

Mis Proyectos con Arduino

Duemilanove Atmega 328P-PU.

2ª Versión. Actualizada.

20/08/2017

José Miguel Castillo Castillo



MIS PROYECTOS CON ARDUINO

INDICE DE CONTENIDO. MIS PROYECTOS CON ARDUINO

1. INTRODUCCIÓN

2. CONOCIENDO ARDUINO

- 2.1 Placa Hardware PCB
- 2.2 Software de Desarrollo
- 2.3 Elementos de la Placa Arduino

3. CARACTERÍSTICAS DE LA PLACA ARDUINO DUEMILANOVE ATMEGA328P

- 3.1. Análisis de la placa Arduino Duemilanove
- 3.2. Alimentación
- 3.3. Memoria
- 3.4. Entrada y Salida
- 3.5. Comunicación
- 3.6. Programación
- 3.7. Reseteo por Hardware y Software
- 3.8. Protección de sobrecarga de USB

4. GRABACIÓN DEL BOOTLOADER

5. TRABAJAR DIRECTAMENTE CON LA PLACA DE PROTOTIPO *PROTOBOARD*

6. PLACA DE DESARROLLO DIY ATMEGA 328P

7. COMPONENTES HARDWARE PARA CONEXIÓN CON ARDUINO

8. ENTORNO DE DESARROLLO DE ARDUINO

- 8.1. Menú Archivo
- 8.2. Menú Editar
- 8.3. Menú Sketch
- 8.4. Menú Herramientas

9. LENGUAJE DE PROGRAMACIÓN ARDUINO

- 9.1. Estructura básica de un programa
- 9.2. Funciones
- 9.3. Variables
- 9.4. Constantes
- 9.5. Tipos de Datos
- 9.6. Funciones de E/S Digitales
- 9.7. Funciones de Entradas Analógicas
- 9.8. Función map()
- 9.9. Función de Interrupción
- 9.10. Funciones de Tiempo y Matemáticas
- 9.11. Sentencias condicionales (Control de Flujo)
- 9.12. Crear nuestras propias funciones
- 9.13. Operadores Aritméticos
- 9.14. Operadores Aleatorios
- 9.15. Comunicación Serial
- 9.16. Ejemplos de programación básica de Arduino

MIS PROYECTOS CON ARDUINO

10. COMENZAR A TRABAJAR CON ARDUINO

- 10.1. Ejemplo de programación: Blink
- 10.2. Prácticas con Arduino

11. PROYECTO: CRUCE REGULADO POR SEMÁFOROS

- 11.1. Datos para la programación
- 11.2. Instrucciones y códigos de programación
- 11.3. Descripción y funcionamiento del circuito. Plano eléctrico

12. PROYECTO: SEMÁFORO PARA PASO DE PEATONES

- 12.1. Datos para la programación
- 12.2. Instrucciones y códigos de programación
- 12.3. Descripción y funcionamiento del circuito. Plano eléctrico.

13. PROYECTO: CONTROL DE LA LUZ NOCTURNA DE ENTRADA A UN EDIFICIO

- 13.1. Datos para la programación
- 13.2. Instrucciones y códigos de programación
- 13.3. Descripción y funcionamiento del circuito. Plano eléctrico.

14. PROYECTO: CENTRALITA DE ALARMA 2 ZONAS NC

- 14.1. Datos para la programación
- 14.2. Instrucciones y códigos de programación
- 14.3. Descripción y funcionamiento del circuito. Plano eléctrico.

15. PROYECTO: SIMULADOR DE PRESENCIA ANTIRROBO

- 15.1. Datos para la programación
- 15.2. Instrucciones y códigos de programación
- 15.3. Descripción y funcionamiento del circuito. Plano eléctrico.

MIS PROYECTOS CON ARDUINO

1. INTRODUCCIÓN

Para realizar un diseño electrónico donde deseamos utilizar el menor número de componentes y, que a su vez pueda ser modificable su funcionamiento por software en futuras versiones, nos debemos de plantear la utilización de un microcontrolador.

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.

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.

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.

<http://www.alldatasheet.es/datasheet-pdf/pdf/241077/ATMEL/ATMEGA328P.html>

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 unos subsistemas o bloques:

- **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.
- **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.
- **Unidad de memoria**. Esta memoria almacenará los programas que se van a ejecutar y los resultados derivados de dicha ejecución.
- **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.
- **Periféricos**. Pueden ser otros dispositivos microprogramables o simplemente circuitos digitales que permiten al usuario interactuar con el sistema.

MIS PROYECTOS CON ARDUINO

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. Tener claro de qué información y datos necesitamos y cual disponemos, dónde la podemos conseguir, etc. Al principio, estos puntos nos servirán de base para tener una 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. 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 mediante DATABOOK, por ejemplo, en <http://www.alldatasheet.es>, etc.
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 Led, deberemos utilizar las digitales, a no ser que se desee jugar con la intensidad luminosa de dicho diodo Led.

En definitiva en un proyecto con microcontrolador intervienen dos partes fundamentales:

1. **El Hardware.** 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 nuestro proyecto está bien conseguido.

MIS PROYECTOS CON ARDUINO

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

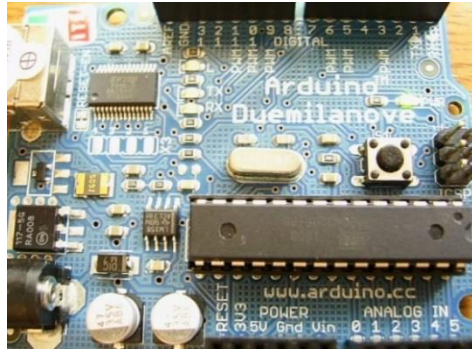
Cuando hablamos de *placa hardware* nos estamos refiriendo en concreto a una PCB (del inglés *Printed Circuit Board*, o sea, placa de circuito impreso). Las PCBs son superficies fabricadas de un material no conductor (normalmente resinas de fibra de vidrio reforzada, cerámica o plástico) sobre las cuales 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.
<http://myelectronic.mipropia.com/arduino.html>

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.



MIS PROYECTOS CON ARDUINO

Por ejemplo, en nuestro caso estudiaremos la placa Arduino **Duemilanove Atmega-328P-PU** que veremos en el siguiente apartado 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** (<http://www.atmel.com>).



reset	1	28	analog 5
pin 0 rx	2	27	analog 4
pin 1 tx	3	26	analog 3
pin 2	4	25	analog 2
pin 3 pwm	5	24	analog 1
pin 4	6	23	analog 0
+5 volts	7	22	ground
ground	8	21	not connected
crystal	9	20	+5 volts
crystal	10	19	pin 13
pin 5 pwm	11	18	pin 12
pin 6 pwm	12	17	pin 11 pwm
pin 7	13	16	pin 10 pwm
pin 8	14	15	pin 9 pwm

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 (<http://www.wiring.co>).

Esta placa surgió en 2003 como proyecto personal de Hernando Barragán, estudiante por aquel entonces del Instituto de Diseño de **Ivrea** (lugar donde surgió en 2005 precisamente la placa 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.

MIS PROYECTOS CON ARDUINO

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 programación 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 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.

Tanto el entorno de desarrollo como el lenguaje de programación Arduino están inspirado en otro entorno y lenguaje libre preexistente: Processing (<http://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++.

```
#include <stdio.h>
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
```

C/C++

MIS PROYECTOS CON ARDUINO

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.
- **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 reinicio:** 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.

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.

<http://www.ardumania.es/aprende/>

MIS PROYECTOS CON ARDUINO

3. CARACTERÍSTICAS DE LA PLACA ARDUINO DUEMILANOVE ATMEGA-328P

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.

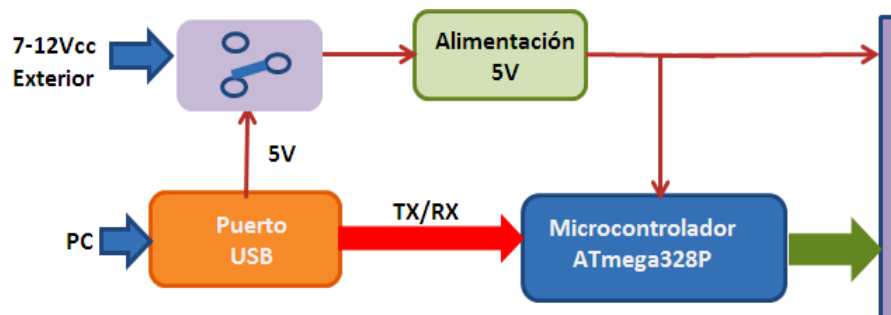
Aunque existen diferentes versiones de la placa de Arduino, en nuestro caso, nos centraremos en la placa Arduino **Duemilanove** con el microcontrolador **Atmega328P-PU**.



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:

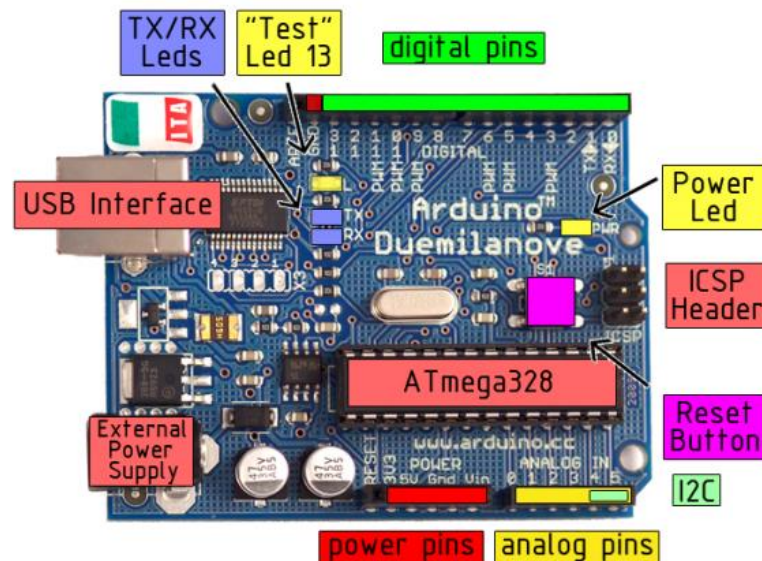
1. **Alimentación.** La placa Arduino se puede seleccionar entre la tensión de 5 V que ofrece el puerto de comunicación USB o una alimentación externa comprendida entre 7 a 12 voltios, que se regula y estabiliza a 5 voltios. Con la tensión ofrecida por el puerto USB se podrá controlar dispositivos de 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.



MIS PROYECTOS CON ARDUINO

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.



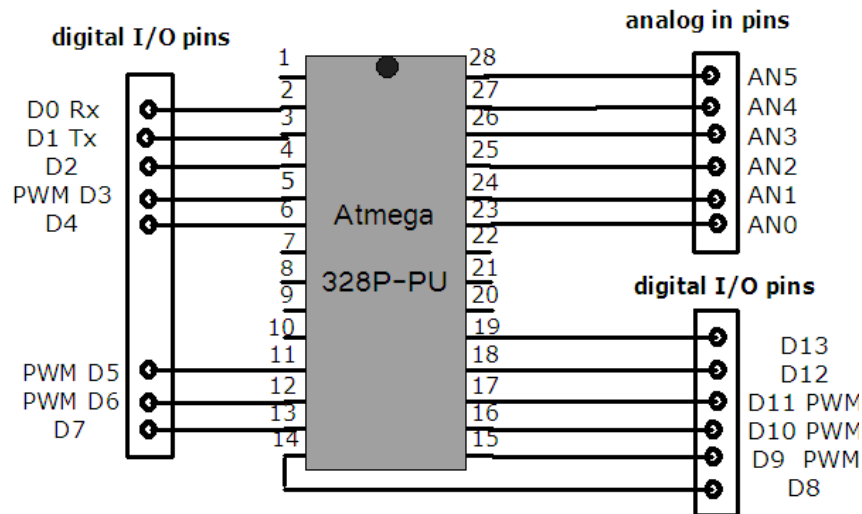
Pines de conexiones de E/S

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:
 - **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.
 - **3,3 V.** Este pin proporciona 3,3 voltios. Es posible que algún sensor o componente requiera de esa tensión para poder funcionar correctamente.
 - **5 V.** Este pin proporciona 5 voltios para alimentar los dispositivos, sensores y/o componentes electrónicos conectados a Arduino.
 - **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.
 - **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.

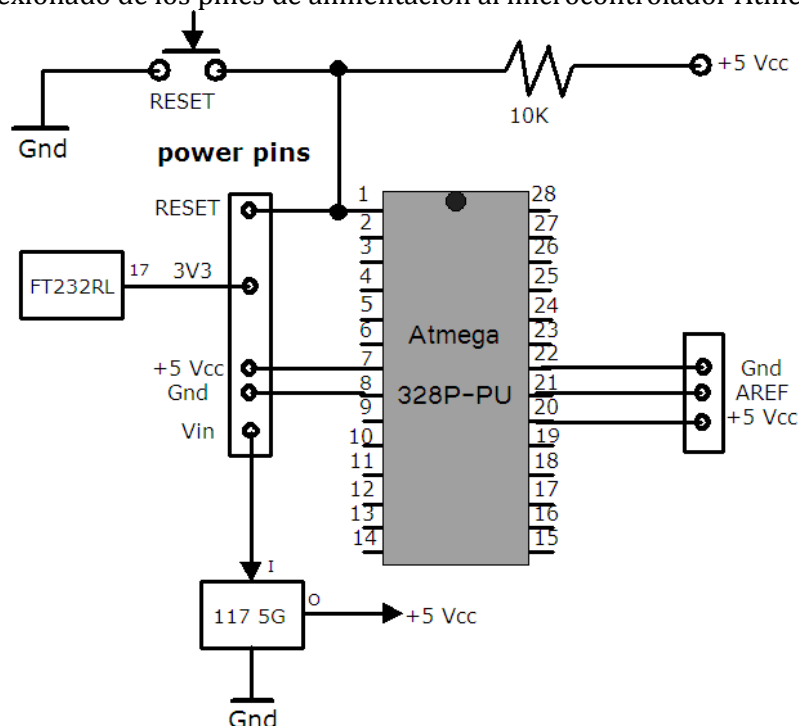
MIS PROYECTOS CON ARDUINO

Esquema de conexionado de los pines digitales y analógicos al 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.

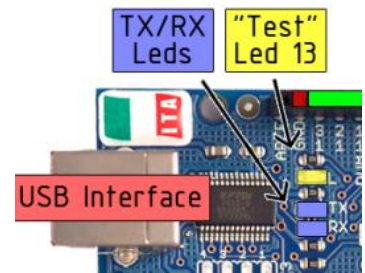


MIS PROYECTOS CON ARDUINO

Indicador TX/RX Leds

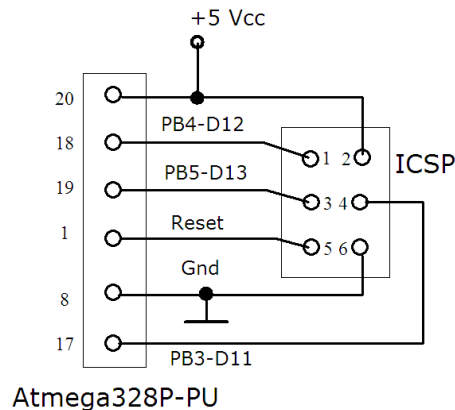
Indica que Arduino se está comunicando vía serie con el PC. Cuando esto ocurre, los indicadores parpadean, alertando de la transmisión y recepción de la información transmitida entre Arduino y el ordenador.

Los pines 2 y 3, **D0 (Rx)** y **D1 (Tx)**, del microcontrolador Atmega 328P 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 Leds**. 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

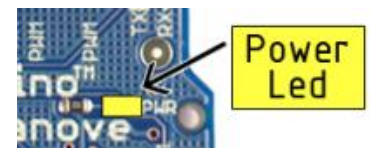
Se utilizan cuando se desea programar Arduino desde un entorno diferente del IDE y de la conexión típica por USB. Para realizar esta operación se requiere de un programador externo que irá conectado a los conectores mencionados. Si se desea programar Arduino de este modo, se deberá hacer en lenguaje 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 y la alimentación es correcta. Al mismo tiempo el ordenador detecta un nuevo dispositivo y lo instala.

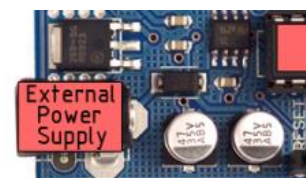


El pin AREF en Arduino.

Proporciona 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.



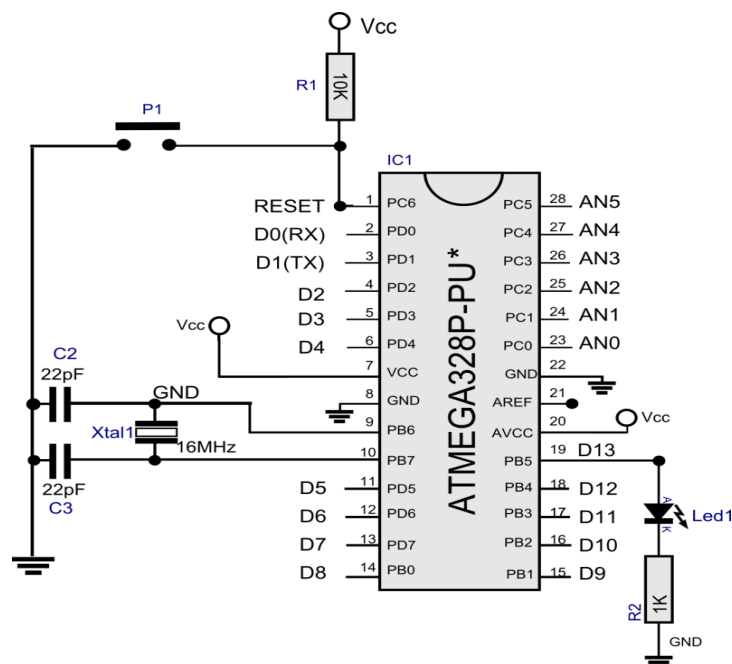
MIS PROYECTOS CON ARDUINO

Microcontrolador Atmega 328P

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	Descripción
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.

<http://www.alldatasheet.com/datasheet-pdf/pdf/241077/ATMEL/ATMEGA328P.html>



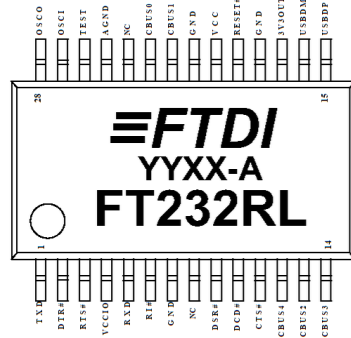
MIS PROYECTOS CON ARDUINO

3.2. Alimentación

Hay tres modos diferentes de alimentar la placa Arduino:

1. **Mediante el conector USB.** Cuando Arduino está conectado al PC, la placa está alimentada para poder programarla. El puerto USB proporciona 5 voltios, tensión suficiente para activar la placa y que empiece a funcionar.

El circuito que utiliza de puente entre el PC y la placa Arduino es el Circuito integrado **FTDI FT232RL** que nos provee de alimentación 3,3 V y 5 voltios.
<http://www.alldatasheet.com/datasheet-pdf/pdf/144591/FTDI/FT232RL.html>



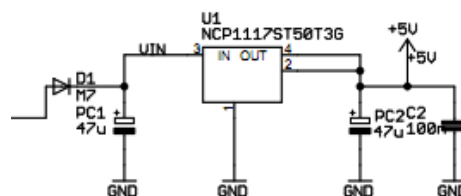
C.I. FTDI FT232RL



2. **Mediante un Jack de 2,1 mm del tipo rop de entrada.** La tensión deberá ser de 7 a 12 voltios en continua, siendo lo normal una tensión de 9 voltios, como una pila de petaca. Para ello se dispone de un regulador de tensión de 5 voltios instalado en la propia placa Arduino. Si es suministrada con menos de 7 V, puede suministrar menos de 5 voltios y la placa podría ser inestable. Si usas más de 12 voltios, el regulador de tensión puede sobrecalentarse y dañar la placa. El rango recomendado es de 7 a 12 voltios en continua.



Suministro externo.



Regulador 5V

3. **Mediante el pin de entrada Vin.** Tiene la misma función que el conector Jack de 2,1 mm, sólo que aquí se puede prescindir de este conector y se puede alimentar la placa mediante una tensión continua entre 7 y 12 voltios. Este pin se encuentra en el conector de alimentación (**power pins**) de la placa Arduino y además contiene los siguientes pines:

- **Vin.** Este terminal 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 9 mm.
- **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.

MIS PROYECTOS CON ARDUINO

3.3. 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).

<http://www.arduino.cc/en/Referencia/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.

<http://arduino.cc/en/Tutorial/Memory>

3.4. 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 – 50 KOhmios.

Algunos pines tienen funciones principales:

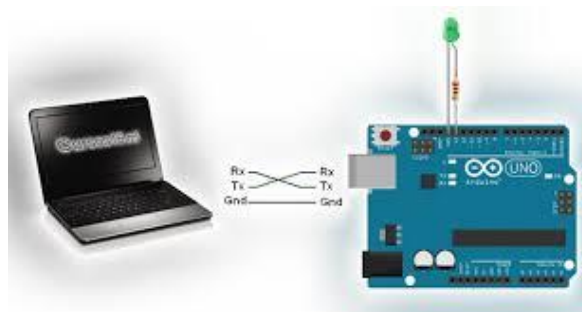
- **Serial: 0 (Rx) y 1 (Tx).** Usados para recibir (Rx) y transmitir (Tx) datos TTL, en serie. Estos pines están conectados a los pines correspondientes del chip FTDI USB-a-TTL Serie.
- **Interruptores externos: 2 y 3.** Estos pines pueden ser configurados para disparar un interruptor en un valor bajo, un margen creciente o decreciente, o un cambio de valor. Mirar la función *attachInterrupt()* <http://www.arduino.cc/en/Reference>
- **PWM: 3, 5, 6, 9, 10 y 11.** Proporcionan salida PWM de 8 bits con la función *analogWrite()*. <http://www.arduino.cc/en/Reference>
- **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** (analog pins).

MIS PROYECTOS CON ARDUINO

3.5. 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, 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 a datos de texto simple ser enviados a y desde la placa Arduino.

Una librería *SoftwareSerial* permite comunicación serie en cualquiera de los pines digitales del Duemilanove. <http://www.arduino.cc/en/Reference/SoftwareSerial>



3.6. 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. <http://www.arduino.cc/en/Main/Software>

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

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



3.7. 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.

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();
}
```

3.8. 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.

4. GRABACIÓN DEL BOOTLOADER

El *bootloader* de Arduino es un software alojado en la memoria flash que nos permite programar Arduino a través del puerto serie sin necesidad de usar un programador externo.

El *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 querer 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.

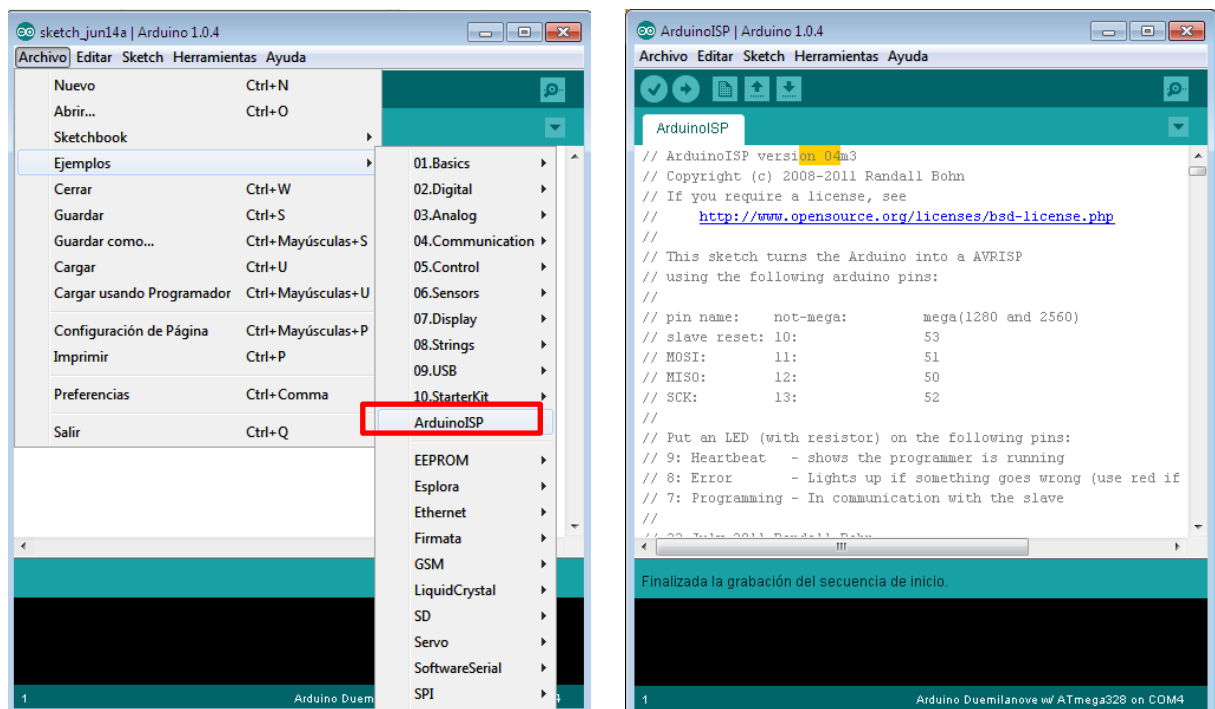
En cualquiera de los casos, **no necesitamos disponer de un programador externo** sino que podemos grabar el *bootloader* de un Arduino usando otro Arduino como programador.

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 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, en configuración *Slave*.

Para grabar el cargador de arranque, usando una placa de Arduino, hay que seguir estos pasos:

1. Cargue primeramente el programa **Archivo/Ejemplos/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.)



MIS PROYECTOS CON ARDUINO

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.

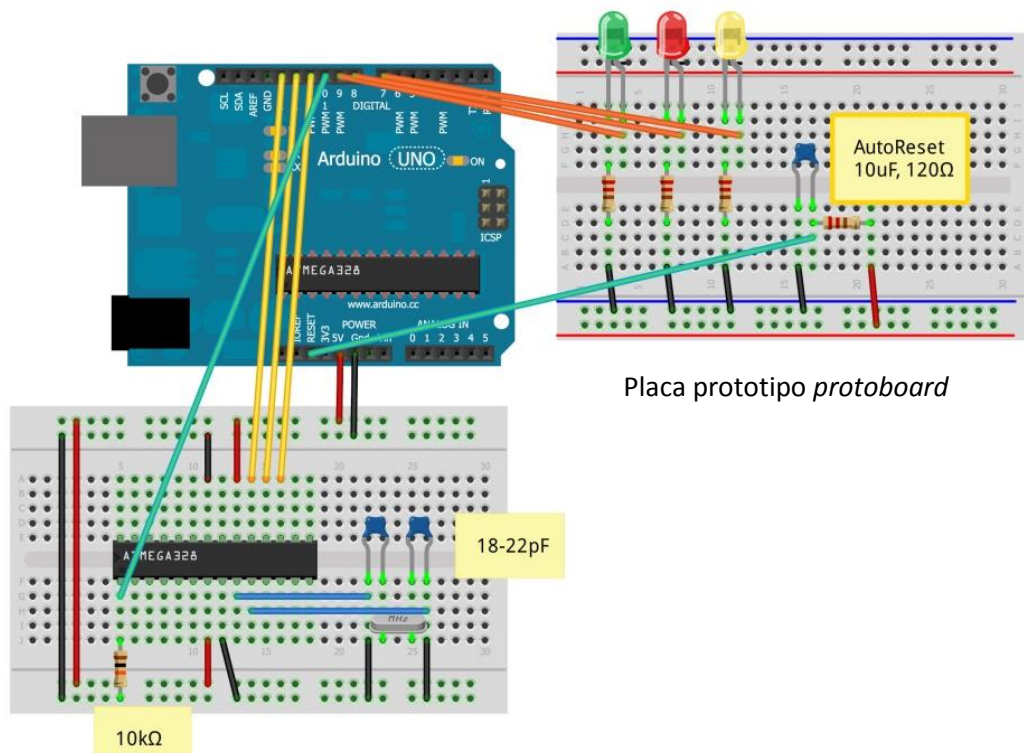
- Desconecte la placa de Arduino **Master** del PC
- Conecte la placa Arduino **Master** y el microcontrolador, sin el gestor de arranque, en la placa prototipo *protoboard* **Slave**, tal como se muestra en el siguiente esquema.

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.

Material necesario:

- 1. Placa Arduino Atmega328P-PU Duemilanove (Master)
- 1. Placa de prototipo protoboard (Slave)
- 1. ATmega328P-PU (Sin Bootloader o defectuoso)
- 1. Cristal de cuarzo a 16Mhz
- 2. Condensadores 18-22pF
- 1. Resistencia 10k Ω
- 3. Resistencias de 220-330 Ω (para los LEDs)
- 1. Resistencia 120 Ω
- 1. Condensador 10uF

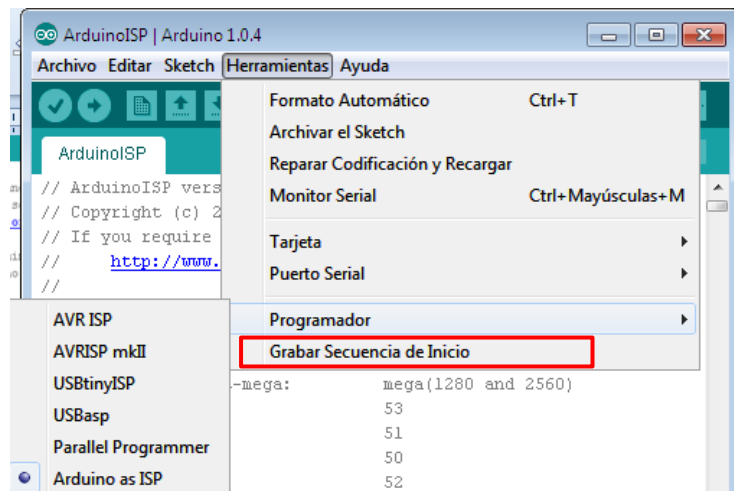
NOTA: La placa prototipo o *protoboard* se trata de una placa con conexiones internas en la que podemos *pinchar* nuestros componentes para realizar nuestros prototipos, sin tener que realizar un solo punto de soldadura de estaño, tantas veces como queramos. Debemos tener en cuenta como se distribuyen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



4. Conecte 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. Seleccione en el menú **Herramientas/Tarjeta/ "Arduino Duemilanove w/ATmega328"**.
6. Seleccione **Herramientas/Programador/"Arduino as ISP"**

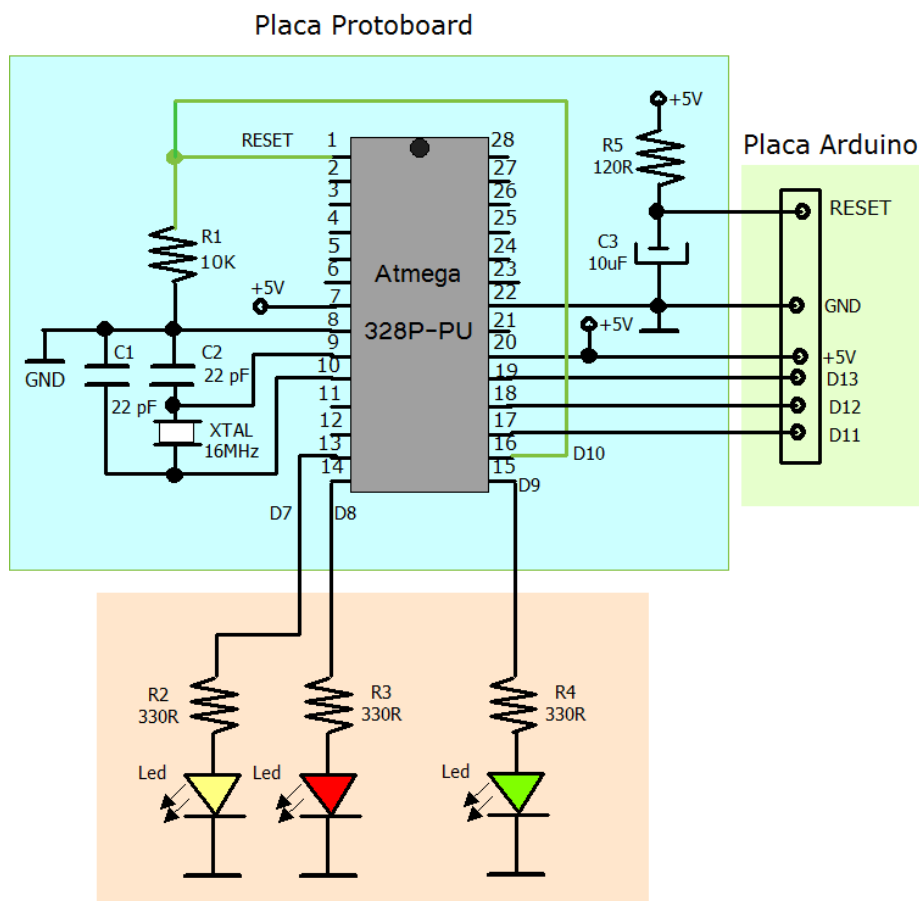
MIS PROYECTOS CON ARDUINO

7. 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/Grabar Secuencia de Inicio**



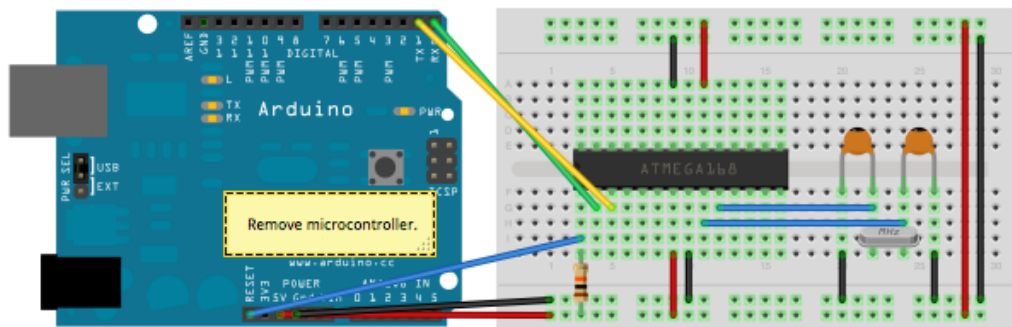
El proceso encenderá los Leds de la placa Arduino, **Lamp**, **Tx** y **Rx** y los Leds de la placa *protoboard* durante aproximadamente unos 15 segundos. Si todo va bien, se encenderá el LED amarillo de la placa Arduino y al cabo de un minuto tendremos nuestro Atmega328p listo para ser programado.

NOTA: Sólo debe grabar el cargador de arranque una vez. Después de haberlo hecho, puede quitar los cables de puente conectados a la placa de prototipo *protoboard Slave* y podrá insertar el microcontrolador en la placa Arduino y cargar un programa que sepamos su funcionamiento como "Blink" y comprobar que funciona correctamente.



5. TRABAJAR DIRECTAMENTE CON LA PLACA DE PROTOTIPO *PROTOBOARD*

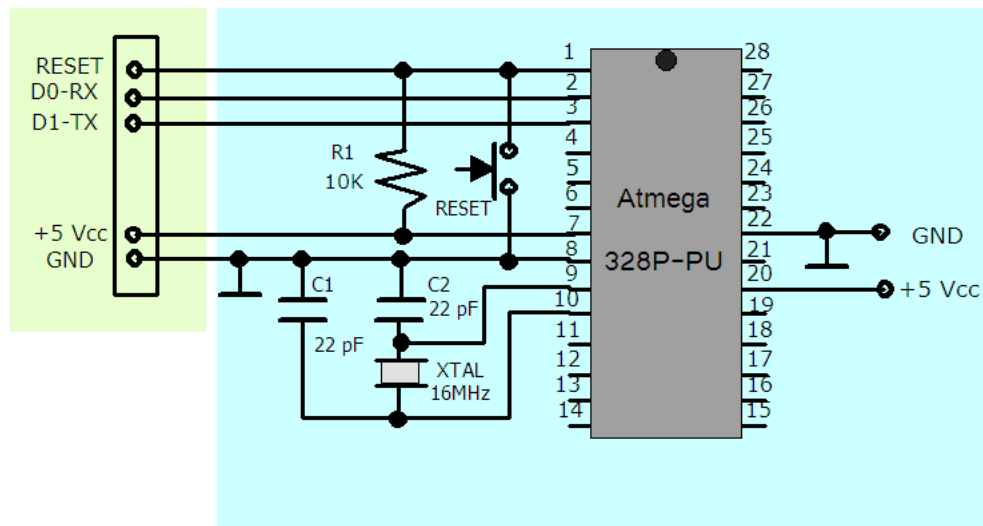
Una vez que su Atmega328P tiene cargado el gestor de arranque en él, puede cargar programas a él utilizando la propia placa de prueba *protoboard*, haciendo unas pequeñas modificaciones. Para ello, debe quitar el microcontrolador que viene en la placa de Arduino para que el chip de FTDI pueda comunicarse con el microcontrolador de la placa de prototipo *protoboard*. 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 "**Arduino Duemilanove w / ATmega328**" en el menú **Herramientas/ Tarjeta**. A continuación, sube un programa y trabaja sobre la placa *protoboard*.



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

Placa Arduino sin
Microcontrolador

Placa Proto-Board con
Microcontrolador



En esta configuración conectaremos los pines de comunicación **Rx** y **Tx** de la placa Arduino a los pines 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 y la correspondiente alimentación de +5 voltios y masa GND. A partir de ahora podemos ir trabajando con todos los componentes sobre la misma placa de prueba *protoboard* que contiene el microcontrolador.

NOTA: Es muy importante tener mucho cuidado en la manipulación del microcontrolador: en su desmontaje y montaje puede doblar alguna patilla. Evitar tocar el patillaje con los dedos.

MIS PROYECTOS CON ARDUINO

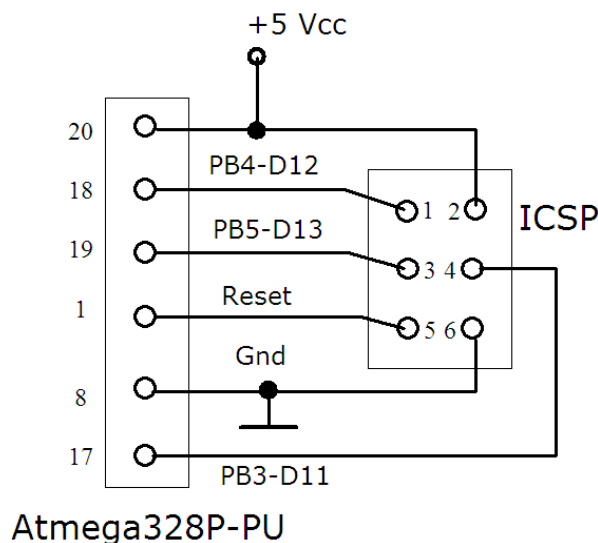
6. PLACA DE DESARROLLO DIY ATMEGA328P PARA ARDUINO UNO R3 SA

DIY ATmega328P es una placa de desarrollo para Arduino UNO R3 construido en Arduino UNO *bootloader*, para el proyecto Arduino. Tiene las mismas funciones de una placa Arduino, excepto que posee un zócalo especial que nos permite insertar fácilmente un Atmega328P-PU, tantas veces como queramos para grabar varios microcontroladores con el mismo programa, o grabarlo e instalarlo en una PCB o placa de prototipo. Se alimenta a través del puerto USB y no se puede suministrar tensión externa.



Características de la placa:

- Utiliza FT232 USB al módulo TTL, módulo CH340G, para grabar el programa.
- 100% compatible con el programa Arduino UNO R3, pantallas de expansión, entorno de desarrollo IDE.
- Microcontroladores: ATmega328P-PU
- Tensión de servicio: 5V
- IO voltaje lógico de la interfaz: 5V
- Corriente de funcionamiento: 500mA (máx.)
- Pines de E / S digitales: 14 (de los cuales 6 pueden proporcionar salida PWM)
- Pines de entrada analógica: 6
- Frecuencia de reloj: 16MHz
- LED de indicador de prueba de a bordo, posibilidad de programar para controlarlo.
- Indicador de alimentación *Power Led*.
- *Bootloader* que ejecuta Interfaz: ICSP
- Interfaz de programación: GND, 5V, RX, TX, DTR



7. COMPONENTES HARDWARE PARA CONECTAR CON ARDUINO

En los proyectos electrónicos intervienen unas series de componentes electrónicos que nos sirven para polarizar, señalizar, 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 un componente importante 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, sensores en las puertas automáticas cuando entramos en un supermercado, en los detectores de aproximación del vehículo, etc.

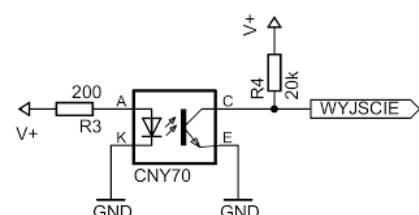
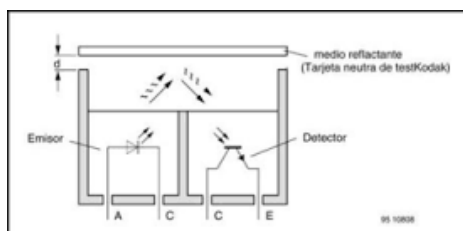
SENSOR DE CONTACTO O BUMPER

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 fines 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.



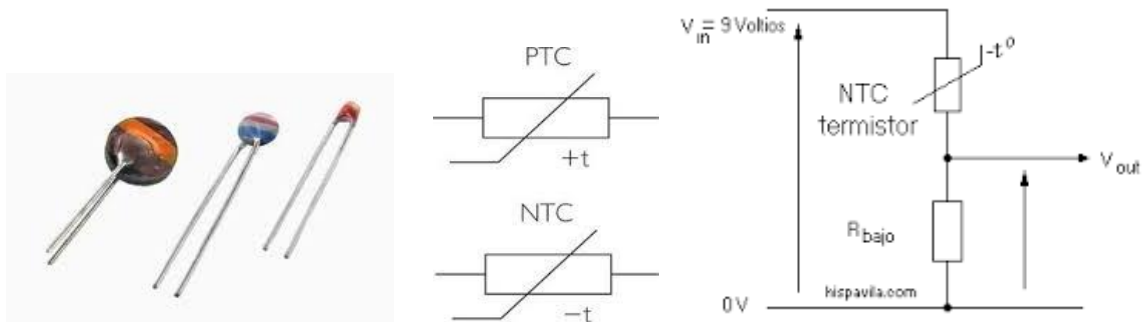
SENSOR ÓPTICO CNY70

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 reflectan. 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.



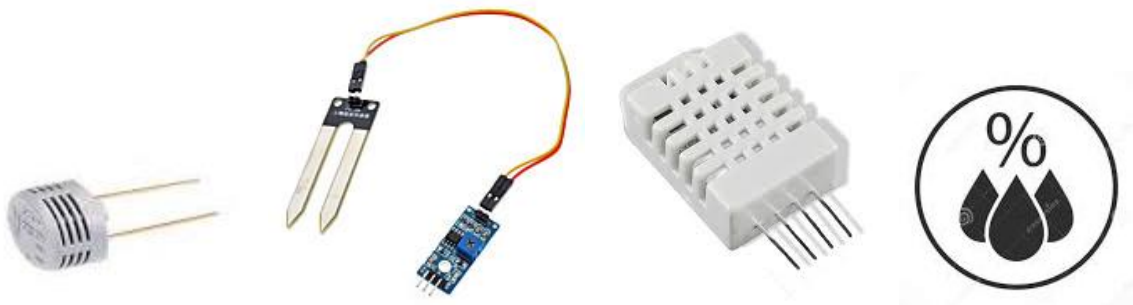
SENSOR DE TEMPERATURA

Se trata de resistencias cuyo valor asciende con la temperatura (termistor PTC) o bien disminuye con la temperatura (termistor NTC). Por lo tanto, depende de la temperatura el hecho de que este sensor permita o no el paso de la corriente por el circuito del sistema de control. La principal aplicación de los sensores térmicos, como es lógico, es la regulación de sistemas de calefacción y aire acondicionado, además de las alarmas de protección contra incendios, etc. Los principales son los termistores.



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.



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.



SENSOR 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 infrarrojos VS1838

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

Un sensor detector de movimientos por infrarrojos PIR, es un dispositivo pasivo que están formados por dos infrarrojos y una lente de *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. Pueden activar un relé, una lámpara, un timbre, una electroválvula, etc.



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.

SENSOR DETECTOR DE OBSTACULOS IR INFRARROJOS

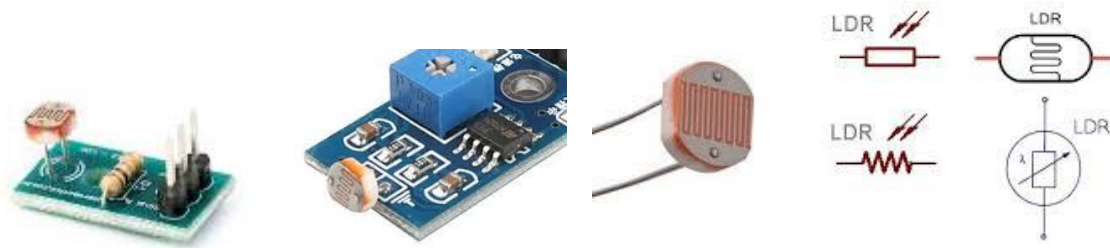
Este sensor de infrarrojos detecta obstáculos a una distancia de 2 a 30 cm. Con un ángulo de detección de 35°. Está constituido por un diodo emisor de infrarrojos y un fotodiodo receptor de infrarrojos que en el caso de detectar un objeto envía una señal, pudiendo activar un relé.



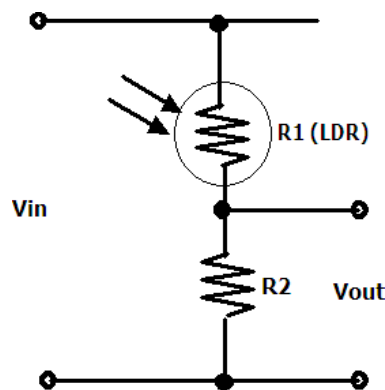
SENSORES DE LUZ LDR

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.



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

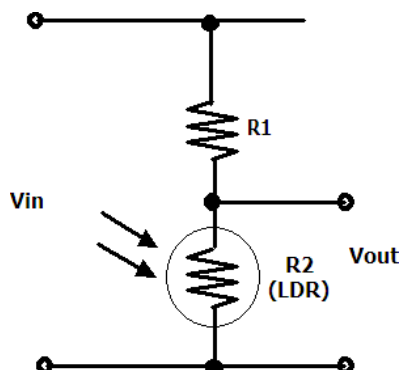


$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in}$$

Como se ha explicado anteriormente, tal como está configurado este divisor, la tensión resultante o de salida será la R2 entre la suma de la LDR y la resistencia R2, y todo multiplicado por la tensión de entrada Vin.

Si la R1 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 (R1) con la R2, quedando de esta manera, la LDR abajo y la R2 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.



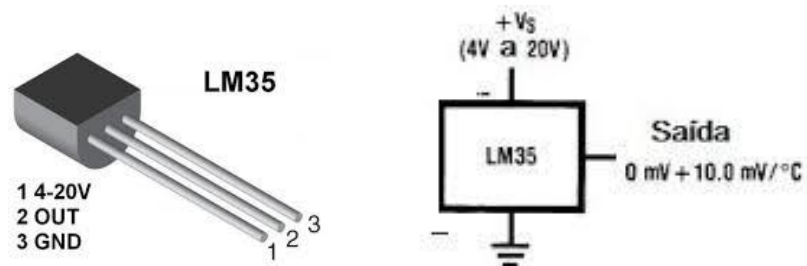
SENSORES DE TEMPERATURA NTC

Un sensor de temperatura NTC es un componente electrónico que 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.



SENSORES DE TEMPERATURA LM35

El sensor de temperatura LM35 produce una tensión proporcional a la temperatura a la que éste se vea sometido. Tiene tres patillas: Dos de alimentación y otra que nos entrega el valor de temperatura tomado. Éste trabaja de forma lineal.

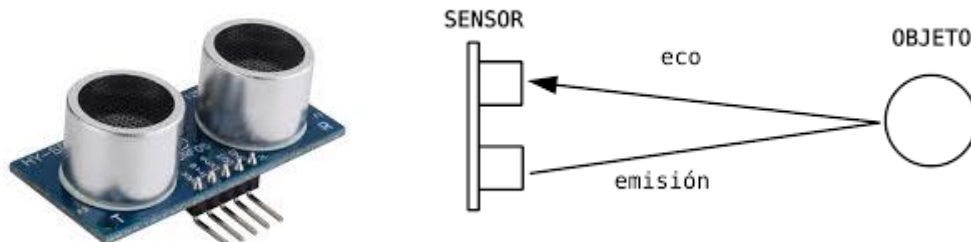


Este sensor entrega 10 mV por cada grado centígrado de temperatura. Debido a la naturaleza de la información resultante (en milivoltios), este sensor se deberá conectar en las entradas analógicas de Arduino y se deberá tratar matemáticamente para obtener unos valores en grados acorde con las temperaturas que normalmente utilizamos.

```
valor=analogRead(temp);  
Grados= (5.0*valor*100.0) / 1024;
```

SENSORES DE DISTANCIA (POR ULTRASONIDOS) SR04

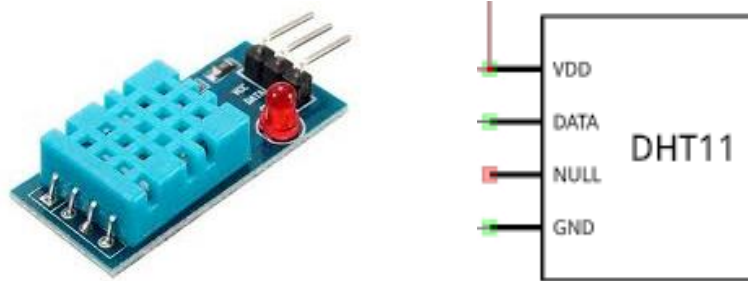
El sensor SR04 está compuesto por dos elementos, un emisor de ultrasonidos y un micrófono capaz de "escuchar" esas frecuencias que captaría las ondas sonoras que rebotan contra los objetos que se interpongan. De esta manera sabrá que tenemos un objeto delante y la distancia a la que éste se encuentra, calculando lo que la onda ha tardado en volver, ya que conocemos la velocidad del sonido en el aire.



MIS PROYECTOS CON ARDUINO

SENSORES DE TEMPERATURA Y HUMEDAD DHT11/22

El sensor DHT11 o DHT22 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.



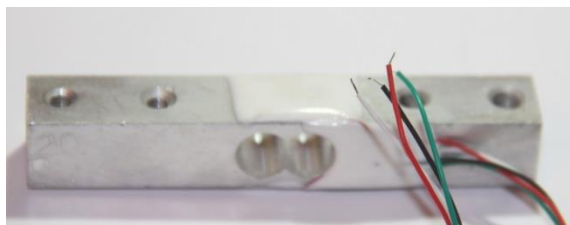
SENSOR DETECTOR DE POLUCION CONTAMINACIÓN MQ-135

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.



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.



MIS PROYECTOS CON ARDUINO

SENSORES ACELERÓMETROS

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 MICRÓFONOS

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.



SENSOR DE VELOCIDAD

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, por ejemplo.



MIS PROYECTOS CON ARDUINO

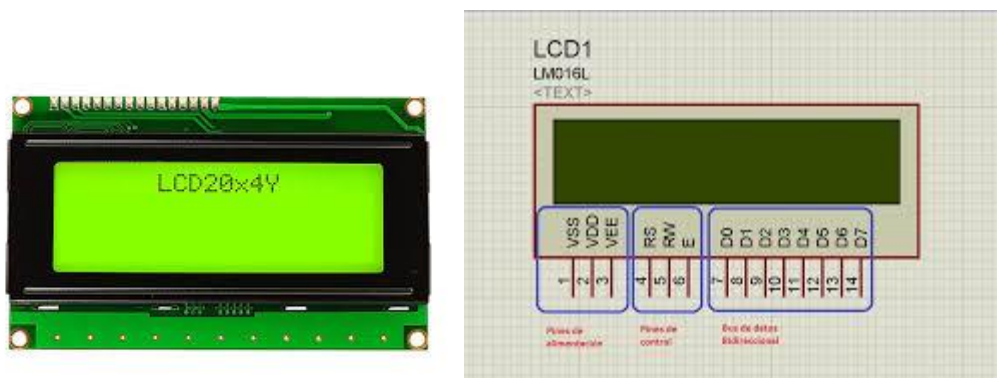
ACTUADORES LED

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 (-).



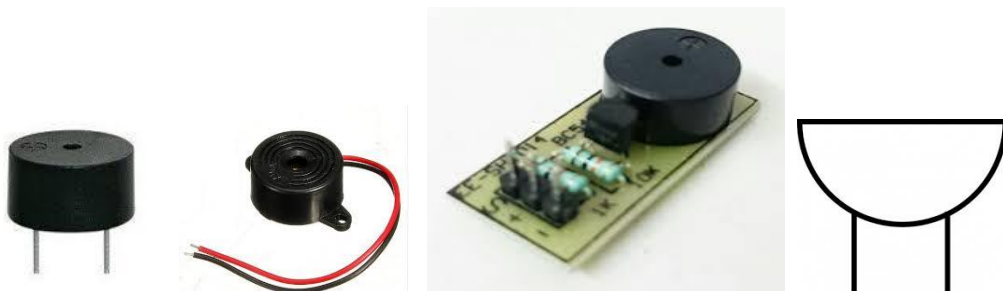
ACTUADORES PANTALLA LCD

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 ZUMBADORES

Produce 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.



MIS PROYECTOS CON ARDUINO

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 que cuando se pulsa, el circuito se cierra, con lo que la corriente circula por todo el circuito y, los hay que son normalmente cerrados, que cuando se pulsa el circuito se abre, con lo que la corriente deja de circular por el circuito eléctrico.



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.



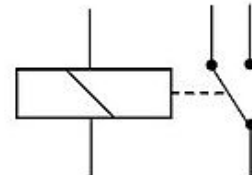
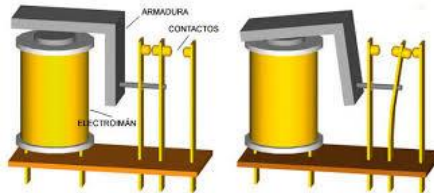
SERVOMOTORES

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°.



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. 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

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. Dispone de tres patillas. 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.

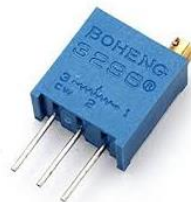
(1) Resistencia ajustable (2) Potenciómetro manual (3) Resistencia ajustable de precisión.



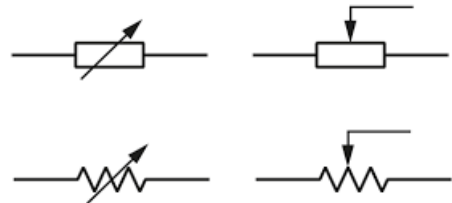
(1)



(2)

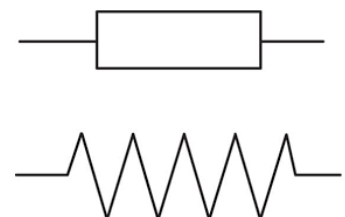


(3)



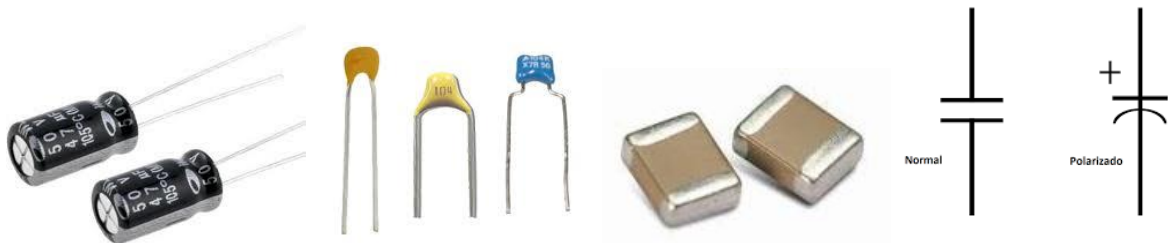
RESISTENCIAS

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.



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 potencia, adquieren una determinada carga eléctrica, positiva en una de ellas y negativa en la otra, siendo nula la variación de carga total.



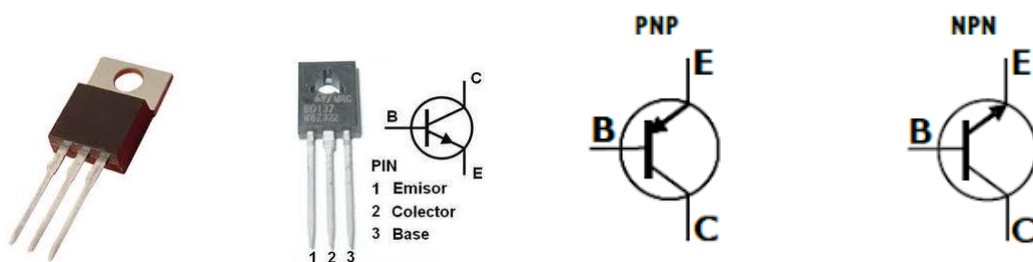
OPTOACOPLADORES

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. 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.



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.



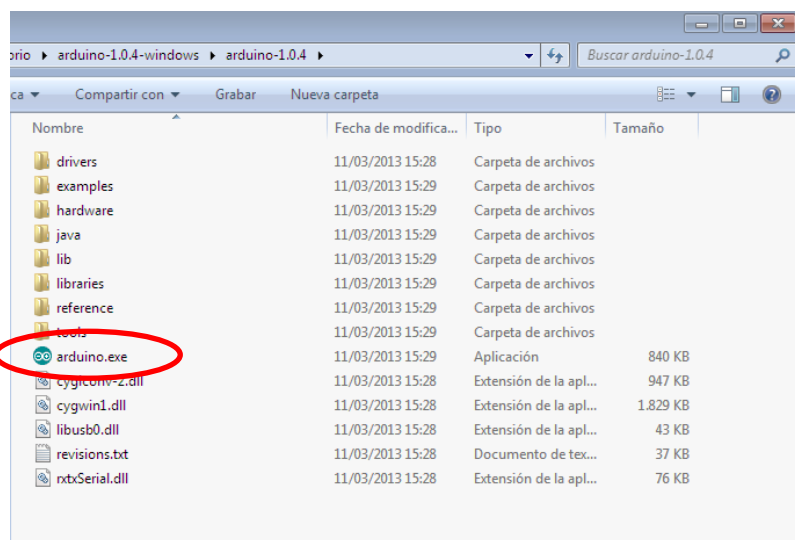
8. ENTORNO DE DESARROLLO DE ARDUINO

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.

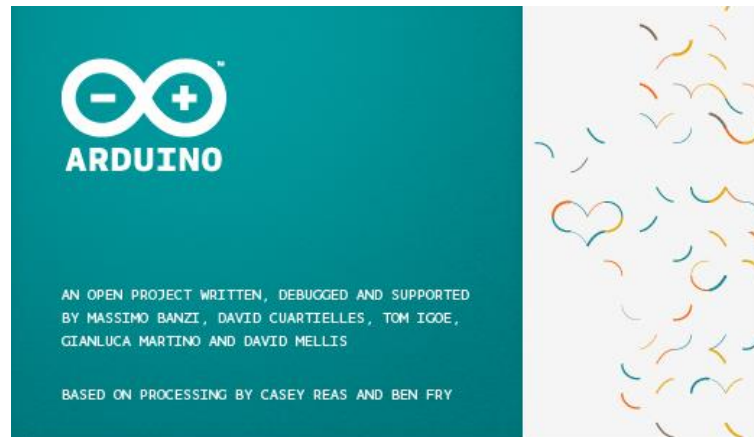


Una vez conectados PC y Arduino, el sistema operativo Windows detecta el dispositivo conectado al puerto USB. Normalmente, en los IDE actuales (1.5, 1.6, y en adelante) los drivers USB se instalan en el momento en que se está instalando el IDE; de esta forma, se detecta a Arduino automáticamente cada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

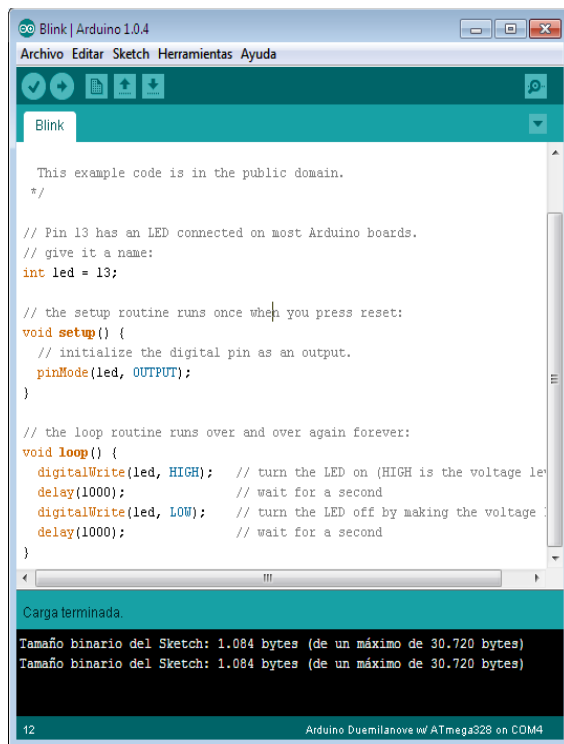
De todas formas si necesitamos instalar o reinstalar los drivers para el chip FTDI de la placa. Estos pueden encontrarse en el directorio drivers /FTDI USB Drivers de la distribuidora Arduino: <http://www.ftdichip.com/Drivers/VCP.htm>.

MIS PROYECTOS CON ARDUINO

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



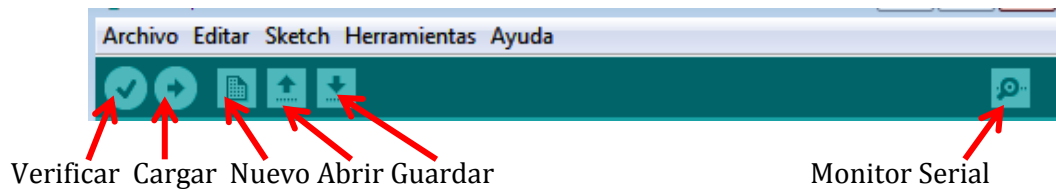
En las siguientes imágenes se muestran el aspecto del entorno de programación de Arduino IDE, con el programa **Blink** cargado y compilado correctamente y otro con error de compilación por faltarle un “**punto y coma**” en una de las instrucciones del programa.



Como se puede ver en las imágenes aparece un entorno de trabajo de Arduino donde podemos escribir nuestros códigos e instrucciones y el propio entorno nos va ayudando señalizando si el código introducido está bien, así como las llaves, paréntesis, valores, etc.

MIS PROYECTOS CON 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 sintácticos.

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.

MIS PROYECTOS CON ARDUINO

8.1. Menú Archivo

Este menú permite gestionar los archivos arduino con formato ***.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**.

Archivo	Editar	Sketch	Herramientas	Ayuda
Nuevo				Ctrl+N
Abrir...				Ctrl+O
Sketchbook				▶
Ejemplos				▶
Cerrar				Ctrl+W
Guardar				Ctrl+S
Guardar como...				Ctrl+Mayúsculas+S
Cargar				Ctrl+U
Cargar usando Programador				Ctrl+Mayúsculas+U
Configuración de Página				Ctrl+Mayúsculas+P
Imprimir				Ctrl+P
Preferencias				Ctrl+Comma
Salir				Ctrl+Q

8.2. Menú Editar

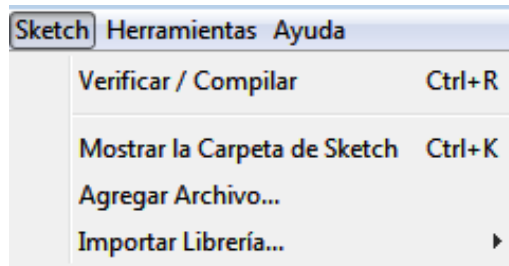
Este menú permite realizar la gestión del contenido de los archivos de arduino: copiar, cortar, pegar, seleccionar, etc. **En copia para el Foro**, nos permite copiar el código de nuestra rutina al portapapeles de forma conveniente para postear en un foro.

Editar	Sketch	Herramientas	Ayuda
Deshacer			Ctrl+Z
Rehacer			Ctrl+Y
Cortar			Ctrl+X
Copiar			Ctrl+C
Copiar para el Foro			Ctrl+Mayúsculas+C
Copiar como HTML			Ctrl+Alt+C
Pegar			Ctrl+V
Seleccionar Todo			Ctrl+A
Comentar/Descomentar			Ctrl+Slash
Incrementar Margen			Ctrl+Close Bracket
Reducir Margen			Ctrl+Open Bracket
Buscar...			Ctrl+F
Buscar Siguiente			Ctrl+G
Buscar Anterior			Ctrl+Mayúsculas+G
Utilizar Selección para Buscar			Ctrl+E

MIS PROYECTOS CON ARDUINO

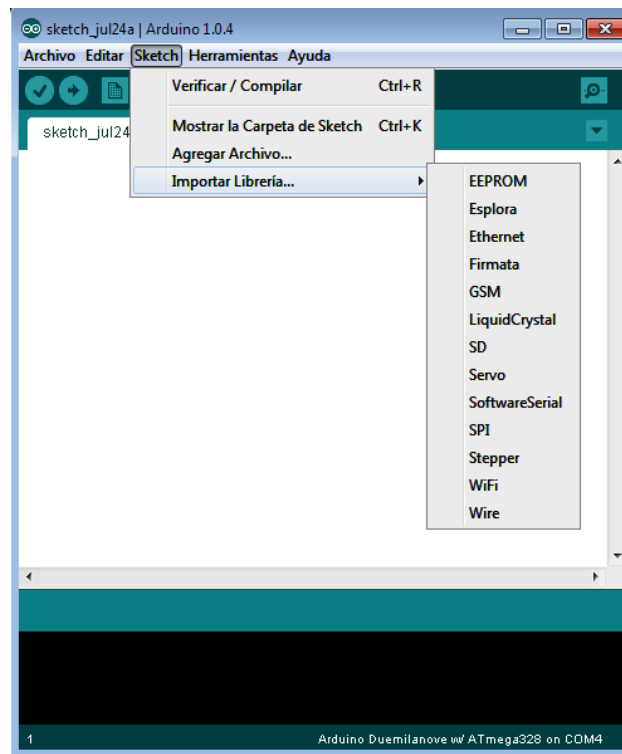
8.3. Menú Sketch

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.

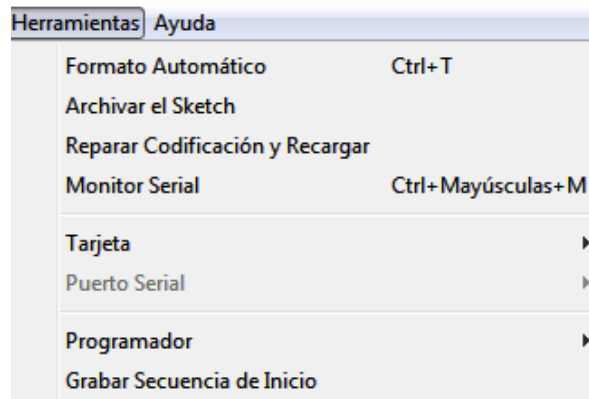


Las librerías que incorpora por defecto Arduino y que se deben invocar o cargar manualmente se pueden observar si hacemos clic en **Sketch / Importar Librería...** en el IDE de Arduino.

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 la hemos descargado e instalado nosotros mismos.

8.4. Menú Herramientas

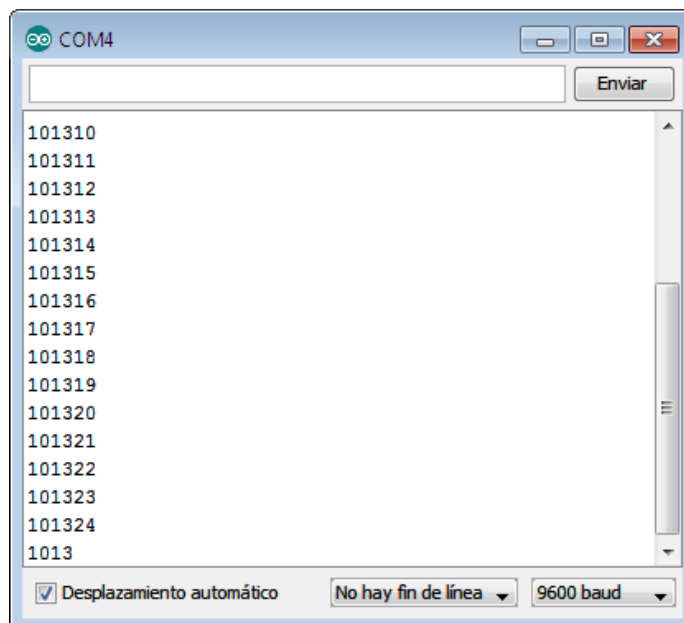
Este menú permite utilizar diversas opciones para mejorar la usabilidad de las instrucciones y códigos de nuestro programa.



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 códigos 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.



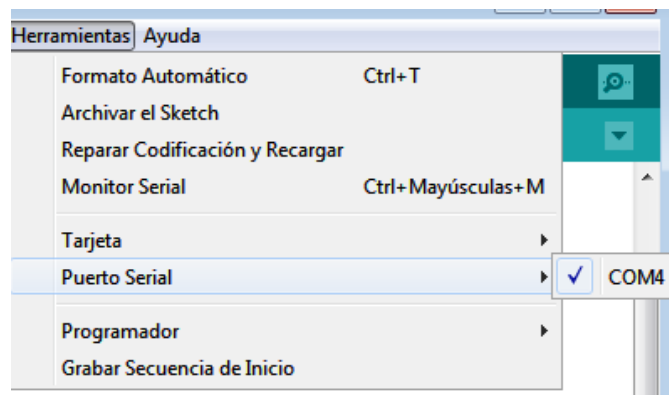
NOTA: Para realizar la conexión mediante puerto serie únicamente es necesario conectar nuestra placa Arduino empleando el mismo puerto que empleamos para programarlo. A continuación abrimos el IDE Standard de Arduino y hacemos click en el “Monitor Serial”.

MIS PROYECTOS CON ARDUINO

La opción **Tarjeta**, nos permite seleccionar, entre todas ellas, la placa de Arduino que corresponde con la que tenemos disponible. En nuestro caso es la **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).



MIS PROYECTOS CON ARDUINO

La opción de **Programador** permite grabar un *bootloader* 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 **Atmegs** 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**.



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

1. Obtener una placa Arduino Duemilanove Atmega328P-PU
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 encienden el LED verde de POWER, en 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 Rx y Tx
10. Ver que parpadea el LED naranja L del pin D13 cada 1 segundo.

9. LENGUAJE DE PROGRAMACIÓN 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.

9.1. 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*/
```

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
}
```

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
}
```

MIS PROYECTOS CON ARDUINO

Las llaves { }

Definen el comienzo y el final de bloques de función y bloques de declaraciones como *void loop()* 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;  
}
```

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 multilínea 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.

9.2. 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.

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
}
```

9.3. 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.

```
int inputValue = 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 se valores al 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 un código la misma aplicación te indica si está correcto, cambiando el color de la fuente.

9.4. Constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. 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);  
{  
  Ejecutar las instrucciones;  
}
```

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
```

9.5. 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; //declara"unaVariable" como variable de tipo //entero
```

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.

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.

Arrays

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  
  }  
}
```

9.6. 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. 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. Lectura: Con la sentencia **digitalRead()** leeremos el estado de un pin almacenado como HIGH o como LOW.

Función pinMode(pin, mode)

En las funciones de **void setup()** se configuran los pines específicos para la programación, estos pines se pueden comportar como 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/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
```

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
}
```

9.7. Funciones de Entradas Analógica

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 analógicas, **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-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"
```


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.

9.8. 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.

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:

```
/* 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 y destino minimo y maximo
  if (valor==255)
  { // si el valor de la ldr es igual a 255
    Serial.println("LUZ MAXIMA"); // mensaje de aviso.
  }
}
```

Con esta función es fácil incluirla en un programa si fuese necesario para resolver con éxito un proyecto de este tipo.

9.9. 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.

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.

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

Ejemplo:

```
/* 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);
  }
}
```

9.10. Funciones de tiempo y matemáticas

Gestión del tiempo en Arduino

En el lenguaje de programación de Arduino existen una serie 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
```

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
}
```

MIS PROYECTOS CON ARDUINO

- **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:

- **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 especificados
```

Si **valor** es menor que 100 **valor** recogerá su propio valor si **valor** es mayor de 100 **valor** pasará 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
// "valor" y 100
```

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

9.11. 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. **If**: Si se cumple se ejecutan las sentencias del bloque. 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 **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 él 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. Cuando se llama a una función esta se ejecuta y una vez que termina la función vuelve el flujo del programa a donde se llamó a la función. Ver el apartado 9.10 Crear nuestras propias funciones.

If (si condicional)

If es una declaración que se utiliza para probar si una determinada condición de 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;
}
```

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 qué solo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada.

```
if (inputPin < 500)
{
  Instrucciones;
}
else if (inputPin >= 1000)
{
  Instrucciones;
}
else
{
  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;
}
```

MIS PROYECTOS CON ARDUINO

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 `i` en el 0, y la condición es probar que el valor es inferior a 20 y si es cierta `i` se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves:

```
for (int a=0; a<20; a++)    // declara 1 y prueba si es menor que 20, incrementa i
{
  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
}
```

Este código 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 testea 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)    // testea si es menor que 200
{
  Instrucciones;           // ejecuta las instrucciones entre llaves
  unaVariable++;           // incrementa la variable en 1
}
```

do ... while

El bucle **do while** 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;
}
While (unaVariable ? 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.

```
do
{
x=leeSensor();
delay(50);
}
while (x<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
}
}
```

9.12. 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.

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();
```

Veamos un ejemplo para esclarecer los conceptos:

```
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  
}
```

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()*.

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 claudador que cierra el *void loop*.

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.

9.13. 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;  x= x -7;  i= j * 6;  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`.

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

NOTA: 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 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, **AND** (&&), **OR** (||) y **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.

MIS PROYECTOS CON ARDUINO

Lógica AND

Este operador permite validar 2 o más condiciones, las cuales se deben cumplir todas a la 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

Se cumple cuando ninguna de las instrucciones se cumple. 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.

MIS PROYECTOS CON ARDUINO

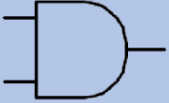
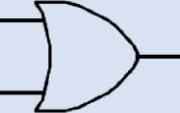
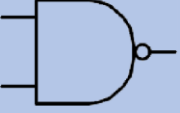
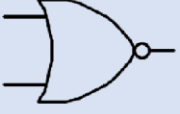
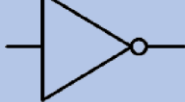
Operador	Nomenclatura en Arduino	Significado	Símbolo
AND	&&	Devuelve true cuando la primera Y la segunda condición se cumplen	
OR		Devuelve true cuando la primera O la segunda condición se cumple	
NAND	(i=) && (i=)	Devuelve true cuando NI la primera NI la segunda condición se cumplen	
NOR	(i=) (i=)	Devuelve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

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

9.14. 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.

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.
```

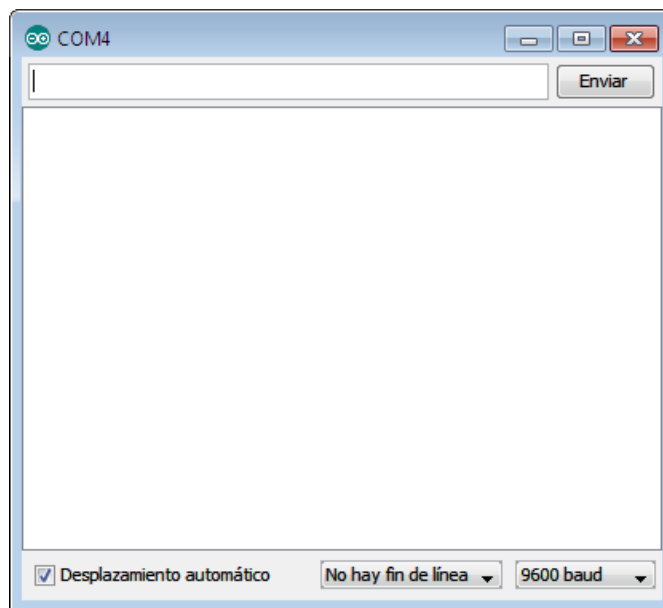
El siguiente ejemplo genera un valor aleatorio entre 0 – 255 y lo envía a una salida analógica PWM:

```
int randNumber; // 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 de //la  
  función millis  
  
  ranNumber = random(255); // genera numero aleatorio entre 0-255  
  
  analogWrite(led, ranNumber); // envía a la salida led de tipo PWM el //valor  
  delay(500); // espera 0,5 segundos  
}
```

9.15. 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. Por ejemplo: 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.

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. 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);
}

void loop(){
  Serial.println('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.

MIS PROYECTOS CON ARDUINO

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')**.

Cuando usamos **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.

NOTA: 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 //9600
  bps
}
```

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
}
```

MIS PROYECTOS CON ARDUINO

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.

Ejemplo:

```
Serial.print(b); // Vuelca o 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); //Vuelca o envía el valor de b como un numero //hexadecimal
en caracteres ASCII "4F"
Serial.print(b, BIN); // Vuelca o 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); // Vuelca o 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.available() /* 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: ");
    Serial.println(incomingByte, DEC);
  }
}
```

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
  {
    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;
void setup()
{
  pinMode(13, OUTPUT); // Declaramos que utilizaremos el pin 13 como salida
  Serial.begin(9600); // establecemos el rango en baudios en 9600
}
void loop()
{
  if (Serial.available()>0){
    input=Serial.read();
    if (input=='1'){
      digitalWrite(13, HIGH); //Si el valor de input es 1, se enciende el led
    }
    else
    {
      digitalWrite(13, LOW); //Si el valor de input es diferente de 1, se apaga
el LED
    }
  }
}
```

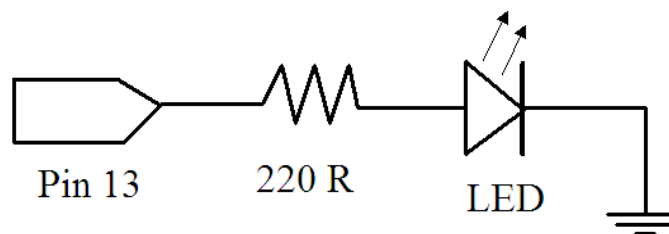

9.16. Ejemplos de programación básica de Arduino

A continuación se describen algunos ejemplos básicos de la programación para utilizarlo en componentes auxiliares que podemos conectar a nuestra placa Arduino.

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 veremos más adelante.

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 Arduino ya incluye en la tarjeta esta resistencia.



```
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); // activa el LED

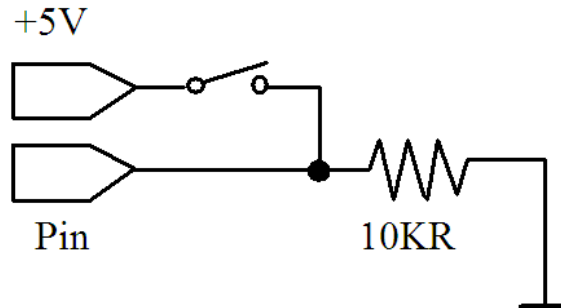
  delay(1000); // espera 1 segundo

  digitalWrite(ledPin, LOW); // desactiva el LED

  delay(1000); // espera 1 segundo
}
```

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.



```
int ledPin = 13;    // pin 13 asignado para el LED de salida
int inPin = 2;      // pin 2 asignado para el pulsador

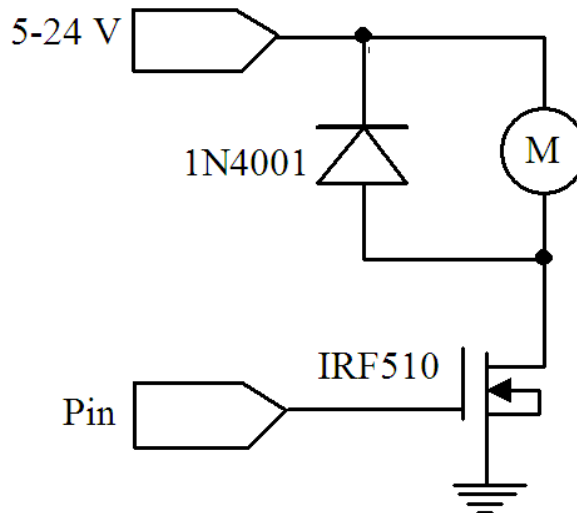
void setup()        // configura entradas y salidas
{
  pinMode(ledPin, OUTPUT);    // declara LED como salida
  pinMode(inPin, INPUT);      // declara pulsador como entrada
}

void loop()
{
  if (digitalRead (inPin) == HIGH)    // testea si la entrada esta HIGH
  {
    digitalWrite(ledPin, HIGH);    // enciende el LED
    delay(1000);                    //espera 1 segundo
    digitalWrite(ledPin, LOW);      // apaga el LED
  }
}
```

Salida de alta corriente de consumo

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

NOTA: El esquema muestra un motor con un diodo de protección por ser una carga inductiva. En los casos que las cargas no sean inductivas no será necesario colocar el diodo.



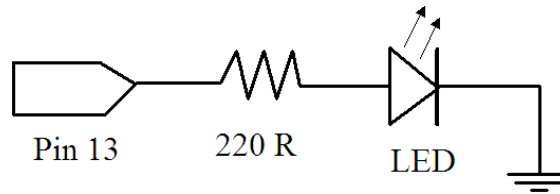
```
int outPin = 5; // pin de salida para mosfet

void setup()
{
  pinMode (outPin, OUTPUT); // pin5 como salida
}

void loop()
{
  for (int i=0; i<=5; i++) // repetir bucle 5 veces
  {
    digitalWrite(outPin, HIGH); // active el MOSFET
    delay(250); // espera 250 milisegundos
    digitalWrite (outPin, LOW); // desactiva el MOSFET
    delay(250); // espera 250 milisegundos
  }
  delay(1000); // espera 1 segundo
}
```

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.



```
int ledPin=9;    // pin PWM para el LED
void setup()     // no es necesario configurar nada
void loop()
{
  for (int i=0; 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 ii
    delay(100);                // pausa durante 100ms
  }
}
```

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.

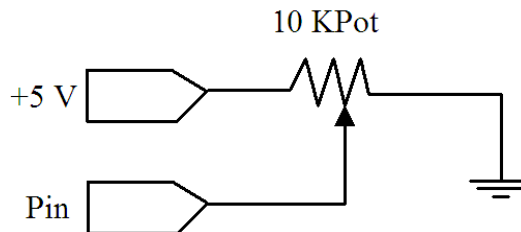
Veamos otro ejemplo:

```
/*Luminosidad variable de un led mediante pulsos PWM*/
void setup(){
  pinMode(5,OUTPUT); // declaramos de salida el pin 5
}
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
  }}
```

Entrada con potenciómetro (Entrada analógica)

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 para controlar el tiempo de parpadeo de un LED.

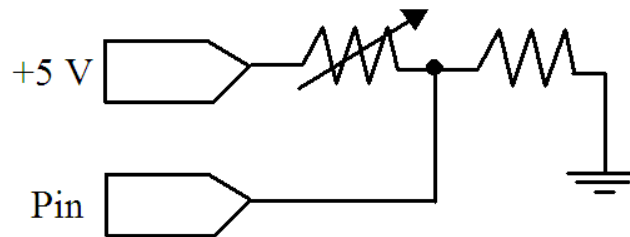


```
int potPin=0;      // pin entrada para potenciómetro
int ledPin=13;     // pin de salida para el LED
void setup()
{
  pinMode (ledPin, OUTPUT);  // declara ledPin como SALIDA
}
void loop()
{
  digitalWrite(ledPin, HIGH); // pone ledPin en ON
  delay(analogRead(potPin);   // detiene la ejecución un tiempo "potPin"
}
```

Entrada conectada a resistencia variable (Entrada analógica)

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 y establecer un tiempo de retardo. Este tiempo controla el brillo de un diodo LED conectado en la salida.



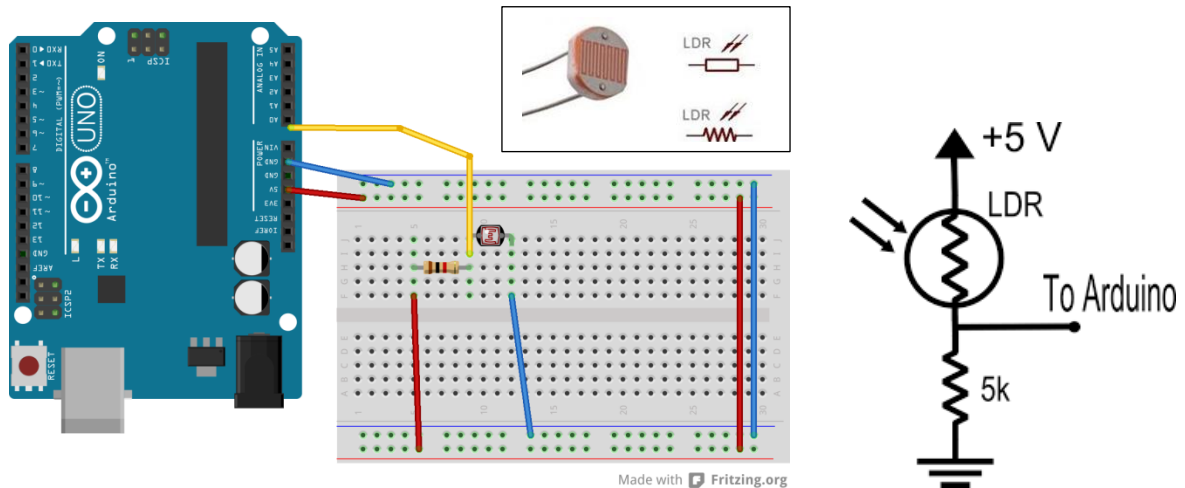
```
int ledPin=9;           // salida analógica PWM para conectar un LED
int analogPin = 0;      // resistencia variable conectada a la entrada analógica
// pin 0
void setup()            // no es necesario configurar entradas y salidas
void loop()
{
  for (int i=0; i<=255; i++) // incrementa de valor de i
  {
    analogWrite (ledPin, i); // configura el nivel brillo con el valor de i
    delay(delayVal());      // espera un tiempo
  }
  for (int i=255; i>=0; i--) // decrementa el valor de i
  {
    analogWrite(ledPin, i); // configura el nivel de brillo con el valor de i
    delay(delayVal());      // espera un tiempo
  }
}
int delayVal()
{
  int v;
  v= analogRead(analogPin) // crea una variable temporal (local)
  // lee valor analógico
  v/=8;                    // convierte el valor leído de 0-1024 a 0-128
  return v;                // devuelve el valor v
}
```

A continuación realizaremos una pequeña experiencia en la cual se obtendrán mediciones de una fotorresistencia o LDR.

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 MΩ o más en la oscuridad.

MIS PROYECTOS CON ARDUINO

Para esta experiencia utilizamos la configuración siguiente:



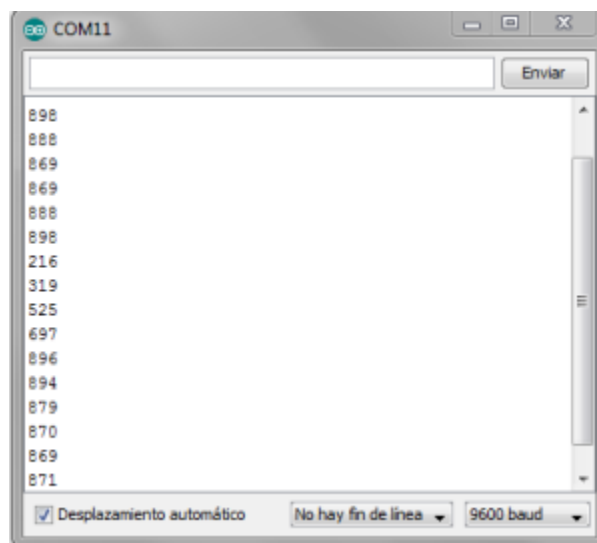
Abrimos el entorno de Arduino IDE e introducimos los siguientes códigos:

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

void loop()
{
  int m= analogRead(A0);
  Serial.println(m);
  delay(5000);
}
```

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

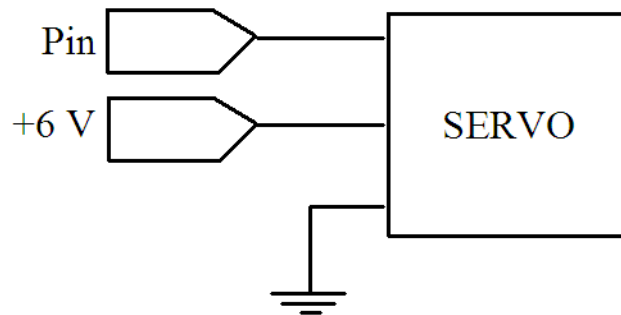
Este código obtendrá una medición cada 5 segundos correspondientes a las fracciones del voltaje de aplicación que hay en el punto de medición.



En el código anterior advertimos que los valores que captamos mediante la fotocélula LDR están comprendidos entre 0 a 1.023. Estos son los valores que pueden tomar los sensores con Arduino.

Salida conectada a servo

Los servos de modelismo tienen un motor y unos engranajes cuya salida se puede mover en un arco de 180° y contienen la electrónica necesaria para ello. Todo lo que se necesita es un pulso enviado cada 20 ms. Este ejemplo utiliza la función **servoPulse** para mover el servo de 10° a 170°.



```
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 10° y gira hasta 170°
{
  for (myAngle=10; myAngle<=170, myAngle ++ )
  {
    servoPulse (servoPin, myAngle); // servo vuelve desde 170° hasta 10°
  }
  for (myAngle=170; myAngle>=10; myAngle);
}
}
```

Control de servomotor

Básicamente un servomotor se compone de un motor de corriente continua, un reductor de velocidad y un multiplicador de fuerza, todo ello gestionado por una electrónica adicional e introducida bajo un chasis de plástico, generalmente de color negro.

Para controlar el servomotor se le envía pulsos cada 20 ms es decir 50Hz. La anchura del pulso es lo que codifica el ángulo de giro, es decir lo que se conoce como PWM, codificación por ancho de pulso. Esta anchura varía según el servomotor pero normalmente va entre 0.5 y 2.5 ms aunque pueden variar.

Dependiendo del tamaño del servo y su consumo es posible que no puedas alimentarlo desde tu placa arduino, en ese caso es necesaria una fuente de 5V independiente para poder moverlo. Sobre el peso que pueden levantar se puede deducir con el par del servo. Normalmente los servos indican el par o torque que pueden realizar para un servo estándar suele ser 5kg/cm es decir puede mover 5kg a 1 cm de distancia. En caso de querer moverlo a 5 cm el servo solo podrá mover 1kg.

A continuación vamos a ver como controlar en Arduino un servomotor. Para ello iniciamos el entorno de Arduino y vamos a los *Sketch* de ejemplos en **Archivo/Ejemplos/Servo/Knob**

Ejemplo de control de una posición de servo mediante un potenciómetro (resistencia variable)

```
// 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
}
```

10. 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 tarjeta está conectada.

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.

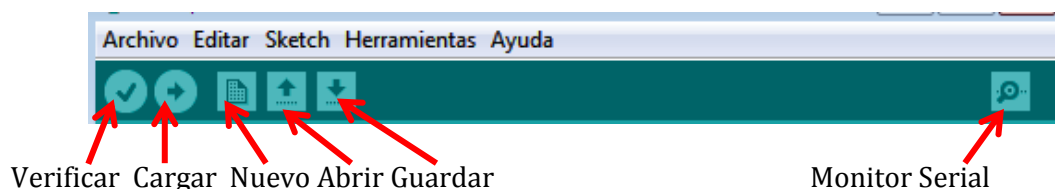
Posteriormente vamos al menú “**Herramientas**” y en la opción “**Tarjeta**” seleccionamos el tipo de tarjeta que coincida con la que tenemos conectada, en este caso sería **Arduino Duemilanove w/ATmega328**.

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).

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 fuente es el correcto. 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.

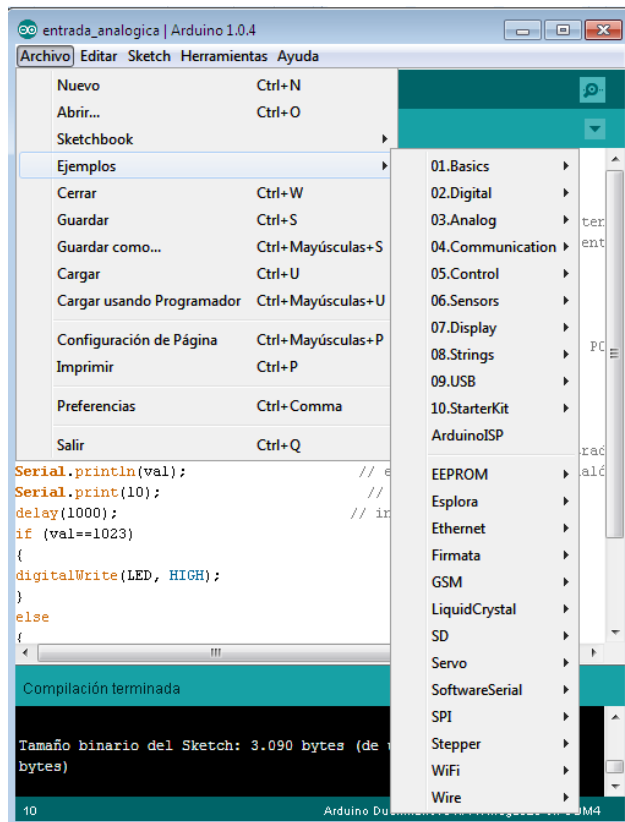


Durante la carga del programa, en la placa de Arduino, se encenderán los LEDs de color naranja de comunicación **Tx/Rx** que nos indican que se está produciendo transmisión y recepción de datos por el **Puerto Serial**.

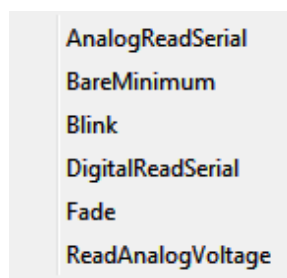
NOTA: Para mayor rapidez, se puede obviar la verificación y pulsar directamente “cargar”, puesto que también el programa antes de cargarlo en la memoria flash lo verifica.

MIS PROYECTOS CON ARDUINO

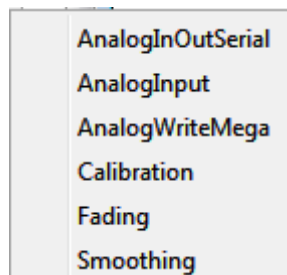
En el entorno de desarrollo de Arduino nos permite cargar diversidad de ejemplos de Sketch de diferentes autores y utilidades específicas: Digital, Analógico, Control, Sensores, Display, comunicación, etc., accediendo al menú **Archivo/Ejemplos**:



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



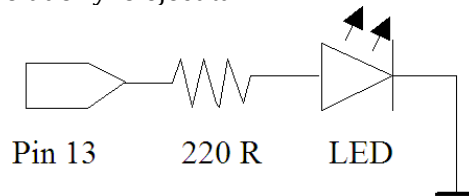
En la opción **Analog**, tenemos:



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.

10.1. Ejemplo de programación: 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.
  */

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

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

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

10.2. Prácticas con Arduino

Aumentar y disminuir intensidad luminosa de Led (fading)

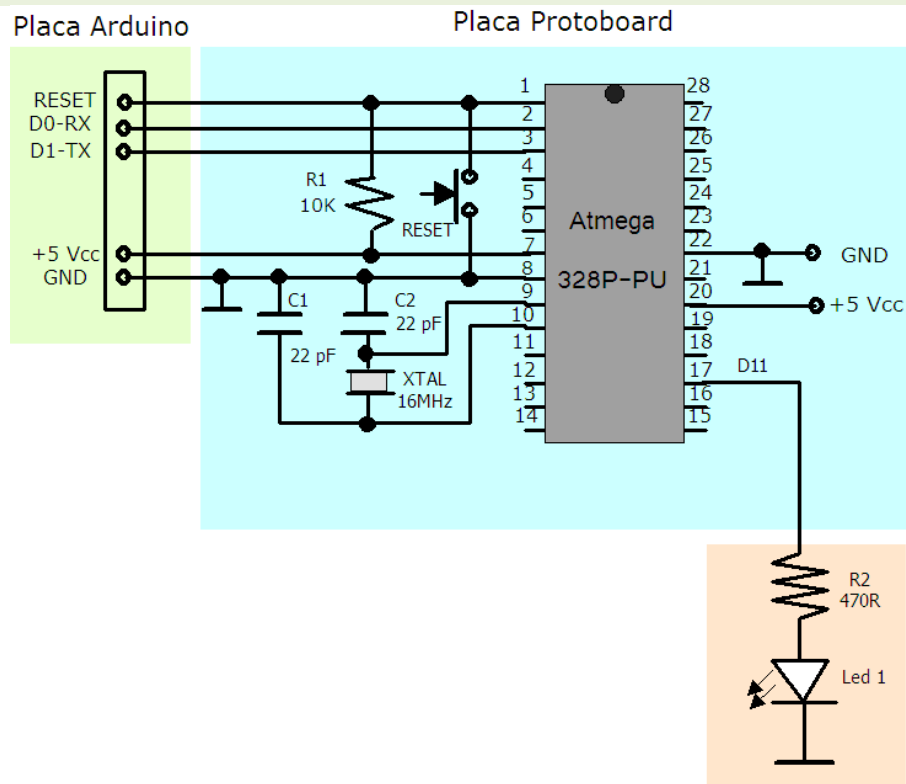
En esta práctica programaremos un Led que irá aumentando y disminuyendo la luminosidad usando la capacidad de ofrecer una tensión variables que da una salida analógica. Para ello se conecta un Led al pin D11 y se provoca que su luminosidad pase de mínima a máxima, para luego ir de máxima a mínima. Los 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
}
```



Aumentar luminosidad de Led con pulsador (fading)

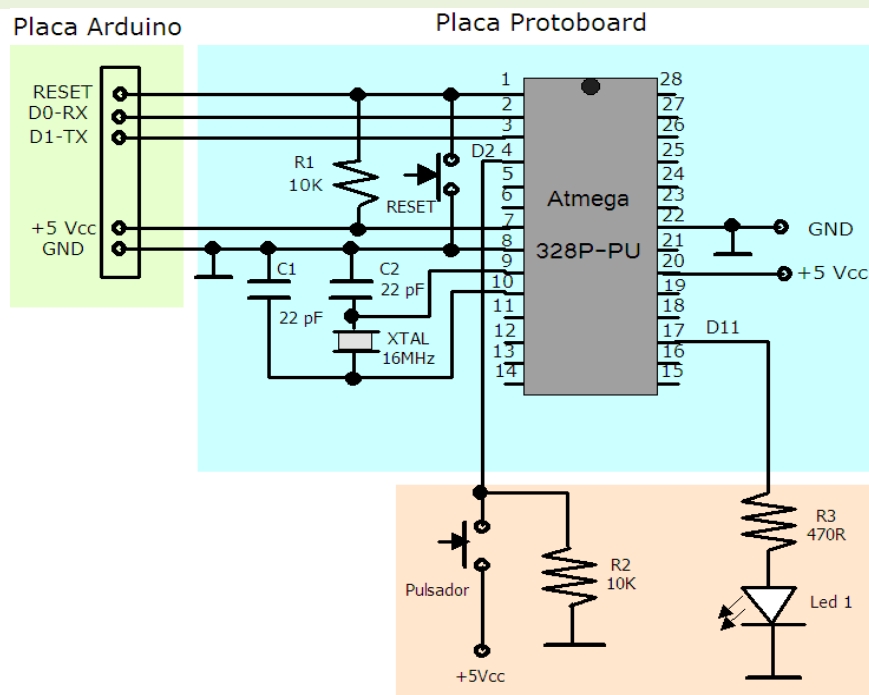
En esta otra práctica muy semejante a la anterior, aumenta y disminuye la luminosidad de un Led pero en este caso controlada mediante un pulsador. El pulsador debe estar conectado a un pin digital D2 como entrada y como salida tendremos el diodo Led en el pin D11.

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).

```
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;
  }
  if (x>255)
  {
    x=0; // asigna el valor 0 a x
    analogWrite(led,0); // apaga el Led
  }
}
```



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
}
```

MIS PROYECTOS CON ARDUINO

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

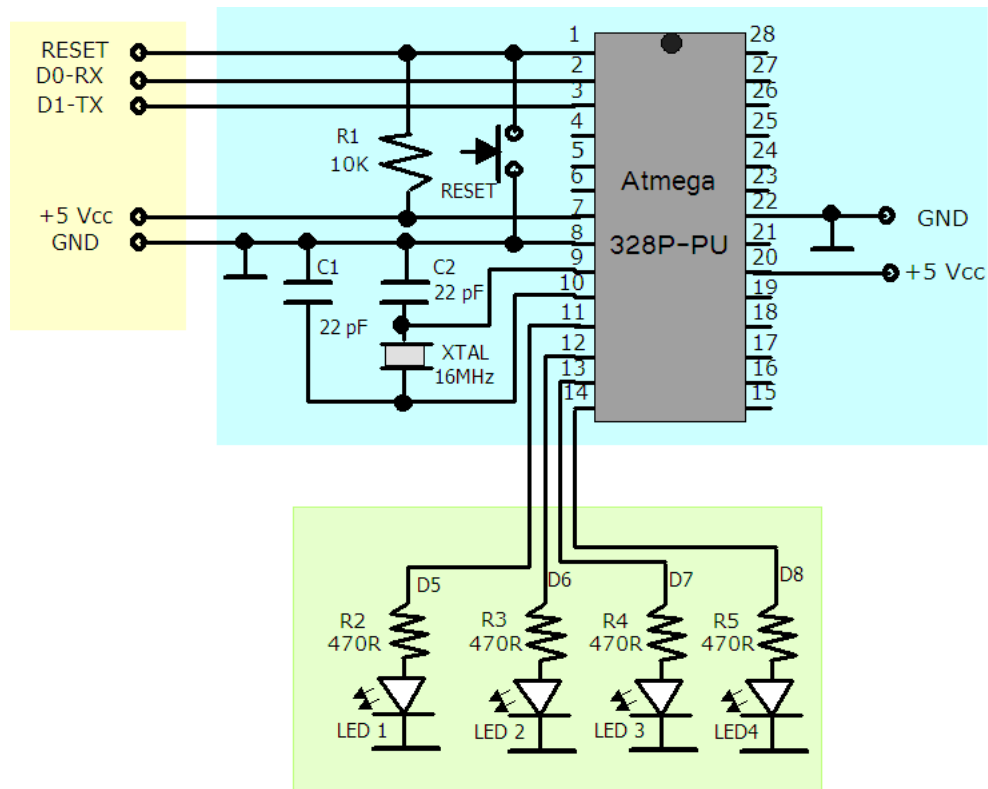
    for(n=0; n<4; n++) // contador de n de 1 a 4
    {
        pinMode(Leds[n],OUTPUT); // toma valores Leds con salida
    }
}

void secuencia()
{
    // bloque de instrucciones secuencia
    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
}
```

MIS PROYECTOS CON ARDUINO

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



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
  }
}
```

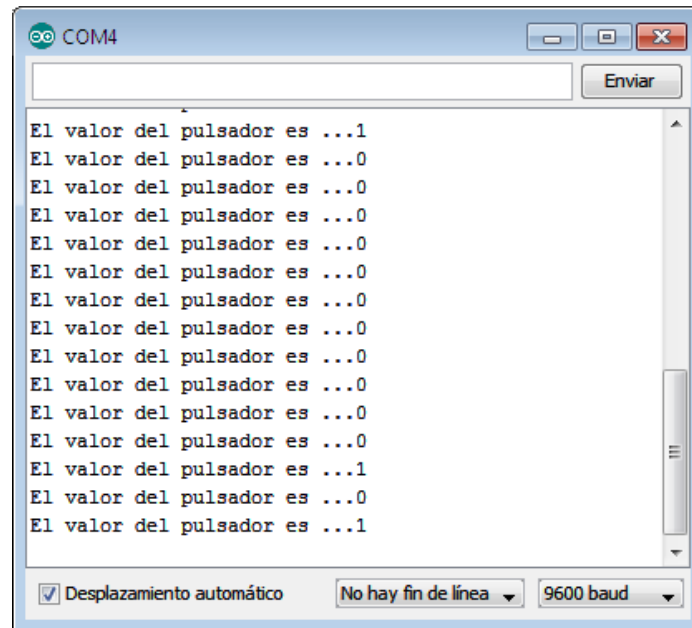
MIS PROYECTOS CON ARDUINO

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.

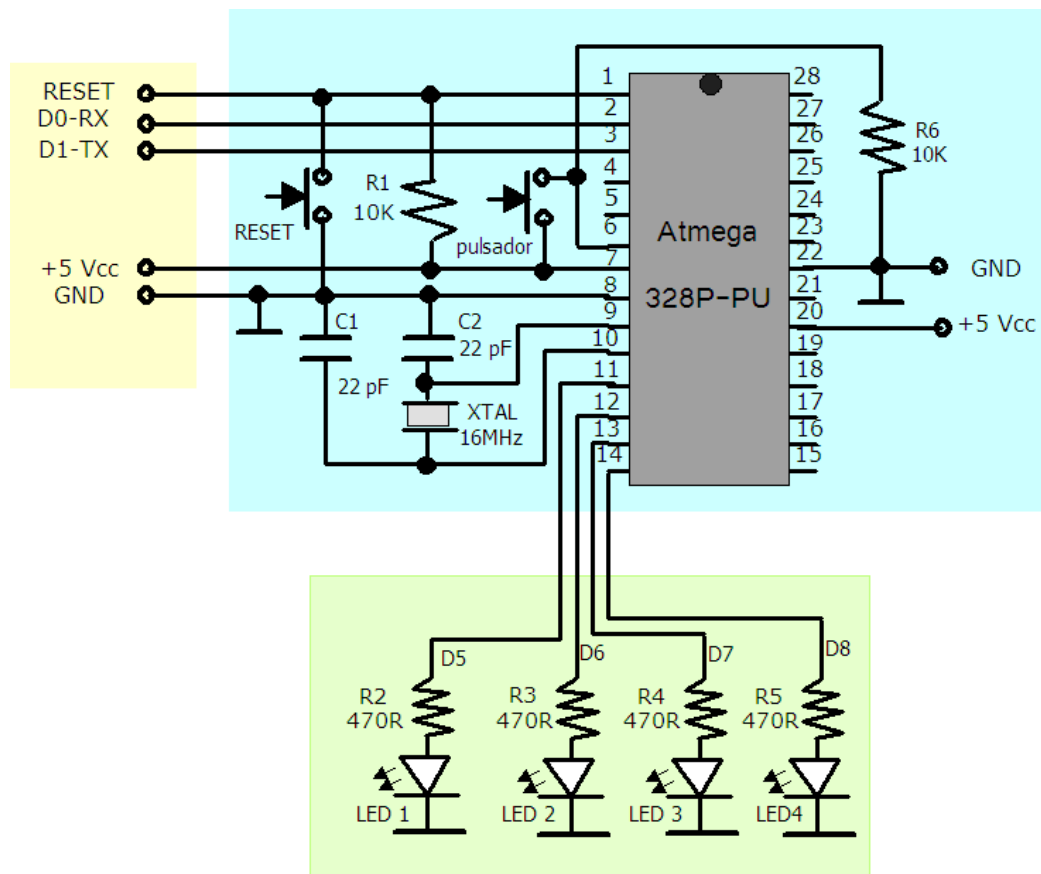
```
int leds[]={5,6,7,8}; // declaramos los pin a la variable leds
int tiempo=200;       // declaramos la variable tiempo igual a 200
int pulsador=4;       // declaramos pulsador en el pin 4
int n=0;              // declaramos la variable n
int valorpulsador=0;  // declaramos la variable valorpulsador

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);
  monitoriza();
  if (valorpulsador==1)
  {
    secuencia();
  }
}
```

MIS PROYECTOS CON ARDUINO



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



Luz de Led en función de la luz. (Versión 1)

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

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

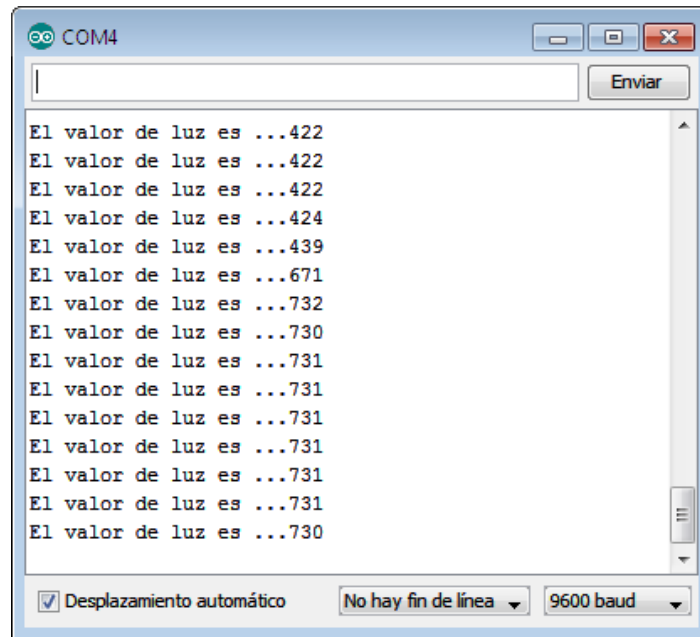
Para ello conectaremos una **LDR** a la entrada analógica AN0 y un Led al pin D9. Cuando la luz se encuentre entre 0 y 512 el Led debe colocarse en el nivel de potencia máxima (255), si la luz se encuentra entre valores 512 y 1024 el Led debe lucir al nivel de potencia 64. Además se deberá visionar el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

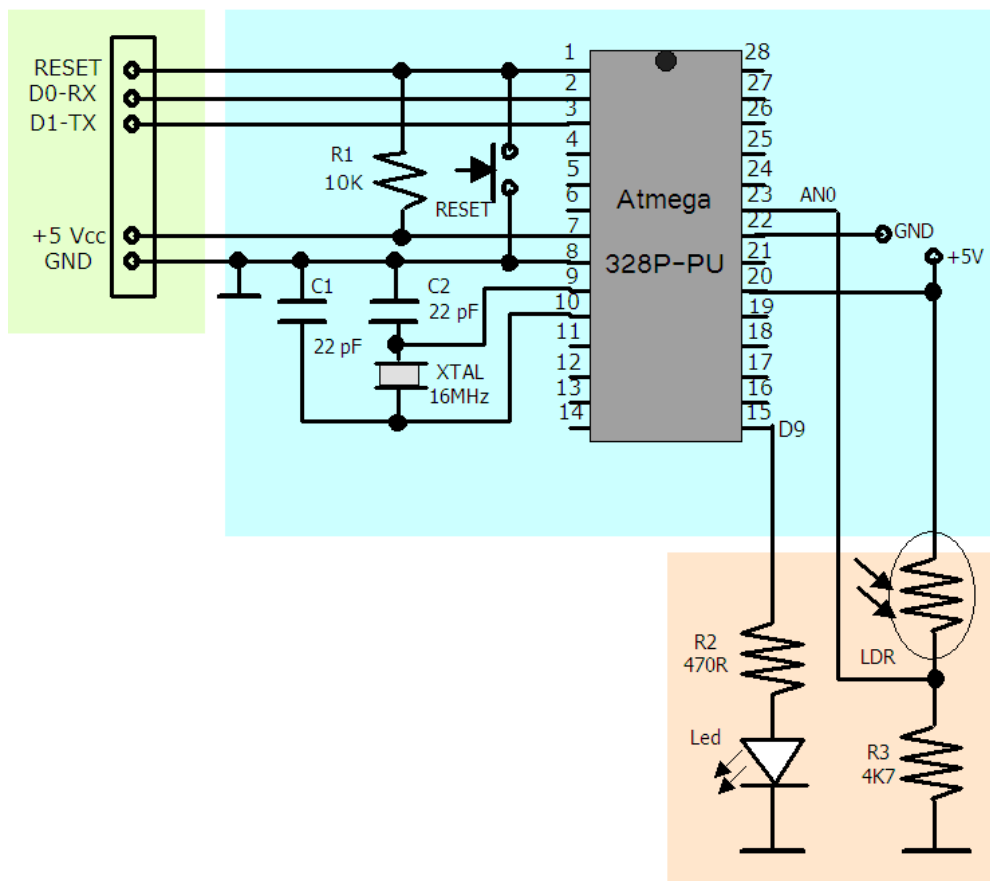
void setup()  //comienza la configuracion
{
  pinMode(9,OUTPUT);    // el pin D9 es de salida analógica PWM
  Serial.begin(9600);    // active la comunicacion serial
}
void monitoriza()    // bloque de instrucciones monitoriza
{
  Serial.print("El valor de luz es ..."); // texto indice en la consola
  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
  }
}
```


MIS PROYECTOS CON ARDUINO



Placa Arduino

Placa Protoboard



Luz de Leds en función de la luz (Versión 2)

Esta práctica 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 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
}

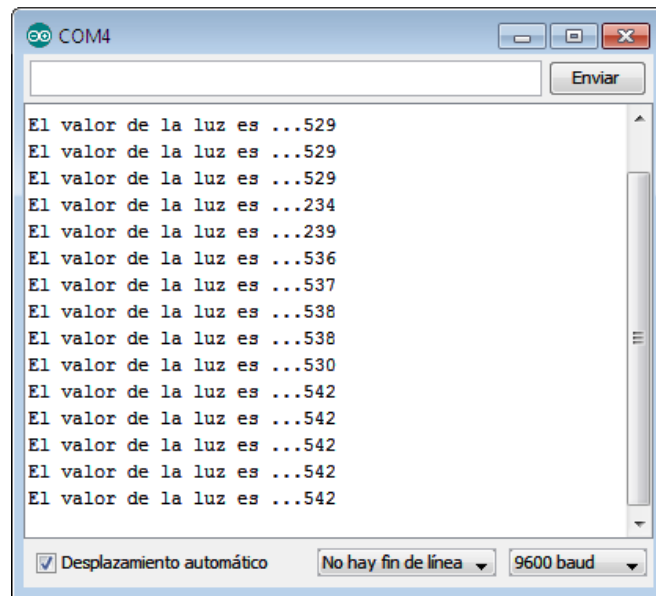
void monitoriza()        // bloque de instrucciones monitoriza
{
  Serial.print("El valor de la luz es ..."); // visualiza en índice
  Serial.println(luz);    // escribe el valor de luz
  delay(1000);            // pausa de 1 segundo
}

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++)
    {
      analogWrite(leds[n],127);
      delay(tiempo);
    }
  }
  if (luz<=511 && luz>=256)
  {

```

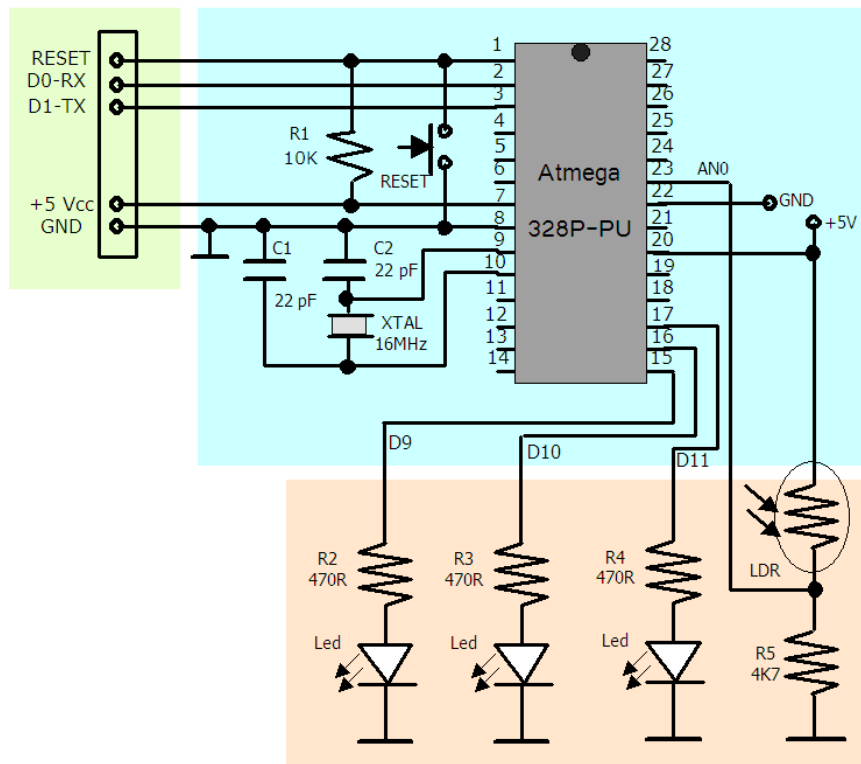
MIS PROYECTOS CON ARDUINO

```
for (n=0;n<=3;n++)
{
analogWrite(leds[n],191);
delay(tiempo); // pausa de 300 milisegundos
}
}
if (luz<=255 && luz>=0)
{
for (n=0;n<=3;n++)
{
analogWrite(leds[n],255);
delay(tiempo); // pausa de 300 milisegundos
}}
}
```



Placa Arduino

Placa Protoboard



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
}

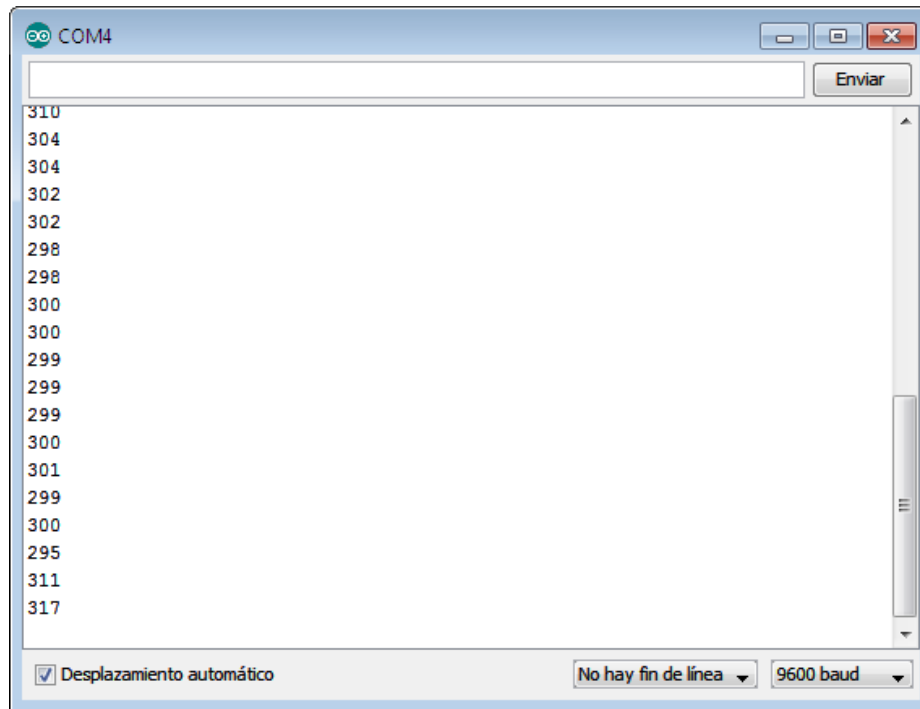
void loop()
{
  val=analogRead(ldr); // lee valor de LDR
  Serial.println(val); // visualiza por el monitor serial el valor de val
  delay(10000); // pausa para evitar la histéresis entre los valores de activacion
  if(val<=valor){ // si el valor de val es mayor o igual que el umbral se enciende
    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.

En las siguientes imágenes del Monitor Serial podemos observar los valores obtenidos mediante el monitor serie con una LDR y la diferencia de valores que se obtiene al variar la cantidad de luz y con la resistencia R6 de 4K7 para polarizar la LDR.

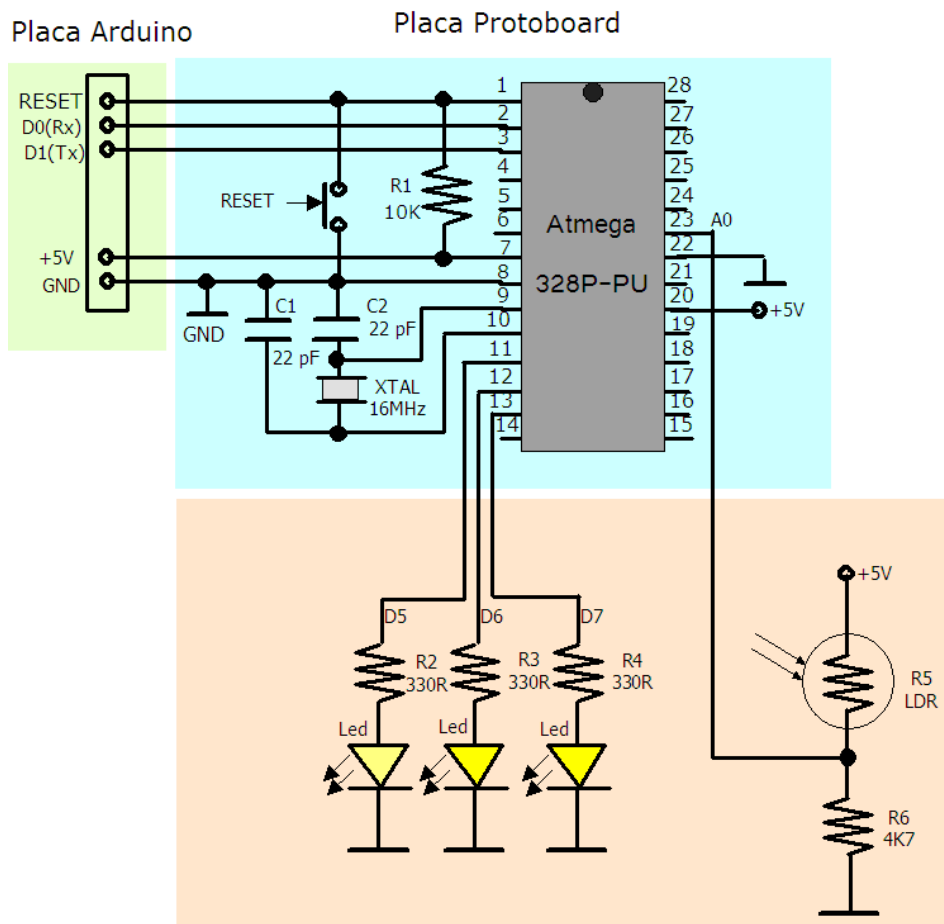
NOTA: Para obtener un ajuste más preciso del crepúsculo (oscuridad) se puede utilizar un potenciómetro o resistencia ajustable para obtener el punto de conexión de la LDR.

MIS PROYECTOS CON ARDUINO



Con resistencia de polarización de 4K7.

Con resistencia de 4K7 proporciona unas medidas de más de 300 y conforme se va oscureciendo disminuye a niveles de 275. Para ello, en la programación hemos puesto el valor de activación de las luces de jardín menor o igual a 300.



Encendido aleatorio de un Led bipolar

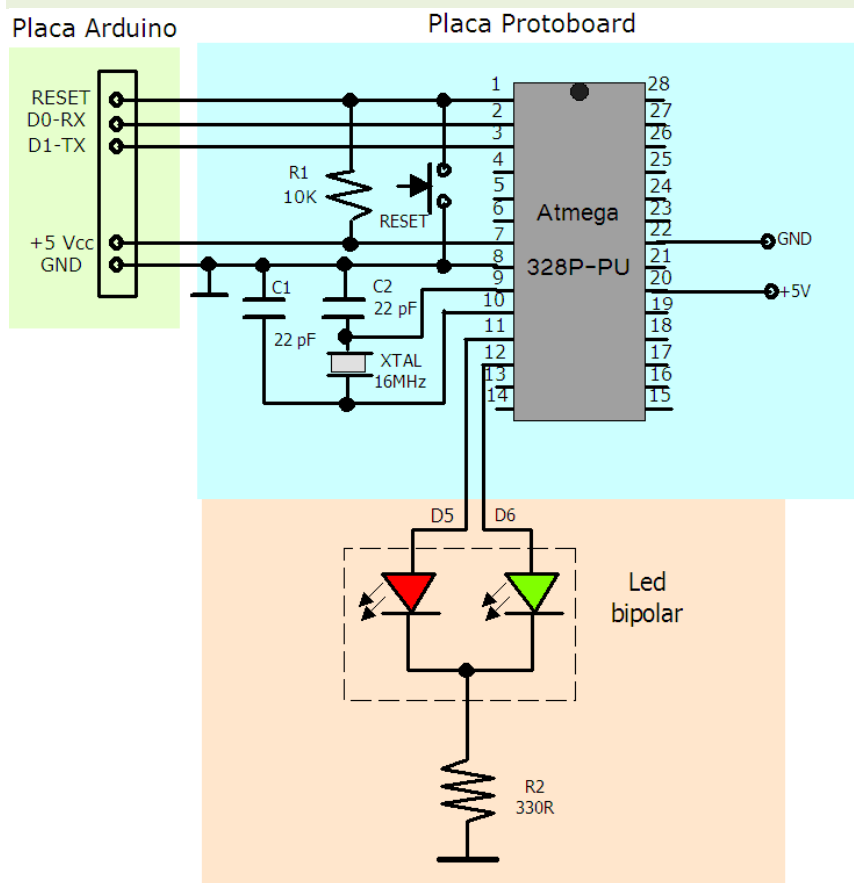
Esta práctica consiste en aplicar la función aleatoria `RandomSeed()` y `Random()`, que nos aportarán un poco de aleatoriedad a nuestra práctica para trabajar con estas instrucciones.

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
}
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
}
```



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.

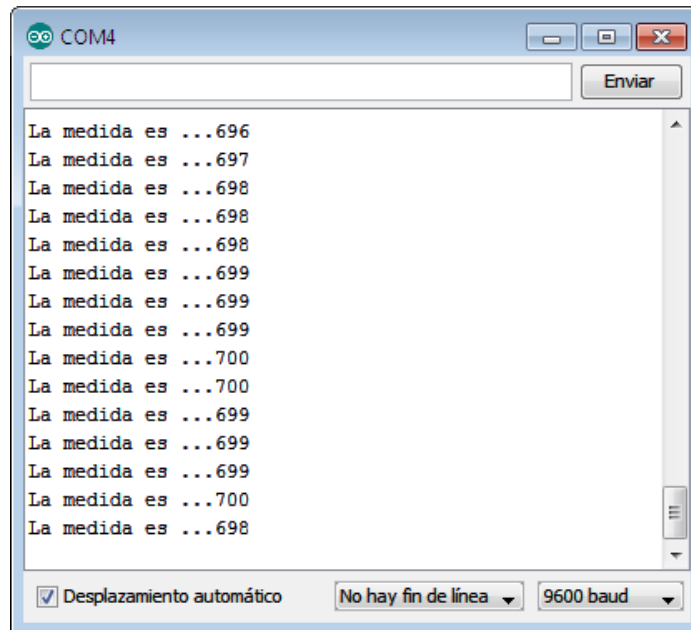
La resistencia **NTC** (Coeficiente de Temperatura Negativo) es una resistencia dependiente de la temperatura, conforme detecta el aumento de la temperatura su resistencia interna comienza a disminuir, por lo tanto, lo que la caracteriza es que a mayor temperatura menor resistencia.

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;
int ntc=0;
int motor=10;
int medida=0;
int nivel=700; //variable que guarda el límite de temperatura al que se activa el
ventilador
void setup()
{
  pinMode(led,OUTPUT);
  pinMode(motor,OUTPUT);
  Serial.begin(9600);
}
void monitoriza() //procedimiento que envía al puerto serie, para ser leído en el
//monitor,
{
  Serial.print("La medida es ...");
  Serial.println(medida);
  delay(1000);          //para evitar saturar el puerto serie
}
void loop()
{
  medida=analogRead(ntc);
  monitoriza();
  if(medida>nivel)
  {
    digitalWrite(led,HIGH);    //si la señal del sensor supera el nivel marcado:
    digitalWrite(motor,HIGH); //se enciende un aviso luminoso
    //arranca el motor
  }
  else
    // si la señal está por debajo del nivel marcado
  {
    digitalWrite(led,LOW);
    digitalWrite(motor,LOW);  // el motor se para
  }
}
```

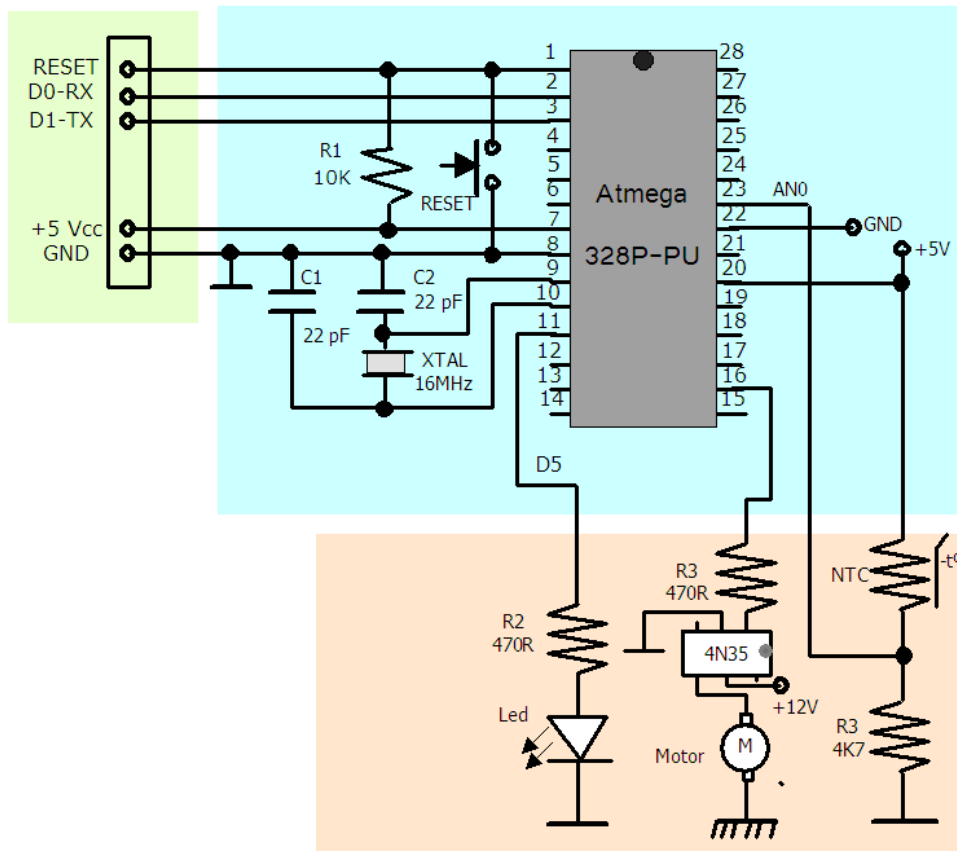
MIS PROYECTOS CON ARDUINO

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.



Placa Arduino

Placa Protoboard



Termostato con velocidad de 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.

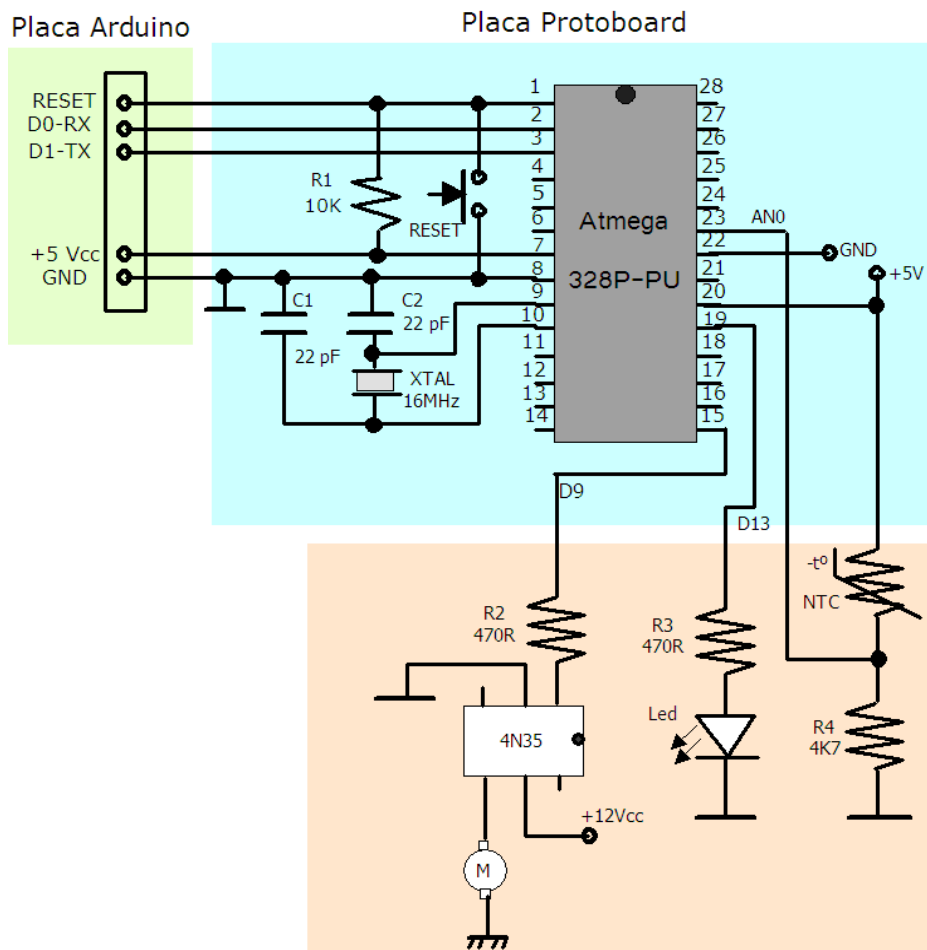
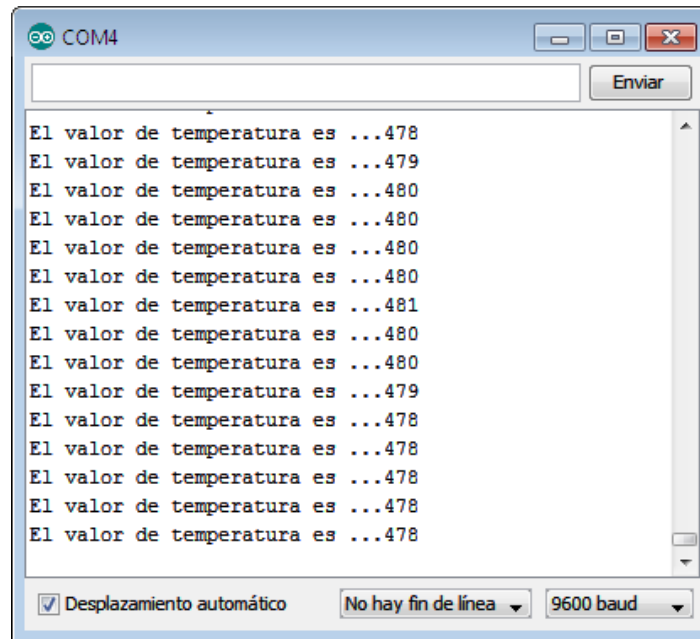
```
int motor=9;
int led=13;
int ntc=0;
int temperatura=0;
void setup()
{
  pinMode(led,OUTPUT);
  pinMode(motor,OUTPUT);
  Serial.begin(9600);
}

void monitoriza()
{
  Serial.print("El valor de temperatura es ...");
  Serial.println(temperatura);
  delay(1000);
}

void loop(){
  temperatura=analogRead(ntc);
  monitoriza();
  if(temperatura>478)
  {
    digitalWrite(led,HIGH);
    analogWrite(motor,200);
  }
  else
  {
    digitalWrite(led,LOW);
    digitalWrite(motor,LOW);
  }
}
```

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.

MIS PROYECTOS CON ARDUINO



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.

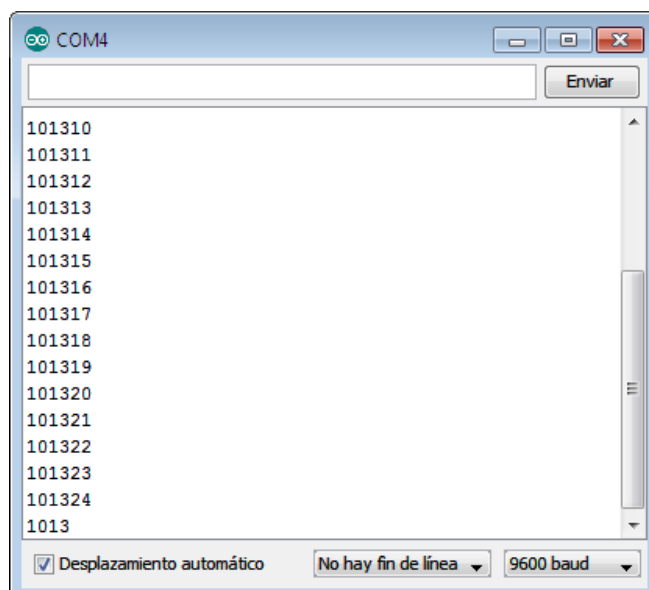
MIS PROYECTOS CON ARDUINO

Programa 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.

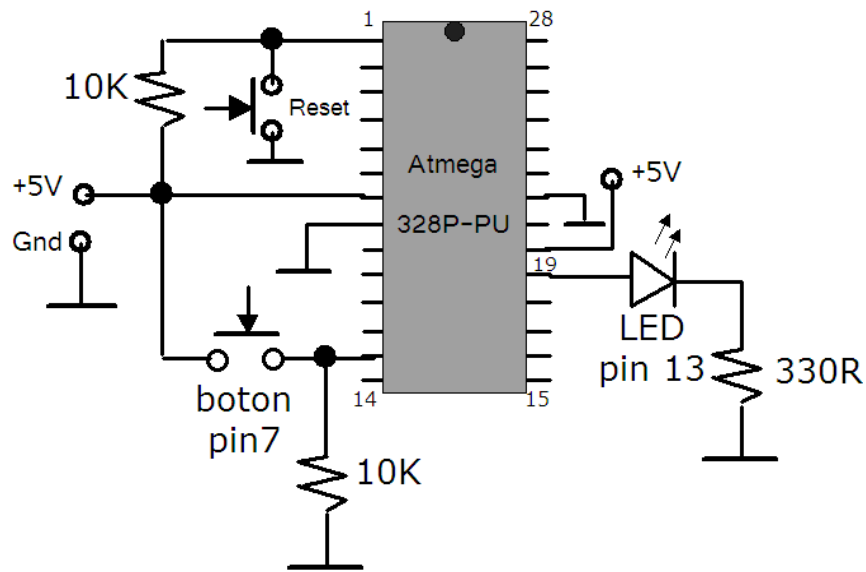
```
// Programa Contador

int LED=13;
int boton=7;
int valor=0;
int contador=0;
int estadoboton=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)            // condicocna y compara con el valor 1
    {
      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;
}
```



MIS PROYECTOS CON ARDUINO

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.



Programa Contador de 0 a 10

La siguiente programación detecta si el botón conectado a la entrada pin 7 ha sido presionado y enciende el LED. Envía al PC el valor de la variable de cuenta "Contador" vía puerto serie.

```
// Programa Contador de 0 a 10

int LED=13;
int boton=7;
int valor=0;
int contador=0;
int estadoboton=0;

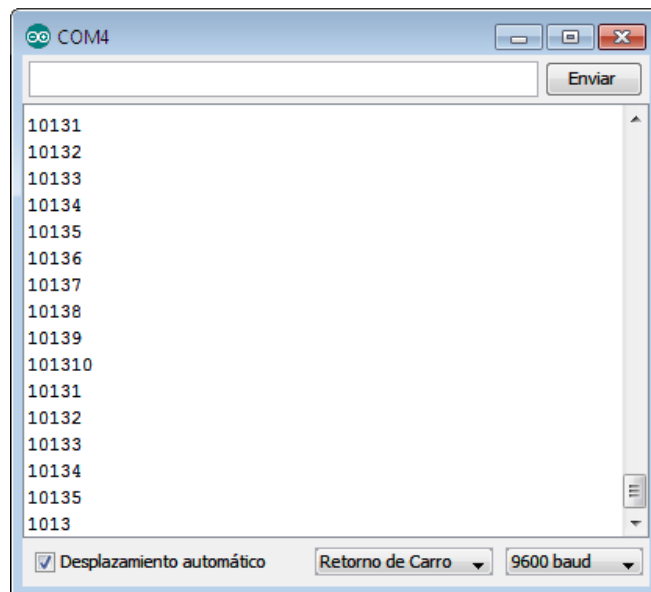
void setup()
{
  Serial.begin(9600);      // configure velocidad de transmisión a 9600 b
  pinMode(LED, OUTPUT);    // pone de salida digital el LED verde el pin 13
  pinMode(boton, INPUT);   // pone de entrada digital el botón pin 7
}

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

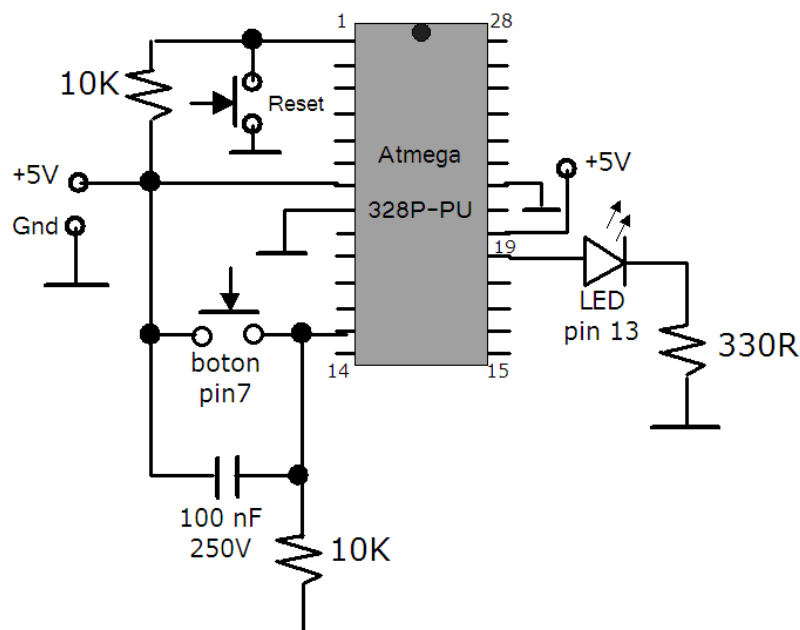
  if(valor!=estadoboton)    // condiciona igualdad estado variable valor
  {
    if(valor==1)            // condiciona y compara con el valor 1
    {
      contador++;           // suma 1 al contador
      Serial.println(contador); // lanza el valor de contador al pulsar boton
      Serial.print(10);     // lanza al monitor serial 10
    }
  }
}
```

MIS PROYECTOS CON ARDUINO

```
Serial.print(13);           // lanza al monitor serial 13
if(contador==10)           // limita la cuenta el valor a 10
{
  contador=0;              // pone el contador a cero
}
}
}
estadoboton=valor;
}
```



Tal como se observa en el *Monitor Serial* cada vez que pulsamos el botón de conteo se va incrementando de uno en uno el contador hasta el 10, volviendo a empezar desde cero.



NOTA: Para evitar rebotes en la pulsación del botón del contador se añade un condensador de 100 nF en paralelo con el pulsador, esto evitará que al pulsar una sola vez se produzca una doble pulsación.

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.

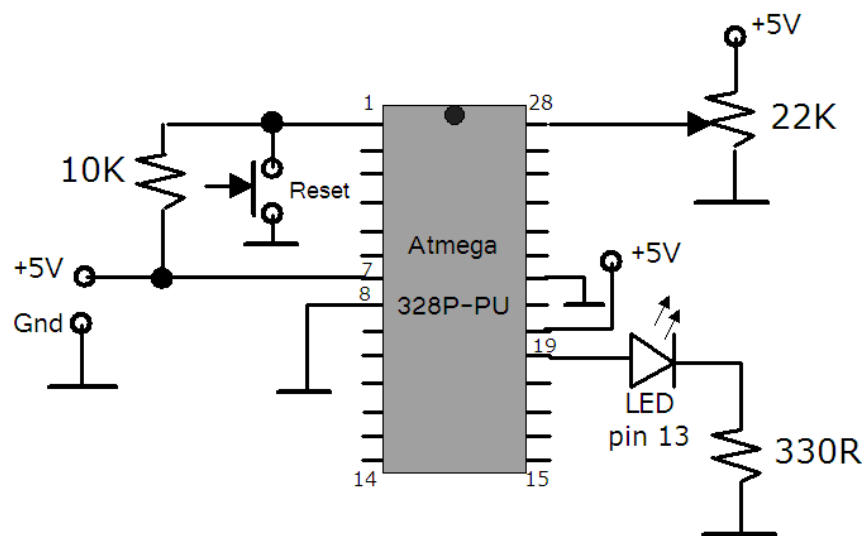
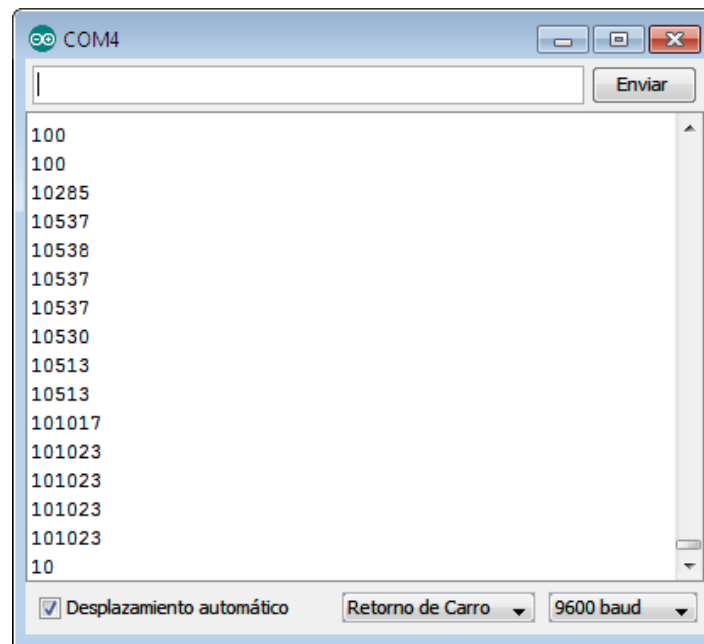
```
// Entrada Analógica

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;
void setup()
{
  pinMode(LED, OUTPUT);
  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)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
  }
}
```

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 pasado de 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.

MIS PROYECTOS CON ARDUINO



11. PROYECTO: CRUCE REGULADO POR SEMÁFOROS

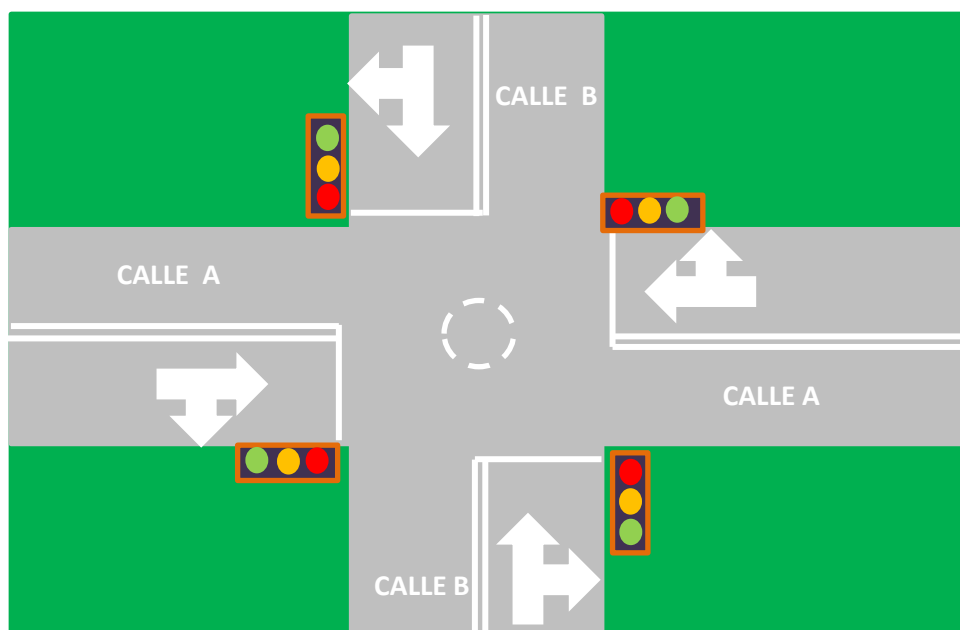
Todo proyecto conlleva unas series de pasos antes y durante su desarrollo. Según las dimensiones del proyecto se necesitarán más recursos o menos. Es decir, en el caso de este proyecto que se plantea el diseño de la regulación de un cruce con semáforos, los recursos son pocos, en el caso de una ampliación 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.

11.1. Datos para la programación

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.

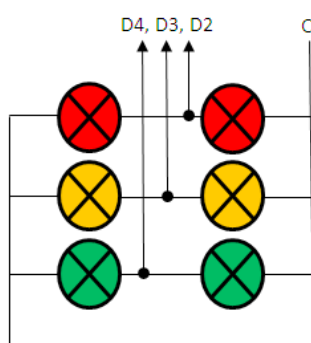
11.2. Códigos de programación:

```
/*
  Semaforo
  Secuencia de 6 led encendiendose por medio de una temporización delay diferentes
  y conmutadas
  */
void setup()
{
  // inicializa el programa para pin de salida.
  // declaracion de los pin de salida digital son:
  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
  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
}
```

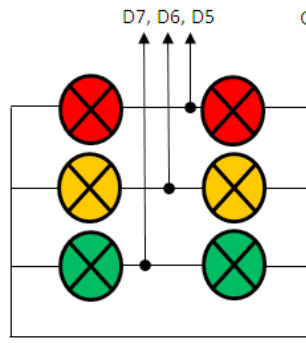
11.3. Descripción y funcionamiento del circuito

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.

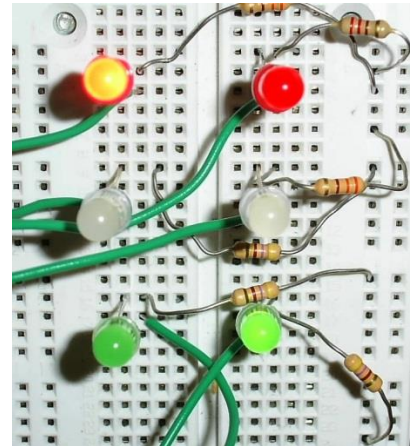
En la siguiente página se muestra el plano eléctrico completo de este proyecto. La circuitería electrónica para el control de dos semáforos A y B, conmutados, para completarlo se necesitan la instalación de dos semáforos A puesto en paralelo y otros dos B también en paralelo.



A



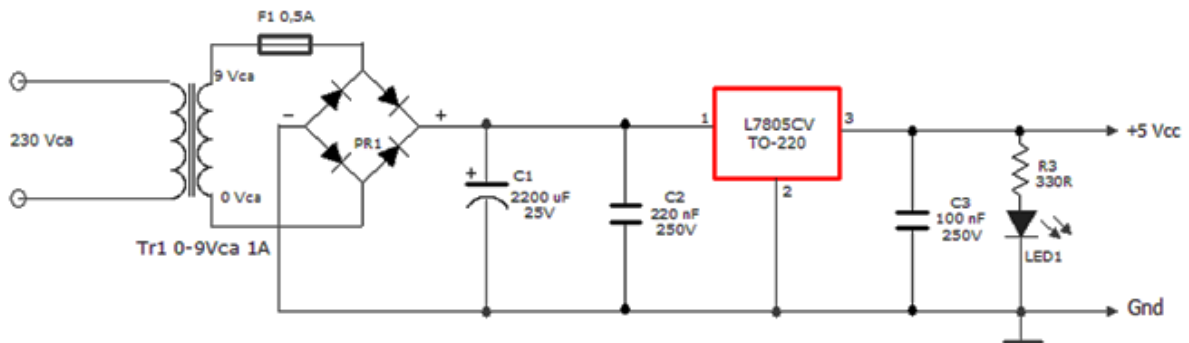
B



A

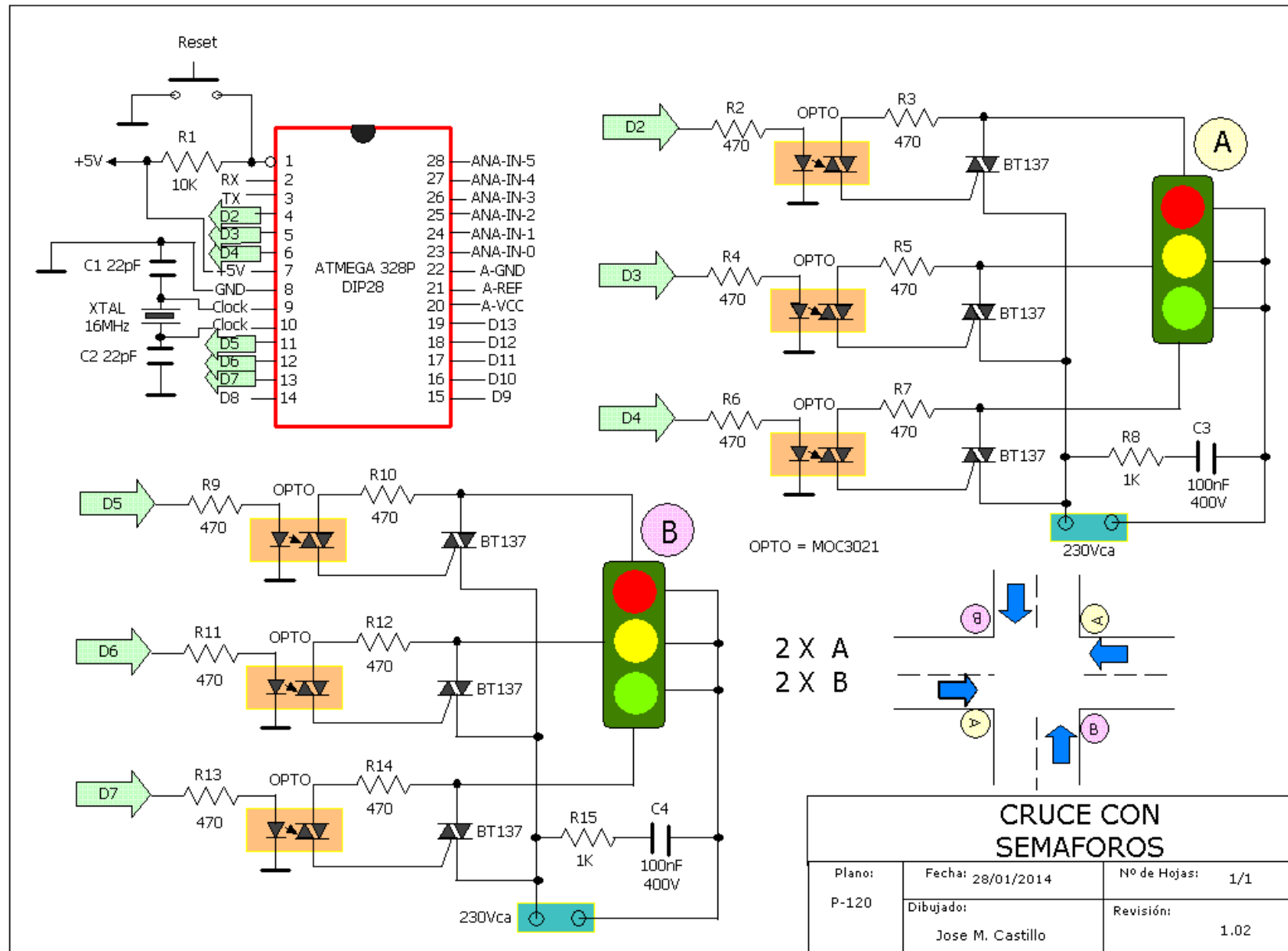
B

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).



NOTA: Hay que tener mucha precaución con las tensiones de que alimentan las lámparas (230Vca) y respetar el aislamiento con las tensiones y señales de salida del microcontrolador y por medio de un optoacoplador.

MIS PROYECTOS CON ARDUINO



12. PROYECTO: SEMÁFORO PARA PASO DE PEATONES

Este proyecto contempla dos semáforos para cada una de las direcciones de una vía de doble sentido. Los 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.

12.1. Datos para la programación

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.

12.2. Códigos de programación:

```
/* Este programa consiste en controlar un semaforo y 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
}
```

MIS PROYECTOS CON ARDUINO

```
void loop()
{
    estadopulsador=digitalRead(pulsador); // Ponemos la variable al estado
    //obtenido en el pin pulsador

    if (estadopulsador == HIGH) // condicionante si esta en alto se hace //lo
    siguiente

    {
        delay(100); // ponemos un retardo de 0,1 segundo
        digitalWrite(led1,LOW); // apagamos el led1 naranja
        delay(100); // dejamos 100 ms de tiempo
        digitalWrite(led2,HIGH); // encendemos el led2 naranja semaforo
        delay(2000); // retardo de 0,2 segundos
        digitalWrite(led2,LOW); // apagamos el led2 naranja semaforo
        digitalWrite(led3,HIGH); // encendemos el led3 rojo semaforo
        delay(1000); // ponemos un retardo de 0,1 segundo
        digitalWrite(led5, LOW); // apagamos el led 5 rojo del peaton
        delay(500); // retardo de 0,5 segundos
        digitalWrite(led4,HIGH); // encendemos el led 4 verde del peaton
        delay(20000); // tiempo de paso de 1 minuto
        digitalWrite(led4,LOW); // se apaga el led 4 verde del peatos
        delay(500); // retardo
        digitalWrite(led4,HIGH); // enciende led 4 verde peaton
        delay(500); // retardo
        digitalWrite(led4,LOW); // apaga led 4 verde peaton
        delay(500); // retardo
        digitalWrite(led4,HIGH);
        delay(500); // retardo
        digitalWrite(led4,LOW);
        delay(500); // retardo
        digitalWrite(led4,HIGH);
        delay(500); // retardo
        digitalWrite(led4,LOW);
        delay(500); // retardo
        digitalWrite(led5, HIGH);
        delay(500); // retardo

    } else { // de lo contrario hace esto otro

        digitalWrite(led3, LOW); // apaga el led3 rojo vehiculo
        digitalWrite(led4, LOW); // apaga el led4 verde peaton
        digitalWrite(led5,HIGH); // enciende el led5 rojo peaton
        delay(100); // establece un tiempo
        digitalWrite(led1, LOW); // apaga el led1 naranja
        digitalWrite(led2, HIGH); // enciende led 2 naranja
        delay (400); // deja un retardo de 0,4 segundos
        digitalWrite(led2, LOW); // apaga el led 2 naranja
        delay(100); // deja un retardo de 0,1 segundos
        digitalWrite(led1,HIGH); // enciende el led 1 naranja
        delay(400); //deja un retardo de 0,4 segundos

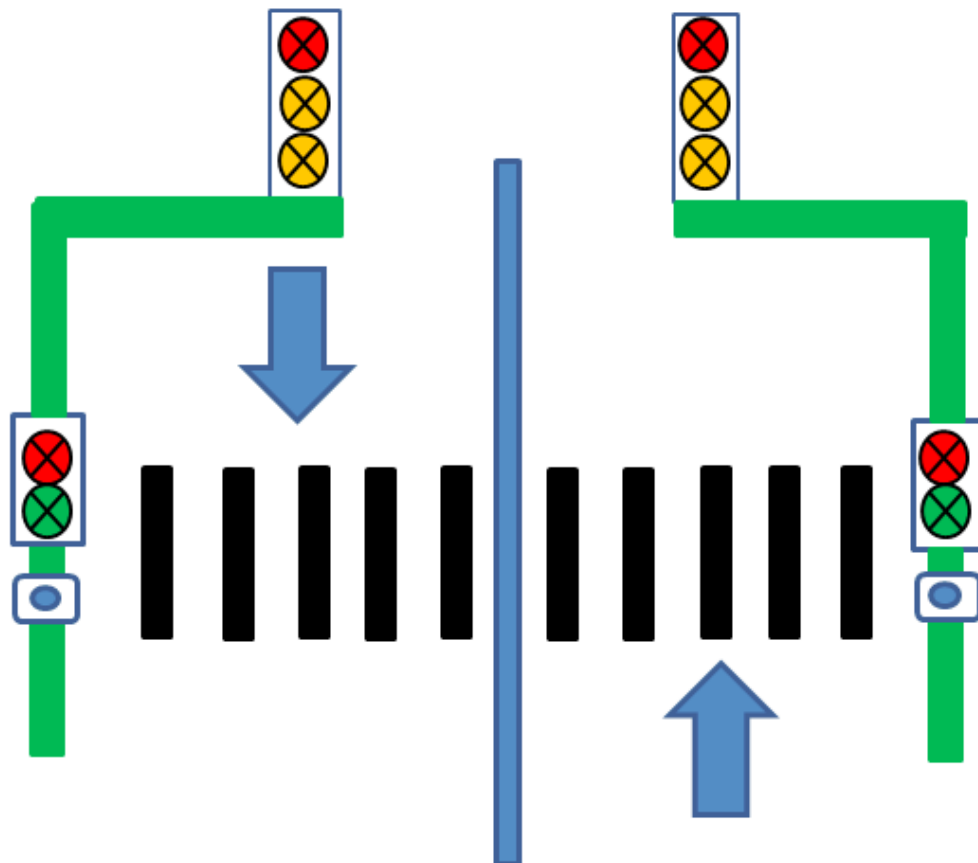
    }
}
```

12.3. Descripción y funcionamiento del circuito

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.

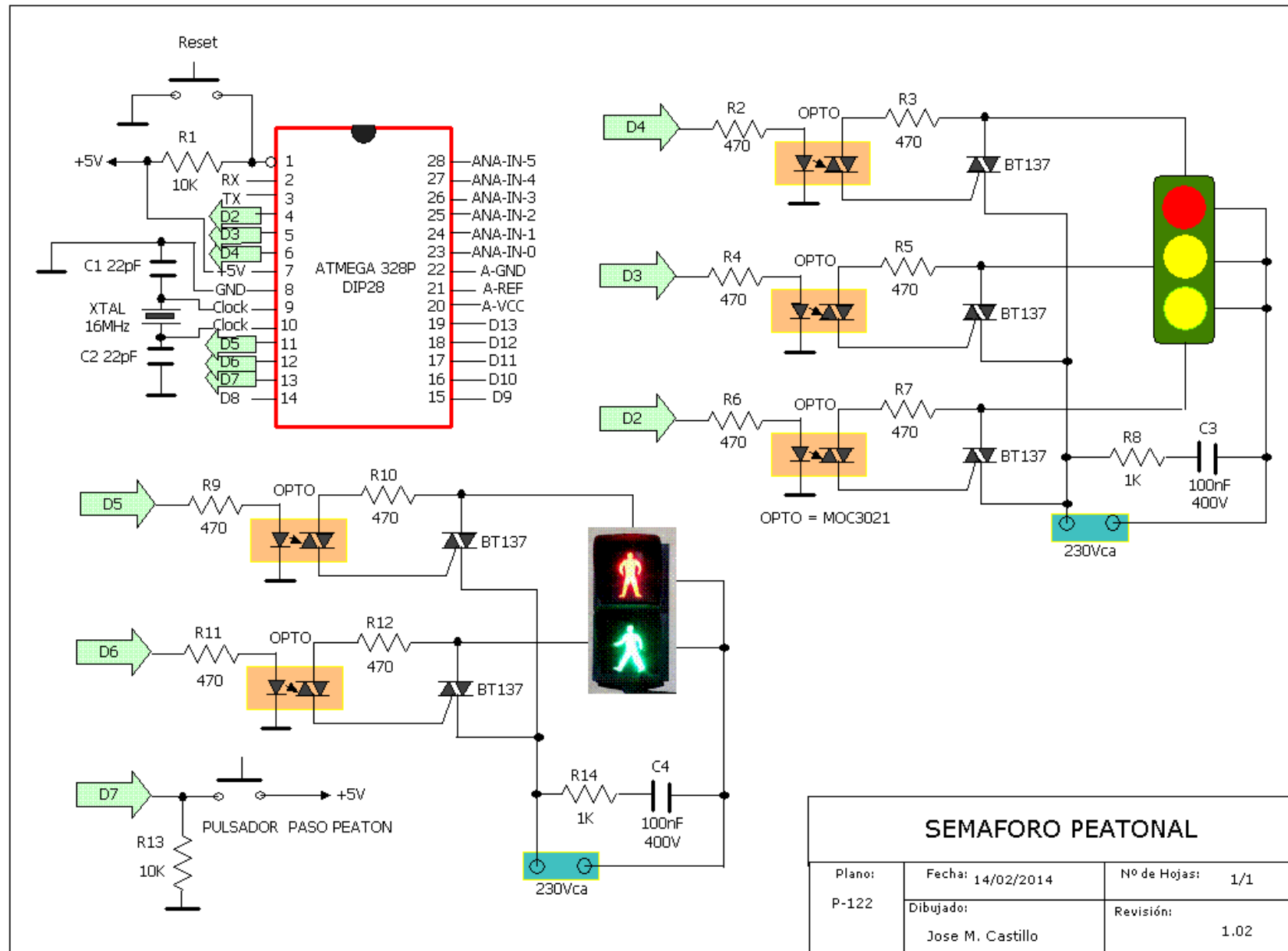
El Atmega328P de 28DIP se le conecta para la señal de reloj, en los pines 9 y 10, un cristal de cuarzo de 16 MHz para sincronizar la señal de reloj y el funcionamiento de todos los elementos conectados al microcontrolador. Los pines de salida de D02 a D06 se aplican una resistencia de 470 Ohmios en serie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

MIS PROYECTOS CON ARDUINO

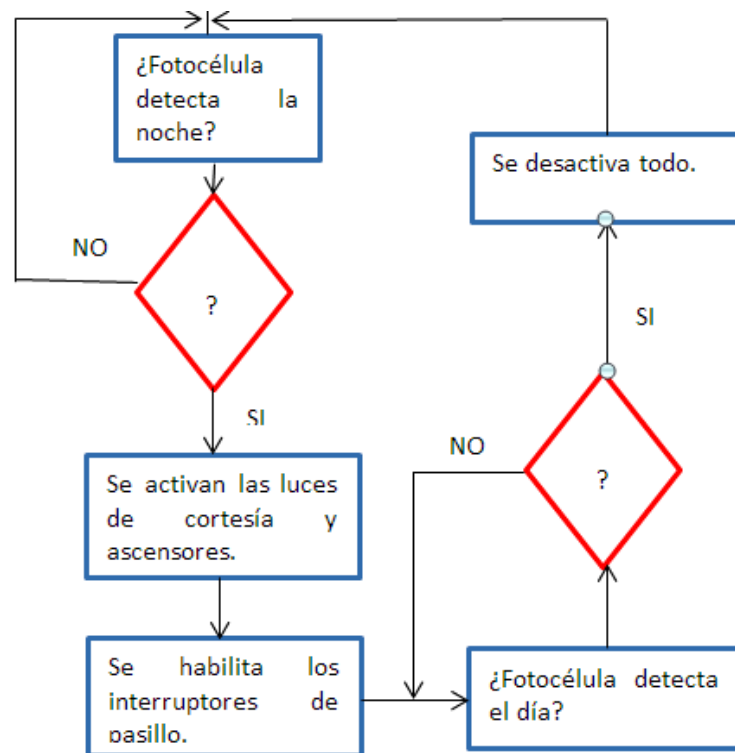


13. PROYECTO: CONTROL DE LA LUZ NOCTURNA DE ENTRADA A UN EDIFICIO

Este proyecto consiste en controlar el encendido automático de la iluminación de la entrada y pasillos de un edificio de oficinas cuando se hace de noche y su desconexión cuando se hace de día.

El sistema contempla el empleo de una célula fotoeléctrica LDR que detectará cuando es de noche. Mediante el ajuste de una resistencia variable se podrá elegir la cantidad de luz. Esto permitirá establecer, cuando es de noche, el encendido permanente de una o dos lámparas de bajo consumo de cortesía en la entrada del edificio y ascensores y con ello habilitar un pulsador (o más de uno en paralelo o en su caso un detector de movimiento del tipo volumétrico) para encender las luces del pasillo y zonas comunes durante un tiempo programado (en este caso de 1 minuto y medio), pasado este tiempo se apagarán, y estará de nuevo en disposición de recibir una nueva pulsación.

En el momento que la célula fotoeléctrica LDR detecte la luz de día todo el sistema se deshabilitará por completo (desconexión), no funcionando los pulsadores del pasillo y apagándose las lámparas de entrada de cortesía.



13.1. Datos para la programación

1. Declaramos de entrada el uso de un potenciómetro para polarizar la fotocélula LDR donde se ajusta para obtener la activación de la secuencia de programación.
2. Declaramos de entrada el uso de un pulsador para encender las luces del pasillo.
3. Declaramos de salida las luces de cortesía que se quedan encendidas permanentemente.
4. Declaramos de salida las luces del pasillo que se activan mediante el pulsador

Por lo tanto existen principalmente dos códigos que lo declaramos de entrada y otros dos de salida. Lo primero que se debe cumplir es que la fotocélula detecte la noche y polarice el pin de entrada para que éste pueda establecer el flujo de la programación: enciende luces de cortesía y active los pulsadores de pasillo y luces del pasillo.

13.2. Códigos de programación

```
/*
  Elementos necesarios:
  - 2 Leds de 5mm
  - 2 Resistencia de 1/4W 680 Ohmios
  - 1 Resistencia de 1/4 W 10K Ohmios
  - 1 Resistencia de 1/4W 4K7 Ohmios
  - 1 Resistencia ajustable 4K7 Ohmios
  - 1 Celula fotoeléctrica LDR
  - 1 Micropulsador
*/
int led1 = 2; // Declaramos pin de salida para encender la luz de entrada //de
cortesía y ascensores
int led2 = 3; // Declaramos pin de salida para encender las luces del //pasillo
cuando actuamos en el pulsador
int boton = 11; // Declaramos pin de entrada para el pulsador de encendido //de las
luces del pasillo
int ldr = 10; // Declaramos pin de entrada para el sensor fotoelectrico LDR
int estadoboton = 0; // Ponemos a cero la variable
int sensor = 0; // Ponemos a cero la variable

void setup()
{
  pinMode (led1, OUTPUT); // Configuramos el led1 de salida
  pinMode (led2, OUTPUT); // Configuramos el led2 de salida
  pinMode (boton, INPUT); // Configuramos la variable de entrada
  pinMode (ldr, INPUT); // Configuramos la variable de entrada
}
void loop()
{
  sensor=digitalRead (ldr); // Inicializamos la variable al valor tomado por //ldr
  digitalWrite(led1, sensor); // Pone a la salida del pin led1 la lectura //obtenida
  por la variable sensor
  if(sensor == HIGH) // Condicionante comparativa si el valor de la //variable
  esta en alto o no
  {
    estadoboton=digitalRead(boton); // Si esta en alto lee el valor del pin //boton
    {
      digitalWrite(led2, estadoboton); // Pone el pin led2 segun el valor de
      //estadoboton
    }
    if (estadoboton == HIGH) // Si el estadoboton esta en alto
    {
      delay(70000); // Pone una pausa de un minuto y veinte segundos //encendido el
      led2
    }
    else // de lo contrario apaga el led2
    {
      digitalWrite(led2, LOW); // Pone a nivel bajo el led2
    }
  }
}
```

13.3. Descripción y funcionamiento del circuito

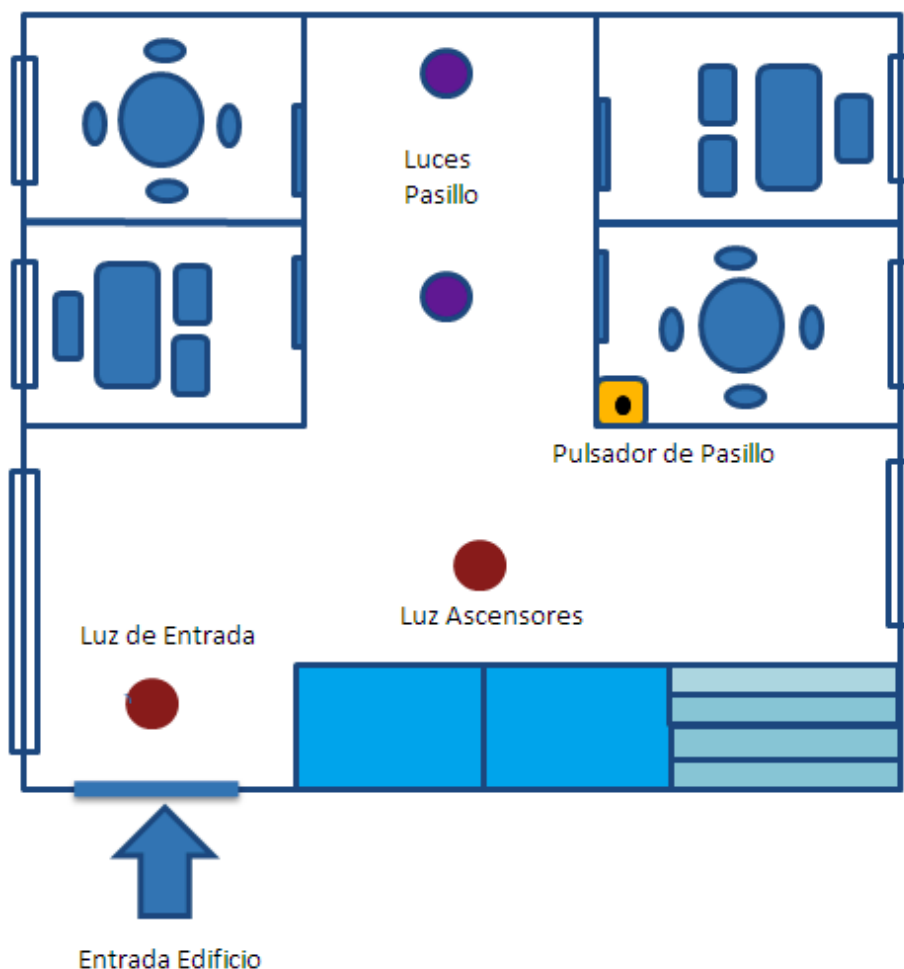
El circuito se compone principalmente del microcontrolador Arduino ATmega328P, programado para dos pines de entrada D10 y D11 y dos pines de salida D2 y D3.

La entrada del pin D10 corresponde a la fotocélula LDR conectada en serie con una resistencia ajustable de 4K7 Ohmios y otra de 1K Ohmio a +5V. Con la resistencia variable de 4K7 ajustaremos la LDR para activarse en el punto crepuscular o anocheciendo que nos convenga. La instalación de esta fotocélula LDR debe de estar totalmente independiente y fuera de otros focos de luces, especialmente en un lugar cuya iluminación sea natural.

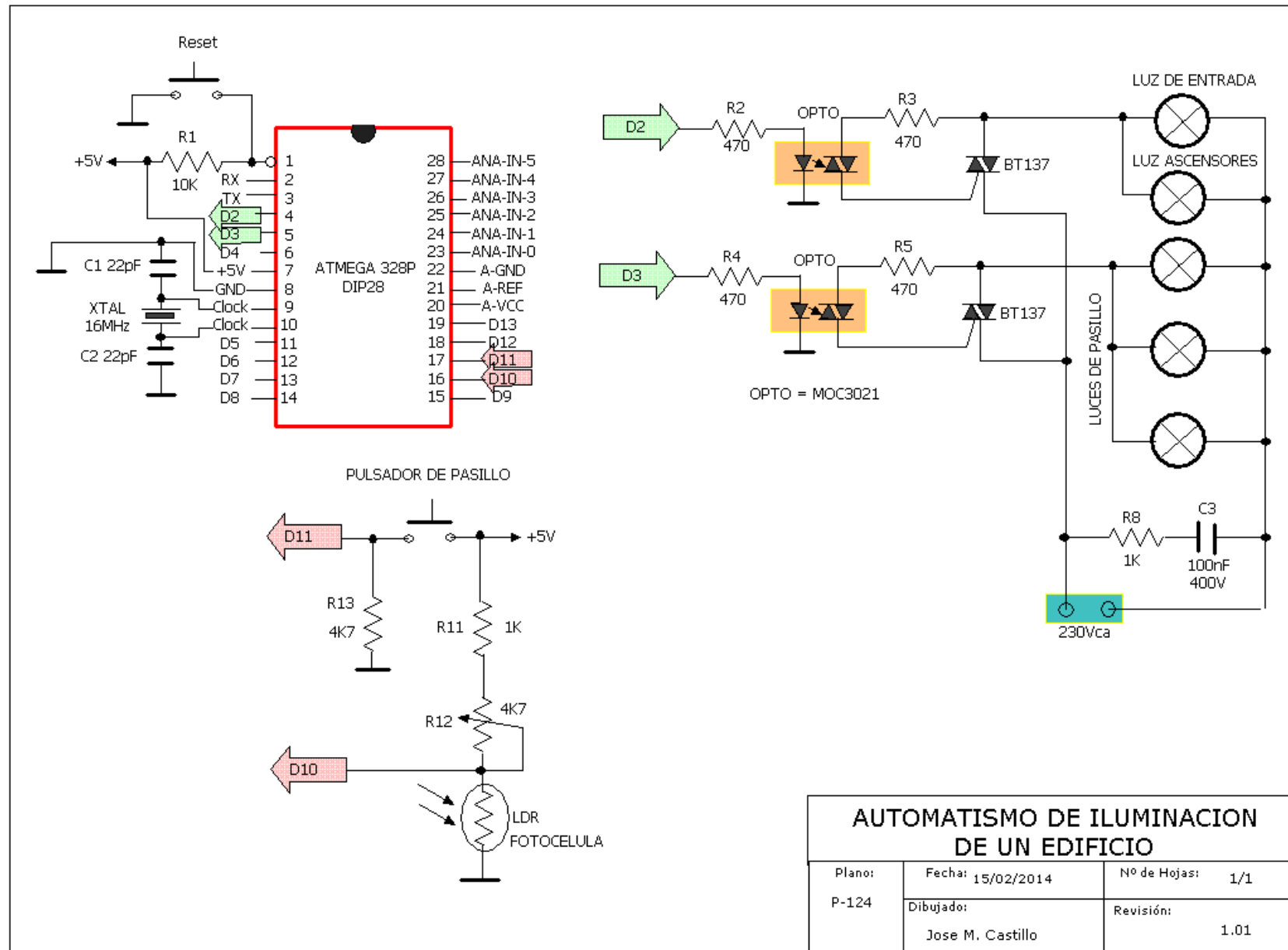
La otra entrada del pin D11 corresponde a la entrada de pulsadores polarizado con +5V y una resistencia de polarización pull-up de 4K7. Para disponer de más pulsadores se podrán instalar en configuración paralelo y con un condensador de 10 nF.

La salida del pin D2 cuando está activo, enciende las luces de entrada y ascensores, ésta polariza con 5V la entrada al Optoacoplador y por medio de este hace conducir el Diac interno para polarizar la puerta del Triac y establecer el encendido de las lámparas (lámparas de cortesía y ascensor) a 230Vca.

La salida del Pin D3, se activa cuando pulsamos el pulsador de pasillos. Tiene la misma configuración que el anterior circuito y corresponde al encendido de las luces del pasillo que se activa cuando se pulsa el pulsador. Por medio del optoacoplador MOC3021 hace circular la corriente del Diac y éste polariza la puerta del Triac haciéndola conducir.



MIS PROYECTOS CON ARDUINO



14. PROYECTO: CENTRALITA DE ALARMA DE DOS ZONAS NC

Este proyecto consiste en el diseño de una centralita de alarma que disponga de dos zonas NC (Normalmente Cerradas) y la conexión retardada. En el momento que se produzca una intrusión nos avise tanto acústicamente como ópticamente y la zona alarmada.

Las centralitas de alarmas se encargan de gestionar los diversos dispositivos de entrada como son los detectores, sensores, etc., y los dispositivos de salida para avisarnos que se ha producido una alarma por medio de señalización acústica y óptica. Los dispositivos de entradas deben de estar todos conectados en serie haciendo un lazo cerrado. Al detectar uno de ellos alarma, el lazo se abre y le indica a la centralita que se ha producido una alarma, ésta la recibe, la señaliza y hace conectar los dispositivos de salida que nos avisan óptica y acústicamente.

Gracias a los microcontroladores se pueden minimizar los recursos hardware que en el caso de componentes convencionales necesitaríamos de mayor cantidad: resistencias, transistores, condensadores, diodos, etc.

Este proyecto es idóneo para trabajar con microcontroladores por la necesidad de elementos de entrada y salidas, temporizaciones, etc.

La temporización, es una de las necesidades que se suelen utilizar con mucha frecuencia en electrónica, principalmente formados por componentes RC, resistencias y condensadores, que nos permite obtener un retardo mediante la carga y descarga con una constante de tiempo.

Existen dos modos de programar una temporización: temporización a la conexión y temporización a la desconexión. El primero, el circuito, se encuentra desconectado y, cuando se pulsa, al cabo de un determinado tiempo se conecta. El segundo se encuentra conectado y, cuando se pulsa, al cabo de un tiempo se desconecta. En nuestro proyecto utilizamos un retardo a la conexión, cuando conectamos nuestra centralita tenemos un margen de tiempo para salir del recinto. En la Zona 1 utilizamos un retardo a la desconexión, cuando entramos en el recinto y nos detecta tenemos un margen de tiempo para desconectarla.

En un microcontrolador la programación hace la tarea de temporizar, ejemplo, utilizando un contador por bloque, con un bucle creado en **for**, retardo **delay**, un **do... while**, etc.

La señalización y visualización óptica mediante Leds es esencial en estos equipos, pues debemos ver en cualquier momento y las 24 horas, como está el estado de nuestras zonas de alarma, esté conectada como si no está conectada la centralita, si hubiera alguna zona que estuviera alarmada no se podría conectar la centralita hasta que desaparezca la alarma, posiblemente por una ventana o una puerta abierta, un detector que se encuentra en mal estado, cable cortado, etc.

Existen dos zonas: zona 1 (retardada para E/S) y zona 2 (instantánea), normalmente cerrada, donde se podrá distribuir los dispositivos de detección, por ejemplo, en la zona 1 colocar todos los contactos magnéticos correspondiente a las puertas de entrada y detectores volumétricos de entrada a la vivienda. En la zona 2, detectores volumétricos de cocina, magnéticos de garaje y ventanas. Con dos zonas lo hacen más eficaz en el momento de localizar por qué parte de la vivienda o negocio se ha detectado la intrusión.

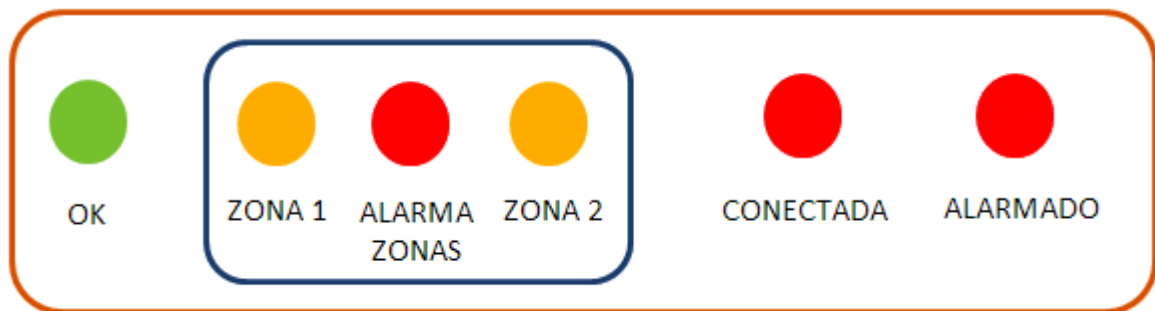
Los dispositivos que utilicemos para la detección tanto magnética como electrónica tendrán que ser sus contactos, normalmente cerrados y, abriéndose, cuando se detecta la alarma.

14.1. Datos para la programación

Este proyecto contienen los siguientes datos:

- a) 2 zonas de entrada de alarma NC: **zona1**(Retardada) y **zona2** (Instantánea),
- b) Cada zona de alarma tiene asignado un Led naranja de señalización: **led04** y **led05**
- c) 2 pulsadores de entrada para la conexión y desconexión: **pulse** y **reset**
- d) Un zumbador para señalar acústicamente la salida y entrada: **zum**
- e) Un Led verde para indicar que todo está correcto y se puede conectar la centralita: **led03**
- f) Un Led rojo para indicar que alguna zona está abierta o con fallos: **led01**
- g) Un Led rojo para indicar que la centralita está conectada y en estado de alerta: **led02**
- h) Una salida de ALARMADO cuando se activa cualquier zona de alarma: **led06**

Para la señalización óptica comprende 6 diodos LEDs, un diodo Led de color verde, **OK**, para indicar que todo está correcto, zonas 1 y 2 cerradas y alimentación. Dos diodos Leds de color naranja o amarillo que nos indica el estado de cada zona: cerrada o abierta. Un diodo Led rojo, **ALARMA ZONAS**, para indicar que las zonas están abiertas, un diodo Led rojo para indicar que la centralita está **CONECTADA**, y un diodo Led rojo, **ALARMADO**, que nos indica que se ha producido una intrusión mientras la centralita estaba conectada. En el caso de que se produzca una intrusión en una de las dos o las dos zonas de protección, se quedaría permanentemente encendido los Leds de la zona correspondiente y el Led de ALARMA ZONAS. Esto nos informaría en que zona, Zona 1 o Zona 2, se ha producido la intrusión durante la noche o en el tiempo que ha estado conectada la centralita.



Mientras no pulsemos el botón de activación de la centralita, los Leds del panel de zonas 1 y 2, Led OK, y Alarma de zonas, se irán encendiendo y apagando conforme se habrá una puerta o ventana, o detecte el dispositivo volumétrico un movimiento. Si todos los Leds estuvieran apagados se encendería el Led verde de **OK**. En este momento se puede conectar la centralita de alarma, escuchando un zumbador de aviso durante el margen de tiempo que se tiene para salir de la vivienda o local y conectarse definitivamente pasado este tiempo. Los Leds verde de OK y rojo de CONECTADA se quedarían encendido (centralita en alerta). Siempre habrá un Led o más de uno encendido, nunca deben estar todos apagados, a no sea por fallos de alimentación.

En el caso de que las zonas 1 y 2 estén cerradas, el Led verde de OK encendido, el circuito de conexión está a la espera de que pulsemos el **pulsador** (pulse) D7 para comenzar la cuenta, esto está formado por un bloque condicionante **do...while**, hasta 40 veces de repetición, a la misma vez se escucha un pitido que es un **zumbador** (zum) D11 que nos va avisando hasta que finaliza la cuenta, conectándose seguidamente el Led rojo de **CONECTADA** D4 y se queda permanentemente encendido hasta que no desconectemos la centralita. Para desconectar la rutina de alerta se debe pulsar el botón OFF (Reset).

MIS PROYECTOS CON ARDUINO

Estando conectada la centralita, si se produce una intrusión en la **ZONA 1** (retardada) comienza la temporización a la desconexión con un tiempo establecido, avisándonos mediante un zumbador para que desconectemos la centralita. Pasado este tiempo y no hemos desconectado la centralita se activa la salida de alarma **ALARMADO** actuando sobre los dispositivos de aviso. La **ZONA 2** (instantánea) al producirse una intrusión inmediatamente se activa la salida de alarma y se enciende los Leds correspondientes a **ALARMADO** y cada una de las zonas se quedarán permanentemente encendidos hasta que no desconectemos o reseteemos la centralita.

La señalización acústica por medio de un zumbador será un pitido intermitente, para ello, la frecuencia será de 120 milisegundos entre los ciclos **HIGH** y **LOW**.

14.2. Códigos de programación

```
/* Centralita de alarma de dos zonas con tiempo de retardo a la
conexión, zona 1 retardada a la desconexión y zona 2 instantanea */

int zum=11;           // declaramos la salida del zumbador
int pulse=7;          // declaramos la entrada del pulsador de conexion
int led01=13;         // declaramos led rojo de alarma
int led02=4;          // configuracion led rojo de conectado alarma
int led03=3;          // declaramos led verde de OK
int led04=10;         // declaramos led amarillo zona 1
int led05=9;          // declaramos led amarillo zona 2
int led06=2;          // declaracion de la salida de alarma con LED rojo
int zona1=6;          // declaramos zona 1 digital de entrada
int zona2=8;          // declaramos zona 2 digital de entrada
int var01=0;          // variable para entrada pulsador
int var02=0;          // variable para led verde
int var03=0;          // variable led amarillo zona 1
int var04=0;          // variable led amarillo zona 2
int var05=0;          // variable para la zona 1
int var06=0;          // variable para la zona 2
int var07=0;          // variable para almacenar led de alarma
int var08=0;          // variable para el Led de alarmado
int var09=0;          // variable para salida alarma zona 1 retardada
int x=0;
int y=0;

void setup()
{
  pinMode(zum,OUTPUT);    // configuramos el zumbador de salida
  pinMode(pulse, INPUT);  //configuramos el pulsador de entrada
  pinMode(zona1,INPUT);    // configuramos la zona 1 como entrada
  pinMode(zona2,INPUT);    // configuramos la zona 2 como entrada
  pinMode(led01,OUTPUT);   //configuramos led rojo de salida
  pinMode(led02, OUTPUT);  // configuramos el led rojo de conexión
  pinMode(led03,OUTPUT);   //configuramos led verde de salida
  pinMode(led04,OUTPUT);   //configuramos led amarillo 1 de salida
  pinMode(led05,OUTPUT);   //configuramos led amarillo 2 de salida
  pinMode(led06, OUTPUT);  // configuracion led de salida alarma
}
void loop()
{
  var01=digitalRead(pulse); // almacena el valor del pulsador
  var02=digitalRead(led03);  // almacena el estado del led verde
```

MIS PROYECTOS CON ARDUINO

```
if(var01==0 && var02==1 && var07==0 && var08==0) // condicional AND si el valor es
//distinto en los cuatro
do
{
    x=x+1;
    digitalWrite(zum, HIGH); // Suena el zumbador
    delay(120); // pausa de ciento veinte milisegundo
    digitalWrite(zum, LOW); // se calla el zumbador
    delay(120); // pausa de ciento veinte milisegundo
    digitalWrite(led02, HIGH); // enciende el led rojo
    delay(120); //pausa de ciento veinte milisegundo
    digitalWrite(led02, LOW); // apaga el led rojo
    delay(120); // pausa de ciento veinte milisegundo
}
while (x<40); // asignamos a x que cuente hasta 40
if (var02==1 && x==40) // condicional activación del led rojo conexion
{
    digitalWrite(led02,HIGH); // enciende el led rojo de conectado alarma
}
/* determina si se enciende cualquier led amarillo de zona se apagara el
led verde de OK*/

var03=digitalRead(led04); //lee el estado del led amarillo zona 1
var04=digitalRead(led05); // lee el estado de led amarillo zona 2
var08=digitalRead(led06); // lee el estado de led alarmado
if(var03==1 || var04==1 || var08==1) // condicionante cualquier led de zona
{
    digitalWrite(led01, HIGH); // enciende el led rojo de alarma
}
else
{
    digitalWrite(led01, LOW); // apaga el led rojo de alarma
}
/* determina que se encienda los led de cada zona */

var05=digitalRead(zona1); // lee el valor de la entrada digital zona 1
if (var05==0) // si variable esta a nivel bajo
{
    digitalWrite(led04, HIGH); // enciende el led amarillo zona 1
}
else
{
    digitalWrite(led04,LOW); // apaga el led amarillo zona 1
}
var06=digitalRead(zona2); // lee el valor de la entrada digital zona 2
if (var06==0) // si variable esta a nivel bajo
{
    digitalWrite(led05,HIGH); // enciende el led amarillo zona 2
}
else
{
    digitalWrite(led05,LOW); // apaga el led amarillo zona 2
}
/* esta parte conmuta el led verde de OK con el led rojo de alarma*/
if (var03 || var04==1 || var08==1) // condicionante nivel alto zona 1 OR zona 2
{
    digitalWrite(led03, LOW); // apaga el led verde
}
else
{
    digitalWrite(led03, HIGH); // enciende el led verde
```

MIS PROYECTOS CON ARDUINO

```
}

/* Ponemos una serie de variables a nivel alto para condicionar la
activacion de la alarma */

var07=digitalRead(led02); // lee el valor de led rojo y lo guarda en variable
if (var07==1 && var06==0) // condicionante alarma zona 2 instantanea
{
    digitalWrite(led06,HIGH); // activamos la salida de alarma instantanea
}
/* determina la memorizacion de la alarma instantanea zona 2 */
if(var07==1 && var04==1)
{
    digitalWrite(led05,HIGH);
    digitalWrite(led01, HIGH);
}
var09=digitalRead(zona1);
if (var09==0 && var07==1 && var02==1) // condicionante para activar la zona 1
retardada
do
{
    y=y+1;
    digitalWrite(zum, HIGH); // Suena el zumbador
    delay(120); // pausa de ciento veinte milisegundo
    digitalWrite(zum, LOW); // se calla el zumbador
    delay(120); // pausa de ciento veinte milisegundo
    digitalWrite(led02, HIGH); // enciende el led rojo
    delay(120); //pausa de ciento veinte milisegundo
    digitalWrite(led02, LOW); // apaga el led rojo
    delay(120); // pausa de ciento veinte milisegundo
}
while (y<40); // asignamos a y que cuente hasta 40
if (var07==1 && y==40) // condicionante para activar zona 1 retardada
{
    digitalWrite(led06,HIGH); // activamos la alarma de salida y led rojo
    digitalWrite(led04,HIGH); // activamos el led amarillo zona 1
    digitalWrite(led01,HIGH); // activamos led rojo alarma
    digitalWrite(led02,HIGH); // activamos led rojo de conexion alarma
}
}
```


14.3. Descripción y funcionamiento del circuito

El circuito se compone principalmente de un microcontrolador Atmega328P-PU, DIP de 28 pines, con la programación cargada, que realiza todas las tareas de gestión de la centralita de alarma de dos zonas, con la programación descrita anteriormente.

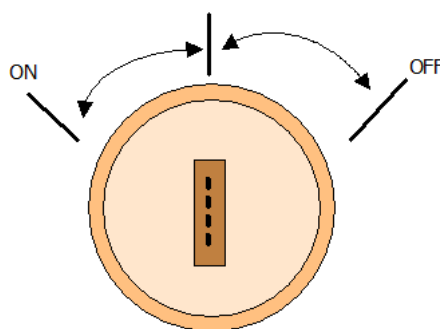
Como es sabido, la polarización de los componentes utilizados en este proyecto, es importante de tenerlo en cuenta, para evitar que los pines de E/S se sobrecarguen a las salidas y entradas al microcontrolador, para ello, se proveen resistencias de 470 Ohmios para polarizar los diodos Leds, de 3K3 para polarizar las bases del TIC 31C. Esto evita que se puedan estropearse alguna entrada o salida del chip.

La ZONA 1 (D6) y ZONA 2 (D8), de entrada al microcontrolador, están tomadas en la programación con valor alto, para ello, se ha previsto de la conexión a +5Vcc con una resistencia de polarización en serie de 4700 Ohmios de protección, suficiente para poner los dispositivos en serie y cerrarse el lazo.

En el esquema eléctrico se muestra seis partes fundamentales del circuito:

1. El Microcontrolador Atmega328P-PU con todas sus entradas y salidas indicadas, alimentación, señal de reloj, y reset.
2. La Fuente de alimentación de +12V y +5V estabilizadas.
3. Salida de Zumbador y circuito de Relé.
4. Todos los Leds de señalización óptica.
5. Conexión y desconexión de la centralita de alarma
6. Entradas de la zona 1 y la zona 2 NC.

Con el microcontrolador Atmega328P-PU programado y perfectamente polarizado en alimentación 7 y 8, para la alimentación, y la señal de reloj pines 9 y 10. El circuito de reset lo tenemos en el pin 1 con una polarización de 10 K a +5V, utilizado en este caso también para inicializar la centralita de alarma, lo mismo ocurre con el pulsador de conexión (ON) conectado al pin 13 del microcontrolador con una resistencia de 10K polarizada a +5V. Estos pulsadores perfectamente se pueden sustituir por llaves tubulares de seguridad que sean del tipo pulsador de tres posiciones: ON--OFF y que vuelvan a su posición central:



Puede haber otros tipos de conexión y desconexión de la centralita como son los llaveros de radiofrecuencia que se compone de un mando tipo llavero con dos pulsadores ON—OFF, etc.

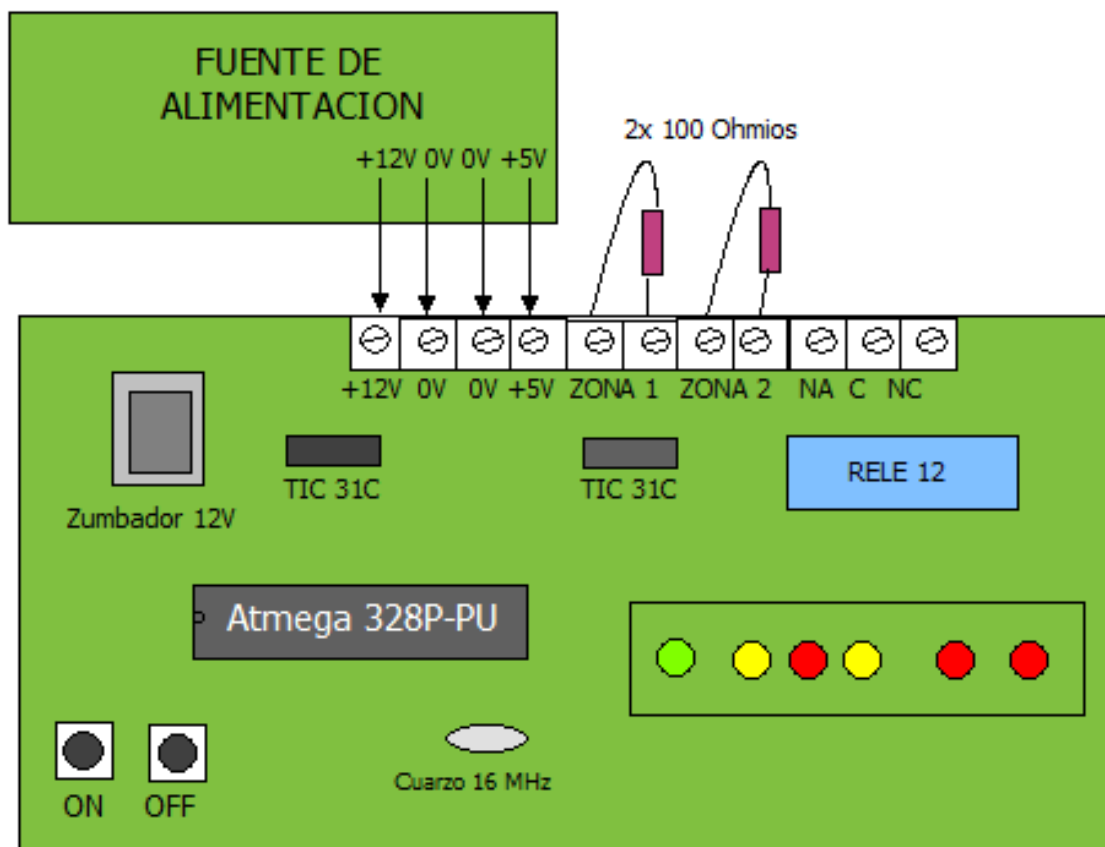
La fuente de alimentación es del tipo lineal con dos reguladores de tensión positiva de 12 y 5 voltios. Con los reguladores L7812 y 7805.

MIS PROYECTOS CON ARDUINO

Una salida de relé electromagnético de 12 Vcc que activará exteriormente otros dispositivos, cuando se produce la alarma, podrá activar sirenas, campanas, luces, marcadores telefónicos, etc. Un zumbador de 12 voltios de salida, avisará cuando hemos conectado la centralita y estamos saliendo o cuando estamos entrando para desconectar la centralita. Estos elementos requieren de una alimentación de 12 Vcc.

Un panel de 6 Leds nos indica en cualquier momento el estado de la centralita de alarma:

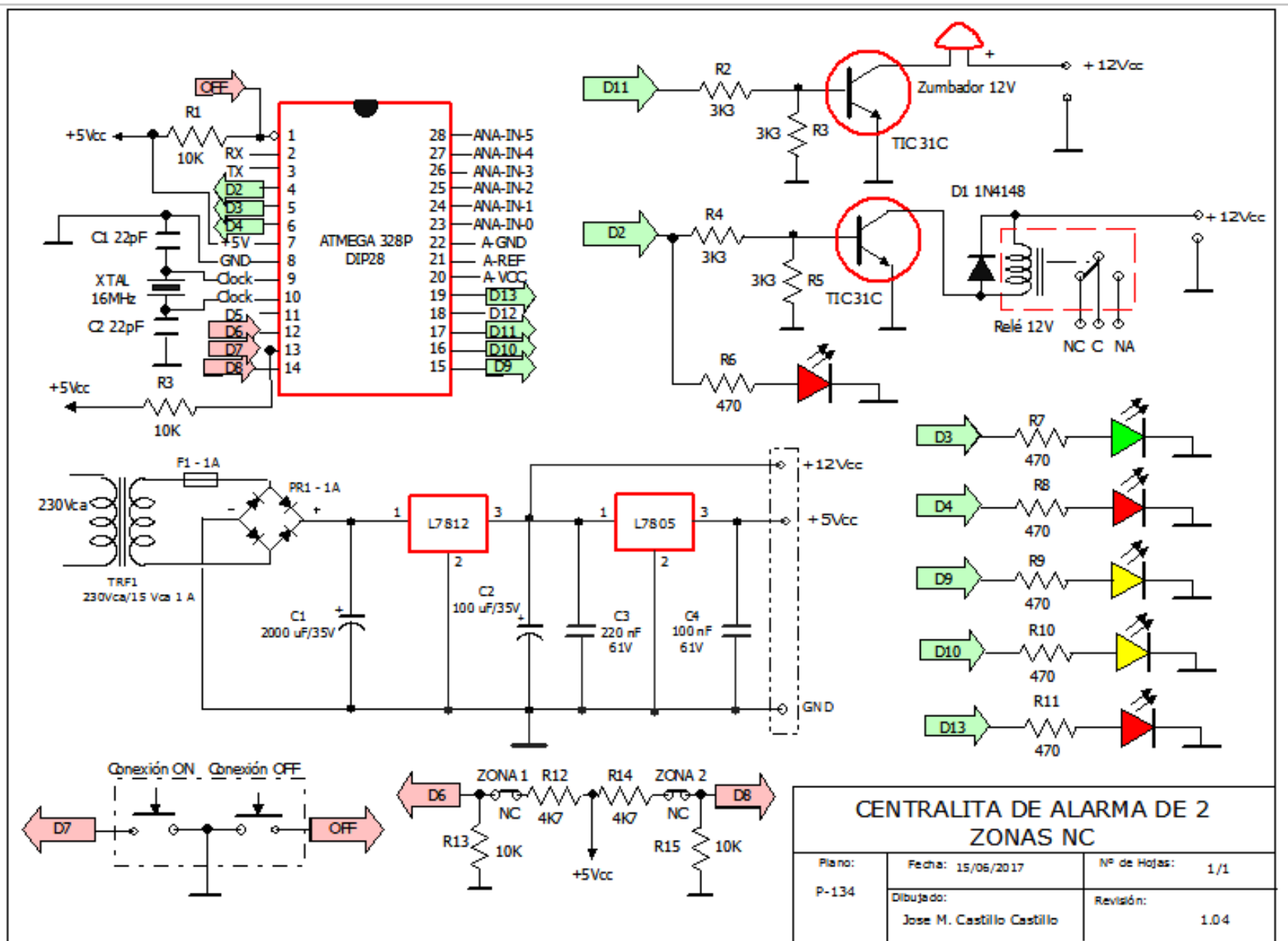
1. Led verde de OK (D3). Encendido indica que está correcto y sin alarma
2. Led naranja ZONA 1 (D9). Encendido indica que se ha producido una intrusión (lazo abierto)
3. Led rojo de ALARMA (D13). Encendido indica ALARMA de ZONAS abiertas.
4. Led naranja ZONA 2 (D10). Encendido indica que se ha producido una intrusión (lazo abierto)
5. Led rojo de CONECTADO (D4). Encendido indica que la centralita está CONECTADA.
6. Led rojo de ALARMADO (D2). Encendido indica que se ha producido una intrusión estando conectada y se ha activado los elementos de señalización, óptica y acústica.



Los transistores de acoplamiento NPN TIC 31C sirven de amplificación de las señales de salidas D2 y D11 del microcontrolador y polarizar con +12 Vcc el zumbador y el relé de salida que trabajan a 12Vcc. En este caso no utilizamos Optoacopladores puesto que la masa de las tensiones de +5Vcc y +12Vcc es la misma.

En los bornes de conexión del relé se dispondrá de la conexión de un diodo **1N4148** para evitar y proteger contra inducciones el circuito electromagnético.

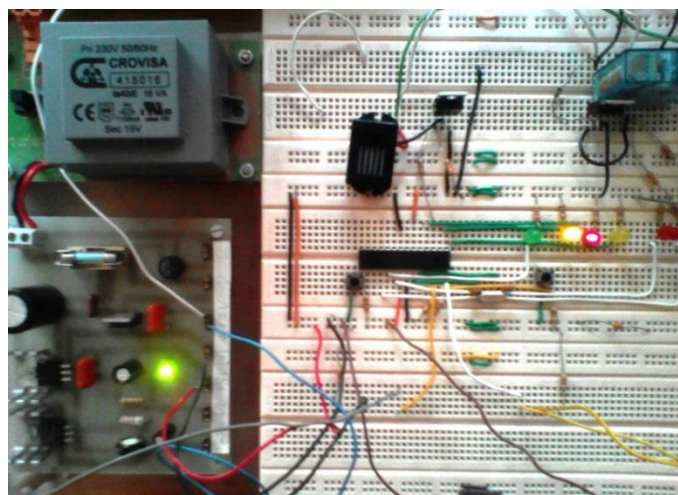
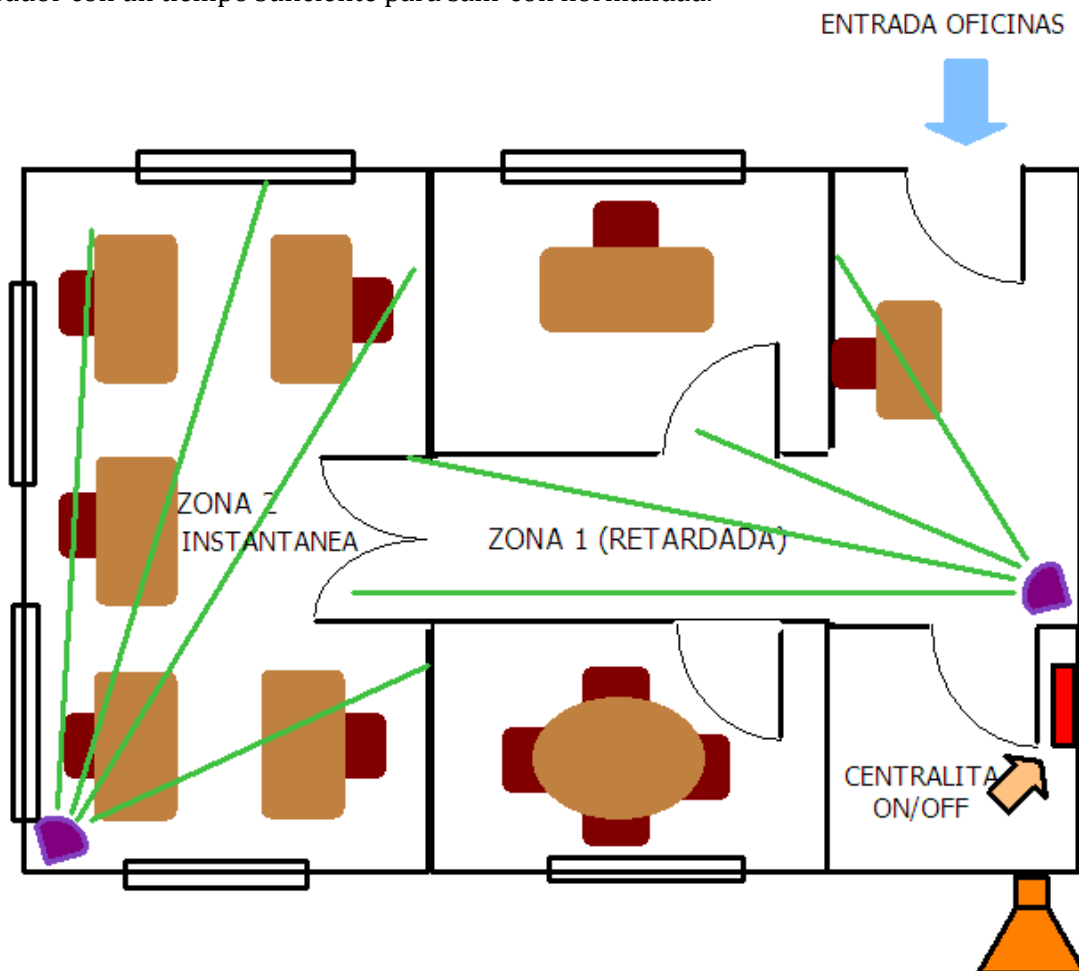
MIS PROYECTOS CON ARDUINO



MIS PROYECTOS CON ARDUINO

En la siguiente figura se muestra como ejemplo la instalación de un sistema de alarma contra robo en un edificio de oficinas. Como se podrá observar se ha dividido en dos zonas: Zona 1 retardada y Zona 2 instantánea. Cada zona posee un detector volumétrico de infrarrojos pasivo de poco más de 15 metros. Al entrar o salir del edificio de oficinas, hay programado un tiempo para desconectar y conectar la centralita. En el caso de entrar por la Zona 1 se activa y suena un zumbador, avisándonos, de un breve tiempo que nos permite desconectar la centralita de alarma, pasado este tiempo y si no se ha desconectado se activa la sirena de alarma. Si entrásemos por cualquiera de las ventanas de la zona 2 se activaría de inmediato la centralita de alarma.

Para poder conectar la centralita debe de estar todas las zonas sin alarma, en este caso, se encenderá el Led verde de OK y se podrá conectar la centralita y salir del recinto, avisándonos un zumbador con un tiempo suficiente para salir con normalidad.



15. PROYECTO: SIMULADOR DE PRESENCIA ANTIRROBO

La simulación de presencia en una casa o en un negocio es un medio muy eficaz de disuasión contra los robos por intrusión. La forma más corriente de simulación de presencia consiste en el empleo de luces que se enciende y se apagan, de manera más o menos automática y aleatoria, durante la noche en las habitaciones de una casa temporalmente desocupada o en las oficinas de una empresa.

Este tema se ha tratado con cierta frecuencia en la prensa especializada, en la que se encuentran ideas muy sencillas para controlar movimientos (cortinas, puertas, sombras, etc.) y ruidos, especialmente de tipo doméstico.

En el mercado existen numerosas empresas que ofrecen instalaciones de alarma, con un gran despliegue de publicidad, ensalzando sus características, normalmente a un precio bastante elevado.

En cambio, la simulación no recibe estos alardes publicitarios, puesto que su instalación suele estar presidida por la discreción. Sin embargo, una simulación de presencia puede complementar muy eficazmente un sistema de alarma. En estos últimos, el sistema se activa cuando ya se ha roto algo, mientras que la simulación de presencia es una disuasión.

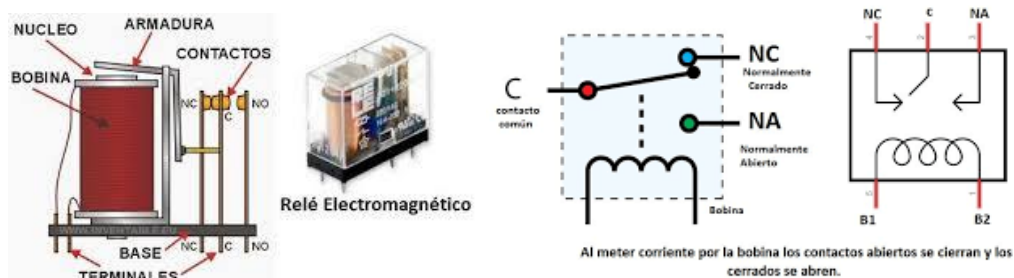
Como se verá a continuación, nuestro proyecto contempla una simulación de presencia antirrobo que es bastante más económica que un sistema de alarma y bastante más fácil de realizar con la placa Arduino, formado por la programación de un microcontrolador Atmega328P-PU y con unos pocos componentes como resistencias, transistores, reguladores de tensión y relés.

15.1. Datos para la programación (recopilamos información)

En este proyecto se tiene que incorporar una fotocélula LDR que, en el crepúsculo, active el programa y enciendan y apaguen varias veces las luces de manera programada. Por tanto, es posible realizar un programa de simulación que dure hasta la mañana siguiente, o sea, cuando la fotorresistencia recibe una cantidad de luz suficiente para finalizar la programación y quedar en estado de *standby* en espera de que se haga de noche para volver a activarse. Naturalmente, el sistema sólo puede funcionar correctamente si la fotorresistencia está orientada hacia una ventana y al abrigo de las luces que ella misma controla.

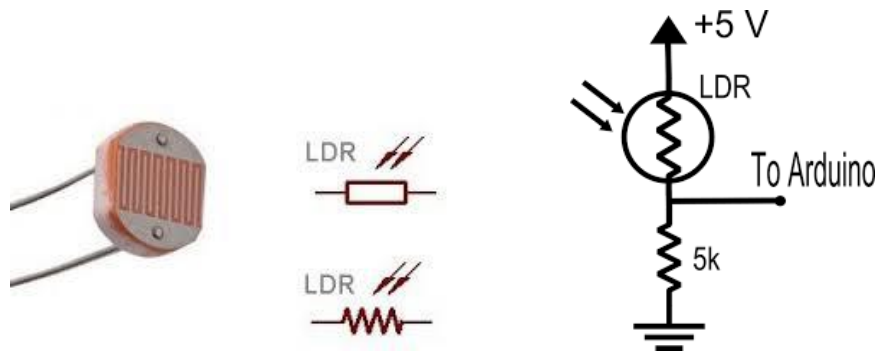
Sin embargo, en los casos que pueda haber una falsa señal o información que pueda activar o desactivar la programación, se dispone de una tiempo prudencial que nos permita volver a preguntarle a la fotocélula si efectivamente se ha hecho de día o de noche en cuyo caso comenzará, continuará o detendrá la programación.

Los circuitos de control de salida que constituyen el proyecto están basados principalmente en circuitos con relés, donde se utilice las conexiones de salida correspondiente a un interruptor normalmente abierto NA, del conmutado NA-C-NC, que cuando se active cierra el contacto y haga pasar la corriente eléctrica de 230 Vca a una bombilla del tipo económica de 11W e incluso también se puede añadir una radio que funcione a 230 Vca.



MIS PROYECTOS CON ARDUINO

La fotorresistencia LDR nos sirve para iniciar los ciclos en cada crepúsculo y se chequea varias veces al principio para confirmar el estado que es de noche, y no se produzcan falsas conexiones. De esta manera, en el momento que la fotocélula detecta la luz diurna la programación se detiene.



Vamos hacer la secuencia de la programación como si realmente estuvieran viviendo un matrimonio, sin hijos, diariamente en la vivienda. Suponiendo, como ejemplo, la vivienda del dibujo, donde consta de 2 habitaciones, salón-comedor, cocina y cuarto de baño. El proceso seria secuencial tomando primeramente como referencia la señal de la fotocélula LDR que nos informa cuando se ha establecido la noche y seguidamente se inicia la programación.

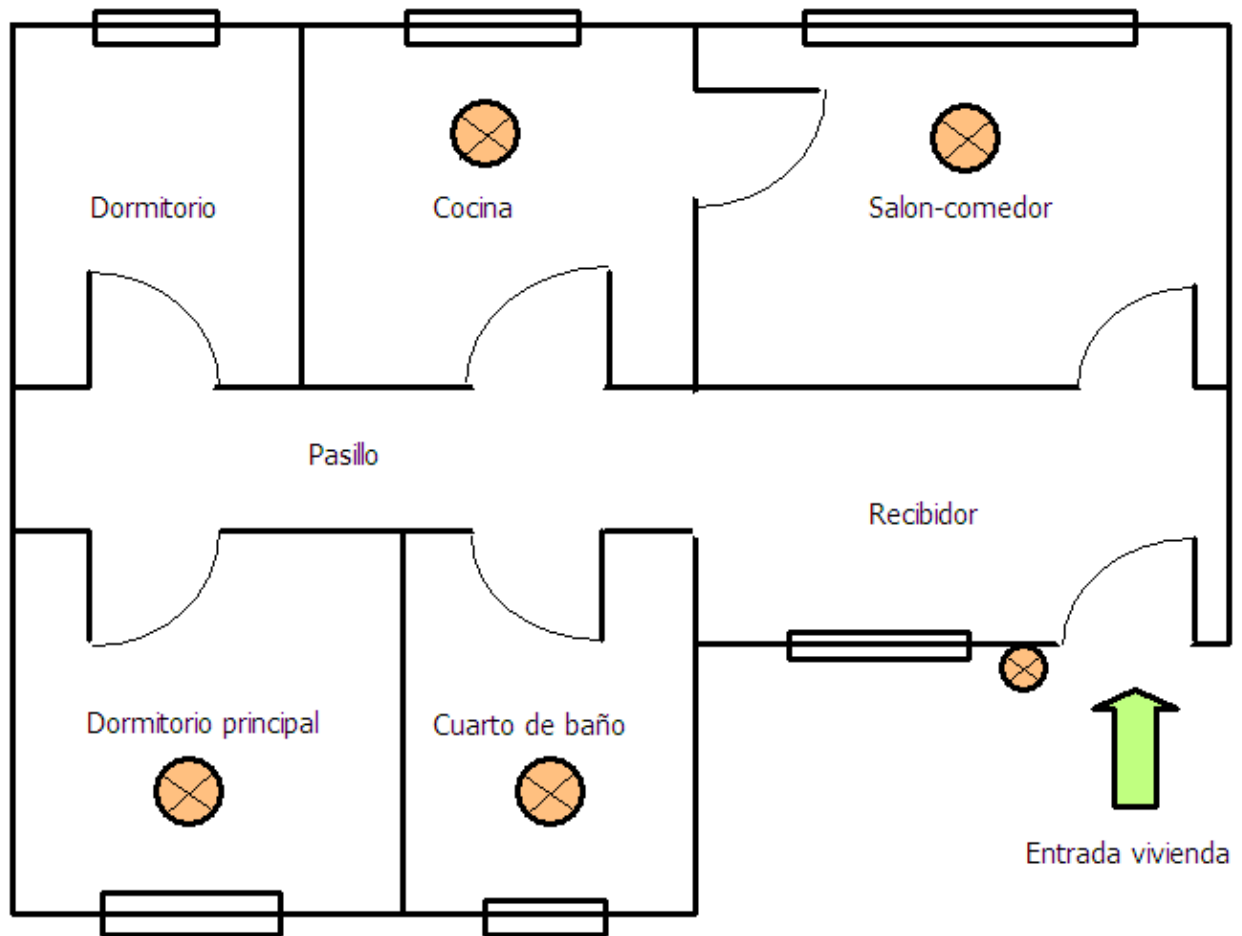
1. Se hace de noche, la LDR detecta este ciclo crepuscular pero antes de comenzar la programación establecemos varios retardos (delay) para verificar que efectivamente se ha hecho de noche. Si efectivamente es de noche se activa y comienza la programación. Hay que tener en cuenta que la fotocélula no se debe disparar por falsa detección o por histéresis. Para ello ponemos un retardo que transcurrido un cierto tiempo vuelva a preguntar si sigue la fotocélula activa y por lo tanto se pueda proceder a comenzar el programa.
2. Al comenzar la programación, la farola de la calle se enciende (queda permanentemente encendida hasta que la fotocélula detecte el día y se apaga). Este Led01 de salida va incorporado en el equipo y nos servirá para indicar que el sistema está activado o desactivado.
3. Transcurridos un breve tiempo se enciende la luz del salón-comedor.
4. Un poco más tarde se enciende la luz del cuarto de baño (se enciende y se apaga después de unos minutos)
5. Después de un tiempo se enciende la luz de la cocina
6. Se enciende la luz del dormitorio principal (se enciende y se apaga en unos minutos)
7. Se apaga la luz de la cocina después de un buen tiempo encendida
8. Se apaga la luz del salón comedor
9. Se enciende la luz del cuarto de baño (se enciende y apaga durante 15 minutos)
10. Se enciende la luz del dormitorio principal (permanentemente durante 60 minutos)
11. Queda 60 minutos todo apagado
12. Se enciende la luz del dormitorio
13. Se enciende la luz del cuarto de baño y se apaga a los 20 minutos
14. Se apaga la luz del dormitorio.
15. Pasan 60 minutos todo apagado
16. Se enciende la luz del dormitorio
17. Se enciende la luz de la cocina durante 20 minutos
18. Se apaga la luz del dormitorio
19. Damos tiempo para llegar al día
20. La fotocélula detecta día y se detiene el programa

MIS PROYECTOS CON ARDUINO

Vamos a utilizar los siguientes pines de salida digital:

- D5(11)= Farola de la calle
- D6(12)= Salon_comedor
- D7(13)= Cocina
- D8(14)= Cuarto de Baño
- D9(15) = Dormitorio Principal

Plano de la vivienda:



En la imagen anterior se indican los puntos de luz a controlar. Hay que tener en cuenta para ello que se debe utilizar los puntos de la vivienda que con mayor frecuencia se están utilizando por el dueño. En este caso hemos utilizados cuatro puntos internos de la vivienda: la cocina, el salón-comedor, cuarto de baño y el dormitorio principal.

El tiempo en la programación del encendido y apagado de las lámparas en cada recinto de la vivienda es clave para determinar que no sea un fraude, es decir, si vamos 10 veces consecutivas al baño, apagando y encendiendo las luces, parecería una discoteca, un síntoma poco usual. Lo suyo sería dedicarle un tiempo prudencial o incluso largo para crear un ambiente de normalidad.

MIS PROYECTOS CON ARDUINO

Una de las formas que se suele utilizar para configurar el encendido del día y la noche utilizando una fotocélula LDR es recurriendo al Monitor Serial que nos permite visualizar los valores analógicos de entrada al pin AN0 tomado de la fotocélula LDR para determinar el valor del nivel de luz o sensibilidad del día y de la noche.

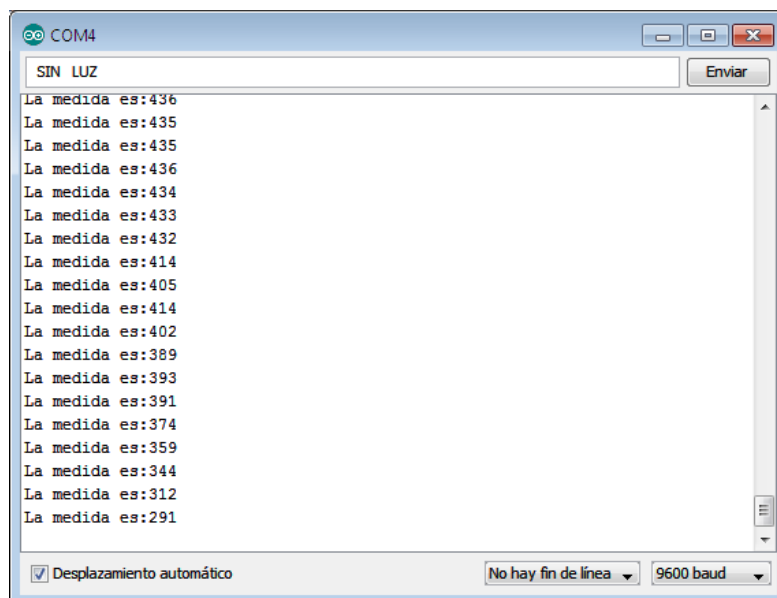
En este caso advertimos que los valores captados mediante la LDR pueden estar comprendidos entre 0 a 1.023. Estos son valores que puede tomar la LDR. (En el apartado 9.8 tenemos la función `map()` que nos permite delimitar estos valores).

Según el código del programa siguiente podemos visualizar en el Monitor Serial el valor numérico:

```
int medida=0;
int nivel=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()
{
  medida=analogRead(ldr); // tomamos lectura de la ldr
  if (medida<nivel) // condiciona que si la medida es menor que 300
  }
```

Este código es esencial para determinar el nivel de luminosidad del día y la noche. Se ha observado que cuando desciende la luminosidad de la fotocélula (más oscuridad) ésta experimenta un descenso del nivel de la medida hacia abajo, y creciendo cuando la luminosidad es más alta. Por ello, en este caso hemos puesto una condición que si la medida es menor que el nivel prefijado de 300 se active la programación, es cuando tenemos tapada la fotocélula.

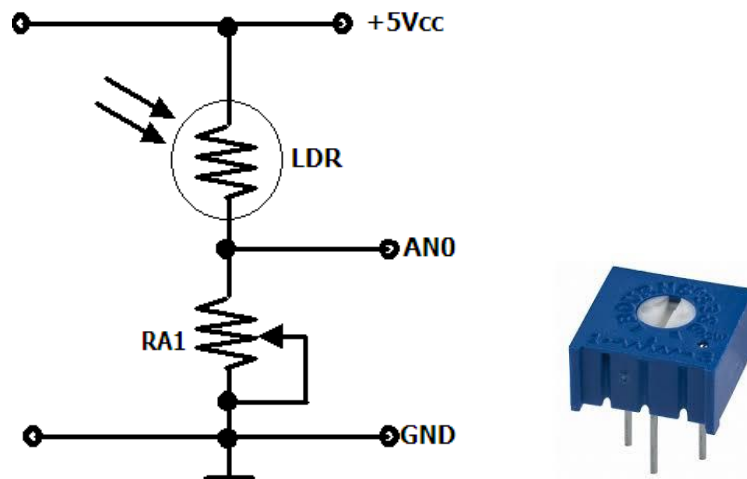
El valor de sensibilidad que aparece en el código del proyecto tiene sólo carácter orientativo, ya que el lector deberá ajustar el valor de dicha variable según la luz ambiente de que disponga en el momento de configurar la LDR.



MIS PROYECTOS CON ARDUINO

Observar en el Monitor Serial que la medida de luz, conforme va haciéndose más intensa la luz aumenta la medida 402,414, 436... y conforme va disminuyendo la intensidad de luz incluso tapando la fotocélula LDR la medida va bajando por debajo de los 400, 344, 312, 291... es decir a mayor oscuridad de la fotocélula el valor de la medida es menor. Si establecemos el valor de la luz a 400 en vez de 300, se activaría demasiado pronto el simulador, siendo todavía de día. Estas variaciones se producen debido a la polarización de la fotocélula con diferentes valores del divisor de tensión y su resistencia de polarización, en nuestro caso hemos utilizado el valor de una resistencia ajustable de 10K.

Con este tipo de resistencia ajustable la sensibilidad se consigue con mayor exactitud que con una resistencia fija. Esto nos permite ajustar gradualmente el punto umbral de oscuridad que deseamos para que se active el programa.



El potenciómetro o resistencia ajustable no es más que una resistencia en la que se puede ajustar o variar su valor resistivo girando una ruedecita de derecha a izquierda o viceversa, aumentando la resistencia o disminuyendo a nuestro gusto. Estas resistencia ajustables poseen tres terminales; el que está en medio, es el cursor y se utiliza para ajustar el valor de la resistencia en ese momento.

En nuestro caso solamente hay que ponernos con la fotocélula LDR, en un lugar adecuado, orientándolo hacia un ambiente natural nocturno, y al abrigo de las demás luces, focos o farolas que puedan perturbar la medición. Cuando tengamos el umbral de nocturnidad que deseamos, iremos moviendo la ruedecita de ajuste de la resistencia, hasta conseguir que se active el programa o se encienda la lámpara Led01 donde comenzará la programación del simulador.

15.2. Código de Programación

```
/* Programa simulador de presencia antirobo, se compone de
5 pines de salida D5, D6, D7, D8 y D9 cuya funciones se encarga de
encender y apagar la luz de los recintos de la vivienda y un
pin de entrada analogico AN0 que constituiria la señal LDR
para establecer la noche y el día.
La conexión y desconexión del simulador será a través
de un interruptor que de paso de corriente continua al circuito*/
int ldr=0; // declaramos pin de entrada analogico de la ldr
int led01=5; // declaramos pin de salida farola exterior y conexion
int led02=6; // declaramos pin de salida salon-comedor
int led03=7; // declaramos pin de salida cocina
int led04=8; // declaramos pin de salida cuarto de baño
int led05=9; // declaramos pin de salida dormitorio principal
int medida=0; // declaramos la variable medida
int nivel=300; // establecemos el valor de luz a 300

void setup()
{ // configuramos los elementos de entrada y salida
pinMode(ldr,INPUT); // asociamos el pin ldr de entrada
pinMode(led01, OUTPUT); // asociamos el pin led01 a salida
pinMode(led02, OUTPUT); // asociamos el pin led02 a salida
pinMode(led03, OUTPUT); // asociamos el pin led03 a salida
pinMode(led04, OUTPUT); // asociamos el pin led94 a salida
pinMode(led05, OUTPUT); // asociamos el pin led05 a salida
}
void finaliza() // bloque de función para apagar todas las luces
{
digitalWrite(led01,LOW); // apaga farola encendida
digitalWrite(led02,LOW); // apaga salon-comedor
digitalWrite(led03,LOW); // apaga cocina
digitalWrite(led04,LOW); // apaga cuarto de baño
digitalWrite(led05,LOW); // apaga dormitorio de matrimonio
delay(1000); //pausa
}
void loop(){ // comienza la rutina
medida=analogRead(ldr); // leemos el nivel de umbral de la LDR
if (medida<nivel) // ponemos condicion si es menor que el prefijado
{
delay(60000); // retardo de minutos para preguntar de nuevo
}
else
{
finaliza(); // apaga todas las luces
}
medida=analogRead(ldr); // volvemos a medir la cantidad de luz
if (medida<nivel) // condiciona si es menor que el nivel prefijado
{
delay(60000); // retardo de minutos para confirmarlo y volverlo a preguntar
}
else
```

MIS PROYECTOS CON ARDUINO

```
{
  finaliza(); // apaga todos las luces
}
medida=analogRead(ldr); // volvemos a medir la cantidad de luz

if (medida<nivel) // si la medida es menor que el nivel prefijado comienza
programa
{
  delay(180000); // retardo de 3 minutos antes de encender la farola
  digitalWrite(led01, HIGH); // se enciende la farola permanentemente
  delay(600000); // retardo de 10 minutos para encender luz salon
  digitalWrite(led02, HIGH); // se enciende la luz del salon-comedor
  delay(900000); // retardo de 15 minutos para encender la luz del cuarto de baño
  digitalWrite(led04, HIGH); // se enciende la luz del cuarto de baño
  delay(600000); // retardo de 10 minutos para apagar la luz del cuarto de baño
  digitalWrite(led04, LOW); // se apaga la luz del cuarto de baño
  delay(900000); // retardo de 15 minutos para encender la luz de la cocina
  digitalWrite(led03, HIGH); // se enciende la luz de la cocina permanentemente
  delay(1500000); // retardo de 25 minutos para encender la luz del dormitorio
  digitalWrite(led04, HIGH); // se enciende la luz del dormitorio
  delay(1500000); // retardo de 25 minutos para apagar la luz del dormitorio
  digitalWrite(led04, LOW); // se apaga la luz del dormitorio
  delay(3600000); // retardo de 1 hora
  digitalWrite(led03, LOW); // se apaga la luz de la cocina
  delay(900000); // retardo 15 minutos para encender la luz del cuarto de baño
  digitalWrite(led04, HIGH); // enciende la luz del cuarto de baño
  delay(900000); // retardo de 15 minutos para apagar la luz del cuarto de baño
  digitalWrite(led04, LOW); // se apaga la luz del cuarto de baño
  delay(3600000); // retardo de una 1 hora para apagar la luz del salon comedor
  digitalWrite(led02, LOW); // se apaga la luz del salon
  delay(150000); // retardo de 2 minutos para encender la luz del cuarto de baño
  digitalWrite(led04, HIGH); // enciende luz del cuarto de baño
  delay(300000); // retardo de 5 minutos para encender la luz del dormitorio
  digitalWrite(led05, HIGH); // enciende la luz del dormitorio
  delay(600000); // retardo 10 minutos para apagar la luz del cuarto de baño
  digitalWrite(led04, LOW); // se apaga luz del cuarto de baño
  delay(3600000); // retardo de 1 hora
  digitalWrite(led05, LOW); // apaga luz del dormitorio
  delay(7200000); //retardo de 2 horas para encender la luz del dormitorio
  digitalWrite(led05, HIGH); // enciende la luz del dormitorio
  delay(300000); //retardo de 5 minutos para encender la luz del cuarto de baño
  digitalWrite(led04, HIGH); // enciende la luz del cuarto de baño
  delay(600000); // retardo de 10 minutos para apagar la luz del cuarto de baño
  digitalWrite(led04, LOW); // se apaga la luz del cuarto de baño
  delay(600000); // retardo de 10 minutos para apagar la luz del dormitorio
  digitalWrite(led05, LOW); // se apaga la luz del dormitorio
  delay(7200000); // retardo de 2 horas
}
else
{
  finaliza(); // apaga todas las luces
}
}
```

15.3. Descripción y funcionamiento del circuito

El circuito se compone de cuatro partes:

- **La fuente de alimentación,**
- **el circuito de control**
- **circuito de entrada**
- **y el circuito de salida**

La **fuente de alimentación** es del tipo lineal formada por dos reguladores de tensión de 12 y 5 voltios. Los 5 voltios se consigue a través de la salida del regulador de 12 voltios y estos 5 voltios están dedicados a suministrar tensión al microcontrolador y Leds. Los 12 voltios están para alimentar los relés a través de los transistores TIC 31C que activa los relés con poca corriente que les llega a sus bases mediante los pines de salidas digitales del microcontrolador. Al tener la misma masa GND no hace falta utilizar los optoacopladores.

El **circuito de control** comprende el microcontrolador Atmega 328p con los componentes auxiliares necesarios para su correcta polarización y funcionamiento: condensadores, cristal de cuarzo de 16MHz, pulsador, resistencias, etc. Éste dispone de los pines de entrada y salida configurable con el programa cargado, y nos permite activar las salidas D5, D6, D7, D8 y D9 para activar los relés dependiendo de la señal de entrada AN0 de la LDR.

El **circuito de salida** lo forman el grupo de relés y sus contactos de salidas conmutados que conectamos a las lámparas de los diferentes recintos de la vivienda, para suministrarle corriente eléctrica alterna de 230 Vca. Cuando se produce la salida HIGH de un pin del microcontrolador, que tiene muy poca corriente, se aplica a la base del transistor TIC 31C y lo hace conducir circulando corriente a través del emisor al colector y alimentando con suficiente corriente y tensión al relé que se activa y mueve los contactos conmutados, que se encuentran independiente, cerrando el circuito y encendiendo las lámparas que trabajan a 230 voltios alterna y cuya potencia no sea más de 11 vatios y del tipo económica.

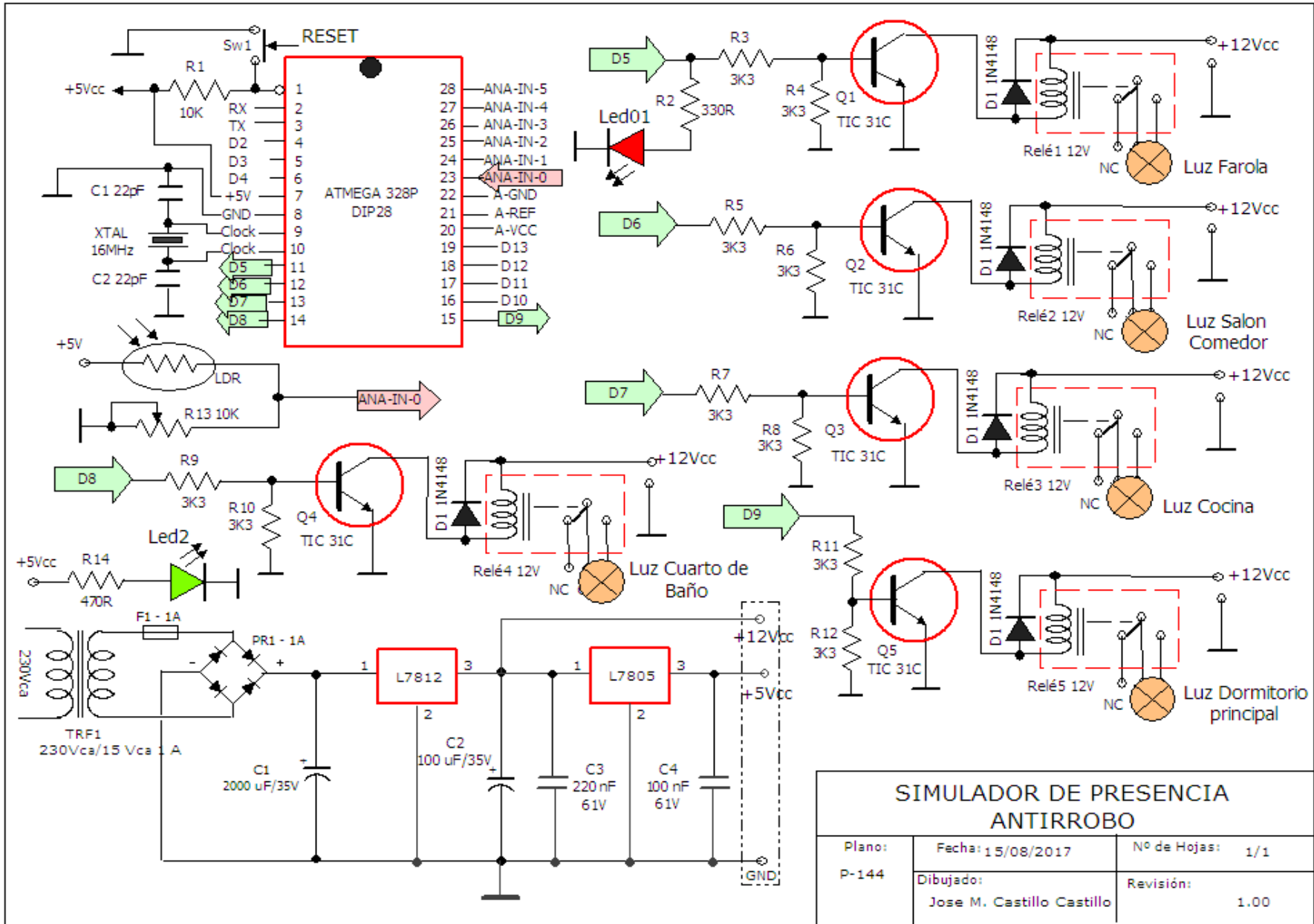
El **circuito de entrada** está comprendido por la fotocélula LDR que es la encargada de controlar la sensibilidad de luz para que se ponga la programación en marcha. Ésta se puede ajustar por medio de una resistencia ajustable la sensibilidad o cantidad de luz ambiental que deseamos se active el programa, girando la ruedecita hasta conseguir encender la lámpara con la oscuridad deseada.

El microcontrolador Atmega 328P-PU está sincronizado con la señal de reloj de 16 MHz, generada por medio de un cristal de cuarzo en sus pines 9 y 10 que hace que la frecuencia sea muy estable.

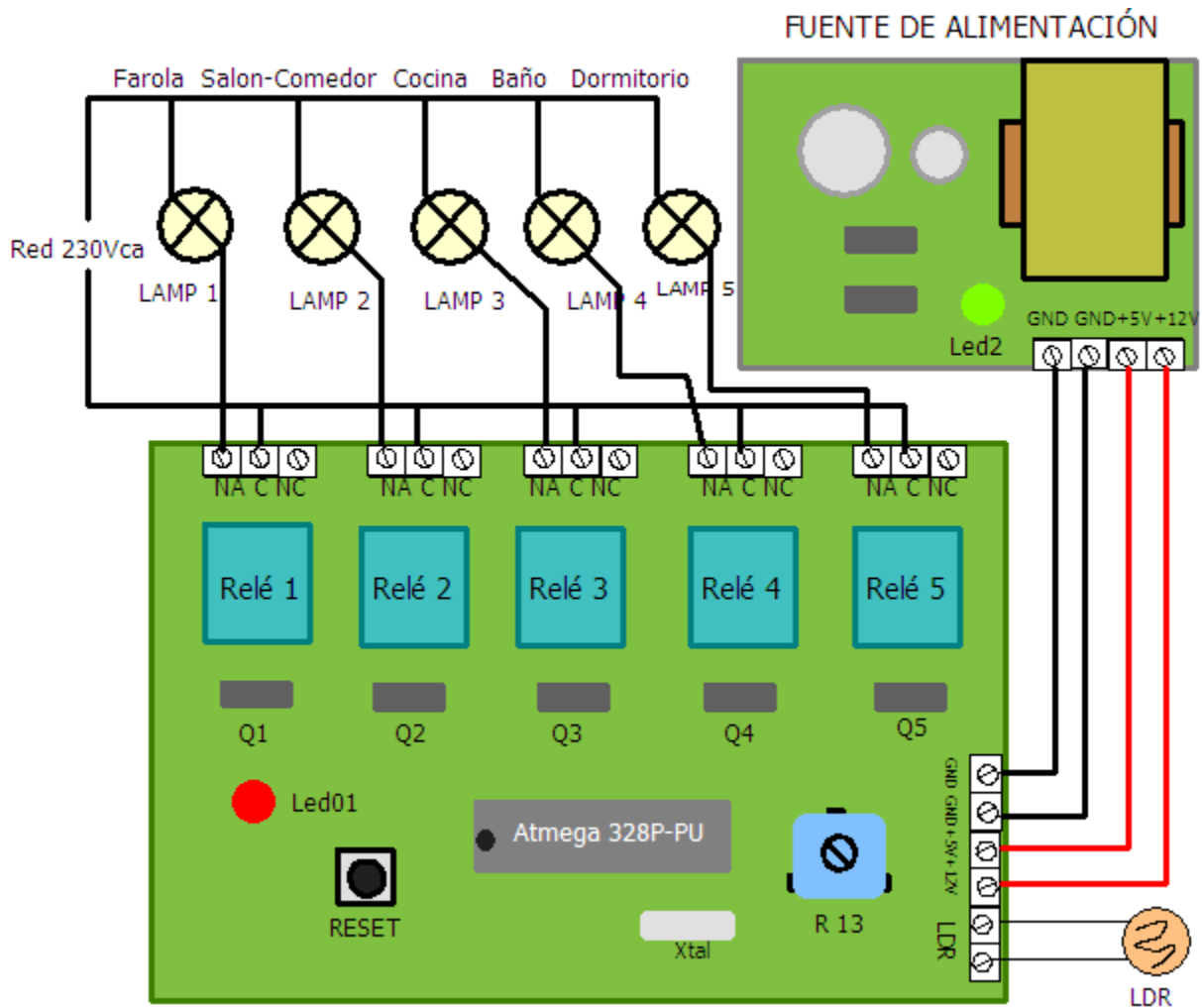
También se dispone de un pulsador para reset, que en cualquier momento, si se pulsa, inicializa el programa desde el principio (es como si apagamos y encendemos el equipo).

Se añaden dos Leds de señalización, el Led 2, de color verde, nos indica que la tensión de 12 y 5 voltios están correctamente en buen estado. El Led 01 de color rojo nos indica cuando está encendido que el programa se está ejecutar.

MIS PROYECTOS CON ARDUINO



MIS PROYECTOS CON ARDUINO



Esquema del conjunto completo de los componentes utilizados en el equipo simulador: placa del circuito de control y fuente de alimentación con sus componentes.

Como se muestra en el dibujo, se aprecia la fuente de alimentación de +12 y +5 voltios con sus salidas a la placa de control y en la placa de control el grupo de 5 relés con sus respectivos transistores. Las salidas de los relés se escoge un interruptor C-NA que cuando se activa el relé se cierre el circuito y hace pasar corriente para encender o apagar las luces del recinto de la vivienda.

R13 es la resistencia ajustable que permitirá escoger el punto de umbral de nocturnidad para activar la programación.

Para inicializar la programación se utiliza un pulsador Reset que nos permita, debido a cualquier incidencia, se pueda volver a recuperar la programación.