RELOJ LEDS 24 HORAS CON DISPLAYS

En esta versión 2, del reloj vamos a utilizar displays de 7 segmentos, aunque se podría hacer con leds al igual que la primera, lo que va a ser completamente diferente, es la programación. En vez de utilizar componentes electrónicos, vamos a usar el Arduino para que se encargue de todo.
Y como va a ser mucho más pequeño que el otro porque los displays son chiquitines, vamos a usar una caja de cartón del chino para meter todo dentro.

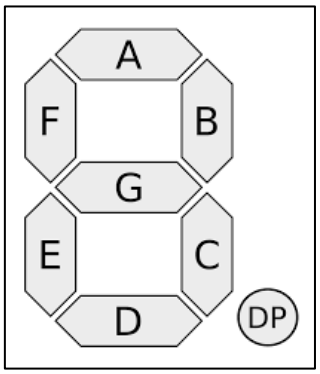
Materiales:

- Soldador
- Estaño
- Caja de cartón
- Cúter
- unos restos de madera para poder atornillar el Arduino (si tenemos impresora 3D le hacemos una base a medida, dejo el link al final)

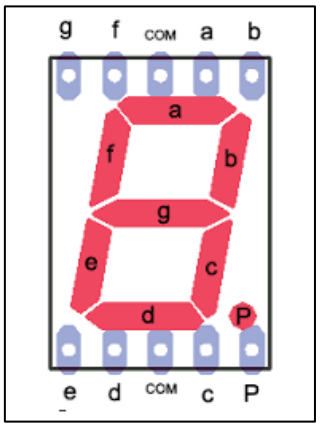
Listado de componentes:

- 1 Arduino con cable usb
- 4 displays de 7 segmentos
- Alimentador usb 5v/1ª
- 2 resistencias de 10K
- 2 pulsadores
- 1 placa multiperforada.

Introducción:



Esto sería un display de 7 segmentos, un componente que contiene 10 patillas, con 7 leds para formar cualquier número y uno más para el punto.
Si alimentamos B y C tenemos el 1.
Si alimentamos A, B, G, E y D tenemos el 2.
Creo que se entiende, ¿no?

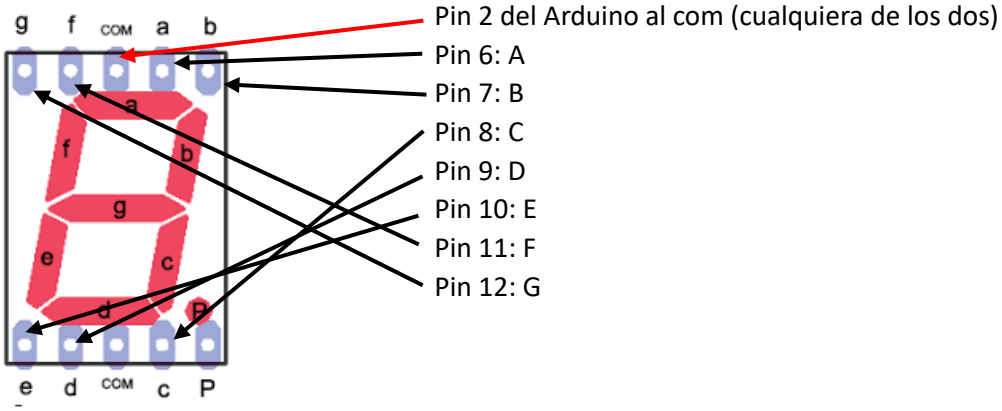


Las 10 patillas están distribuidas como se ve en la imagen, se alimenta en la central arriba o abajo con +5v
Y todas las demás son negativas.

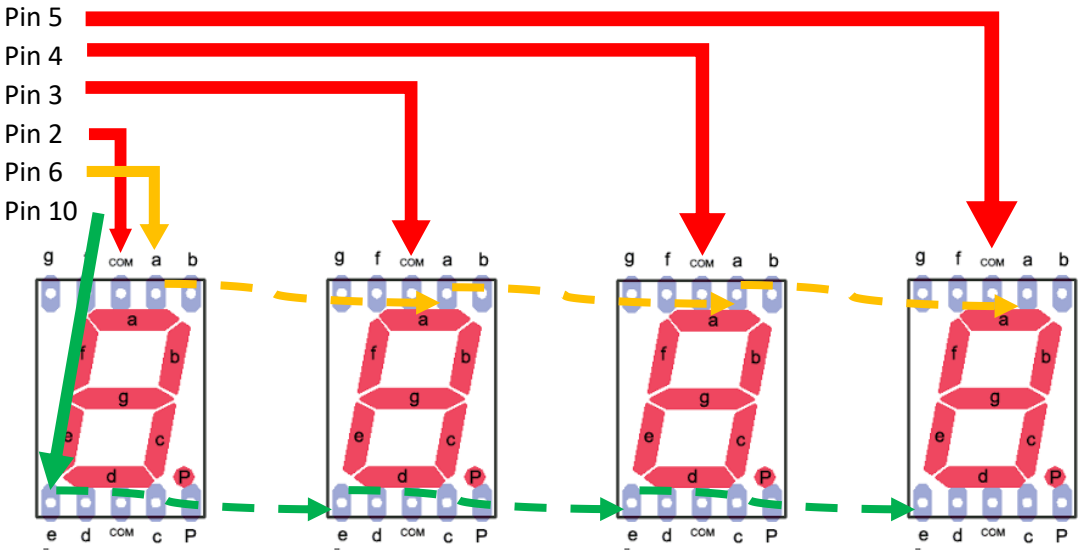
Parte 1: conexión al Arduino

Vamos a utilizar los pines del Arduino 2, 3, 4 y 5 para alimentar los 4 displays.
Y los pines 6, 7, 8, 9, 10, 11, 12 y 13* para los negativos de cada segmento.
*el 13 sería para el punto y no lo vamos a utilizar.
El listado es:
Pin 2: decenas de las horas (DH)
Pin 3: unidades de las horas (UH)
Pin 4: decenas de los minutos (DM)
Pin 5: unidades de los minutos (UM)
Pin 6: segmento A
Pin 7: segmento B
Pin 8: segmento C
Pin 9: segmento D
Pin 10: segmento E
Pin 11: segmento F
Pin 12: segmento G
Pin 13: Punto. (sin conexión)

Suponiendo que este display es DH



Hasta aquí sencillo, pero al colocar los 4 displays se nos complica porque realmente el pin 6 va a todas las A, el 7 a todas los B, y así sucesivamente… trataré de dibujarlo sin que sea un batiburrillo de flechas.



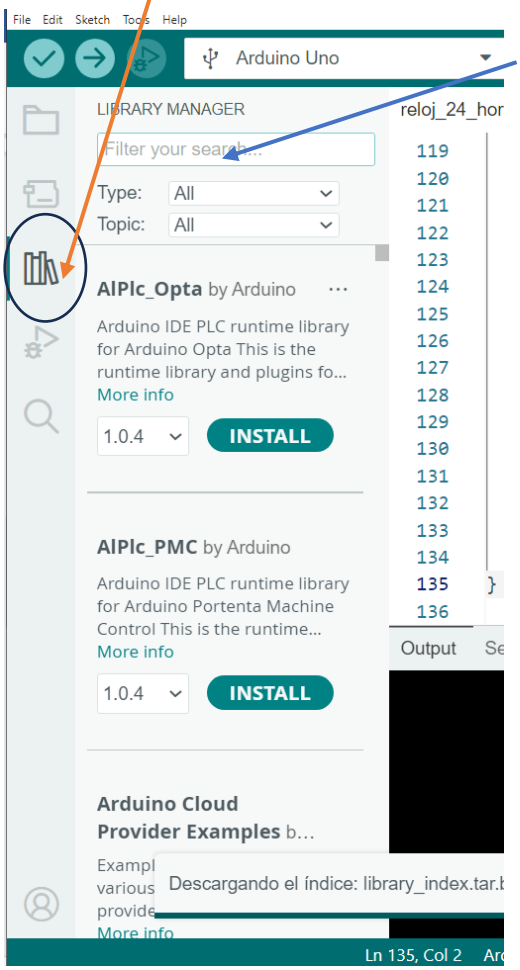
Solo puse 2 pines “negros”, porque si pongo las 28 flechas “negras” no se va a ver nada. Espero que se entienda.

Primera contradicción, os digo negros, negativo siempre negro, pero es que en este caso no es negativo, es el cable de control que enciende y apaga, por eso mejor poner colores diferentes para saber cuál es cual; si falla alguno entre 28 negros, es la muerte a pellizcos.

PARTE 2: CODIGO DE ARDUINO

El código de Arduino es supercomplicado, si lo explico línea por línea no se acaba nunca este manual, así que voy a explicar solo lo que tendréis que instalar y modificar para ajustarlo a vuestro gusto.

Primero descargar e instalar el programa Arduino IDE.
Conectar por usb el Arduino, cuando lo reconozca, instalar las 2 librerías necesarias para el código que vamos a usar.
Apartado librerías tiene forma de varios libros



Y en el cuadro de búsqueda escribimos sevseg y pinchamos en install
y bounce2 -> install

Vale ahora viene el tocho-código:

```
#include "SevSeg.h"
#include <Bounce2.h>

SevSeg sevseg;

int horas = 0;
int minutos = 0;
unsigned long previousMillis = 0;
unsigned long interval = 60000; // Intervalo de 60 segundos

const byte pinMinutos = 14; // Pin para el pulsador de ajuste de minutos
```

```
const byte pinHoras = 15; // Pin para el pulsador de ajuste de horas

Bounce botonMinutos = Bounce();
Bounce botonHoras = Bounce();

const unsigned long tiempoAvanceRapido = 200; // Tiempo en milisegundos para el
avance rápido
const unsigned long tiempoAvanceRapidoH = 400;

unsigned long tiempoAnteriorMinutos = 0;
unsigned long tiempoAnteriorHoras = 0;

void setup() {
byte numDigits = 4;
byte digitPins[] = {2, 3, 4, 5}; // 5UM,4DM,3UH,2DH
byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13}; // 6A-12G-13P
bool resistorsOnSegments = false;
byte hardwareConfig = COMMON_ANODE;
bool updateWithDelays = false;
bool leadingZeros = true;
bool disableDecPoint = false;

Serial.begin(9600);
sevseg.begin(hardwareConfig, numDigits, digitPins, segmentPins,
resistorsOnSegments,
updateWithDelays, leadingZeros, disableDecPoint);
sevseg.setBrightness(-80);

pinMode(pinMinutos, INPUT_PULLUP);
pinMode(pinHoras, INPUT_PULLUP);

botonMinutos.attach(pinMinutos);
botonMinutos.interval(50); // Establecer un intervalo de rebote de 50 ms para el
pulsador de minutos

botonHoras.attach(pinHoras);
botonHoras.interval(50); // Establecer un intervalo de rebote de 50 ms para el
pulsador de horas
}

void loop() {
unsigned long currentMillis = millis();

// Verificar si ha pasado el intervalo deseado
if (currentMillis - previousMillis >= interval) {
previousMillis = currentMillis;

// Incrementar los minutos
minutos++;

if (minutos == 60) {
minutos = 0;
horas++;
```

```
if (horas == 24) {
horas = 0;
}
}
}

// Actualizar el estado del pulsador de minutos
botonMinutos.update();
// Verificar si el pulsador de minutos ha sido presionado y no está en estado de
rebote

if (botonMinutos.fell()) {
minutos++;
if (minutos >= 60) {
minutos = 0;
}

tiempoAnteriorMinutos = currentMillis; // Reiniciar el tiempo anterior al presionar el
botón
} else if (botonMinutos.read() == LOW && currentMillis - tiempoAnteriorMinutos >=
tiempoAvanceRapido) {
minutos++;
if (minutos >= 60) {
minutos = 0;
}
tiempoAnteriorMinutos = currentMillis; // Actualizar el tiempo anterior durante el
avance continuo
}


// Actualizar el estado del pulsador de horas
botonHoras.update();
// Verificar si el pulsador de horas ha sido presionado y no está en estado de rebote
if (botonHoras.fell()) {
horas++;
if (horas >= 24) {
horas = 0;
}
tiempoAnteriorHoras = currentMillis; // Reiniciar el tiempo anterior al presionar el
botón
} else if (botonHoras.read() == LOW && currentMillis - tiempoAnteriorHoras >=
tiempoAvanceRapidoH) {
horas++;
if (horas >= 24) {
horas = 0;
}
tiempoAnteriorHoras = currentMillis; // Actualizar el tiempo anterior durante el
avance continuo
}

// Mostrar las horas y los minutos en los displays de 7 segmentos
int tiempo = horas * 100 + minutos;
```

```
sevseg.setNumber(tiempo, 3); // Tercer dígito para las horas
sevseg.setNumber(tiempo, 2); // Segundo dígito para las horas
sevseg.setNumber(tiempo, 1); // Primer dígito para los minutos
sevseg.setNumber(tiempo, 0); // Cuarto dígito para los minutos

sevseg.refreshDisplay();
}
```

UFF! Nada más y nada menos.
Bien, lo que podemos modificar está resaltado en rojo, es la velocidad a la que queremos que avancen los números cuando pulsamos el botón para ponerlo en hora. Por defecto 200 para los minutos y 400 para las horas, ¿por qué? Porque los minutos son mas que las horas entonces prefiero que vaya un poco más rápido y no estar 4 días para ponerlo en hora. Pero como siempre al gusto.
El siguiente valor que podemos editar es el brillo de los leds, si tenemos 140leds a pleno rendimiento, nos van a ver los ecologistas desde 1000km de distancia y nos van a quemar la casa. Lo vamos a regular con ese valor (+100/-100). Creo recordar.
Otro valor que se puede modificar es el de horas == 24, por 12; para convertirlo en un reloj de 12 horas. Hay que hacerlo en todos los comandos. Si no recuerdo mal aparece 3 veces en este código.

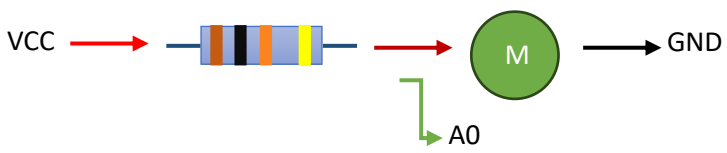
Otro detalle que podéis ver en el código es que los pulsadores están conectados al pin 14 y 15, cuando en la placa aparecen como A0 y A1. Si os diese error puede que haya que modificarlo, pero a mí me los reconoce mejor así.

Pues si no me olvidé de nada, solo nos falta el tema: pulsadores.
Los pulsadores van una resistencia de “Pullup”. No voy a explicar la parte técnica de como funciona un pulsador en electricidad que sería lo más fácil, para después explicar la diferencia en electrónica y bla bla bla. No es el fin de este manual. Es de Arduino. Aunque se dé algún apunte en cierto momento.
Solo decir, que si dejamos un pulsador abierto, es decir conectado a la nada, en electrónica da problemas.
Para evitar esa conexión a la nada, se usan las resistencias pull-up y pull-down.
Como en el código le hemos puesto:

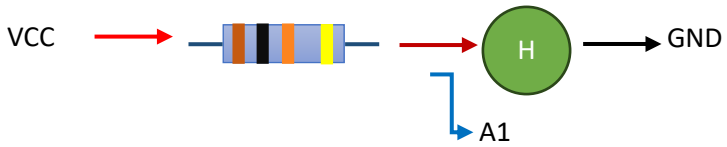
```
pinMode(pinMinutos, INPUT_PULLUP);
pinMode(pinHoras, INPUT_PULLUP);
```

pues habrá que conectar los pulsadores con sus resistencias pull-up.

Se conecta la alimentación vcc a un lado la resistencia de 10K, el extremo de la resistencia al pulsador, y el otro extremo del pulsador a negativo. Y en la unión de la resistencia con el pulsador, sacamos la señal para A0.



Lo mismo con el pin A1 (horas).

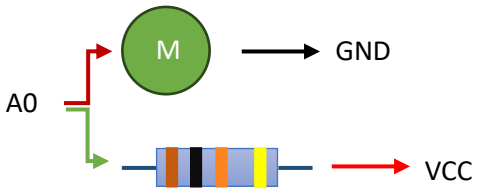


Las flechas serían los cables y los pongo de diferentes colores para saber cuál es el que entra en M el que sale, el que entra en R y el que sale. Si hay problemas se identifica mucho más rápido. Truquillo!
Solo respeto dos colores: rojo // negro negativo

Rojo=positivo=+5v=Vcc= y demás variantes
Negro=negativo=masa=tierra=GND= y demás

Si no os gusta la teoría, saltad a la parte 3…
Vamos a explicar un poquitín con este ejemplo, la necesidad de la resistencia “pull-up/down”.
La función del pulsador M es avanzar los minutos para poner el reloj en hora.
El funcionamiento nos dice que hay que meter un 0v-negativo-low (como más os guste) en el pin A0, para que empiece a contar.
Perfecto! La lógica humana diría: “fácil. Un lado del pulsador a GND y el otro al pin A0. Cuando pulso lo estoy conectando a GND, por lo tanto le meto un 0, y ya funciona.”
Sí y no.
Cuando se pulsa, sí que va a funcionar, pero mientras no está pulsado ese pin A0 está conectado a la nada, ni a negativo, ni a positivo, ni a ningún valor.
Y esto puede volver loco a cualquier aparato electrónico.

Vamos a ver ahora que sucede cuando metemos esa resistencia



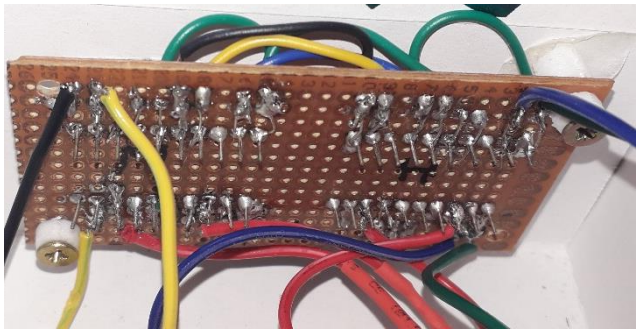
Lo que provoca esa resistencia es que A0 tenga dos caminos por los que ir. Siempre va a escoger el más cómodo, el que menos resistencia tenga.
En el primer caso, pulsador abierto (resistencia enorme-infinita). Por un lado tiene una resistencia de 10K y por el otro infinita, se va por la resistencia. Ese pin A0 está conectando a un valor vcc menos la caída en la resistencia, pero siempre positivo.
Cuando pulsamos, la resistencia pasa a ser prácticamente cero. Entonces por un lado tiene 10K y por el otro 0K. se va por el pulsador a masa.

Si todavía no esta claro vamos con un ejemplo de la vida.
Si vamos por una carretera, llegamos a un cruce y por la derecha tardamos 10Minutos (R10K) en llegar a casa y por la izquierda menos de 1 minuto (pulsado). Todos iríamos por la izquierda, pero, si hay una valla cortando la carretera izquierda (pulsador abierto) pues tenemos que ir por la derecha 10minutos.
En cuanto quiten la valla (pulsado) todos por la izquierda.

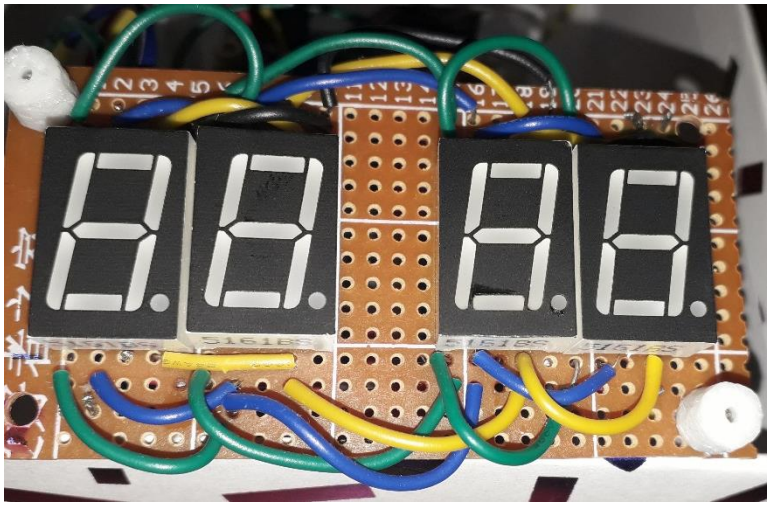
PARTE 3: SOLDADURAS

En la placa multiperforada colocamos 2 displays pegados le damos una separación de 1cm aprox y colocamos los otros 2 también pegados.
Algo así:

Dejamos por lo menos 3 agujeritos por arriba y 3 por abajo porque hay que soldar el cable que viene del pin y el otro que hace de puente.

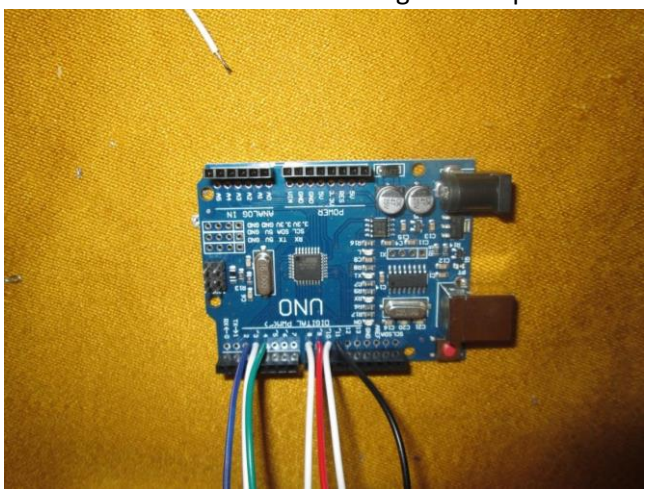


Soldaduras y cables pin



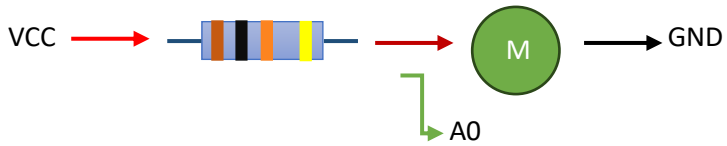
Puentes

Soldar en el Arduino no tiene ninguna complicación.

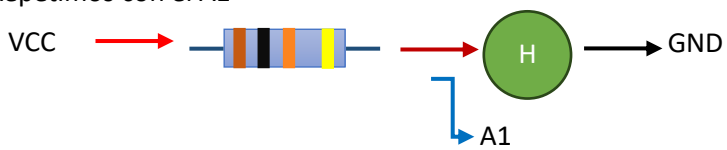


Cada color que hayamos decidido en su pin…

Ahora en otro trocito de placa multiperforada, soldamos las resistencias 2 de 1K. por un lado a masa las dos (cables negros) y por otro le dejamos un cable largo, uno de cada color, por ej: azul minutos, amarillo horas) que van a ir a los pulsadores y todavía no hemos decidido donde van.
Tenemos dos opciones o le hacemos dos agujeros a la tapa de la caja y van por fuera. O les colocamos unos soportes dentro de la caja, para que no se vean por fuera y parezca una caja de cartón al uso.
Yo le hice un soporte en 3D para que quedasen por dentro, pero también se puede hacer en madera.
Una vez colocados en su sitio el cable del pin A0 va al pulsador y el otro de la resistencia, al otro extremo del pulsador

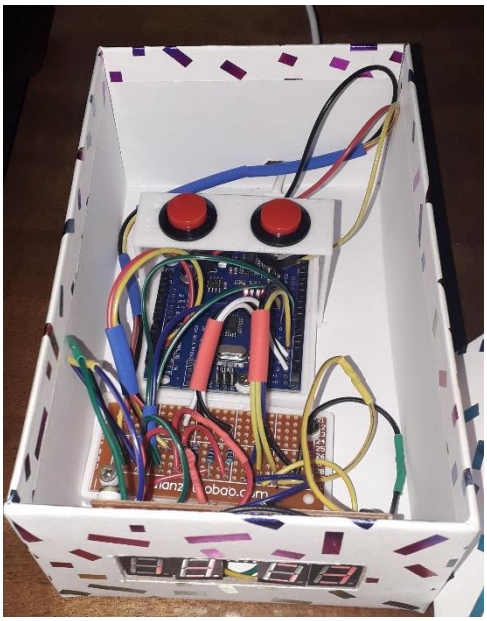


Repetimos con el A1



Resumiendo, las dos resistencias pueden ir unidas por un lado, y a positivo.
Y los dos pulsadores pueden ir unidos por un lado, y a negativo

A mí me quedó así:



Le puse al arduino una base, y pegado, por fuera de la base, un puente elevado para los pulsadores.
Tambien veis que hay una placa pequeña en la que van las resistencias de 1K pero le puse mas resistencias para proteger los displays, porque no sabia que el codigo de Arduino me permitia modificar la intensidad de los leds y no hacian falta esas resitencias.