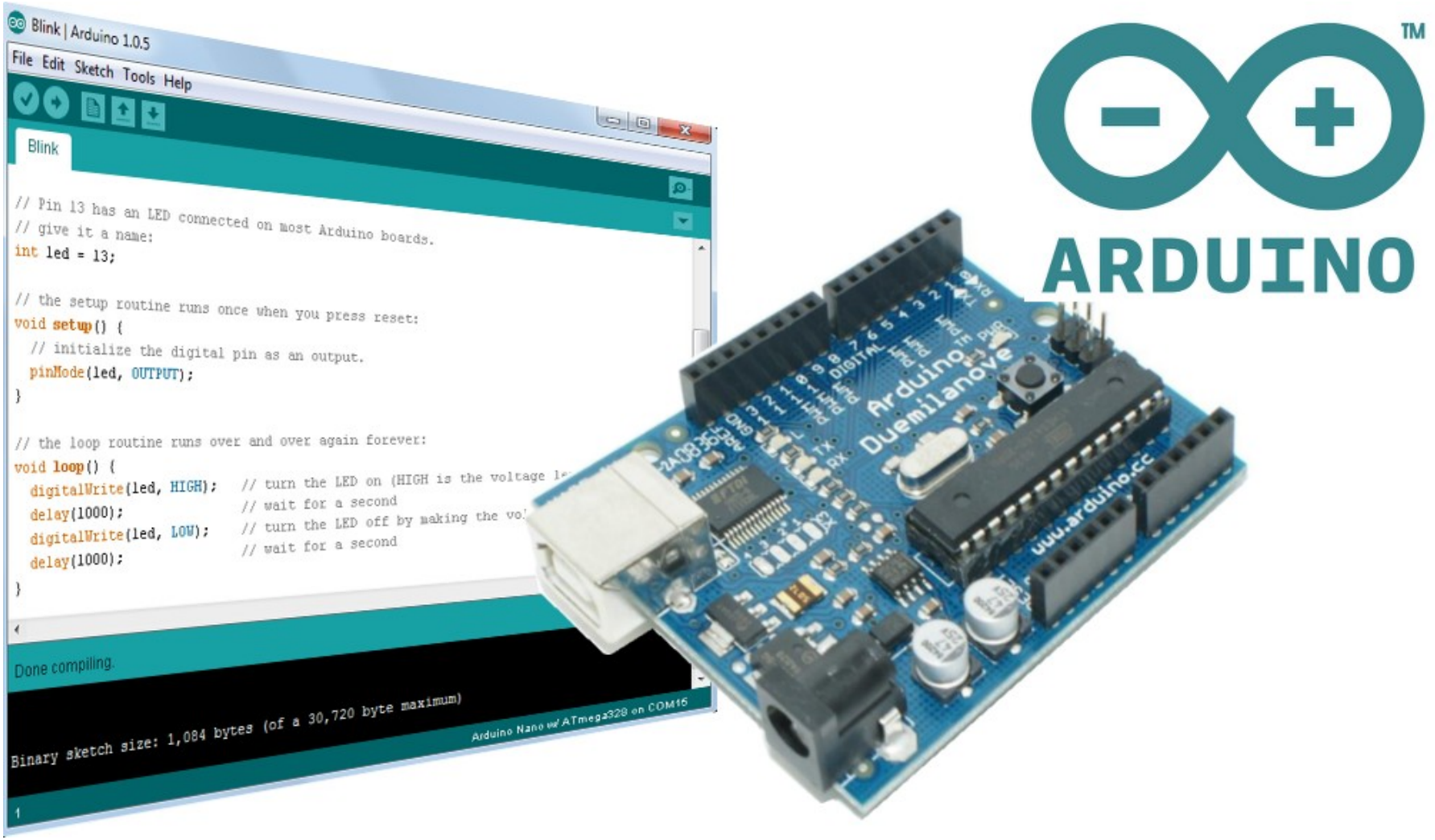
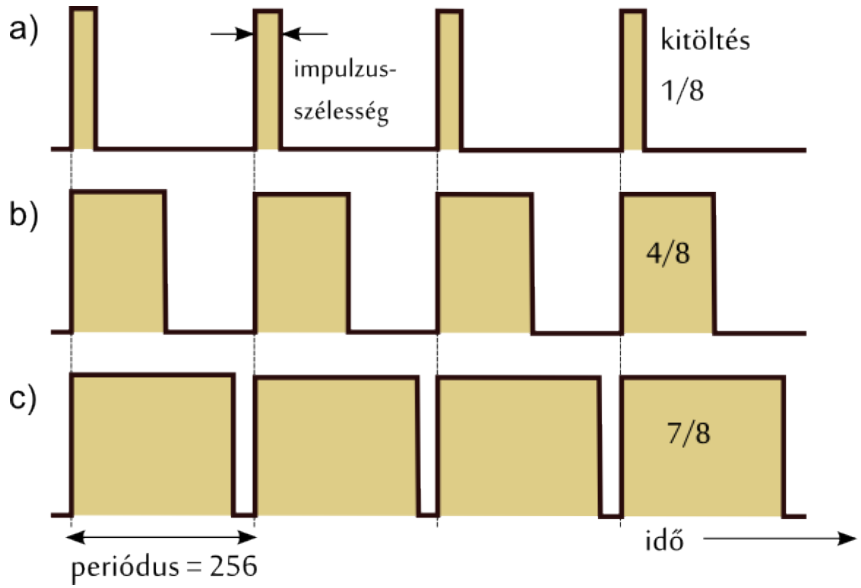
Bevezetés az elektronikába



13. Arduino programozás – analóg I/O

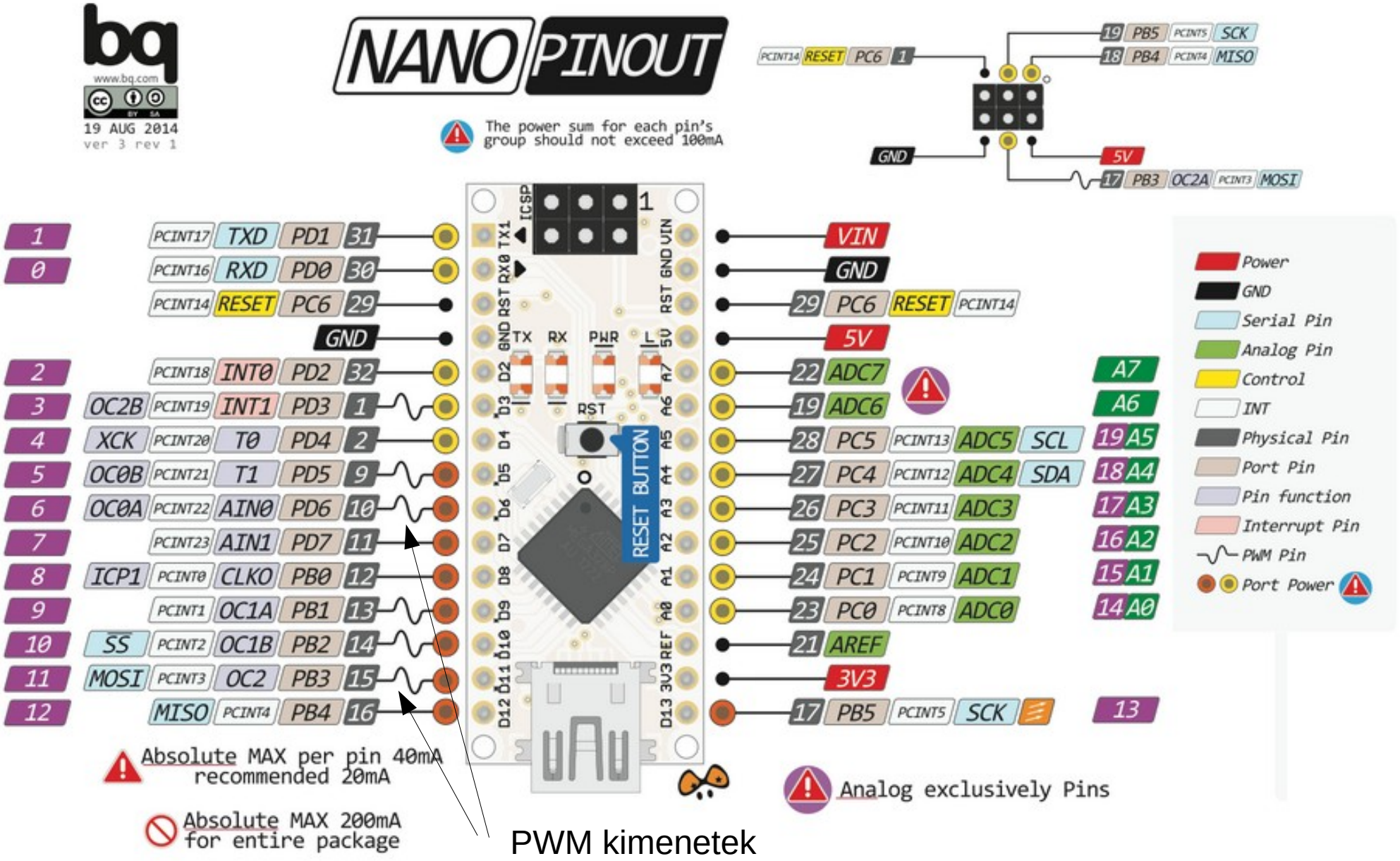
PWM: impulzus-szélesség moduláció

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



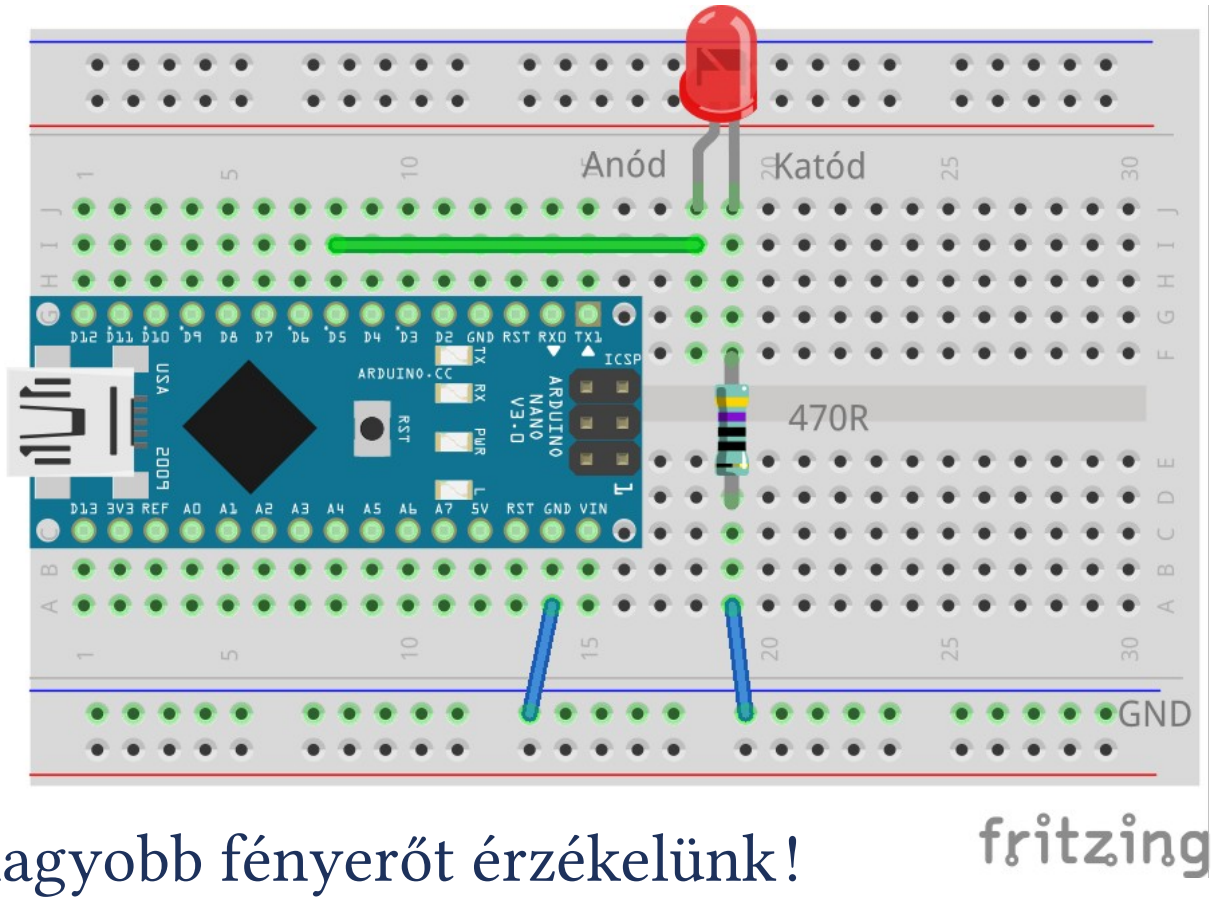
Időzítő	Csatorna	Kivezetés	Frekvencia
Timer0	OC0A, OC0B	6, 5	980 Hz
Timer1	OC1A, OC1B	9, 10	490 Hz
Timer2	OC2A, OC2B	11, 3	490 Hz

Az Arduino nano kártya kivezetései



„Lélegző” LED – led_fade.ino

- A **D5** kivezetésre kötött LED fényerejét fokozatosan növeljük, majd a maximum elérése után fokozatosan csökkentjük
- Az **`analogWrite(pin,duty)`** függvény két paramétere a vezérelni kívánt digitális kivezetés sorszáma (csak 3, 5, 6, 9, 10, 11 lehet) és a PWM kitöltési tényező (0 – 255 közötti érték)
- Mivel a LED úgy van bekötve, hogy magas szintű jel esetén világít, így a kis kitöltés kisebb fényerőt, a nagy kitöltés nagyobb fényerőt jelent.
- Szemünk nem lineárisan érzékel, kétszer nagyobb kitöltésnél nem kétszer nagyobb fényerőt érzékelünk!



led_fade.ino

- A **D5** kivezetésre kötött LED fényerejét változtatjuk („lélegző” LED)

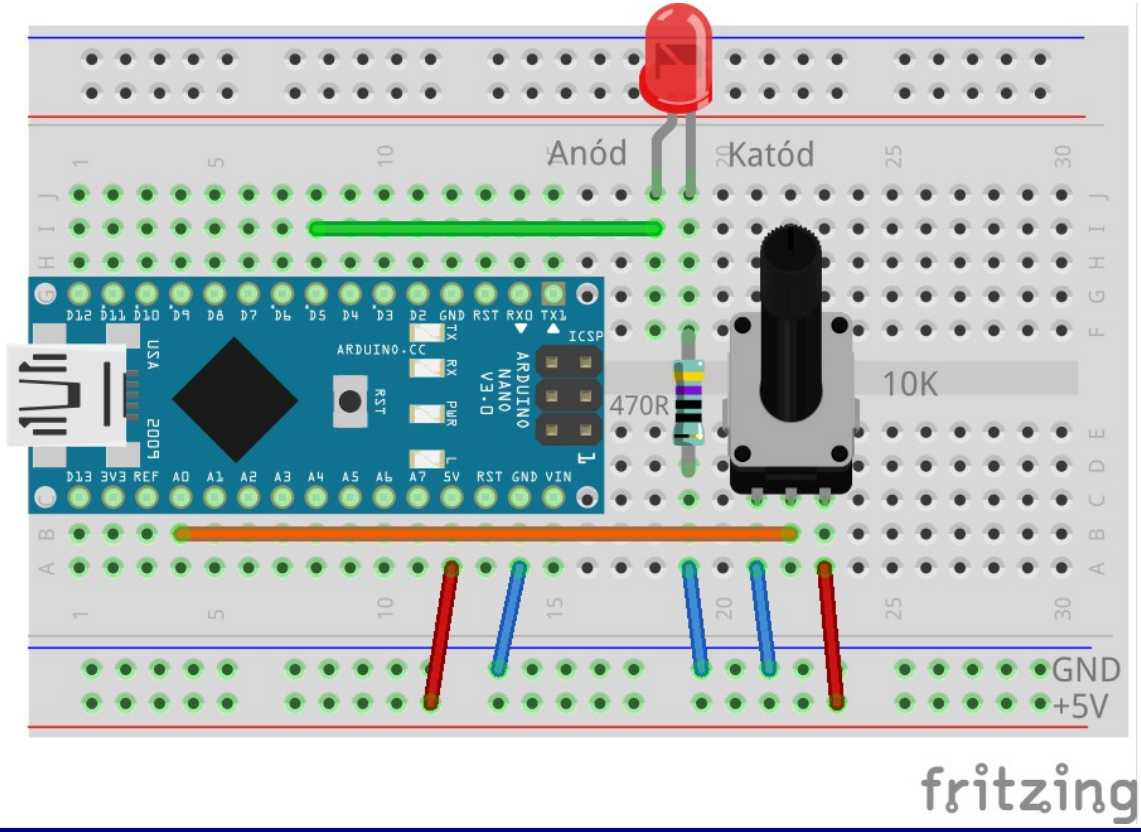
```
Const int led = 5;          // ide van kötve a LED
int brightness = 0;         // fényerő értéke
int fadeAmount = 5;         // a változás léptéke

void setup() {
  pinMode(led, OUTPUT);     // D5 legyen kimenet (ez a sor elhagyható)
}
void loop() {
  // A LED kimeneten beállítjuk a kitöltést (a LED fényerejét)
  analogWrite(led, brightness);
  // megnöveljük a fényerő értékét az általunk megadott
  // léptékben, mindig, mikor lefut a loop, hozzáadódik
  brightness = brightness + fadeAmount;
  // a végén megfordítjuk a változás irányát:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // Várunk, hogy a szemünk is láthassa a változást:
  delay(50);
}
```

Forrás: Harsányi Réka - Juhász Márton András: Fizikai számítástechnika

Teljesítményvezérlés potméterrel

- A potméterrel leosztott feszültséget megmérjük az `analogRead(A0)` függvényhívással (0 – 1023 közötti értéket kapunk)
- A kapott számot átskálázzuk a 0 – 255 tartományba, majd az `analogWrite()` függvényhívással erre az értékre állítjuk be a **D5** PWM csatorna kitöltését
- **LED Arduino**
  anód – D5
  katód – GND
  (470 Ω-on keresztül)
- **Potméter Arduino**
  teteje +5 V
  csúszka A0
  alja GND
  (a potméter 10 kΩ-os)



led_pwm.ino

- Az ADC 0 – 1023 közötti számot ad vissza, ezt legalább néggyel kell osztani, hogy 0 – 255 közötti számot kapjunk a PWM vezérléséhez
- Itt most 8-cal osztjuk az ADC-ből kapott értékeket, s osztás helyet jobbraléptetést végzünk

```
const int led = 5;       // ide van kötve a LED

void setup() {
  // Nincs tennivaló
}

void loop() {
  int reading = analogRead(A0);
  analogWrite(led, reading>>3 );  // Osztás 8-cal
  // Várunk, hogy a szemünk is láthassa a változást:
  delay(50);
}
```

Vezérlési szerkezetek

A műveletek végrehajtási sorrendjét határozzák meg.

- Utasítások és blokkok
- Feltételes elágazás
  - **if – else** utasítás
  - **else – if** utasítás
  - **switch** utasítás ← Ma erről lesz szó
- Ciklusszervezés
  - **while** utasítás
  - **for** utasítás
  - **do – while** utasítás
- A **break** és **continue** utasítások
- A **goto** utasítás és **a címkék**

Forrás: BRIAN W. KERNIGHAN – DENNIS M. RITCHIE: A C programozási nyelv, 3. fejezet

A switch utasítás

- A **switch** utasítás egy kifejezés értékét több egész értékű állandó értékével hasonlítja össze, és azt az ágat hajtja végre, amelyiknél egyezést talál
- A switch utasítás általános felépítése:

```
switch (kifejezés) {
    case állandó kifejezés: utasítások; break
    case állandó kifejezés: utasítások; break
    . . .
    default: utasítások
}
```

A **break** szerepe az, hogy kiugorjon a **switch** utasítás törzséből

- Egy példa:

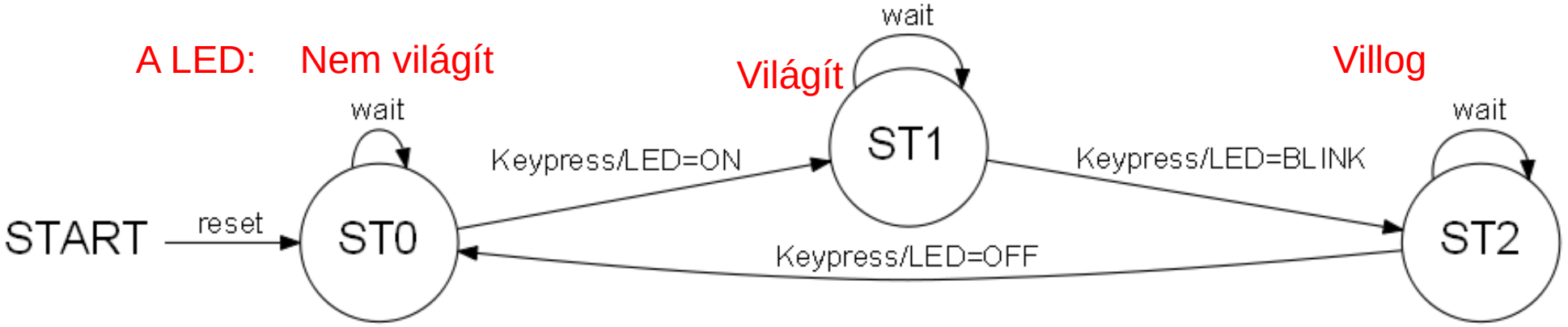
```
switch (led_state) {
  case 0: digitalWrite(RED_LED, LOW); break;
  case 1: digitalWrite(RED_LED, HIGH); break;
  default: digitalWrite(RED_LED, !digitalRead(RED_LED));
}
```

Újraírjuk a led_tristate.ino programot

- Emlékszünk még a piros LED vezérlésére nyomógomb segítségével?
  - első lenyomásra a piros LED világít
  - második lenyomásra a piros LED villog (kb. ~~1 Hz~~ 2 Hz)
  - harmadik lenyomásra a piros LED kialszik

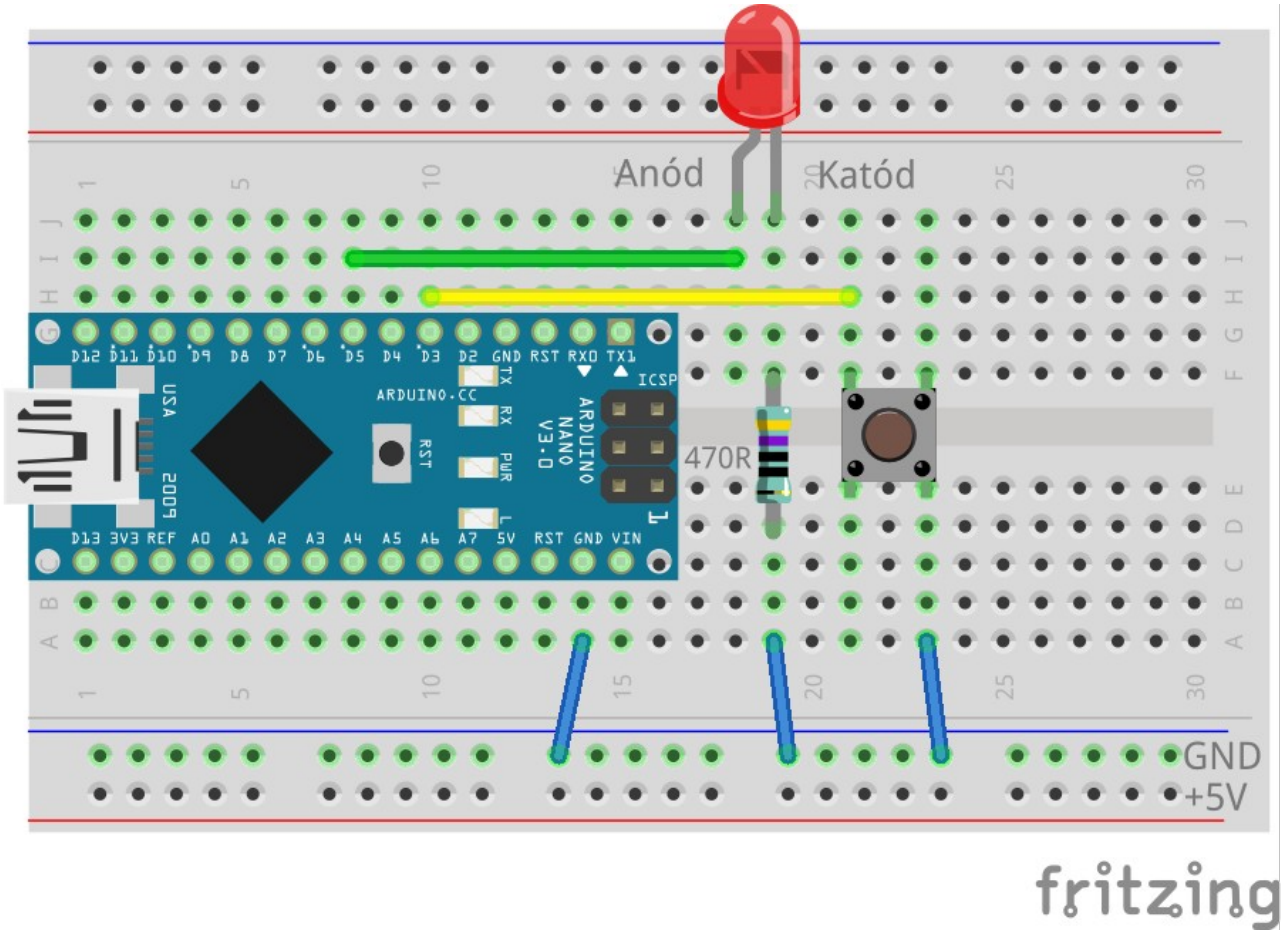
Most egy kicsit gyorsabbra vesszük!

- Minden gomblenyomás egy újabb állapotba viszi a LED-et:
  - 0: A LED nem világít (a kimenet LOW)
  - 1: A LED folyamatosan világít (a kimenet HIGH)
  - 2: A LED villog (a kimenet állapotát ~~500~~ 250 ms-onként átbillentjük)
- A LED állapotát a **led_state** változóban tároljuk



A kapcsolás

- Hozzávalók: Arduino, LED, nyomógomb, 470 Ω-os ellenállás
- LED anódja **D5**-re a nyomógomb pedig **D3**-ra csatlakozik



Ez volt a led_tristate.ino program

```
const int RED_LED = 5;              // D5-re van kötve a LED
const int PUSH2 = 3;                // D3-ra van kötve a nyomógomb

// Hardverfüggetlen rész (MSP430 Launchpad kártyán is futtatható!)
boolean waitforpress = true;        // Kezdetben lenyomásra várunk
int led_state = 0;                  // Kezdetben a LED ki van kapcsolva
int timecount = 25;                 // Villogás félperiódusának ideje
                                    // 20 ms egységekben (500 ms = 25*20 ms)
void setup() {
  pinMode(RED_LED,OUTPUT);          // Legyen kimenet
  pinMode(PUSH2,INPUT_PULLUP);      // Bemenet belső felhúzással
}
```

Folytatás a következő oldalon ...

A változók szerepe:

waitforpress: a nyomógomb *lenyomás – felengedés* ciklusának nyilvántartására használjuk (true = lenyomásra várunk, false = felengedésre várunk)

led_state: ebben tartjuk nyilván a LED állapotát (0 = nem világít, 1 = világít, 2 = villog)

timecount: a 20 ms-os időszeletek (vissza)számlálója villogtatáskor

Ez volt a led_tristate.ino program

```
const int RED_LED = 5;              // D5-re van kötve a LED
const int PUSH2 = 3;                // D3-ra van kötve a nyomógomb

// Hardverfüggetlen rész (MSP430 Launchpad kártyán is futtatható!)
boolean waitforpress = true;        // Kezdetben lenyomásra várunk
int led_state = 0;                  // Kezdetben a LED ki van kapcsolva
int timecount = 25;                 // Villogás félperiódusának ideje
                                    // 20 ms egységekben (500 ms = 25*20 ms)
void setup() {
  pinMode(RED_LED,OUTPUT);          // Legyen kimenet
  pinMode(PUSH2,INPUT_PULLUP);      // Bemenet belső felhúzással
}
```

Folytatás a következő oldalon ...

A változók szerepe:

waitforpress: a nyomógomb *lenyomás – felengedés* ciklusának nyilvántartására használjuk (true = lenyomásra várunk, false = felengedésre várunk)

led_state: ebben tartjuk nyilván a LED állapotát (0 = nem világít, 1 = világít, 2 = villog)

timecount: a 20 ms-os időszeletek (vissza)számlálója villogtatáskor

Ez volt a led_tristate.ino program

```
void loop() {
  if(waitforpress) {                  //Ha lenyomásra várunk és
    if(!digitalRead(PUSH2)) {         //Ha lenyomás történt...
      if(++led_state > 2) led_state = 0;     //Állapot léptetés
      digitalWrite(RED_LED, led_state > 0); //LED állapot beállítása
      waitforpress = false;           //Következő stáció: felengedésre várunk
    }
  }
  else {                              //Ha felengedésre vártunk és
    if(digitalRead(PUSH2)) {          //Ha felengedést észlelünk...
      waitforpress = true;            //Következő stáció: lenyomásra várunk
    }
  }
  delay(20);                          //pergésmentesítő késleltetés
  if(led_state == 2)                  //Ha villogtatás állapotban vagyunk
    if(--timecount == 0) {            //Ha az eltelt idő 25*20 ms, akkor
      digitalWrite(RED_LED,!digitalRead(RED_LED)); //LED átbillentése
      timecount = 25;                 //Következő billentés 25*20 ms múlva...
    }
}
```

KEYPRESS

BLINK

Ez volt a led_tristate.ino program

```
void loop() {
  if(waitforpress) {                  //Ha lenyomásra várunk és
    if(!digitalRead(PUSH2)) {         //Ha lenyomás történt...
      if(++led_state > 2) led_state = 0;     //Állapot léptetés
      digitalWrite(RED_LED, led_state > 0); //LED állapot beállítása
      waitforpress = false;           //Következő stáció: felengedésre várunk
    }
  }
  else {                              //Ha felengedésre vártunk és
    if(digitalRead(PUSH2)) {          //Ha felengedést észlelünk...
      waitforpress = true;            //Következő stáció: lenyomásra várunk
    }
  }
  delay(20);                          //pergésmentesítő késleltetés
  if(led_state == 2)                  //Ha villogtatás állapotban vagyunk
    if(--timecount == 0) {            //Ha az eltelt idő 25*20 ms, akkor
      digitalWrite(RED_LED,!digitalRead(RED_LED)); //LED átbillentése
      timecount = 25;                 //Következő billentés 25*20 ms múlva...
    }
}
```

KEYPRESS

A LED kezelése átkerül ide

BLINK

Ezt a részt átírjuk egy switch utasításra

Ez lett az új program: led_tristate2.ino

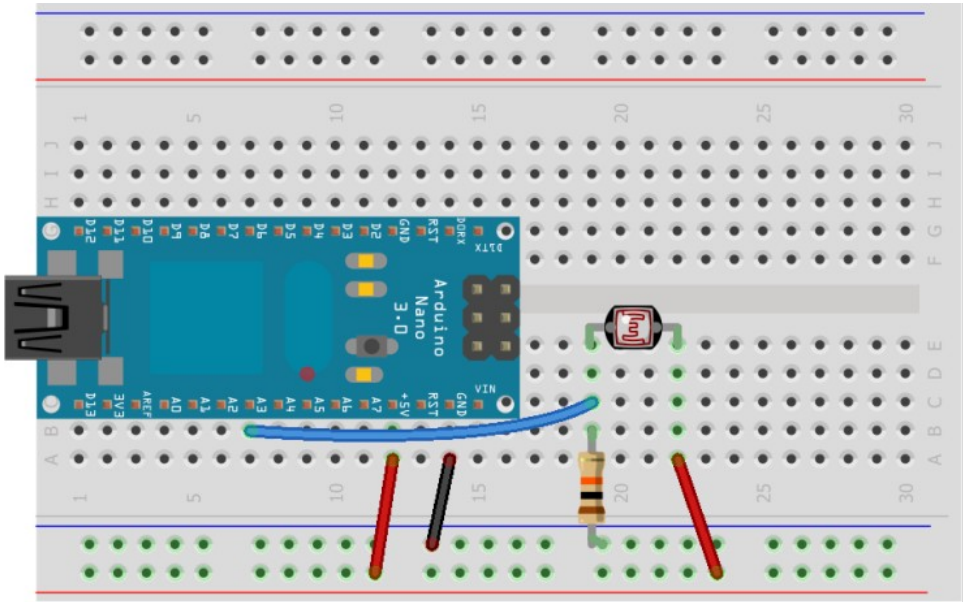
```
const int RED_LED = 5;
const int PUSH2 = 3;
boolean waitforpress = true;      //Kezdetben lenyomásra várunk
int led_state = 0;                //Kezdetben a LED ki van kapcsolva
void setup() {
  pinMode(RED_LED, OUTPUT);       //legyen kimenet
  pinMode(PUSH2, INPUT_PULLUP);   //Bemenet belső felhúzással
}
void loop() {
  if (waitforpress) {             //Ha lenyomásra várunk és
    if (!digitalRead(PUSH2)) {    //Ha lenyomás történt...
      if (++led_state > 2) led_state = 0; //Állapot léptetés (2 után körülfordulás...)
      waitforpress = false;       //Következő stáció: felengedésre várunk
    }
  }
  else {                          //Ha felengedésre vártunk és
    if (digitalRead(PUSH2)) {     //Ha felengedést észlelünk...
      waitforpress = true;        //Következő stáció: lenyomásra várunk
    }
  }
  switch (led_state) {
    case 0: digitalWrite(RED_LED, LOW); break;
    case 1: digitalWrite(RED_LED, HIGH); break;
    default: digitalWrite(RED_LED, !digitalRead(RED_LED));
  }
  delay(250);
}
```

Innen kikerült a LED kimenet vezérlése

Kompromisszum a „lusta” megoldáshoz

Fénymérés fényérzékeny ellenállással

- A kapcsolás feszültségosztóként működik, amelyikben a felső tag egy CdS fényérzékeny ellenállás, melynek ellenállása a megvilágítástól függően széles határok között változik. A megvilágítás hatására az ellenállása csökken…
- Az ellenállásosztó közös pontját az **A3** analóg bemenetre kötöttük
- Az eredményt soros porton kiküldjük a számítógépre, s a terminálablakban jelenik meg.



Photoresistor.ino

```
void setup () {
   Serial.begin(9600);
   analogReference(DEFAULT);   //VCC a referencia
   Serial.print("Photoresistor");
}

void loop () {
   long reading = 0;
   for(int i=0; i<4750; i++) {
      reading += analogRead(A3);
   }
   // Trükkös osztás 1024-gyel
   float voltage = reading>>10;
   Serial.print(voltage,0);
   Serial.print(" mV, ");
   // Átszámítás kOhm-ra
   // Rx = VCC*10k/voltage - 10k
   float rx = 47500/voltage - 10;
   Serial.print(rx,3);
   Serial.println(" kOhm");
   delay (5000);
}
```

4750 db mérés eredményét összeadjuk és nem szorzunk Vref-fel!

Photoresistor
3145 mV, 5.103 kOhm
3149 mV, 5.084 kOhm
3036 mV, 5.646 kOhm
1393 mV, 24.099 kOhm
1322 mV, 25.930 kOhm
4406 mV, 0.781 kOhm
4449 mV, 0.677 kOhm
4134 mV, 1.490 kOhm
2856 mV, 6.632 kOhm
2836 mV, 6.749 kOhm

Az érzékelési tartomány sávokra osztása

- Módosítsuk az előző programot úgy, hogy az ADC-vel mért értékeket sávokra osztjuk fel, s az eredmény szöveges értékelését kiíratjuk a soros porton!
- Legyenek a kiírandó szövegek:
  - sötét
  - derengő fény
  - közepes fény
  - erős fény
- Az eredmény szöveges értékeléséhez használjuk a most tanult **switch** utasítást!

switch_case.ino

```
const int sensorMin = 0;          // szenzor minimum (tapasztalat alapján)
const int sensorMax = 800;        // szensor maximum (tapasztalat alapján)
void setup() {
  Serial.begin(9600);             // Soros port inicializálása
}

void loop() {
  int reading = analogRead(A3); // Szenzor kiolvasás
  // A mérési tartomány leképezése négy esetre
  int range = map(reading, sensorMin, sensorMax, 0, 3);
  switch (range) {
    case 0:     // Eltakart szenzor esetén
      Serial.println("sötét");
      break;
    case 1:     // kezed közel a szenzorhoz
      Serial.println("derengő fény");
      break;
    case 2:     // kezed 10 cm-re a szenzortól
      Serial.println("közepes fény");
      break;
    case 3:     // kezed nincs a fény útjában
      Serial.println("erős fény");
      break;
  }
  delay(1000);  // Késleltetés a kiíratások között
}
```