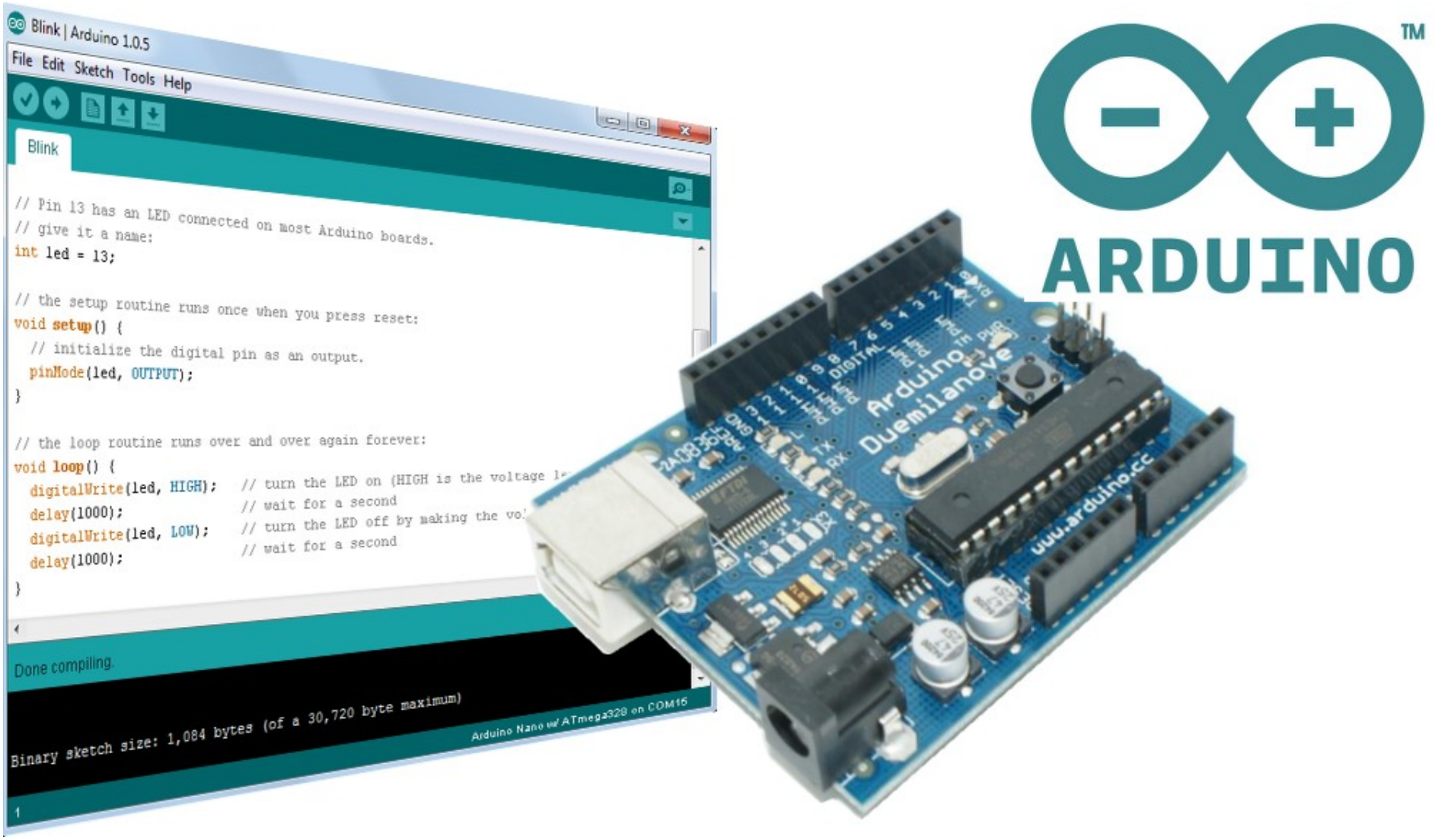
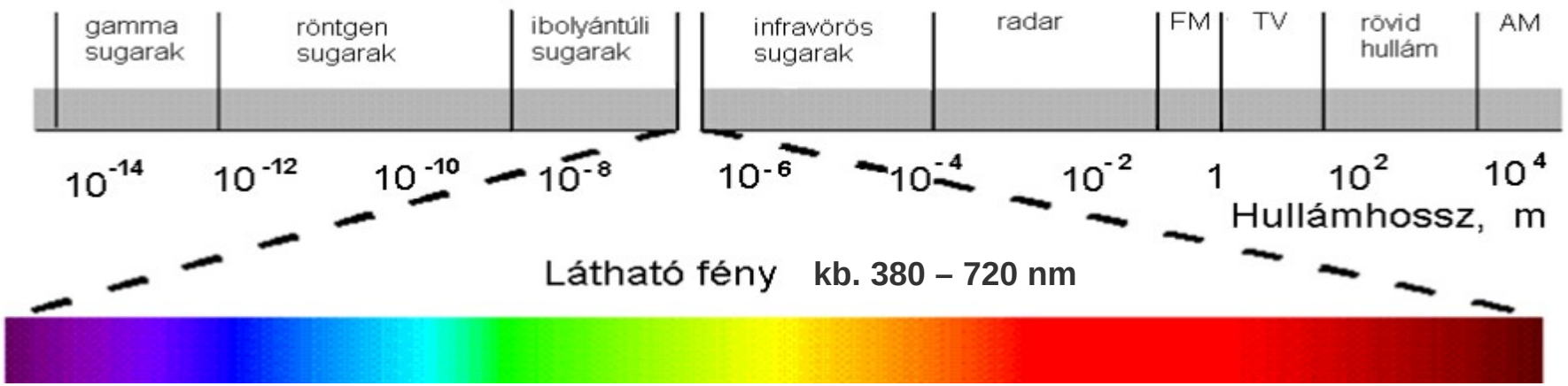
Arduino tanfolyam kezdőknek és haladóknak



15. Analóg és digitális RGB LED-ek vezérlése

Mit nevezünk színnek?

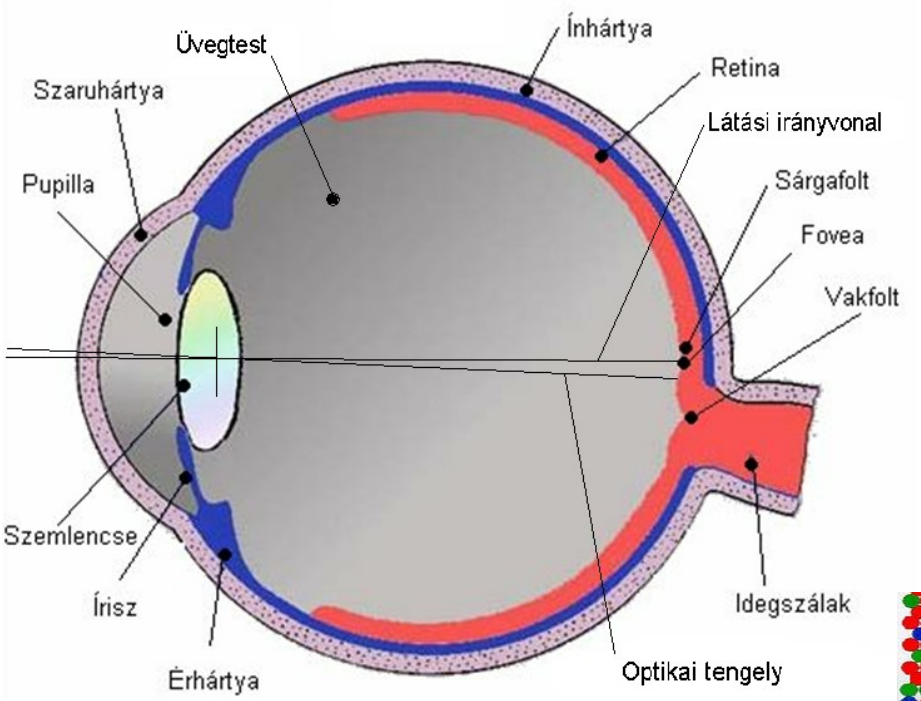
- Forrás: **Műszaki Optika** (Ábrahám Gy., Wenzelné Gerőfy K., Antal Á., Kovács G.)

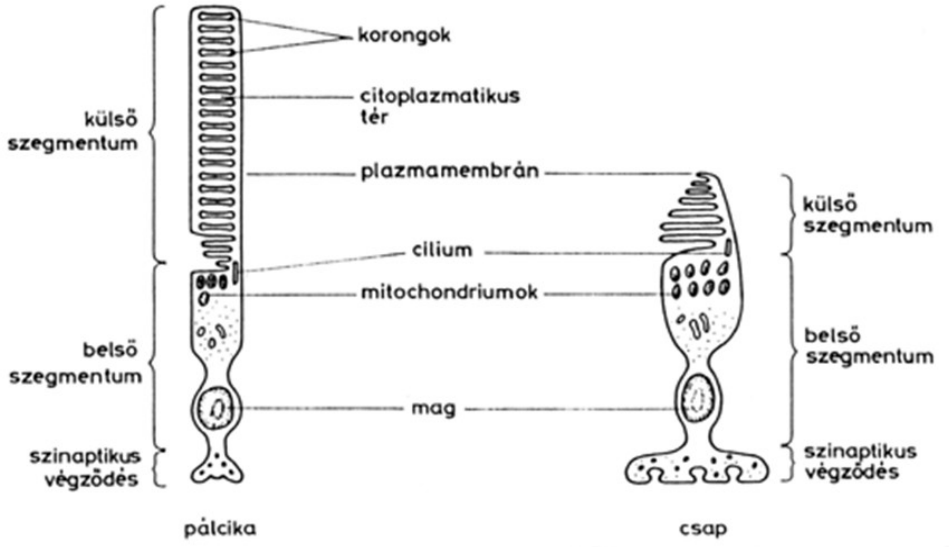


- A szín a szemünkbe érkező fénynek azon tulajdonsága, hogy különböző hullámhosszúságú összetevői nem azonos intenzitásúak
- A magyar szabvány (**MSz 9620**) definíciója szerint a szín *„A látható sugárzásnak az a jellemzője, amelynek alapján a megfigyelő a látótér két azonos méretű, alakú és szerkezetű, egymáshoz csatlakozó része között különbséget tud tenni, és ezt a különbséget a megfigyelt sugárzások spektrális eloszlásának eltérése okozhatja.”*

Az emberi szem és a színlátás

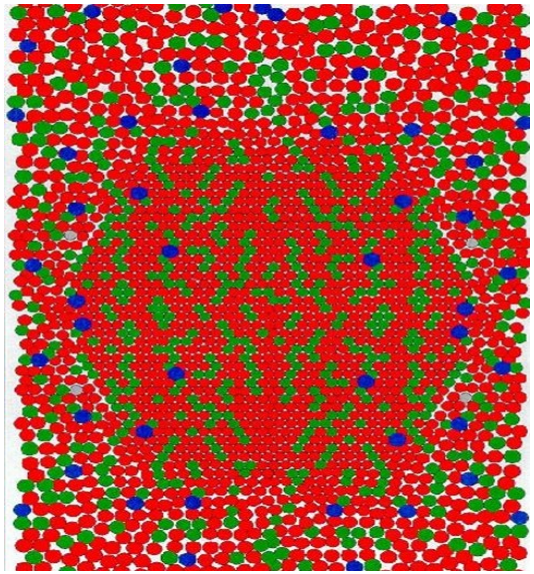
- A szem érzékelő része a **retina** amely pálcikákat és csapokat tartalmaz

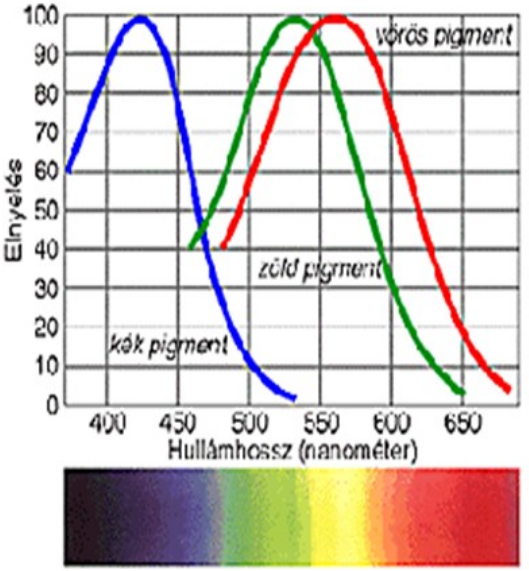




- A színek érzékelése az ún. csapok segítségével történik, melyekből kék, zöld, és vörös fényre érzékenyek találhatók az ábrán látható eloszlásban

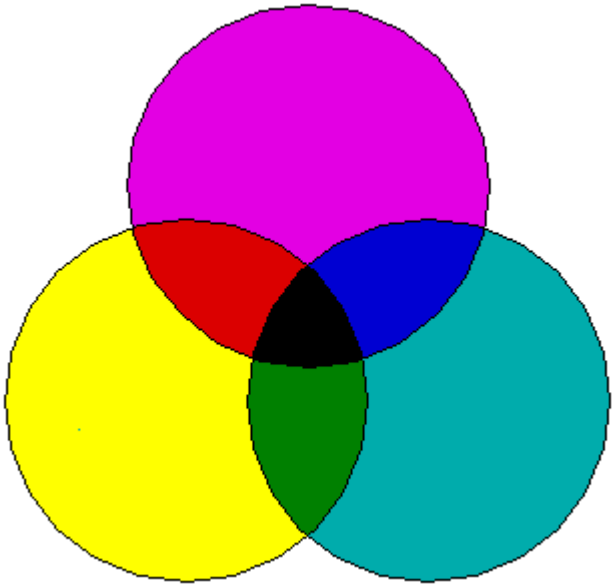
Forrás: **Műszaki Optika**

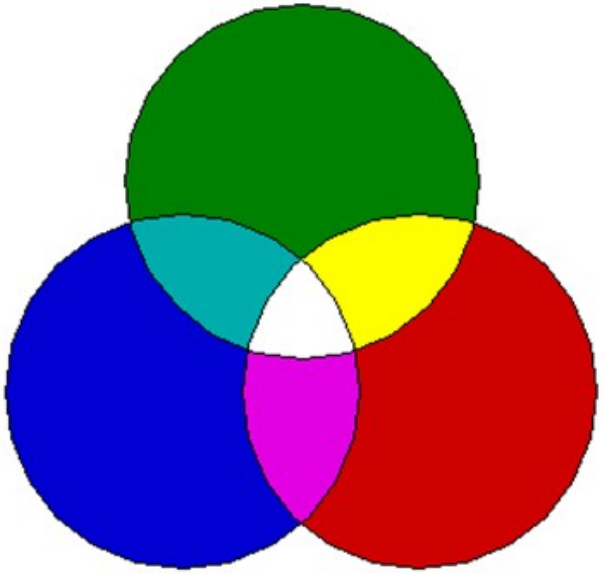




Színkeverés

- Két módszere ismert:
  - **szubtraktív színkeverés** az emberi szemtől függetlenül, a fények természetes spektrális módosulása útján jön létre (pl. nyomdászat)
  - **additív színkeverés** az emberi látórendszerben alakul ki (pl. színes TV)



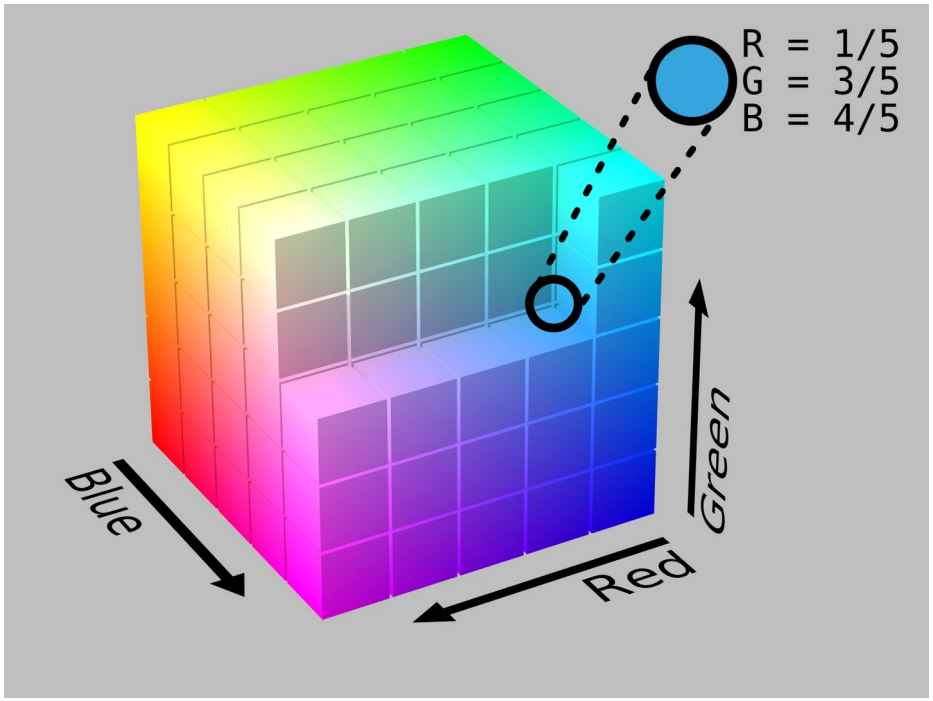


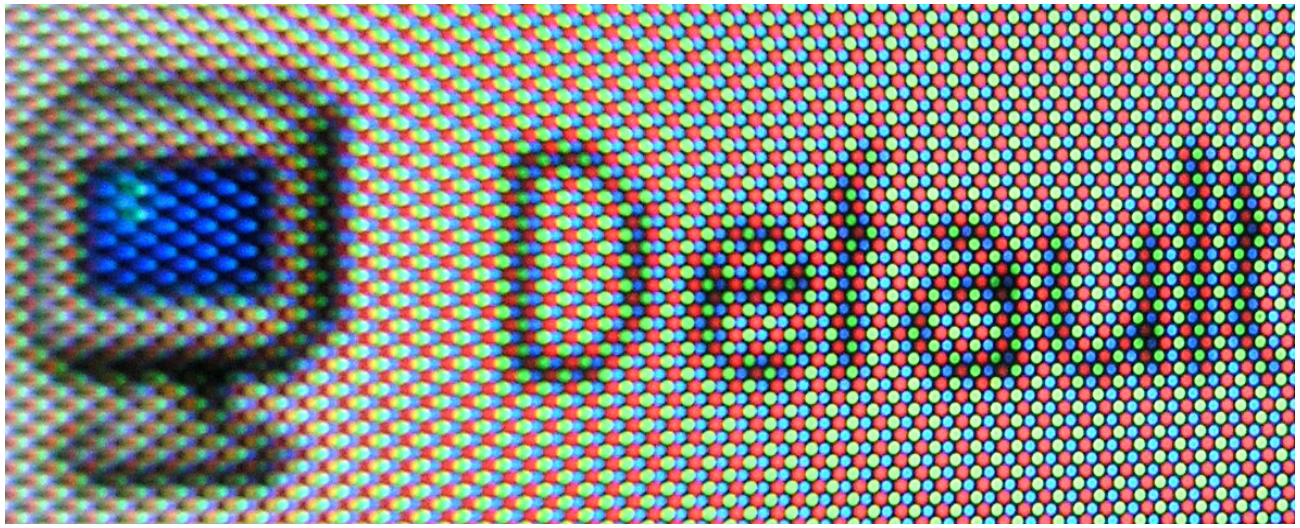
- Szubtraktív színkeverés
  (**CYMK modell,** a tinta gyengíti a fényvisszaverést bizonyos hullámhosszon)
- Additív színkeverés
  (**RGB modell,** különböző hullámhosszúságú fénysugarakkal)

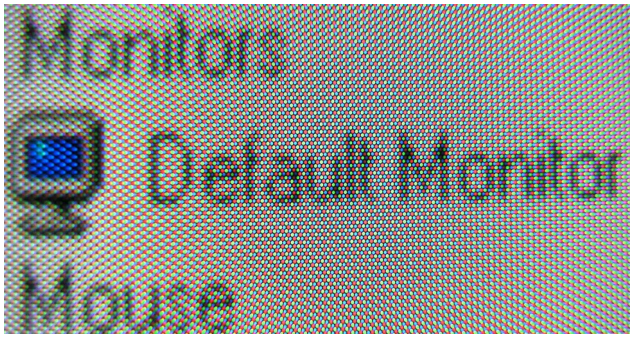
Forrás: **Műszaki Optika**

Az RGB színtér modell

- Az **RGB** additív színtér modellben vörös, zöld és kék összetevők keverésével próbáljuk leírni, vagy előállítani a színeket
- Minden színárnyalat az R, G és B ”koordinátákkal” jellemezhető
- Az alábbi képen egy színes monitor fényképe látható kinagyítva







[Wikipedia: RGB color model](), [Color spaces w RGB primaries]()

A HSV (vagy HSI) színtér modell

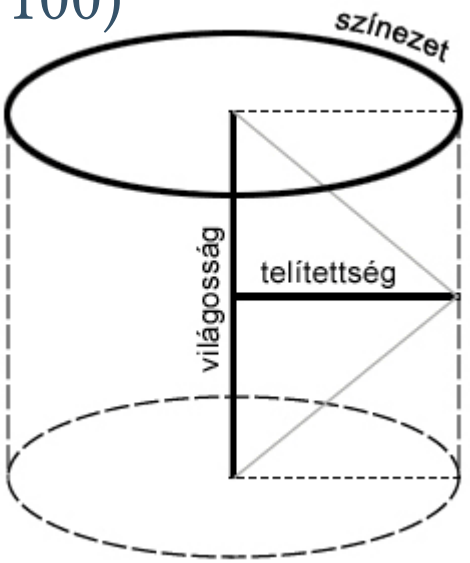
- A színérzetek jellemzőit hengerkoordináta rendszerben is ábrázolhatjuk, ahol a Φ szög a színezet (*Hue*, 0 – 360°), a tengelytől való távolság a telítettség (*Saturation*, 0 – 100), a tengelymenti távolság pedig a világosság (*Value, Intensity*, 0 – 100)
- A henger inkább kettőskúp, mert fekete, ill. fehér szín esetében nincs értelme telítettségről, vagy színezetről beszélni
- Összefüggés a HSI és az RGB értékek között:

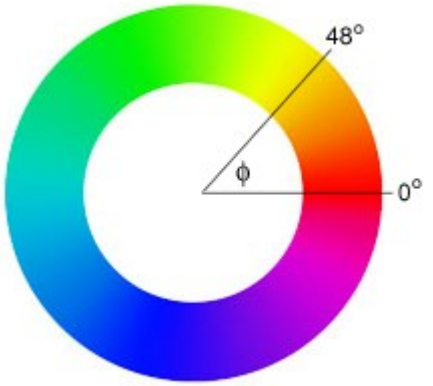
$$I = \frac{R+G+B}{3}$$

$$S = 1 - \frac{3}{(R+G+B)} \min(R,G,B)$$

$$H = \cos^{-1}\left(\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \qquad \text{assuming } G > B$$

If $B > G$, then $H = 360 - H$.





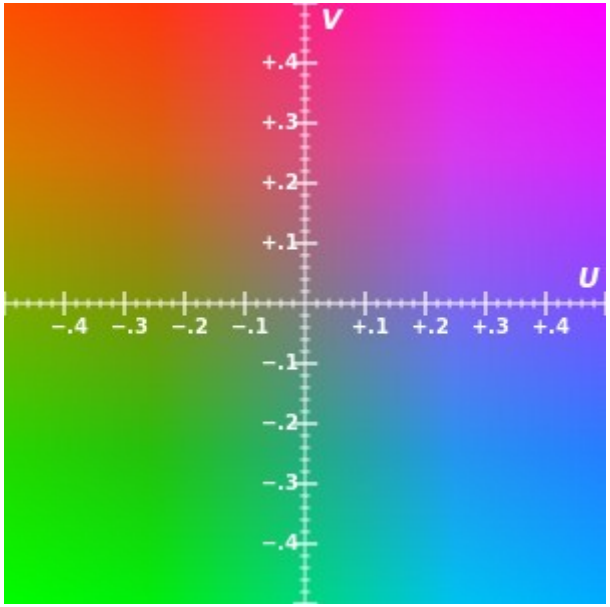
Források:

Földvári Melinda: Szín + Kommunikáció

Wikipedia: RGB color model

A YUV színtér

- Az RGB a legalkalmasabb a monitorokhoz, megjelenítőkhöz,
- A YUV színtér modell tipikus felhasználási területe a színes képek és mozgóképek feldolgozása, tömörítése, továbbítása. Először az analóg TV sugárzásnál használták a fekete-fehér adással való kompatibilitás megőrzésére
- A YUV három komponenséből az első a világosság jel (Luma), U a kék, és V a piros vetület
- A YUV és az RGB értékek közötti átszámítás az alábbi mátrixműveletekkel végezhető el:

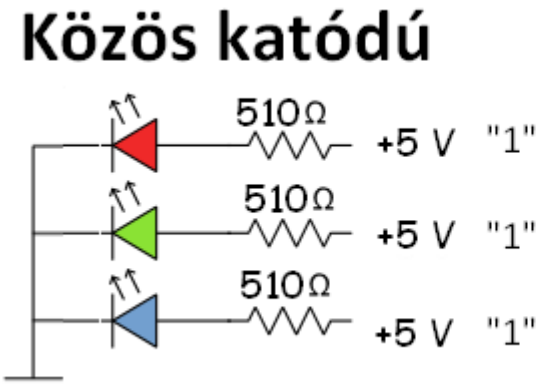


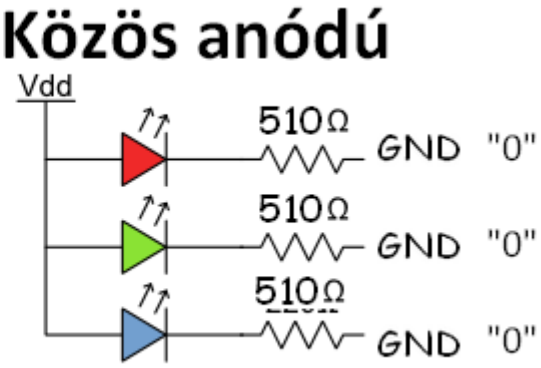
The conversion between RGB and YUV for HDTV (BT.709).

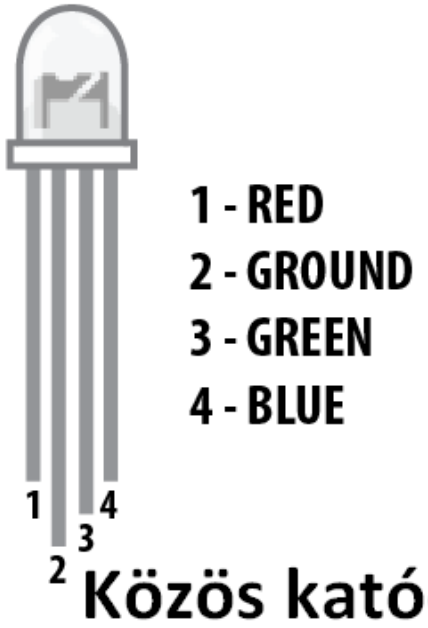
$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}; \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

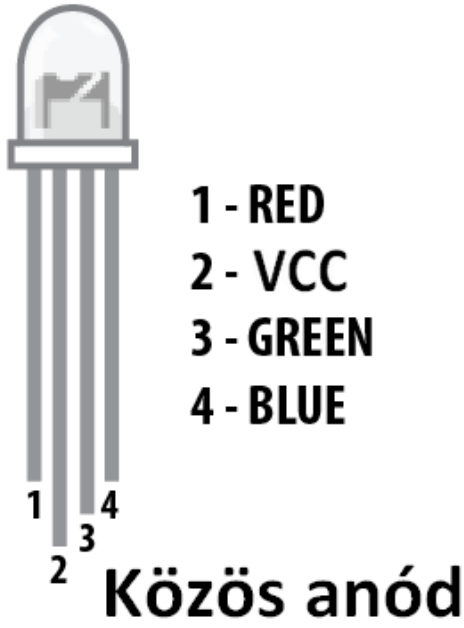
RGB LED-ek

- Az RGB LED három, különböző színű (vörös, zöld, kék) LED-et tartalmaz egy közös tokban, s vagy az anódok, vagy a katódok össze vannak kötve
- Az áramkorlátozásról nekünk kell gondoskodni (pl. ellenállás)
- **Közös katód** esetén az anódokat magas szintre kell húzni
- **Közös anód** esetén a katódokat lefelé kell húzni (inverz logika)
- Kísérleteinknél elég lesz színenként 5 – 10 mA



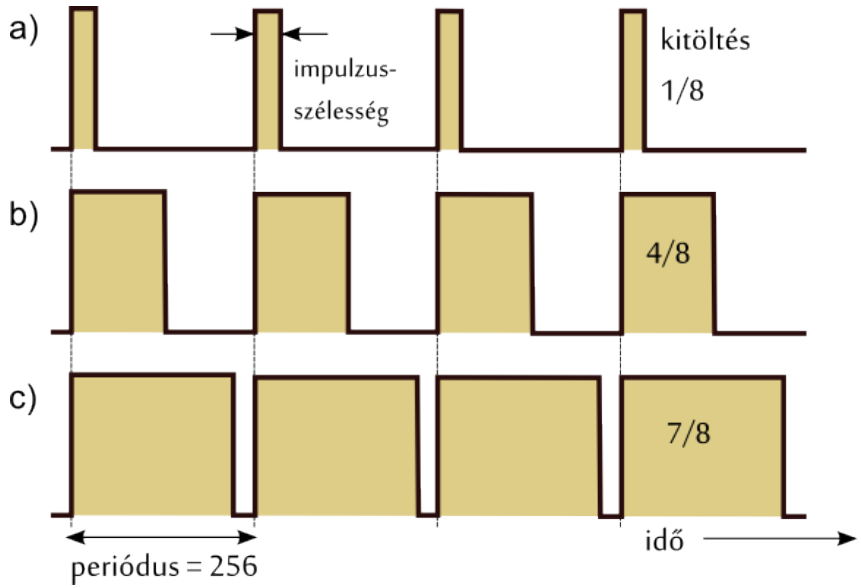






RGB LED „analóg” vezérlése

- **PWM** = pulse width modulation (impulzus-szélesség moduláció)
- A frekvencia állandó
- A kitöltés változtatható 0- 255 között
- `analogWrite(`*pin,data*`)` függvény:
  - csak bizonyos lábakra használható
  - data = 0 – 255 közötti szám (kitöltés)
- Arduino Uno/Nano (**Atmega 328P**):



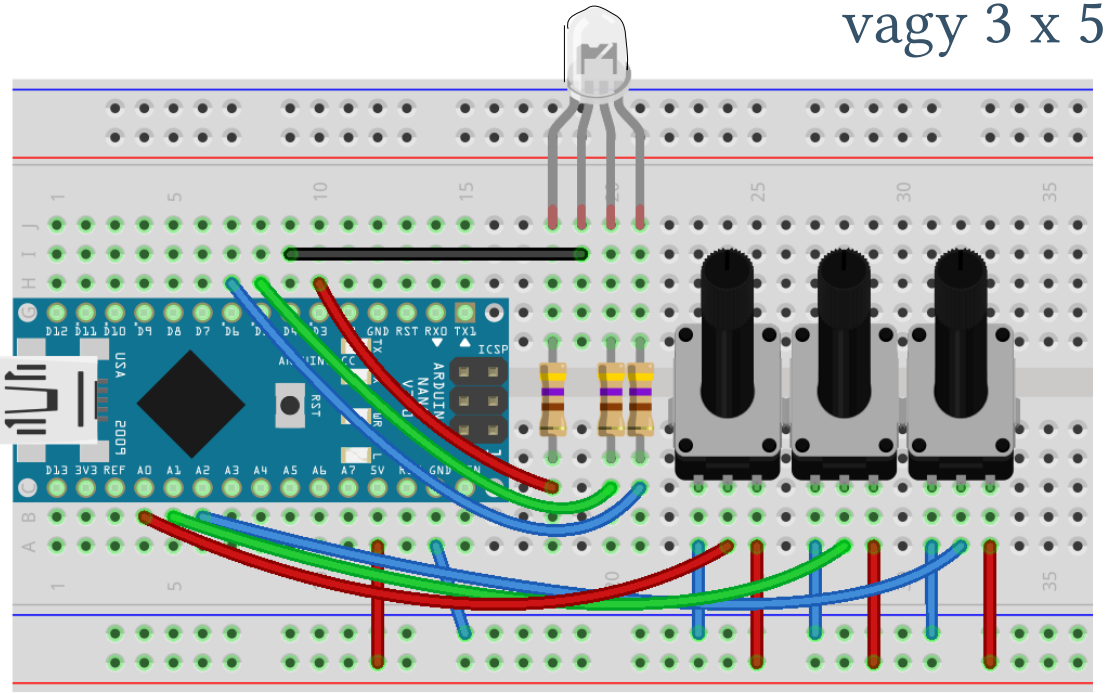
Időzítő	Csatorna	Kivezetés	Frekvencia
Timer0	OC0A, OC0B	6, 5	980 Hz
Timer1	OC1A, OC1B	9, 10	490 Hz
Timer2	OC2A, OC2B	11, 3	490 Hz

PWM = kvázi analóg vezérlés, amely a kitöltés változtatásával a teljesítményt szabályozza

Kézivezérelt színkeverés

- **Kapcsolási elrendezés:**
  Itt a tápfeszültséget lustaságból a **D4** lábról adjuk, nem követendő példa!
- **D3** – Red (piros)
- **D4** – közös elektróda
- **D5** – Green (zöld)
- **D6** – Blue (kék)

A0 – 1. potméter csúszka (piros)
A1 – 2. potméter csúszka (zöld)
A2 – 3. potméter csúszka (kék)
Az áramkorlátozás 3 x 470 Ω, vagy 3 x 510 Ω legyen!



Az RGB LED lábaira 510 Ω-os ellenállásokat forrasztottunk és egy tüskesorra építettük.

Így akár az Arduino nano mellé is dughatjuk.

A megoldás hátránya, hogy mindhárom LED árama a közös **D4** ágon folyik (max. 20 mA)

RGBled_pwm.ino

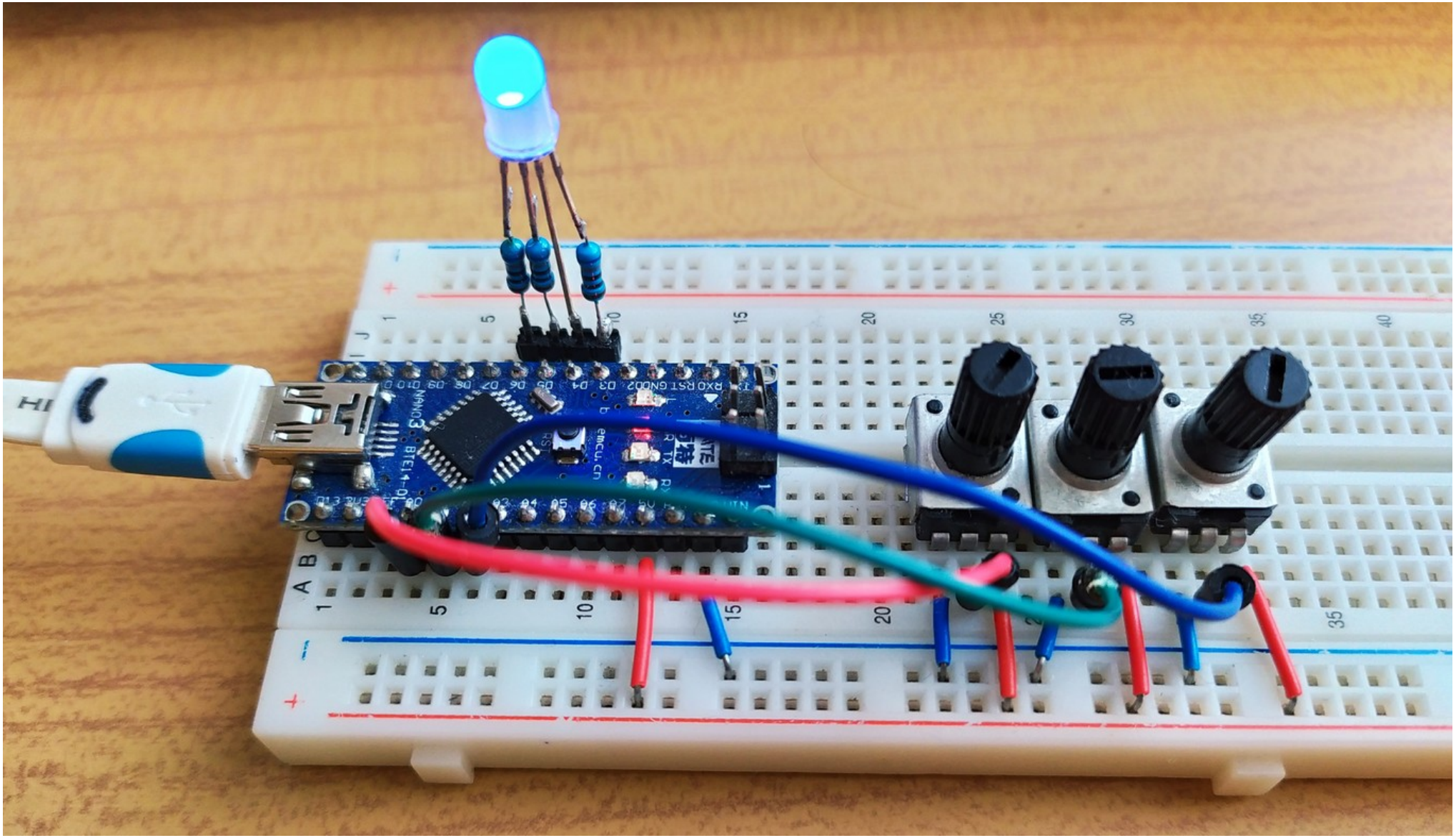
- Az **A0, A1, A2** bemenetekre kötött potméterekkel vezéreljük a **D3, D4, D5, D6** kivezetésekre kötött RGB LED-et *(**D4** a közös láb)*

```
#define red    3           // Piros LED = D3
#define common 4           // Közös elektróda = D4
#define green  5           // Zöld LED = D5
#define blue   6           // Kék LED = D6
#define common_level HIGH // Közös elektróda (LOW: katód, HIGH: anód)
```

```
byte r = 0, g = 0, b=0;    // Szinkomponensek
int t = 25;                // 25ms késleltetés

void setup() {
  pinMode(common,OUTPUT);
  setRGB();
}

void loop() {
  r = analogRead(A0)>>2;   // ADC: 0-1023, ezt
  g = analogRead(A1)>>2;   // 0-255 közé
  b = analogRead(A2)>>2;   // transzformáljuk
  SetRGB();                // LED beállítása
}
```

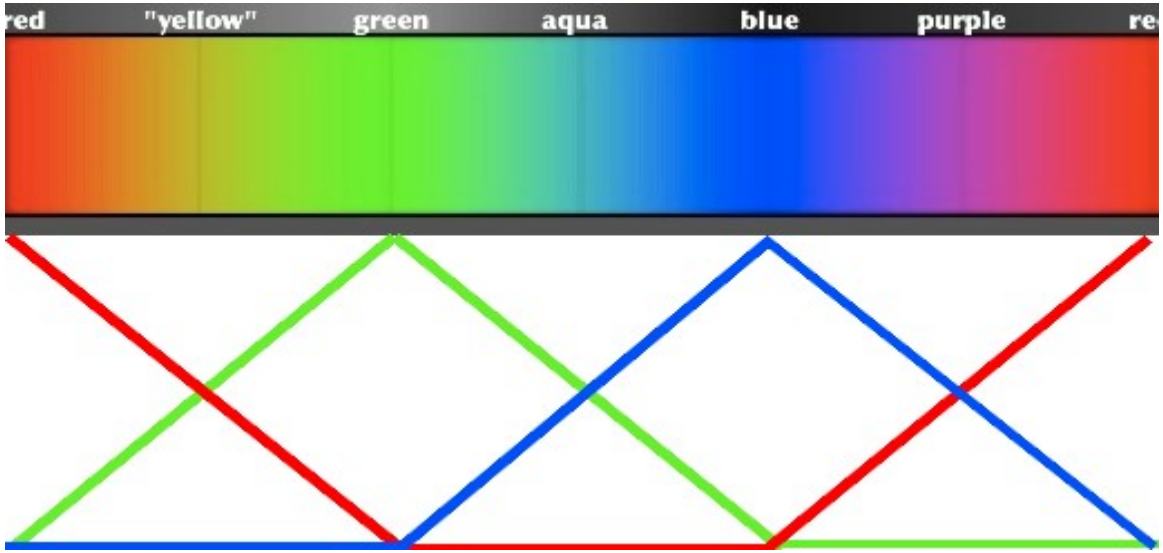
```
void setRGB(void) {
  if (common_level == HIGH) {
    digitalWrite(common, HIGH);
    analogWrite(red,255-r);
    analogWrite(green, 255-g);
    analogWrite(blue, 255-b);
  } else {
    digitalWrite(common, LOW);
    analogWrite(red,r);
    analogWrite(green, g);
    analogWrite(blue, b);
  }
  delay(t);
}
```

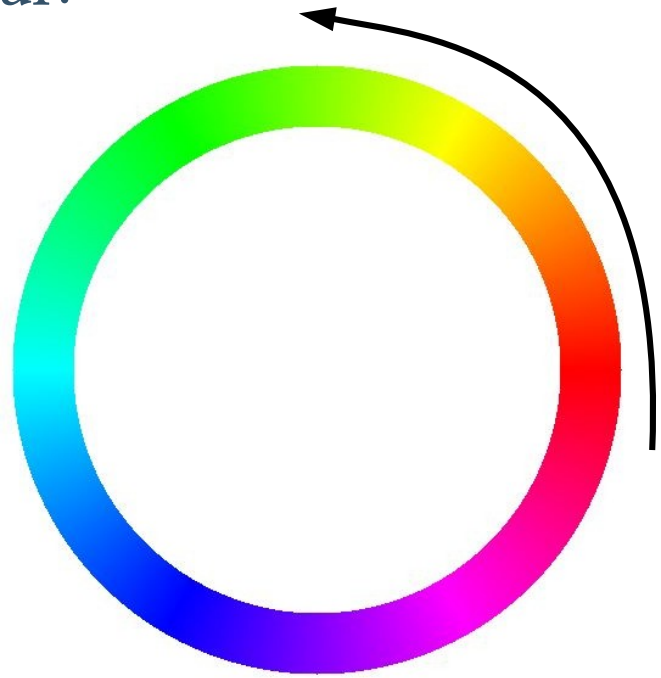
A megépített kapcsolás



Automatikus színátmenet

- Állítsuk elő a színkerék folytonos színátmeneteit!
- Három színátmenettel oldhatjuk meg, például:
  - Piros ⟶ zöld átmenet
  - Zöld ⟶ kék átmenet
  - Kék ⟶ piros átmenet
- A színkomponensek intenzitásának változásait az alábbi ábrán láthatjuk





A kapcsolás ugyanaz, mint a 10. oldalon, csak a potmétereket most nem használjuk

RGB_colorwheel.ino

- A **D3, D4, D5, D6** kivezetésekre kötött RGB LED-et vezéreljük *(**D4** a közös láb). A kapcsolás megegyezik a 10. oldalon bemutatottal*

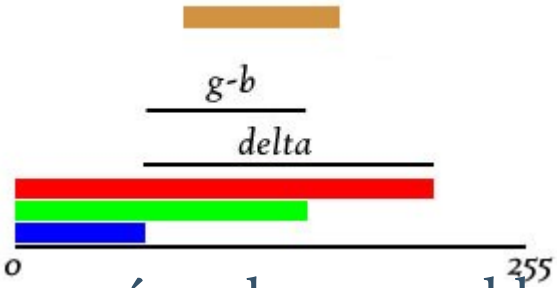
```
#define red    3           // Piros LED = D3
#define common 4           // Közös elektróda = D4
#define green  5           // Zöld LED = D5
#define blue   6           // Kék LED = D6
#define common_level HIGH // Közös elektróda (LOW: katód, HIGH: anód)
```

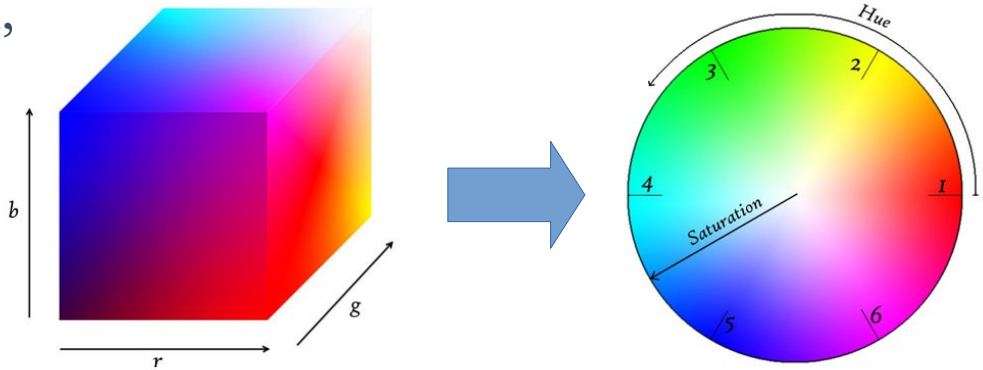
```
byte r = 255, g = 0, b=0;
int t = 25;

void setup() {
  pinMode(common,OUTPUT);
  setRGB();
}

void loop() {
  for (g=0; g < 255; g++, r--)
    setRGB(); // Piros -> zöld átmenet
  for (b=0; b < 255; b++, g--)
    setRGB(); // Zöld -> kék átmenet
  for (r=0; r < 255; r++, b--)
    setRGB(); // Kék -> piros átmenet
}
```

```
void setRGB(void) {
  if (common_level == HIGH) {
    digitalWrite(common, HIGH);
    analogWrite(red,255-r);
    analogWrite(green, 255-g);
    analogWrite(blue, 255-b);
  }
  else
  {
    digitalWrite(common, LOW);
    analogWrite(red,r);
    analogWrite(green, g);
    analogWrite(blue, b);
  }
  delay(t);
}
```

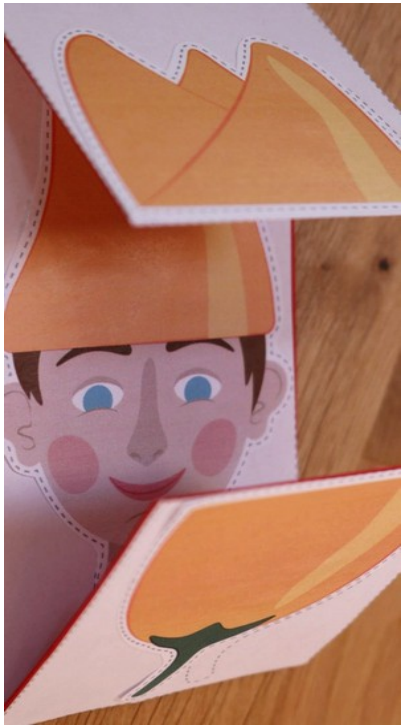
RGB → HSV konverzió

- Forrás: https://www.codeproject.com/Articles/9207/An-HSV-RGBA-colour-picker
- Induljunk ki egy r,g,b színhármasból, pl (209,146,65) ami egy drapp szín





- **A fényesség** a legnagyobb komponens értéke (max) osztva 255-tel: $val = \frac{max}{255}$
- **A telítettség** a szín tisztaságától, azaz a szürke tartalomtól függ: *delta* és a *maximum* hányadosa: $sat = \frac{delta}{max}$
  Tiszta színnél ez 1 (100 %), szürke színnél pedig nulla
- **A színezet** értékét a maximális komponenshez tartozó bázisszög (vörös 0°, zöld 120°, kék 240°) és 60°-nak a minimális komponenstől mért eltérések arányában vett hányadának összege adja meg
  A fenti példában $hue = \frac{g-b}{delta} \cdot 60^o = \frac{146-65}{209-65} \cdot 60^o = 33.75^o \approx 34^o$



HSV ⟶ RGB konverzió

- A színezettség (*Hue*) adatból kiindulva meghatározhatjuk a fő színt: ha Hue/60 = 0, vagy 5, akkor a vörös, ha 1, vagy 2 , akkor a zöld, egyébként pedig a kék a domináns szín, értéke: ***max = val · 255***
- A legkisebb komponens értékét az alábbi képlettel kapjuk meg:

```
base = (255.0f * (1.0 - sat) * val); // max*(1-sat)
```

- A középső komponens értéke ***delta · (hue%60)/60 + base***, azaz

```
(255.0f * val - base) * ((h%60)/60.0f) + base;
```

ha ***Hue/60 = 0, 2, vagy 4*** egyébként pedig

```
(255.0f * val - base) * (1 - ((h%60)/60.0f)) + base;
```

- Megjegyezzük, hogy a későbbiekben a fentiektől eltérő skálázással is találkozhatunk, ahol az egészaritmetika kedvéért a **telítettség** és a **fényesség** 0 – 255 közötti értékeket vehet fel

„Lélegző” LED, változó színben

- Korábban láttunk már mintaprogramot változó fényerejű LED-re (pl. **led_fade** a 2019. november 14-i előadásban). Most ezt azzal kombináljuk, hogy az elhalványodó majd kivilágosodó RGB LED
  - színét (*Hue*) változtathatjuk az **A0** bemenetre kötött potméter segítségével
  - fényességét (*Val*) periodikusan változtatjuk 0 és 255 között
  - telítettsége (*Sat*) pedig állandóan maximális értéken (255) van
- Amint látható, HSV rendszerben számolunk, s csak a megjelenítéskor térünk át RGB reprezentációra
- Az emberi látás nemlineáris jellegét egy korrekciós táblázattal próbáljuk figyelembe venni
- A kapcsolás ugyanaz, mint a 10. oldalon, de csak az **A0**-ra kötött potmétert használjuk
- **Felhasznált forrás:**
  kasperkamperman.com/blog/arduino/arduino-programming-hsb-to-rgb/

RGBLED_fade.ino 4/1. oldal

- A korrekciós táblázat

```
/**************************************************************************
 * A dim_curve korrekciós táblázat az emberi látás nemlineáris voltát kompenzálja.
 * A getRGB() függvényben használjuk a fényesség és a telítettség korrekciójára
 * (az utóbbihoz invertálva), a fényerőszabályozás így természetesebbnek tűnik.
 * Az alábbi értékeket közelítőleg az y = round(pow( 2.0, [x+64]/40.0) - 1)
 * képlettel írhatjuk le, ahol x = 0 - 255 közötti érték, de vannak eltérések.
 **************************************************************************/
const byte dim_curve[] = {
  0,   1,   1,   2,   2,   2,   2,   2,   2,   3,   3,   3,   3,   3,   3,   3,
  3,   3,   3,   3,   3,   3,   3,   4,   4,   4,   4,   4,   4,   4,   4,   4,
  4,   4,   4,   5,   5,   5,   5,   5,   5,   5,   5,   5,   5,   6,   6,   6,
  6,   6,   6,   6,   6,   7,   7,   7,   7,   7,   7,   7,   8,   8,   8,   8,
  8,   8,   9,   9,   9,   9,   9,   9,   10,  10,  10,  10,  10,  11,  11,  11,
  11,  11,  12,  12,  12,  12,  12,  13,  13,  13,  13,  14,  14,  14,  14,  15,
  15,  15,  16,  16,  16,  16,  17,  17,  17,  18,  18,  18,  19,  19,  19,  20,
  20,  20,  21,  21,  22,  22,  22,  23,  23,  24,  24,  25,  25,  25,  26,  26,
  27,  27,  28,  28,  29,  29,  30,  30,  31,  32,  32,  33,  33,  34,  35,  35,
  36,  36,  37,  38,  38,  39,  40,  40,  41,  42,  43,  43,  44,  45,  46,  47,
  48,  48,  49,  50,  51,  52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,
  63,  64,  65,  66,  68,  69,  70,  71,  73,  74,  75,  76,  78,  79,  81,  82,
  83,  85,  86,  88,  90,  91,  93,  94,  96,  98,  99,  101, 103, 105, 107, 109,
  110, 112, 114, 116, 118, 121, 123, 125, 127, 129, 132, 134, 136, 139, 141, 144,
  146, 149, 151, 154, 157, 159, 162, 165, 168, 171, 174, 177, 180, 183, 186, 190,
  193, 196, 200, 203, 207, 211, 214, 218, 222, 226, 230, 234, 238, 242, 248, 255,
};
```

RGBLED_fade.ino 4/2. oldal

```
#define sensorPin   A0    // ide van kötve a potméter
#define ledPinR      3    // pwm kimenet a piros LED számára
#define ledPinG      5    // pwm kimenet a zöld LED számára
#define ledPinB      6    // pwm kimenet a kék LED számára
#define commonPin    4    // LED közös kivezetés
#define common_level HIGH // Közös anód: HIGH, katód: LOW

int sensorVal = 0;        // Az ADC-vel mért érték
int fadeVal   = 0;        // fényerősség, 0-255 között változik
int fadeSpeed = 4;        // az elhalványodás 'sebessége'

// A getRGB() függvény ebbe a tömbbe írja be az RGB értékeket
// amelyet a piros, zöl és kék LED-ek beállításához használunk
int rgb_colors[3];

int hue;                  // Színezet
int saturation;           // Telítettség
int brightness;           // Fényesség

void setup() {
  pinMode(commonPin, OUTPUT);             // 'Lusta' bekötéshez
  digitalWrite(commonPin, common_level);  // 'Lusta' bekötéshez
}
```

RGBLED_fade.ino 4/3. oldal

- A ***loop*** függvényben: analóg mérés, fading és megjelenítés

```
void loop()  {
  sensorVal = analogRead(sensorPin);      // Analóg beolvasás
  fadeVal = fadeVal + fadeSpeed;          // a fadeVal módosítása
  fadeVal = constrain(fadeVal, 0, 255);   // fadeVal 0 és 255 között legyen
  if (fadeVal == 255 || fadeVal == 0) {   // lefelé/fölfelé irányváltás
    fadeSpeed = -fadeSpeed;
  }
  hue = map(sensorVal, 0, 1023, 0, 359); // színezet 0 - 360 közötti szám
  saturation = 255;                       // telítettség 255 legyen
  brightness = fadeVal;                   // fényesség 0 - 255 közötti érték
  getRGB(hue, saturation, brightness, rgb_colors);  // HSB → RGB konverzió
  if (common_level == HIGH) {             // Közös anód esete
    analogWrite(ledPinR, 255 - rgb_colors[0]);      // red value
    analogWrite(ledPinG, 255 - rgb_colors[1]);      // green value
    analogWrite(ledPinB, 255 - rgb_colors[2]);      // blue value
  } else {                                // Közös katód esete
    analogWrite(ledPinR, rgb_colors[0]); // red value
    analogWrite(ledPinG, rgb_colors[1]); // green value
    analogWrite(ledPinB, rgb_colors[2]); // blue value
  }
  delay(20);                              // Fade sebességét szabályozza
}
```

RGBLED_fade.ino 4/4. oldal

```
void getRGB(int hue,int sat,int val,int colors[3])
{ val = dim_curve[val];
  sat = 255 - dim_curve[255 - sat];
  int r, g, b, base;
  if (sat == 0) { // Hue doesn't mind.
    colors[0] = val;
    colors[1] = val;
    colors[2] = val;
  } else  {
  base = ((255 - sat) * val) >> 8;
  switch (hue/60) {
    case 0:
      r = val;
      g = (((val-base)*hue)/60)+base;
      b = base;
      break;
    case 1:
      r = (((val-base)*(60-(hue%60)))/60)+base;
      g = val;
      b = base;
      Break;
     case 2:
      r = base;
      g = val;
      b = (((val-base)*(hue%60))/60)+base;
        break;
```

```
case 3:
  r = base;
  g = (((val-base)*(60-(hue%60)))/60)+base;
        b = val;
        break;

case 4:
  r = (((val-base)*(hue%60))/60)+base;
  g = base;
  b = val;
  break;

case 5:
  r = val;
  g = base;
  b = (((val-base)*(60-(hue%60)))/60)+base;
  break;
 }

   colors[0] = r;
   colors[1] = g;
   colors[2] = b;
  }
}
```

RGB LED programkönyvtárak

- Az előző példaprogram elég hosszú volt ahhoz, hogy rájöjjünk, célszerű programkönyvtárat keresni vagy írni az RGB LED-ek vezérléséhez, színátmeneteinek kényelmes kezeléséhez
- Szempontok:
  - Legyen objektumorientált
  - Kezelje transzparensen a közös anód illetve katód különbségeit
  - Használhassunk előre definiált színeket
  - Nyújtson kényelmes kezelhetőséget a kívánt feladatokra
  - HSV ⟶ RGB, esetleg RGB ⟶ HSV konverzió lehetősége

Néhány ismertebb programkönyvtár:

- **RGBLed** (by Steven Wilmouth, github.com/wilmouths/RGBLed)
- **RGBLED2** (RGBLED by Bret Stateham, github.com/BretStateham/rgbled)
- **FastLED** (főleg digitális vezérlésű LED-hez, github.com/FastLED/FastLED)

RGBLed (Steven Wilmouth)

- Egyszerű felépítésű programkönyvtár, színmegadás RGB számhármassal vagy előre definiált színnevekkel

- **off**();
- **brightness**(int rgb[3], int brightness);
- **brightness**(int red, int green, int blue, int brightness);
- **setColor**(int rgb[3]);
- **setColor**(int red, int green, int blue);
- **flash**(int rgb[3], int duration);
- **flash**(int rgb[3], int onDuration, int duration);
- **flash**(int red, int green, int blue, int duration);
- **flash**(int red,int green,int blue,int onDuration,int duration);
- **fadeOut**(int rgb[3], int steps, int duration);
- **fadeOut**(int red, int green, int blue, int steps, int duration);
- **fadeIn**(int rgb[3], int steps, int duration);
- **fadeIn**(int red, int green, int blue, int steps, int duration);

RGBLed_flash.ino

```
#include <RGBLed.h>
RGBLed led(3, 5, 6, COMMON_ANODE);

void setup() {
  pinMode(4,OUTPUT);
  digitalWrite(4,HIGH);
}

void loop() {
  led.flash(RGBLed::RED, 1000);
  led.flash(0, 255, 0, 1000);
  for(int i=0; i<8; i++)
    led.flash(RGBLed::BLUE, 50);
}
```

Konklúzió

- Továbbfejlesztésre érdemes
- A közös anódú LED esete nem kidolgozott a fade fv-ekben
- HSV kezelést enm támogat

RGBLED2 (Bret Stateham)

- Az előzőnél szerényebb függvénykészlet, de van benne **HSV** kezelés és a **mapValue()** fv. gondoskodik a közös anódú esetről
- void **writeRGB**(int red, int green, int blue);
- void **writeHSV**(int h, double s, double v);
- void **writeRed**(int red);
- void **writeGreen**(int green);
- void **writeBlue**(int blue);
- void **turnOff**(void);
- void **writeRandom**(void);
- void **writeColorWheel**(int dly);
- int **mapValue**(int value);

rgbled_colorwheel2.ino

```
#include <RGBLED2.h>
RGBLED rgbLed(3,5,6,COMMON_ANODE);

int delayMs = 100;
void setup() {
  pinMode(4,OUTPUT);      // "lusta" bekötéshez
  digitalWrite(4,HIGH);   // Közös anód
}

void loop() {
  // Körbejárjuk a színkereket
  rgbLed.writeColorWheel(delayMs);
}
```

FastLed (†Daniel Garcia & Mark Kriegsman)

- **FastLED:** gyors, hatékony (és bonyolult) programkönyvtár a címezhető LED szalagok és egyéb kijelzők kényelmes kezeléséhez
- Támogatott LED-ek: NeoPixel, WS2811, WS2812, WS2812B, DotStar, APA102, APA104, GW6205/GW6205_400, P9813 Total Control Lighting LEDs, USC1903_400, Pixelmatix SmartMatrix, WS2801, LPD8806, LPD1886, TM1809, TM1804, TM1803, and SM16716
- Teljes támogatás **HSV** és **RGB** kezeléshez, konverzióhoz
- Gyors matematikai és memória műveletek
- Honlap: fastled.io/
- Dokumentáció: github.com/FastLED/FastLED/wiki/Overview
- Forráskód: github.com/FastLED/FastLED

AnalogOutput.ino

- A **FastLed** programkönyvtár elsősorban a címezhető LED-ek vezérlésére készült, támogató függvényeinek azonban analóg LED-ek, vagy LED-szalagok vezérlésénél is használt vehetjük
- Az alábbi programban az alapszínek felvillantása után a színkereket járjuk körbe

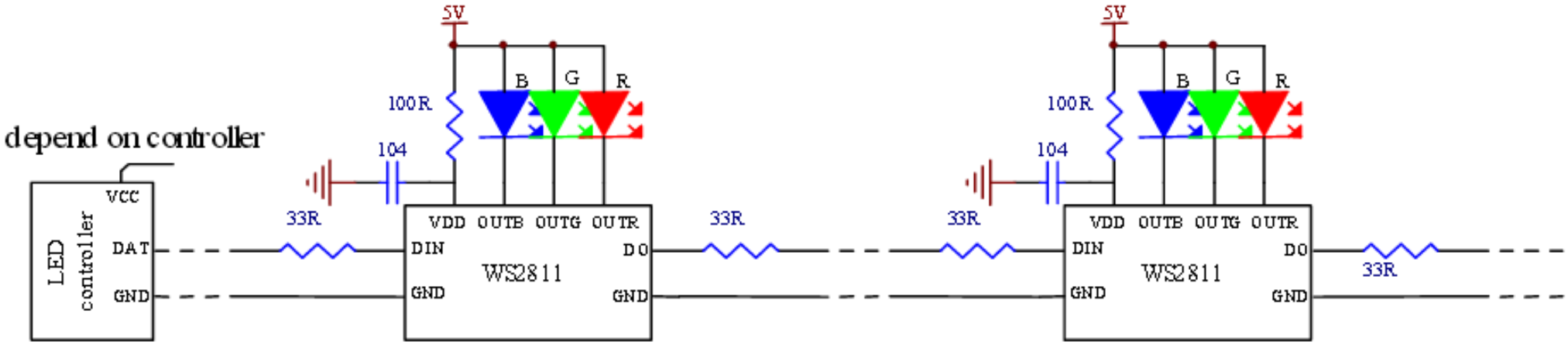
```
#include <FastLED.h>

#define REDPIN   3
#define GREENPIN 5
#define BLUEPIN  6

void showAnalogRGB( const CRGB& rgb) {
  analogWrite(REDPIN,   255-rgb.r );
  analogWrite(GREENPIN, 255-rgb.g );
  analogWrite(BLUEPIN,  255-rgb.b );
}

void colorBars() {
  showAnalogRGB( CRGB::Red );   delay(500);
  showAnalogRGB( CRGB::Green ); delay(500);
  showAnalogRGB( CRGB::Blue );  delay(500);
  showAnalogRGB( CRGB::Black ); delay(500);
}
```

```
void setup() {
  pinMode(REDPIN,   OUTPUT);
  pinMode(GREENPIN, OUTPUT);
  pinMode(BLUEPIN,  OUTPUT);
  pinMode(4,OUTPUT);        // lusta bekötés
  digitalWrite(4,HIGH);     // közös anód
  // Flash R, G, B, black.
  colorBars();
}

void loop() {
  static uint8_t hue;
  hue = hue + 1;
  // Use FastLED automatic HSV→RGB conversion
  showAnalogRGB( CHSV( hue, 255, 255) );
  delay(20);
}
```

A szokásostól eltérő konvenció: hue 0 - 255 közötti szám

WS2812 alapok

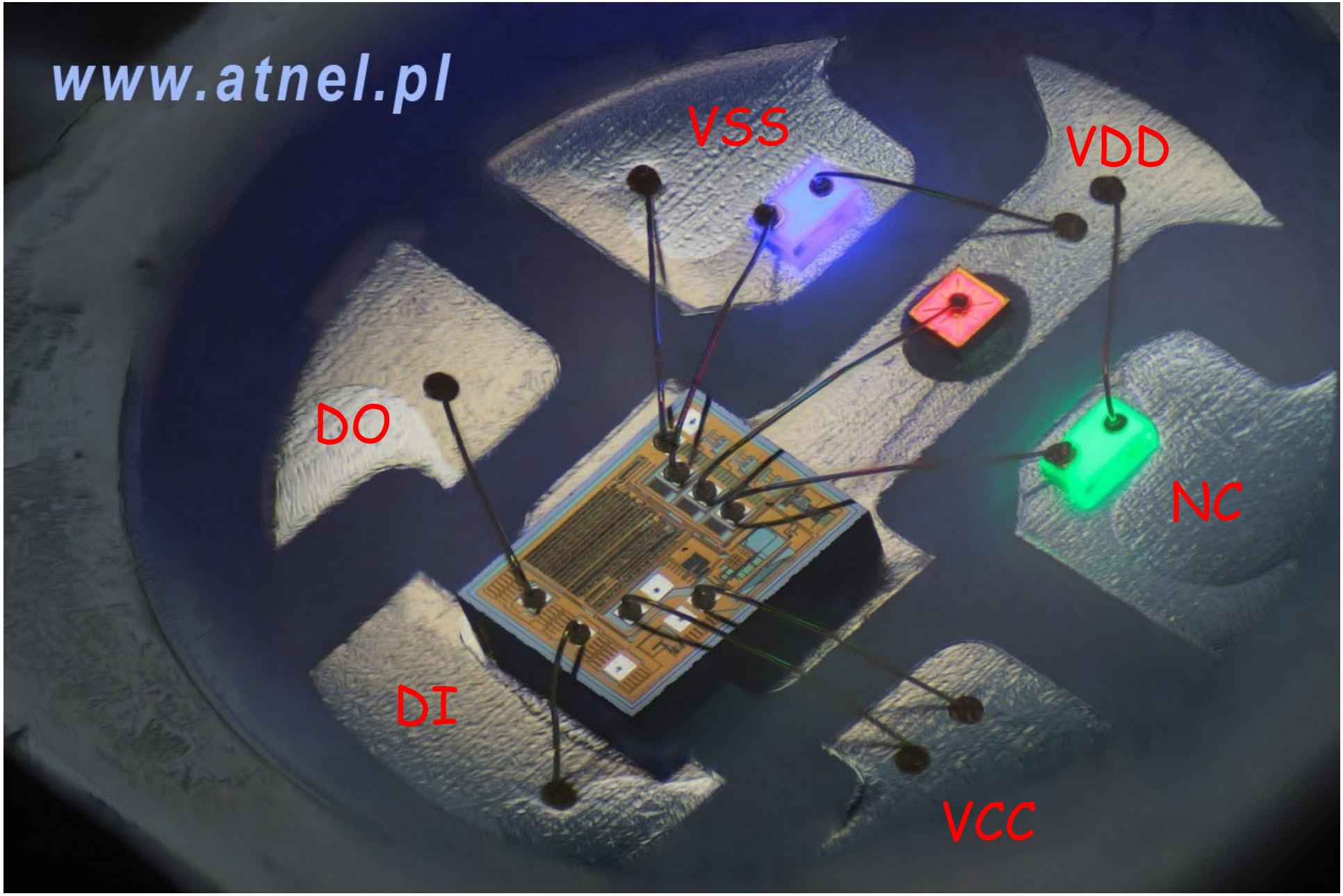
Kezdetben volt a **WS2801** és a **WS2811** LED vezérlő IC (WS = World Semi, a gyártó).



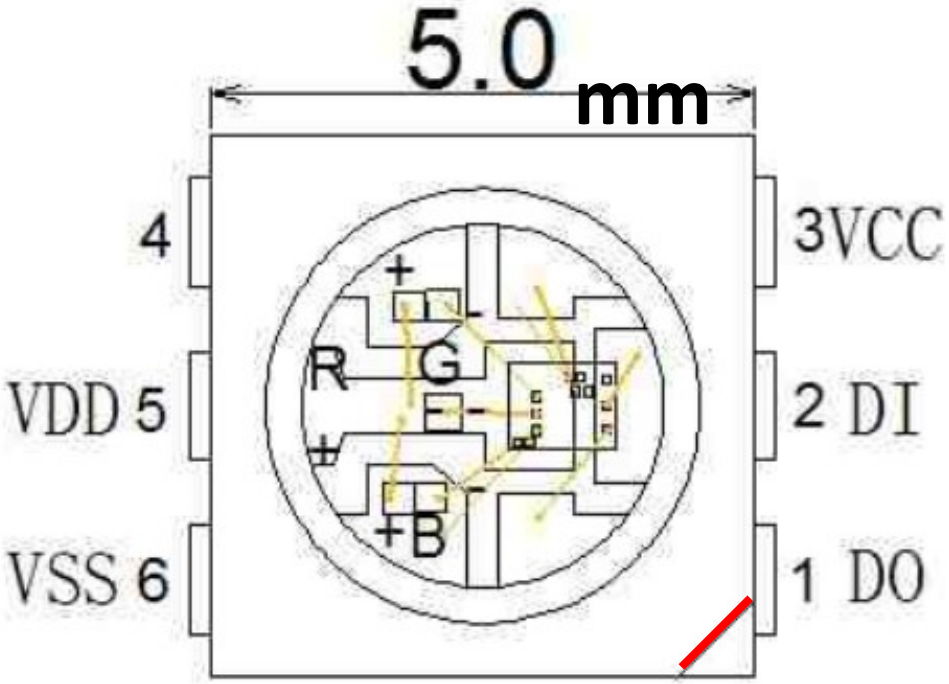
A **WS2812** (NeoPixel) viszont egy olyan 5x5 mm-es RGB LED, amelybe már be van építve egy WS2811-hez hasonló vezérlő!

5050 méretű RGB LED **5050 méretű WS2812**

A WS2812 belső felépítése



WS2812 lábkiosztás, paraméterek



Láb	Név	Funkció
1	DO	Adat kimenet
2	DI	Adat bemenet
3	VCC	Vezérlő tápfeszültsége
4	NC	Nincs bekötve
5	VDD	LED-ek tápfeszültsége
6	VSS	Közös pont (GND)

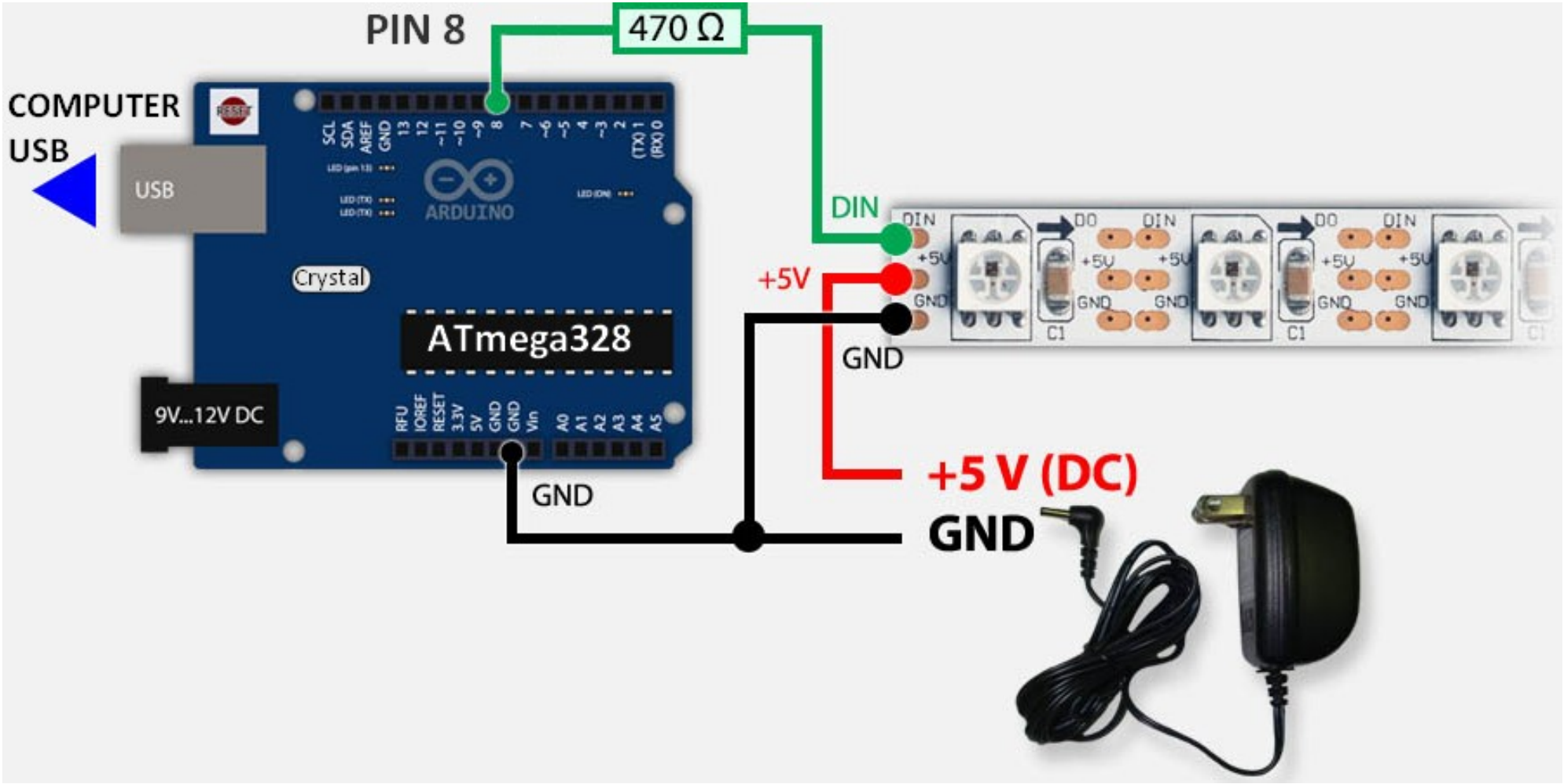
Paraméter	Jel	Határérték	Egység
Tápfeszültség (vezérlő)	VCC	+6,0 – 7,0	V
Tápfeszültség (LED)	VDD	+6,0 – 7,0	V
Bemeneti jelszint	DI	-0,5 – VDD+0,5	V
Áramfelvétel LED-enként	I	0 – 20	mA

Megjegyzések: Erős a gyanú, hogy az adatlapban van némi keveredés…

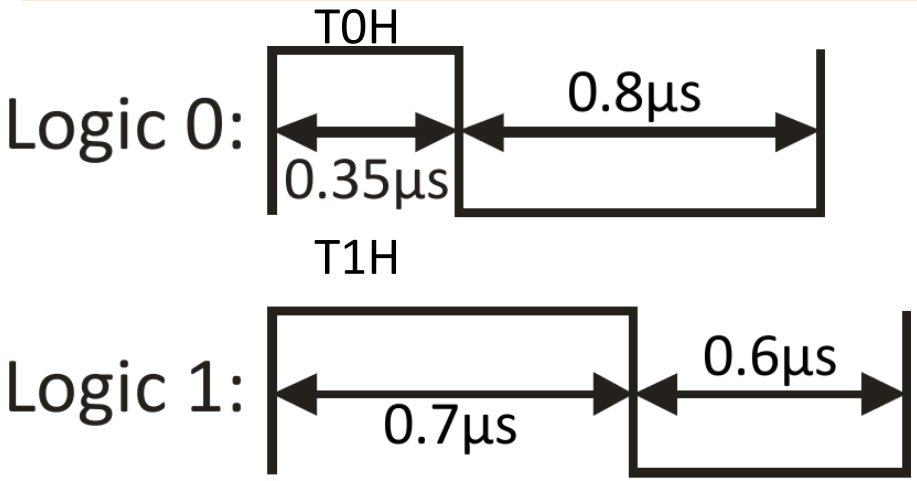
1. A tápfeszültség inkább +5,0 – 7,0 V, nem pedig +6,0 – 7,0 V!
2. A bemeneti jelszint inkább a vezérlő, mintsem a LED-ek tápfeszültségéhez igazodik!

WS2812 szalag bekötése

Az áramfelvétel számításához LED-enként 3 x 20 mA maximális áramot vegyünk!
Méterenként 30 db LED-del számolva ez 1.8A/m áramfelvételt jelent.



WS2812 kommunikáció

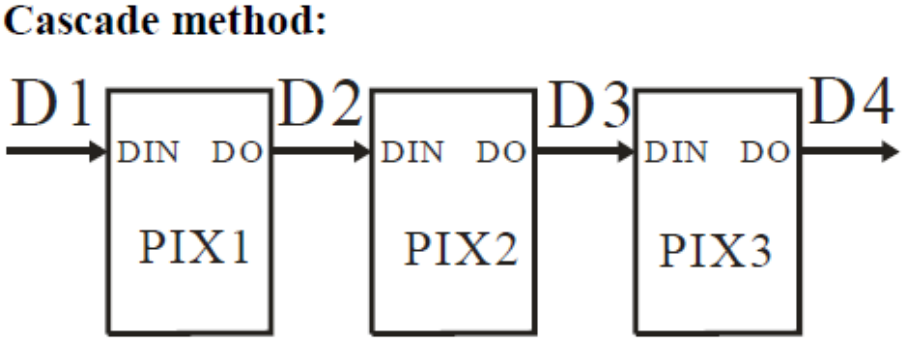


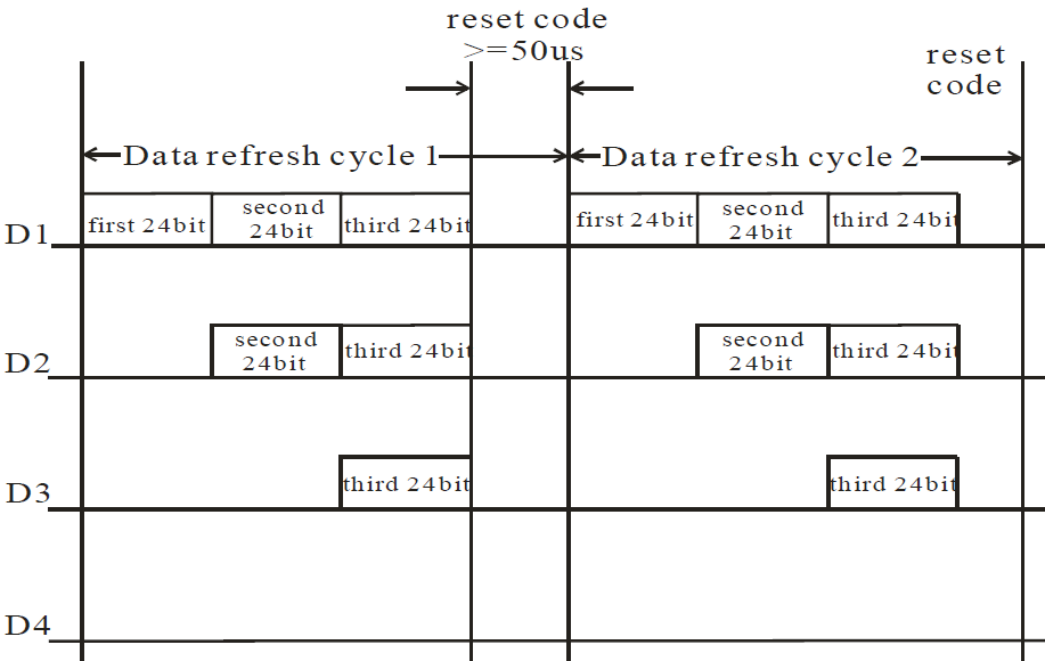
Az egyes bitek értéke a magas szinten eltöltött időtől függ.
T0H: (350 ± 150) ns
T1H: (700 ± 150) ns

RESET: alacsony szint T ≥ 50 000 ns ideig.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

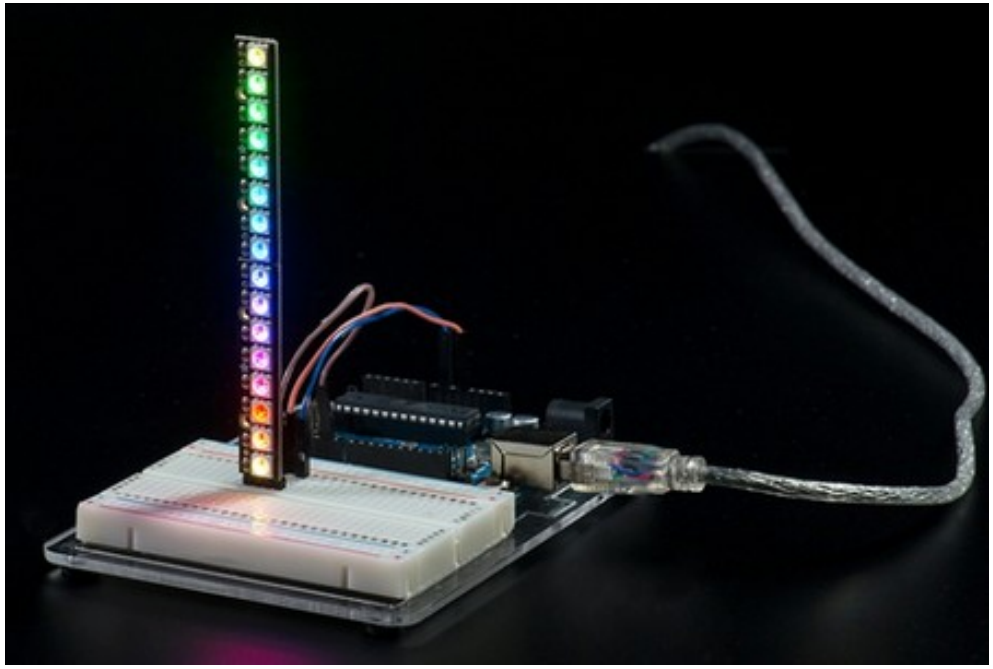
Adatformátum: 3 x 8 bit, GRB sor-rendben, magas helyiértékű bit elöl.

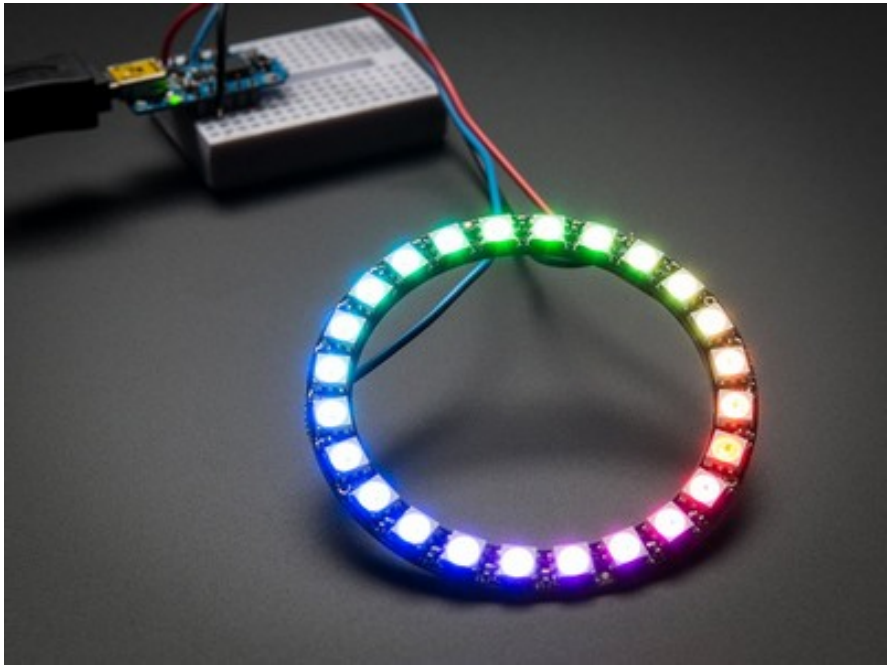




Adafruit Neopixel programkönyvtár

- Az **Adafruit_Neopixel** programkönyvtár használatára már mutattunk példákat a **2016. április 6-i** előadásban, bővebben lásd ott!
- Az alábbiakban most a **FastLed** programkönyvtár használatára mutatunk néhány egyszerű példát egy, illetve nyolc, felfűzött LED esetére, a LED-ek száma természetesen bővíthető





Neopixel.ino

- Egy Neopixel (WS2812) LED vezérlése **FastLed** programkönyvtár használatával: piros, és zöld szín felvillantása felváltva
- Bekötés a 30. oldal szerint (**D8** a vezérlő kimenet)

```
#include <FastLED.h>
#define NUM_LEDS 1              // Felfűzött LED-ek száma
#define DATA_PIN 8              // A vezérlő kivezetés

CRGB leds[NUM_LEDS];

void setup() {
  FastLED.addLeds<NEOPIXEL,DATA_PIN>(leds, NUM_LEDS);
}

void loop() {
  leds[0] = CRGB::Red;          // Piros szín beállítása
  FastLED.show();
  delay(1000);
  leds[0] = CRGB::Green;        // Zöld szín beállítása
  FastLED.show();
  delay(1000);
}
```

neopixel_hsv.ino

- Egy Neopixel (WS2812) LED vezérlése **FastLed** programkönyvtár használatával: a *hue* léptetésével végigvezetjük a színkörön
  Megjegyzés: ne feledjük, hogy itt *hue* 0 – 255 közötti szám lehet!
- Bekötés a 30. oldal szerint (**D8** a vezérlő kimenet)

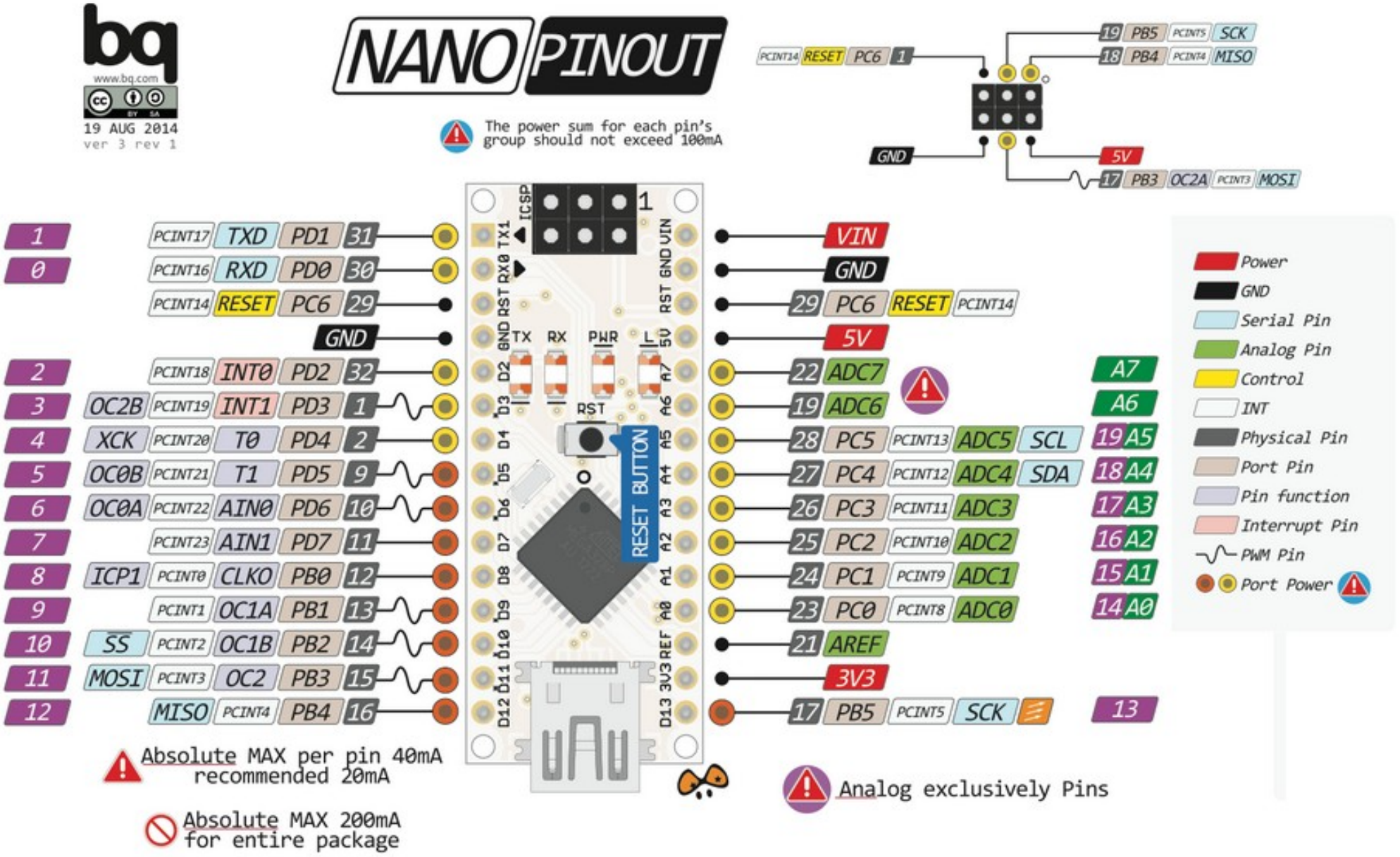
```
#include <FastLED.h>
#define NUM_LEDS 1
#define DATA_PIN 8

CRGB leds[NUM_LEDS];

void setup() {
  FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
}

void loop() {
  static uint8_t hue;
  leds[0] = CHSV(hue++, 255, 255);
  FastLED.delay(33);
}
```

neopixel_8.ino

- **8 db** felfűzött Neopixel (WS2812) LED vezérlése **FastLed** programkönyvtár használatával: a hue léptetésével végigfuttatjuk a a színskálát a LED-eken
- Bekötés a 30. oldal szerint (**D8** a vezérlő kimenet)

```
#include <FastLED.h>
#define NUM_LEDS 8
CRGBArray<NUM_LEDS> leds;

void setup() {
  FastLED.addLeds<NEOPIXEL, 8>(leds, NUM_LEDS);
}

void loop() {
  static uint8_t hue = 0;
  for (int i = 0; i < NUM_LEDS; i++) {
    leds.fadeToBlackBy(40);           // fade everything out
    leds[i] = CHSV(hue, 255, 255);    // let's set an led value
    hue += 5;
    FastLED.delay(40);
  }
}
```

Az Arduino nano kártya kivezetései



Ellenállás színkódok