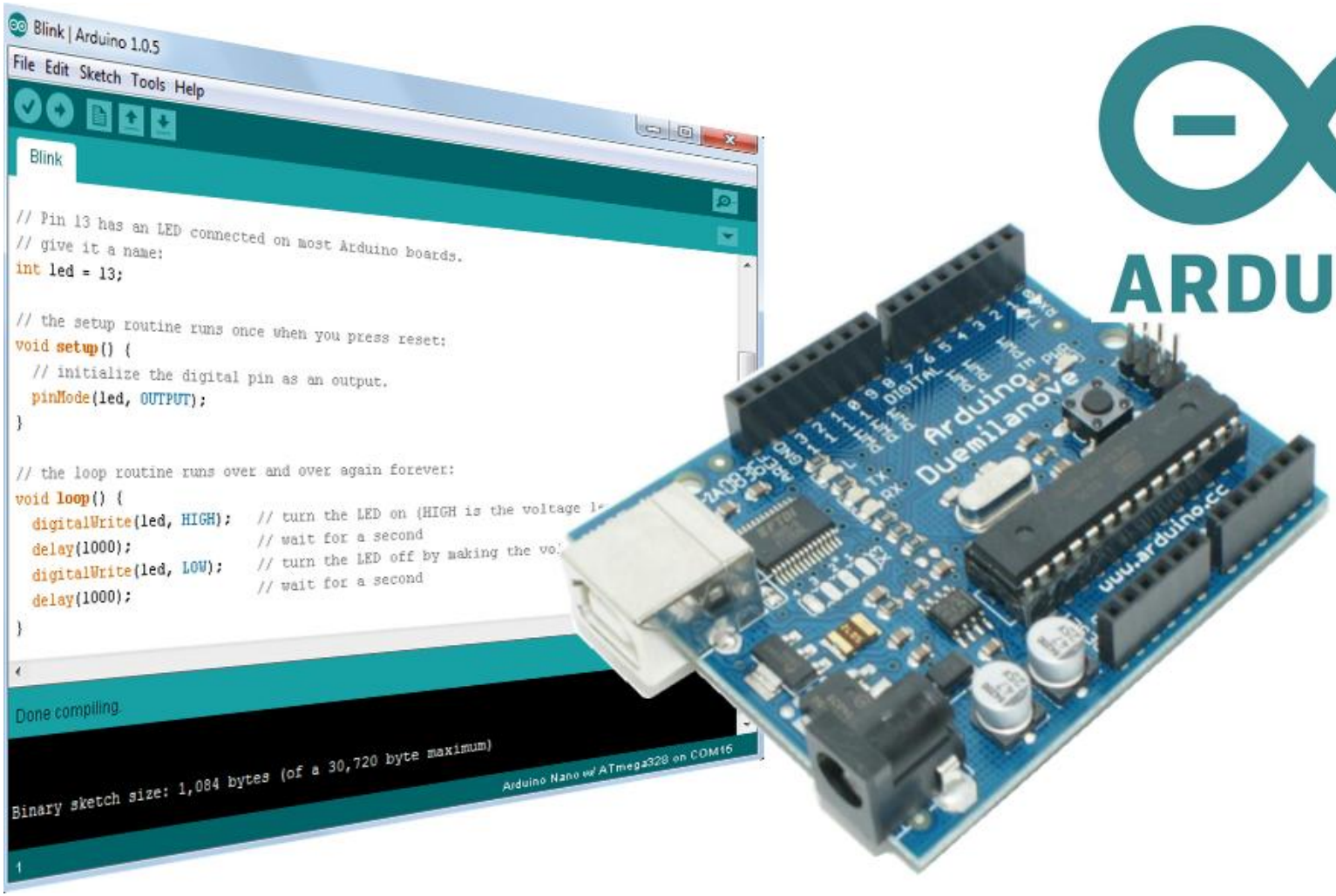


Bevezetés a mikrovezérlők programozásába:

Vezérlési szerkezetek, relációs operátorok

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 12 projektek

ledswitch – LED ki-bekapcsolgatása nyomógomb segítségével (nyomógomb kezelés alapprogram!)

LEDtristate –LED három állapotának (ki, be, villog) ciklikus váltogatása nyomógomb segítségével

RGBledswitch – RGB LED színeinek ciklikus váltogatása nyomógomb segítségével (8 állapot: 0 – 7 színkód)

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**

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

Utasítások és blokkok

- Egy olyan kifejezés, **mint x = 0, i++** vagy **digitalWrite(LED,HIGH)** **utasítássá válik,** ha egy pontosvesszőt írunk utána. Például:

```
x = 0;
i++;
digitalWrite(LED,HIGH);
```

Folyamatábra rajzjele

A C nyelvben a pontosvessző az utasításlezáró jel (terminátor)

- A **{}** kapcsos zárójelekkel deklarációk és utasítások csoportját fogjuk össze egyetlen **összetett utasításba vagy blokkba.** Ilyen a függvények utasításait összefogó zárójelpár, vagy az **if, else, while, for,** ill. hasonló utasítások utáni utasításokat összefogó zárójelpár. A blokk végét jelző jobb kapcsos zárójel után soha nincs pontosvessző. A változók bármelyik blokk belsejében deklarálhatók.

```
void loop() {          <-- Blokk kezdete
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  Serial.print("Humidity: ");
  Serial.print(h,1);
  Serial.print(" Temperature: ");
  Serial.print(t,1);
}                      <-- Blokk vége
```

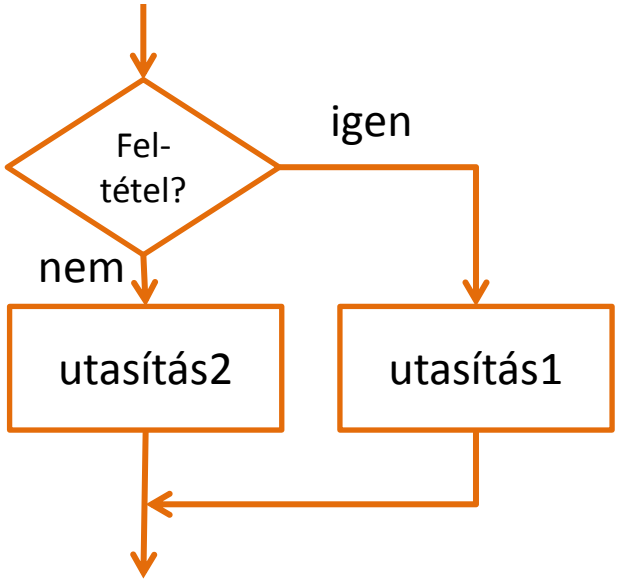
Részletek egy tavalyi mintaprogramunkból (DHT22 hőmérő és relatív páratartalom mérő)

Feltételes elágazás: if – else utasítás

Feltételvizsgálat, s ezt követően a végrehajtás menete a feltételvizsgálat eredményétől függ.

Szintaxis:

```
if(feltétel) utasítás1
else utasítás2
```



Az **else** ág opcionális…
A **feltétel** itt egy kifejezés, ami aritmetikai és relációs operátorokat is tartalmazhat. A kiértékelés eredménye **igaz,** ha a kifejezés értéke **true,** vagy 0-tól különböző, és **hamis** akkor, ha a kifejezés értéke **false,** vagy nulla.
Ha a feltétel teljesül, akkor **utasítás1** kerül végrehajtásra, egyébként pedig **utasítás2** lesz végrehajtva. Példa:

```
if(digitalRead(PUSH2)) digitalWrite(RED_LED,LOW);
else  digitalWrite(RED_LED,HIGH); //Amíg a gomb lenyomva, a LED világít
```

Speciális esetben (értékadásnál) használható a **rövid alak** (melyben a baloldal elhagyható…):
(változó =)feltétel?érték1:érték2; A feltétel teljesülése esetén érték1, egyébként érték2 lesz a visszatérési érték. Példa: inline maximum függvény definíciója a rendszerkönyvtárakban:

```
#define max(a,b)  ((a)>(b)?(a):(b))
```

Relációs és logikai operátorok

Logikai kifejezésekben, illetve feltételvizsgálatoknál az alábbi operátorok (műveletek) állnak rendelkezésre:

Relációs operátorok

==	egyenlő	pl. x == y
!=	nem egyenlő	pl. x != y
<	kisebb mint	pl. x < y
>	nagyobb mint	pl. x > y
<=	kisebb, vagy egyenlő	
>=	nagyobb, vagy egyenlő	

Logikai operátorok

Logikai változókon vagy logikai kifejezéseken végezett műveletekhez használhatók

!	Tagadás (negáció)	pl. !a
&&	ÉS művelet	pl. a && b
||	VAGY művelet	pl. a || b

Megjegyzések:

1. A relációs, illetve a logikai műveletek eredménye egy logikai (boolean) mennyiség, ami csak 0 (ha hamis) vagy 1 (ha igaz) értéket vehet fel.
2. A C fordítók kiterjesztett értelmezéssel bármilyen 0-tól különböző számot is elfogadnak igaz értékként. Így például `if(x != 0)` helyett egyszerűen `if(x)` -et is írhatunk.
3. Ne keverjük össze a logikai operátorokat a bitenként logikai operátorokkal!
   Bitenkénti műveletek: ~, &, |, ^ Logikai műveletek: !, &&, ||

if utasítások egymásba ágyazása

Egymásba ágyazott **if-else** szerkezeteknél, hiányzó **else** ágak esetén nem világos, hogy a meglévő **else** ág melyik **if** utasításhoz tartozik. Például az

```
if (n > 0)
    if (a > b) z = a;
    else z = b;
```

programrészletben az **else** a belső **if** utasításhoz tartozik.

Általános szabály: az **else** mindig a hozzá legközelebb eső, **else** ág nélküli **if** utasításhoz tartozik. Ha nem így szeretnénk, akkor használjunk kapcsos zárójeleket! Például:

```
if (n > 0) {
    if (a > b) z = a;
}
else z = b;
```

A nem egyértelmű helyzet különösen zavaró az olyan szerkezetekben, mint az alábbi:

```
if (n >= 0)
    for (i = 0; i < n; i++)
        if (s[i] > 0) {
            printf("...");
            return i;
        }
else      /* ez így hibás */
    printf("n értéke negatív") ;
```

A tagolás ellenére a fordító itt az **else** ágat a belső **if** utasításhoz kapcsolja.

Az ilyen, nehezen felderíthető hibák megelőzésére használjunk kapcsos zárójeleket a második **if** utasítás körül!

Az else if szerkezet

Az **else if** szerkezet adja a többszörös elágazások programozásának egyik legáltalánosabb lehetőségét.

A szerkezet úgy működik, hogy a program sorra kiértékeli a *kifejezéseket* és ha bármelyik ezek közül igaz, akkor végrehajtja a megfelelő *utasítást*, majd befejezi az egész vizsgáló láncot.

Itt is, mint bárhol hasonló esetben, az utasítás helyén kapcsos zárójelek között elhelyezett blokk is állhat.

Az utolsó **else** ág alapértelmezés szerint a *„fentiek közül egyik sem”* esetet kezeli. Ha ilyenkor semmit sem kell csinálni, ez az ág elhagyható.

Szintaxis:

```
if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
else if (kifejezés)
    utasítás
.
.
.
else
    utasítás
```

A switch utasítás

A switch utasítás is a többirányú programelágaztatás egyik eszköze. Az utasítás úgy működik, hogy összehasonlítja egy kifejezés értékét több egész értékű állandó kifejezés értékével, és az ennek megfelelő utasítást hajtja végre.

A switch utasítás általános felépítése:

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

Példa:

```
switch (led_state) {
    case 0: digitalWrite(LED,0);
    case 1: digitalWrite(LED,1);
    . . .
   default: digitalWrite(!digitalRead(LED));
            delay(250);
}
```

Példaprogramok feltételes elágazással

1. **ledswitch.ino:** Írjunk programot, amelyben egy nyomógomb segítségével ki-be kapcsolgatunk egy LED-et!
2. **LEDtristate.ino:** Írjunk programot, amelyben egy nyomógomb segítségével vezérelünk egy LED-et, melynek a ki- és bekapcsolt állapota mellett egy harmadik, villogó állapota is van!
3. **RGBswitch.ino:** Váltogassuk egy nyomógombbal egy RGB LED színeit!

Emlékeztető: egyszerű I/O vezérlés

Digitális I/O

- ✓ **pinMode**(pin, mode) – kivezetés üzemmódjának beállítása
- ✓ **digitalWrite**(pin, state) - kimenetvezérlés
- ✓ **digitalRead**(pin) – bemenet állapotának lekérdezése

mode: **OUTPUT, INPUT, INPUT_PULLUP** state: **LOW, HIGH**

Analóg I/O

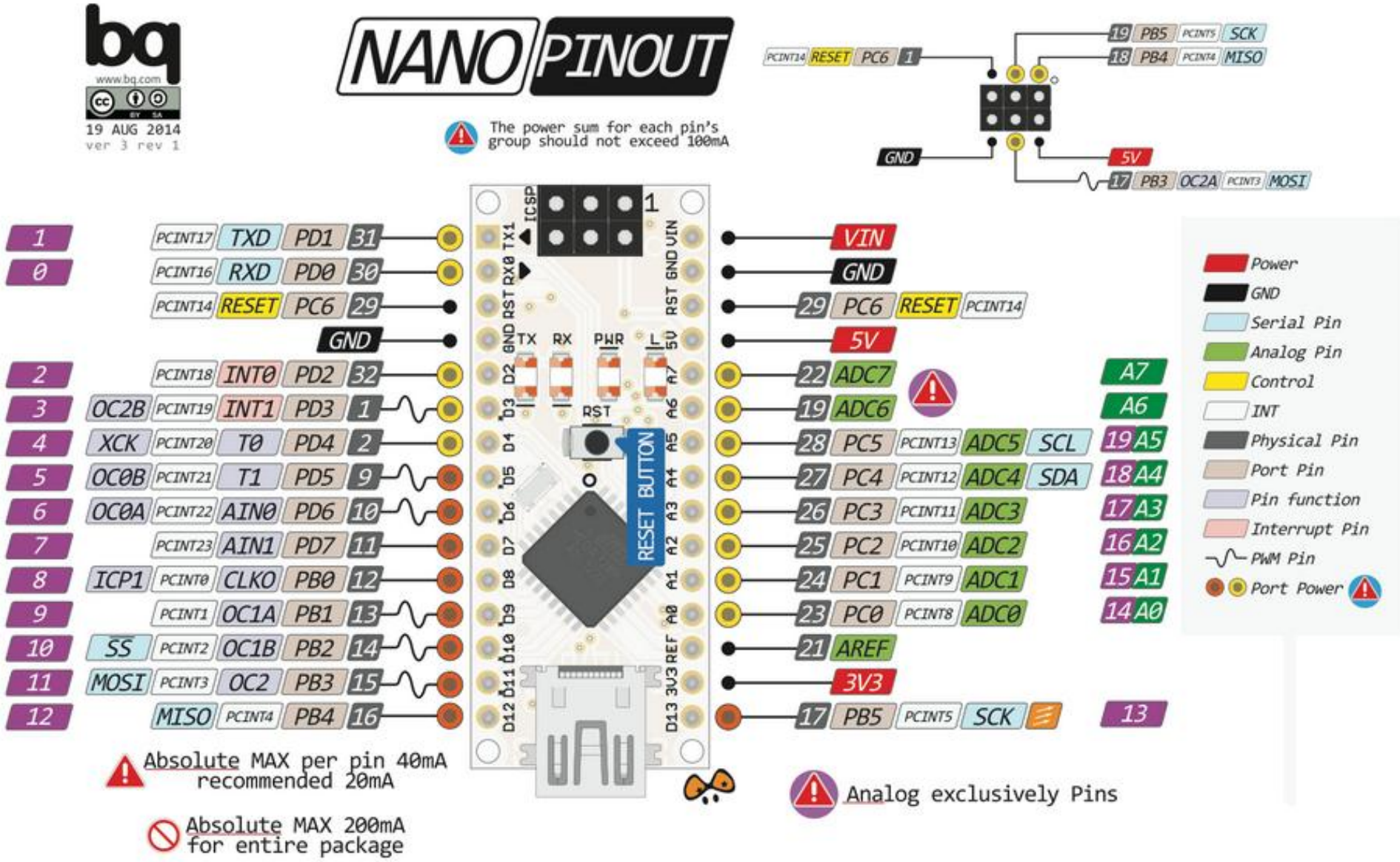
- **analogReference**(ref) – ADC referenciájának megadása
- **analogRead**(chan) – analóg-digitális konverzió eredménye
- **analogWrite**(pin) - PWM teljesítményvezérlés

ahol:

ref: **DEFAULT** (VCC), **INTERNAL** (1V1) vagy **EXTERNAL** (Arduino)

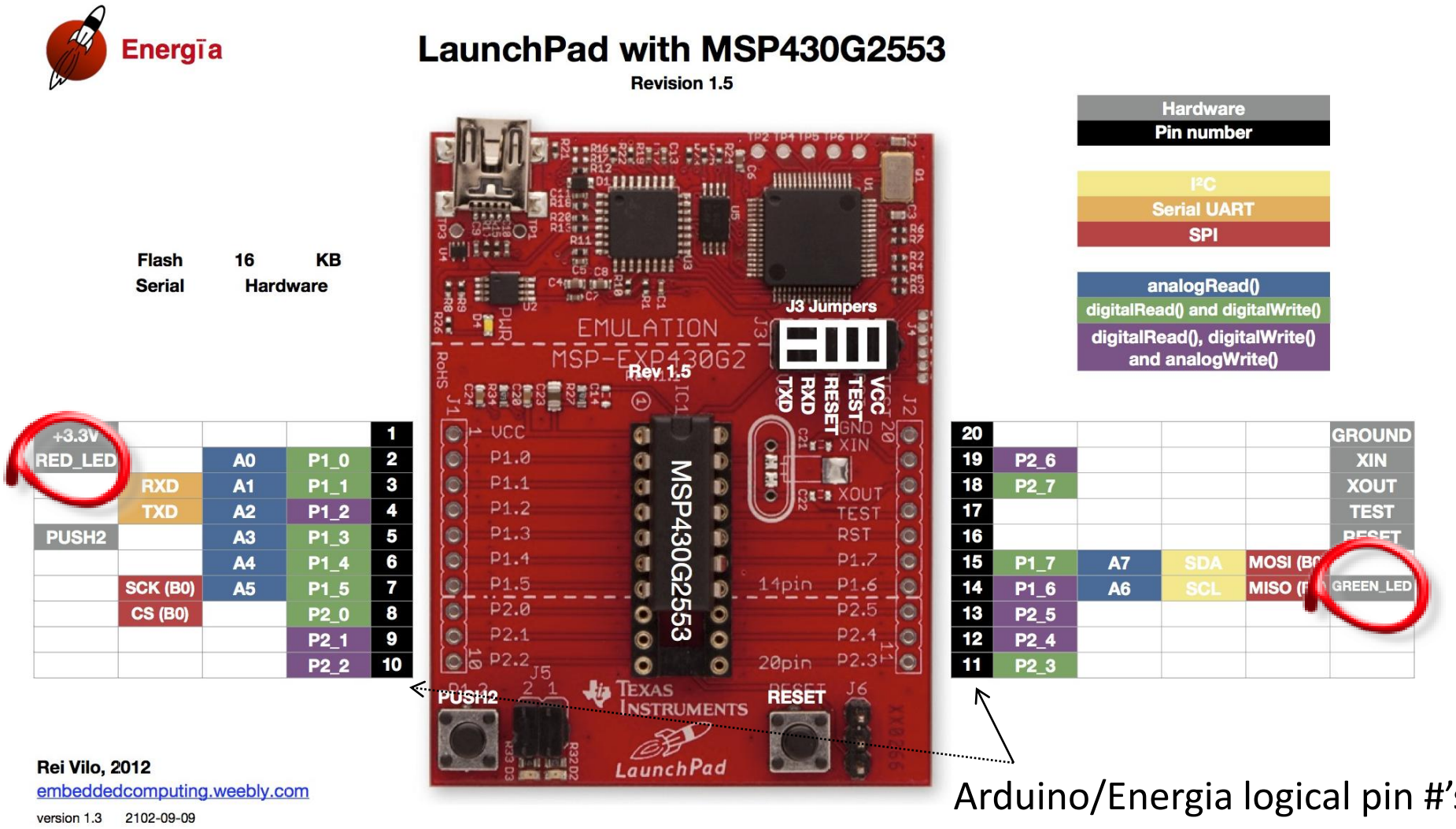
Energia esetén *INTERNAL* helyett **INTERNAL1V5** vagy **INTERNAL2V5** használható

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

A kapcsolási elrendezés

1. Feladat: A piros LED kapcsolgatjuk a nyomógomb segítségével:

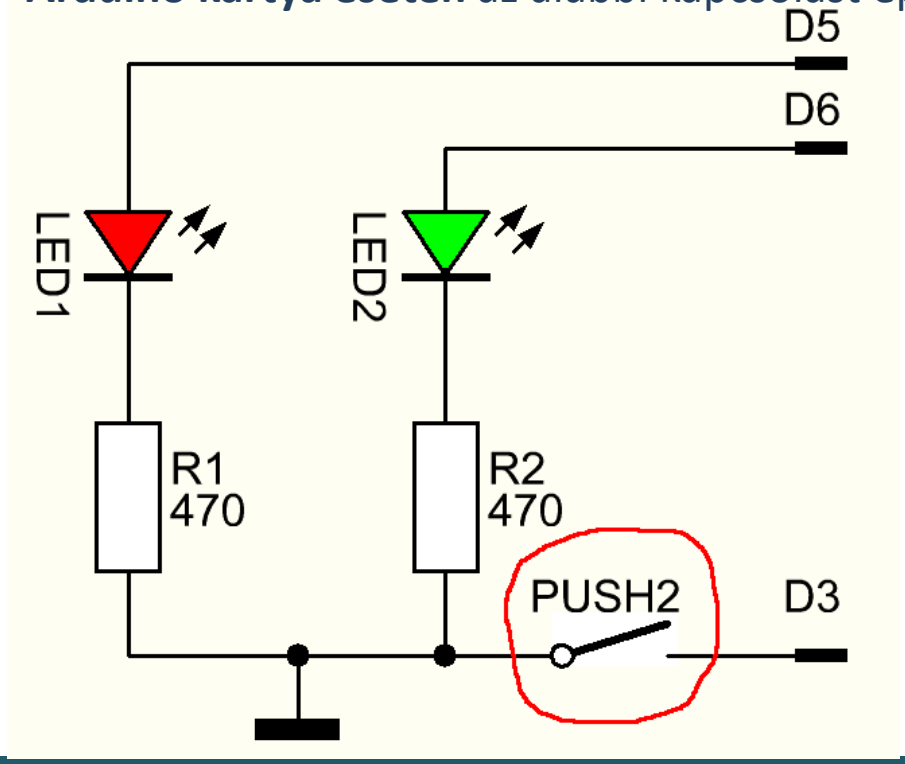
Első (minden páratlan) lenyomásra a piros LED világítson!

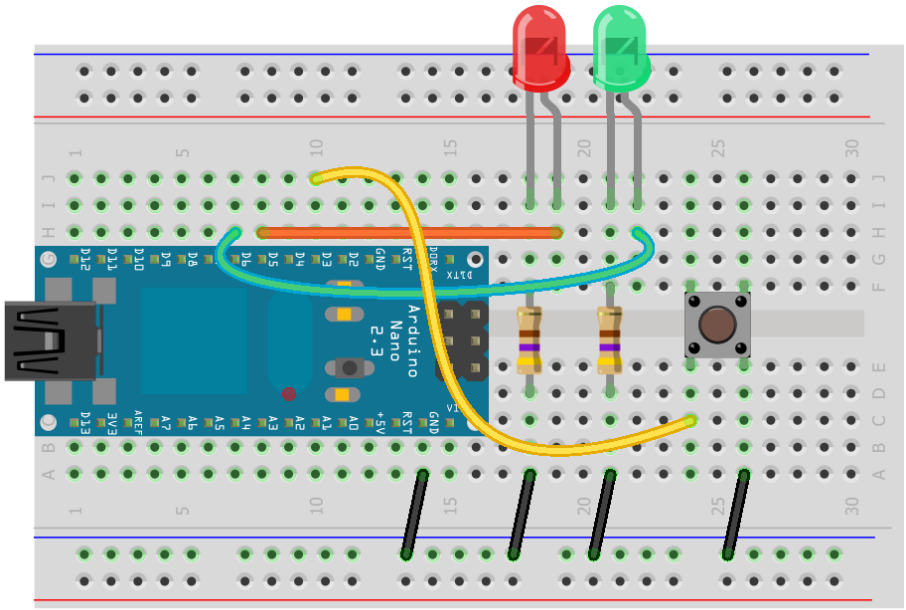
Második (minden páros) lenyomásra a piros LED kialszik!

A kapcsolás lehet ugyanaz, amit az előző foglalkozáson használtunk!

Megjegyzés: Az **MSP430 Launchpad kártya esetében** a gyárilag ráépített LED-et és az SW2 nyomógombot használhatjuk.

Arduino kártya esetén az alábbi kapcsolást építsük meg!





Made with Fritzing.org

1. ledswitch.ino

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

Hardverfüggő rész,
csak az Arduino kártyához kell!

```
//Hardverfüggetlen rész, MSP430 Launchpad kártyán is futtatható
boolean led_state = false;        //Kezdetben a LED ki van kapcsolva
boolean waitforpress = true;      //Kezdetben lenyomásra várunk

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

Folytatás a következő oldalon…

A változók szerepe:
led_state: ebben tartjuk nyilván a LED állapotát (true = világít, false = nem világít)

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

1. ledswitch.ino

```
void loop() {
  if(waitforpress) {                //Ha lenyomásra várunk és
    if(!digitalRead(PUSH2)) {       //Ha lenyomás történt...
      led_state = !led_state;       //LED állapotának átbillentése
      digitalWrite(RED_LED, led_state);
      waitforpress = false;         //Következő stáció: felengedésre várunk
    }
  }
  else {                            //Ha felengedésre vártunk és
    if(digitalRead(PUSH2)) {        //Ha felengedést észlelünk...
      waitforpress = true;          //Következő stáció: lenyomásra várunk
    }
  }
  delay(20);                        //pergésmentesítő késleltetés
}
```

Nyomógomb esetében *lenyomás – felengedés* ciklusokban kell gondolkodni:

- Ha lenyomásra vártunk és lenyomva találjuk a gombot, akkor átkapcsoljuk a LED állapotát és átlépünk a felengedés várása állapotba.
- Ha felengedésre vártunk és felengedve (magas állapotban) találjuk a nyomógombot, akkor vége egy *lenyomás-felengedés* ciklusnak, újra lenyomásra várunk.

2. LEDtristate.ino

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

Hardverfüggő rész,
csak az Arduino kártyához kell!

```
//Hardverfüggetlen rész, MSP430 Launchpad kártyán is futtatható
boolean waitforpress = true;      //Kezdetben lenyomásra várunk
int led_state = 0;                //Kezdetben a LED ki van kapcsolva
int timecount = 25;               //Villogás félperiódusának ideje

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

Csak itt változtattunk

Folytatás a következő oldalon…

A változók szerepe:
waitforpress: a nyomógomb *lenyomás – felengedés* ciklusának nyilvántartására használjuk (true = lenyomásra várunk, false = felengedésre várunk)
led_state: ebben tartjuk nyilván a LED állapotát (0 = nem világít, 1 = világít, 2 = villog)
timecount: a 20 ms-os időszeletek (vissza)számlálója villogtatáskor

2. LEDtristate.ino

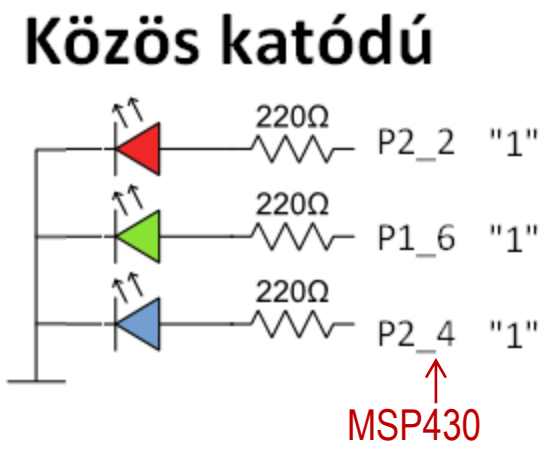
```
void loop() {
  if(waitforpress) {                //Ha lenyomásra várunk és
    if(!digitalRead(PUSH2)) {       //Ha lenyomás történt...
      if(++led_state > 2) led_state = 0;  //Állapot léptetés (2 után körülfordul)
      digitalWrite(RED_LED,led_state > 0);//LED állapot beállítása
      waitforpress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

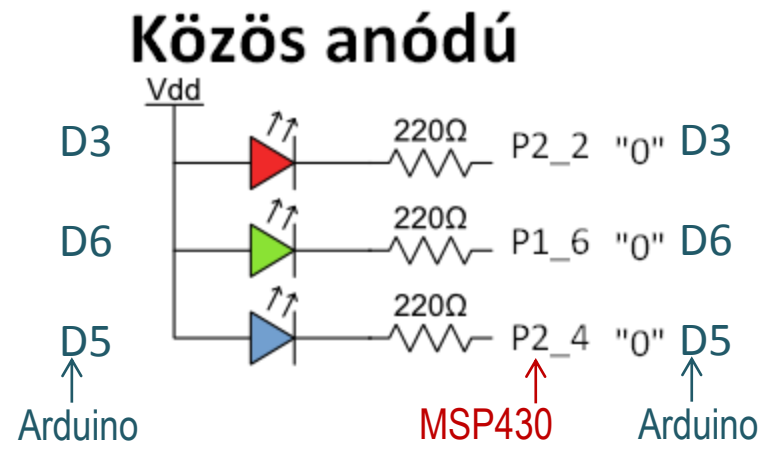
Ügyeljünk rá, hogy a villogtatást a nyomógomb állapotát figyelő feltételeken kívül kell végezni!

Az RGB LED bemutatása

Az RGB LED három, különböző színű LED egy közös tokban.
A három szín a három alapszín, amelyből minden más szín kikeverhető (additív színkeveréssel):

Red = piros
Green = zöld
Blue = kék

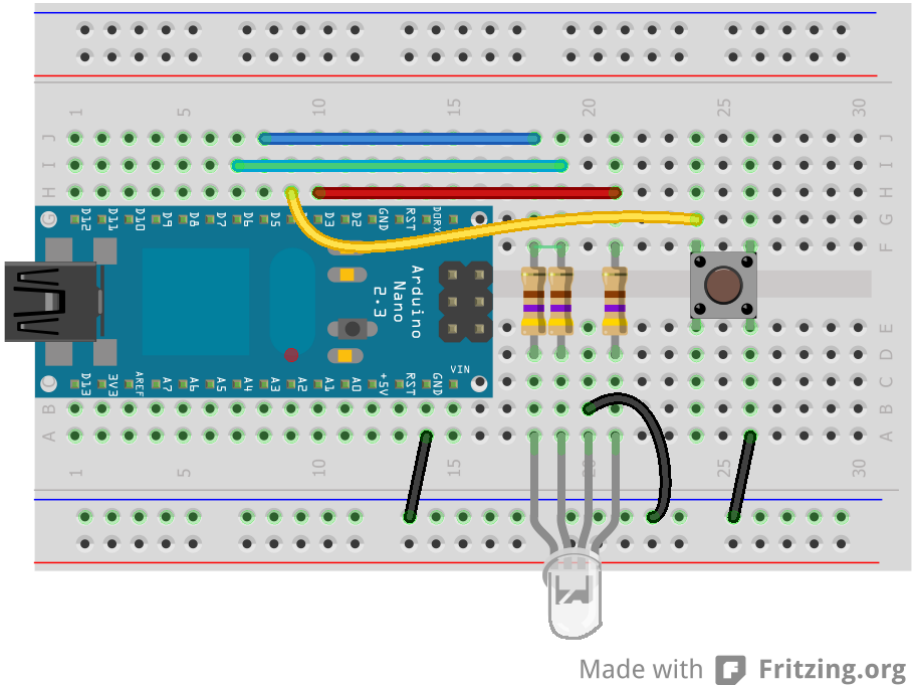




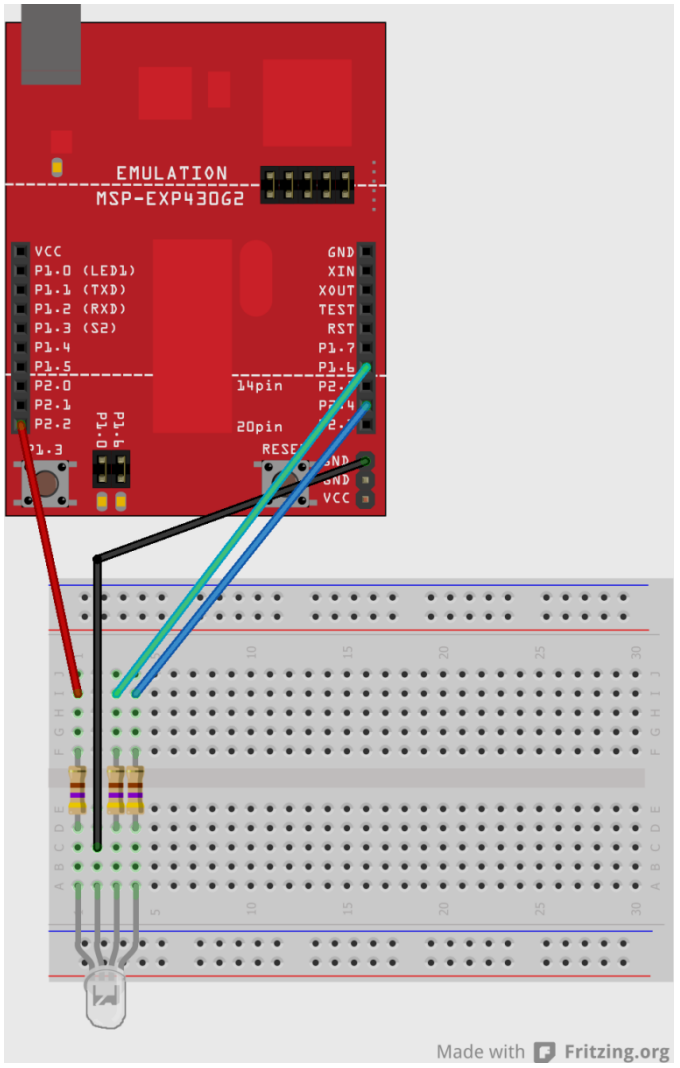
Arduino esetén 470Ω-os ellenállásokat használjunk!

Huzalozási vázlat

R LED = D3
G LED = D6
B LED = D5
PUSH2 = D4



R LED = P2_2
G LED = P1_6
B LED = P2_4
PUSH2 = P1_3



3. RGBledswitch.ino

Nyomógombbal 0 és 7 között sorban léptetjük az RGB LED színkódjait, melynek értelmezése: 0. bit = piros, 1. bit = zöld 2. bit = kék alapszín.

Hardverfüggő rész

ARDUINO kártyához

```
#define LED_RED     3
#define LED_GREEN   6
#define LED_BLUE    5
#define PUSH2       4
```

MSP430 Launchpad kártyához

```
#define LED_RED     P2_2
#define LED_GREEN   P1_6
#define LED_BLUE    P2_4
#define PUSH2       P1_3
```

```
//--- Hardverfüggetlen rész ----
int color = 0;                          //színkód (0..7)
boolean waitforpress = true;

void setup() {
  // Inicializáljuk a ki- és bemeneteket
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_BLUE, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(PUSH2,INPUT_PULLUP);
}
```

Folytatás a következő oldalon…

A nyomógomb *lenyomás – felengedés* ciklusainak lekezelése itt is ugyanúgy történik, mint az 1. példaprogramban. Az RGB LED színváltásainak megvalósítása a bekeretezett sorokban történik – csak ez az új elem ebben a programrészben.

```
void loop() {
  if(waitforpress) {                  //Ha lenyomásra várunk és
    if(!digitalRead(PUSH2)) {         //Ha lenyomás történt...
      color = ++color & 0x07;         //Szín léptetése
      digitalWrite(LED_RED,  color & 1);   // piros LED ki
      digitalWrite(LED_GREEN,color & 2);   // zöld LED ki
      digitalWrite(LED_BLUE, color & 4);   // kék LED ki
      waitforpress = false;           //Következő stáció: felengedésre várunk
    }
  }
  else {                              //Ha felengedésre vártunk és
    if(digitalRead(PUSH2)) {          //Ha felengedést észlelünk...
      waitforpress = true;            //Következő stáció: lenyomásra várunk
    }
  }
  delay(20);                          //pergésmentesítő késleltetés
}
```