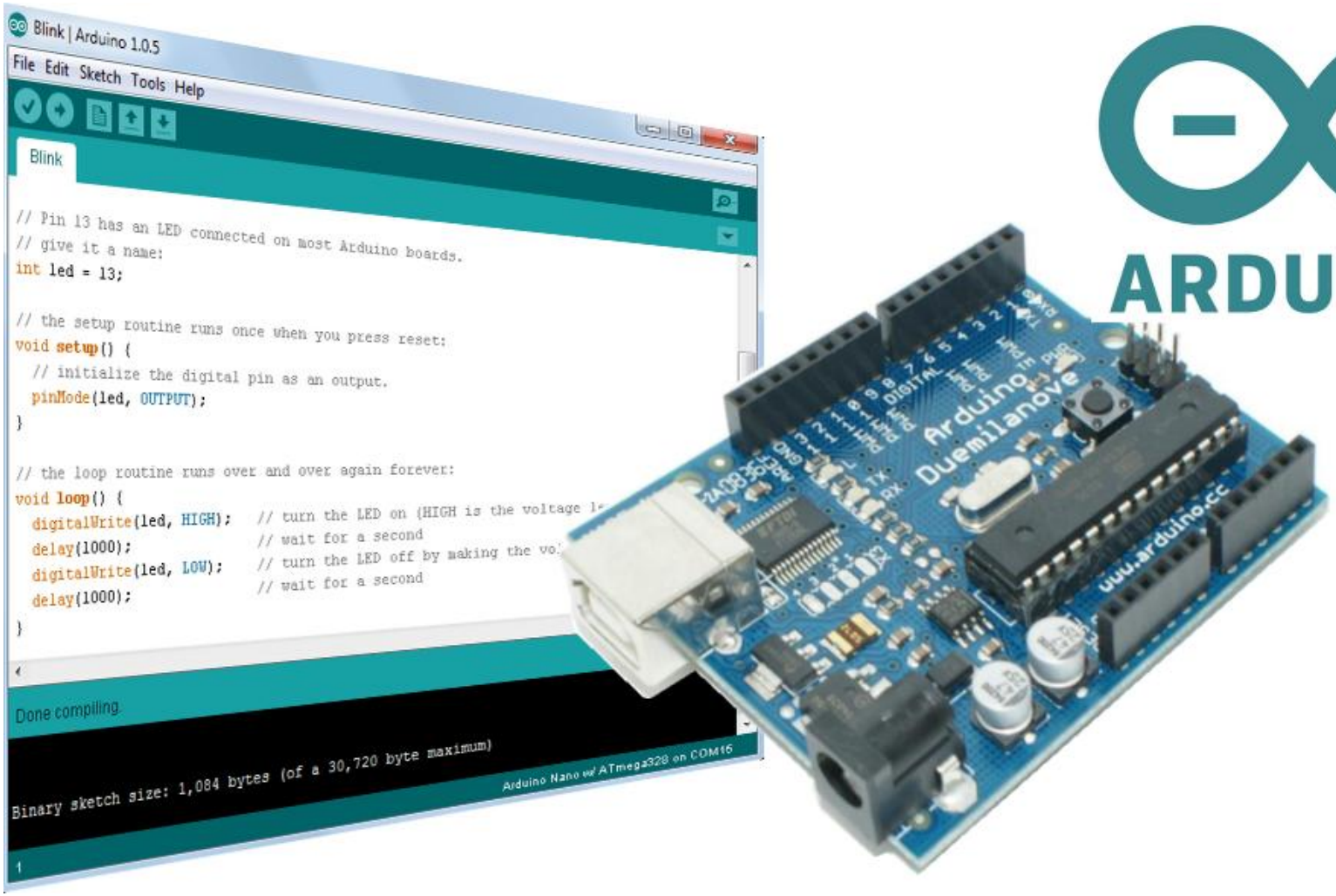


Bevezetés a mikrovezérlők programozásába:

Programciklusok szervezése, analóg I/O

Ajánlott irodalom

- Aduino LLC.: Arduino Language Reference
- ATMEL: ATmega328p mikrovezérlő adatlapja
- Brian W. Kernighan, Dennis Ritchie: A C programozási nyelv
- Cseh Róbert: Arduino programozási kézikönyv
- Ruzsinszki Gábor: Mikrovezérlős rendszerfejlesztés C/C++ nyelven I. – PIC mikrovezérlők
- Ruzsinszki Gábor: Mikrovezérlős rendszerfejlesztés C/C++ nyelven II. – Arduino

Lab 13 projektek

- **AnalogThermometer** – Hőmérséklet mérése MCP9700 analóg hőmérő szenzorral
- **AnalogThermometer2** – Hőmérséklet mérése MCP9700 analóg hőmérő szenzorral, átlagolással
- **AnalogThermometer3** – Hőmérséklet mérése MCP9700 analóg hőmérő szenzorral, átlagolással, grafikus kijelzéssel
- **Photoresistor** – Fény érzékelése fényérzékeny ellenállás segítségével
- **RGB_led** – RGB LED színeinek véletlenszerű váltogatása, folytonos színátmenettel (PWM mintapélda)

Vezérlési szerkezetek

Egy nyelv vezérlésátadó utasításai az egyes műveletek végrehajtási sorrendjét határozzák meg. Ebben az előadásban az alábbi szerkezetek tulajdonságairól lesz szó:

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

A talk12 előadásban tárgyalt vezérlési szerkezetek

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

Ciklusszervezés while utasítással

while (*feltétel*) *utasítás vagy blokk* *(elöltesztelő ciklus…)*

A *feltétel* itt egy kifejezést jelent, melynek bármely nullától különböző értéke „igaz” jelentéssel bír. Ha a feltétel igaz, a **while** utasításhoz kapcsolódó utasítás (vagy blokk) végrehajtásra kerül, majd a végrehajtás visszakerül a feltétel kiértékeléséhez. A programciklus addig ismétlődik, amíg a feltétel hamissá (nulla értékűvé) nem válik. Példák:

1.
```
while (true) { //LED villogtatás végtelen ciklusban
  digitalWrite(LED,!digitalRead(LED));
  delay(500);
}
```

2.
```
while (false)digitalWrite(LED, HIGH); //Sohasem hajtódik végre!
```

3.
```
int i = 0;
while (i < 5) {        //ötszöri LED villantás
  digitalWrite(LED,HIGH);
  delay(50);
  digitalWrite(LED,LOW);
  delay(500);
  i++;                 //Ciklusváltozó léptetése (FONTOS!)
}
```

Nyomógomb kezelése mégegyszer

```
const int RED_LED = 5;
const int PUSH2 = 3;
```

Hardverfüggő rész,
csak az Arduino kártyához kell!

```
void setup() {
  pinMode(RED_LED,OUTPUT);          //legyen kimenet
  pinMode(PUSH2,INPUT_PULLUP);      //Bemenet belső felhúzással
}

void loop() {
#  while(digitalRead(PUSH2);        //Lenyomásra várunk...
   digitalWrite(RED_LED,!digitaRead(RED_LED)); //LED átbillentés
   delay(20);                       //pergésmentesítő késleltetés
#  while(!digitalRead(PUSH2);       //Felengedésre várunk...
   delay(20);                       //pergésmentesítő késleltetés
}
```

Ez a múlt órai **ledswitch.ino** program egyszerűsített megvalósítása, **while** utasítás segítségével.
Előnye: egyszerű, áttekinthető struktúra
Hátránya: blokkoló várakozások, amelyek megakasztják a programot *(#-vel jelölt sorok)*

Ciklusszervezés for utasítással

kezdőérték feltétel ciklusváltozó léptetés

for (*1. kifejezés; 2. kifejezés; 3. kifejezés*) *utasítás* – ami egyenértékű az alábbival:

```
1. kifejezés;
while(2. kifejezés) {
        3. kifejezés;
        utasítás
}
```

*(a **for** ciklus is elöltesztelő ciklus…)*

Az előző oldal 3. példáját például így írhatjuk át:

kezdőérték
feltétel

```
for(i = 0; i < 5; i++) {
  digitalWrite(LED,HIGH);
  delay(50);
  digitalWrite(LED,LOW);
  delay(500);
}
```

FOR ciklus

```
int i = 0;
while (i < 5) {
  digitalWrite(LED,HIGH);
  delay(50);
  digitalWrite(LED,LOW);
  delay(500);
  i++;
}
```

ciklusváltozó léptetés

WHILE ciklus

Ciklusszervezés do - while utasítással

A **while** és **for** utasításokkal szemben a **do - while** utasítás ún. *hátultesztelő* ciklust szervez, azaz a ciklusmag legalább egyszer végrehajtásra kerül. Általános formája:

```
do
  utasítás
while (kifejezés);
```

A mikrovezérlő először végrehajtja az *utasítást* és csak utána értékeli ki a *kifejezést*. Ha a kifejezés értéke igaz, az utasítás újból végrehajtódik. Ez ismétlődik mindaddig, amíg a kifejezés értéke hamis nem lesz, ekkor a ciklus lezárul és a végrehajtás az utána következő utasítással folytatódik. Példa:

```
/* uitoa: az n előjel nélküli számot s karaktersorozattá alakítja */
void uitoa(int n, char s[]) {
   int i;
   i = 0;
   do { /* fordított sorrendben generálja a számjegyeket */
      s[i++] = n % 10 + '0'; /* a következő számjegy */
   } while((n /= 10) > 0);   /* törli azt */
   s[i] = '\0';
   reverse(s);
}
```

A break és continue utasítások

A **break** utasítás lehetővé teszi a **for, while** vagy **do** utasításokkal szervezett ciklusok idő előtti elhagyását, valamint a **switch** utasításból való kilépést. A **break** hatására a legbelső ciklus vagy a teljes **switch** utasítás fejeződik be.

Példa:

```
for (x = 0; x < 255; x ++) {
    analogWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold)break;
    delay(50);
}
```

Kilépünk a ciklusból, ha elértük a küszöböt

A **continue** utasítás lehetővé teszi a ciklusmag hátralevő részének átugrását (a programvégrehajtás a ciklusváltozó léptetéséhez ugrik).

```
for (x = 0; x < 255; x ++) {
    if (x > 40 && x < 120) continue; //így pl. kihagyunk egy sávot
    analogWrite(PWMpin, x);
    delay(50);
}
```

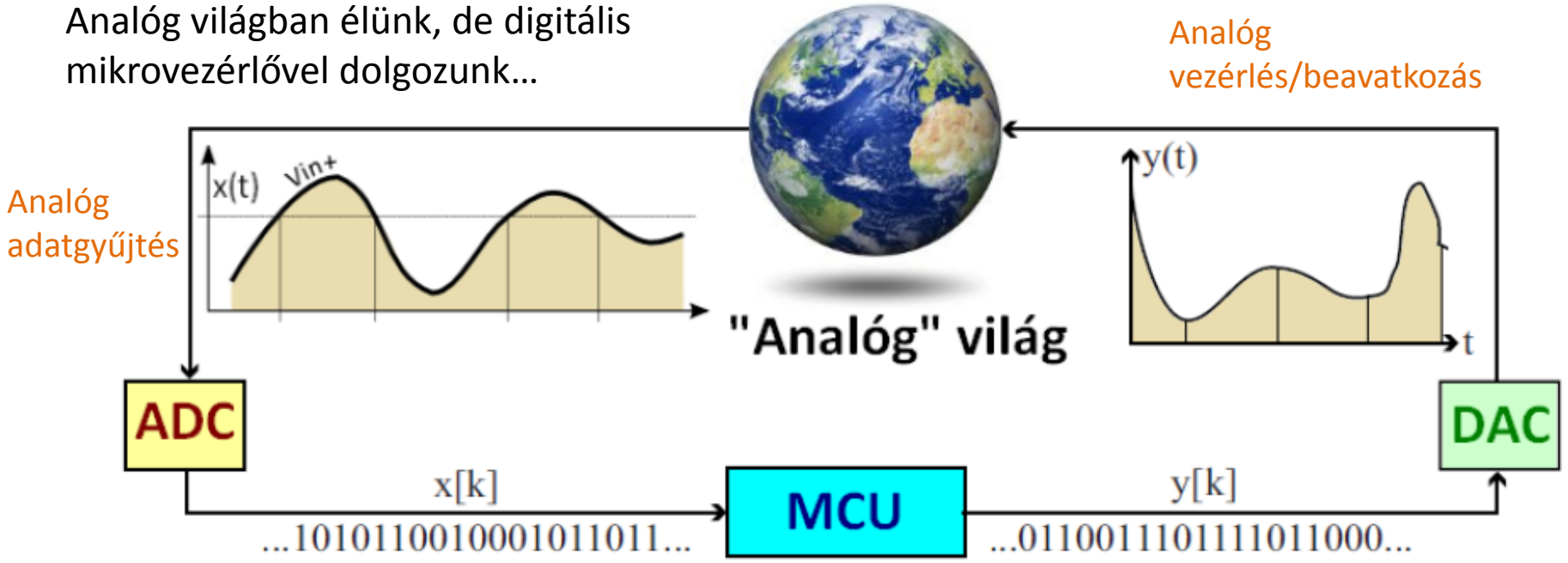
A goto utasítás és a címkék

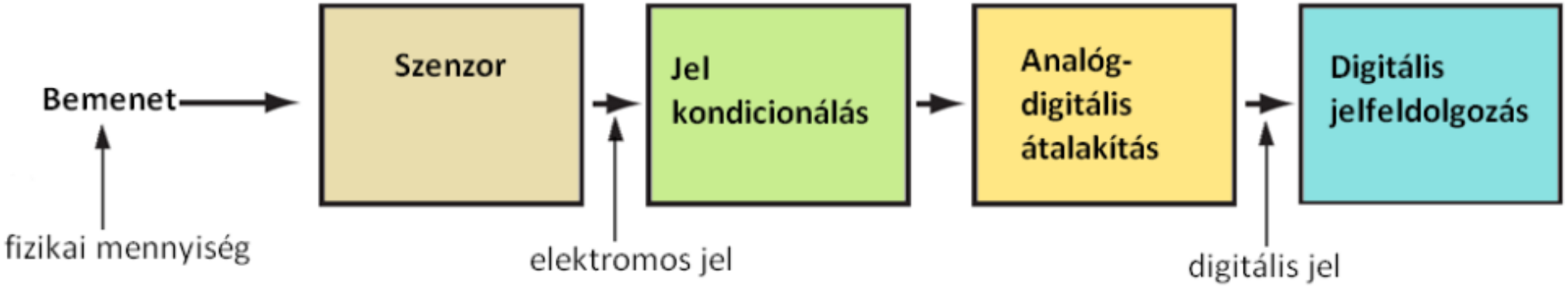
A **goto** utasítás segítségével megadott címkékre ugorhatunk. Nagyon ritkán használjuk, Például többszörös ciklusból történő kiugráshoz (ahol a **break** utasítás nem használható, mert csak a legbelső ciklusból ugrana ki…)

```
for(...)
    for(...) {
        ...
        if (zavar)
            goto hiba;          <-----------    //ugrás
    }
    ...
hiba:                           <-----------    //címke
    a hiba kezelése
```

Analóg jelfeldolgozás

Analóg világban élünk, de digitális mikrovezérlővel dolgozunk…





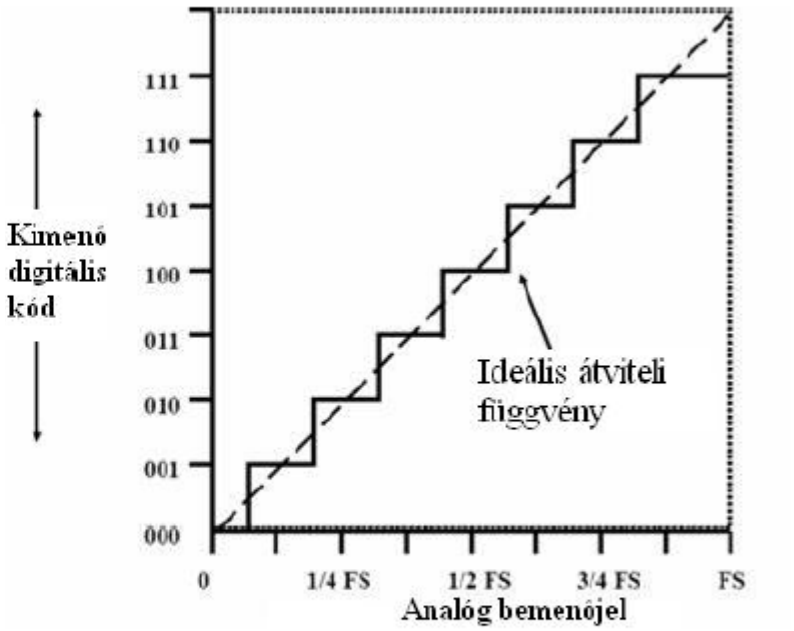
Az analóg adatgyűjtő ág elemei

- **Szenzor:**
  - A folytonos fizikai mennyiségeket (pl. hőmérséklet, nyomás, páratartalom, sebesség, áramlási sebesség, elmozdulás, gyorsulás, szöggyorsulás) elektromos jellé alakítja (feszültséggé vagy árammá).
- **Jel kondicionálás (szűrés, erősítés, stb.):**
  - A mérendő mennyiség elektromos jellé alakítása után még szűrésre, jel erősítésre, impedancia illesztésre is szükség lehet, hogy az analóg-digitális átalakító (ADC) bemeneti tartományába transzformáljuk az átalakítandó jelet.
- **Analóg-Digitális Átalakító (ADC):**
  - *Bemenet:* a mérendő jel
  - *Kimenet:* a mérendő jellel arányos számot reprezentáló digitális kód

Analóg–digitális átalakító (ADC)

Az ADC feladata, hogy diszkrét digitális kóddá alakítsa a bejövő folytonos (analóg) jelet

Egy 3-bites átalakító ideális átviteli függvénye



A konverzió digitális értéke (N_{ADC}):

Végkitérés: N_{ADC}= 1023, ha a bemenő jel ≥ V_{R+} -0.5*LSB
Nulla: N_{ADC} = 0, ha a bemenő jel ≤ V_{R-} + 0.5 LSB
Közbeeső értékekre: $\mathbf{N_{ADC} = 1023 * (V_{IN} - V_{R-})/(V_{R+} - V_{R-})}$

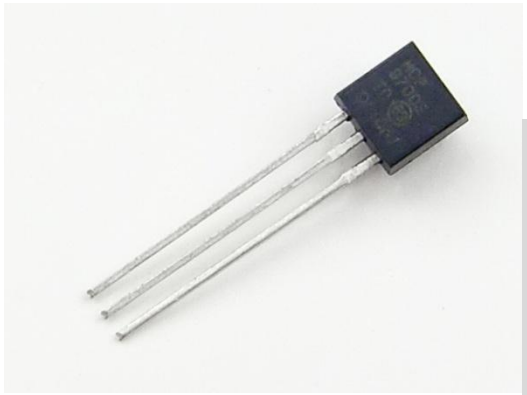
$\mathbf{V_{R-}}$ esetünkben = 0 V (**VSS**, a közös pont)
$\mathbf{V_{R+}}$ az, amit az `analogReference()` függvényhívással beállítunk (VCC, belső- , vagy külső referencia)

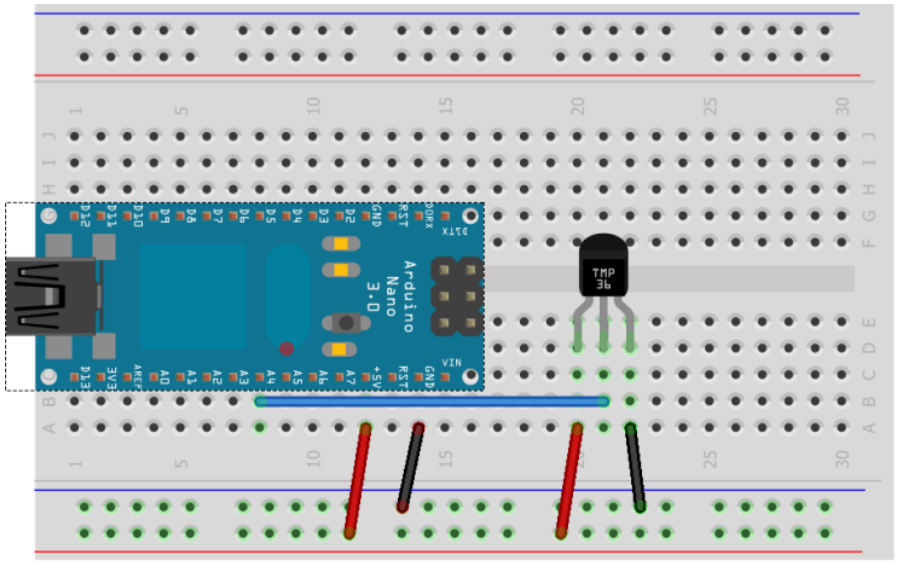
Többnyire a konverzió $\mathbf{N_{ADC}}$ eredményéből a $\mathbf{V_{IN}}$ bemenő feszültséget kell meghatároznunk. A fenti képletből $\mathbf{V_{IN}}$-t kifejezve, ezt kapjuk:

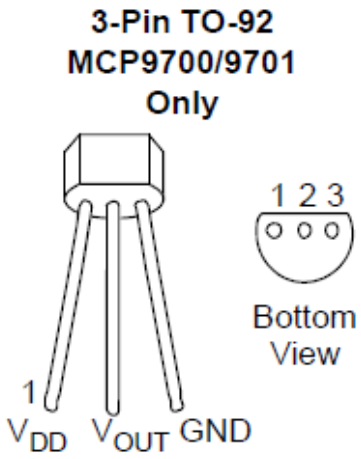
$$\mathbf{V_{IN} = (V_{R+} - V_{R-}) * N_{ADC}/1023 + V_{R-}}$$

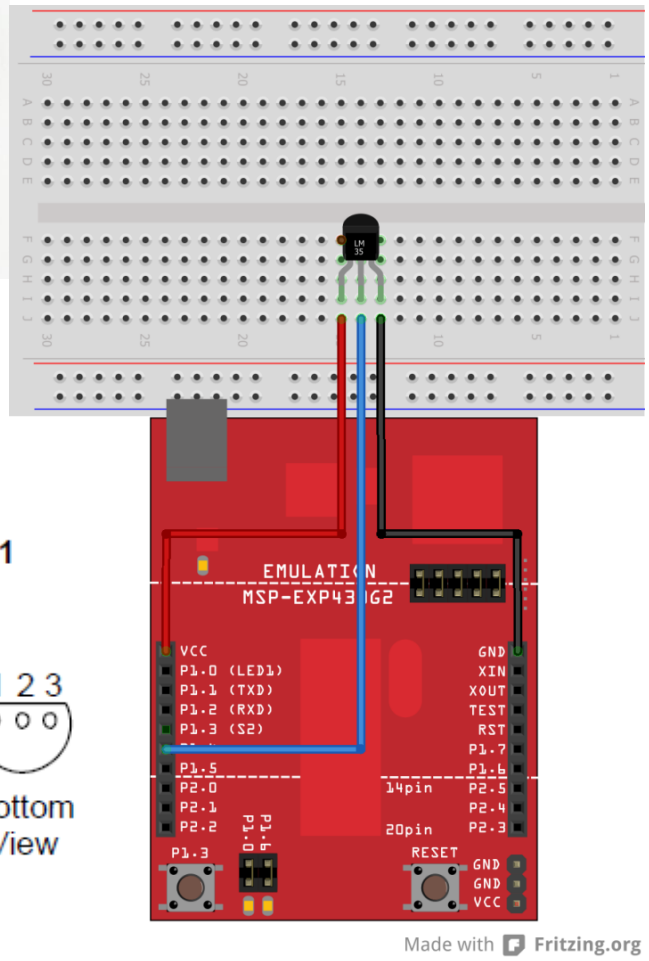
Hőmérés analóg hőmérővel

Microchip MCP9700
VDD = 2,5 – 5,5 V
Mérési tart.: -40 – 150 °C
Érzékenység: 10 mV / °C
Nullapont: 500 mV @ 0 °C









AnalogThermometer.ino

```
void setup() {
  Serial.begin(9600);                 //Soros kapcsolat 9600 bit/s
  analogReference(INTERNAL);          //1,1 V-os belső referencia
}

void loop() {
  delay(1000);                        //egy kis várakozás
  int reading = analogRead(A4);       //a hőmérő leolvasása

  //--- millivolt -------
  float voltage = ((long)reading * 1100) / 1023;
  Serial.print (voltage,0);           //Kiírás mV-okban
  Serial.print (" mV, ");
  //--- Celsius ---------
  float tempC = (voltage - 500)/10; //Átszámítás C fokokra
  Serial.print (tempC,1);
  Serial.print (" C, ");
  //--- Fahrenheit ------
  float tempF = (tempC * 9/5) + 32; //Átszámítás F fokokra
  Serial.print (tempF,1);
  Serial.println (" F");
  delay (1000);
}
```

Vref

ADC felbontása

Megjegyzés: Arduino-nál 1.1V, az MSP430 Launchpad esetében pedig 1.5V a belső referencia, ezért az utóbbinál INTERNAL helyett INTERNAL1V5-öt, 1100 helyett pedig 1500-at kell írnunk!

AnalogThermometer2.ino

A zavaró ingadozások kiszűrésére **sok mérést végzünk**, és a kapott adatokat **átlagoljuk.**
Az összegző változó *long* (vagy uint32_t) típusú legyen, nehogy túlcsorduljon!

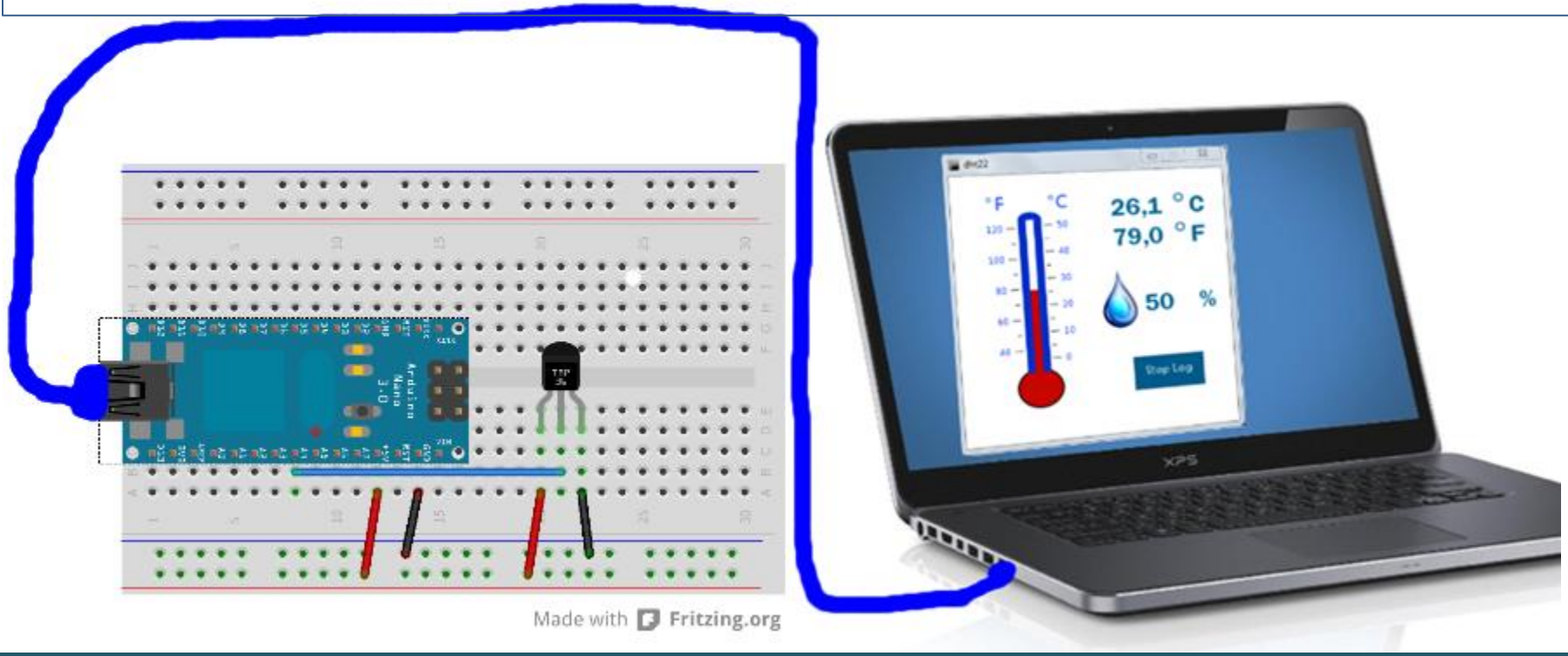
```
long mysum;             //ebben összegezzük az eredményt
void setup() {
  Serial.begin(9600);                //Soros kapcsolat 9600 bit/s
  analogReference(INTERNAL);         //1,1 V-os belső referencia
}

void loop() {
   //---  Mérés átlagolással ---
   mysum = 0;
   for(int i=0; i<1100; i++) {
      mysum += analogRead(A4);      //Rendre hozzáadjuk az új kiolvasást
   }
  //--- Átszámítás mV-okra ----
  float voltage = mysum>>10;        //Osztás 1024-gyel
  //A program többi része változatlan ...
=========================================
}
```

Megjelenítés a PC-n

Ebben a változatban csak a kiíratás formátumát változtatjuk meg, hogy az egy hasonló projekthez készült TRHlogger alkalmazással kompatibilis legyen. A hőmérés eredményét a PC-n futó alkalmazás grafikusan megjeleníti és kívánságra naplózza is.

A Lab13.zip csomagban mellékelt TRHlogger alkalmazás csak akkor működik, ha az Arduino a legkisebb sorszámú soros port a rendszerben a program indításakor (a program automatikusan az elsőhöz próbál csatlakozni).

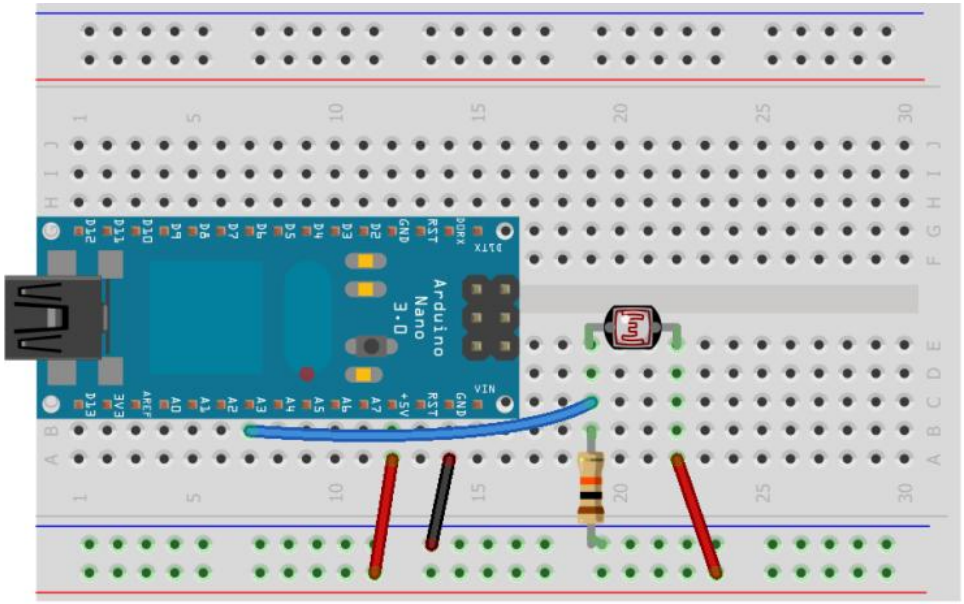


Fényérzékeny ellenállás

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 kössük!

Az eredményt soros porton kiküldjük a számítógépre, s a terminálablakban jelenik meg.



Photoresistor.ino

Hardverfüggő paraméter: Arduino esetén 5000, MSP430 esetén 3500

```
long mysum;                     //ebben összegezzük majd az eredményt
const int VCC = 5000;           //A tápfeszültség mV-ban
void setup() {
  Serial.begin(9600);           //Soros kapcsolat 9600 bit/s
  analogReference(DEFAULT);     //VCC a referencia
}

void loop() {
  //---  Mérés átlagolással ---
  mysum = 0;
  for(int i=0; i<VCC; i++) {
     mysum += analogRead (A3);  //Rendre hozzáadjuk az új kiolvasást
  }
  //--- Átszámítás mV-okra ----
  float voltage = mysum>>10;    //Osztás 1024-gyel
  Serial.print (voltage,0);     //Feszültség kiírása
  Serial.print (" mV ");
  // Átszámítás kOhm-ra      Rx = VCC*10k/voltage - 10k
  float rx = 10*VCC/voltage - 10;
  Serial.print(rx,3);
  Serial.println(" kOhm");
  delay (2000);
}
```

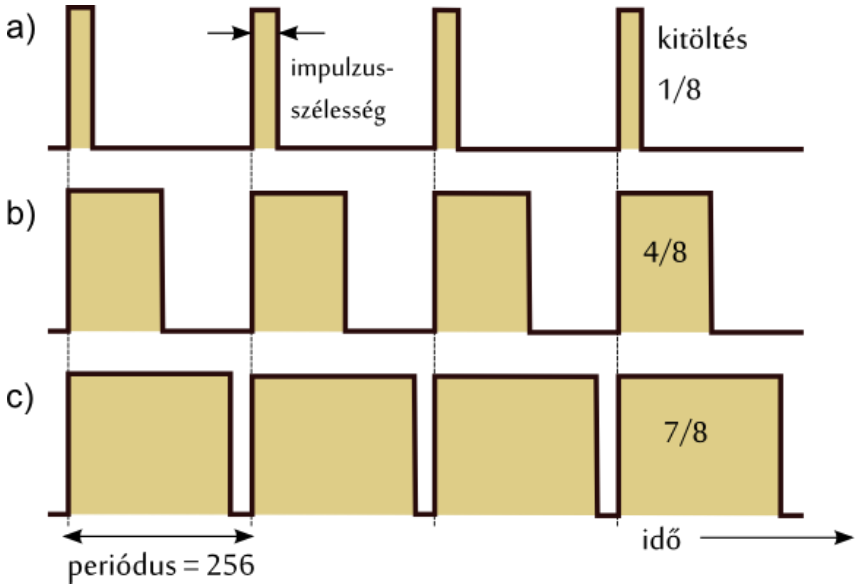
PWM: impulzus-szélesség moduláció

PWM = pulse width modulation (impulzus-szélesség moduláció)

A frekvencia állandó,

a kitöltés változtatható 0- 255 között.

Az **analogWrite()** függvény csak bizonyos lábakra vonatkozóan használható



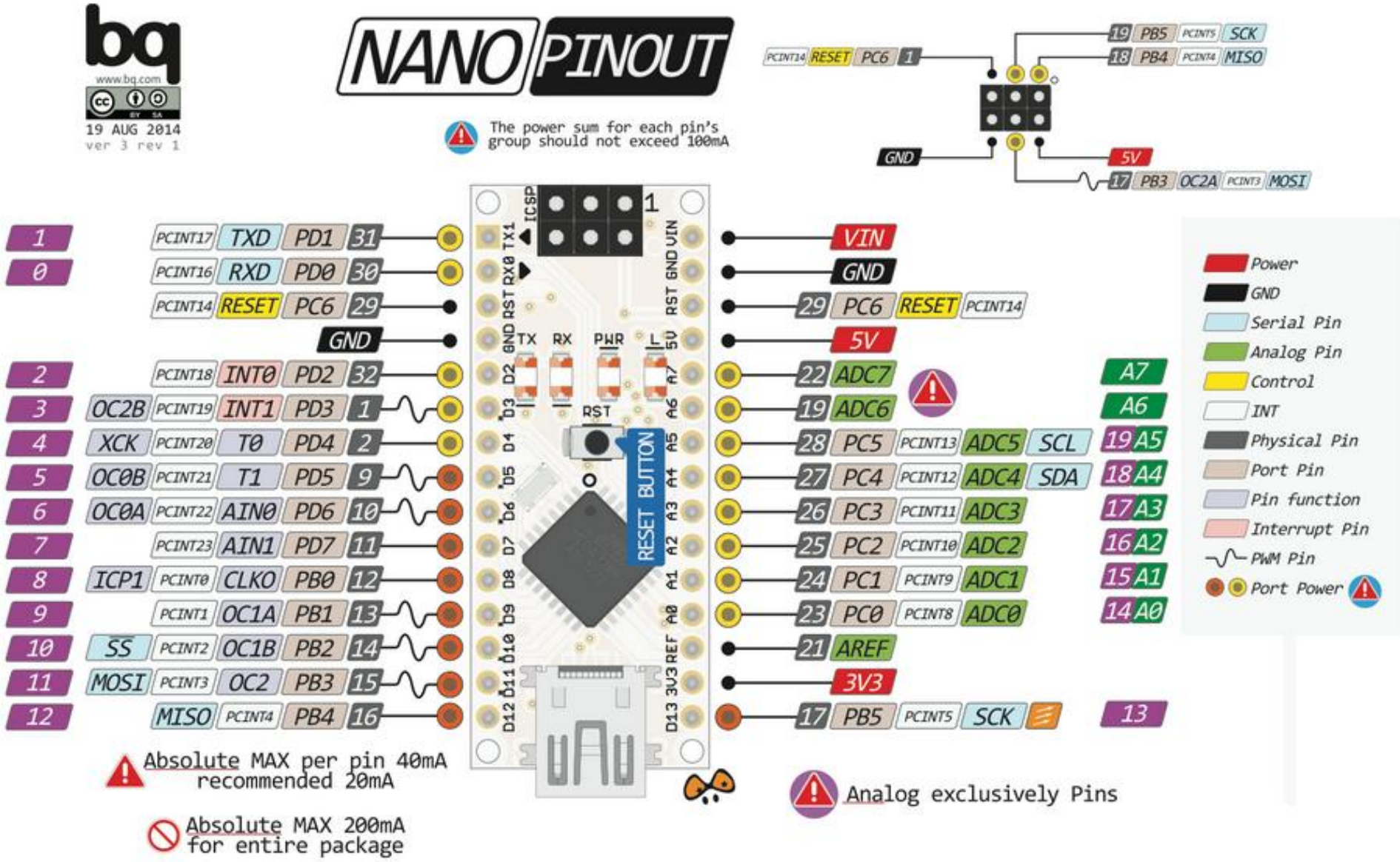
Arduino

Időzítő	Csatorna	Digitális kimenet
Timer0	OC0A, OC0B	6,5
Timer1	OC1A, OC1B	9,10
Timer2	OC2A, OC2B	11,3

MSP430 Launchpad

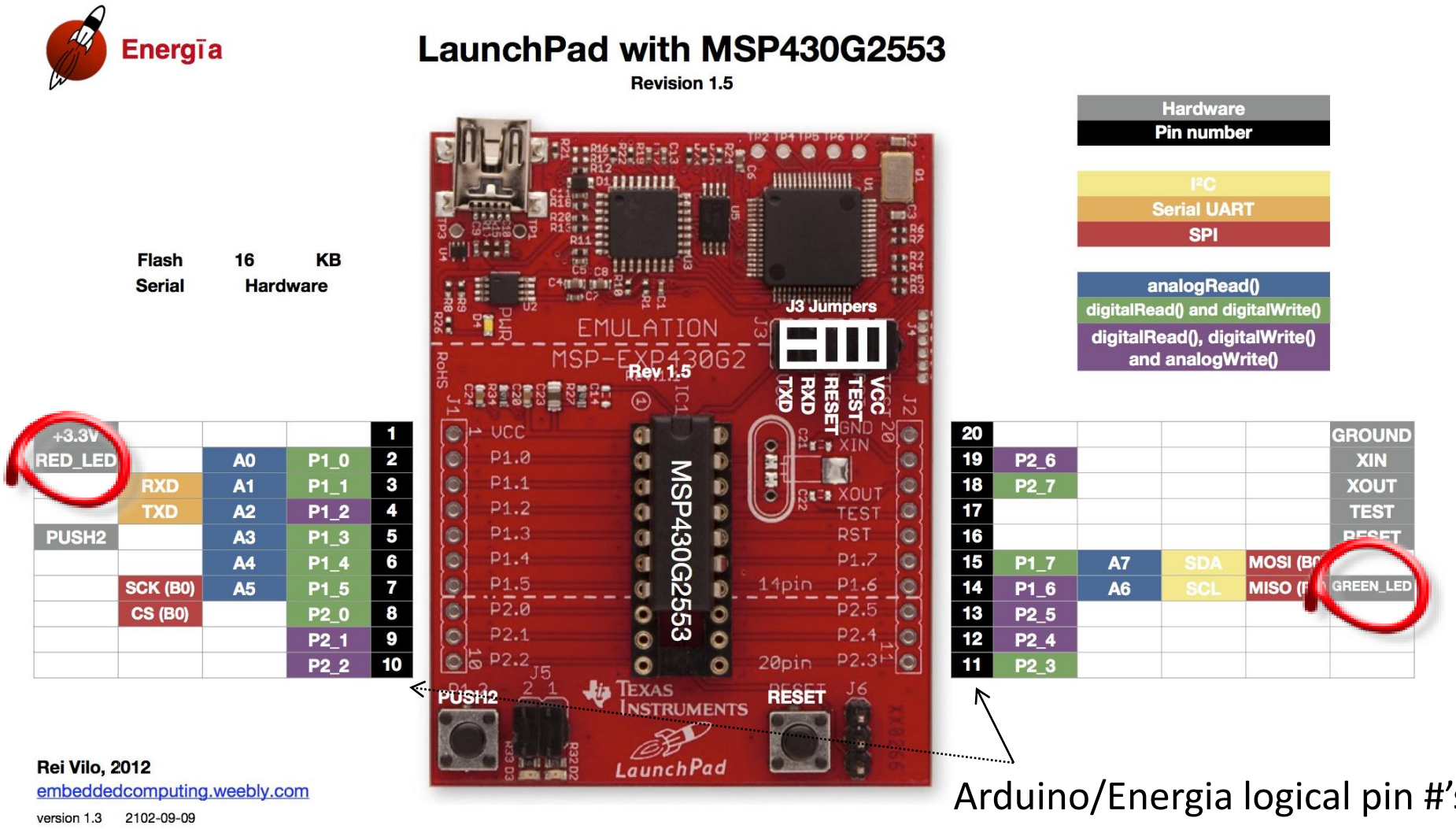
Időzítő	Csatorna	Választható kivezetés
Timer0	TA0.1	P1_2, P1_6 vagy P2_6
Timer1	TA1.1	P2_1 vagy P2_2
Timer1	TA1.2	P2_4 vagy P2_5

Emlékeztető: Arduino nano v3.0



MSP430 Launchpad : Energia Pinout

http://github.com/energia/Energia/wiki/Hardware

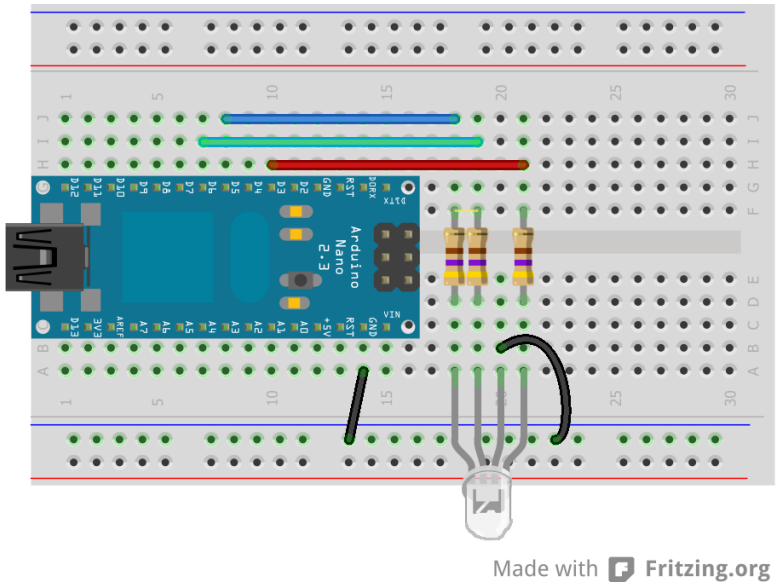


Rei Vilo, 2012
embeddedcomputing.weebly.com
version 1.3 2102-09-09

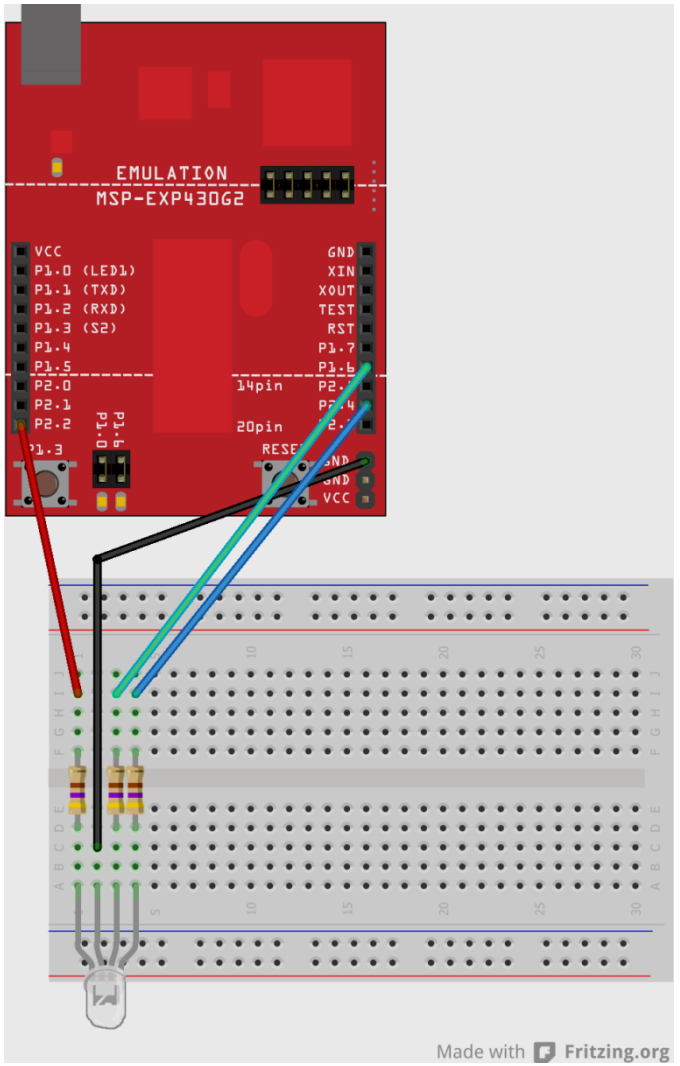
Arduino/Energia logical pin #'s

Huzalozási vázlat

R LED = D3
G LED = D6
B LED = D5



R LED = P2_2
G LED = P1_6
B LED = P2_4



RGB_led.ino program

```
#define LED_RED   3          //Launchpad: P2_2
#define LED_BLUE  5          //Launchpad: P2_4
#define LED_GREEN 6          //Launchpad: P1_6

float RGB1[3];
float RGB2[3];
float INC[3];
float SUM;
int red, green, blue;

void setup() {
//--- Kiindulási színek generálása, véletlenszerűen
  randomSeed(analogRead(0));
  for (int x=0; x<3; x++) {
    RGB1[x] = random(256);
    RGB2[x] = random(256);
  }
}
```

RGB_led.ino program

```
void loop() {
//--- Az 1. színből a 2. színbe történő átmenet lépéseinek meghatározása
  for (int x=0; x<3; x++) {
    INC[x] = (RGB1[x] - RGB2[x]) / 256;
  }
//--- Közelítés 256 lépésben
  for (int s=0; s<256; s++) {
    red = int(RGB1[0]);
    green = int(RGB1[1]);
    blue = int(RGB1[2]);
//--- A közelítő szín kijelzése
    analogWrite (LED_RED, red);
    analogWrite (LED_GREEN, green);
    analogWrite (LED_BLUE, blue);
    delay(20);
    for (int x=0; x<3; x++) {
      RGB1[x] -= INC[x];   //Közelítünk a másik színhez
    }
  }  //folyt. köv.
}
```

RGB_led.ino program

```
//--- Új színt konstruálunk, véletlen számok generálásával
  do { //Ciklikus ismétlés, míg SUM >= 300 nem teljesül
    SUM = 0;
    for (int x=0; x<3; x++) {
      RGB2[x] = random(956)-700;
      RGB2[x] = constrain(RGB2[x], 0, 255);
      SUM += RGB2[x];
    }
  }
  while(SUM < 300);          //Nem engedjük túlságosan elsötétedni
}
```

Alkalmazás a gyakorlatban: Hangulatlámpa folytonos színátmenettel

Projektleírás és a képek forrása:
http://users.atw.hu/wattmep/RGB_LED_627A/RGB_LED_627A.html