Gépészmérnöki és Informatikai Kar
Robert Bosch Mechatronikai Intézeti Tanszék

# Mechatronikai laboratóriumok 1.

## GEMRB011-B (4 kredit)

**2ó ea.+2ó gy.** (Gyakorlati jegy)

**Oktatók:**

**Lénárt József**
**egyetemi tanársegéd**

**Rónai László**
**PhD hallgató**

**A tananyagot összeállította: Rónai László**

**Robert Bosch Mechatronikai Intézeti Tanszék**

**2019**

# Bevezetés

- A tananyag elsősorban az Arduino fejlesztő platformon található mikrovezérlők programozásával, azon belül is az Atmega328 típusú mikrokontrollerrel foglalkozik.
- A szükséges elméleti ismereteken felül gyakorlati feladatokat is tartalmaz a segédlet.
- Az előadás- és a gyakorlati anyagot összeállította: Rónai László, PhD hallgató

AZ EMBERI ERŐFORRÁSOK MINISZTÉRIUMA ÚNKP-17-3 KÓDSZÁMÚ ÚJ NEMZETI KIVÁLÓSÁG PROGRAMJÁNAK TÁMOGATÁSÁVAL KÉSZÜLT.

# A félévvel kapcsolatos információk

- Szeptember 09-től November 8-ig tart a szorgalmi időszak
- Összesen: 9 hét
- Félév zárása:
  - Gyakorlaton programozási feladat megoldása
  - Zárthelyi az órán leadott tananyagból
  - Aláírás + Gyakorlati jegy
- A tervek szerint a 8. héten lesz a zárthelyi és a programozási feladat

# Ajánlott irodalom

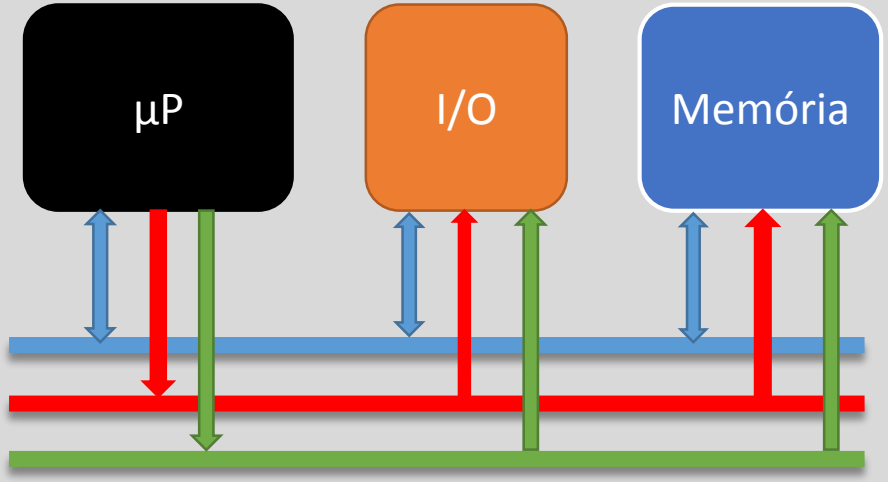
- Gárdus Z.: Digitális rendszerek szimulációja, Bíbor Kiadó, 2009
- Lambert M.: Szenzorok-elmélet, Budapest, ISBN 978-963-87401-1-3, 2009
- Brian W. E.: Arduino programozási kézikönyv, (Cseh R. fordította), Budapest, 2011
- ATmega328 Datasheet
- Arduino IDE elérhetősége: https://www.arduino.cc/
- Fritzing szoftver elérhetősége: http://fritzing.org/home/

# Mikrovezérlő

- A mikrovezérlő felfogható egy rendszer a lapkán (SoC) áramkörnek. Tartalmaz egy, vagy több CPU-t, ezen felül memóriával, saját buszrendszerrel, illetve ki- és bemeneti portokkal rendelkezik. Vezérlési feladatokra tervezett „célszámítógép”, amely kis fogyasztással rendelkezik. A beágyazott rendszerek lelkét alkotják. Általában a RISC architektúra jellemzi őket.
- Mikroprocesszor: Egy olyan regiszter alapú integrált áramkör, amely az utasítások értelmezéséért és végrehajtásáért felelős.
- 1971: Első mikroprocesszor: Intel 4004 → 70-es évektől megjelennek a mikrovezérlők (Texas Instruments számológépek)
- 1993: EEPROM megjelenése

# Mikroprocesszor részei

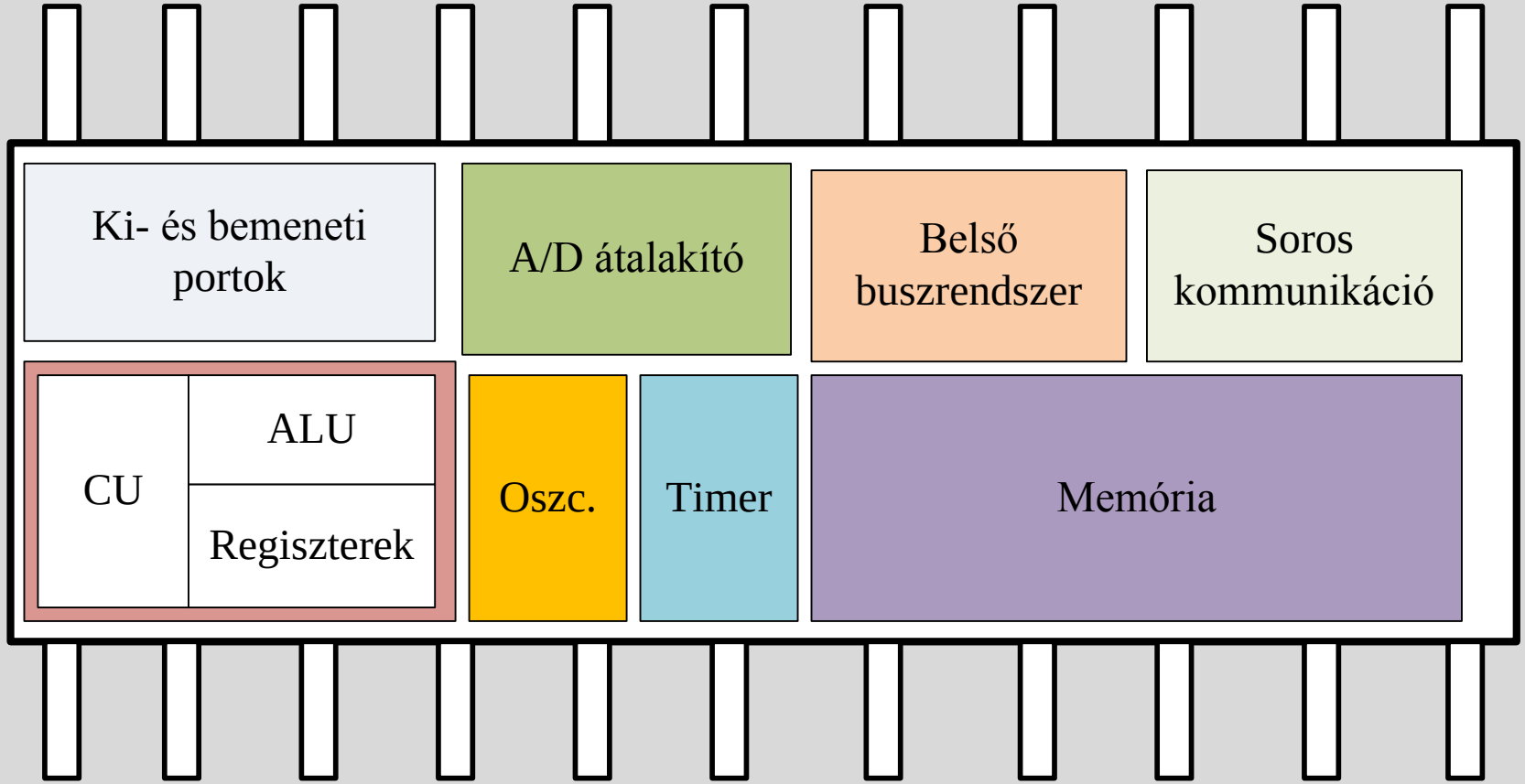
- CU: A vezérlőegység értelmezi az utasításokat és a vezérlőjeleket előállítja a végrehajtáshoz
- ALU: Aritmetikai- és a logikai műveletek végrehajtása
- Regiszterek: A számuk processzoronként változó
- (Cache memória): Gyorsítótár, amely eltérő sebességű eszközöknél célszerű. A lassabb eszköz tartalmának egy része a cache-be kerül, így a CPU gyorsabban el tudja azt érni egy következő olvasásnál.




- Belső BUSZ rendszer részei: Ez lehet adat-, cím-, és vezérlő busz
  - Adatbusz: kétirányú, adat továbbításhoz
  - Címbusz: egyirányú, adatok továbbítási címe számára
  - Vezérlő busz: processzor kialakítás függő, művelet irányításához

# Mikrovezérlő

- **A mikrokontroller általános felépítése:**
  - **Memória: ROM, RAM, PROM, stb.**
  - **Az egységeket sínrendszer köti össze**
  - **Analóg/Digitál átalakító**
  - **I/O portok**
  - **Időzítők számlálók**
  - **Watchdog, stb.**

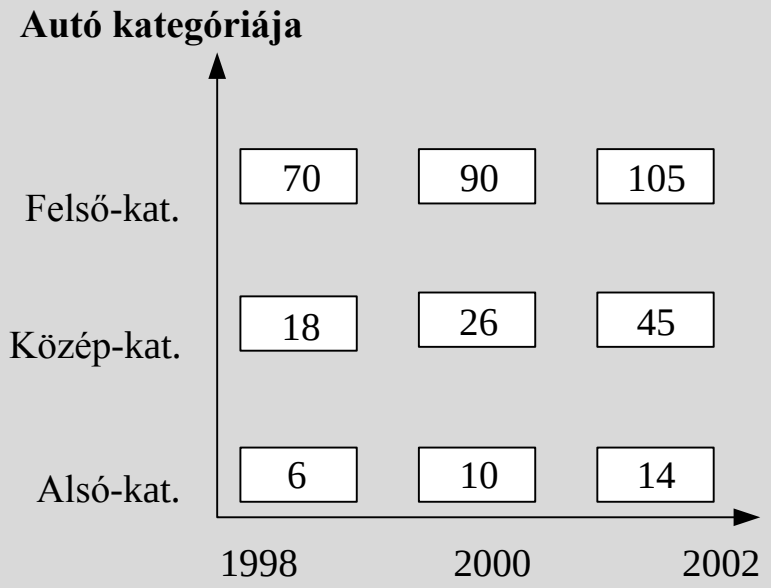


# A mikrovezérlő jellemzői

- Egy asztali számítógéphez viszonyítva:
  - Lényegesen kisebb fogyasztás,
  - Kisebb memória és tárhely kapacitás,
  - Általában RISC architektúra,
  - Kisebb számítási teljesítmény jellemzi őket.

# Felhasználási területei

- Szinte minden elektromos berendezésben található mikrokontroller, néhány példa:
  - Háztartási berendezések: szoba-mérleg, TV, stb.
  - Gépjármű: különböző szenzorok által mért jelek feldolgozására, kiértékelésére
  - Ipari alkalmazások: robotizálás, automatizálás
  - Katonai felhasználás
  - Űrtechnika, stb.

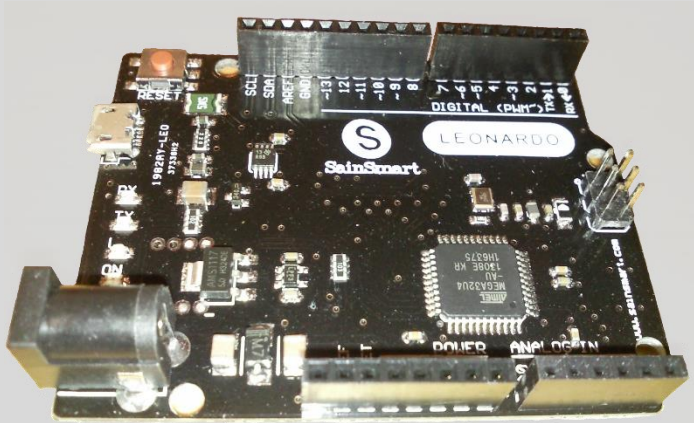


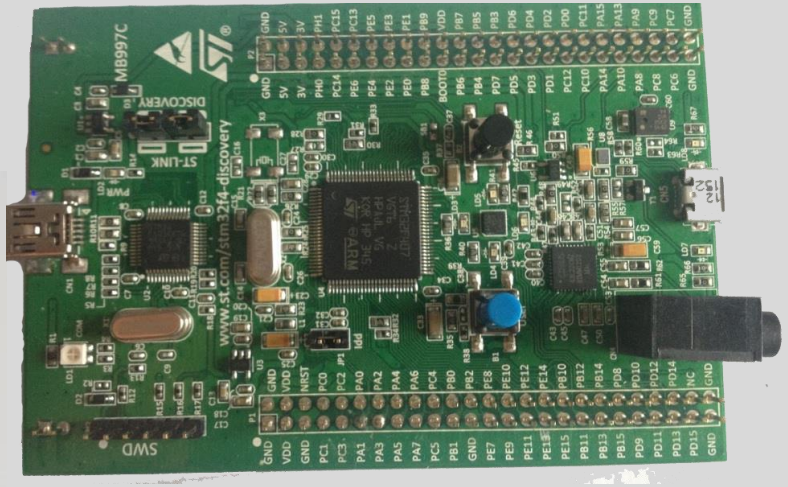
# Mikrovezérlő paraméterei

- Órajel: 1 – 100 MHz közötti
- Feszültségtartomány általában 3,3 V, vagy 5 V
- Áramfelvételük mA-es nagyságrendű.
- Lábszám alapján 8 – 100 db.
- A RAM és a Flash memória: általában kB nagyságrendű
- A belső adatméret alapján 8, 16 és 32 bites mikrovezérlők terjedtek el.

# Mikrovezérlő gyártók

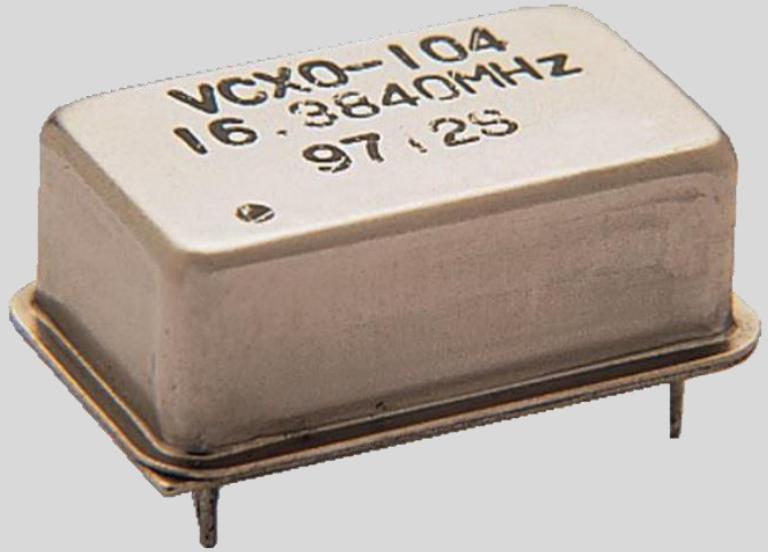
- **Pár nevesebb gyártó:**
  - **Atmel**
  - **Microchip Technology**
  - **Intel**
  - **Texas Instruments**
  - **STMicroelectronics**
  - **Xilinx, stb.**




# Oszcillátor

- Olyan eszköz, amely elektromos energia hatására, stabil és periodikus elektromágneses rezgést kelt
- Az előállított jel alakja szerint lehet
  - Szinuszos jel
  - Négyszögjel
- Lehet külső és/vagy belső oszcillátor
- Típusai
  - LC oszcillátor → 100 kHz feletti frekvenciákra
  - Meisner oszcillátor → visszacsatoló áramkör transzformátor, az egyik tekercs párhuzamosan van kötve egy kondenzátorral, így egy rezgőkört alkotnak.
  - RC oszcillátor: Kis frekvenciás tartományban

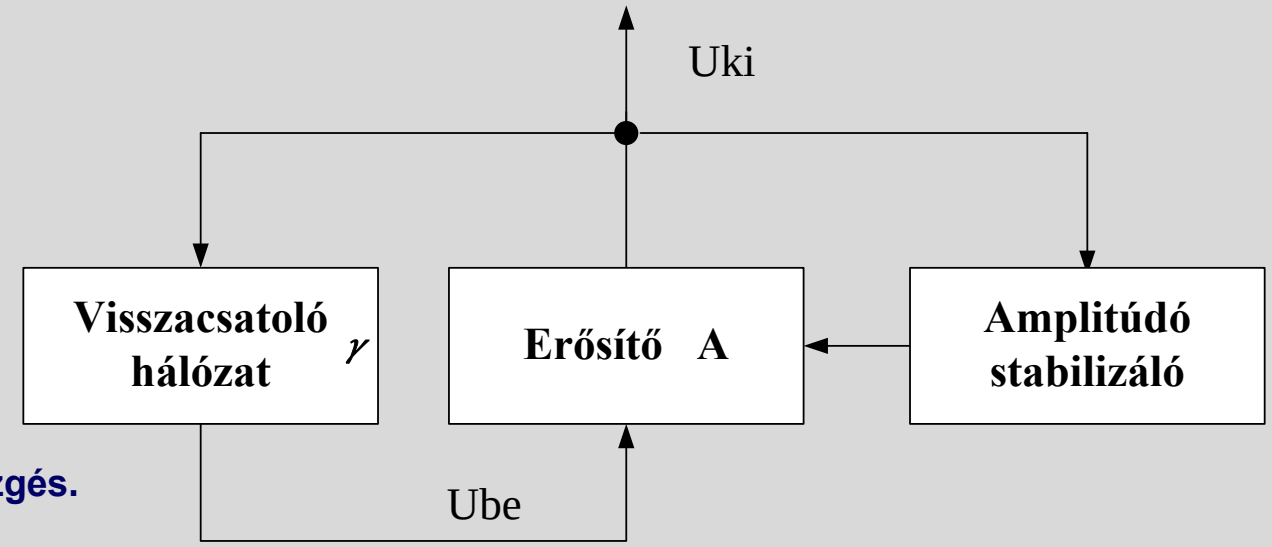


# Oszcillátor

## Működése:

$$A_{vcs} = \frac{A}{1 - \gamma A}$$

- Egy adott frekvencián pozitív visszacsatolás → eredő erősítés növekszik
- Fázisfeltétel: A visszacsatoló hálózat és az erősítő fázisforgatása 0° kell, hogy legyen $A \cdot \gamma > 0$
- Amplitúdófeltétel: $A \cdot \gamma \geq 1$
  - Ezt biztosítja a stabilizáló áramkör, így adott frekvencián a rezgés önfenntartó lesz
  - A gyakorlatban nem szokott teljesülni, hogy a szorzat értéke pontosan 1
  - Ha a szorzat kisebb, mint 1, akkor csillapodik a rezgés.



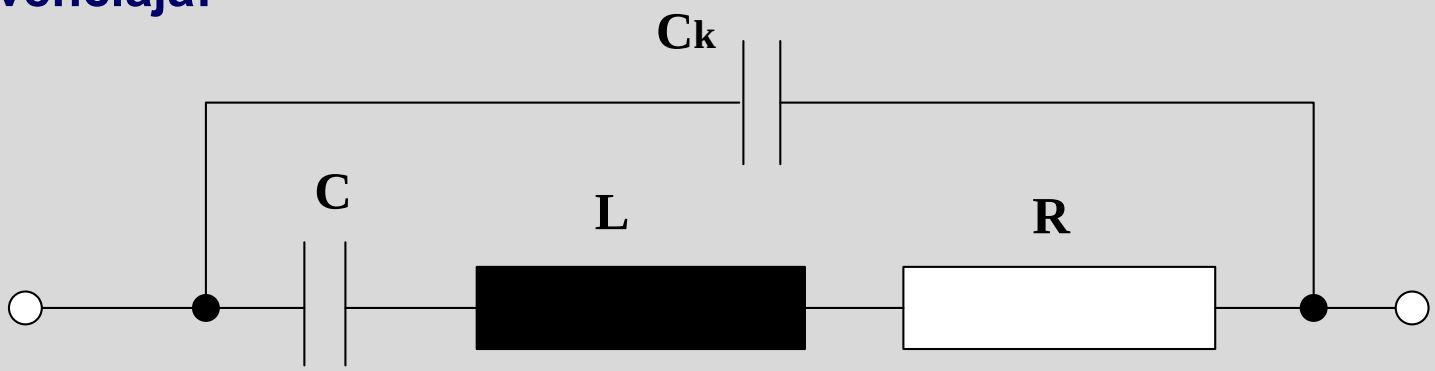
# Kristály-oszcillátor (XO)

- Rezgőkör helyett rezgőkristály alkalmazása → piezoelektromos tulajdonság
- Elektrostrikciót használjuk ki, a kristályra rákapcsolt feszültség hatására rezgésbe jön
- Kvarckristályból adott méretű lapka készítés
- A lapka két oldalára ezüst réteget visznek fel, majd kivezetéseket forrasztanak
- Váltakozó feszültség hatására a kristály rezegni kezd.

# Kristály-oszcillátor

- A helyettesítő kapcsolása
  - $C_k$ az ezüstréteg és a kivezetések együttes kapacitása
  - R, L, C a kristály tulajdonságából adódik
- A rezgőkör soros rezonanciafrekvenciája:

$$f_{rs} = \frac{1}{2\pi\sqrt{LC}}$$




A párhuzamos rezonancia frekvencia:

$$f_{rp} = \frac{1}{2\pi\sqrt{LC}}\sqrt{1+\frac{C}{C_k}}$$

Az ezüstréteg és a kivezetések tehát befolyásolják a rezonanciafrekvenciát, ezért célszerű párhuzamosan kapcsolni egy nagy értékű kapacitást.

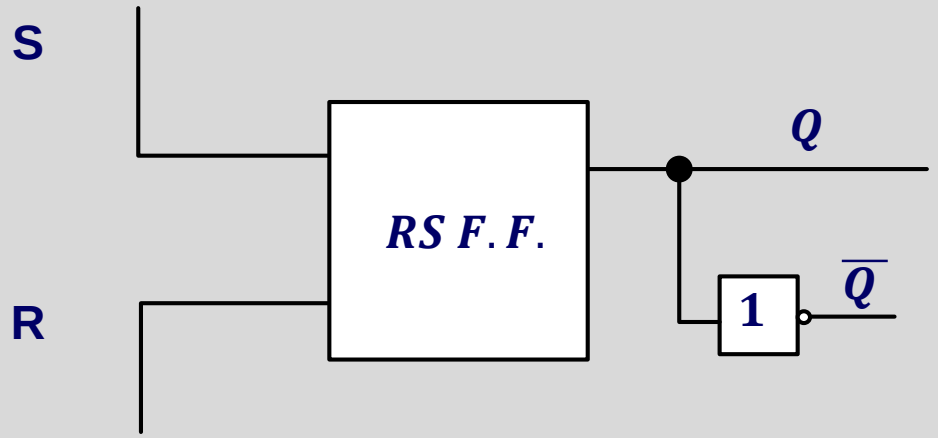
# Regiszterek

- A regiszter a mikroprocesszor flip-flopokból (elemi tárolóelemek) felépített nagy sebességgel írható és olvasható ideiglenes tárhelye. A hozzáférési ideje egy regiszternek 10 ns nagyságrendű.
- Flip-Flop: 1 bit információ tárolására alkalmas
- A következő regiszterek mindegyik processzorban jelen vannak:
  - Programszámláló (PC)
  - Veremtár mutató (SP)
  - Akkumulátor (AC)

# Regiszterek

- **Funkciójuk alapján besorolhatók:**
  - **Statikus regiszterek: Információ ideiglenes tárolására, megvalósítások RS (bistabil multivibrátor) és D Flip-Flopokból**
  - **Léptető, vagy SHIFT regiszterek: Információ tárolására és helyiértékenkénti léptetésre használhatók**
  - **Visszacsatolt regiszterek: A regiszter F.F.-jainak kimenetéről visszacsatolást eszközölünk annak bemenetére**

| R | S | Funkció |
|---|---|---|
| 0 | 0 | Tárolás |
| 0 | 1 | Beírás |
| 1 | 0 | Törlés |
| 1 | 1 | Tiltott |

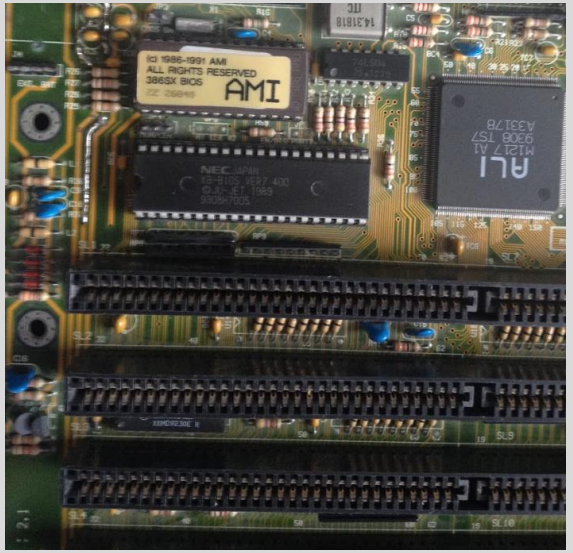


# Memóriák

- RAM: Random Access Memory, azaz közvetlen hozzáférésű, írható/olvasható memória.
  - SRAM: Sztatikus RAM → bipoláris tranzisztorokból, vagy MOSFET-ekből állnak, a tartalma a tápfeszültség megszűnéséig megőrződik
  - DRAM: Dinamikus RAM → MOSFET-ekből áll, a tartalma a kondenzátorok kisüléséig megmarad
  - Az első dinamikus RAM-ot az Intel mutatta be 1970-ben
- Feladata: Ideiglenes adatok tárolása, mint pl. programutasítások, adatok.

# Memóriák

- **ROM: Csak olvasható memória, BIOS, firmware és egyéb értékek megőrzésre használatos**
  - **A ROM lassabban olvasható, mint a RAM. → Gyorsítás: ROM tartalmának áthelyezése RAM területre (Shadow memory)**
  - **PROM: Csak egyszer írható, speciális eszközzel**
  - **EPROM: A tartalma UV fénnyel törölhető, majd újraírható, pl.: régi alaplapok BIOS-a**
  - **EEPROM (Flash memória): Elektromosan törölhető a tartalma, majd újraírható**

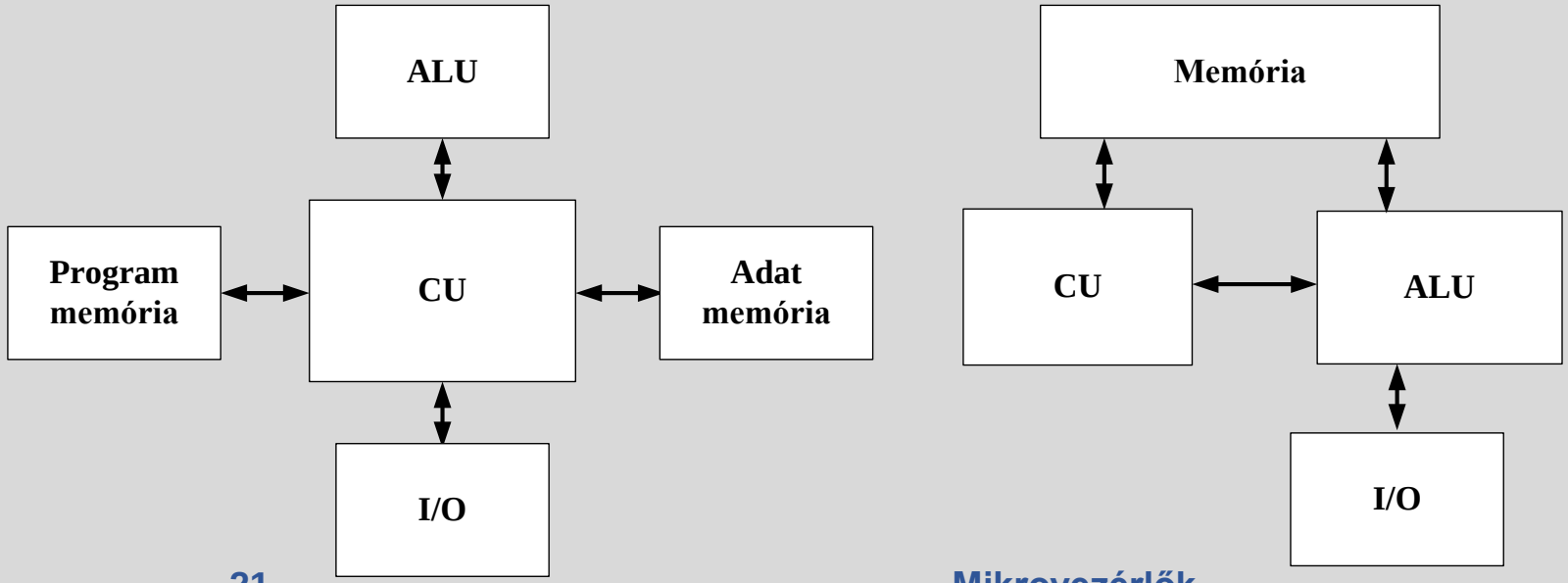


# Architektúrák

- **Neumann architektúra:**
  - **1945: Neumann János által kidolgozott modell**
  - **Számítógép a következő elemekből épül fel: ALU, CU, regiszterek, operatív tároló, háttértár**
  - **Egyetlen egy tároló elem adatok és utasítások számára**
  - **Utasítás kiolvasása és adattal végzett művelet egy időben nem hajtható végre → 1 db. adatbusz van**
  - **Ez a rendszer teljesítményét korlátozza → Harvard architektúra**

# Architektúrák

- **Harvard architektúra esetében az adat- és program memória külön helyezkedik el, ezért az utasításbeolvasás és az adathozzáférés nem egy útvonalon történik.**
  - **Párhuzamosítás → Teljesítménynövekedés**



# Utasításkészlet

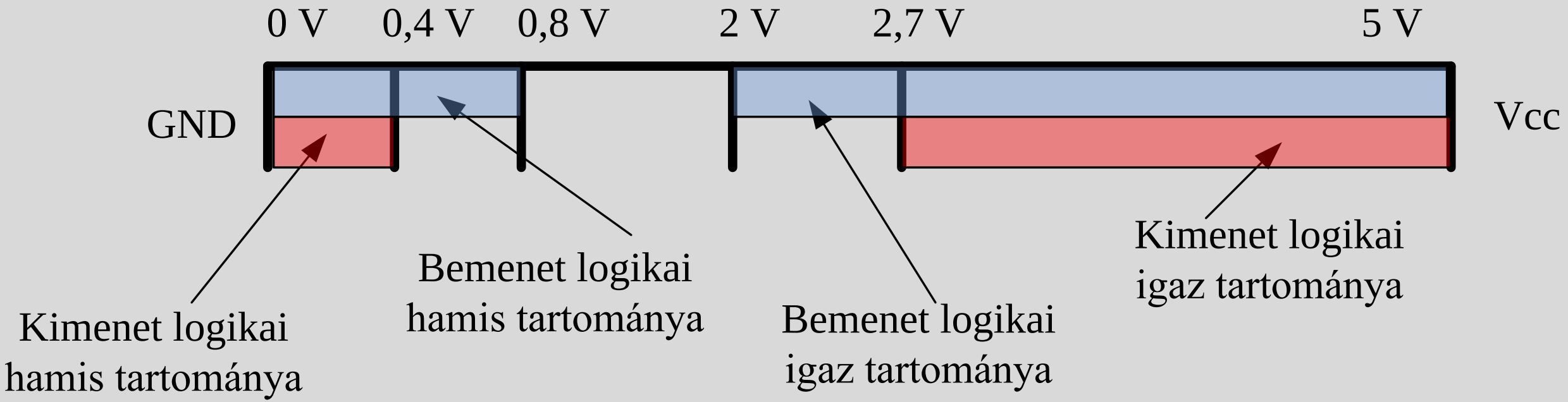
- Összetett utasításkészlet (CISC – Complex Instruction Set Computer):
  - Bonyolultabb utasítások → lassabb működés (hátrány)
  - Több elemi műveletből tevődik össze egy utasítás
  - Változó hosszúságú utasítások
  - Összetett műveletek
  - Ez volt a domináns architektúra
  - A nagy utasításkészlet nagyobb belső programtárat kíván
  - Kevés a regiszterek száma
  - Híres CISC processzorok: Intel 286, 386, 486, stb.

# Utasításkészlet

- Csökkentett utasításkészlet (RISC – Reduced Instruction Set Computer):
  - Load/Store architektúra: memóriaelérés csak ezzel a két művelet segítségével
  - Egyszerűsített címzési mód
  - Utasítások hossza egyforma
  - Az utasítások ciklusideje is egyforma
    - Pipe-Line: Több utasítás párhuzamosan is végrehajtható
  - Meglehetősen sok regisztert használ
  - Kisebb teljesítményű processzorok létrehozása
  - Ilyen processzor pl. az IBM PowerPC

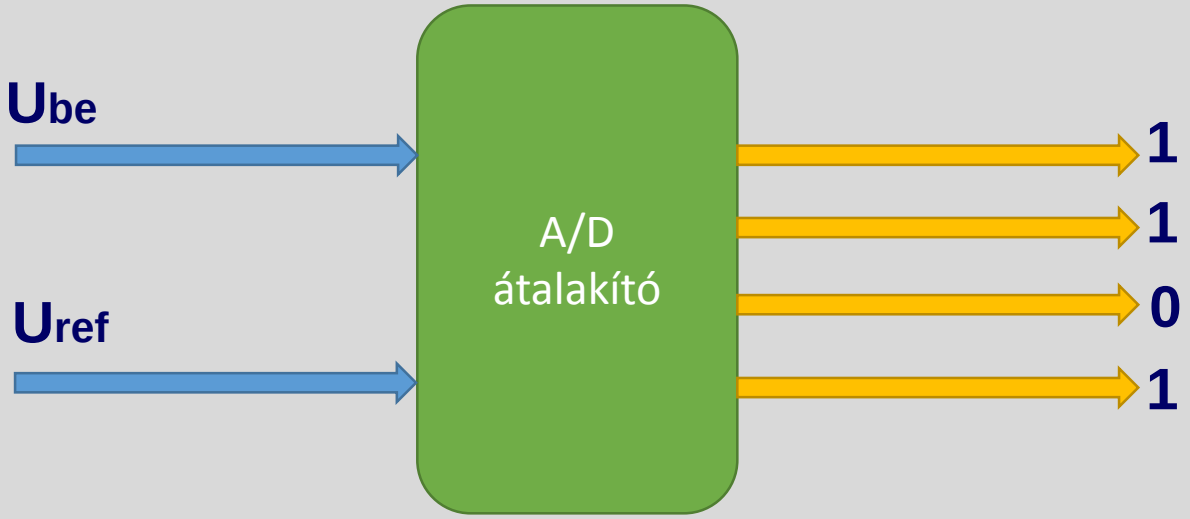
# Logikai tartományok definiálása

- **TTL → Tranzisztor-tranzisztor logika esetén az alábbi tartományok figyelhetők meg:**



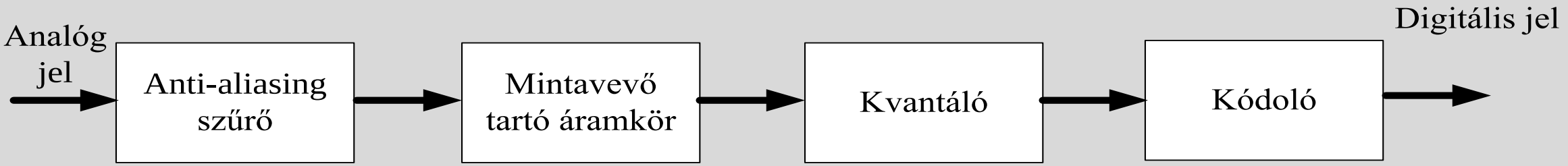
# A/D átalakítás

- A környezet jeleit mérjük. A fizikai paraméterek értékei többnyire folytonosak, ezt át kell alakítani diszkrét jelekké, hogy a mikrovezérlő számára feldolgozható legyen.
- Analóg jelek például: nyomás, hőmérséklet, erő…
- Megmutatja, hogy az adott órajelnél a bemeneti feszültség a referencia érték hányad része, ez alapján generál egy multi-bit számsort.



# A/D átalakítás

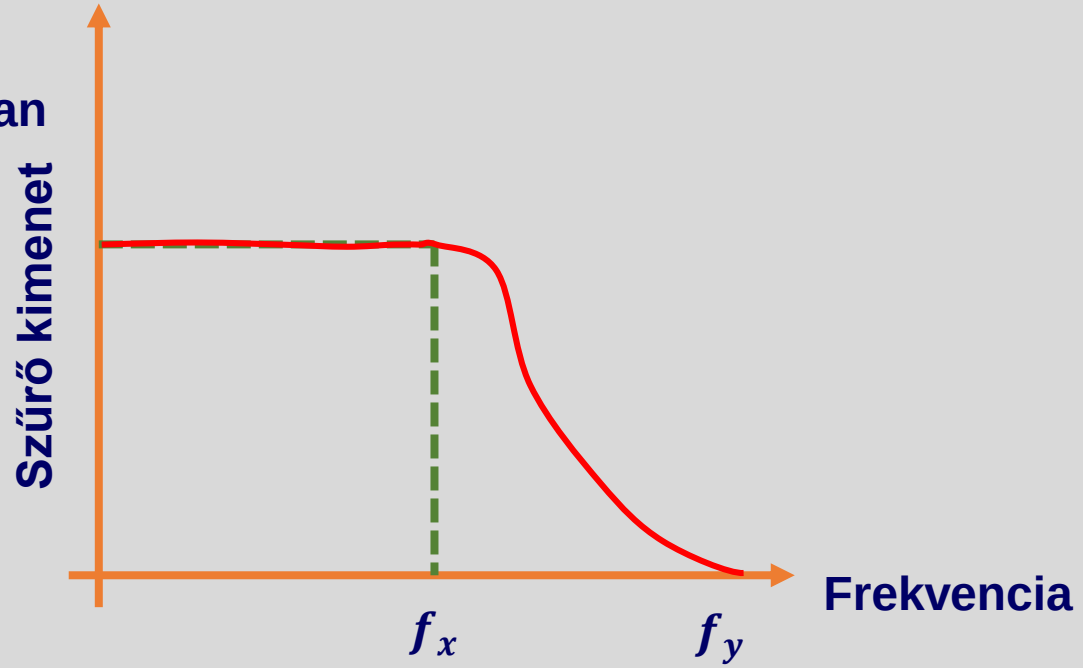
- **Az átalakítás blokkvázlata:**




- **Anti-aliasing szűrő: jelre szuperponálódott zajok kiszűrése →aluláteresztő szűrő**
- **Mintavevő tartó: Folytonos jelből mintát vesznek, tartókondenzátorral a következő mintavételig megőrzik.**
- **Kvantáló: Intervallumokra bontása a folytonos értéktartománynak, (lineáris, nem lineáris)**
- **Kódoló: A kvantálási intervallumokhoz azonosító hozzárendelése → pl. bináris kód**

# A/D átalakítás

- **Az anti aliasing szűrő: Lényegében egy meghatározott frekvencia érték feletti jeleket nem engedi át.**
  - **Analóg szűrő**
  - **Ideális szűrő: Az $f_x$ frekvencia felett nem enged át**
  - **A gyakorlatban van egy átmeneti sáv $[f_x, f_y]$, ahol ugyan gyengítve (fokozatos elnyomással), de még átenged bizonyos frekvenciákat**

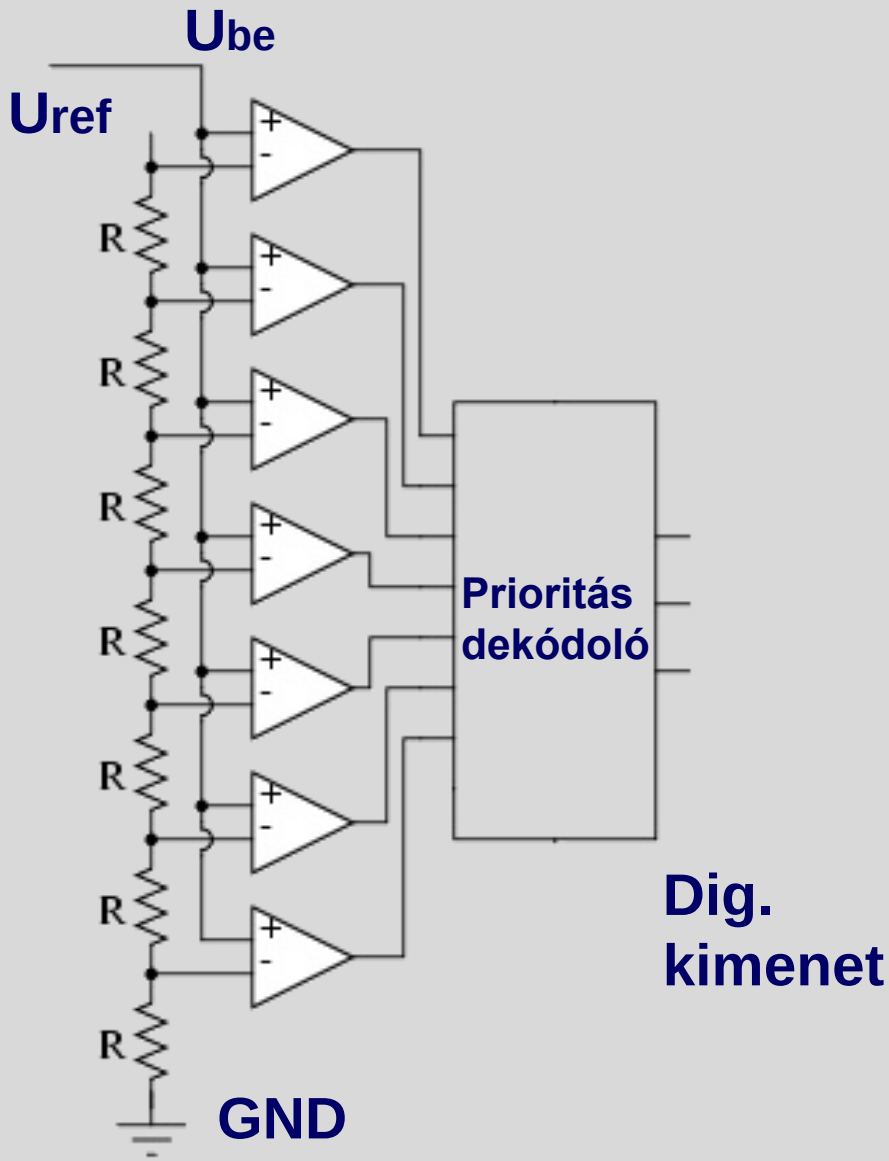


# A/D átalakítás

- **Mintavevő tartó:**
- **Konverzió közben a jel nem változhat.**
- **Mintavételi (Shannon) törvény: $f_{minta} \geq 2f_{max}$, a mintavételi frekvenciának legalább kétszer akkorának kell lennie, mint a maximális frekvencia.**

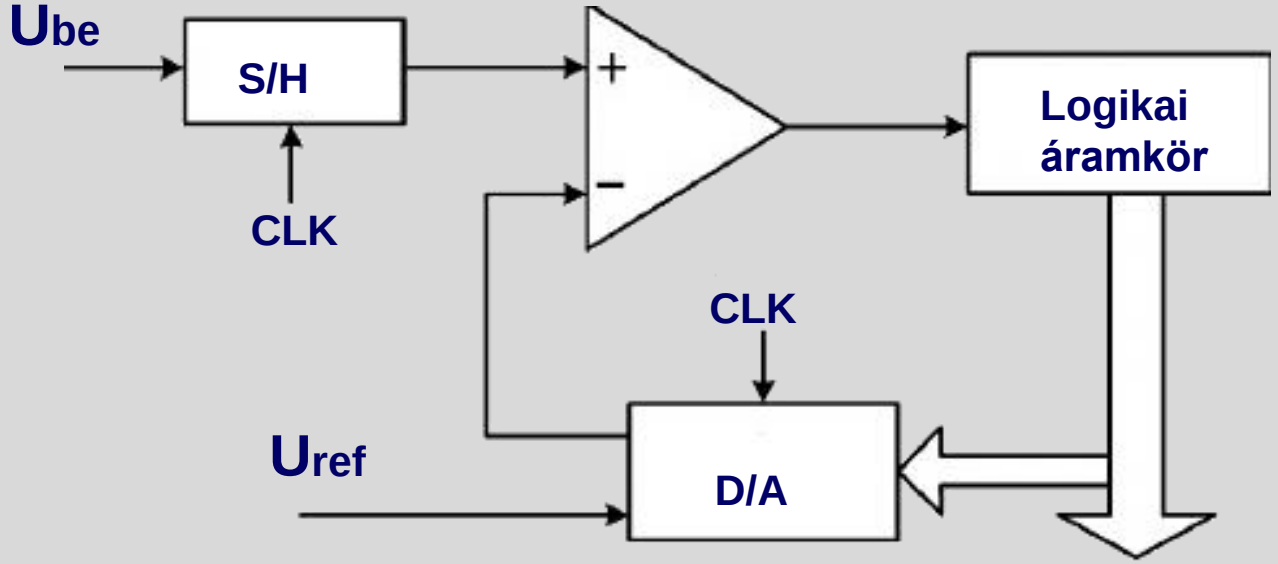
# A/D átalakítók fajtái

- **Flash, vagy Párhuzamos A/D**
  - **Nagy sebesség, drága, nagy megbízhatóság**
  - **Feszültségosztóval a referencia feszültség felosztása.**
    **Komparátorok figyelik, ha a ref. fesz. nagyobb, mint a bemeneti fesz.,**
    **akkor 0-át ad, ha fordítva, akkor 1-et.**
    **Ahol az átmeneti rész van (első 1), azt alakítja át digitális jellé.**
  - $2^N - 1$ **komparátorra van szükség**



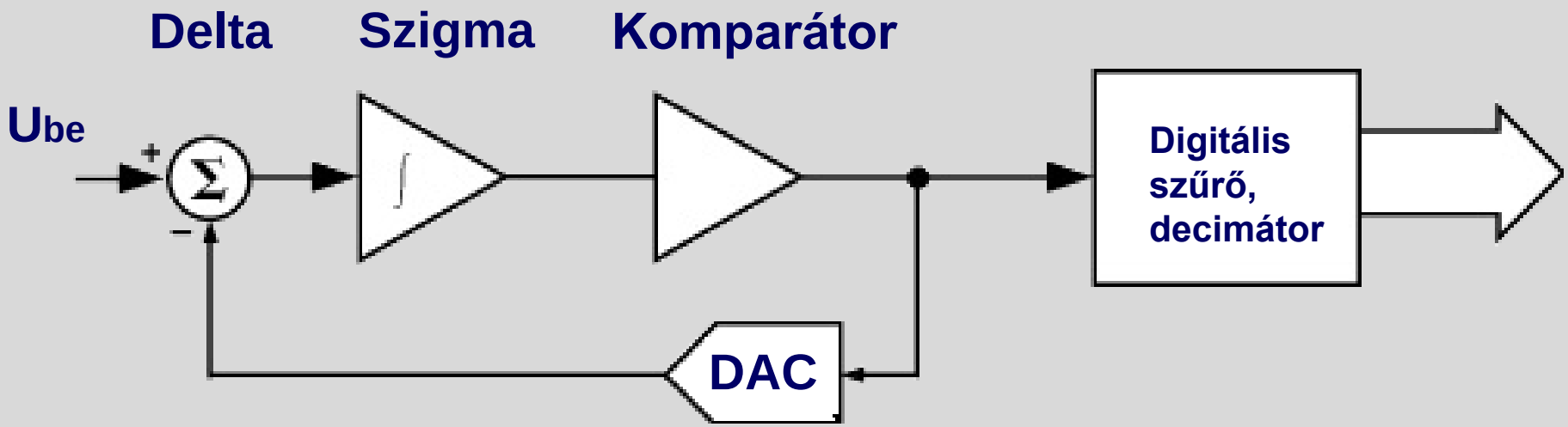
# A/D átalakítók fajtái

- **Kétoldali közelítéses (SAR)**
  - **Közepes sebesség és ár**
  - **Folyamatos közelítéssel megközelíti a bemeneti feszültség értékét**
  - **Jelátalakítás annyi lépés, amekkora az átalakító felbontása**
  - **Ha a bemeneti feszültség nagyobb, mint a referencia feszültség, akkor 1-et ad vissza, ha kisebb, akkor 0-t**



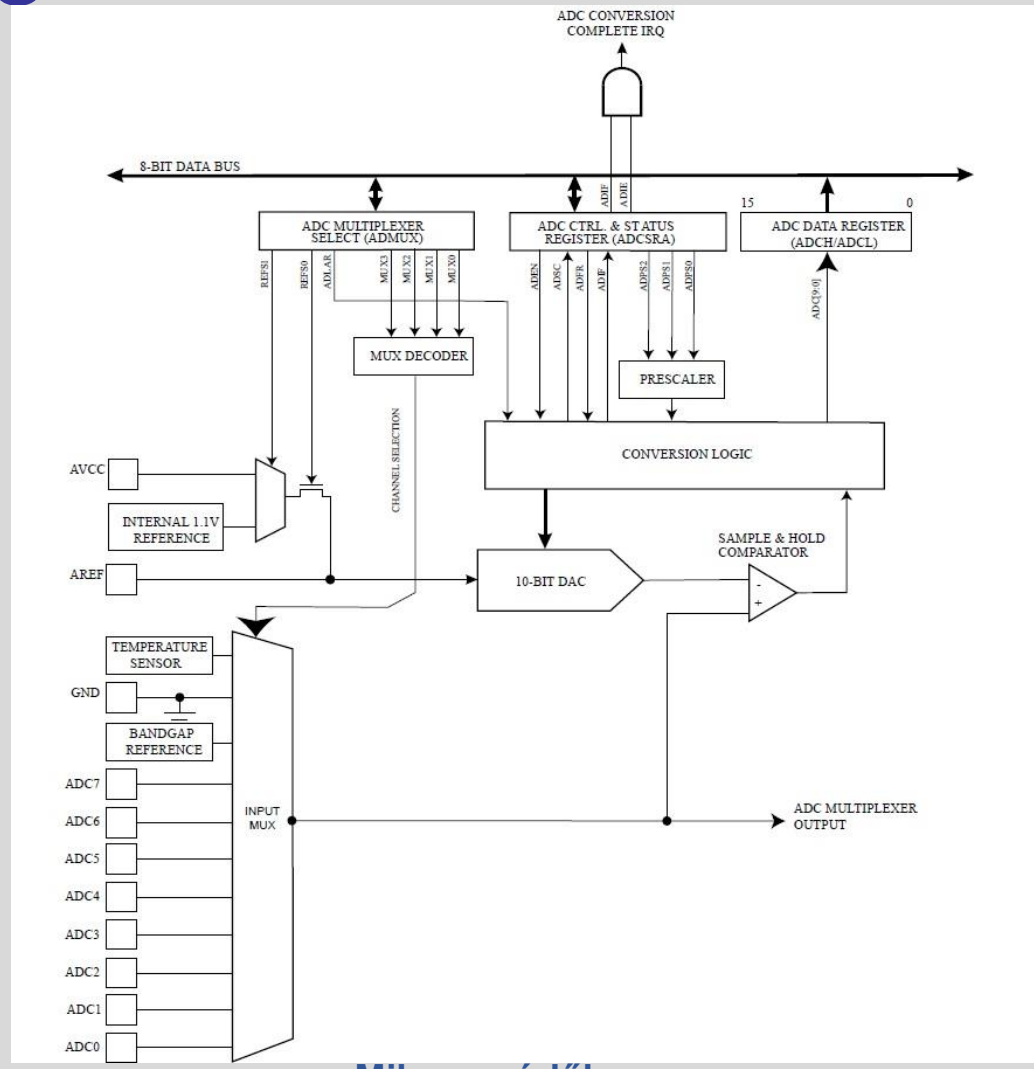
# A/D átalakítók fajtái

- **Szigma delta**
  - **Lassú, nagyon jó a felbontása**
  - **Minden lépésben integrálást hajt végre (Σ), és különbséget képez (Δ)**
  - **Működésének alapja a túlmintavételezés → bementeti jelet többszörös mintavételi frekvenciával mintavételezzük**



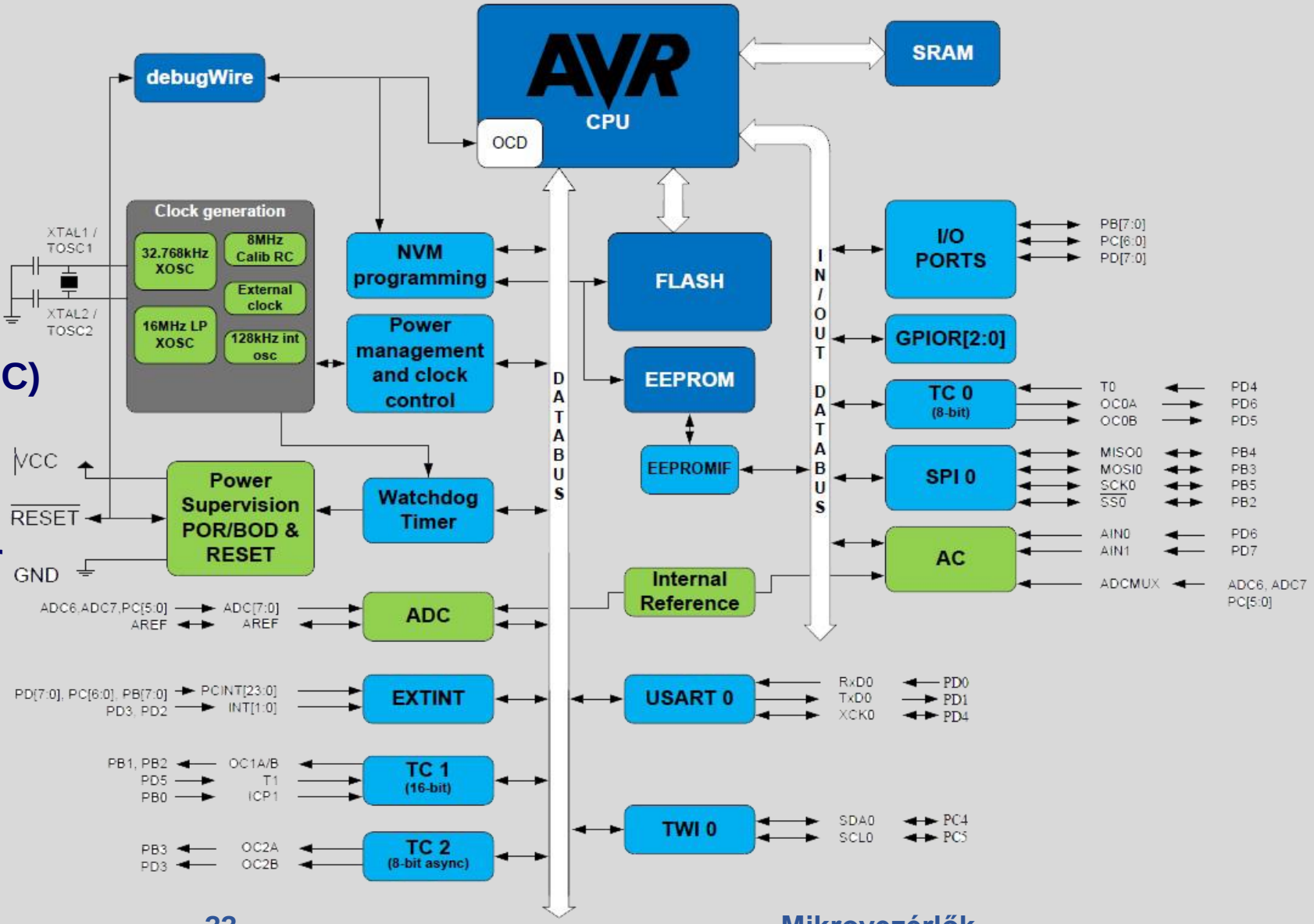
# ATmega328 A/D konvertere

- 10 bites felbontás → 1024 db diszkrét értékre történő szétbontás
- Kétoldali közelítéses (SAR)
- 13-260 µs konverziós idő



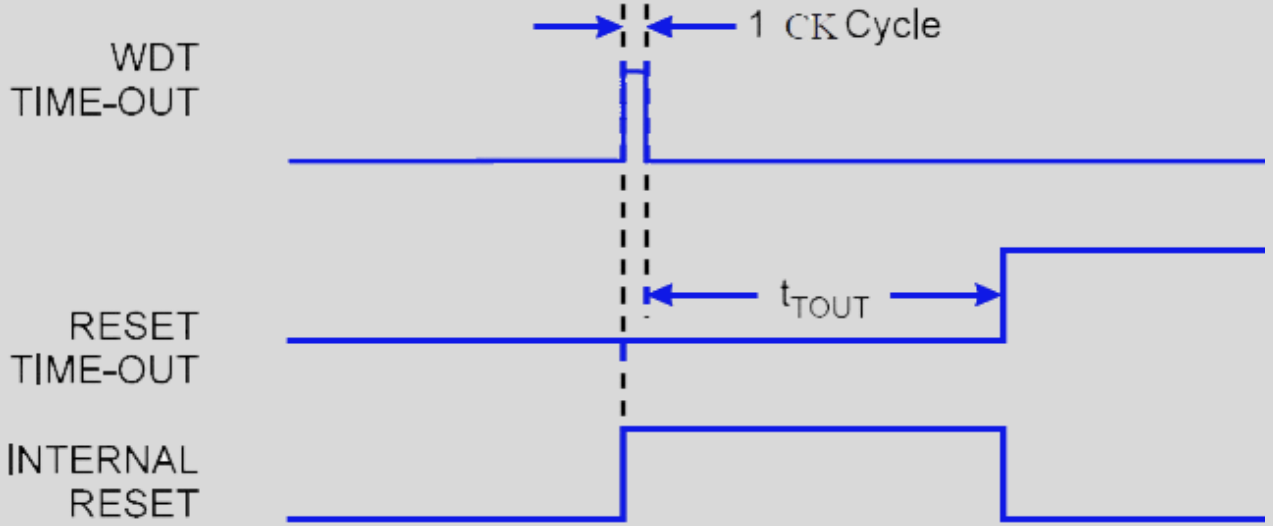
# ATmega328

- **ATmega328 mikrovezérlő:**
  - **Csökkentett utasításkészlet (RISC)**
    - **131 utasítás**
  - **Harvard architektúra**
  - **32 darab általános célú regiszter**
    - **8 bitesek (1 byte adat tárolása)**
  - **8 bites mikrovezérlő**
  - **1 KB EEPROM**



# WatchDog

- A mikrovezérlő lefagyása, tétlensége esetén generál egy órajelnyi RESET jelet, hogy újrainduljon a rendszer.
- A WatchDog Timer folyamatosan számol, ha elér egy bizonyos értéket, akkor → RESET
- WatchDog Timer reset utasítással a rendszernek újra kell indítania a számlálót.



# Programozási alapok (C++)

# Adatok típusa

- Karakter típus: char, char16_t, char32_t, wchar_t
  - Karakterek tárolására
- Egész típus: int
- Lebegőpontos: float, double, long double
  - Egyszeres/kétszeres pontosságú
- Logikai típus: bool
  - True/False
- Void típus: nem meghatározott

# Adatok típusa

- Típusminősítők:
  - const: változó értéke nem módosítható
  - volatile: a változó értékét egy másik futó folyamat is megváltoztathatja (megírt programtól függetlenül)
- Típusmódosítók: Típusleíró méretét és előjelességét, vagy előjel nélküliségét jelzik
  - short
  - long
  - long long
  - signed, unsigned

# Operátorok

- Szimbólumok összessége, amelyek az operandusok (kifejezés tagja) feldolgozására adnak előírást
- Precedenciával bírnak!
- A C++ tartalmazza a C nyelv összes operátorát is.

# Operátorok

- Aritmetikai: +, -, /, *, %
- Logikai és összehasonlító: ||, &&, <, >, <=, >=, ==, !=
- Léptető: ++, --
  - postfix, prefix alakok

```
int x, y, z;
z=1;
y=z++;
x=++z;
```

Mi lesz az x és az y értéke?

# Operátorok

- Bitművelet: ~, &, |, ^, <<, >>
- Értékadó: +=, -=, *=, /=, %=
- Feltételes: három operandusos operátor: ?:

```
int a=3, b=6;
int c;
c = a > b ? a : b; Mennyi lesz a c értéke?
```

# Speciális (vezérlő) karakterek

- Backslash után kell őket írni.
- Kocsi-vissza (carriage return): ’ \r’
- Újsor: ’ \n ’
- Lapdobás: ’ \f ’
- Tabulálás:
  - Vízszintes: ’ \t ’
  - Függőleges: ’ \v’

# Szintaktika/szemantika

- **Szintaktika: Formai hiba (nagyrészt elírások miatt)**
  - **double a==12;**
- **Szemantika: Tartalmi hiba, a programot a compiler lefordítja, de nem az elvártaknak megfelelően működik**

# Utasítások

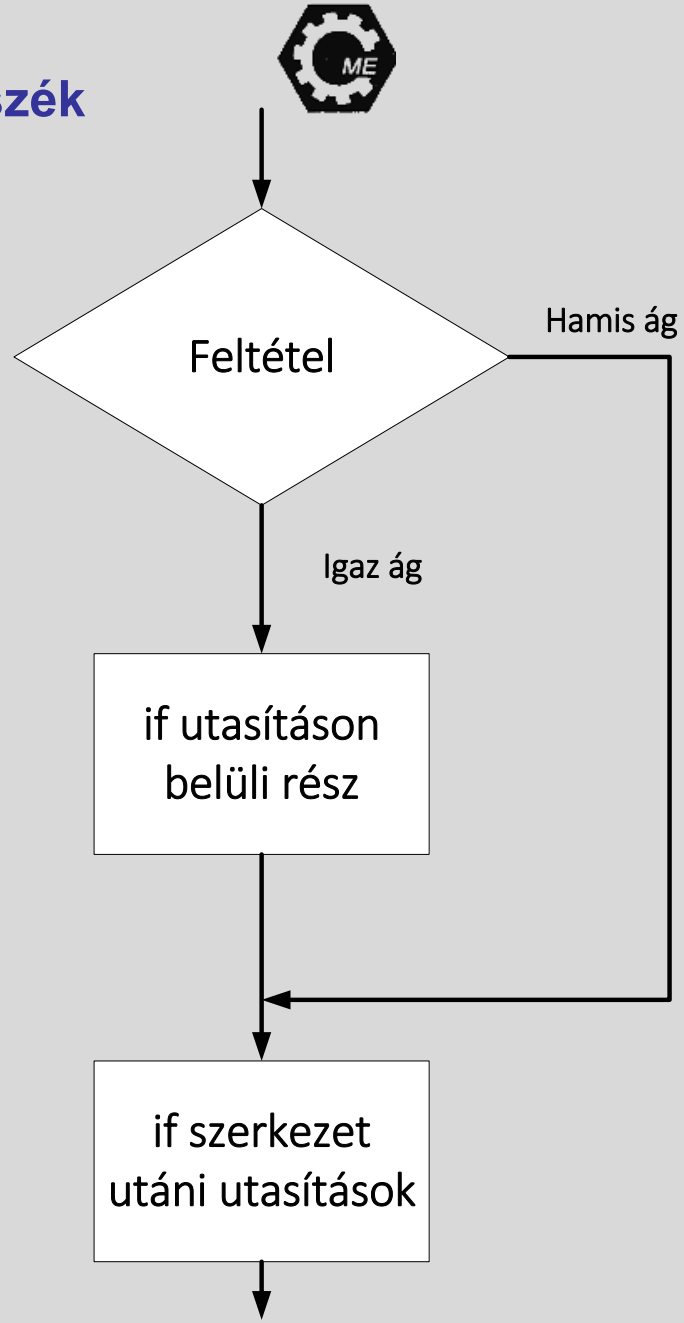
- Elvégzendő tevékenységekből épül fel a program, ezen tevékenységek egysége az utasítás.
  - Üres utasítás
  - Szelekciós utasítások: if, if-else, switch
  - Ciklusok
  - Vezérlésátadó utasítások
  - Címkézett utasítások
  - Összetett utasítások

# Utasítások

- Üres utasítás: Egy darab pontosvesszőből áll: ;
- Szintaktikai szabályok előírják az adott helyen az utasítás meglétét, azonban logikailag ott nem akarunk tevékenységet definiálni.

# if szerkezet

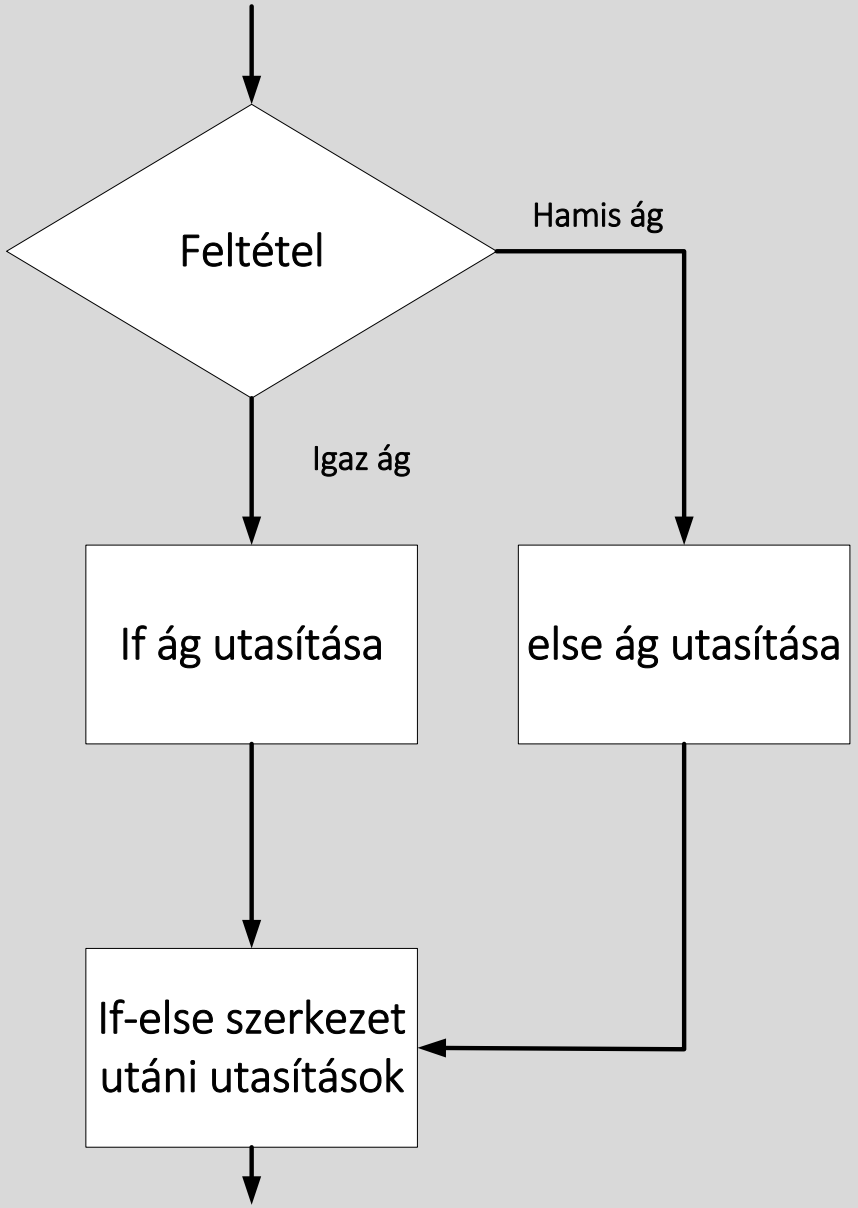
```
int valtozo=42;
if (valtozo > 100)
  {
        // ha igaz a feltétel,
        // akkor ez a rész végrehajtódik
  }
```



# if-else szerkezet

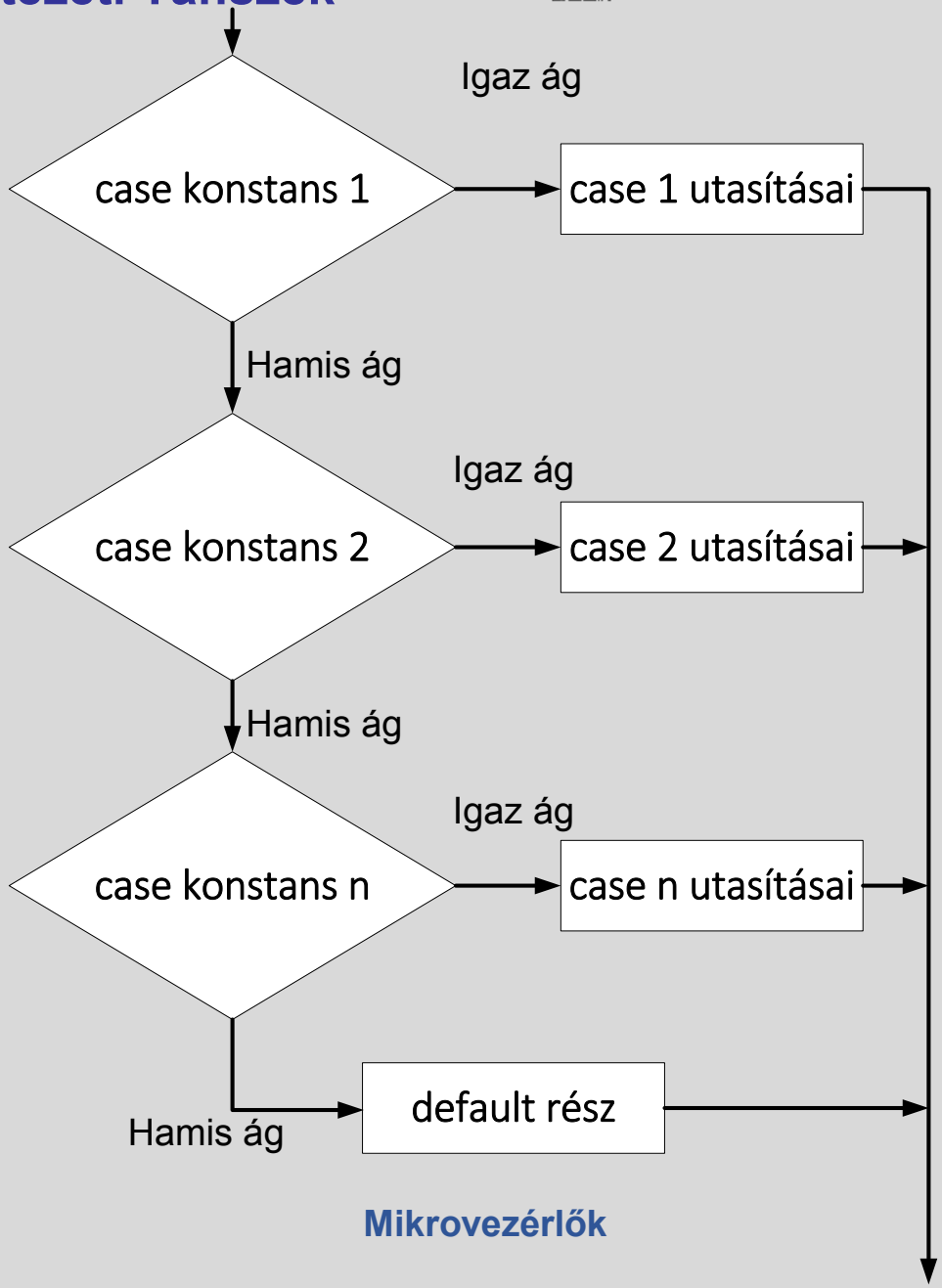
- **Fontos: az if utasítások egymásba is ágyazhatók!**
  **→ Többirányú elágaztatás lehetősége**

```
int valtozo=42;
if (valtozo > 100)
 {
        // ha igaz a feltétel,
        // akkor ez a rész végrehajtódik
 }
else
 {
        // ha hamis a feltétel,
        // akkor ez a rész végrehajtódik
 }
```



# switch-case szerkezet

```
switch (x)
  { case ’a’:
     utasítás(ok);
     break;
     case ’b’:
     utasítás(ok);
     break;
     default:
     utasítás(ok);
     break;}
```



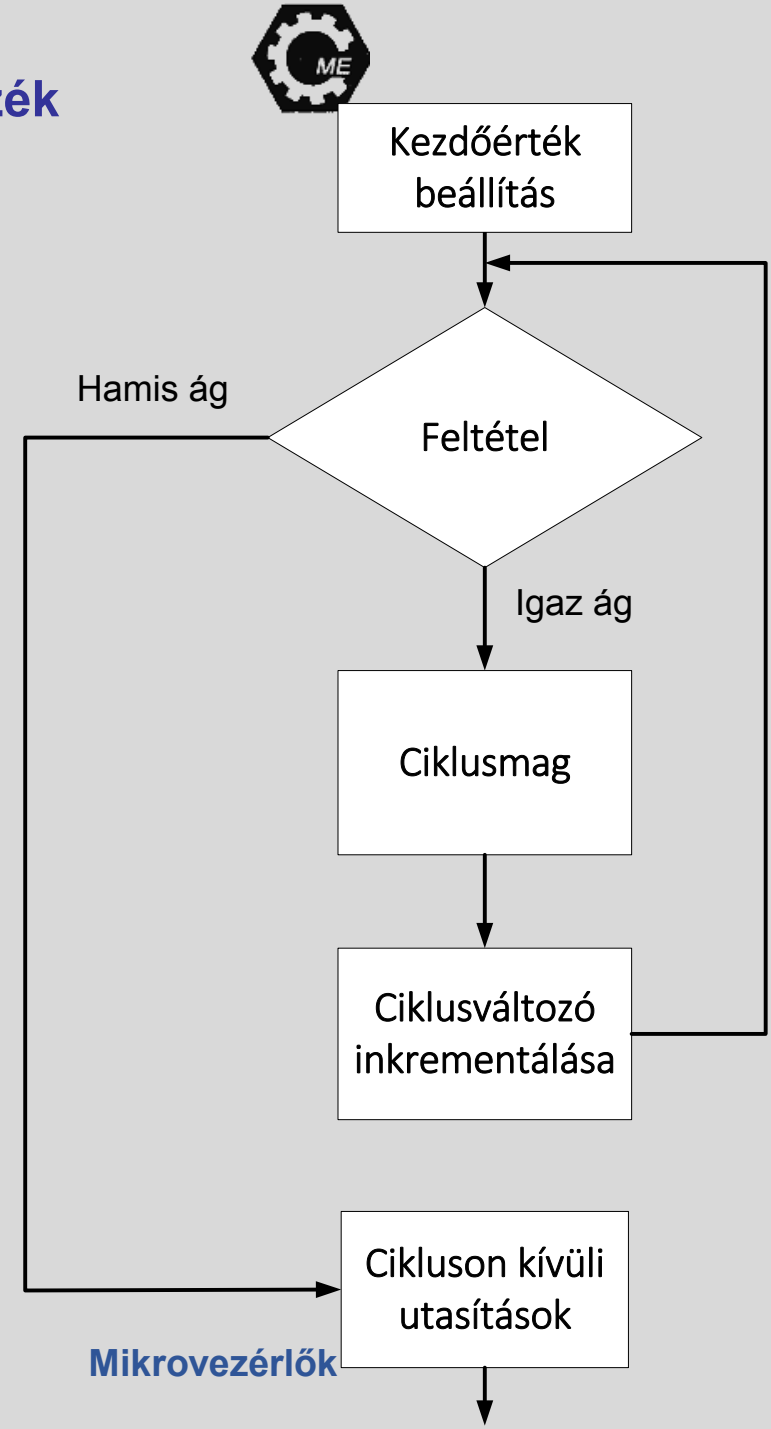
# Ciklusok

- Olyan programszerkezetek, amelyek egy bizonyos feltétel (feltételrendszer) megfelelése mellett bizonyos utasítások automatikus ismétlését biztosítják.
  - For ciklus
  - Elöltesztelő ciklus (while)
  - Hátultesztelő ciklus (do-while)

# for ciklus

- **A ciklusmagban megadott utasítást meghatározott mennyiségben akarjuk végrehajtani.**

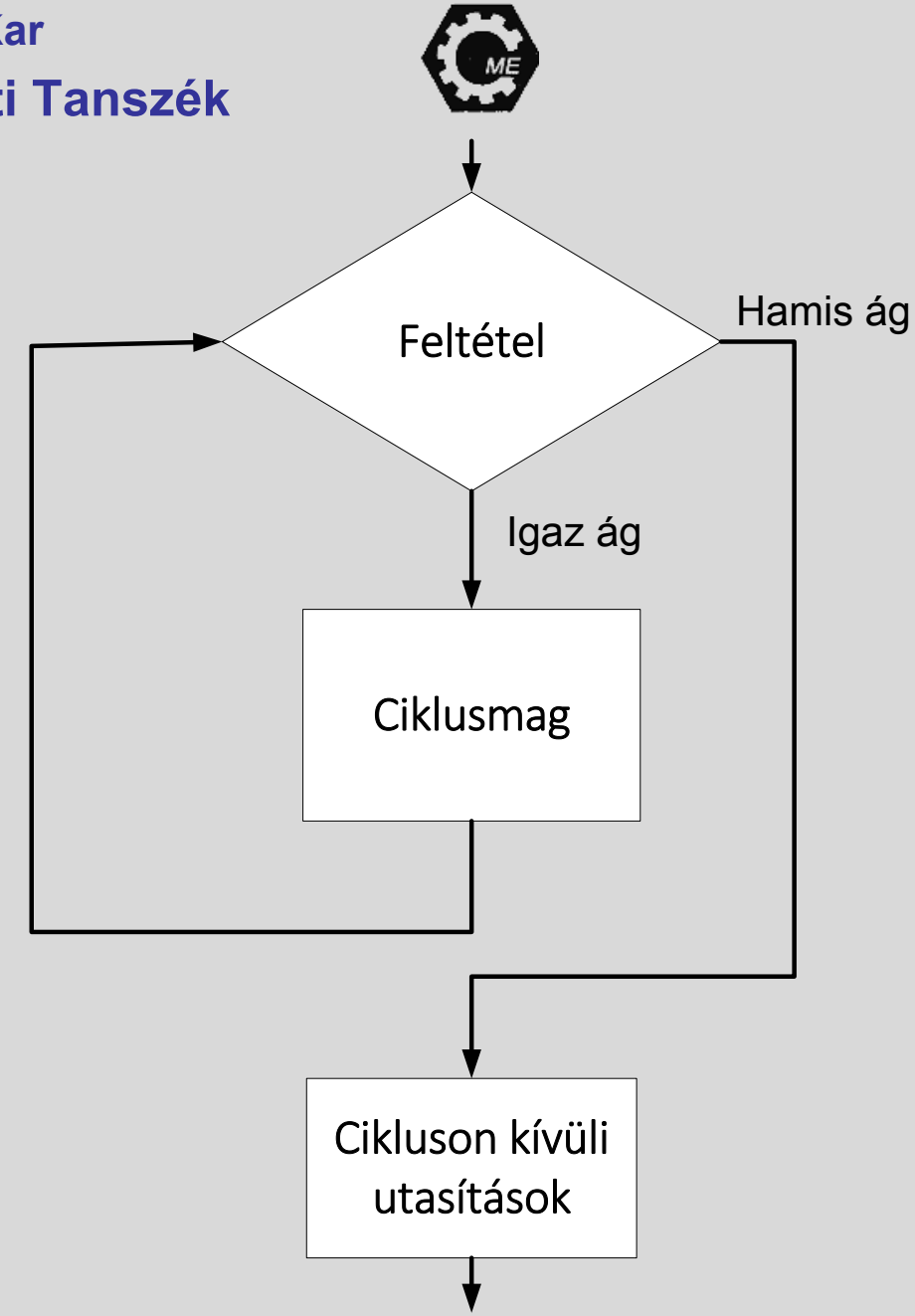
```
// kezdő érték; feltétel; léptetés
for (int i=0; i<100; i++)
{
   utasítás(ok);
}
```



# while ciklus

- A megadott feltétel, ha igaz, akkor végrehajtja a ciklusmagban megadott utasításokat.
- Elöltesztelő: Tehát először van feltétel vizsgálata

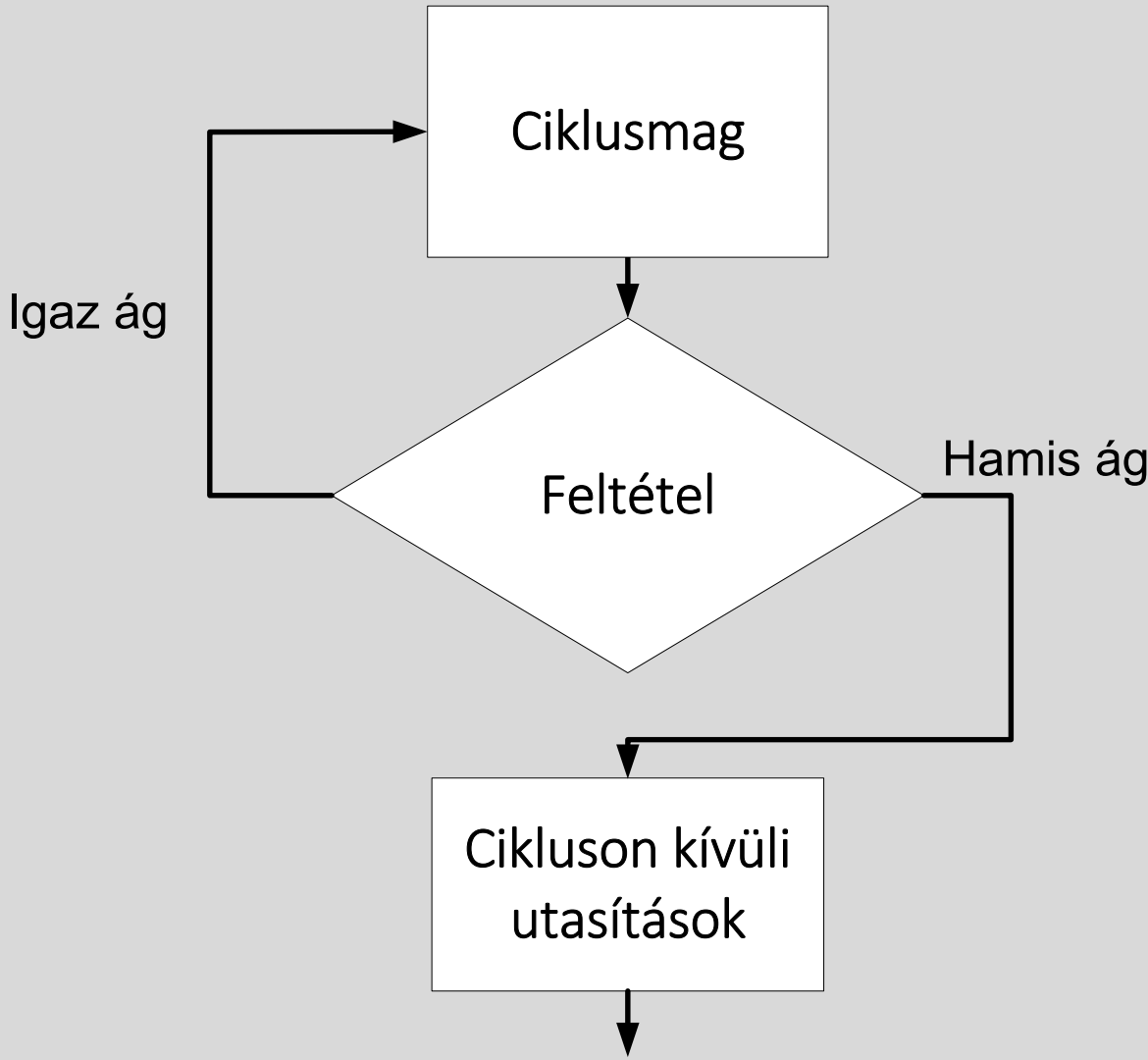
```
while(feltétel)
 {
   utasítás(ok);
 }
```



# do-while ciklus

- **A ciklusmagban található utasítások végrehajtása után van a feltétel vizsgálat → hátultesztelő ciklus**

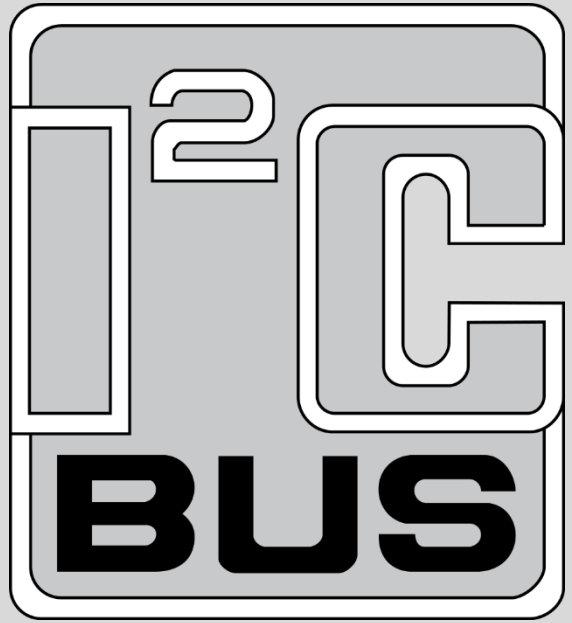
```
do
 {
   utasítás(ok);
 }
while (feltétel);
```



# Kommunikációs szabványok

# Mikrovezérlőknél alkalmazott kommunikációs szabványok

- $I^2C$: Inter-Integrated Circuit
- SPI: Serial Peripheral Interface
- UART: Universal Asynchronous Receiver/Transmitter

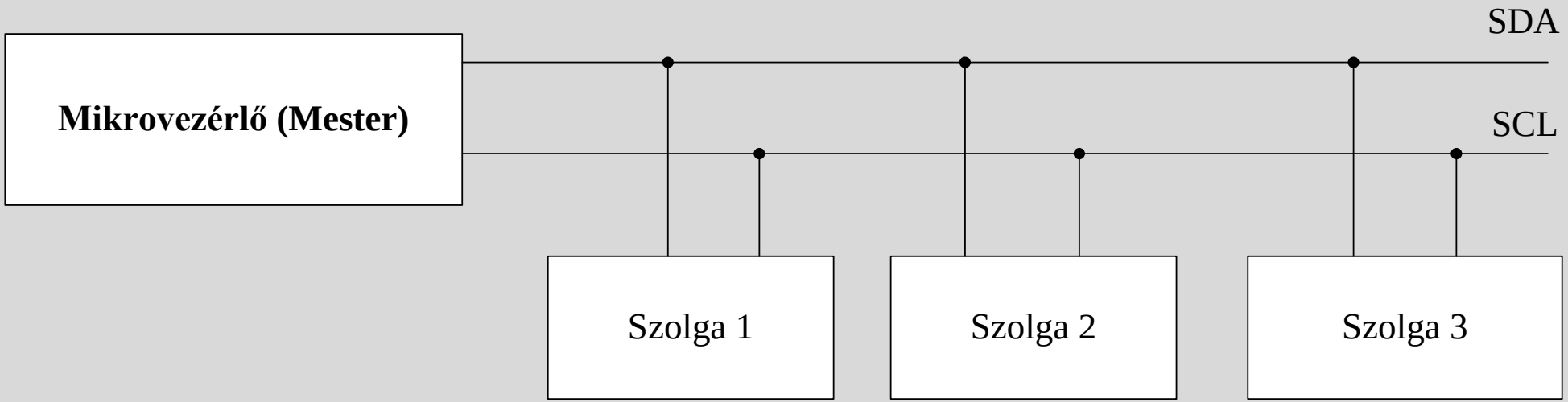


# Kommunikáció módok

- **Szimplex: Egyirányú kommunikáció, az információ csak egy irányban áramlik, erre jó példa  a GPS**
- **Half-duplex: Két irányú a kommunikáció, de egyidőben csak egy irányban lehetséges**
- **Full-duplex: Külön vezeték van a küldésre és fogadásra → SPI**

# I2C protokoll

- IC-k közötti kommunikációra találták ki.
- Két vezetéket használ, egyet az órajelnek (SCL), egyet a soros adatnak (SDA)
  - A két vezeték felhúzó ellenállásokkal logikai 1 értéken van (általában ≤ 4,7 kΩ)
  - 5 V és a 3,3 V-os feszültségszint az elterjedt, de lehet más feszültséggel is használni
- 1982-ben a Philips cég fejlesztette



# I2C

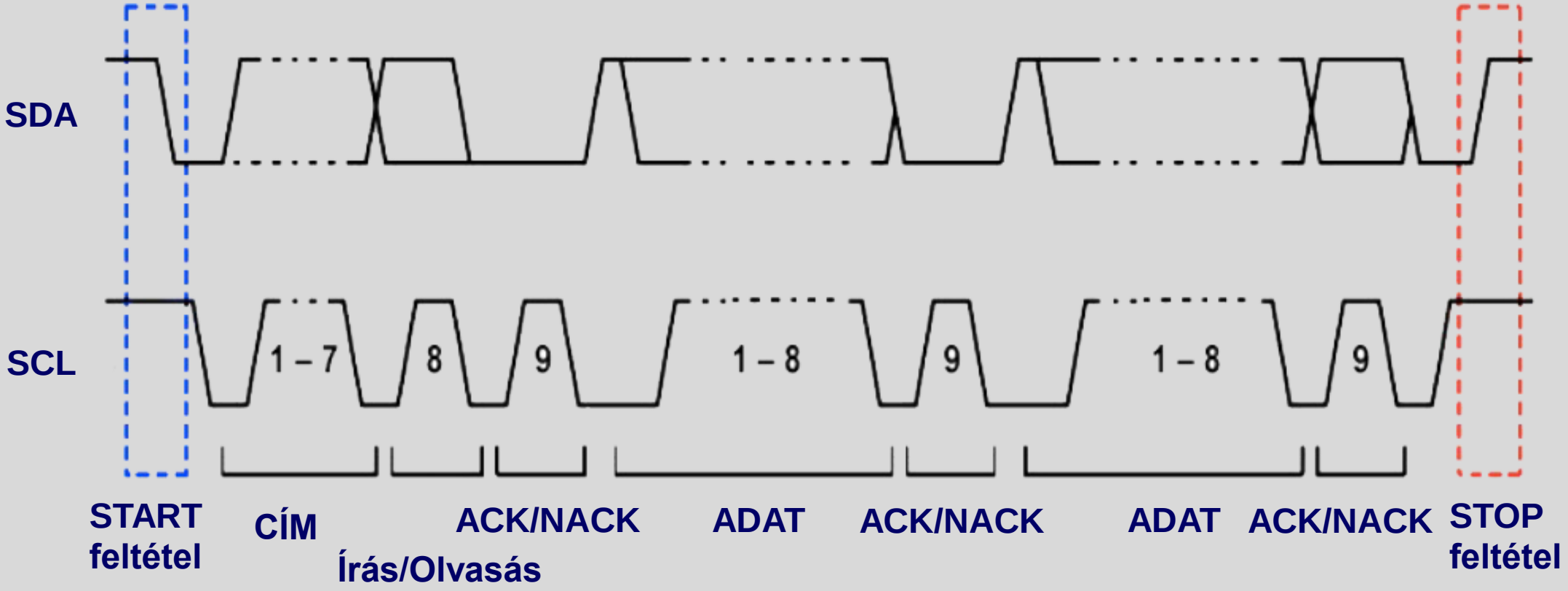
- Eredetileg 7 bites címzés és 100 kHz-es adatátviteli sebesség
- 1992-ben megjelent a 10 bites címzés és 400 kHz-es adatátviteli sebesség
- Mára már az 5 MHz adatátviteli sebesség sem lehetetlen
- Multi-master rendszert támogatja
- Szinkron adatátviteli rendszer
- Nyitott Drain → az eszközök a jelvezetéket le tudják húzni 0-ra
- 2-3 m távolságon belül használható nagy megbízhatósággal
- Slave lehet: A/D, D/A átalakító, vagy mikrovezérlő

# I2C

- A protokoll
  - A kommunikációt a master eszköz kezdeményezi a START feltétellel → SDA logikai 0 lesz, még az SCL előtt
  - Ha több master eszköz van, akkor azé a kommunikációs jog, amelyik az adatvezetéket hamarabb húzza le
  - Címzési rész, ez eredetileg 7 bites: MSB → legnagyobb helyértékű bájt
  - Ezután következik az írás, vagy olvasás bit, ha logikai 1, akkor olvasást, ha logikai 0, akkor írást kezdeményez a master.
  - NACK/ACK bit: A keret 9. bitje, ha a kiválasztott slave eszköz elérhető, akkor ezt ezzel a bittel visszajelzi az adatvezetéken (a 9. órajel előtt logikai 0-ra teszi az SDA-t).
  - Adatkeret: SDA-n mennek az adatok, mellette a master továbbra is generálja az órajelet. Ezután ACK/NACK.
  - STOP feltétel: Ha az adat elküldése megtörtént, akkor a master fogja generálni. SCL 0→1, utána SDA 0→1

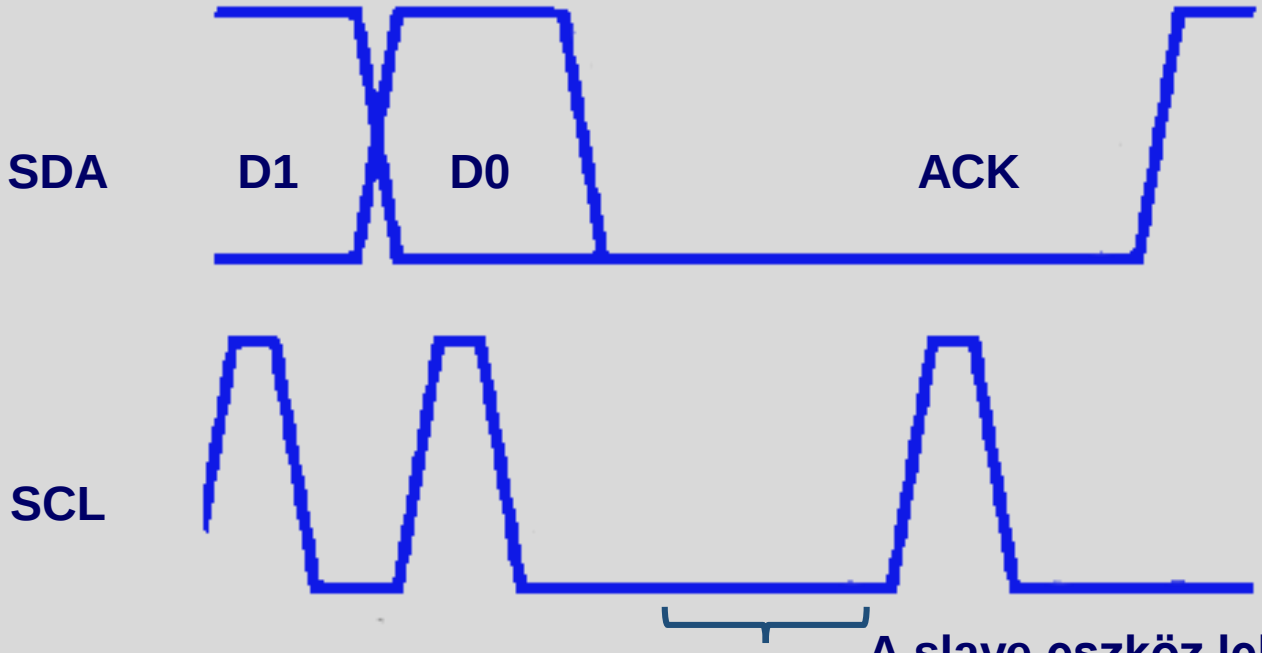
# I2C

- **A kommunikációs protokoll:**



# I2C

- **Órajel „nyújtás” (Clock stretching):**
  - **Amikor a slave eszköz nem áll készen újbóli adatok fogadására, akkor az órajel vezetéket lehúzza, ekkor a master eszköz megvárja, míg a slave elengedi a vezetéket.**
- **Az esetre jó példa lehet egy slave-ként bekötött A/D átalakító, amely még nem fejezte be a konverziót.**

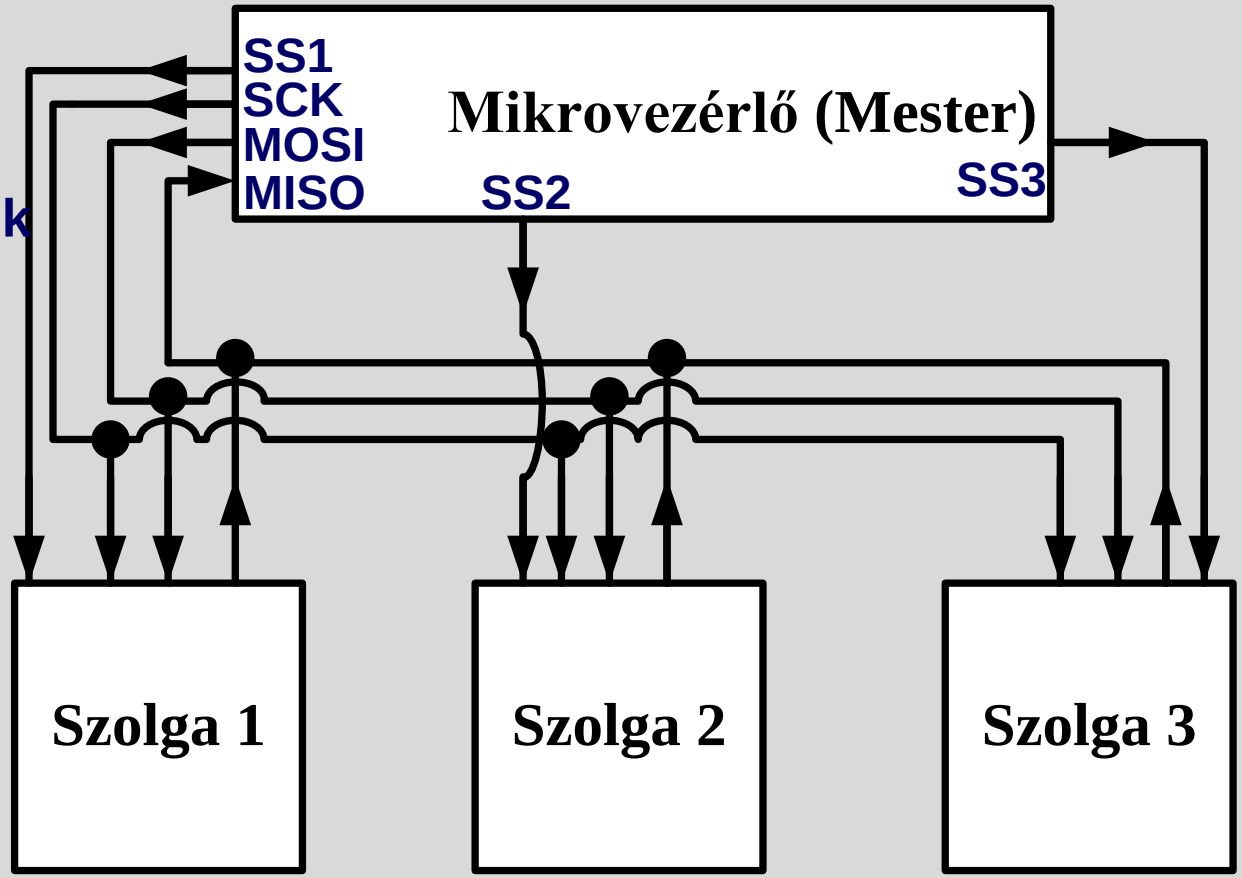


# SPI

- Kétirányú szinkron soros kommunikáció
- Master/slave kapcsolat itt is megvan
- Minimum négy vezeték szükséges, több slave esetén újabb vezetékek szükségesek
- Slave eszközök lehetnek pl. különféle szenzorok, shift regiszterek, stb.
- A fogadó egység egy egyszerű shift regiszter is lehet
- Egy master, több slave egység lehet

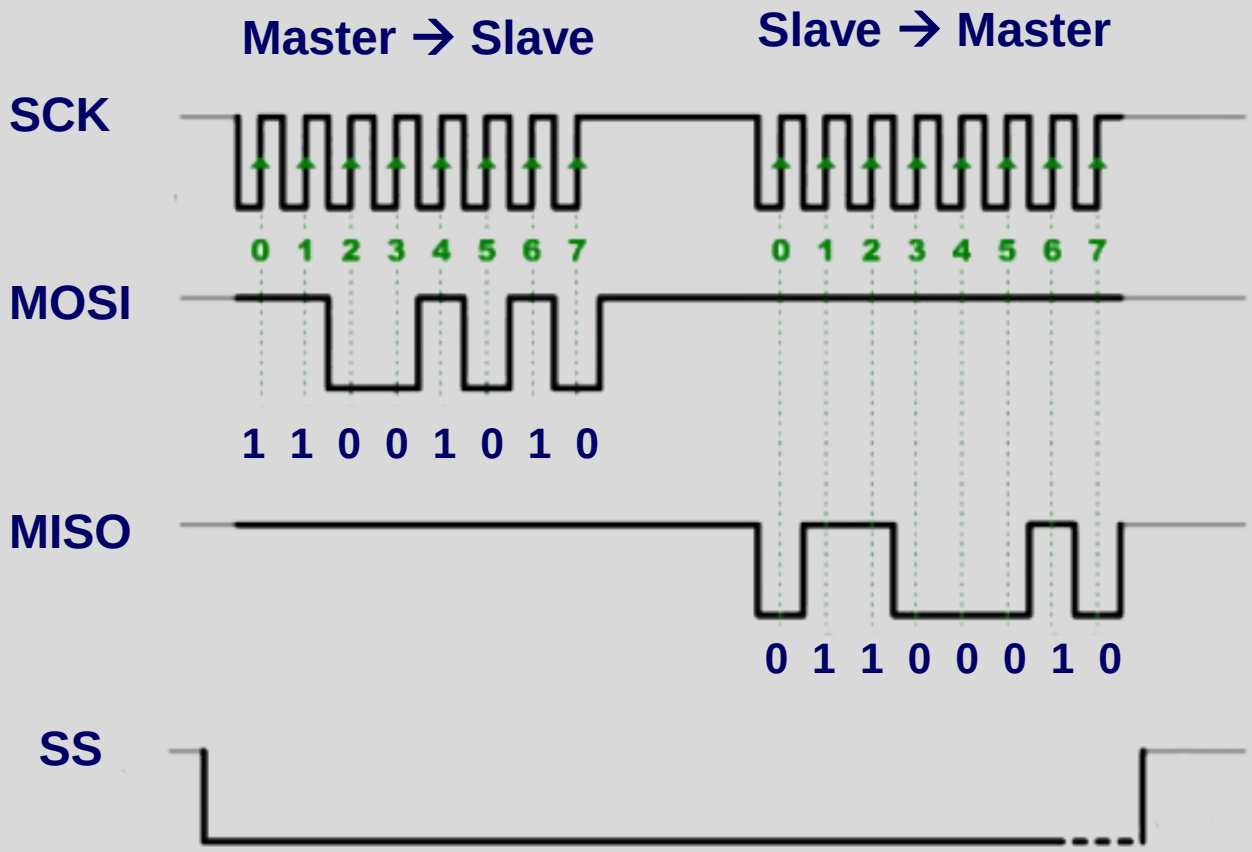
# SPI

- SCK: Órajel, amelyet a master generál
- MOSI: Master Out Slave In, adat küldés a slave-nek
- MISO: master In Slave out, adat küldés a master-nek
- SS: Slave Select: Slave választása adat küldésre/fogadásra → több slave esetén több SS láb szükséges!
- Lehet egy darab SS vezetéket használni pl. LED meghajtóknál



# SPI

- **A kommunikáció folyamata:**
  - **A bájtsorrend állítható**
  - **Gyors működés: Mikrovezérlő órajelét osztjuk le (/2, /128, stb.)**



# SPI

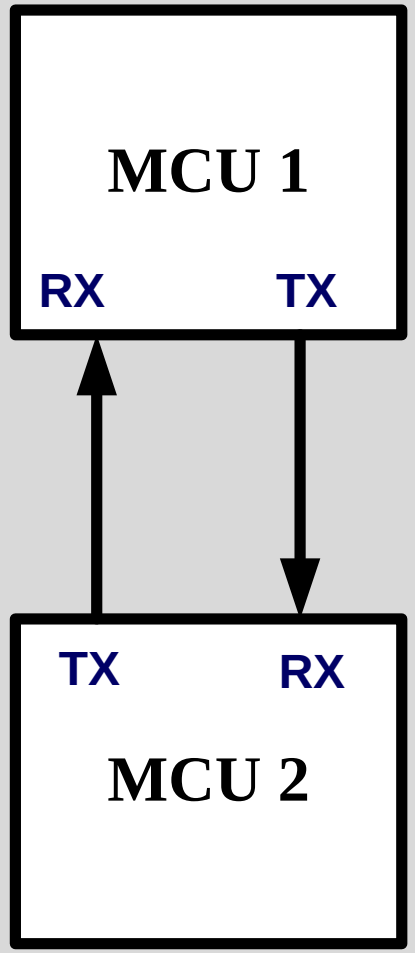
- Előnyök:
  - Gyorsabb az aszinkron soros átvitelnél
  - Több slave támogatása
  - Shift regiszter lehet a fogadó hardver
- Hátrányok
  - Több vezeték szükséges a kommunikációhoz
  - Slave-ek nem kommunikálnak egymással
  - Általánosságban több SS vezeték szükséges, ha több szolga van a rendszerben

# UART

- Aszinkron adatátvitel
- Két darab vezeték:
  - TX
  - RX
- A soros és a párhuzamos interfészek „közvetítője”.
- Az UART egyik végén van az adatbusz és vezérlő pin-ek, a másik végén pedig az RX és a TX kivezetés
- Lehet különálló IC is, de a mikrovezérlők többségébe ez már be van építve (ATmega328 1 db. UART modult tartalmaz)

# UART

- START bit: logikai 0 állapot
- ADAT bitek
- PARITÁS bit: adat bitek után, ha használják: az 1-ek száma páros, vagy páratlan legyen
- STOP bitek: logikai 1 állapot
- Általában LSB a bájtsorrend



# UART

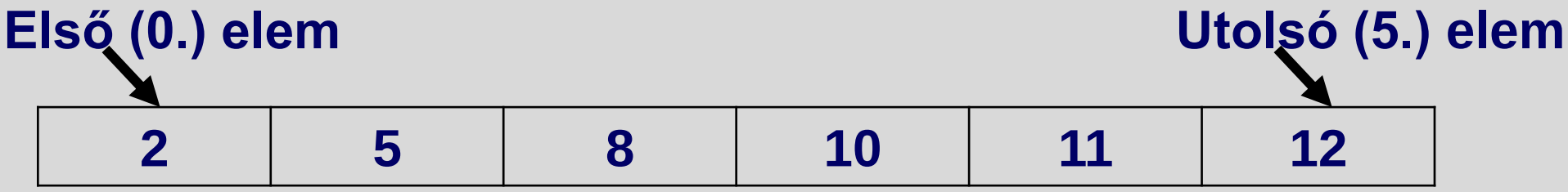
- A kommunikáció folyamata:


- A kommunikáció sebessége:
  - 9600 bit/sec
  - 19200 bit/sec
  - 38400 bit/sec
  - 115200 bit/sec
  - Stb.

# Tömbök, struktúrák kezelése

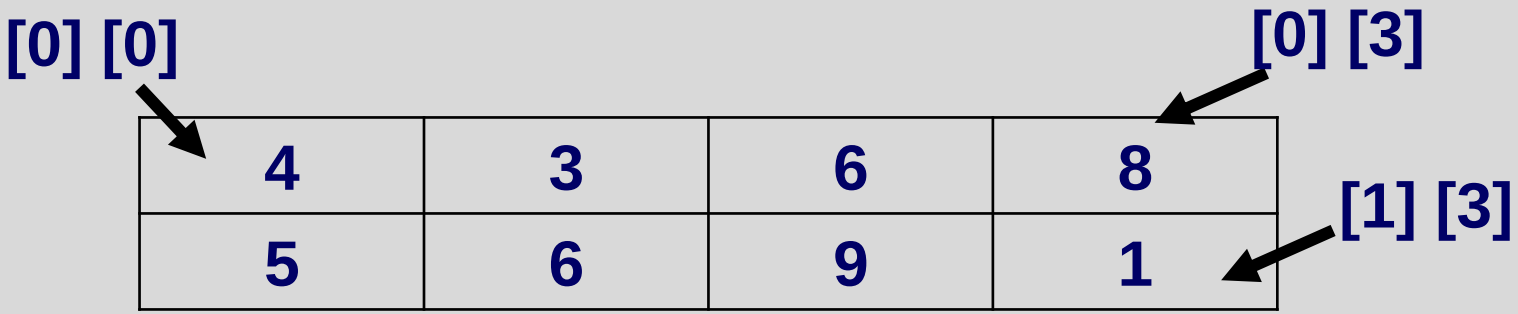
# Tömbök

- Egy változónév alatt azonos értékek tárolása
- Indexálás 0-tól indul, tömbméret-1 lesz az utolsó elem
- Az index csak egész típus legyen
- Deklarálásra példa: int pins []={2, 5, 8, 10, 11, 12};
  int pins [6]={2, 5, 8, 10, 11, 12};



# Többdimenziós tömbök

- Deklarálása: típus tömbnév [x] [y];
  - Mint egy táblázat, amelynek x sora és y oszlopa van.
- Pl. int 2dtomb [2] [4] = {{4, 3, 6, 8},
  {5, 6, 9, 1}};
- int 2dtomb [2] [4] = {4, 3, 6, 8, 5, 6, 9, 1};



# Többdimenziós tömbök

- **Példaprogram:**

```
void loop () {
int valami[3][4] = { {2,3,4,5}, {6,7,8,9}, {10,11,12,13}};
int i, j;
for ( i = 0; i < 3; i++ ) {
      for ( j = 0; j < 4; j++ ) {
         Serial.print("A tömb elemének sora és oszlopa:");
         Serial.print(i, j);
         Serial.print("Az értéke:");
         Serial.print(valami[i][j]);
         Serial.println();
         delay(100);}
 }
}
```

# Struktúrák

- Változók együttese, amelyek összetartoznak, kapcsolatban vannak.
- Egy egységként kezelhetjük őket
- A struktúrán belüli változók típusa lehet különböző.
- Erre jó példa lehet
  - Egy könyvtári rendszer, amiben számon tartják a könyveket
  - Autókereskedésnél az autók rendszerezése
  - Stb.

# Struktúrák

- **Példaprogram struktúra definiálására:**

```
typedef struct {
        char cime [30];
        int kiadas_eve;
        char szerzo [20];
        int isbn;
} Konyv;
```

# Struktúrák

- **Példaprogram struktúra definiálására:**

```
Konyv k1;
void setup() {
strcpy(k1.cime, "Programozas");
k1.kiadas_eve= 1999;
strcpy(k1.szerzo, "Anonymous");
k1.isbn=1000000;
Serial.begin(9600);
}
void loop () {
Serial.print("A konyv cime: ");
Serial.print(k1.cime);  // stb…
}
```

# Megszakításkezelés

# Események jellege

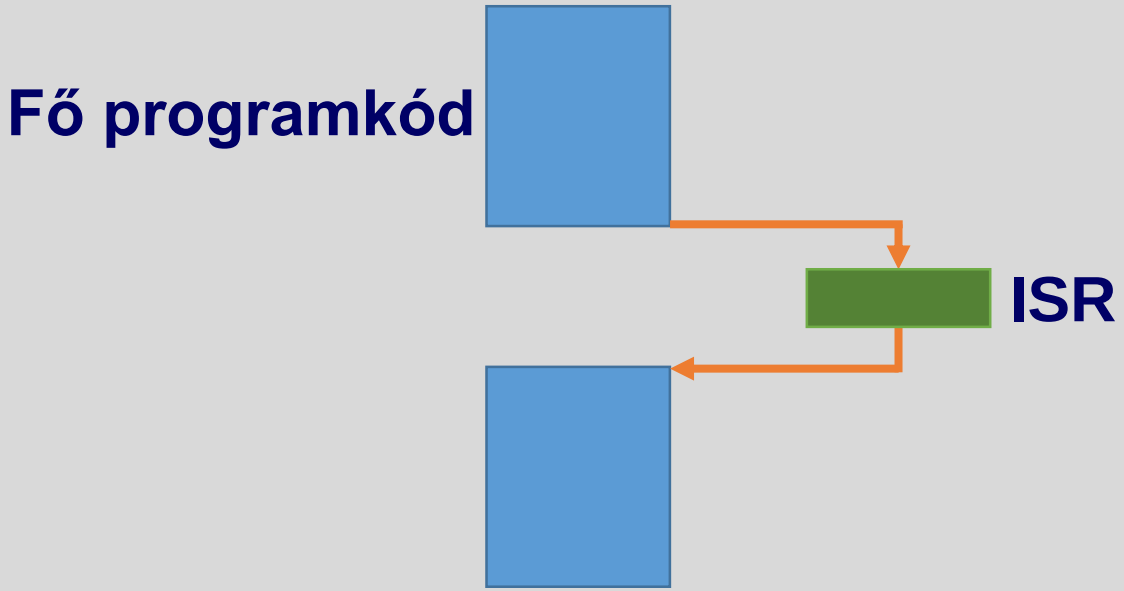
- Szinkron jellegű (belső) esemény: a program futása közben meghatározható a keletkezés helye és időpontja
- Aszinkron jellegű esemény: várható események, de a bekövetkezés idejét nem lehet tudni pl. billentyűzet esete
- Váratlan aszinkron jellegű esemény: Az esemény bekövetkezése nem várt pl. hardverhiba.

# Megszakítások

- Szoftveres és hardveres megszakítások:
  - Szoftveres megszakítás: az utasítások milyensége miatt
    - pl. a 0-val való osztás esete (fault), trap, stb.
  - Hardveres megszakítás: hálózaton, perifériákon keresztül érkező megszakítási kérelmek
    - pl. egy gomb megnyomása váltja ki, egér mozgatása stb.

# Megszakítások

- A főprogram futásának megállítása valamilyen esemény hatására, majd egy magasabb prioritású programrész futtatása.
- A felfüggesztett program állapota mentésre kerül, hogy a későbbiekben folytatható legyen.



# Megszakítások

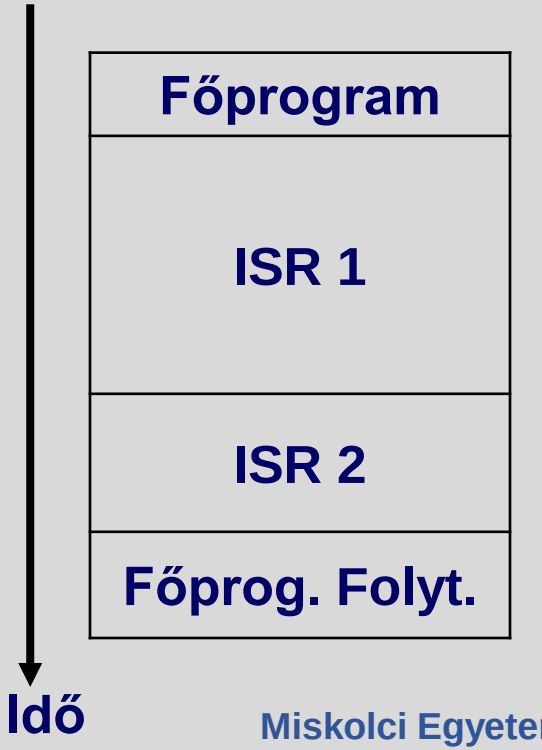
- Nem a főprogram váltja ki, hanem valamilyen külső esemény
- Megszakítási kérelem érkezésekor az éppen futó utasítás nem szakad meg
- Megszakítás-kezelő rutin feladatai (ISR - Interrupt Service Routine):
  - Az aktuális program állapotának elmentése átmeneti tárolóba
  - További megszakítások tiltása (engedélyezése)
  - Megszakítás-kezelő programrész futtatása
  - Megszakított program folytatása

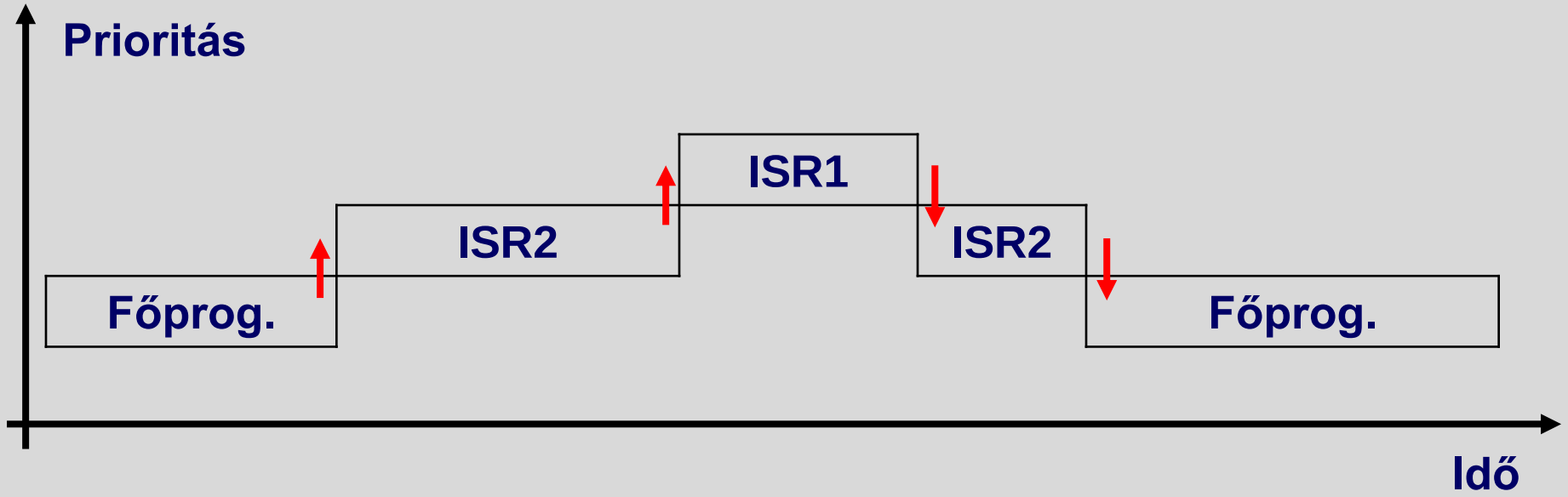
# Megszakítások maszkolása

- Egy adott megszakítás engedélyezésére, vagy tiltására alkalmazható
  - Regiszterek átállításával
- Maszkolt megszakítás: a program írója tilthatja/engedélyezheti az adott megszakítást
  - Valós idejű, kritikus folyamatoknál lesz fontos
- Nem maszkolható megszakítás nem tiltható le.

# Megszakítás szintjei

- Lehet:
  - Egyszintű-
  - Többszintű megszakítás végrehajtás: prioritási szintek vannak



# Megszakítások az Arduino platformon

- ATMega328 mikrovezérlő esetében: pin 2 és pin 3 használható külső megszakítás kezelésére
  - Más mikrokontrollereknél több, vagy kevesebb pin is lehetséges
- Gyakorlati alkalmazási példa:
  - Motor fordulatszámának mérése
    - Forgó részen állandó mágnes
    - pl. Hall-érzékelő adja a megszakítási kérelmet

# Megszakítások az Arduino platformon

- attachInterrupt(pin, ISR, mód); használata
  - Pin: meg kell adni a digitális pin számát, amely generálja majd a megszakítást
  - ISR (Interrupt Service Routine): egy típusnélküli függvény, amely a megszakításkéréskor meghívódik
  - Mód: Mikor triggerelődjön a megszakítás
    - LOW: megszakítás csak akkor, ha az adott lábon logikai 0 van
    - CHANGE: amikor az adott pin-en jelszint változás történik
    - RISING: logikai 0 → logikai 1 átmenetnél
    - FALLING: logikai 1 → logikai 0 átmenetnél

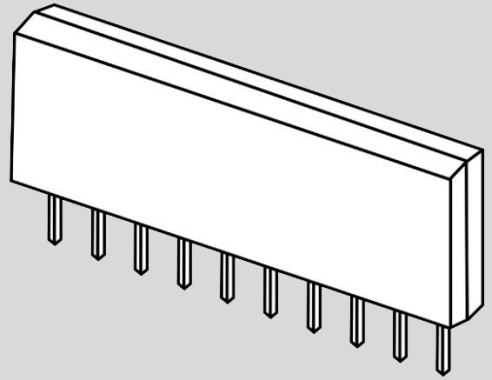
# Megszakítások az Arduino platformon

- Globális változó használata az ISR és a főprogram miatt.
- Az ISR által használt változó volatile legyen a deklarálásnál
- Maga az ISR rövid legyen → főprogram minél hamarabb folytatódjon
- Nincs az ISR-nek visszatérési értéke

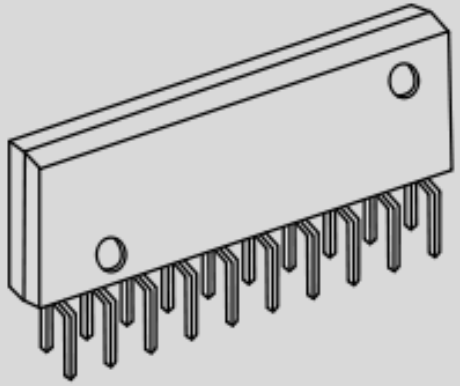
# Megszakítások az Arduino platformon

```
const int megszakitas_pin = 2;
const int led =  5;
const int an_pin = A0;
int be=0;
volatile int gomb_allapot = 0;
void setup() {
  pinMode(megszakitas_pin, INPUT); pinMode(led, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(megszakitas_pin), megszakitas_fgv, CHANGE);
  Serial.begin(9600);}
void loop()
{
  be=analogRead(an_pin);
  Serial.println(be);}
void megszakitas_fgv() {
  gomb_allapot = digitalRead(megszakitas_pin);
  digitalWrite(led, gomb_allapot);}
```

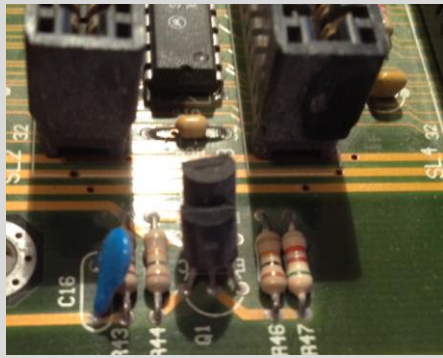
# Tokozások

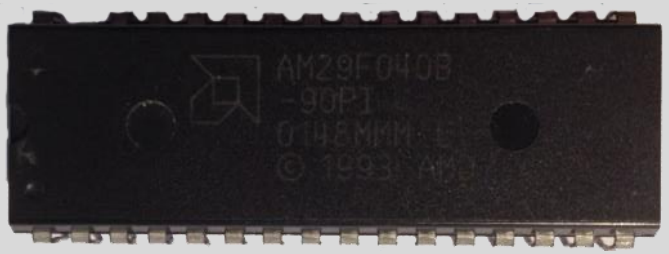


- **Egyoldali (SIL → Single In-Line) furatszerelt (THT) kivezetés**
  - **SIP → Single in-line package (Egysoros)**
  - **ZIP → Zig-zag In-Line Package, ez kétsoros**
  - **TO → Transistor Outline**



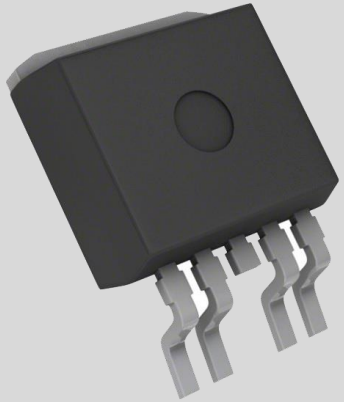
- **Kétoldalas (Dual In-Line) furatszerelt kivezetés**
  - **DIP → Dual in-line package (Kétsoros), 1965-ben jelent meg, a lábak távolsága 2,54 mm**
    - **PDIP → Műanyag tokozás**
    - **CDIP → Kerámia tokozás**
    - **SPDIP → kisebb a kivezetések közötti távolság**

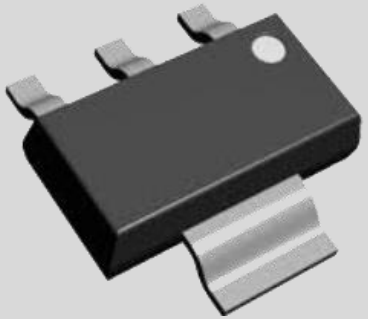




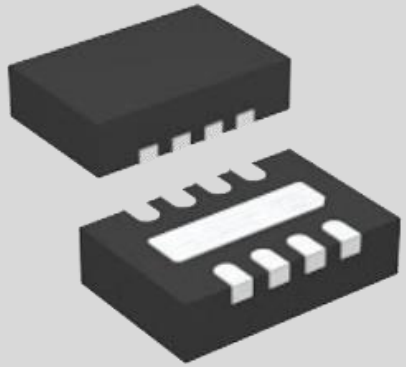
# Tokozások

- **Egyoldali felületszerelt (SMT) kivezetés**
  - SOT tokozások (Small Outline Transistor)
  - DPAK tokozások (DecaWatt Package)



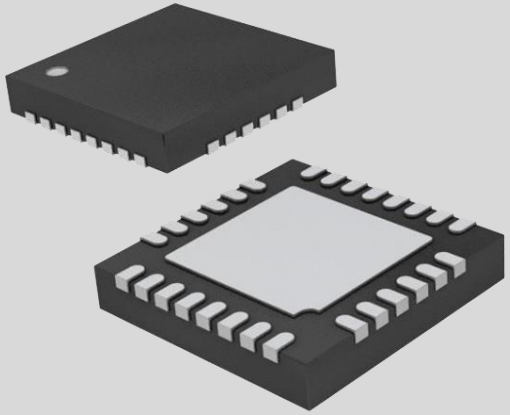


- **Kétoldalas felületszerelt kivezetés**
  - SO tokozások
    - SOIC: Small Outline IC
    - SSOP, TSOP, stb.
  - DFN: Dual Flat No Leads

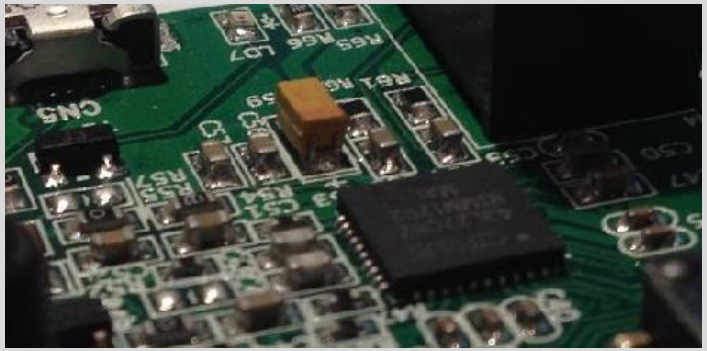
# Tokozások


- A négy oldalas kivezetés a 90-es évek elején terjedt el Európában:
  - QFP → Quad Flat Package, lábszám 32 - 304 között, vastagság 2 – 3,8 mm
    - LQFP, CQFP, SQFP, TQFP, stb.
    - TQFP: Nagyon vékony kivitelű tok, lábszám 32 - 256 közötti, vastagsága 0,8 – 1,4 mm, a lábak közötti távolság: 0,4 - 0,8 mm
  - QFN → Quad Flat No Lead, nincsenek lábak, helyette forrszemek vannak, sok variánsa van.



# Tokozások

- A lapka alján vannak a kivezetések (mátrix)
  - PGA → Pin Grid Array: Processzorok kedvelt tokozási típusa
    - A tok anyaga apján itt is van megkülönböztetés: CPGA, PPGA, stb.
  - LGA → Land Grid Array: Alján több sorban vannak a kivezetések
  - BGA → Ball Grid Array: Apró forrszemek, ráolvasztják a NYÁK-ra
    - RAM, GPU-knál jellemző tokozás típus



# Mikrovezérlők választásának irányelvei

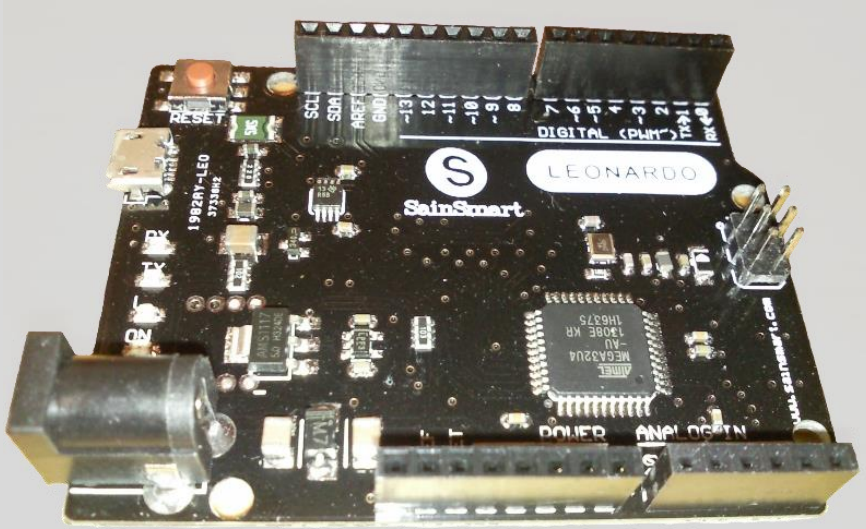
1. Az adott feladathoz szükséges be- és kimenetek számának meghatározása
2. Alkalmazandó A/D átalakító felbontása, sebessége
3. Feszültségszintek: LVTTL, vagy TTL
4. A mikrokontroller, vagy a board befoglaló mérete
5. Operatív tár mérete
6. PWM csatornák száma
7. Működési hőmérséklet intervalluma
8. Órajel

# Mikrovezérlő gyártók

1. Atmel www.atmel.com/
2. Microchip Technology http://www.microchip.com/
3. Intel https://www.intel.com/content/www/us/en/homepage.html
4. Texas Instruments https://www.ti.com/
5. STMicroelectronics www.st.com/
6. NXP Semiconductors https://www.nxp.com/
7. Cypress www.cypress.com/

# Atmel


- **Pár ismertebb 8 bites mikrokontroller:**
  - **ATmega2560**
    - **TTL, órajel: 16MHz**
    - **Digitális ki-/bemenet: 54 db., ebből 15 db. PWM, 16 db. analóg bemenet (10 bites)**
  - **ATmega168**
    - **TTL, órajel: 16 MHz**
    - **Digitális ki-/bemenet: 23 db., ebből 6 db. PWM, 8 db. analóg bemenet (10 bites)**
  - **ATmega32u4**
    - **TTL, órajel: 16 MHz**
    - **Digitális ki-/bemenet: 20 db., ebből 7 db. PWM, 12 db. analóg bemenet (10 bites)**

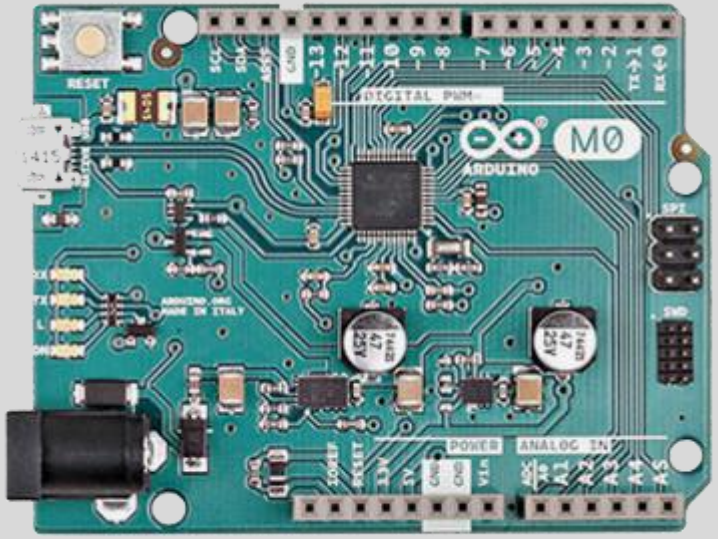


# ARM architektúrára épülő mikrovezérlők

- ARM → Advanced RISC Machine, 1985-ben jelent meg
- RISC architektúrára épül
- 32/64 bites mikrovezérlők
- A 32 bites RISC mikrokontrollerek több, mint ¾-e ARM
- Mobiltelefonokban is ARM  van
- Harvard architektúrára épül

# Atmel

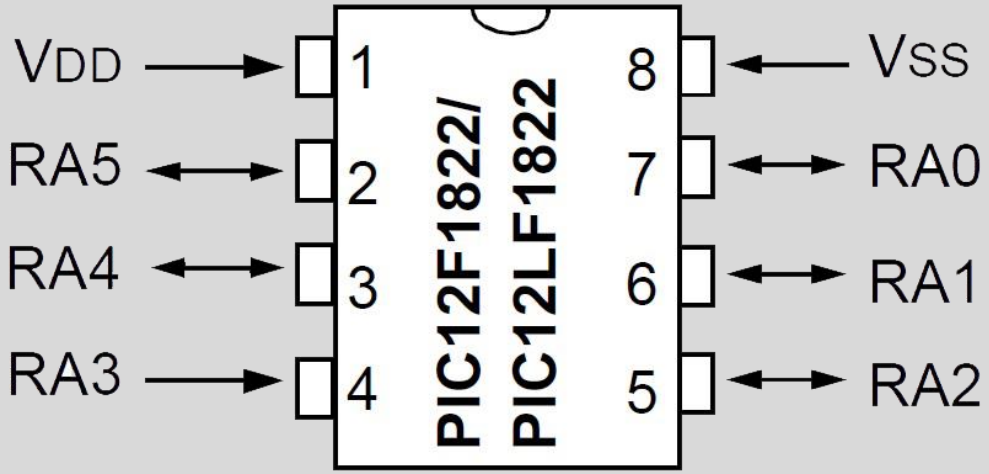

- **AT91SAM3X8E**
  - **ARM architektúra**
  - **LVTTL**
  - **Órajel: 48 MHz**
  - **6 db. Analóg bemenet, 1 db. DAC, 20 db. Digitális ki-/bemenet, ebből 12 db. PWM**




- **ATSAMD21G18**
  - **ARM architektúra**
  - **LVTTL**
  - **Órajel: 84 MHz**
  - **12 db. Analóg bemenet, 2 db. DAC, 54 db. Digitális ki-/bemenet, ebből 12 db. PWM**

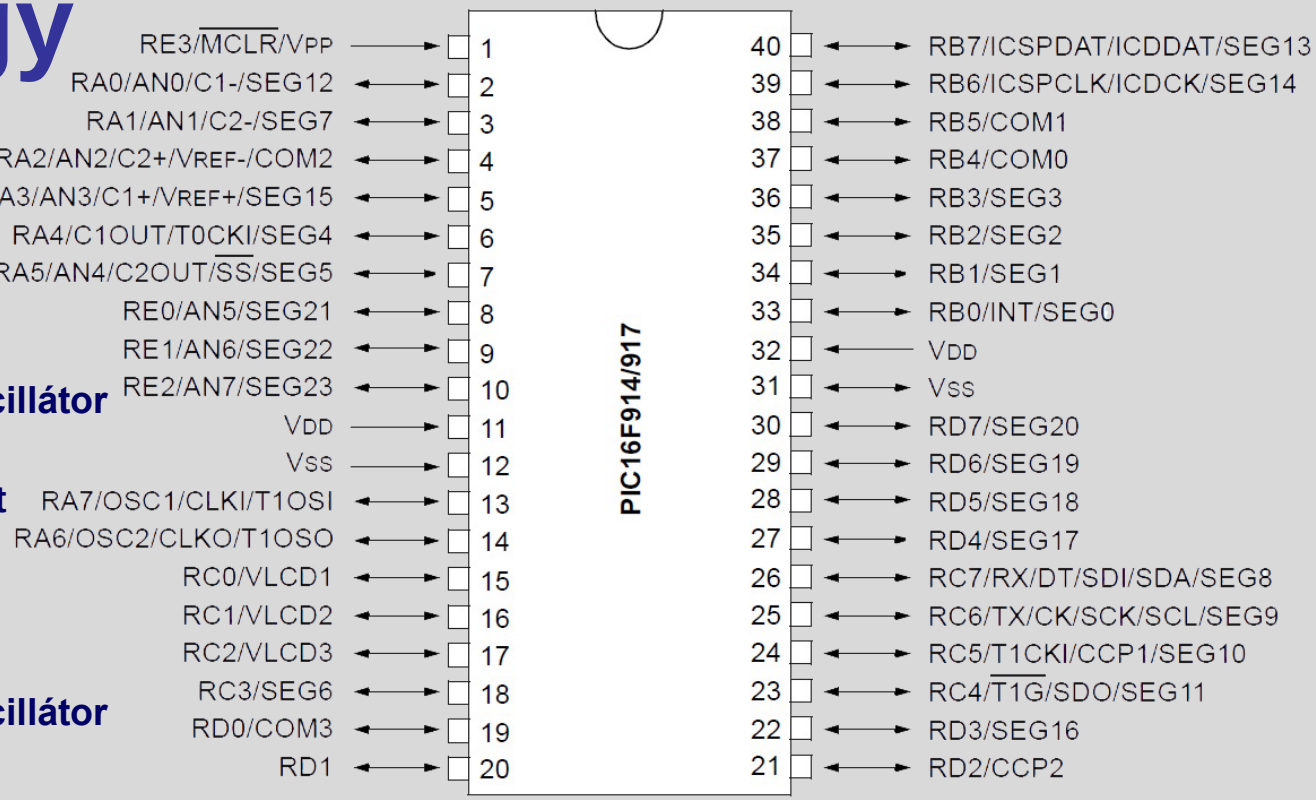
# Microchip Technology

- PIC12F, PIC16F, PIC18F család
  - 8 bites mikrovezérlők
  - 12F-nek 8 lába van, a 16F-nek 14-64 lábszáma van
- Néhány PIC12F mikrovezérlő:
  - PIC12F675
    - 5 MIPS, 64 byte RAM, PM: 1,75 kB, belső 4 MHz oszc.
    - 4 db. analóg bemenet (10 bites felbontás)
  - PIC12F1822
    - 8 MIPS, 128 byte RAM, PM: 3,5 kB, belső 35 MHz oszc.
    - 4 db. analóg bemenet (10 bites felbontás)

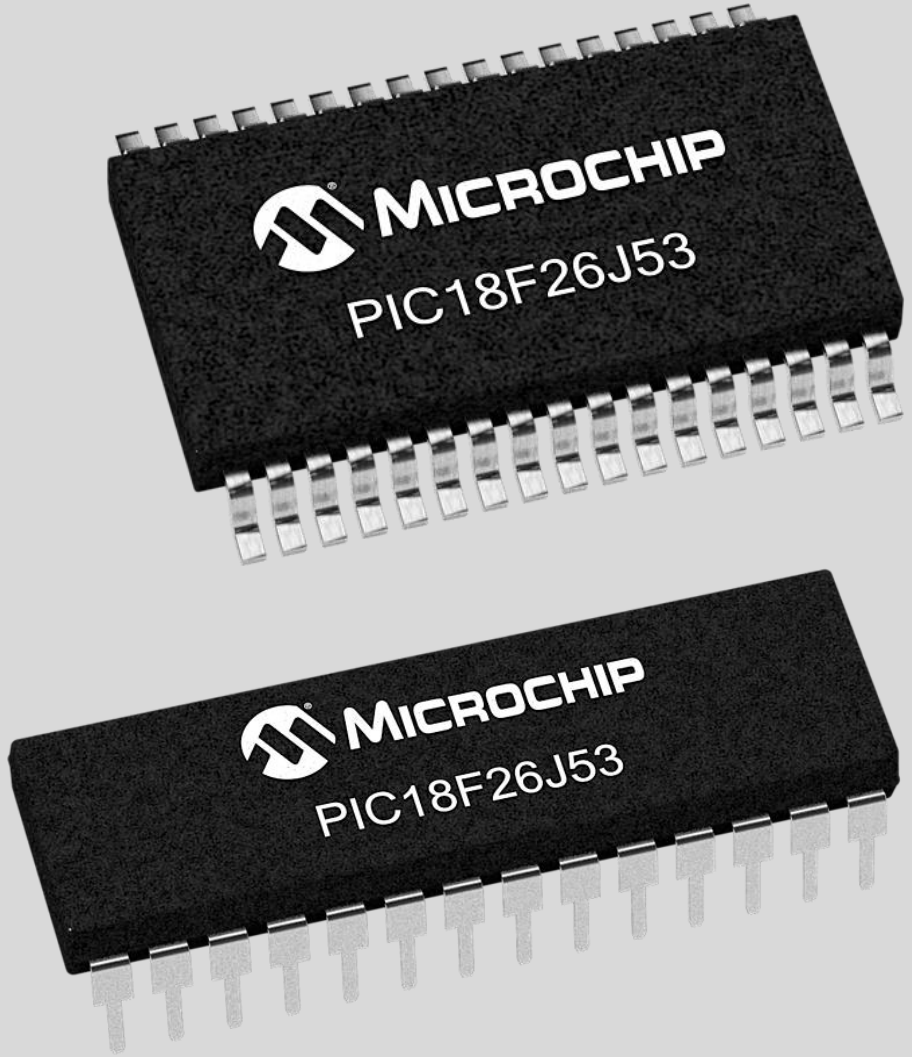
# Microchip Technology

- **Néhány PIC16F mikrovezérlő:**
- **PIC16F917**
  - **5 MIPS, 352 byte RAM, PM: 14 kB, belső 32 kHz – 8 MHz oszcillátor**
  - **8 db. analóg bemenet (10 bites felbontás), 35 db. ki-/bemenet**
- **PIC16F877**
  - **5 MIPS, 368 byte RAM, PM: 14 kB, belső 32 kHz – 8 MHz oszcillátor**
  - **8 db. analóg bemenet (10 bites felbontás), 33 db. ki-/bemenet**

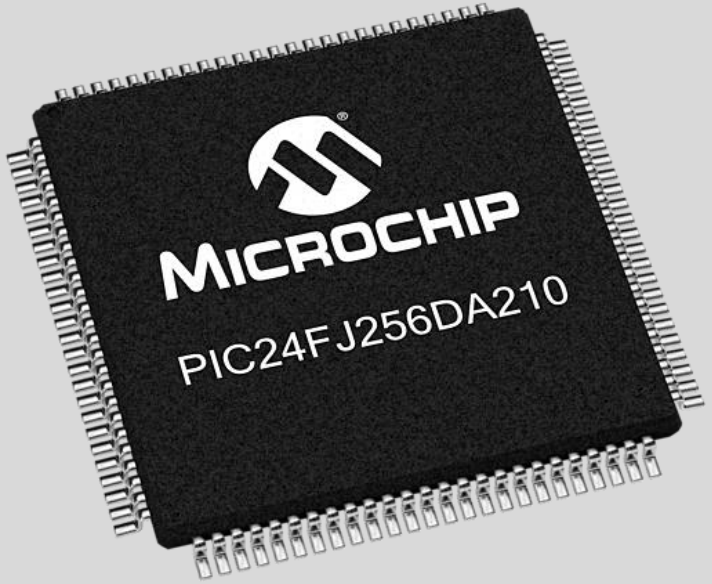



# Microchip Technology

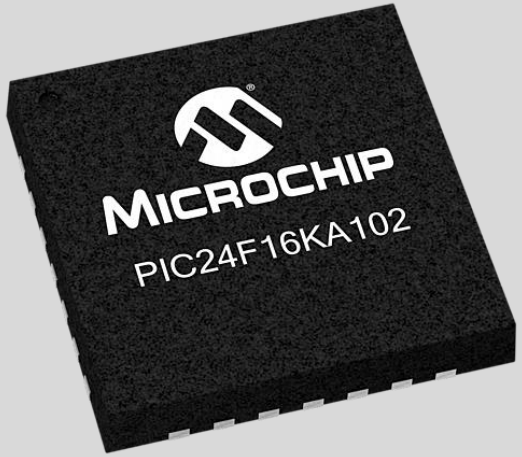
- **Néhány PIC18F mikrovezérlő:**
- **PIC18F458**
  - **10 MIPS, 1536 byte RAM, PM: 32 kB, Órajel max. 40 MHz**
  - **8 db. analóg bemenet (10 bites felbontás)**
- **PIC18F26J53**
  - **12 MIPS, 3800 byte RAM, PM: 64 kB,**
  - **13 db. analóg bemenet (12 bites felbontás)**



# Microchip Technology




- PIC24F család
  - 16 bites mikrovezérlő, 24 bites PM
- Néhány típus:
- PIC24FJ256DA210
  - 16 MIPS, 96 kB RAM, PM: 256 kB, 8 MHz, 32 kHz belső oszc.
  - 24 db. analóg bemenet (10 bites felbontás), 84 db. I/O
- PIC24F16KA102
  - 16 MIPS, 1,5 kB RAM, PM: 16 kB, 8 MHz, 32 kHz belső oszc.
  - 9 db. analóg bemenet (12 bites felbontás), 24 db. I/O



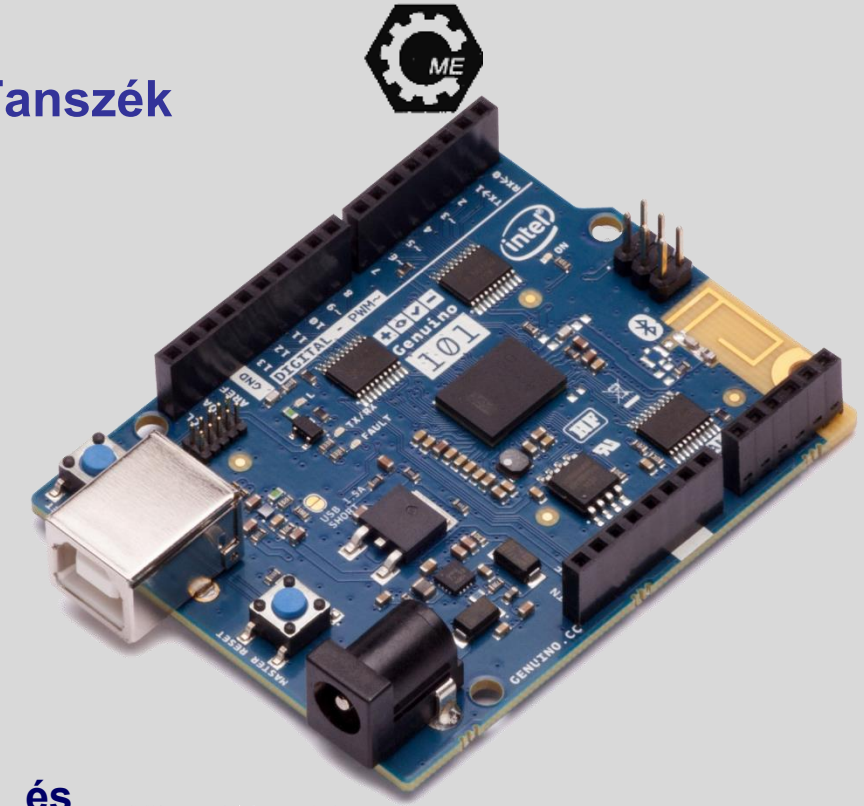
# PIC programozók

- MPLAB ICD-3 programozó és debugger
- PICkit programozók
  - PICkit 2
  - PICkit 3

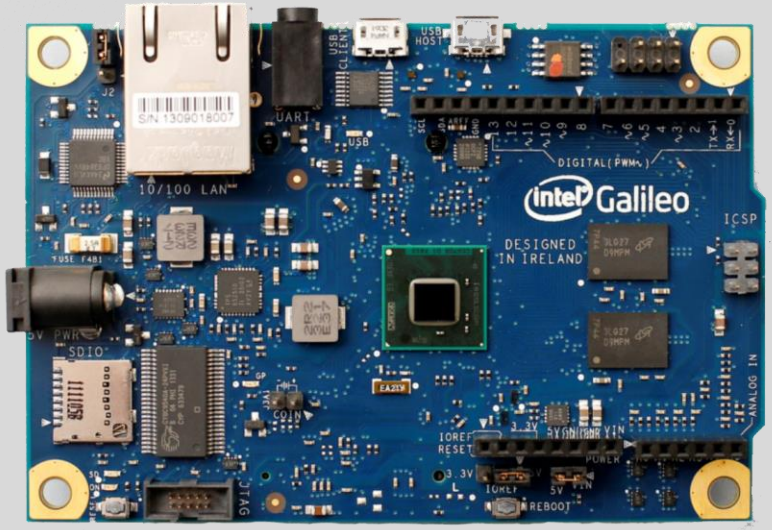

# ARC architektúra

- ARC – Argonaut RISC Core
  - 32 bites CPU család, SoC eszközöknél széles körben használják
  - Eredetileg az Argonaut Games keretében 3D-s pipeline fejlesztési projektek miatt jött létre
  - RISC architektúrára épül
  - A processzorai kamerákban, televíziókban, autókban is megtalálhatók
  - Nagymértékben testre-szabhatók az ilyen processzorok
  - Támogatja a felhasználó által egyedileg definiált utasításokat

# Intel

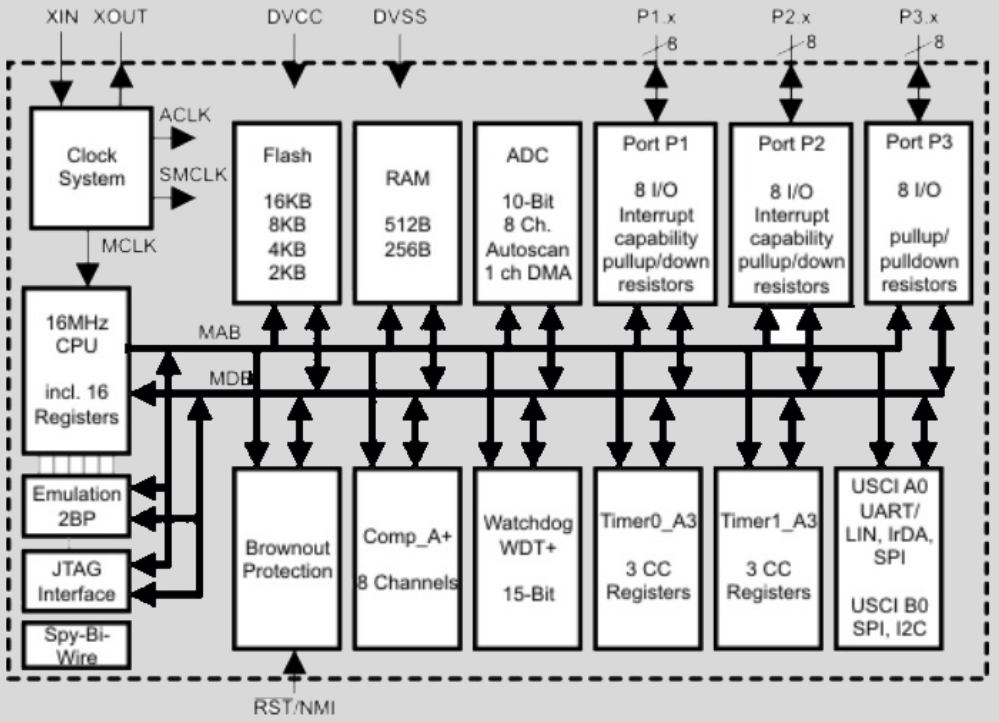


- Intel Curie
  - Intel Quark SE C1000, LVTTL (5 V toleráns I/O), órajel: 32 MHz
  - ARC architektúra
  - 14 db. digitális be-/kimenet, ebből 4 PWM, 6db. analóg bemenet
  - 32 bites címbusz, Bluetooth-al, 6 tengelyes gyorsulásmérővel és giroszkóppal
- Intel Galileo (Gen 1, Gen 2)
  - 32 bites Intel Quark X1000 (Intel Pentium), 400 MHz , 256 MB DDR3 RAM
  - 6 db. analóg bemenet: AD7298 (12 bites felbontás), 14 digitális be-/kimenet, ebből 6 PWM



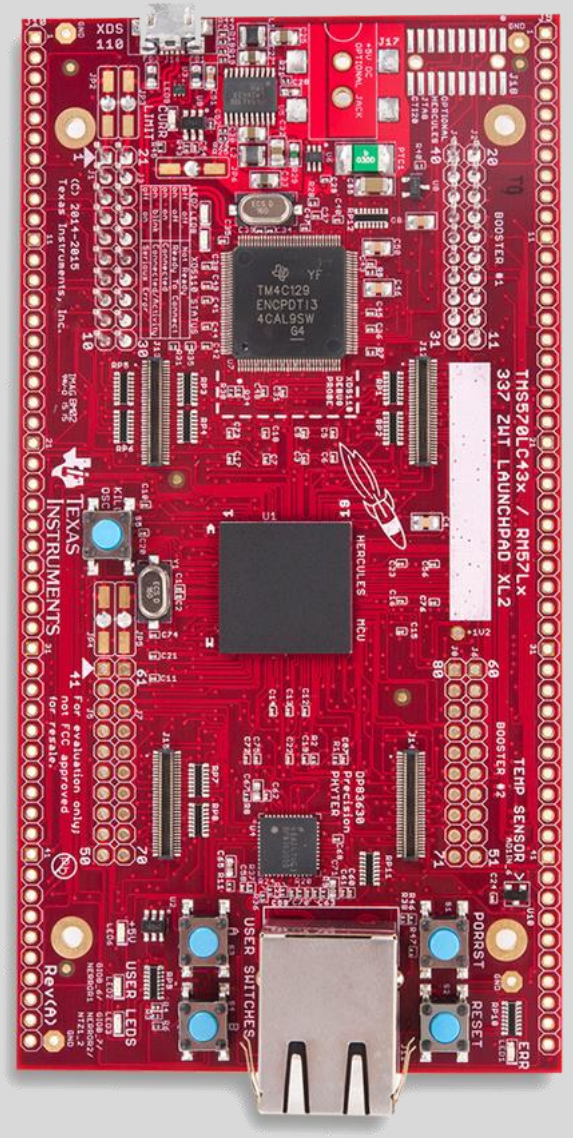
# Texas Instruments

- **MSP-EXP430G2 fejlesztői Kit (MSP430G2553 mikrokontrollerrel)**
  - **16 bites mikrovezérlő, órajel 16 MHz, 0,5 kB RAM, LVTTL**
  - **10 bites ADC, 8 csatornával, 24 db. GPIO láb**



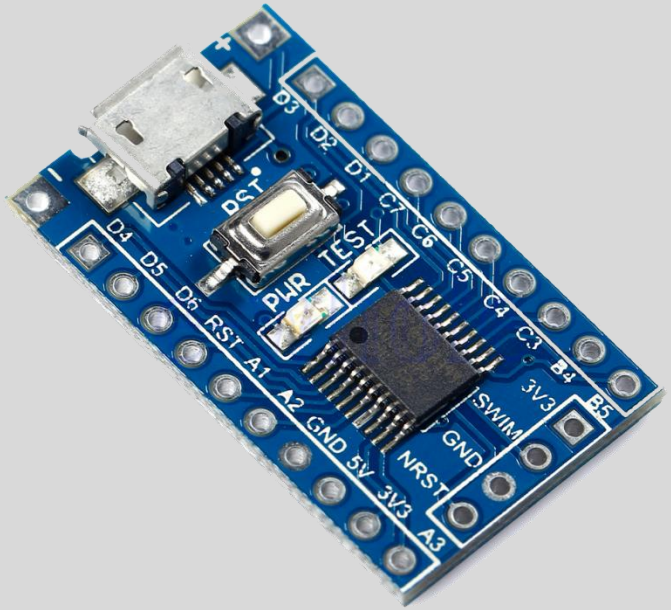
# Texas Instruments

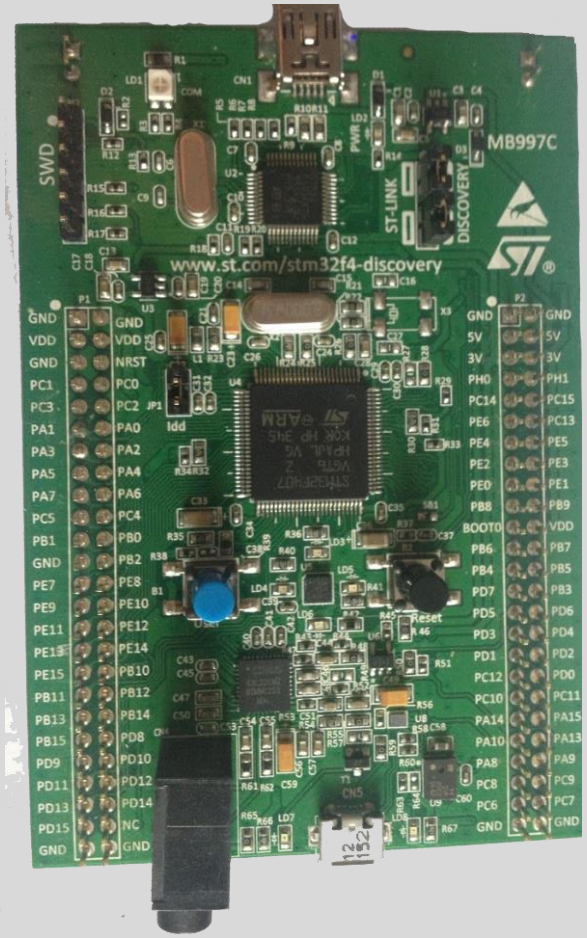
- **Hercules RM57Lx fejlesztői Kit (RM57L843 Hercules™ mikrokontrollerrel)**
  - **Kétmagos 32 bites RISC CPU, órajel 330 MHz, 512 kB RAM (ECC), LVTTL**
  - **12 bites 2 db. MibADC (Multibuffered) 32 és 25 csatornával, 145 db. GPIO láb, ebből 16 külső megszakításra használható**
  - **ARM architektúrára épül**



# STMicroelectronics

- **STM8S103F3**
  - 8 bites mikrovezérlő, órajel 16 MHz, 1 kB RAM, Harvard architektúra
  - 5 db. multiplexált analóg csatorna 10 bites felbontással
  - 28 db. be-/kimenet
- **STM32F4 Discovery**
  - STM32F407VGT6 32 bites mikrovezérlő, 192 kB RAM
  - Órajel: 168 MHz, ARM architektúra
  - 82 db. be-/kimenet
  - 2 db. 12 bites DAC, 3 db. 12 bites ADC (16 csatornával)

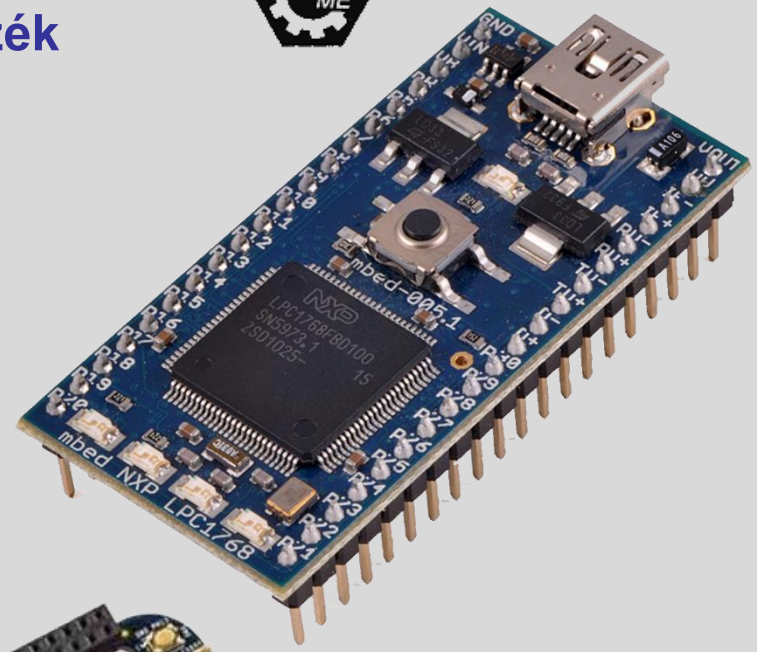


# Power architektúra

- POWER → Performance Optimization With Enhanced RISC
- 80-as évek végén az IBM alkotta meg
- RISC architektúra (load/store)
- Első megvalósítás: RS/6000 gépekben
- Később: PowerPC architektúra
- Az első PowerPC implementáció 1993-ban a PowerPC 601

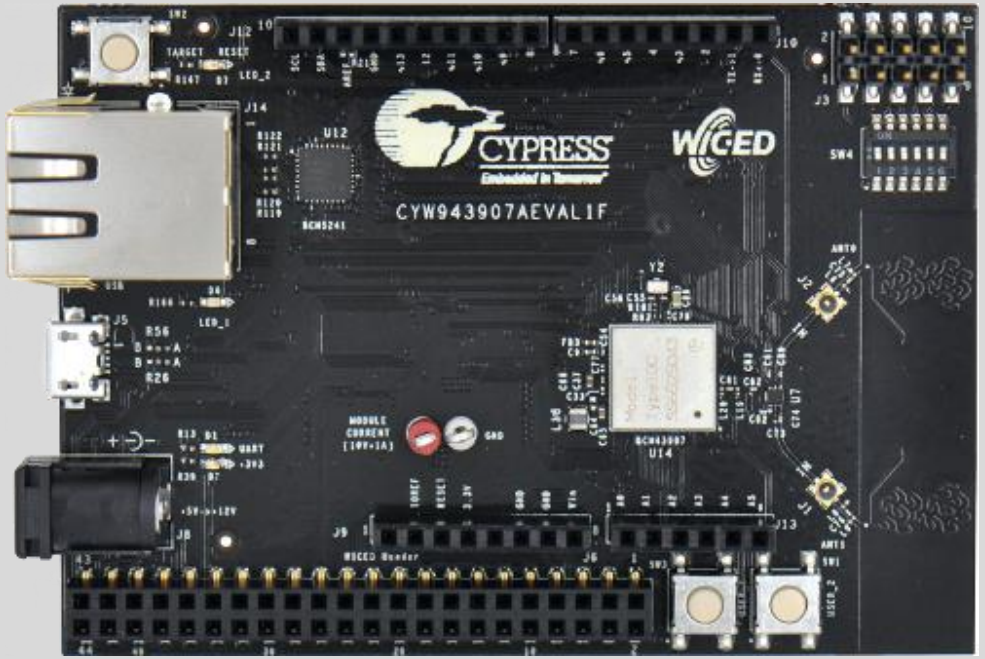
# NXP Semiconductors



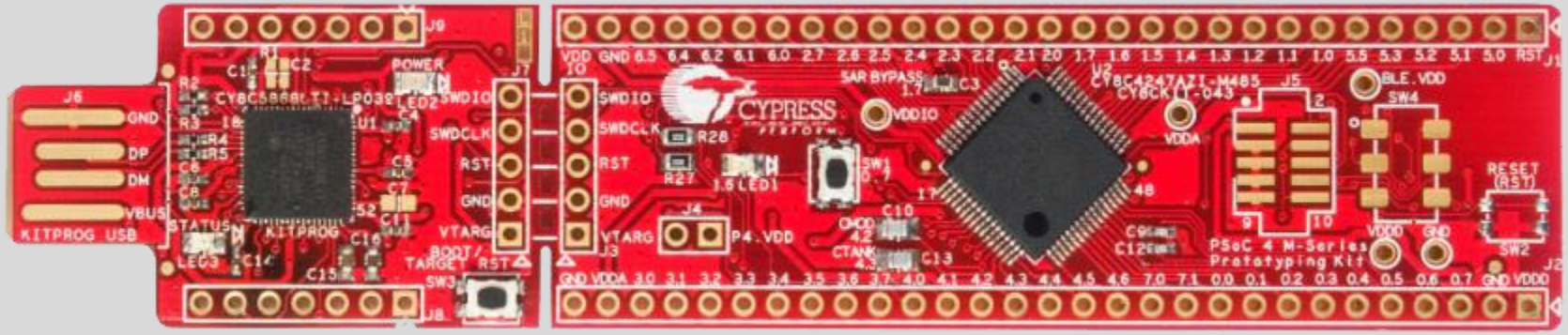

- **LPC1768 board (LPC1768FBD100 MCU-val)**
  - 32 bites mikrovezérlő, 64 kB SRAM, órajel 100 MHz, ARM architektúra
  - 8 db. analóg csatorna 12 bites felbontással, DAC (10 bites)
  - 70 db. GPIO
- **MPC5748G board**
  - 32 bites MCU, 2x160 MHz, 1x80 MHz CPU-k, 768 kB SRAM
  - Power architektúra, RISC
  - Maximum 246 db. GPIO
  - 2 db. 10 és 12 bites ADC (48 és 16 csatorna)

# Cypress

- **CYW943907AEVAL1F kit (Wi-Fi SoC)**
  - 32 bites mikrovezérlő, RISC, órajel max.: 320 MHz, SRAM: 2 MB
  - Kifejezetten az Internet of Things alkalmazásokra fejlesztve
  - 17 db. GPIO, ARM architektúra, 6 db. PWM modul



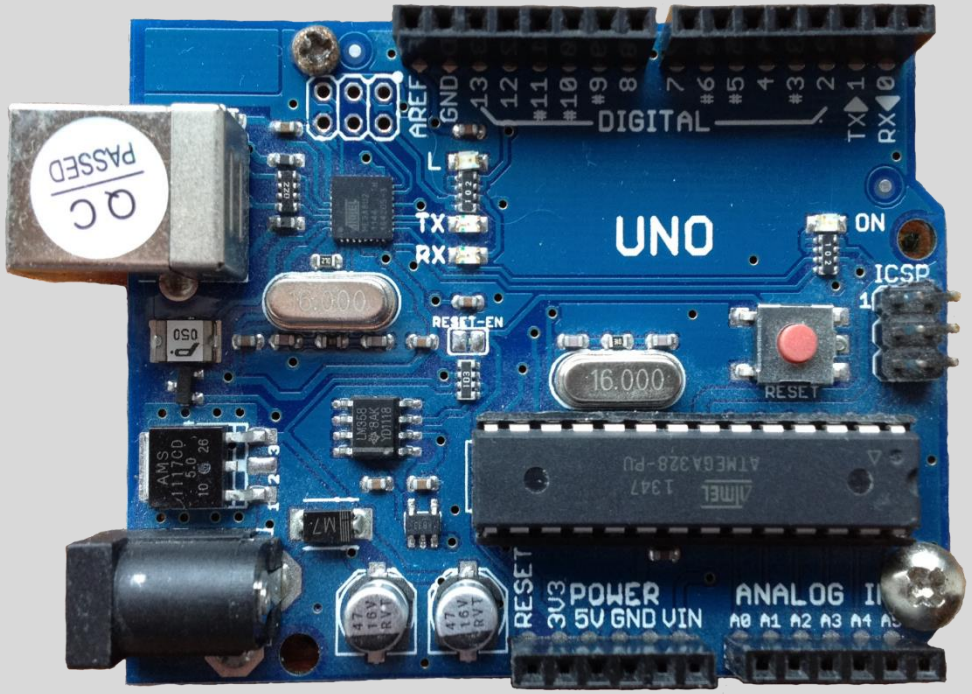

- **PSoC 4 prototyping kit (PSoC 4200M)**
  - 32 bites mikrovezérlő, RISC, órajel max.: 48 MHz, 16 KB SRAM
  - ARM architektúra, 51 db. GPIO
  - 1 db. SAR 12 bites ADC



# Gyakorlati feladatok

# Az Arduino fejlesztőplatform

- 2003: Wiring platform megalkotása, célja egy egyszerű és olcsó platform létrehozása, amely mindenki számára elérhető, nyílt forráskódú.
- 2005-ben Olaszországban született meg, a Wiring alapján.
- Sok modell jelent meg, a teljesség igénye nélül:
  - Arduino Mega, Arduino Leonardo, Arduino UNO, Arduino Nano, stb.



# Az Arduino Nano

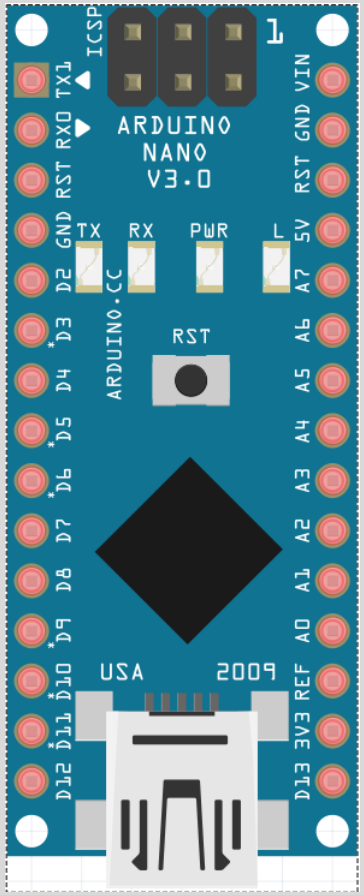
- **Specifikációi:**

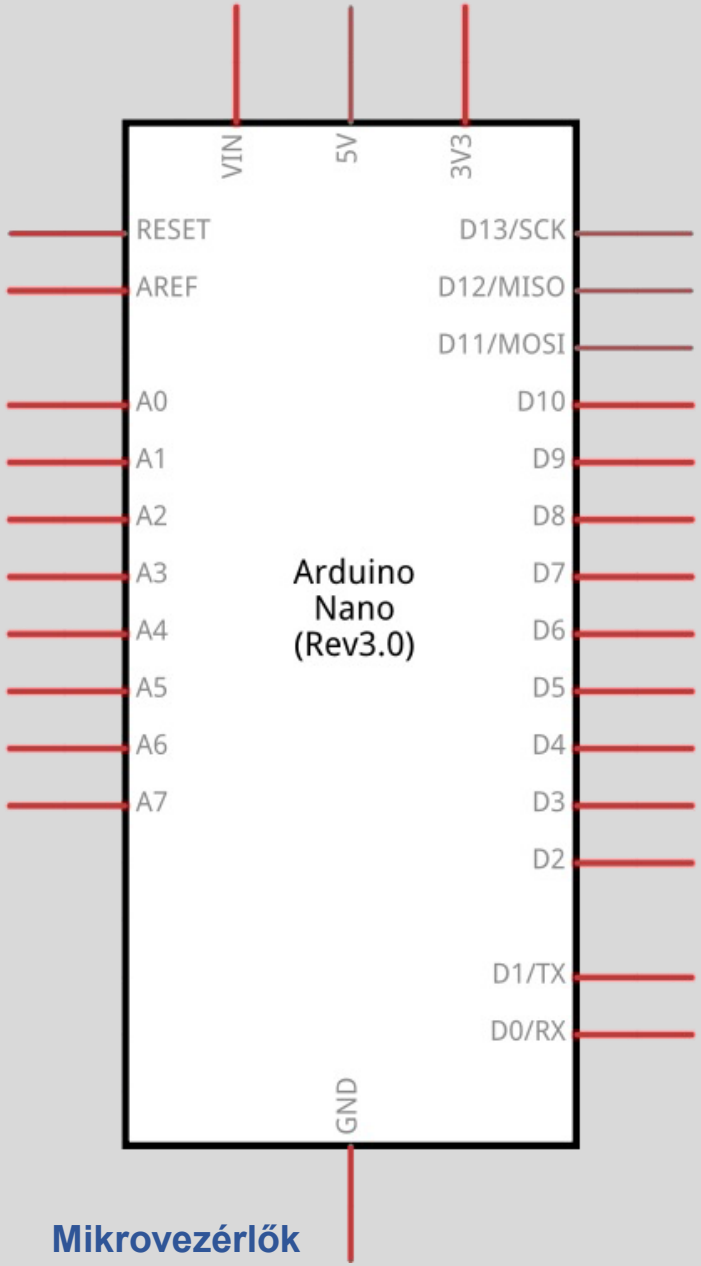
| **Megnevezés** | **Érték** |
|---|---|
| Mikrovezérlő: | Atmega168, **ATmega328** |
| Üzemi feszültség: | 5 V |
| Digitális I/O portok száma: | 22 |
| Analóg bemenetek száma: | 8 |
| Órajel: | 16 MHz |
| Flash memória: | 32 KB |
| Áramfelvétel: | 19 mA |

# Nano részei és a lábkiosztása

**Felülnézetből**

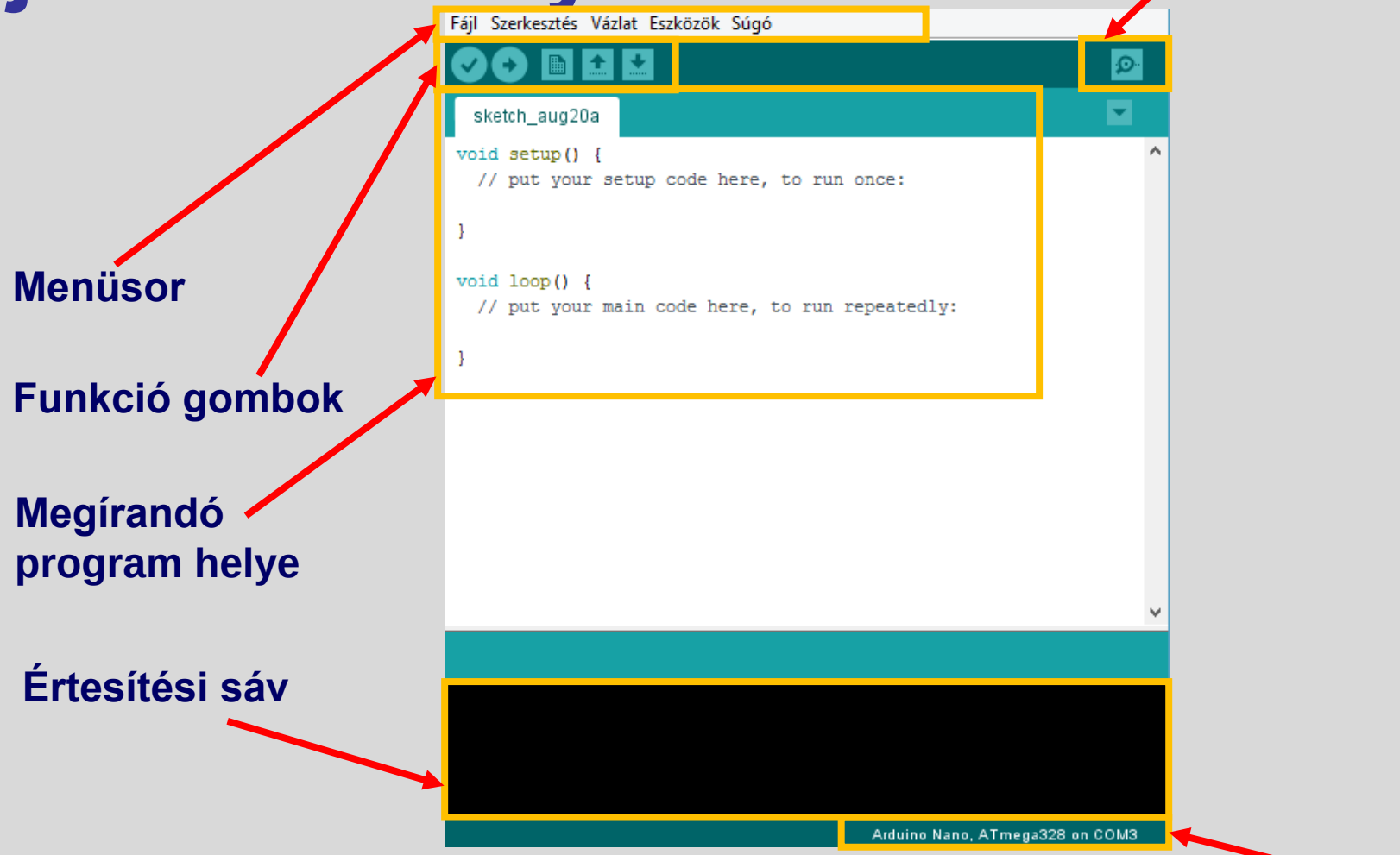

**Alulnézetből**



# Arduino IDE fejlesztői környezet

- C alapokon nyugvó C++ implementáció
- Két fő programrész
  - void setup { utasítások } a vezérlő indításakor csak egyszer fut le
  - void loop { utasítások } egy végtelen ciklusnak tekinthető

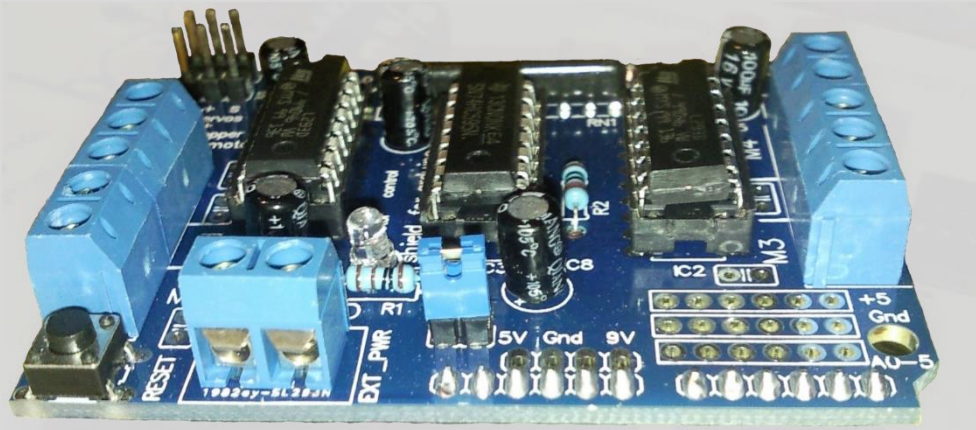


# Compiler

- Fordítóprogram: A megírt programot fordítja le a mikrovezérlő számára érthető nyelvre
- Arduino IDE a Build megnyomása után átalakítja a programot C++ szerkezetűre: .ino → .cpp
  - Több programrész esetén a rendszer összefűzi őket
  - Arduino.h header file hozzáadásra kerül
- Az avr-gcc fordítóprogram gépi kóddá alakítja
- Majd kombinálja az Arduino library-val
- A végeredmény 1 db. .hex kiterjesztésű file
- Ez a file kerül áttöltésre

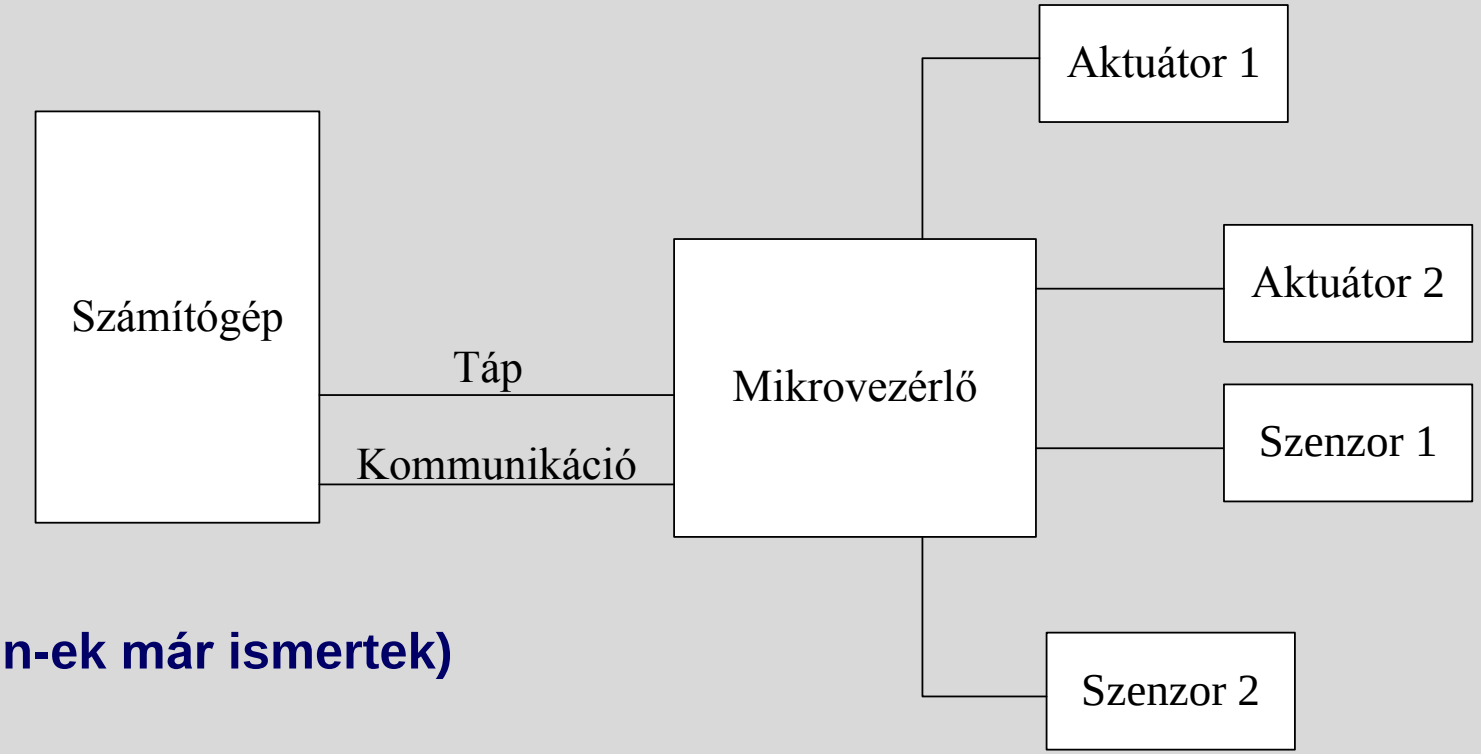
# Shield-ek használata

- Helytakarékosság és praktikusság és nem utolsó sorban funkciókibővítés
- Bővítő modulok a mikrovezérlőre, bizonyos feladatok célirányos megoldására
- Shield-ekre néhány példa:
  - Motor shield
  - Ethernet shield
  - Proto shield
  - LCD shield



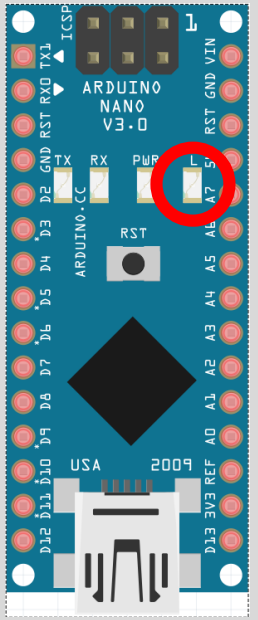
# Feladat megoldásának módszertana

1. Az adott feladat átgondolása
2. Áramkör megtervezése
   - (pl.: Fritzing, Eagle)
3. Ellenőrzés
4. Szükséges elemek beszerzése
5. Áramkör megépítése
6. Ellenőrzés
7. Programírás (A használni kívánt pin-ek már ismertek)
8. Ellenőrzés
9. A megírt program mikrokontrollerre töltése
10. Ellenőrzés



# Beépített LED villogtatása

- Első lépés a program megírása
- Majd a megírt kód ellenőrzése
- A megírt program mikrokontrollerre töltése
  - A mikrovezérlő csatlakoztatása után ki kell választani kommunikációs portot, valamint a platform típusát (Arduino Nano) és a processzort (Atmega328)



# Beépített LED villogtatása

- **A program kód:**

```
int led=13;

void setup() {

  pinMode(led, OUTPUI);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(100);
  digitalWrite(led, LOW);
  delay(100);
}
```

**// Pin deklarálása**

**//Egyszer fut le**

**//Pin kimenetként való definiálása**

**// A ciklusunk**

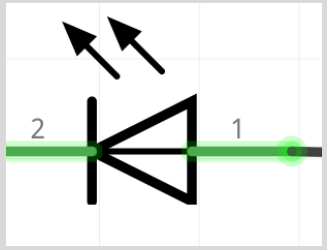
**//Kimenet logikai 1**

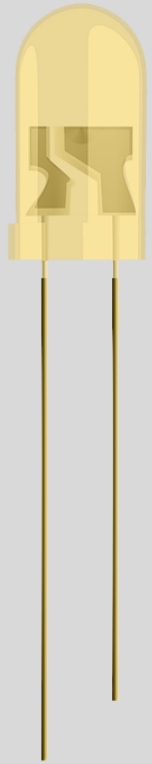
**// 0,1 sec várakozás**

**//Kimenet logikai 0**

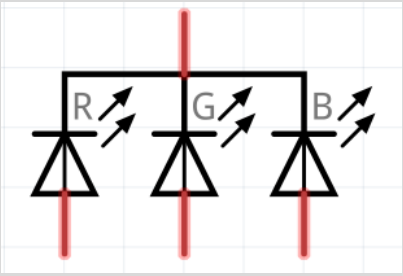
**//0,1 sec várakozás**

# LED-ek

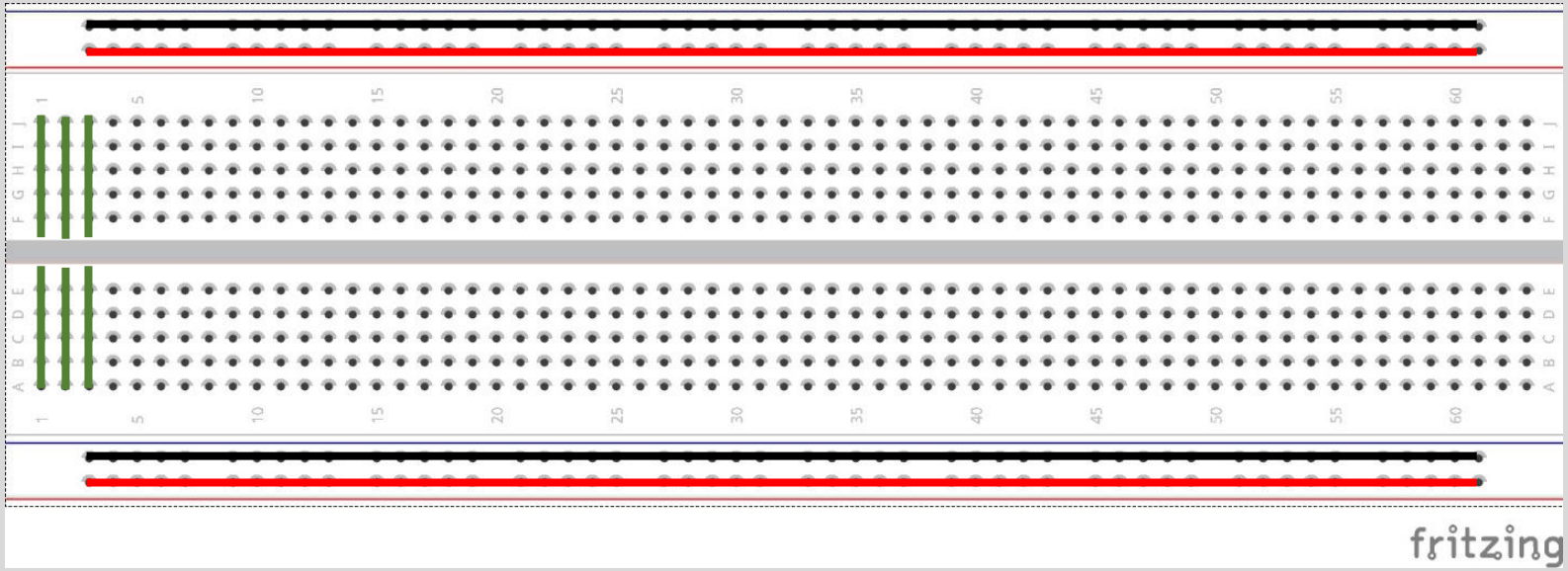


- LED: Light Emitting Diode, az elsőt 1962-ben találták fel
  - Kibocsátott fény hullámhossza függ a félvezető anyagától
- Polaritás érzékeny!
  - Fordított bekötésnél nem fog világítani!
  - Két kivezetése van: Anód (+), Katód (-)
- Kivitel szerint lehetnek
  - SMD, azaz felületszereltek
  - Hagyományos, furatszereltek
- Dual/Duo LED-ek
  - Párhuzamos → 2 kivezetés
  - Közös anód → 3 kivezetés
  - Közös katód → 3 kivezetés
- RGB LED-ek
  - Közös anód → 4 kivezetés
  - Közös katód → 4 kivezetés



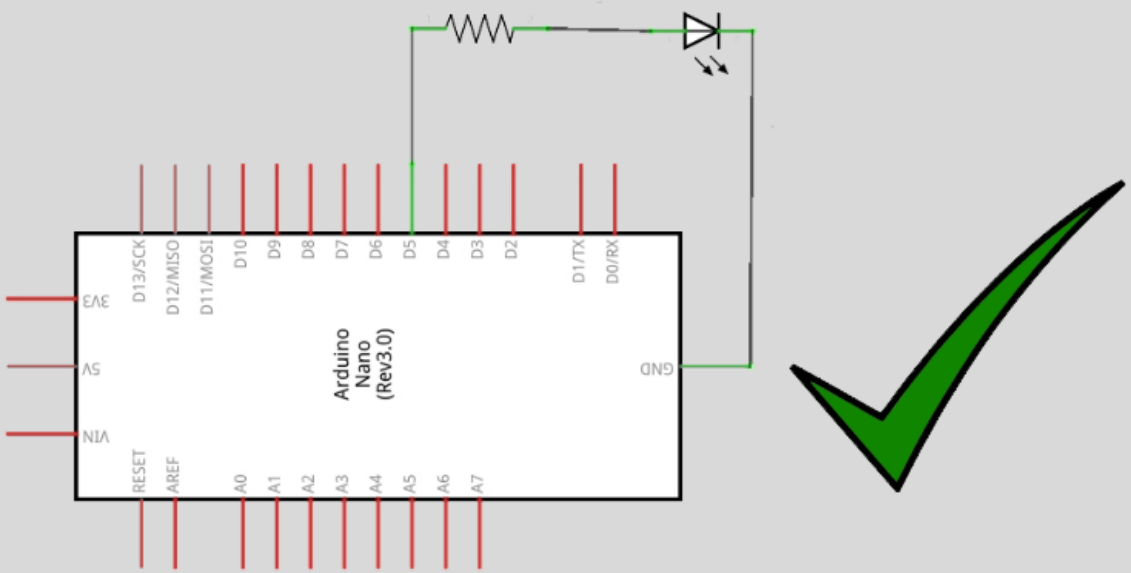
# Breadboard ismertetése

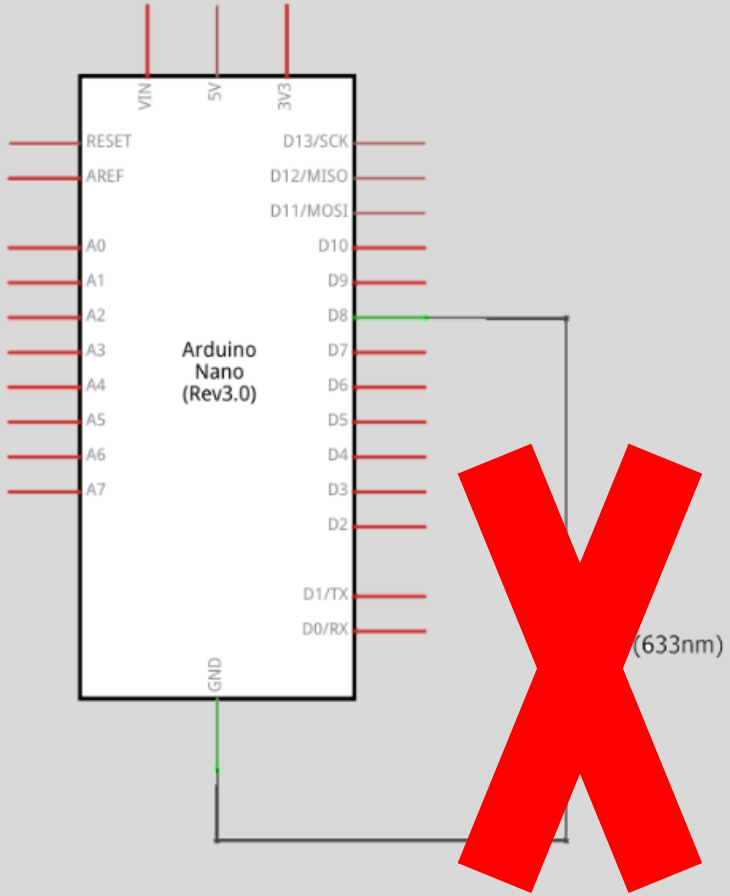



- Egyszerű használat
- Forrasztás nélkül tesztelhető a megépített áramkör
- Többféle kivitel, feladatokhoz mérten

# Külső LED villogtatása

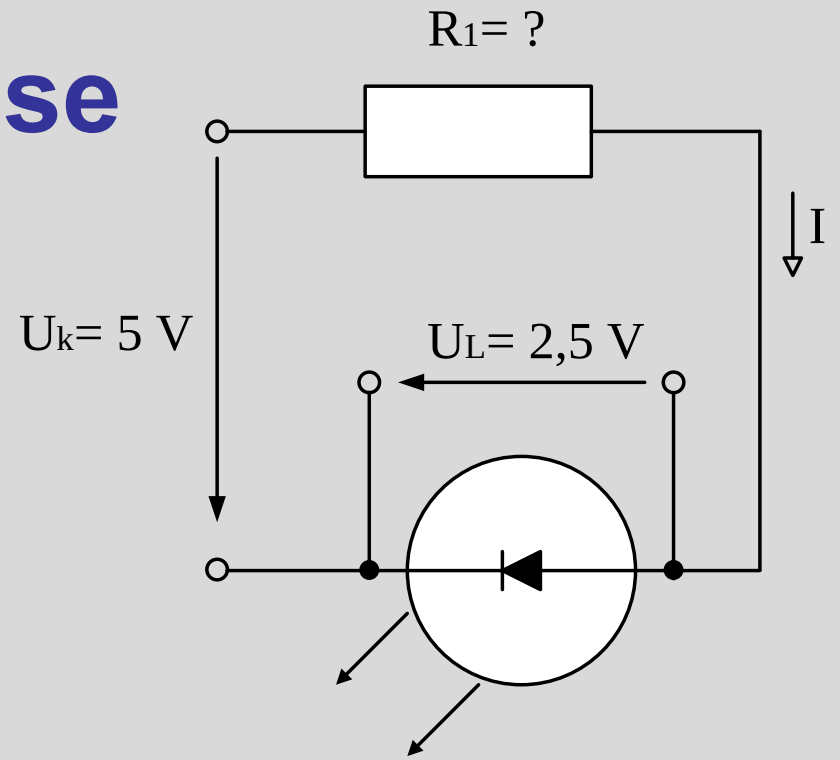
- **Első lépés az áramkör megtervezése**
- **A hagyományos LED közvetlenül rákötve a mikrovezérlőre tönkremegy! → előtét ellenállás szükséges**



# Előtét ellenállás méretezése

- **Egy hagyományos piros fényű LED adatai:**
  - **Üzemi feszültség: 2 - 2,5 V**
  - **Hullámhossz: 625 nm**
  - **Burkolat átmérő 5 mm**
  - **Világítási szög: 30°**
  - **LED-en átfolyó áram legyen: 12 mA**




**Hurok törvény:**

$$U_k = U_1 + U_L \rightarrow U_1 = U_k - U_L$$

**Ohm törvény:**

$$R = \frac{U}{I} \rightarrow R_1 = \frac{U_1}{I}$$

$$U_1 = 5\ V - 2{,}5\ V = 2{,}5\ V$$

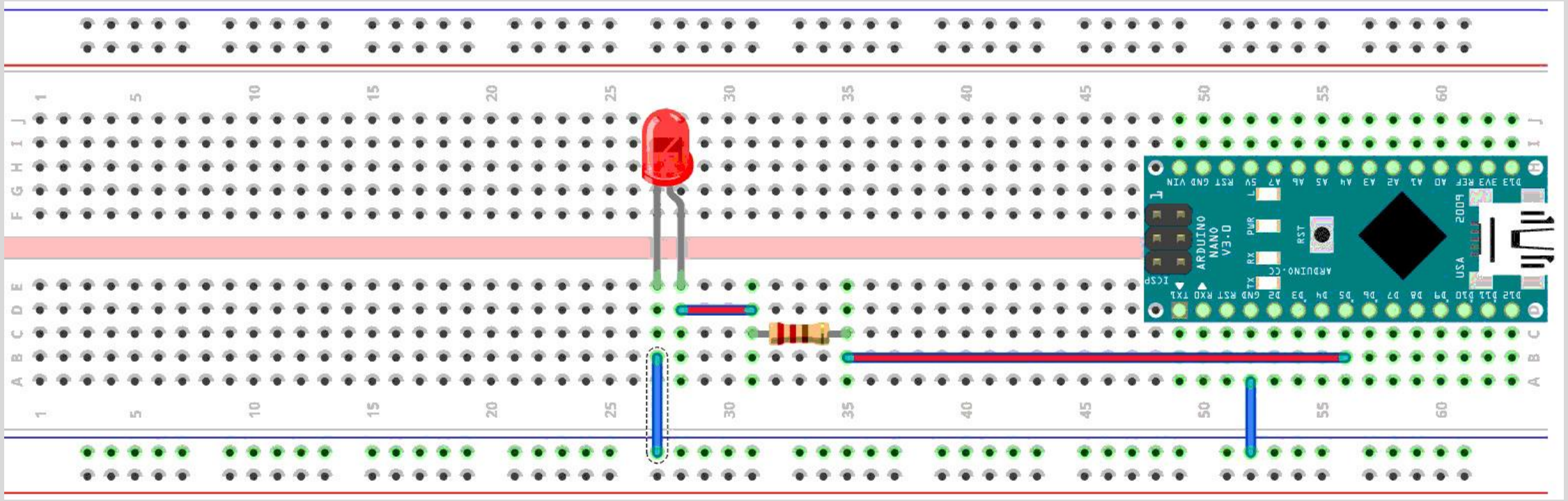
$$I = 12\ mA = 0{,}012\ A$$

$$R_1 = \frac{2{,}5\ V}{0{,}012\ A} = 208{,}33\ \Omega$$

$$R_{1szabv} = 220\ \Omega$$

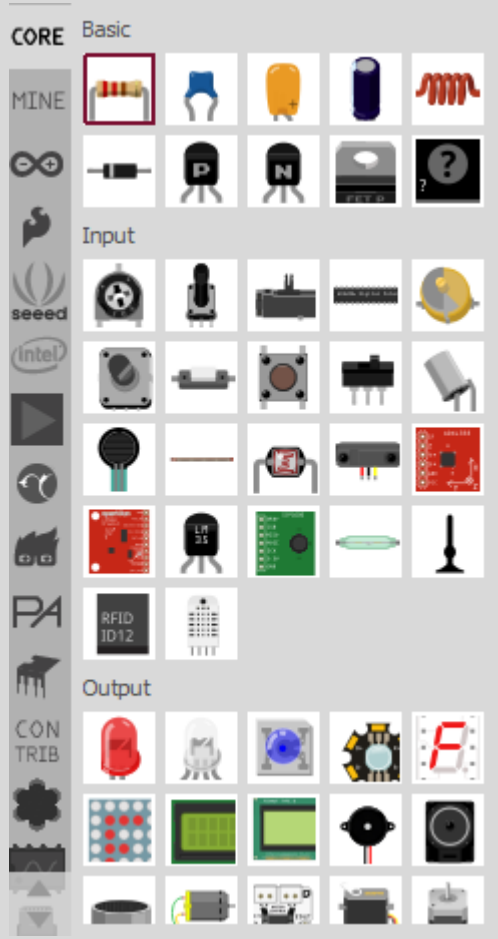
# Külső LED villogtatása

- **Bekötési séma**



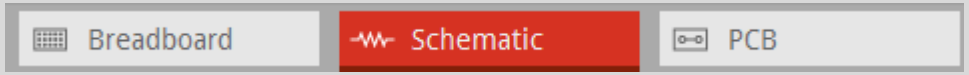
# Fritzing szoftver ismertetése

- Potsdami Egyetemen fejlesztették
- Nyílt forráskódú (A forráskód C++ nyelven íródott)
- „hobby” CAD szoftver
- Prototípusok elkészítésére hozták létre
- Támogatott operációs rendszerek: Unix, MAC OSX, Windows
- NYÁK tervezés

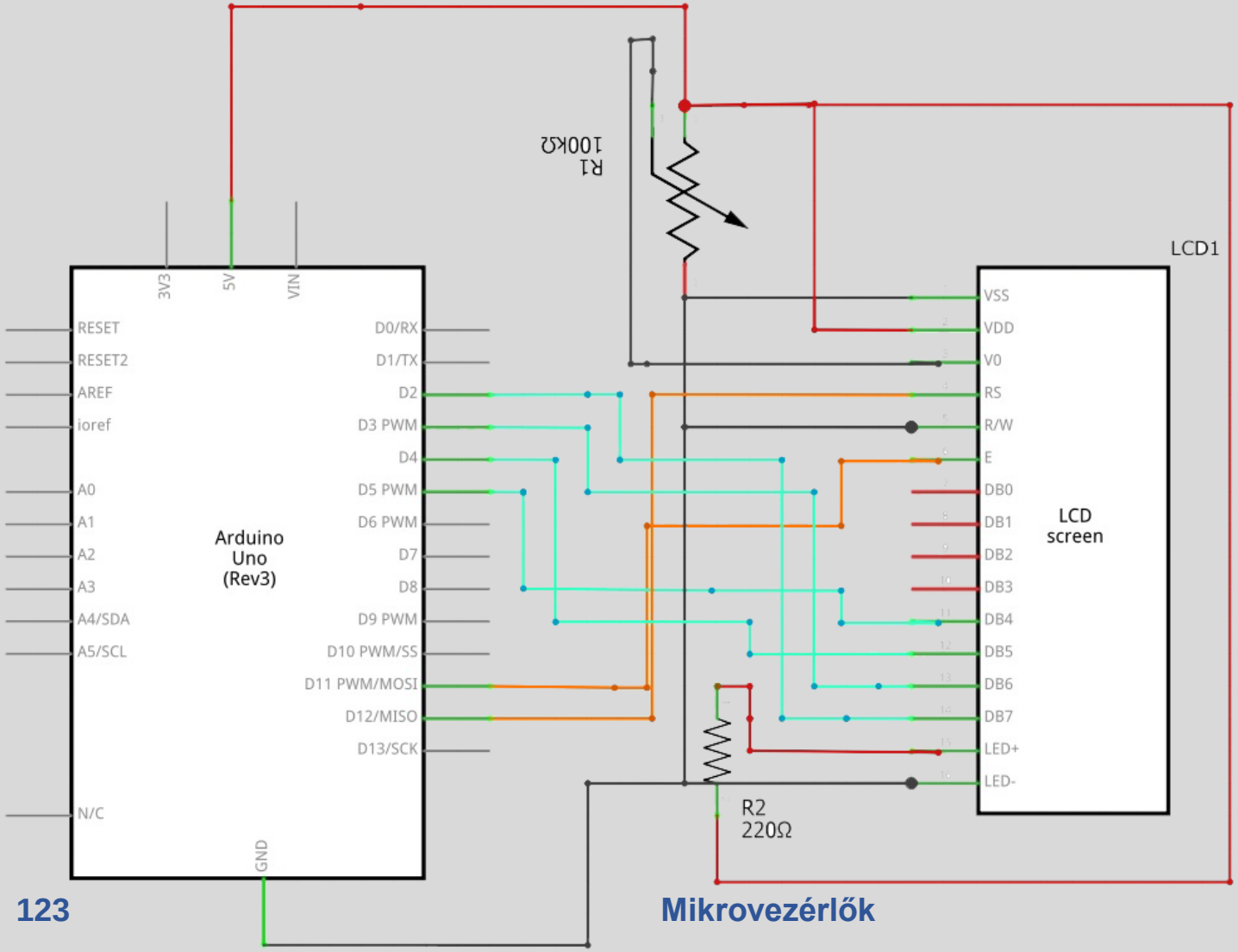


# Fritzing szoftver ismertetése

- **Kapcsolási rajz elkészítése →**

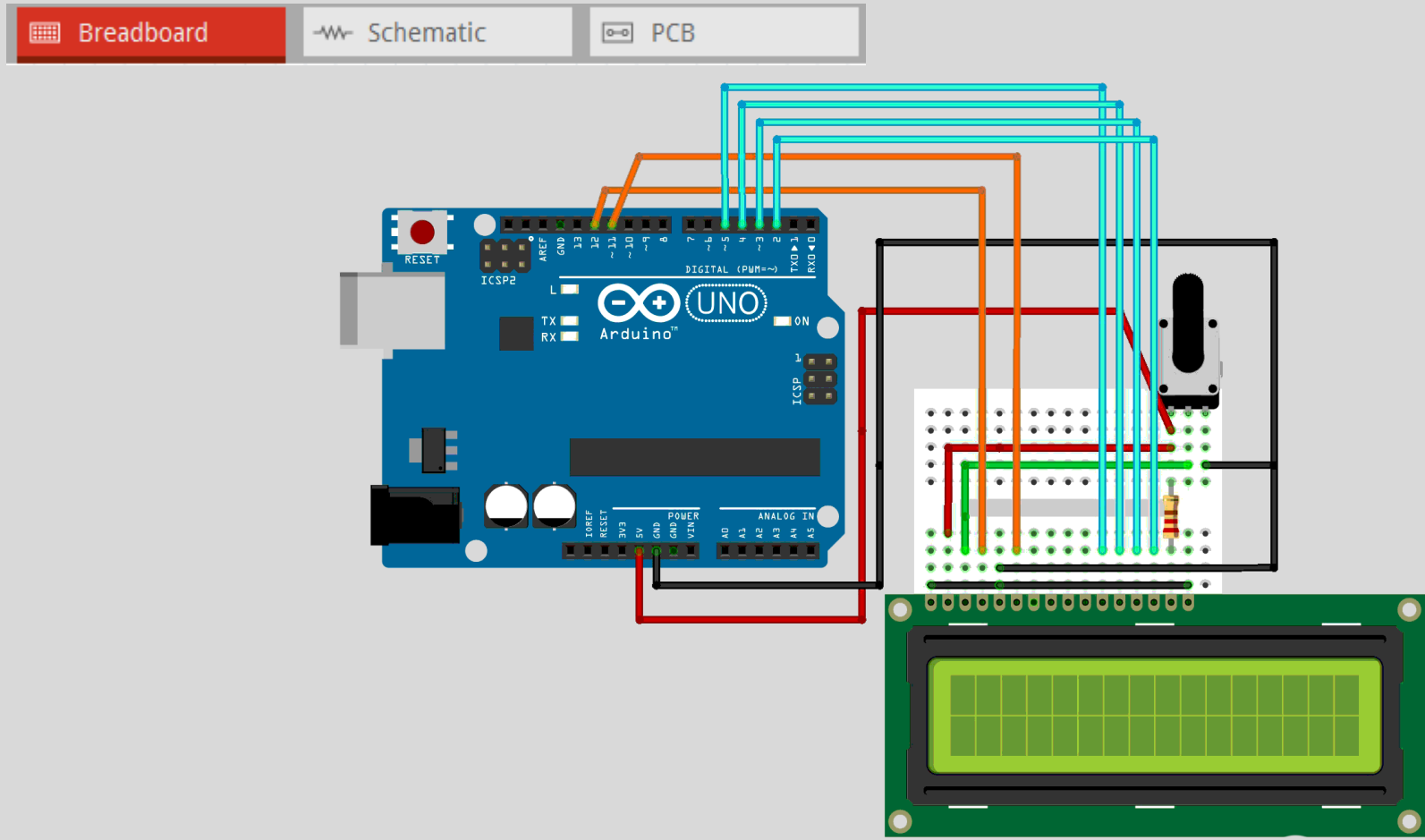



  - **Vezetékek színét lehet módosítani**
  - **Töréspontok betehetők**
  - **A kész kapcsolás exportálható számos formátumban**



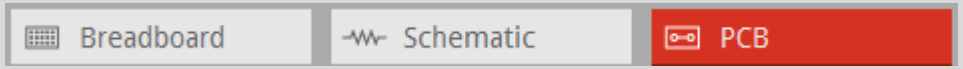
# Fritzing szoftver ismertetése

- **Bekötési séma szimulálása →**
  - **Többféle breadboard**
  - **Vezetékek színei módosíthatók**
  - **Töréspontok helyezhetők el**
  - **Alkatrész adatbázis**

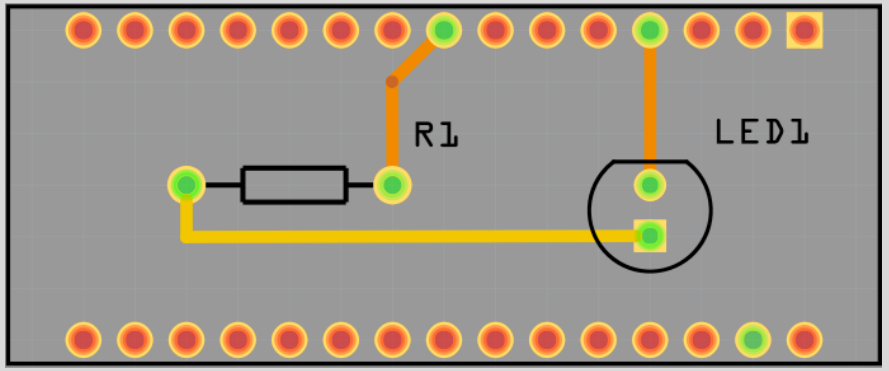


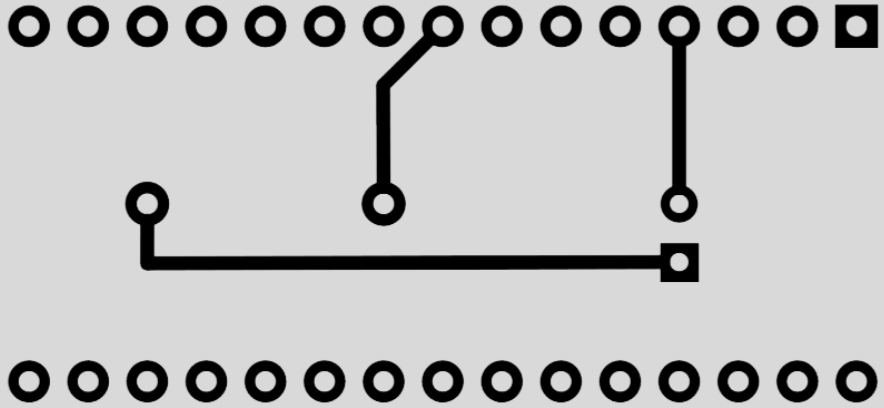
# Fritzing szoftver ismertetése

- **NYÁK tervezés →**




  - **Autoroute funkció**
  - **Tetszőleges módosítási lehetőségek**
  - **Exportálás (pl.: .pdf file)**
  - **Rétegek választása**



# LED fényerejének változtatása

- **Impulzusszélesség moduláció segítségével → A feszültséget, áramerősséget gyors kapcsolgatással szabályozzák.**
  - **Motor fordulatszáma, vagy esetünkben egy LED fényereje tetszőlegesen változtatható**
  - **A kitöltési idő változtatható, a periódusidő fix. Arduino PWM frekvencia: 500 Hz**

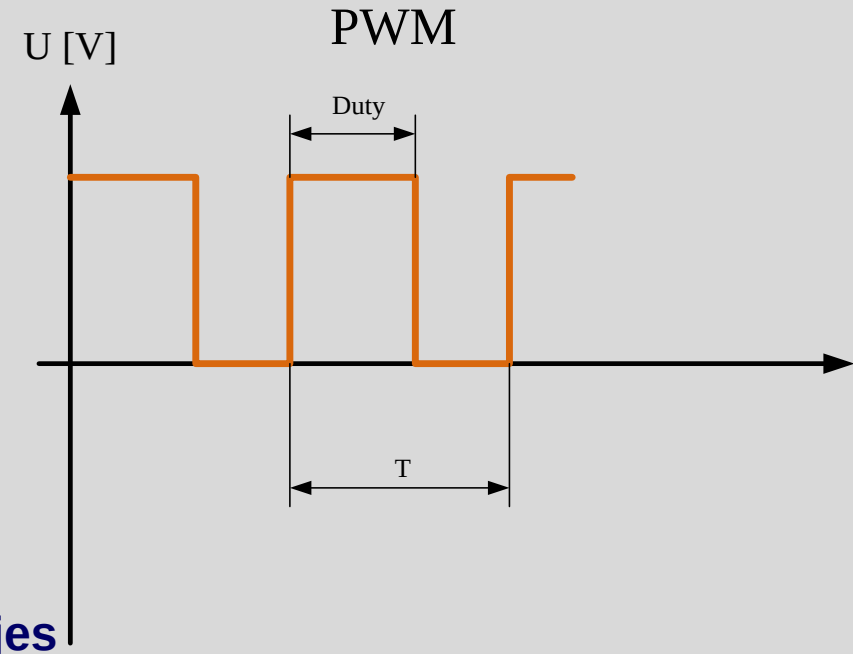
| Kitöltési tényező [%] | Értéke |
|---|---|
| 25 % | 64 |
| 50 % | 127 |
| 75 % | 191 |
| 100 % | 255 |

**0**

**LED nem világít**

**255**

**LED teljes fényerővel világít**



# LED fényerejének változtatása

- Összesen 6 láb használható PWM kimenetként, ezek: D3, D5, D6, D9-11

## Diszkrét értékekkel:

```
int pwm=5;
byte ertekek[]= {20, 40, 255, 67, 121, 190, 200, 255};

void setup() {

  pinMode(pwm, OUTPUT);
}

void loop() {
  for(int i=0; i<8; i++)
  {
    analogWrite(pwm, ertekek[i]);

    delay(200);
  }

}
```

## Összes lehetőségen végigszaladunk:

```
int pwm=5;

void setup() {

  pinMode(pwm, OUTPUT);
}

void loop() {
  for(int i=0; i<256; i++)
  {
    analogWrite(pwm, i);

    delay(25);
  }

}
```

# LED fényerejének változtatása

- Próbáljunk meg függvényt létrehozni a feladatra

**Egy lehetséges megoldás:**

```
const int pwm = 5;
void fenyero_valt()
{
  for( int i=0; i<256; i++)
  {
    analogWrite(pwm, i);
    delay(10);
  }
}
void setup(){
  pinMode(pwm, OUTPUT);}
void loop()
{fenyero_valt();}
```

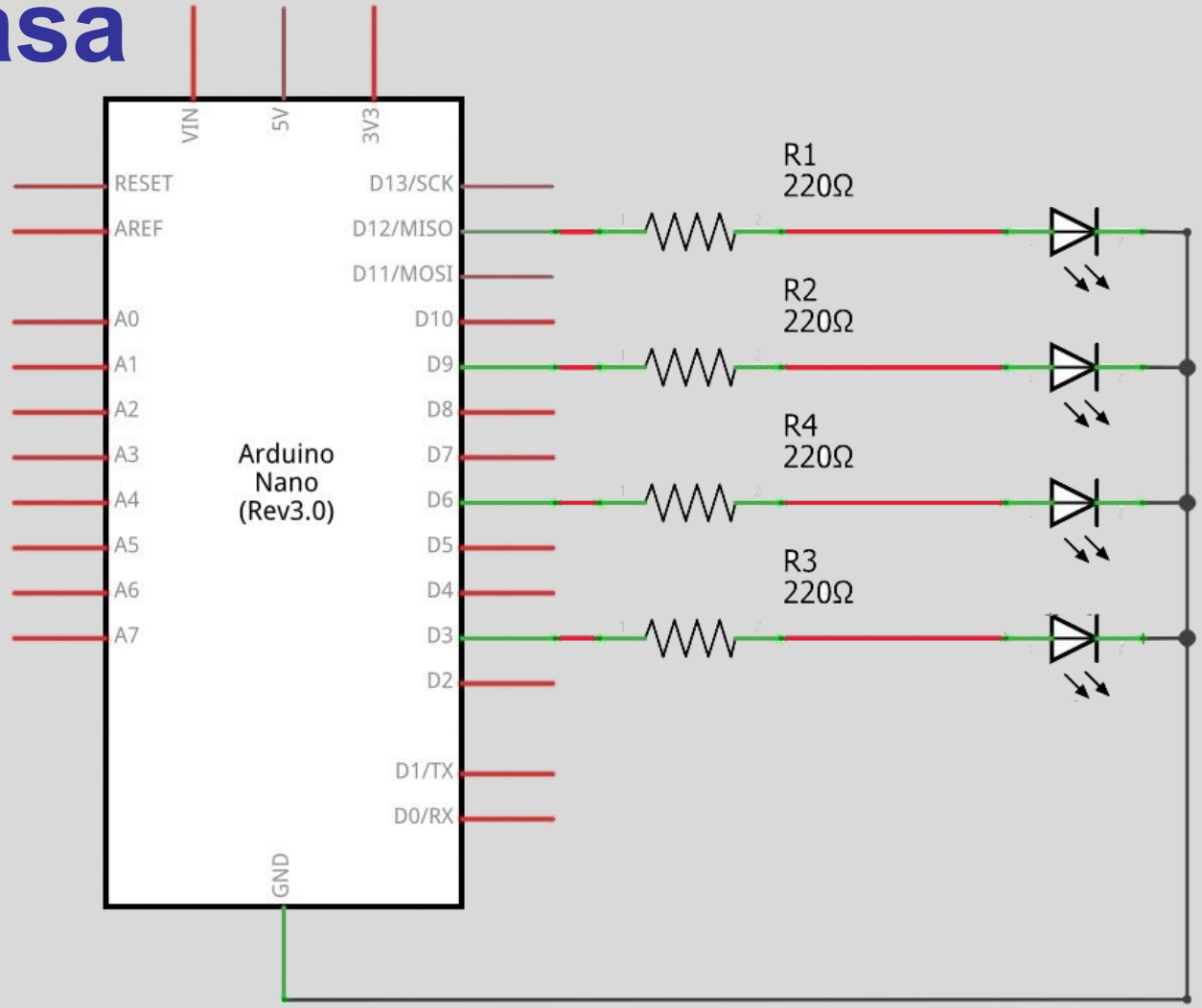
- A függvények létrehozása hasznos, ha az adott programrészletre többször is szükség van egy adott programban.
- Moduláris, kompaktabb lesz a program.
- Az átláthatóságot is javítja.

Függvény deklarálása:

```
visszatérési_érték_típus függvénynév( paraméter lista )
{
  // utasítások, helyi változók deklarálása
return kifejezés;
}
```

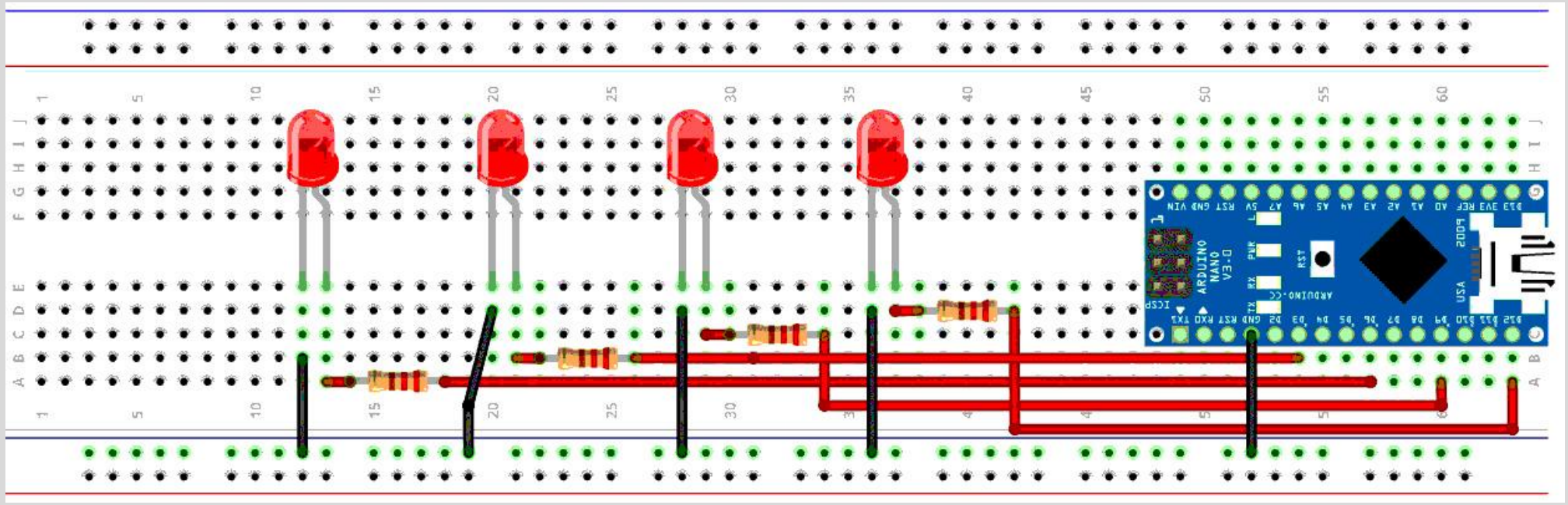
# Futófény programozása

- **Áramkör megtervezése:**
  - **4 db LED bekötése előtét ellenállással, 4 különböző digitális kimenetre**



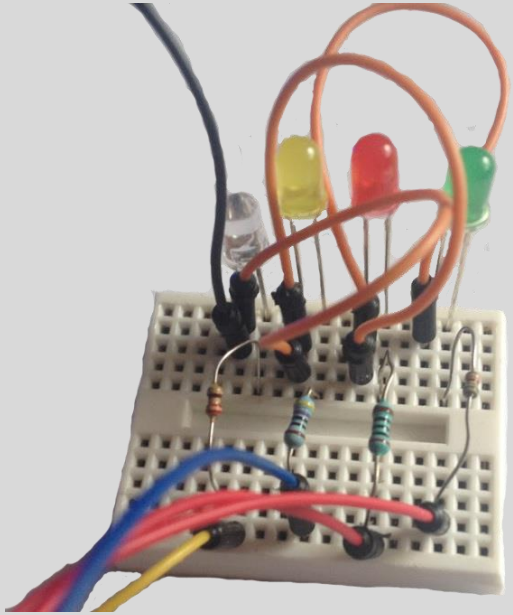
# Futófény programozása

- **Bekötési séma:**



# Futófény programozása

- **A megírt program:**



```
int led[]={4, 5, 6, 7};
void setup() {
  for(int i=0; i<4;i++)
  {
  pinMode(led[i], OUTPUT);
  }
}
void loop() {
  for(int j=0; j<4 ;j++)
    {
      digitalWrite(led[j], HIGH);
      if(j>0)
      digitalWrite(led[j-1], LOW);
      delay(200);
    }
  for(int g=4; g>0 ;g--)
    {
      digitalWrite(led[g], LOW);
      digitalWrite(led[g-1], HIGH);
      delay(200);
    }
}
```

// Pinek deklarálása

// Pinek kimenetként való definiálása

// 1. LED kivillan, majd elalszik, 2. LED…

// A deklarált pin-től kezdve tegye logikai 0-ba a kimenetet

// Visszafelé halad a futófény

# EEPROM írása és olvasása

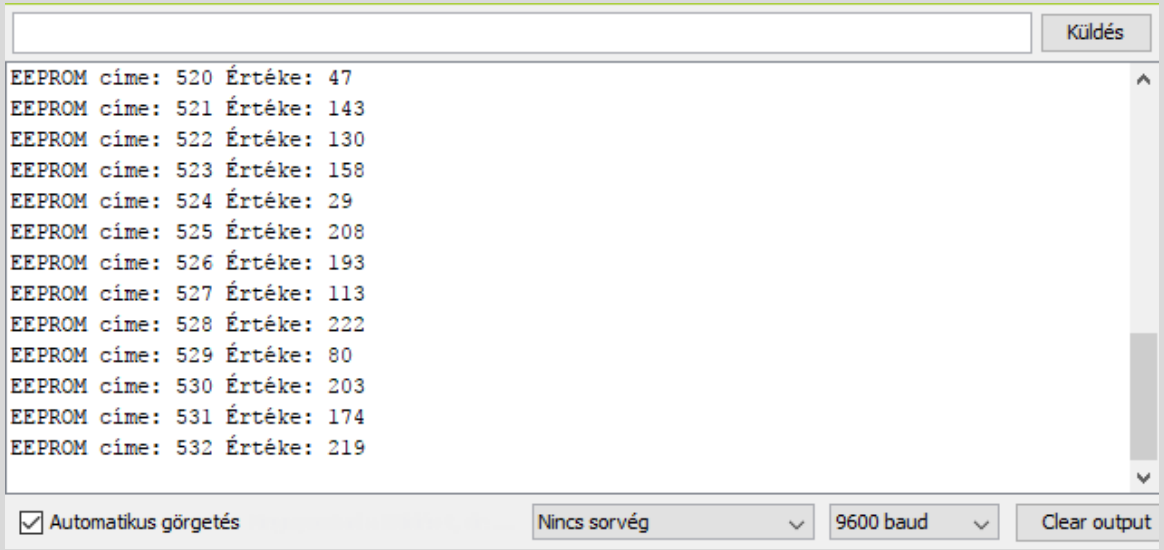
- Tároljunk el véletlen számokat, majd jelenítsük meg őket a soros monitoron.
- Rendelkezésre áll 1 KB méretű EEPROM bájtok írására és olvasására
  - 100000 írás és törlés
  - Header file használata: EEPROM.h
  - EEPROM.write(cím, adat); és az EEPROM.read(cím); függvények használata
  - Esetünkben a cím: 0 és 1023 közötti egészek
  - 1 bájton 0-255 közötti egészek tárolhatók

# Soros port használata

- **Port megnyitás: Serial.begin(baudrate);**
- **Érték kiíratás ASCII formátumban, kocsi vissza+új sorba: Serial.println(valtozo);**
- **Érték kiíratás ASCII formátumban, egymás után: Serial.print(valtozo);**

**Pár példa:**
**Serial.print("Cím"); → Cím kiíratás**
**Serial.print(125, BIN) → „1111101"**
**Serial.print(125, OCT) → „175"**
**Serial.print(125, HEX) → „7D"**
**Serial.println(1.23456, 0) → "1"**
**Serial.println(1.23456, 2) → "1.23"**
**Serial.println(1.23456, 4) → "1.2346"**
**Serial.print("\t"); → Tabulátor**
**Serial.write(64); → @ (bájt küldéséhez)**

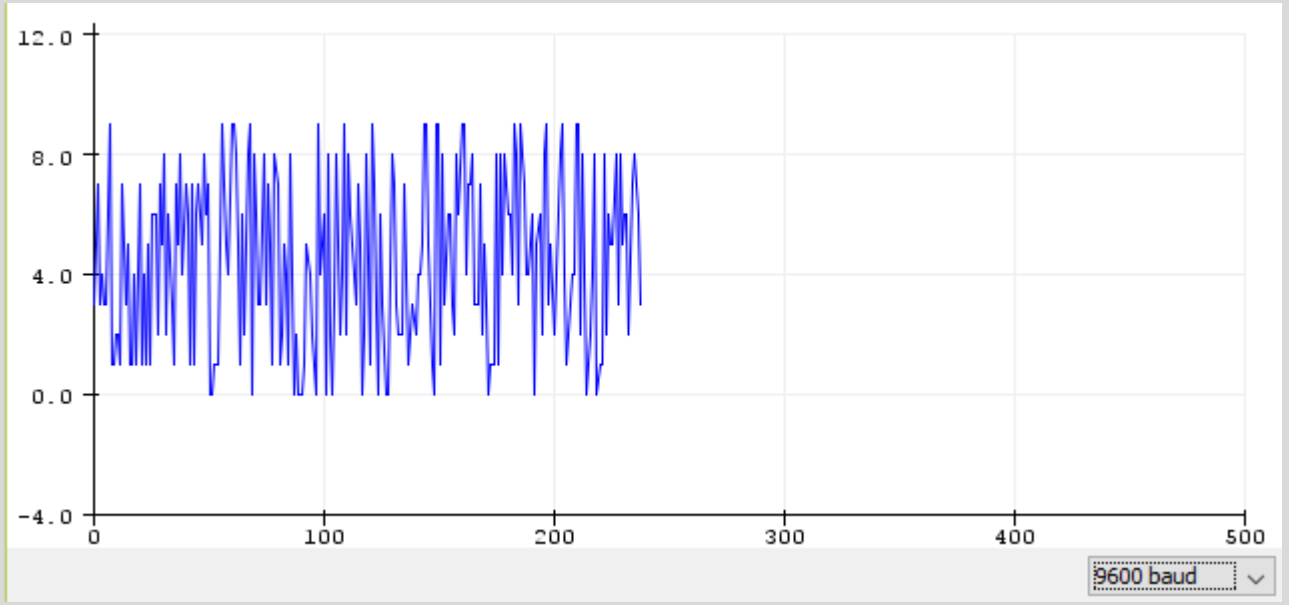
- **Soros monitor:**



# A soros plotter használata

**Véletlen számok generálása 1-10-ig:**

```
int veletlen;
void setup()
{
  Serial.begin(9600);
  // Random bemenet megadása a véletlen számgenerátor inicializálásához
  randomSeed(analogRead(2));
}
void loop()
{
    // 0-10 közötti véletlen számok,
    // random(minimum, maximum); vagy random(maximum);
    veletlen=random(10);
    Serial.println(veletlen);
    delay(50);
}
```



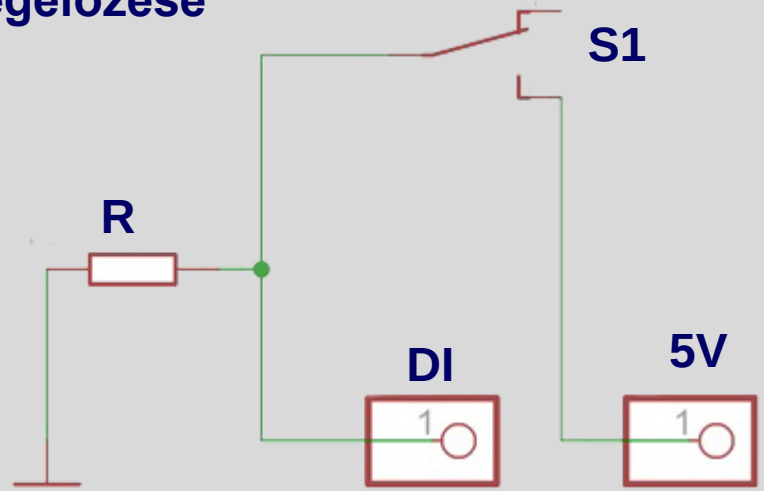
# EEPROM írása és olvasása

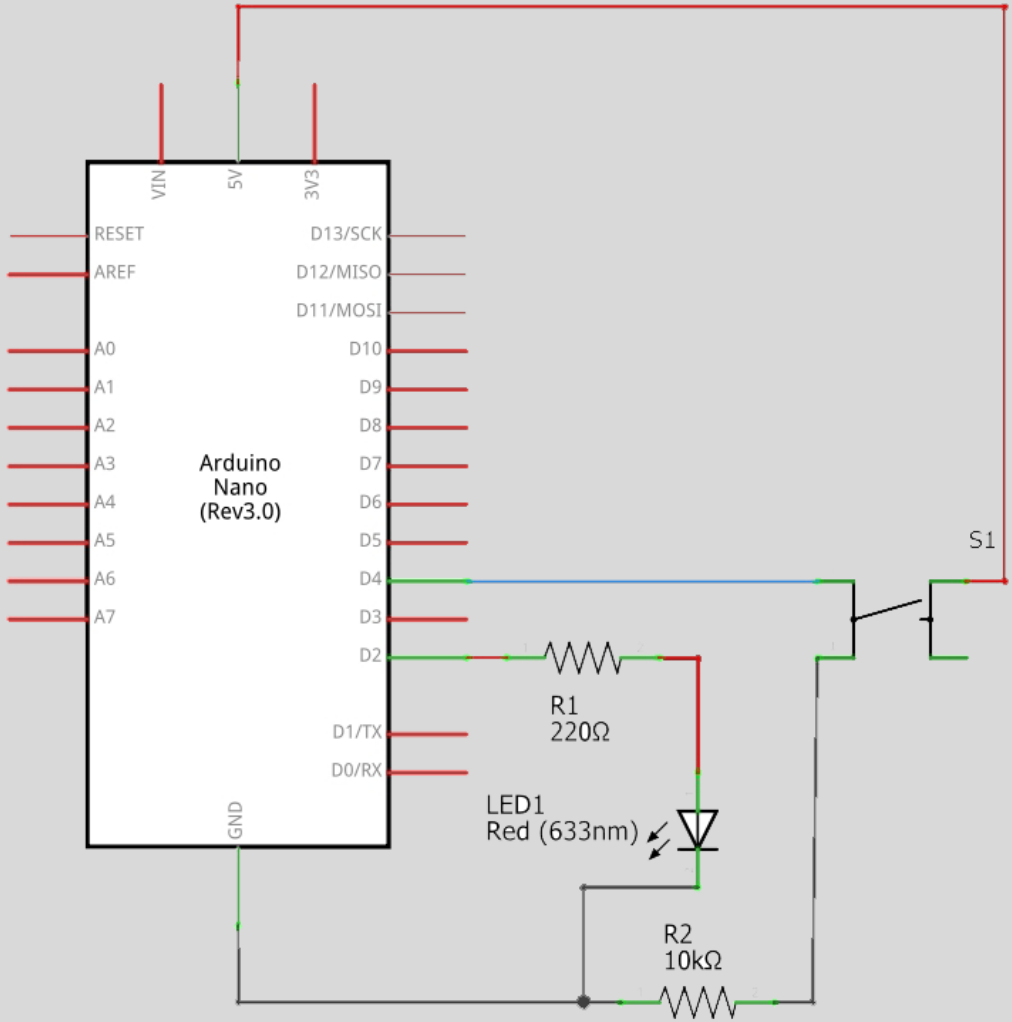
```
#include <EEPROM.h>
int veletlen;

void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(2));
}
void loop()
{
  Serial.println("Random számok írása és olvasása");

  for (int i = 0; i < EEPROM.length(); i++)
  {
    veletlen=random(255);
    EEPROM.write(i, veletlen);
  }
  Serial.println();
  for (int j=0; j< EEPROM.length(); j++)
  {
    veletlen = EEPROM.read(j);
    Serial.print("EEPROM címe: ");
    Serial.print(j);
    Serial.print(" Értéke: ");
    Serial.println(veletlen);
    delay(100);
  }
}
```

- Az EEPROM.length() helyett használható: int hossz=1024, mivel jelen esetben ismert ez EEPROM mérete

# Nyomógombbal működtetett LED

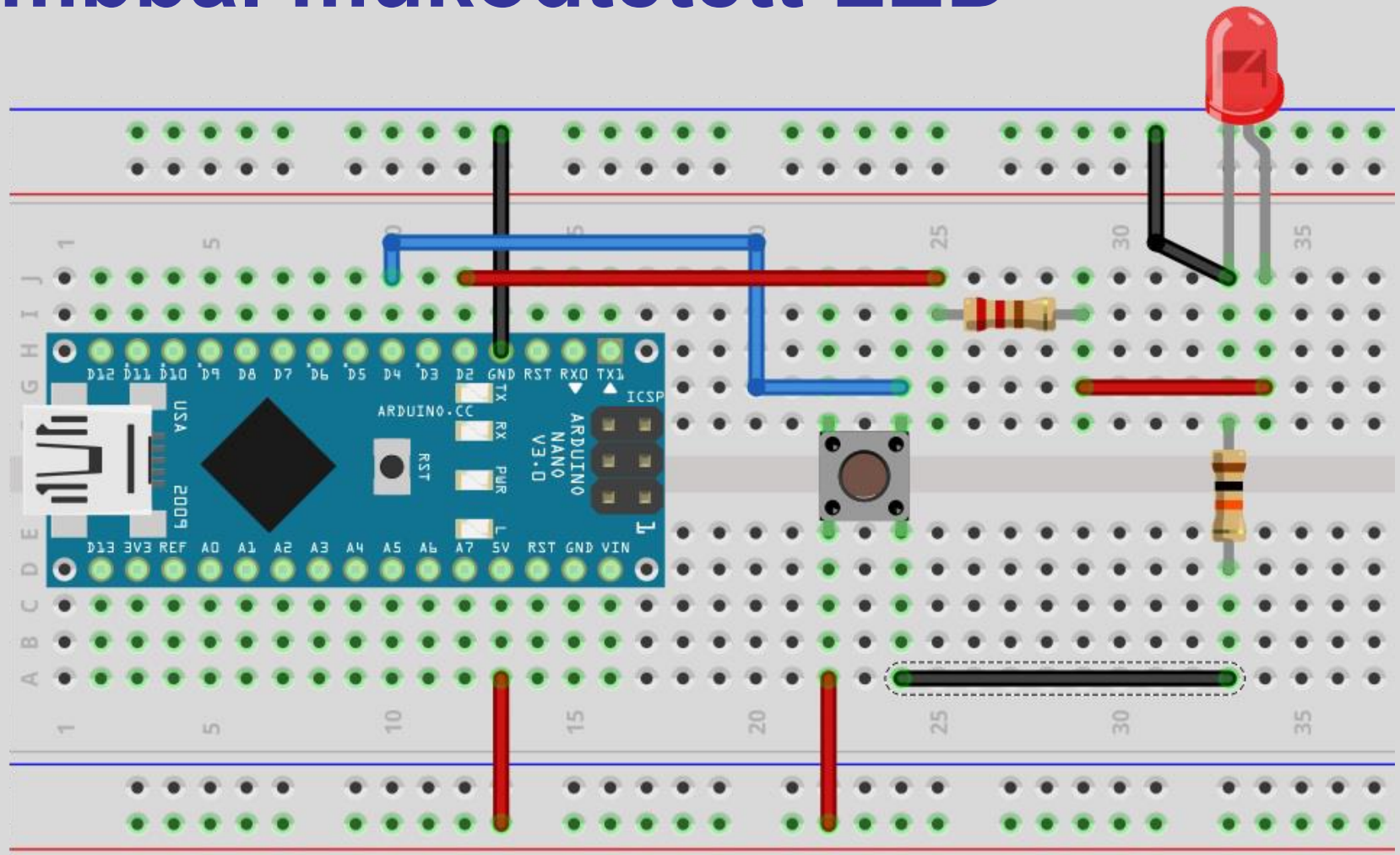
- **Áramkör megtervezése:**
  - 1 db. LED bekötése
  - 1 db nyomógomb bekötése
- **Lehúzó ellenállás (Pull Down Resistor)**
  - Alaphelyzetben a bementként konfigurált láb logikai 0 lesz
  - Bizonytalan állapot megelőzése



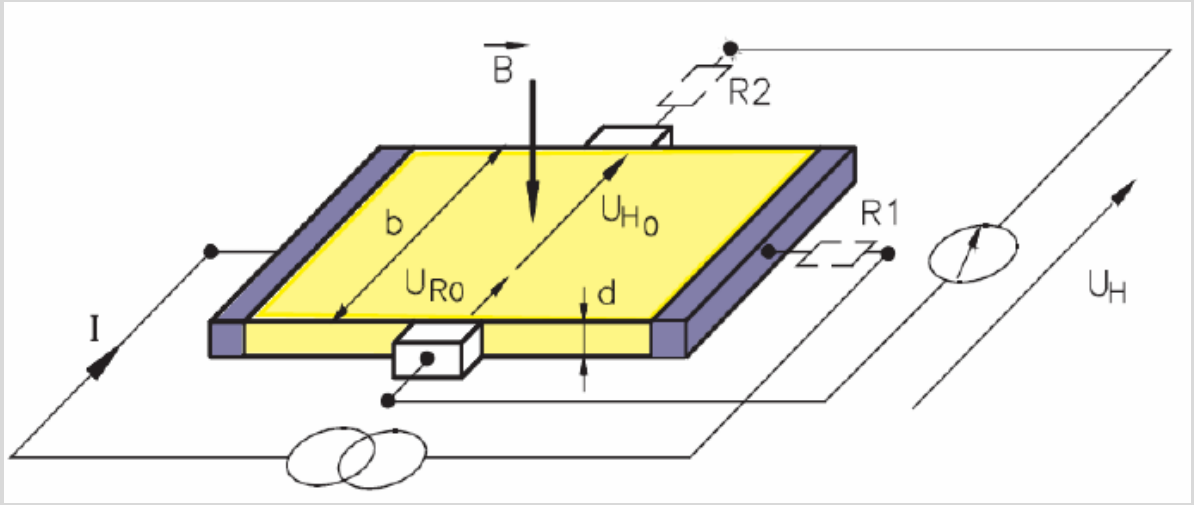



# Nyomógombbal működtetett LED

- **Bekötési séma:**

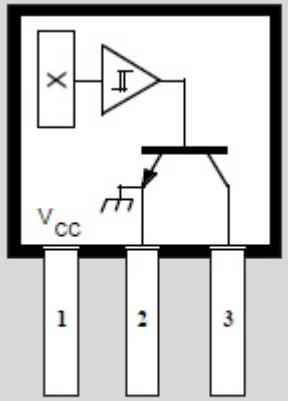


# Hall érzékelő




- **Áramkör megtervezése:**
  - **1 db. LED bekötése**
  - **1 db. Hall érzékelő bekötése**
- **I áram folyik át egy félvezetőn, amelyet mágneses térbe helyezve, az elektronokra a Lorentz-erő fog hatni, így megjelenik a hasáb két oldalán az $U_H$ Hall feszültség. A mágneses indukció vonalak merőlegesek a félvezető lapkára. A feszültség akkora, hogy a töltéshordozókra ható Lorentz erőt kompenzálja.**

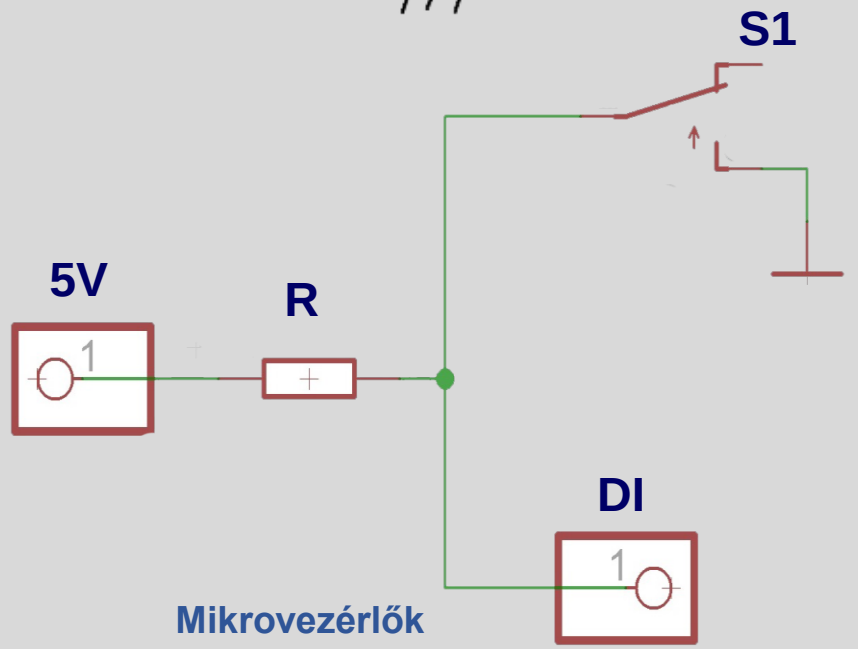
# Hall érzékelő







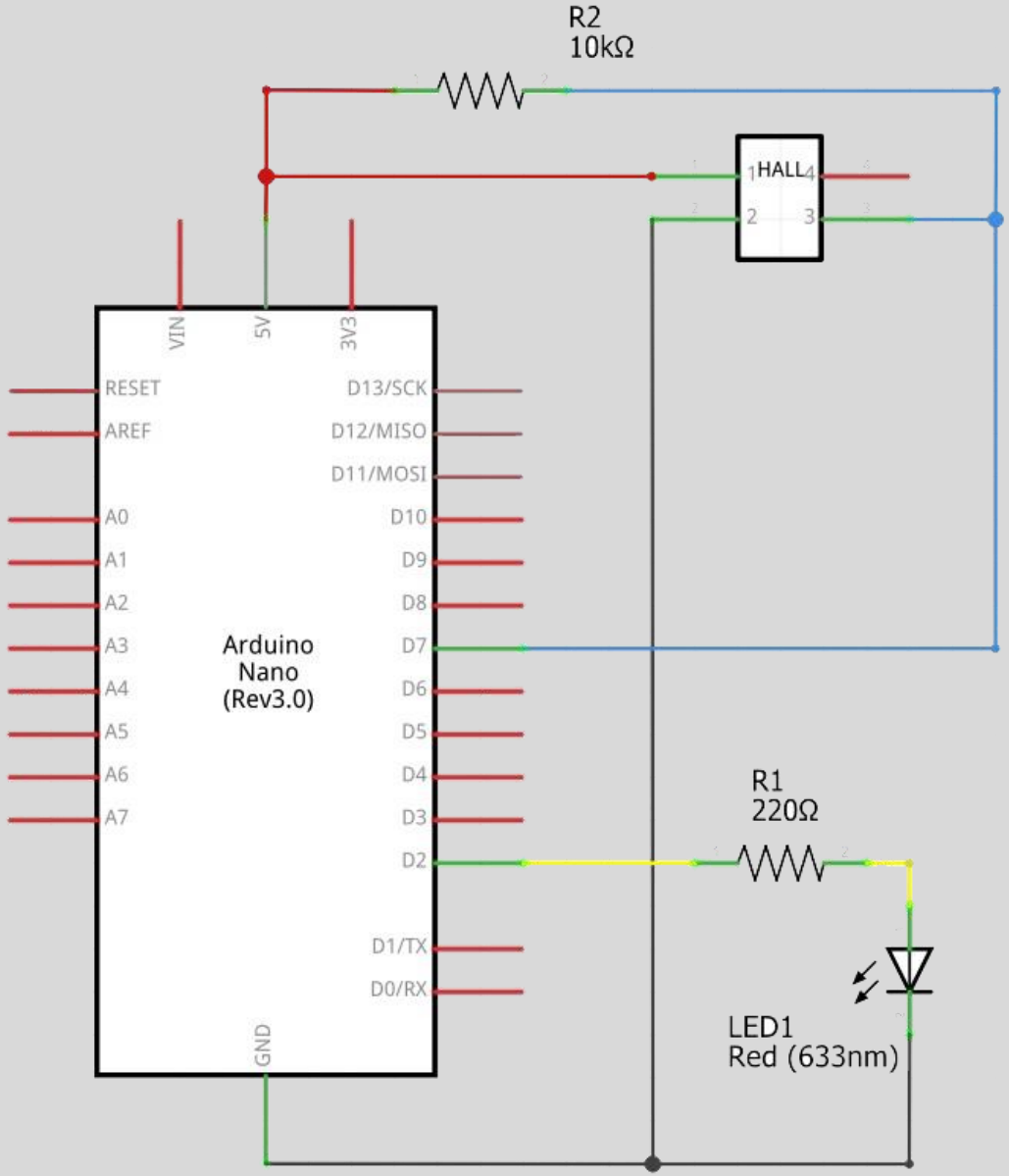
- **Érzékelő adatai:**
  - Nyitott kollektoros
  - Üzemi feszültségszint: 4,5 – 24 V
  - Típuskód: A3144
  - Tartalmazza: Feszültség regulátor, védődióda, négyszög jelű Hall feszültség generátor hőmérsékletkompenzáló áramkör, jelerősítő, Schmitt trigger
  - Schmitt trigger (bistabil billenőkör): küszöbérték kapcsoló, hiszterézis tapasztalható, amplitúdó uniformizálásra alkalmazzák. A kimeneti jele a bemenő jel amplitúdójának nagyságától függ.




- **Felhúzó ellenállás (Pull Up Resistor)**
  - Alaphelyzetben a bementként konfigurált láb logikai 1 lesz
  - Bizonytalan állapot megelőzése

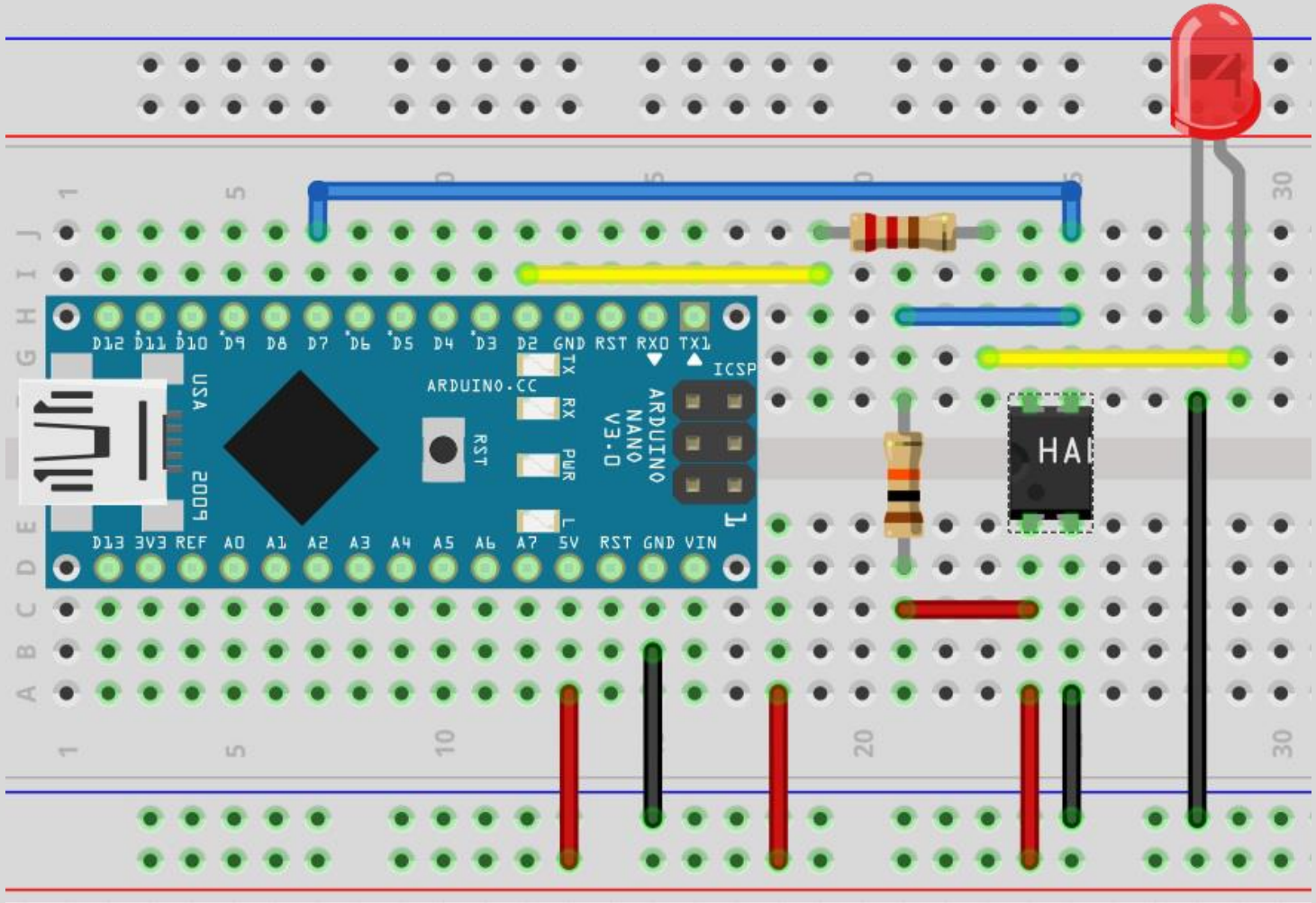
# Hall érzékelő

- **A kapcsolási rajz:**



# Hall érzékelő

- **A bekötési séma:**



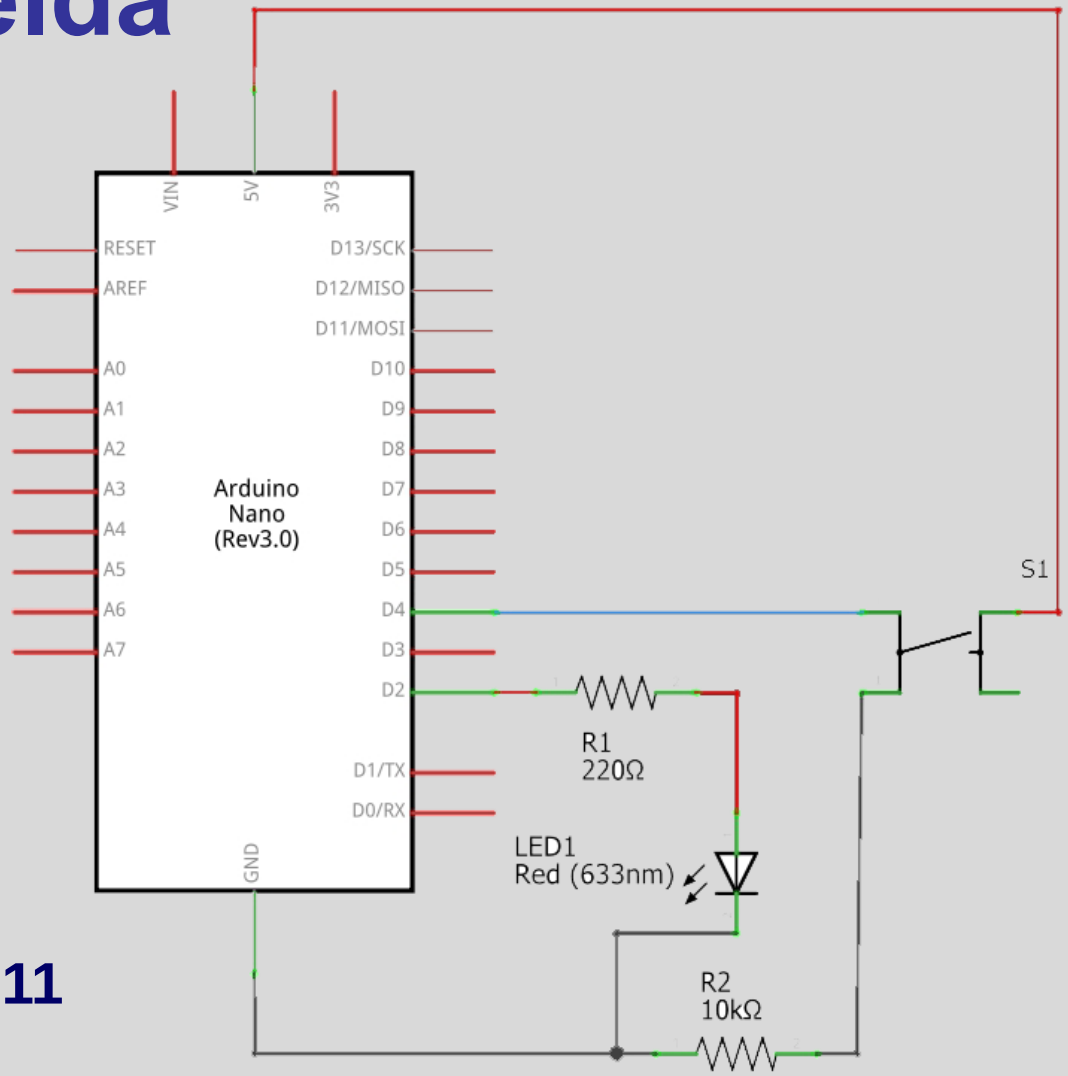
# Hall érzékelő

- **A megírt program:**

```
int hall_jel = 7;                                   // Valtozok deklaralasa
int led =  2;
int allapot = 0;
void setup() {                                      // Pin-ek konfigurálása
  pinMode(hall_jel, INPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600); }                             // Baud ráta beállítása
void loop(){
  allapot = digitalRead(hall_jel);                  // Hall szenzor állapotának olvasása
 if (hall_jel == LOW)                               // Ha nem érzékel, a LED ne világítson
 {
   digitalWrite(led, HIGH);
   Serial.println("Erzekel");                       // Soros porton küldje ki az állapotot
 }
 else {                                             // Ha érzékel, a LED világítson
   digitalWrite(led, LOW);
   Serial.println("Nem Erzekel"); }}                // Soros porton küldje ki az állapotot
```

# Megszakításkezelésre példa

- A kapcsolási rajz:
- Szükséges eszközök:
  - Lehúzó ellenállás
  - LED, előtét ellenállással
  - Nyomógomb



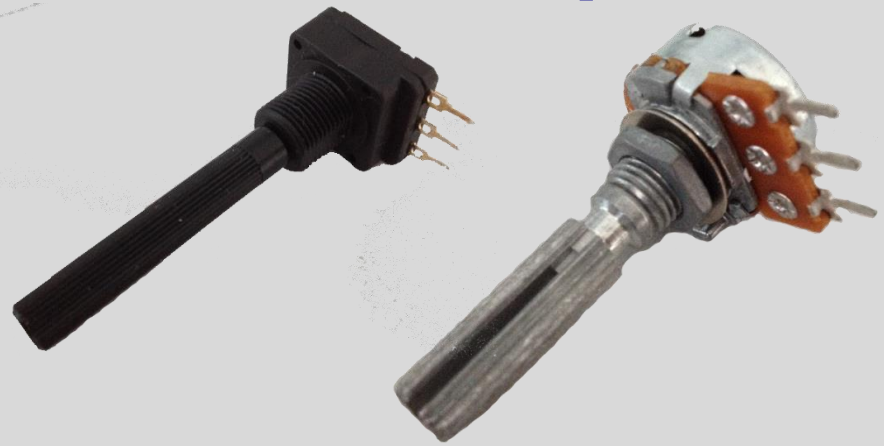

- Emlékeztető: PWM lehetőség: D3, D5, D6, D9-11

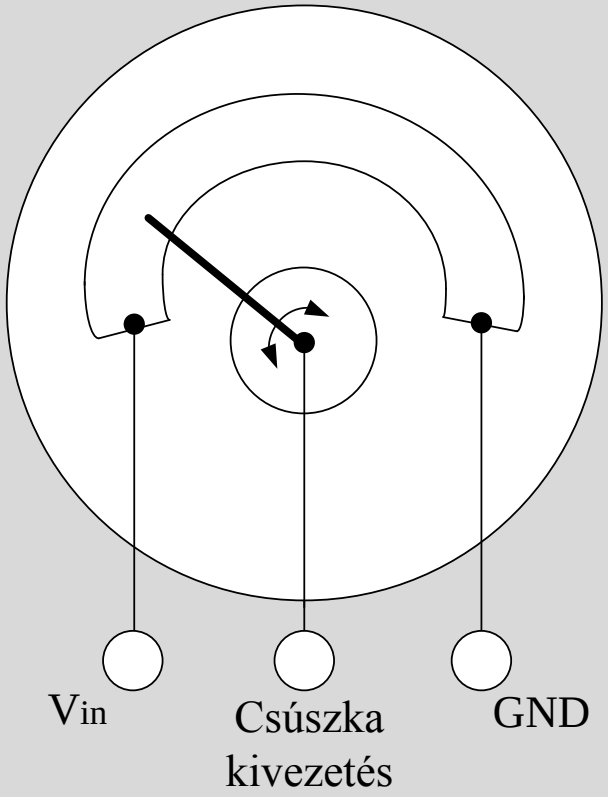
# Megszakításkezelésre példa

```
const int megszakitas_pin = 2;  //nyomogombot ide kotjuk
const int led =  5;
volatile int gomb_allapot = 0;
void setup()
{
   pinMode(megszakitas_pin, INPUT);
   pinMode(led, OUTPUT);
   attachInterrupt(digitalPinToInterrupt(megszakitas_pin), megszakitas_fgv, CHANGE);
 }
void loop()
{
}
void megszakitas_fgv()
{
  gomb_allapot = digitalRead(megszakitas_pin);
  digitalWrite(led, gomb_allapot);
}
```

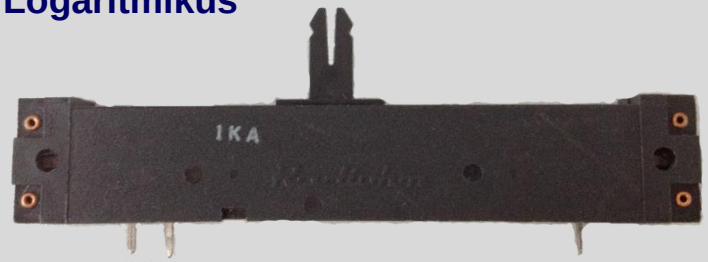
# LED fényerejének változatása potenciométerrel

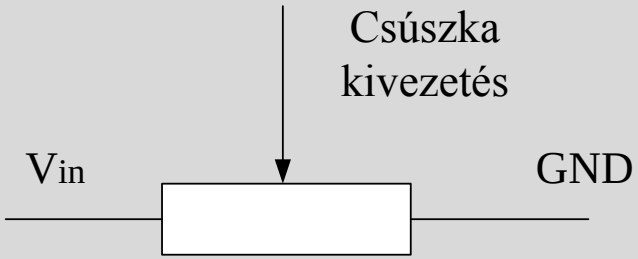
- **Áramkör megtervezése:**
  - 1 db. LED + ellenállás
  - 1 db. Potenciométer





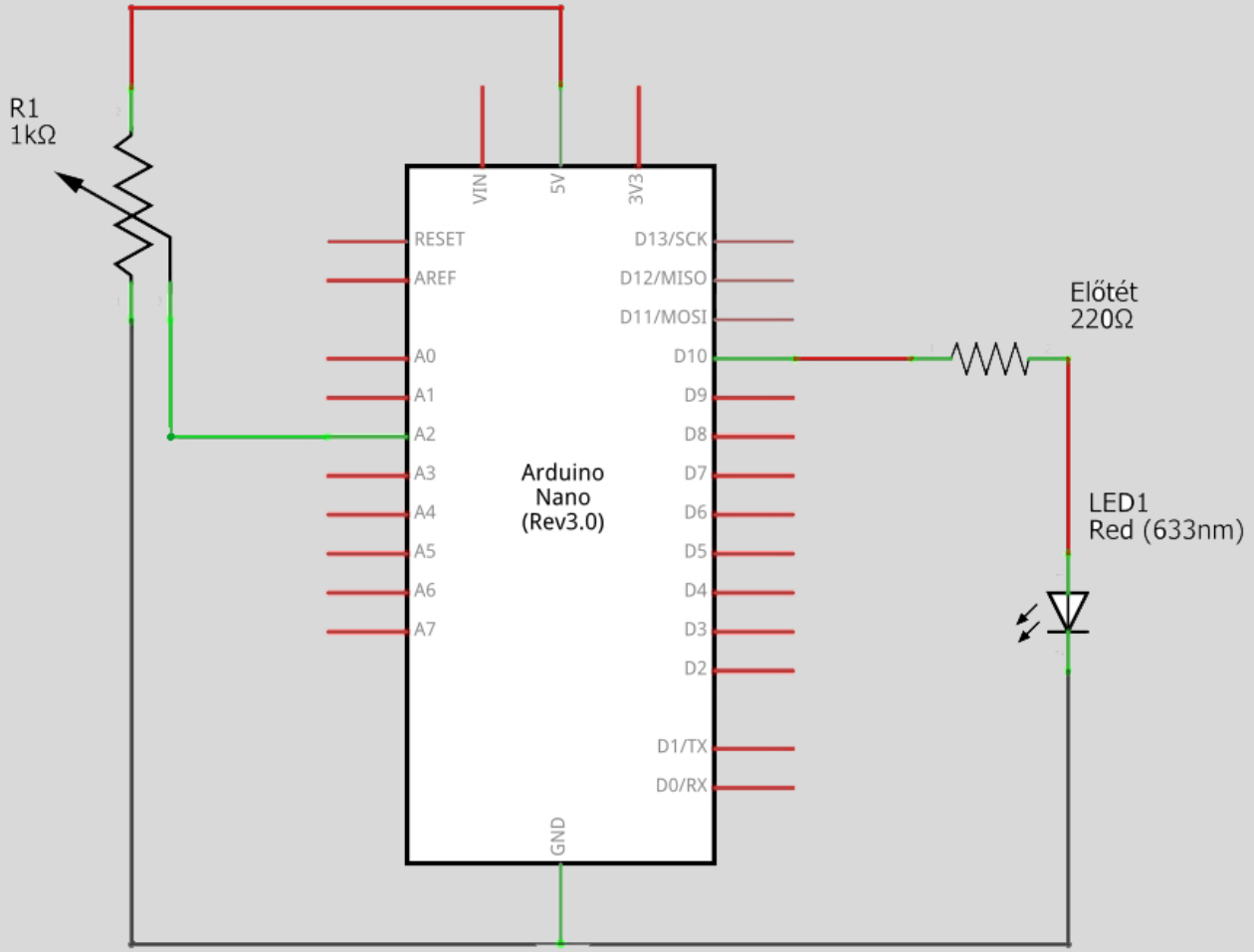

- **Potenciométer: Változatható értékű ellenállás (Osztásarányt a csúszkával állítjuk)**
  - Lineáris
  - Logaritmikus



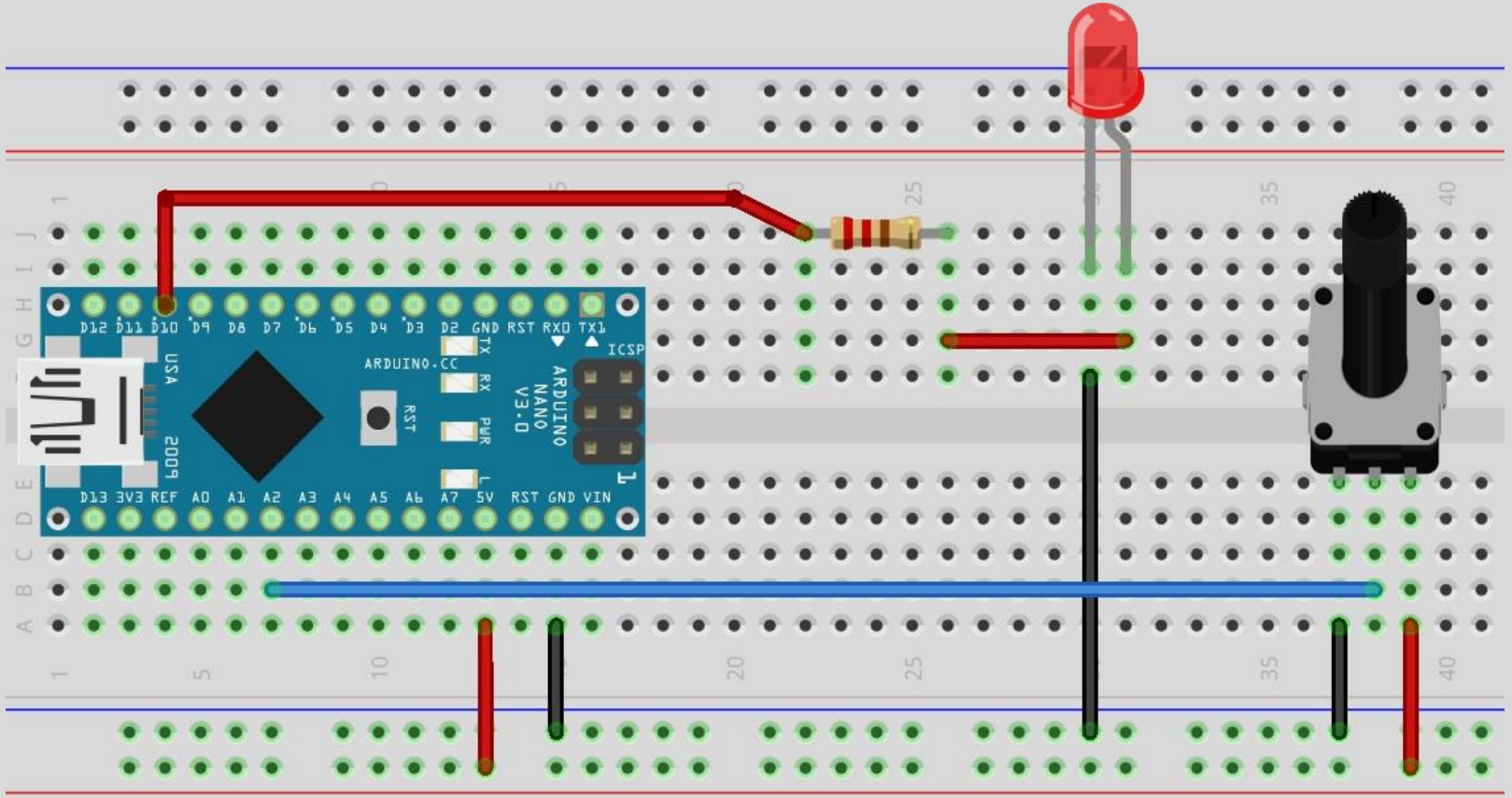
# LED fényerejének változatása potenciométerrel

- **A kapcsolási rajz:**
  - **A0-A7 analóg lábak valamelyikére köthető a potenciométer csúszka kivezetése**



# LED fényerejének változatása potenciométerrel

- **Bekötési séma:**



# LED fényerejének változatása potenciométerrel

- **Skálázás:**
  - A beolvasott analóg jel A/D átalakítás után 0-1024 közötti értékeket adhat, ezt az adott feladathoz mérten egy adott intervallumra be kell állítani
  - PWM-et használunk a LED fényerejének változtatásához → Kitöltési tényező 0-255 közötti értékű lehet
- **Két lehetőség 8 bitre történő skálázásra:**
  - map(); függvény használata
  - Gyalog módszer

```
Be_max = 1023
Be_min = 0
Ki_min = 0
Ki_max = 255
```

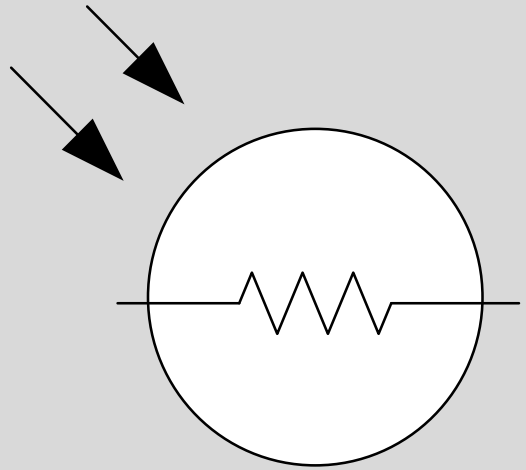
```
long map(long bejovo, long be_min, long be_max, long ki_min, long ki_max)
{
 return (bejovo - be_min) * (ki_max - ki_min) / (be_max - be_min) + ki_min;
}
```

# LED fényerejének változatása potenciométerrel

- **A program:**

```
int pot_bemenet = 2;                    // Csúszka kivezetés
int led = 10;                           // LED
int bejovo = 0;                         // Beolvasandó érték

void setup()
{
  pinMode(led, OUTPUT);
}
void loop()
{
   bejovo = analogRead(pot_bemenet);    //Analóg érték beolvasása
   bejovo = map(bejovo, 0, 1023, 0, 255); // Skálázás
   analogWrite(led, bejovo);            // ISZM alkalmazása
}
```

# Alkonykapcsoló

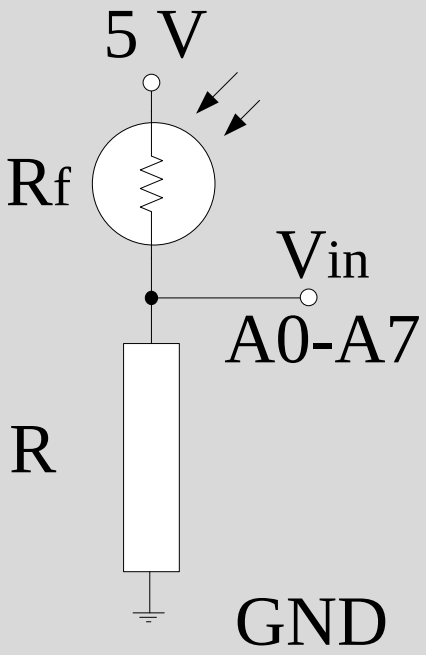


- **Áramkör megtervezése:**
  - 1 db. LED
  - 1 db. Fotoellenállás
  - Ellenállások

- **Fotoellenállás (LDR): Fény hatására az ellenállás értéke változik**
  - Fényérzékeny rétegek kialakítása: ólom-szulfid, kadmium-szulfid, stb.
  - A megvilágítás mértéke ha csökken, akkor az ellenállás értéke megnő (a fény gerjeszti a töltéshordozókat).
  - Spektrális érzékenység: meghatározott fény hullámhossznál a legérzékenyebb
  - Erősen hőmérsékletfüggő
  - Lassú reagálás (10 msec nagyságrendű)
  - Sötét- és világos ellenállás megadása

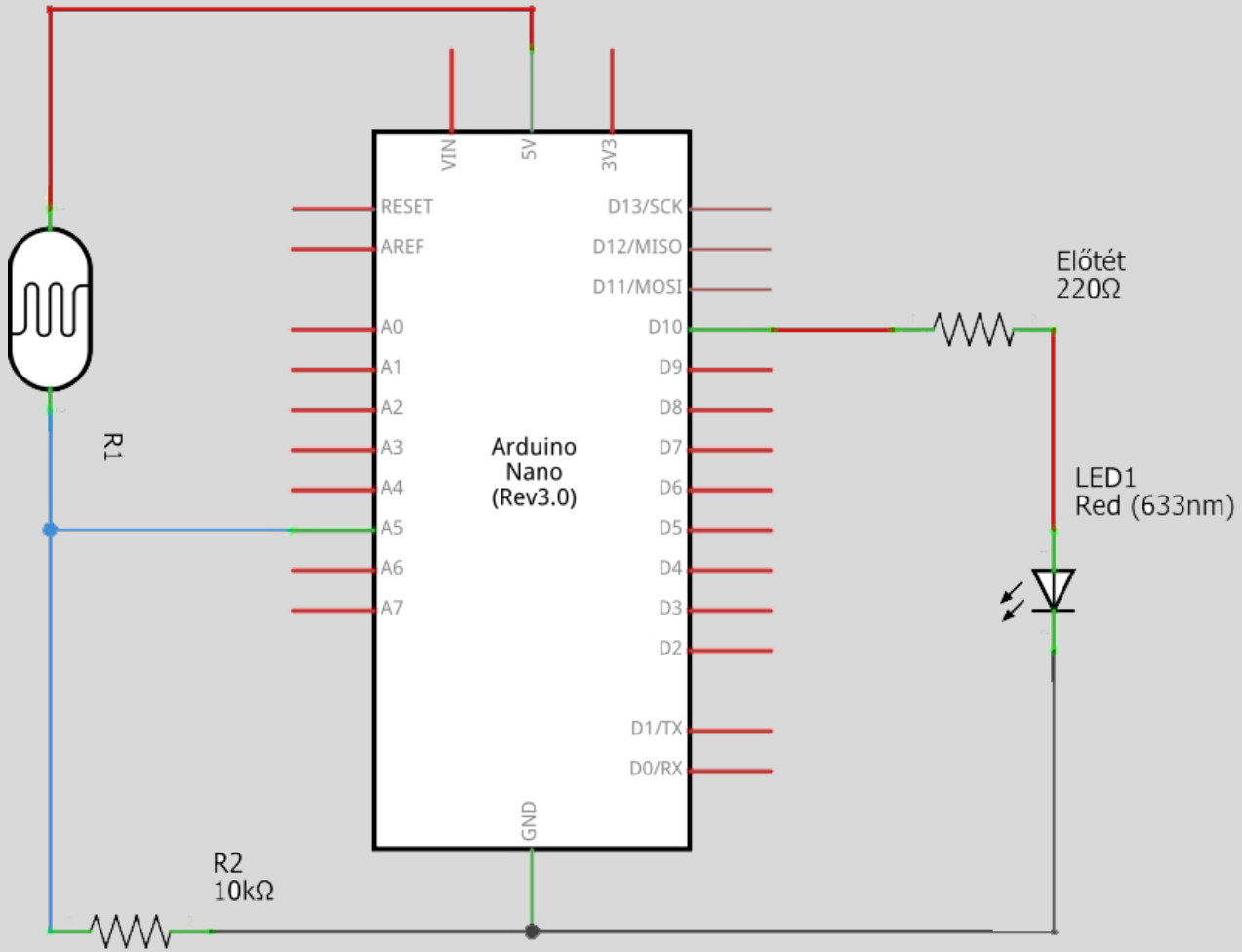
# Alkonykapcsoló

- **Feszültségosztó alkalmazása**
  - **Fotoellenállás értékének átalakítása feszültség változássá**
  - **Referencia pont**



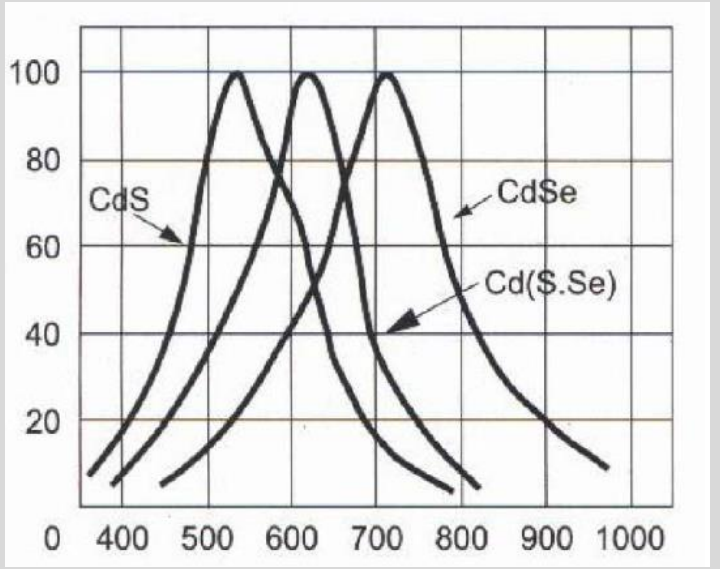

$$V_{in} = U_{be}\frac{R}{R + R_f}$$

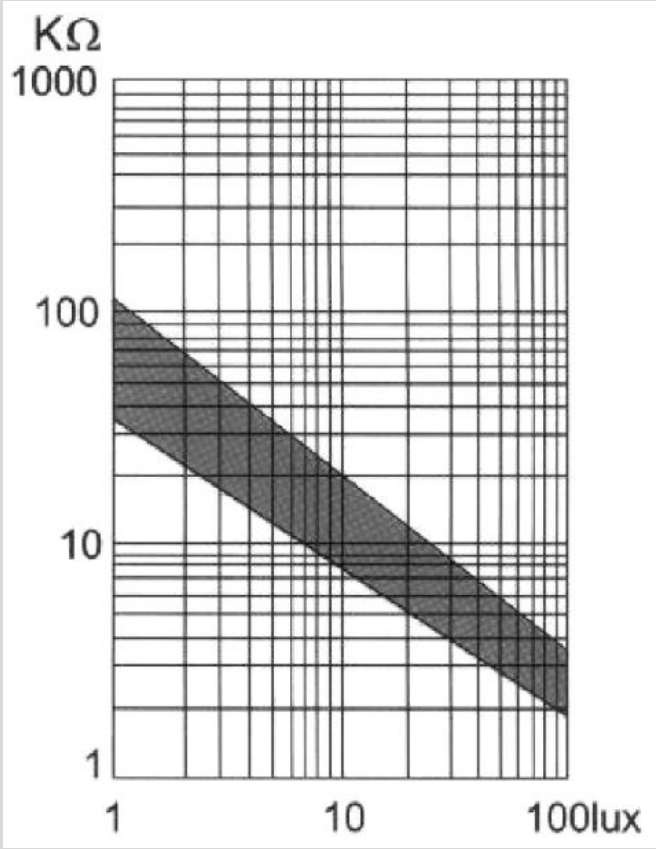
$$R = \sqrt{R_{vil} \cdot R_{söt}}$$



# Fotoellenállás

- **GL5528 típusú fotoellenállás adatai:**
  - $R_{vil.} = 8 - 20\ k\Omega$ $10\ Lux\ esetén$
  - $R_{söt.} = 1\ M\Omega$ $0\ Lux\ esetén$
  - **Maximális feszültség:** $U_{max} = 150\ V$
  - **Anyaga: Kadmium szulfid**
  - **Spektrális érzékenységi csúcs:** $540\ nm$
  - **Működési hőmérséklet tartomány:** $-30\ -\ +70\ °C$
  - **100 Lux-nál** $R_f = 2\ k\Omega$ →



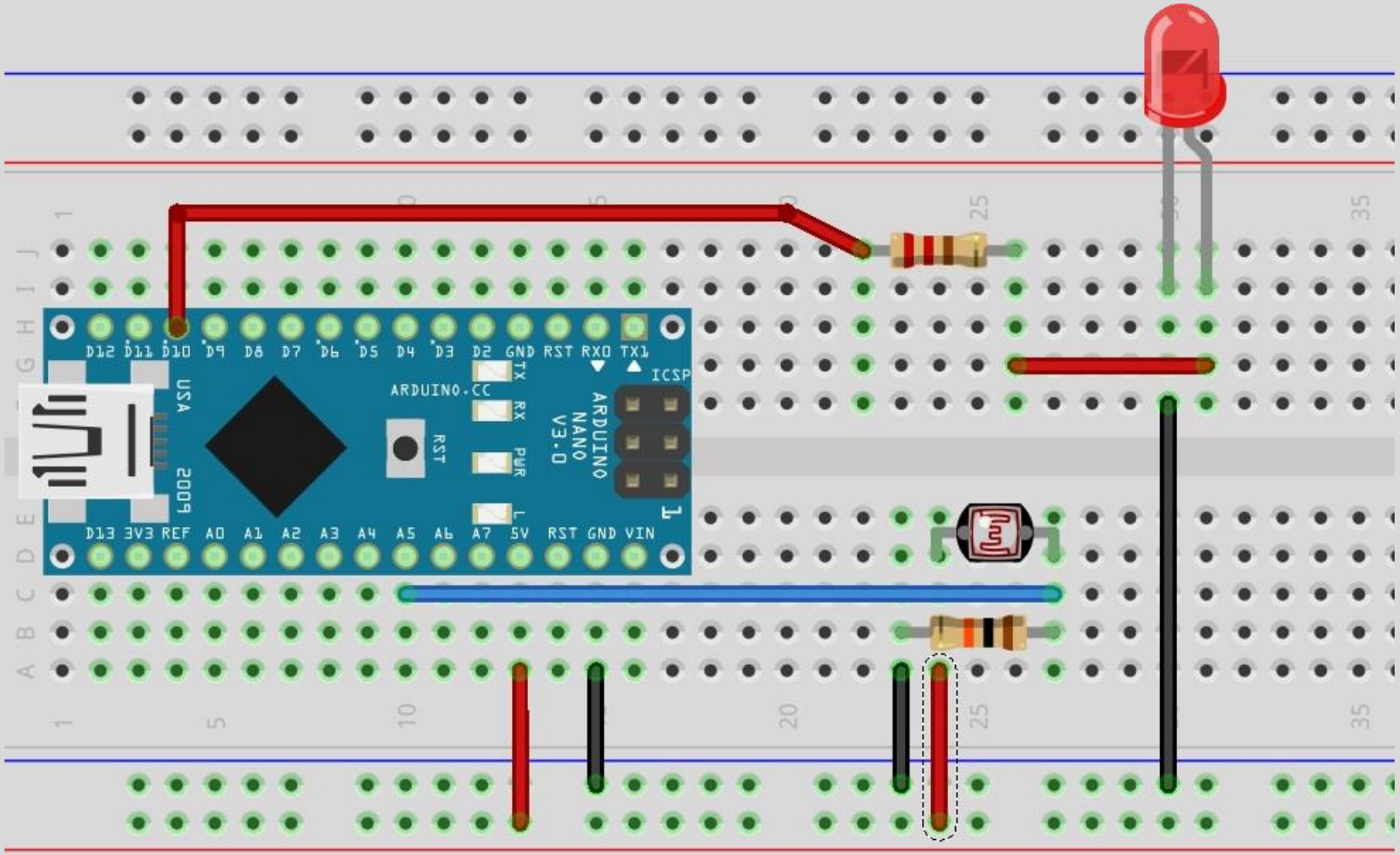




$$V_{in} = 5\ V \frac{10000\ \Omega}{10000\ \Omega + 2000\ \Omega} = 4{,}167\ V,$$

**ez a mikrovezérlőn 852 − 853 kvantált értéket jelent majd**

# Alkonykapcsoló

- **Bekötési séma:**



# Alkonykapcsoló

- A program:

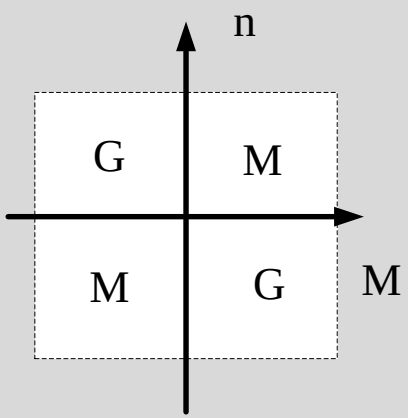
```
int fotor_bemenet = 2;                    // Változók deklarálása
int led = 10;
int bejovo =0;

void setup()
{
  pinMode(led, OUTPUT);
}
void loop()
{
   bejovo = analogRead(fotor_bemenet);
   if (bejovo <= 600)                     // if else kétirányú elágazás
   digitalWrite(led, HIGH);
   else
   digitalWrite(led, LOW);
}
```

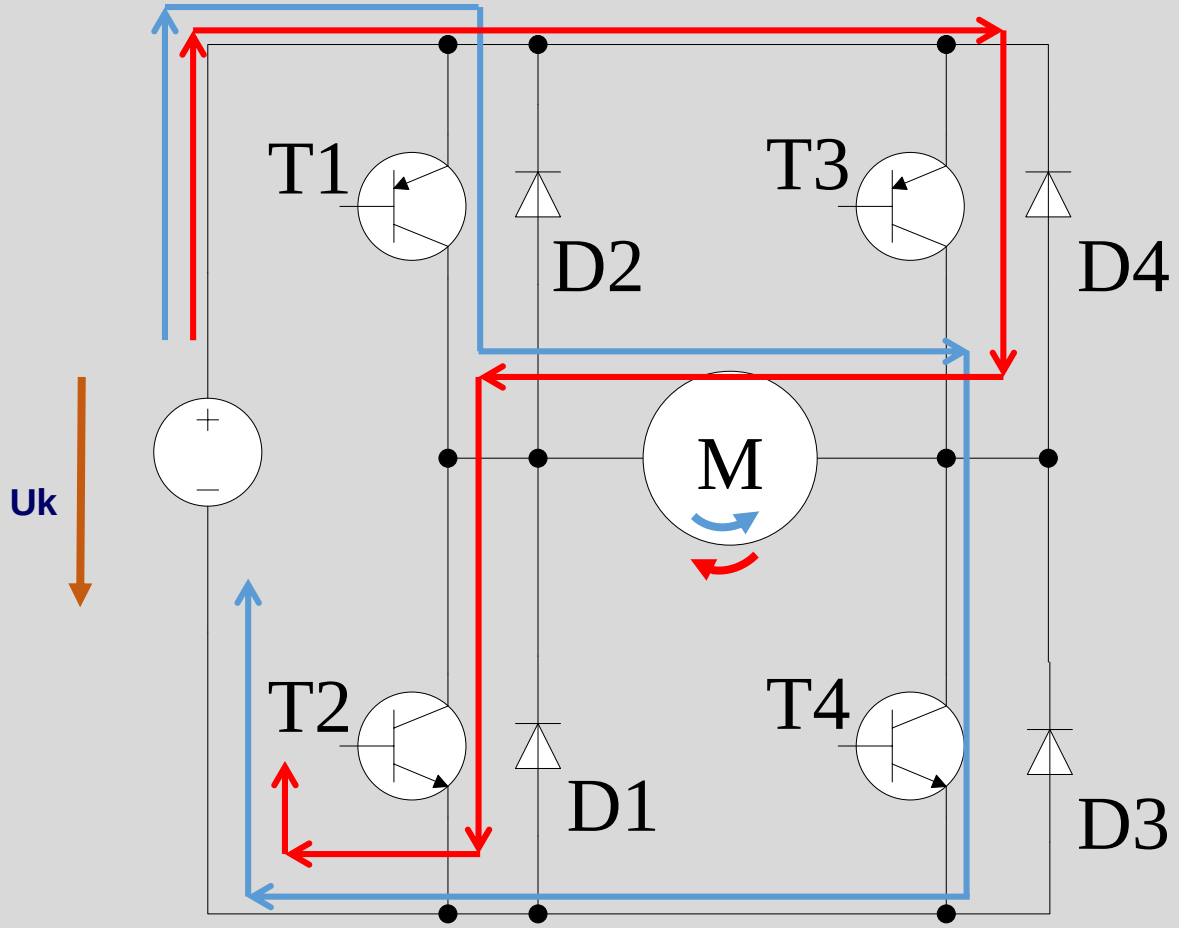
# DC motor fordulatszámának szabályozása

- **Áramkör megtervezése:**
  - **1 db. DC motor**
  - **1 db. H-híd**
  - **1 db. Potenciométer**
- **A motorok áramfelvétele nagy, a mikrovezérlő 40 mA áramot képes leadni.**
  - **Külső tápegységet kell használni**
  - **H-hidas motorvezérlő IC-t fogunk bekötni**

# DC motor fordulatszámának szabályozása

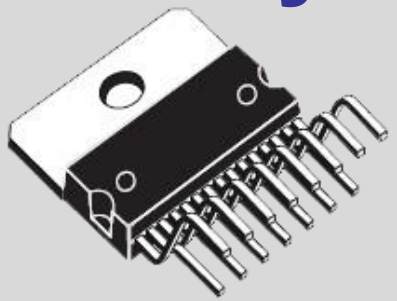
- **H-híd:**
  - **T1 és T4 vezet**
  - **T3 és T2 vezet**
  - **Védődiódák: Induktív terhelések esetére.**
  - **Félvezető lehet: Tranzisztor, MOSFET, stb.**
  - **Azonos oszlopban levőket összekapcsolva rövidzár!**
    - **A készen kapható H-hidakban ez ki van küszöbölve**



# DC motor fordulatszámának szabályozása

- **Az általunk használt H-híd:**
  - Védődiódák nem IC-n belül, hanem a panelen vannak rajta.
- **L298N motor meghajtó IC** → →

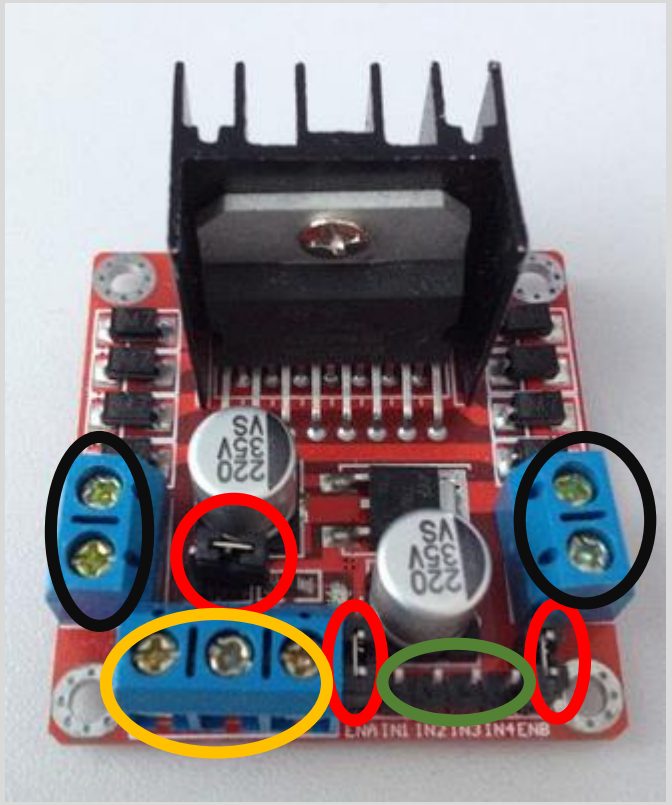


Multiwatt15

  - Engedélyező feszültségszint: max.: 7 V
  - Külső tápforrás feszültségszint: 5 – 35 V
  - Logikai áram nagysága: max. 36 mA
    Meghajtás maximális árama: 2 A
  - Storage temperature: -25 – +130 °C
  - Maximális teljesítmény: 25 W
  - Az elektronika mérete: 43 x 43 x 27mm

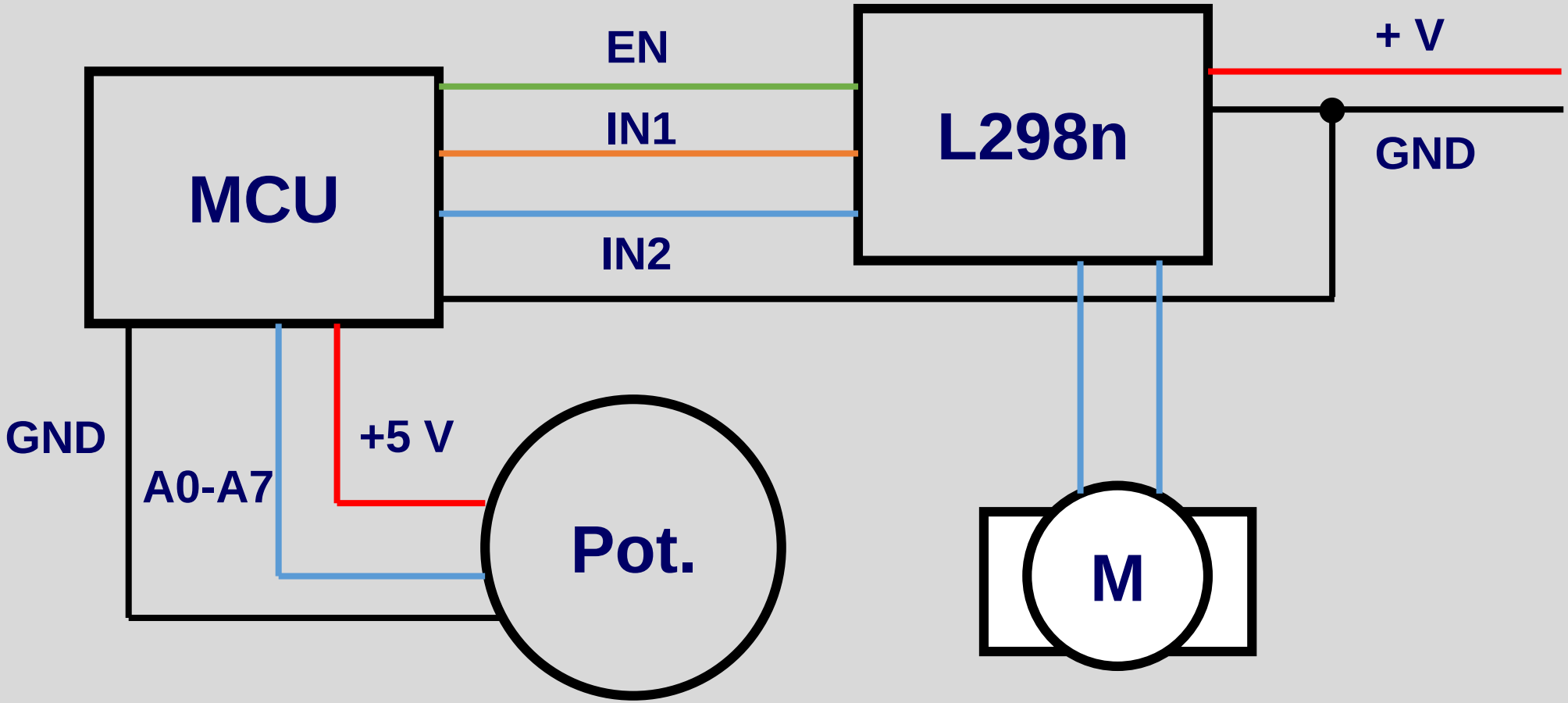
# DC motor fordulatszámának szabályozása

- **H-híd panel lábkiosztása:**
  - **3 db. Jumper: Külső tápforrásból 5 V előállítása, motor engedélyező lábak használata (PWM-re ezeket használjuk)**
  - **Az in1 és in2 az első motort-, az in3 és in4 a második motort vezérli**
  - **A külső tápegység sorkapcsa és az 5 V levétele pl. az Arduino számára**
  - **A két motor sorkapcsa**



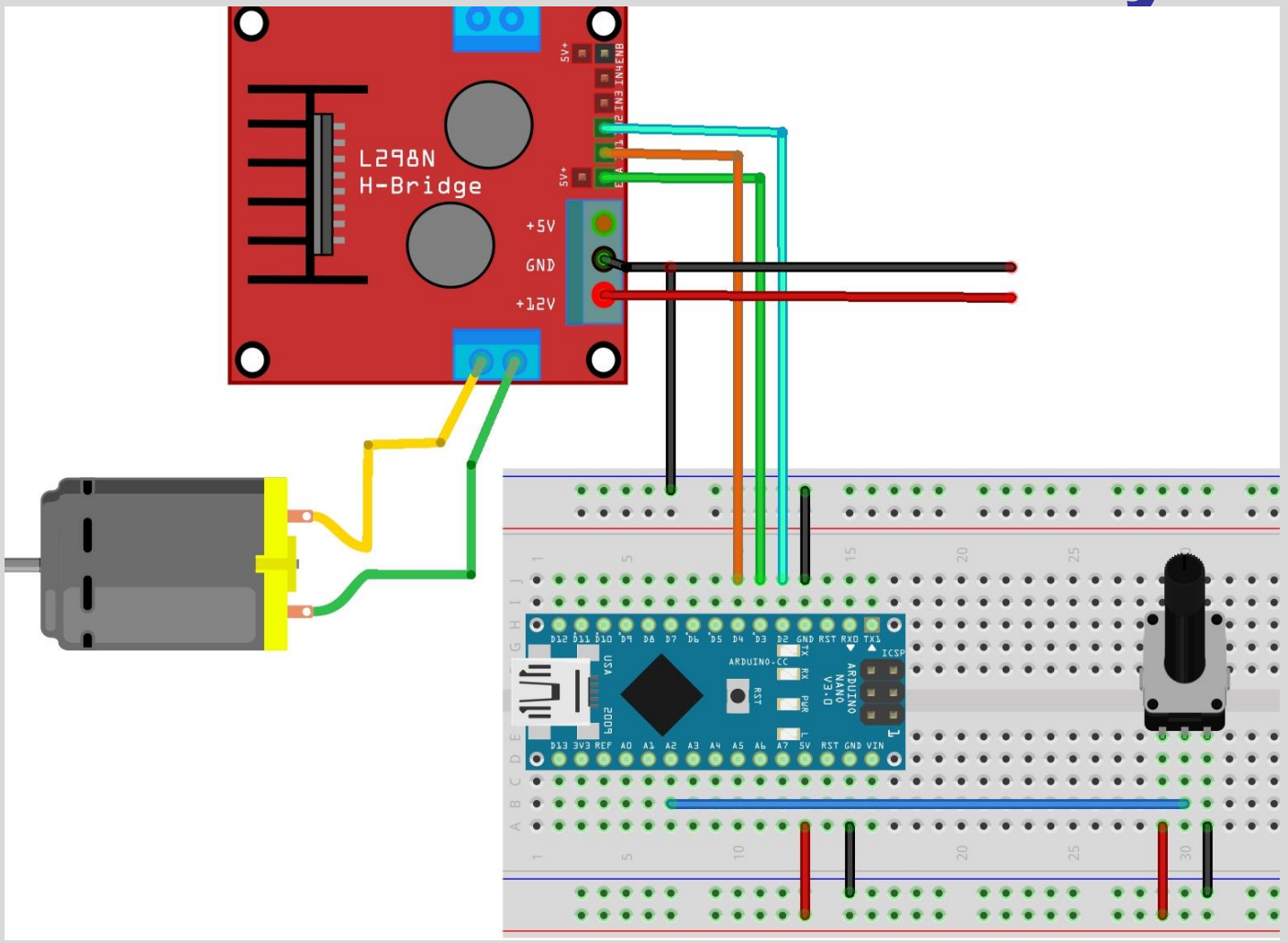
# DC motor fordulatszámának szabályozása

- **A kapcsolási rajz:**



# DC motor fordulatszámának szabályozása

- **A bekötési séma:**



# DC motor fordulatszámának szabályozása

- **A megírt program:**

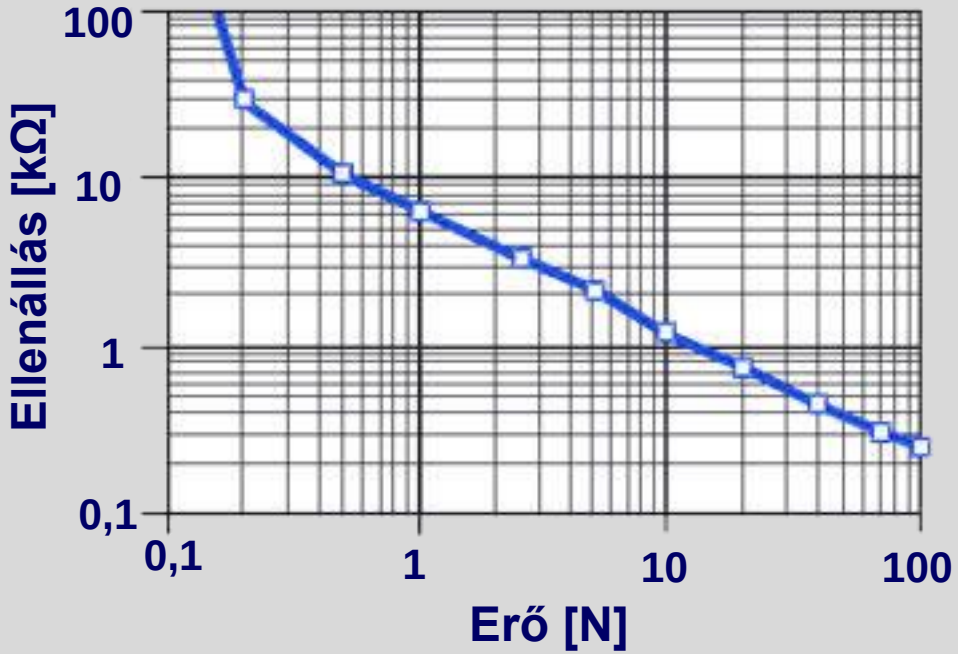
```
int IN1 = 4; int IN2 = 2; int en_pwm = 3;  int bejovo = 0; int pot_bemenet = 2;
void setup()
{
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(en_pwm,OUTPUT);
}
void loop()
{
  bejovo = analogRead(pot_bemenet);
  bejovo = map(bejovo, 0, 1023, 0, 255);
  analogWrite(en_pwm, bejovo);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
}
```

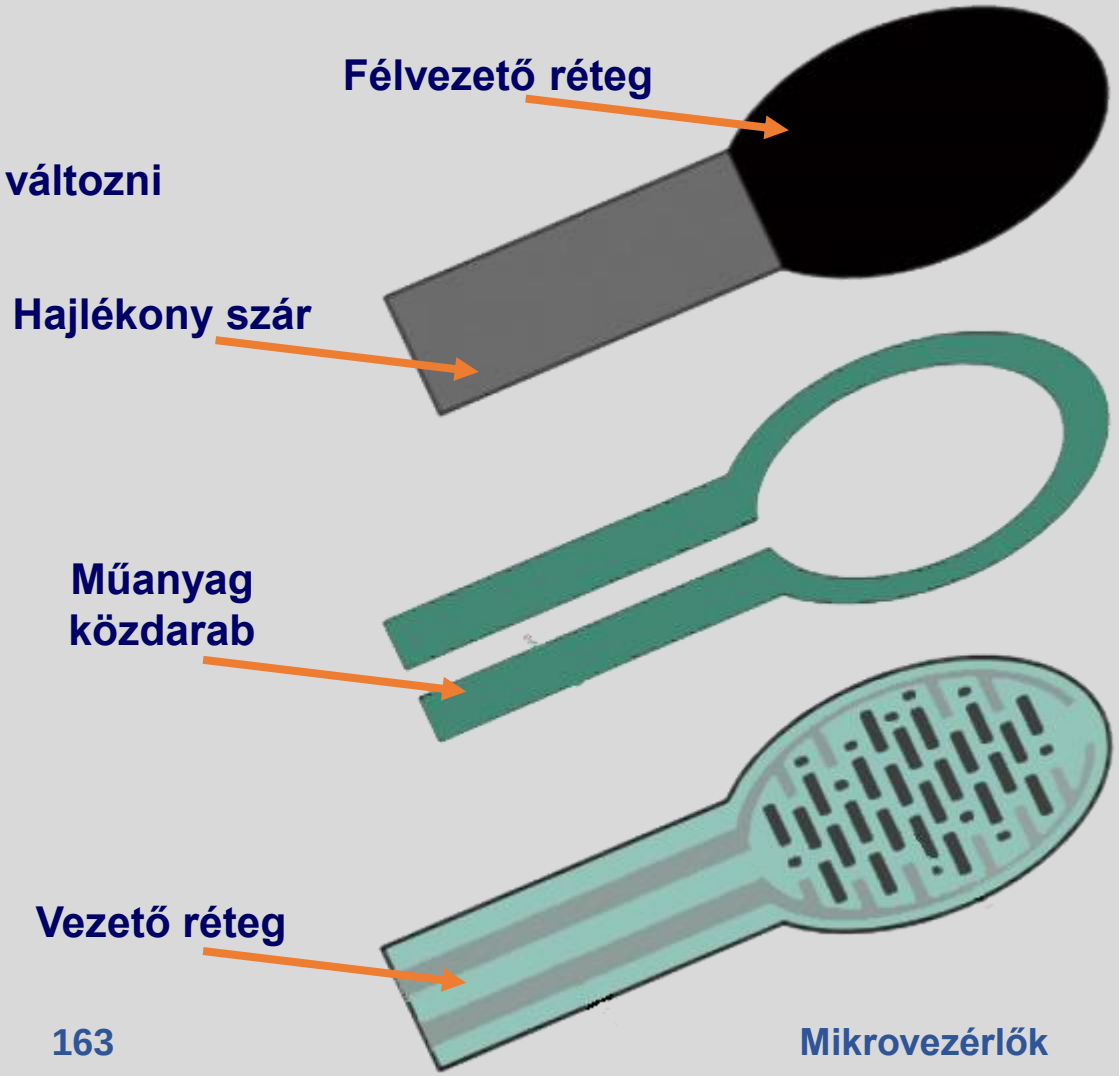
# 2 darab elektromágnes működtetése

- **Áramkör megtervezése:**
  - **2 db. elektromágnes**
  - **1 db. H-híd**
- **A feladat az előzőhöz hasonló.**

# Érdekesség: FSR (Force Sensitive Resistor)

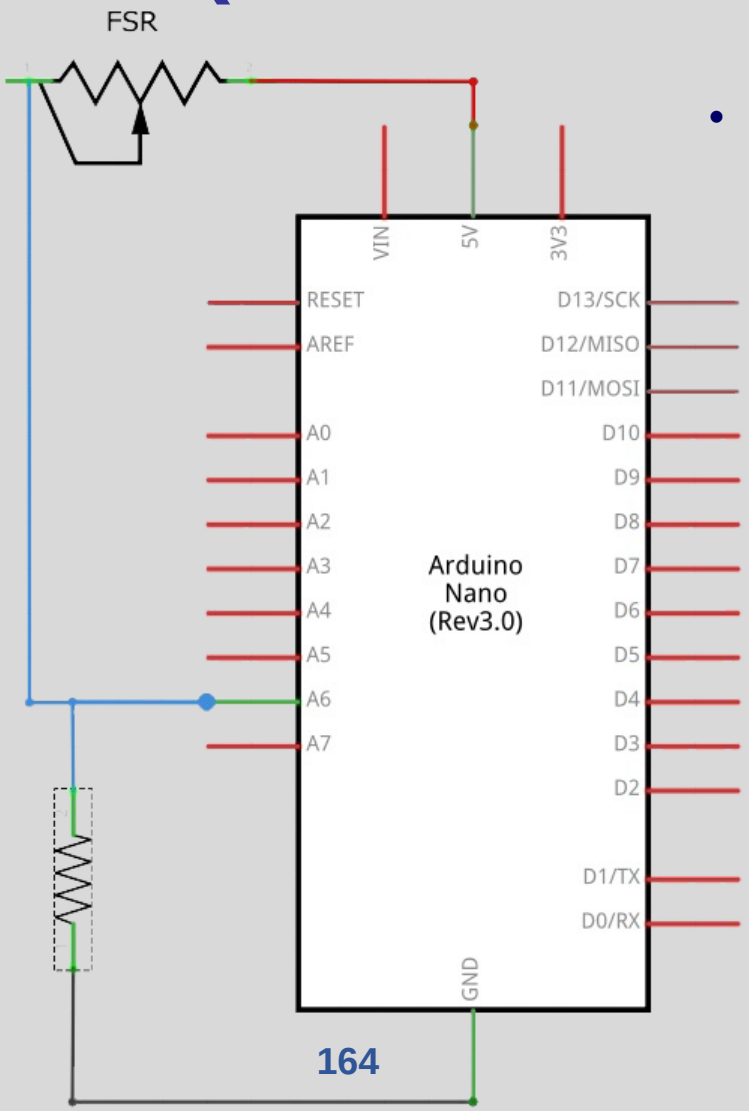
- **Erőmérés FSR használatával**
  - Az ellenállás értéke az erő függvényében fog változni
  - Viszonylag olcsó, és egyszerű a használata



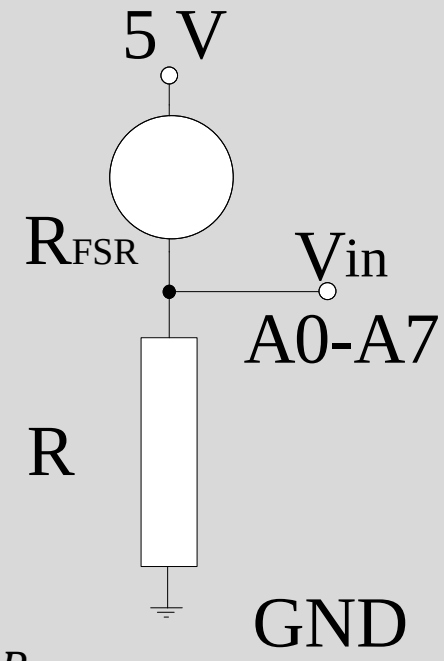



# Érdekesség: FSR (Force Sensitive Resistor)

- **A kapcsolási rajz:**
  - **10 kΩ lehúzó ellenállás**
  - **1 db. FSR érzékelő**



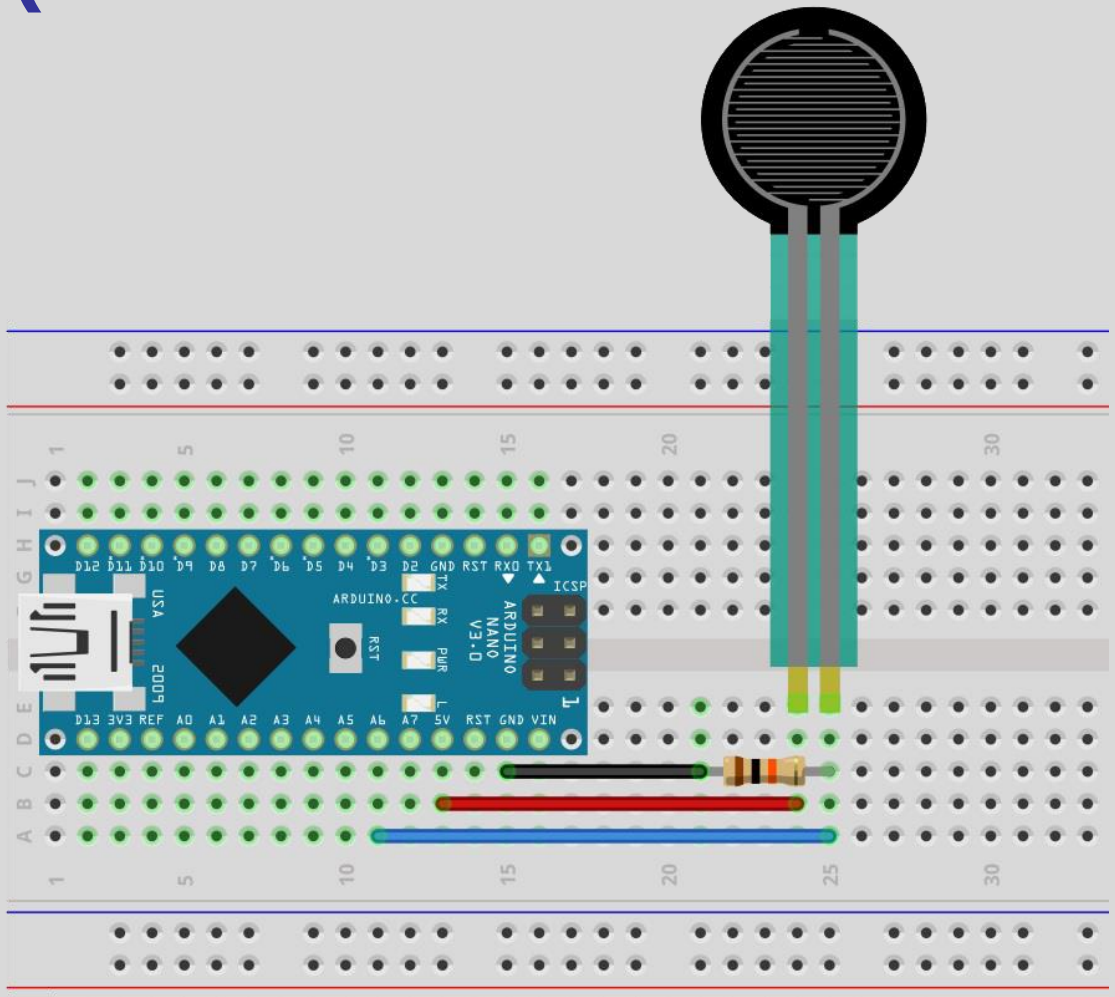

- **Feszültségosztó alkalmazása:**




$$V_{in} = U_{be}\frac{R}{R + R_{fsr}}$$

# Érdekesség: FSR (Force Sensitive Resistor)

- **A bekötési séma:**



# Köszönetnyilvánítás

AZ EMBERI ERŐFORRÁSOK MINISZTÉRIUMA ÚNKP-17-3 KÓDSZÁMÚ ÚJ NEMZETI KIVÁLÓSÁG PROGRAMJÁNAK TÁMOGATÁSÁVAL KÉSZÜLT.