Parte 2

Lección 2

RGB LED

Resumen

RGB LED son una forma divertida y fácil para agregar color a sus proyectos.
Puesto que es como regular 3 LED en uno, el uso y conenxión no es muy diferente.

Vienen en 2 versiones: ánodo común o cátodo común.
Ánodo común utiliza 5V en el pin común, mientras que el cátodo común se conecta a tierra..
Como con cualquier LED, tenemos que conectar algunas resistencias en línea (3 total) así que podemos limitar la corriente absorbida.

En nuestro bosquejo, se comienzan con el LED en el estado de color rojo, entonces se descolora a verde, luego se descolora azul y finalmente hacia el color rojo.

Haciendo esto que nos pasará por la mayor parte del color que se puede lograr.

Component Required:

(1) x Elegoo Uno R3

(1) x protoboard de 830 puntos de amarre

(4) x M M de cables (cables de puente de macho a macho)

(1) x RGB LED

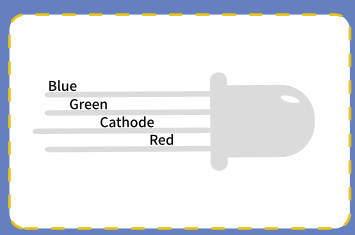
(3) x resistencias de 220 ohmios

Introducción del componente

RGB:

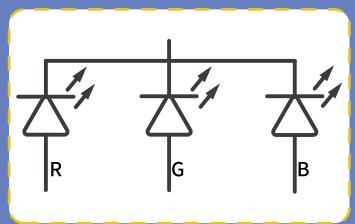
A primera vista, LEDs RGB (rojo, verde y azul) sólo parecen regular LED. Sin embargo, dentro del paquete del LED generalmente, hay realmente tres LEDs, uno rojo, uno verde y sí, uno azul. Controlando el brillo de cada uno de los LEDs individuales, usted puede mezclar prácticamente cualquier color quedesee.

Mezclamos colores del mismo modo que sería mezclar pintura en una paleta - ajustando el brillo de cada uno de los tres LEDs. La manera dura para hacer esto sería usar valor diferentes resistencias (o resistencias variables) como hicimos con el en la lección 2, pero eso es un mucho trabajo! Afortunadamente para nosotros, Kit UNO R3 tiene una función analogWrite que puede utilizar con pines marcados con un ~ a la salida de una cantidad variable de energía los LEDsapropiados.



El LED RGB tiene cuatro conductores. Hay un cable a la conexión positiva de cada uno de los LEDs individuales dentro del paquete y un patilla única que está conectado a los tres lados negativos de los LEDs.

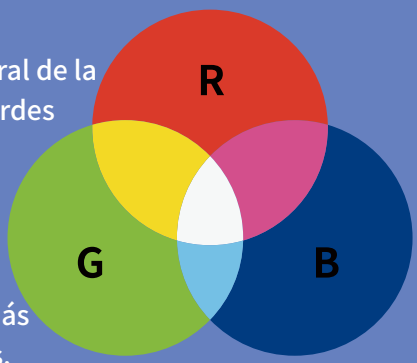
La común conexión negativa del paquete LED es el segundo pasador de la parte plana. También es el más largo de las cuatro patas y se conectarán a la tierra.



Cada LED dentro del paquete requiere su propio resistor de 220Ω para prevenir demasiada corriente que fluye a través de él. Los tres conductores del positivo de los LEDs (uno rojo, uno verde y uno azul) están conectados a los pines de salida UNO con estas resistencias.

COLOR:

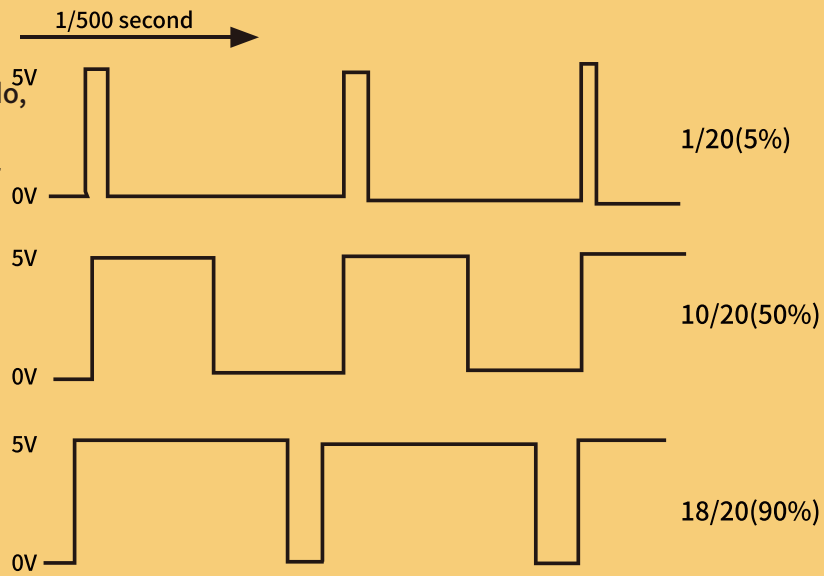
- **La** razón por la que usted puede mezclar cualquier color usted tiene gusto variando las cantidades de rojo, verde y azul de la luz es que el ojo tiene tres tipos de receptor de luz (rojo, verde y azul). Su ojo y el cerebro procesan las cantidades de rojo, verde y azul y conviertan en un color delespectro.
- **En** cierto modo, mediante el uso de los tres LEDs, estamos jugando un truco en el ojo. Esta misma idea se utiliza en televisores, donde la pantalla LCD tiene puntos de color rojo, verde y azul junto a unos a otros que componen cada píxel.
- **Si** establece el brillo de todos los tres LEDs al ser el mismo, el color general de la luz será blanco. Si apagamos el LED azul, para que sólo los LEDs rojo y verdes son el mismo brillo, la luz aparecerá amarillo.Que podemos controlar el brillo de cada una de las partes de rojas, verdes y azules del LED por separado, lo que es posible mezclar cualquier color que nos gusta.



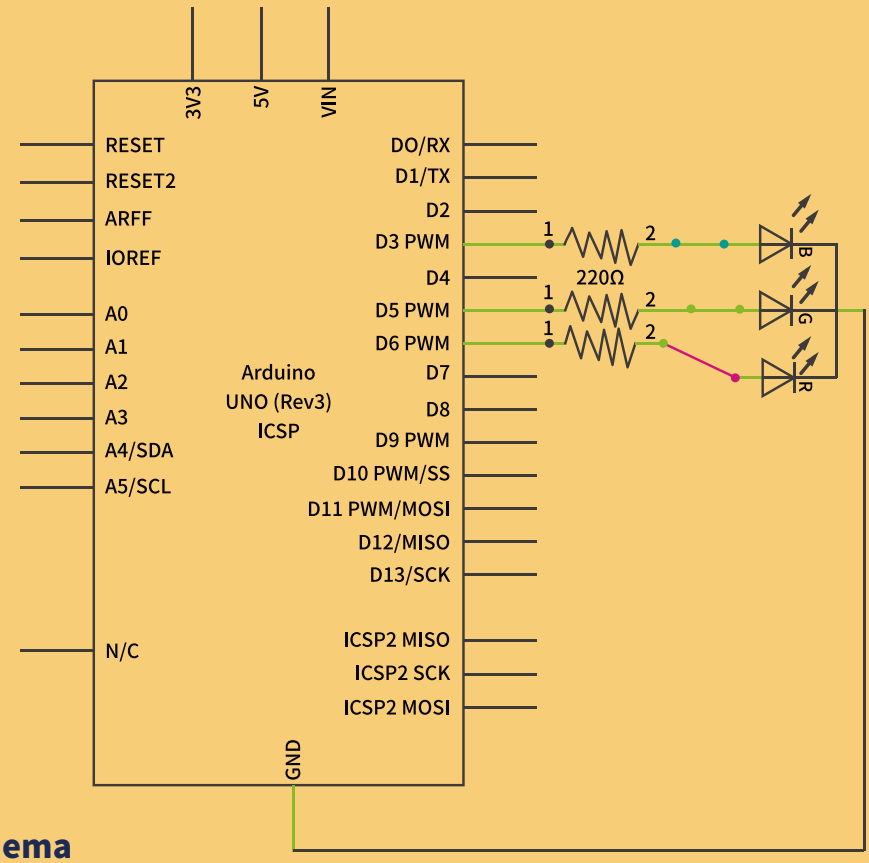
- **Negro** no es tanto un color como una ausencia de luz. Por lo tanto, lo más cercano que podemos llegar a negro con el LED es apagar los tres colores.

Teoría (PWM)

- Modulación de ancho de pulso (PWM) es una técnica para el control de potencia. También utilizamos aquí para controlar el brillo de cada uno de los LEDs.
  El siguiente diagrama muestra que la señal de uno de lo PWM pins en la UNO.



- Aproximadamente cada 1/500 de segundo, la salida PWM producirá un pulso. La duración de este pulso es controlada por la función 'analogWrite'. Así 'analogWrite(0)' no producirá ningún pulso en todo 'analogWrite(255)' producirá un pulso que dura todo el camino hasta el pulso siguiente vencimiento, para que la salida es en realidad todo el tiempo.
- Si especificamos un valor en el analogWrite que está en algún lugar entre 0 y 255, se producir un pulso. Si el pulso de salida es alto para el 5% del tiempo, entonces lo que nosotros estamos manejando sólo recibirá el 5% de potencia.
- Si, sin embargo, la salida es 5V para el 90% del tiempo, la carga recibirá el 90% de la potencia entregada a él. No podemos ver los LEDs de encendido y apagado a esa velocidad, así que, sólo parece que está cambiando el brillo.



Conexión Esquema

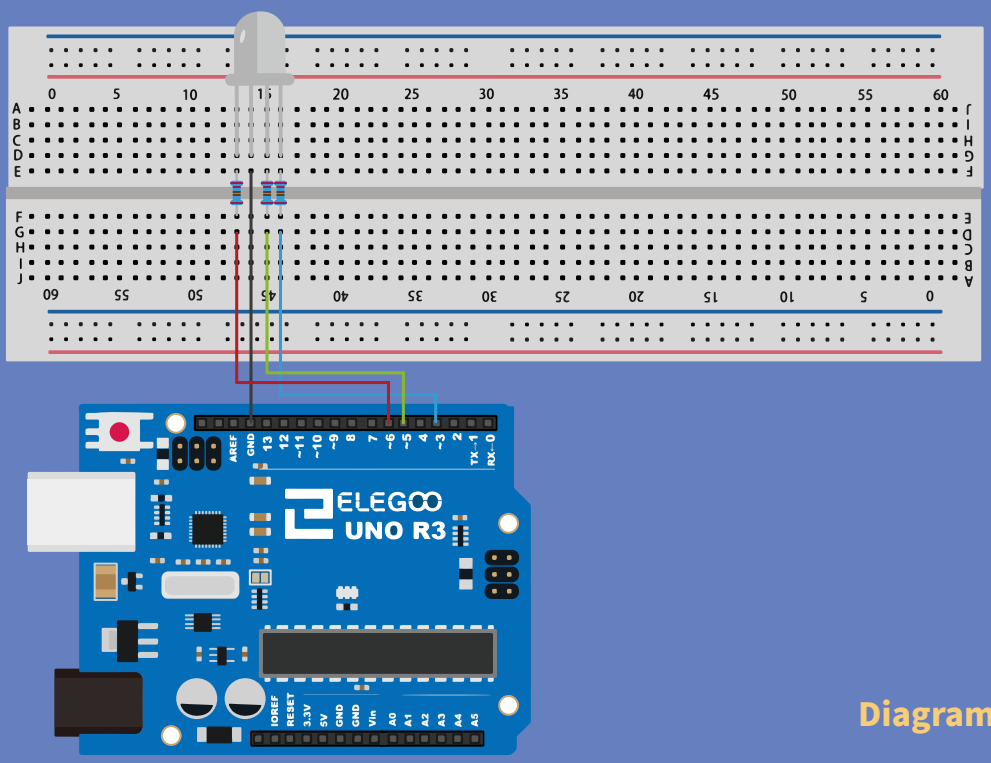
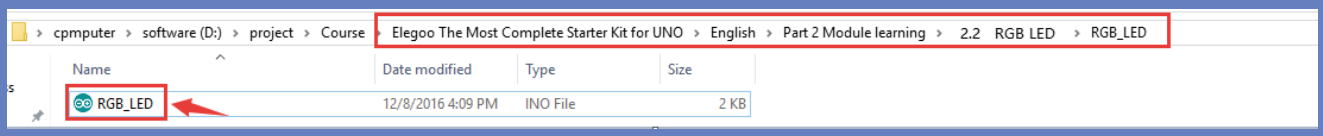


Diagrama de cableado

Code

- Después del montaje, abra la carpeta del Proyecto en la ruta: \ Elegoo The Most Complete Starter Kit for UNO\English\Part 2 Module Learning\2.2 RGB LED, y haga clic en SUBIR para cargar el programa.



- El dibujo comienza especificando que los pernos van a utilizar para cada uno de los colores:

```
// Define Pins
#define BLUE 3 // The compiler will replace any mention of ledPin with the value 3 at compile time.
#define GREEN 5
#define RED 6
```

#define constantName value

Es un útil componente de C ++ que permite al programador asignar un nombre a un valor constante antes de compilar el programa. Las constantes definidas en arduino no ocupan espacio de memoria en el programa. El compilador reemplazará las referencias a estas constantes con el valor definido en la compilación.

Esto puede tener algunos efectos secundarios no deseados si, por ejemplo, un nombre constante que había sido definido, es incluido en algún otro nombre de constante o variable. En ese caso, el texto sería reemplazado por el #número definido (o texto).

Parámetros

constantName: El nombre de la macro a definir.

value: El valor a asignar a la macro.

Notas y advertencias

- No hay punto y coma después de la declaración #define. Si lo incluye, el compilador arrojará errores críticos más abajo en el proceso.
- **#define ledPin 3; // Esto es un error**
- Del mismo modo, incluir un signo igual después de la declaración #define también generará un error crítico del compilador, más adelante en el proceso.
- **#define ledPin = 3 // Esto también es un error**

El siguiente paso es escribir la función 'setup'. Como hemos aprendido en lecciones anteriores, la función de configuración se ejecuta una sola vez después de que el Arduino se ha restablecido. En este caso, todo lo que tiene que hacer es definir los tres pines que estamos utilizando como salidas.

pinMode(pin, mode) Configura el pin especificado para que se comporte como pin de entrada o de salida.

A partir de Arduino 1.0.1, es posible habilitar las resistencias pullup internas con el modo INPUT_PULLUP. Además, el modo INPUT inhabilita explícitamente los pullups internos.

```
void setup()
{
pinMode(RED,OUTPUT);
pinMode(GREEN,OUTPUT);
pinMode(BLUE, OUTPUT);
digitalWrite(RED,HIGH);
digitalWrite(GREEN,LOW);
digitalWrite(BLUE, LOW);
```

Parámetros

pin: El número de pin de la placa Arduino del que se quiere establecer el modo.

mode: INPUT, OUTPUT o INPUT_PULLUP. Consulte la página “Pines digitales” para obtener una descripción más completa de la funcionalidad.

Int: Son tipos de datos. Enteros es su tipo de datos principal para el almacenamiento de números.

```
// define variables
int redValue;
int greenValue;
int blueValue;
```

En el Arduino Uno (y otras placas basadas en ATmega) un dato “INT” almacena un valor de 16 bits (2 bytes). Esto produce un rango de -32,768 a 32,767 (un valor mínimo de -2^15 y un valor máximo de (2^15) - 1). En las placas basadas en Arduino Due y SAMD (como MKR1000 y Zero), un dato de tipo “INT” almacena un valor de 32 bits (4 bytes). Esto produce un rango de -2,147,483,648 a 2,147,483,647 (un valor mínimo de -2^31 y un valor máximo de (2^31) - 1).

“INT” almacena números negativos con una técnica llamada (matemática complementaria de 2). El bit más alto, a veces denominado bit de “signo”, marca el número como un número negativo. El resto de los bits se invierten y se agrega 1.

Sintaxis

```
int var = val;
```

Parámetros

var: nombre de la variable.

val: el valor que asigna a esa variable.

- Antes de echar un vistazo a la función 'loop', veamos la última función en el proyecto.

Las variables de definición

```
redValue = 255; // choose a value between 1 and 255 to change the color.
greenValue = 0;
blueValue = 0;
```

- Esta función tiene tres argumentos, uno para el brillo de los LEDs rojos, verdes y azules. En cada caso de que el número será en el rango 0 a 255, donde 0 significa apagado y 255 significa brillo máximo. La función entonces llama 'analogWrite' para ajustar el brillo de cada LED.

Trate de añadir algunos colores de los tuyos el dibujo y ver el efecto en tu LED.

```
for (int i = 0; i < 255; i += 1) // fades out red bring green full when i=255
{
redValue -= 1;
greenValue += 1;
// The following was reversed, counting in the wrong directions
// analogWrite(RED, 255 - redValue);
// analogWrite(GREEN, 255 - greenValue);
analogWrite(RED, redValue);
analogWrite(GREEN, greenValue);
delay(delayTime);
}
```

for

[Estructura de control]

Descripción

La declaración “FOR” se usa para repetir un bloque de declaraciones encerradas entre llaves. Generalmente se usa un contador para incrementar y terminar el ciclo. La declaración “FOR” es útil para cualquier operación repetitiva, y es usada a menudo en combinación con matrices para operar en colecciones de datos/pines.

Sintaxis

```
for ( inicialización;  condición; incremento) {
  // declaracion(es);
}
```

Parámetros

initialization: Se ejecuta primero y exactamente una vez.

condition: cada vez que da la vuelta el bucle, se comprueba la condición; si es cierto, el bloque de instrucción y el incremento se ejecuta, entonces la condición se comprueba nuevamente. Cuando la condición se vuelve falsa, el ciclo termina.

increment: Se ejecuta cada vuelta del ciclo cuando la condición es verdadera.

=

[Operadores aritméticos]

Descripción

El signo igual solo ‘=’ en el lenguaje de programación C ++ se llama operador de asignación. Tiene un significado diferente que en clase de álgebra, donde indica una ecuación o igualdad. El operador de asignación le dice al microcontrolador que evalúe cualquier valor o expresión que esté en el lado derecho del signo igual y lo almacene en la variable a la izquierda del signo igual.

Notas y advertencias

La variable en el lado izquierdo del operador de asignación (signo ‘=’) debe ser capaz de mantener el valor almacenado en él. Si no es lo suficientemente grande para contenerlo, el valor almacenado en la variable será incorrecto.

No confunda el operador de asignación [=] (signo igual solo) con el operador de comparación [==] (signos de doble igual), que evalúa si dos expresiones son iguales.

+= / -=

[Operadores compuestos]

Descripción

Son una abreviatura adecuada para realizar sumas/restas en una variable con otra constante o variable.

Syntax

```
x += y; // equivalent to the expression x = x + y;
x -= y; // equivalent to the expression x = x - y;
```

Parámetros

x: variable. Tipos de datos permitidos: INT, FLOAT, DOUBLE, BYTE, SHORT, LONG.

y: variable o constante. Tipos de datos permitidos: INT, FLOAT, DOUBLE, BYTE, SHORT, LONG.