsador debe estar conectado al pin D4 y los Leds a los pines D5, D6, D7 y D8.

Se deben encender y posteriormente apagar los Leds desde el pin D5 al D8, con un tiempo de duración de encendido y apagado de 200 milisegundos.

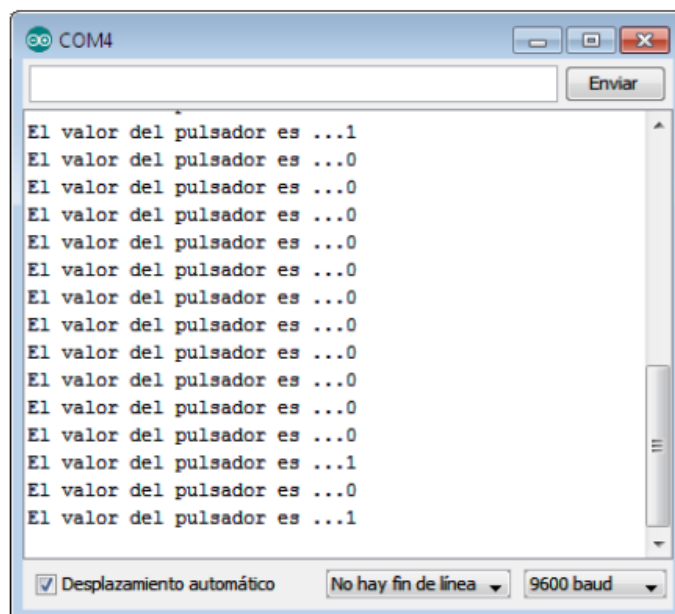
En este caso nos vamos a familiarizar aprendiendo a conectar una entrada digital a arduino (pulsador), a declarar variables tipo lista de valores y declarar función y llamarla cuando sea necesario.

```
int cadenaleds[]={5,6,7,8};    // declara variables de cadena pin 5,6,7 y 8
int pulsador=4;                // declara variable pulsador pin 4
int tiempo=200;                // variable tiempo igual a 200
int n=0;                       // declara variable n
void setup(){                  // comienza la configuracion
for(n=0; n<4; n++) {          // bucle for para n hasta 4
pinMode(cadenaleds[n],OUTPUT); // declaramos cadenaleds de salida
}
pinMode(pulsador,INPUT);      // declaramos pulsador de entrada
void flash() {                 // bloque de secuencias flash
for (n=0; n<4; n++) {
digitalWrite(cadenaleds[n],HIGH); // pone a nivel alto el valor n de la
cadena
delay(tiempo);                // pausa 200 milisegundos
digitalWrite(cadenaleds[n],LOW); // pone a nivel bajo el valor n de la
cadena
delay(tiempo);                // pausa 200 milisegundos
}
}
void loop() {
if(digitalRead (pulsador)==HIGH) { // condicional si pulsador igual alto
flash();                       // ejecuta el bloque flash
}
}
```

17. Secuencia de Leds con pulsador

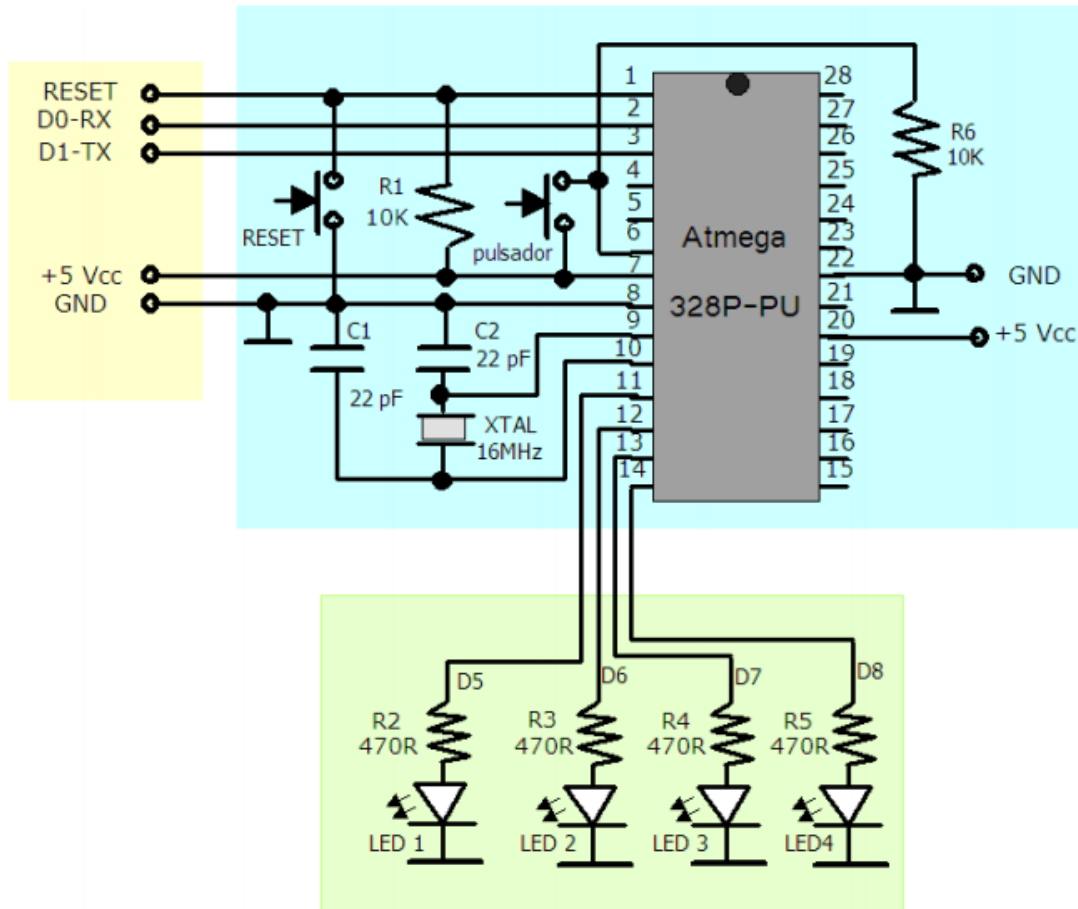
En esta otra forma se puede ver mediante el panel serial el valor del pulsador (pulsado 1) (sin pulsar 0) y cuando sea uno se activa la secuencia de Leds y lo indica en el panel.

```
int leds[]={5,6,7,8};           // declaramos los pin a la variable leds
int tiempo=200;                 // declaramos la variable tiempo igual a 200
int pulsador=4;                 // declaramos pulsador en el pin 4
int n=0;                        // declaramos la variable n
int valorpulsador=0;           // declaramos la variable valorpulsador
void setup(){                  // Comienza la configuracion
  for(n=0;n<4;n++){           // bucle for con n de 1 a 4
    pinMode(leds[n],OUTPUT);   // declaramos Leds de salida
  }
  pinMode(pulsador,INPUT);     // declaramos pulsador de entrada
  Serial.begin(9600);          // activa la comunicacion serial a 9600
}
void monitoriza(){             // bloque de rutinas monitoriza
  Serial.print("El valor del pulsador es ..."); // texto en índice de linea
  Serial.println(valorpulsador); // muestra en línea el valorpulsador
  delay(1000);                 // pausa de 1 segundo
}
void secuencia(){              // bloque de rutinas secuencia
  for(n=0;n<4;n++){           // bucle for de n de 1 a 4
    digitalWrite(leds[n],HIGH); // pone leds a nivel alto
    delay(tiempo);             // pausa de 200 milisegundos
    digitalWrite(leds[n],LOW); // pone leds a nivel bajo
    delay(tiempo);
  }
}
void loop() {
  valorpulsador=digitalRead(pulsador); //leemos pulsador y lo guardamos en
//valorpulsador
  monitoriza();                //funcion monitoriza
  if (valorpulsador==1) {      // si valorpulsador es igual a 1
    secuencia();               //llamamos a la funcion secuencia
  }
}
```



17. Secuencia de Leds con pulsador

Esquema de conexionado de la placa Arduino a la placa de prototipo con los cuatro Leds conectados en D5, D6, D7 y D8 y el pulsador conectado al pin D4 correspondiente al patillaje 6 del Atmega 328P-PU.



18. Luminosidad de un Led en función de la luz exterior

A continuación iremos viendo dos versiones relacionadas con la iluminación artificial en función a la luz solar que recibamos.

En esta primera versión se trata de un dispositivo que haga lucir un Led más o menos en función de la luz externa.

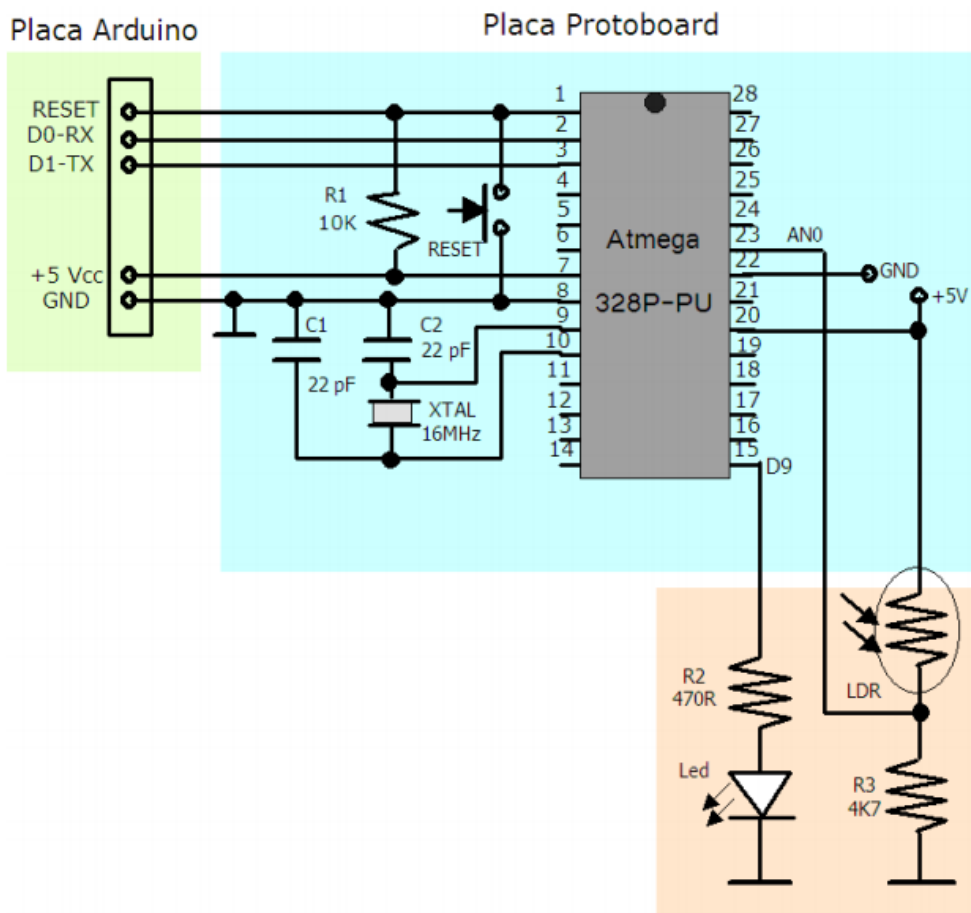
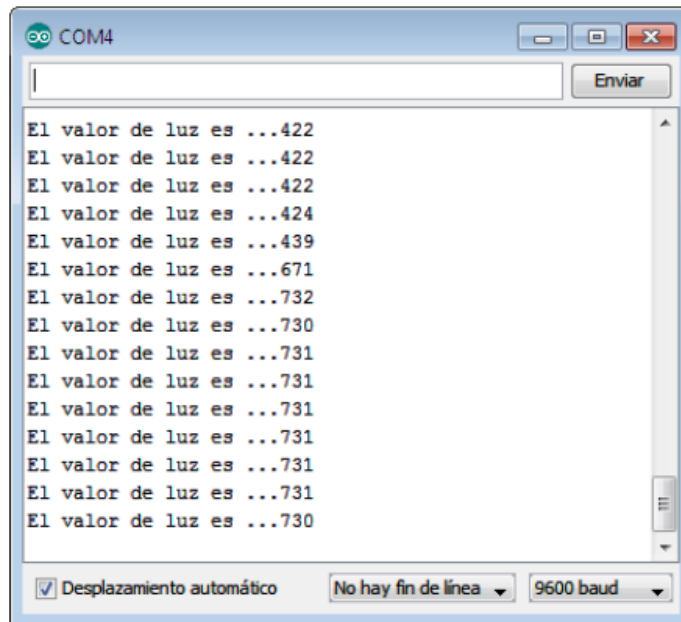
Para ello conectaremos una LDR a la entrada analógica AN0 y un Led al pin D9. Cuando la luz se encuentre entre 0 y 512 el Led debe colocarse en el nivel de potencia máxima (255), si la luz se encuentra entre valores 512 y 1024 el Led debe lucir al nivel de potencia 64. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en la consola del PC.

Se pretende en esta práctica de asimilar conceptos relacionados con las conexiones de entradas analógicas pin AN0 con componentes analógicos en este caso una LDR, conexiones de salidas analógicas pin D9 (PWM), con órdenes como **analogWrite()**, ordenes de control como If, else y la visualización de datos en la consola del PC mediante el puerto serial y el Monitor.

```
int led=9;           // declara Led en el pin D9
int ldr=0;          // declara ldr en el pin AN0
int luz=0;          // declara variable luz
void setup() {      //comienza la configuracion
  pinMode(9,OUTPUT); // el pin D9 es de salida analógica PWM
  Serial.begin(9600); // active la comunicacion serial
}
void monitoriza(){  // bloque de instrucciones monitoriza
  Serial.print("El valor de luz es ..."); // texto indice en la consola
  Serial.println(luz); // escribe el valor de luz
  delay(1000);       // espera 1 segundo
}
void loop(){        // comienza el bucle
  luz=analogRead(ldr); // luz es igual al valor que tengamos en ldr
  monitoriza();       // llama al bloque de instrucciones monitoriza
  if(luz<512 && luz>=0){ // condiciona a un nivel alto entre mayor de 0 y 512
    analogWrite(led,255); // pone el valor a led
  }
  if(luz>=512 && luz<=1024){ // condicionante a nivel alto entre 512 y 1024
    analogWrite(led,64); // pone el valor 64 a led
  }
}
```

18. Luminosidad de un Led en función de la luz exterior

Volcado en pantalla del monitor serie de los diferentes valores de luz que incide en la LDR:



18. Luminosidad de un Led en función de la luz exterior

En ésta otra versión, que es similar a la anterior, pero en este caso consiste en lucir tres Leds más o menos en función de la luz externa. Para ello conectamos una resistencia dependiente de la luz LDR a la entrada analógica AN0 y los tres Leds a los pines D9, D10 y D11.

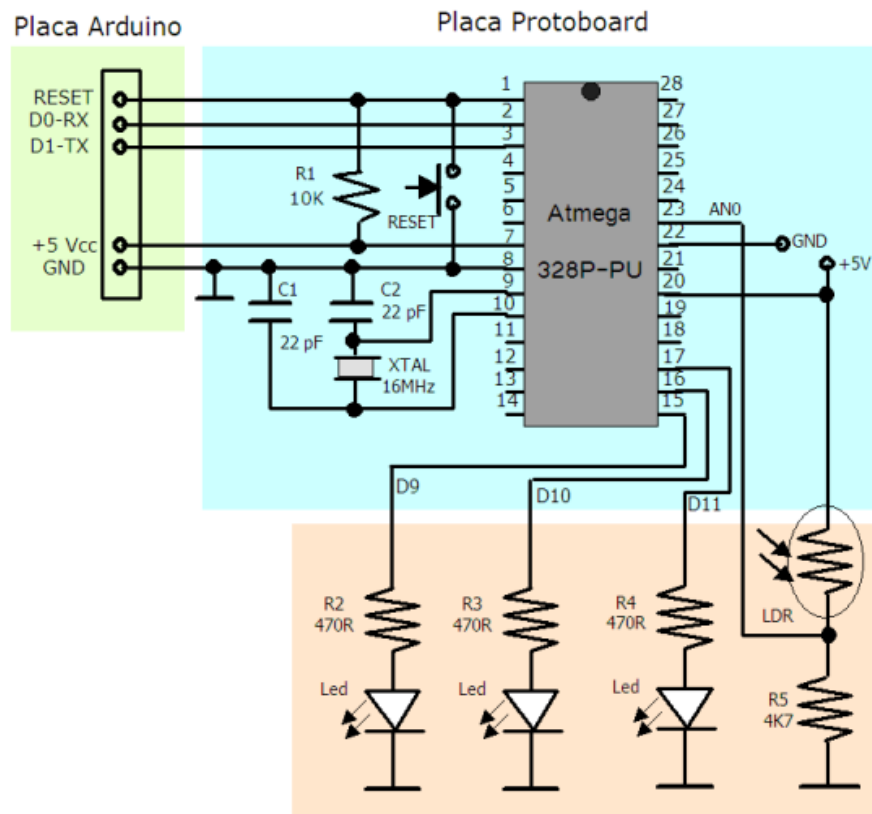
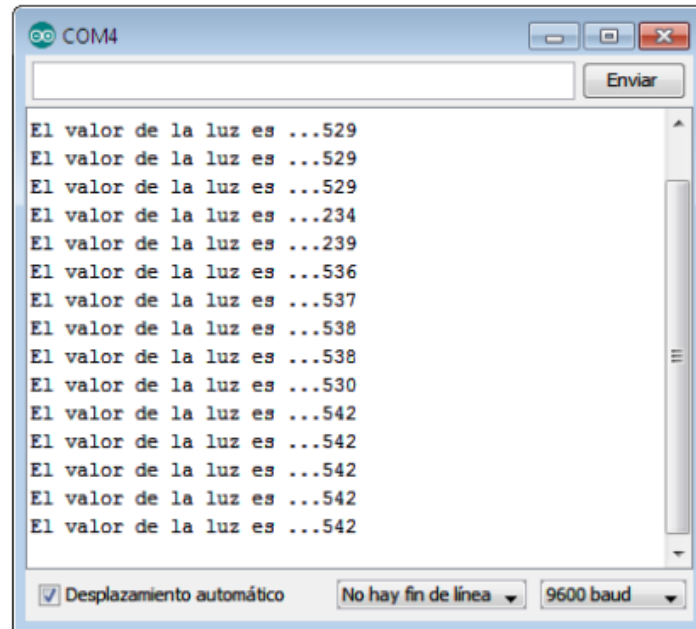
Cuando la luz se encuentre entre los valores 768 y 1023 los Leds deben colocarse en el nivel de potencia 64, si la luz se encuentra entre valores de 512 y entre 768 y 1023 los Leds deben lucir al nivel de potencia 127, si la luz se encuentra entre valores 9 y 255 los Leds deben lucir al nivel de potencia 255. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en la consola del PC a través del Monitor Serial.

En esta práctica recordaremos las entradas analógicas para la fotoresistencia LDR. Las salidas analógicas a través de los pines PWM D9, D10 y D11. Las ordenes **analogWrite()**. Los datos de consola a través del puerto serie y las ordenes **Serial.begin**, **Serial.print** y las ordenes de control de flujo **If**, **else**.

```
int leds[]={9,10,11}; //declaramos los pin de salida analogicas PWM 9, 10
y 11
int tiempo=300; // declaramos el valor de tiempo
int ldr=0; // declaramos la entrada analógica AN0 para la LDR
int n=0; // declaramos variable n
int luz=0; // declaramos variable luz
void setup() { // comienza la configuracion
for(n=0;n<=3;n++){ // bucle contador del valor n
pinMode(leds[n],OUTPUT); // ponemos Leds de salida
}
Serial.begin(9600); // comienza el monitor serial
}
void monitoriza(){ // bloque de instrucciones monitoriza
Serial.print("El valor de la luz es ..."); // visualiza en índice
Serial.println(luz); // escribe el valor de luz
delay(1000); // pausa de 1 segundo
}
void loop(){ // comienza el bucle del programa
luz=analogRead(ldr); // toma luz el valor de la LDR
monitoriza(); // llamada al bloque de instrucciones
if (luz<=1023 && luz>=768){ // condiciona luz entre - 1023 y + 768
for (n=0;n<=3;n++){ // bucle contador del valor n
analogWrite(leds[n],64); // toma valor de n para leds
delay(tiempo); // pausa de 1 segundo
}
}
if (luz<=767 && luz>=512){
for (n=0;n<=3;n++){ //bucle temporizado
analogWrite(leds[n],127); //enciende los leds
delay(tiempo); //espera 300 milisegundos
}
}
if (luz<=511 && luz>=256){
for (n=0;n<=3;n++)
{
analogWrite(leds[n],191);
delay(tiempo); // pausa de 300 milisegundos
}
}
```

18. Luminosidad de un Led en función de la luz exterior

```
}  
if (luz<=255 && luz>=0){  
  for (n=0;n<=3;n++) {  
    analogWrite(leds[n],255);  
    delay(tiempo);           // pausa de 300 milisegundos  
  }  
}  
}
```



19. Control de luminosidad mediante fotocélula LDR

El microcontrolador Arduino es una herramienta con un gran potencial y una gama de usos increíble. Entre estas se encuentra la capacidad que posee de obtener mediciones analógicas de elementos resistivos.

Las resistencias variables como los sensores de luz LCD, los termistores, sensores de esfuerzo, fotocélulas, etc., se conectan a las entradas analógicas para recoger valores de parámetros físicos.

Este ejemplo hace uso de una función para leer el valor analógico de una **LDR** y establecer el encendido y apagado de un diodo led según la cantidad de luz que le llegue al recinto mediante la fotocélula.

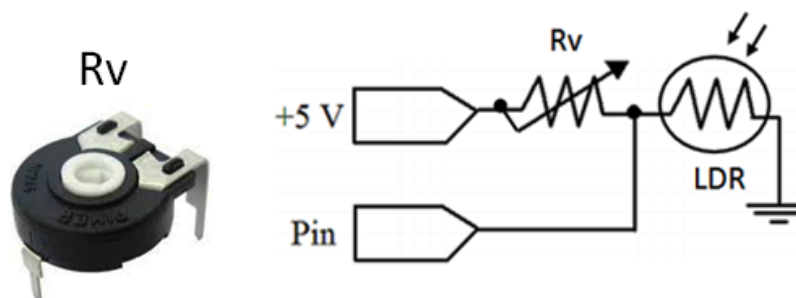


Una **LDR** puede determinar cuándo accionar un determinado dispositivo, por ejemplo un servomotor, una bombilla, un buzzer o un aspersor, en el momento que detecte cierta cantidad de luz.

Una fotorresistencia o **LDR** (*light-dependent resistor*) es un componente electrónico cuyo valor resistivo disminuye con el aumento de la intensidad de luz que inciden en él. Comúnmente fabricados con un semiconductor de alta resistencia como el sulfuro de cadmio (CdS) recubiertos por una placa impermeable y transparente, pueden llegar a valores de $1\text{ M}\Omega$ o más en la oscuridad.

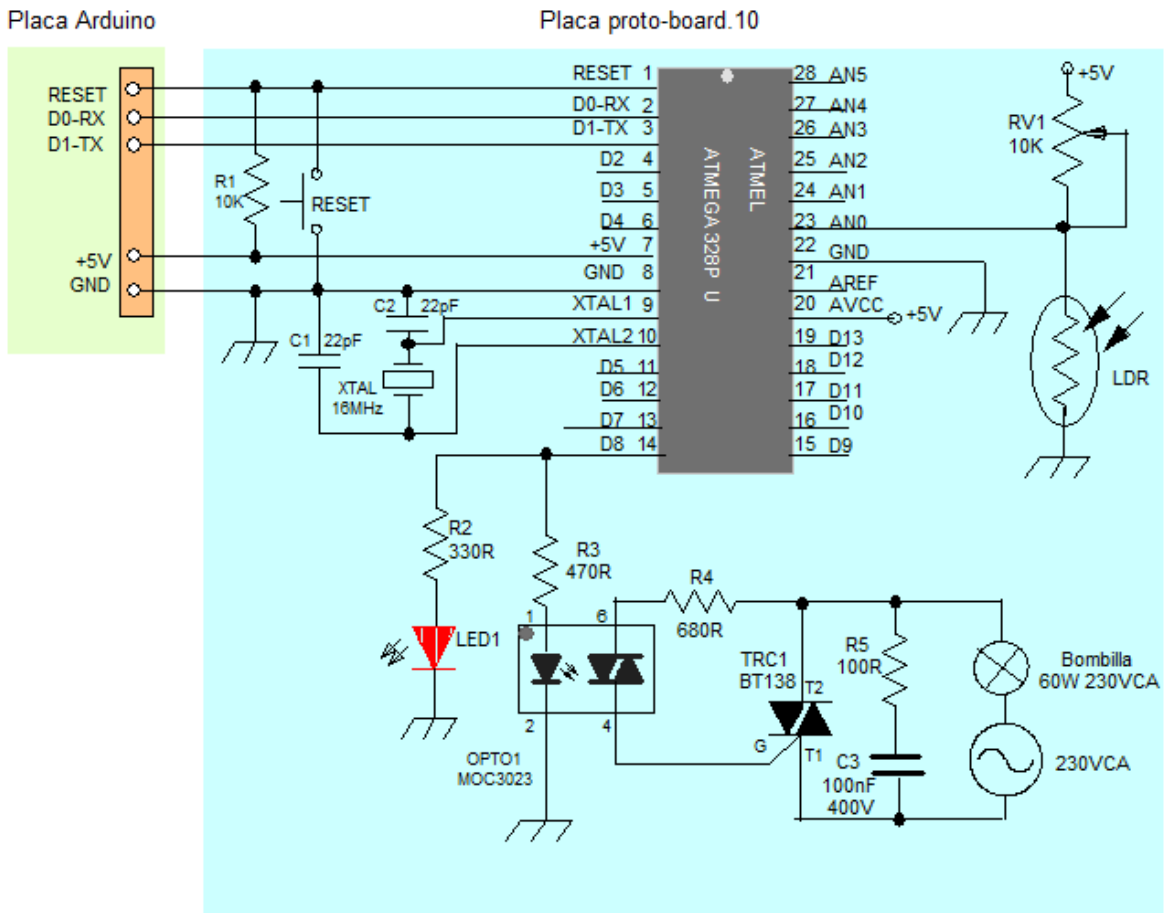
El valor que nos proporciona una **LDR** variará dependiendo de la cantidad de luz que incida sobre ella. Este valor de la resistencia será bajo cuando la luz incida sobre ella y será alto cuando no incida ninguna luz.

En el siguiente esquema la **LDR** está polarizada por una resistencia ajustable **Rv** que nos permite ajustar el punto que deseamos se active el **Pin** y con ello encienda una bombilla, por ejemplo, si está oscureciendo y en el interior del local se encuentra oscuro, retocamos la resistencia ajustable para activar la fotocélula y encender las luces, quedándose en esa posición de ajuste.



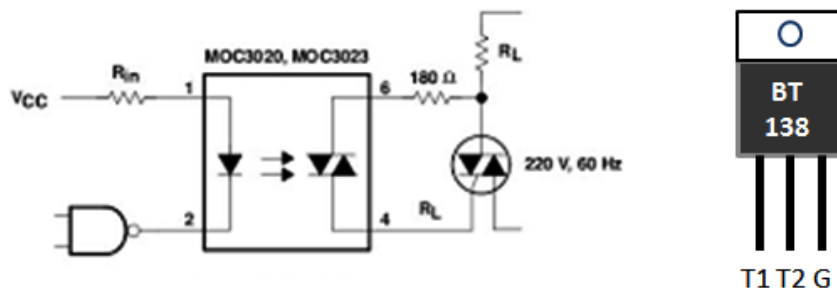
19. Control de luminosidad mediante fotocélula LDR

En el siguiente esquema se muestra la conexión de la fotoresistencia **LDR** mediante una resistencia ajustable de **4K7**, conectada al pin 23, puerto analógico **AN0**, y un diodo **LED1**, conectado al pin 14 del puerto digital **D8**, que nos permite, cuando está encendido, que la fotoresistencia LDR está activa, debido a que no le llega luz.



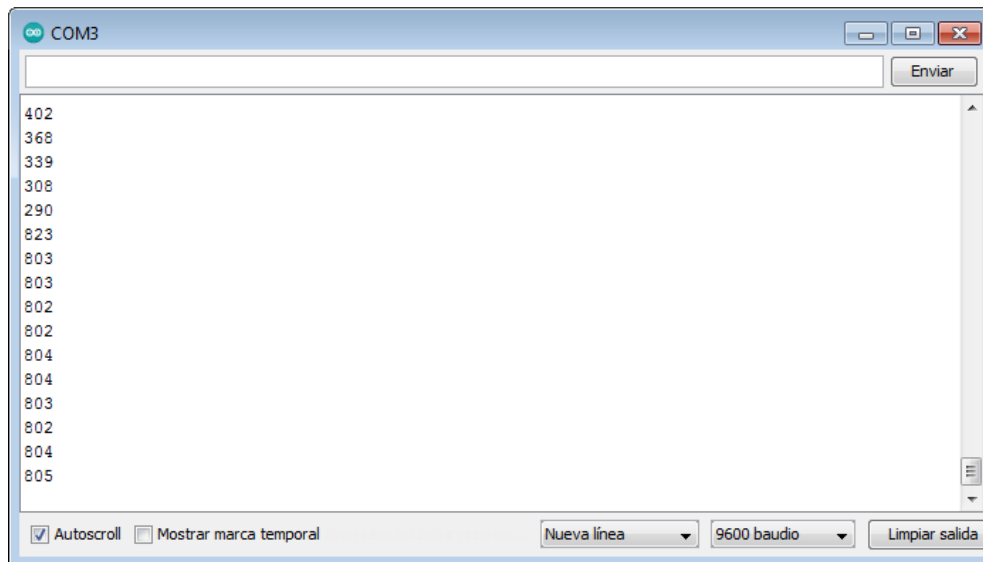
La resistencia ajustable RV1 de 10K se encarga de variar el margen que tenemos seleccionado en la programación, encendido y apagado, de forma que no hay que modificar los valores en la programación.

El circuito que se encarga de encender y apagar la bombilla está formado por un optoacoplador del tipo FotoTriac MOC3023 que se conecta a la puerta G del Triac BT138 y permite encender o apagar las bombillas de 230VCA.



Para la programación necesitamos visualizar los valores que la fotoresistencia LDR proporciona, para ello activamos el monitor serie. Aquí nos aparecen valores desde 0 al 1023, estos valores, dependerá de la luz que incida sobre la LDR y, seleccionar los valores más adecuados para cuando no haya luz se encienda la bombilla. En este caso se ha elegido un valor mayor de 400, que a partir de este valor, es cuando se enciende la bombilla.

19. Control de luminosidad mediante fotocélula LDR



Los valores que aparecen en la imagen del monitor serie, captados mediante la fotocélula LDR están comprendidos entre 0 a 1.023. Estos son los valores que pueden tomar los sensores con Arduino.

```
/*Control de la luminosidad de una fotoresistencia LDR */  
  
int led=8;           //asignamos pin digital D8 a led  
int LDR=A0;         //asignamos pin analogico A0 a la LDR  
int valor;          //asignamos la variable valor  
  
void setup(){  
  Serial.begin(9600); //Habilitamos el monitor serie  
  pinMode(led, OUTPUT); //configuramos el led de salida  
  pinMode(LDR, INPUT); //configuramos LDR como entrada  
}  
void loop() {  
  valor=analogRead(LDR); //guardamos el valor leído de la LDR en valor  
  Serial.println(valor); //visualizamos por el monitor serie los valores de LDR  
  delay(1000);           //esperamos un segundo para ver los datos  
  
  if (valor>400){        //ponemos condicion si valor es mayor de 400  
    digitalWrite(led, HIGH); //si es mayor enciende led  
  }  
  else{  
    digitalWrite(led, LOW); //apaga el led  
  }  
  delay(1000);          //espera un segundo  
}
```

La función **analogRead()**; toma el voltaje de aplicación y lo divide en 1024 partes que corresponden a los 10 bits que maneja por defecto.

Este código obtendrá una medición cada 1 segundos correspondientes a las fracciones del voltaje de aplicación que hay en el punto de medición.

20. Control de las luces del jardín mediante LDR

En esta práctica se emulará un sistema de encendido y apagado de las luces de un jardín en función de la luz ambiente existente.

Cuando en el jardín la luz sea la adecuada (de día), las luces del jardín permanecerán apagadas. Si la luz ambiente decae (tarde-noche), las luces deben encenderse para iluminar nuestro jardín.

Dependiendo del valor de la resistencia, obtendremos diferentes rangos de valores cuando Arduino interactúe con la LDR.

```
/* control luces jardín mediante LDR */

int ldr=0;           // asignamos al pin A0 como pin de entrada analógico
int valor=300;      // valor umbral
int led01=5;        // declaramos pin 5
int led02=6;        // declaramos pin 6
int led03=7;        // declaramos pin 7
int val=0;          // declaramos variable val

void setup() {
  Serial.begin(9600);
  pinMode(led01, OUTPUT); // declaramos led01 de salida
  pinMode(led02, OUTPUT); // declaramos led02 de salida
  pinMode(led03, OUTPUT); // declaramos led03 de salida
  pinMode(ldr, INPUT);    // declaramos ldr de entrada
}
void loop(){

  val=analogRead(ldr);    // lee valor de LDR
  Serial.println(val);    //visualiza por el monitor serial el valor de val
  delay(10000);           //pausa para evitar la histéresis entre los valores de
  activacion
  if(val<=valor){        // si el valor de val es mayor o igual que el umbral se
  enciende
  las luces del jardín
  digitalWrite(led01, HIGH); // enciende led 01
  digitalWrite(led02, HIGH); // enciende led 02
  digitalWrite(led03, HIGH); // enciende led 03
  }
  else {
  digitalWrite(led01, LOW); // apaga led 01
  digitalWrite(led02, LOW); // apaga led 02
  digitalWrite(led03, LOW); // apaga led 03
  }
}
```

El valor de sensibilidad que aparece en el código de la práctica tiene sólo carácter orientativo, ya que habrá que ajustar el valor de dicha variable según la luz ambiente de que disponga en el momento de realizar la práctica.

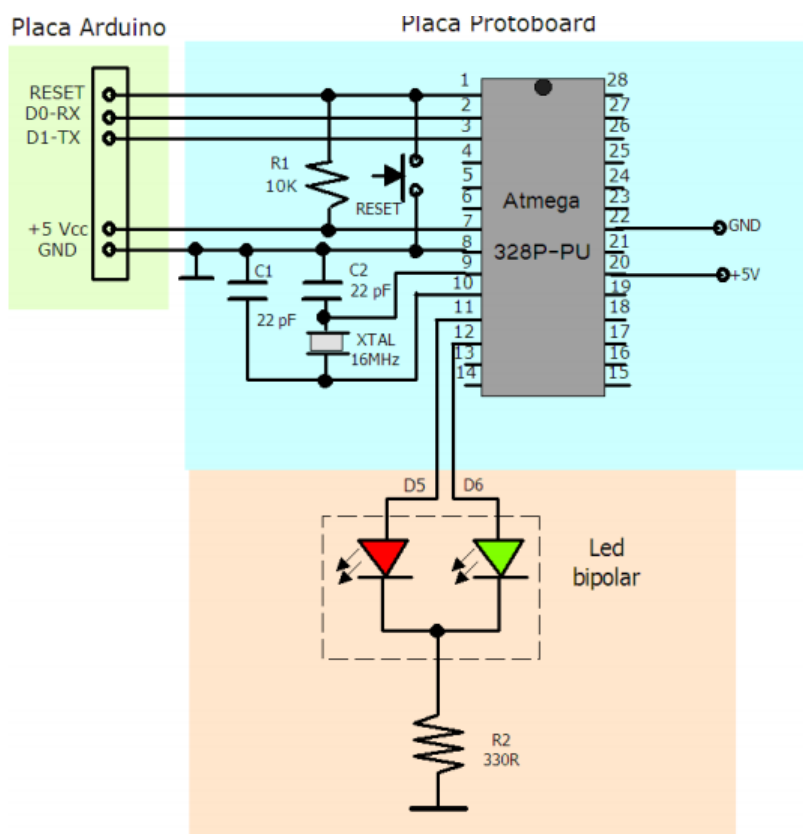
21. Encendido aleatorio de un Led bipolar

Esta práctica consiste en aplicar la función aleatoria **RandomSeed()** y **Random()**, que nos aportarán un poco de aleatoriedad a nuestra práctica para trabajar con estas instrucciones.

En esta práctica se creará un circuito en el que un Led bipolar emule la luz de una vela, es decir, deberá cambiar su intensidad luminosa de forma aleatoria.

Para ello se utiliza un diodo Led bipolar que consiste en dos diodos Leds de color verde y rojo y con los terminales del cátodo unidos en común, integrados en una misma capsula. Se establece la siguiente rutina que produce el encendido aleatorio de los dos Leds. El efecto se asemeja a una vela artificial o el fuego en un belén.

```
/* aleatoriedad de dos led simulando una vela o fuego*/
int max=255;           // declaramos una variable asignando valor 255
int min=255;          // declaramos una variable asignando valor 255
void setup (){
  pinMode (5, OUTPUT); // declaramos pin digital 5 de salida
  pinMode(6, OUTPUT);  // declaramos pin digital 6 de salida
}
void loop (){
  randomSeed (millis()); // creamos una semilla para aplicar
  aleatoriedad
  analogWrite(5, random(max)); // escribimos en el pin 5 valores aleatorios
  delay(25); // mantener visible durante 25 milisegundos
  analogWrite(6,random(min)); // escribimos en el pin 6 valores aleatorios
  delay (25); // mantener visible durante 25 milisegundos
}
```



22. Termostato

A continuación veremos unas prácticas dedicadas al sensor de temperatura (termostato). En esta práctica se trata de un dispositivo que haga funcionar un motor y un Led cuando la temperatura supera cierto valor umbral. Para ello conectaremos una resistencia dependiente de la temperatura **NTC** a la entrada analógica AN0, un Led al pin D5 y un motor de corriente continua al pin D10. Cuando la temperatura llegue a cierto umbral de voltaje (entre 0 y 1024) que nosotros decidamos, se conectarán a la vez el diodo Led y el motor que puede tener unas aspas de ventilador en su eje para enfriar la NTC. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en el Monitor serial de la consola del PC.

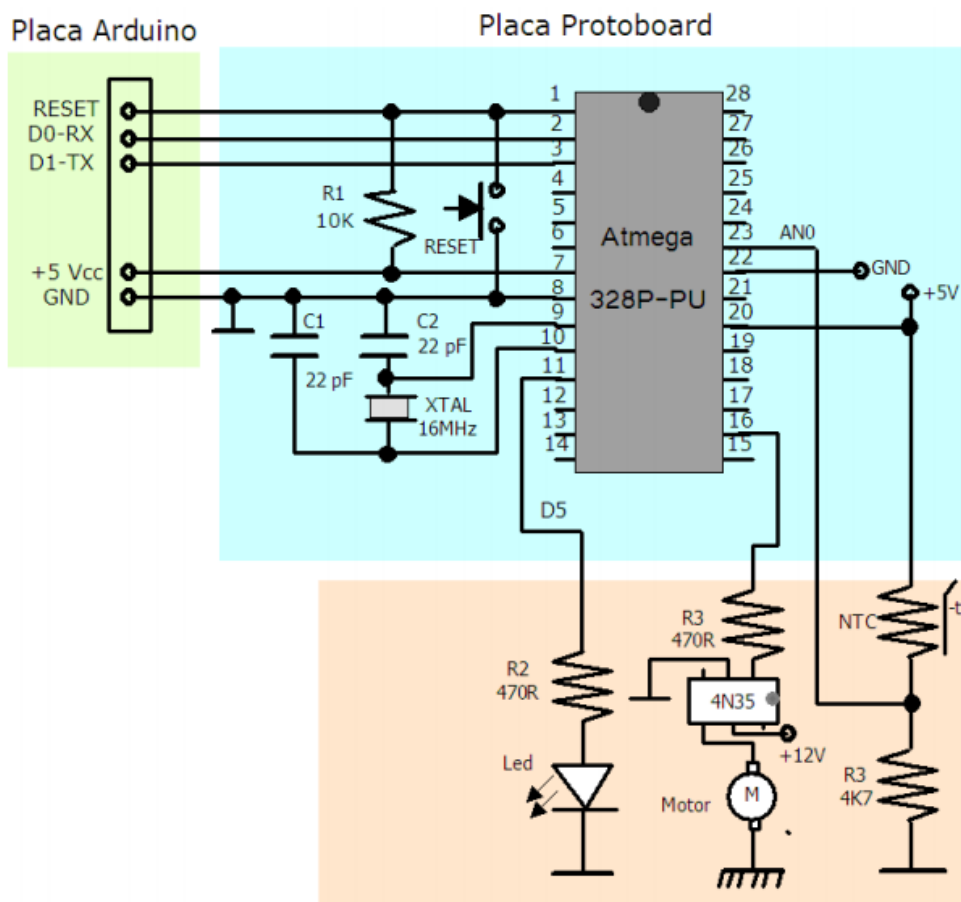
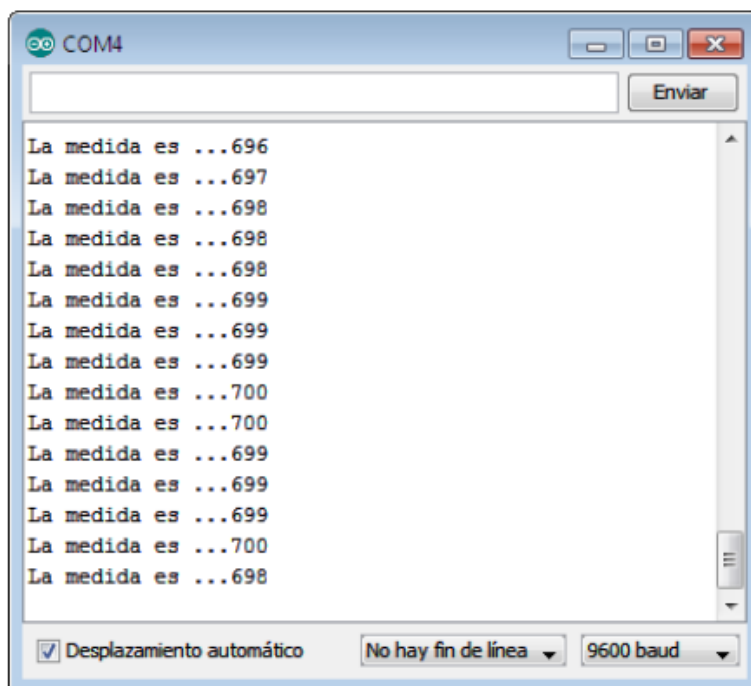
La resistencia **NTC** (*Coeficiente de Temperatura Negativo*) es una resistencia dependiente de la temperatura, conforme detecta el aumento de la temperatura su resistencia interna comienza a disminuir, por lo tanto, lo que la caracteriza es que a mayor temperatura menor resistencia.

Repasaremos las conexiones de entrada analógicas AN0 para el sensor NTC. Utilizaremos órdenes como **analogRead()**. Visualizaremos datos de la consola del PC a través del puerto serie y el Monitor Serial utilizando las ordenes **Serial.begin**, **Serial.print** y por último veremos ordenes de control **if**, **else**.

```
int led=5;           //asignamos el puerto digital D5 a led
int ntc=0;          //asignamos el puerto analógico A0 a ntc
int motor=10;       //asignamos el puerto digital D10 a motor
int medida=0;       //asignamos la variable medida
int nivel=700;      //asignamos la variable nivel con el valor 700
void setup() {
  pinMode(led,OUTPUT); //configuramos el led de salida
  pinMode(motor,OUTPUT); //configuramos el motor de salida
  pinMode(ntc,INPUT); //configuramos la ntc de entrada
  Serial.begin(9600); //configuramos el monitor serie a 9600
}
void monitoriza(){ //procedimiento que envía al puerto serie, para ser leído
en el monitor
  Serial.print("La medida es ..."); //muestra por monitor serie el texto
  Serial.println(medida); // muestra por el monitor serie valor medida
  delay(1000); //para evitar saturar el puerto serie
}
void loop() {
  medida=analogRead(ntc); //guardamos en medida los datos de ntc
  monitoriza(); //ve a monitoriza
  if(medida>nivel) { //si la señal del sensor supera el nivel marcado
    digitalWrite(led,HIGH); //se enciende un aviso luminoso
    digitalWrite(motor,HIGH); //arranca el motor
  }
  else { // si la señal está por debajo del nivel marcado
    digitalWrite(led,LOW);
    digitalWrite(motor,LOW); // el motor se para
  }
}
```

22. Termostato

En la consola del PC en el monitor serial se observa como aplicando calor a la resistencia NTC se percibe las variaciones de la medida hasta conseguir activar el motor y Leds. En este caso en la medida 700.



23. Termostato con velocidad del motor variable

En esta práctica muy semejante a la anterior trata de hacer lucir un Led y funcionar el motor de un ventilador cuando la temperatura llegue a cierto valor umbral (entre 0 y 1024). Para ello conectamos una NTC a la entrada analógica AN0, el Led al pin D13 y el motor en el pin D9. El motor debe funcionar a cierto nivel de potencia a elegir entre 0 y 255. Además se deberá visionar el valor de voltaje en la entrada analógica (valor entre 0 y 1024) en una consola en el PC.

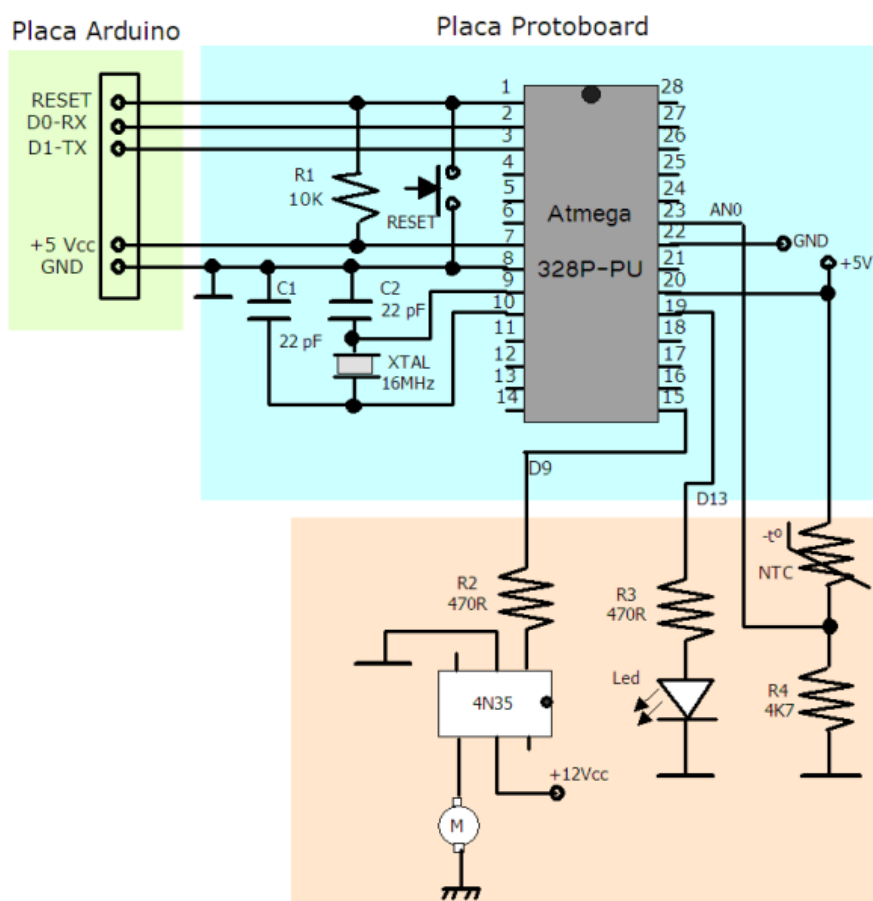
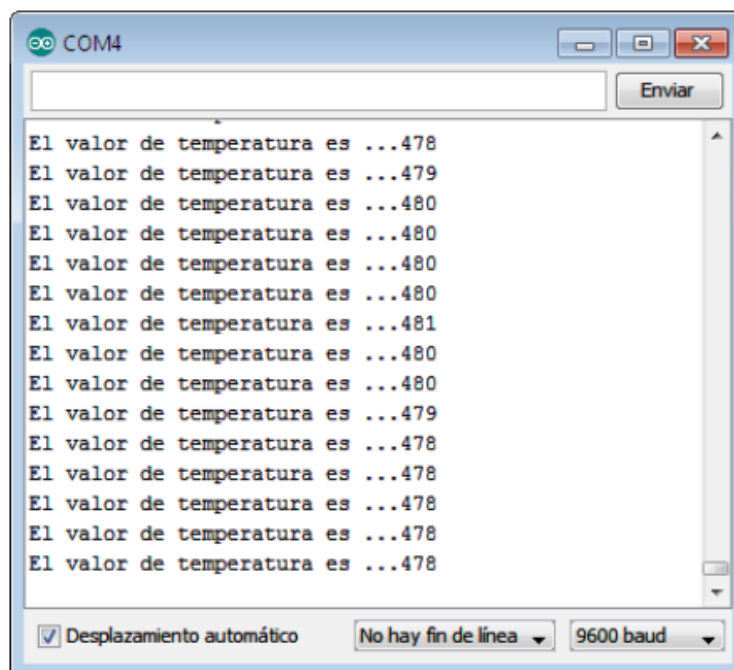
La resistencia **NTC** (*Coeficiente de Temperatura Negativo*) es una resistencia dependiente de la temperatura, conforme detecta el aumento de la temperatura su resistencia interna comienza a disminuir, por lo tanto, lo que la caracteriza es que a mayor temperatura menor resistencia.

Se repasa las conexiones de entrada analógicas para entradas con señales variables (NTC), salidas analógicas a través de pines digitales **PWM**, las ordenes **analogWrite()**, visualización de datos a través de la consola del PC mediante el monitor serial y por último veremos las ordenes condicionales de **If, else**.

```
int motor=9;           //asignamos el pin digital D9 a motor
int led=13;           //asignamos el pin digital D13 a led
int ntc=0;            //asignamos el pin analógico A0 a ntc
int temperatura=0;    // asignamos la variable temperatura a 0
void setup(){
  pinMode(led,OUTPUT); //configuramos led de salida
  pinMode(motor,OUTPUT); //configuramos motor de salida
  pinMode(ntc,INPUT); //configuramos ntc de entrada
  Serial.begin(9600); //configuramos el monitor serie a 9600 baudios
}
void monitoriza() { //creamos la funcion monitoriza
  Serial.print("El valor de temperatura es ..."); //muestra el texto por
  pantalla
  Serial.println(temperatura); //muestra el valor de la temperatura por
  pantalla
  delay(1000); //damos un margen de un segundo para ver los datos
}
void loop(){
  temperatura=analogRead(ntc); //guarda el valor leído de ntc en temperatura
  monitoriza(); // ve a la funcion monitoriza
  if(temperatura>478) { //si temperatura es mayor de 478
    digitalWrite(led,HIGH); //enciende el led
    analogWrite(motor,200); //activa el motor
  }
  else { //de lo contrario
    digitalWrite(led,LOW); //apaga el led
    digitalWrite(motor,LOW); //desconecta el motor
  }
}
```

En la consola del PC en el monitor serial se observa como aplicando calor a la resistencia NTC se percibe las variaciones de la medida hasta conseguir activar el motor y Leds. En ese caso se activará a partir de la medida 479.

23. Termostato con velocidad del motor variable



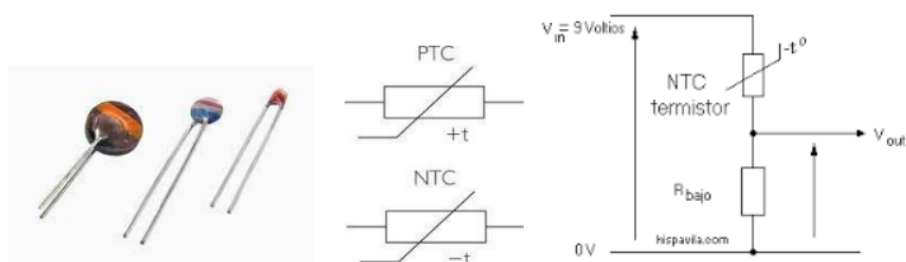
En esta práctica se utiliza un optoacoplador 4N35, constituido por un diodo Led y un fototransistor, que permite controlar un motor en continua de 12 voltios, totalmente aislado de la alimentación de +5 voltios que alimentamos el microcontrolador.

24. Medidor y control de temperatura mediante NTC

Los sensores de temperatura son unos dispositivos muy útiles que se utilizan en sistemas diseñados para medir temperaturas en lugares accesibles o no accesibles por el ser humano.

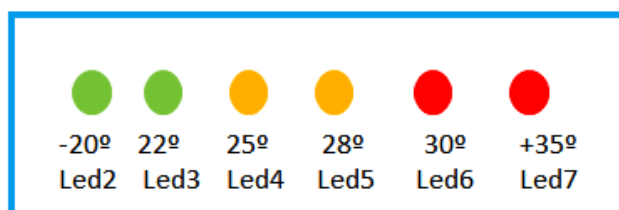
Un ejemplo muy fácilmente donde nos encontramos uno de estos sensores son en los equipos electrónicos que requieren, por su potencia, de un control de la temperatura mediante la activación de ventiladores.

El componente que se va a utilizar en esta práctica es el sensor de temperatura **NTC**. Este componente electrónico varía su resistividad en función de la temperatura ambiente. Al aumentar la temperatura disminuye la resistencia. La relación entre ambos parámetros no es lineal, sino exponencial.

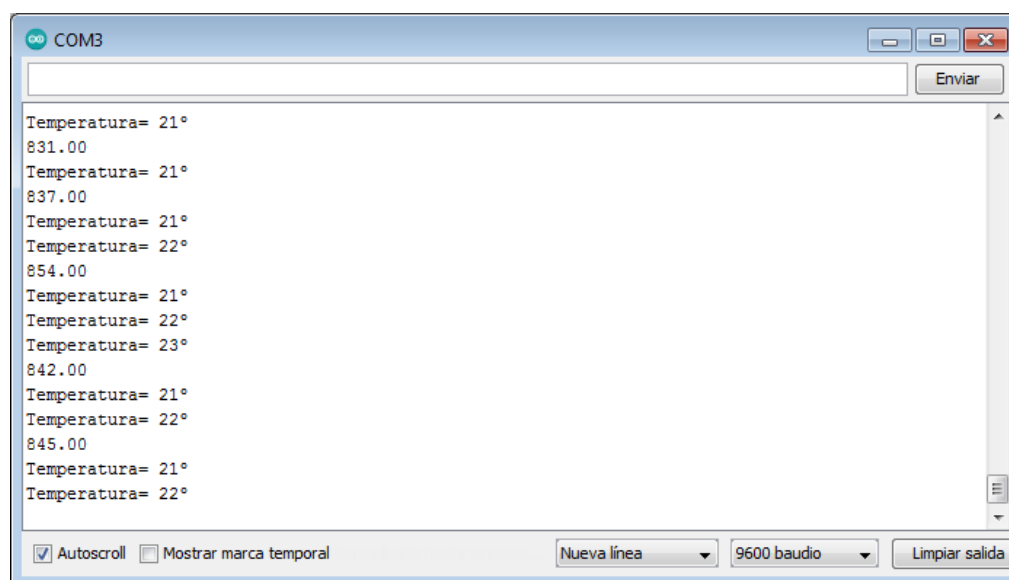


Estructura física y simbología de las NTC y PTC

Esta práctica consiste en controlar la temperatura de un recinto mediante de 6 diodos Led con diferentes colores que van del verde, amarillo y rojo. Cada Led controla una temperatura, comenzando desde los 18º y subiendo a más de los 35º.



Los datos se han conseguido mediante la utilización de la función **Serial.begin(9600)** donde se han ido visualizando los diferentes valores de la NTC según va cambiando la temperatura ambiente.



24. Medidor y control de temperatura mediante NTC

```
/* Medidor de temperatura con el sensor termico NTC */

int valor=0;//asignamos la variable valor a 0
int temp=A0;      //asignamos el valor del puerto analogico A0 a temp
int piezo=A2;     //asignamos el valor del puerto analogico A2 a piezo
int led2=2;       //asignamos el valor del puerto digital 2 a led2 verde
int led3=3;       //asignamos el valor del puerto digital 3 a led3 verde
int led4=4;       //asignamos el valor del puerto digital 4 a led4 amarillo
int led5=5;       //asignamos el valor del puerto digital 5 a led5 amarillo
int led6=6;       //asignamos el valor del puerto digital 6 a led6 rojo
int led7=7;       //asignamos el valor del puerto digital 7 a led7 rojo

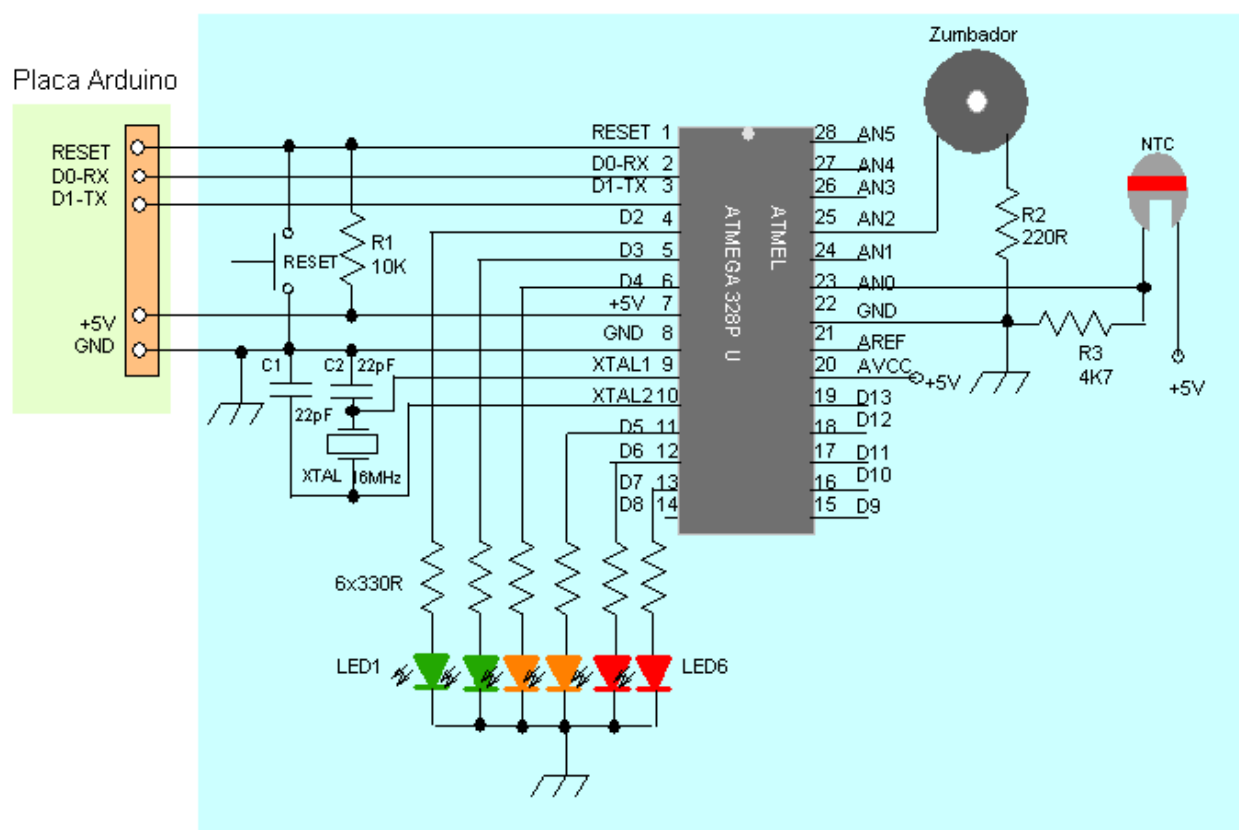
void setup(){
pinMode (temp, INPUT);    //configuramos el pin temp de entrada
pinMode (piezo,OUTPUT);   //configuramos el pin piezo de salida
pinMode (led2, OUTPUT);   //configuramos el pin led2 de salida
pinMode (led3, OUTPUT);   //configuramos el pin led3 de salida
pinMode (led4, OUTPUT);   //configuramos el pin led4 de salida
pinMode (led5, OUTPUT);   //configuramos el pin led5 de salida
pinMode (led6, OUTPUT);   //configuramos el pin led6 de salida
pinMode (led7, OUTPUT);   //configuramos el pin led7 de salida
}
void loop() {
valor=analogRead(temp); //se guarda en valor la lectura de temp

if (valor>780 && valor<819) {
    digitalWrite(led2, HIGH); //enciende el led2 si se encuentra dentro del
margen
}
else {
    digitalWrite(led2, LOW);//de lo contrario apaga el led2
}
if (valor>835){
digitalWrite(led3, HIGH); //enciende el led3 si se encuentra dentro del
margen
}
else {
    digitalWrite(led3, LOW); //de lo contrario apaga el led3
}
if (valor>845){
    digitalWrite(led4, HIGH); //enciende el led4 si se encuentra dentro del
margen
}
else {
    digitalWrite(led4, LOW); //de lo contrario apaga el led4
}
if (valor>855){
    digitalWrite(led5, HIGH); //enciende el led5 si se encuentra dentro del
margen
}
else {
    digitalWrite(led5, LOW);//de lo contrario apaga el led5
}
if (valor>865){
```

24. Medidor y control de temperatura mediante NTC

```
digitalWrite(led6, HIGH);//enciende el led6 si se encuentra dentro del margen
}
else {
  digitalWrite(led6, LOW);//de lo contrario apaga el led6
}
if (valor>875){
digitalWrite(led7, HIGH);//enciende el led7 si se encuentra dentro del margen
tone (piezo,320,500);           //activa tonos f320 d500
tone (piezo,420,500);           //activa tonos f420 d500
delay(500);                       // espera medio segundo
}
else {
  digitalWrite(led7, LOW);// de lo contrario apaga el led7
  noTone (piezo);// desactiva el sonido
}
delay(3000); //espera 3 segundos para actualizar la lectura de temperatura
}
```

Placa proto-board.10



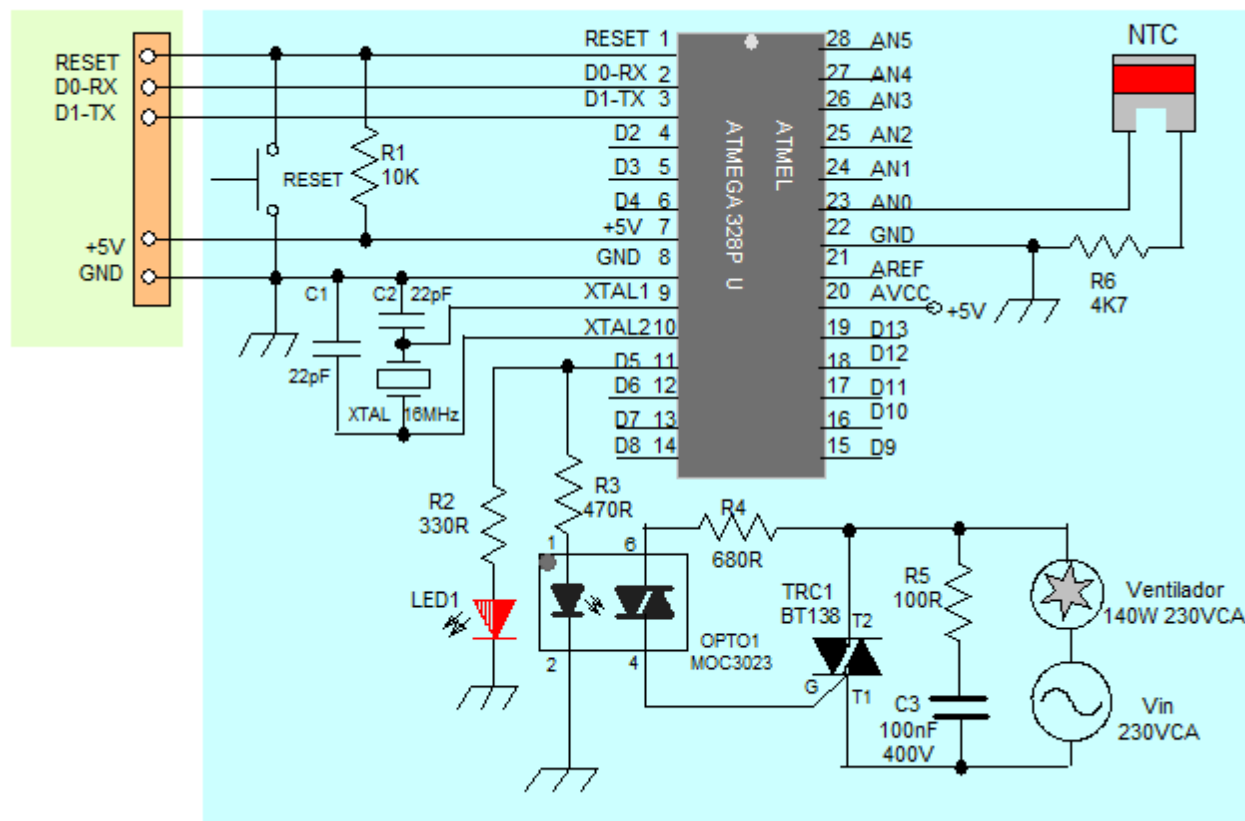
Por otro lado se ha previsto añadir en ésta práctica, relacionada con la resistencia térmica NTC, de solamente una salida D5 que permita controlar un ventilador que se activa cuando la temperatura supera los 25 grados, y se desactiva cuando la temperatura baja por debajo del nivel de los 25 grados.

24. Medidor y control de temperatura mediante NTC

En el siguiente esquema eléctrico se muestra una salida del pin digital D5 del microcontrolador ATmega328P que va a señalar un diodo led rojo LED1 que se encenderá cuando la temperatura exceda de los 25 grados y a la misma vez activará un optoacoplador OPTO SIM3023 que pondrá en marcha el circuito interruptor de 230V de corriente alterna formado por el Triac BT138 que pondrá en funcionamiento el ventilador.

Placa Arduino

Placa proto-board.10



```

/* activacion de un ventilador controlado mediante el sensor de temperatura
NTC */
int valor=0;//asignamos la variable valor a 0
int temp=A0;      //asignamos el valor del puerto analogico A0 a temp
int led5=5;      //asignamos el valor del puerto digital 5 a led5 rojo
void setup(){
pinMode (temp, INPUT);  //configuramos el pin temp de entrada
pinMode (led5, OUTPUT); //configuramos el pin led5 de salida
}
void loop() {
valor=analogRead(temp); //se guarda en valor la lectura de temp

if (valor>845){
    digitalWrite(led5, HIGH);    //enciende el led5 si se encuentra dentro del
margen
}
else {
    digitalWrite(led5, LOW);    //de lo contrario apaga el led5
}
delay(3000); //espera 3 segundos para actualizar la lectura de temperatura
}

```

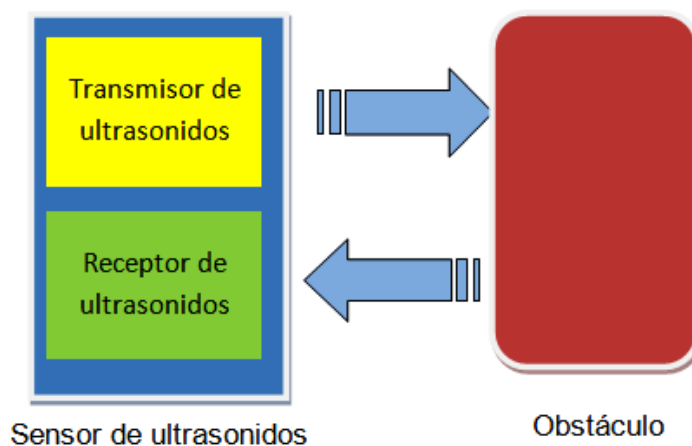
25. Detector de alarma mediante ultrasonidos HC-SR04

Uno de los sensores más utilizados por sus aplicaciones y utilidad es el sensor de ultrasonidos.

Un sensor de ultrasonidos puede servir para medir distancias a objetos, utilizarlo como “sonar” en lugares poco accesibles y como detector de obstáculos para robots móviles y de personas, cuando se mueven por su campo de acción. Para este último caso, se ha pensado programar un sensor de ultrasonidos que nos pueda servir para una sencilla alarma que nos detecte la presencia de una persona o animal en un determinado lugar de una habitación o recinto.

Los ultrasonidos son aquellos sonidos que se generan a partir del rango de los 20KHz hasta los 400KHz y no son percibidos por las personas.

Su funcionamiento consiste en emitir un pulso corto en el rango de los ultrasonidos, este pulso que se emite choca contra un objeto y a su vez es reflejado por el objeto y es entonces cuando el sensor receptor captura el eco producido y lo procesa.



Para averiguar la distancia de un objeto u obstáculo aplicando ultrasonidos, se resuelve aplicando mediante la siguiente formula:

$$d = \frac{v \cdot t}{2}$$

d= distancia
v= velocidad del sonido
t= tiempo de rebote

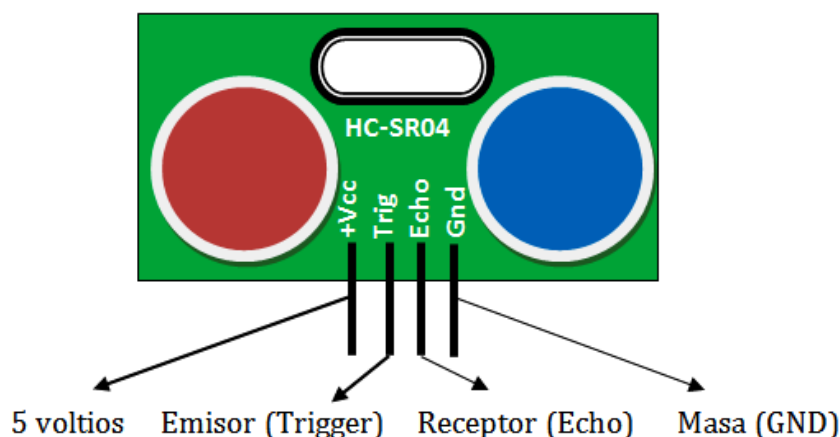
Si sabemos a la velocidad a la que se desplaza el sonido (340m/s), sabemos el tiempo que ha tardado en ir y volver el pulso de ultrasonido, de este modo, podemos saber la distancia a la que se encuentra el objeto.

En esta práctica utilizaremos el sensor de ultrasonido HC-SR04, que es compatible con Arduino.

Este dispositivo, que es uno de los sensores más utilizados en robótica, nos permite medir distancias o detectar objetos. Tanto su conexión como su programación son de fácil implementación.

25. Detector de alarma mediante ultrasonidos HC-SR04

Dispone de cuatro terminales perfectamente serigrafiados para su correcta conexión. Los dos terminales de los extremos son para alimentar el dispositivo (+5Vcc y masa). El terminal *trig* será el encargado de emitir la señal o pulso del ultrasonido y el terminal *echo* será el encargado de recibir la señal o pulsos.



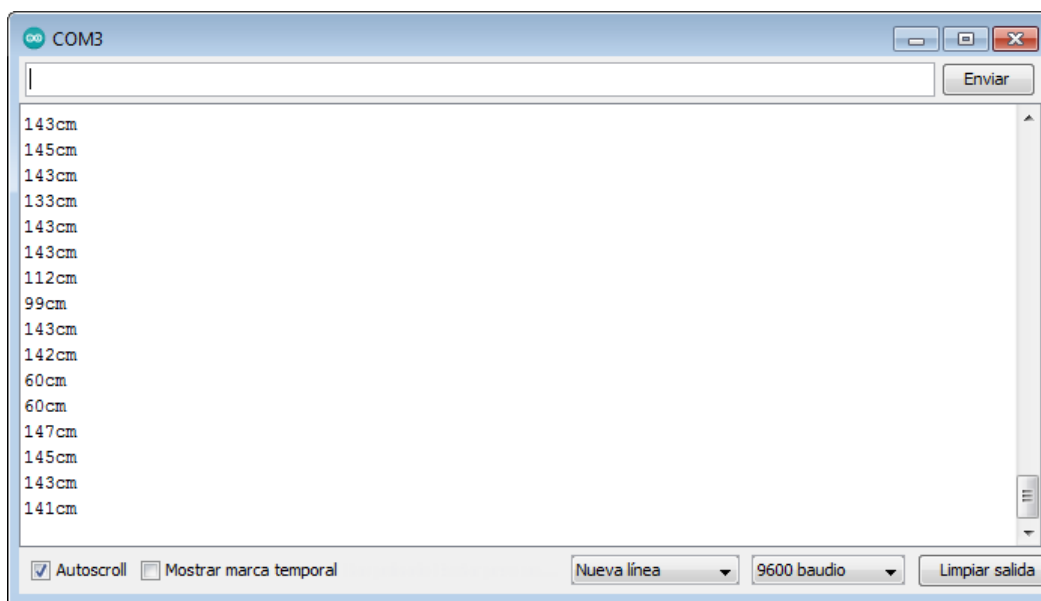
La conexión con Arduino es bastante sencilla: basta con asignar un pin al emisor(trigger) y un pin para el receptor (Echo), y habilitar el monitor serie para ver los valores que van apareciendo según la distancia a la que el objeto u obstáculo está fijo y cuando se produce un cambio de distancia o movimiento del objeto.

Al ser una práctica dedicada a un detector de alarma, el dispositivo de ultrasonidos HC-SR04 se va a encargar de enviar y recibir pulsos de ultrasonido en el interior de un recinto. Cuando la señal emitida y recibida sea la misma, se entiende que el objeto no se ha movido, en este caso no se producirá alarma. En cambio cuando se produce una alteración de los pulsos por la aparición de un cambio de obstáculo o el movimiento de una persona por la zona protegida en este caso se producirá la alarma.

En la programación se ha añadido tres diodos Leds que nos van indicando en todo momento la distancia de tres zonas de detección: distancia alta (led verde), distancia media (led naranja) y distancia corta (led rojo). Tendremos que habilitar el monitor serie para ir viendo por la pantalla del PC el valor desde el 0 al 1024 de las distancias en centímetros que van apareciendo en las diferentes zonas conforme nos movemos por las zonas. Como una opción para avisar de la alarma se puede utilizar la salida del pin D4, donde se encuentra el led rojo y conectarle en paralelo un optoacoplador y éste alimentar un relé.

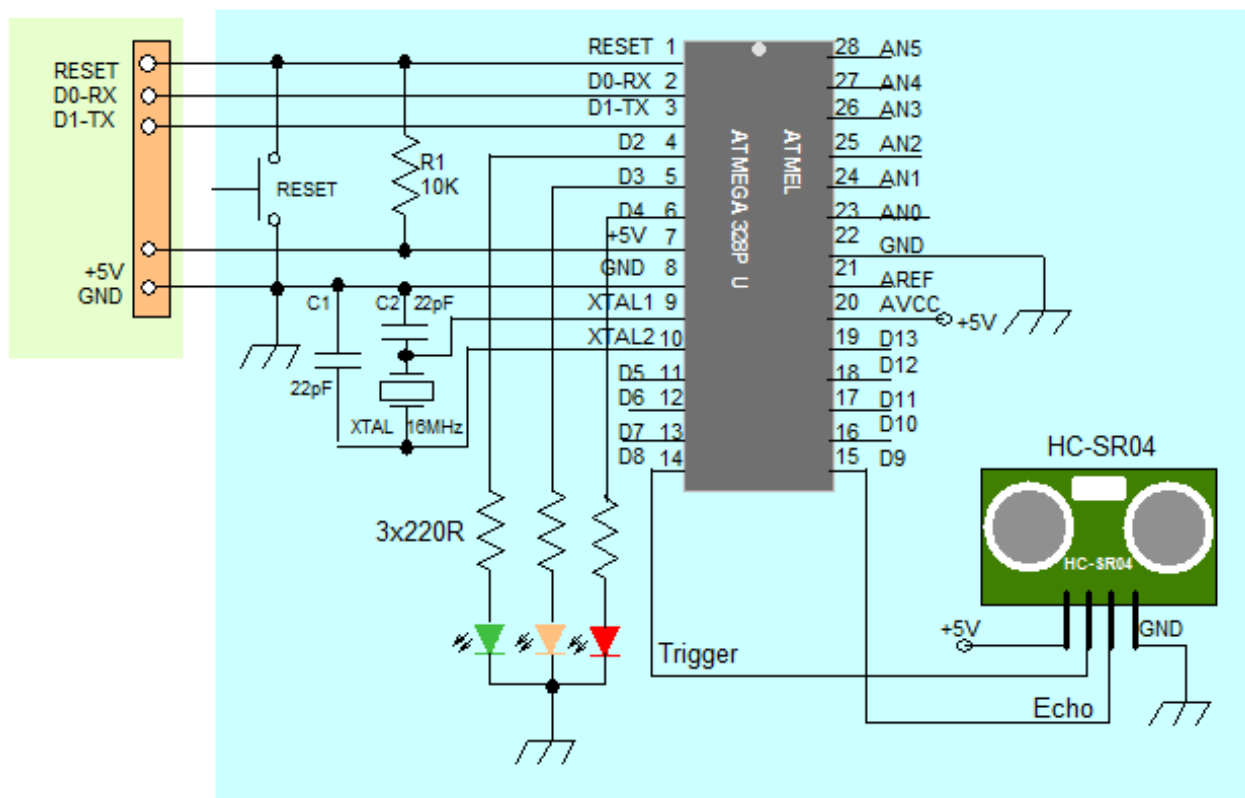
En el monitor serie nos aparece la distancia en centímetros de los objetos u obstáculos que pongamos frente al dispositivo de ultrasonidos. La distancia de detección de este tipo de ultrasonido no es grande, aproximadamente de unos 200 cm. Para ubicarlo en un pasillo, un recinto pequeño, etc.

25. Detector de alarma mediante ultrasonidos HC-SR04



Placa Arduino

Placa proto-board.10



25. Detector de alarma mediante ultrasonidos HC-SR04

```
/* Detector de alarma mediante ultrasonidos HC-SR04 de Arduino */
#define disparo8 //pin digital para enviar el pulso de disparo
#define eco 9 //pin digital para recibir el pulso de recepción o eco
unsigned int tiempo; //variable integer sin signo para almacenar el tiempo
unsigned int distancia; // variable integer sin signo para almacenar
distancia en cm
int led2=2; //asignamos el valor del puerto digital 2 a led2 VERDE
int led3=3; //asignamos el valor del puerto digital 3 a led3 AMARILLO
int led4=4; //asignamos el valor del puerto digital 4 a led4 ROJO

void setup(){
  Serial.begin(9600); //configuramos la comunicación por el monitor serie
  pinMode (disparo, OUTPUT); //configuramos pin de disparo ultrasonido de salida
  pinMode (eco, INPUT); //configuramos pin de eco ultrasonido de entrada
  pinMode (led2, OUTPUT); //configuramos el pin led2 de salida
  pinMode (led3, OUTPUT); //configuramos el pin led3 de salida
  pinMode (led4, OUTPUT); //configuramos el pin led4 de salida
}

void loop() {
  digitalWrite (disparo, LOW); //desactivamos el disparo
  delayMicroseconds(2); //esperamos 2 microsegundos antes de activar el disparo
  digitalWrite(disparo, HIGH); //activamos el disparo
  delayMicroseconds(10); //esperamos 10 microsegundos antes de desactivar el
disparo
  digitalWrite (disparo, LOW); //desactivamos el disparo
  tiempo=pulseIn(eco, HIGH); //guardamos en la variable tiempo la duracion
disparo-eco
  distancia=tiempo/58; //divide el tiempo entre 29µs+29µs tiempo ida y vuelta
  Serial.print(distancia); //visualizamos la distancia por el monitor serie
  Serial.println("cm"); // en centímetros
  delay(1000); //esperamos 1 segundo

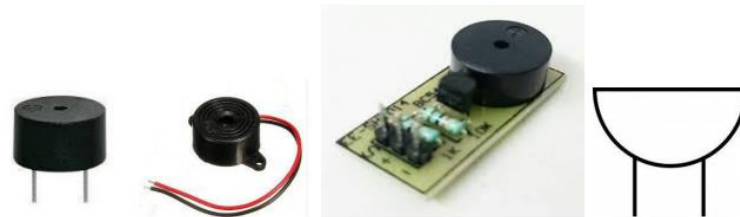
  if (distancia>140) {
    digitalWrite(led2, HIGH); //enciende el led2 VERDE si se encuentra dentro
del margen
  }
  else {
    digitalWrite(led2, LOW); //de lo contrario apaga el led2 VERDE
  }
  if (distancia>110 && distancia<140){
    digitalWrite(led3, HIGH); //enciende el led3 AMARILLO si se encuentra
dentro del margen
  }
  else {
    digitalWrite(led3, LOW); //de lo contrario apaga el led3 AMARILLO
  }
  if (distancia>2 && distancia<110){
    digitalWrite(led4, HIGH); //enciende el led4 ROJO si se encuentra dentro
del margen
  }
  else {
    digitalWrite(led4, LOW); //de lo contrario apaga el led4 ROJO
  }
  delay(500); //espera medio segundo para actualizar la lectura de temperatura
}
```

26. Aviso acústico para aparcamiento

En este ejercicio sobre ultrasonidos nos permite detectar la distancia, alejamiento y cercanía, de un obstáculo que se encuentra inmóvil. Este es un ejemplo típico de lo que sería un avisador por tonos de aparcamiento para un vehículo cuando está realizando la maniobra de marcha atrás.

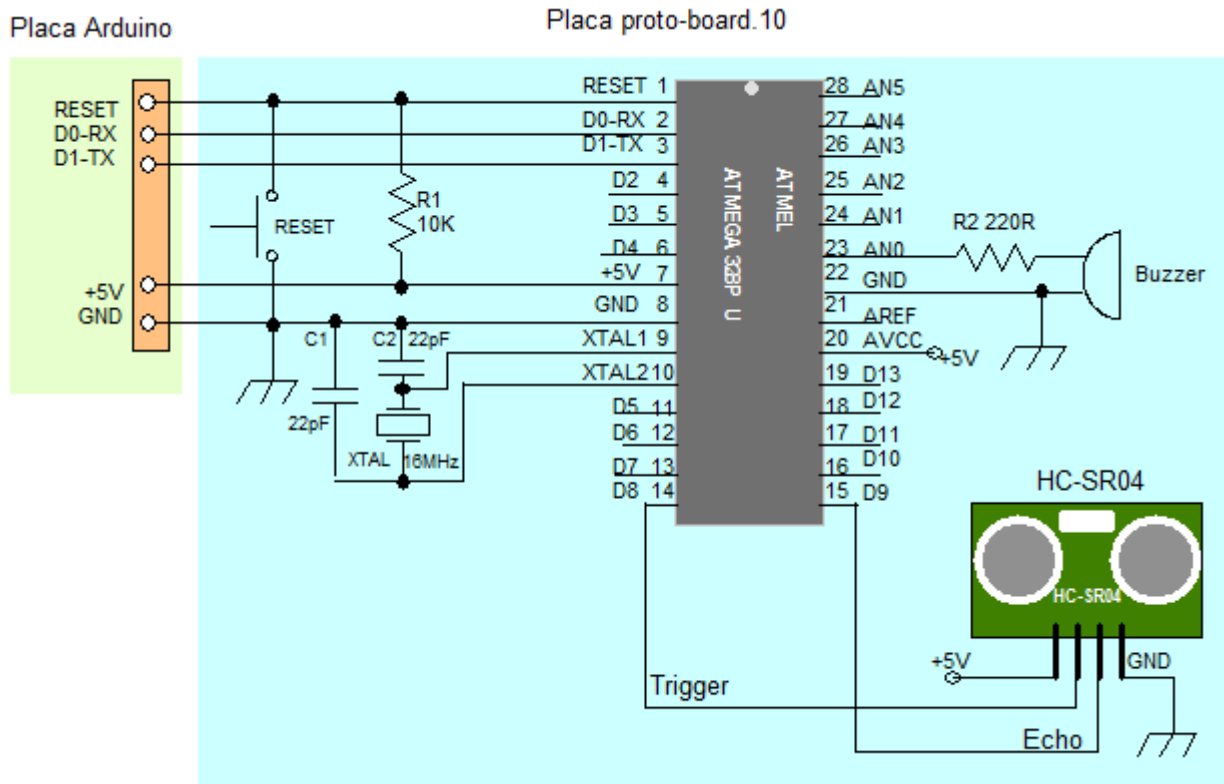
Para ello, se utiliza el dispositivo de ultrasonidos HC-SR04 y un zumbador (buzzer) para avisar cuando más nos acerquemos al obstáculo parado los tonos son más rápidos.

Los zumbadores o buzzer están compuestos de una placa interior que, al igual que la membrana de un altavoz, vibra y transforma la señal en sonido. Su tensión de trabajo varían desde los 5 voltios a los 12 voltios de continua.



Zumbadores o buzzer

Este zumbador, como se muestra en el esquema de conexión, necesita una resistencia de protección de 100 a 220 Ω para la tensión de +5V. Al tratarse de un dispositivo analógico, la conexión del pin positivo deberá establecerse a un pin analógico, en este caso, se ha seleccionado el pin A0, donde se va a generar la salida de tonos, su frecuencia y duración.



26. Aviso acústico para aparcamiento

```
/* Emision de tonos para aparcamiento con el HC-SR04 de Arduino */
#define disparo 8 //pin digital para enviar el pulso de disparo
#define eco 9 //pin digital para recibir el pulso de recepci3n o eco
unsigned int tiempo; //variable integer sin signo para almacenar el tiempo
unsigned int distancia; // variable integer sin signo para almacenar
distancia en cm
int zumbador=A0; //asignamos el zumbador en el puerto analogico A0

void setup(){
Serial.begin(9600); //configuramos la comunicaci3n por el monitor serie
pinMode (disparo, OUTPUT); //configuramos pin de disparo ultrasonido de salida
pinMode (eco, INPUT); //configuramos pin de eco ultrasonido de entrada
pinMode (zumbador, OUTPUT); //configuramos el zumbador de salida
}
void loop() {
digitalWrite (disparo, LOW); //desactivamos el disparo
delayMicroseconds(2); //esperamos 2 microsegundos antes de activar el disparo
digitalWrite(disparo, HIGH); //activamos el disparo
delayMicroseconds(10); //esperamos 10 microsegundos antes de desactivar el
disparo
digitalWrite (disparo, LOW); //desactivamos el disparo
tiempo=pulseIn(eco, HIGH); //guardamos en la variable tiempo la duracion
disparo-eco
distancia=tiempo/58; //divide el tiempo entre 29µs+29µs tiempo ida y vuelta

Serial.print(distancia); //visualizamos la distancia por el monitor serie
Serial.println("cm"); // en centimetros
delay(100); //esperamos 100 milisegundos

if (distancia<50) {
tone(zumbador,320,100); //suena tonos dentro del margen establecido
delay(50); //intervalo de tiempo rapido para tonos
}
else {
noTone(zumbador); //de lo contrario no suena tonos
}
if (distancia>60 && distancia<150) {
tone (zumbador,300,100); //suena tonos dentro del margen establecido
delay(300); //intervalo de tiempo lento para tonos
}
else {
noTone(zumbador); //de lo contrario no suena tonos
}
}
```

27. Control de aforo a un local

En este otro ejemplo consiste en controlar el número de aforo en un local de fiesta. Pues, rebasar el número de asistentes establecido para ese local podría ser motivo de una espléndida multa.

En este caso se va a utilizar una Fotorresistencia LDR y un diodo Laser.

La palabra **Láser** viene del inglés *Light Amplification by Stimulated Emission of Radiation*, o amplificación de luz por emisión estimulada de radiación.

Estos componentes son muy utilizados en los reproductores o grabadores de DVD-ROM, DVD, impresoras, escaneo de códigos de barras, sensores, etc.



Diodo laser concentra toda la luz en un punto



Diodo laser para Arduino

La luz que emitimos por un diodo Led, o una bombilla convencional, por ejemplo, es una luz dispersa, mientras que la luz que emitimos con un diodo laser es una luz concentrada en un punto.

Por lo tanto si nos fijamos en el dibujo siguiente observaremos que si enfrentamos una Fotorresistencia **LDR** y un diodo **Láser** debemos de apuntarlo perfectamente para que coincida el haz del Láser con la superficie de detección de la Fotorresistencia LDR. De lo contrario nos daría problemas a la hora de recibir la señal de la Fotorresistencia LDR.



Fotorresistencia LDR



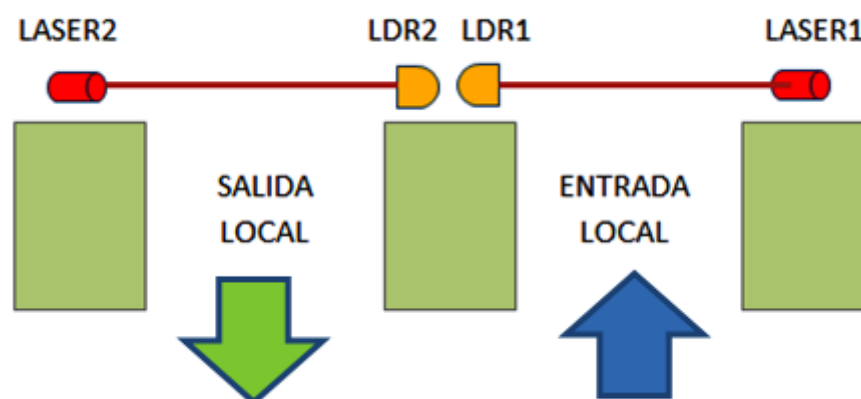
Diodo Láser

Para realizar un sistema de este tipo, de forma mucho más profesional y apropiada, se debería utilizar un emisor y receptor de infrarrojos activos, de solo un haz, creando una barrera de detección, pero como económico y práctico se ha optado por utilizar la Fotocélula LDR y el diodo Láser.

Los componentes que se necesitan en esta práctica consistirán en dos Fotorresistencias LDR y dos diodos Láser, que controlan, uno la entrada y el otro la salida. El sistema debe contabilizar el número de personas que cruzan el acceso de entrada y por otro lado las personas que abandonan el local por la zona de salida y, por la pantalla del PC, mediante el monitor serial del IDE de Arduino, se mostrará en todo momento el número de personas que se encuentran dentro del local.

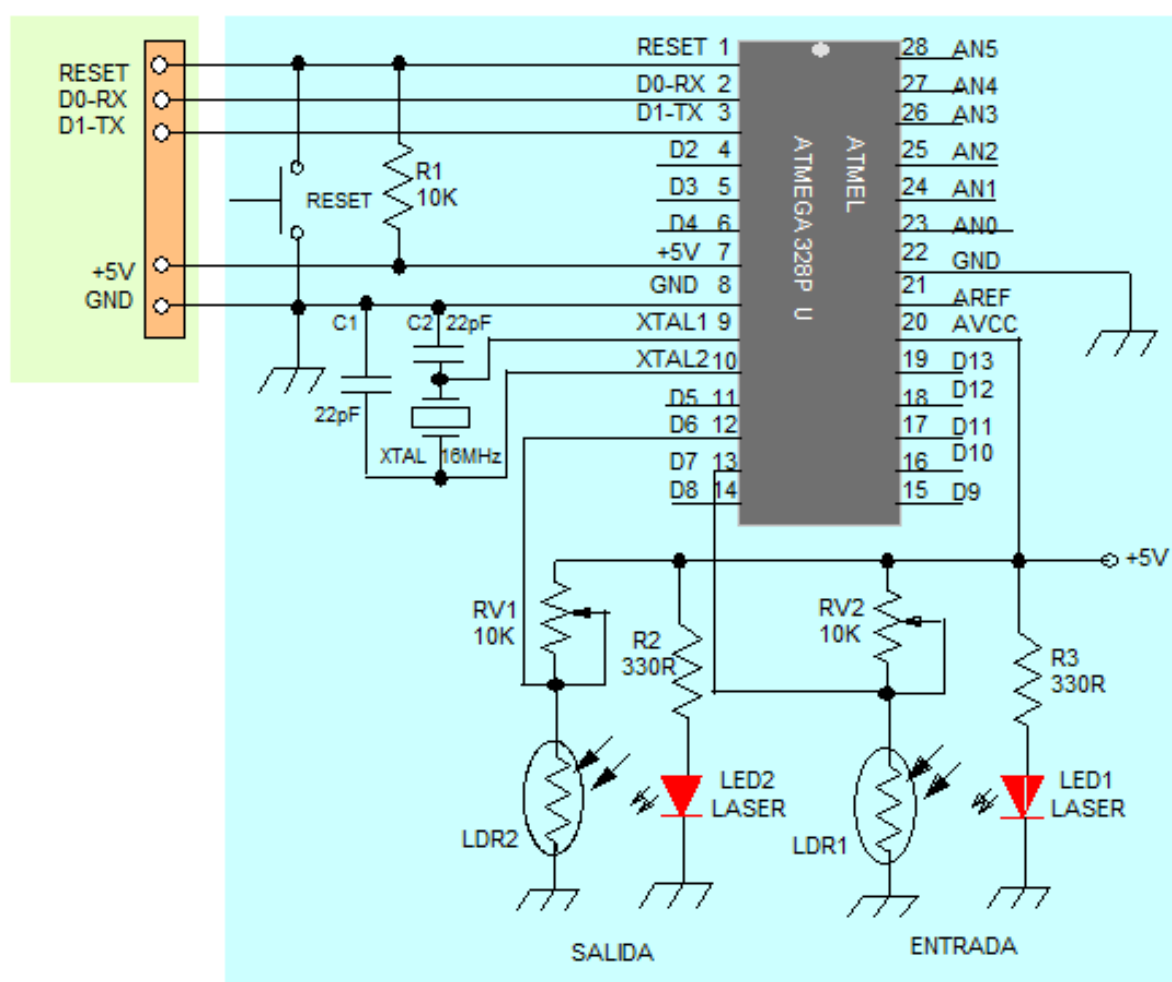
27. Control de aforo a un local

Los diodos láser **LASER1** y **LASER2** estarán conectados y alimentados directamente sin depender de la programación del microcontrolador. La Fotorresistencia **LDR1** (Entrada) va al pin digital **D7** y la Fotorresistencia **LDR2** (Salida) va al pin digital **D6**. El sistema detecta un nivel bajo cuando corta el haz láser.



Placa Arduino

Placa proto-board.10



Las resistencias RV1 y RV2 de 10K son ajustables para obtener un mejor control de la sensibilidad.

27. Control de aforo a un local

```
/* control de aforo a un local*/
/* los diodos laser no se declaran en este sketch y deben de estar siempre
activos y alimentados con +5V y perfectamente alineados con las LDRs*/

int ldr1=7;           //asignamos pin digital D7 conexion LDR1 de entrada
int ldr2=6;           //asignamos pin digital D6 conexion LDR2 de salida
int contadorE=0;      //asignamos contador de aforo de entrada
int contadorS=0;      //asignamos contador de aforo de salida
int contador=0;       //asignamos contador total
int valorE;           //asignamos variable LDR1 para la entrada
int valorS;           //asignamos variable LDR2 para la salida

void setup() {
  Serial.begin (9600); //configuramos monitor serie a 9600 bps
  pinMode (ldr1, INPUT); //configuramos pin ldr1 de entrada
  pinMode (ldr2, INPUT); //configuramos pin ldr2 de entrada
}
void loop() {

  valor1=digitalRead(ldr1); //se lee el valor de la LDR1 de entrada
  if (valor1==LOW) {        // si la LDR1 de entrada detecta corte de luz
contadorE++;                //almacena cada vez que se corte la luz
  delay(1000); /* los retardos utilizados evita que la visualización de los
datos en el monitor serie se vean muy rapidos */
  }
  valor2=digitalRead(ldr2); //se lee el valor de la LDR2 de salida
  if (valor2==LOW) {        //si la LDR2 de salida detecta corte de luz
contadorS++;                //almacena cada vez que se corte la luz
  delay(1000);              //esperamos un segundo
  }
contador=contadorE-contadorS; /* establecemos el valor de contador
con la diferencia entre el valor del contadorE de entrada
y el valor del contadorS de salida */

if (contador==200) { /*establecemos el número maximo de aforo al local de 200
personas si llega a este valor*/
Serial.println ("ATENCION!! AFORO COMPLETO"); //se visualiza en el PC
  delay(1000); //esperamos un segundo
  }
  else {
    Serial.println (200-contador); /* visualiza en el PC el numero de
personas que estan dentro del local*/
    delay(1000); //esperamos un segundo
  }
}
}
```

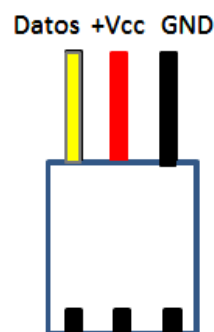
28. Control de Servomotores

Los servomotores analógicos y digitales, son iguales en cuanto al nivel de usuario: conformados por la misma estructura (motor DC, engranajes reductores, potenciómetro y placa de control) y se controlan con el mismo tipo de señal **PWM**. La principal diferencia se debe a la adición de un microprocesador en el circuito de control de los servomotores digitales, el cual se encarga de procesar la señal **PWM** de entrada y controlar el motor mediante pulsos con una frecuencia 10 veces mayor a la frecuencia de los servomotores analógicos. Dentro de los servomotores de corriente continua hay dos tipos: los clásicos de escobillas, y los de nueva tecnología sin escobillas BLDC.

Para programar un servomotor en Arduino podemos utilizar una librería especial para los servomotores, esta librería se denomina **Servo.h** y viene por defecto en el IDE de Arduino. Las instrucciones más usuales que podemos introducir en un programa para controlar al servomotor son:

- **Attach()**: asocia la variable servo a un pin de Arduino. Por ejemplo `servo.attach(pin)`.
 - Pin: pin de Arduino
- **Deattach()**: realiza la función de desactivar o desasociar el servomotor con el pin asociado en la instrucción `attach()`.
Por ejemplo: **servo.deattach(pin)**
 - Pin: pin de Arduino
- **Write()**: escribe un valor para el servo, controlando así su eje. Si el servo es estándar, ajustará el ángulo de rotación en grados. Si el servo es de rotación continua, ajustará la velocidad del servo, donde 0 será la velocidad máxima en una dirección, 180 será la velocidad máxima en la otra dirección y 90 servirá para detener el servo. Por ejemplo, **servo.write(90)**
- **Read()**: leerá el ángulo actual del servomotor, que es pasado por la instrucción `write()`. Por ejemplo, **servo.read()**

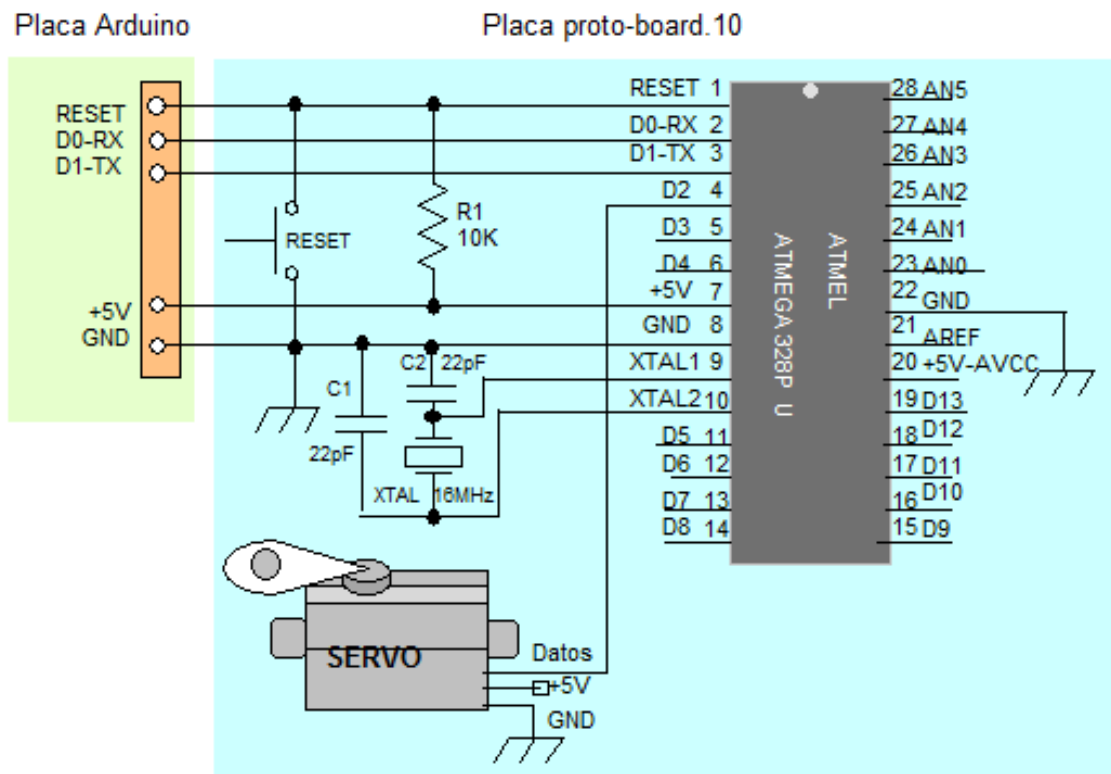
El funcionamiento del servomotor se debe a la modulación por ancho de pulso (PWM), cuentan con tres cables, dos para alimentación Vcc y Gnd (variando de 4.8 V a 6 V - verificar hoja de datos) y un tercer cable para aplicar un tren de pulsos de control, con la finalidad de que el circuito de control diferencial interno ubique el servomotor en la posición indicada.



La conexión del servomotor normalmente posee tres cables de conexión:

- Cable amarillo para el pin de datos.
- Cable rojo para los +5 voltios
- Cable negro para el negativo o GND

28. Control de Servomotores



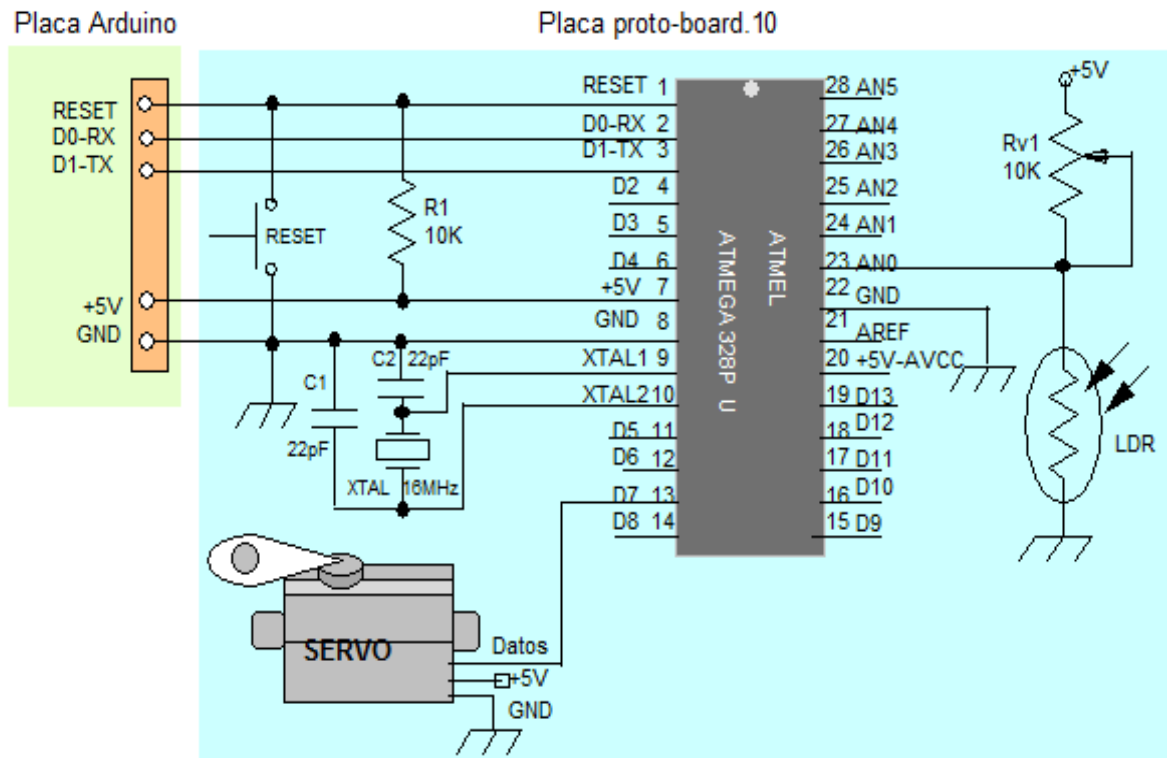
```
int servoPin = 2; // servo conectado al pin digital 2
int myAngle; // angulo del servo de 0-180
int pulseWidth; // anchura del pulso para la función servoPulse
void setup() {
  pinMode (servoPin, OUTPUT); // configura pin 2 como salida
}
void servoPulse(int servoPin, int myAngle){
  pulseWidth=(myAngle*10) + 600; // determina retardo
  digitalWrite(servoPin, HIGH); // active el servo
  delayMicroseconds(pulseWidth); // pausa
  digitalWrite(servoPin, LOW); // desactiva el servo
  delay(20); // retardo de refresco
}
void loop(){// el servo inicia su recorrido en 10º y gira hasta 170º
  for (myAngle=10; myAngle<=170, myAngle ++){
  servoPulse (servoPin, myAngle); // servo vuelve desde 170º hasta 10º
  }
  for (myAngle=170; myAngle>=10; myAngle);
  }
}
```

En este otro ejercicio gobernaremos un servomotor mediante una LDR, emulando un toldo o persiana automatizada que reacciona a la cantidad de luz.

Cuando la luz que incide por la ventana sea mayor a un umbral determinado, la persiana deberá bajar para atenuar la luz.

Todo lo contrario deberá suceder cuando la luz que incide en la ventana sea escasa. En este caso, la cortina deberá replegarse para dejar pasar la máxima luz del exterior.

28. Control de Servomotores



```

/* codigo para el control de una persiana según la luz que incida */

#include <Servo.h> //incluimos la librería Servo.h
int ldr=A0; //asignamos el pin analogico A0 de entrada para sensor de luz
int sensibilidad=95; //variable que contiene el valor umbral
int val=0; //variable para almacenar el valor adquirido por la LDR
Servo motor_persiana; //creamos el objeto motor_persiana

void setup() {
Serial.begin(9600); //configuramos el monitor serie
pinMode (ldr, INPUT); //configuramos el pin como entrada para la LDR
motor_persiana.attach (7); //indicamos a Arduino de controlar nuestro servo
por el pin 7
}
void loop() {

val=analogRead(ldr); //lee el valor de la LDR y lo almacena en la variable val
Serial.println(val); //muestra por el monitor serie el valor que tiene val
delay(100); //espera 100 milisegundos
if (val<=sensibilidad) { //si el valor de val es igual o menor que umbral,
hay poca luz
motor_persiana.write(180); //gira y recoge la cortina
delay(3000); //gira durante 3 segundos, este tiempo dependera de la longitud
de la persina
motor_persiana.write(90); //paramos el motor
}
else { //de lo contrario baja la persina
motor_persiana.write(0); //gira en sentido contrario al anterior
motor_persiana.write(90); //paramos el motor
}
}
}

```

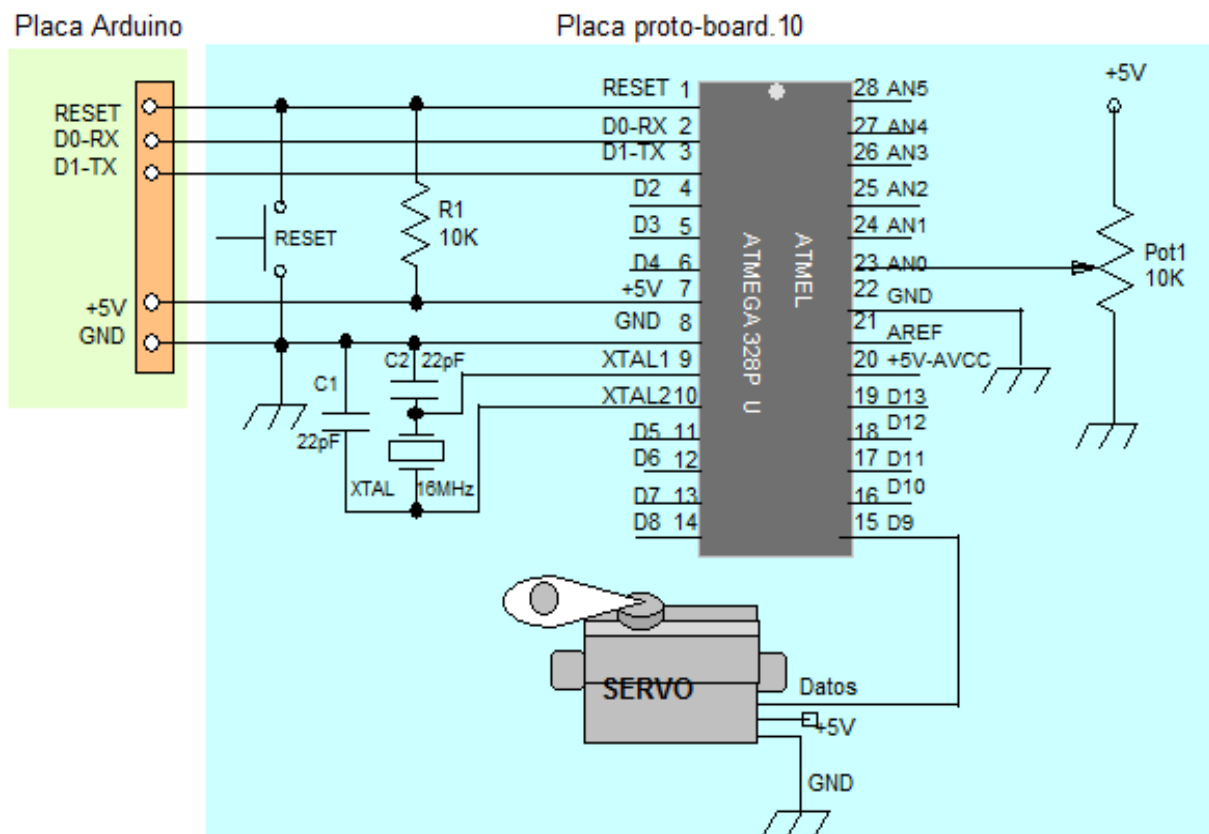
28. Control de Servomotores

Otro ejemplo sería el control de una posición de servo mediante un potenciómetro (resistencia variable). Para ello iniciamos el entorno de Arduino y vamos a los Sketch de ejemplos en **Archivo/Ejemplos/Servo/Knob**. Como se podrá apreciar, en el siguiente ejemplo, será necesario insertar en el sketch la librería *servo.h*.

```
/*Controlling a servo position using a potentiometer (variable resistor)
 by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>*/

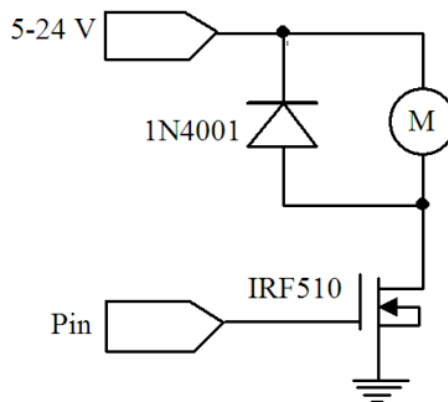
#include <Servo.h>
Servo myservo;    // create servo object to control a servo
int potpin = 0;   // analog pin used to connect the potentiometer
int val;         // variable to read the value from the analog pin

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}
void loop() {
  val = analogRead(potpin); // reads the value of the potentiometer
                             (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 179); // scale it to use it with the servo
                                   (value
                                   between 0 and 180)
  myservo.write(val); // sets the servo position according to
                      the
                      scaled value
  delay(15);          // waits for the servo to get there
}
```



29. Salida de alta corriente de consumo

A veces es necesario controlar cargas de más de los 40 mA que es el límite que suministra el microcontrolador Arduino. En este caso se hace uso de un transistor MOSFET que puede alimentar cargas de mayor consumo de corriente. En el siguiente ejemplo se muestra como un transistor MOSFET conmuta un motor 5 veces cada segundo.



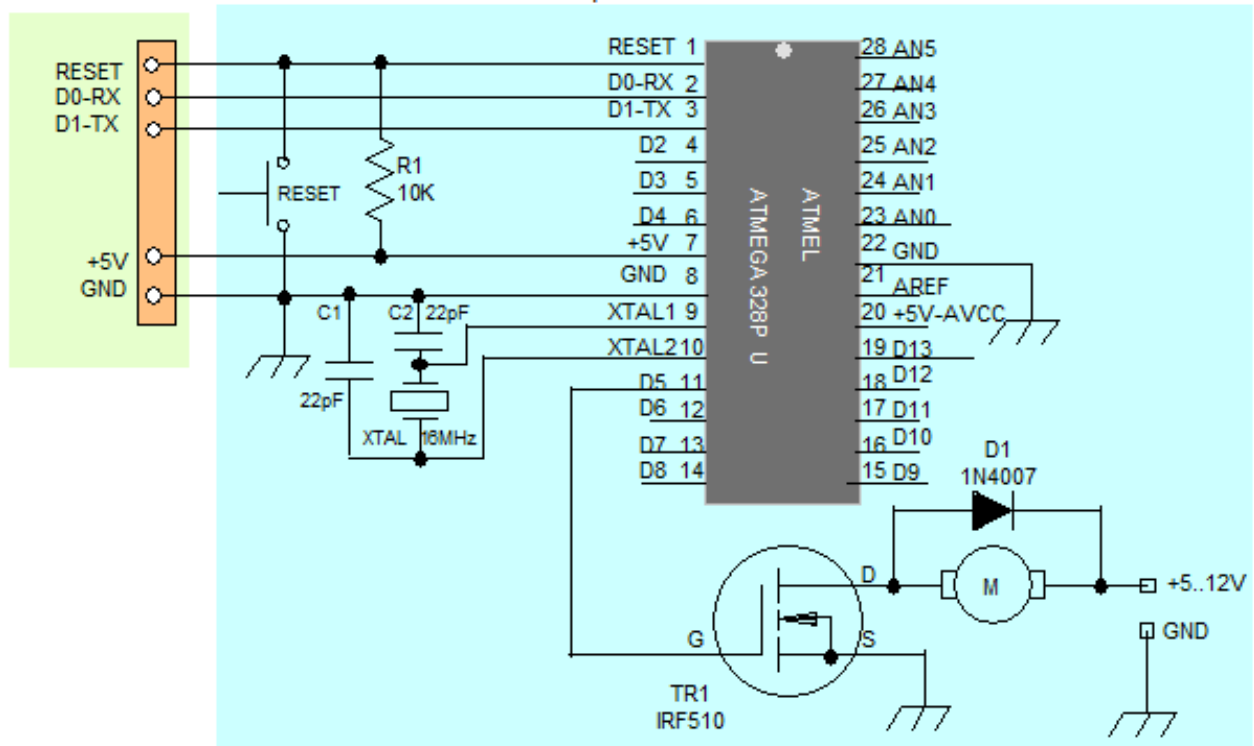
Pin introduce una oscilación que mueve el motor girandolo en un sentido y parandose

La señal de oscilación generada por la programación en Arduino por el pin D5 se introduce en el terminal G (gate) del transistor **MOSFET** Canal N, IRF510, cuya salida por el terminal D (drenador) hace mover el motor girándolo en un sentido y parándose, según la secuencia programada.

En el siguiente esquema se muestra un transistor TR1 MOSFET IRF510 que controla un motor con un diodo de protección D1 por ser una carga inductiva. En los casos que las cargas no sean inductivas no será necesario colocar el diodo.

Placa Arduino

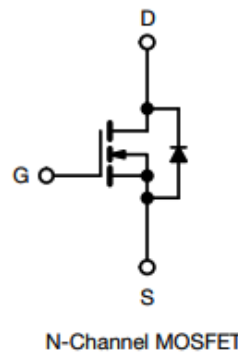
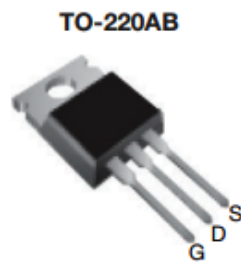
Placa proto-board.10



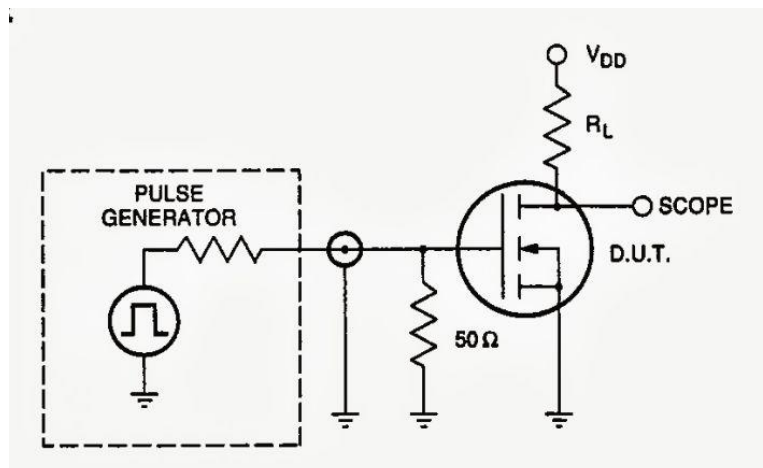
29. Salida de alta corriente de consumo

```
/* control de giro y parada de un motor*/
int outPin = 5; //asignamos el pin 5 de salida para entrada G del mosfet
void setup()
{
  pinMode (outPin, OUTPUT);    // pin5 como salida
}
void loop()
{
  for (int i=0; i<=5; i++)    // bucle de repeticion 5 veces
  {
    digitalWrite(outPin, HIGH); // active el MOSFET
    delay(250);                // espera 250 milisegundos
    digitalWrite (outPin, LOW); // desactiva el MOSFET
    delay(250);                // espera250 milisegundos
  }
  delay(1000);                // espera 1 segundo
}
```

Características del transistor MOSFET Canal N IRF510



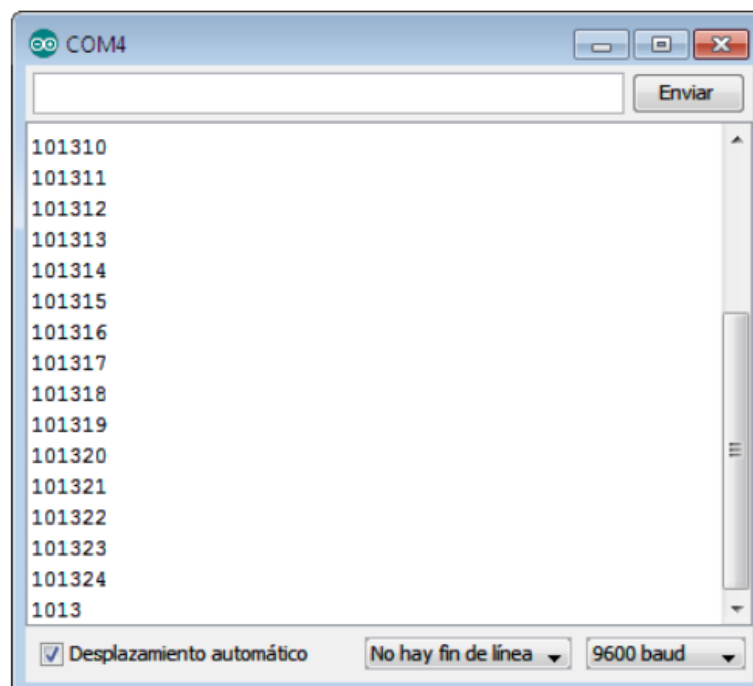
PRODUCT SUMMARY		IRF510	
V_{DS} (V)	100		
$R_{DS(on)}$ (Ω)	$V_{GS} = 10$ V	0.54	
Q_g max. (nC)	8.3		
Q_{gs} (nC)	2.3		
Q_{gd} (nC)	3.8		
Configuration	Single		



30. Programación de un contador

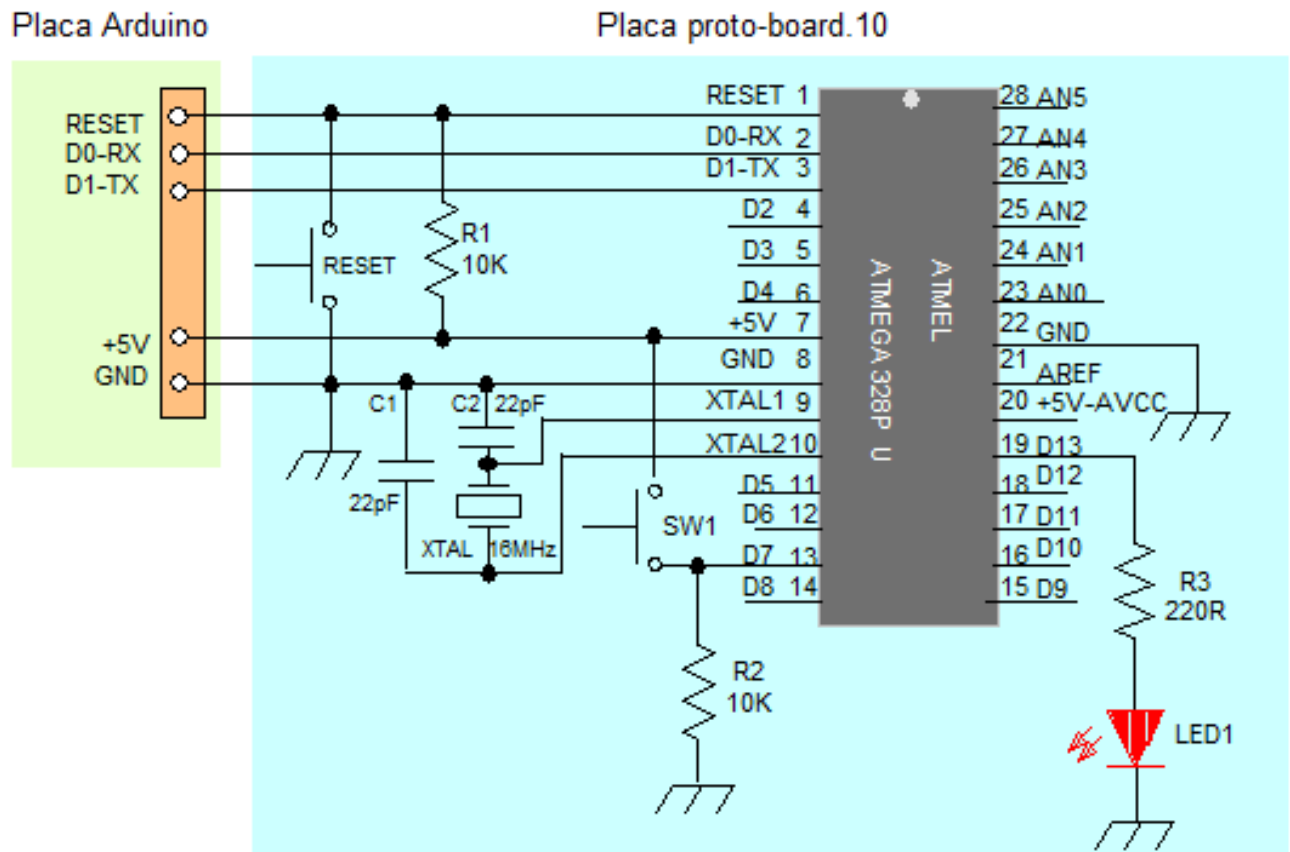
El programa que se va a exponer a continuación consiste en detectar que se ha pulsado el Botón, entrada del pin D7, y enciende el LED D13. Enviando al PC el valor de la variable de cuenta "Contador" vía puerto serie.

```
/* Programa Contador */
int LED=13;           //asignamos el pin 13 digital a LED
int boton=7;         //asignamos el pin 7 digital a boton
int valor=0;         //asignamos la variable valor a 0
int contador=0;      //asignamos la variable contador a 0
int estadoboton=0;   //asignamos la variable estadoboton a 0
void setup() {
  Serial.begin(9600); // configure velocidad de transmisión a 9600 b
  pinMode(LED, OUTPUT); // pone de salida digital el LED verde el pin 13
  pinMode(boton, INPUT); // pone de entrada digital el botón pin 7
}
void loop() {
  valor=digitalRead(boton); // lee el valor de la entrada digital pin 7
  digitalWrite(LED, valor); // enciende el LED si el nivel es alto
  if (valor!=estadoboton){ // condiciona igualdad estado variable valor
    if(valor==1){          // condiciona y compara con el valor 1
      contador++;          // suma 1 a contador
      Serial.println(contador); // lanza el valor de contador al pulsar boton
      Serial.print(10);    // lanza al monitor serial 10
      Serial.print(13);    // lanza al monitor serial 13
    }
  }
  estadoboton=valor;      //almacena valor en estado boton
}
```



30. Programación de un contador

Observándose en el Monitor Serial, cada vez que pulsemos el botón se va incrementando el contador de uno en uno, teniendo 1013 como índice del contador.



31. Cruce regulado por semáforos

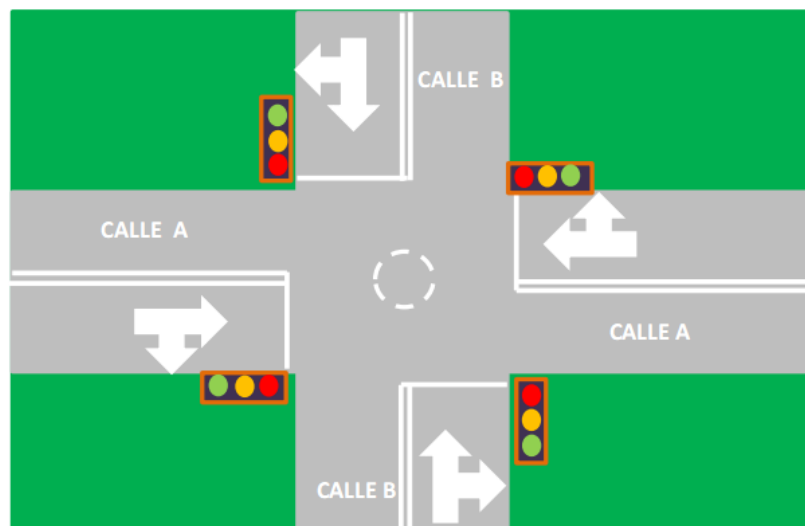
Todo proyecto conlleva unas series de pasos antes y durante su desarrollo. Según las dimensiones del proyecto se necesitarán más recursos o menos. Es decir, en el caso de este proyecto que se plantea el diseño de la regulación de un cruce con semáforos, los recursos son pocos, en el caso de una ampliación con más calles perpendiculares y el control del paso de peatones el diseño se complica aún más y los recursos aumentan considerablemente. Lo mismo ocurre, cuando diseñamos un proyecto de un sistema de alarma para una vivienda que controla únicamente dos zonas de protección que otro proyecto de seguridad donde se protege un museo donde tiene que haber, por ejemplo, muchísima más zonas de protección.

Pues bien, según el proyecto que se quiera realizar será más o menos complejo. En la programación será un tanto compleja contra más elementos exteriores necesitemos de controlar: un botón de paso de peatón, indicadores de salidas, ciclos de temporizaciones, etc.

En este proyecto de regulación de un cruce con semáforos, es realmente simbólico y sus recursos son mínimos, pero eso no quita que, hay que recopilar y plantear todos los pros y contras y tener la información suficiente para que no nos salga como un churro.

Primeramente y, es bien conocido por todos nosotros, que un cruce son dos calles o avenidas que se cruzan, con diferentes sentidos de circulación, y el objetivo es que puedan circular los vehículos de una de las calle y los vehículos de la otra estén detenidos, para posteriormente se produzca una conmutación en viceversa, es decir, la que estaba circulando se detienen y la que estaban parados comience a circular... hasta aquí, todo claro, OK? Omitimos en este proyecto el control del paso de personas (peatones), por existir pasos subterráneos.

Si observamos en la siguiente imagen, el cruce se compone de dos calles, la calle A y la calle B. La calle A la secuencia de regulación del semáforo es la misma en sus dos sentidos de circulación, por lo tanto habrá dos semáforos con la misma secuencia, lo mismo ocurre en la calle B pero siendo necesariamente la secuencia de regulación diferente entre cada una de las calles. Para ello, en la calle A colocamos los dos semáforos en paralelo con la misma secuencia y hacemos lo mismo en la calle B pero las secuencias de regulación de encendido y apagado son diferentes y conmutadas.



Cada semáforo contempla tres indicaciones de color rojo, verde y naranja.

31. Cruce regulado por semáforos

Si nos fijamos en los datos que se especifican a continuación, se trata de un cambio conmutado del semáforo de la calle A con el de la calle B, que sigue la siguiente secuencia:

1. Semáforo calle A se encuentra en verde, semáforo calle B se encuentra en rojo.
2. Semáforo calle A se enciende el naranja, semáforo calle B se encuentra en rojo
3. Semáforo calle A se encuentra en rojo, semáforo calle B se encuentra en verde (Del verde al rojo se encenderá antes el color naranja un breve tiempo y se pondrá en rojo, e inmediatamente pasara la calle B del rojo al verde).
4. Al pasar la calle B del verde al rojo se encenderá antes el color naranja un breve tiempo y se pondrá en rojo, e inmediatamente pasará la calle A del rojo al verde y vuelve a comenzar el ciclo.

```
/* Semaforo
   Secuencia de 6 led encendiendose por medio de una temporización delay
   diferentes y conmutadas */

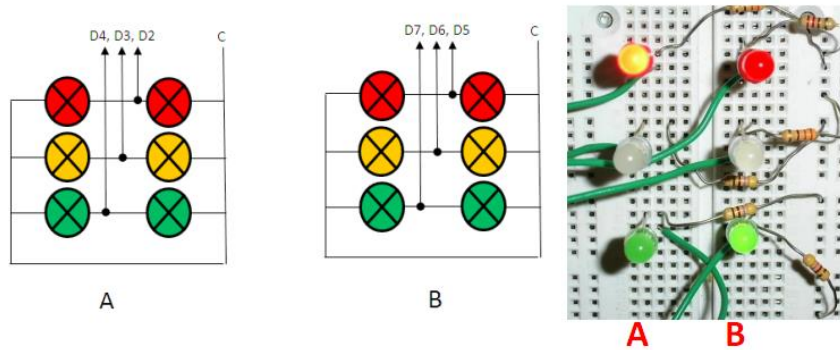
void setup() {          // configura el programa para pines de salida.
  pinMode(2, OUTPUT); // led rojo A
  pinMode(3, OUTPUT); // Led naranja A
  pinMode(4, OUTPUT); // Led verde A
  pinMode(5, OUTPUT); // Led rojo B
  pinMode(6, OUTPUT); // Led naranja B
  pinMode(7, OUTPUT); // Led verde B
}

void loop() {
  digitalWrite(2, HIGH); // Enciende el led rojo A
  digitalWrite(7, HIGH); // enciende el led verde B
  delay (25000);         // intervalo de encendido led rojo A y verde B
  digitalWrite(6, HIGH); // enciende el led naranja de la B
  delay (5000);          // tiempo encendido de 5 segundos
  digitalWrite(6,LOW);  // apaga el led naranja
  delay (100);           // tiempo apagado
  digitalWrite(7, LOW); // apaga el led verde B
  digitalWrite(2, LOW); // apaga el led rojo A
  delay(100);            // tiempo apagado led verde B y rojo A
  digitalWrite(5, HIGH); // Enciende el led rojo B
  digitalWrite(4, HIGH); // Enciende el led verde A
  delay (25000);         // tiempo encendido del led rojo B y led verde A
  digitalWrite(3, HIGH); // enciende el led naranja de la A
  delay (5000);          // tiempo encendido
  digitalWrite(3, LOW);  // apaga el led naranja de la A
  delay (100);           // tiempo apagado
  digitalWrite (5, LOW); // Apaga led rojo B
  digitalWrite (4, LOW); // Apaga led verde A
  delay (100);           // tiempo apagado led rojo B y led verde A
}
```

31. Cruce regulado por semáforos

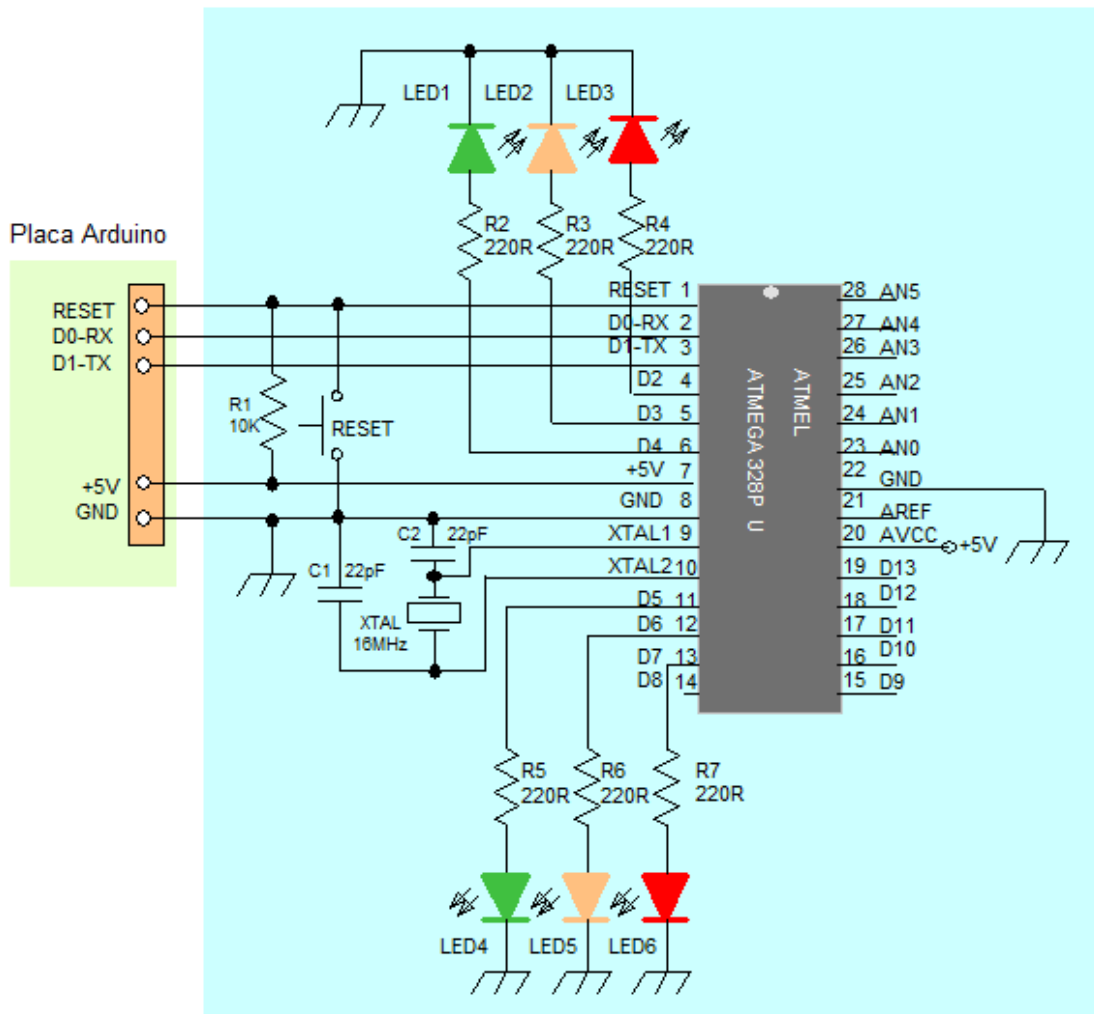
El circuito está constituido por el microcontrolador **Atmega328P-PU**. Las salidas de los pines de datos digitales 4, 5, 6, 11, 12 y 13, que corresponde a los datos D2, D3, D4, D5, D6 y D7. Estas salidas se insertan a las entradas de los OPTO MOC3021 para activar los TRIAC y estos encender o apagar las lámparas cuya tensión sea de unos 230 Vca.

En la siguiente página se muestra el plano eléctrico completo de este proyecto. La circuitería electrónica para el control de dos semáforos A y B, conmutados, para completarlo se necesitan la instalación de dos semáforos A puesto en paralelo y otros dos B también en paralelo.



En el microcontrolador **Atmega328P-PU** se conectan la señal de reloj, en los pines 9 y 10, un cristal de cuarzo de **16 MHz polarizado** por dos condensadores de 22 pF. La fuente de alimentación puede ser conmutada o lineal con salida estabilizada de +5 voltios (7805).

Placa proto-board.10



32. Semáforo para paso de peatones

Este proyecto contempla el control del paso de peatones de dos semáforos para cada una de las direcciones de una vía de doble sentido y se ejecuta mediante un pulsador peatonal.

Estos semáforos tienen la misma secuencia de regulación y está exclusivamente para el paso de peatones, que permitirá el paso, cuando se le pulsa el botón para pasar, con lo cual los vehículos se detendrán y, pasados un determinado tiempo, volverán a su estado normal, volviendo a circular los vehículos.

La secuencia de programación comprende tres ciclos de funcionamiento:

- **1º Ciclo.** Mientras no se pulse el botón de paso de peatón, el muñeco o Led rojo de peatón estará encendido y los 2 Leds naranja del semáforo estarán encendiéndose secuencialmente.
- **2º Ciclo.** Cuando se pulsa el botón para pasar (pulsador de peatón) se establece un breve tiempo antes de activar la secuencia de conmutación para cambiar los 2 Leds naranjas del semáforo, al Led rojo y, después de unos segundos, se apaga el muñeco o Led rojo de peatón y se enciende el Led verde de paso.
- **3º Ciclo.** Se establece un tiempo prudencial para que pase el peatón, el cual una vez finalizado se procede a ponerse primeramente activo el Led rojo de peatón y apagándose el Led verde, posteriormente y con un breve tiempo se apagará el Led rojo del semáforo y se activarán los dos Leds naranja secuencialmente, volviéndose al ciclo 1º.

NOTA: Se duplicarán la señalización para el paso peatonal (2 juegos de muñecos verde y rojo conectados en paralelo) y en el caso de una vía en doble sentido se duplicarán el semáforo, conectándose en paralelo.

```
/* Este programa consiste en controlar un semaforo para el paso peatonal a
través de un pulsador */

int led1=2;           // Semaforo naranja
int led2=3;           // Semaforo naranja
int led3=4;           // Semaforo en rojo
int led4=5;           // Paso peaton en verde
int led5=6;           // Paso peaton en rojo
int pulsador=7;       // Pulsador de paso peatonal
int estadopulsador=0; // Declaramos la variable a cero

void setup(){
  pinMode(led1, OUTPUT); // Configuramos el led 1 de salida
  pinMode(led2, OUTPUT); // Configuramos el led 2 de salida
  pinMode (led3,OUTPUT); // Configuramos el led 3 de salida
  pinMode (led4, OUTPUT); // Configuramos el led 4 de salida
  pinMode (led5, OUTPUT); // Configuramos el led 5 de salida
  pinMode (pulsador, INPUT); //Configuramos el pulsador de entrada
}
void loop() {
```

32. Semáforo para paso de peatones

```
estadopulsador=digitalRead(pulsador); //guardamos valor de pulsador
if (estadopulsador == HIGH){ // condicionante si esta nivel alto

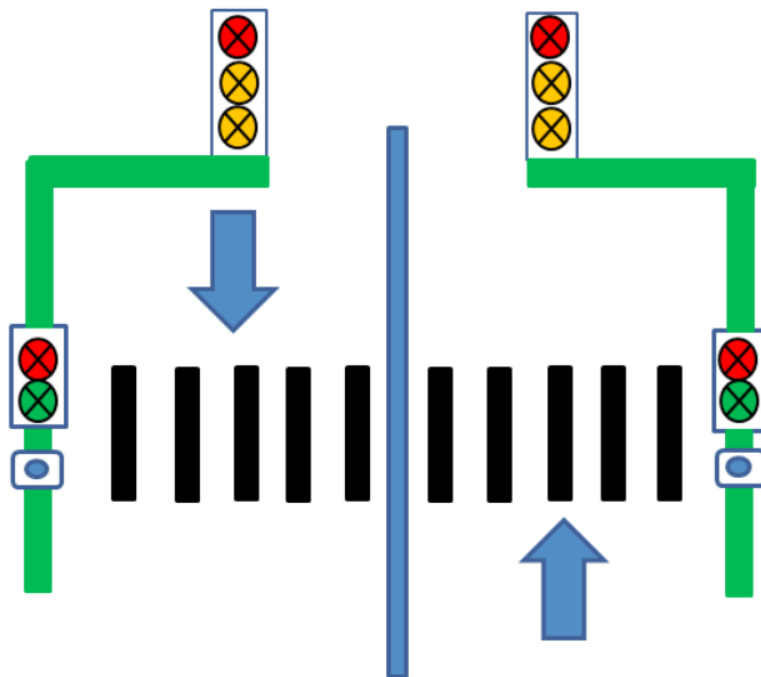
    digitalWrite(led1,LOW); // apagamos el led1 naranja
    delay(100); // dejamos 100 ms de tiempo
    digitalWrite(led2,HIGH); // encendemos el led2 naranja semaforo
    delay(3000); // retardo de 3 segundos
    digitalWrite(led2,LOW); // apagamos el led2 naranja semaforo
    digitalWrite(led3,HIGH); // encendemos el led3 rojo semaforo
    delay(5000); // ponemos un retardo de 5 segundo
    digitalWrite(led5, LOW); // apagamos el led 5 rojo del peaton
    delay(100); // retardo de 100 milisegundos
    digitalWrite(led4,HIGH); // encendemos el led 4 verde del peaton
    delay(10000); // tiempo de paso de 10 segundos
    for (int x=0; x<20; x++) { // ponemos un temporizador
        digitalWrite(led4,LOW); // enciende led 4 verde peaton
        delay(300); // retardo
        digitalWrite(led4,HIGH); // apaga led 4 verde peaton
        delay(300); // retardo
    }
}
else {
    digitalWrite(led4, LOW); // apaga el led4 verde peaton
    digitalWrite(led5,HIGH); // enciende el led5 rojo peaton
    delay(200); // establece un tiempo
    digitalWrite(led3, LOW); // apaga el led3 rojo vehiculo
    delay(200);
    digitalWrite(led1, HIGH); //apaga el led1 naranja
    delay(100);
    digitalWrite(led1, LOW); //enciende led 2 naranja
    delay(100);
    digitalWrite(led2, HIGH); // enciende led 2 naranja
    delay(100);
    digitalWrite(led2, LOW);
}
}
```

32. Semáforo para paso de peatones

El circuito electrónico de este proyecto está constituido principalmente por el microcontrolador Atmega328P-PU, en este caso utilizamos 6 pines de datos D02, D03, D04, D05, D06 y D07 para establecer las salidas digitales para los diferentes puntos de control: Leds naranjas, led rojo, muñeco led verde, muñeco led rojo y una entrada de pulsador. En el pin 13 lo utilizamos para conectar un pulsador, activado a nivel alto de 5V y polarizado por una resistencia de 10K a masa para evitar frustraciones de nivel de tensión que pudieran dar errores.

El Atmega328P de 28DIP se le conecta para la señal de reloj, en los pines 9 y 10, un cristal de cuarzo de 16 MHz para sincronizar la señal de reloj y el funcionamiento de todos los elementos conectados al microcontrolador. Los pines de salida de D02 a D06 se aplican una resistencia de 470 Ohmios en serie para polarizar el Led del Optoacoplador.

Al ser de dos direcciones de doble sentido de circulación tenemos que conectar en paralelo los dos semáforos y los señalizadores de peatones. Lo mismo ocurre con el pulsador de paso que también irá en paralelo.

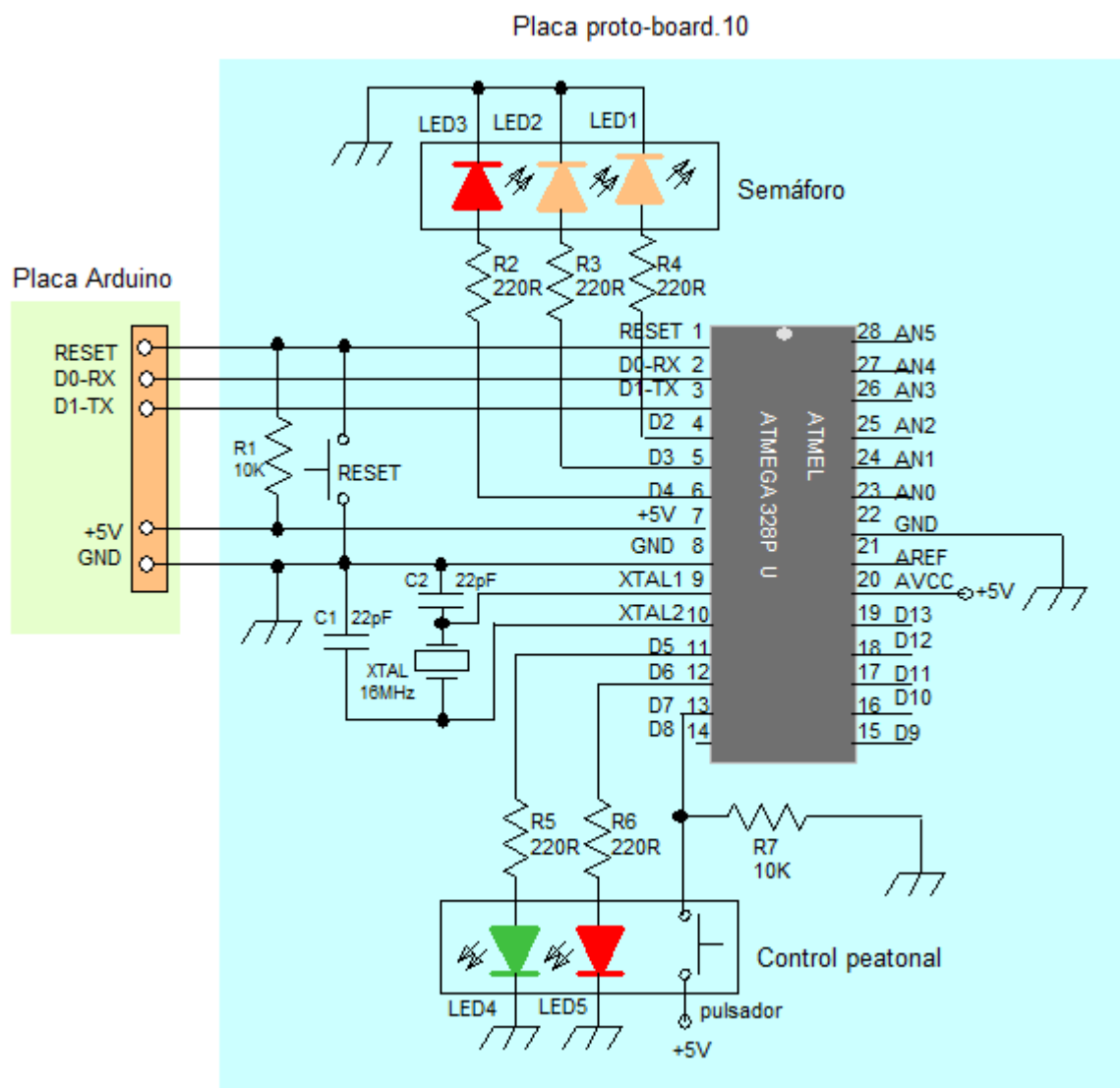


En las siguientes páginas se muestran el esquema eléctrico de este proyecto. La circuitería electrónica para el control de un semáforo y el control del paso de peatones, para que esté completo se necesita de la instalación de dos semáforos, en paralelo, para cada sentido de circulación y la instalación de dos controles de paso de peatón, en paralelo, para colocarlo en cada extremo de la calle.

32. Semáforo para paso de peatones

En el siguiente esquema eléctrico se muestra la configuración básica de un semáforo con tres leds, dos de color naranja y uno de color rojo y, por otra parte en el control peatonal, dos leds, uno de color verde y otro de color rojo y un pulsador (Normalmente Abierto). Al tener doble sentido de circulación se tendrá que disponer de dos semáforos y dos controles peatonal. Esto se realiza poniendo los semáforos en paralelo lo mismo que los controles peatonal.

Sin haber pulsado el pulsador para el paso de peatones, el led5 rojo se encuentra encendido y los leds 1 y 2 color naranja comienzan a parpadear alternativamente. Cuando pulsamos el pulsador para pasar, se apaga el led1 y el led2 se queda un breve tiempo encendido para dar paso al led3 rojo, que se queda encendido durante un determinado tiempo, el led 5 rojo se apaga y se enciende el led4 verde que indica que podemos pasar, al cabo de un tiempo se pone a parpadear indicando que va a cambiar, apagándose y encendiéndose el led5 y los leds 1 y 2 parpadean.



33. Dado electrónico

Este proyecto está dedicado al entretenimiento y se construye un dado electrónico cuya programación consiste en el control aleatorio de seis números del 1 al 6 que se visualiza mediante un decodificador de 7 segmentos en un display.

En esta programación se van a utilizar dos funciones esenciales para producir valores aleatorios, estas son: **random(seed)** y **random(max, min)**:

Función randomSeed(seed)

Esta función nos permite inicializar, a partir de una variable o de otra función, una semilla para generar números aleatorios y como punto de partida para la función **random()**.

```
randomSeed(valor); // hace que valor sea la semilla del random
```

Por ejemplo:

randomSeed(millis) generará números aleatorios a partir del valor de la función `millis`.

Recordemos que esta función devuelve en **milisegundos** el tiempo desde que Arduino está ejecutando el programa actual.

Debido a que Arduino es incapaz de crear un verdadero número aleatorio, `randomSeed` le permite colocar una variable, constante u otra función de control dentro de la función `random`, lo que permite generar números aleatorios "al azar". Hay una variedad de semillas o funciones, que pueden ser utilizados en esta función, incluido `millis()` o incluso `analogRead()` que permite leer ruido a través de un pin analógico.

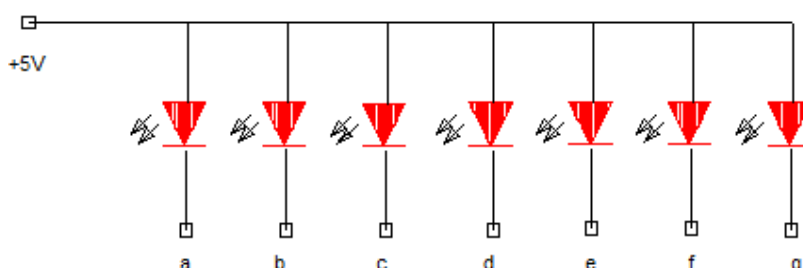
Función random(max) y random(min,max)

Para utilizar la función `random`, primero debemos emplear la función **randomSeed()**.

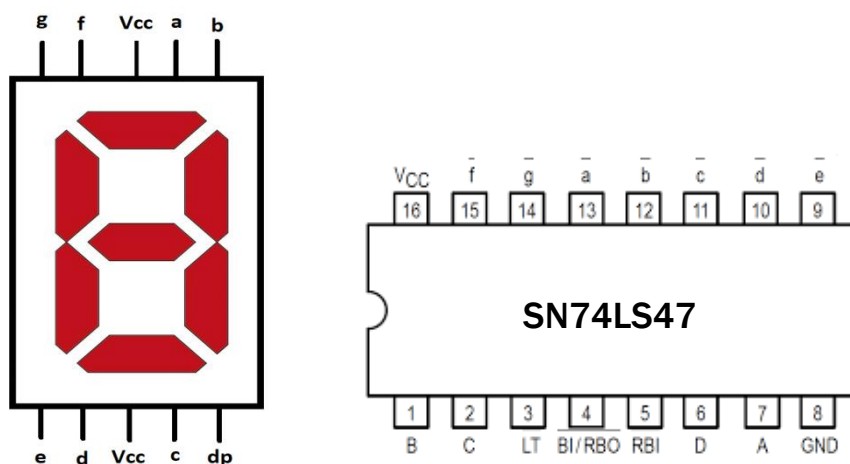
La función **random(aleatorio)** genera números aleatorios en un rango de 0 a un máximo, o un rango preestablecido por el usuario con las variables `max` y `min`:

- **random(max)** devuelve un valor aleatorio entre 0 y `max`.
- **random(min, max)** devuelve un valor aleatorio entre `min` y `max`.

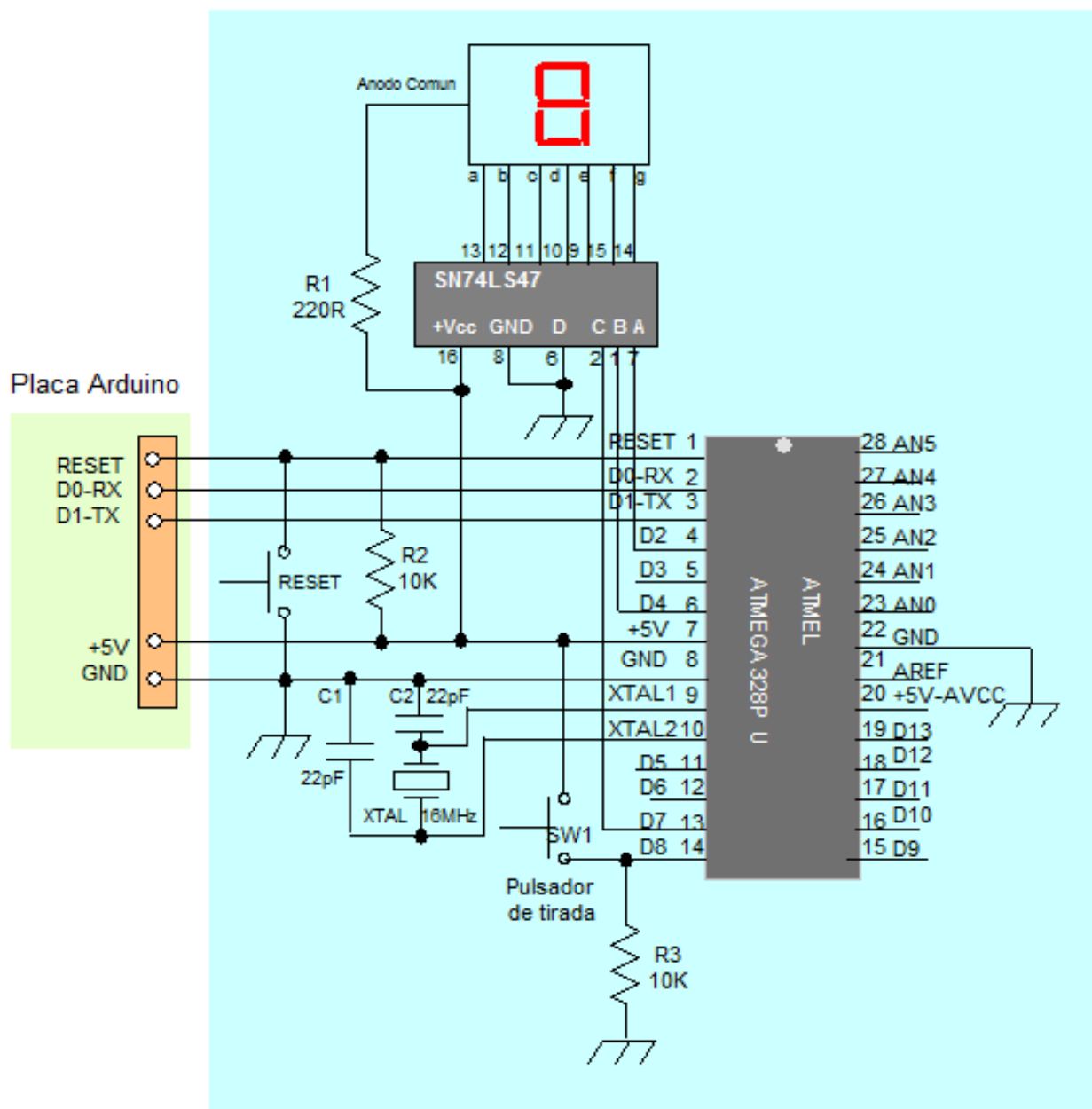
En el esquema eléctrico se ha añadido un decodificador SN74LS47 que es un BCD a 7 segmentos. De las cuatro entradas una de ellas, la entrada pin 6 (D), no se usa y se pondrá a masa, las otras tres A, B y C se usan y van conectados a los pines digitales de salida D2, D4 y D7 del microcontrolador ATmega328P. Las 7 salidas del decodificador se activan a nivel bajo y se conectarán a los cátodos del display y se pondrá una resistencia de 220Ω que va conectado al ánodo común +5V del display.



33. Dado electrónico



Placa proto-board.10



33. Dado electrónico

En el pin digital de entrada D8 se pondrá un pulsador-botón para la tirada del dado.

```
/*Programacion para un dado electronico*/
int A=2;           //asignamos A al pin digital D2
int B=4;           //asignamos B al pin digital D4
int C=7;           //asignamos C al pin digital D7
int boton=8;       //asignamos boton al pin digital D8
int estadopulsador; //creamos variable
int maximo=6;      //asignamos la variable maximo a 6
int randomNumber;  //creamos variable

void setup() {
  pinMode (A, OUTPUT); //configuramos A como salida
  pinMode (B, OUTPUT); //configuramos B como salida
  pinMode (C, OUTPUT); //configuramos C como salida
  pinMode (boton, INPUT); //configuramos boton como entrada
}

void loop() {
  estadopulsador=digitalRead(boton); //lee el estado del pulsador
  if (estadopulsador==HIGH){         // si estadopulsador esta en alto
  for (int w=0; w<5; w++) {          // bucle repetitivo hasta 5
  dado1();                            //ve a la función dado1
  delay(100);                          //espera 100 milisegundos
  dado2();                            //ve a la función dado2
  delay(100);                          //espera 100 milisegundos
  dado3();                            //ve a la función dado3
  delay(100);                          //espera 100 milisegundos
  dado4();                            //ve a la función dado4
  delay(100);                          //espera 100 milisegundos
  dado5();                            //ve a la función dado5
  delay(100);                          //espera 100 milisegundos
  dado6();                            //ve a la función dado6
  }
  randomSeed(millis());                //se genera numeros aleatorios
  randomNumber=random(maximo);        //guarda el valor obtenido de random en
  randomNumber
  if (randomNumber==1){                //si es igual a 1
  dado1();                             //ve a la función dado1
  }
  if (randomNumber==2){                //si es igual a 2
  dado2();                             //ve a la función dado2
  }
  if (randomNumber==3){                //si es igual a 3
  dado3();                             //ve a la función dado3
  }
  if (randomNumber==4){                //si es igual a 4
  dado4();                             //ve a la función dado4
  }
  if (randomNumber==5){                //si es igual a 5
  dado5();                             //ve a la función dado5
  }
  if (randomNumber==6){                //si es igual a 6
  dado6();                             //ve a la función dado6
  }
}
```

33. Dado electrónico

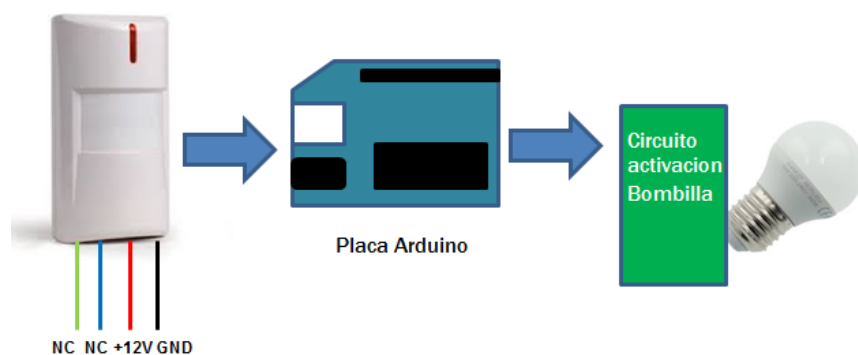
```
}  
}  
void dado1(){ // funcion dado1  
digitalWrite (A, HIGH); // pon a nivel alto el pin A  
digitalWrite (B, LOW); // pon a nivel bajo el pin B  
digitalWrite (C, LOW); // pon a nivel bajo el pin C  
}  
void dado2(){ // funcion dado2  
digitalWrite (A, LOW); // pon a nivel bajo el pin A  
digitalWrite (B, HIGH); // pon a nivel alto el pin B  
digitalWrite (C, LOW); // pon a nivel bajo el pin C  
}  
void dado3(){ // funcion dado3  
digitalWrite (A, HIGH); // pon a nivel alto el pin A  
digitalWrite (B, HIGH); // pon a nivel alto el pin B  
digitalWrite (C, LOW); // pon a nivel bajo el pin C  
}  
void dado4(){ // funcion dado4  
digitalWrite (A, LOW); // pon a nivel bajo el pin A  
digitalWrite (B, LOW); // pon a nivel bajo el pin B  
digitalWrite (C, HIGH); // pon a nivel alto el pin C  
}  
void dado5(){ // funcion dado5  
digitalWrite (A, HIGH); // pon a nivel alto el pin A  
digitalWrite (B, LOW); // pon a nivel bajo el pin B  
digitalWrite (C, HIGH); // pon a nivel alto el pin C  
}  
void dado6(){ // funcion dado6  
digitalWrite (A, LOW); // pon a nivel bajo el pin A  
digitalWrite (B, HIGH); // pon a nivel alto el pin B  
digitalWrite (C, HIGH); // pon a nivel alto el pin C  
}
```

34. Detector de movimientos con activación de luz

Este modelo de proyecto se suele utilizar en los lugares públicos: servicios, entradas a edificios, pasillos, etc. Consiste en detectar la persona y encenderse la luz del lugar y pasado un determinado tiempo y no habiéndose detectado más movimiento se apaga.

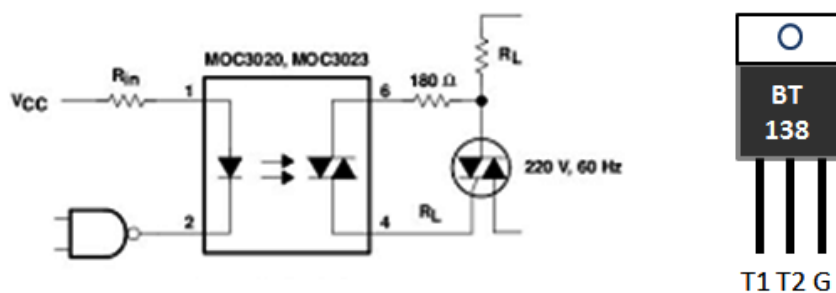
El elemento principal consiste en un volumétrico infrarrojo que al detectar movimiento hace encender una bombilla de 230Vca y se queda encendida durante un minuto y medio y, se apaga posteriormente si no existe movimientos. Este tiempo puede cambiarse en la programación del microcontrolador.

El detector volumétrico de infrarrojo, cuando está conectado a la alimentación de 12 voltios, sus terminales de salida se encuentran normalmente cerrados, pero cuando se produce una detección su terminal de salida cambia y pasa a estar abierto con lo que se produce el encendido de la bombilla.

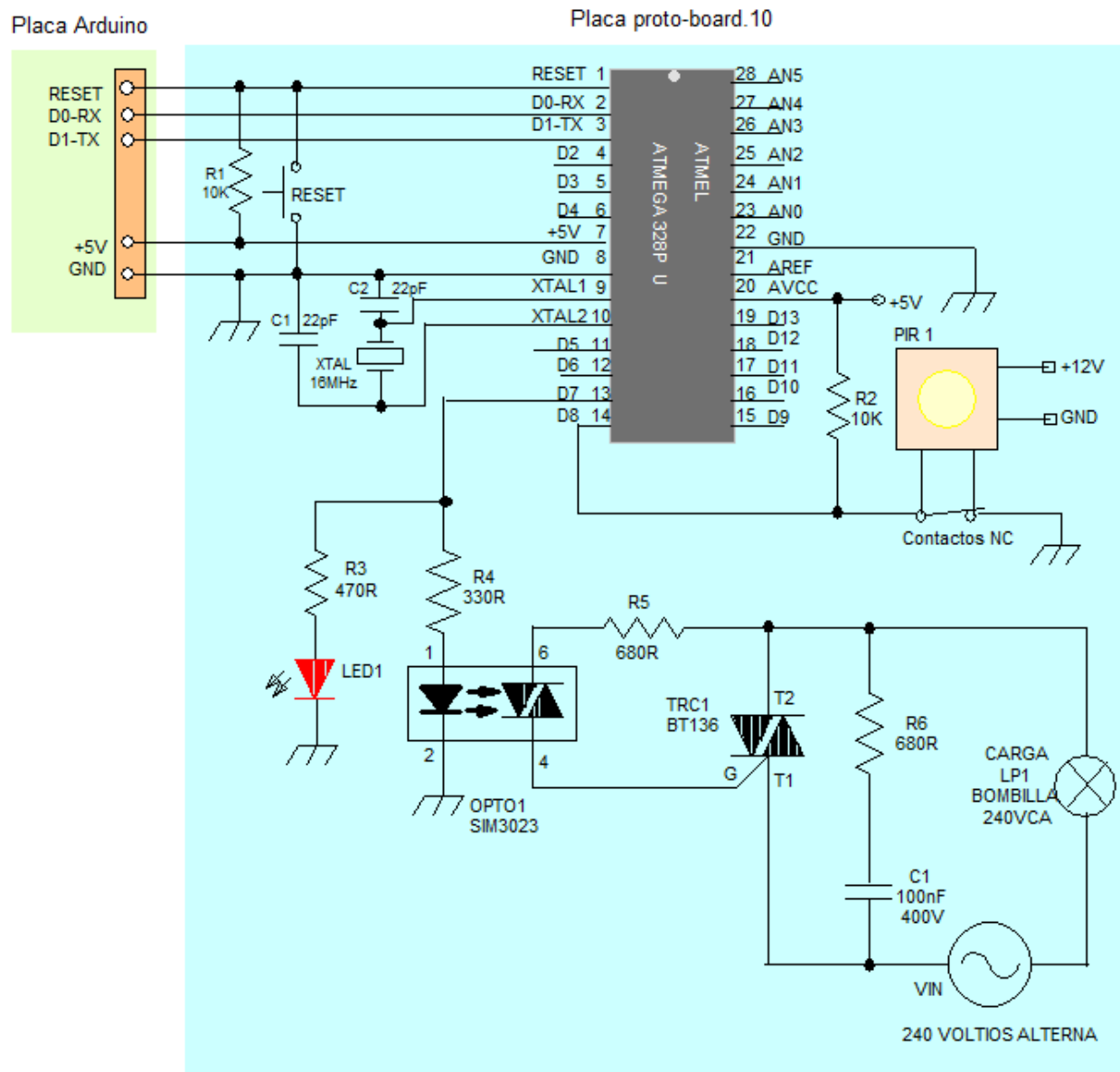


Para este caso se ha seleccionado un puerto de entrada digital D8 y le hemos llamado "detec" que se activará con un nivel alto (5 voltios) cuando exista un movimiento, de lo contrario estará a un nivel bajo (0 voltios) cuando no detecte movimiento.

Para señalar esto, cuando se produce movimiento, se utiliza el pin digital de salida D7 que activa un diodo led rojo y la conexión a un optoacoplador MOC3023 y un Triac BT138 para encender una bombilla de 230 voltios que estará aproximadamente encendida un minuto y 38 segundos, según el valor que le demos al bucle repetitivo.



34. Detector de movimientos con activación de luz



```

/*Código para detectar movimiento y encender una luz durante un determinado
tiempo*/
int pin=7;           //asignamos el pin con el pin digital 7
int detec=8;        //asignamos detec al pin digital 8
int movi;           //asignamos la variable movi
void setup() {
  pinMode (pin, OUTPUT); //configuramos el pin de salida
  pinMode (detec, INPUT); // configuramos detect de entrada
}
void loop() {
  movi=digitalRead(detec); //lee el valor de detec y guardalo en movi
  if (movi==HIGH) {       //si movi esta a nivel alto
    for (int w=0; w<500; w++){ /* creamos un bucle repetitivo que dura 1 minuto
    y 38 segundos para mayor tiempo cambiamos el valor a mas de 500 */
      digitalWrite(pin, HIGH); //ponemos el pin a nivel alto y enciendo led
      delay(200); //esperamos 200 milisegundos que se añade a variable w
    } }
    digitalWrite(pin, LOW); //termina el bucle y apagamos led }
  }

```

35. Interruptor por control remoto

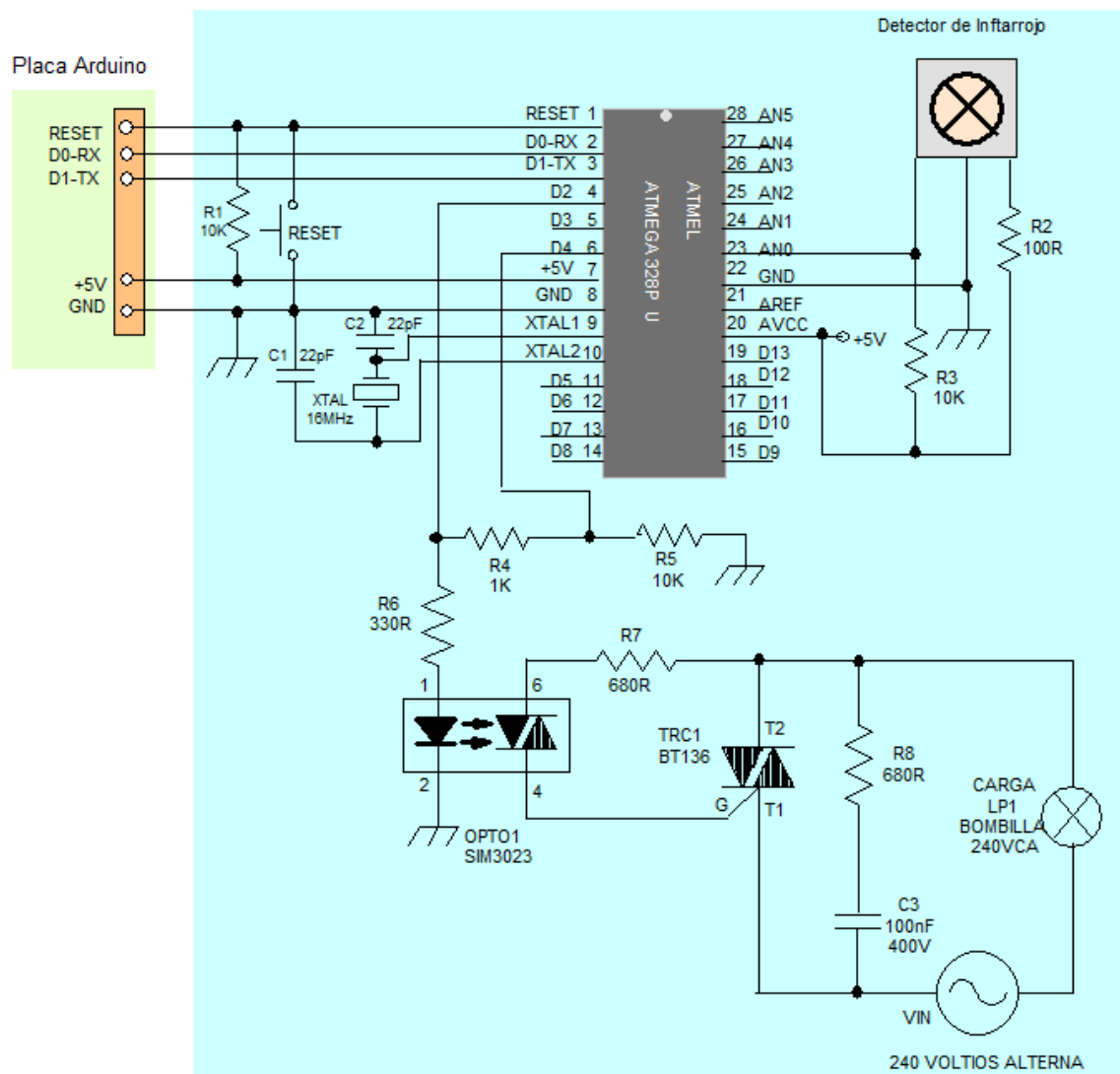
Este programa consiste en el control de la entrada de señales de un diodo receptor de infrarrojo a través de cualquier mando a distancia para encender o apagar una bombilla de corriente alterna de 230 voltios.

El espectro electromagnético consiste en una franja de ondas electromagnéticas cuya frecuencia es muy baja para que nuestros ojos la detectaran. Esta franja son los **infrarrojos**. Pues bien, existen diodos capaces de emitir luz infrarroja y transistores sensibles a este tipo de ondas y que por lo tanto detectan las emisiones de los diodos. Esta es la base del funcionamiento de los mandos a distancia de nuestros televisores y detectores de posicionamiento de objetos.



Un diodo emisor de infrarrojo envía una señal y es captada por un transistor de infrarrojo que actúa sobre un circuito que hace activar un motor, una bombilla, un canal de TV, etc.

Placa proto-board.10



35. Interruptor por control remoto

```
/* encendido y apagado por mando a distancia de infrarrojo*/
int valor; //asignamos la variable
int valor2; //asignamos la variable 2
int rea=4; // asignamos el puerto digital 4 a rea
int sensor=A0; //asignamos el puerto analógico a sensor
int led=2; //asignamos el puerto digital 2 a led

void setup() {
  pinMode (sensor, INPUT); // configuramos sensor de entrada
  pinMode (led, OUTPUT); //configuramos led de salida
  pinMode (rea, INPUT); //configuramos el pin rea como entrada
}
void loop(){
  valor=analogRead(sensor); // almacena el valor del sensor analogico en valor
  valor2=digitalRead(rea); // almacena el valor de rea digital en valor2
  if (valor<100 && valor2==LOW){ //condicionante si valor menor de 100 y
  valor2 bajo
  digitalWrite(led, HIGH); //encendemos led
  delay(500); //espera medio segundo
  }
  if (valor<100 && valor2==HIGH){ //condicionante si valor menor de 100 y
  valor2 alto
  digitalWrite (led, LOW); // apagamos led
  delay(500); //espera medio segundo
  }
}
```