Arduino C/C++ programozás 3.

18. Élfigyelés
19. Hétszegmenses kijelzők multiplex vezérlése
20. Arduino kivezetések közvetlen elérése
21. Léptető motor vezérlése
22. Szervó motor vezérlése
23. I^2C kommunikáció
24. Hőmérséklet mérés
25. Feladatok

Felhasznált forrás, és ajánlott irodalom:
Ruzsinszki Gábor: Programozható Elektronikák

Felhasznált és ajánlott online szimulációs felület: www.tinkercad.com

18.1. Élfigyelés

Élfigyelés

A bemenetek lekérdezésekor vannak olyan esetek amikor nemcsak az a kérdés, hogy adott kapcsoló/nyomógomb le van-e éppen nyomva (vagy nem), hanem fontos azt is tudnunk, hogy

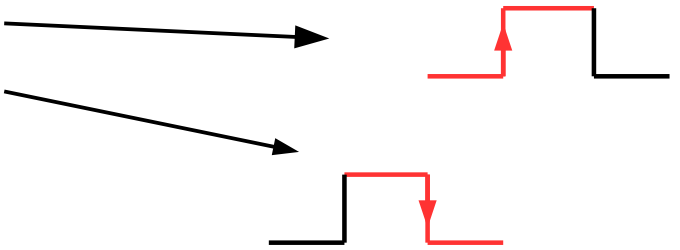
- folyamatosan nyomva tartjuk már egy ideje
- vagy már felengedtük és most újra lenyomtuk

Tehát számolni szeretnénk a lenyomásokat

- azért mert léptetni szeretnénk valamit
- vagy minden új lenyomásra egy új állapotba kell kerülnie a vezérelt rendszerünknek

Az élfigyelés kétféle lehet, figyelhetünk

- felfutó élt → váltás 0 állapotból 1 állapotba
- lefutó élt → váltás 1 állapotból 0 állapotba



Élfigyelés megvalósítása

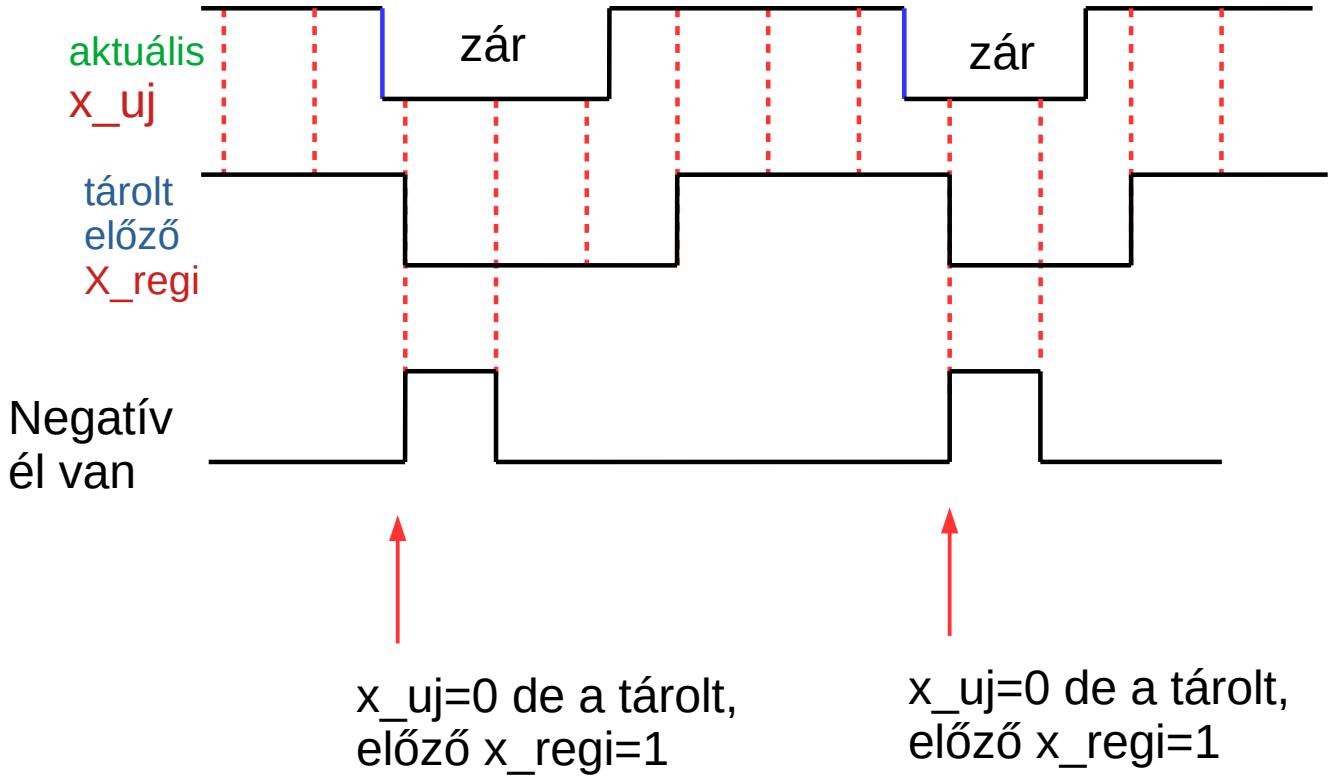
Többféleképpen lehetséges

- megszakítás használatával → bemenet állapot változásának figyelése
  ARDUINO esetén nem mindegyik bemenetnél lehetséges !!
- egyszerűen ciklikusan lekérdezzük a bemenet állapotát és összehasonlítjuk az aktuális és az előző lekérdezés értékeit
  → tárolni kell egy változóban a régi (előző) lekérdezés eredményét
  → a lekérdezési frekvenciát annak megfelelően válasszuk meg, hogy milyen gyorsan kell reagálnunk a kapcsoló működtetésére

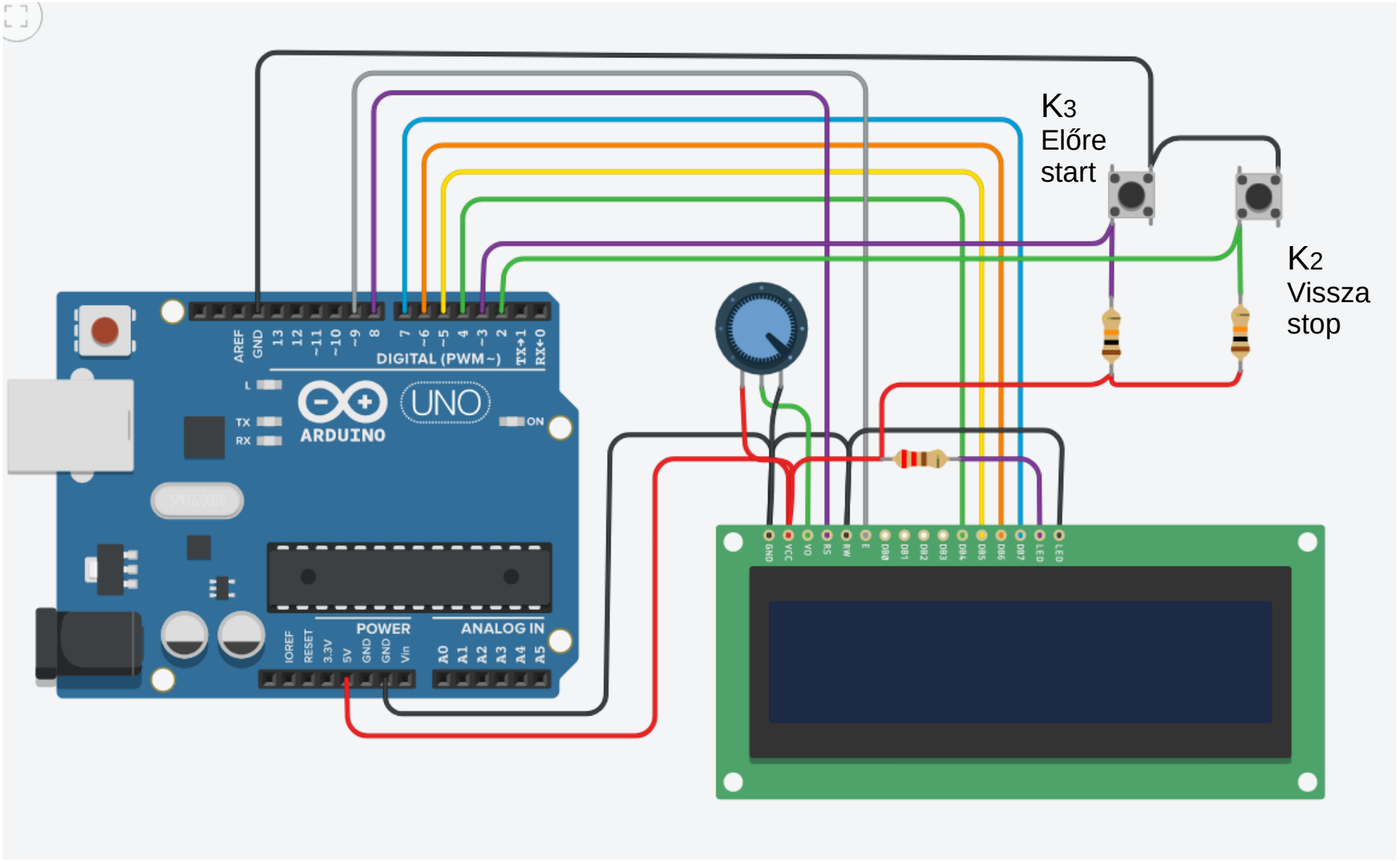
18.2. Élfigyelés

Élfigyelés bemenet ciklikus lekérdezésével

x láb figyelése → digitalRead(x)
bekötés: kapcsoló nyitva – H szint, kapcsoló zárva – L szint → negatív él figyelése !!



18.3. LCD kijelző vezérlése élfigyeléssel



18.4. Élfigyelés megvalósítása

1. mintafeladat

- A kijelző első sorában jelenítsünk meg egy üdvözlő üzenetet, a 2. sorban pedig a ’0’ értéket
- a K3 minden lenyomására a 2. sorban folyamatosan növeljük a kijelzett számot 1-el
  → 0-1-2-3-4-5-6-7-8-9-10-11- …

```
// ezt be kell importálni a kijelző használatához !!
#include <LiquidCrystal.h>

// LCD kijelző bekötése (RS, EN, D4, D5, D6, D7)
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
const byte k3=3;                    // K3   nyomógomb
byte k3_old=1;                      // K3  előző állapota
byte k3_new=1;                      // K3 új, aktuális állapota

byte szam=0;          // változó a számláláshoz

void setup()
 {
       pinMode(k3, INPUT);      //  'K3' bemenet lesz (nyomógomb)
       lcd.begin(16, 2);        // 16x2 karakter/soros LCD kijelző beállítása
       delay(1000);             // késleltetünk 1000ms-t
       lcd.setCursor(0, 0);     // kurzor pozíció 1. sor 1. karakter
       lcd.print("Hello !");    // szöveg kiíratása (1. sorban)
       lcd.setCursor(0, 1);     // kurzor pozíció 2. sor 1. karakter
       lcd.print(szam);         // változó értékének kiíratása
       delay(2000);
 }
```

18.5. Élfigyelés megvalósítása

1. mintafeladat folytatása

```
void loop()
{
        lcd.setCursor(0, 1);          // kurzor pozíció 2. sor 1. karakter
        lcd.print("          ");      // előző érték törlése
        lcd.setCursor(0, 1);          // kurzor pozíció 2. sor 1. karakter
        lcd.print(szam);              // változó aktuális értékének kiíratása

        k3_new = digitalRead(k3);             // nyomógomb lekérdezése, és új állapot tárolása
        if((k3_new==0)&&(k3_old==1))          // negatív él !! → +1
              { szam++; }
        k3_old=k3_new;                        // előző állapot letárolása

        delay(100);                           // kicsit késleltetünk

}
```

18.6. Élfigyelés megvalósítása

2. mintafeladat

- Az 1. mintafeladatot picit bővítsük → a K2 nyomógomb lenyomására 1-el csökkentsük a számot !

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
const byte k3=3;            // K3  nyomógomb
const byte k2=2;            // K2  nyomógomb
byte k3_old=1;              // K3  előző állapota
byte k3_new=1;              // K3 új, aktuális állapota
byte k2_old=1;              // K2  előző állapota
byte k2_new=1;              // K2 új, aktuális állapota

byte szam=0;          // változó a számláláshoz

void setup()
 {
       pinMode(k3, INPUT);      //  'K3' bemenet lesz (nyomógomb)
       pinMode(k2, INPUT);      //  'K2' bemenet lesz (nyomógomb)
       lcd.begin(16, 2);        // 16x2 karakter/soros LCD kijelző beállítása
       delay(1000);
       lcd.setCursor(0, 0);     // kurzor pozíció 1. sor 1. karakter
       lcd.print("Hello !");    // szöveg kiíratása (1. sorban)
       lcd.setCursor(0, 1);     // kurzor pozíció 2. sor 1. karakter
       lcd.print(szam);         // változó értékének kiíratása
       delay(2000);
 }
```

18.7. Élfigyelés megvalósítása

2. mintafeladat folytatása

```
void loop()
{
        lcd.setCursor(0, 1);        // kurzor pozíció 2. sor 1. karakter
        lcd.print("         ");     // előző érték törlése
        lcd.setCursor(0, 1);        // kurzor pozíció 2. sor 1. karakter
        lcd.print(szam);            // változó aktuális értékének kiíratása

        k3_new = digitalRead(k3);           // előre nyomógomb lekérdezése, és új állapot tárolása
        if((k3_new==0)&&(k3_old==1))        // K3 negatív él !! → +1
              { szam++; }
        k3_old=k3_new;                      // előző állapot letárolása

        k2_new = digitalRead(k2);           // hátra nyomógomb lekérdezése, és új állapot tárolása
        if((k2_new==0)&&(k2_old==1))        // K2 negatív él !! → -1
              { szam--; }
        k2_old=k2_new;                      // előző állapot letárolása

        delay(100);                         // kicsit késleltetünk
}
```

19.1. Több hétszegmenses kijelző vezérlése

A hardver szempontjából többféleképpen megoldható több hétszegmenses kijelző, de mindenféleképpen bonyolódni fog a dolog

1. eset

- Teljesen függetlenül kezeljük a 7szegm. kijelzőket → ezzel egy nagy probléma van → minden kijelző 8 (vagy 7, ha a pontot nem használjuk) digitális kimenetet használ el !! → nem sokat lehetséges így rákötni a mikrovezérlőre
  Persze hogy mennyit, az függ a mikrovezérlő típusától (mennyi digitális kimenete van) és attól hogy milyen egyéb digitális bemeneteket, kimeneteket kell használnunk

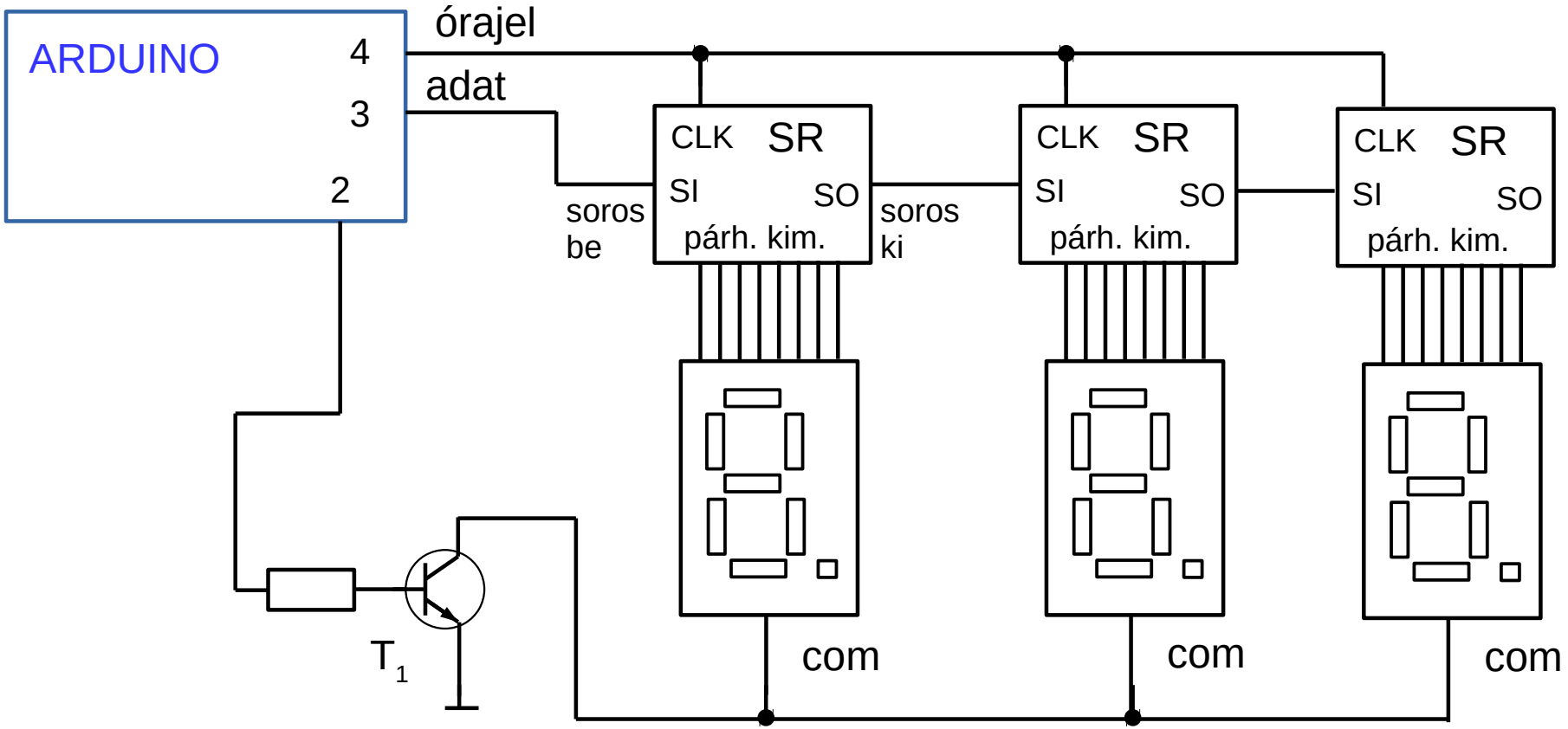
2. eset, multiplex vezérlés

- A 7szegmenses kijelzőket párhuzamosan kötjük rá a 8 (vagy 7) digitális kimenetre
- természetesen így mindig ugyanazt az értéket látnánk egyszerre mindegyiken → meg kell oldani, hogy egyenként tudjuk őket bekapcsolni, kikapcsolni
- a közös (COM) bemenetükre nem kapnak közvetlenül 0 (vagy Ut) értéket, hanem tranzisztorokkal kapcsoljuk ezeket
- a tranzisztorokat a mikrovezérlővel kapcsolgatjuk → ez kijelzőként egy plusz digitális kimenetet igényel !
- ha elég gyorsan kapcsolgatjuk a kijelzőket → úgy tűnik mintha egyszerre világítanának

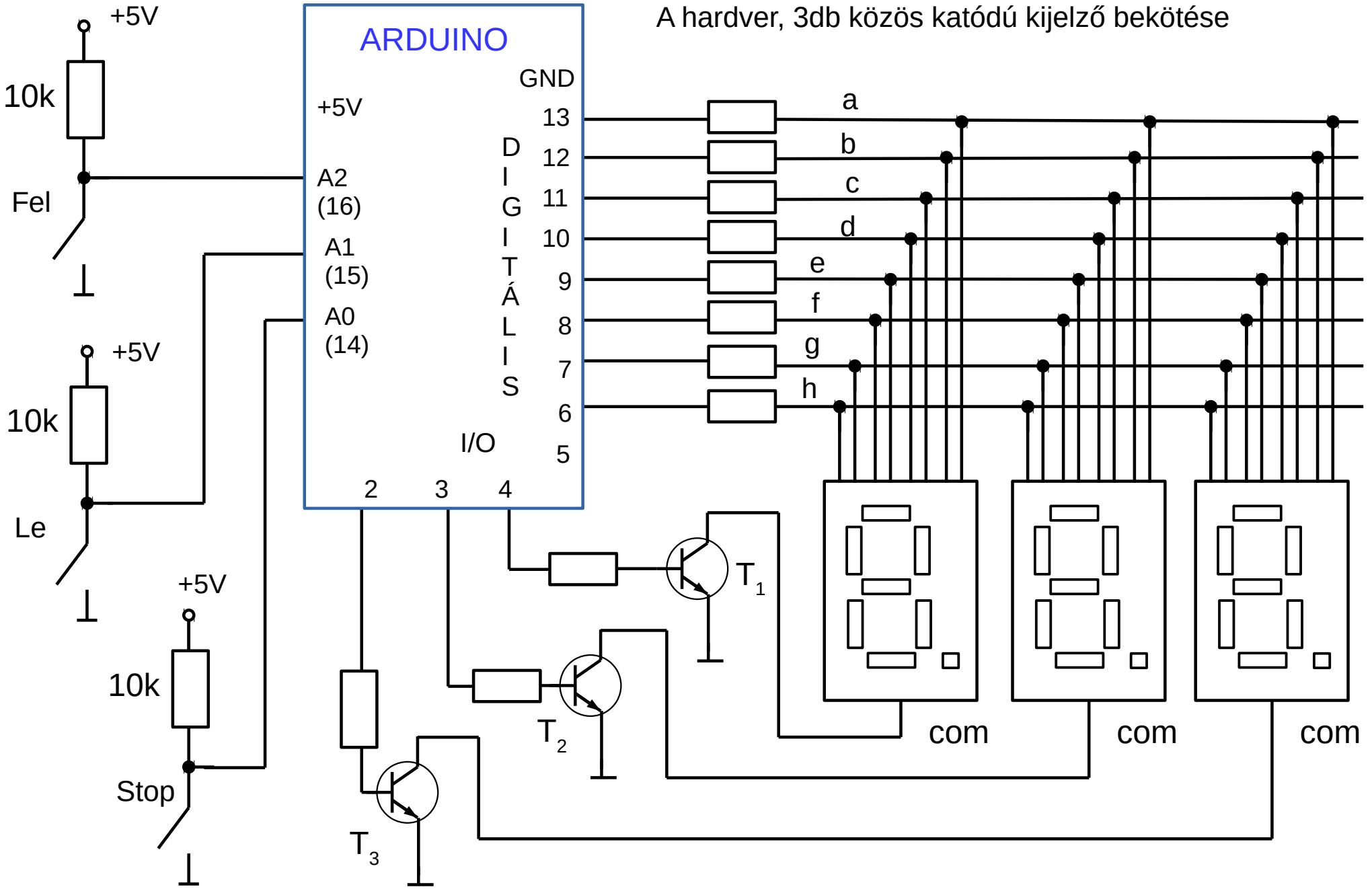
19.2. Több hétszegmenses kijelző vezérlése

3. eset, soros vezérlés shift regiszterekkel

- A 7szegm. kijelzőket közvetlenül shift regiszterekre kötjük
  (vagy egyéb vezérlő chipre)
- a shift regisztereket pedig sorban felfűzzük és sorosan küldjük ki nekik a biteket a mikrovezérlőről
- nagyon kevés digitális kimenettel megoldható (2, vagy 3) sok kijelző vezérlése



19.3. Hétszegmenses kijelzők multiplex vezérlése



19.4. Hétszegmenses kijelzők multiplex vezérlése

1. mintafeladat

- A 1-2-3-1-2-3-1 … értékek kijelzése folyamatosan (2s-ig mindegyik)
- de most mindegyik szám másik kijelzőn !!
  '1' szám → 1. kijelzőre '2' szám → 2. kijelzőre
  '3' szám → 3. kijelzőre

- 1 → szegmensek beállítása → és
  T1 tranzisztor bekapcsolása, T2 és T3 kikapcsolása →

- 2 → szegmensek beállítása → és
  T2 tranzisztor bekapcsolása, T1 és T3 kikapcsolása →

- 3 → szegmensek beállítása → és
  T3 tranzisztor bekapcsolása, T1 és T2 kikapcsolása →

A tranzisztorok bekapcsolása → bázisra 1-es szint (+5V)
A tranzisztorok kikapcsolása → bázisra 0-ás szint (0V)

19.5. Hétszegmenses kijelzők multiplex vezérlése

1. mintafeladat megoldása

```
// globális változók a szegmensek és nyomógombok lábakhoz rendelésére
const byte a7=13;     // 'a' szegmens  13. lábon van
const byte b7=12;      // 'b' szegmens  12. lábon
const byte c7=11;
const byte d7=10;
const byte e7=9;
const byte f7=8;
const byte g7=7;
const byte h7=6;
const byte stop=14;   // stop nyomógomb A0 lábon
const byte le=15;     // le nyomógomb  A1 lábon
const byte fel=16;    // fel nyomógomb  A2 lábon
const byte t1=4;      // T1 tranzisztor  4. lábon
const byte t2=3;      // T2 tranzisztor  3. lábon
const byte t3=2;
```

19.6. Hétszegmenses kijelzők multiplex vezérlése

1. mintafeladat folytatása

```
// Megmondjuk melyik lábak kimenetek (a hétszegmenses kijelzőt vezérlők)
// illetve melyik lábak bemenetek (amelyekre a nyomógombokat kötöttük)

void setup( )
 {
      pinMode(a7, OUTPUT);        //  'a7'  kimenet lesz
      pinMode(b7, OUTPUT);        //  'b7'  kimenet lesz
      pinMode(c7, OUTPUT);
      pinMode(d7, OUTPUT);
      pinMode(e7, OUTPUT);
      pinMode(f7, OUTPUT);
      pinMode(g7, OUTPUT);
      pinMode(h7, OUTPUT);
      pinMode(t1, OUTPUT);        //  't1'  kimenet lesz, 1. kijelző fel/le kapcsolása
      pinMode(t2, OUTPUT);
      pinMode(t3, OUTPUT);
      pinMode(fel, INPUT);        //  'fel' bemenet lesz (nyomógomb)
      pinMode(le, INPUT);
      pinMode(stop, INPUT);

      delay(2000);

 }
```

19.7. Hétszegmenses kijelzők multiplex vezérlése

1. mintafeladat folytatása

```
void loop()
 {
      // mindegyik kijelző lekapcsolása
      digitalWrite(t1, LOW);
      digitalWrite(t2, LOW);
      digitalWrite(t3, LOW);
      szam7szegm(1);          // 1-es szám a szegmensekre, függvény hívás !!
      digitalWrite(t1, HIGH);  // 1. kijelző  felkapcsolása
      delay(5);                // nagyon pici késleltetés !!
      // mindegyik kijelző lekapcsolása
      digitalWrite(t1, LOW);
      digitalWrite(t2, LOW);
      digitalWrite(t3, LOW);
      szam7szegm(2);          // 2-es szám a szegmensekre
      digitalWrite(t2, HIGH);  // 2. kijelző  felkapcsolása
      delay(5);                // nagyon pici késleltetés !!
      // mindegyik kijelző lekapcsolása
      digitalWrite(t1, LOW);
      digitalWrite(t2, LOW);
      digitalWrite(t3, LOW);
      szam7szegm(3);
      digitalWrite(t3, HIGH);
      delay(5);

 }
```

19.8. Hétszegmenses kijelzők multiplex vezérlése

1. mintafeladat folytatása

```
void szam7szegm(byte x)
{
  // a számokat megjelenítő függvény

        if(x==1)  {
              digitalWrite(a7, LOW);  digitalWrite(b7, HIGH);
              digitalWrite(c7, HIGH);   digitalWrite(d7, LOW);
              digitalWrite(e7, LOW);   digitalWrite(f7, LOW);
              digitalWrite(g7, LOW);   digitalWrite(h7, LOW);    }

        else if(x==2)  {
              digitalWrite(a7, HIGH);  digitalWrite(b7, HIGH);
              digitalWrite(c7, LOW);   digitalWrite(d7, HIGH);
              digitalWrite(e7, HIGH);   digitalWrite(f7, LOW);
              digitalWrite(g7, HIGH);   digitalWrite(h7, LOW);      }

        else if(x==3)  {
              digitalWrite(a7, HIGH);  digitalWrite(b7, HIGH);
              digitalWrite(c7, HIGH);   digitalWrite(d7, HIGH);
              digitalWrite(e7, LOW);   digitalWrite(f7, LOW);
              digitalWrite(g7, HIGH);   digitalWrite(h7, LOW);    }
}
```

19.9. Feladatok

Írj programokat az előző 3db multiplexelt 7 szegmenses kijelző vezérlésére

1. feladat

- folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-… DE sorban mindig másik kijelzőn ! → '0' az elsőn, '1' a másodikon, '2' a harmadikon, '3' az elsőn, '4' a másodikon, ….

2. feladat

- folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-… az első kijelzőn (1s-ig mindegyik)
- 'Stop' kapcsolót lenyomva → leáll a számlálás, felengedése után folytatódik !

3. feladat

- folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-… a második kijelzőn (1s-ig mindegyik)
- 'Fel' kapcsolót lenyomva → a számlálás a harmadik kijelzőn történik, felengedése után újra a középsőn folytatódik !
- 'Le' kapcsolót lenyomva → a számlálás az első kijelzőn történik, felengedése után újra a középsőn folytatódik !

19.10. Feladatok

Írj programokat az előző 3db multiplexelt 7 szegmenses kijelző vezérlésére

4. feladat

- 'Fel' kapcsolót lenyomva → folyamatos számlálás indul mindhárom kijelzőt használva
  → 000-001-002-….-010-011-….-998-999-000-001-...
- 'Stop' kapcsolót lenyomva → leáll a számlálás
- 'Le' kapcsolót lenyomva → folyamatos számlálás indul lefelé (visszafele)
  mindhárom kijelzőt használva

5. feladat

- 'Fel' kapcsolót lenyomva → egyet lép felfelé a számláló,
  mindhárom kijelzőt használjuk
- 'Le' kapcsolót lenyomva → egyet lép lefelé a számláló,
  mindhárom kijelzőt használjuk
- induláskor a számláló 000 állásban

Tipp a megoldáshoz:
Élfigyelés kell !!

20.1. Bitműveletek, ismétlés

Bitműveletek

- bitenkénti ÉS (AND): &
  szam1&szam2
  pl. 0b1011&0b1101 → 0b1001

- bitenkénti VAGY (OR): |
  szam1|szam2
  pl. 0b1010|0b0100 → 0b1110

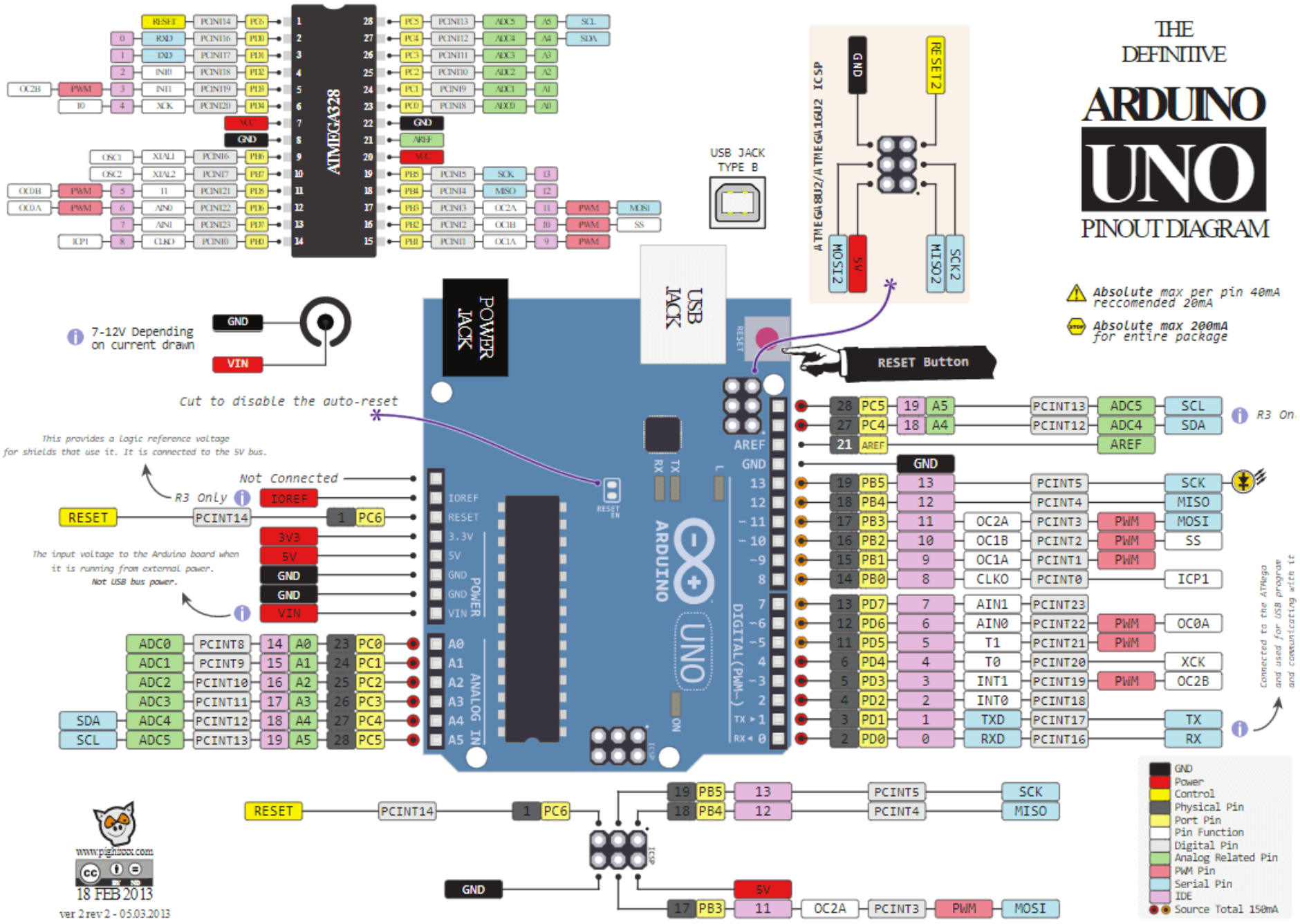
- bitenkénti NEM (NOT): ~
  ~szam1
  pl. ~0b1011 → 0b0100

- kizáró VAGY (XOR): ^
  szam1^szam2
  pl. 0b1110^b0100 → 0b1010

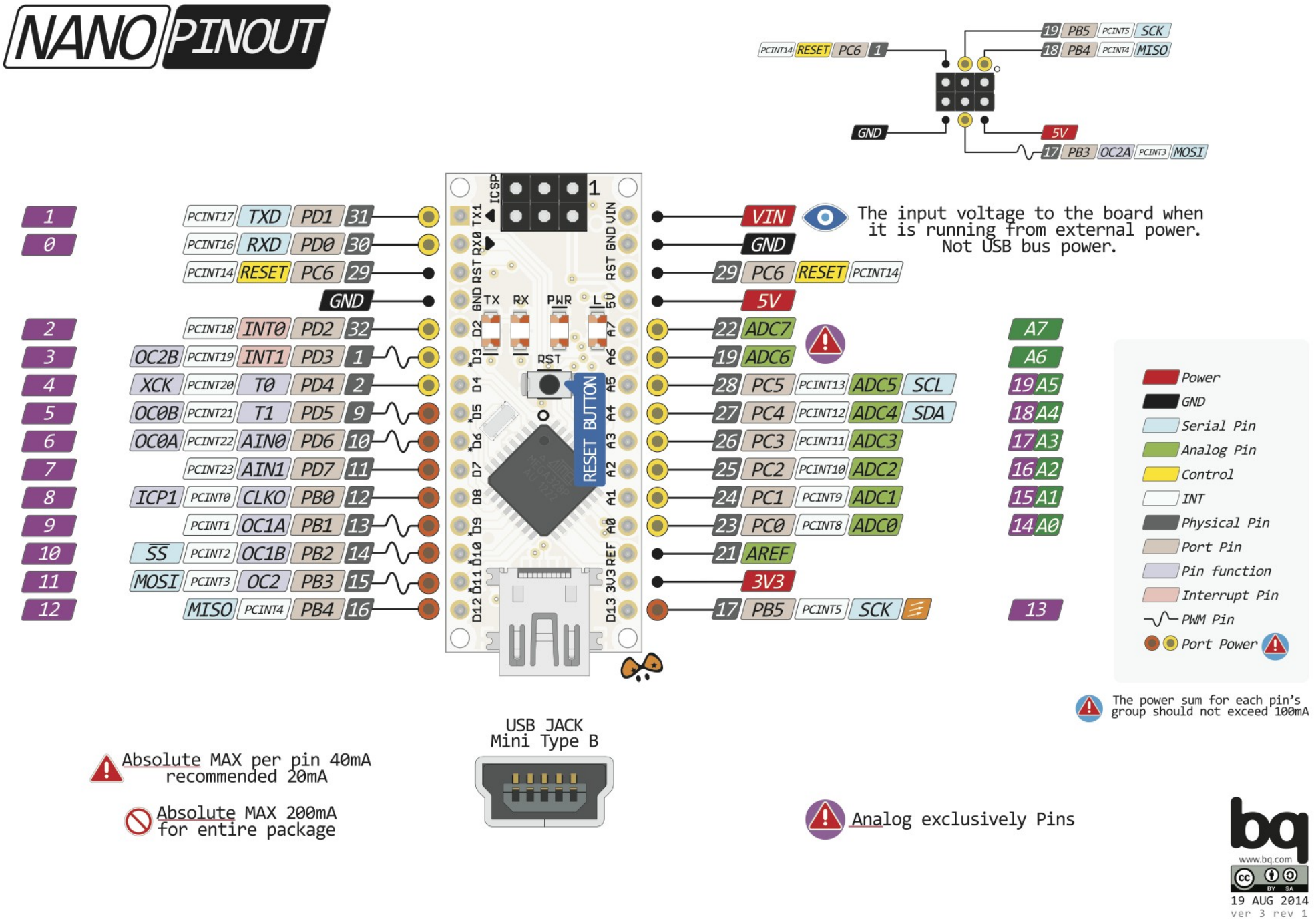
- eltolás balra (left shift) <<
  szam1<<x // eltolás ← x bittel
  pl. 0b00001001<<2 → 0b00100100

- eltolás jobbra (right shift) >>
  szam1>>x // eltolás → x bittel
  pl. 0b11001010>>3 → 0b00011001

20.2. Arduino Uno lábkiosztása



20.3. Arduino Nano lábkiosztása



20.4. Arduino lábak vezérlése

Atmega 328 perifériák

1. Az Arduino Uno és Nano modellekben található Atmega 328, Atmel AVR mikrovezérlő sok periféria áramkört tartalmaz:
- 23 db digitális bemenetet/kimenetet (programból kell állítani, hogy be vagy ki)
- 10 bites A/D átalakítót 6 (ill. 8) csatornával → 6 (8) bemenet lehet analóg bemenet
- 3 időzítőt/számlálót (timer), ebből 2db 8 bites, 1 pedig 16 bites
- 6 PWM kimenetet
- kommunikációs portokat, USART, SPI, I^2C

2. A legtöbb kivezetésnek/lábnak több funkciója is van ! → a kívánt funkciót programozással, a megfelelő vezérlő regiszterek bitjeinek állításával lehet kiválasztani

Digitális bemenetek, kimenetek

- A 23 db digitális bemenetet/kimenetet 8-as csoportokba van szervezve, és minden csoporthoz 3db vezérlő regiszter tartozik
  - PB0, PB1, PB2, ...PB5,(PB6,PB7) → DDRB, PORTB, PINB regiszterek
  - PC0, PC1, PC2, ...PC5,(PC6) → DDRC, PORTC, PINC regiszterek
  - PD0, PD1, PD2, ...PD7 → DDRD, PORTD, PIND regiszterek

- nem mindig használható mind a 23 digitális I/O funkcióra, mert közülük 1-3 egyéb célra használt ! (reset, külső órajel csatl.)

20.5. Arduino lábak vezérlése

DDRx, PORTx, PINx regiszterek

- DDRx regiszter bitjei állítják be az irányokat,
  ha 1 → az adott láb kimenet, ha 0 → akkor bemenet
- PORTx regiszter bitjein keresztül a hozzá rendelt lábakra lehet írni
- PINx regiszterbe kerülnek a digitális bemenetként beállított lábak értéki → innen tudjuk kiolvasni

DDRD

7	6	5	4	3	2	1	0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0

PD0, PD1,PD2, ...PD7 lábak irány beállítása,
1→ kimenet, 0→ bemenet

PORTD

7	6	5	4	3	2	1	0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0

PD0, PD1,PD2, ...PD7 lábak írása

PIND

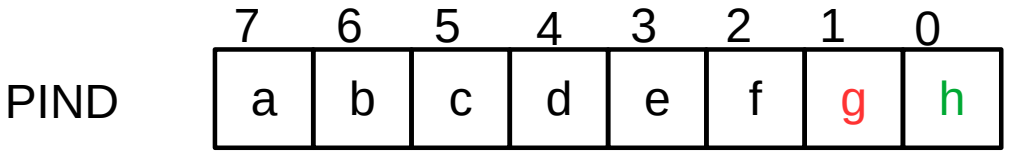
7	6	5	4	3	2	1	0
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0

PD0, PD1,PD2, ...PD7 lábak olvasása

DDRC, PORTC, PINC és DDRB, PORTB, PINB hasonlóan

20.6. A bemenetek lekérdezése

Bemenet értékének lekérdezése bitenkénti ÉS művelettel



PIND g bit vizsgálható a következő művelettel:

if(PIND&0b00000010)

→ igaz, ha g=1 hamis, ha g=0

PIND h bit vizsgálható a következő művelettel:

if(PIND&0b00000001)

→ igaz, ha h=1 hamis, ha h=0

mert:

a b c d e f g h

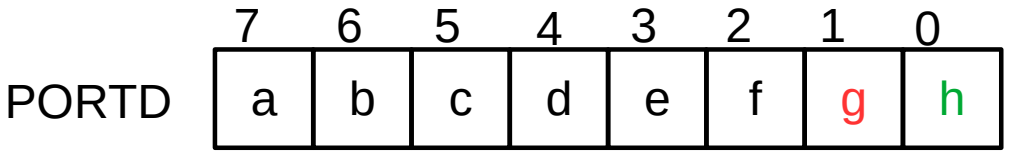
& 0 0 0 0 0 0 1 0

= 0 0 0 0 0 0 g 0 → 0, ha g=0 vagy 2, ha g=1

de minden 0-tól különböző szám igaz logikai értékké konvertálódik !!

20.7. A kimenetek állítása

Kimenet értékének állítása bitenkénti ÉS, VAGY művelettel



PORTD g bit törlése (0-ba állítása) a következő művelettel:

PORTD=PORTD&0b11111101)

→ a többi bit marad az ami volt !!

mert:

a b c d e f g h

& 1 1 1 1 1 1 0 1

= a b c d e f 0 h

Mert a*1=a …. b*1=b ….

de g*0=0

PORTD h bit 1-be állítása a következő művelettel:

PORTD=PORTD|0b00000001)

→ a többi bit marad az ami volt !!

mert:

a b c d e f g h

| 0 0 0 0 0 0 0 1

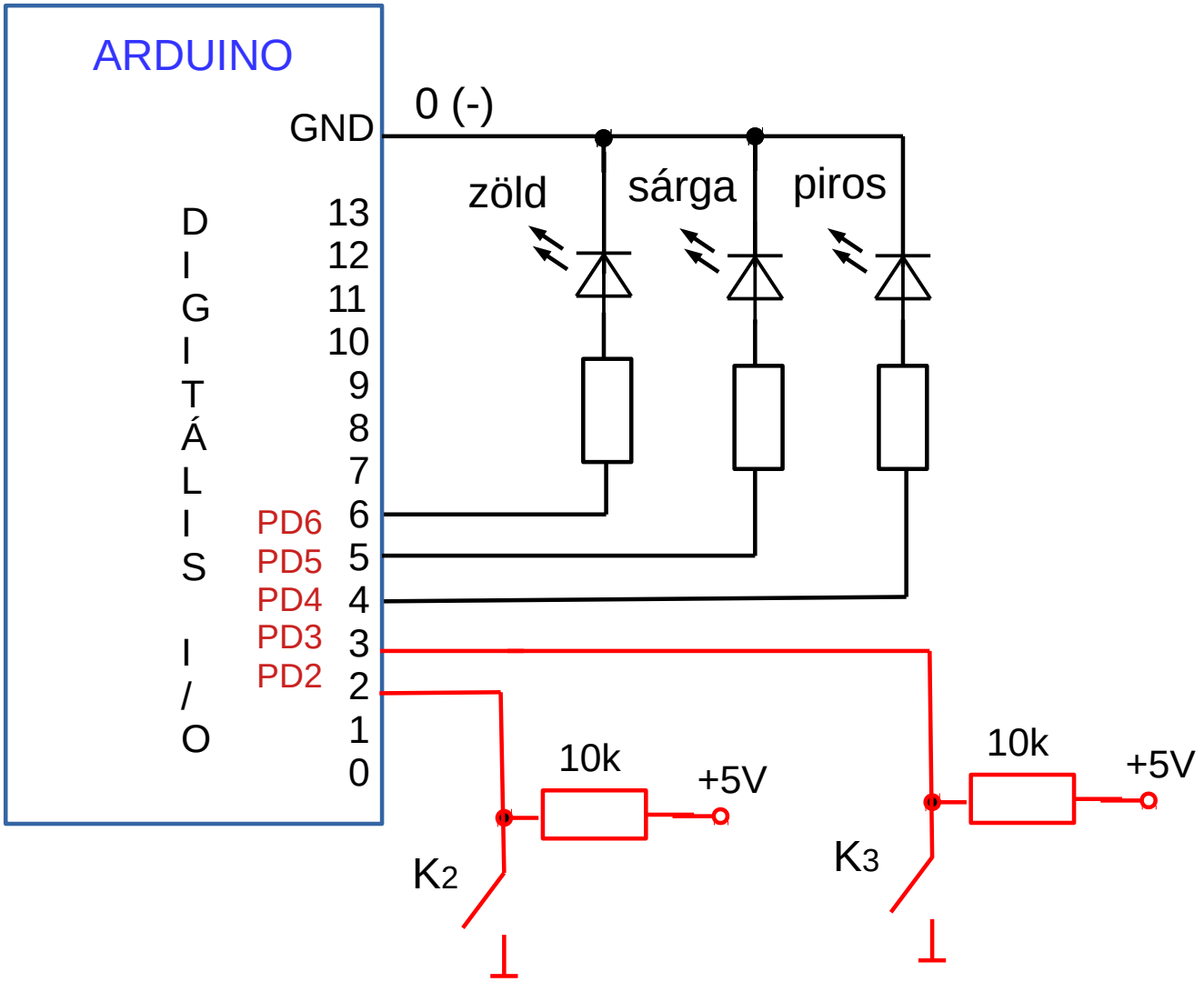
= a b c d e f g 1

Mert a|0=a …. b|0=b ….

de h|1=1

20.8. Vezérlés regiszterekkel

LED-ek vezérlése, hardver



20.9. Vezérlés regiszterekkel

1. mintafeladat

```
// Először felkapcsoljuk a piros LED-et
// majd a zöld LED villog (1 másodpercenként)
   void setup( )
    {
            DDRD=0b11110000;       //  PD7, PD6, PD5,PD4 láb  kimenet lesz
                                          //  PD3, PD2, PD1,PD0 láb  bemenet lesz

            PORTD=PORTD&0b10001111;       //  mindegyik LED-et  lekapcsoljuk
            PORTD=PORTD |0b00010000;      //  piros LED-et  fel
            delay(2000);

    }

   void loop( )
    {
            PORTD=PORTD |0b01000000;      //  zöld LED-et  fel
            delay(1000);
            PORTD=PORTD&0b10111111;      //  zöld LED-et le
            delay(1000);
    }
```

20.10. Vezérlés regiszterekkel

2. mintafeladat

```
// Az előző feladat megoldása másképp (XOR használatával)
// Először felkapcsoljuk a piros LED-et
// majd a zöld LED villog (1 másodpercenként)

void setup( )
{
        DDRD=0b11110000;       //  PD7, PD6, PD5,PD4 láb  kimenet lesz
                                //  PD3, PD2, PD1,PD0 láb  bemenet lesz

        PORTD=PORTD&0b10001111;       //  mindegyik LED-et  lekapcsoljuk
        PORTD=PORTD |0b00010000;      //  piros LED-et  fel
        delay(2000);
}

void loop( )
{
        PORTD=PORTD ^ 0b01000000;     //  zöld LED-et  invertáljuk !!
        delay(1000);
}
```

20.11. Vezérlés regiszterekkel

3. mintafeladat

```
// K2-t lenyomjuk →  a sárga LED villog (1 másodpercenként)
// K3-t lenyomjuk →  a zöld LED villog (1 másodpercenként)
byte melyik=0;   // 32 – sárga  0b00100000,  64 – zöld  0b01000000
void setup( )
 {
        DDRD=0b11110000;       //  PD7, PD6, PD5,PD4 láb  kimenet lesz
                                //  PD3, PD2, PD1,PD0 láb  bemenet lesz
        PORTD=PORTD&0b10001111;       //  mindegyik LED-et  lekapcsoljuk
        delay(2000);

 }
void loop( )
 {
        if(!(PIND&0b00001000))   melyik=64;     // K3 lenyomva → zöld
        if(!(PIND&0b00000100))   melyik=32;     // K2 lenyomva → sárga
        PORTD=PORTD ^ melyik;       //  XOR, invertál ahol 1-es értékű a 'melyik' változó
        delay(500);

 }
```

21.1. Léptető motorok

1. Léptető motorok

- szénkefe nélküli egyenáramú motorok, amelyek adott számú lépésből tesznek meg egy fordulatot → általában néhány fokos szögelfordulás jellemző lépésenként (pl. 1,8° → 200 lépés egy körülforgás)
- vezérlésük digitális → a tekercseikre megfelelő sorrendben kell feszültséget kapcsolni (majd lekapcsolni)
- elég pontosan egy adott pozícióba lehet forgatni ! (1-5%)
- fordulatszámuk viszont elég alacsony lehet, néhány száz fordulat percenként (pl. 500)

2. Léptető motor típusok

Felépítés alapján lehet:

- változó mágneses ellenállású (VR) → a forgórész fogazott, lágy mágneses anyag
- állandó mágneses (PM) → a forgórészen állandó mágnes(ek)
- hibrid motor (HB) → az előbbi kettő kombinációja

Vezérlés szempontjából:

→ unipoláris, az álló rész minden pólusán két tekercs (egy tekercs középleágazással)

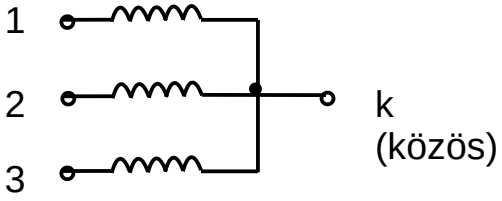
→ bipoláris, minden fázishoz egy tekercs → az egyes tekercseken az áram irányát is változtatni kell ! → bonyolultabb vezérlés

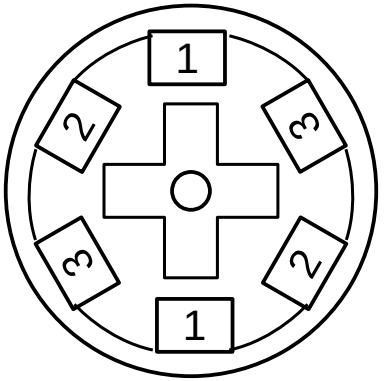
Nagyon jó, szemléletes leírás található a következő oldalon:
http://qtp.hu/elektro/leptetomotor_mukodese.php

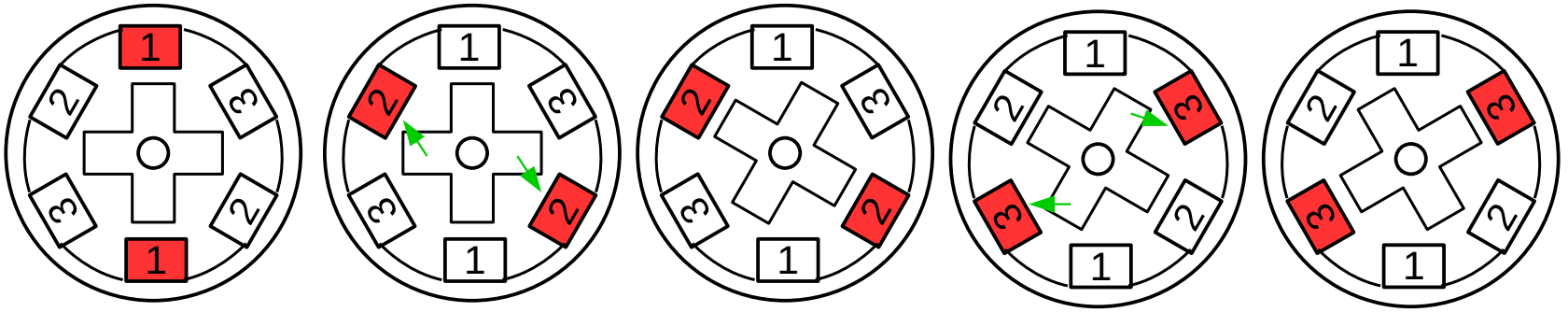
21.2. Léptető motorok

3. Változó mágneses ellenállású léptető motorok

- variable reluctance (VR), a forgórész fogazott, lágy mágneses anyag
- az álló részen 3 darab vezérelhető tekercs, 4 kivezetéssel (vagy 5 tekercs, 6 kivezetéssel)
- a tekercsekre megfelelő sorrendben feszültséget kapcsolva (1,2,3,1,2,3,1,...)
  lépeget az egyik irányba, a sorrendet megfordítva (3,2,1,3,2,1,3,...) a másik irányba





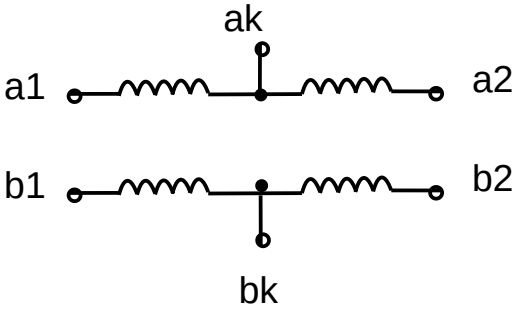


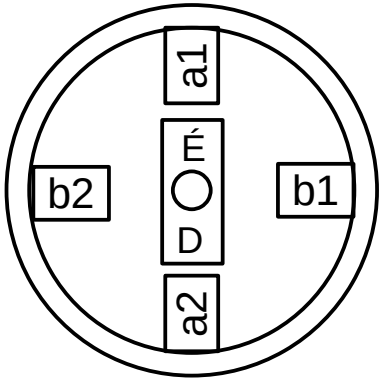
21.3. Léptető motorok

4. Unipoláris léptető motor

- állórészen általában 4 darab vezérelhető tekercs → 5 vagy 6 kivezetéssel
- forgórészen állandó mágnes(ek) → PM motor, vagy esetleg plusz fogazás is →HB motor
- többféle vezérlés lehetséges, de az a lényeg hogy a tekercsekre megfelelő sorrendben feszültséget kapcsolva lépeget az egyik, a sorrendet megfordítva a másik irányba

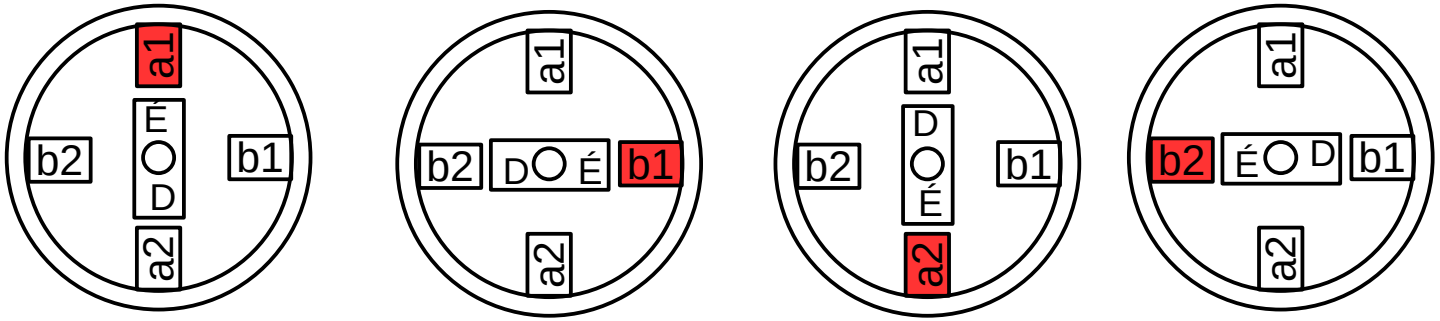
Unipoláris, 6 vezetékes
az 5 vezetékes lényegében ugyanez, csak a két közös vezeték (ak és bk) össze van kötve





Vezérlési módok:
- egyfázisú (hullám)
- féllépéses (half stepping)
- teljes lépéses (full stepping)
- mikro lépéses (micro stepping)

egyfázisú (hullám) vezérlés
- a 4 tekercsre egyenként, sorban kapcsolunk feszültséget → a1-b1-a2-b2-a1-b1-a2-b2-a1-...



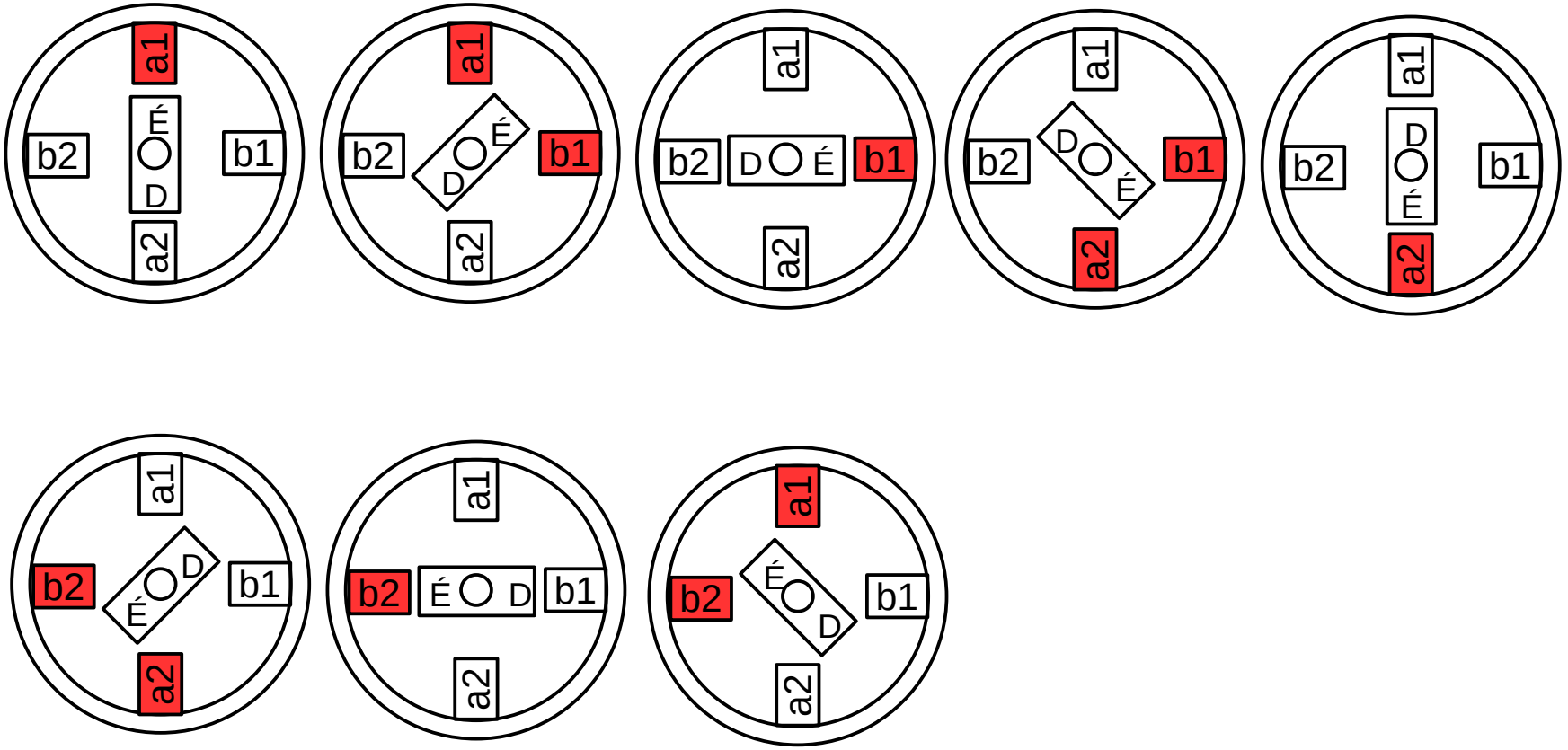
21.4. Léptető motorok

4. Unipoláris léptető motor

féllépéses (half stepping) vezérlés

-az egyfázisú vezérlés lépései közé plusz lépéseket iktatunk be úgy, hogy egyszerre két tekercsre kapcsolunk feszültséget → a lépések száma megduplázódik !

Tehát a vezérlés → a1 – a1,b1 – b1 – b1,a2 – a2 – a2,b2 – b2 – b2,a1 – a1,b1 – b1 – ...

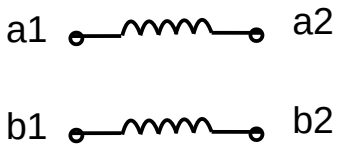


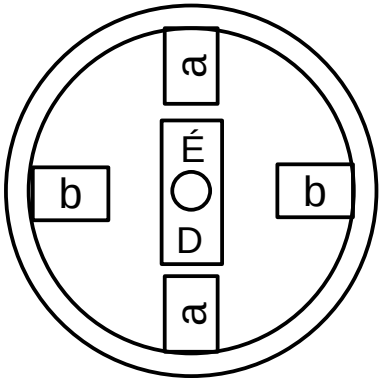
21.5. Léptető motorok

5. Bipoláris léptető motor

- állórészen általában 2 darab vezérelhető tekercs, 4 kivezetés
  vagy lehet több tekercs is → pl. 4 tekercs (8 kivezetés)
- forgórészen állandó mágnes(ek) → PM motor, vagy esetleg plusz fogazás is →HB motor
- az egyes tekercseken az áram irányát is változtatni kell ! → H-híd vezérlés

Bipoláris, 4 vezetékes

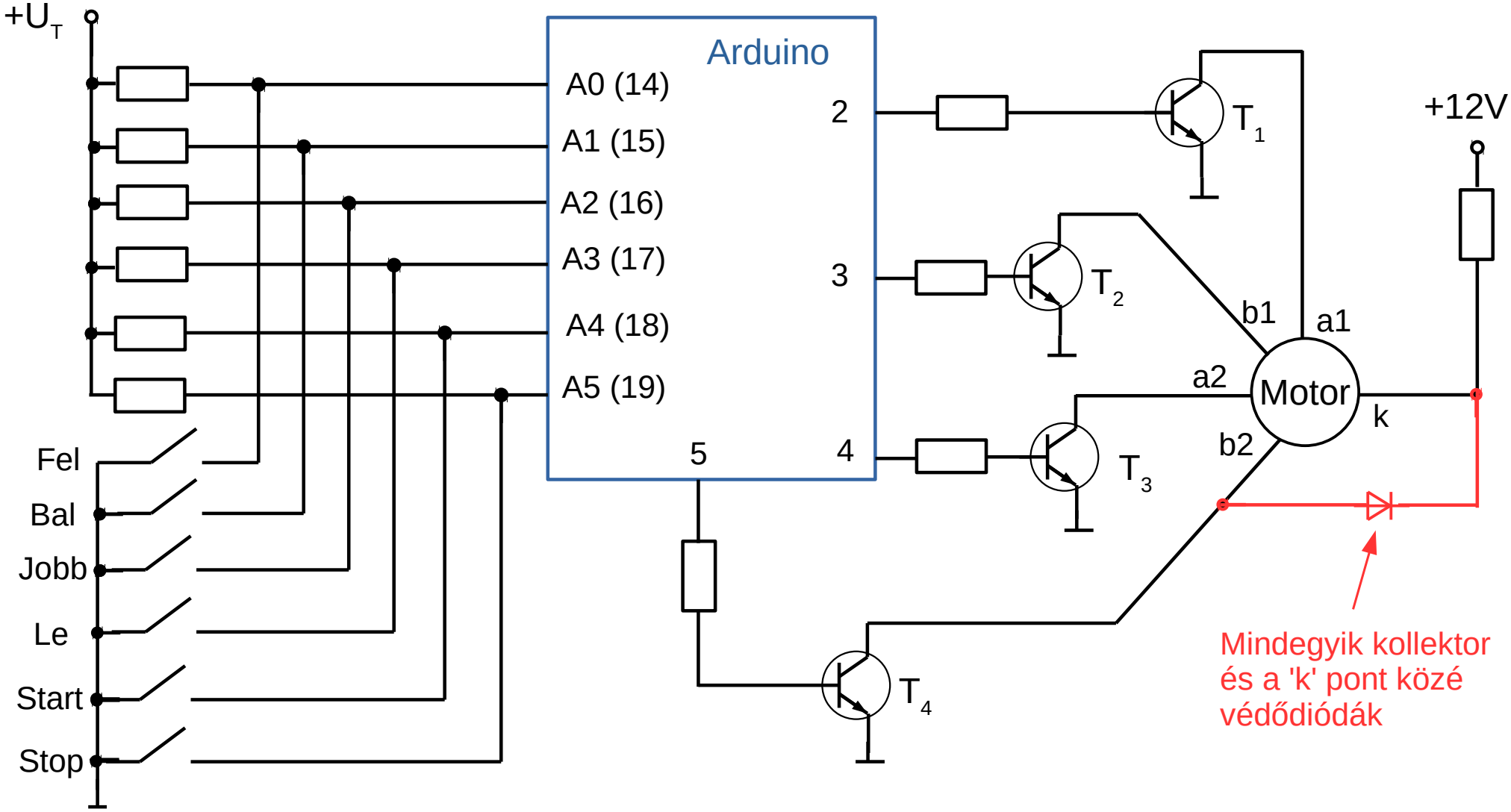




21.6. Unipoláris léptető motor vezérlése

A hardver

- a motor 4 tekercsére tranzisztorokkal kapcsolunk feszültséget, mert jellemzően 5V-nál nagyobb feszültséget igényelnek, és nagy az áram felvételük is !
- tehát a négy tranzisztort kell sorban kapcsolgatni (a bázisukra 1 ill. 0 szint kapcsolásával)



21.7. Unipoláris léptető motor vezérlése

1. mintafeladat

a 2. 3. 4. és 5. digitális kimenetekre kapcsolt unipoláris léptető motor vezérlése a következőképpen:

- A JOBB kapcsolót nyomva tartva → a motor lépkedjen 'jobbra' (óramutató járásával megegyezően)
- A BAL kapcsolót nyomva tartva → a motor lépkedjen ellenkező irányba

A kapcsolók a 14. 15. … 19. lábakra vannak kötve !! →
Ezek analóg bemenetek is lehetnek (A0, A1, … A5) !!

A motor léptetése jobbra → feszültség (1-es szint) kapcsolása sorban egyenként
a1,b1,a2,b2 tekercsekre → 2,3,4,5 lábakra

A motor léptetése balra → feszültség (1-es szint) kapcsolása sorban egyenként
b2,a2,b1,a1 tekercsekre → 5,4,3,2 lábakra

Tehát egyfázisú, hullám vezérlést használunk
- egyik irány → a1-b1-a2-b2-a1-b1-a2-b2-a1-…
- másik irány → b2-a2-b1-a1-b2-a2-b1-a1-b2-…

21.8. Unipoláris léptető motor vezérlése

1. mintafeladat megoldása

```
// globális változók a motorvezérlő kimenetek és nyomógombok lábakhoz rendelésére
const byte a1=2;       // 1. motor tekercs
const byte b1=3;       // 2. motor tekercs
const byte a2=4;       // 3. motor tekercs
const byte b2=5;       // 4. motor tekercs
const byte fel=14;     // fel nyomógomb  A0 lábon
const byte bal=15;     // bal nyomógomb  A1 lábon
const byte jobb=16;    // jobb nyomógomb  A2 lábon
const byte le=17;      // le nyomógomb  A3 lábon
const byte stop=19;    // stop nyomógomb A5 lábon
byte i=0;              // változó a tekercsek kapcsolgatására

void setup( )
 {
       pinMode(a1, OUTPUT);    //  'a1'  kimenet lesz
       pinMode(b1, OUTPUT);    //  'b1'  kimenet lesz
       pinMode(a2, OUTPUT);
       pinMode(b2, OUTPUT);
       pinMode(fel, INPUT);        //  'fel' bemenet lesz (nyomógomb)
       pinMode(bal, INPUT);        //  'bal' bemenet lesz (nyomógomb)
       pinMode(jobb, INPUT);
       pinMode(le, INPUT);
       pinMode(stop, INPUT);
       delay(2000);
       leptet(0);   delay(200);  leptet(1);  delay(200);
       leptet(2);   delay(200);  leptet(3);  delay(200);
       leptet(0);   // alaphelyzetbe állás
       delay(500);
 }
```

21.9. Unipoláris léptető motor vezérlése

1. mintafeladat folytatása

```
void loop()
 {
      while(digitalRead(jobb)==0)    // motor jobbra ha 'JOBB' gomb  lenyomva
      {
            i++;
            if(i>3) i=0;
            leptet(i);                     // léptető függvény hívása
            delay(500);
      }

      while(digitalRead(bal)==0)    // motor balra ha 'BAL' gomb  lenyomva
      {
            if(i>0) i--;
            else i=3;
            leptet(i);                     // léptető függvény hívása
            delay(500);
      }

 }
```

21.10. Unipoláris léptető motor vezérlése

1. mintafeladat folytatása

```
// a léptető motort vezérlő függvény
void leptet(byte x)
{
        if(x==0)   // a1 be
         {
                digitalWrite(a1, HIGH);  digitalWrite(b1, LOW);
                digitalWrite(a2, LOW);   digitalWrite(b2, LOW);
         }
        if(x==1)  // b1 be
         {
                digitalWrite(a1, LOW);  digitalWrite(b1, HIGH);
                digitalWrite(a2, LOW);   digitalWrite(b2, LOW);
         }
        if(x==2)   // a2 be
         {
                digitalWrite(a1, LOW);  digitalWrite(b1, LOW);
                digitalWrite(a2, HIGH);   digitalWrite(b2, LOW);
         }
        if(x==3)  // b2 be
         {
                digitalWrite(a1, LOW);  digitalWrite(b1, LOW);
                digitalWrite(a2, LOW);   digitalWrite(b2, HIGH);
         }
}
```

21.11. Unipoláris léptető motor vezérlése

2. mintafeladat

a 2. 3. 4. és 5. digitális kimenetekre kapcsolt unipoláris léptető motor vezérlése a következőképpen:

- A JOBB kapcsolót lenyomva → a motor lépkedjen 'jobbra' (óramutató járásával megegyezően)
- A STOP kapcsolót lenyomva → a motor álljon le

Célszerű a motor állapotát külön változóban (vagy változókban) tárolni →
jelenleg egy változó is elég →
a motorbe változót használjuk:
- 1 értékű, ha forognia kell (JOBB kapcsolót megnyomtuk)
- 0 értékű, ha állnia kell (még el sem indítottuk vagy a STOP kapcsolót megnyomtuk)

A motor léptetése jobbra →
egyfázisú, hullám vezérlést használunk → a1-b1-a2-b2-a1-b1-a2-b2-a1-…

21.12. Unipoláris léptető motor vezérlése

2. mintafeladat megoldása

```
// globális változók a motorvezérlő kimenetek és nyomógombok lábakhoz rendelésére
const byte a1=2;       // 1. motor tekercs
const byte b1=3;       // 2. motor tekercs
const byte a2=4;       // 3. motor tekercs
const byte b2=5;       // 4. motor tekercs
const byte fel=14;     // fel nyomógomb  A0 lábon
const byte bal=15;     // bal nyomógomb  A1 lábon
const byte jobb=16;    // jobb nyomógomb  A2 lábon
const byte le=17;      // le nyomógomb  A3 lábon
const byte stop=19;    // stop nyomógomb A5 lábon
byte i=0;              // változó a tekercsek kapcsolgatására
byte motorbe=0;        // változó a motor állapotának tárolására, 0-áll   1-forog
void setup( )
 {
       pinMode(a1, OUTPUT);    //  'a1'  kimenet lesz
       pinMode(b1, OUTPUT);    //  'b1'  kimenet lesz
       pinMode(a2, OUTPUT);
       pinMode(b2, OUTPUT);
       pinMode(fel, INPUT);        //  'fel' bemenet lesz (nyomógomb)
       pinMode(bal, INPUT);        //  'bal' bemenet lesz (nyomógomb)
       pinMode(jobb, INPUT);
       pinMode(le, INPUT);
       pinMode(stop, INPUT);
       leptet(0);   delay(200);  leptet(1);  delay(200);
       leptet(2);   delay(200);  leptet(3);  delay(200);
       leptet(0);   // alaphelyzetbe állás
       delay(500);
 }
```

21.13. Unipoláris léptető motor vezérlése

2. mintafeladat folytatása

```
void loop()
 {
      if(motorbe==1)        // motor forog jobbra
      {
            i++;
            if(i>3) i=0;
            leptet(i);                    // léptető függvény hívása
      }

      if(digitalRead(jobb)==0)        // motor indítása jobbra, ha 'JOBB' gomb  lenyomva
            {    motorbe=1; }
      if(digitalRead(stop)==0)        // motor leállítása, ha 'STOP' gomb  lenyomva
            {    motorbe=0; }

      delay(500);

 }
```

21.14. Unipoláris léptető motor vezérlése

2. mintafeladat folytatása

```
// a léptető motort vezérlő függvény
void leptet(byte x)
{
        if(x==0)   // a1 be
         {
                digitalWrite(a1, HIGH);  digitalWrite(b1, LOW);
                digitalWrite(a2, LOW);   digitalWrite(b2, LOW);
         }
        if(x==1)  // b1 be
         {
                digitalWrite(a1, LOW);  digitalWrite(b1, HIGH);
                digitalWrite(a2, LOW);   digitalWrite(b2, LOW);
         }
        if(x==2)   // a2 be
         {
                digitalWrite(a1, LOW);  digitalWrite(b1, LOW);
                digitalWrite(a2, HIGH);   digitalWrite(b2, LOW);
         }
        if(x==3)  // b2 be
         {
                digitalWrite(a1, LOW);  digitalWrite(b1, LOW);
                digitalWrite(a2, LOW);   digitalWrite(b2, HIGH);
         }
}
```

21.15. Unipoláris léptető motor vezérlése

<u>3. mintafeladat</u>

a 2. 3. 4. és 5. digitális kimenetekre kapcsolt unipoláris léptető motor vezérlése a következőképpen:

- a JOBB kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra' (óramutató járásával megegyezően)
- a STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van
- a FEL kapcsoló lenyomása után → a motor gyorsabban lépkedjen (kb. 3-szoros sebességgel)
- a LE kapcsoló lenyomása után → a motor térjen vissza az alap sebességhez

Célszerű a motor állapotát külön változóban (vagy változókban) tárolni → jelenleg két változó célszerű→

motorbe változó

- 1 értékű, ha forognia kell (JOBB kapcsolót megnyomtuk)
- 0 értékű, ha állnia kell (még el sem indítottuk vagy a STOP kapcsolót megnyomtuk)

gyors változó

- 1 értékű, ha nagyobb sebességű
- 0 értékű, ha lassú

A sebességet legegyszerűbben a késleltetés módosításával tudjuk megváltoztatni →

nagy késleltetés → lassú

Kis késleltetés → gyors

21.16. Unipoláris léptető motor vezérlése

3. mintafeladat folytatása

A ’setup’ és ’leptet’ függvények nem változnak, ugyanazok mint az előző két mintafeladatban. És a globális konstansok, változók is ugyanazok, csak egy új globális változó kell, és a ’loop’ függvény változik →

```
...
byte gyors=0;          // sebesség,  1-gyors   0-lassú
…
void loop()
 {
      if(motorbe==1)        // motor forog jobbra
       {
            i++;
            if(i>3) i=0;
            leptet(i);                    // léptető függvény hívása
       }
      if(digitalRead(jobb)==0)  motorbe=1;      // motor indítása jobbra
      if(digitalRead(stop)==0)  motorbe=0;      // motor leállítása
      if(digitalRead(fel)==0)   gyors=1;        // gyorsítás
      if(digitalRead(le)==0)    gyors=0;        // lassítás
      if(gyors==1)  delay(200);             // gyors → kis késleltetés
      else delay(600);                      // lassú → nagy késleltetés

 }
```

21.17. Feladatok

Írj programokat unipoláris léptető motor vezérlésére
(a kapcsolás az előzővel megegyező)

1. feladat

- a program indulásakor a motor lépjen 10-et jobbra gyorsan (300ms késleltetés), majd 5-öt balra lassan (1000ms késleltetés)

2. feladat

- a program indulásakor a motor lépjen 10-et jobbra gyorsan (300ms késleltetés),
- a BAL kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'balra'
- a STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van

3. feladat

- a JOBB kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra'
- a STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van
- a FEL kapcsoló lenyomása után → a motor gyorsabban lépkedjen (kb. 4-szeres sebességgel → 0,5 másodpercenkénti lépések)
- a LE kapcsoló lenyomása után → a motor térjen vissza az alap sebességhez, 2 másodpercenként lépjen
- Megoldandó feladat: a nagy késleltetések ellenére, reagáljon gyorsan a kapcsolók lenyomására

21.18. Feladatok

Írj programokat unipoláris léptető motor vezérlésére
(a kapcsolás az előzővel megegyező)

4. feladat

- a FEL kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra'
- a LE kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'balra'
- a STOP kapcsoló lenyomására → a motor álljon meg stabilan a pozícióban
- a JOBB kapcsoló minden lenyomására → a motor lépjen egyet 'jobbra'

Késleltetés a lépések között legyen 400ms

5. feladat

- a program indulásakor várjon a START kapcsoló lenyomására ! → ezután a motor lépjen 20-at jobbra gyorsan (250ms késleltetés), majd álljon le
- ezután
  - a FEL kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra'
  - a LE kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'balra'
  - a STOP kapcsoló lenyomására → a motor álljon meg stabilan a pozícióban
  - a JOBB kapcsoló minden lenyomására → a motor lépjen egyet 'jobbra'
  - a BAL kapcsoló minden lenyomására → a motor lépjen egyet 'balra'

    Késleltetés a lépések között legyen 400ms

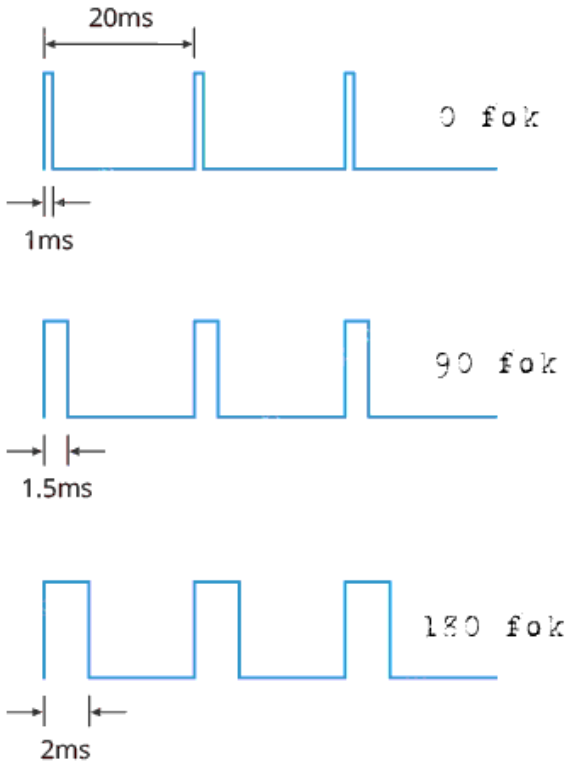
22.1. Szervó motor

PWM ismét

- PWM (Pulse Width Modulation → impulzus szélesség moduláció)
- Egy négyszögjel kitöltési tényezőjét változtatjuk ! (mennyi ideig van magas szinten ill. alacsony szinten)
- A négyszögjel frekvenciája és amplitúdója nem változik.

Szervó motor

- Bonyolult felépítésű DC motor, saját vezérlő áramkörrel egybeépítve
- pozíciója (szögelfordulása) nagyon pontosan beállítható
- vezérlése egy vezetéken, digitálisan, PWM jellel történik
- a leggyakoribb típus 0-180˚-os tartományban vezérelhető
- tipikusan a PWM jel periódus ideje 20ms, és a középálláshoz (90 fok) 1,5 ms impulzusidő tartozik



22.2. Szervó motor kezelő könyvtár

„Servo.h”

Először a programunk elején ezt a header állományt be kell importálnunk, hogy elérjük a szükséges objektumokat, függvényeket !

```
#include <Servo.h>
```

„Servo” objektum

Ezután létre kell hozni egy Szervó vezérlő objektumot,
Az objektum metódusaival (lényegében függvények) tudjuk vezérelni a motort !

```
Servo servo;      // ezután a „servo” nevű objektum metódusait kell használnunk
```

„attach” metódus

Ezzel adjuk meg, hogy a motort melyik Arduino-s lábbal akarjuk vezérelni.
PWM-et nem kell feltétlen tudnia a lábnak (szoftveresen megoldja, bár ilyenkor a 9. és 10. láb analóg (PWM) kimenetként nem használható !!

```
servo.attach(pin);
servo.attach(pin,min,max);   // a   min. és max. impulzusidőket is megadhatjuk
                             //  mikroszekundumban ( 0 és 180 fokhoz tartoznak)
                            // speciálisabb szervók esetén

pl.  servo.attach(9);        // 9. lábbal vezéreljük
```

22.3. Szervó motor kezelő könyvtár

„write” metódus

A vezérlés: megadjuk milyen szögelfordulásba álljon be.

```
servo.write(angle);          // pozíció fokban
pl.  servo.write(120);       // 120°-os pozíció
```

„read” metódus

A motor pozíciójának lekérdezése. Fokban adja vissza (0 – 180 fok)

```
servo.read();
pl.  poz = servo.read();
```

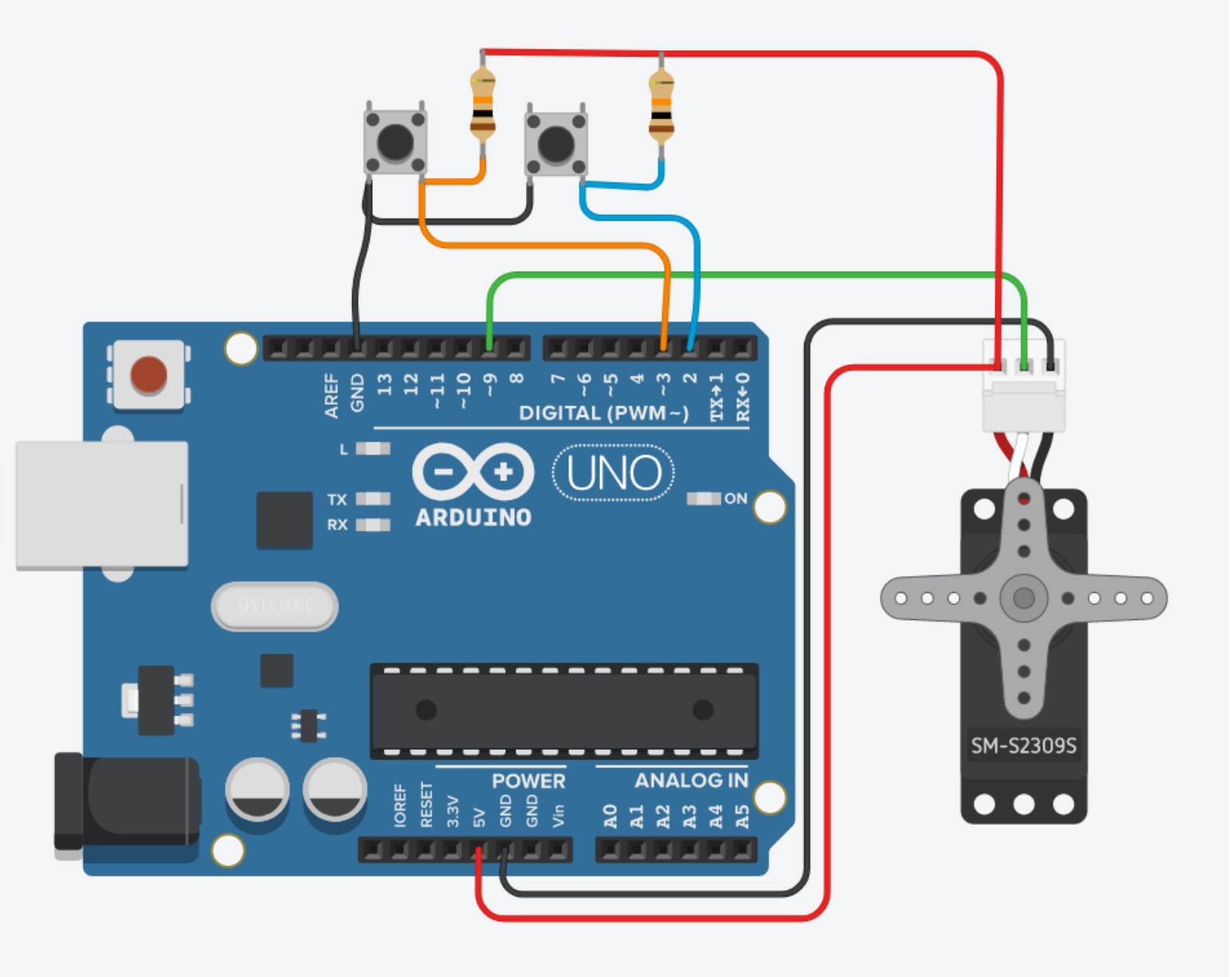
„detach” metódus

A servo objektumot lecsatolja a lábról.
Minden servo objektumot le kell csatolni ha analóg kimenetkent akarjuk
a 9. és 10. lábat használni !!

```
servo.detach();
```

22.4. Szervó motor vezérlés

A hardver



22.5. Szervó motor vezérlés

1. mintafeladat

Forgás oda-vissza automatikusan, 0 és 180 fok között

```
#include <Servo.h>          //  a servo könyvtár betöltése

const byte servoPin = 9;          //  változó rendelése a 9-es kimenethez

Servo servo;                      //  a szervó objektum létrehozása
int angle = 0;                    // szervó pozíció változója és értékadása


void setup()
{
      servo.attach(servoPin);   // szervó vezérlő  összerendelése a digitális kimenettel
}

void loop()
{
    jobbra();
    delay(200);
    balra();
    delay(200);
}
```

22.6. Szervó motor vezérlés

1. mintafeladat folytatása

```
void jobbra()
{
    // 0-tól 180 fokig mozdítja el a szervót
        for(angle = 0; angle < 180; angle++)
        {
                servo.write(angle);          //kirakja az értéket a szervóra
                delay(15);
        }
}

void balra()
{
   // vissza számlálás 180-tól 0-ig
        for(angle = 180; angle > 0; angle--)
        {
                servo.write(angle);          //kiíratja a szervóra
                delay(15);
        }
}
```

22.7. Szervó motor vezérlés

2. mintafeladat

Forgás jobbra, vagy balra, a két nyomógombot lenyomva

```
#include <Servo.h>          //  a servo könyvtár betöltése

const byte servoPin = 9;        //  változó rendelése a 9-es kimenethez
const byte  jobbny=2;           // nyomógomb, jobbra
const byte  balny=3;            // nyomógomb, balra
Servo servo;                    //  a szervó objektum létrehozása
int angle = 90;                 // szervó pozíció változója, középső pozícióból indulunk
int i=0;                        // ciklus változó


void setup()
{
      pinMode(jobbny,INPUT);
      pinMode(balny,INPUT);
      servo.attach(servoPin);         // szervó vezérlő  összerendelése a digitális kimenettel
}
```

22.8. Szervó motor vezérlés

2. mintafeladat folytatása

```
void loop()
{
	if(i%5==0)
	{
		if(!digitalRead(jobbny))		//  léptetés jobbra
		{
			if(angle<180)  angle++;
		}

		if(!digitalRead(balny))		//  léptetés balra
		{
			if(angle>0)  angle--;
		}
	}

	servo.write(angle);		//kirakja az értéket a szervóra
	delay(15);

	i++;
}
```

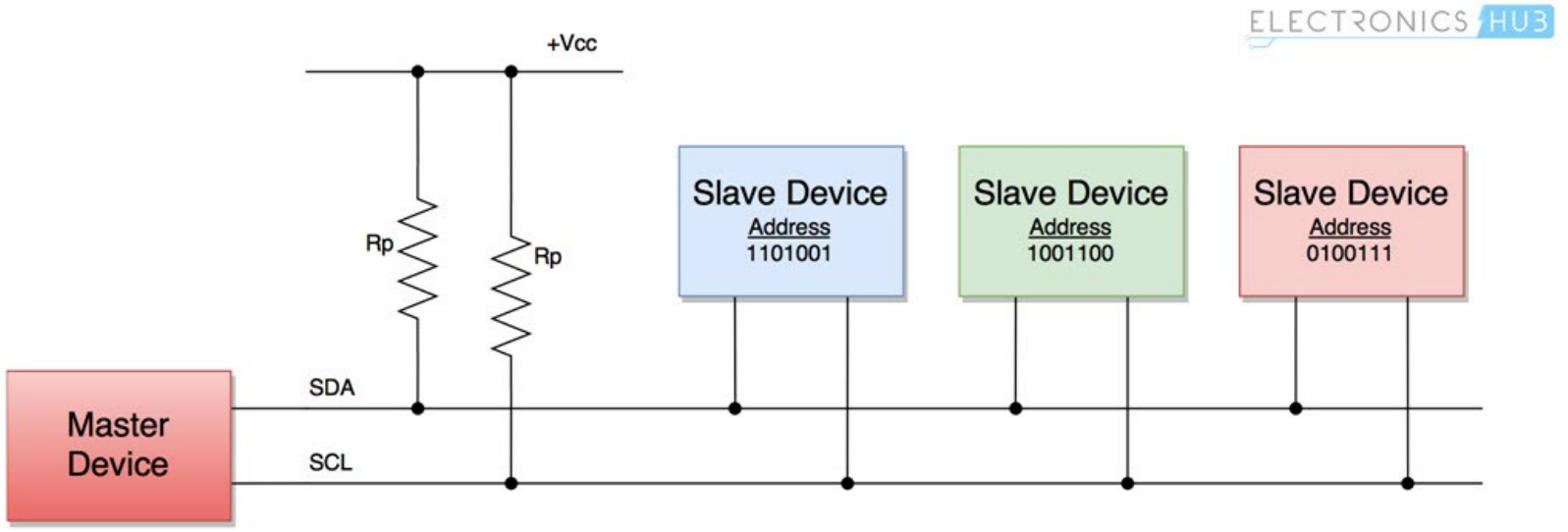
23.1. I^2C kommunikáció

I^2C

- Inter-Integrated Circuit
- szinkron soros kommunikációs protokoll, mikrovezérlők és perifériák, kijelzők, szenzorok közötti adatcseréhez
- másik elnevezése: TWI (Two Wire Interface)
- átviteli sebességek: 100kbit/s (Standard Mode), 400kbit/s (Fast Mode), 1Mbit/s (Fast Mode Plus), 3,4Mbit/s (High Speed Mode),

I^2C bus

- két vezeték: adat (SDA, serial data) + órajel (SCL, serial clock)
- open-drain kimenetek → felhúzó ellenállások kellenek !!
- egy eszköz master és slave módban működhet
- általában egy master és több slave eszköz van → címek szükségesek ! (7 bit)



23.2. Arduino I^2C kezelő könyvtár

„Wire.h”

Először a programunk elején ezt a fejállományt be kell importálnunk, hogy elérjük a szükséges függvényeket !

```
#include <Wire.h>
```

„begin” metódus

I^2C kommunikáció indítása.
A Master és Slave üzemmód beállítása

```
Wire.begin();                    // indítás Master módban
Wire.begin(SLAVE_ADDR);          // indítás Slave módban, címet megadva
```

„requestFrom” metódus

Adat kérése adott című eszköztől !!

```
Wire.requestFrom(address,byte_number);      //  a kérés után a busz felszabadul
Wire.requestFrom(address,byte_number,stop);
// ha van 3. paraméter →  ha értéke „igaz” a kérés után a busz felszabadul,
      //  ha hamis akkor viszont foglalt marad
                                  // speciálisabb szervók esetén

pl.  Wire.requestFrom(10,2);   // a 10-es címről kérünk 2 byte-ot
```

23.3. Arduino I^2C kezelő könyvtár

„SetClock” metódus

A Master így állítja be az órajelet

```
Wire.SetClock(cl);
```

„beginTransmission” metódus

Írás, olvasás előtt a slave eszköz beállítása.
A Master így kezdi a kommunikációt.

```
Wire.beginTransmission(address);
```

„endTransmission” metódus

A Master befejezi a kommunikációt a slave eszközzel.

```
Wire.endTransmission();
Wire.endTransmission(stop);
// ha van paraméter →  ha értéke „igaz” a kérés után a busz felszabadul,
      //  ha hamis akkor viszont foglalt marad
```

„write” metódus

Adat írása (küldése) a buszra.

```
Wire.write(value);      //  byte  vagy  string
Wire.write(byte_tömb,tömb_hossza);    //  tömb küldése
```

23.4. Arduino I^2C kezelő könyvtár

„available” metódus

A belső pufferben elérhető byte-ok, amiket olvasni lehet

```
Wire.available();
```

„read” metódus

Adat (byte) olvasása a buszrendszerről

```
Wire.read();
```

„onReceive” metódus

Slave esetén, megadja hogy melyik függvényt kell hívni, ha adat érkezett a mastertől.

```
Wire.onReceive(function_name);
```

„onRequest” metódus

Slave esetén, megadja hogy melyik függvényt kell hívni, ha adatot vár a master (adatot kell küldeni).

```
Wire.onRequest(function_name);
```

23.5. I^2C minta programok

1. mintafeladat

- Master mint vevő, slave mint adó
  Az arduino.cc tutorialból (https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterReader)

```
// Master (vevő)
// by Nicholas Zambetti <http://www.zambetti.com>

#include <Wire.h>

void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);        // start serial for output
}

void loop()
{
  Wire.requestFrom(8, 6);    // request 6 bytes from slave device #8

  while (Wire.available())
  {
    // slave may send less than requested
    char c = Wire.read();         // receive a byte as character
    Serial.print(c);                 // print the character
  }

  delay(500);
}
```

23.6. I^2C minta programok

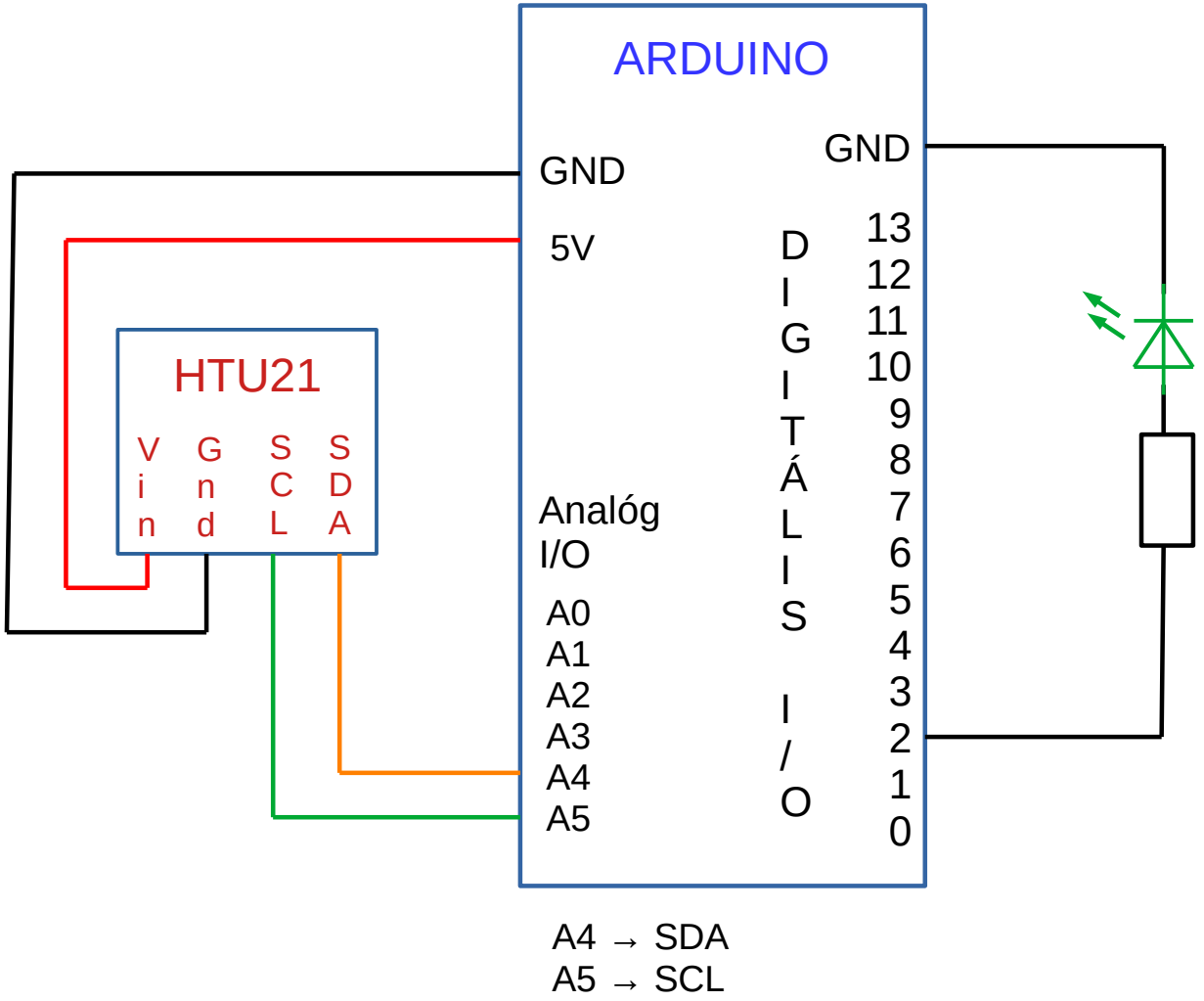
1. mintafeladat

- Master mint vevő, slave mint adó
  Az arduino.cc tutorialból (https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterReader)

```
// Slave (adó)
// by Nicholas Zambetti <http://www.zambetti.com>

#include <Wire.h>

void setup()
{
    Wire.begin(8);                 // join i2c bus with address #8
    Wire.onRequest(requestEvent);  // register event
}

void loop()
{
    delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
    Wire.write("hello ");        // respond with message of 6 bytes
                                 // as expected by master
}
```

23.7. HTU21 hőszenzor lekérdezése

A hardver



23.8. HTU21 hőszenzor lekérdezése

2. mintafeladat

```
// Master, Arduino (vevő)
// htu21 szenzor, mint slave és adó

#include <Wire.h>

byte Cim=0x40;                // szenzor címe
byte TempCmd = 0xE3;    // hőmérséklet adat lekérdezése
float homerseklet;
const byte Led=2;

void setup()
{
    Wire.begin();       // join i2c bus (address optional for master)
    Serial.begin(9600);     // start serial for output
}

void loop()
{
    homerseklet= tempread();
    Serial.println(homerseklet);
    pinMode (Led, OUTPUT);
    digitalWrite(Led, HIGH);
    delay(750);
    digitalWrite(Led, LOW);
    delay(750);
}
```

23.9. HTU21 hőszenzor lekérdezése

2. mintafeladat

```
float tempread()
{
        unsigned int result;
        Wire.beginTransmission(Cim);          //begin
        Wire.write(TempCmd);                  //send command
        delay(100);
        Wire.endTransmission();

        Wire.requestFrom(Cim, 3);             // request 3 bytes from slave
        while(Wire.available() < 3) {  ; }    //wait
        result = ((Wire.read()) << 8);
        result += Wire.read();
        result &= ~0x0003;                    // clear two low bits (status bits)
        return (-46.85 + 175.72 / 65536.0 * (float)result);
}
```

24.1. TMP36 hőszenzor használata

TMP36 szenzor

Forrás: http://www.tavir.hu/tipp-tmp36-homero-es-az-arduino/

Nagyon kicsi fogyasztású, pontossága ± 1-2 ˚C,
csak tápfeszültséget kell adni rá (2,7 – 5,5 V) és a
hőmérséklettel arányosan változik a kimenetének feszültsége (10mV / ˚C)

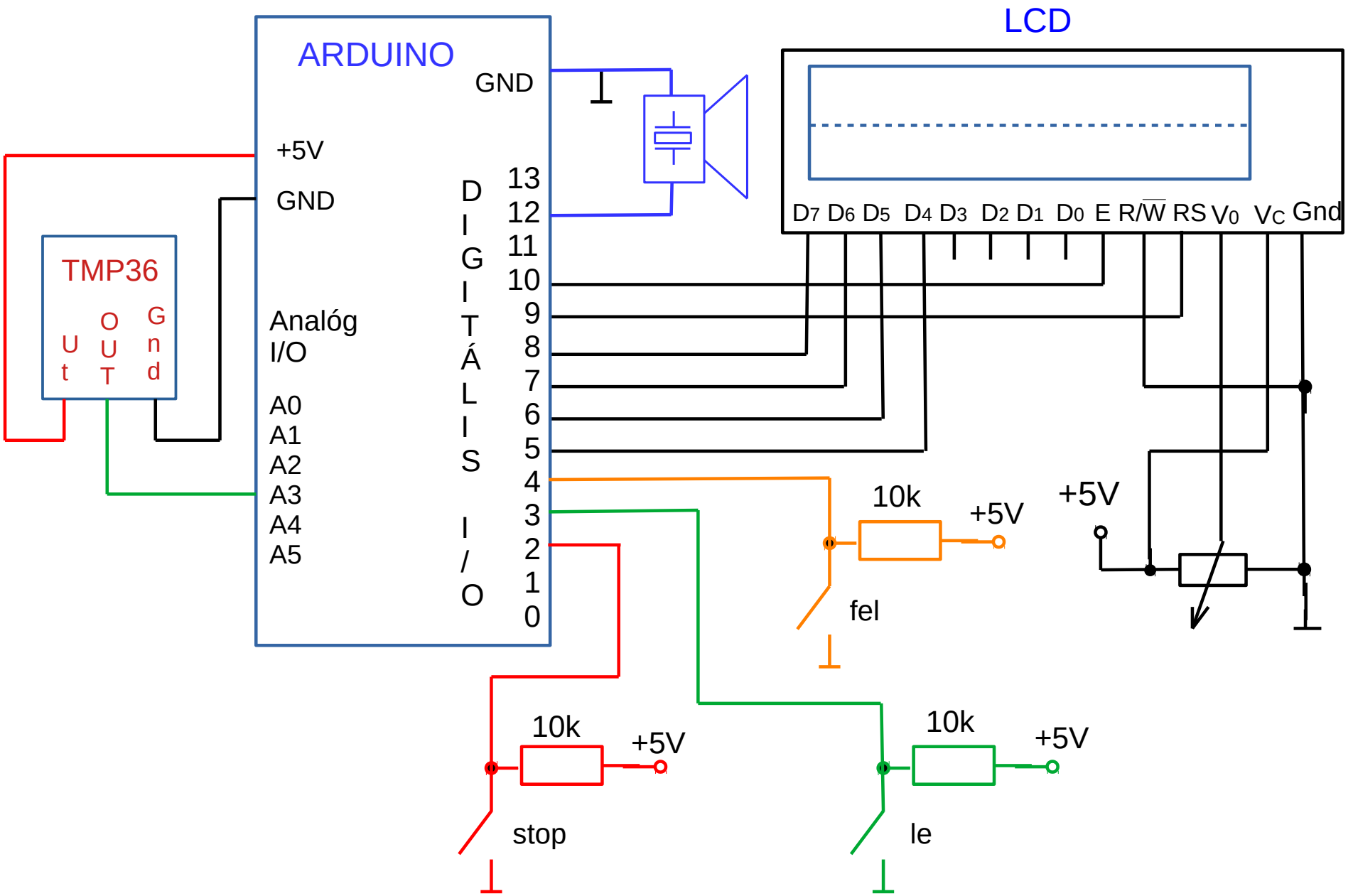
Mérési tartomány: -40 ˚C (0,1 V) …. + 125 ˚C (1,75 V)
0 ˚C esetén → U_{ki} = 0,5 V és
50 ˚C esetén → U_{ki} = 1 V →

A hőmérséklet számítása az adatlap alapján:
$T = 100*(U_{ki} – 0,5)$

U_{ki} számítása a beolvasott digitalizált érték (D) alapján:
$U_{ki} = D * 5 /1024$

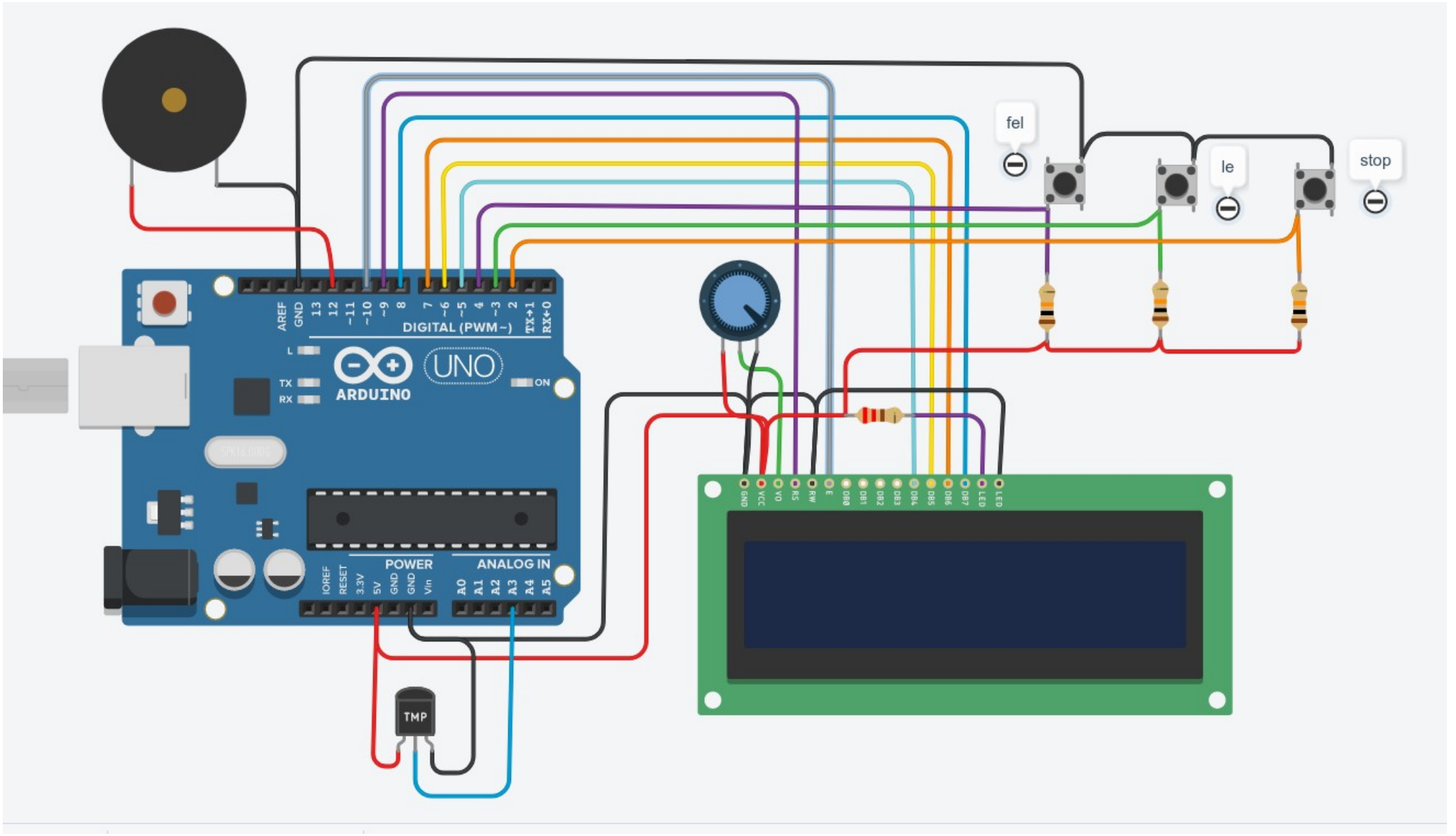
24.2. TMP36 hőszenzor használata

A hardver



24.3. TMP36 hőszenzor használata

A hardver



24.4. TMP36 hőszenzor használata

Beállítások

Alapvető beállítások a következő feladatokhoz

```
#include <LiquidCrystal.h>

// kimenetek
LiquidCrystal lcd(9, 10, 5, 6, 7, 8);
const byte hang=12;          // 12. lábon hangjelző

// bemenetek
const byte hom=17;           // A3 analóg bemeneten hőszenzor
const byte stop=2;           //  nyomógomb a 2. lábon
const byte le=3;             //  nyomógomb a 3. lábon
const byte fel=4;            //  nyomógomb a 3. lábon

// digitalizált feszültség,  (1023 ->  5V)
int dfesz=153;    // alapérték (25 fok)

float afesz=0.75;            // a tényleges feszültség
float fok=25.0;              // hőmérséklet

void setup()
{
        pinMode(hom, INPUT);            //  'hom'  bemenet lesz
        pinMode(hang, OUTPUT);          //  'hang'  kimenet lesz
        lcd.begin(16, 2);               // 16x2 karakter/soros LCD kijelző beállítása
        delay(1000);                    // késleltetünk 1000ms-t
        lcd.print("Hello !");           // Szöveg kiíratása
        delay(2000);
}
```

24.5. TMP36 hőszenzor használata

1. mintafeladat

Feladat: A mért feszültség és hőmérséklet kijelzése az LCD kijelzőn.
Másodpercenként mérjünk !

```
void loop()
{
      dfesz=analogRead(hom);       // analóg bemenet olvasása
      afesz=dfesz*5.0/1024.0;
      fok=100*(afesz-0.5);
      lcd.setCursor(0, 1);         // kurzor pozíció 2. sorba
      lcd.print("              ");  // előző érték törlése
      lcd.setCursor(0, 1);          // kurzor pozíció 2. sorba
      lcd.print(afesz);
      lcd.print("   ");
      lcd.print(fok);
      delay(1000);

}
```

24.6. TMP36 hőszenzor használata

2. mintafeladat

Feladat: A mért hőmérséklet kijelzése az LCD kijelzőn. Másodpercenként mérjünk !
Ha a hőmérséklet nagyobb mint +40 fok akkor adjunk folyamatos hangjelzést !

```
void loop()
{
      dfesz=analogRead(hom);       // analóg bemenet olvasása
      afesz=dfesz*5.0/1024.0;
      fok=100*(afesz-0.5);
      lcd.setCursor(0, 1);              // kurzor pozíció 2. sorba
      lcd.print("                ");
      lcd.setCursor(0, 1);               // kurzor pozíció 2. sorba
      lcd.print(fok);

      if(fok>40)              // ha a hőmérséklet  > 40 fok
       {
            tone(hang,600);      // 600 Hz-es jel felkapcsolása
       }
      else                         // egyébként (ha nem nagyobb)
        {
            noTone(hang);        // hang lekapcsolása
        }

      delay(1000);

}
```

24.7. Feladatok

Írj programokat az előző kapcsolásra (24.3.)

1. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után másodpercenként mérjünk hőmérsékletet ! A mért értéket írjuk ki az LCD kijelző 2. sorába
- a „STOP” nyomógomb lenyomása után leáll a mérés

2. feladat

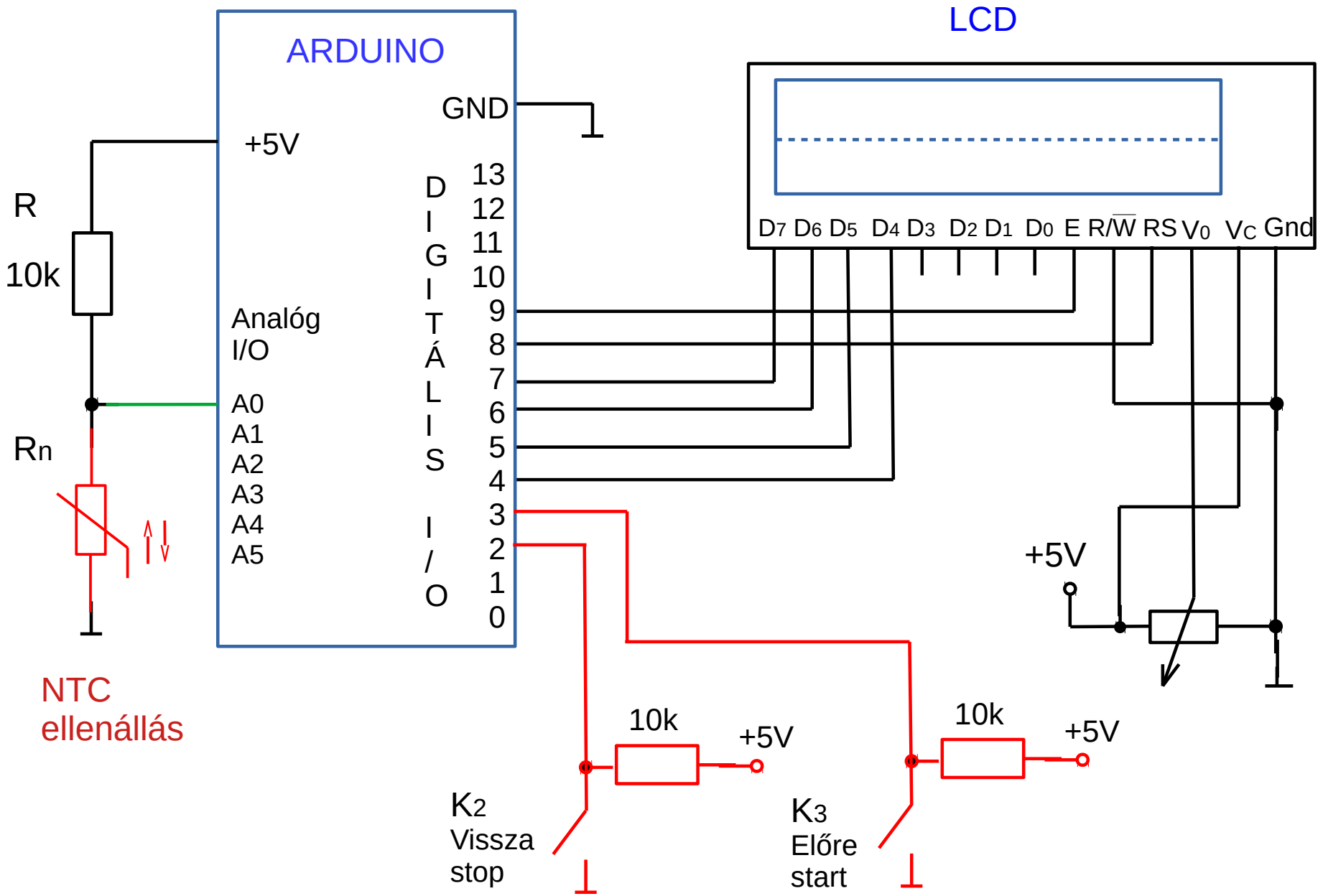
- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után másodpercenként mérjünk hőmérsékletet ! A mért értéket írjuk ki az LCD kijelző 2. sorába
- ha a hőmérséklet kisebb mint 0 fok akkor adjunk folyamatos hangjelzést !
- a „STOP” nyomógomb lenyomása után leáll a mérés

3. feladat

- Induláskor írjuk ki a ’60’ értéket az LCD kijelző 1. sorába
- a „FEL” és „LE” nyomógombok lenyomásával tudjuk egy számláló értékét növelni – csökkenteni, +40 és +100 között (a számláló alap értéke 60) !
- a számláló értékét mindig írjuk ki az LCD kijelző 1. sorába !
- másodpercenként mérjünk hőmérsékletet, értékét írjuk ki az LCD kijelző 2. sorába ! ha a hőmérséklet nagyobb mint a számláló értéke, akkor adjunk folyamatos hangjelzést !

24.8. Hőmérséklet függő ellenállás használata

A hardver



24.9. Hőmérséklet függő ellenállás használata

Termisztor (NTC, PTC ellenállás)

A hőmérséklet függvényében erősen változik az ellenállása

NTC – hőmérséklet növelésekor csökken az ellenállása,

PTC – hőmérséklet növelésekor nő az ellenállása

A sorba kötött fix értékű ellenállással egy feszültség osztót alkotnak, amelynek kimeneti feszültsége változik a hőmérséklet változásának hatására →
ezt vezetjük az Arduino analóg bemenetére !

A beolvasott digitalizált értékből (D) kiszámoljuk a feszültségosztó kimeneti feszültségét:

$U_{ki} = D * 5 /1024$

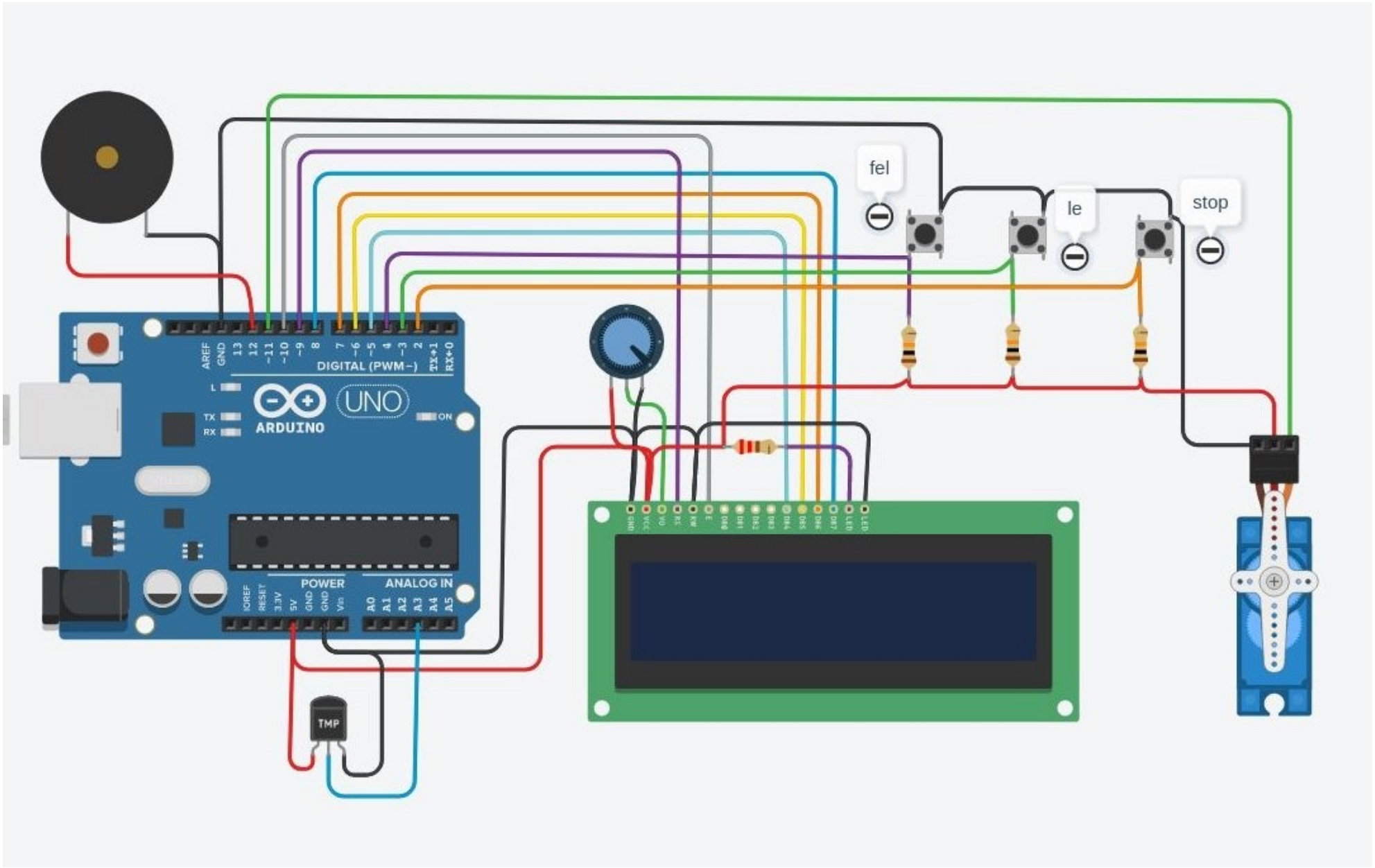
A feszültségosztó kimeneti feszültségéből, R ismeretében számolható az NTC aktuális értéke (Rn):

Mivel $U_{ki} = U_t * R_n / (R_n + R)$ → $R_n = R / (U_t/U_{ki} -1)$ → $R_n = 10 / (5 / U_{ki} - 1)$ kOhm-ban

Az aktuális Rn ismeretében az NTC adatlapja alapján számolható az aktuális hőmérséklet
Bonyolult, mivel nem teljesen lineáris általában a karakterisztika !

25.1. Feladatok

A hardver



25.2. Feladatok

Írj programokat az előző kapcsolásra (25.1.)

1. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után a szervómotor álljon be 90°-os pozícióba,
  ezután másodpercenként mérjünk hőmérsékletet !
  A mért értéket írjuk ki az LCD kijelző 2. sorába
- ha a hőmérséklet nagyobb mint 60°C fok akkor adjunk folyamatos hangjelzést !
- a „STOP” nyomógomb lenyomása után leáll a mérés, a motor visszaáll alaphelyzetbe !

2. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után másodpercenként mérjünk hőmérsékletet !
  A mért értéket írjuk ki az LCD kijelző 2. sorába
- ha a hőmérséklet kisebb mint 0°C fok akkor a szervómotor álljon 180°-os pozícióba, és adjunk folyamatos hangjelzést , ha a hőmérséklet 0 fok fölé megy → motor 0°-ba és hang leáll !
- a „STOP” nyomógomb lenyomása után leáll a mérés, a motor visszaáll alaphelyzetbe !

3. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után a szervómotor álljon be 90°-os pozícióba, adjunk rövid hangjelzést és írjuk ki az LCD kijelző 2. sorába → „90 fok”
- a „LE” nyomógomb lenyomása után a szervómotor álljon be 180°-os pozícióba, adjunk rövid hangjelzést és írjuk ki az LCD kijelző 2. sorába → „180 fok”
- a „STOP” nyomógomb lenyomása után a motor visszaáll alaphelyzetbe !

25.3. Feladatok

Írj programokat az előző kapcsolásra (25.1.)

4. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után adjunk rövid hangjelzést, a szervómotor álljon be 180°-os pozícióba, és írjuk ki az LCD kijelző 2. sorába → „180 fok”
- a „LE” nyomógomb lenyomása után adjunk rövid hangjelzést, a szervómotor álljon be 0°-os pozícióba, és írjuk ki az LCD kijelző 2. sorába → „0 fok”

5. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- a „FEL” nyomógomb minden egyes lenyomására a szervómotor pozíciója 10°-al növekedjen ! (180°-ig) Az aktuális pozíciót mindig írjuk ki az LCD kijelző 2. sorába
- a „LE” nyomógomb minden egyes lenyomására a szervómotor pozíciója 10°-al csökkenjen ! (0°-ig) Az aktuális pozíciót mindig írjuk ki az LCD kijelző 2. sorába

6. feladat

- Induláskor írjunk ki egy üzenetet az LCD kijelző 1. sorába
- másodpercenként mérjünk hőmérdékletet, és a mért értéket írjuk ki az LCD kijelző 2. sorába
- ha a hőmérséklet 0 és 100 °C fok között van akkor a szervómotor álljon ugyanolyan értékű szögelfordulásba, pozícióba (pl. 45°C esetén 45 fokos pozícióba)
- ha a hőmérséklet kisebb mint 0 fok akkor a szervómotor álljon 0-os pozícióba, és adjunk folyamatos hangjelzést , ha a hőmérséklet 0 fok fölé megy → hang leáll !
- ha a hőmérséklet nagyobb mint 100°C fok akkor a szervómotor álljon 100°-os pozícióba, és adjunk folyamatos hangjelzést , ha a hőmérséklet 100 fok alá megy → a hang leáll !

25.3. Feladatok

Írj programokat az előző kapcsolásra (25.1.)

7. feladat

- Induláskor írjunk ki egy üzenetet („Hello”) az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után a szervómotor álljon be 180°-os pozícióba, ezt a pozíciót írjuk ki az LCD kijelző 2. sorába („180 fok”)
- a „LE” nyomógomb lenyomása után a szervómotor álljon be 90°-os pozícióba, ezt a pozíciót írjuk ki az LCD kijelző 2. sorába („90 fok”)
- a „STOP” nyomógomb lenyomása után adjunk rövid hangjelzést, és a motor álljon vissza alaphelyzetbe !

8. feladat

- Induláskor írjunk ki egy üzenetet („Hello”) az LCD kijelző 1. sorába
- a „FEL” nyomógomb lenyomása után induljon egy számláló, amely másodpercenként számláljon egyesével felfelé 20-ig (induló érték 0), majd ott álljon le.
A számláló értékét írjuk ki az LCD kijelző 2. sorába
- a „LE” nyomógomb lenyomása után induljon egy számláló, amely másodpercenként számláljon egyesével lefelé 0-ig, majd ott álljon le.
A számláló értékét írjuk ki az LCD kijelző 2. sorába
- a „STOP” nyomógomb lenyomása után adjunk rövid hangjelzést, és a számlálás álljon le