Harsányi Réka – Juhász Márton András

Fizikai számítástechnika: elektronikai alapok és Arduino programozás

A következő leckék művészeti képzésben tanuló diákoknak szólnak, így az első öt fejezet, az elektronikai alapokat tárgyaló rész csak a legfontosabb definíciókat, alkatrészeket és működésüket, az alapvető számításokat, illetve mértékegységeket tartalmazza. A negyedik fejezet néhány játék szétszedésén és újrafelhasználásán keresztül az olcsó, programozást nem igénylő érzékelőket tárgyalja. A hatodik fejezet pedig már bevezet az Arduino programozásba. A hivatalos (arduinoArduino.cc) leckéken haladunk sorban, kiegészítve a szükséges háttér-információkkal. Az anyagban a Duemilanove típusú Arduino szerepel, azonban időközben megjelent egy újabb verziója, Uno néven. Ezeken kívül is számtalan típusa van az Arduinonak, melyek méretben, be- és kimenet számban és a ráépített mikrochip fajtájában térnek el egymástól.

Kulcsszavak: elektronikai alapok, alkatrészek, forrasztás, Arduino Duemilanove, Arduino programnyelv

BUDAPESTI KOMMUNIKÁCIÓS ÉS ÜZLETI FŐISKOLA

Typotex Kiadó

2014

Copyright: © 2014-2019, Harsányi Réka, Juhász Márton András, Budapesti Kommunikációs és Üzleti Főiskola

Creative Commons NonCommercial-NoDerivs 3.0 (CC BY-NC-ND 3.0)

A szerző nevének feltüntetése mellett nem kereskedelmi céllal szabadon másolható, terjeszthető, megjelentethető és előadható, de nem módosítható.

Lektorálta: Fernezelyi Márton

ISBN 978 963 279 189 0

Készült a Typotex Kiadó gondozásában

Felelős vezető: Votisky Zsuzsa

Készült a TÁMOP-4.1.2/A/1-11/1-2011-0010 számú, „Digitális és kollaboratív művészet” című projekt keretében.



Tartalom

1. Alapok és eszközök 7
1.1. Bevezető 7
1.1.2. Mi a fizikai számítástechnika? 7
1.2. Elektronikai alapok 8
1.2.2. Az általunk használt eszközök 8
1.2.3. Mértékegységek és fogalmak 13
2. Soros és párhuzamos kötés; Kapcsoló, relé; A kondenzátor és a teljesítmény fogalma; 34
2.1. Alkatrészek soros és párhuzamos bekötése 34
2.1.1. Ellenállások 34
2.1.2. Áramforrások 36
2.1.3. LED-ek 37
2.2. Kapcsoló és relé működése 39
2.3. A kondenzátor működése 47
2.4. A tranzisztor működése 51
2.4.1. Oszcillátor építése 53
3. A forrasztás menete és eszközei 62
4. Játékokból nyert szenzorok és piezo érzékelő 81
4.1. Low tech szenzorok és aktuátorok 81
4.1.1. Összetett rendszerek építése 82
4.1.2. Egyedi játékok és szerkentyűk belseje 87
4.2. Készítsünk kontakt mikrofont piezo szenzorból 103
5. Logikai kapuk 111
5.1. Integrált áramkörök 111
5.1.1. Kettes számrendszer 113
5.1.2. Logikai kapuk, logikai függvények 113
5.1.3. Az alkatrészszám leolvasása 116
5.1.4. Logikai áramkörök összerakása 117
6. Bevezetés az Arduino világába 134
6.1. A programozás alapja 134
6.1.1. Definíciók 134
6.1.2. Tervezés 135
6.1.3. Vezérlési szerkezetek 137
6.1.4. Adatszerkezetek 140
6.1.5. Függvények 140
6.2. Mi az Arduino? 141
6.3. Arduino Duemilanove részei 142
6.3.1. Mi a különbség az analóg és a digitális jel között? 144
6.4. Arduino fejlesztőkörnyezet telepítése 145
6.5. Programozás: szükséges minimum kód 152
6.6. Digitális pinek használata 153

6.6.1. LED villogtatása, kiolvasás digitális pinből 153
6.6.2. Nyomógomb használata, kiírás digitális pinre 155
6.7. Analóg pinek használata 157
6.7.1. Potméter használata, kiolvasás analóg pinből 157
6.7.2. Analóg jel feszültség értékének kiírása serial monitoron 159
6.8. Impulzus-szélesség moduláció 160
6.8.1. PWM pinek működése 160
6.8.2. LED fényerejének szabályozása PWM-mel 161
7. Példák digitális pinekhez 164
7.1. LED villogtatása delay() nélkül 164
7.2. Nyomógomb LED-del 166
7.3. Késleltetés (debounce) 168
7.4. Nyomógombos számláló 170
7.5. Pinek beépített felhúzó ellenállása 172
7.6. Dallam lejátszása tone() függvénnyel 173
7.7. Generált hajlítás lejátszása tone() függvénnyel 178
7.8. Analóg billentyűzet tone() függvénnyel 181
7.9. Hang lejátszása több kimeneten tone() függvénnyel 184
8. Példák analóg pinekhez 187
8.1. Analóg bemenet kiolvasása 187
8.2. Analóg bemenettel vezérelt PWM 188
8.3. Analóg szenzor kalibrálása 191
8.4. LED fényerejének szabályozása PWM-mel 193
8.5. Simítás 195
9. Soros kommunikáció 198
9.1. Adatátvitel PC-ről Arduinora 202
9.1.1. Soros adat küldése Processing segítségével 204
9.1.2. Soros adat küldése Max/MSP segítségével 206
9.1.3. Vezérlés karakterekkel 206
9.1.4. Switch feltétel használata soros kommunikációban 210
9.2. Adatátvitel Arduinoról PC-re 212
9.2.1. Adatok grafikus ábrázolása Processing segítségével 214
9.2.2. Adatok grafikus ábrázolása Max/MSP segítségével 217
9.2.3. Több független adat átvitele 218
9.3. Kétirányú kapcsolat – meghívás és válasz 222
9.3.1. Handshaking módszer Processing oldalról 224
9.3.2. Handshaking módszer Max/MSP oldalról 227
10. Vezérlő struktúrák 228
10.1. If állítás 228
10.2. For ciklus 230
10.3. Tömbök 232
10.4. While ciklus 234
10.5. Switch utasítás szenzor kiolvasással 237
10.6. Switch utasítás soros bemenettel 239

11. Szenzorok, motorok 242
11.1. ADXL3xx gyorsulásmérő 242
11.2. Piezo szenzor 247
11.3. Ultrahangos távolságszenzor 250
11.4. Sharp infravörös távolságszenzor 252
11.5. Motorok fajtái és működése 254
11.5.1. Az elektromágneses mező 254
11.5.2. Az elektromágneses indukció 255
11.5.3. Egyenáramú villanymotor 256
11.6. Egyenáramú villanymotor vezérlése H-híddal 258
12. Műveletek karakterláncokkal 263
12.1. Karaktertömbök 263
12.2. A string osztály példányai 264
12.2.1. String osztály további függvényei 265
13. Kommunikációs könyvtárak 276
13.1. Könyvtárak szerkezete 276
13.2. SoftwareSerial könyvtár 279
13.3. Firmata könyvtár 285
13.3.1. StandardFirmata használata Processingből 290
13.3.2. StandardFirmata használata Max/MSP-vel 291
13. 4. Wire könyvtár 292
13. 5. SPI könyvtár 299
13. 6. Ethernet könyvtár 305
13.6.1. Ethernet osztály 307
13.6.2. IPAddress osztály 308
13.6.3. Server osztály 309
13.6.4. Client osztály 311
13.6.5. EthernetUDP osztály 313
14. Könyvtárak hasznos eszközökhöz (háttértár, kijelző, motorok, programozó) 317
14.1. EEPROM könyvtár 317
14.2. LiquidCrystal könyvtár 319
14.3. Servo könyvtár 328
14.4. Stepper Motor könyvtár 332
14.4.1. Unipoláris léptetőmotorok 333
14.4.2. Bipoláris léptetőmotorok 335
14.4.3. A stepper motor könyvtár használata 337
14.4.4. A stepper motor könyvtár függvényei 340
14.5. Arduino, mint ISP 341
14.5.1. Arduino programozása Arduinoval 342
14.5.2. ATmega mikrokontroller programozása Arduinoval 342
Szintaxis Gyűjtemény 344
SZERKEZETEK 344
Vezérlési szerkezetek 344
További szintaxisok 349

Összehasonlító operátorok 351
Aritmetikai operátorok 351
Boolean / logikai operátorok 352
Pointerek 353
Bitenkénti operátorok 353
Léptető és bitenkénti operátorok 356
ADAT TÍPUSOK 358
Állandók 358
Adattípusok 361
Típusmódosítók 372
Típusminősítők 373
Alaptípus 375
FÜGGVÉNYEK 375
Digitális be-/kimenetek 375
Analóg be-/kimenetek 376
Haladó szintű be-/kimenetek 377
Idő 379
Matematika 380
Trigonometria 382
Random számok 383
Bitek és byte-ok 383
Soros port kommunikáció 385

1. Alapok és eszközök

1.1. Bevezető

A következő 5 fejezetben az elektronikai alapokkal ismerkedünk meg. Mivel a könyv művészeti képzésben részesülő diákoknak szól, az elmélet helyett inkább a gyakorlatban történő megértés kerül a középpontba. A legszükségesebb alkatrészek működését áramkörök összerakásán keresztül mutatjuk be. A 6. fejezetben az Arduino programozását kezdjük el. A tanulók gyakorlati feladatokon keresztül jutnak el a be- és kimenetek programozásától az egyéb szoftverekkel való kommunikáción át a különböző könyvtárak használatáig.

1.1.2. Mi a fizikai számítástechnika?

Tágabb értelemben nem más, mint annak a szemléletmódnak az elsajátítása, hogy miként kommunikálnak az emberek számítógépeken keresztül. Szűkebb értelemben olyan interaktív rendszerek létrehozását jelenti hardverek és szoftverek segítségével, melyek képesek érzékelni a világban létrejövő jeleket és regálni is tudnak rá. A szó a hétköznapi használatban gyakran saját készítésű csináld-magad projektek vagy alkotások megnevezésére szolgál, melyek mikrovezérlők (vezérlési feladatokra optimalizált extra kicsi számítógépek) be- és kimeneteire kötött különféle digitális és analóg érzékelők segítségével kontrollálnak elektromechanikus eszközöket, mint pl. a motorok, a világítás vagy egyéb eszközöket, de akár szoftvereket is.

Ahhoz hogy létrehozzunk ilyen interakciókat, meg kell tanulnunk a megfelelő nyelvezetet, hogy hogyan változik át fény-, hang-, hő- (hatás), stb. elektromos jellé, és hogyan tudunk ezekhez az analóg adatokhoz hozzáférni és digitális jellé alakítani, valamint reagálni rájuk. Tanulunk szenzorokról, alacsony feszültségű motorokról. Ez egy aktív részvételt igénylő tanulási folyamat, ami azt jelenti, hogy sok időt kell eltölteni áramkörök építésével, forrasztással, programozással és mindezek elemzésével, megértésével. Persze mindezekkel párhuzamosan az elektronika alapjait is el kell sajátítaniuk a diákoknak. Azt, hogy melyek a legalapvetőbb alkatrészek és mik a tulajdonságaik, hogyan kell őket áramkörbe kötni. A fizikai számítástechnika fontos részét képezi még a DIY (do it yourself = csináld magad) vonal. Például némelyik szenzor otthon is elkészíthető megfelelő anyagokból, lényegesen költséghatékonyabban.

1.2. Elektronikai alapok

1.2.2. Az általunk használt eszközök

Néhány eszközt most mutatok be, míg a forrasztáshoz használtakat később. Az alap szerszámok a következők lesznek:



1.1. ábra: Csípőfogó: ezzel vágjuk el a vezetéket.

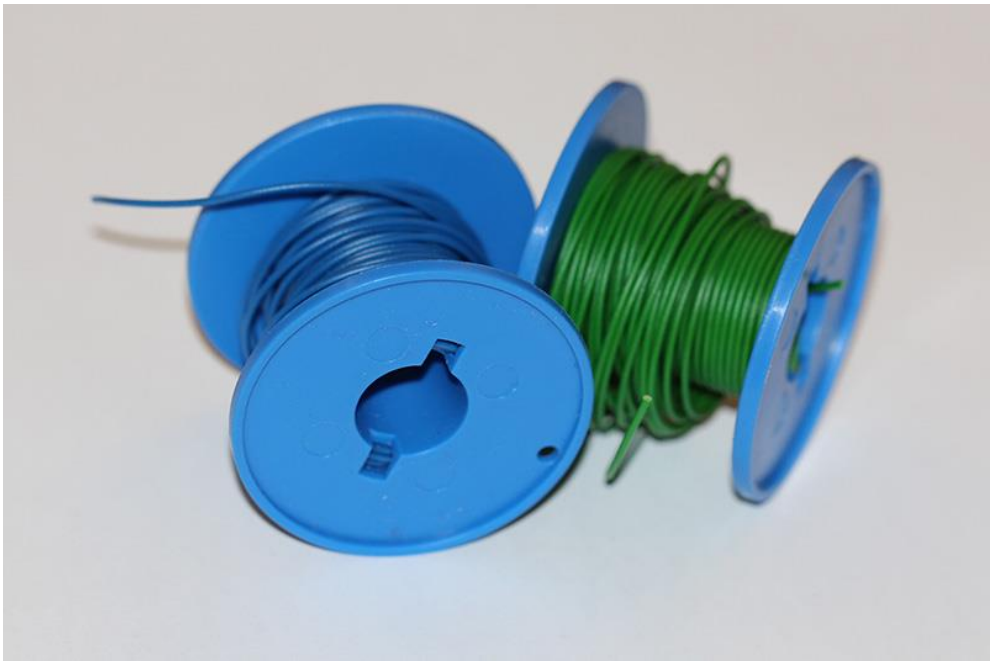


1.2. ábra: Laposfogó: ezzel tudjuk megfogni a forrasztás közben melegedő alkatrészt, hogy elkerüljük az égési sérüléseket.

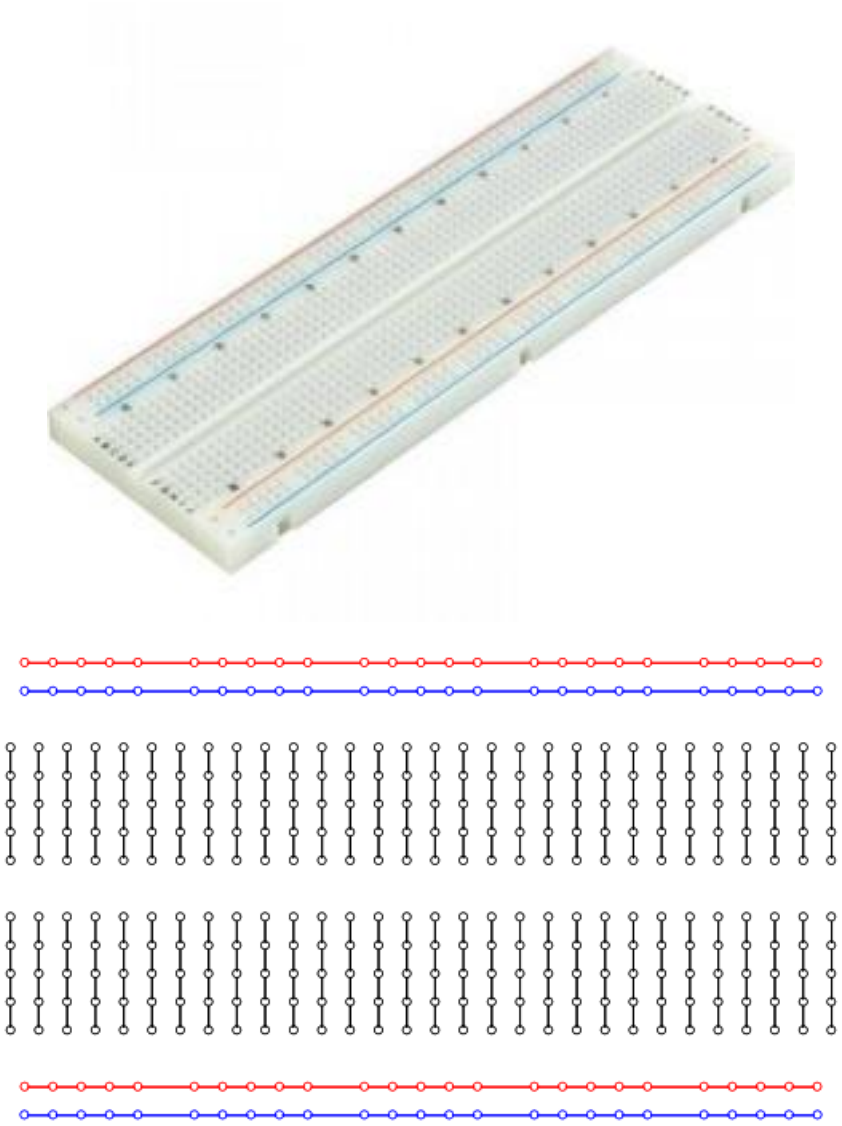
 © Harsányi Réka, Juhász Márton András



1.3. ábra: Állítható blankoló fogó: előnye, hogy a különböző vastagságú drótok méretéhez állíthatjuk, kényelmes a használata. A drót végének lecsupaszítására, azaz műanyag burkolatának (szigetelésének) levágására alkalmas eszköz.

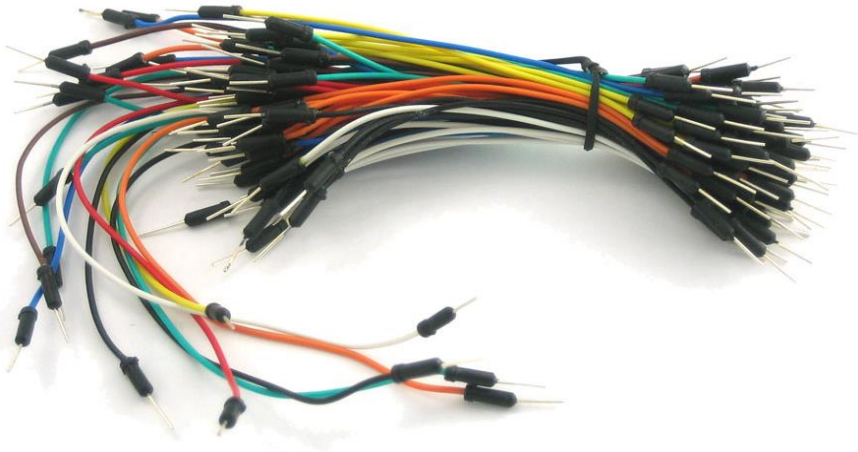


1.4. ábra: Drótok: célszerű tömör vezetékkel dolgozni, mert a többszálas végei szétállnak, így azokat nem lehet egykönnyen beledugni a dugaszolós próbapanelbe.



1.5. ábra: Dugaszolós próbapanel és összeköttetése.

A dugaszolós próbapanel úgy működik, hogy a burkolat alatt a sorok és oszlopok egy fém összeköttetéssel rendelkeznek, az 1.5 rajz szerint. Ezzel úgy lehet kapcsolatot létesíteni az egyes alkatrészek között, hogy a megfelelő helyekre be kell dugni azok kivezetéseit, azaz lábait. Előnye, hogy nem kell forrasztani és bármikor szétszedhető az áramkör, így az alkatrészeket újra fel lehet használni egy másik áramkörben. Alapvetően gyors prototipizálásra alkalmas, és ha már minden megfelelően működik, akkor véglegesen össze lehet forrasztani az alkatrészeket egy úgynevezett protoboardon, azaz prototípus NYÁK lemezlapon, melyekről a forrasztás meneténél bővebben írok.



1.6. ábra: Dugaszolós próbapanelhez kapható vezetékcsomag, de blankolt drótdarabokat is használhatunk.



1.7. ábra: Krokodilcsipeszek: drótok és alkatrészlábak összefogására használjuk. (Fontos, hogy az áramkör többi fém részéhez ne érjen a csipesz fém része, mert akkor olyan áramkört is zárhatunk, amit nem akartunk!)

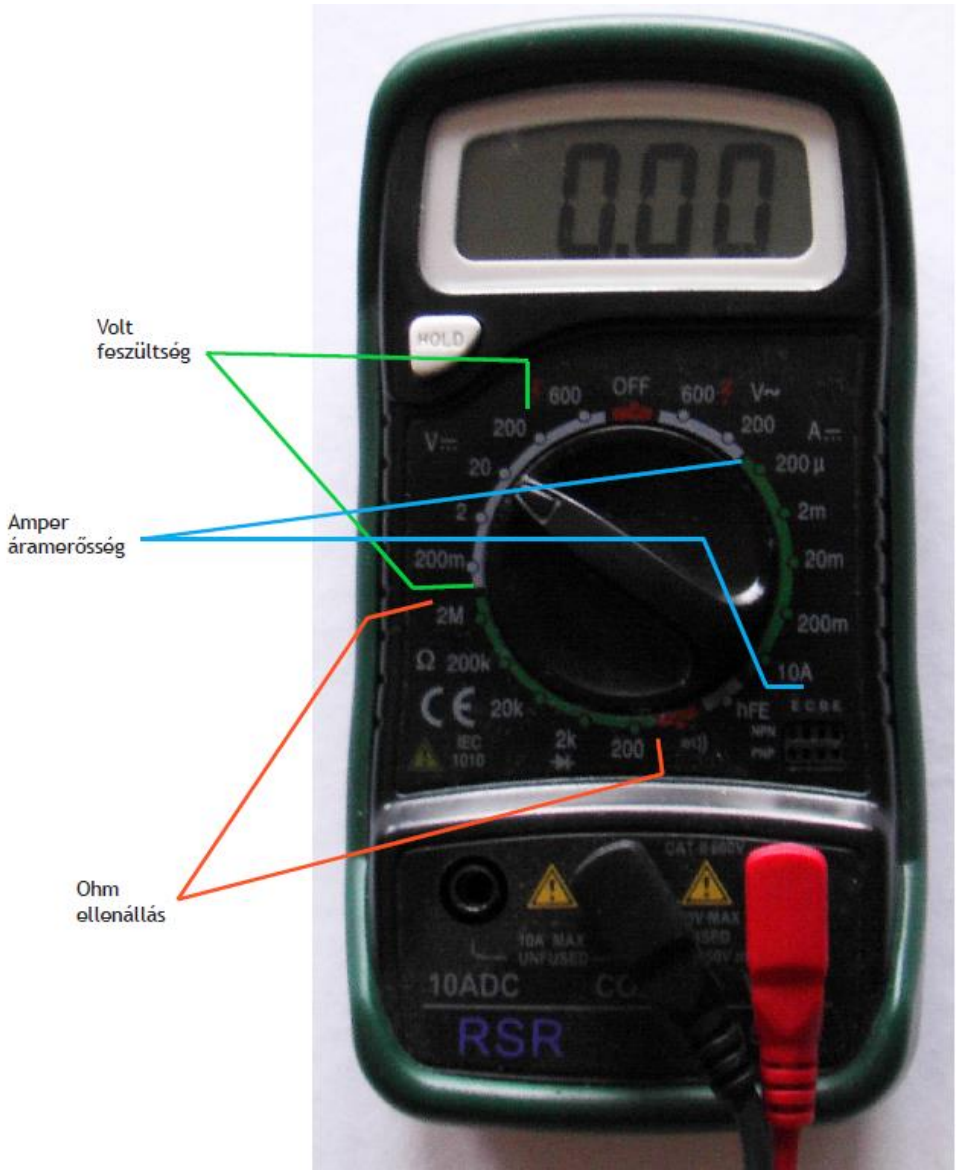


1.8. ábra: Digitális multiméter

A multiméter egy elemmel működő elektronikus berendezés, mellyel lemérhetőek az áramkörbe szerelt egyes alkatrészek, így megtalálható, hogy melyik nem érintkezik megfelelően, illetve az ellenállások is megmérhetőek vele. Az eredmény a kijelzőn látható.

Az alábbi három érték mérhető a digitális multiméterrel: feszültség voltban, áramerősség amperben, illetve ellenállás ohmban. Ehhez a megfelelő állásba kell fordítani a számtárcsát, mely mellett számok láthatóak. A mérésre szánt érték nem lehet ennél a számnál nagyobb. Pl. 6V-os áramkört nem lehet úgy mérni, hogy 2V-ra van állítva a multiméter, ilyenkor 20V-ra kell állítani! A két fém tüskét hozzá kell érinteni az alkatrészhez. Ha valamit helytelenül mérünk le, akkor „E” vagy „L” hibaüzenetet kapunk. Két kiszedhető mérőtüskével rendelkezik a multiméter, de alul három (10ADC, COM, V/Ω/mA) vagy négy (COM, V/Ω, mA, A) hely van nekik. A fekete kábel a negatív pólus, így az a COM (common = közös) lyukba kerül. A piros kábel a pozitív, melyet abba a lyukba kell beilleszteni, amit mérni szeretnénk vele. Mivel kisfeszültségű egyenárammal dolgozunk, a nagyfeszültségű elektromos áram (azaz váltóáram) mérésére alkalmas ADC lyukra nem lesz szükség, csak a többire. Némelyik multiméter kapacitás (Farád) mérésére is alkalmas, de mi most nem ilyet fogunk használni.

 © Harsányi Réka, Juhász Márton András



1.9. ábra: A számtárcsa beállításai

1.2.3. Mértékegységek és fogalmak

1.2.3.1. Volt és feszültség

A feszültség fogalmának megértéséhez végezzük el a következő kísérletet: Érintsünk a nyelvünkhöz egy 9 voltos elemet úgy, hogy a nyelvünkkel zárjuk annak két pólusát. Ekkor bizsergést érezhetünk a nyelvünkön, melynek oka, hogy a nedvesség vezeti az elektromos áramot. Ha szárazabbra töröljük a nyelvünket és megismételjük a kísérletet, azt tapasztaljuk, hogy kisebb bizsergést érzünk. Mivel az elem kisfeszültségű, ez a kísérlet nem veszélyes, de nagyfeszültségű áramba tilos belenyúlni, hiszen az emberi test nagy

része víz, így jól vezeti az áramot, ami – ha bizonyos irányba folyik át a testünkön – halálos is lehet! Hogy megértsük, mi történt, le fogjuk mérni multiméterrel az elemünket. Ennek összeszerelését és beállítását korábban bemutattam.



1.10. ábra: Kísérlet egy 9 voltos elemmel

A multiméterrel az elem feszültségét fogjuk lemérni, ezért a V feliratú lyukba tesszük a piros mérőtüskét. Fontos, hogy a tárcsa melletti értékek a maximum mérhető értékeket jelentik. Mivel az elem 9V-os, a tárcsát úgy kell beállítani, hogy maximum 20 voltot tudjon mérni. Ha az elem végéhez érintjük a megfelelő színű tüskéket, azaz a feketét a negatív, a pirosat a pozitív pólusához, akkor a multiméter kijelzőjén megjelenik az átfolyó áramfeszültség értéke, ami jelen esetben 9 volt.

Ha az áramot folyadékként képzeljük el, akkor a feszültséget, ami egy elektromos nyomás, úgy tudjuk leginkább megérteni, ha egy teli víztartály aljára képzelünk egy lyukat, amin folyik ki a víz. A tartályban lévő vízoszlop magassága egyenesen arányos az alul kiáramló víz sebességével, azaz a nyomásával. Ha ezt egy elemre vonatkoztatjuk, akkor az elem feszültsége a használat során egyre csökken, míg az elektronok el nem fogynak belőle.

A FESZÜLTSÉGRŐL
Az elektromos nyomást, amit feszültségnek nevezünk, voltban mérjük.
A feszültség jele: U, mértékegysége: V (volt).

Volt átváltása:
0,001 volt – 1 millivolt – 1 mV
0,01 volt – 10 millivolt – 10 mV
0,1 volt – 100 millivolt – 100 mV
1 volt – 1.000 millivolt – 1V

 © Harsányi Réka, Juhász Márton András

1.2.3.2. Ohm és ellenállás

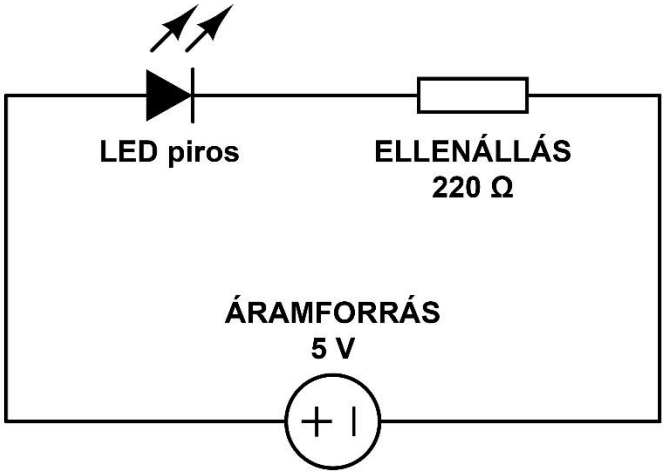
Az előző kísérletre reflektálva, most multiméterrel mérjük meg nyelvünk ellenállását! Ehhez át kell állítani a tárcsát ohmra (Ω)! A K kiloohmot, az M megaohmot jelent, melyek váltószámai alább találhatóak! Állítsuk 100 K-ra az értéket és tegyük a nyelvünkre a mérőtüskéket egymástól kb. 2,5 cm távolságra. A kijelzőn 50 K körüli érték jelenik meg. Ha a tüskék távolságát csökkentjük, a szám csökken, mely azt jelenti, hogy rövidebb távon kevesebb ellenállást kell leküzdenie az áramnak. Töröljük le a nyelvünket és ismételjük meg a mérést. Azt tapasztaljuk, hogy a kijelzőn egy nagyobb érték jelenik meg, mely azt jelenti, hogy a száraz nyelvnek nagyobb az ellenállása, azaz nehezebben haladnak rajta át az elektronok. Például, ha a karunkon mérjük meg az ellenállást, előfordulhat, hogy nem ír ki semmit a multiméter, hiszen itt nincs nedvesség, ami vezetné az áramot.

AZ ELLENÁLLÁSRÓL
Az elektromos ellenállást ohmban mérjük.
Az ellenállás jele: R, mértékegysége: Ω (ohm).

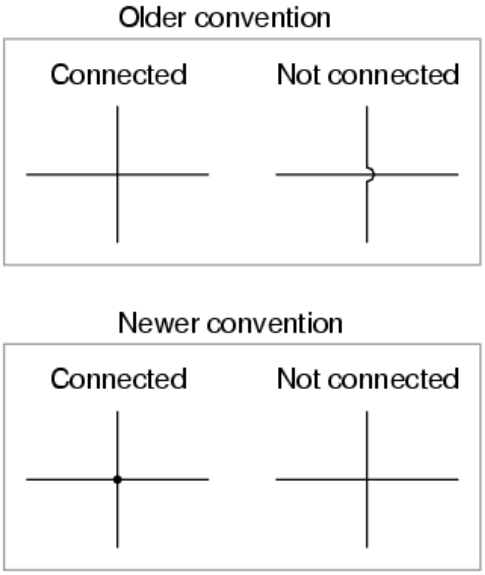
Átváltása:
KΩ = kiloohm, ami 1 000 Ω-t jelent,
MΩ = megaohm, ami 1 000 000 Ω-t jelent.
1,000 ohm - 1 kiloohm - jele: 1KΩ vagy 1K
10,000 ohm - 10 kiloohm - jele: 10KΩ vagy 10K
100,000 ohm - 100 kiloohm - jele: 100KΩ vagy 100K
1,000,000 ohm - 1 megaohm - jele: 1MΩ vagy 1M
10,000,000 ohm - 10 megaohm - jele: 10MΩ vagy 10M

*Azokat az anyagokat, melyeknek nagy az ellenállásuk, **szigetelők**nek hívjuk. Ide tartozik a legtöbb műanyag. A kis ellenállású anyagokat **vezetők**nek nevezzük, mint például a réz, az alumínium, az ezüst vagy az arany. A **félvezetők** csak részben vezetik az áramot.*

Mit jelent az **áramkör**? Az egyszerű áramkör egy elektromos árammal működő, vezetővel összekapcsolt alkatrészek összessége, tehát minimum tartalmaz egy fogyasztót és egy áramforrást. Az áramforrások egyéb energiákat (pl. mozgási-, fény-, vegyi-, hő-) alakítanak át villamos energiává. A fogyasztó, mint neve is mutatja, elhasználja ezt, és közben egy másfajta energiát szolgáltat, pl. az izzólámpa fényt, a melegítő hőt, a motor mozgást állít elő. Ha kapcsoló is van az áramkörben, akkor az csak zárt állapotban engedi át az elektronokat a két pólus között. Fajtáját tekintve lehet analóg vagy digitális.



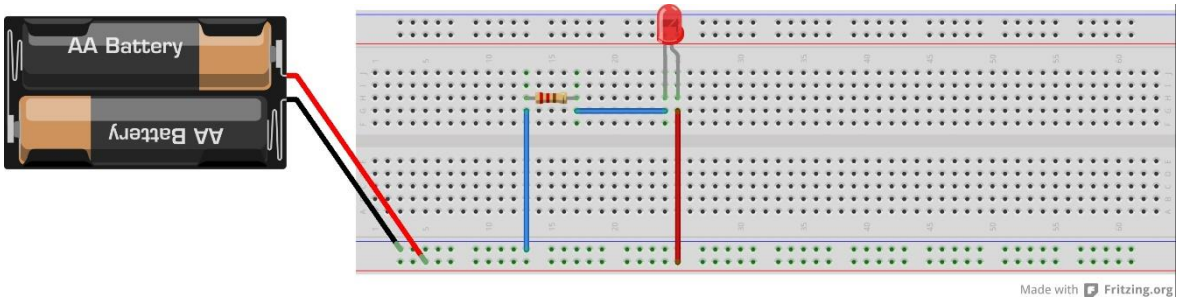
1.11. ábra: Egyszerű analóg áramkör kapcsolási rajza, felül egy fogyasztóval és egy kapcsolóval, alul egy áramforrással.



1.12. ábra: A vezetéket vonal jelzi, a régi verzió szerint, ha érintkeznek, akkor keresztezik egymást, ha nem, akkor az egyik hurokkal megy át a másikon. Az újabb jelölési típusnál, ha érintkeznek, egy pont jelzi azt, ha nem érintkeznek, akkor csak simán keresztezik egymást.

A **kapcsolási rajz** útmutatást ad, hogyan és milyen alkatrészeket kell összekötni az adott áramkörben. A **rajzjel** az egyes alkatrészek kapcsolási rajzban használatos jelképe.

© Harsányi Réka, Juhász Márton András



1.13. ábra: Fizikailag így néz ki a fenti áramkör

Az elektronikában az **áram iránya** az áramforrás pozitív sarkától a negatív sarka felé folyik, ezt nevezzük technikai áramiránynak. Éppen ellenkezőleg mondjuk, mint ahogy valójában mozognak a részecskék, ezt nevezzük fizikai áramiránynak, tehát a fizikailag negatív oldalt jelezzük plusszal, a fizikailag pozitív oldalt pedig mínusszal. Ennek oka, hogy az 1700-as években folytatott kísérletek így kerültek rögzítésre és később a fizikai törvényeket is e mentén határozták meg, így megtartották a régi fogalmat.

1.2.3.4. Egyszerű áramkör összerakása áramforrás, LED és ellenállás felhasználásával

Hogy megértsük, mit jelent az ellenállás és milyen szerepet játszik az áramkörben, rakjuk össze első áramkörünket! Ehhez a következő alkatrészekre lesz szükség: 4 db AA vagy AAA típusú (ceruzaelem) 1,5 voltos elemre és hozzá illő elemtokra, egy LED-re, alligátor-csipeszekre vagy próbapanelre és 470 ohm, 1 kiloohm és 2.2 kiloohm értékű ellenállásokra. Először nézzük végig, mik is ezek az alkatrészek.

Az **elem** a legegyszerűbb áramforrás, melyben egy vegyi folyamat játszódik le, melynek következtében szabad elektronok keletkeznek, ezek egyik irányból a másikba folynak. Például képzeljük el az elem belsejét két víztartályként, az egyik legyen tele, a másik pedig üres. Ha képzeletben alul összekötjük őket egy csővel, akkor a víz addig folyik át rajta az egyik irányba, amíg az üres tartályban ugyanannyi víz nem lesz, mint a korábban teli tartályban, tehát a két mennyiség kiegyenlítődik. Ugyanezt csinálják az elektronok az elem belsejében, egyik szélétől a másik felé haladnak. Az elemeknek több fajtája van, melyek méretben, feszültség és áramerősség értékben is eltérnek egymástól. A benne lezajló folyamat nem visszafordítható, ami azt jelenti, hogy ha lemerült, nem működik tovább. Ezt kipróbálhatjuk, ha három félbevágott citromot cink- és rézdarabok segítségével (3 x kb. 1 V), valamint egy LED-et sorba kötünk úgy, hogy a pozitív oldalon legyen a réz- és a negatívon a cinkdarab.

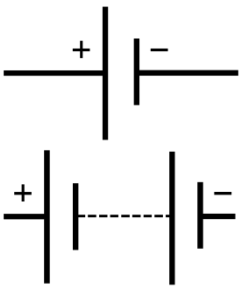
A vegyi folyamat megértéséhez tudni kell, hogy egy atom semleges töltésű neutronokból, pozitív töltésű protonokból és negatív töltésű elektronokból áll. Most a két utolsó lesz fontos. Egy atomot feldarabolni nagyon sok energiát igényel, de ahhoz, hogy néhány

elektron leváljon róla, kevés energia is elég. Például, ha cinkelektróda savba merül, az elektronok vándorlása addig történik, még a cinkelektródán nem lesz több szabad hely.

Az elem veszélyes hulladék, ezért csak használt elem gyűjtő konténerbe dobjuk el! Az akkumulátorok is hasonló elven működnek, azzal a különbséggel, hogy a bennük lezajló folyamat visszafordítható, azaz feltölthetőek.

Most térjünk ki arra, mi a **rövidzár**! Ha az áramforrás két kivezetését közvetlenül összekötjük, egy rövidzárat hozunk létre. Ilyenkor melegedés tapasztalható. Kipróbálhatjuk ezt egy 1,5 voltos ceruzaelemmel. (Használhatunk hozzá elemtokot is, de vigyázzunk, mert hamar felforrósodik!) Ugyanez történik, ha véletlenül összeérnek alkatrészek az áramkörben vagy épp rosszul kötjük be őket, ezért kísérlet közben figyeljünk oda, mert tönkremehetnek az alkatrészek.

Az elem rajzjele:





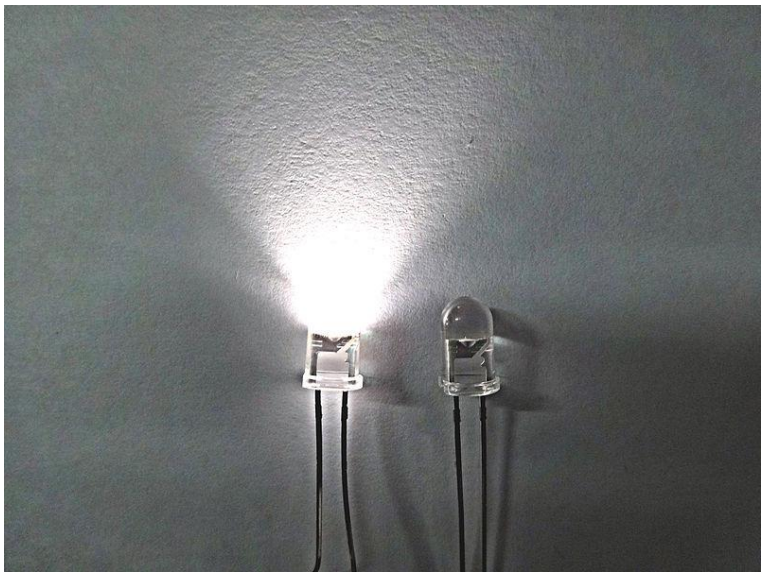
1.14. ábra: Különböző típusú szárazelemek

A hagyományos elemen kívül alternatív energiaforrások is elérhetőek, mint például a napelem vagy a dinamó. Fontos, hogy az általa előállított feszültség és áramerősség

 © Harsányi Réka, Juhász Márton András

megegyezzen az áramkör alkatrészei által használttal, illetve a megfelelő alkatrészek (pl. ellenállások) beszerelésével csökkentsük a megfelelő szintre az értékét. A leckékben ceruzaelemeket és 9 voltos elemet fogunk használni.

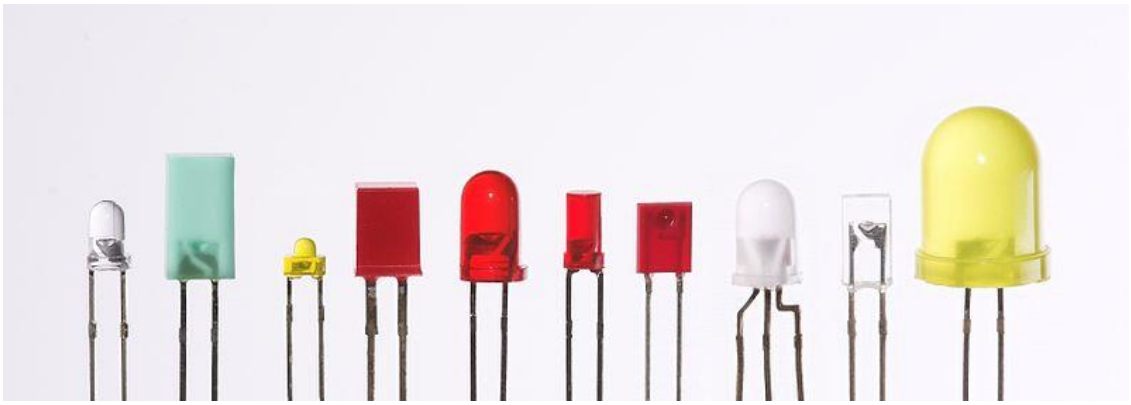
A **LED,** ami a Light Emitting Dioda rövidítése, magyarul fénykibocsátó dióda. Különböző méretű, színű, iránykarakterisztikájú, feszültségű típusa létezik.



1.15. ábra: A LED iránykarakterisztikája a fényszórási szöget jelenti

Látható fény tartományban lehet fehér, zöld, sárga, kék, piros és narancssárga, látható tartományon kívül infravörös vagy ultraibolya, illetve RGB is lehet, ilyenkor minden szín egy lábra van kivezetve, így minimum négy lába van. Méretben és alakban is eltérőek lehetnek.

LED rajzjele: 



1.16. ábra: Különféle LED-ek

Színek, hullámhosszok és a hozzájuk tartozó feszültségérték:

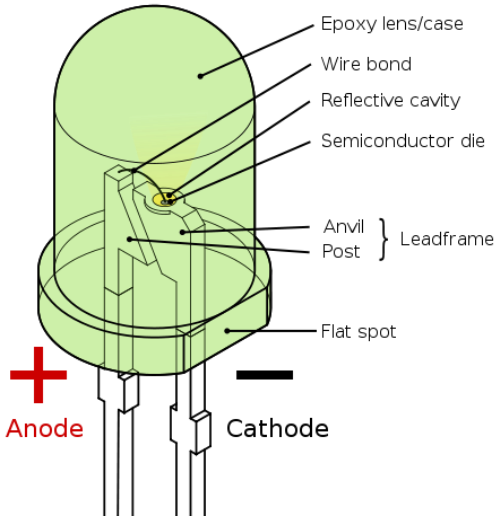
- ultraibolya (400 nm, 749 THz, 3,1 eV környéke),
- kék (470 nm, 638 THz, 2,6 eV környéke),
- zöld (568 nm, 523 THz, 2,2 eV környéke),
- sárga (588 nm, 510 THz, 2,1 eV környéke),
- narancs (625 nm, 480 THz, 2,0 eV környéke),
- vörös (640 nm, 468 THz, 1,9 eV környéke),
- infravörös (900-950 nm, 333-316 THz, 1,4–1,3 eV),
- rózsaszín (kék chip és piros fénypor),
- fehér (RGB megfelelő arányú keveréke, illetve ultraibolya chip és fénypor),
- többszínű LED: közös tokozásban található, például piros és zöld szín, de lehet az R-G-B mindhárom színe, melyet szivárvány LED-nek is neveznek.

Mi most a legelterjedtebb 5 mm átmérőjű, furatszerelt tokozású, piros színű verziót fogjuk használni.

A LED-nek az anód a pozitív, hosszabb szárú, a katód a negatív, rövidebb szárú oldala. Az anód egy olyan elektródát (fém kivezetést) jelent, amiből elektronok lépnek be az áramkörbe. A katód az ellentétes töltésű elektróda, amire átáramlanak az elektronok a LED-en belül.

Közelebbről jól látszik, hogy az egyik szélén egyenes a burkolat, ez a negatív oldal. (Ha le van vágva egyforma hosszúra a két lába, innen be tudjuk azonosítani a lábait.)

 © Harsányi Réka, Juhász Márton András



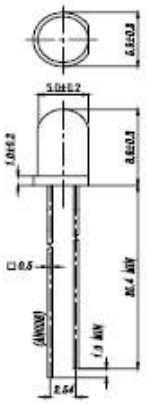
1.17. ábra: LED felépítése: a katód szár tart egy „tálcát”, amihez az anódlábból egy kis vezeték érkezik, a műanyag burkolat oldala a katódlábnál egyenesre van vágva.

Minden alkatrésznek van adatlapja, melyet vagy hozzácsomagolnak, vagy az interneten elérhető. Ebben megtalálható, hogy mekkora feszültség szükséges az üzemeltetéséhez. A különböző színű LED-nél ez eltérő lehet, ezért ha nem piros 5 mm-es LED-et használunk, a példákban megjelölt ellenállásérték nem biztos, hogy megfelel. Ilyenkor a későbbiekben tárgyalt Ohm-törvény segítségével kiszámolható, hogy adott esetben milyen értékű ellenállásra van szükség.

Technical Data Sheet
5.0mm Round Type LED Lamps

333-2SDRT/S530-A3

Package Dimensions



Notes: 1. All dimensions are in millimetres
2. The height of flange must be less than 1.5mm(0.059").
3. Without special declared,the tolerance is±0.25mm.

■ Absolute Maximum Ratings at Ta = 25℃

Parameter	Symbol	Rating	Unit
Forward Current	IF	25	mA
Operating Temperature	Topr	-40 to +85	℃
Storage Temperature	Tstg	-40 to +100	℃
Electrostatic Discharge	ESD	2000	V
Soldering Temperature	Tsol	260 ± 5	℃
Power Dissipation	Pd	60	mW
Reverse Voltage	VR	5	V

Note: *1:Soldering time ≦ 5 seconds.

© Harsányi Réka, Juhász Márton András

Technical Data Sheet
5.0mm Round Type LED Lamps

333-2SDRT/S530-A3

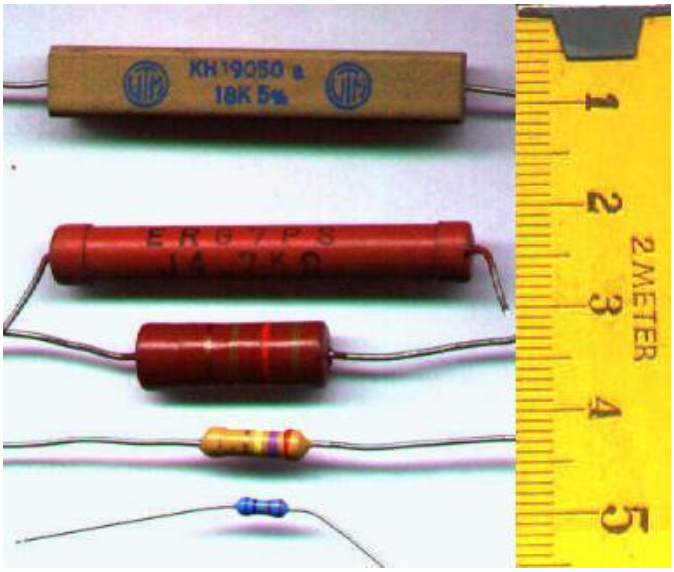
Electro-Optical Characteristics (Ta=25°C)

Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
Forward Voltage	VF	I_F= 20 mA	/	2.0	2.4	V
Reverse Current	IR	V_R= 5 V	/	/	10	μA
Luminous Intensity	Iv	I_F= 20 mA	500	1250	/	mcd
Viewing Angle	2θ 1/2	I_F= 20 mA	/	10	/	deg
Peak Wavelength	λp	I_F= 20 mA	/	632	/	nm
Dominant Wavelength	λd	I_F= 20 mA	/	624	/	nm
Spectrum Radiation Bandwidth	△λ	I_F= 20 mA	/	20	/	nm

1.18. ábra: LED adatlapja (datasheet)

- V_F forwarded voltage: az a feszültség, amin működik, azaz világítani kezd a LED;
- I_V luminous intensity: a LED fényereje, mértékegysége az mcd = millicandela;
- I_F forwarded current: a LED által felhasznált áramerősség;
- Viewing Angle: a szórási szög fokban megadva.

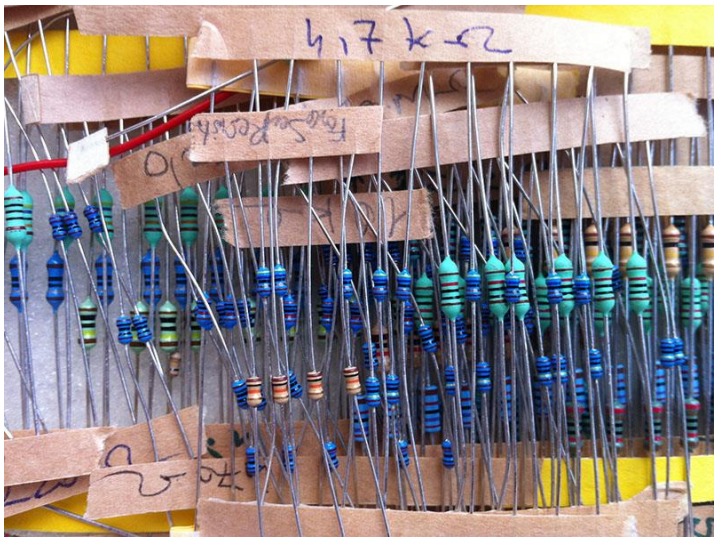
Az **ellenállások** fő jellemzője nevéből adódóan az ellenállásérték (Ohm). A mögé kötött alkatrész számára a megfelelő feszültségérték hozható létre vele. A soron következő áramkörben az ellenállásértékekkel fogunk kísérletezni.



1.19. ábra: Ellenállások különböző kivitelben léteznek

Ellenállás rajzjele Európában: 

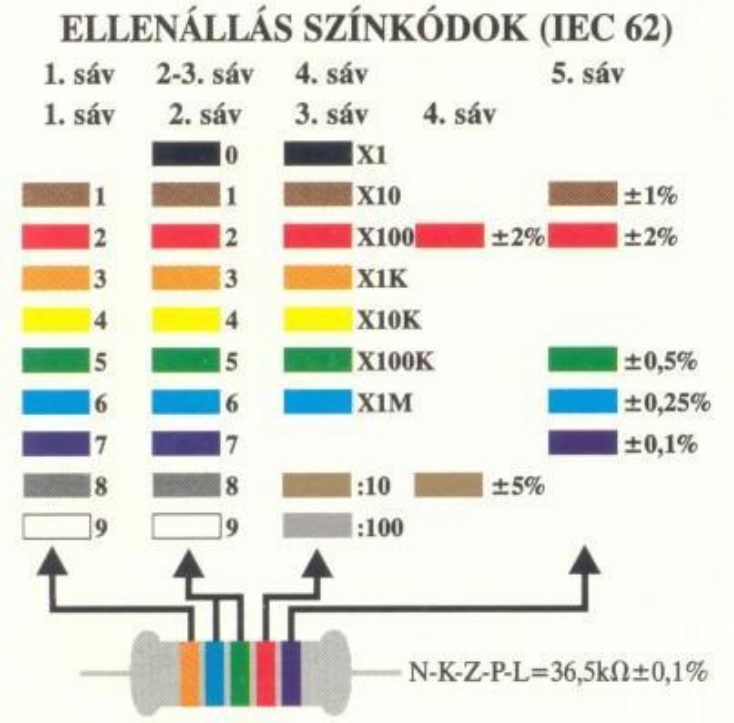
Amerikában: 



1.20. ábra: Különböző értékű ellenállások

Honnan tudhatjuk meg az ellenállások értékét? Mivel az általunk használt 0,25 wattos ellenállások nagyon kicsik, nem férne el rajtuk ez a sok adat, így színkódolással vannak ellátva. Más fajtákon egy szám jelzi az értéküket.

 © Harsányi Réka, Juhász Márton András



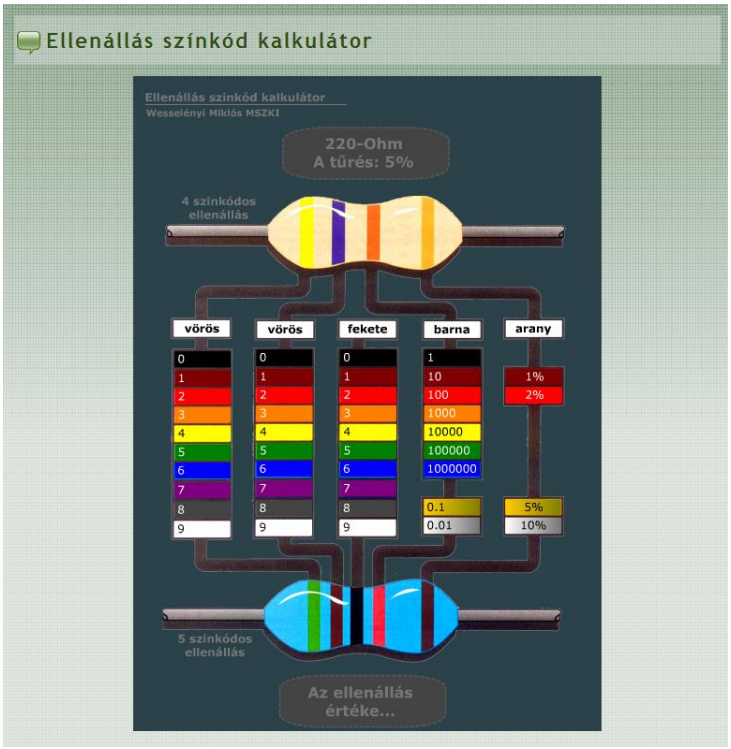
1.21. ábra: Ellenállás színkód értékei

Az ellenállások színkódolását vissza tudjuk fejteni a 1.21 számú ábra segítségével. Az ellenállás alapszíne nem fontos, csak a csíkok színe és sorrendje. Lehet négy- vagy ötsávos a kódolás. Ha van rajta arany vagy ezüst csík, annak a jobb oldalon kell lennie a sorrend leolvasásakor. Az ezüst csík 10%-os, az arany 5%-os tűréshatárt jelent, illetve – amint a 1.21 számú ábrán látható – az ötödik csík színe is a különböző tűrésértékeket jelenti. Ötsávos ellenállásokat fogunk használni, ezeknél az első három sáv mutatja meg az értékét. Például vegyük a piros-piros-fekete-fekete-barna sorrendet: ez azt jelenti, hogy az első oszlopból piros = 2, másodikból piros = 2, harmadikból fekete = 0, a negyedik azt mutatja meg, hogy mennyivel kell beszorozni az eddig kapott értéket, itt a fekete = 1. Ha ezt leírjuk egymás mellé: 220 x 1 = 220 ohm. Az eredményt ohmban kapjuk, melyet át kell váltani kilo- vagy megaohmra. Az utolsó, ötödik sáv a tűréshatárt jelenti, mely jelen esetben barna = 1%, ami azt jelenti, hogy az adott ellenállás értéke megközelítőleg 218–222 ohm. Ez elég bonyolultnak tűnhet, ezért multiméterrel is lemérhető az ellenállások értéke.

Léteznek online kalkulátorok is, ahol, ha megfelelő sorrendben megadjuk a színeket, megkapjuk az ellenállás értékét. De van közöttük olyan, amelyik még arra is figyelmeztet, ha rossz sorrendben adtuk meg a csíkokat, ezáltal nem szabványos az értéket kaptunk, – ilyenkor a fordított sorrend lesz a jó megoldás. Ezek a szoftverek arra is alkalmasak, hogy az általunk kiválasztott értéket és a színkódok csíkjainak számát megadjuk, akkor megmutatják a csíkok színét is, sorrendben.

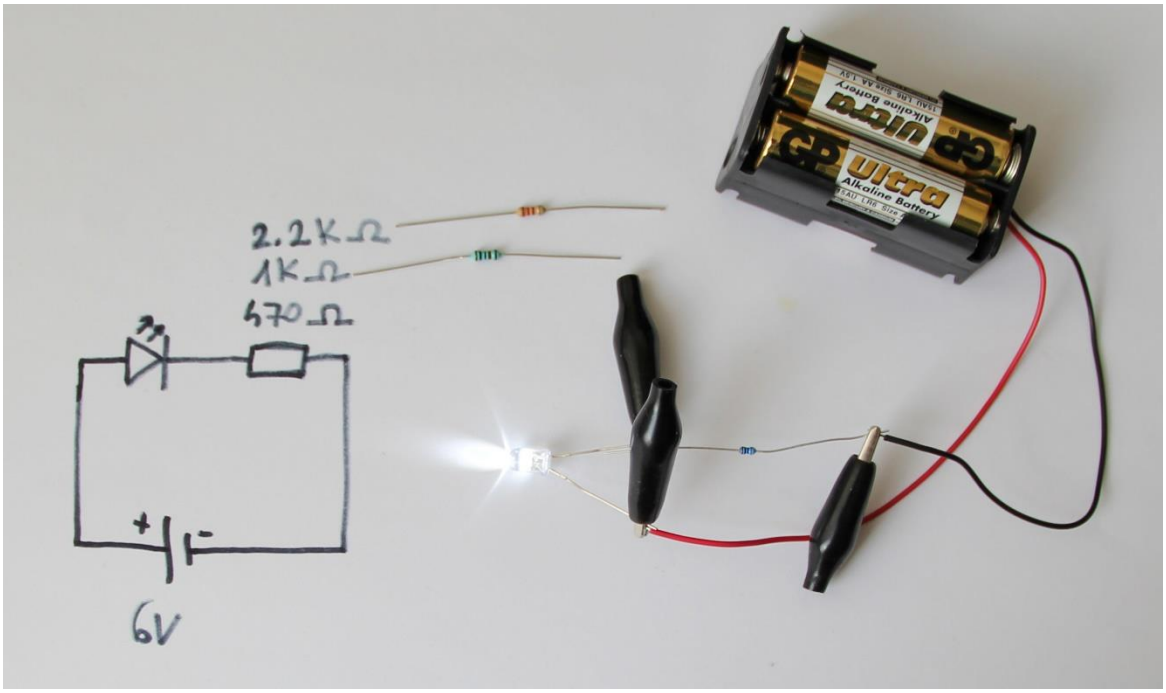
Az alábbi linkeken találhatunk ellenállás-kalkulátorokat:

- http://www.hobby-hour.com/electronics/resistorcalculator.php
- http://www.wmszki.hu/honlap/szinkod
- http://www.hobbielektronika.hu/segedprogramok/?prog=ellenallas
- http://elektrotanya.com/?q=hu/ellenallas_szinkod



1.22. ábra: Online kalkulátor az ellenállás értékének meghatározásához (http://www.wmszki.hu/honlap/szinkod)

Most szereljük össze az áramkört a következő kapcsolási rajz szerint! 4 db ceruzaelemet, azaz 6 volt feszültséget használunk Először a 2,2 K ellenállást építsük be, melynek színkódja öt sáv esetén: piros-piros-fekete-barna és a tűréshatár színkódja. Azt tapasztaljuk, hogy a LED fénye nagyon kicsi. Ha kicseréljük az ellenállást az 1 K-ra, a LED fénye nő, de még mindig nem az igazi. Most a 470 ohmos ellenállást is kipróbáljuk. Így már egészen jól világít a LED.



1.23. ábra: Ellenállások kipróbálása LED-es áramkörben

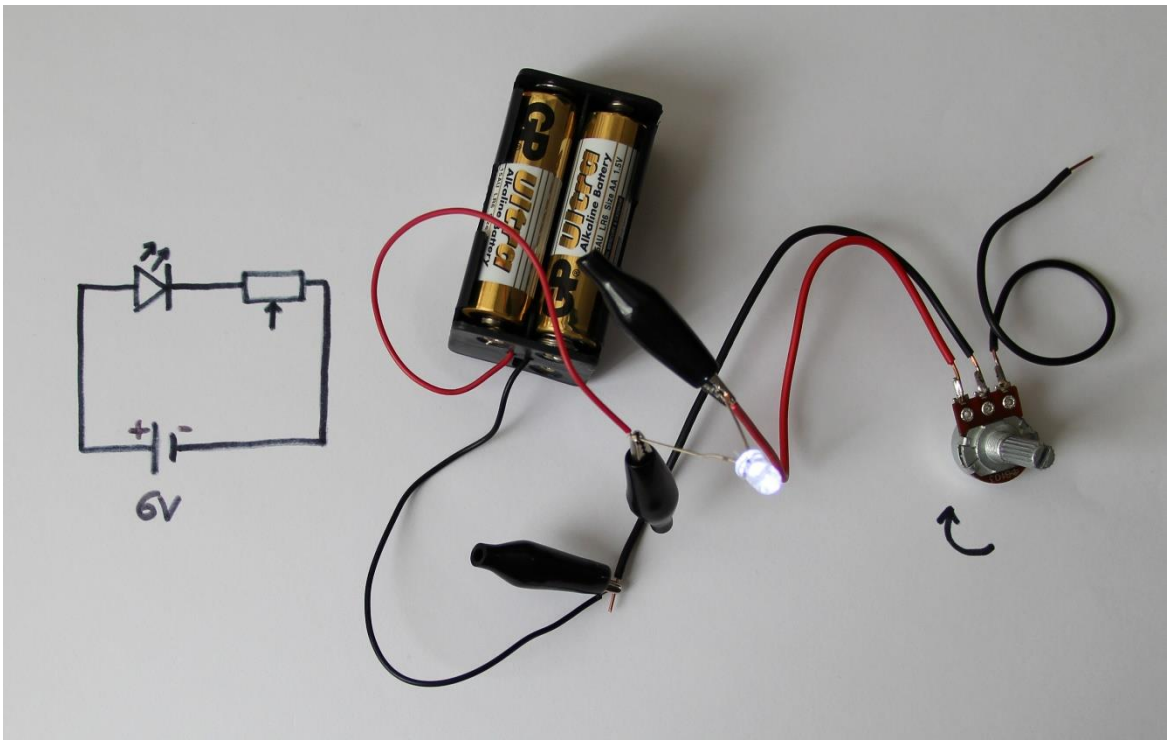
Fontos megemlíteni ennél a résznél, hogy léteznek változtatható értékű ellenállások, ezek a potenciométerek. Az ellenállás két végpontja közé egy csúszóérzékelő kerül, ezt lehet mozgatni. Lehet csúszkás, lineárisan tolható vagy tengely körül forgatható. Az ellenállásérték változása pedig lehet lineáris vagy logaritmikus. Ezen kívül fontos még, hogy mekkora a maximális ellenállásértéke, a maximális teljesítménye és hány fokban hajlítható el. Általában teljes körnél kicsivel kevesebb ez a szög (pl. 330 fok), de vannak speciális, többmenetes fajták is, melyek több kört tudnak fordulni.

Potenciométer rajzjele: 



1.24. ábra: Egyfordulatú tekerős potméter, ez a legelterjedtebb

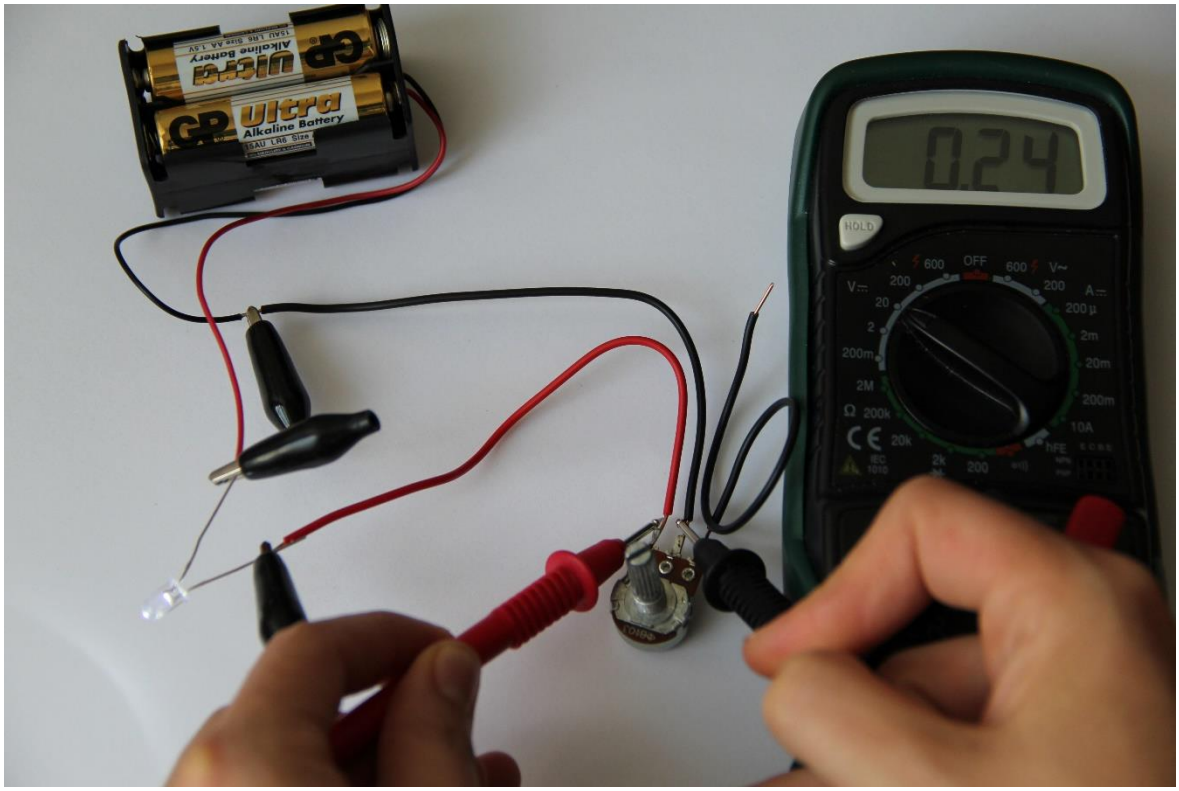
Az előző áramkörben cseréljük ki az ellenállást egy 2 kiloohmos lineáris, tekerős potméterre! Amikor összerakjuk az áramkört, a potméter legyen letekerve. Ha felülről nézzük, úgy, hogy a lábai előre mutassanak, tekerjük el teljesen az óra járásával ellentétes irányba! Balról az első lábára kötjük az elem negatív pólusát, a középső lábra a LED rövidebb, negatív szárát. A LED hosszabb, pozitív szára ugyanúgy marad az elem pozitív pólusára kötve. Nagyon lassan tekerjük a potmétert. Azt tapasztaljuk, hogy egy fokon túl a LED elromlik, kiég. Ez azért van, mert túl sok feszültséget kapott, az ellenállás értéke túl kevés lett, amikor feltekertük a potmétert. Cseréljük ki a potmétert egy 10 kiloohmosra és a LED-et is egy újra, majd ismételjük meg a kísérletet. Ebben az esetben nem megy tönkre a LED-, mivel a nagyobb ellenállás megfelelően korlátozza a feszültséget.



1.25. ábra: Potméter LED-es áramkörben

A LED lábain és a potméter két bekötött lábán multiméterrel lemérhető az ellenállás-változás, miközben tekerjük a potmétert. Állítsuk a multimétert 20 V állásba! Mivel voltot mérünk, a feszültségváltozást fogjuk látni a kijelzőn.

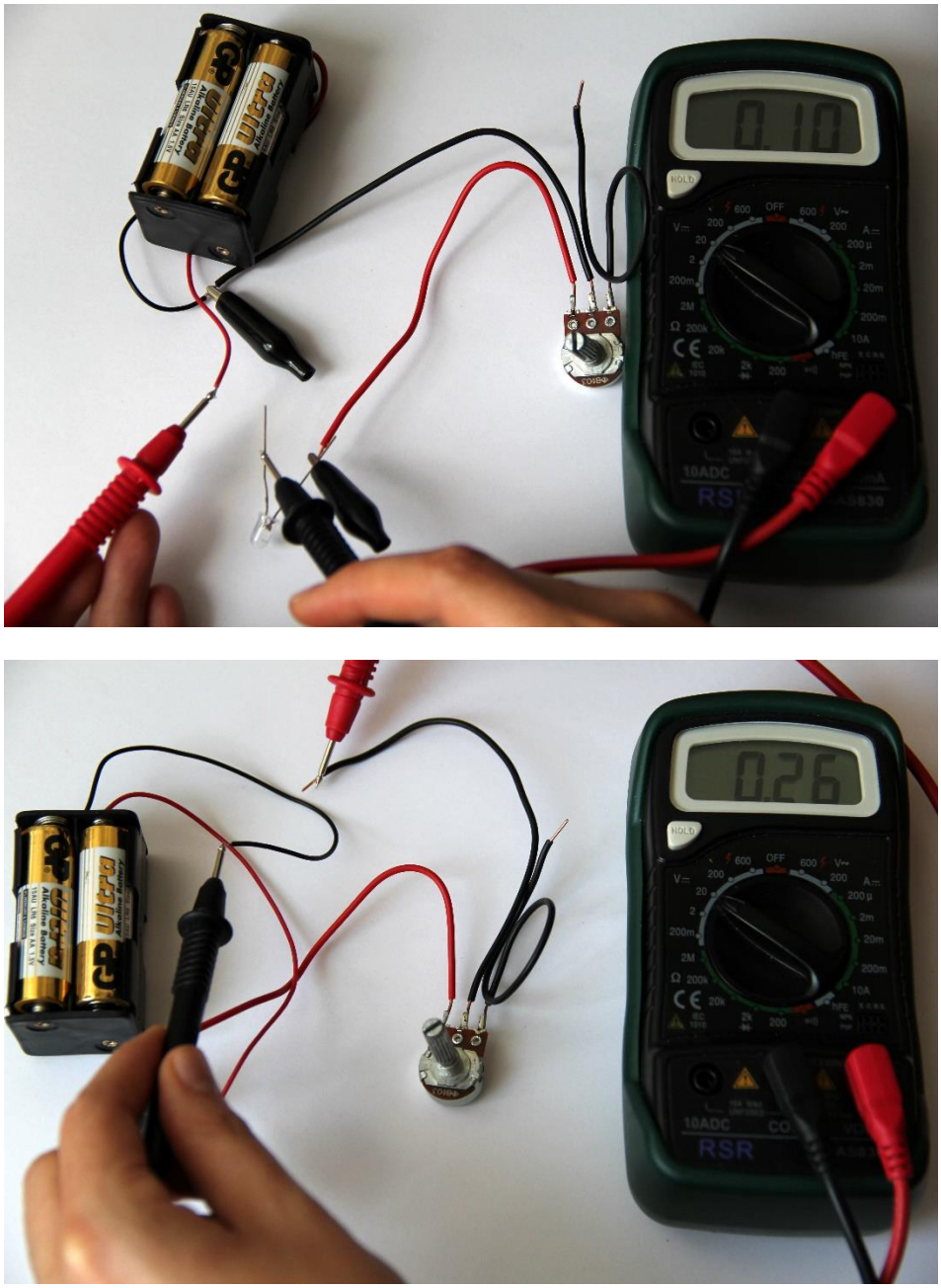
 © Harsányi Réka, Juhász Márton András



1.26. ábra: Multiméterrel lemérhető a feszültségváltozás a potméter lábain

1.2.3.4. Amper és áramerősség

Ha áramerősséget szeretnénk mérni a multiméterrel, át kell kapcsolni milliamper mérésre. Tekerjük a multimétert a maximum értékre. Némelyiken ilyenkor a piros kábelt is át kell dugni az mA csatlakozóba. Mérjük le az áramerősséget: helyezzük a piros mérőtüskét az elem pozitív végéhez, a LED pozitív szárát kössük ki az áramkörből és érintsük hozzá a fekete mérőtüskét, így az áram a multiméteren keresztül jut el a LED-hez. Változtassuk a potméter értékét és mérjük újra! Látható, hogy minél magasabb az ellenállás, annál kisebb az áramerősség értéke. Az áramkör bármely alkatrészénél elvégezve a mérést azt tapasztaljuk, hogy az áramerősség megegyezik. Ez sorba kötött alkatrészek esetén igaz csak, párhuzamosan kötötteknél eltérő lehet az érték. A következő fejezetben lesz szó arról, mit jelent ez a két fogalom.



1.27. ábra: Ampermérés digitális multiméterrel

AZ ÁRAMERŐSSÉGRŐL

A vezetéken áthaladó elektronok mennyiségét az áramerősséggel határozzuk meg. Ha a folyadék-analógiánál maradunk, minél kisebb a tartály alján lévő lyuk (ez a vezeték keresztmetszetét szimbolizálja), amin a víz kiáramlik, annál nagyobb az áramlás (ami az áramerősséget jelenti).

 © Harsányi Réka, Juhász Márton András

Az áramerősség jele: I, mértékegysége: A (amper).

1 amper az áramerősség, ha az adott vezeték keresztmetszetén 1 másodperc alatt 6,24 x 1018 darab elektron halad át, ami 1 coulombnak felel meg.

Amper átváltása:
0,001 amper – 1 milliamper – 1 mA
0,01 amper – 10 milliamper – 10 mA
0,1 amper – 100 milliamper – 100 mA
1 amper – 1.000 milliamper – 1A

1.2.3.5. Ohm-törvény

Az előző méréseknél megfigyelhettünk egy összefüggést az ellenállás, az áramerősség és a feszültség között. 6 V feszültséget használtunk, amikor 3 K ellenállást mértünk, akkor az áramerősség 2 mA volt. Viszont 1 K ellenállás esetén az áramerősség 6 mA volt. Észrevehető, hogy az ellenállás és az áramerősség értékeinek szorzata mindig ugyanannyi, és a feszültségértékkel egyenlő. Ez az Ohm-törvény.

AZ OHM-TÖRVÉNY

A feszültség, az ellenállás és az áramerősség közötti összefüggést írja le: a feszültség kiszámítható az ellenállás és az áramerősség szorzataként. Az áramerősség (I) a feszültséggel (U) arányosan nő vagy csökken adott keresztmetszetnél, ha az ellenállás értéke változatlan marad.

Képletek: U = I x R (volt = kiloohm x milliamper)
További képletek: I = U / R, R = U / I
Ahol az R az ellenállás, I az áramerősség és U a feszültség.

Ennek segítségével már kiszámolható, hogy milyen ellenállást kell használni egy alkatrész megvédésére az adott feszültségű áramkörben. Például a LED adatlapján azt látjuk, hogy 2,5 volt feszültséggel és 20 mA áramerősséggel működik. Tudjuk, hogy 6 voltról üzemel az áramkör, ami 4 db ceruzaelemnek fele meg. 6 – 2,5 = 3,5. Tehát olyan ellenállásra lesz szükség, ami 3,5 volttal csökkenti a feszültséget, így a LED-hez már csak 2,5 volt jut el. Az Ohm-törvény alkalmazásával váltsuk át a milliampert Amperbe!

V = 3,5 V
I = 0,02 A
R = V / I = 3,5 / 0,02 =175 ohm

Ez nem szabályos érték, ezért kerekítve egy 180 vagy 220 ohmos ellenállást tudunk beépíteni, ezek elég közeli értékek még.

1.2.3.6. Watt és teljesítmény

A TELJESÍTMÉNYRŐL
Az elektronoknak az áramkörben haladva egyfajta ellenállást kell leküzdeniük és eközben munkát végeznek, ezt wattban mérjük. Az izzók is wattban vannak hitelesítve, innen lehet ismerős a kifejezés.

A teljesítmény jele: P, mértékegysége: W (watt).

Számítása: P = U x I (Watt = Volt x Amper).
Ebből eredő képletek: U = P / I, I = P / U.
Ahol P a teljesítmény, I az áramerősség és U a feszültség.

Watt átváltása:
0,001 watt – 1 milliwatt – 1 mW
0,01 watt – 10 milliwatt – 10 mW
0,1 watt – 100 milliwatt – 100 mW
1 watt – 1000 milliwatt – 1 W

Korábban az ellenállások tulajdonságainál már beszéltünk a wattról, de most már ki is tudjuk számolni, milyenre van szükségünk, 0,25 wattosra, 0,50 wattosra vagy esetleg nagyobb értékűre. A leckék során a 0,25 wattos ellenállás megfelelő lesz. A korábban példaként felhozott 2,5 V és 20 mA fogyasztású LED-es áramkörre alkalmazva számoljuk ki a szükséges teljesítmény értékét! Mivel 6 V-ról üzemeltetjük, minimum 3,5 V-ot kell az ellenállásnak lecsökkentenie.
V = 3,5 V
I = 20 mA = 0,002 A
P = U x I = 3,5 x 0,02 = 0,07 watt. Mivel ilyen értékű gyárilag nincs forgalomban, a 1/8 wattos megfelelő lesz, de a leckék során 1/4 watt teljesítményű ellenállásokat is használhatunk. A lényeg, hogy nem lehet kisebb annál, amit kiszámoltunk! Sokkal nagyobb értékű használata nem ajánlott, mert esetleg nem jut elég áram az alkatrészhez.

1.2.3.7. A mértékegységek decimális szorzói

Mindegyik váltóérték rendelkezik egy névvel, egy rövidítéssel, amit a mértékegység rövidítése elé írunk, és egy értékkel, ami a 10 valamelyik pozitív vagy negatív hatványa, ezzel kell beszorozni a mértékegységet, hogy megkapjuk az átváltott értéket. Ezek a következőek:

 © Harsányi Réka, Juhász Márton András

- tera: $T – 10^{12}$
- giga: $G – 10^{9}$
- mega: $M – 10^{6}$
- kilo: $k – 10^{3}$
- hekto: $h – 10^{2}$
- deka: $dk – 10$
- deci: $d – 10^{-1}$
- centi: $c – 10^{-2}$
- milli: $m – 10^{-3}$
- mikro: $m – 10^{-6}$
- nano: $n – 10^{-9}$
- piko: $p – 10^{-12}$
- femto: $f – 10^{-15}$
- atto: $a – 10^{-18}$

2. Soros és párhuzamos kötés; Kapcsoló, relé; A kondenzátor és a teljesítmény fogalma;

2.1. Alkatrészek soros és párhuzamos bekötése

A soros kapcsolás azt jelenti, hogy egymás után, sorban kapcsolódnak az alkatrészek egymáshoz. Ezzel szemben párhuzamos kapcsolás esetén az alkatrészek egyik oldala a pozitív, másik oldala a negatív pólusra van kötve, így külön áramköröket alkotnak. -

Ha nincs kéznél adott értékű ellenállás, több értékből is összeállítható a szükséges ellenállás. Ha nagyobb feszültségű vagy épp áramerősségű áramforrásra van szükség, az elemek megfelelő típusú összekapcsolásával előállíthatóak. Az sem mindegy, hogy több LED-et sorosan vagy párhuzamosan akarunk-e kötni, mivel ilyenkor összeadódik a feszültség vagy az áramerősség értékük. Ismerkedjünk meg ezekkel a lehetőségekkel.

2.1.1. Ellenállások

Ha sorba kötünk két vagy több ellenállást, az értékük összeadódik, amit ohmban határozunk meg. $R = R_1 + R_2 + R_3$ stb.

Mivel ismerjük az ellenállásértékét és a feszültséget, így ki tudjuk számítani az áramerősséget. A feszültség, amivel számolnunk kell, nem mindig egyezik meg az áramforrás feszültségével, azaz a tápfeszültséggel, mert lehet, hogy már áthaladt az áram egy másik ellenálláson. Az ellenállások előtti feszültséget eredő feszültségnek nevezzük. Soros kötés esetén csak a feszültség csökken az áramkörben, az áramerősség ugyanannyi marad.

$U = U_1 + U_2$

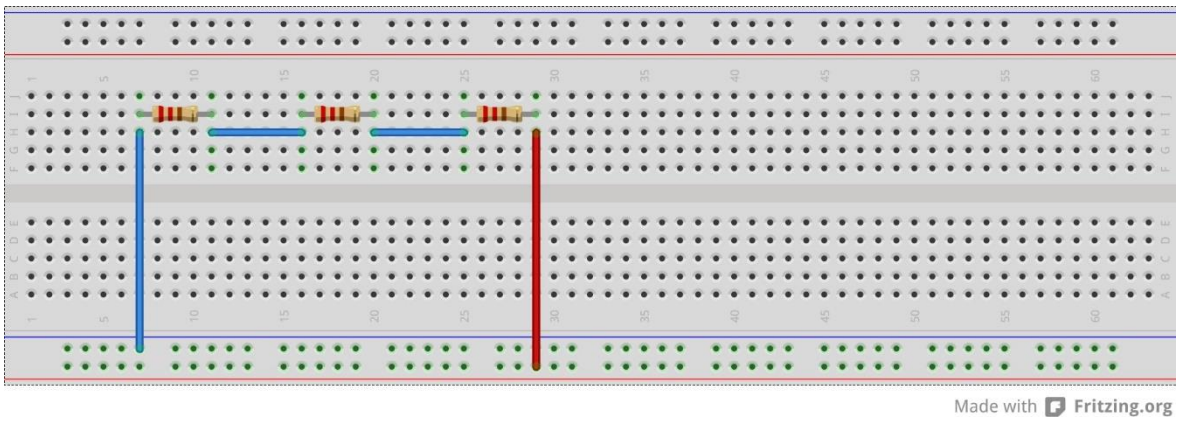
$I = I_1 = I_2$

Az adott ellenállásra jutó feszültséget (U_1, U_2, stb.) úgy tudjuk kiszámítani, hogy ez eredő feszültséget megszorozzuk az adott alkatrész ellenállás érték ($R_{kiszemelt}$-tel jelölve) és az eredő ellenállásréték (R_{soros}-sal jelölve) hányadosával.

$$U_{R_{kiszemelt}} = U_{be} \cdot \frac{R_{kiszemelt}}{R_{soros}}$$

$$R_{soros} = R_1 + R_2 + \cdots + R_n$$

© Harsányi Réka, Juhász Márton András



2.1. ábra: Ellenállások sorosan kötve, ilyenkor a feszültségük összeadódik

Ha párhuzamosan összekapcsoljuk az ellenállásokat, tehát a lábaikat egymással csokorba kötjük, akkor mindegyikre ugyanannyi feszültség jut, az áramerősség viszont eloszlik közöttük, az alkatrészek ellenállás értékének függvényében.

$U = U_1 = U_2$

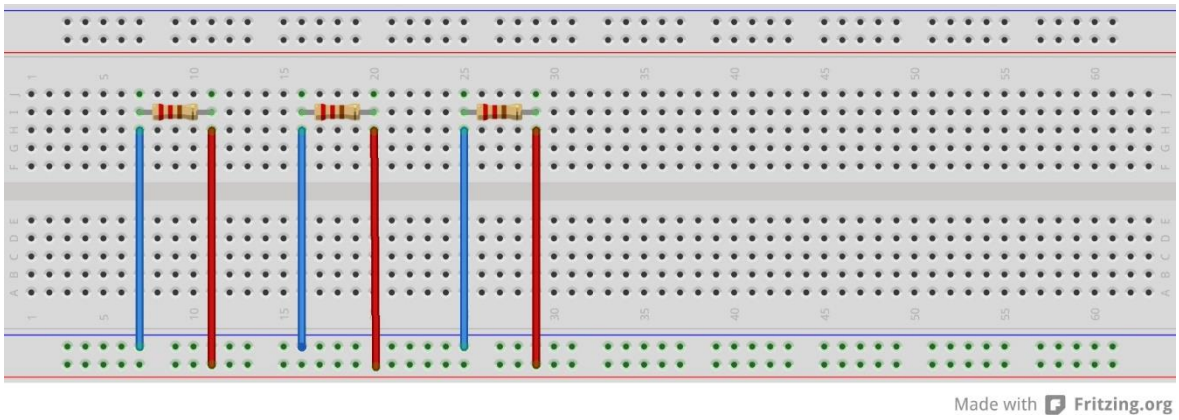
$I = I_1 + I_2$

Az adott alkatrészre jutó áramerősséget (I_1, I_2 stb.), hogy az eredő feszültséget megszorozzuk az eredő vezetés érték (ez az egyes ellenállások vezetésének összege, $R_{parhuzamos}$-sal jelölve) és az adott ellenállás vezetés értékének hányadosával ($R_{kiszemelt}$-tel jelölve). A vezetés az ellenállás reciprokaként számítható ki (1/R).

$$R_{parhuzamos} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}}$$

Ezt behelyettesítve megkapjuk az adott ellenállásra jutó áramerősséget, ahol az U_{be} az eredő feszültséget jelenti:

$$I = U_{be} \cdot \frac{\frac{1}{R_{kiszemelt}}}{\frac{1}{R_{parhuzamos}}} = U_{be} \cdot \frac{R_{parhuzamos}}{R_{kiszemelt}}$$

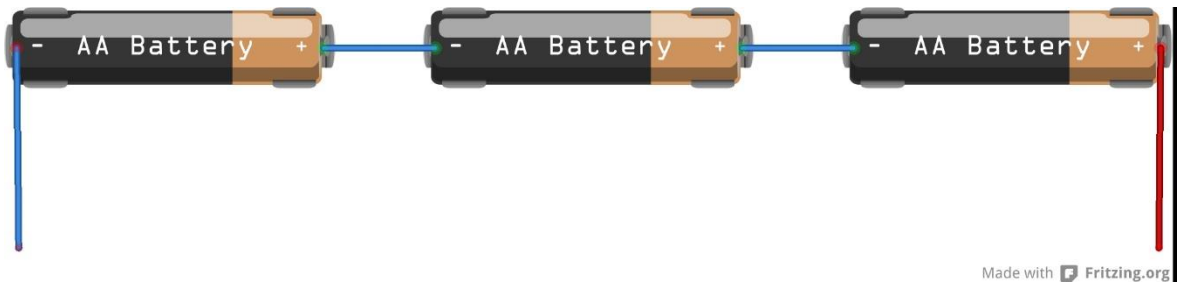


2.2. ábra: Ellenállások párhuzamosan kötve, ilyenkor áramerősségük összeadódik

2.1.2. Áramforrások

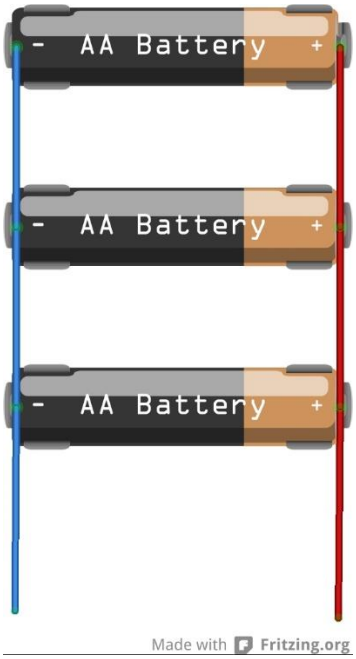
Az áramforrások soros kapcsolása úgy történik, hogy az egyik pozitív sarkához kötjük a másik negatív oldalát, és a többit is így tovább, tehát az elsőnek a negatív lába, az utolsónak a pozitív lába szabad marad. Azonos feszültségű és áramerősségű elemeket lehet így kötni, ha eltérőeket használunk akkor tönkre mennek.

Ebben az esetben az egyes elemek **feszültsége összeadódik**. Főként galvánelemeknél és akkumulátoroknál alkalmazzák a soros kötést.



2.3. ábra: Elemek sorosan kötve

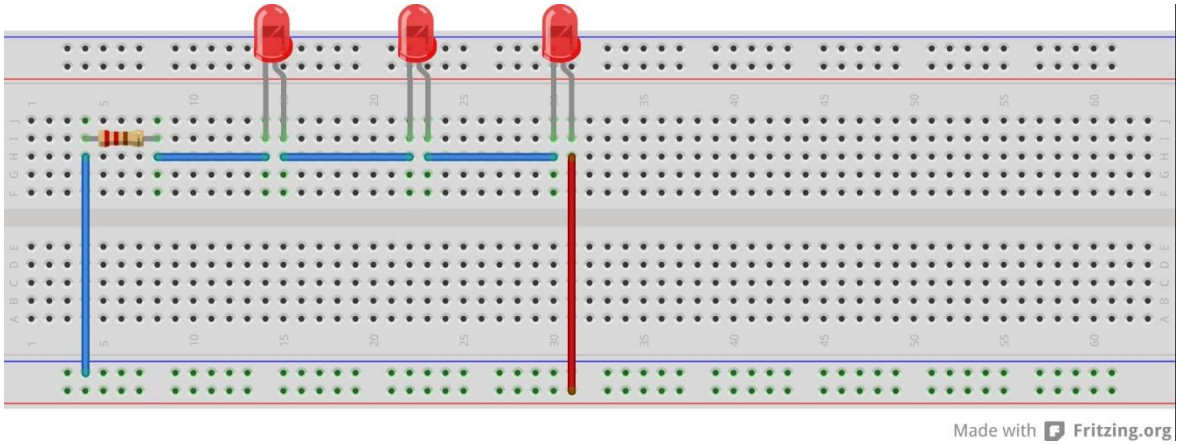
Áramforrások párhuzamos kötése esetén a pozitív sarkokat egybe kötik egy gyűjtővezetékhez, és ugyanígy a negatív oldalakat is. Ebben az esetben a feszültség megegyezik az egyes elemek feszültségével, de az **áramerősségük összeadódik.** Ebben az esetben is fontos, hogy egymással megegyező áramforrásokat használjunk.



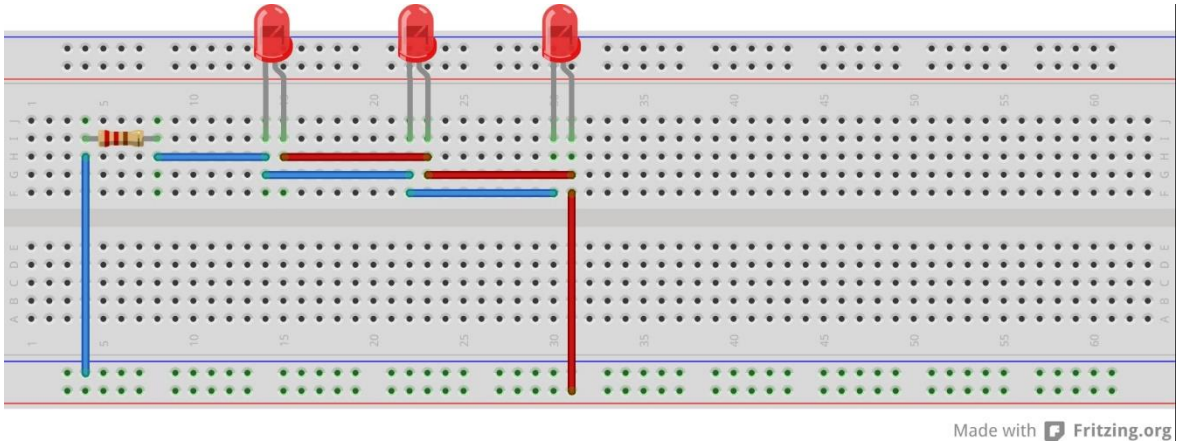
2.4. ábra: Elemek párhuzamosan kötve

2.1.3. LED-ek

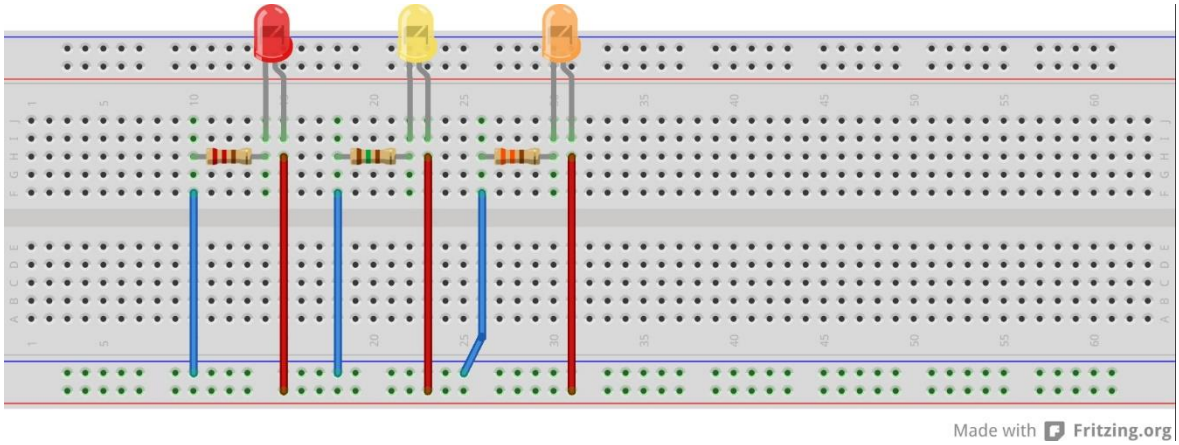
A LED-et is – mint a legtöbb alkatrészt – ellenállással kell megvédeni a túlfeszültségtől, azaz attól, hogy az üzemeltetéséhez szükségesnél nagyobb érkezzen hozzá, így biztosíthatjuk a megfelelő működését. A polaritásra is ügyeljünk, nézzük meg, melyik a pozitív és melyik a negatív lába! Az adatlapjából keressük ki, hogy mekkora feszültséggel és áramerősséggel működik, mivel ezek alapján ki tudjuk számolni, hogy mekkora ellenállásra lesz szükségünk, illetve azt is, hogy a sorba vagy párhuzamosan kötött LED-jeinknek mekkora lesz a fogyasztása! Ugyanaz igaz náluk is, ha sorba kötjük a LED-eket, a feszültség eloszlik, de az áramerősség azonos marad, míg párhuzamos kötés esetén az feszültség ugyanaz, de az áramerősség eloszlik azonos színű LED-ek esetén. Ha különböző színeket kötünk párhuzamosan, akkor mindegyikhez egy megfelelő értékű ellenállást kell bekötni.



2.5. ábra: LED soros bekötése



2.6. ábra: Egyforma LED-ek párhuzamos bekötése



2.7. ábra: Különböző LED-ek párhuzamos bekötése

 © Harsányi Réka, Juhász Márton András

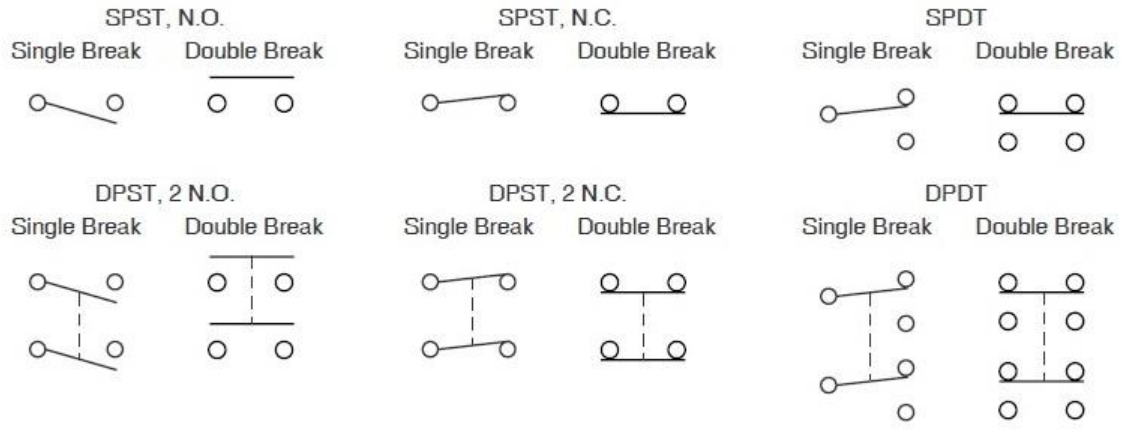
2.2. Kapcsoló és relé működése

Mielőtt összeraknánk a következő áramkört, ismerkedjünk meg két újabb alkatrésszel, a kapcsolóval és a relével. Ezek segítségével módosíthatjuk az áramkörünket, nyithatjuk vagy zárhatjuk a részeit.

A **kapcsolókat** működésük szerint csoportosítjuk, lehetnek mechanikusan, elektromágnessel, hő hatására vagy elektronikusan kapcsolhatóak.

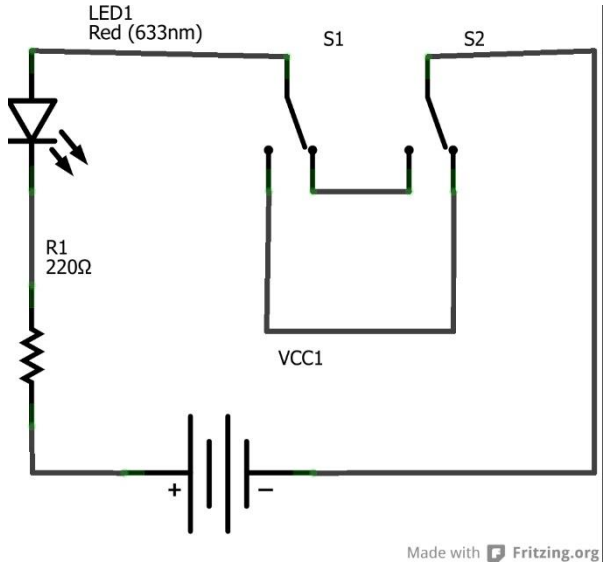
A mechanikus kapcsolóknak három típusát különböztetjük meg, melyek mindig az alapállapotukra vonatkoznak. Lehetnek záró-, nyitó- vagy váltókapcsolók, mint ahogy a nevük is mutatja, vagy zárják, vagy nyitják az áramkört, vagy átkapcsolnak két áramkör között.

Fontos tudni, hogy milyen az aktuális áramkörünk feszültsége és áramerőssége, mivel a kapcsolók egyik jellemző tulajdonsága, hogy mekkora feszültséget bírnak el nyitott állapotban, illetve mekkora áramerősség folyhat át rajtuk zárt állapotban anélkül, hogy megsérülnének. Mindkét érték függ az áram frekvenciájától . A frekvencia az elektromos jel egy másodpercnyi egységre jutó hullámhosszát jelenti. Mértékegysége a hertz (Hz). A kapcsolóknál általában egyenáramon és 50 Hz-en van megadva a korábban említett megengedett átfolyó áramerősség mértéke. Ellenállásuk zárt állapotban olyan kicsi, hogy elhanyagolható, ha mégis jelentősebb, akkor ez jelezve van rajtuk. Élettartamuk pedig a hibamentes átkapcsolások számát adják meg. A pólus a kapcsoló kivezetésének megnevezése. Lehet egy vagy több pólusú is. A megfelelő kapcsoló kiválasztásánál a legfontosabb adatok a vezérlés módja mellett a terhelhetőség, valamint a pólusok és az állások száma. Az angol rövidítést használjuk fajtájának megnevezésére, illetve mindegyiknek van egy típusjele is, ami a tokozáson szerepel.

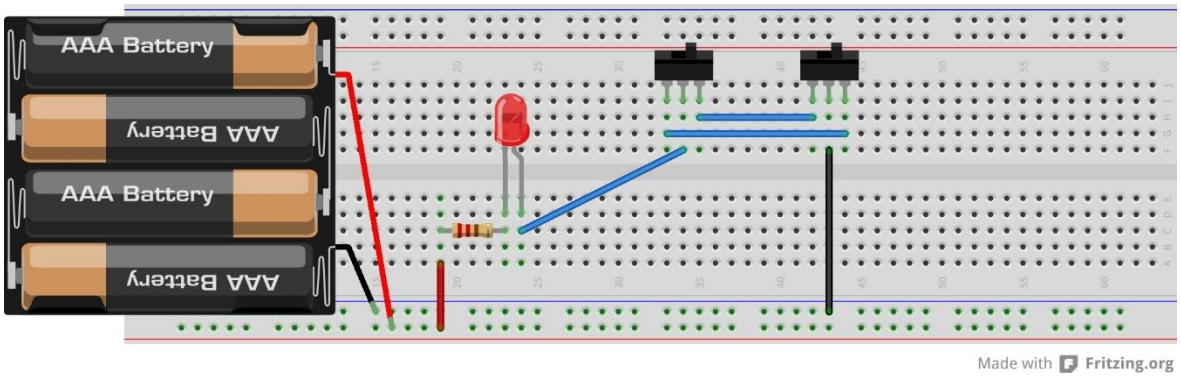


2.8. ábra: *Kapcsolók rajzjelei vegyesen, jobb oldalon a felső kettő a nyomógombok rajzjele (N.O. = Normally Open / Alap állapotban nyitott; N.C. = Normally Closed / Alap állapotban zárt) Egyéb kapcsolófajták is léteznek, de ezeket nem fogjuk használni.*

Az átlagos billenőkapcsolók csak az egyik irányba zárnak (pl. a házban a villanykapcsoló), ezeket egyállású kapcsolónak nevezzük (rövidítése SPST, egyállású, egyirányú). Vannak kétpólusú kapcsolók (SPDT), melyek mindkét irányban tudnak áramkört zárni (átkapcsoláshoz jó). (A következő példához ilyet kell használnunk.)



2.9. ábra: Kapcsolási rajz a két SPDT kapcsolót tartalmazó LED-es áramkörhöz



2.10. ábra: Kísérlet két SPDT kapcsolóval

Rakjuk össze a képen szereplő áramkört! Ehhez két SPDT kapcsolóra, egy LED-re és egy 220 ohmos ellenállásra lesz szükségünk. Amikor átkapcsoljuk a kétállású kapcsolót egyik vagy másik irányba, akkor ugyanabban az irányban zárja az áramkört a középső lábhoz. Ha a két kapcsoló befelé áll, akkor a rövidebb kék dróton folyik az áram, ha mindkettő kifelé áll, akkor a hosszabb kék dróton. Ha azonos irányban állnak, akkor nincs zárva az áramkör, ezért nem ég a LED.

A nyomógombok is mechanikai úton működnek, ami azt jelenti, hogy kézzel meg kell nyomni őket. Lehet visszaálló típusú, melyet, ha elengedünk, akkor az eredeti állapotába kerül, illetve lehet tapadó típusú, amikor meg kell nyomni, hogy visszaálljon eredeti állapotába.

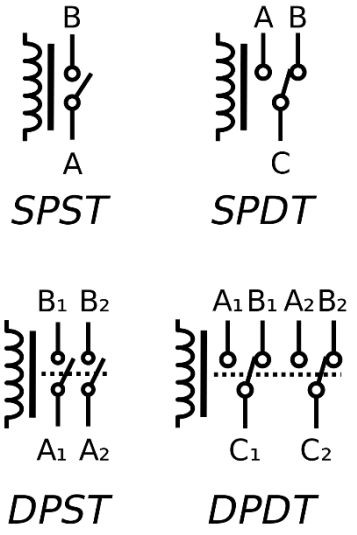


2.11. ábra: NYÁK nyomógomb

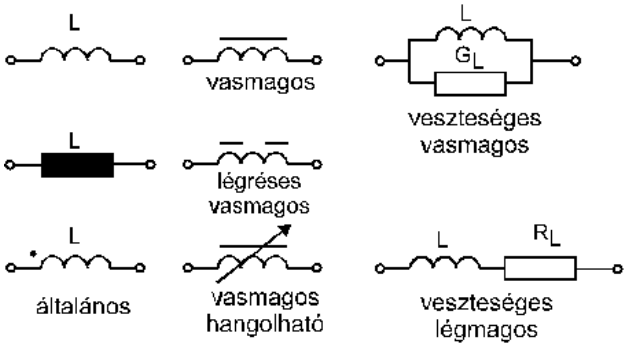
Kapcsolók és nyomógombok jelöléséhez tartozik még az alapállapota és a pillanatnyi tulajdonsága, azaz a benyomott vagy átkapcsolt állapota, ami zárójelbe van téve:

- OFF-(ON) – KI-(BE): amikor lenyomod, csak akkor érintkezik;
- ON-(OFF) – BE-(KI): fordítva működik, lenyomva nyitja az áramkört;
- ON-(OFF)-ON – BE-(KI)-BE: középső állásban van kikapcsolt állapotban, két irányba lehet innen zárni a kapcsolót, ami tehát háromállású.

A **relék**, vagy más néven jelfogók olyan speciális érintkezők, amik elektromágneses módon kapcsolhatóak, hihetetlenül gyorsan. Áramkörök közötti átkapcsolásra alkalmasak. Előnye, hogy a vezérlő áramkör a többitől el van választva, így a relé alkalmas az áramköri alkatrészek megvédésére, mivel nyugalmi állapotban nem engedi át az áramot. Egymástól eltérő felépítésűek léteznek éppúgy, mint a kapcsolóknál. A rá jellemző adatok hasonlóak, mint a kapcsolóknál, tehát, hogy egyen- vagy váltóárammal működik-e, valamint, hogy mekkora a feszültsége és az ellenállása. Ugyanúgy az angol rövidítést használják a különböző típusok megnevezésére, mint a kapcsolóknál. Az ötödik fejezetben az aktuátorok kapcsán még találkozunk ezzel az alkatrésszel. *(**Aktuátor**: elektromos árammal működtetett beavatkozó elem, amely képes valamilyen irányító jelnek megfelelő hatás kifejtésére.)*



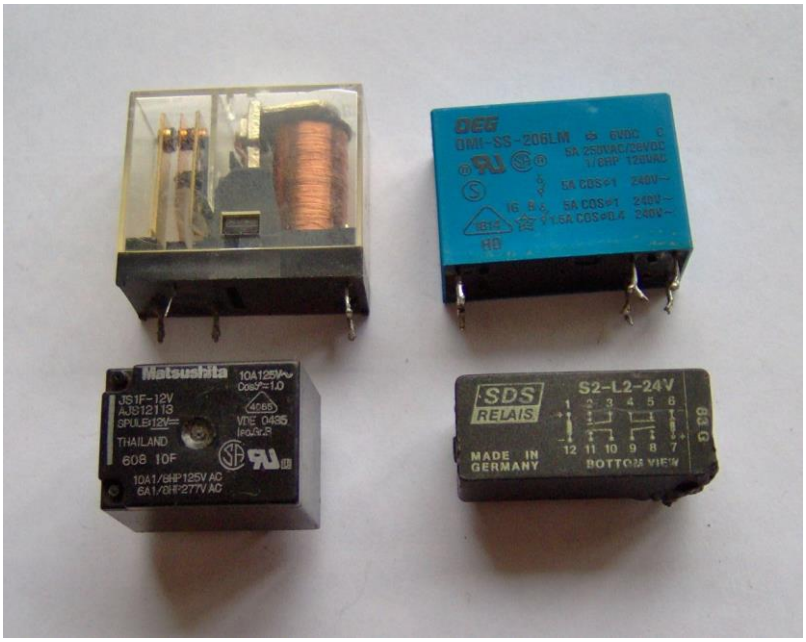
Relék rajzjelei



Tekercsek rajzjelei

A relék rajzjelében a bal szélen a vasmagos tekercs rajzjelét látod, a jobb oldalon a különféle kapcsolókat. Azt, hogy a relé melyik lába tartozik a tekercshez vagy a kapcsolóhoz, az adatlapjából tudjuk kikeresni.

 © Harsányi Réka, Juhász Márton András



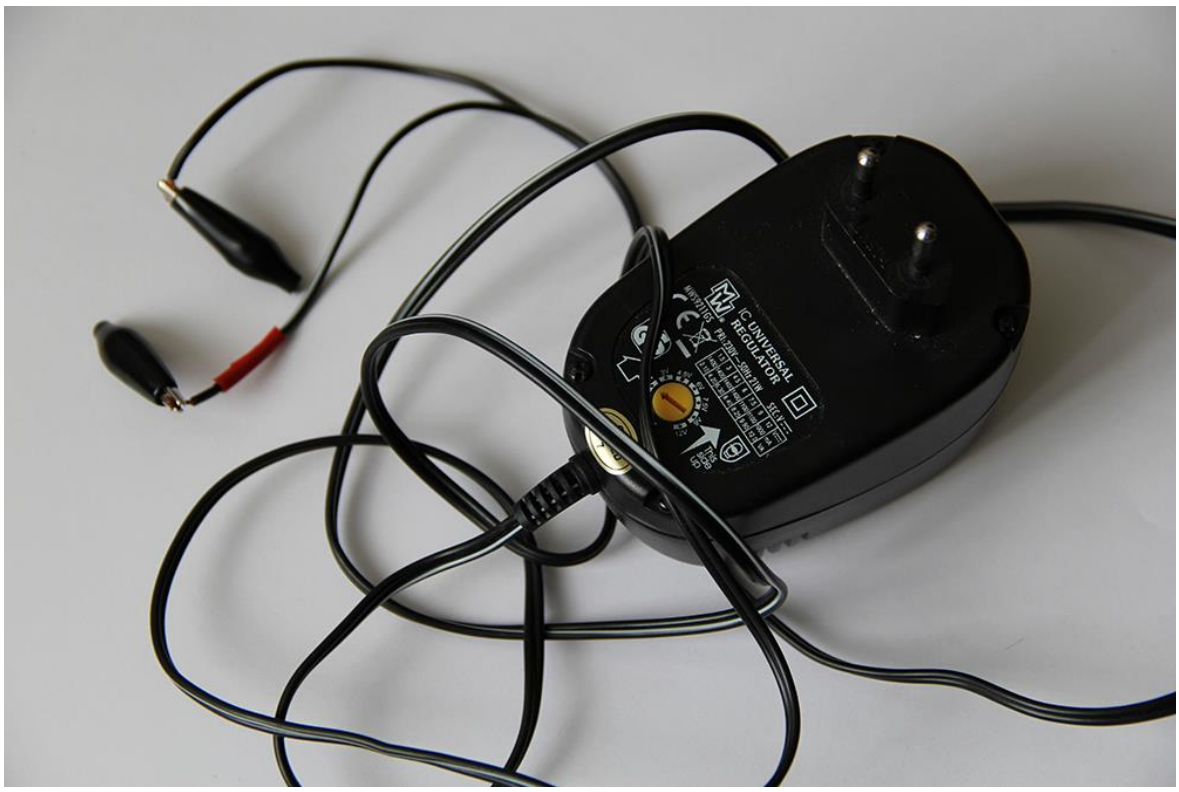
2.12. ábra: Relé, más néven jelfogó. Az elsőn látható, hogy belül egy tekercs található, ami körül feszültség hatására mágneses tér keletkezik, ez mozgatja el azt a hajlékony fémcsíkot, ami az egyik vagy a másik áramkört zárja.

A relé belsejében egy vasrúdra csavart rézdrót tekercs van. Amikor végigfut az áram a tekercsen, a vasrúd mágneses erőt bocsát ki, ami magához húzza a rugalmas fémlapot, mely kapcsolóként funkcionál, zárja az áramkört. Ha megszűnik a feszültség, a fémlap visszakerül eredeti állapotába, így nyitja az adott áramkört. Egyenáramú fajtájánál mindkét irányban keresztülfolyhat az áram, de a biztonság kedvéért meg kell nézni az adatlapján, hogyan kell bekötni. A jellemző tulajdonságai szerint lehet kiválasztani a számunkra megfelelőt:

- a tekercs feszültsége: ennyi feszültséget kaphat maximum a relé;
- a vezérlőfeszültség: az a minimumérték, amin a tekercs zárja a kapcsolót;
- az üzemeltetéshez szükséges áramerősség: a tekercsnek az általában milliamperben megadott áramfogyasztása (néha milliwattban fejezik ki);
- kapcsolási teljesítmény: az a maximális áramerősség, amivel a relében lévő kapcsolók működnek.

Most rakjunk össze egy áramkört kapcsoló, relé, ellenállás és LED-ek felhasználásával, áramforrásként pedig használjunk egy 12 voltos egyenáramú adaptert. Az adapter egy olyan átalakító, ami a nagyfeszültségű váltóáramból kisfeszültségű egyenáramot csinál. Léteznek állítható feszültségűek is, például 3–12V között működőek, különféle végződésű dugókkal. Ha ezt levágjuk, a két drótot beköthetjük, a jelölt szál (pl. fehér csík van rajta) a

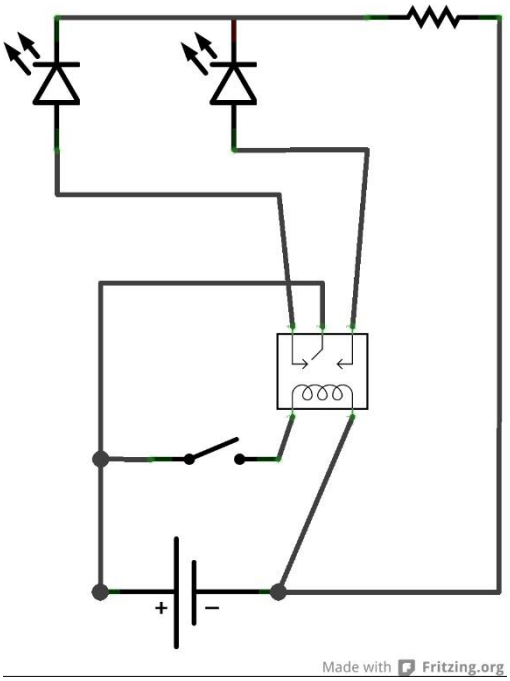
pozitív pólusú. Kézzel sodorjuk össze a végét. Ha szétállnak a szárak, krokodil-csipesszel hozzátoldhatunk egy kis darab tömör végű drótot, hogy bele tudjuk tenni a próbapanelbe. Fontos, hogy a két vége ne érjen össze!



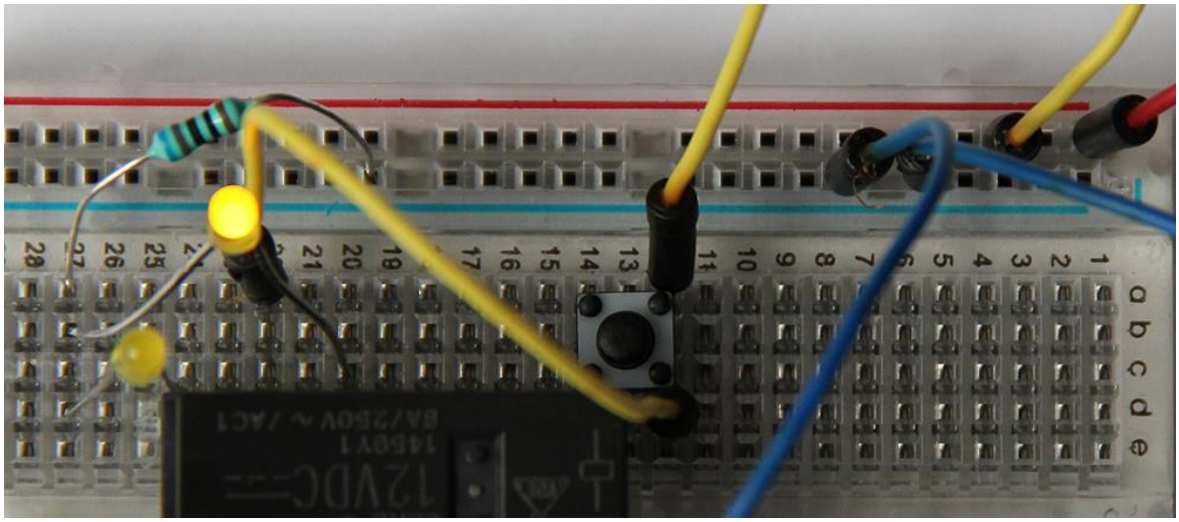
2.13. ábra: Adapter levágott véggel, a fehér csíkkal jelölt vezeték a pozitív oldal

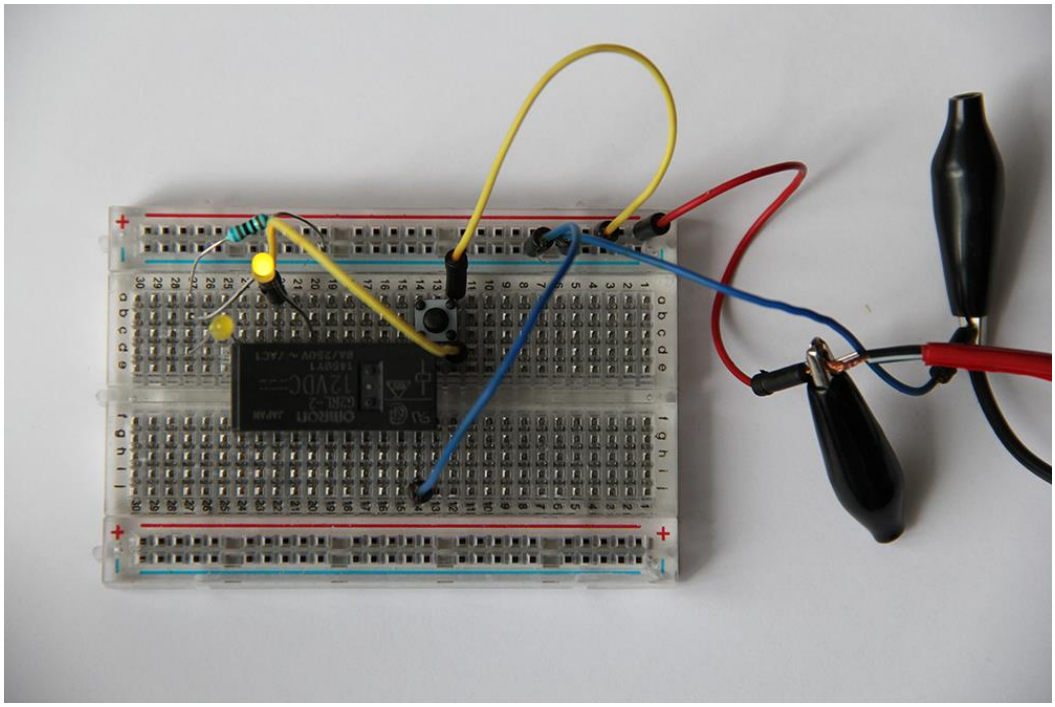
Tehát az áramkörhöz szükség lesz 2 db LED-re, 1 db 12 V egyenáramú DPDT relére, 1 db 680 Ω ellenállásra, 1 db SPST nyomógombra és a preparált adapterre. Az adaptert 12 voltra kell állítani, mivel ezen üzemel a relé, ennél kevesebb feszültség nem tudja bekapcsolni, valamint drótokra és dugaszolós próbapanelre.

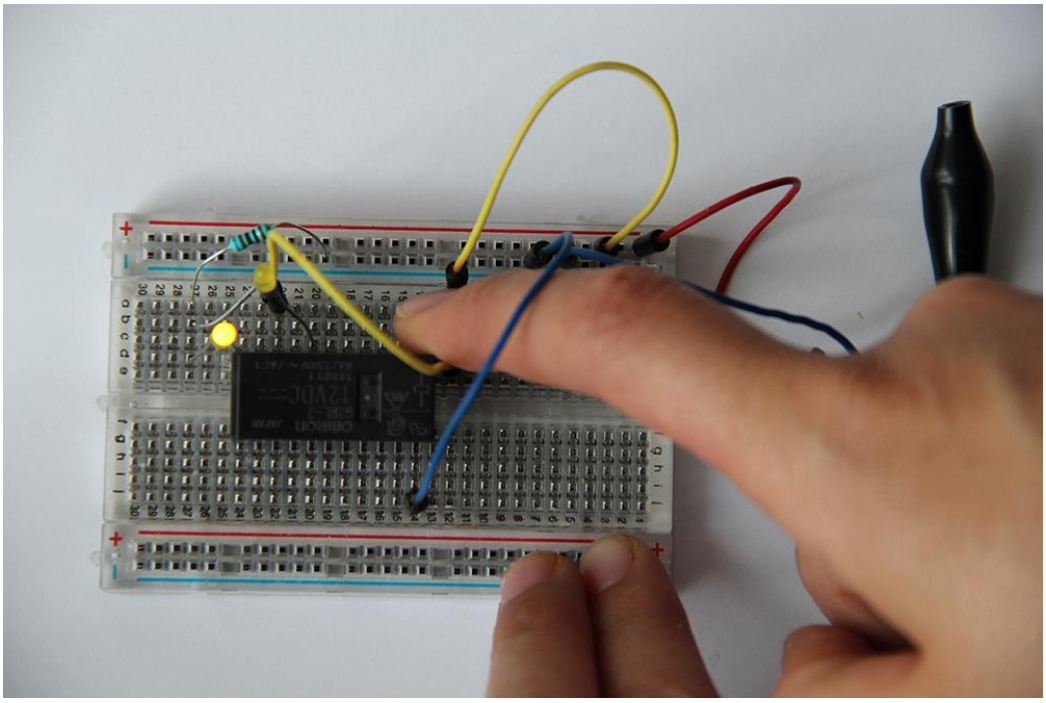
 © Harsányi Réka, Juhász Márton András



2.14. ábra: Relés áramkör kapcsolási rajza







2.15. ábra: Relé nyomógombbal működtetve

 © Harsányi Réka, Juhász Márton András

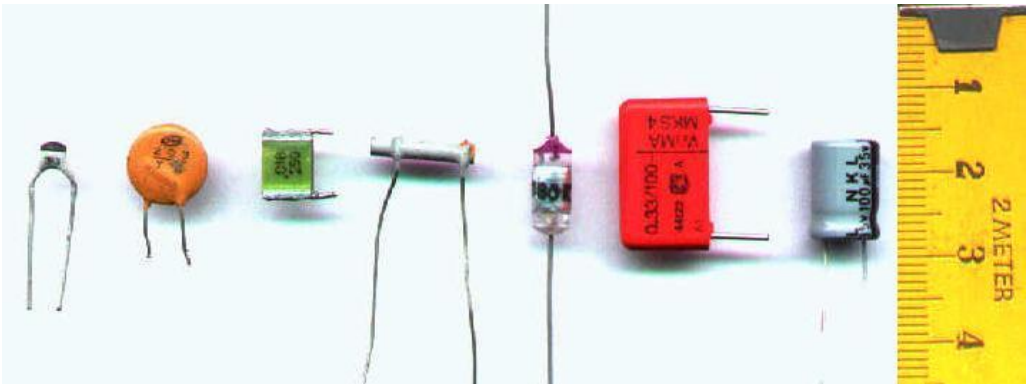
Nézzük meg, mi történik! Ha megnyomjuk a gombot, azzal áram alá helyezzük a relét, ami működésbe lép és a relén belül található kapcsoló nyitja az első és zárja a második áramkört. Az LED-ek külön áramkörbe vannak kötve, így vagy az egyik, vagy a másik LED világít attól függően, hogy megnyomjuk-e a gombot.

2.3. A kondenzátor működése

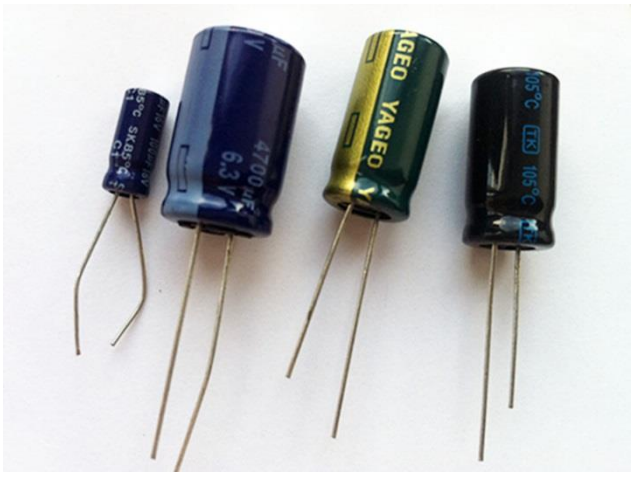
A **kondenzátor** egy olyan elektromos alkatrész, ami töltések tárolására alkalmas. Gyorsan töltődnek fel, egyenletes áramellátást biztosítanak az alkatrészeknek, ha lekapcsoljuk az áramforrást, a kondenzátorból fokozatosan ürülnek ki az elektronok. Olyan, mint egy mini akkumulátor, de ezzel nem tudunk hosszú távon feszültséget adni, mint például egy elemmel, mivel feltöltődési és kisülési ideje is rendkívül rövid azokhoz képest. Az energia tárolása a fémrétegek közötti szigetelőkben történik, mivel a fémfelületeken ellenkező előjelű töltéseket halmozunk fel, így feszültségkülönbség jön létre.

Felépítésük és a bennük lévő szigetelőanyag szerint lehet elektrolit vagy kerámia (illetve egyéb pl. levegő, műanyag, transzformátorolaj, de ezek használatát most mellőzzük), kapacitás szabályozhatóság szempontjából fix értékű vagy változtatható. A kerámia kondenzátorok kétoldalt fémezett kerámia lapocskából állnak, ezek kis kapacitásúak. Bármelyik irányba beköthetőek pont úgy, ahogy az ellenállások. A típusukat számmal jelzik, ami alapján online meg tudjuk keresni az értékét. A nagyobb elektrolit kondenzátorok viszont nem köthetőek be bármelyik irányba, polaritásuk az oldalukon van jelölve, általában a negatív oldal.

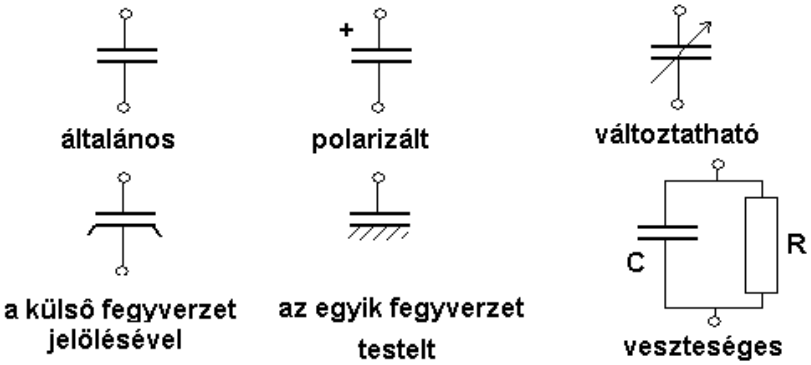
A töltéstárolási képességet farádban (jele: F) mérjük, ami a kapacitás mértékegysége. Lehetnek fix vagy változó kapacitásúak. A maximális feszültség és a farád értéke az elektrolit kondenzátor oldalán fel van feltüntetve. Egy farád meglehetősen nagy kapacitást jelent, általában piko-, nano- és mikrofaráddal dolgozunk.



2.16. ábra: Kondenzátorok



2.17. ábra: Elektrolit kondenzátorok



2.18. ábra: Kondenzátorok rajzjelei

A KONDENZÁTOR FELTÖLTŐDÉSI IDEJÉNEK KISZÁMÍTÁSA

A kapacitás a kondenzátorra jellemző tulajdonság, jele: C, mértékegysége a farád (F).

Számítása: C= Q / U.

Q a kondenzátorban felhalmozódott töltés jele, mértékegysége a coulomb (C).

U a töltések által létrehozott feszültség jele, mértékegysége a volt (V).

Egy coulomb (C) az a töltésmennyiség, amely egy amper (1 A) áramerősség esetén egy másodperc (1 s) idő alatt átfolyik a vezetőn:

1 C = 1 A x 1 s.

Úgy is kiszámítható, hogy a kapacitást a feszültséggel megszorozzuk. A képletet átalakítva Q = C x U. Így 1 C = 1 F x 1 V.

A kondenzátorról leolvasható a kapacitás, azaz, hogy hány farádos. A feszültséget ismerjük.

 © Harsányi Réka, Juhász Márton András

Most nézzük meg, hogyan tudjuk kiszámolni, mennyi idő alatt töltődik fel, illetve merül le a kondenzátorunk. A tárolt töltés mennyiségét kétféleképpen számolhatjuk ki: egyrészt a kapacitást megszorozhatjuk a feszültséggel, másrészt az áramerősséget a feltöltés idejével:

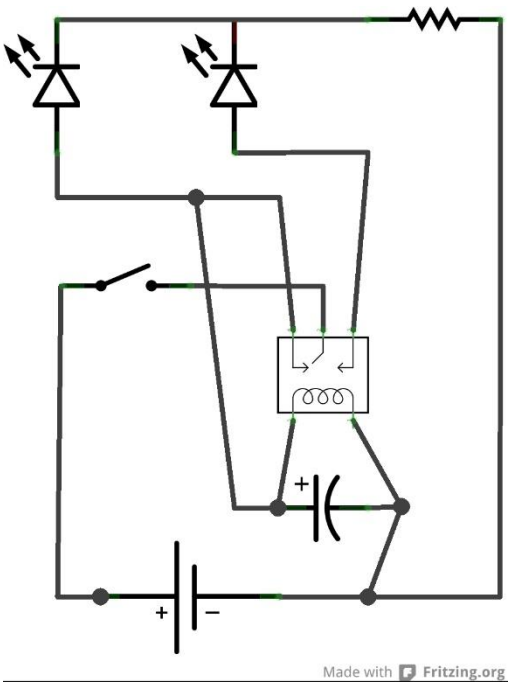
Q = C x U = I x t
t az idő, mértékegysége a másodperc (s).
I az áramerősség, mértékegysége az Amper (A vagy Amp).

Az előző képlet alapján $U = \frac{Q}{C} = \frac{I}{C} x\ t$

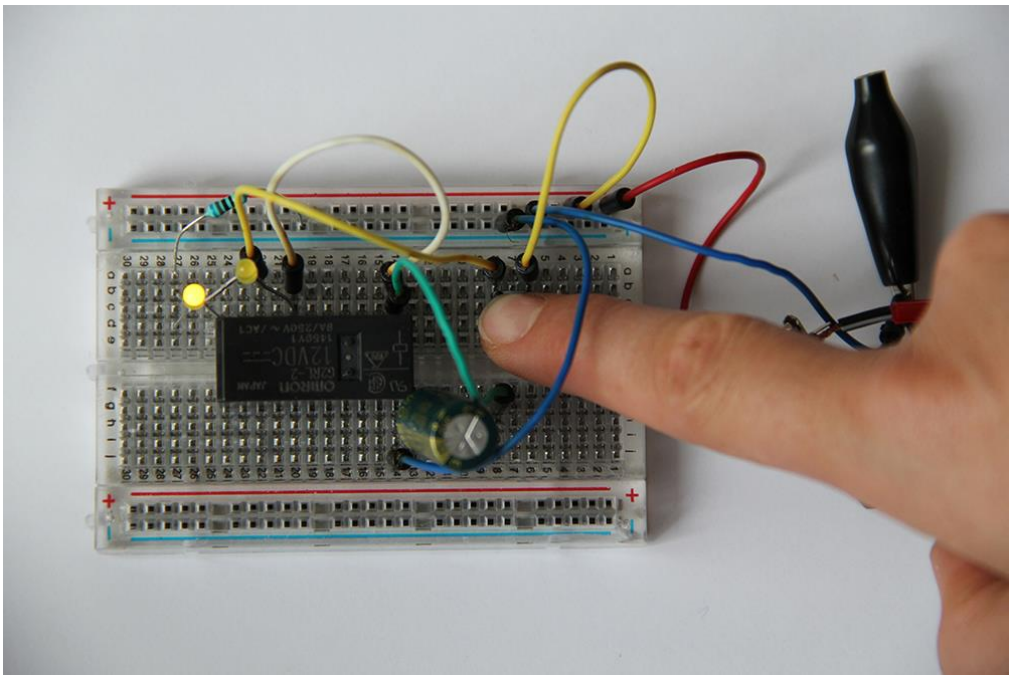
Ha ezt átalakítjuk – mivel ismerjük a feszültséget és a kapacitást – az áramerősséget és a tárolt töltések mennyiségét is ki tudjuk számolni (vagy fordítva), így végül ezekből már az időt is megkaphatjuk:

A farad átváltása:
0,001 nanofarad = 1 picofarad = 1 pF
0,01 nanofarad = 10 picofarad = 10 pF
0,1 nanofarad = 100 picofarad = 100 pF
1 nanofarad = 1,000 picofarad =1,000 pF
0,001 microfarad = 1 nanofarad =1 nF
0,01 microfarad = 10 nanofarad = 10 nF
0,1 microfarad = 100 nanofarad = 100 nF
1 microfarad = 1,000 nanofarad = 1,000 nF
0.000001 farad = 1 microfarad = 1 mF
0,00001 farad = 10 microfarad = 10 mF
0,0001 farad = 100 microfarad = 100 mF
0,001 farad = 1.000 microfarad = 1,000 mF

Módosítsuk a relé és a nyomógomb lábainak bekötését, és szereljünk be egy 1.000 µF-os kondenzátort is, a 2.19 számú ábra szerint!



2.19. ábra: Kondenzátorral kiegészített relés áramkör kapcsolási rajza



2.20. ábra: Kondenzátorral kiegészített relés áramkör

A következőt tapasztaljuk: ha lenyomjuk a gombot, a relé addig zizeg, amíg feltöltődik a kondenzátor, akkor a relén belüli tekercs működésbe lép és átkapcsolja a fémérintkezőt a másik áramkörre, eközben a tekercs nem jut áramhoz, feszültsége csökken, ilyenkor a

fémérintkező visszakapcsol, a tekercs újra feltöltődik, és ismét átkapcsolódik a fémérintkező. Ez a folyamat addig ismétlődik, míg a nyomógombbal áramot juttatunk a tekercsbe. Ha elengedjük a gombot, akkor még a kondenzátorban vannak felhalmozódott töltések, amik fokozatosan ürülnek ki, ezért a folyamat addig nem áll le, míg azok el nem fogynak. Az első LED világít, ha lenyomjuk a gombot, a második LED, amit a kondenzátorhoz kapcsoltunk, oszcillál, felgyullad a fénye és fokozatosan elhalványodik, olyam ütemben, ahogy a kondenzátorból kirülnek a töltések. Az oszcilláció egy folyamat imétlődését jelenti. Ha elengedjük a gombot, megszakítjuk az áramkört, így nem világítanak a LED-ek, mivel nem jutnak el elektronok hozzájuk.

2.4. A tranzisztor működése

A tranzisztoroknak is több fajtája van, de mi csak bipoláris és unijunction (unipoláris) tranzisztorral fogunk dolgozni a következő példában, nézzük át ezeket!

Elnevezése az angol transfer+resistor szóösszetételből ered. Ez volt az első olyan félvezető alkatrész, ami a főáramkör áramát egy vezérlőjel hatására változtatni tudja, azaz az első áramkörben lévő megfelelő kivezetésre küldött feszültség hatására átjutó árammal vezérli a második áramkör alkatrészeit. Lényegében egy vezérelhető ellenállás. Átkapcsolásra vagy jelerősítésre alkalmasak, tehát egyfajta kapcsolóként használhatjuk őket, mivel a vezérlőáram hatására csak adott irányba engedik át az áramot, azaz a feszültséget.

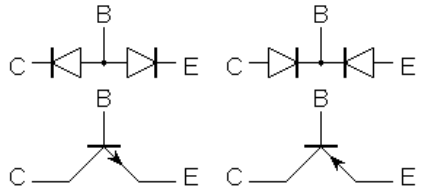
A bipoláris tranzisztorok három rétegből állnak, a rétegek nevét N-nel vagy P-vel jelölik a szennyezettségüktől függően. Ezek a rétegek váltják egymást, így a bipoláris tranzisztor lehet PNP vagy NPN típusú. Hogy mit tartalmaznak ezek a rétegek, arra most nem térünk ki, de fontos, hogy milyen típust használunk, hiszen eltérő módon kell bekötni őket. Különféle tokozásban léteznek a teljesítményüktől függően. Három lába van, egyik a kollektor (C), másik a bázis (B), a harmadik a kibocsátó vagy más néven emitter (E). Ezek a rövidítések szerepelnek a rajzjelen is. A tranzisztor típusszáma a tokozáson szerepel, így az adatlapján meg tudjuk nézni (datasheet) a paramétereit.

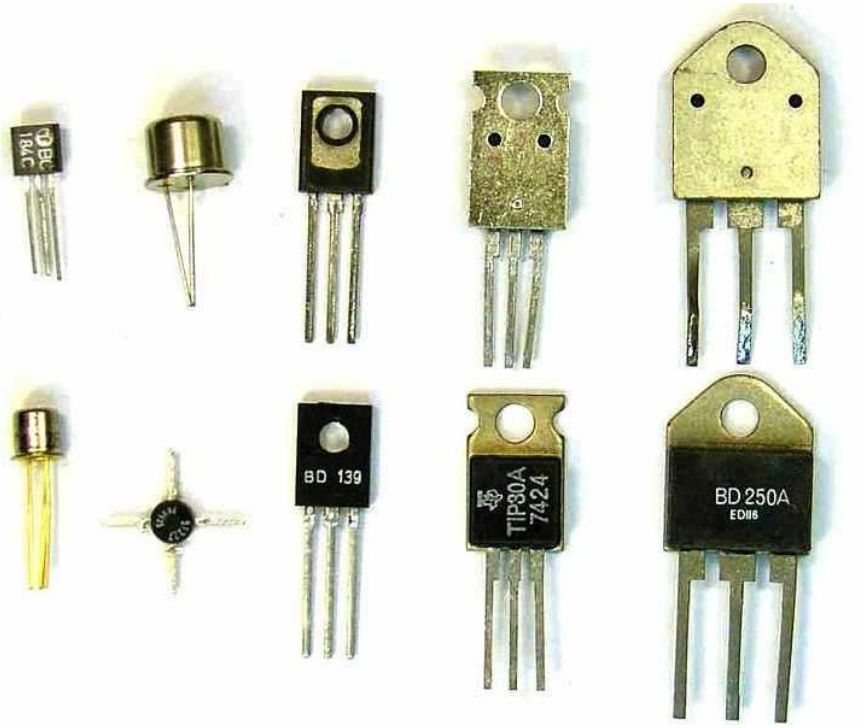
A tranzisztor kémiailag eltérő adalékolású rétegekből álló félvezetőeszköz, amelynek jellemző felhasználási területei az elektromos jelerősítés, valamint jelek ki-be kapcsolása. A belső ellenállásuk változó, a bázis lábra kötött feszültség értékétől függ.

Az NPN és a PNP abban különbözik egymástól, hogy az NPN bázis lábára pozitív, a PNP bázis lábára negatív pólust kell kötni. Ha egyfajta kapcsolóként képzeljük el a tranzisztor belsejét, akkor az NPN alapesetben nyitott állású, és akkor kapcsol be, ha a bázis lábon át pozitív feszültséget kap. A PNP fordítva működik, alapesetben zárt benne az áramkör, ha megkapja a negatív vezérlő áramot, akkor nyitja az áramkört.

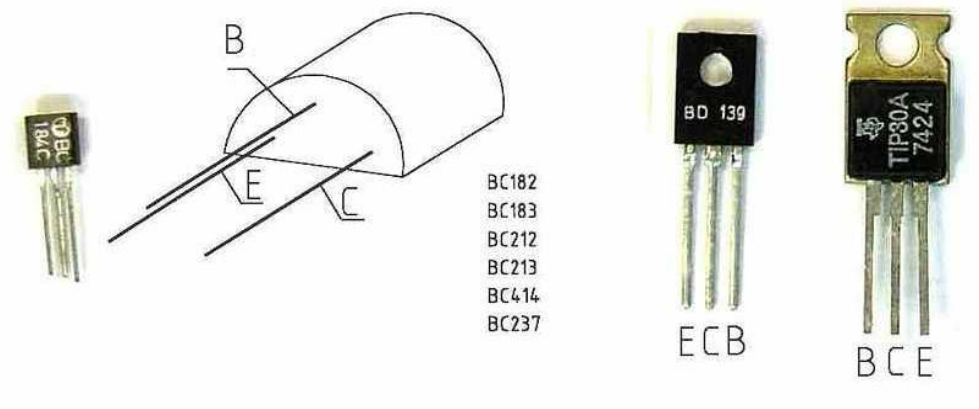
A rajzjelen a kis nyíl jelzi a segédáram haladási irányát, aminek hatására a főáram is beindul, mivel a kollektor–bázis átmenetbe töltéshordozók kerülnek. Műszaki adatai között külön szerepelnek a lábak közötti részek (BE, CB, CE) feszültségei és áramerősségei.

Bipoláris tranzisztor rajzjele, NPN és PNP:





2.21. ábra: Bipoláris tranzisztorok tokozása

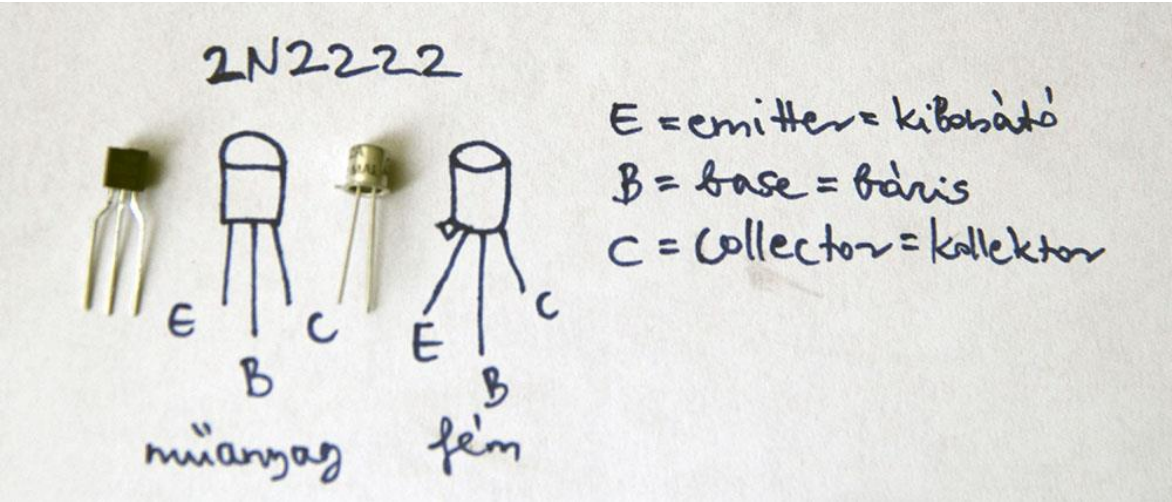


2.22. ábra: A legelterjedtebb bipoláris tranzisztortokok

 © Harsányi Réka, Juhász Márton András

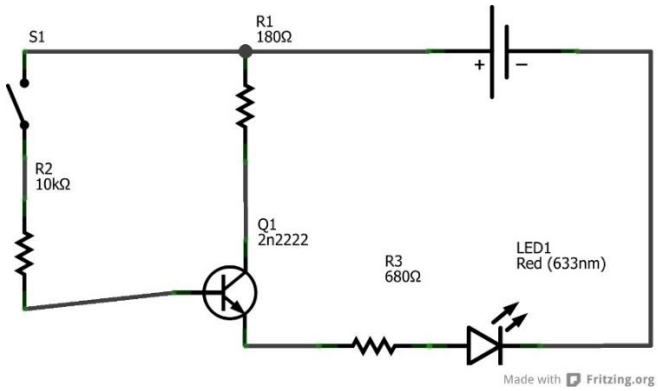
2.4.1. Oszcillátor építése

A próbapanelen és a drótokon kívül szükség lesz az DC (Direct Current = Egyenáram, kisfeszültségű) adapterre 6 voltra állítva, 1 db LED-re, különböző értékű ellenállásokra (470 K, 15 K, 27 K) és kondenzátorokra (2,2 µF) 2 db 2N6027 programozható tranzisztorra, és végül egy 8 ohmos mini hangszóróra (0,25 wattos elég), illetve 1 K ellenállásra és 2N2222 bipoláris tranzisztorra.

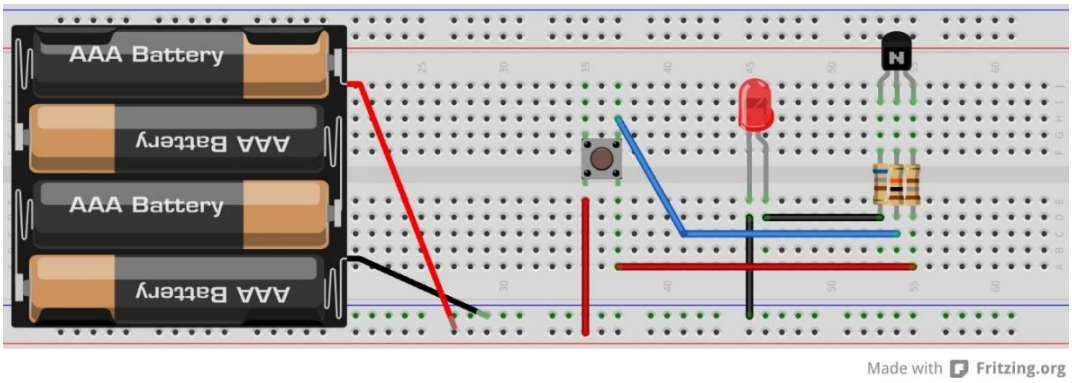


2.23. ábra: 2N2222 tranzisztor lábkiosztása, kétféle tokozásban is kapható, mindegy, melyiket használod, a lábaik elnevezése is megegyezik

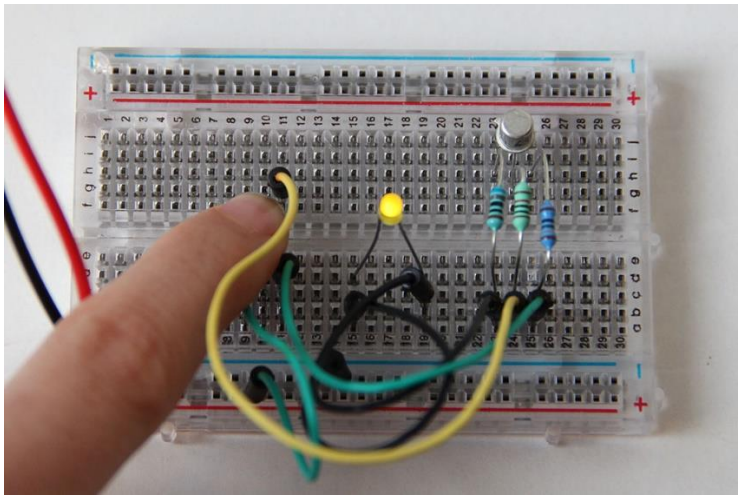
Az első kísérlethez a 2N222 tranzisztort, egy nyomógombot és egy LED-et, valamint 180, 10 K és 680 ohmos ellenállásokra lesz szükségünk.



2.24. ábra: 2N222 bipoláris tranzisztor kapcsolóként üzemeltetve (kapcsolási rajz)



2.25. ábra: 2N222 bipoláris tranzisztor kapcsolóként üzemeltetve



2.26. ábra: 2N222 bipoláris tranzisztor kapcsolóként üzemeltetve

Eleinte a LED sötét, de a gomb megnyomásával világosabbá válik a fénye. Alapesetben a tranzisztor nem engedi át az áramot, de amikor megnyomjuk a gombot, akkor a bázis láb is feszültség alá kerül, és így a benne lévő szilárdtest kapcsoló kinyílik, az áram átjut a kibocsátó lábra, és azon, valamint az ellenálláson keresztül a LED-hez.

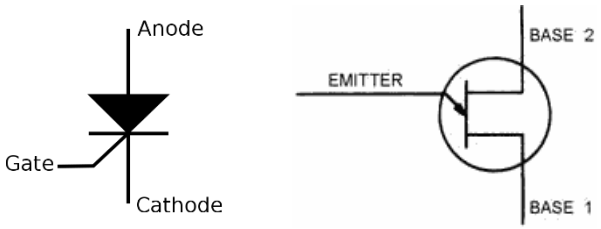
A tranzisztor kollektor lábára kötött feszültséget a bázis vezérli és a kibocsátó küldi tovább. Ebben az esetben a bipoláris tranzisztort átkapcsolásra használtuk, de jelerősítőként is működtethető. A következő kísérlet végén erre is mutatunk egy példát.

Következő feladatunk egy olyan áramkör összeállítása tranzisztorok segítségével, mint amilyen a korábban relével összerakott volt. Ehhez két tranzisztorra lesz szükségünk, amik egymást kapcsolják ki és be. Ha mindkettő bipoláris tranzisztor lenne, akkor nagyon nehezen értenénk meg, mi is történik, ezért úgynevezett programozható, unipoláris (UJT = unipolar junction transistor) tranzisztorokat fogunk használni.

 © Harsányi Réka, Juhász Márton András

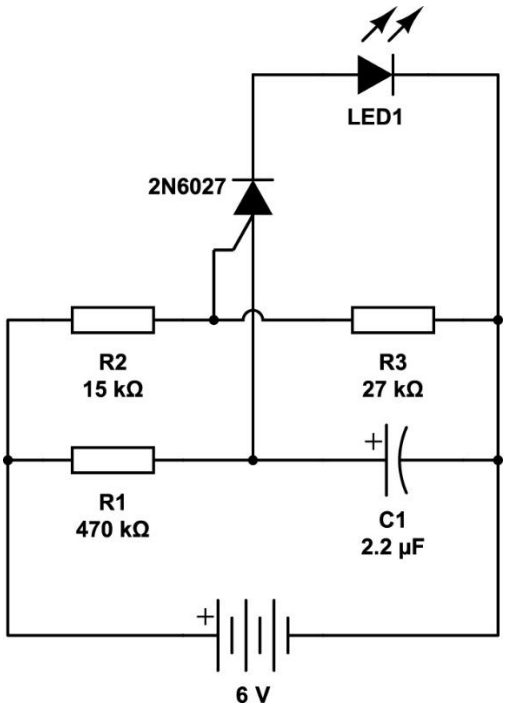
Mit csinál az unipoláris, programozható tranzisztor? Ha az anód lábát rákötjük a feszültségre, aminek értékét lassan növeljük, eleinte nem történik semmi. Ha az anód lábra kötött feszültség elérte a 6 voltot, akkor a tranzisztor átengedi az áramot a katód lábra is. Viszont ha a feszültség lecsökken, akkor az áramlás megszakad. A kapu elnevezésű lábra kötött feszültség határozza meg azt a küszöbértéket, aminek elérésekor az unipoláris tranzisztor átengedi majd az áramot. Ennek értékét ellenállások bekötésével szabályozhatjuk.

Az unipoláris tranzisztor rajzjele:

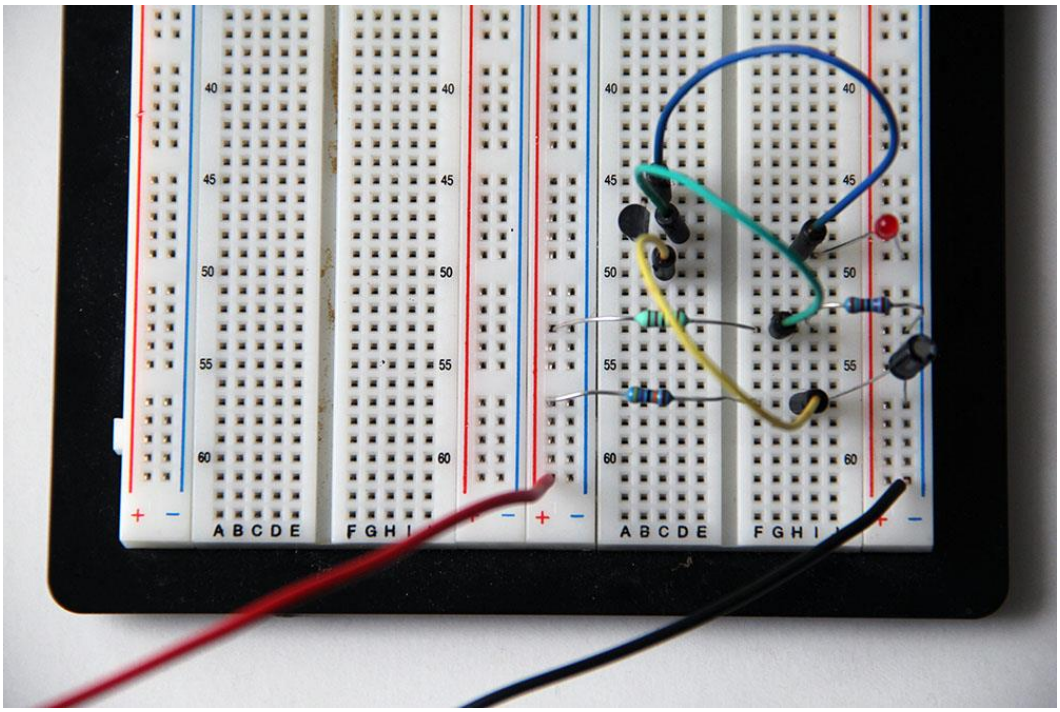


Ha szemből nézzük ezt a tranzisztort, a lábai balról jobbra: Anode = Anód, Gate = Kapu, Cathode = Katód. Viszont egy másik fajtájával is találkozhatunk, a fenti második razjelen két bázis és egy kibocsátó lába van. Most az előbbi verzióval dolgozunk, hogy könnyebben be tudjuk azonosítani a lábakat a bekötésnél.

Oszcilláció létrehozásához még be kell kötni egy kondenzátort is. Mialatt a tranzisztor „kikapcsolt” módban van, a kondenzátor feltöltődik. Az elé kötött ellenállás nagysága meghatározza, milyen gyorsan tölt fel a kondenzátor. Ha nagyobb értékűt kötünk elé, lassabban tölt fel, éppúgy, mint a nagyobb kapacitású kondenzátor használata esetén. Jelen esetben kb. fél másodperc lesz ez az idő. A kondenzátorból az anód lábhoz érkező áram, ha eléri azt a feszültségértéket, amit a kapu lábon a rákötött ellenállásokkal határoztunk meg, akkor az unipoláris tranzisztor „bekapcsol”, így a katód lábon keresztül eljut az áram a LED-hez, ami felvillan. Ekkor a túlfeszültség kimeríti a kondenzátort, így – mivel csökken a feszültség – az unipoláris tranzisztor visszaáll eredeti állapotába, azaz „kikapcsol”. A folyamat elkezdődik elölről, így jön létre az oszcilláció. A végén kipróbálhatunk egy 22 µF kondenzátort is. Ekkor azt tapasztaljuk, hogy sokkal több idő telik el a két felvillanás között. Most rakjuk össze a következő, 6 volt feszültséggel működő áramkört, ahol egy LED-et vezérlünk vele.

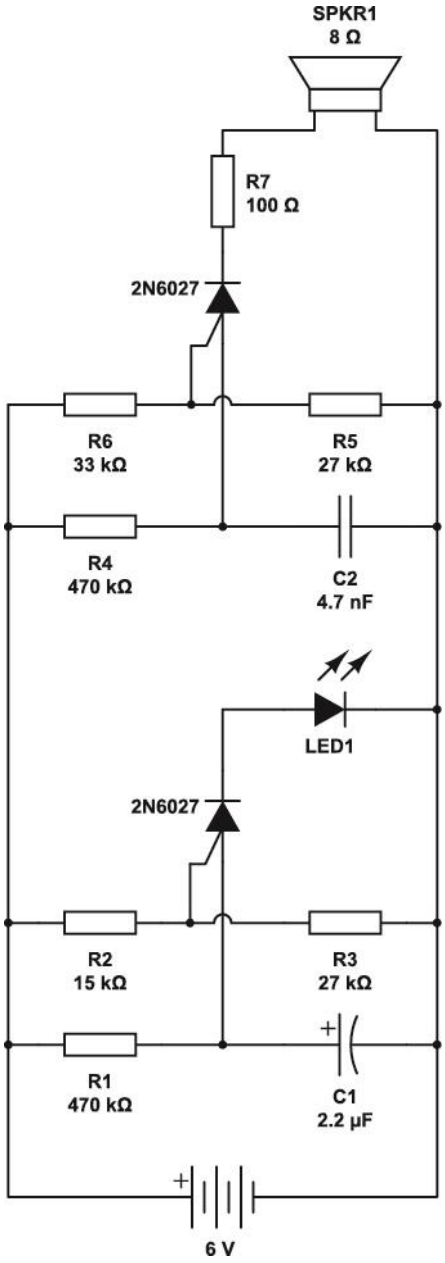


2.27. ábra: LED vezérlése unipoláris tranzisztorral (kapcsolási rajz)

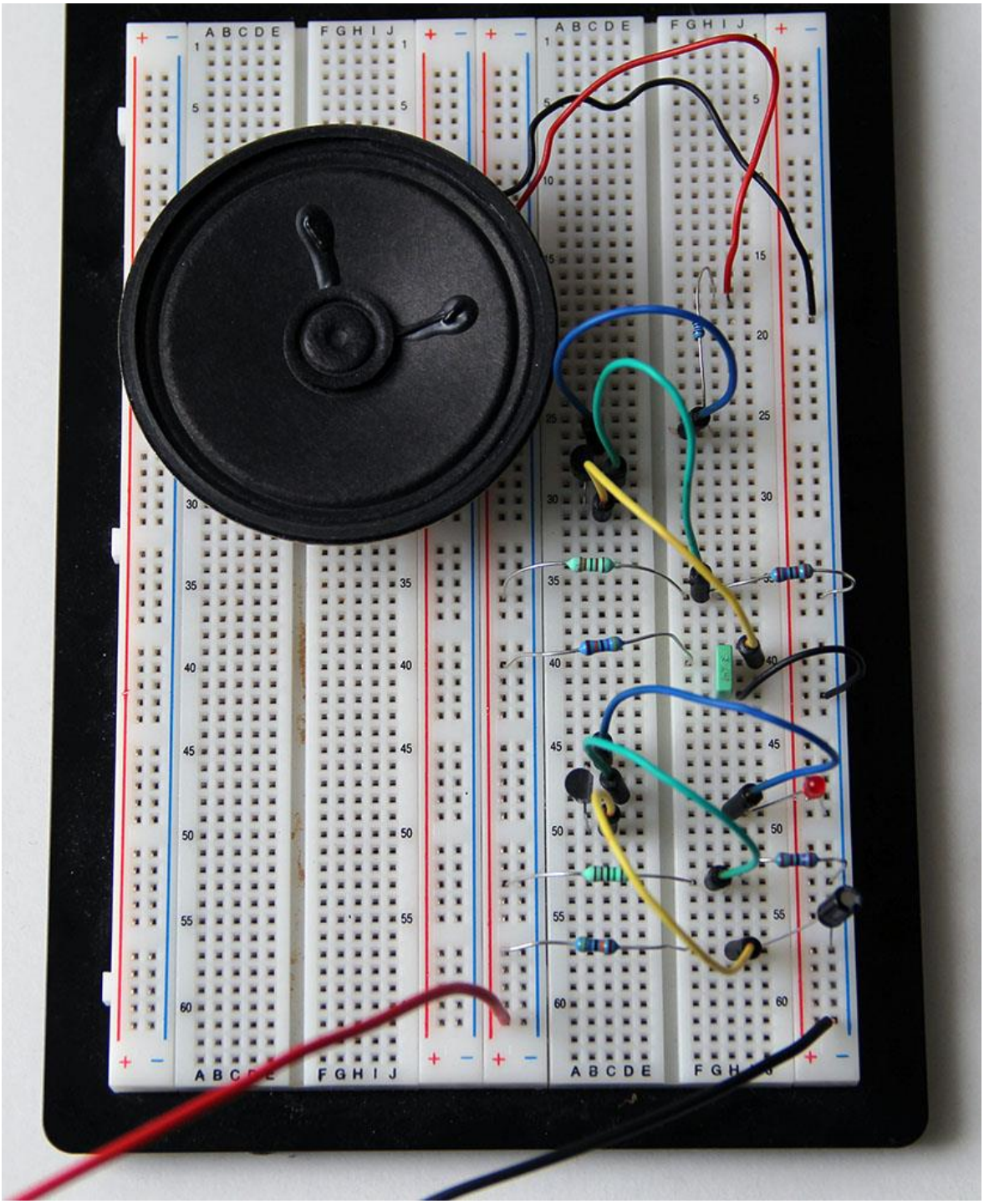


2.28. ábra: LED vezérlése unipoláris tranzisztorral

 © Harsányi Réka, Juhász Márton András



2.29. ábra: Unipoláris tranzisztoros oszcillátor kapcsolási rajza



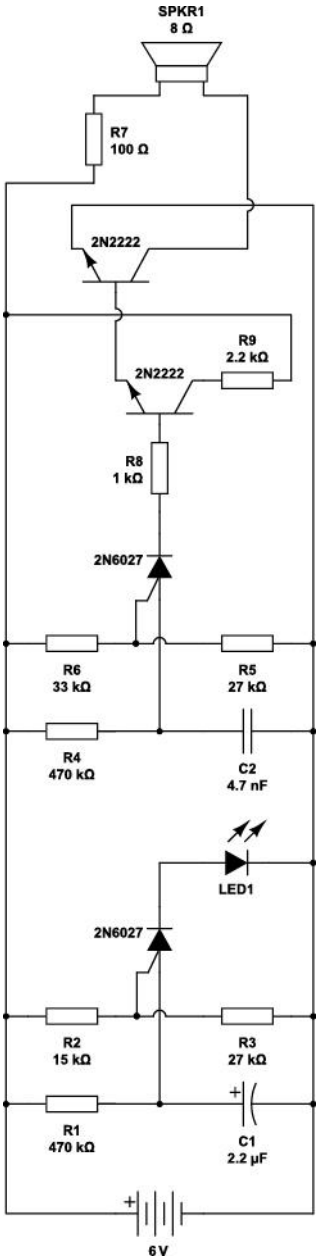
2.30. ábra: Unipoláris tranzisztoros oszcillátor

 © Harsányi Réka, Juhász Márton András

Egészítsük ki az áramkört a többi alkatrésszel a 2.29 számú kapcsolási rajz segítségével. Szükségünk lesz még ellenállásokra: 470 K, 33 K, 27 K és 100 Ω, kondenzátorra: 0.0047 µF, még egy UJT tranzisztorra: 2N6027 és egy 8 Ω (0,25 W) mini hangszóróra. A hangszóró hátulján található egy + és – jelzés, ennek megfelelően kell bekötni az áramkörbe. Ugyanaz történik ebben a részben is, mint az előzőben, csak a LED helyére egy hangszóró került, így most már hallhatjuk is, mi történik. Építettünk egy analóg oszcillátort.

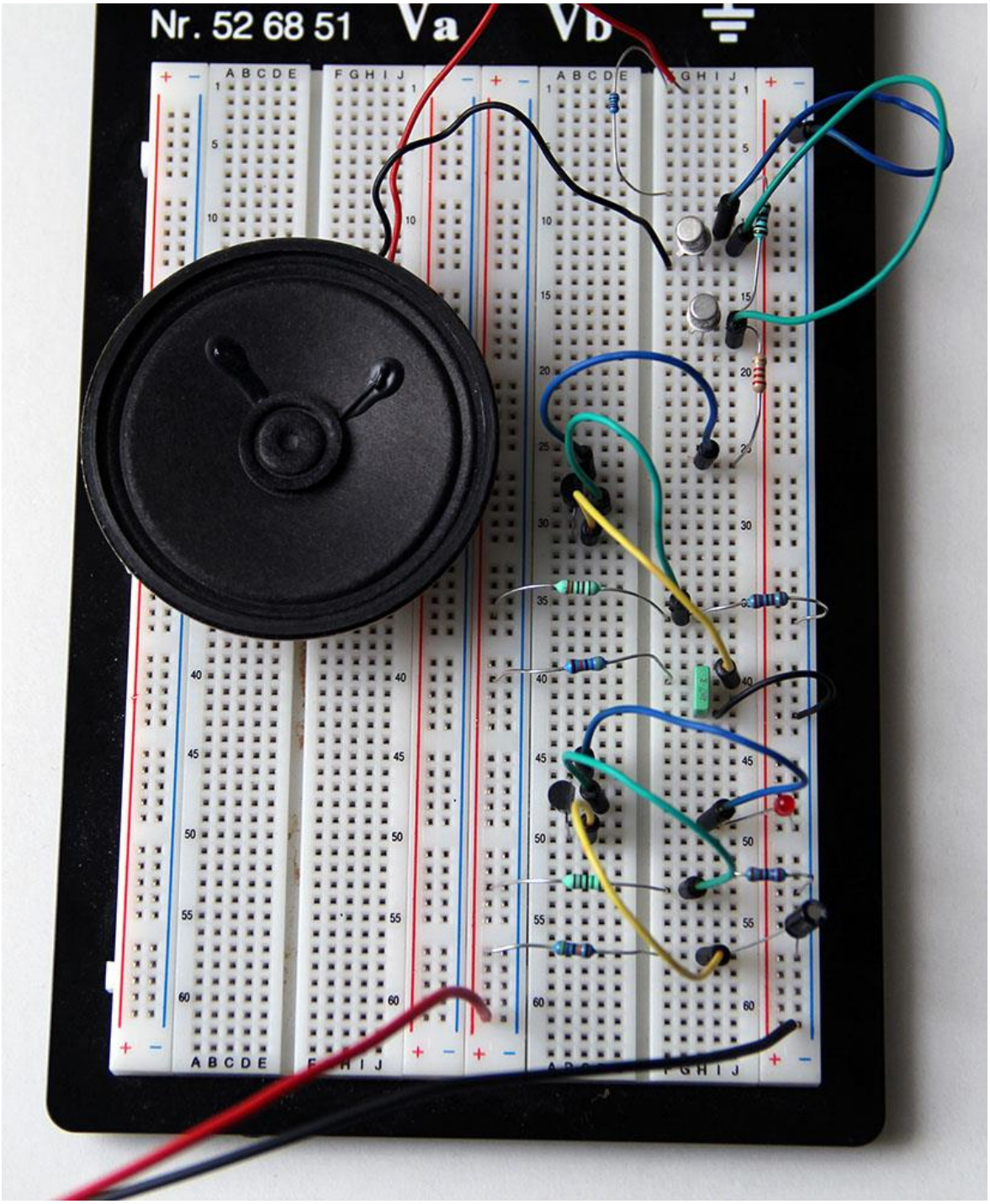
A 2N2222 bipoláris tranzisztor jelerősítésre alkalmas. Az áramerősséget erősiti fel, nem a feszültséget. Egyfajta kapcsolóként funkcionál mivel, ha áram folyik be a bázison, akkor összezárja a kollektorát az emitterével. Az alap lábra érkező minimális változással is nagy áramerősség változást érhetünk el benne. Az emitteren tovább haladó áramerősség sokszorosa lehet az alapra érkezőének. Hogy hányszorosára képes erősíteni a tranzisztor ezt a jelet, a béta-értékkel fejezi ki, ami egy szorzó. Ha egy tranzsztornak 20-as a bétája, akkor ha 1A áramerősség folyik be a bázis lábán, akkor 20A fog a kibocsátón kifolyni.

Ha bekötjük a 2N2222 tranzisztort hangszóró elé, hangosabbá válik az árammal generált hang, ami a kondenzátorok méretével szabályozott időintervallumban hallható, ugyanúgy oszcillál, mint korábban a LED fény. Kössünk be két tranzisztort a 2.31 számú rajz alapján!



2.31. ábra: Oszcillátor hangjának felerősítése két bipoláris tranzisztorral (kapcsolási rajz)

 © Harsányi Réka, Juhász Márton András

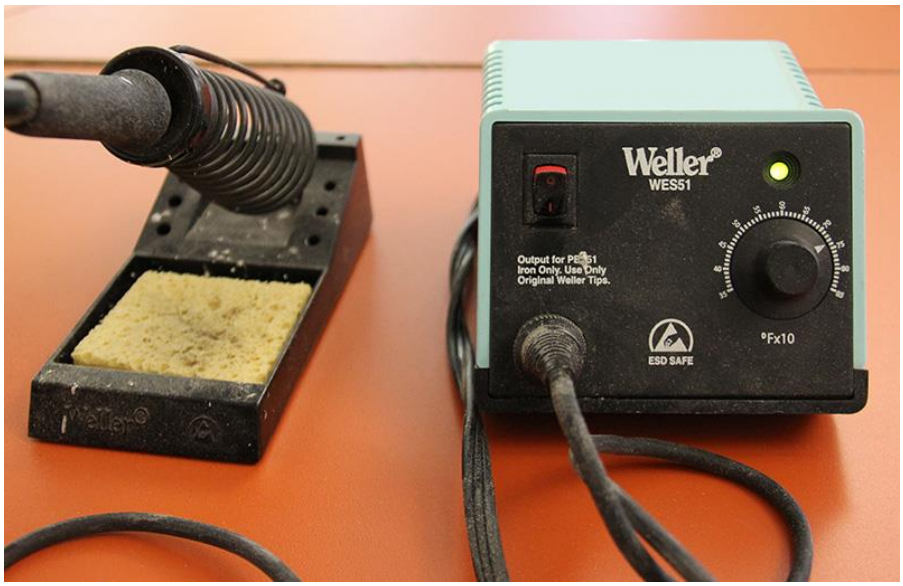


2.32 ábra: Oszcillátor hangjának felerősítése két bipoláris tranzisztorral

3. A forrasztás menete és eszközei

A forrasztás egy olyan, fémek közötti, oldhatatlan kötés, ami vezeti az áramot. Forraszanyaggal készül, amelynek olvadáspontja alacsonyabb, mint az alkatrészeké. Forrasztópákával vagy forrasztópisztollyal előállított hőforrás segítségével jön létre a forrasztás. A legelterjedtebb forraszanyag az ón, bizonyos változata forrasztózsírt is tartalmaz, ami letisztítja a felületet. Ha nincs benne, külön meg kell tisztítani a forrasztásra szánt részeket, mivel a kézről rátapadó zsír és kosz megakadályozza, hogy jól tapadjon az ón, így hibás lesz a kötés, ami nem vezeti majd az áramot. Az erre alkalmas tisztítószereket folyatószereknek nevezzük, mely lehet zsír, krém, flux folyadék stb. Másik feladatuk, hogy megvédjék a hőhatásnak kitett fémet az oxidációtól.

Megkülönböztetünk kemény és lágyforrasztást, a munkahőmérséklet előbbi esetben 450 ^{o}C feletti, a lágyforrasztásnál pedig az alatti. Ez az a hőmérsékletet, amin a felhasznált forraszanyag körül tudja folyni a felületet. Fontos, hogy a hegesztés egy egészen más eljárás, nagyobb felületek esetén alkalmazzák, ezért erre nem térünk ki. Mi pákával lágyforrasztásokat fogunk készíteni. Fontos, hogy a páka milyen hőfokú, az érzékenyebb alkatrészeknél alacsonyabb hőfokra kell állítani. Az első fejezetben szereplőken kívül a következő eszközökre lesz szükségünk.



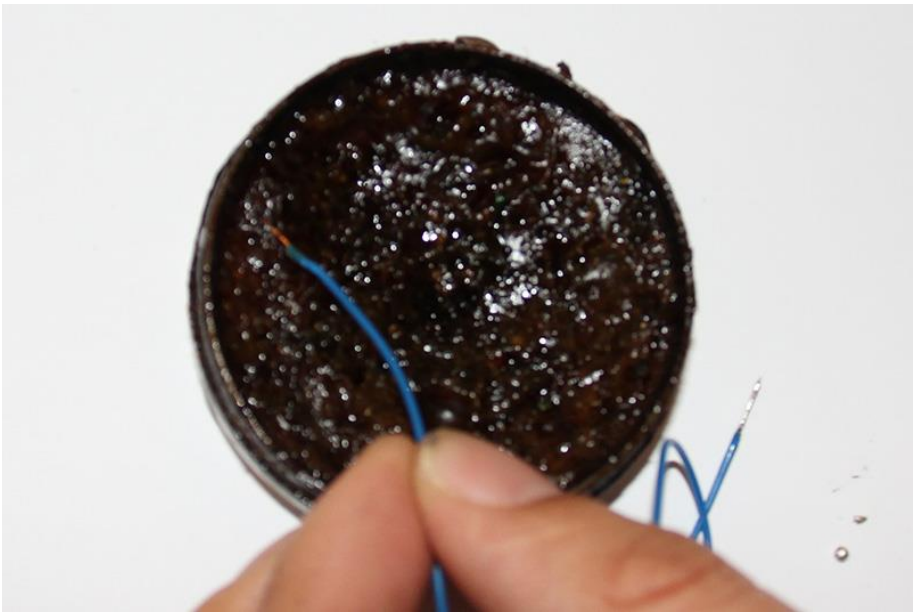


3.1. ábra: Forrasztó egység állítható hőfokszabályozóval, 690 °C-ra állítva. A forrasztó ón olvadási pontja 231,9 °C. Fontos, hogy a forrasztandó részek hőmérséklete meghaladja a forrasztóón olvadáspontját!

A forró pákát mindig a helyére kell visszatenni!



3.2. ábra: Forrasztó ón

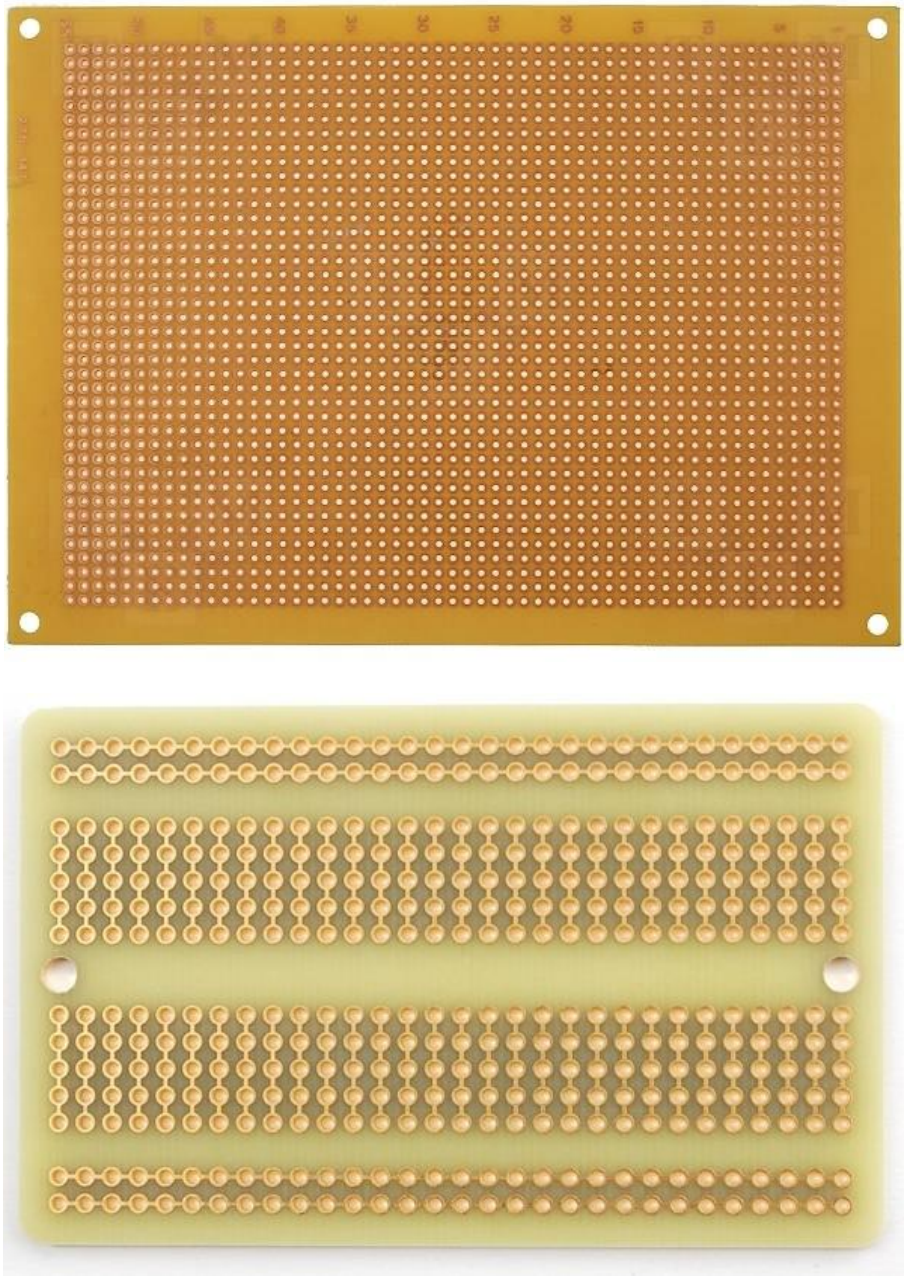


3.3. ábra: Forrasztózsír



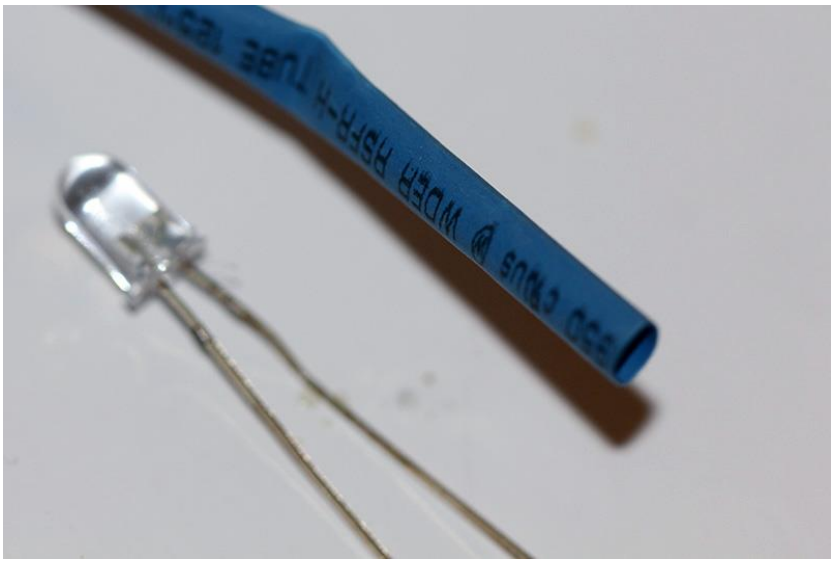
3.4. ábra: Kiforrasztó pumpa és huzal

 © Harsányi Réka, Juhász Márton András



3.5. ábra: Prototípus NYÁK lemezek forrasztásához (különböző típusok léteznek)



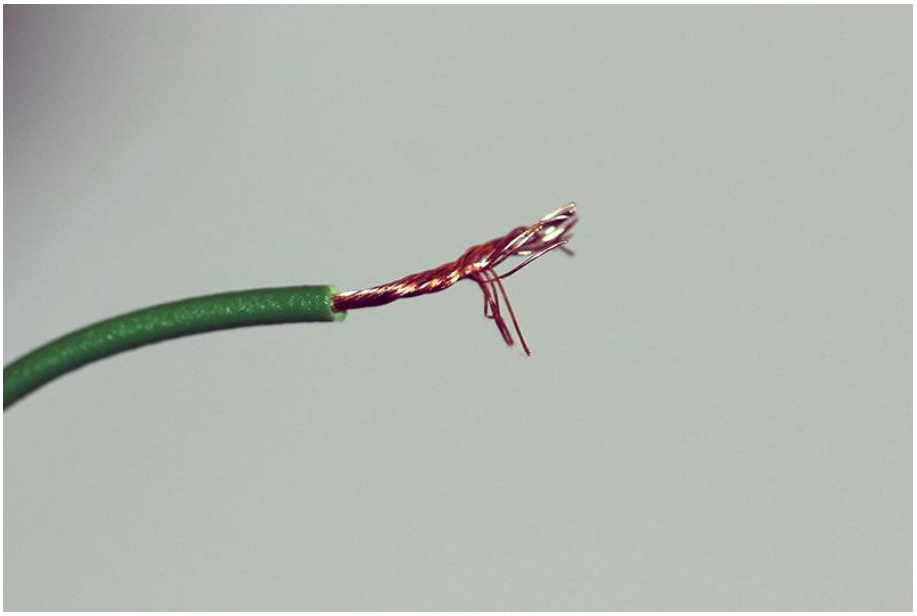


3.6. ábra: Zsugorcsövek különböző méretben és színben (készletben is kaphatóak)

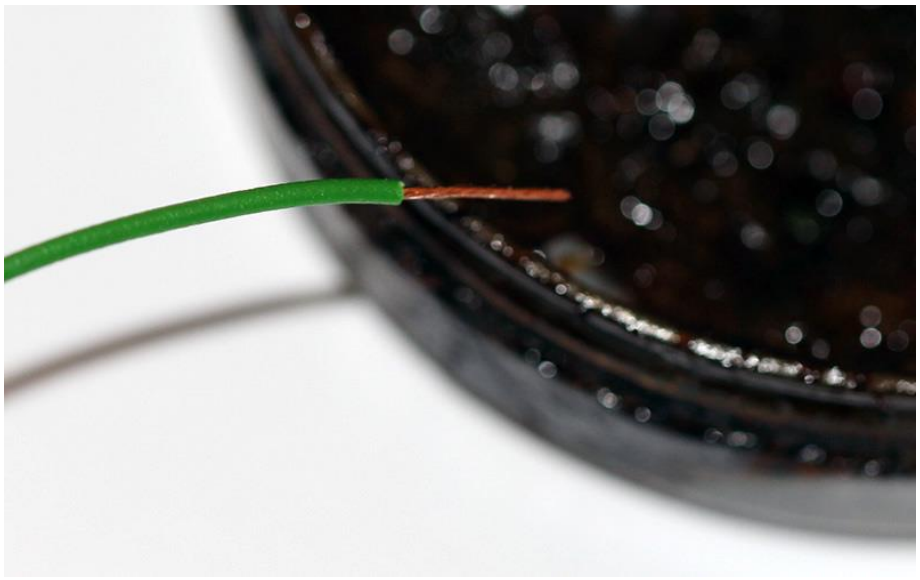


3.7. ábra: „Harmadik kéz” a műszerész forrasztó állvány, mely megkönnyíti a munkát

Két példán keresztül megtanuljuk, hogyan kell forrasztani. Egy LED-et és ellenállást kapcsolunk össze úgy, hogy zsugorcsővel elszigeteljük a lábait egymástól. Ezt követően a prototipizálásra alkalmas NYÁK lemezbe beforrasztunk egy LED-et és egy ellenállást, majd kiforrasztjuk, hogy annak a menetével is megismerkedjünk.

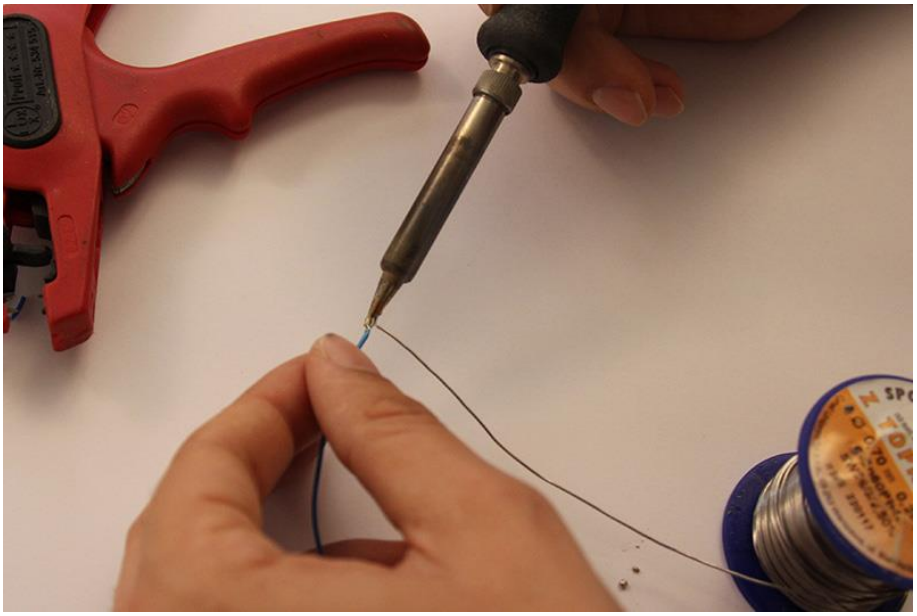


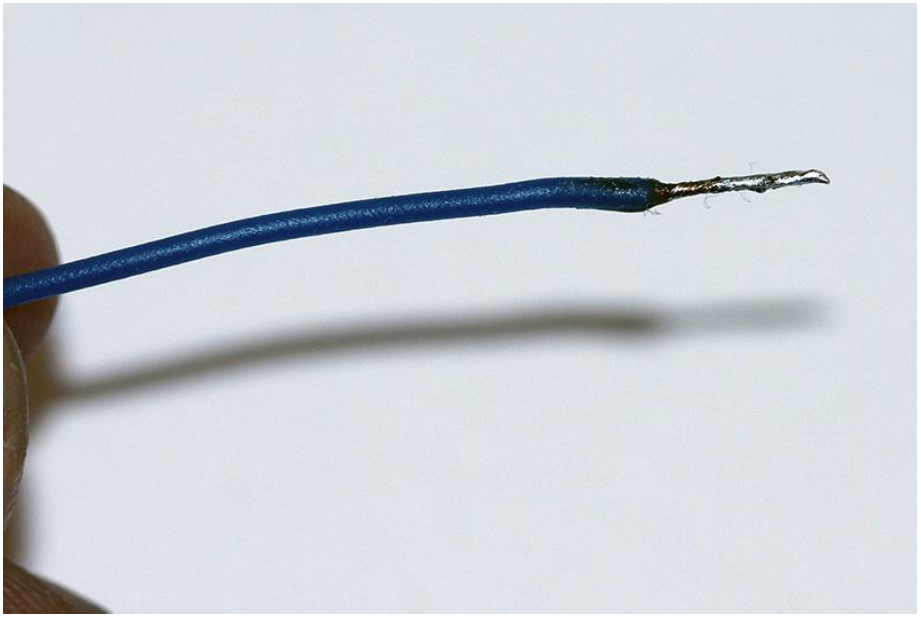
3.8. ábra: Előfordulhat, hogy többszálas a drót, így blankolás után szétállnak a végei.



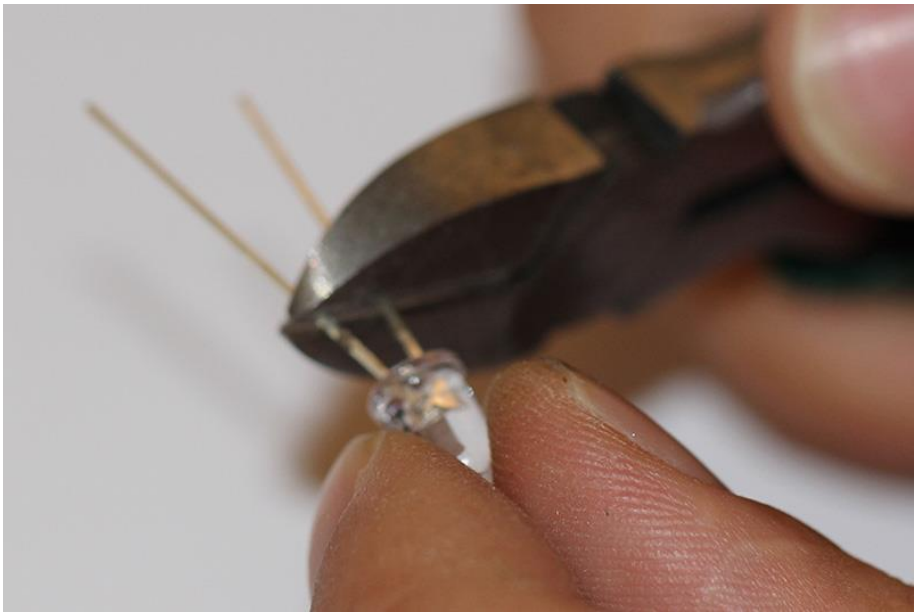
3.9. ábra: Kézzel sodorjuk össze a végét és mártsuk a forrasztózsírba!

 © Harsányi Réka, Juhász Márton András





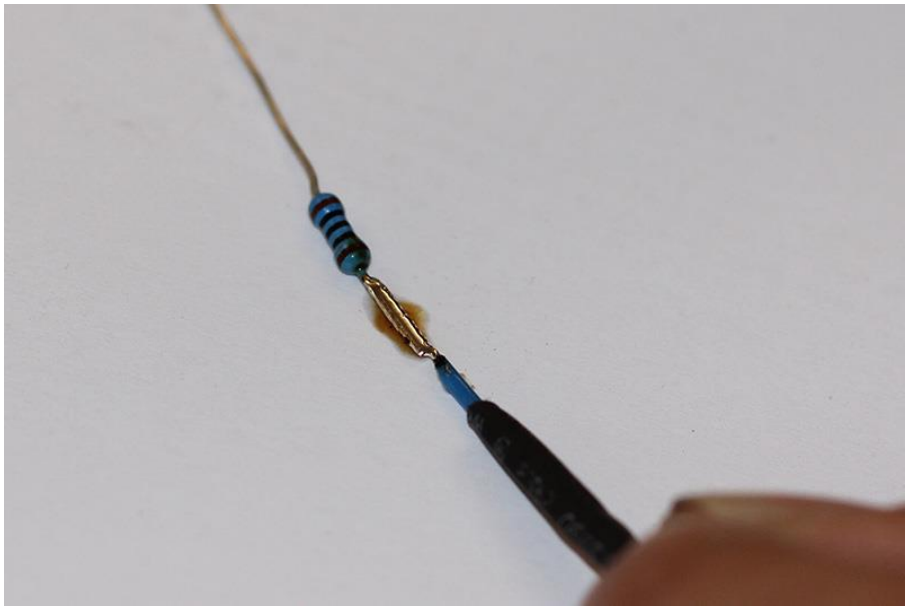
3.10. ábra: Utána ónozzuk be a végét, hogy könnyebb legyen az alkatrészhez forrasztani!

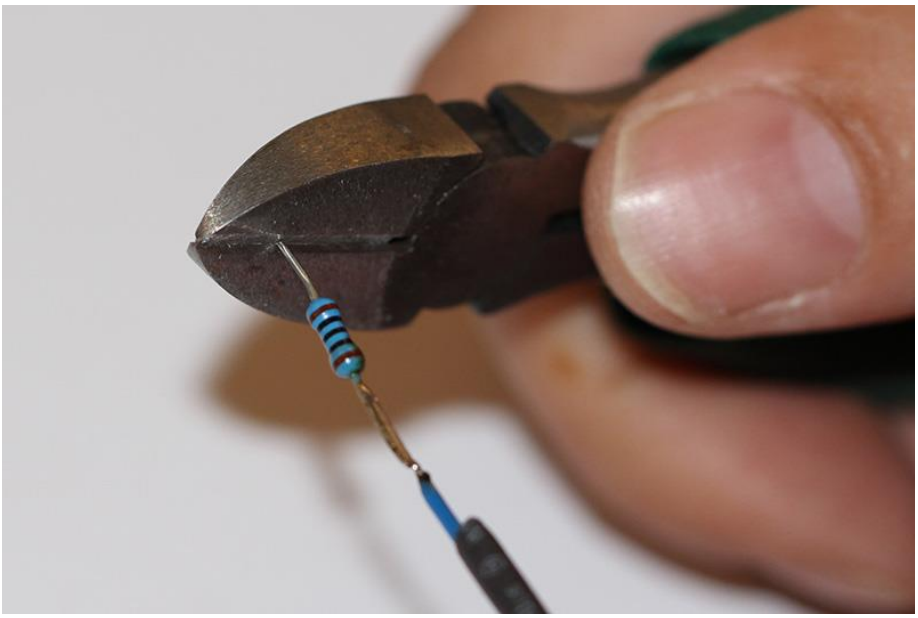




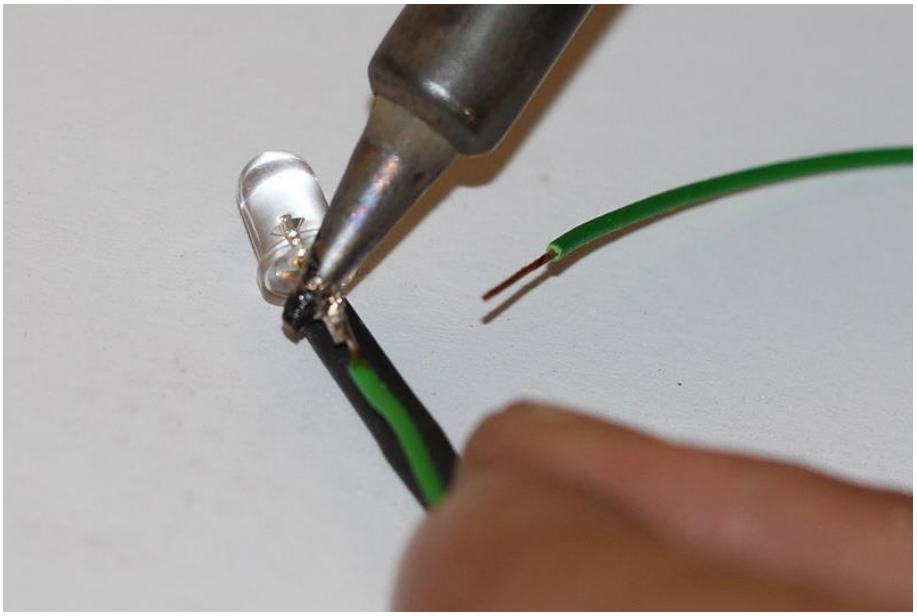
3.11. ábra: A LED lábait vágjuk le!

 © Harsányi Réka, Juhász Márton András

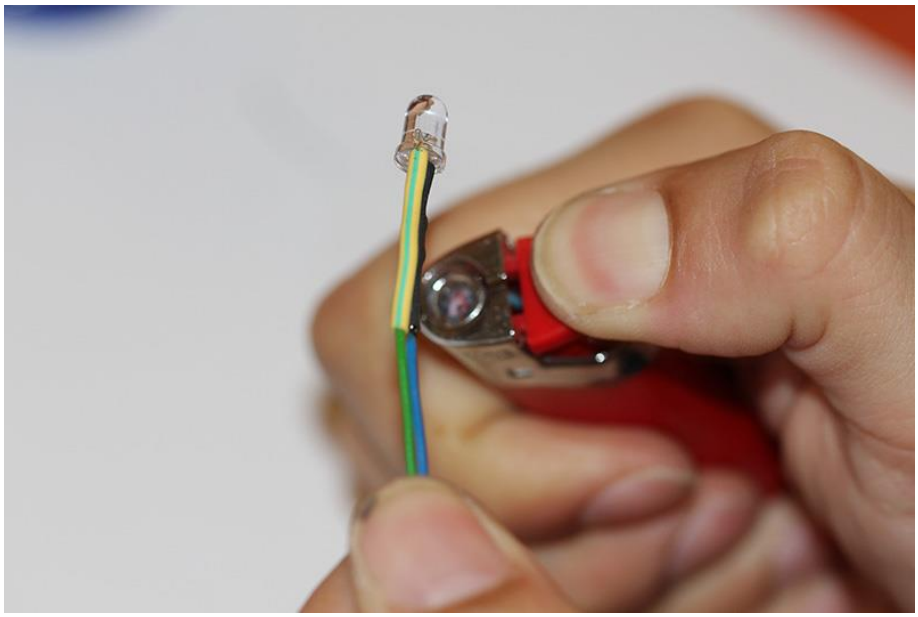




3.12. ábra: Az ellenállás forrasztása előtt húzzunk fel a drótra egy megfelelő hosszúságú és átmérőjű zsugorcső darabot!

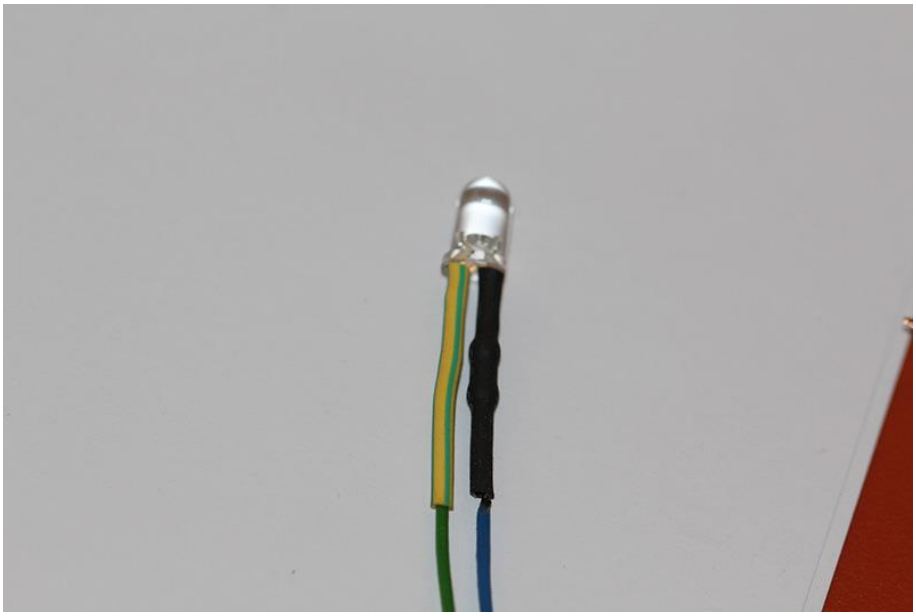


3.13. ábra: Az ellenállást forrasszuk a LED megfelelő lábához és a zsugorcsövet húzzuk rá az egészre, majd a LED másik lábán ismételjük meg ugyanezt!

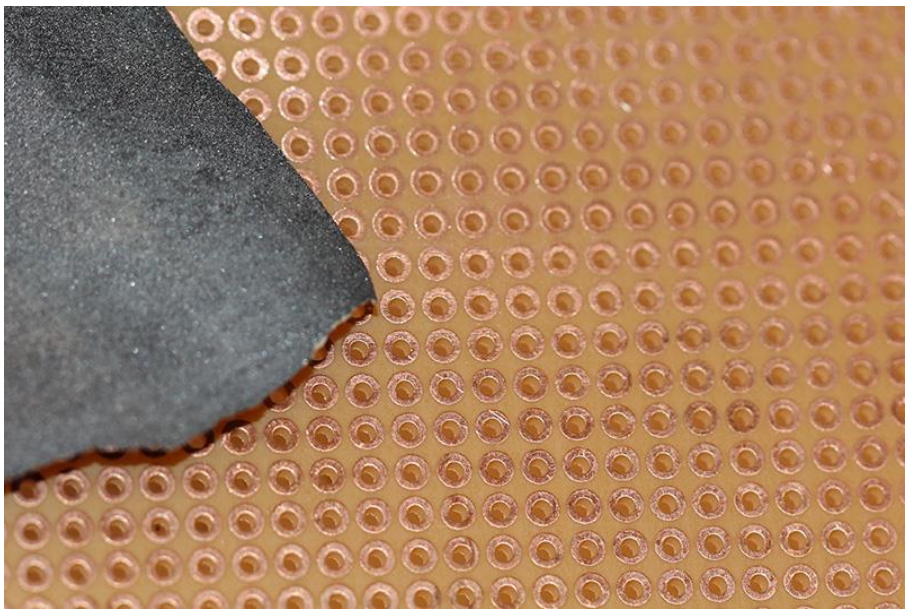


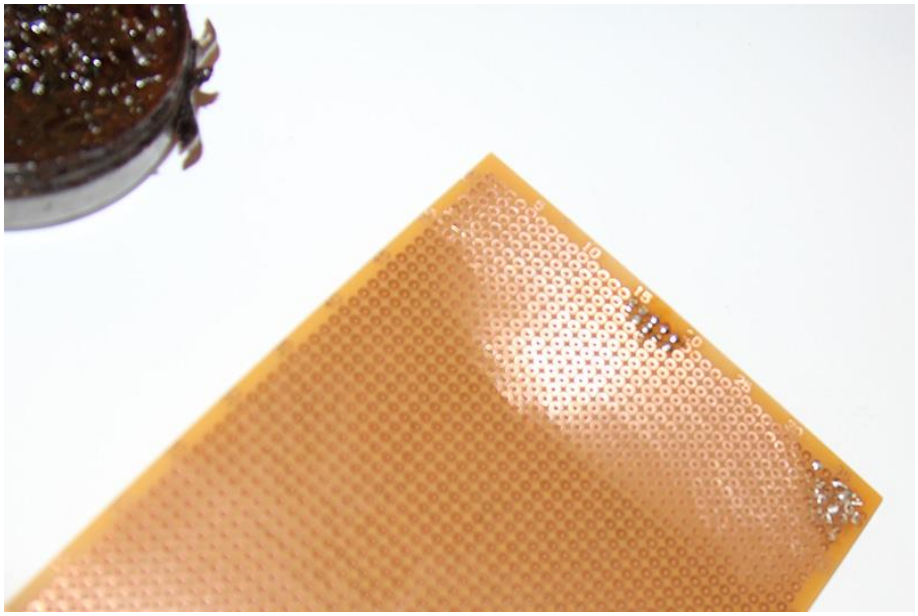
3.14. ábra: A zsugorcső hő hatására összezsugorodik, így ráfeszül a forrasztásra. Mivel műanyag alapú, le is szigeteli azt, így elkerülhetőek a rövidzárok. Tartsuk rövid ideig öngyújtó lángja fölé úgy, hogy a láng ne érjen hozzá (a hő így is elegendő).

 © Harsányi Réka, Juhász Márton András

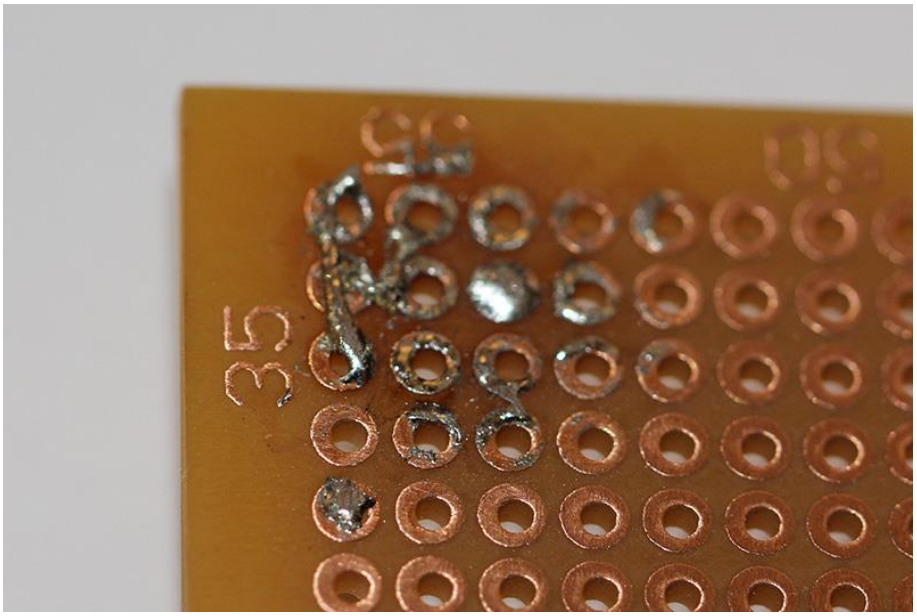


3.15. ábra: Elkészült a zsugorcsővel leszigetelt LED. A képen látszik, hogy a fekete csőben van az ellenállás.



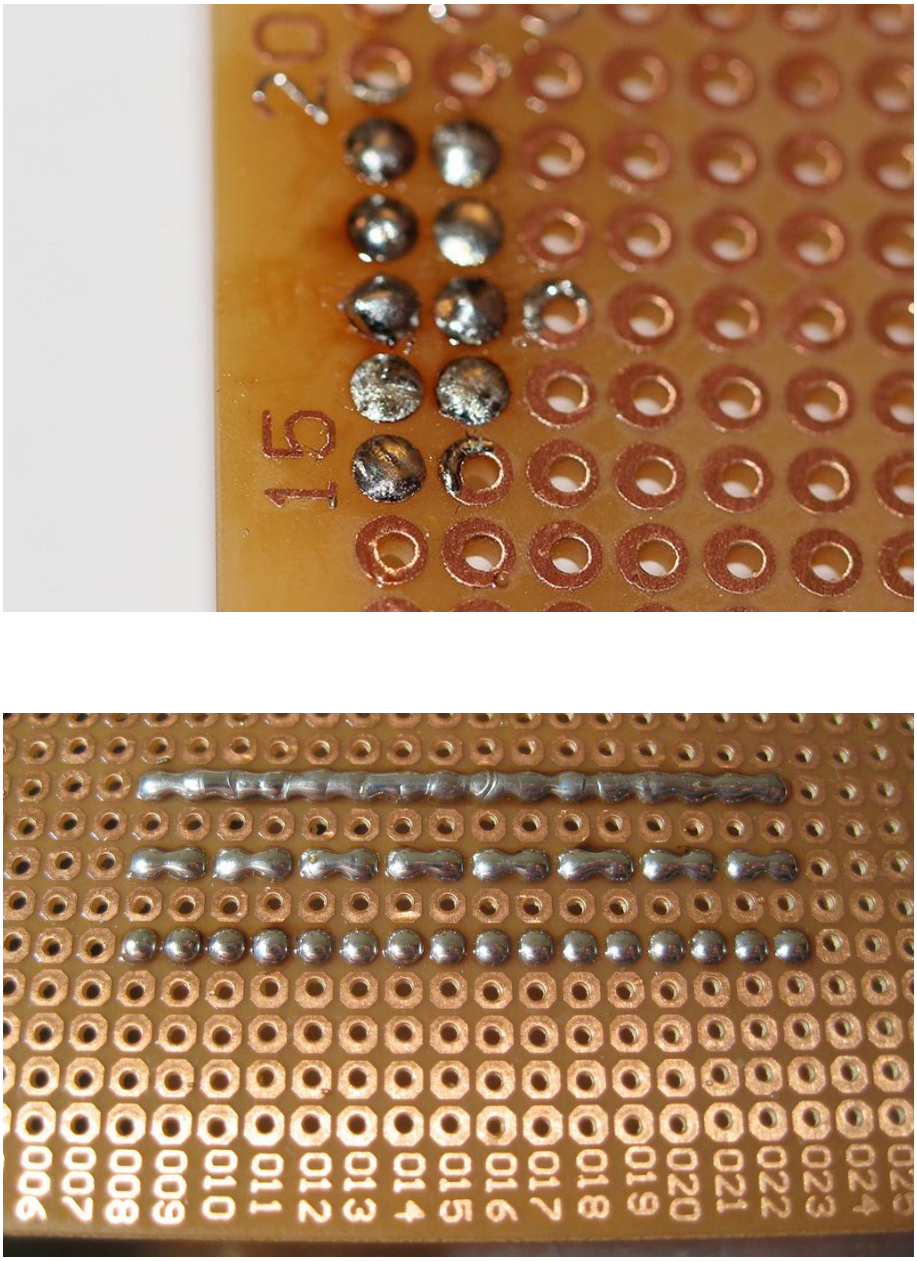


3.16. ábra: A prototipizáló NYÁK lemez esetleg koszos és zsíros lehet, ezért csiszolópapírral óvatosan dörzsöljük át, úgy hogy a rézréteg ne kopjon le!

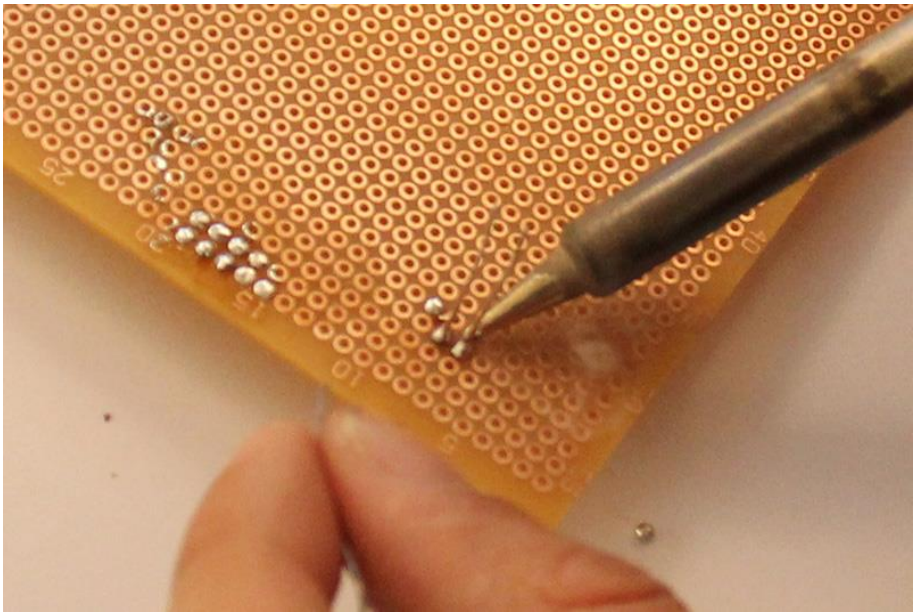


3.17. ábra: Ez a forrasztás hibás, mert átfolyt az ón a szomszédos lyukra is, illetve nincs elég ón rajta, mivel az folyatószer nélkül nem tapad.

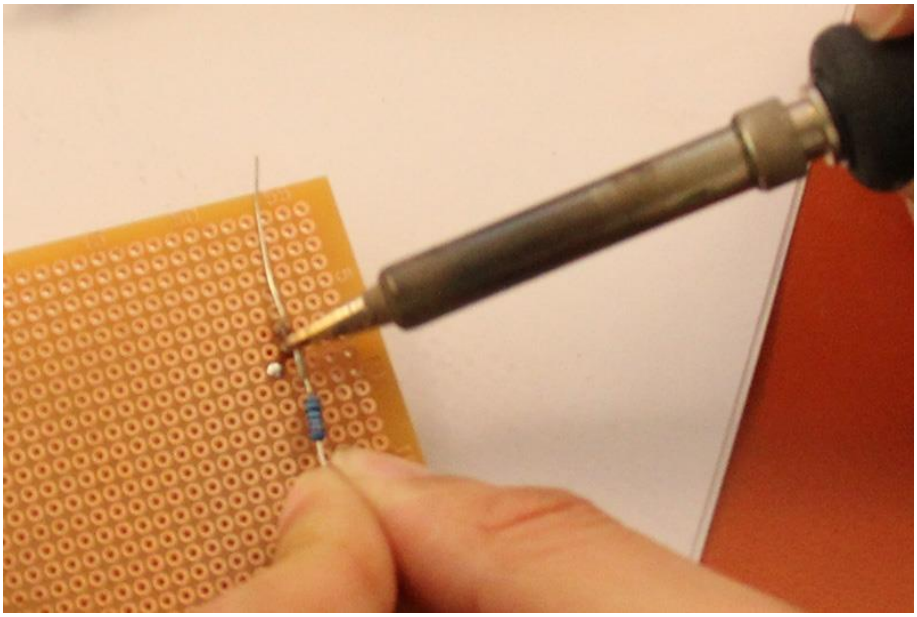
 © Harsányi Réka, Juhász Márton András



3.18. ábra: Ez pedig a helyes, folyatószerrel készült forrasztás, , elég mennyiségű ónnal. Szándékosan is létrehozhatunk kötéseket (alsó kép), vagy használhatunk olyan NYÁK lemezt, ami gyárilag rézzel összekötött szakaszokat tartalmaz.



3.19. ábra: A NYÁK lemezbe dugott LED lábait az ellenkező oldalról tudjuk kényelmesen megforrasztani.



3.20. ábra: Az ellenállás közvetlenül a LED lábához is csatlakozhat, vagy ugyanúgy, mint a LED. Utóbbi esetben alulról drótokkal összeköthetjük a megfelelő részeket.

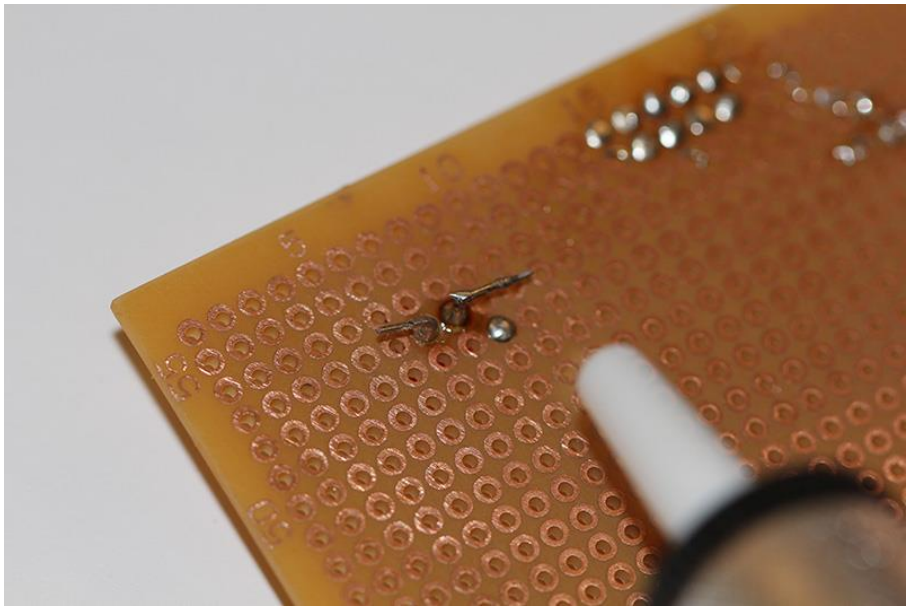
 © Harsányi Réka, Juhász Márton András

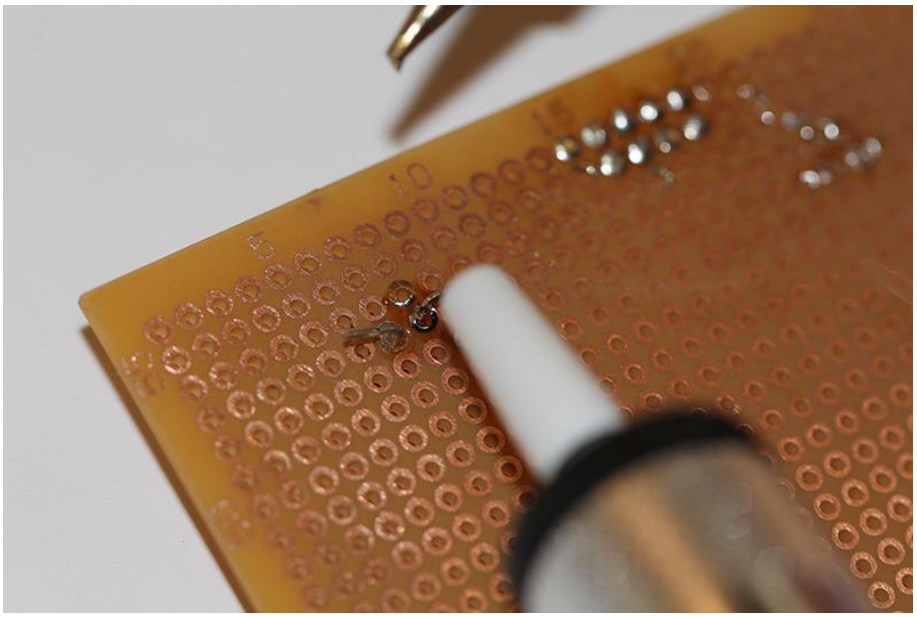


3.21. ábra: FONTOS, hogy minden forrasztás után tisztítsuk le a pákát egy nedves szivaccsal, hiszen koszosan nem ragad az ón. Így a pákahegyet is megóvhatjuk, de ha tönkrement, minden típusnál ki lehet cserélni.



3.22. ábra: Érzékenyebb alkatrészeket alacsonyabb hőfokon kell forrasztani, azok sérülésének elkerülése érdekében.

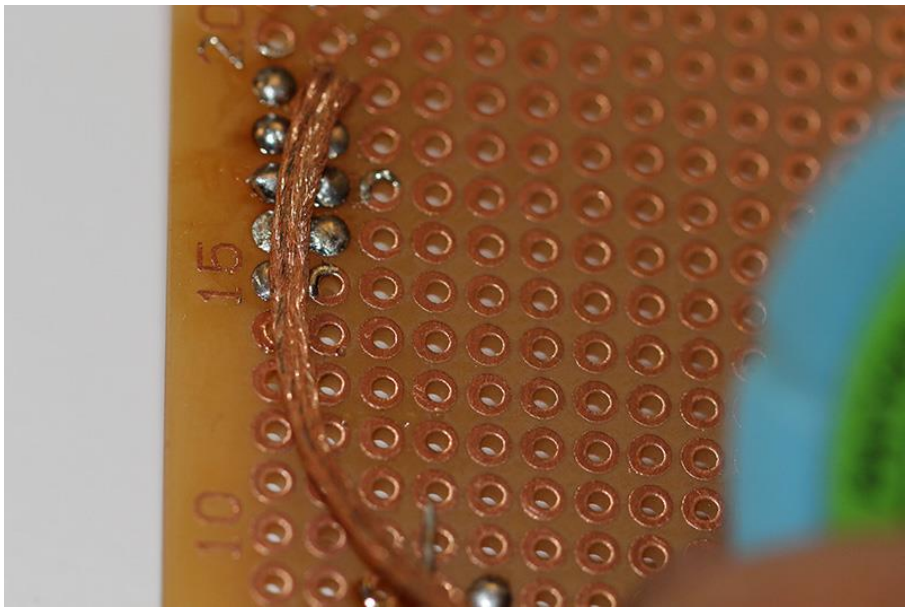


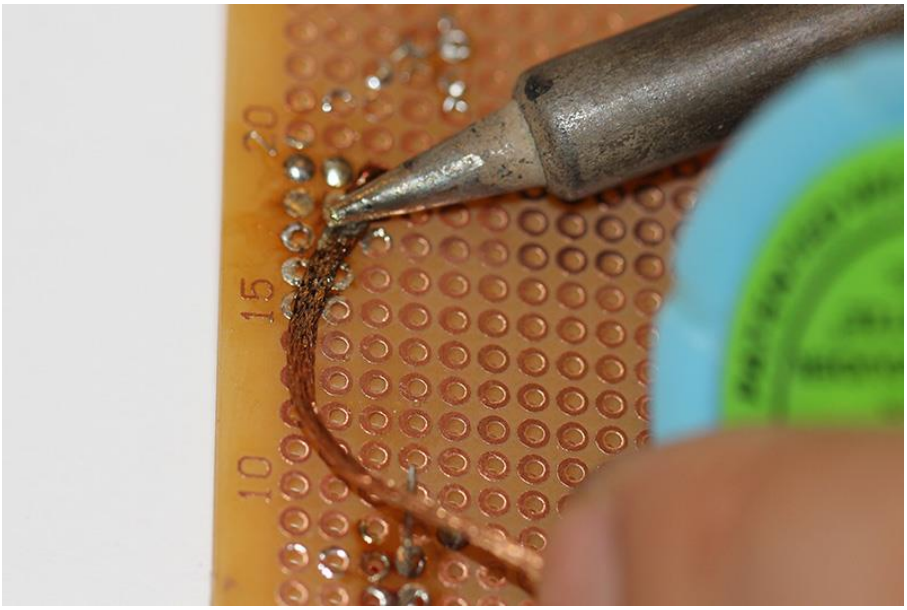


3.23. ábra: Az áramkör szétszedéséhez és a forrasztás eltávolításához használhatunk kiforrasztó pumpát, melynek a végét benyomva odatartjuk a forrasztáshoz, felolvasztjuk a pákával az ónt és eközben megnyomjuk a pumpa gombját. A pumpában keletkező vákuum kiszívja a felolvadt ónt.

© Harsányi Réka, Juhász Márton András


35+YEARS


15
10



3.24. ábra: A másik lehetőség a kiforrasztó huzal, ami egy többszálas fonott lapos rézdrót. A pákával nyomjuk rá a szétszedni kívánt részre, és így a rézszálak közé átfolyik az ón az alkatrészről.

Most már a saját áramköreinket is össze tudjuk forrasztani.

 © Harsányi Réka, Juhász Márton András

4. Játékokból nyert szenzorok és piezo érzékelő

Ebben a leckében játékok belsejében lévő alkatrészeket veszünk sorra, amiket később is tudunk hasznosítani, mivel némelyiket Arduinoval is összeköthetünk.

4.1. Low tech szenzorok és aktuátorok

Ez az alfejezet **Usman Haque és Somlai-Fischer Szabolcs: Low tech sensors and actuators** c. könyvén alapul.

Weboldala: http://lowtech.propositions.org.uk/
Itt elérhető az eredeti könyv pdf formátumban és a hozzá tartozó videók is.
By-Nc-Sa licenc alatt: http://creativecommons.org/licenses/by-nc-sa/2.5/
Videója: http://www.youtube.com/watch?feature=player_detailpage&v=FBLuBi-rVT8

© 2005 Usman Haque & Adam Somlai-Fischer,
info@haque.co.uk / studio@aether.hu
This work is released under the Creative Commons Attribution - ShareAlike License.



Ebben az alfejezetben olyan könnyen és olcsón beszerezhető eszközökről vagy játékokról lesz szó, amelyek számunkra hasznos alkatrészeket tartalmaznak, programozás nélkül is működnek. A low tech ebben az esetben egyszerű, kisfeszültségű áramköröket jelent. -

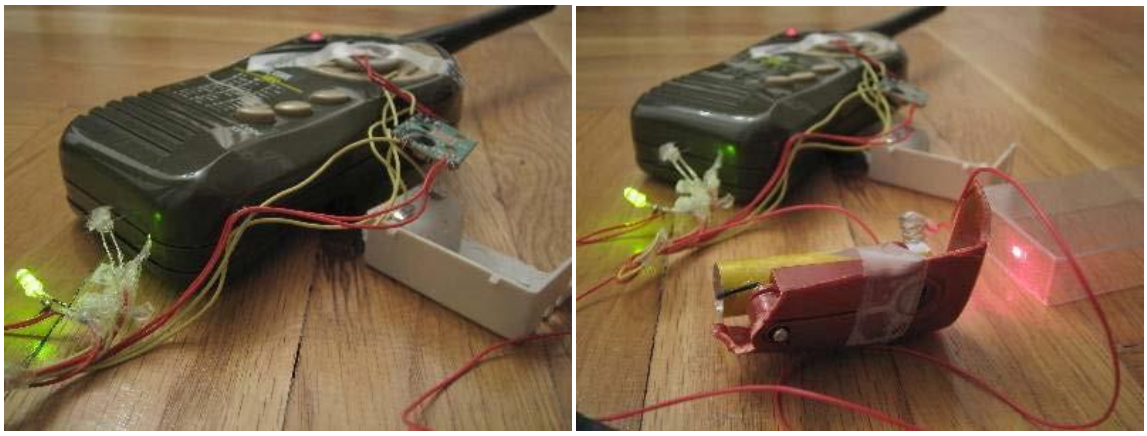
Az első részben komplett rendszereket rakunk össze, a második részben szétszedett játékokból és eszközökből készítünk új áramköröket.

4.1.1. Összetett rendszerek építése

4.1.1.1. Hanggal távirányított lézer

Eszközök: walkie-talkie, hangérzékeny macska, lézer mutatópálca.

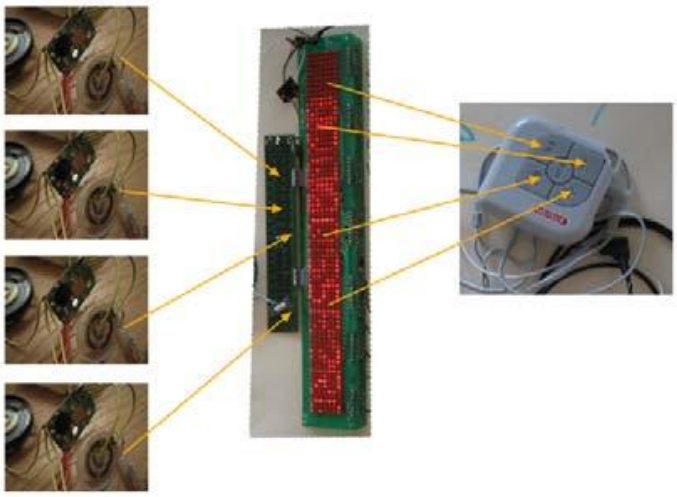
A macskából kiszerelt mikrofont a walkie-talkie hangszórójára helyezzük, hogy érzékelje a másik, távol lévő walkie-talkie-ból jövő hangot. A LED-et a relén keresztül a lézeres mutatópálcához kötjük. Így létrehoztunk egy olyan összetett rendszert, ahol egy másik környezetből jövő távoli hang be tudja kapcsolni a közelünkben lévő lézert, ekkor a LED is felkapcsol (ha ezt kivesszük az áramkörből, tovább kitart majd az elem).



4.1.1.2. Érintésre induló, többhangos rendszer (komparátorral)

Eszközök: hangérzékeny macska, LED mátrix panel és MP3 lejátszó

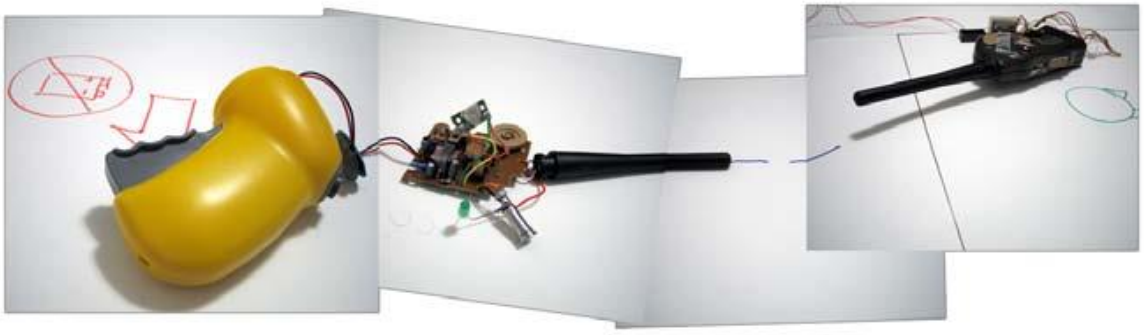
A hangérzékeny macskába számos különböző tárgyat szereltek be, de a borítás megvédi őket, pl. a hangérzékelő szenzort, ami a borítás nélkül ideális érintésszenzorrá válhat. Ezeket a LED mátrix bemenetére rögzítjük, amelynek a kimenete majd az MP3 lejátszónak küld jelet, ami ezáltal a különböző érintésszenzorok megnyomására különböző hangmintákat játszik le. Ez utóbbi attól függ, milyen sorrendben nyomkodjuk a szenzorokat.



4.1.1.3. Önmagát árammal ellátó, távirányított lépésérzékelő

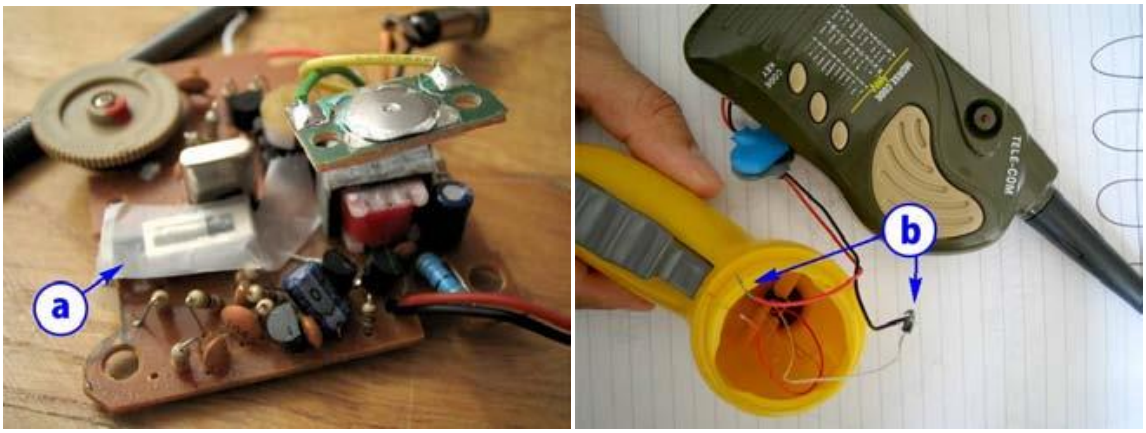
Eszközök: mechanikus elemlámpa, walkie-talkie és hangérzékeny macska

Ez az összetett rendszer valamilyen mechanikus erővel aktiválható, pl. rálép valaki, szélre vagy vízlöketre, amire rádiójelet küld a fogadó egységhez. Mindehhez nincs szükség elemre, mert az áramot mechanikus erő állítja elő. A vevőre bármit szerelhetünk, így ha az megkapja a rádiójelet, lezajlik a folyamat.

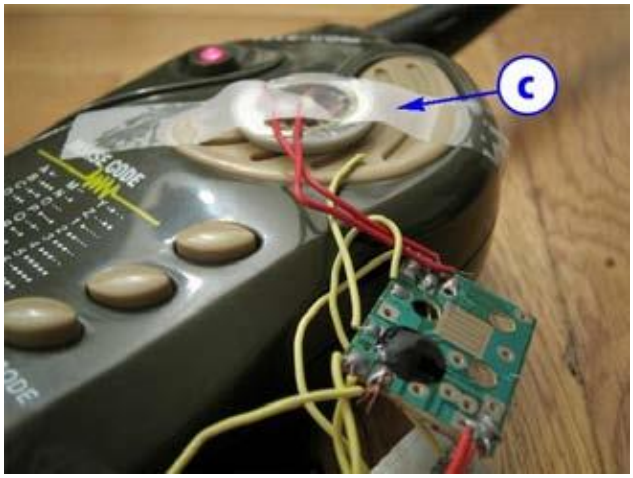


Kezdjük a macskával, szedjük szét és szereljük ki a szemeiből a LED-eket és a mikrofont, a hangszóróra most nem lesz szükség. Az egyik walkie-talkie-ból is szedjük ki az elemeket, és jegyezzük meg, hová voltak csatlakoztatva. Némelyiknek van Morse gombja, ami sípoló hangot vezérel. Ha van ilyen a walkie-talkie-n, véglegesen iktassuk ki, majd az alapgombot is, amelynek megnyomásával beszélni lehet. Szereljük szét a mechanikus elemlámpát, és az azt árammal ellátó drótokat szereljük át a walkie-talkie-ra, figyelve a polaritásra, és emlékezve, hogy az eredeti elem hogyan volt beszerelve. Így van egy olyan eszközünk, ami megnyomásra rádiójelet küld.

Most a macskából kiszedett mikrofont szereljük fel egy ragasztószalaggal a még sértetlen walkie-talkie hangszórójára. A macska LED szemeit pl. egy relén keresztül vezérelhetjük.



A beszélésre szolgáló gombot ragasszuk le a walkie-talkie-n és az elem helyére kössük be a mechanikus elemlámpa áramellátó drótjait.



A macska szenzora érzékeli a walkie-talkie-ból érkező hangokat.

4.1.1.4. Hely feltérképező lézer

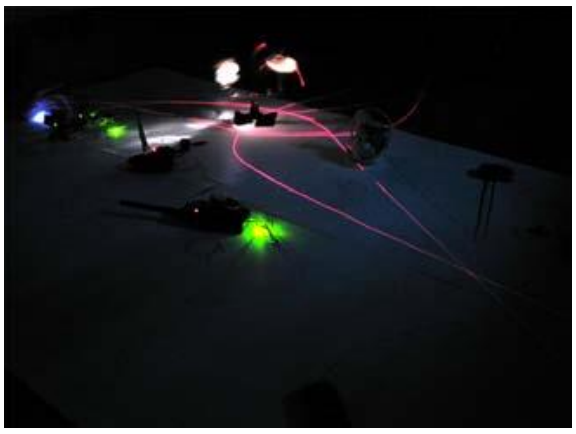
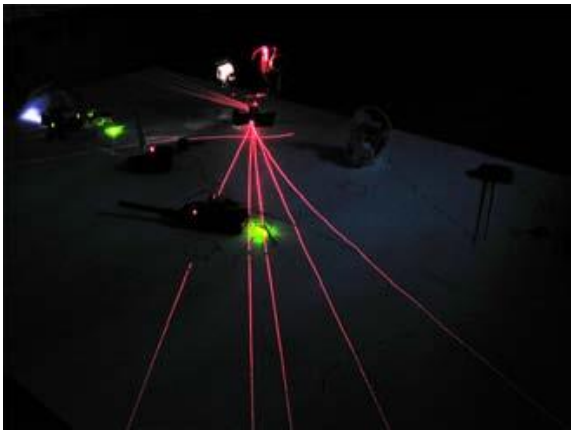
Eszközök: távirányítós autó és lézeres mutatópálca

A sötét térben gyorsan forgó lézerpont a falon egy fénycsíkot képes létrehozni, ami megtörik a sarkokon, így kirajzolódik a tér. Egyszerre több tengely mentén mozgó csíkot is használhatunk a komplexebb látványhoz.



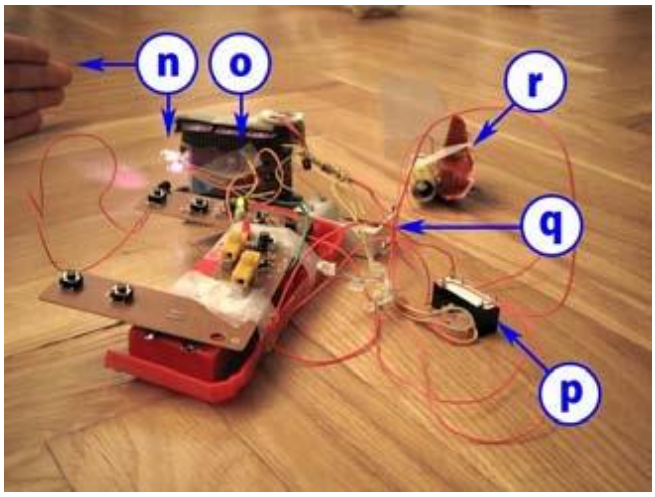

Összerakása nagyon egyszerű, nincs szükség elektronikus szerelésre, mivel csak az autó kerekére kell ragasztószalagozni a lézer mutatópálcát. Több autót is használhatunk, ha összeragasztószalagozzuk őket. Az alábbi képeken a kerekek fixen állnak és az autók teste forog a rászerelt lézerrel.



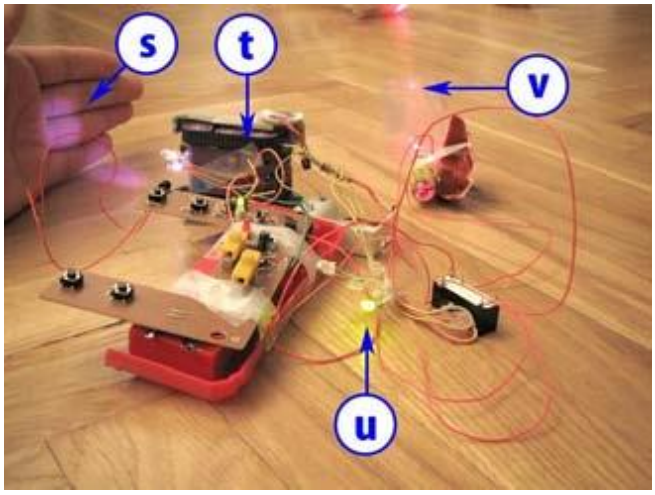



4.1.1.5. Testre reagáló lézer

Eszközök: tank, hangérzékeny macska, relé és lézer mutatópálca

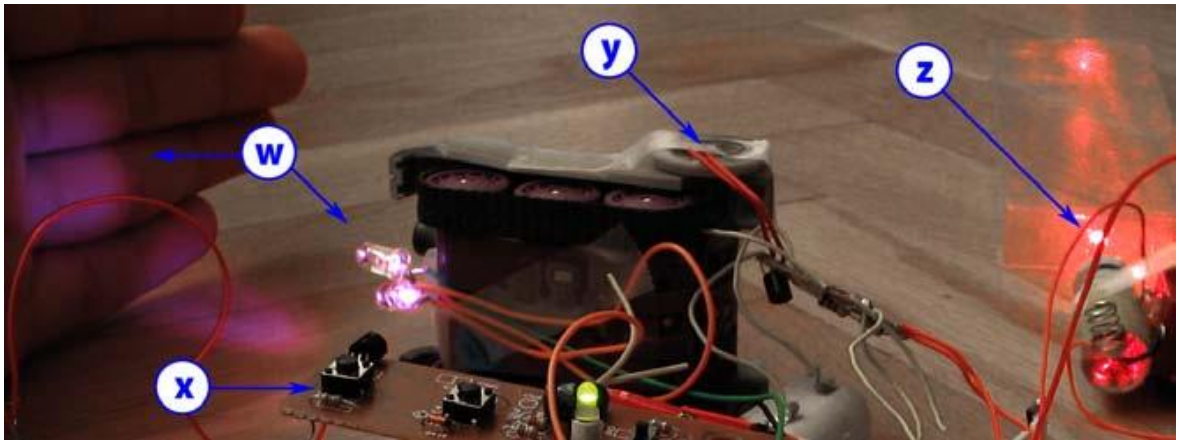


A képen az **n** szenzor mutatja az infravörös LED diódát és a közelében lévő kezet, az **o** egy játéktank, ami az oldalán fekszik, a **q** a macska, ami a relét indítja el (ez a **p)**, így bekapcsol a lézer, ami az **r**.



Az **s** az infrafény, ami felé a kéz közeledik, ami bekapcsolja a **t** jelzésű tankot, amely vibrálni kezd. Ez aktiválja a macska LED szemeit **(u)** és egyúttal a relét is, ez utóbbi bekapcsolja a **z** jelzésű lézert.

Egy közelebbi kép:



w – a két infra LED, **x** – a korábban csatlakoztatott gomb, **y** – a macskából kiszerelt mikrofon, amit arra használunk, hogy érzékelje a tank vibrálását, **z** – a macska által bekapcsolt relé és az utána kötött lézerfény.

4.1.2. Egyedi játékok és szerkentyűk belseje

Most nézzük meg, milyen alkatrészekből állnak ezek a játékok.

4.1.2.1. Relé

LEÍRÁS

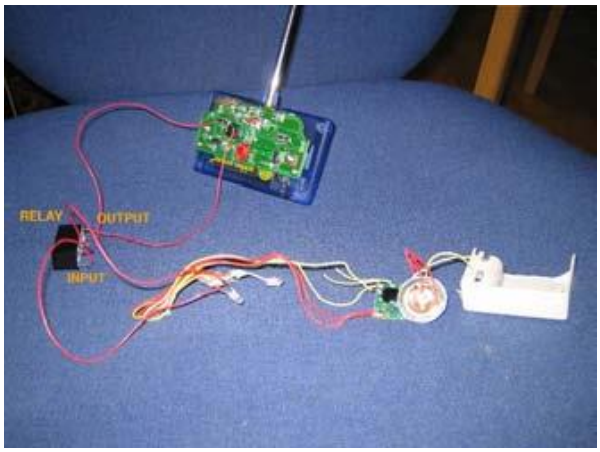
Az egyik legfontosabb alkatrész a használtak közül, lehetővé teszi, hogy különálló eszközöket összekössünk anélkül, hogy elrontsuk őket. A relé lényegében egy „kapcsoló”, ami, ha áramot kap a bemenetén, akkor bezár egy kapcsolót a kimenetén. Ez a legjobb útja annak, hogy vezéreljünk egy eszközt egy másikon keresztül, mivel némelyik analóg szenzor nagyon érzékeny, és ha közvetlenül összekötjük őket, annak előre nem látható következménye lehet. Szinte minden összetett rendszernél használtjuk.

MÓDSZER

Az indító eszköz kimenetét a relé bemenetéhez kell kötni, a kimenetére pedig annak az eszköznek a bemenete kerül, amit el akarunk indítani. A lábak funkciója relénként eltérő lehet, némelyiknek három kimeneti lába van, egy föld és kettő másik, ahol a bejövő jeltől függően nyit vagy zár az áramkör, illetve van olyan, aminek 4 kimeneti lába van, mely páronként nyit vagy zár, ha áram érkezik a bemeneti lábára. Általában kisfeszültségű relét használunk. Érdemes megnézni az adatlapját, hogy ne kelljen annyi verziót kipróbálni. A képen a kis fekete bal oldali hasáb az.

BEMENET
Elektromosság (digitális) – általában 3–24 volt

KIMENET
Kapcsoló (digitális) – ki vagy be állapot



4.1.2.2. Hangérzékeny macska

LEÍRÁS

Hasznos játék, mivel könnyen lehet vele hangra reagáló és világtó dolgokat készíteni. Például tapsra bekapcsolódó vagy hangos léptekre reagáló fényt is alkothatunk vele.

MÓDSZER

A példánkban használt hangérzékelős macska szemei villogni kezdenek és hangosan nyávog, ha a közelében erős hangot, pl. tapsot érzékel. Szedjük szét a játékot, vigyázva a mikrofonra. A szemeit használhatjuk elektromos kimeneteknél, pl. jelzőfényként. a mikrofont pedig hangérzékelőként bármilyen eszköz, pl. motor vezérlésére.

BEMENET
Hang (analóg) – taps, kiabálás, bútortologatás
Érintés (analóg) – ütés, lépés

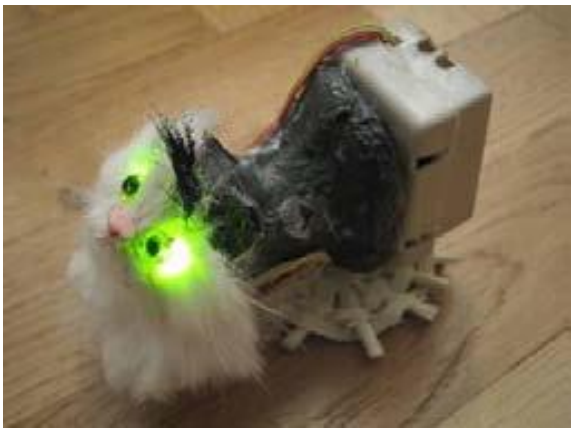
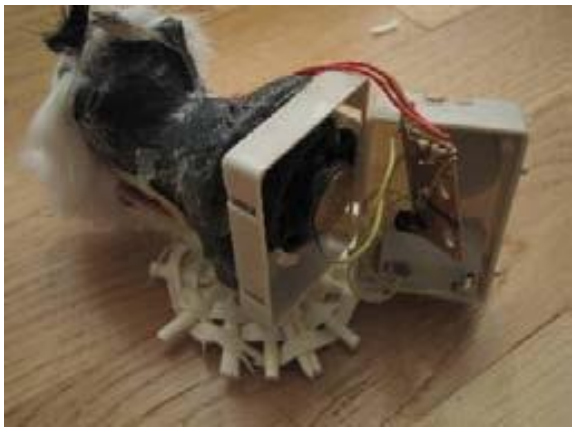
KIMENET
Fény (digitális) – LED szemek villognak
Hang – nyávogás
Elektromosság (digitális) – feszültség halad át a LED-eken

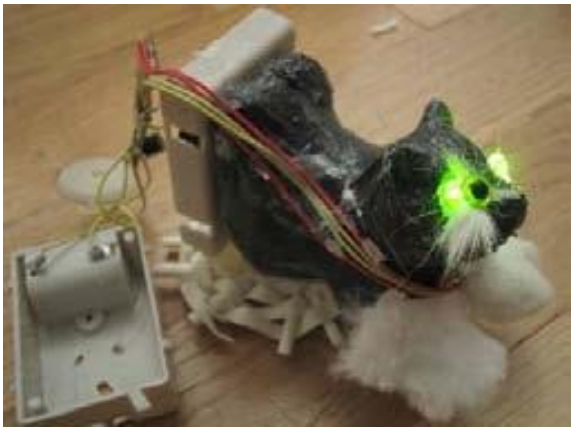




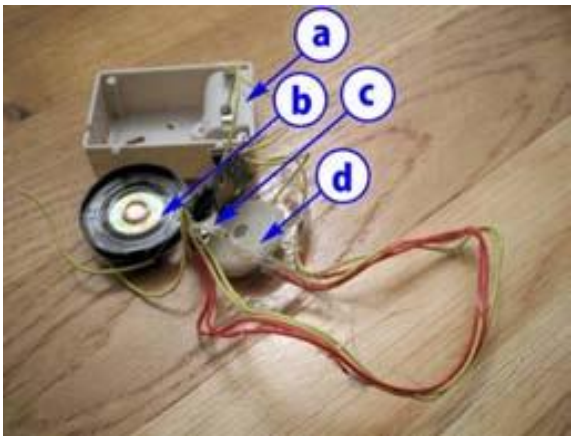
SZÉTSZERELÉSE

A szőrburok alatt a dobozban vannak a szenzorok, ezt kell óvatosan felnyitni, majd ellenőrizni, hogy működik-e még, ha nem, elszakadt valami.

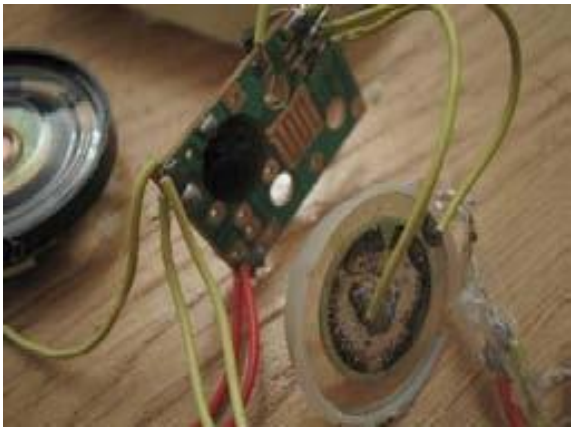



A LED-eket ki kell húzni a fejéből és a dobozból kiszedni az elektronikát: **a** – elem, **b** – hangszóró, **c** – LED-ek, **d** – mikrofon.





Ha közelebbről megnézzük, találunk benne egy integrált áramkört, mely az interakciót vezérli. Az az a zöld tábla, amire az összes kábel rá van kötve. Amit mikrofonnak neveztünk, az nem igazán mikorofon, hanem sokkal inkább rezgésérzékelő, de mivel nagyon érzékeny. a levegő mozgását is észleli. A következő dolgokra lehet használni: A bemenetén a közvetlen környezetében érzékelni hangot, vagy egy eszközre szerelve az abból jövö hangot. A kimenetén villogtatni a LED-et, nyávogó hangot lejátszani, vagy a LED feszültségével beindítani egyéb eszközöket.





4.1.2.3. Elemlámpa

LEÍRÁS

Manapság sok olyan, könnyen elérhető eszköz van, ami zöld energiával működik, pl. napelemes mobiltöltő. Ezeket az eszközöket felhasználhatjuk projektünk áramellátására. Attól függően, hogy hogyan működnek, reagálhatnak kézmozgásra, lépésre, szélre vagy vízsugárra stb.

MÓDSZER

Olyan eszközre van szükség, ami nem elemmel működik és kézmozgással generál áramot. Mi egy elemlámpát használtunk, melynek ha folyamatosan nyomkodjuk a nyelét, akkor világítanak a LED-jei. Ezeket a LED-eket lecserélhetjük a saját rendszerünkre.

BEMENET

Mozgás (analóg) – általában tengely és lendkerék.

KIMENET

Fény (digitális) – feszültség megy a vezetéken át a LED-ekhez (pozitív = piros drót).

SZÉTSZERELÉSE

Viszonylag egyszerű, de nagyon oda kell figyelni, hogy az áramkört ne tegyük tönkre. Ezért jobb, ha nem szedjük szét a burkolatot, csak a LED-eket szereljük ki. Ehhez kell kötni saját rendszerünket, és meg kell tartani az áramgeneráló mechanikát. Alternatív megoldásként relét is vezérelhetünk ezzel a feszültséggel. Ha építünk neki speciális talpat, akár taposó kapcsolóként is működtethető (utolsó kép).



4.1.2.4. LED mátrix

LEÍRÁS

A LED mátrix egy olyan eszköz, amit gyakran láthatunk boltok kirakatában, különböző szövegek futnak rajta. Manapság olcsó, bár az általunk használtak közül ez a legdrágább alkatrész. Nagyon hasznos, pl. bemenetén át vezérelve betűket, szövegeket jeleníthetünk meg a kimenetén, de szekvenszerként (olyan elektronikus eszköz, ami egymás után sorozatban következő utasítások végrehajtására alkalmas) is működhet, ha előre programozott mintákkal vezérlünk eszközöket, és végül komparátorként is használhatjuk. A komparátor egy összehasonlító áramkör, mely két bejövő értéket hasonlít össze. Lehet analóg (kimeneti érték: referencia értékhez képest mér), vagy digitális (kimeneti érték: kisebb, egyenlő, nagyobb lehet).

MÓDSZER

Ideális esetben olyan LED mátrixunk van, ami billentyűzettel rendelkezik, és saját üzenetet lehet beprogramozni neki. Mivel kimeneti eszköz, a LED-eket hozzá tudjuk kötni a második rendszerhez, hogy vezérelni tudjuk velük. Mivel komparátorként használjuk, egy másik eszközt is közvetlenül a billentyűzetbe kötünk, tehát ennek kimenete manipulálja a LED kijelzőn megjelenő mintát.

BEMENET

Relé (digitális) – a LED mátrix bemenete elé kössünk egy relét, hogy megóvjuk annak belső áramköreit.

KIMENET

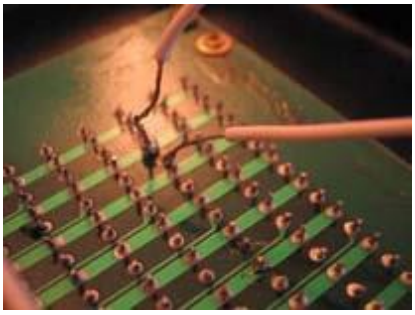
Fényminta (digitális) – LED kijelzőn megjelenő betűk, grafikák.

Elektromosság (digitális) – feszültség megy a vezetéken át a LED-ekhez és tovább.




SZÉTSZERELÉSE

Csavarozzuk le a hátulját és óvatosan húzzuk szét a két burkolót. Egy drót fut az elemcsatlakozótól a LED-ekig. Vigyázzunk, hogy ne szakadjon el! A kijelző hátulján látható, hogy sorokba és oszlopokba vannak rendezve a LED-ek. Ezekre a pontokra tudjuk forrasztani azokat az eszközöket, amiket vezérelni akarunk velük. A LED negatív lábához kell kötni az általa vezérelni kívánt eszközt, azon az oldalon kicsit le van csapva a LED széle. Célszerű a kettő közé relét szerelni.

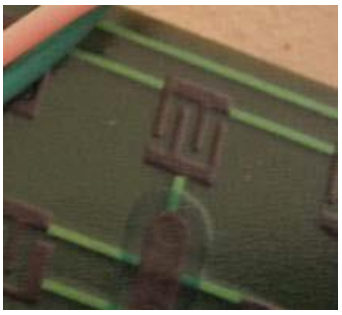





Komplexebb módszer, ha egy relén keresztül egy másik eszköz bemenetébe kötjük a kijelző kimenetét, és szintén egy relén át bekötjük a másik eszköz kimenetét a LED mátrix billentyűzetébe, így a megjelenő üzenetet valós időben generálhatjuk.

Először is a billentyűzetet kell megtalálnunk. Általában ez egy lógó gumilemez lazán ráhelyezve a zöld nyomtatott áramköri lemezre. Ezen keressük meg a betűhöz tartozó részt. Emlékezzünk rá, hogy milyen irányba kell visszatenni. Amikor a betű gombja nincs

megnyomva, akkor az alatta lévő két drót nem ér össze. Ezeket nem kell forrasztani, csak a reléből futó drótokat kell ugyanoda helyezni. Fontos, hogy egy drót csak egy érintkezőhöz érjen hozzá! Ezután ragasszunk rá valami puha anyagot, hogy megvédjük a nyomtatott áramkört.







4.1.2.5. Lézeres mutatópálca

LEÍRÁS

Ezzel az eszközzel távolról fénypont hozható létre egy tárgyon. Alternatív megoldásként ráragaszthatjuk mozgó vagy forgó tárgyakra, hogy fény-vonalat rajzoljunk vele.

MÓDSZER

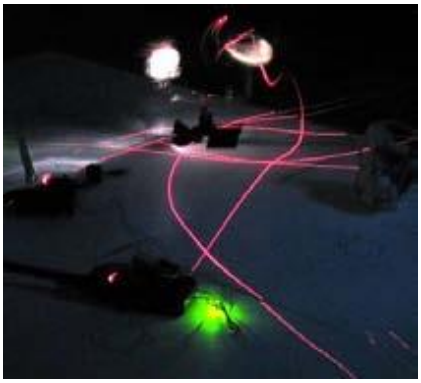
Az alkotó részein belül valószínűleg nincs borítás. Könnyű épségben tartani és egyszerűen csatlakoztatható egy relén át. Távirányítós autó kerekére szerelve fény-vonalakat tud rajzolni.

BEMENET

Elektromosság (digitális) – 4,5 V 3 db ceruzaelemből.

KIMENET

Lézer fény (digitális) – pontot világít tárgyakra.


4.1.2.6. Rulett kerék

LEÍRÁS

Ál-random kimenettel rendelkezik – hasznos más eszközök irányításánál.

MÓDSZER

Relén keresztül interfésznek is használható, vagy ha a LED-ekből tovább vezetjük az áramot, már eszközök is vezérelhetők vele. Bemenetére szintén kell egy eszköz relén át, amivel a forgás szekvencia elindítható.

BEMENET

Elektromosság (digitális) – 4,5 V DC (egyenáramú).

Nyomógomb (digitális) – elindítja a szekvenciát.

KIMENET

LED (digitális) – LED kimenetek sorozata, pár másodperc után megáll az egyiken.




4.1.2.6. MP3 lejátszó

LEÍRÁS

Indításra előre felvett hangot játszik le.

MÓDSZER

MP3 audio fájlt kell rátölteni. Egy relén keresztül lehet a bemenetén át vezérelni a lejátszást vagy megállítást, ami vezérelhet egy hangszórót vagy fejhallgatót.

BEMENET

Elektromosság (digitális) – 1,5 V DC.

Nyomógomb (digitális) – indítás, pillanat állj vagy zenekiválasztás.

KIMENET

Hang (digitális) – hangszóró vagy fejhallgató kell hozzá.



4.1.2.7. Vízzel működő digitális óra

LEÍRÁS

Vízzel állít elő elektromosságot – nem kell hozzá elem.

MÓDSZER

Az elem helyére kell bekötni. Alacsony feszültséget produkál.

BEMENET

Víz (analóg) – vizet engedünk a tartályba.

KIMENET

Elektromosság (analóg) – kb. 50 mV.



4.1.2.8. USB kamera

LEÍRÁS

Olcsó USB kamera. Az interakció részeként embereket és tárgyak fotózhatunk.

MÓDSZER

A zárkioldó gombot használhatjuk relén keresztül, mint bemenetet. Később a képeket USB-n letölthetjük.

BEMENET

Nyomógomb (digitális) – zárkioldó gomb.

KIMENET

Fotó (digitális) – kb. 320 x 240 pixel, kis felbontású képek.



4.1.2.9. Napelemmel működő kerti lámpa

LEÍRÁS

„Napfény-érzékelőként” és áramforrásként is használható.

MÓDSZER

A LED-ből kössük össze egy relén keresztül egy másik eszközzel.

BEMENET

Napfény (analóg) – kapcsoló.

KIMENET
Elektromosság (digitális) – kb. 3 V.



4.1.2.10. Ködgép

LEÍRÁS
Hasznos „atmoszféra” létrehozásánál, vagy, hogy felfogja a fényalapú kimeneteket.

MÓDSZER
Használd lézerfénnyel vagy vetítő eszközzel együtt.

BEMENET
Elektromosság (digitális) – hálózati áramot egyenárammá alakító adapter. Egy gyűjtőedényben a vízfelszínen ki-bekapcsol az áram.

KIMENET
Köd (analóg) – a gyűjtőedény pereménél bukkan fel.



4.1.2.11. Infravörös játékpisztoly, távolságérzékelés

LEÍRÁS
A távolság érzékelése nagyon hasznos, elmenő embereket, mozgó tárgyakat érzékelhetünk. Mivel nincs szükség ehhez érintkezésre, ezért a mechanikus meghibásodás kizárt. Általában a kereskedelmi forgalomban kapható távolság szenzorok ultrahanggal működnek, a visszaverődő hangot és a közben eltelt idő nagyságát figyelik (mint a

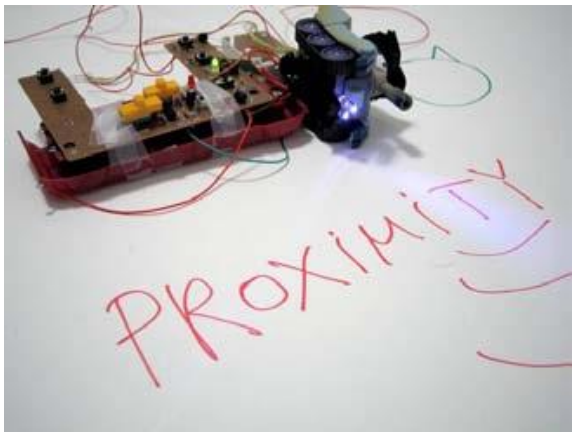
denevérek), ezt számolják át távolsággá. Álalában arra használják őket, hogy megtudják egy tárgy vagy személy adott helyen van-e, vagy nincs ott semmi. Ezt a funkciót most egy olcsóbb szenzorral hozzuk létre.

MÓDSZER

Infravörös fény visszaverése:

A legegyszerűbb módja a távolság mérésének az infra LED és mellé egy infravörös vevőegyüttes alkalmazása. Az infrafény visszaverődik a tárgyról, tehát ugyanúgy megvilágítja, mint bármilyen fény, csak ezt nem látjuk. Manapság léteznek olyan játékok, melyekbe ezek vannak beszerelve, pl. játékpisztolyok, ahol mindkettő adóval és vevővel is rendelkezik. Mi egy játéktankot szereltünk szét.

A LED-et és a fogadót egymással szemben kell elhelyezni, különben nem látják egymást, így két eszközre lesz szükségünk. Az első képen kék színnel jeleztük az infra fényt, mivel az emberi szem számára láthatatlan, de pl. mobil kamerán keresztül kék vagy piros fénye szokott lenni. Ez hasznos lehet, ha le akarjuk követni, hol van.





BEMENET

Nem átlátszó dolgok távolságát méri, pl. mozgó dolgokét is.

KIMENET

Vibrálás, mozgás, LED.

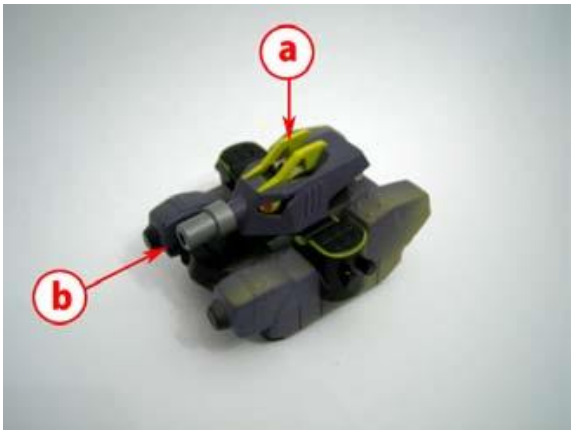
SZÉTSZERELÉSE

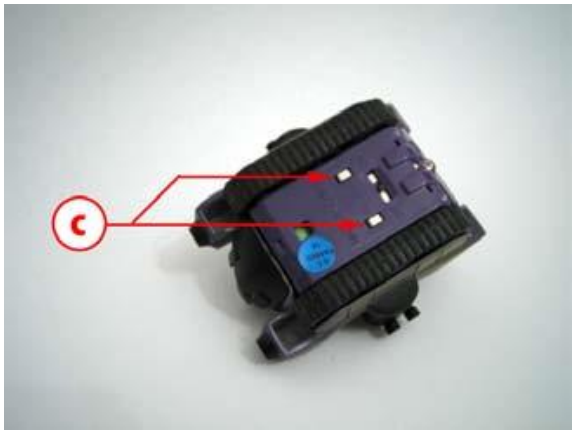
Ebben a példában egy kis távirányítású játékot használunk, aminek egy infravörös fegyvere van, mellyel fogja az ugyanolyan játékból jövő jelet és lő rá.

Mit kell tennünk:
- szedjük le a borítást, hogy hozzáférjünk a távirányítóhoz és az infra eszközökhöz,
- a kocsit és az infra LED-et helyezzük ugyanabba az irányba,
- találjuk ki, hogyan lehetne folyamatosan lenyomva a távirányító gombja,
- találjunk ki egy interakciót, amit ezzel a jellel indítunk el.

A tank:

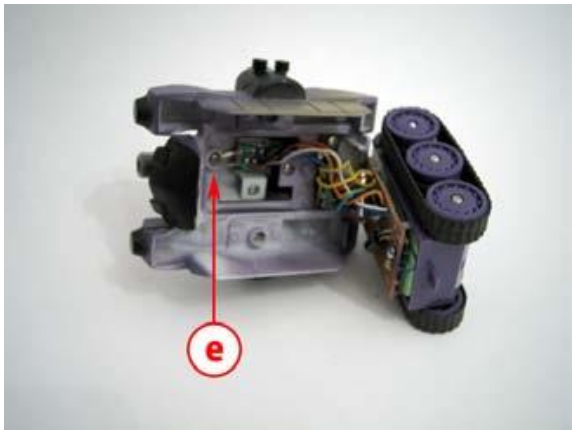
A kis LED-ek lehetnek küldők vagy fogadók – **a, b**. Néha fényes átlátszó fekete lappal vannak fedve. Áramellátás – **c**, itt lehet újratölteni.





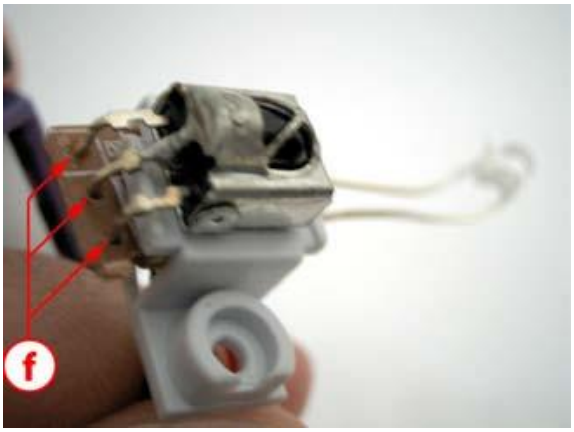
Szedjük le óvatosan a műanyag borítást – lehet, szét kell törni, mert általában ragasztottak –, majd nézzük meg, hogy az infra LED-hez futó drótok épek-e. A LED-ek helyét **d, e** mutatja.



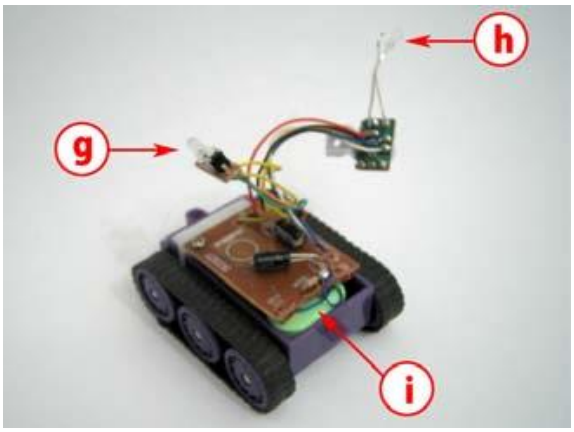


 © Harsányi Réka, Juhász Márton András

Szabadítsuk ki az infra fogadót, aminek három lába van – **f**. A sima LED-eknek kettő van.

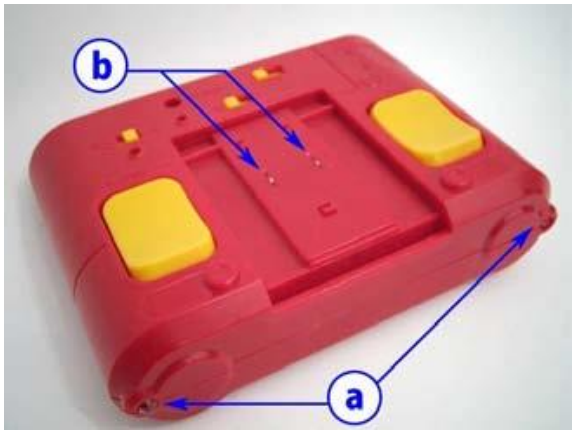


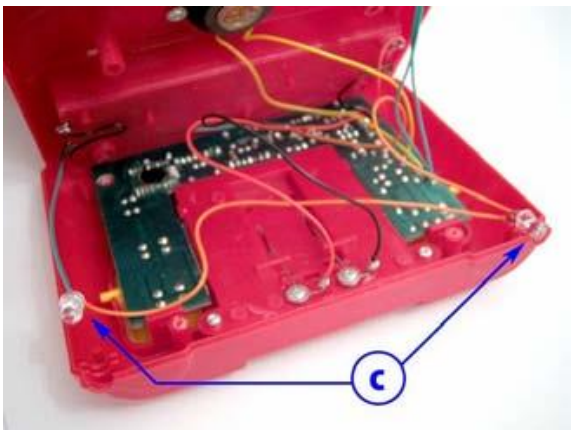
A dróton több dolgot találunk: **g** – infra vevő, **h** – infra adó, **i** – elemek.



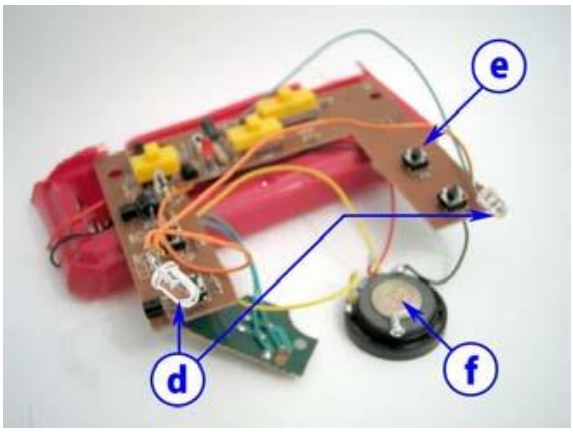
A távirányító:

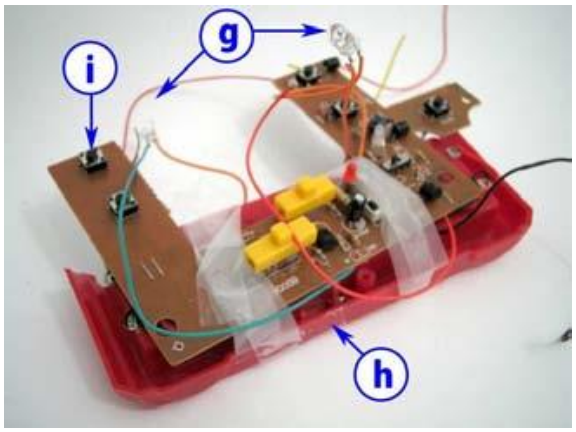
Először az **a**-val jelölt LED-eket keressük meg, b – a töltőegység, c – szintén a LED-ek. Ha felnyitjuk a burkolatot, könnyebb kiszedni őket.





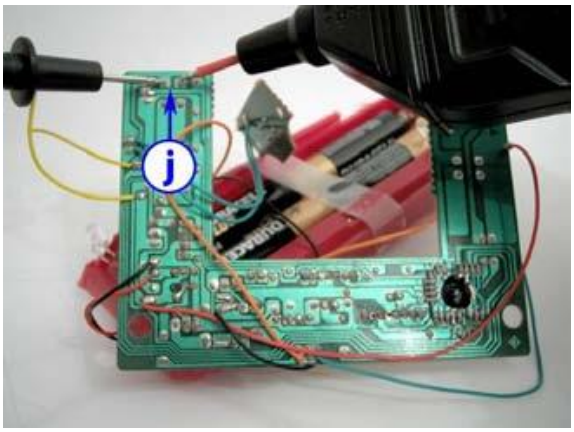
Távolítsuk el a tokot, hogy hozzáférjünk a belső részekhez. Az elemet (**h**) ragasztószalagozzuk rá. Diódák – **d, g**. Hangszóró – **f** (ezt levághatjuk). Kis nyomógombok – **e, i**, ezekkel lehet a kocsi mozgását irányítani.

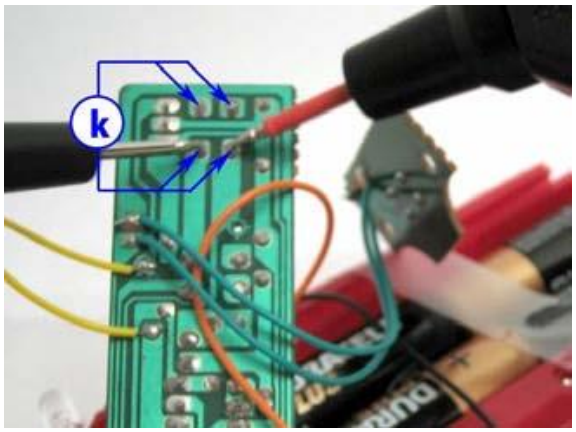


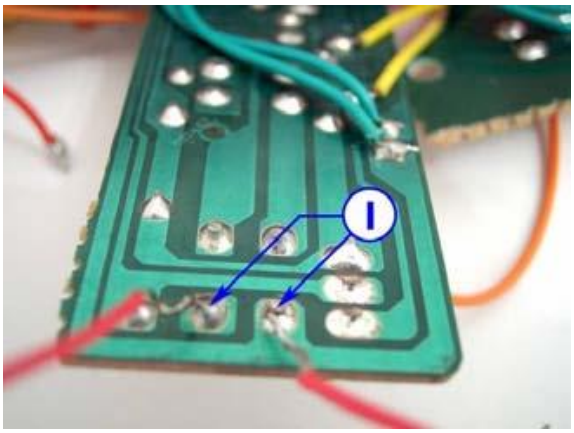


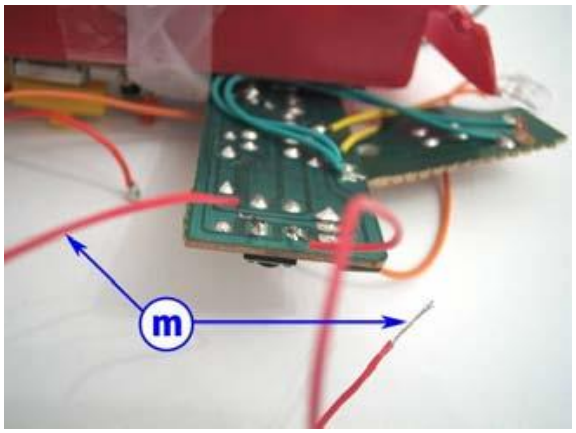
Fordítsuk meg a nyomtatott áramkört!

A **j** és **k** mutatják a nyomógombok hátulját, 4–4 lábat találunk. Ezekre forraszthatjuk az eszközöket, hogy vezérelni tudjuk őket. A nyomógombok két felső lábát kell használni hozzá – **i, m**.









 © Harsányi Réka, Juhász Márton András

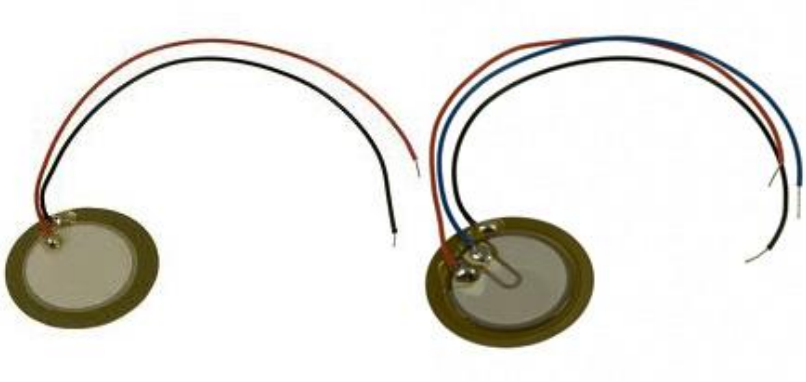
4.2. Készítsünk kontakt mikrofont piezo szenzorból

A piezo érzékelő nagyon olcsó és elterjedt, több méretben kapható és könnyű vele rezgést érzékelni. Van csupasz verziója, melyet, ha ráragasztunk egy tárgyra, pl. hangszerre, akkor érzékelhetjük a hanghullámokat is. Létezik műanyag burkolatos piezo, vagy más néven buzzer, mely furcsa, zümmögő hangot bocsát ki, de pl. kopogás érzékelésére is használható, ha megfelelően le van rögzítve (ilyen kopogásérzékelőt fogunk majd összerakni Arduinoval egy későbbi lecke során). Most egy csupasz piezot szereljünk rá egy gitárra, és a jelet jack kábelen továbbítsuk egy laptopba vagy erősítőbe!

Ehhez a következő alkatrészekre lesz szükségünk:

- elemmel működő erősítő (vagy számítógépen audio bemenet),
- csupasz piezo lemezek,
- 2 méteres árnyékolt kábeldarabok,
- Jack csatlakozók (vagy olyan, ami az erősítőbe kell),
- forrasztó eszközök és szigetelőszalag,
- krokodil-csipeszek,
- tárgyak, amik rezegnek és ráhelyezhetjük a szenzort, pl. fémdarab, gitárhúr.

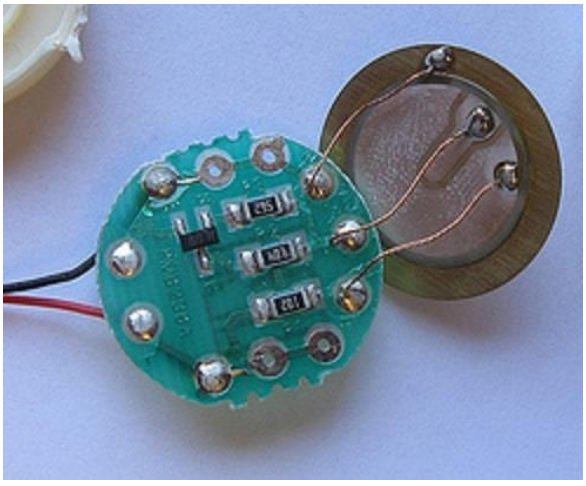
Most készítsük el a saját „kontakt mikrofonunkat”!



4.1. ábra: Csupasz piezo érzékelők



4.2. ábra: Piezo berregők



4.3. ábra: Burkolatból kiszerelt piezo NYÁK-kal

 © Harsányi Réka, Juhász Márton András



4.4. ábra: Krokodil-csipesz



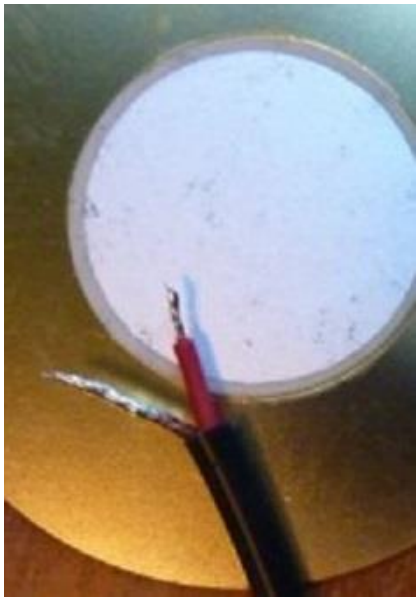
4.5. ábra: Sztereó jack dugó



4.6. ábra: Árnyékolt audio kábel

Kicsit nehéz ráforrasztani a lemezre a drótot, de folyatószer használatával sikerülni fog. Érdemes több piezot vásárolni arra az esetre, ha az első nem sikerülne. Olcsó és könnyen beszerezhető az elektronikai boltokból. Csupasz lemezre van szükségünk, ezért ha csak burkolt van, akkor óvatosan le kell szedni róla a műanyag részt, hogy ne sérüljön meg. Ha van rajta kis nyomtatott áramkör, vágjuk le, de a drótot, ami a piezohoz csatlakozik, hagyjuk meg. Két vagy három vezeték csatlakozik a piezohoz. A fém peremhez csatlakozik a föld, egy pedig a kristályos részről, középről jön, ez a pozitív szár. Néha van egy harmadik vezeték is, ami egy nyelvszerű keskeny részhez csatlakozik, ami el van választva a kristálytól, ezt nem szükséges bekötni. Forrasszuk rá a piezora az árnyékolt kábelt is. Blankolás közben figyelni kell, hogy a fémszálakat ne vágjuk el. A drót közepén futó piros szálat a kristályos részhez, az oldalsó vagy fekete szálat a piezo fém pereméhez kell kötni! Látható, hogy az árnyékolt kábel (a középső jelkábel) több burkolattal rendelkezik, ez szigeteli el a fő részt, a külső környezeti zajoktól.

 © Harsányi Réka, Juhász Márton András



4.7. ábra: Piezo csatlakoztatása hangkábellel

A kábel másik végére szereljük rá a jack dugót! A jack dugó belsejében van két „tartó” szár, rajta kis lyukkal, és egy nagyobb csatlakozórész is. A lyukasba bele lehet fűzni a kábel végét, így könnyebb a forrasztása. Ha nincs rajta ilyen, a krokodil-csipeszekkel össze lehet fogni az elemeket. Ahhoz, hogy jel érkezzen a dugóhoz a kábelen keresztül a piezoból, még rá kell forrasztani a jackre a jelvezetéket, ami a piros középső szál, és a szélesebb, kijjebb eső fülre a földvezetéket, ami a szélső szál. A jack dugó és az audio kábel lehet mono vagy sztereó is, a sztereóban két szál fut középen és a dugónak két kilyukasztott füle van.



4.8. ábra: Sztereó jack dugó belseje



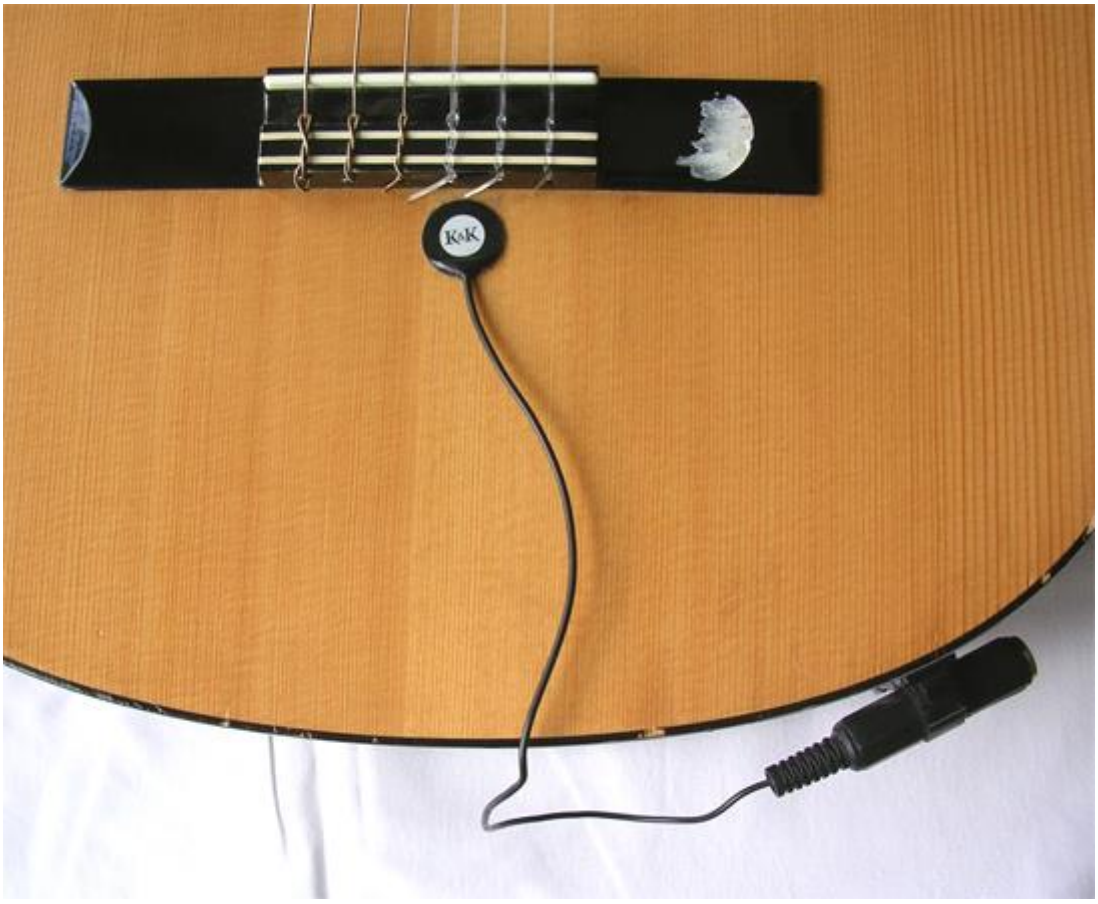
4.9. ábra: Sztereó jack dugó összeforrasztva hangkábellel

 © Harsányi Réka, Juhász Márton András



4.10. ábra: Mono jack dugók hangkábellel

A piezoban keletkező feszültségváltozás a jack dugón keresztül audio jellé alakul. A piezot a fémlapjával ragasszuk fel a rezgő tárgyra, és dugjuk be az erősítőbe vagy a számítógépbe a jack dugót! Utóbbi esetben egy valamilyen hang digitalizálására is alkalmas szoftver is kell, amivel felvehetjük a bejövő jelet, majd visszajátszhatjuk.



4.11. ábra: Felragasztott piezo szenzor

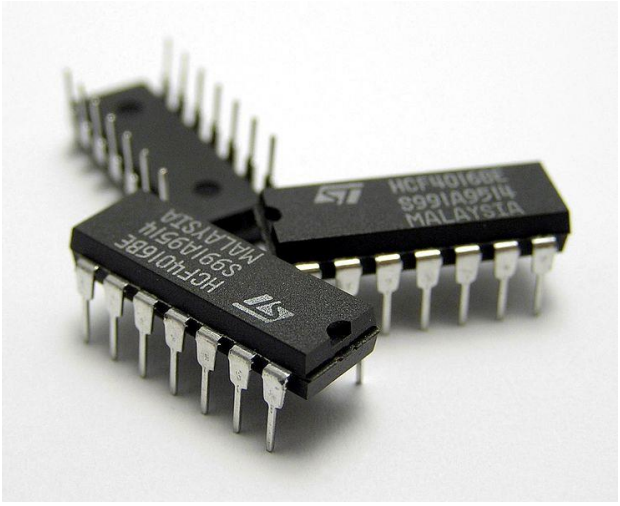
 © Harsányi Réka, Juhász Márton András

5. Logikai kapuk

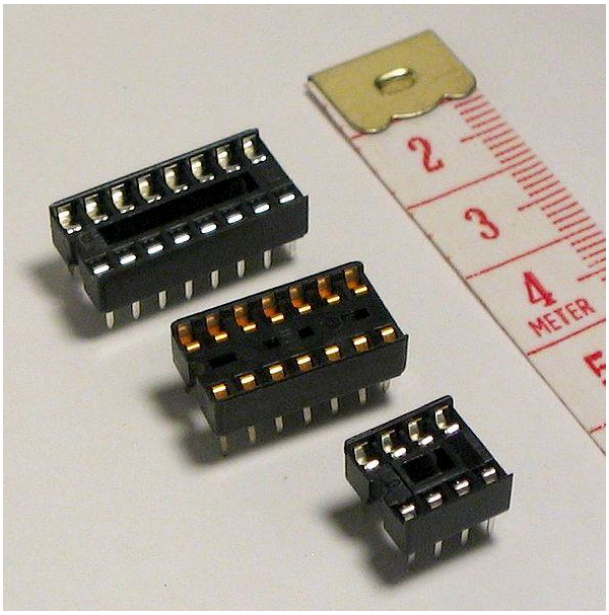
5.1. Integrált áramkörök

Az integrált áramkörök (rövidítése: IC) egy bizonyos feladat ellátására képes mini áramkörök, amik akár több száz analóg alkatrészt is tartalmazhatnak, például tranzisztorokat, ellenállásokat. Viszonylag olcsóak és könnyen beszerezhetőek.

Különböző tokozásúak lehetnek, a legelterjedtebb a DIP = Dual-Inline Pin műanyag tokozás (mi is ilyet fogunk használni a kísérletben). Egy szögletes, műanyag, fekete téglatest alakú házból, két oldalon egy-egy sorban kiállnak a lábak, más néven pinek. Hosszuk és a lábaik száma változó, lehet 8, 14, 16, 28, vagy 40 pines. Ezt a tokozási típus könnyen használható dugaszolós próbapanelben, ha pedig be szeretnénk forrasztani az áramkörbe, DIP foglalatra lesz szükségünk, így megvédhetjük az IC-t az esetleges forrasztás okozta károktól, mivel csak utólag bele kell tenni a tokba.

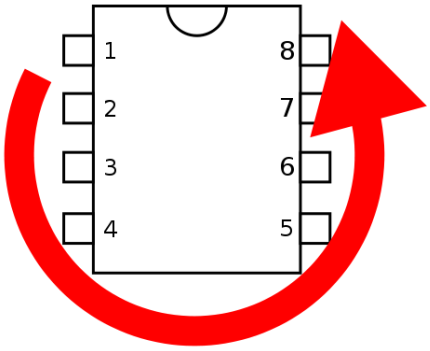


5.1.1. ábra: DIP tokozású integrált áramkörök jelölése



5.1.2. ábra: DIP foglalatok

A fekete tok egyik oldalán egy félkör alakú bemélyedés látható, innen lehet beazonosítani, hogy melyik irányból kezdődik a lábak számozása. Ha felfelé áll a jel, a bal oldali felső az egyes láb, innen az óramutató járásával ellentétes irányban számozzák a többit. Azt, hogy az adott IC milyen feladat ellátására képes, az adatlapján nézhetjük meg, mint ahogy azt is, hogy melyik kivezetése mit csinál.



5.1.3. ábra: DIP tokozású integrált áramkör kivezetéseinek számozása

A chipek logikai feladatokat végeznek el, ezért most térjünk rá a logikai kapukra és az általuk alkalmazott kettes számrendszerre!

5.1.1. Kettes számrendszer

A kettes számrendszer ismerete fontos a logikai kapuk és a későbbiekben tárgyalt digitális jelek megértése szempontjából.

A mindennapokban a **decimális**, azaz a tízes számrendszert használjuk. Helyiértékes számrendszer, amiben a számjegyek helyiértékei a tíz hatványai, elnevezései pedig: egyesek, tízesek, százasok, ezresek, stb. A számban jobbról balra értelmezve nőnek. Minden helyiértéken ugyanazokat a jeleket használjuk. Számjegyei: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ezek egyben a számok úgynevezett alaki értékei, ugyanaz a számjegy más-más értékű aszerint, hogy hol helyezkedik el a számban. Nem egész számok is lehetnek benne, úgynevezett tizedes törtek. Például:

$864 = 8x10^2 + 6 \times 10^1 + 4 \times 10^0 = 8 \times 100 + 6 \times 10 + 4 \times 1.$

A digitális információk tárolására a kettes (**bináris**) számrendszert használjuk. Olyan helyiértékes számrendszer, aminek két számjegye van: 0 és 1, melyeket bitnek is neveznek. A helyi értékek kettő hatványaiként írhatók le, jobbról balra haladva növekvő sorrendben. Például így lehet átszámolni négy bitet tízes számrendszerbe:

$1010 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 8 + 0 + 2 + 0 = 10$

Bitekből állnak a bájtok. Egy bájt 8 bitnek felel meg, de pozitív vagy negatív is lehet egy bináris szám Ilyenkor az előjelet is ez a két érték tárolja (úgynevezett előjelbit), a 0 a plusznak, az 1 a mínusznak felel meg, ezután következik a szám abszolút értéke. Ez egy bájt esetén azt jelenti, hogy a 8 bitnyi helyből egyet elfoglal az előjel, így csak hét hely marad a számnak, tehát az előjeles számok legnagyobb vagy legkisebb értéke kisebb, mint a nem előjeleseké, mert ott mind a nyolc bit a szám értékét tárolja. Ennek a programozásban lesz jelentősége a különböző adattípusoknál. Például nézzünk meg egy előjeles bájtot:

01101001 = plusz előjelű mert 0, ez után következnek a bitek: $1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ = plusz előjel $1 \times 64 + 1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$ = plusz előjel $64 + 32 + 0 + 8 + 0 + 0 + 1 = +105$

5.1.2. Logikai kapuk, logikai függvények

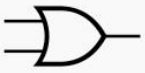
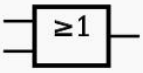
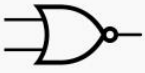
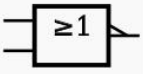
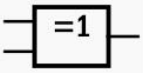
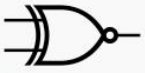
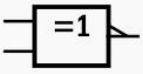
George Boole angol matematikus dolgozta ki a matematika nyelvén a logikai törvényszerűségeket, amit Boole-algebrának vagy **logikai algebrának** hívunk. Alapvetően arról szól, hogy egy állítás igazságértéke lehet „igaz” vagy „hamis”. Ezt a két értéket könnyen lehet a bináris számrendszerben reprezentálni; ha a kijelentés igaz, akkor logikai értéke 1, a hamisnak pedig 0.

A **logikai függvények** segítségével tudjuk matematikailag leírni a logikai feltételek és az események közötti összefüggéseket. Független változónak a logikai feltételeket, függő változónak a hatására bekövetkező eseményt hívjuk. A logikai függvényeket többféle módon is leírhatjuk, pl. szövegesen, táblázatosan, logikai vázlatként, algebrai alakban, grafikusan. Számunkra a kombinációs logikai hálózatok lesznek fontosak a programozásban, ezen belül a logikai kapuk. A logikai függvényeket két csoportra osztjuk, egyrészt a **kombinációs hálózatokra**, ezek időfüggetlenek, másrészt a **szekvenciális** (sorrendben futó) **hálózatokra**, ezek időfüggőek. A különbség a kettő között, hogy a szekvenciális hálózatokban a pillanatnyi bemeneti feltétel mellet egy korábbi időpillanatban létrejött kimeneti jel is befolyásolja az aktuális kimeneti értéket.

Némelyik integrált áramkör akár több logikai kaput is tartalmazhat, mivel a kimeneti értékük is logikai, így továbbítható a következő bementére. A **logikai kapu** olyan áramkör, ami a logikai alapfüggvényeket hajtja végre. Ezek a következők lehetnek: NEGÁLÁS (NOT), ÉS (AND), VAGY (OR) és ezek kombinációja: NEGÁLT ÉS (NAND), NEGÁLT VAGY (NOR), KIZÁRÓ VAGY (XOR), NEGÁLT KIZÁRÓ VAGY (XNOR). Ezek is rendelkeznek rajzjellel, a négyszögletes a nemzetközi szabvány jel, de a másik, amerikai szabvány terjedt el. A logikai kapuk 2, 3, 4 vagy 8 bemenetűek lehetnek, de kimenetük csak kétféle értéket adhat, igazat vagy hamisat. Igazságtáblázatuk különböző, azt mutatják meg, hogy milyen logikai értéket tud kiadni a kapu a különböző bemenetek kombinációitól függően. Ne felejtsük el, hogy mindez bináris számrendszerben történik, így a bemenet is 0 vagy 1 lehet éppúgy, mint a kimenet.

A későbbiekben olyan áramköröket rakunk majd össze a 74xx család tagjainak segítségével, amelyek során megértjük, melyik logikai kapu hogyan működik.

 © Harsányi Réka, Juhász Márton András

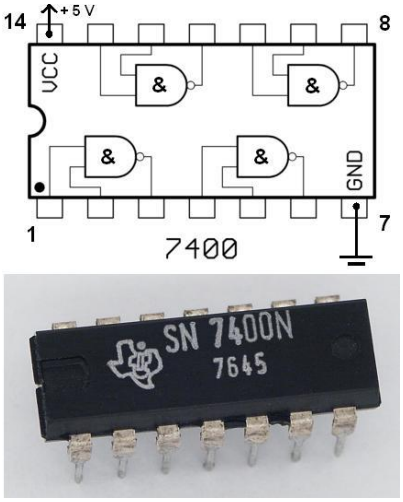
<table>
<tr><th>Kapu</th><th>hagyományos jel</th><th>szögletes jel</th><th>művelet</th><th>Igazságtábla</th></tr>
<tr><td>AND (és)</td><td>  </td><td>  </td><td>$A \cdot B$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td>A</td><td>B</td><td>A AND B</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>OR (megengedő vagy)</td><td>  </td><td>  </td><td>$A + B$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td>A</td><td>B</td><td>A OR B</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>NOT (negálás)</td><td>  </td><td>  </td><td>$\overline{A}$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td colspan="2">A</td><td>NOT A</td></tr><tr><td colspan="2">0</td><td>1</td></tr><tr><td colspan="2">1</td><td>0</td></tr></table></td></tr>
<tr><td colspan="5">A NOT kaput az elektronikában nevezik még invertemek is, hiszen gyakorlatilag megfordítja, idegen szóval invertálja a bemenetként kapott igazságértéket.</td></tr>
<tr><td>NAND (negált és)</td><td>  </td><td>  </td><td>$\overline{A \cdot B}$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td>A</td><td>B</td><td>A NAND B</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>NOR (negált vagy)</td><td>  </td><td>  </td><td>$\overline{A + B}$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td>A</td><td>B</td><td>A NOR B</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>XOR, EXOR vagy MOD2 (kizáró vagy, antivalencia)</td><td>  </td><td>  </td><td>$A \oplus B$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td>A</td><td>B</td><td>A XOR B</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>XNOR vagy EXNOR (negált kizáró vagy, ekvivalencia)</td><td>  </td><td>  </td><td>$\overline{A \oplus B}$</td><td><table><tr><th colspan="2">bemenet</th><th>kimenet</th></tr><tr><td>A</td><td>B</td><td>A XNOR B</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
</table>

5.1.2.1. ábra: Logikai kapuk típusai, rajzjelei és igazságtáblázatai

5.1.3. Az alkatrészszám leolvasása

A chipek tokozásán látható egy alkatrészszám, például SN74HC00N (ha kétsoros a felirat, a másodikkal nem kell foglalkozni). Ez a szám információkat tartalmaz számunkra. A fenti példában a 74 szám azt jelenti, hogy a 7400-as logikai családba tartozik (a 74xx is ugyanennek a jelölése). A számot megelőző nagybetűk a gyártóra utalnak, ez pillanatnyilag nem fontos információ számunkra, ezért át is ugorjuk. A logikai családot követő nagybetűk már fontosak, a nemzedéket jelzik (L, LS, C, HC, AHC stb.). Az újabbak gyorsabbak és sokoldalúbbak az előzőeknél. A HC generáció már ideális számunkra. Ezt követően ismét számokat találunk, minimum kettőt, melyek szintén a logikai kapura utalnak (például a 00 NAND kaput jelent). Ennek a három kódnak az ismeretével már meg tudjuk keresni a számunkra szükséges IC-t egy katalógusból. Végül ismét betűket látunk, melyek a tokozás típusát jelentik (N a DIP tok jele).

Bizonyos digitális IC-k a logikai kapuk mechanikai megvalósulási formái, elektromos áramkörök. Programozható verzióban is léteznek. A CMOS szabványtípusban a következő logikai kapuk fordulnak elő (a mikrochip tetején egy szám jelzi, hogy milyen feladat ellátására alkalmas): 7400 : NAND, 7402 : NOR, 7404 : NOT, 7408 : AND, 7432 : OR, 7486 : XOR). Általában 5 voltos tápfeszültséggel működtethetőek, a megfelelő kimenetre kötve.



5.1.3.1. ábra: 7400 típusú mikrokontroller négy NAND kapuval, SN a gyártó kódja, N a DIP tok jelölése

TTL chipek – a bipoláris tranzisztorokhoz hasonlóan – sokkal inkább az áramra érzékenyek, sem mint a feszültségre. Emiatt a működéshez jelentős mennyiségű áramot igényelnek. A CMOS chipek a programozható, unijunction tranzisztorokhoz hasonlítanak. A feszültségre érzékenyek, ami lehetővé teszi, hogy minimális legyen az áramfelvételük a bemeneti oldalon.

Mindkét család a – TTL és CMOS – ma is létezik. A CMOS széria érzékeny és könnyen tönkremegy statikus elektromosság hatására, – tehát ha feltöltődik a ruhánk és „megrázzuk” az IC-t, miközben hozzáérünk, –, mégis értékes az alacsony áramfelvétele miatt. A TTL IC-k sokkal több áramot fogyasztanak, viszont kevésbé érzékenyek és nagyon gyorsak. Ha pl. egy számítógépet akarunk építeni, akkor a TTL család a jó választás, viszont ha egy kis kütyüt szeretnél, ami hetekig megy elemről, akkor a CMOS család a megfelelő. A CMOS gyártók mindent megtesznek azért, hogy ezeket a hátrányokat csökkentsék. A modern CMOS chipeknél ez bizonyos mértékig meg is valósult.

Az alábbi táblázatban a két IC családot hasonlítjuk össze. A CMOS oszlopban lévő kérdőjelek arra utalnak, hogy fejlesztések eredményeként bizonyos IC generációknál elhanyagolhatók lehetnek a különbségek.

IC családok összehasonlítása	TTL	CMOS
érzékeny statikus elektromosságra	kevésbé	jobban?
sebesség	gyorsabb	lassabb?
energiafogyasztás	magasabb	nagyon alacsony
tápfeszültség terjedelme	5 V keskeny sáv	3 V – 6 V széles sáv
bemeneti ellenállás	alacsony	nagyon nagy

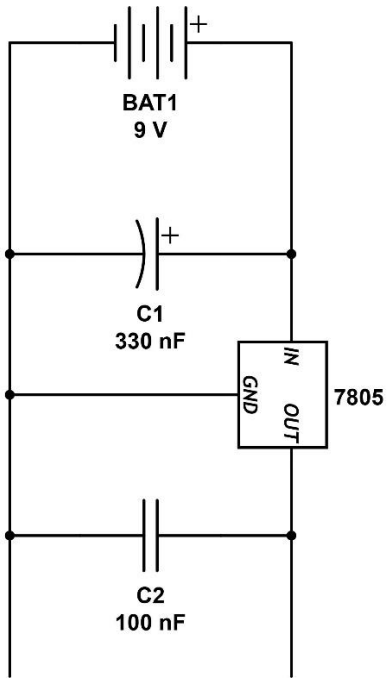
A CMOS családba tartozó 74HCxx generációs IC-ket használjuk ebben a könyvben, mert széles körben elérhetők/elterjedtek, és a bemutatott áramköri alkalmazások nem igényelnek sem nagy sebességet, sem nagy energiát. Az IC-k cikkszáma és a lábkiosztás megegyezik a TTL családba tartozó IC-kkel, a be- és kimeneti feszültségek azonban nem.

5.1.4. Logikai áramkörök összerakása

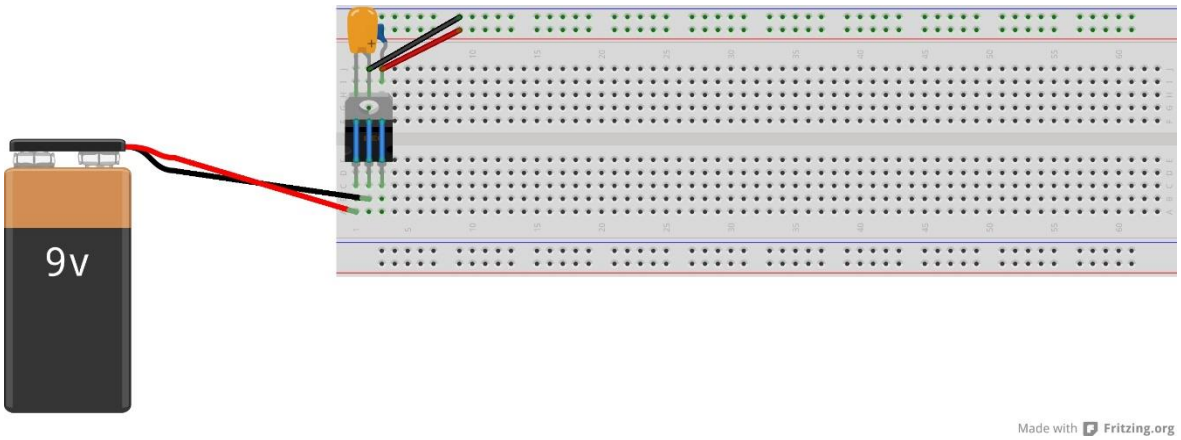
A következő kísérletekben a 74xx IC család tagjait fogjuk használni, hogy megértsük, melyik mit csinál, mindegyiket áramkörbe fogjuk építeni. Beszabályozott áramforrást használjunk, ami pont 5 voltos! Mivel a lábak nagyon érzékenyek, ne érintsük hozzá az áramforrás kivezetéseit, mert tönkremegy az IC. A két pólust mindig kapcsolóval vagy nyomógombbal és ellenállással együtt kötjük be, nem csak szimplán, közvetlenül a lábra! Ha több logikai kapu kimenetét akarjuk együtt használni, kössünk mögéjük diódát is, hogy megvédjük őket egymástól, mivel a dióda visszafelé nem engedi át az áramot!

5.1.4.1. LM7805 feszültség stabilizátor működése

Mivel mindegyik IC-nek stabil 5 voltra van szüksége, ezért egy feszültségszabályozóval és kondenzátorokkal együtt használjuk. A példákban 9 V tápfeszültséget alkalmazunk, így a következő alkatrészekre lesz szükség: 9 V elem és tok, LM7805 feszültség stabilizátor, 0,33uF elektrolit és 0,1uF kerámia kondenzátor. Ezeket a következő módon kötjük össze:



5.1.4.1. ábra: LM7805 feszültség stabilizátor bekötésének kapcsolási rajza

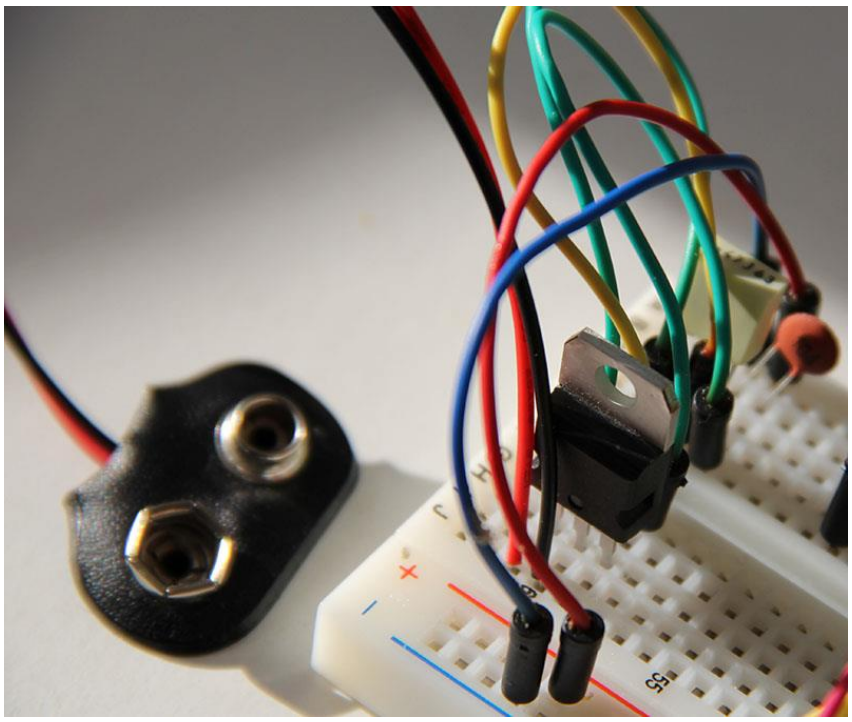


5.1.4.2. ábra: LM7805 feszültség stabilizátor bekötése próbapanelen

 © Harsányi Réka, Juhász Márton András

Az egyik lábán beérkező magasabb feszültséget lecsökkenti és a másik lábán már az alacsonyabb feszültséget adja tovább. A középső a közös negatív láb. A bemeneti és a kimeneti feszültség közötti különbség, ami esetünkben (9 – 5 =) 4 volt, hővé alakul, ezért a tokozás hátulja fémből készült, melynek célja a hő leadása. A tetején egy lyuk található, ezért ha felcsavarjuk egy nagyobb alumínium darabra, jelentősen megnövelhető a hőleadás hatékonysága. Ebben az esetben erre nem lesz szükség.

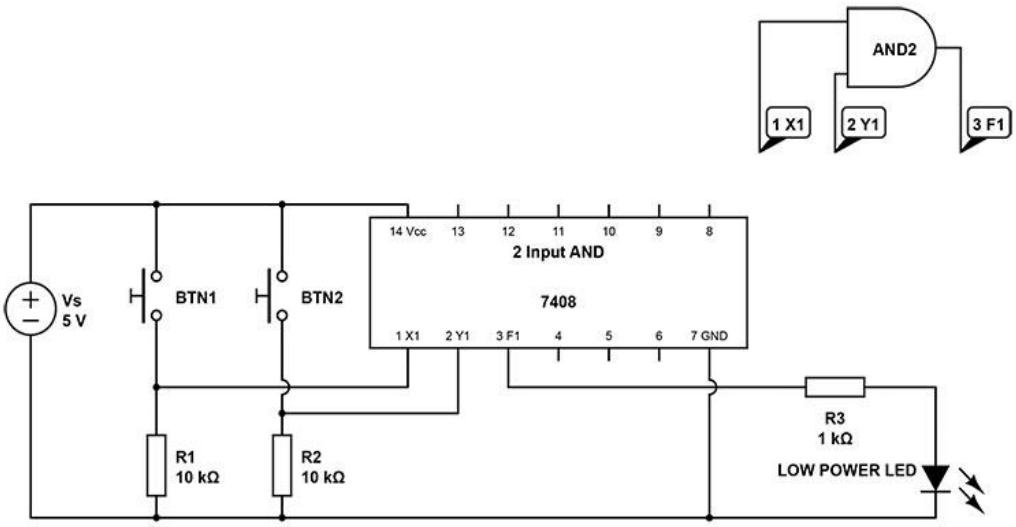
Minden logikai kaput az így kapott 5 V feszültségről üzemeltetünk. A logikai kapuk kapcsolási rajzában ezek az alkatrészek már nem lesznek külön feltüntetve, csak az 5 V áramforrás.



5.1.4.3. ábra: LM7805 feszültség stabilizátor és kondenzátorok.
(Minden további áramkörben szükség lesz ezekre az alkatrészekre.)

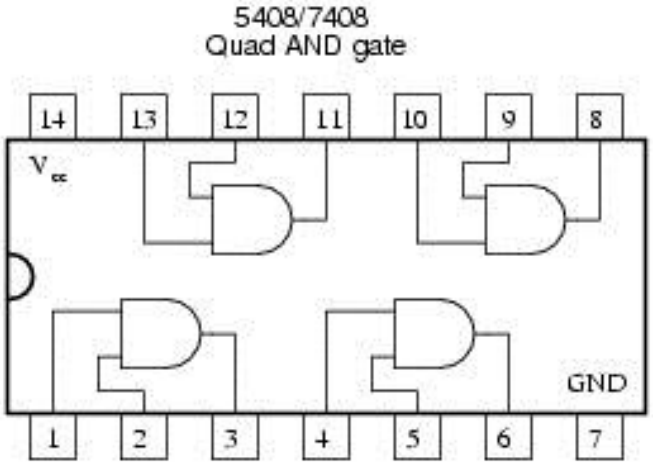
5.1.4.2. 7408 AND kapu

Szükség lesz az áramkör megépítéséhez 1 db 74HC00 chipre, 1 amperes alacsony fogyasztású LED-re, 2 db SPST mikro nyomógombra, 2 db 10 K és 1 db 1 K ellenállásra. A feszültség stabilizátort már bekötöttük a kondenzátorokkal, az így kapott 5 voltról üzemeltetjük az áramkört. Először az AND IC-t építsük be a kapcsolási rajz szerint. Az IC irányát a végén lévő kis bemélyedésből tudjuk megállapítani.



5.1.4.4. ábra: AND kapu bekötésének kapcsolási rajza: az X1 és Y1 a bemenetei, ezek kimenete az F1 az ábrán, jobb oldalt felül külön kiemelve látjuk a kapu rajzjelét.

A 14. lábra kell kötni az áramforrás pozitív pólusát, a 7. lábra pedig a negatívat. A 74xx család többi tagjára is igaz ez. Ez a típusú IC négy darab AND kaput tartalmaz, mindegyiknek két logikai bemenete és egy kimenete van.

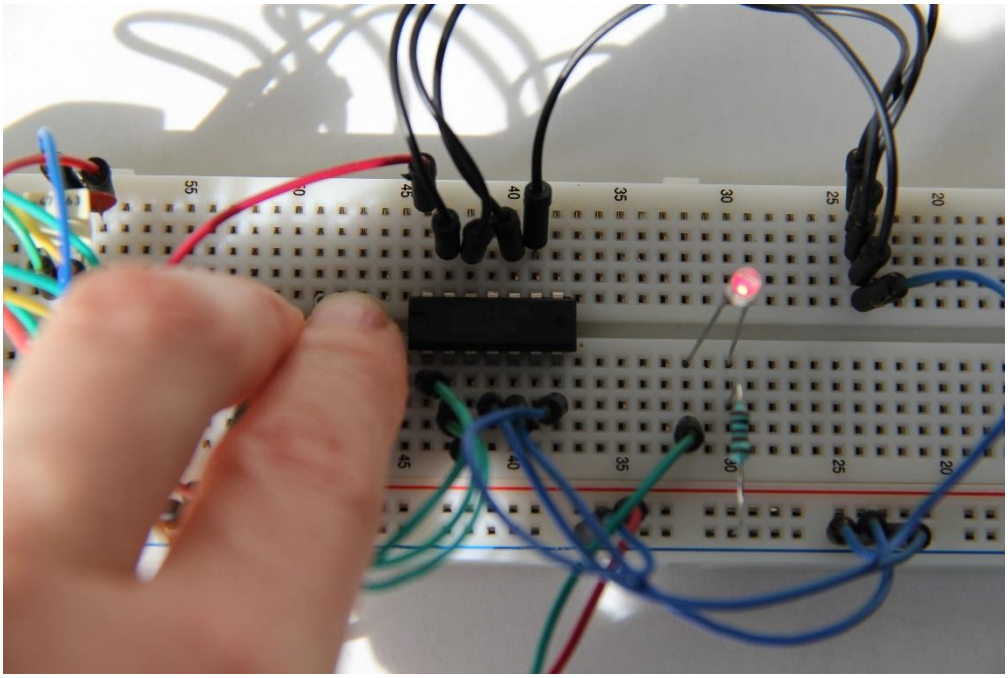


5.1.4.5. ábra: Kapuk és kivezetéseik a 7408 AND IC-ben

 © Harsányi Réka, Juhász Márton András

Az áramkörben alapesetben a két bemenet nem kap feszültséget, ilyenkor a kimenete is nulla, ezért a LED nem világít. Ha csak az egyik gombot nyomjuk le, a helyzet nem változik, mert ilyenkor csak az egyik bemenetre érkezik jel. Az tapasztalható, hogy csak akkor világít a LED, ha a kapu az első ÉS a második bemenetén is áramot kap, tehát, ha mindkét gombot lenyomjuk, ilyenkor a kapu kimenete átvált pozitívra.

Az igazságtáblát a „Logikai kapuk típusai, rajzjelei és igazságtáblázatai” című kép jobb oldalán találjuk mindegyik kapuhoz.

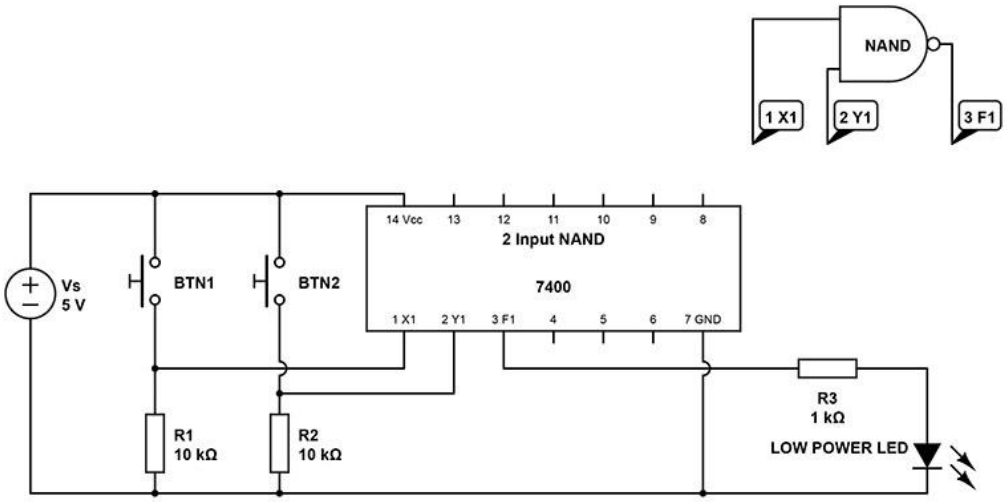


5.1.4.6. ábra: AND kapu működése

A most nem használt kapuk bemeneteit az áramforrás negatív oldalára köthetjük a 5.1.4.6. számú ábra szerint. Az ellenállás a LED pozitív vagy a negatív kivezetésére is köthető.

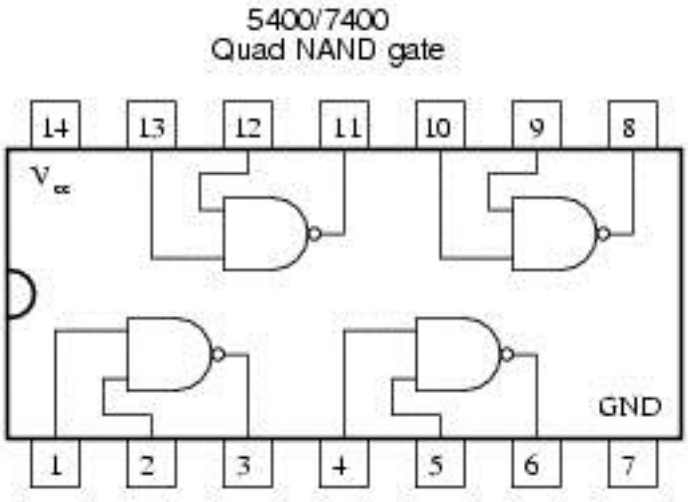
5.1.4.3. 7400 NAND kapu

A következő kísérlethez csak ki kell cserélni az IC-t a 74HC00-ra, mivel a lábkiosztása a NAND kapunak ugyanolyan, mint az AND-nek. Eközben ne legyen áram alatt!



5.1.4.7. ábra: NAND kapu bekötésének kapcsolási rajza: az X1 és Y1 a bemenetei, ezek kimenete az F1 az ábrán, jobb oldalt felül külön kiemelve látjuk a kapu rajzjelét.

Ez a típusú IC négy darab NAND kaput tartalmaz, mindegyiknek két logikai bemenete van és egy kimenete.



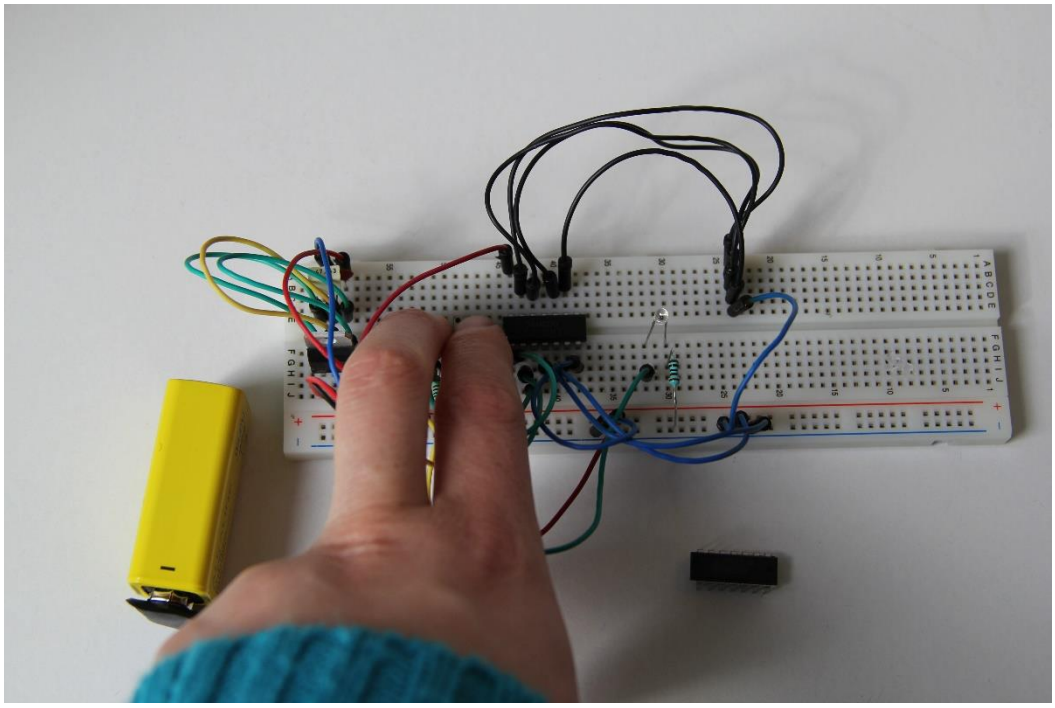
5.1.4.8. ábra: Kapuk és kivezetéseik a 7400 NAND IC-ben

 © Harsányi Réka, Juhász Márton András

Ha áram alá helyezzük a NAND kaput, világítani kezd a LED. Az IC első két lába a két logikai bemenet, a harmadik a kimenet, ezen megy a feszültség az 1 K ellenálláson át az alacsony fogyasztású LED-hez. Alaphelyzetben a nyomógombok megszakítják az áramkört és minkét logikai bemenet feszültsége nulla. Két nyomógomb közül előbb az egyiket, utána a másikat nyomjuk meg. Mindkét esetben tovább világít a LED, mivel az egyik bemenetére érkezett csak feszültség. Ha mindkét gombot egyszerre nyomjuk meg, azaz mindkét bemenetre érkezik feszültség, akkor kialszik a fénye, azaz a NAND kapu kimenete nulla lesz.

A NAND a Not AND rövidítése = Negált ÉS kapu. Ennek logikai kimenete alapesetben pozitív, de akkor NEM, ha az első ÉS a második bemenete is pozitív, tehát ilyenkor a kimenete negatívra vált, azaz nem engedi át az áramot. Ha csak az egyik vagy a másik bemenet pozitív, akkor nem teljesül a feltétel, ilyenkor a kimenet marad alapesetben.

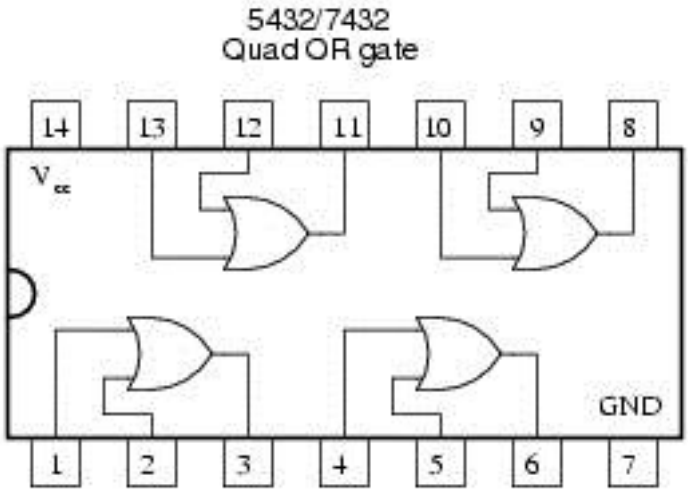
Az AND kapu kimeneti értékeivel ellentétesen működik a NAND kapu.



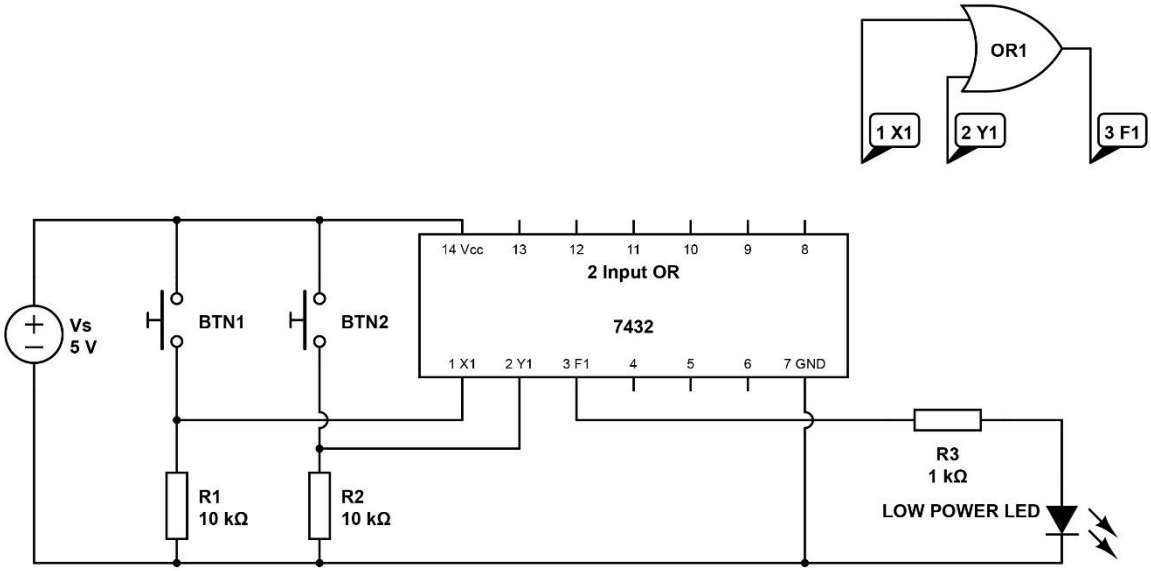
5.1.4.9. ábra: NAND kapu működése

5.1.4.4. 7432 OR kapu

A 74HC32 IC négy OR = Megengedő VAGY kaput tartalmaz. Most erre cseréljük ki az előző áramkörben a chipet!

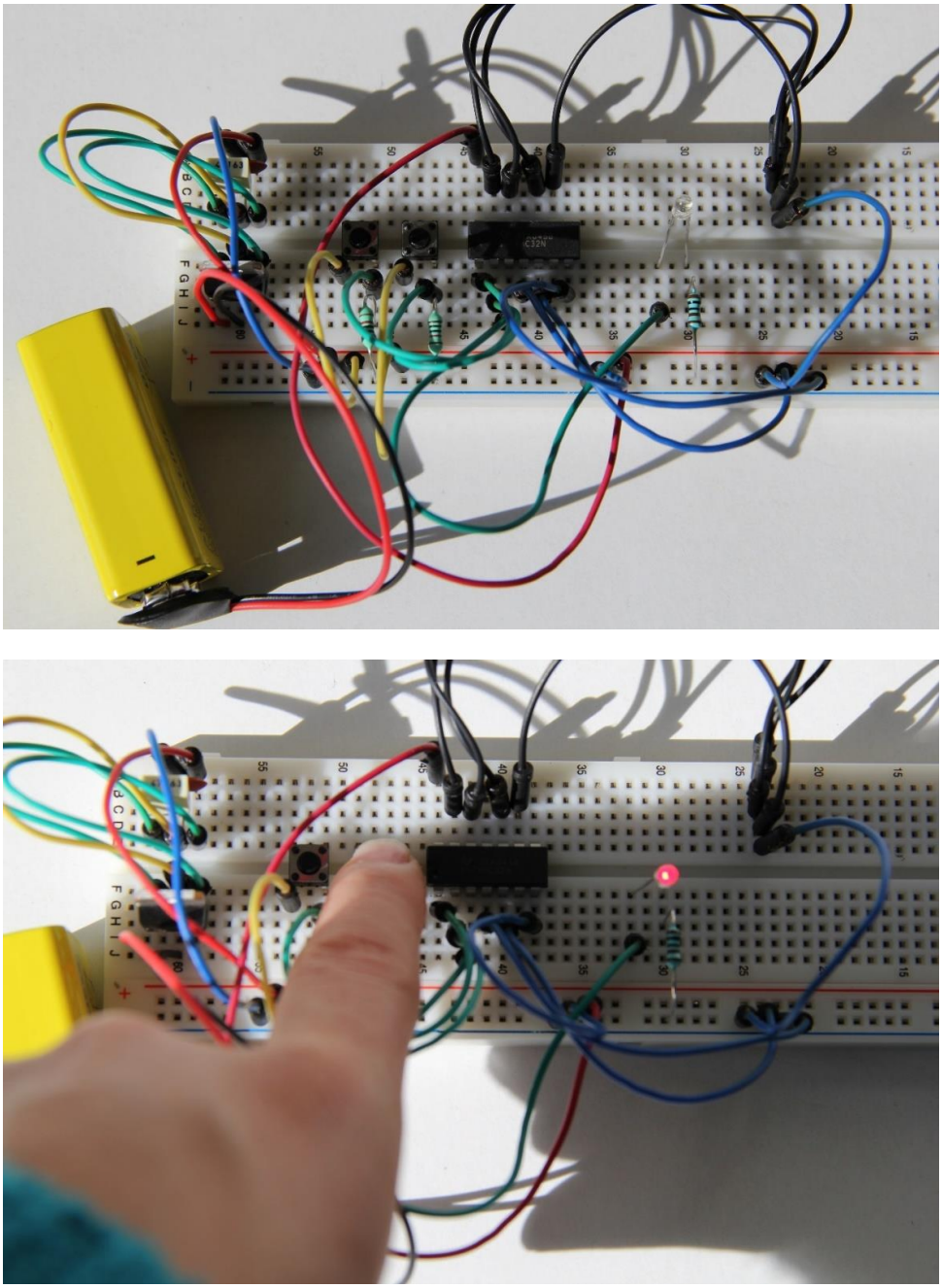


5.1.4.10. ábra: Kapuk és kivezetéseik a 7432 OR IC-ben



5.1.4.11. ábra: OR kapu bekötésének kapcsolási rajza: az X1 és Y1 a bemenetei, ezek kimenete az F1 az ábrán, jobb oldalon felül külön kiemelve látjuk a kapu rajzjelét.

 © Harsányi Réka, Juhász Márton András

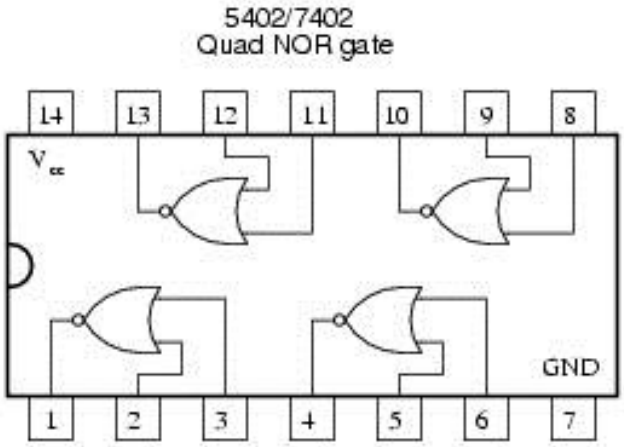


5.1.4.12. ábra: OR kapu működése

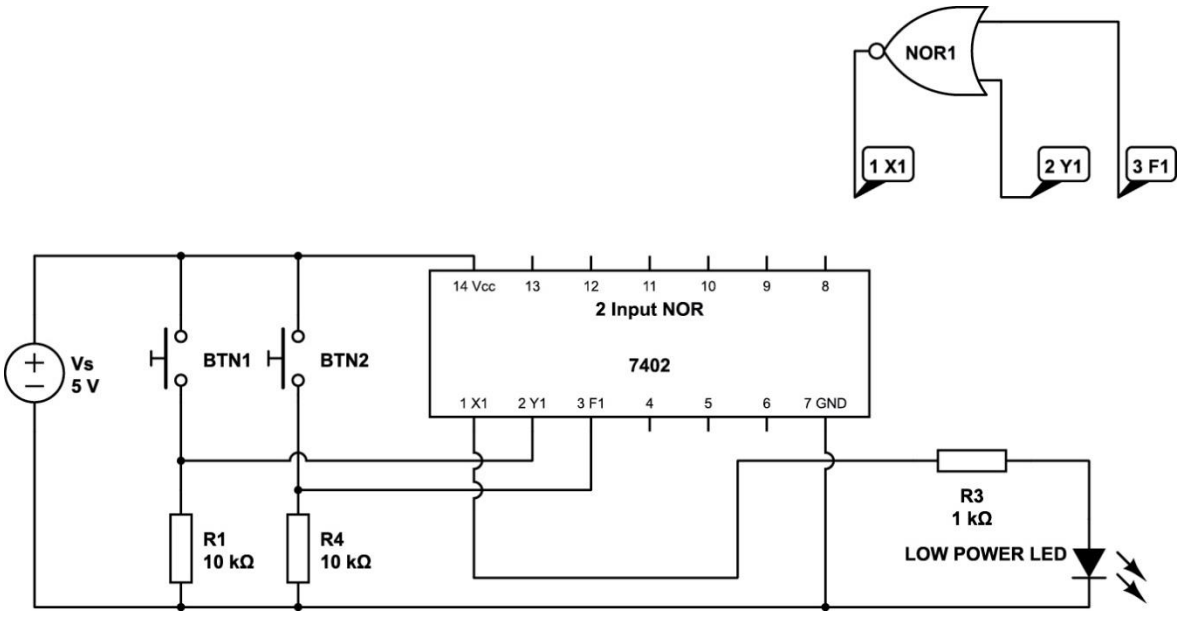
Az OR, azaz a Megengedő VAGY kapu működését tekintve, ha negatív jelet kap a két bemenetére, akkor a kimenete negatív. Alapesetben a gombok megnyomása nélkül nem világít a LED. Ha egyik vagy másik, illetve ha mindkettő bemeneti lábra pozitív jelet kap, akkor a kimenete pozitívra vált. Ezt azt jelenti, hogy ha az egyik, másik, vagy mindkettő gombot lenyomjuk, a LED világítani kezd.

5.1.4.5. 7402 NOR kapu

A NOR kapuval végzett kísérlethez a nyomógombok és a LED bekötését fel kell cserélni a kapcsolási rajz szerint, mivel ennél a kapunál a kimenet kivezetése az első láb, a bemeneteké pedig a második és a harmadik láb.



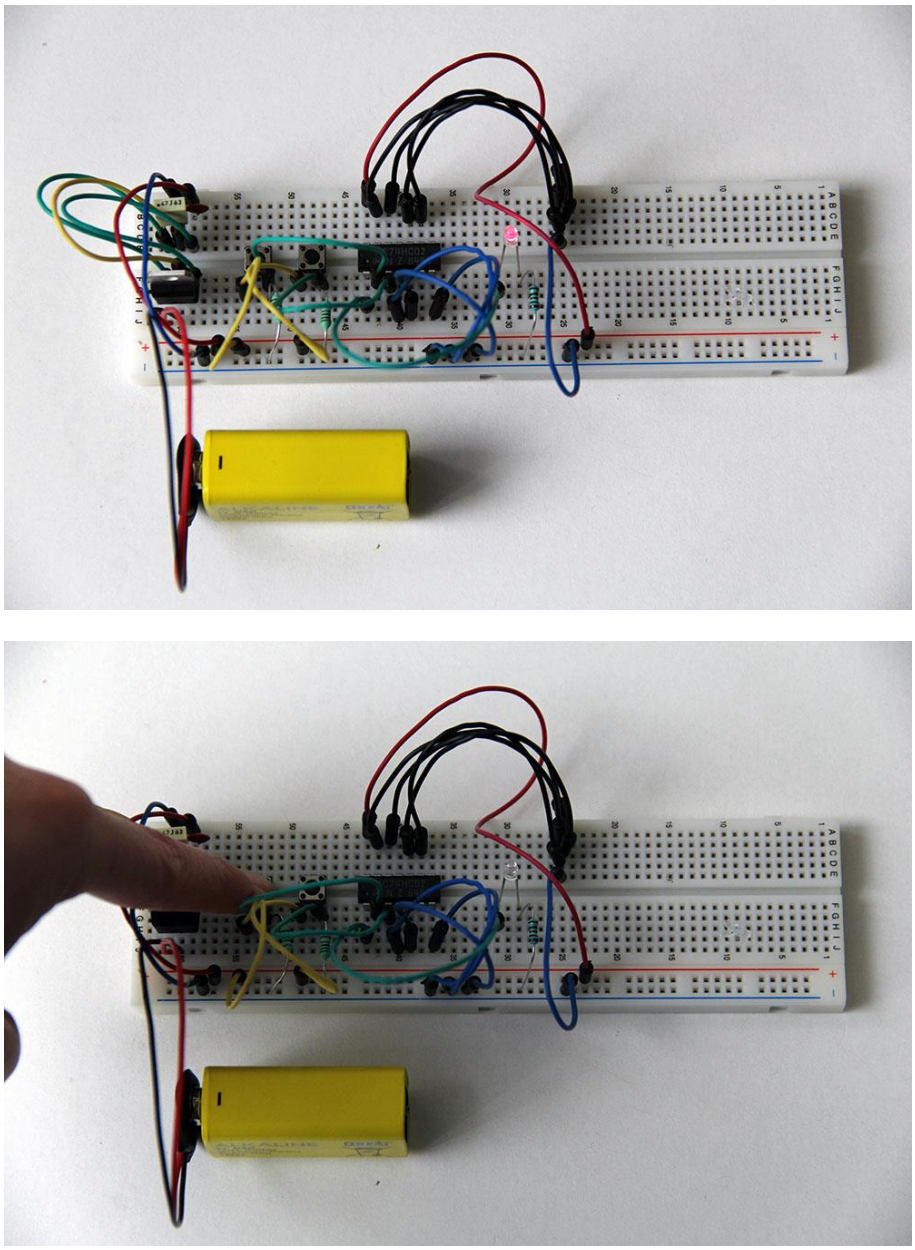
5.1.4.13. ábra: Kapuk és kivezetéseik a 7402 NOR IC-ben, 4 kaput tartalmaz.



5.1.4.14. ábra: Kapcsolási rajz NOR kapuhoz

 © Harsányi Réka, Juhász Márton András

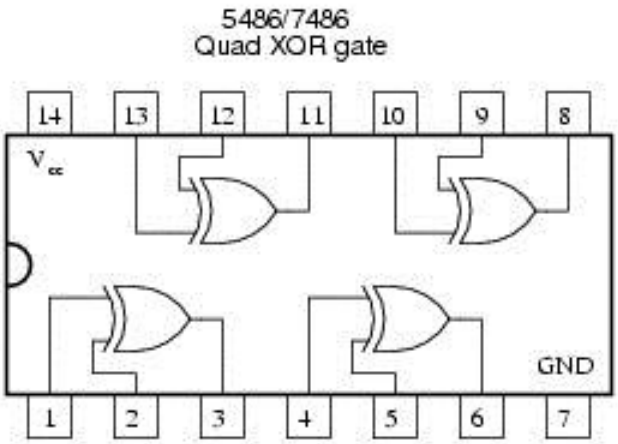
A NOR = Negált VAGY kapu működését tekintve, alapesetben negatív jelet kap a két bemenetére, akkor a kimenete pozitív, ezért világít a LED. Ha egyik vagy másik, illetve ha mindkettő bemeneti lábra pozitív jelet kap a gombok megnyomása által, akkor a kimenete negatívra vált, így elalszik a LED. A NOR kapu kimeneti értékei az OR kapuénak ellentétei.



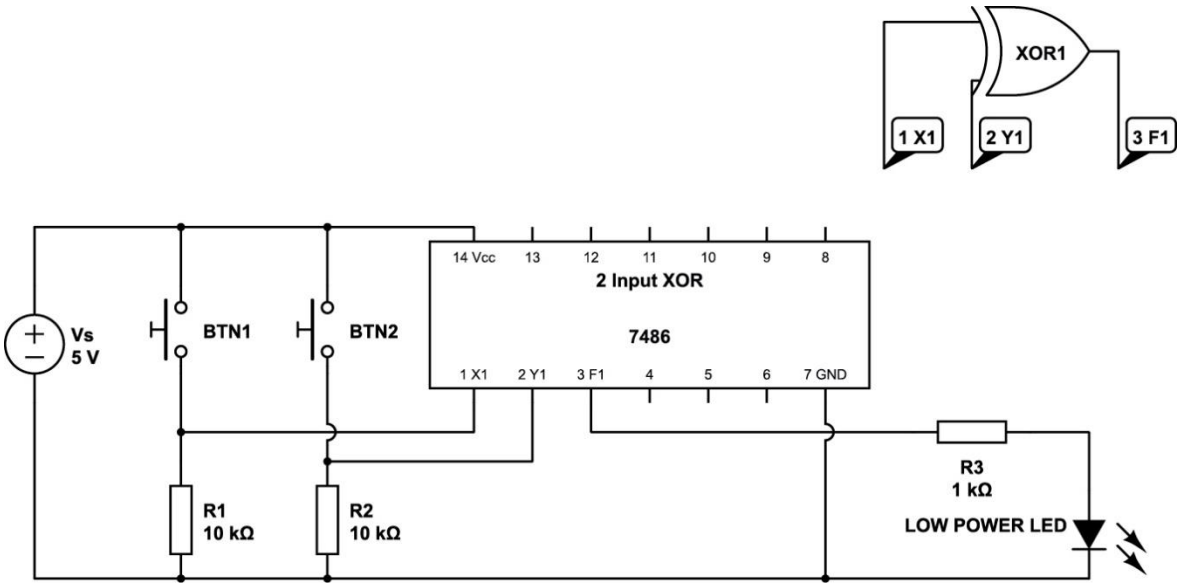
5.1.4.15. ábra: NOR kapu működése

5.1.4.6. 7486 XOR kapu

A XOR kapu esetén a nyomógombokat az első két lábra, a LED-et a harmadik lábra kötjük, éppúgy, mint korábban az OR kapu használata esetén.

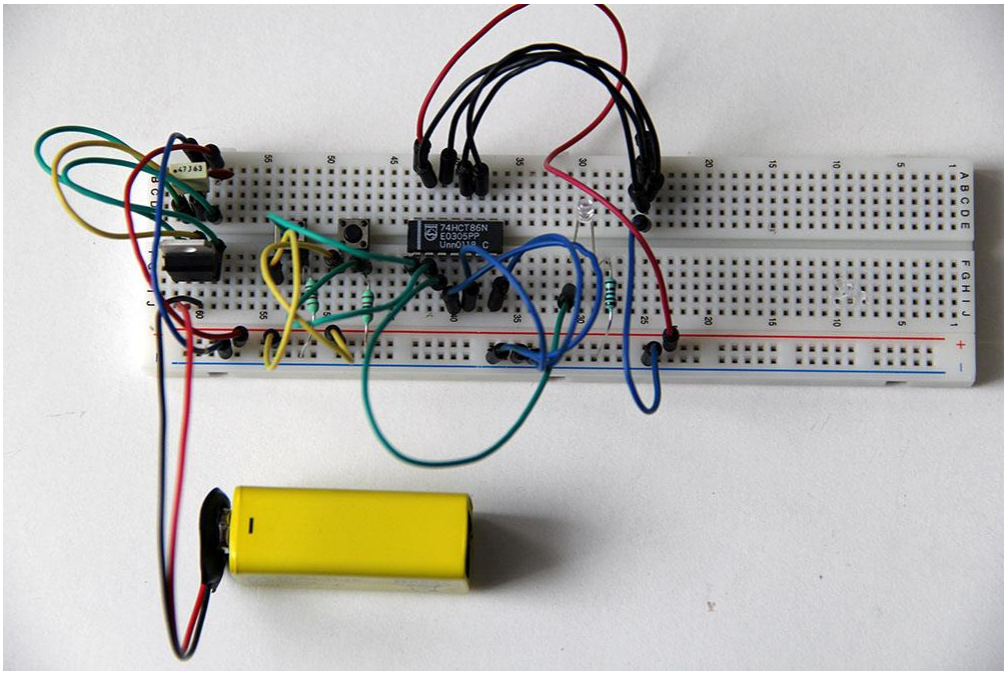


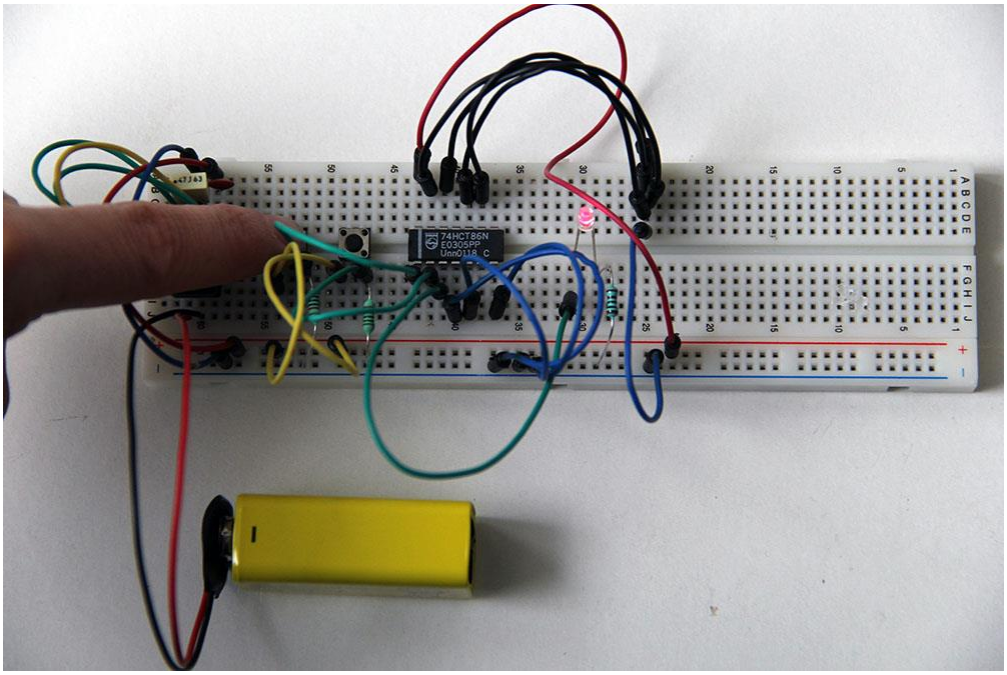
5.1.4.16. ábra: Kapuk és kivezetéseik a 7486 XOR IC-ben

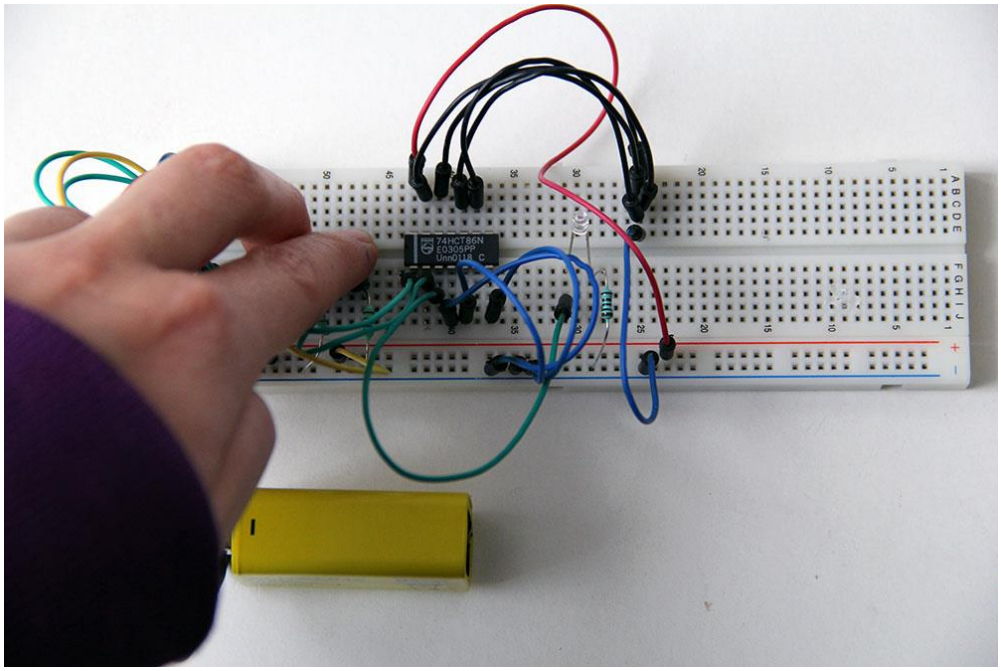


5.1.4.17. ábra: Kapcsolási rajz XOR kapuhoz

 © Harsányi Réka, Juhász Márton András


74HCT86N
E0305PP
Unn0118 C


74HCT86N
E0305PP
Unn0118 C

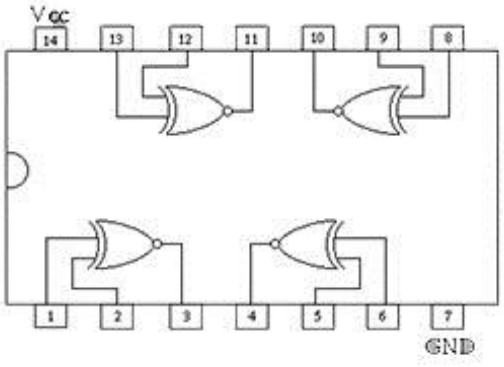


5.1.4.18. ábra: XOR kapu működése

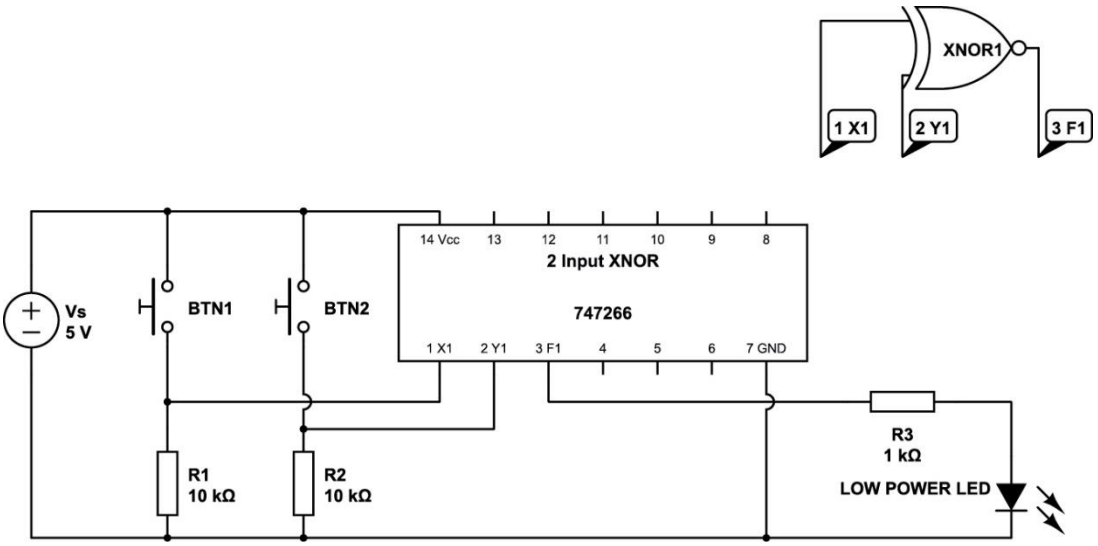
A XOR = Kizáró VAGY kapu működését tekintve, ha a két bemenetén azonos jelet kap, akkor a kimenete nem változik, negatív marad, de ha az egyik bemeneten pozitív jel érkezik, akkor a kimenete is pozitívra vált. Ha mindkét gomb alaphelyzetben áll és nem enged át feszültséget, vagy mindkét gombot lenyomjuk, hogy átengedjék az áramot a két bemenethez, akkor a kimeneten nem változik, a LED nem világít. Viszont ha az egyik gombot lenyomjuk, a kapu kimenetén keresztül áram alá kerül a LED és világítani kezd.

5.1.4.7. 747266 XNOR kapu

A XNOR = Negált Kizáró VAGY kapu működését tekintve pont ellentétes az XOR kapuéval. Kimenete alapesetben pozitív, ami nem változik, ha azonos jelet kap a bemenetére, de ha az egyiken pozitív jelet kap, akkor kimenete negatívra vált. Alapesetben világít a LED, és akkor is, ha mindkét gombot lenyomjuk, de ha csak az egyiket, akkor kialszik a LED fénye.



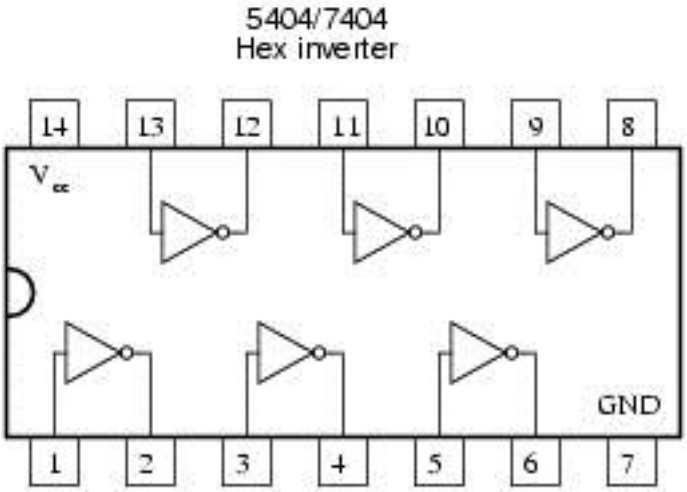
5.1.4.19. ábra: Kapuk és kivezetéseik a 747266 XNOR IC-ben



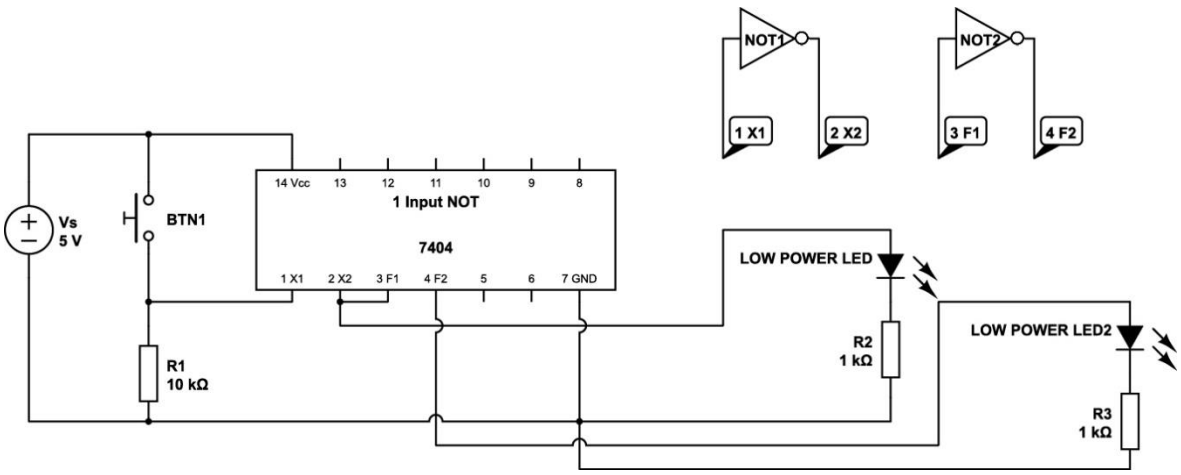
5.1.4.20. ábra: Kapcsolási rajz XNOR kapuhoz

5.1.4.8. 7404 NOT kapu

Építsük be a 7404 logikai kaput két LED-del és egy nyomógombbal a kapcsolási rajz alapján! Ez az IC hat kaput tartalmaz és az eddigiekkel ellentétben egy bemenettel és egy kimenettel rendelkeznek a kapuk.

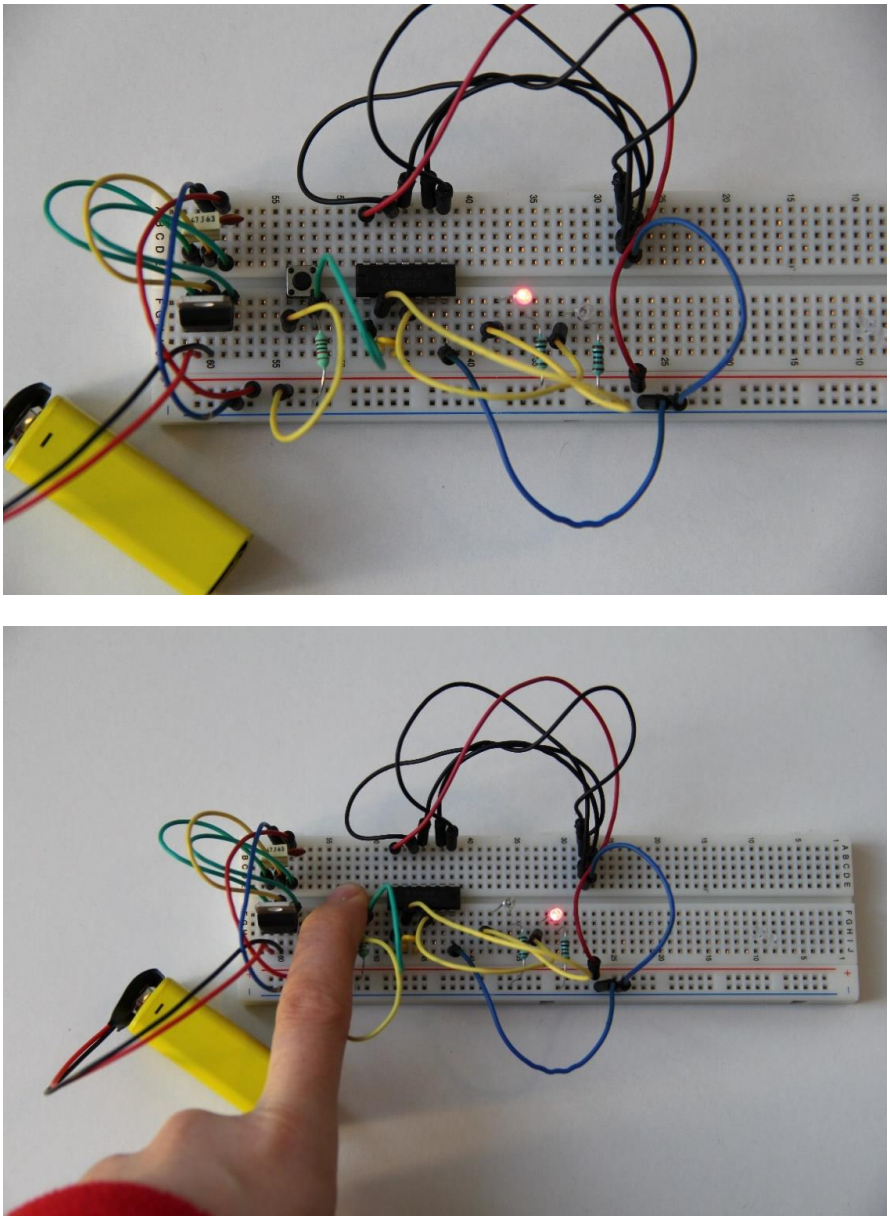


5.1.4.21 ábra: Kapuk és kivezetéseik a 7404 NOT IC-ben



5.1.4.22. ábra: Kapcsolási rajz a NOT kapuhoz

A NOT = Negálás kapu működését tekintve, ha negatív jelet kap a bemenetére, akkor a kimenete pozitívra vált, illetve fordítva. Inverternek is nevezik.



5.1.4.23. ábra: NOT kapu működése

Az áramkörben a két NOT kapu kimenetére egy-egy alacsony fogyasztású LED-et kötöttünk. Alapállapotban a chip első lábának feszültsége nulla, ennek ellentéte lesz a NOT kapu kimenetén, azaz egy, ami jelen esetben 5 V feszültségnek felel meg, így a kettes lábra kötött LED világít. Mivel ez van bekötve a következő kapu bemenetére, a harmadik lábra, ennek ellentétje, azaz 0 V érkezik meg a kimeneti negyedik lábon keresztül a második LED-hez, tehát az alapesetben nem világít. Ha lenyomjuk a gombot, akkor az első NOT kapu feszültséget kap, a kimenete 0 volt lesz, így az első LED elalszik, a második kapuhoz is ez érkezik, így annak kimenete 5 V lesz, ezért a második LED felgyullad.

6. Bevezetés az Arduino világába

6.1. A programozás alapja

6.1.1. Definíciók

„*Programozás*: számítógép-algoritmusok és adatszerkezetek megtervezése és megvalósításuk valamely programozási nyelven.

Algoritmus: Valamely probléma megoldására alkalmas véges számú cselekvéssor, amelyet véges számú alkalommal mechanikusan megismételve a probléma megoldását kapjuk. Azért véges számú az ismétlés, mert nem tarthat végtelen ideig a folyamat.

Adatszerkezet: az adatelemek egy olyan véges halmaza, amelyben az adatelemek között szerkezeti összefüggések vannak. Az adatelem az a legkisebb adategység, amelyre hivatkozni lehet.

Programozási nyelv: a számítástechnikában használt olyan, az ember által olvasható és értelmezhető utasítások sorozata, amivel közvetlenül, vagy közvetve (például: gépi kódra fordítás után) közölhetjük a számítógéppel egy adott feladat elvégzésének módját.

Program = algoritmus + adatszerkezet.

Kódolás: az algoritmus és az adatszerkezet megvalósítása valamely programnyelven.

Specifikáció: a program feladatának, bemeneti és kimeneti adatainak megadása.

Tervezés: a feladat, azaz a specifikáció ismeretében választjuk ki a programnyelvet (senki sem szerkesztene weblapot C-ben…), az algoritmusokat és az adatszerkezeteket.

Kódolás: A kiválasztott nyelven megvalósítjuk a megtervezett algoritmusokat és adatszerkezeteket. Ez részben egyszerre történik a tervezéssel, azaz bizonyos algoritmikus, ill. adatszerkezeti kérdésekben a kódolás közben döntünk.

Tesztelés, hibajavítás: Senki sem ír hibátlan kódot, azaz mindig van mit javítani. A hibáknak két csoportja van: szintaktikai (formai) és szemantikai (tartalmi) hibák.

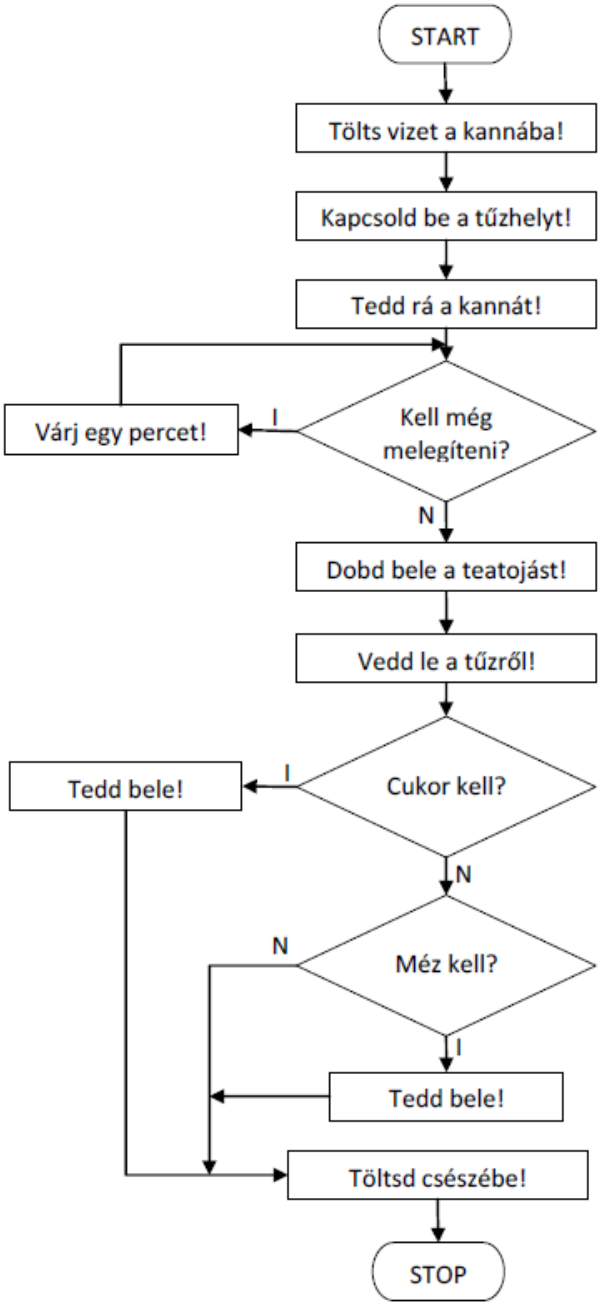
Szintaktikai hiba a formai szabályok megsértése. Valamit rosszul gépelünk, lemarad egy zárójel, stb. Bár a kezdő programozónak sok bosszúságot okoznak a szintaktikai hibák, ezekkel könnyű elbánni, mert a fordítóprogram jelzi, hol találhatók.

Szemantikai hiba esetén a fordítóprogram nem talál hibát, mert a program formailag jó, csak nem azt csinálja, amit szeretnénk, hanem azt, amire utasítottuk.”

forrás: Pohl László:A programozás alapjai (2010)

6.1.2. Tervezés

A programunk megtervezéséhez kétféleképpen kezdhetünk hozzá. Létre hozhatunk egy grafikus folyamatábrát, vagy leírhatjuk pszeudokóddal. A *folyamatábra* (más néven végrehajtási gráf, vezérlésfolyam-gráf, blokkdiagram) blokkokat és végrehajtásuk utáni irányokat ábrázolja, lehetnek benne elágazások is, ilyenkor attól függően folytatódik a kód, hogy a blokkban szereplő állítás igaz vagy hamis-e. A *pszeudokód* egy az algoritmusok leírására használt mesterséges nyelv, ami a programozási nyelv felépítését követi azáltal, hogy egy-egy szót felletet meg az egyes lépéseknek. Vegyünk egy konkrét példát, írjuk le a teafőzés folyamatát a fent említett két módszerrel!



Teafőzés folyamatábrával

forrás: Pohl László:A programozás alapjai (2010)

 © Harsányi Réka, Juhász Márton András

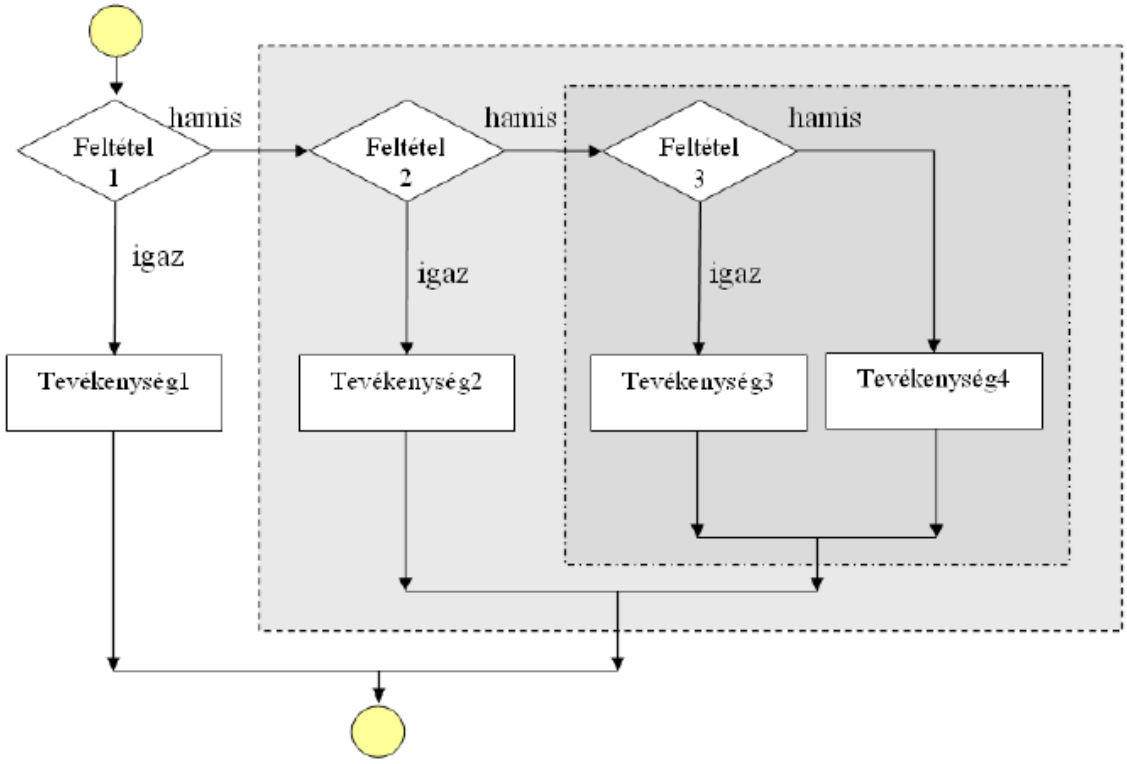
```
„Főzz teát!” parancs:
        Tölts vizet a kannába!
        Kapcsold be a tűzhelyet!
        Tedd rá a kannát!
        Amíg nem forr
                Várj egy percet!
        Dobd bele a teatojást!
        Vedd le a tűzről!
        Ha kell cukor
                Tedd bele!
        Egyébként
                Ha kell méz
                        Tedd bele!
        Töltsd csészébe!
VÉGE.
```

Teafőzés pszeudo kóddal

forrás: Pohl László:A programozás alapjai (2010)

6.1.3. Vezérlési szerkezetek

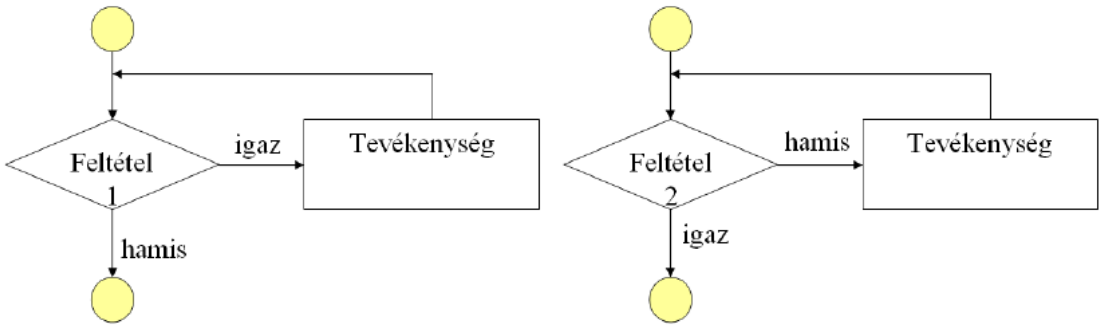
A jól felépített program jól átlátható részegységekből áll. Az egyes részek kapcsolata előre meghatározott sémákon alapul, a leggyakrabban használt ezek közül a *sorozat* (szekvencia), az *elágazás* (szelekció) és a *ciklus* (iteráció). *Strukturált programnak* nevezzük az olyan programot, amely csak ezt a három alapvető tevékenységszerkezetet használja. A sorozat azt jelenti, hogy egymás után végzünk el feladatokat, amelyeknek nem kell egymással összefüggésben állniuk. Az elágazások egy feltételen alapulnak, a program megvizsgálja, hogy a blokkban szereplő állítás igaz vagy hamis, és az eredménytől függően fut tovább. Több ágú logikai feltételeket is alkalmazhatunk, de egymásba is ágyazhatjuk őket.



Többágú elágazás, egymásba ágyazott feltételekkel

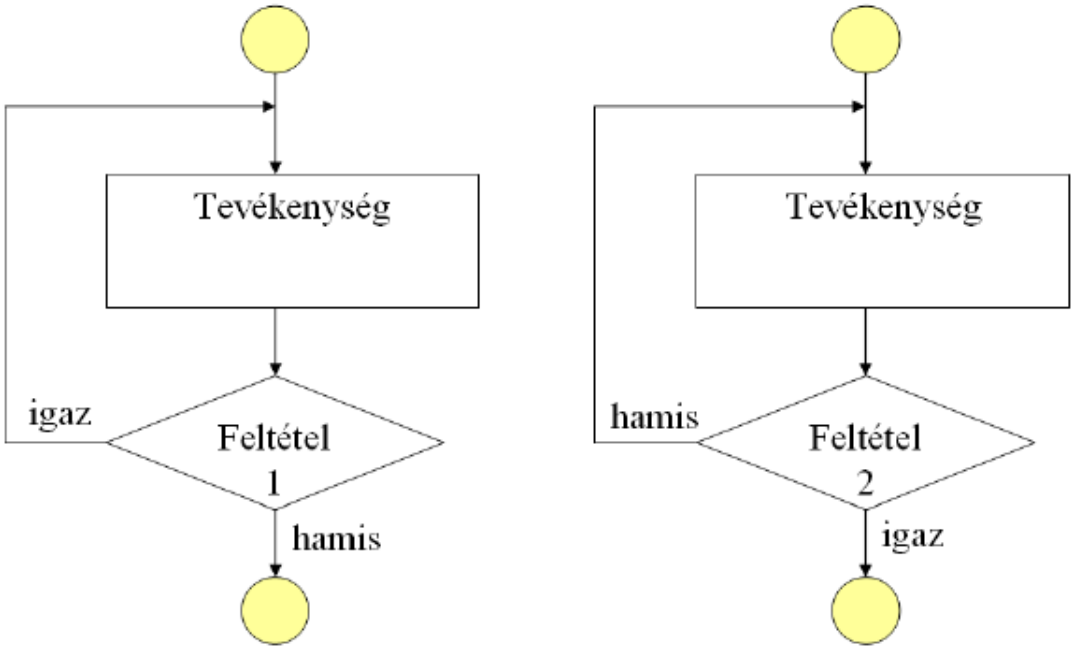
forrás: Simon Gyula:A programozás alapjai (2011)

Az ismétlődő tevékenységek leírására ciklusokat alkalmazunk. Minimum egy feltételt és egy tevékenységet tartalmaznak. Attól függően, hogy a benne szereplő feladat elvégzése előtt vagy után vizsgálja meg a feltételt, elöltesztelő vagy hátultesztelő ciklusnak nevezzük. Az elöltesztelő először megvizsgálja a feltételt, ami ha teljesül lefut a kód. A hátultesztelőnél mindenképp lefut a parancssor és csak utána vizsgálja meg, hogy a feltétel igaz vagy hamis-e és ennek függvényében megismétli, ha kell, majd ismét leellenőrzi a feltételt. A feltételnek is két típusa lehet, bennmaradási vagy kilépési feltétel. Ezek kombinációjával négy típusú ciklust hozhatunk létre (Elöltesztelő ciklus bennmaradási feltétellel; Elöltesztelő ciklus kilépési feltétellel; Hátultesztelő ciklus bennmaradási feltétellel; Hátultesztelő ciklus kilépési feltétellel). A *bennmaradási feltétel* azt jelenti, hogy amíg a feltétel teljesül addig folyamatosan fut a ciklus. A *kilépési feltétel* pedig azt, hogy addig fut a ciklus, amíg a feltétel hamissá nem válik. Folyamatábrával illusztrálva:



Elöltesztelő ciklus bennmaradási és kilépési feltétellel

forrás: Simon Gyula:A programozás alapjai (2011)



Hátultesztelő ciklus bennmaradási és kilépési feltétellel

forrás: Simon Gyula:A programozás alapjai (2011)

6.1.4. Adatszerkezetek

Adatszerkezetnek nevezzük azt a struktúrát, amit az adatokon belül, több kisebb adat egymáshoz rendezésével, egymással kialakított viszonyrendszer által hozunk létre. Adattípusokon végzünk el műveleteket. Az adattípus lehet *statikus*, állandó (konstans) vagy *dinamikus*, változó. Az állandók használata esetén a bennük tárolt adatok nem változnak. Ezzel szemben a változókban igen, például egy ciklus parancssora irányulhat a változóban tárolt érték folyamatos változtatására. Attól függően, hogy hol hozzuk létre az állandót vagy a változót beszélhetünk *globális* vagy *lokális* adattípusról. A különbség köztük, hogy a globális változót bárhol meghívhatjuk a kódban, de a lokálisat csak azon a cikluson belül tudjuk alkalmazni ahol létrehoztuk. Számtalan fajtája létezik az adattípusoknak attól függően, hogy mekkora tartományban, milyen előjellel tárol értéket, illetve egész számot vagy tizedes törtet, esetleg karaktert, beszélhetünk *integer*-ről, *float*-ról, *long*-ról, *char*-ról stb. A karaktereket a számítógép adattá konvertálva kezeli, minden jelnek megfeleltet egy értéket, ez az ASCII kódolás. *Típusmódosítók*kal átalakíthatjuk a változónk tulajdonságait a program bármely részében. A *tömb* egy speciális adattároló, amiben több értéket kezelhetünk, azokat tetszőleges sorrendben is kiolvashatjuk. A későbbiekben példákon keresztül alkalmazzuk az adattípusokat. A műveleteket *operátor*okkal (műveleti jelekkel) hajtjuk végre. Ide tartoznak az alapvető *algebrai,* (+,-,*,/) *relációs* (<, >, <=, >=) és *logikai* (&&, ||, !) jelek.

A *szintaxis* a programozási nyelvben azt a szabályrendszert jelenti, amely meghatározza, hogy az adott nyelvben hogyan lehet az utasításokat létrehozni.

A *deklaráció* során meghatározzuk egy elem típusát vagy működését.

Az *inicializálás* a kezdő érték megadását jelenti.

6.1.5. Függvények

A *függvény*ek logikailag összefüggő szekvenciális programszerkezetek kezelésére, paraméterek átadására és megváltoztatására is alkalmasak. Egymással logikailag összefüggő, szekvenciális feladatokat tartalmaznak, végeredményként egy értéket adnak vissza. Ugyanakkor az egyes különálló függvények között nem szükséges kapcsolat megléte, a kód többi részétől független egységet képeznek. A *szubrutin* is hasonló folyamatot jelent, az egy olyan különálló programrész, amit beilleszthető, meghívható a főprogramban. Több szubrutint is alkalmazhatunk, illetve szubrutinokon belül is meghívhatjuk őket.

6.2. Mi az Arduino?

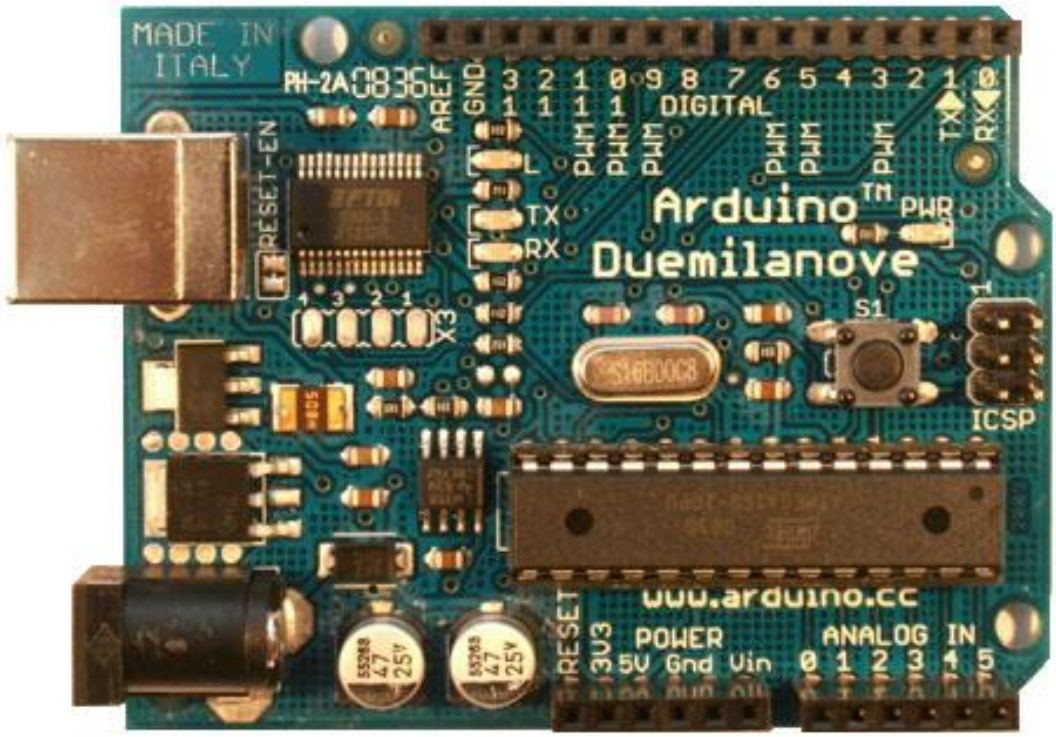
Egyrészt egy mikrokontroller, másrészt a programozási környezet neve. Több fajtája létezik, mi az órán az Arduino Duemilanove-t használjuk. Az Arduino egy nyílt forrású, gyors prototipizáláshoz ideális mikrokontroller modul. Használata a világos felépítésű elektronikának és jól értelmezhető programnyelvnek köszönhetően könnyen és gyorsan elsajátítható. Hirtelen elterjedése interaktív művészeti alkotásokban, hobbi alkalmazásokban és gyors vezérléstechnikai megoldások kivitelezésében egy jól kiépült nemzetközi tudásbázist hozott létre, hozzáférhetővé téve az elektronika egyszerű használatát minden érdeklődő számára. Sokrétű ki- és bemenetein keresztül fel tudja venni a kapcsolatot bármilyen jellegű szenzorral és programjának megfelelő válaszadásra képes, meghajtva lámpákat, motorokat, kijelzőket. A mikrokontrollert Arduino nyelven lehet programozni, ami a Wiring nyelvén alapszik, felhasználói felületét pedig a Processing alapja nyújtja. Az Arduino egyedülálló áramkörként is működtethető, vagy összekapcsolható számítógéppel, amin Flash, Processing, MaxMSP, GarageBand, illetve egyéb soros kommunikációra képes programokkal képes együttműködni. A modulokat készen és KIT-ben is meg lehet vásárolni, a programok ingyen letölthetők honlapukról. A kapcsolások és a nyomtatott áramköri tervek szintén hozzáférhetők és magáncélra szabadon felhasználhatók. Az Arduino a 2006-os Ars Electronica digitális közösségek szekciójában is elismerésben részesült. A csapat: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, és David Mellis.

A tananyag ezen részét a Massimo Banzi és társai által kidolgozott hivatalos leckék adják, amik Creative Common License-szel rendelkeznek:



CC BY-SA 3.0 - Creative Commons License

6.3. Arduino Duemilanove részei



6.3. ábra: Arduino Duemilanove

A Duemilanove 2009-es modell, ATmega168 típusú vagy ATmega328 típusú mikrokontrollerrel felszerelt verzióban létezik. 14 digitális ki-/bemenete van amiből 6 speciális PWM funkciót is képes ellátni, 6 analóg bemenettel rendelkezik, valamint egy 16 MHz-es osszcillátorral, egy USB, egy hálózati és egy ICSP csatlakozóval rendelkezik. Egy reset (újraindító) gomb is található rajta. Minden alkatrészt tartalmaz, ami a mikrokontroller működéséhez szükséges, így ha csatlakoztatjuk USB kábelünkkel (5V) a számítógéphez, megkezdhetjük használatát, de akár elemet, akkumulátort vagy AC-to-DC adaptert (a nagyfeszültséget átalakítja kisfeszültségű egyenárammá) dughatunk a hálózati csatlakozójába, ezek ajánlott feszültség értéke maximum 12V lehet. A DC a Direct Current rövidítése, ami kisfeszültségű egyenáramot jelent, ezzel fogunk dolgozni.

A Duemilanove adatai:

- mikrokontroller típusa: ATmega 168 vagy ATmega 328
- működtető feszültségérték: 5V

- bementi feszültség – ajánlott értéke: 7-12V
- bemeneti feszültség – határ értéke: 6-20V
- digitális ki-/bemeneti pinek száma: 14, ebből 6 PWM. 5V-on működnek, mindegyik rendelkezik egy belső felhúzó (pull-up) ellenállással, ami egy speciális paranccsal bekapcsolható (alap állapotban ki van kapcsolva), értéke 20-50 kOhm lehet.
- analóg bemeneti pinek száma: 6, ezek 10 bites felbontáson tudnak egy értéket felvenni 0-1023 között.
- az egyes pinekre jutó áramerősség értéke: 40mA
- a 3.3V elnevezésű pinre jutó áramerősség értéke: 50mA
- flashmemória mérete: 16KB (ATmega168), 32KB (ATmega328) amiből 2KB-ot a bootloader foglal el. A flashmemória egy olyan újraprogramozható, adattárolásra alkalmas eszköz, aminek nincs szüksége tápfeszültségre ahhoz, hogy megtartsa a benne tárolandó adatot. A bootloader magyarul rendszer betöltő program, ezzel képes elindulni az eszköz.
- SRAM mérete: 1 KB (ATmega168), 2 KB (ATmega328). Ez egy statikus memória, amiben a tárolt adat a tápfeszültség megszűnéséig marad meg.
- EEPROM mérete: 512 byte (ATmega168), 1 KB (ATmega328). Elektronikusan törölhető és újraírható, azaz programozható tároló memória. Olyan mint a flashmemória.
- Órajel sebessége: 16 MHz. Ez egy olyan jel, amely két vagy több áramkör tevékenységét koordinálja, szinkronizálja. Egy alacsony és egy magas jelszint között osszcillál.

Áramot kezelő pinek:

- Vin: bemeneti feszültség, ha külső áramforrást használunk az üzemeltetéshez.
- Gnd: földelés.
- 5V: értelemszerűen 5V jön belőle.
  3V3: az alaplapra szerelt FTDI chip által termelt 3,3 V feszültség.

Némelyik pin speciális funkciók ellátására is képes, ezeket a későbbiekben használat közben ismerjük meg.

Speciális funkciókkal ellátott pinek:

- digitális 0(RX) és 1(TX): Ezek a pinek össze vannak kötve a FTDI USB-to-TTL nevű soros chip megfelelő részeivel, ezért soros port kommunikációhoz használhatjuk őket. Az RX fogadja, a TX pedig küldi az adatot.

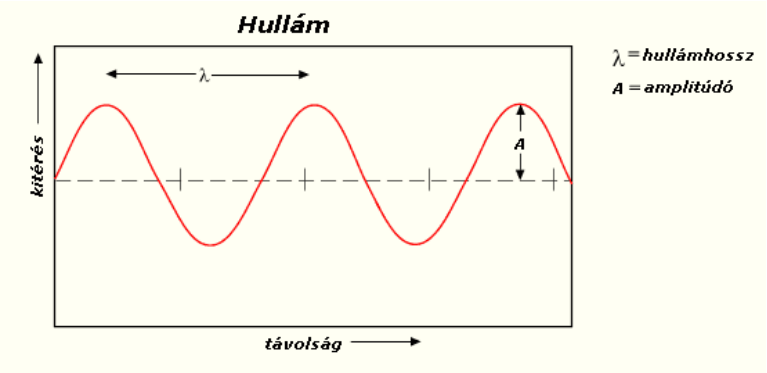
- digitális 2 és 3 pin külső megszakításhoz: a csatlakozón fellépő fel- vagy lefutó él hatására fellépő megszakításkéréshez használható, ez a mikrovezérlő valamelyik bemenetének értékváltozására generálódik. Az *attachInterrupt()* funkcióval használjuk.
- digitális PWM pinek: 3, 5, 6, 9, 10, és 11 (pulse width modulation = impulzus-szélesség moduláció). Az *analogWrite()* paranccsal 8 biten küldhetünk rá adatot.
- SPI pinek: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK), SPI busz (Serial Peripheral Interface = soros periféria illesztő) kommunikációhoz használhatjuk, az *SPI Library* meghívásával.
- LED 13 pin: a beépített LED van ide kivezetve ide.
- Analóg A4 (SDA) és A5 (SCL) pinek: I^2C (TWI) busz kommunikációra alkalmasak, *Wire Library* meghívásával.

Egyéb pinek:

- AREF pin: Alapvetően az analóg pinek 0-5V között mérik a bementi adatot. Ennek felső értéke ezzel a pinnel és az *analogReference()* paranccsal megváltoztatható.
- Reset pin: ha a *shield*et használunk és van rajta reset gomb, akkor használjuk ezt a pint. LOW-ra állítva működik.

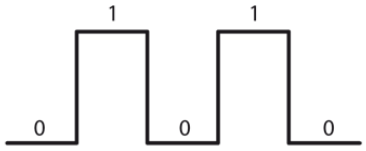
6.3.1. Mi a különbség az analóg és a digitális jel között?

Az analóg jel idő és amplitúdó szerint folyamatosan változó jel, számtalan értéket képes felvenni. Lehet szabályos időközönként, periodikusan változó vagy szabálytalan, azaz a két szélsőérték között bármekkora lehet a pillanatnyi értéke. Az amplitúdó a jel nagyságát határozza meg, a frekvencia pedig az ismétlődések időtartamát, ami a periódusidő vagy a hullámhossz.



6.3.1.1. ábra: Analóg jel, szinusz hullám, szabályos (periodikus) jel

A digitális jel két értéket vehet fel, nullát vagy egyet. Arduino esetében ez azt jelenti, hogy ha bemenetére 5 V feszültség érkezik, az érték 1 lesz, ha nem érkezik semmi, akkor 0. Kimenetként használva ugyanez igaz, ha 1 értékkel 5 V feszültséget küldünk. Az Arduinonak vannak speciális PWM pinjei, erről később lesz szó.



6.3.1.2. ábra: Digitális jel, bináris számrendszerben

Az arduino mikrokontrollerben található egy Analog-to-Digital-Converter (ADC), ami a bemeneti analóg feszültségjelet átalakítja egy digitális jellé. Konkrét példán keresztül a későbbiekben ismerjük meg a használatát.

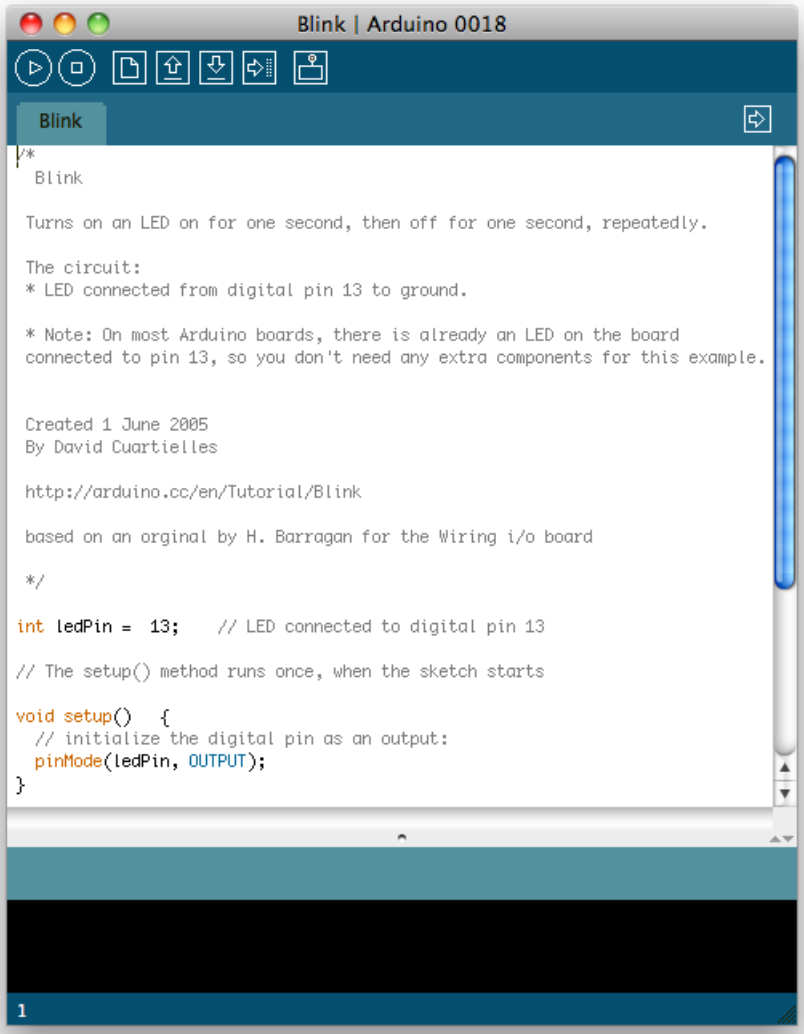
6.4. Arduino fejlesztőkörnyezet telepítése

Az Arduino Boardhoz egy Arduino nevű open source fejlesztőkörnyezet tartozik, aminek legfrissebb verziója a hivatalos oldalról letölthető háromféle operációs rendszerhez: Windows-hoz, Mac OS X-hez és Linuxhoz. A letöltött fájlt kicsomagoljuk, ha kell, és máris futtathatjuk az Arduino szoftvert, nem kell külön telepíteni. Letöltés erről a linkről lehetséges: http://arduino.cc/en/Main/Software

A Windows automatikusan telepíti az USB drivert, ha rákötjük az Arduinot. Macintoshon le kell tölteni, majd telepíteni kell. Az interneten FTDI USB Serial Driver néven találjuk meg.

A következők szerint töltsünk fel az Arduino Boardunkra egy minta fájlt, hogy le tudjuk ellenőrizni, megfelelően működik-e:

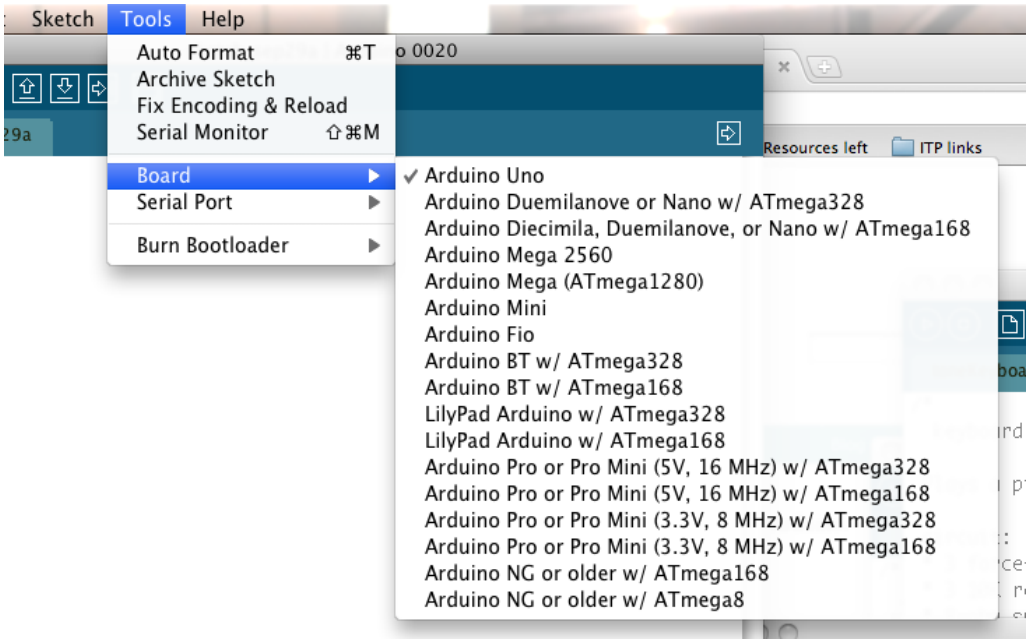
- Nyissuk meg az Arduino szoftvert!
- Keressük ki a mintapéldák között a Blink fájlt: **File/Examples/1. Basics/Blink**!



6.4.1. ábra: Blink megnyitása

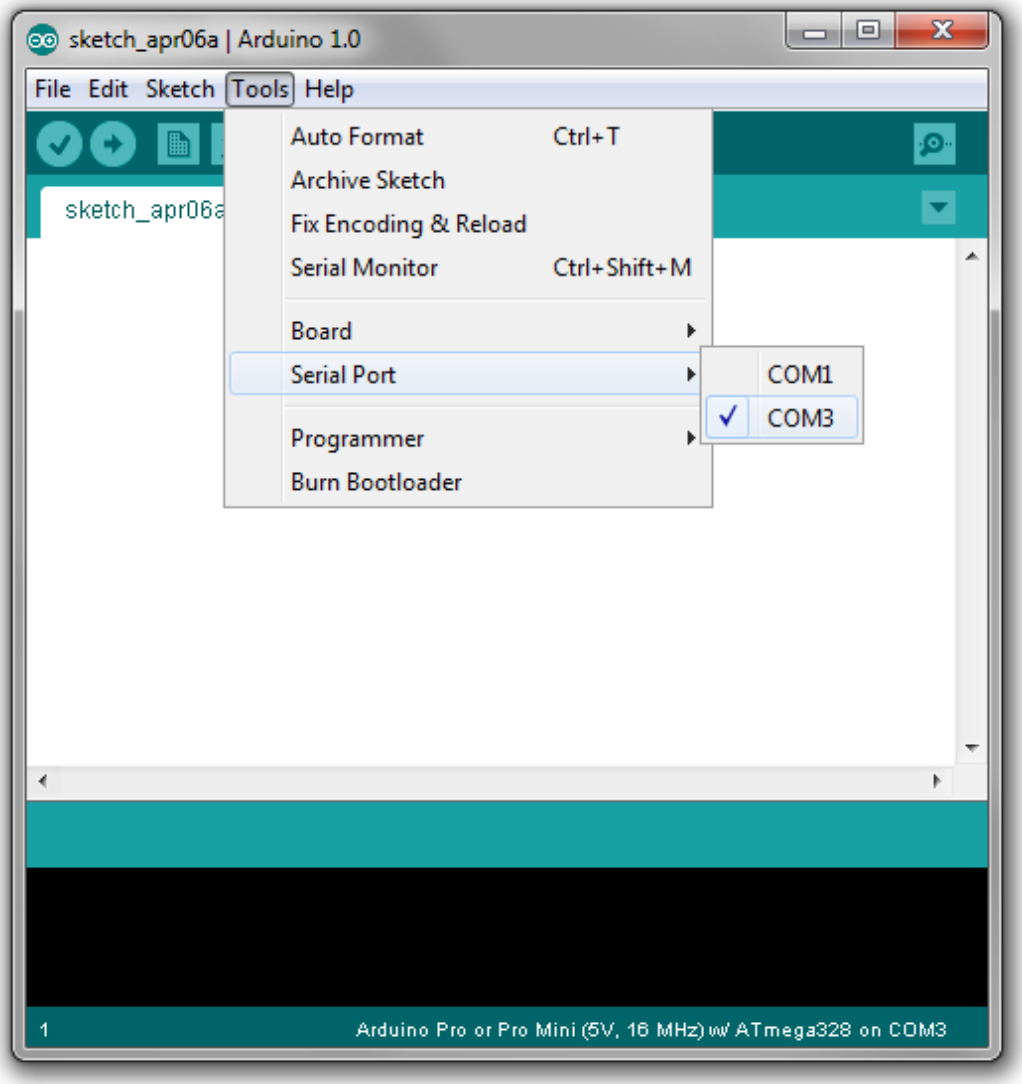
- Állítsuk be az általunk használt Arduino típusát. Ezt a **Tools/Board** lenyíló menüben tehetjük meg, úgy hogy klikkeléssel áttesszük a pipát az adott típushoz.

 © Harsányi Réka, Juhász Márton András



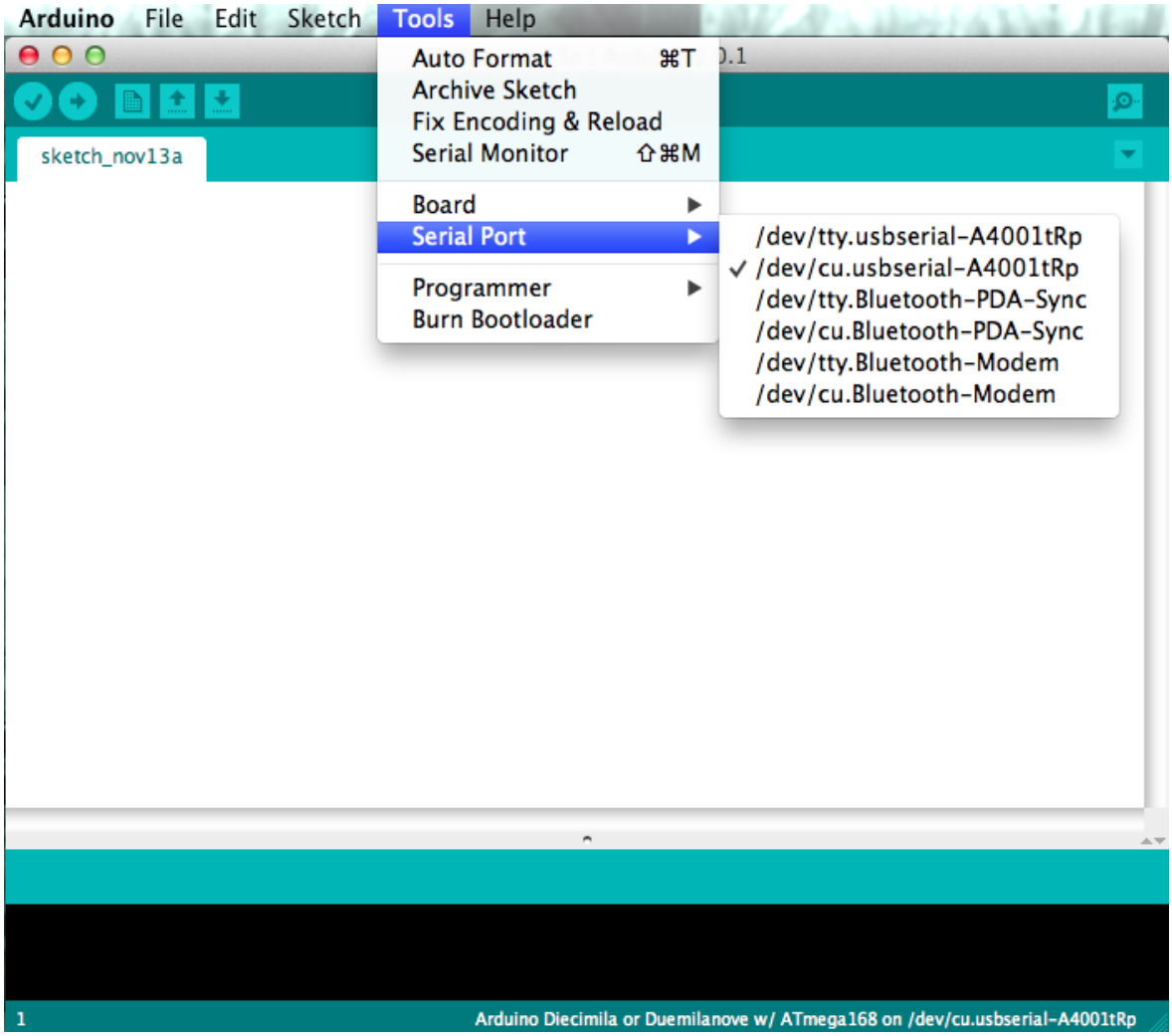
6.4.2. ábra: Arduino típusának kiválasztása

- Ezután ellenőrizzük a soros port beállítását a **Tools/Serial Port** menüben! Hogy épp melyiket használjuk (COM1, COM2, COM3, stb.), az Eszközkezelőben (Device Manager) meg tudjuk nézni. Egyszerűbb, ha megpróbáljuk feltölteni a kódot, és ha nem sikerül, akkor átállítjuk a portot egy másikra.



6.4.3. ábra: Soros port kiválasztása WIN

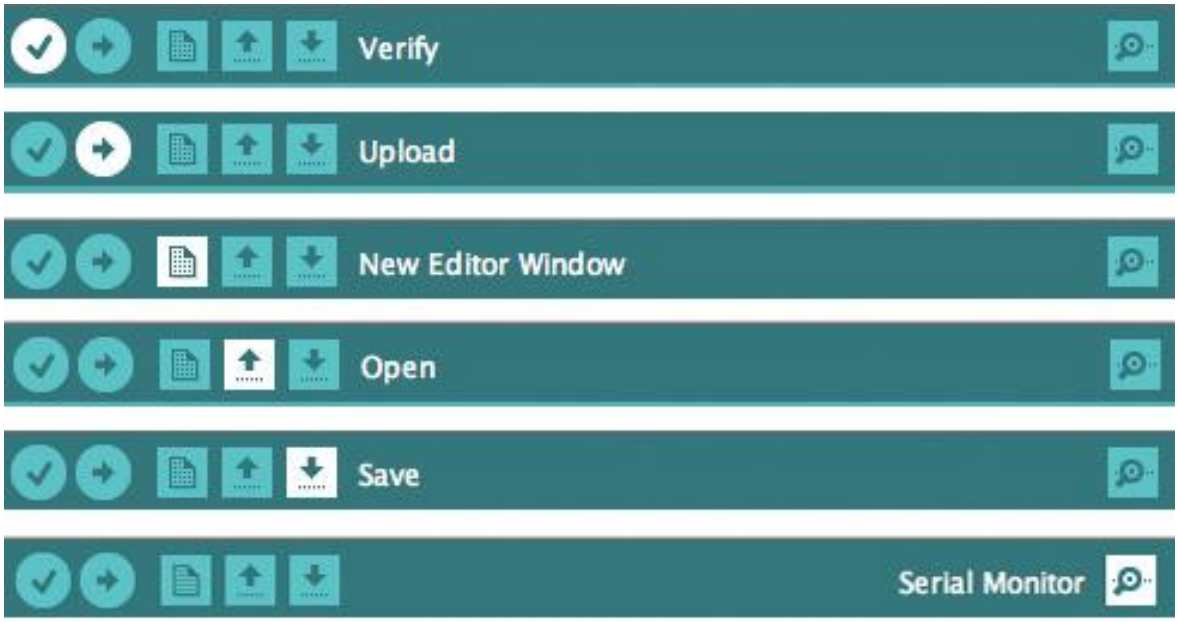
 © Harsányi Réka, Juhász Márton András



6.4.4. ábra: Soros port kiválasztása MAC

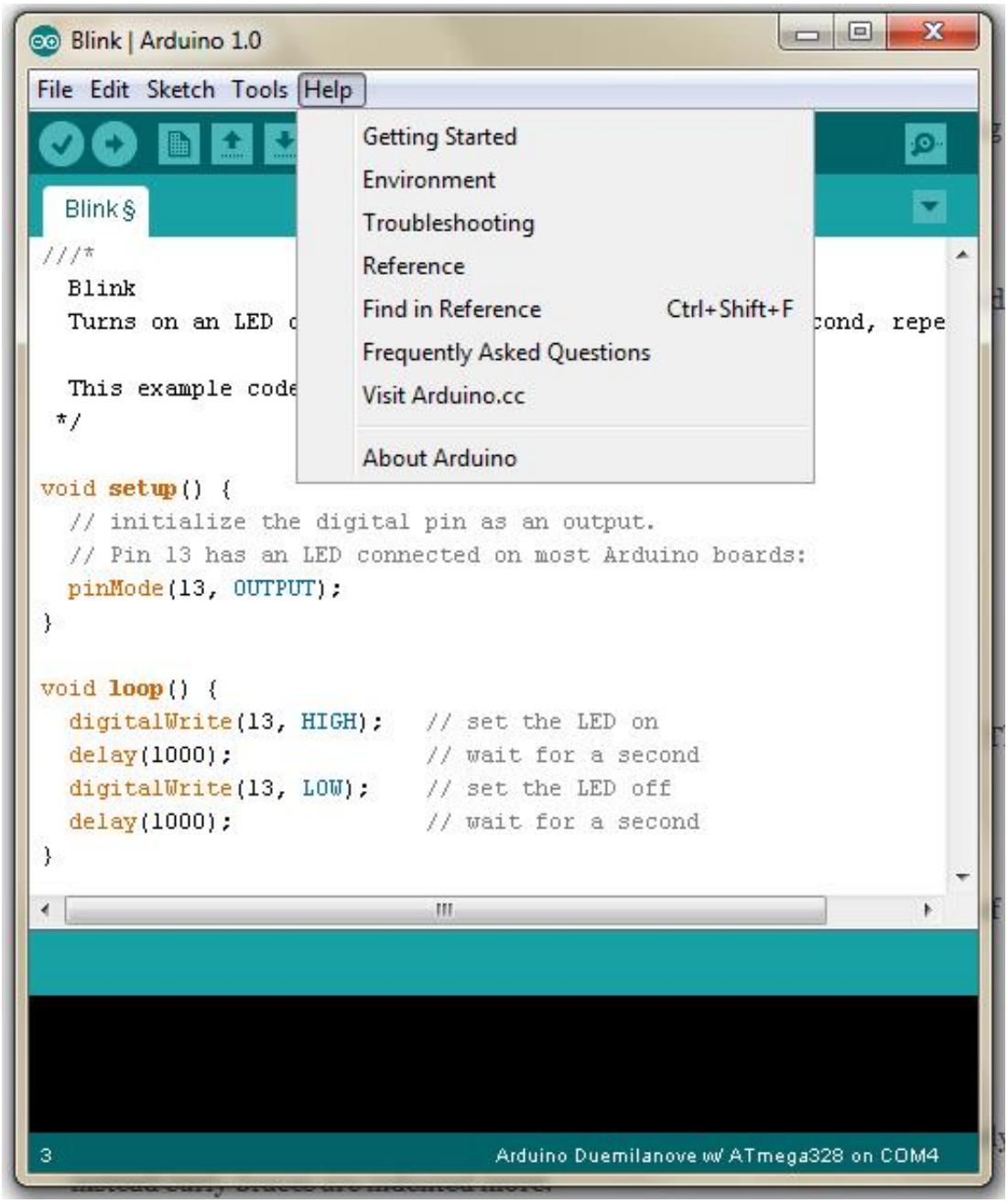
A második gombbal **Upload** tudjuk feltölteni a kódot a mikrokontrollerünkre. **Done Uploading** felirat jelzi, ha ez sikerült. Ekkor a beépített sárga LED villogni kezd úgy, hogy 1 másodpercig ég és 1 másodpercig nem, míg ki nem húzzuk az áramból, azaz az USB-ből.

Gombok balról jobbra: Ellenőrzés, Feltöltés, Új ablak, Megnyitás, Mentés és a jobb oldalon a Serial Monitort bekapcsoló gomb.



6.4.5. ábra: Gombok

Az alapokról, a fejlesztőkörnyezetről, hibakeresésről, gyakran ismételt kérdésekről stb. további információkat online, angol nyelven, a **Help** menüpont alatt találhatunk.



6.4.6. ábra: További segítségek

6.5. Programozás: szükséges minimum kód

Az Arduino által használ kódok, más néven **Sketch**-ek kiterjesztése a régebbi verzióknál **.pde** volt, de az 1.0 verzió óta már **.ino** kiterjesztést használ.

Arduino sketch írásnál a szükséges minimum a *setup()* és a *loop()* függvények meghívása (persze ehhez írhatunk saját függvényeket, de e nélkül nem fog működni a kódunk). Ezek az úgynevezett dedikált funkciók.

A **setup** funkció a program futtatásakor csak egyszer fut le az elején. Itt adhatjuk meg, hogy melyik pint hogyan akarjuk használni (bemenetként vagy kimenetként). Ekkor inicializáljuk a pineket (értékadás 0-val), illetve hívjuk meg a soros portot.

A **loop** funkció, mint a neve is mutatja, többször fut le egymás után. Itt változtathatjuk meg a bemeneti és kimeneti adatokat.

Ezek dedikált funkciók, tehát be vannak építve a fejlesztőkörnyezetbe. *Void*dal kell őket meghívni. -

Szintaxisok:

Setup: A kód legelején definiáljuk benne a változókat (inicializálás), pin módokat, könyvtár használatát stb. Csak egyszer fut le.

```
void setup() {
 // ide tesszük az állandókat
}
```

Loop: A setup után következik, folyamatosan, azaz többször fut le, lehetővé teszi, hogy a program változzon és reagáljon.

```
void loop() {
 // a változók helye  – ismétlődik a lefutás
}
```

További szintaxisok:

; pontosvessző

Az állítások lezárására használjuk.

```
int a = 13;
```

{ } kapcsos zárójel

 © Harsányi Réka, Juhász Márton András

Az arduino programnyelv legfőbb eleme, különböző konstrukciókban használjuk.

Ha van egy nyitó kapcsos zárójel, akkor mindig kell, hogy legyen egy hozzá tartozó záró is. Az Arduino fejlesztőkörnyezetébe be van építve, hogy ha az egyik zárójelet kijelöljük, megmutassa, melyik tartozik hozzá. Hasznos tipp, hogy ne felejtsük el bezárni a zárójelet, tegyük ki egyből mind a kettőt, utána írjunk be közé. Függvényeknél, ciklusoknál és feltételeknél használjuk.

// **megjegyzés**

Két darab perjel mögé beírhatunk magunknak megjegyzéseket a programsorba, melyeket a fordítóprogram figyelmen kívül hagy, így nem tölti fel őket az Atmega chipre.

Ez a kommentelés csak egy sorra vonatkozik. Ha több sort akarunk írni, a csillagos perjeleket használjuk:

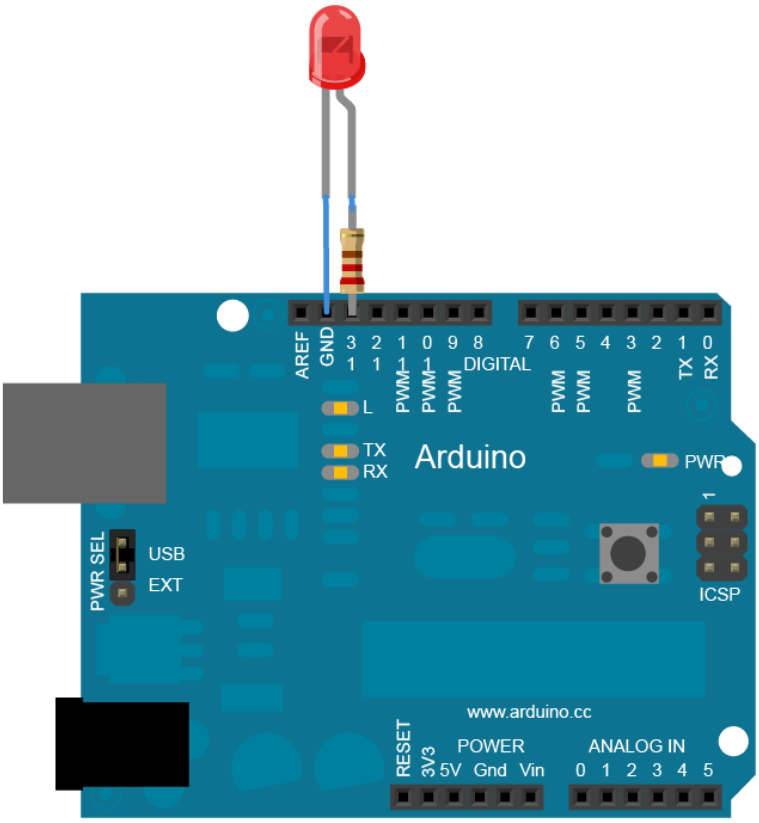
/* ide jön
több sor
megjegyzés ***/**

A SZINTAXISOK A TANANYAG LEGVÉGÉN **SZINTAXISOK GYŰJTEMÉNYE** CÍM ALATT ÖSSZEGYŰJTVE IS MEGTALÁLATÓAK!

6.6. Digitális pinek használata

6.6.1. LED villogtatása, kiolvasás digitális pinből

1 db 220 ohmos ellenállásra és 1 db 5 mm-es LED diódára lesz szükségünk. A LED negatív lába megy a földbe (rövidebb szára), a pozitív (hosszabb) pedig az ellenálláson keresztül a 13 digitális pinbe (ami a beépített LED kivezetése). Az ellenállásra a LED-hez jutó feszültség csökkentése miatt van szükségünk. Az átlagos 5 mm-es LED-ek kb. 20 mA-t fogyasztanak és 2,2–3,4 V-tal működnek. Az adott LED paramétereit a gyártó adatlapja tartalmazza. Az ellenállások színkódolása mutatja meg az értéküket. De használhatjuk az online kalkulátorokat is.



6.6.1.1. ábra: LED bekötése

Az alább található kódban először kimenetként beállítjuk a LEDhez kötött pint: *pinMode(13, OUTPUT);* Majd a loop ciklusban bekapcsoljuk a LED fényét a *digitalWrite(13, HIGH);* paranccsal. Ennek hatására az arduino 5V feszültséggel látja el a LED-et (ez túl sok lenne, de már ellenállás bekötésével lecsökkentettük az értékét). Ez után kikapcsoljuk a fényét a *digitalWrite (13, LOW);* sorral, ilyenkor a pin állapota 0V-os. Hogy a két állapot közötti változást a szemünk is le tudja követni, lelassítjuk a villogást, úgy hogy *delay()*-jel várakoztatjuk a program lefutását, azaz a megadott idő alatt nem történik változás, tehát ha ég a LED akkor úgy is marad adott ideig az állapota. A delay() időtartamát mikiszekundumban adjuk meg, 1000 ms = 1 másodperc.

Arduino kódja:

```
/*
  Villogás
  Bekapcsolja a LED-et egy másodpercre, utána kikapcsolja egy másodpercre, és
ezt ismétli folyamatosan. -
 */
```

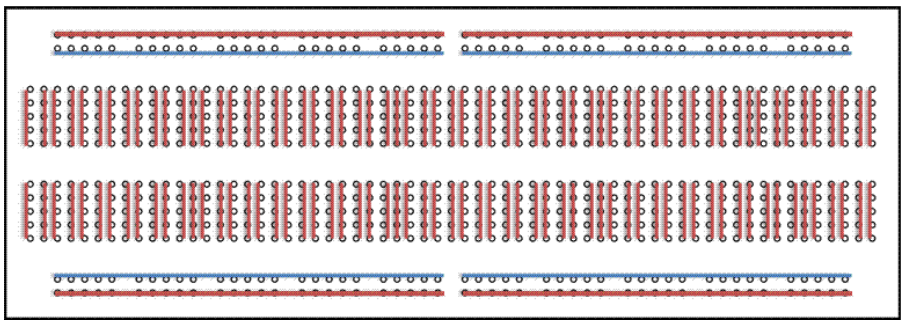
```
// A 13 pinhez egy SMD LED van kötve
// a legtöbb Arduino boardon.
// hozzunk létre egy változót és nevezzük el „led”-nek:
int led = 13;

// a setup rutin egyszer lefut, ha megnyomjuk a reset gombot:
void setup() {
   // a digitális pint kimenetként inicializáljuk:
   pinMode(led, OUTPUT);
}

// a loop rutin folyamatosan fut, mindig újrakezdi
void loop() {
   digitalWrite(led, HIGH);
   // bekapcsol a LED (HIGH a feszültség szint, 5 V)
   delay(1000);               // 1 másodpercig vár
   digitalWrite(led, LOW);
   // kikapcsol a LED (LOW, 0 V a feszültség)
   delay(1000);               // 1 másodpercig vár
}
```

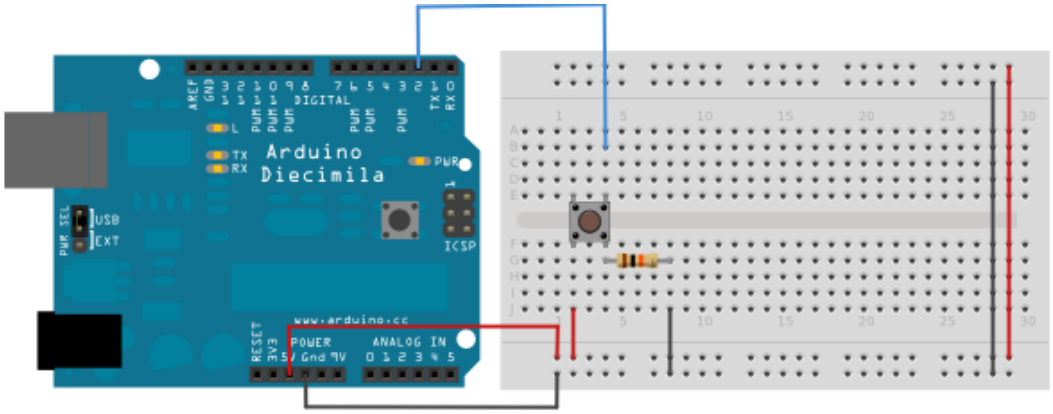
6.6.2. Nyomógomb használata, kiírás digitális pinre

Szükségünk lesz 1 db nyák mikro nyomógombra, 1 db 10 kiloohmos ellenállásra, illetve dugaszolós próbapanelre. Az ábrán látható módon kössük össze őket. A nyomógomb azért van középen, hogy a lábai ne legyenek egymáshoz bekötve, mivel a próbapanel közepén nincs átvezetés.



6.6.2.1. A dugaszolós próbapanel összeköttetései

Bekötése:



6.5.2.2. Nyomógomb bekötése

Amikor a nyomógomb nyitott állapotban van (nincs megnyomva), akkor nincs összeköttetés az 5V pinbe és a digitális pinbe bekötött lábai között, ezért a digitális pinből kiolvasott jel értéke LOW azaz 0V. Ilyenkor a földeléssel (Gnd pin) áll összeköttetésben (az ellenálláson keresztül) az 5V pin. Amikor a nyomógomb zárt állapotba kerül (le van nyomva) , akkor záródik az áramkör az 5V pin és a digitális pin között, ekkor a beérkező jel értéke HIGH azaz 5V. Az alábbi kóddal ez a példa úgy működik, hogy ha lenyomjuk a gombot, világít a LED.

Fordított módon is beköthetjük a nyomógombot, úgy hogy az ellenállást úgy hogy a gombhoz érkező poyitív és negatí oldalakat felcseréljük. Tehát az ellenálláson keresztül HIGH jel érkezik a digitális pinbe, amikor a gomb alap állapotban van (nincs megnyomva). LOW jel érkezik amikor lenyomjuk. Ebben az esetben az alábbi kódot használva, akkor alszik ki a LED fénye, ha a gombot lenyomjuk, azaz a földeléssel zárjuk össze a digitális pint.

Arduino kódja:

```
/*
  DigitalReadSerial
 Kiolvassa a digitális 2 pinből az adatot, amit bemenetnek használ, és az
eredményt kiírja a serial monitoron. -
 */

// digitális 2 pinbe egy nyomógombot csatlakoztatunk
// és elnevezzük:
```

```
int pushButton = 2;

// a setup rutin egyszer fut le, ha megnyomjuk a reset gombot:
void setup() {
   // inicializáljuk a soros kommunikációt 9600 bit/mp-en:
   Serial.begin(9600);
   // pushbutton nevű pint bemenetként meghatározzuk:
   pinMode(pushButton, INPUT);
}

// a loop rutin folyamatosan fut, mindig elölről kezdi:
void loop() {
   // kiolvassuk a bemeneti pint:
   int buttonState = digitalRead(pushButton);
   // kiírjuk a gomb állapotát serial monitoron:
   Serial.println(buttonState);
   // várakozás a kiolvasások között,
   // hogy stabilabban működjön:
   delay(1);
}
```

A jobb felső sarokban lévő gombbal tudjuk megnyitni a serial monitort. Vagy 0-t vagy 1-t ír ki, attól függően, hogy meg van-e nyomva a gomb.

6.7. Analóg pinek használata

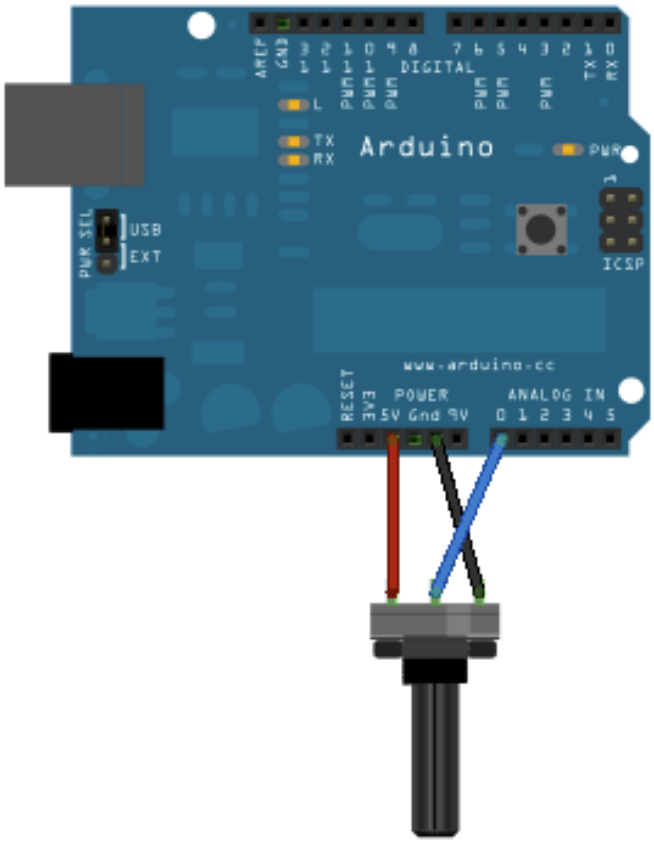
6.7.1. Potméter használata, kiolvasás analóg pinből

A potméter egy változtatható feszültségű ellenállás, egy 10 kiloohmos szükséges a példához. A belőle nyert változó adatokat fogjuk kiolvasni, de ugyanezen elven bármilyen analóg szenzor beköthető.

A potméter tengelyének elfordításával az ellenállás értékét változtathatjuk a középső és minkét szélső kivezetése között, ilyenkor a középső kivezetés feszültség értéke változik. Amikor az 5V pinhez futó és a középső pin közötti ellenállás megközelítőleg nulla (ilyenkor a másik szélső lábon megközelítőleg 10KOhm az ellenállás ebben a példában), akkor a középső pin feszültsége megközelítőleg 5V. Ha teljesen elfordítjuk a tengelyt, azaz megnöveljük az ellenállást, a feszültség 0V lesz. Ezt a feszültség érték változást fogjuk kiolvasni az Arduino analog bemmentén.

Az Arduino egy olyan belső áramkörrel rendelkezik, ami az analóg jelet digitálissá alakítja (**analog-to-digital converter**). Az analóg bement kiolvasáshoz az *analogRead()* funkciót fogjuk használni, ami egy számértéket ad vissza 0 és 1023 között (a használt feszültséghez arányosítva, pl. 5V = 1023; 0V = 0).

Bekötése: Ha a lábai előre mutatnak, akkor balra tekerjük el teljesen a forgógombot, hogy beszerelés közben ne továbbítsa az áramot. Első lába az 5 V pinbe megy, a középső az analóg A0-ba, a harmadikat pedig le kell földelni a GND pinbe.



6.7.1.1. ábra: Potméter bekötése

A kód setup függvényében az egyetlen dolog amit csinálunk, hogy bekapcsoljuk a soros port kommunikációt az Arduino és a számítógépünk között 9600 bit/sec adatsebességgel: *Serial.begin(9600);* Ez után a loop cikluson belül meghatározunk egy integer (egész számértéket tároló) változót, amiben a potméterből kiolvasott adatot (0-1023) tároljuk el: *int sensorValue = analogRead(A0);* Majd a soros ablakban (serial monitor) megjelenítjük az adatok decimális értékét, sortördeléssel: *Serial.println(sensorValue, DEC)*. A kezelőfelületen a jobb oldali felső ikonnal tudjuk megnyitni az ablakot. Ahogyan tekerjük a potmétert úgy változik a megjelenített adat.

Arduino kód:

```
/*
  AnalogReadSerial
  Az analóg 0 pinből olvassa ki az értékeket, amiket utána serial monitoron
kiír. A potméter középső lábát A0-ba kötjük, a két szélsőt az 5 V-ba és a
földbe.  -
 */

void setup() {
   // inicializáljuk a soros kommunikációt 9600 bits/mp-en:
   Serial.begin(9600);
}

void loop() {
   // kiolvassuk a bemenetet analóg 0 pinen:
   int sensorValue = analogRead(A0);
   // kiprinteli a kiolvasott értéket:
   Serial.println(sensorValue);
   delay(1);
   // a delay késlelteti a kiolvasást, a stabilitás érdekében
}
```

6.7.2. Analóg jel feszültség értékének kiírása serial monitoron

A feladathoz az előző példának megfelelően kötjük be a potméterünket, tehát ismét analóg feszültségváltozást olvasunk ki. -

A következő példakód serial monitorra írja ki a beérkező feszültség értékét, amihez – a pontosabb eredmény eléréséhez – egy float változót definiálunk. A kód ugyanúgy kezdődik mint az előző esetben, bekapcsoljuk a soros portot és eltároljuk a potméterből kiolvasott értéket egy integerben. Ez után a feszültség értékhez létrehozunk egy float (tizedes tört értékek tárolására alkalmas) változót, amiben a potméterből jövő adat feszültség értékét eltároljuk: *float voltage= sensorValue * (5.0 / 1023.0);* Vágül ezt megjelenítjük a soros ablakban: *Serial.println(voltage).*

Arduino kód:

```
/*
 ReadAnalogVoltage
 Kiolvassa az analóg 0 pin értékét, az eredményt pedig átszámolja feszültség
értéknek, majd kiírja serial monitoron. A potméter középső lábát A0-ba kötjük,
a két szélsőt az 5 V-ba és a földbe. -
 */

void setup() {
   // inicializáljuk a soros kommunikációt 9600 bits/mp-en:
   Serial.begin(9600);
```

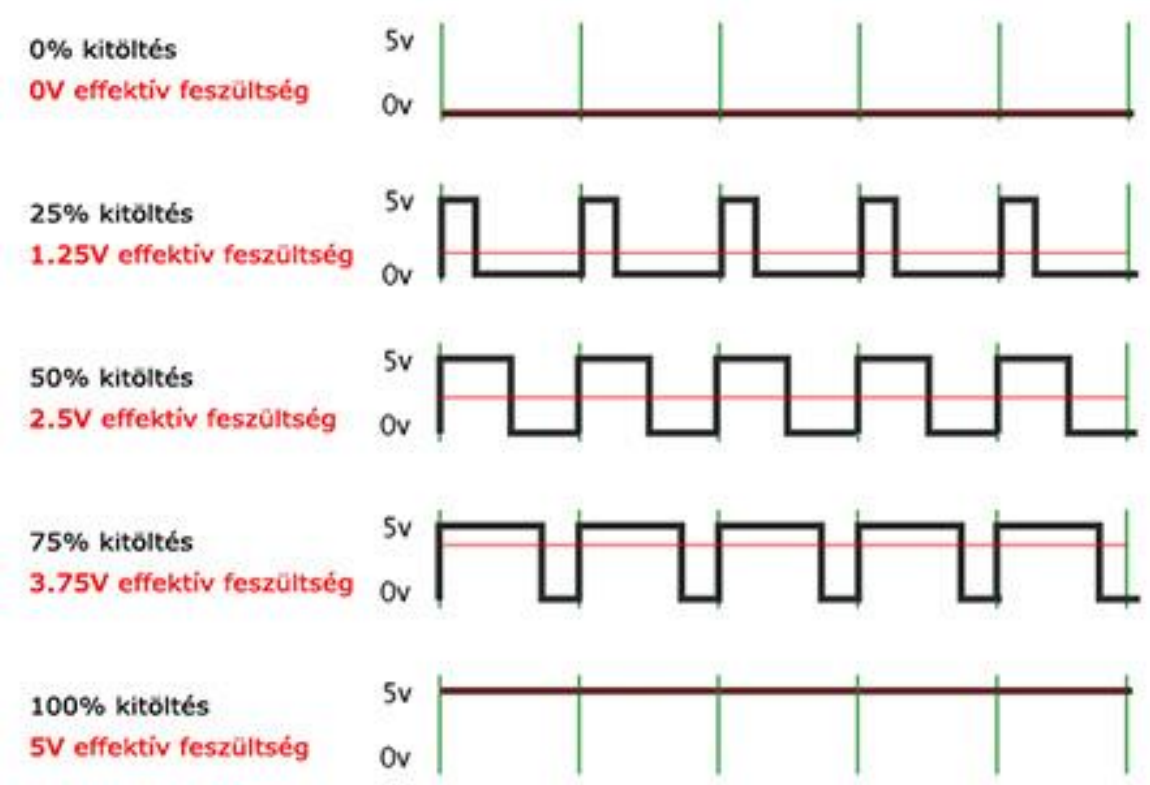
```
}

void loop() {
   // bemenetként használt A0 pin kiolvasása:
   int sensorValue = analogRead(A0);
   // analóg érték (0-1023) átkonvertálása feszültségértékre (0-5V):
   float voltage = sensorValue * (5.0 / 1023.0);
   // kiírja a kiolvasott értéket:
   Serial.println(voltage);
}
```

6.8. Impulzus-szélesség moduláció

6.8.1. PWM pinek működése

Ez a jel is 0-t és 1-t tud értékként felvenni, mint minden digitális jel. Amplitúdója állandó, de változó szélességű impulzusokból áll, így befolyásolható a kimeneten létrejövő tényleges feszültség.

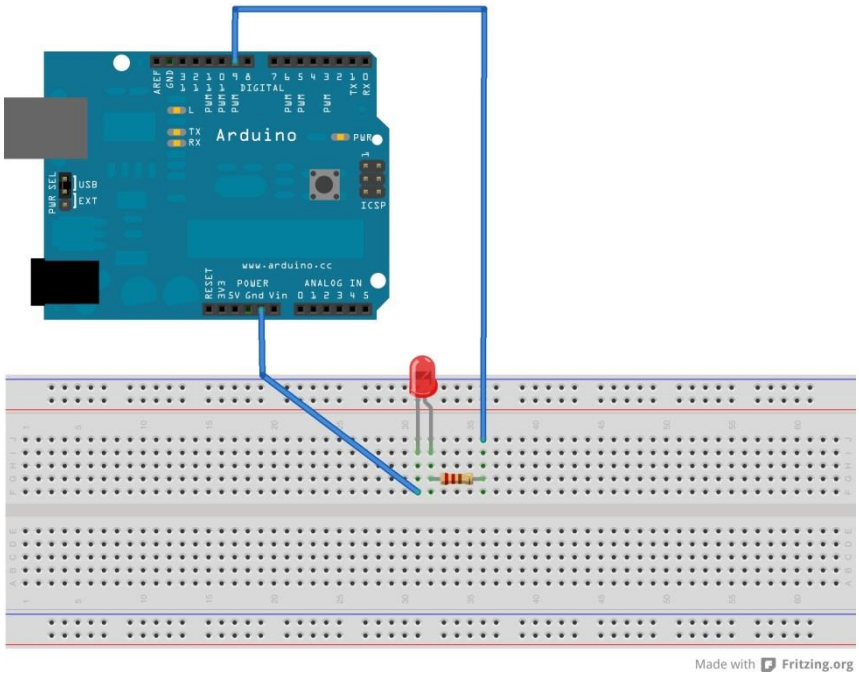


6.8.1.1. ábra: PWM jel
(kép: www.cnv.hu/gallery/cikkek/elektronika/ pwm%20magyarazat.jpg)

6.8.2. LED fényerejének szabályozása PWM-mel

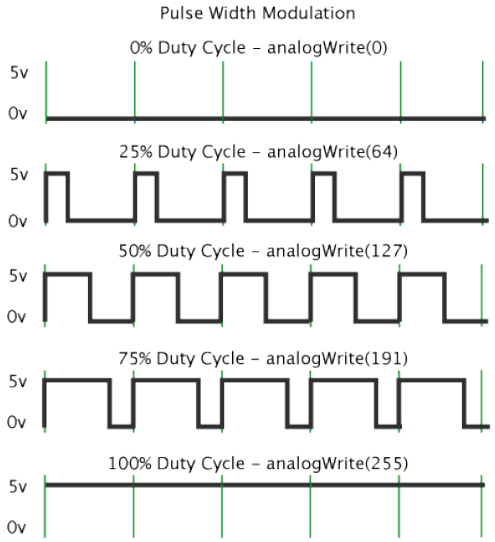
Ebben a feladatban azt vizsgáljuk meg, hogyan lehet a digitális jel manipulációjával szabályozni a feszültséget. Az Arduino néhány digitális pinje mellé oda van írva a PWM, ezek közül használjunk egyet.

Bekötése: a LED pozitív lábát az ellenálláson keresztül – ugyanúgy, mint eddig – a 9-PWM digitális pinbe, a negatív lábát a földbe (Gnd) kötjük.



6.8.2.1. ábra: LED bekötése PWM pinbe

Az *analogWrite()* funkciót használjuk. A PWM hihetetlenül gyorsan kapcsolja a pin állapotát ki-be, így idézi elő az analóg jelre emlékeztető elhalványuló effektust. 0–255 közötti értéket küldhetünk a PWM pinekre, pl. az analogWrite(127); 50%-ot jelent. Az ábrán a zöld vonalak az időt jelölik, 2 milliszekundumonként.



6.8.2.2. ábra: PWM jel magyarázata LED-hez

Először létrehozunk változókat: a LED a 9 pinbe van kötve. A fényerőnek, és az értéknek amivel változtatjuk majd, szintén létrehozunk integer tárolókat. A setupban kimenetként megadjuk a LED pint. A loop ciklusban analogWrite() segítségével megadjuk melyik pinre (*led*) és mekkora értéket (*brigthness*) akarunk küldeni. A PWM pin 0-255 közötti értéket fogad. A fényerő értékét folyammatosan változtatjuk, egy fix értékkel (*fadeAmount*) növeljük, ilyenkor a LED egyre világosabb. Ez addig zajlik még el nem éri a szélső, 255 értéket: *if (brightness == 0 || brightness == 255) {fadeAmount = -fadeAmount ;}* Ekkor a folyamat megfordul, mivel negatív előjelűre vált a léptetés mértékét eltároló változó, így az érték 5-ösével csökkenni fog amíg el nem éri a nullát, tehát a LED fénye halványodik. Az if feltételben azt látjuk hogy ha a fényerő értéke eléri a nullát VAGY a 255 csak akkor teljesül a feltétel, azaz csak akkor fut le a kapcsos zárójelen belül található programrész, aminek hatására megfordul a folyamat iránya. Mivel az *analogWrite()* nagyon gyorsan tudja változtatni a PWM értékeket, a kód végérén található egy késleltetés (*delay*), hogy lássuk a végbemenő változást.

Arduino kód:

```
/*
 Fade
 Ez a példa megmutatja, hogyan halványítsuk a LED fényerejét a 9 PWM pin és az
analogWrite() használatával. -
 */

// létrehozzuk az állandókat, az integer egész számot tárol:
int led = 9;           // ide van kötve a LED
```

```
int brightness = 0;    // fényerő értéke
int fadeAmount = 5;    // a változás léptéke

void setup()  {
   // deklaráljuk a 9 pint kimenetként:
   pinMode(led, OUTPUT);
}

void loop()  {
   // fényerő értékét hozzárendeljük a LED pinhez:
   analogWrite(led, brightness);

   // megnöveljük a fényerő értékét az általunk megadott
   // léptékben, mindig, mikor lefut a loop, hozzáadódik
   brightness = brightness + fadeAmount;

   // a végén megfordítjuk a változás irányát:
if (brightness == 0 || brightness == 255) {
         fadeAmount = -fadeAmount ;
   }

   // 30 milliszekundumot várunk,
   // hogy szemünk is láthassa a változást:
   delay(30);
}
```

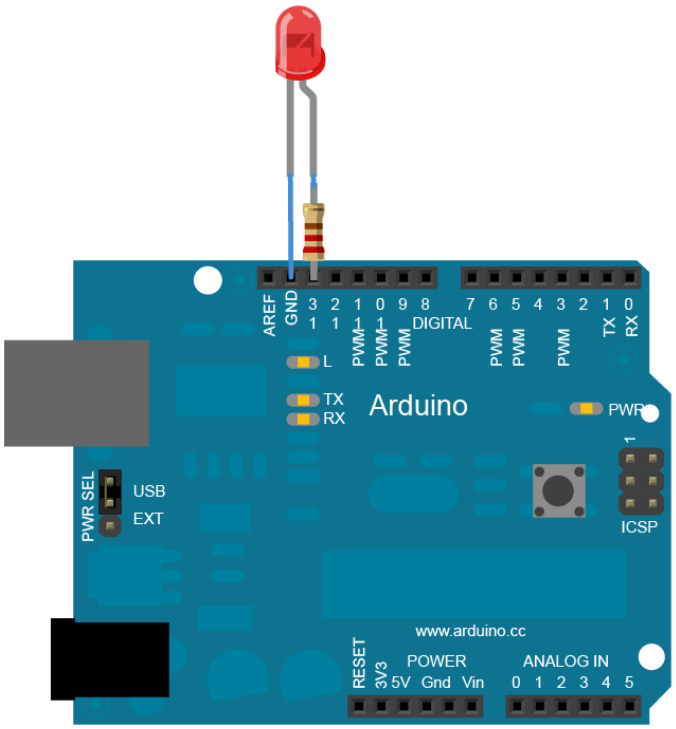
7. Példák digitális pinekhez

A következő alfejezetekben különböző eszközöket fogunk digitális pinekre kötni: LED-et, nyomógombot, kis hangszórót, miközben újabb típusú változókkal és – többek között – az **if** kondícióval, a **for** ciklussal, a **tone** és a **millis** függvényekkel ismerkedünk meg, valamint az Arduino belső felhúzó-ellenállását is használni fogjuk.

7.1. LED villogtatása delay() nélkül

Néha azt szeretnénk, hogy párhuzamosan történjenek dolgok, pl. miközben villog a LED egy nyomógomb állapotát is vizsgálni szeretnénk. Ilyenkor a *delay()*-t nem tudjuk használni, hiszen az az egész kód lefutását késlelteti, és lehet hogy pont akkor nyomjuk meg a gombot, amikor a kód áll, így elmumlasztjuk a kiolvasását. Ezért az alábbi kódban változók segítségével hozzuk létre a villogást úgy, hogy az eltelt idő mértékét vizsgáljuk.

A következő példában a LED mindig az ellenkező állapotába kerül, mint amilyenben épp van, tehát ha be van kapcsolva, akkor kikapcsol, és fordítva. Mivel a loop() folyamatosan fut, minden alkalommal megvizsgálja, hogy az if feltétel igaz-e, és ahhoz képest lép tovább a kódban. Használhatjuk az Arduino beépített LED-jét vagy bekötethünk egyet a szokásos módon – a pozitív lábát 220 ohmos ellenállásom keresztül a 13 pinbe, a negatív lábát pedig a földelésbe (Gnd).



7.1.1. ábra: LED bekötése 13 pinbe

Arduino kód:

```
/* Blink without Delay
Ki- és bekapcsolja a LED-et a delay függvény használata nélkül. Ez azt jelenti,
hogy másik kód is tud futni vele egy időben, hiszen a delay() nem akasztja meg
annak lefutását.

 created 2005 by David A. Mellis
 modified 8 Feb 2010 by Paul Stoffregen
 -
 */

// a konstansok (állandók) nem változnak, ezért velük
// határozzuk meg a pineket
const int ledPin =  13;      // LED pin

// Variables will change:
int ledState = LOW;
// ledState változó tárolja a LED állapotának értékét
long previousMillis = 0;
// a LED frissítésének utolsó időértékét tárolja

// a következő változónk egy long less, mivel az érték
```

```
// gyorsan akkorára nő hogy integerben nem tudjuk tárolni
long interval = 1000;
// villogáshoz intervalluma (milliszekundumban)

void setup() {
   // kimenetnek állítjuk be digitális pinünket
   pinMode(ledPin, OUTPUT);
}

void loop() {
  /* ide az a kód kerül, aminek mindig futnia kell
Szünetet tart, hogy lássa, fel kell-e villantani a LED-et: úgy, hogy
megvizsgálja, a jelenleg eltelt idő és az utolsó felvillanás idejének
különbsége nagyobb-e, mint az intervallum, ami alatt fel akarjuk villantani a
LED-et.
*/

   unsigned long currentMillis = millis();

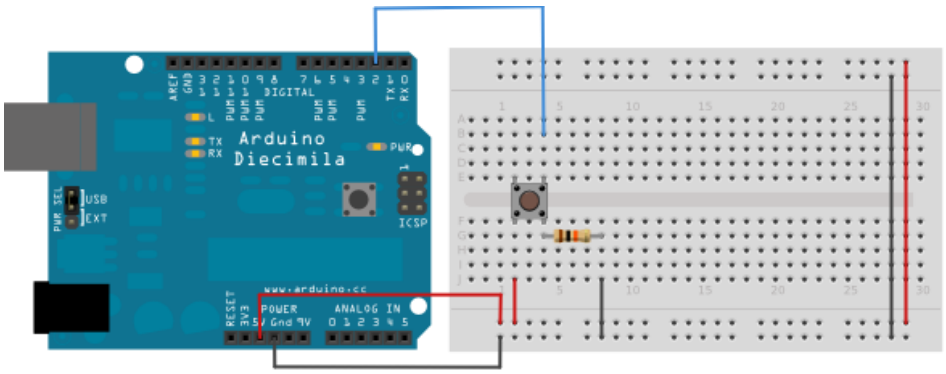
   if(currentMillis - previousMillis > interval) {
       // elmenti a LED felvillanás utolsó időértékét
       previousMillis = currentMillis;

       // ha a LED ki van kapcsolva, akkor bekapcsolja
       // és fordítva:
       if (ledState == LOW)
            ledState = HIGH;
       else
         ledState = LOW;

         // a ledState változóval beállítjuk a 13 pin állapotát:
         digitalWrite(ledPin, ledState);
     }
   }
```

7.2. Nyomógomb LED-del

Ismét a beépített, 13 digitális pinre kivezetett LED-ünket fogjuk használni, valamint egy nyomógombot, amit a 7.2.1. számú ábra szerint kötünk be a digitális 2 pinbe, a feszültségbe (5 V) és egy 10 kiloohmos ellenálláson keresztül a földbe (Gnd).



7.2.1. ábra: Nyomógomb bekötése

Amikor a nyomógomb nyitva van (tehát nincs lenyomva), akkor a lábak között nincs összeköttetés, így – mivel a gombon át a földeléshez vagyunk kötve – a pinünkből LOW jelet tudunk kiolvasni. Ha a gomb lenyomásával zárjuk az áramkört, akkor a jelünk az 5 V pinből érkezik, azaz HIGH lesz.

Ha fordítva kötjük az áramkört, azaz az ellenállással a HIGH jelet tartjuk meg, akkor gombnyomásra LOW-t kapunk, így a LED akkor alszik el, ha nyomjuk a gombot.

Arduino kód:

```
/*
  Button
A 13 pinbe kapcsolt LED-et ki-be kapcsolja, ha megnyomjuk a 2 pinbe kötött
nyomógombot.

 created 2005 by DojoDave <http://www.0j0.org>
 modified 30 Aug 2011 by Tom Igoe
-*/

// konstansokkal beállítjuk a pineket:
const int buttonPin = 2;     // nyomógomb 2 pinbe megy
const int ledPin =  13;      // LED pin

// változók:
int buttonState = 0;
// nyomógomb állapotát tároló változó

void setup() {
   // kimenetként határozzuk meg a LED pint:
   pinMode(ledPin, OUTPUT);
   // a nyomógombot bemenetként:
   pinMode(buttonPin, INPUT);
}
```

```
void loop(){
   // kiolvassuk a nyomógomb állapotának értékét:
   buttonState = digitalRead(buttonPin);

   // megvizsgáljuk, be van-e nyomva a gomb
   // ha igen, tehát a buttonState változó HIGH értékű
   // ha nem, akkor LOW
   if (buttonState == HIGH) {
       // ha ez a feltétel igaz, akkor bekapcsolja a LED-et
       digitalWrite(ledPin, HIGH);
   } else {
       // ha a feltétel nem igaz, kikapcsolja a LED-et
       digitalWrite(ledPin, LOW);
   }
}
```

7.3. Késleltetés (debounce)

Ehhez a feladathoz az előző példa alkatrészeit és bekötési ábráját használjuk. A nyomógombot kapcsolóként használjuk, mely először bekapcsolja, majd második megnyomásnál kikapcsolja a LED-et. Késlelteti a bemeneti jelet, mely azt jelenti, hogy nagyon rövid időn belül kétszer is leellenőrzi, meg van-e nyomva a gomb. Késleltetés használata nélkül előfordulhat, hogy az egyszeri gombnyomást a kód többszörinek veszi. **Millis()** függvény segítségével követhetjük a gombnyomás időtartamát. Ugyanúgy HIGH jelet kap a bemeneti pinünk, ha lenyomjuk a gombot, mit az előző példában, és LOW jelet, ha nincs. Amikor a 2 bemeneti pin állapota LOW-ról HIGH-ra változik, átkapcsolja a kimeneti pint LOW-ról HIGH-ra, és a következő alkalommal vissza.

Arduino kód:

```
/*
 Debounce
 Nyomógomb megnyomásával tudjuk be-, majd újbóli megnyomásával  kikapcsolni a
LED fényét.

 Áramkör:
 * LED a 13 pinbe és a földbe kötve (beépített LED)
 * nyomógomb 2 pinből az 5 V-ba megy
 * 10 kiloohmos ellenállás a 2 pinből

 created 21 November 2006 by David A. Mellis
 modified 30 Aug 2011 by Limor Fried
 - */
```

```
// állandókkal megadjuk a pin értékeket:
const int buttonPin = 2;     // nyomógomb pin
const int ledPin =  13;      // LED pin

// változókat hozunk létre:
int ledState = HIGH;   // kimeneti pin aktuális állapota
int buttonState;   // bemeneti pin aktuális állapota
int lastButtonState = LOW;   // bemeneti pin előző értéke

/* a következő változók long típusúak lesznek, mivel az időt milliszekundumban
mérjük, ezért értéke gyorsan nagyra nő, akkorára, amit integerben nem tudunk
tárolni */

long lastDebounceTime = 0;
// a kimeneti pin átkapcsolásának utolsó időértéke
long debounceDelay = 50;
// késleltetési idő; ha a kimenet vibrál, meg kell növelni

void setup() {
   pinMode(buttonPin, INPUT);
   pinMode(ledPin, OUTPUT);

   // adjuk meg a LED állapotának kezdő értékét
   digitalWrite(ledPin, ledState);
}

void loop() {
   // helyi változóba kiolvassuk a kapcsoló állapotát:
   int reading = digitalRead(buttonPin);

   // leellenőrizzük, hogy elég idő telt-e el a két kiolvasás között:

   // Ellenőrizzük, hogy megnyomás vagy zaj
   // hatására változott-e a kapcsoló:
   if (reading != lastButtonState) {
      // a késleltetés időzítőt újraindítjuk:
      lastDebounceTime = millis();
   }

   if ((millis() - lastDebounceTime) > debounceDelay) {
   /* ha az eltelt időből kivonjuk az utolsó átkapcsolás idejét, és az nagyobb,
mint a késleltetési idő, akkor - mivel ez gombnyomásra utal - a reading nevű
integer változóban eltárolt adatunkat hozzárendeljük a nyomógomb aktuális
értékét tároló változónkhoz */

      // ha a nyomógomb állapota megváltozik:
      if (reading != buttonState) {
            buttonState = reading;

         // ha a nyomógomb állapota HIGH, benyomott, akkor
         // átkapcsoljuk a LED állapotát az aktuális ellenkezőjére
         if (buttonState == HIGH) {
```

```
            ledState = !ledState;
          }
        }
    }

    // és ezzel az értékkel beállítjuk a  LED állapotát:
    digitalWrite(ledPin, buttonState);

    // majd elmentjük. A loop következő lefutásánál már
    // ez lesz a kezdő értéke ennek a változónak
    lastButtonState = reading;
}
```

7.4. Nyomógombos számláló

Ebben a példában egy nyomógombbal működő számlálót készítünk. Ahhoz, hogy a megnyomások számát követni tudjuk, ugyanúgy a bemeneti pin állapotát (HIGH, LOW) kell figyelnünk, és azt kell számolnunk, hogy ez a változás hányszor történt meg. Ezt „állapotváltozás érzékelés”-nek vagy „szélsőérték érzékelés”-nek hívjuk.

A következő kód folyamatosan kiolvassa a gomb állapotát és összehasonlítja az előzőleg kiolvasott értékkel. Amennyiben a két állapot különböző és az aktuális állapota HIGH, akkor a gomb be van nyomva, így a kódban eggyel nő a nyomógomb számlálóértéke. Ezt szintén megvizsgáljuk, és ha az a négy többszöröse, akkor bekapcsolja a beépített (13 pin) LED-ünk, egyéb esetekben kikapcsolja. Továbbra is a második példa alkatrészeit és bekötési ábráját alkalmazzuk.

Arduino kód:

```
/*
  State change detection (edge detection)
Szélsőérték érzékelés: Ez a példa azt mutatja meg, hogyan lehet érzékelni,
amikor a gomb állapota megváltozik.

 created  27 Sep 2005
 modified 30 Aug 2011 by Tom Igoe
- */

const int  buttonPin = 2;
const int ledPin = 13;

// létrehozzuk a megnyomások darabszámát tároló változót:
int buttonPushCounter = 0;
int buttonState = 0;    // gomb aktuális állapotát tárolja
int lastButtonState = 0;  // gomb előző állapotát tárolja
```

```
void setup() {
   // nyomógomb pinjét bemenetként inicializáljuk:
   pinMode(buttonPin, INPUT);
   // LED pinjét pedig kimenetként:
   pinMode(ledPin, OUTPUT);
   // inicializáljuk a soros portot (9600 bit/sec):
   Serial.begin(9600);
}


void loop() {
   // kiolvassuk a nyomógomb bemeneti pinjét:
   buttonState = digitalRead(buttonPin);

   // összehasonlítja a nyomógomb aktuális
   // állapotát az előzővel:
   if (buttonState != lastButtonState) {

      // ha az állapot változott, megnöveli a számláló értékét:
      if (buttonState == HIGH) {
         // ha az aktuális állapot HIGH, akkor
         // a gomb meg lett nyomva
         buttonPushCounter++;
         Serial.println("on");

         // ekkor a serial motor kiírja az on szót,
         // és a megnyomások számát is:
         Serial.print("number of button pushes:  ");
         Serial.println(buttonPushCounter);
       } else {
         // ha a gomb aktuális állapota LOW,
        // akkor a gomb ki lett kapcsolva
        Serial.println("off");
     }
   }

   // az aktuális állapotot elmenti, mint utolsó állapotot
   // a loop következő lefutásához
   lastButtonState = buttonState;

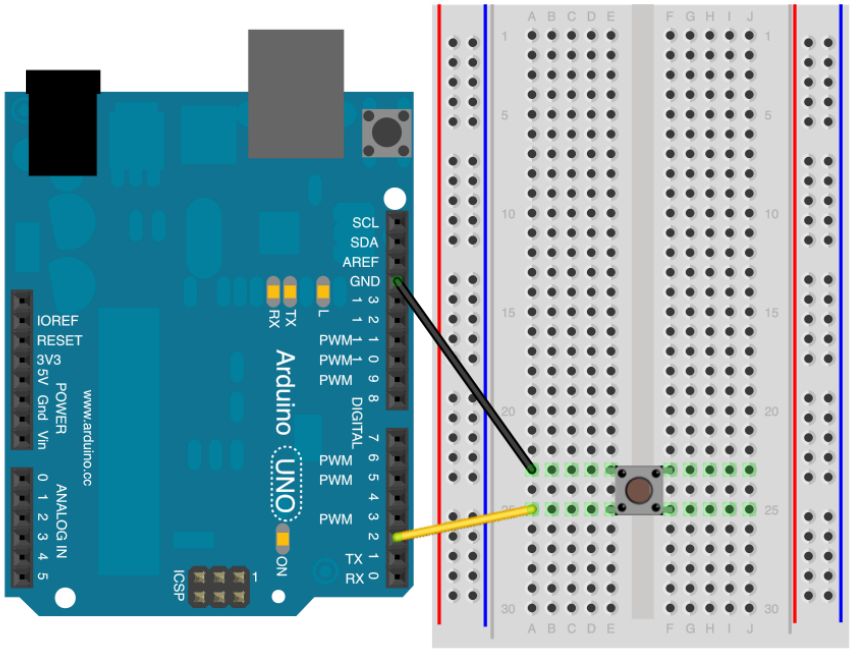
   // minden negyedik gombnyomásra bekapcsolja a LED-et úgy,
   // hogy leellenőrzi a gombnyomás számlálót.
   // a moduló függvény a két szám hányadosának
   // eredményét nézi
   if (buttonPushCounter % 4 == 0) {
      digitalWrite(ledPin, HIGH);
   } else {
      digitalWrite(ledPin, LOW);
   }
}
```

7.5. Pinek beépített felhúzó ellenállása

Minden pin rendelkezik egy beépített, programozható felhúzó ellenállással. Ez a példa azt mutatja meg, hogyan használjuk az INPUT_PULLUP parancsot a pinMode() függvény segítségével. A kapcsoló állapotát figyeli azáltal, hogy soros kommunikációt létesít az USB porton keresztül az Arduino és a számítógép között. A kommunikáció a (belső) soros porton (más néven: COM) történik, általában ki kell választani, hogy melyiket használjuk.

Ezen felül a kódban, ha a bemenet HIGH, akkor az Arduino boardba beépített LED bekapcsol, és LOW állapot esetén a LED kikapcsol. -

Kössünk be egy nyomógombot a 7.5.1. számú ábra szerint, az egyik lábát a földbe (GND), a másikat pedig a digitális 2 pinbe.



7.5.1. ábra: Nyomógomb bekötése INPUT_PULLUP példához

A nyomógombok vagy a kapcsolók alapállapotban nyitva vannak (amennyiben ilyen típust használunk, hiszen több fajta létezik), ami azt jelenti, hogy a két lába között nincs kapcsolat. Mivel a belső ellenállás a digitális 2 pinen aktív és 5 V-ra van csatlakoztatva (mivel ennyi feszültséget kap az USB-ből az Arduino, tehát ennyi jön a digitális pinből is), ezért ekkor HIGH jelet olvasunk ki. Ha a gombot lenyomjuk vagy bekapcsoljuk a kapcsolót, azzal zárjuk az áramkört, így – mivel közvetlen lesz a kapcsolat a földeléssel – LOW jelet kapunk.

 © Harsányi Réka, Juhász Márton András

A kódban először létrehozzuk a soros kommunikációt 9600 bit/sec-on, majd engedélyezzük a 20 kiloohmos belső felhúzó ellenállást, illetve beállítjuk kimenetként a 13 LED-pint. Utána egy változó segítségével kiolvassuk a gomb állapotát, amelynek decimális értékét kiíratjuk serial monitoron – jelen esetben ez 0 vagy 1 lesz (0 = benyomva, 1 = alapállapotban).

Arduino kód:

```
/*
 Input Pullup Serial

 created 14 March 2012
 by Scott Fitzgerald
 */

void setup(){
   //soros kommunikáció elindítása
   Serial.begin(9600);

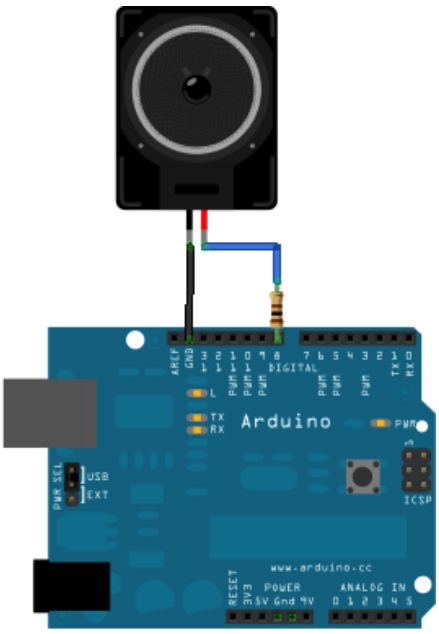
   //2 pint bemenetként használjuk,
   // bekapcsoljuk a belső felhúzó ellenállást
   pinMode(2, INPUT_PULLUP);
   pinMode(13, OUTPUT);
}

void loop(){
   //a nyomógomb kiolvasott értékét egy változóban tároljuk
   int sensorVal = digitalRead(2);
   //és kiírjuk serial monitoron is
   Serial.println(sensorVal);

   /* Tartsuk szem előtt, hogy a belső felhúzó ellenállás használata miatt a
   nyomógomb logikája megfordul, tehát HIGH lesz a jel, ha alapállapotban,
   tehát nyitva van, és LOW, ha be van nyomva. Kapcsoljuk be a LED-et akkor, ha
   be van nyomva a gomb, és ki, ha nincs. */
   if (sensorVal == HIGH) {
       digitalWrite(13, LOW);
   } else {
       digitalWrite(13, HIGH);
   }
}
```

7.6. Dallam lejátszása tone() függvénnyel

Bekötése: A 8 ohmos kis hangszóró (0,25 wattos elég) pozitív oldala a 100 ohmos ellenálláson keresztül a digitális 8 pinbe, a negatív szára a földelésbe (GND) megy.



7.6.1. ábra: Hangszóró bekötése

Szükségünk van a „pitches.h” file-ra, melyhez a következő kódot egy alap szövegszerkesztőbe be kell illeszteni és elmenteni a file-t ezen a néven. A .h a *header file* kiterjesztése, ez tartalmazza a függvény meghatározásokat és definíciókat a *library* számára.

Az *arduino* mappa *libraries* mappájában hozzuk létre a *Melody* nevű mappát és tegyük bele a most készített *pitches.h* file-t – csak így fog működni. Az Arduino program újraindítása után bekerül a Sketch/Import Library legördülő menübe a Melody, innen fogjuk használni. Nyissunk egy új sketch-t, és másoljuk be az Arduino kódot (lejjebb található). Az #include „pitches.h” sort kitörölhetjük, majd vegyük elő az előbb említett legördülő menüből a melody-t, így megjelenik a következő sor: #include <pitches.h>. Most már működik a kód.

A pitches.h file kódja:

```
/*************************************************
* Public Constants
*************************************************/
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
```

 © Harsányi Réka, Juhász Márton András

```
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
```

```
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
```

 © Harsányi Réka, Juhász Márton András

```
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Arduino kód:

```
/*
  Melody / Dallam lejátszása
 (8 ohmos hangszóró a 8 pinbe kötve)

 created 21 Jan 2010
 modified 30 Aug 2011 by Tom Igoe
 */


/////sketch/Import Library-ból szedjük elő:

#include "pitches.h"

// a dallam hangjegyei, megszólalás sorrendjében:
// tömböt használunk
int melody[] = {
   NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};

   // hangjegyek hossza: 4 = negyed hang, 8 = nyolcad hang stb.:
int noteDurations[] = {
   4, 8, 8, 4,4,4,4,4 };

void setup() {
   // a dallam hangjegyeinek ismétlődése,
   // most 8 hangjegyet használunk, ezért:
   for (int thisNote = 0; thisNote < 8; thisNote++) {

      // a hangjegyek hosszának kiszámításához
```

```
        // osszuk el a másodpercet arányosan,
        //pl. a negyed hang = 1000 / 4, a nyolcad = 1000 / 8, stb.
        int noteDuration = 1000/noteDurations[thisNote];
        tone(8, melody[thisNote],noteDuration);

        // hogy elkülönítsük a hangjegyeket,
        // tegyünk közéjük szünetet
        // ennek időtartama +30%-nál úgy tűnik, jól működik:
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);

        // megállítja a lejátszást:
        noTone(8);
    }
}

void loop() {
  // nem ismételjük meg a dallamot.
}
```

Írjuk át a kódot tetszőleges dallamra!

Amennyiben **standard library**-ket akarunk használni a hivatalos oldalról, akkor letöltés után előbb telepíteni kell azokat. Kicsomagolás után egy **.h** és egy .**cpp** kiterjesztésű file-t találunk. Az ***arduino*** mappában lévő ***libraries*** mappába kell elmenteni, bár az már alapból tartalmazza őket. Esetleg itt létrehozhatunk egy saját ***library***-t is. Az Arduino program újraindítása után a ***Sketch/Import Library*** menüből elérhetőek és használhatóak.

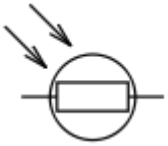
7.7. Generált hajlítás lejátszása tone() függvénnyel

Következő feladatunknál megtartjuk az előzőleg beszerelt 100 ohmos ellenállást és a 8 ohmos (0,25 wattos) kis hangszórót és bekötünk még egy fotóellenállást is egy 4,7 kiloohmos ellenállással.

Fotóellenállást eddig nem használtunk. Ez egy olyan alkatrész, amelynek vezetőképessége fény hatására – a fényelektromos hatás következtében – megnő, ezért fényérzékelőként használható. (LDR = Light Dependent Resistor) Működését tekintve minél jobban megvilágítjuk a fényérzékeny rétegét, annál kisebb lesz az ellenállása. Legnagyobb érzékenységét egy megadott fényhullámhossznál éri el, így vannak speciális fajtái, amelyek adott színekre érzékenyek, ezt spektrális érzékenységnek hívjuk. Jellemző tulajdonságai a világos-ellenállás értéke, melyet megvilágítással lehet elérni: 1000 luxhoz

 © Harsányi Réka, Juhász Márton András

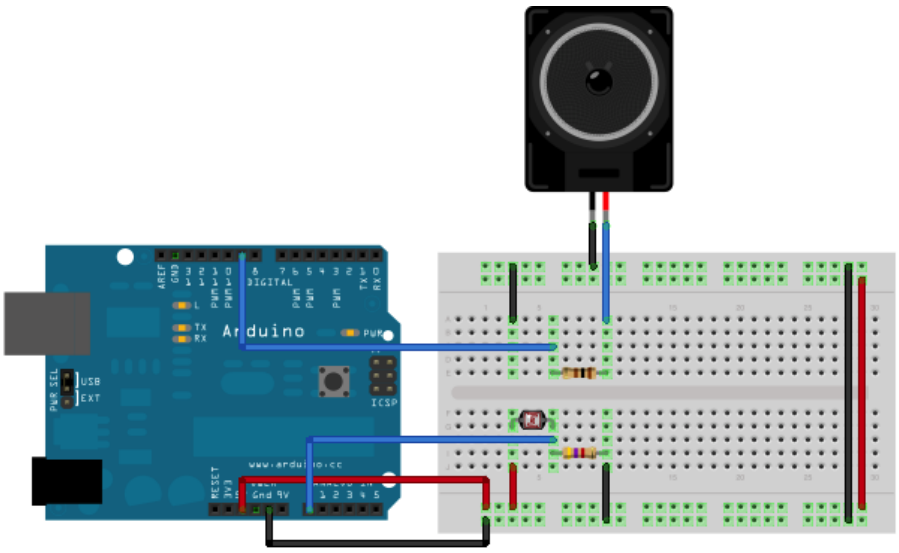
számolják és 100 ohmtól 2 kiloohmig terjedhet a skála; a másik a sötét-ellenállás érték, ami megvilágítás nélküli állapotában jellemző, ez megaohm nagyságrendű.

Fotóellenállás rajzjele: 



7.7. ábra: Fotóellenállás (LDR)

Bekötése: A hangszóró pozitív szára az ellenálláson keresztül a 9 pinbe, negatív oldala a földelésbe kerül bekötésre. A fotóellenállás egyik lábát (mindegy, milyen irányba kötjük be) az 5 V-ba, a másikat az analóg 0 pinbe és a 4,7 kiloohmos ellenálláson át a földelésbe is bekötjük.



7.7.1. ábra: Hangszóró bekötése fotóellenállással

Az analóg szenzorból kiolvasott értéket követi le majd a generált hangunk, ezáltal egy hajlítás jön létre. A kód nagyon egyszerű, a kiolvasott analóg értékeket szétosztja a hallható hangok értékéhez arányosítva (ebben az esetben leszűkítjük ezt a tartományt 120–1500 Hz közé). Az analóg szenzor által használt tartományt is beállítjuk 0–1023 helyett most 400–1000 közé, hogy jobban működjön a kód. A map() utasításban lévő paraméterek átállításával kísérletezhetünk.

Arduino kód:

```
/*
 Pitch follower
 Analóg szenzorból kiolvasott értékhez rendeli a hangszórón lejátszott hajlítás
hangjait.

 created 21 Jan 2010
 modified 31 May 2012
 by Tom Igoe, with suggestion from Michael Flynn
 */

void setup() {
   // soros kommunikációt inicializáljuk,
   // hogy a változást követhessük:
   Serial.begin(9600);
}

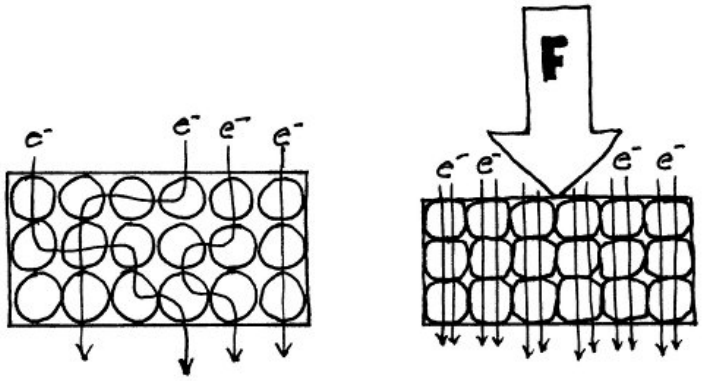
void loop() {
   // a szenzort kiolvassuk:
   int sensorReading = analogRead(A0);
   // kiírjuk, hogy megtudjuk az értékeit:
   Serial.println(sensorReading);
   // szétosztjuk az analógból érkező értéktartományt
   // (ebben az esetben 400-1000 érkezik a fotóellenállásból)
   // a kimeneten a hajlítás értéktartománya (120-1500 Hz)
   // állítsuk be, vagy változtassuk meg ezeket
   // az értékeket a szenzorunknak megfelelően, ha kell
   int thisPitch = map(sensorReading, 400, 1000, 120, 1500);

   // játsszuk le a hajlítást:
   tone(9, thisPitch, 10);
   delay(1);
   // késleltetés a kiolvasásban, a stabilitás miatt
}
```

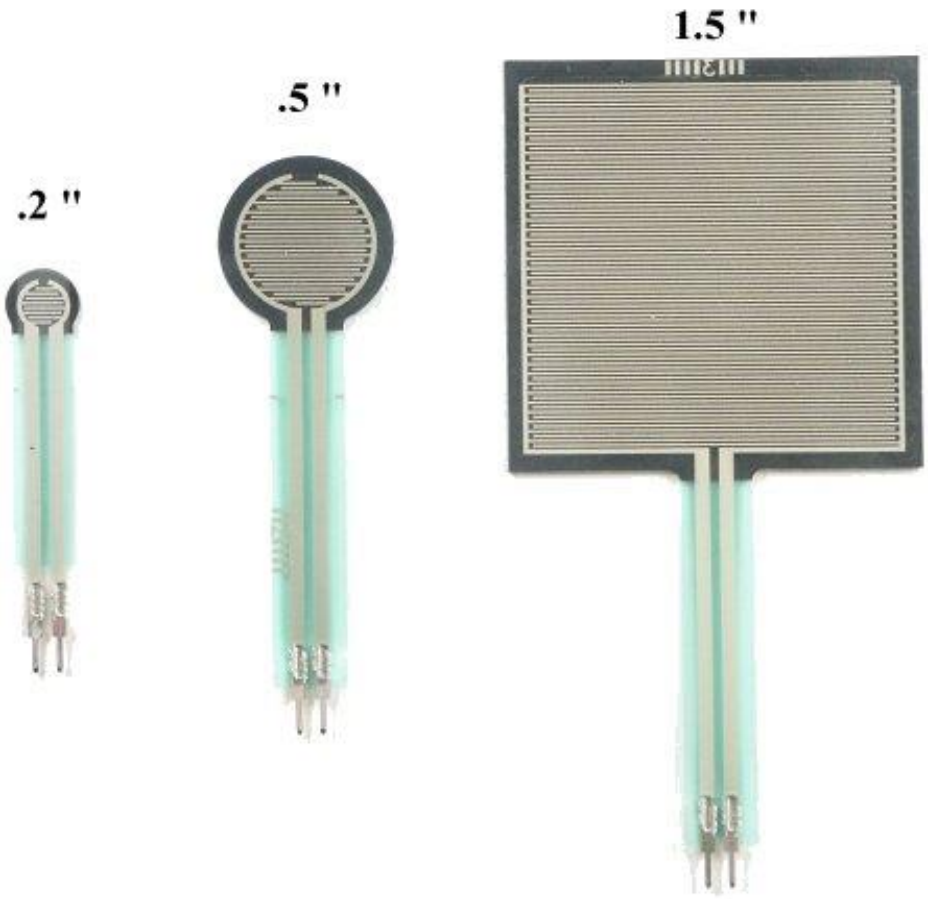
7.8. Analóg billentyűzet tone() függvénnyel

A következő feladathoz szükségünk lesz a 8 ohmos hangszórónkra a 100 ohmos ellenállással és három erőkifejtés-érzékelő ellenállásra, valamint hozzájuk három 10 kiloohmos ellenállásra.

Az FSR = Forse Sensitive Resistor, azaz erőkifejtés-érzékelő ellenállás olyan alkatrész, amelyben összenyomás hatására megváltozik az ellenállás: minél erősebben nyomjuk össze, annál kisebb lesz, tehát annál több áram jut át rajta.

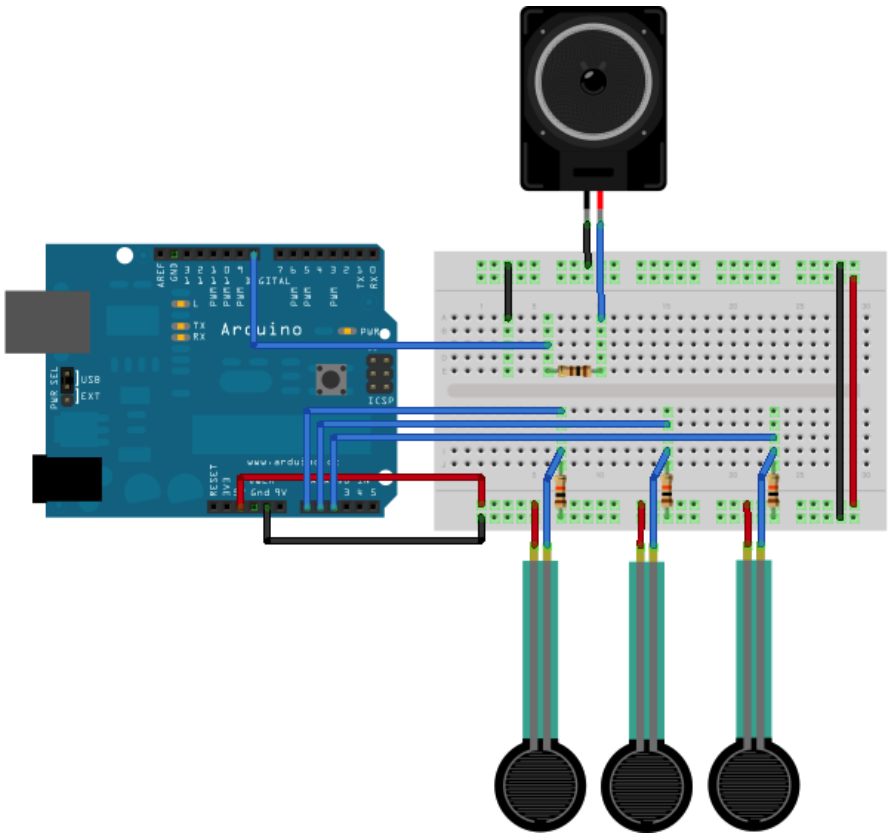


7.8.1. ábra: Elektronok útja az FSR-ben.
Bal oldalon az alap, jobb oldalon az összenyomott állapot látható.



7.8.2. ábra: Több méretben, formában kapható

Bekötése: A hangszóró negatív lába a földelésbe, a pozitív a 100 ohmos ellenálláson keresztül a 8 pinbe kerül bekötésre. A nyomásérzékeny ellenállásokat párhuzamosan bekötjük az 5 V-ba, másik lábukat pedig az analóg 0, 1, 2 pinbe, valamint a 10 kiloohmos ellenállásokon át a földelésbe is a 7.8.3 számú ábra szerint.



7.8.3. ábra: analóg billentyűzet bekötése

Arduino kód:

A kód kiolvassa a szenzorok értékeit – mindegyik egy hangnak felel meg a hangértékeket tároló tömbben. Ha a szenzorok értékének bármelyike a megadott küszöbérték felett van, akkor a hozzárendelt hang megszólal.

Ugyanazt a pitches.h fájlt használjuk, mint a dallam lejátszáskor. (Létrehozása a 7.6 alfejezetben található.) Ez tartalmazza a szokásos hangjegyek Hz-ben definiált generált hangját, pl. NOTE_C4 a közepes C hang.

```
/*
 keyboard
 Analóg bemeneten alapuló interakció hangok lejátszására, három nyomásérzékeny
ellenállás használatával.

 created 21 Jan 2010
 modified 9 Apr 2012 by Tom Igoe
 */

#include "pitches.h"
```

```
const int threshold = 10;
// küszöbérték, a minimum, ami felett megszólal a hang

// hangjegyek, amiket hozzárendelünk a szenzorokhoz
int notes[] = {
   NOTE_A4, NOTE_B4,NOTE_C3 };

void setup() {
}

void loop() {
   for (int thisSensor = 0; thisSensor < 3; thisSensor++) {
      // kiolvassuk a szenzorból az értéket:
      int sensorReading = analogRead(thisSensor);

      // ha elég erősen meg van nyomva a szenzor,
      // azaz a beolvasott érték nagyobb a küszöbértéknél:
      if (sensorReading > threshold) {
      // akkor a szenzorhoz rendelt hang megszólal:
      tone(8, notes[thisSensor], 20);
      }
   }
}
```

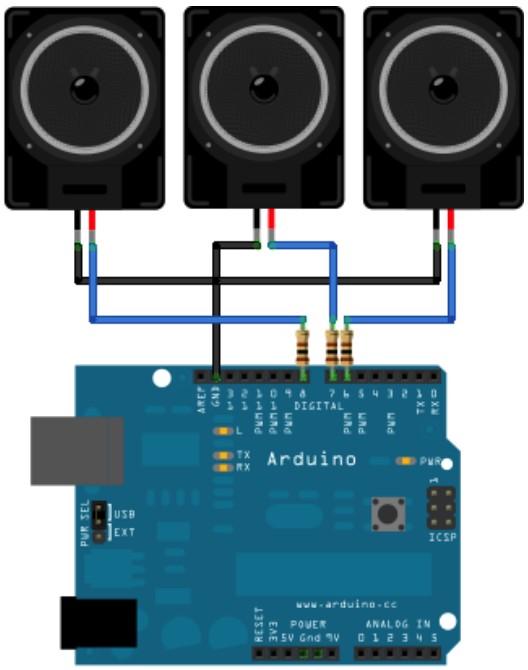
7.9. Hang lejátszása több kimeneten tone() függvénnyel

Ez a példa megmutatja, hogyan játsszunk le különböző hangokat három hangszórón, több digitális kimenet használatával. A tone() utasítás az Atmega egyik belső időzítőjét használja, beállítja azt az általunk megadott frekvenciára és az időzítővel rezegteti a kimeneti pint. Mivel egy időzítőt használ, csak egyesével tudjuk lejátszani a hangokat, azonban több pinen egymást követően több hangot is lejátszhatunk. Ehhez ki kell kapcsolni az időzítőt az adott pinen, mielőtt a következő pinnel használni kezdjük.

A kód sorban minden egyes hangszórón egy hangot játszik le, úgy, hogy ez előző hangszórót előbb kikapcsolja – a hangok hosszát ebben az esetben a késleltetés idejével szabályozzuk.

Alkatrészek bekötése: A 100 ohmos ellenállásokon át a digitális 6, 7, 8 pinbe kötjük a hangszórók pozitív oldalát, a negatívokat pedig a földelésbe.

 © Harsányi Réka, Juhász Márton András



7.9.1. ábra: Több hangszóró bekötése

Arduino kód:

```
/*
  Multiple tone player
 Több hangot egymás után több hangszórón játszik le.
 * áramkör: 3 db 8 ohmos hangszóró a digitális 6, 7, 8 pinben

 created 8 March 2010 by Tom Igoe
 based on a snippet from Greg Borenstein
 */

void setup() {
}

void loop() {
   // tone függvény kikapcsolása a 8 pinen:
   noTone(8);
   // hang lejátszása a 6 pinen 200 milliszekundumig:
   tone(6, 440, 200);
   delay(200);

   // tone függvény kikapcsolása a 6 pinen:
   noTone(6);
   // hang lejátszása a 7 pinen 500 milliszekundumig:
   tone(7, 494, 500);
   delay(500);
```

```
    // tone függvény kikapcsolása a 7 pinen:
    noTone(7);
    // hang lejátszása a 8 pinen 500 milliszekundumig:
    tone(8, 523, 300);
    delay(300);
}
```

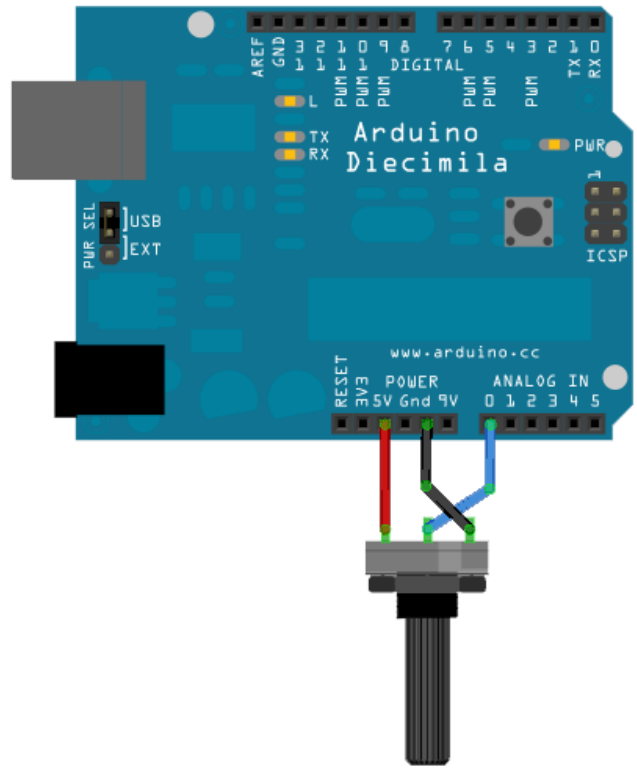
8. Példák analóg pinekhez

8.1. Analóg bemenet kiolvasása

A potméter egy változtatható ellenállású gomb, amit most analóg szenzorként értelmezünk, így analóg pinbe kötve ki tudjuk olvasni az értékeit. A kódban ezzel a kiolvasott értékkel a beépített LED-ünk villogási sebességét változtatjuk.

Az **analogRead()** parancs a beérkező 0–5 V közötti feszültségértéket átalakítja egy 0–1023 közötti digitális értékké. Ezt az Arduino egyik belső áramköre végzi el, amit **ADC**-nek hívunk (**Analog-to-Digital Converter**).

Bekötése: Ha a potméter (pl. 1 kiloohmos lineáris) mindhárom lába előre mutat, akkor a bal szélsőt kössük az 5 V pinbe, a jobb szélsőt a földelésbe, a középsőt pedig az analóg 0-ba. Az alaplapba épített LED-et használjuk, ami a 13 pinre ki van vezetve így opcionálisan beköthető ide egy 5mm-es LED is.



8.1. ábra: Potméter bekötése

Arduino kód:

```
/*
 Analog Input
 Ez a kód bemutatja, hogyan olvassunk ki analóg 0 pinből értéket, és azzal
hogyan vezéreljük a beépítet LED-ünk villogását.

Áramkör:
 * Potentiométer középső lába analóg 0-ba kötve, két oldalsó a földbe és 5 V-ba
 * LED (elhagyható!) pozitív lába a digitális 13-ba, negatív a földbe

 Created by David Cuartielles
 modified 30 Aug 2011 By Tom Igoe
 */

int sensorPin = A0;    // potméter középső lába
int ledPin = 13;      // beépített LED pinje
int sensorValue = 0;  // változó, a beolvasott érték tárolására

void setup() {
   // megjelöljük a LED pint kimenetként
   pinMode(ledPin, OUTPUT);
}

void loop() {
   // kiolvassuk az értéket a szenzorból,
   // 0-1023 közötti értéket kapunk:
   sensorValue = analogRead(sensorPin);

   // LED pint bekapcsoljuk:
   digitalWrite(ledPin, HIGH);

   // „sensorValue” értékű milliszekundummal
   // megállítjuk a program lefutását:
   delay(sensorValue);

   // a ledPin-t kikapcsoljukf:
   digitalWrite(ledPin, LOW);

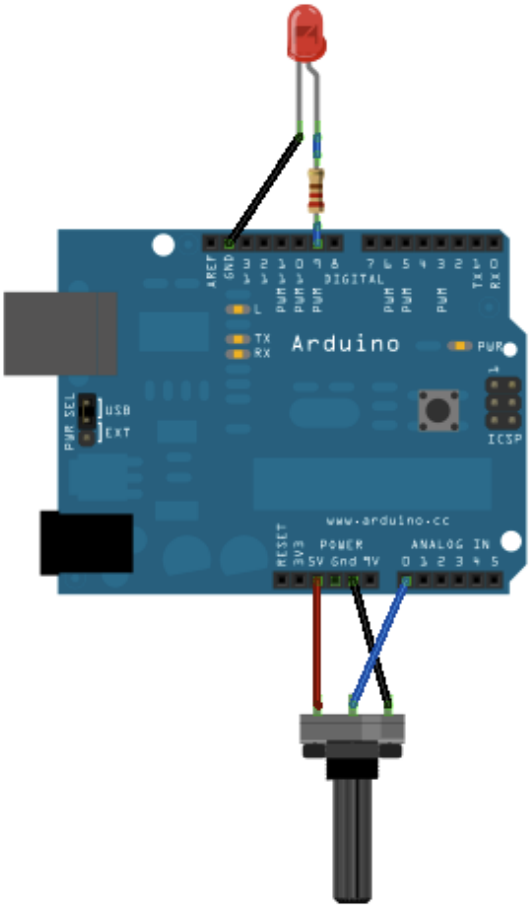
   // „sensorValue” értékű milliszekundummal
   // megállítjuk a program lefutását:
   delay(sensorValue);
}
```

8.2. Analóg bemenettel vezérelt PWM

Ezúttal kiolvasunk egy analóg pint és a kapott értéket 0–255 közötti digitális értékké alakítjuk, skálázzuk **map()** utasítással és ráküldjük egy PWM pinre, amivel egy LED

 © Harsányi Réka, Juhász Márton András

fényerejét fogjuk vezérelni. Ehhez szükségünk lesz egy potméterre (mondjuk 1 kiloohmos lineáris fajtára), egy 220 ohmos ellenállásra és egy hagyományos piros LED-re.



8.2. ábra: LED és potméter bekötése

Bekötése: Ha a potméter mindhárom lába előre mutat, akkor a bal szélsőt kössük az 5 V pinbe, a jobb szélsőt a földelésbe, a középsőt pedig az analóg 0-ba. A LED rövidebb, negatív lába a szokásos módon a földelésbe, a hosszabb, pozitív a 220 ohmos ellenálláson át a digitális 9 pinbe kerül.

A kódban először megjelöljük, melyik pint miként akarjuk használni. Ezután létrehozunk két változót, az egyikbe az analóg pinből **analogRead()**-del kiolvasott érték kerül, a másikba ennek az értéknek az arányosított megfelelője, amit majd a LED-hez fogunk kiküldeni **analogWrite()**-tal. Mivel a feladathoz PWM pint használunk, az impulzusszélesség modulációval különböző erősséggel tud világítani a LED-ünk. A PWM pinre 0–255 közötti értéket küldünk, ehhez az analóg szenzorból kiolvasott 0–1023 közötti értéket **map()** paranccsal szétosztjuk.

Arduino kód:

```
/*
 Analóg input, analóg output, serial output
 Az analóg inputból kiolvasott értéket 0-255 közé skálázza, az eredményt PWM-re
küldi és kiírja serial monitoron is

Áramkör:
 * potméter középső lába bekötve analóg 0 pinbe, a szélsők a földbe és az 5V-ba
 * LED digitális 9-be és a földbe

 created 29 Dec. 2008
 modified 9 Apr 2012
 by Tom Igoe
 */

// Állandókkal jelöljük meg a pineket:
// potméter - bemenet az analóg pin 0-n:
const int analogInPin = A0;
// LED - kimenet a digitális 9-en
const int analogOutPin = 9;

// létrehozzuk a változókat
int sensorValue = 0;        // potméterből kiolvasott érték
int outputValue = 0;        // PWM-re küldött érték

void setup() {
   // soros kommunikáció inicializálása:
   Serial.begin(9600);
}

void loop() {
   // kiolvassuk az analóg bemenetet:
   sensorValue = analogRead(analogInPin);
   // PWM-hez skálázzuk az értéket:
   outputValue = map(sensorValue, 0, 1023, 0, 255);
   // és kiküldjük rá:
   analogWrite(analogOutPin, outputValue);

   // serial monitoron kiírjuk az eredményt:
   Serial.print("sensor = " );
   Serial.print(sensorValue);
   Serial.print("\t output = ");
   Serial.println(outputValue);

   // 2 milliszekundumot várunk, mielőtt újra lefut a loop,
   // hogy az analóg-digitális átalakító stabilan működjön
   delay(2);
}
```

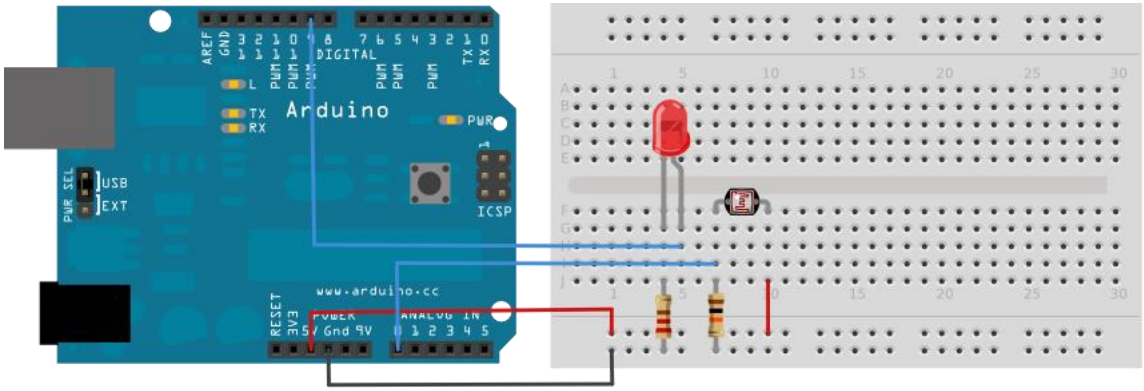
 © Harsányi Réka, Juhász Márton András

8.3. Analóg szenzor kalibrálása

A következőkben megvizsgáljuk a szenzorok kalibrálásának menetét. A folyamat során az Arduino 5 másodpercig megszakítás nélkül olvassa ki az értéket. A program végrehajtása közben a szenzorból kiolvasott adatok meghatározzák a minimum és maximum értékeket.

Az analóg 0 pinbe bármilyen szenzort beköthetünk, potmétert vagy fotóellenállást is; a példában az utóbbit használjuk.

Bekötése: A LED-ünk negatív lábát a szokásos 220 ohmos ellenálláson keresztül bekötjük a földelésbe, pozitív oldalát pedig a 9 pinbe (bármelyik lábához kerülhet az ellenállás, eddig a pozitívhoz tettük), a fotóellenállás (LDR) bármilyen irányban állhat, egyik lába az 5 V-ba, a másik lába az analóg nulla (A0) pinbe és egy 10 kiloohmos ellenálláson át a földelésbe is be van kötve.



8.3. ábra: LED és fotóellenállás bekötése

A kód legelején inicializálnunk kell a két változót, amelyben a minimum és maximum értéket fogjuk eltárolni. Analóg pinről lévén szó, a minimum 1023 és a maximum 0 legyen – most kivételesen fordítva használjuk. A legelején meghatároztuk a maximum értéket (0) és ha a kiolvasott érték a felett van, akkor ezt mentjük majd el, mint új maximum értéket. Ugyanígy, ha a kiolvasott érték az elején beállított minimum érték (1023) alatt van, akkor az lesz az új minimum érték. Ezután ezt a két értéket skálázzuk 0–255 közé, ez fogja vezérelni a PWM-re kötött LED-ünk fényerejét.

Megjegyzés: Ha potmétert használunk, ne tekerjük el a kalibrálás alatt teljesen ütközésig, így köztes értéket tudunk használni, hogy megértsük a kód működését.

Arduino kód:

```
/*
  Kalibráció
  Szenzorok kalibrálása: 5 másodpercig folyamatosan olvassa ki az értéket az
Arduino. A program végrehajtása közben a szenzorból kiolvasott adatok
meghatározzák a minimum és maximum értékeket.

 Áramkör:
 * Analóg szenzor analóg 0 bementi pinbe és földbe
 * LED a digitális 9 pinbe és földbe

 created 29 Oct 2008 by David A Mellis
 modified 30 Aug 2011 by Tom Igoe
 */

// állandókkal meghatározzuk a pineket:
const int sensorPin = A0;    // fotóellenállás pinje
const int ledPin = 9;        // LED pinje

// Változók:
int sensorValue = 0;     // szenzor aktuális értékét tárolja
int sensorMin = 1023;        // minimum szenzorérték
int sensorMax = 0;           // maximum szenzorérték


void setup() {
   // a LED-et bekapcsolja a kalibrálási idő elején:
      pinMode(13, OUTPUT);
      digitalWrite(13, HIGH);

      // a kalibrálás 5 másodpercig tart:
      while (millis() < 5000) {
            sensorValue = analogRead(sensorPin);

         // maximum szenzorérték beállítása
      if (sensorValue > sensorMax) {
            sensorMax = sensorValue;
      }

      // minimum szenzorérték beállítása
      if (sensorValue < sensorMin) {
            sensorMin = sensorValue;
      }
   }

   // kalibrációs idő végén kikapcsolja a LED-et:
   digitalWrite(13, LOW);
}

void loop() {
```

```
    // szenzor kiolvasása:
    sensorValue = analogRead(sensorPin);

    // a kiolvasott analóg értéket átfordítjuk digitálissá:
    sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 255);

    // ha a kiolvasott érték a kalibrációs értékhatárokon
    // kívülre esik, akkor limitáljuk:
    sensorValue = constrain(sensorValue, 0, 255);

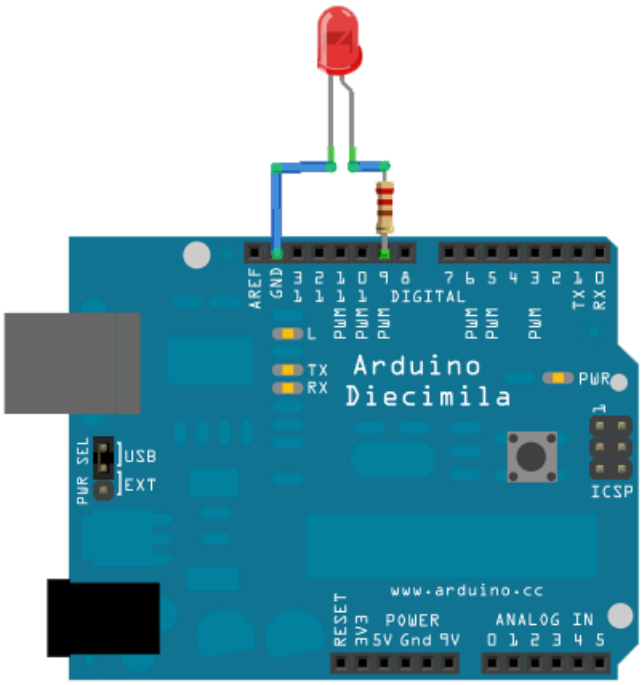
    // beállítjuk a LED fényerejét ezzel az értékkel:
    analogWrite(ledPin, sensorValue);
}
```

Előfordulhat, hogy a kalibráció alatt nem jelennek meg a szélsőértékek, így a map()nem valós értéket állíthat elő (pl. a maximumot felveszi 800-nak az elején, később pedig beolvas 820-at, akkor a map() eredménye már 256), ezért a constrain()-nel beszorítja a kilengéseket a megadott intervallumba.

8.4. LED fényerejének szabályozása PWM-mel

PWM = Pulse Width Modulation, azaz impulzusszélesség moduláció. Egy olyan digitális jel, ami rendkívül gyorsan kapcsolja ki-be a digitális jelet egy meghatározott ütemben, ezáltal felruházva a digitális pinünket analóg-szerű viselkedéssel.

Bekötése: 220 ohmos ellenálláson át bekötjük a LED pozitív lábát a digitális 9 pinbe, a negatívat pedig a földelésbe.



8.4. ábra: LED bekötése digitális 9 pinbe

Arduino kód:

```
/*
 Fading
 LED fényerejének szabályozása analogWrite() függvénnyel.

 Áramkör:
 * LED a digitális 9 PWM pinbe és a földelésbe

 Created 1 Nov 2008 by David A. Mellis
 modified 30 Aug 2011 by Tom Igoe
 */


int ledPin = 9;    // LED digitális 9 pinben

void setup()  {
   // itt semmi sem történik
}
```

```
void loop()  {
   // fényerő növelése minimumtól maximum értékig,
   // ötösével lépkedve:
   for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
      // ezt az értéket kiküldjük a LED pinre:
      analogWrite(ledPin, fadeValue);
      // 30 milliszekundumot várunk, hogy szemmel követhető
      // legyen a fényerő változás:
      delay(30);
   }

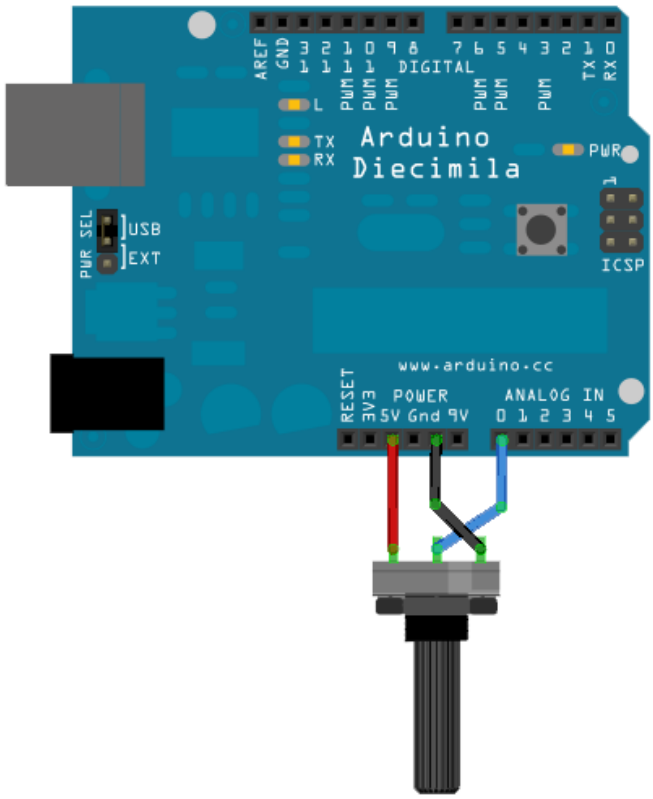
   // fényerő csökkentése maximumtól minimum értékig,
   // ötösével lépkedve:

   for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
       // ezt az értéket kiküldjük a LED pinre:
      analogWrite(ledPin, fadeValue);
      // 30 milliszekundumot várunk, hogy szemmel követhető
      // legyen a fényerő változás:
      delay(30);
   }
}
   --
```

8.5. Simítás

Ez a kód folyamatosan olvassa ki az analóg bemenetet és az értékek átlagát írja ki a számítógépre. Hasznos lehet egyenetlen szenzorok kiolvasásánál, mivel elsimítja az értékek ingadozását. Egyúttal azt is megnézzük, hogyan lehet tömböt használni adattárolásra.

Bekötése: Potmétert használunk, középső lábát analóg nullába kötjük, a két oldalsót pedig az 5 V feszültségbe és a földelésbe.



8.5. ábra: Potméter bekötése A0-ba

A kód egymást követő 10 kiolvasást tárol el egy tömbben, egyesével. Ezeket összeadja, majd elosztja, így egy átlagértéket ad. Mivel a beérkező értékekkel folyamatosan kalkulál, és nem várja meg mind a 10 értéket, ezért nincs késési idő. A numReadings nevű változó értékének lecserélésével tudunk kísérletezni.

Arduino kód:

```
/*
  Smoothing
  Folyamatosan kiolvassa az analóg bemenetet és a tömbben  eltárolt értékeket
folyamatosan átlagolja.

 Áramkör:
    * Analóg szenzor az analóg 0 pinbe.

  Created 22 April 2007
  By David A. Mellis  <dam@mellis.org>
  modified 9 Apr 2012 by Tom Igoe
*/
```

```
//  Meghatározzuk, hány számot akarunk átlagolni.
// Nagyobb számnál simább lesz a kimenet,
// kisebbnél pedig a kimenet gyorsabban reagál.
// Integer állandót használunk a tömb elemeinek
// számának meghatározására.

const int numReadings = 10;

// analóg bemenetet kiolvasó integer változó tömb:
int readings[numReadings];
// aktuális kiolvasás értékét tároló integer változó:
int index = 0;

int total = 0;                  // összes lefutás
int average = 0;                // az átlag

int inputPin = A0;

void setup() {
   // inicializáljuk a soros kommunikációt:
   Serial.begin(9600);
   // inicializáljuk a kiolvasást minden egyes kiolvasáshoz:
   for (int thisReading = 0; thisReading < numReadings; thisReading++)
      readings[thisReading] = 0;
}

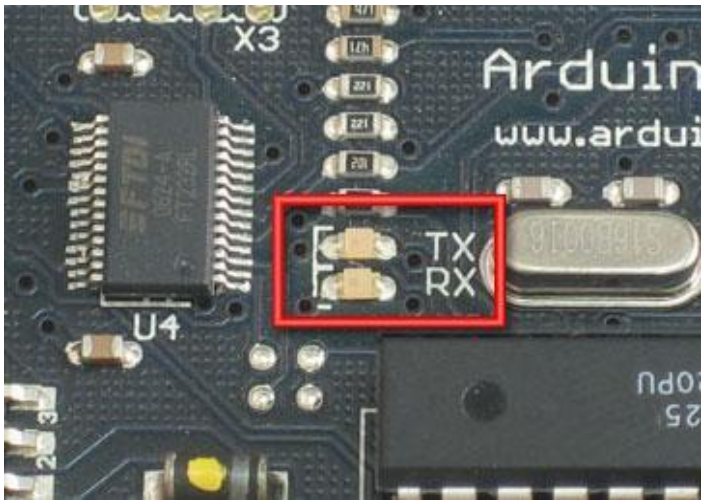
void loop() {
   // levonjuk az utolsó kiolvasást:
   total= total - readings[index];
   // amit a szenzorból olvastunk ki:
   readings[index] = analogRead(inputPin);
   // a kiolvasott értéket hozzáadjuk az egészhez:
   total= total + readings[index];
   // továbblépünk a tömb következő eleméhez:
   index = index + 1;

   // ha a tömb végére érünk...
   if (index >= numReadings)
   // ...visszaugrunk a kezdő értékhez:
      index = 0;

   // kiszámoljuk az átlagot:
   average = total / numReadings;
   // a számítógépre küldjük ASCII számjegyként
   Serial.println(average);
   delay(1);
   // szünet a kiolvasások között a stabilitás miatt
}
```

9. Soros kommunikáció

Az Arduino mikrovezérlőjén (AVR ATmega) van két olyan I/O csatorna (D0 és D1), amelyeknek megkülönböztetett funkciójuk van – a két kivezetést együtt *soros port*nak nevezzük, amin keresztül adatokat cserélhetünk a környezet és a mikrovezérlő között. Az egyik csatorna az adatok fogadásáért (Rx – receiver), a másik pedig küldéséért felel (Tx – transmitter). A soros átvitel szekvenciálisan zajlik, vagyis egy időben egyetlen bit halad át a csatornán. Ennek megfelelően a kommunikáció sebességét a másodpercenként átvitt bitek számával mérjük és BPS-sel (Bits Per Second = Bit per másodperc) jelöljük. Az Arduino maximális átviteli sebessége 115200 bit másodpercenként.



9.1. ábra: A soros átvitelt jelző LED-ek és az USB átalakító

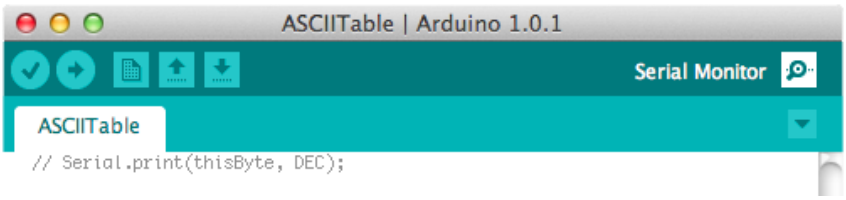
Ahhoz, hogy számítógépünk könnyedén kapcsolódhasson a mikrovezérlőhöz, az Arduino tartalmaz egy soros USB átalakítót (FTDI), mely a fent említett csatornákhoz kapcsolódik és létrehozza a kapcsolatot az univerzális soros busszal (USB). Az adatok ugyan bitenként, de egymástól jól elhatárolható csomagokban közlekednek. Ezek a blokkok START jelből, 8 bit adatból (1 byte) és két STOP jelből állnak.

Az ASCII, vagyis az *American Standard Code for Information Interchange* 1963 óta szabványa a szöveg típusú adatok digitális tárolásának és átvitelének. Az általános kódtáblázat 128 karakterből áll, melynek alapját az angol ABC kis- és nagybetűi, írásjelek, számok és matematikai kifejezések képezik. Ezeken felül 33 grafikusan nem megjeleníthető, úgynevezett vezérlőkarakter is helyet kapott, mint a „soremelés”, „fájl vége”, „fejléc kezdete”, melyek egy része ma már nem használatos. A kódtáblát legegyszerűbben egy számozott sorozatként képzelhetjük el, ahol a nulladik helytől a 31-ig a vezérlőkarakterek, 32-től 126-ig a megjeleníthető karakterek találhatók, a sort pedig a „törlés” nevű vezérlőkarakter zárja. Soros kommunikáció esetén az adat értéke egyszerűen a táblázat egyik elemének sorszámát jelöli.

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

9.1. táblázat – Az ASCII kódtábla

Példánkon keresztül megérthetjük, hogyan értelmezi mindezt az Arduino, és megismerjük a kiterjesztett soros nyomtatási funkciókat. Nyomtassuk ki a soros terminálunkra az ASCII táblázat grafikus elemeit és a hozzájuk tartozó értékeket decimális, hexadecimális, oktális és bináris formátumokban!



9.2. ábra: A soros terminált a keretprogram jobb oldalán található ikonra kattintva érhetjük el.

Példakód

Csak egy számítógéphez kötött Arduinora lesz szükségünk.

```
/*
 ASCII táblázat

 Minden lehetséges formában kiírja a byte értékeket:
 * nyers bináris értékként
 * ASCII kódolású decimális, hexadecimális, oktális és bináris értékekként

 Bővebben az ASCII táblázatról:
 http://hu.wikipedia.org/wiki/ASCII
 http://hu.wikisource.org/wiki/Seg%C3%ADts%C3%A9g:ASCII

 created 2006 by Nicholas Zambetti  <http://www.zambetti.com>
 modified 9 Apr 2012 by Tom Igoe
 */

void setup() {
     //Soros port megnyitása 9600 bit/másodperc sebességgel.
     Serial.begin(9600);
     while (!Serial) {
       ; // addig várunk, amíg a soros port megnyílik (csak Leonardonál
szükséges)
     }

     // port megnyitása után kiírjuk a fejlécet.
     Serial.println("ASCII Table ~ Character Map");
}

// az első grafikus karakter a '!', aminek decimális értéke 33.
int thisByte = 33;
// ASCII karaktereket is megadhatunk két aposztróf között
// mivel a kódtáblázat alapján automatikusan számként képződnek le
// int thisByte = '!';
```

 © Harsányi Réka, Juhász Márton András

```
void loop() {

   Serial.write(thisByte);
   // változtatás nélkül, vagyis binárisan nyomtatja ki a byte-ot
   // A soros monitor automatikusan ASCII-ként fogja értelmezni,
   // ezért a kiküldött 33 (100001) '!'-ként jelenik majd meg

     Serial.print(", dec: ");
     Serial.print(thisByte);
     // A byte decimális értékét ASCII-kódokkal nyomtatja ki (33, tehát két
     // hármasnak megfelelő byte-ot küld).
     // Ez a Serial.print() és Serial.println() függvények alapbeállítása.
     // Meghívhatjuk Serial.print(thisByte, DEC);-ként is, az eredmény ugyanaz.

     Serial.print(", hex: ");
     Serial.print(thisByte, HEX);
     // A byte hexadecimális értékét ASCII-kódok soraként nyomtatja ki,
     // 21, vagyis 2-esnek majd egy 1-esnek megfelelő byte-ot küld.

     Serial.print(", oct: ");
     Serial.print(thisByte, OCT);
     // fentieknek megfelelően csak oktális alapon (41)

     Serial.print(", bin: ");
     Serial.println(thisByte, BIN);
     // binárisan, vagyis kettes alapon (100001).
     // figyeljük meg, hogy az utolsó függvény Serial.println a „soremelés"
     // vezérlőkaraktert is elküldi az adat után

     // az utolsó látszó vagy grafikus karakter a 126, ami a  '~'
     if(thisByte == 126) {     // egyenlőségnél is használhatjuk az ASCII-kódot
                               // (thisByte == '~') {

     // ciklus, amint elérjük az utolsó karaktert, ismétlődik a végtelenségig
       while(true) {
         continue;
       }
     }
     // amíg nem érünk a végére, emeljük a sorszám értékét eggyel - így ugrunk
a
     // következő karakterre.
     thisByte++;
   }
```

ASCII Table ~ Character Map

```
!,      dec:      33,      hex:      21,      oct:      41,      bin:      100001
",      dec:      34,      hex:      22,      oct:      42,      bin:      100010
#,      dec:      35,      hex:      23,      oct:      43,      bin:      100011
$,      dec:      36,      hex:      24,      oct:      44,      bin:      100100
%,      dec:      37,      hex:      25,      oct:      45,      bin:      100101
&,      dec:      38,      hex:      26,      oct:      46,      bin:      100110
',      dec:      39,      hex:      27,      oct:      47,      bin:      100111
(,      dec:      40,      hex:      28,      oct:      50,      bin:      101000
),      dec:      41,      hex:      29,      oct:      51,      bin:      101001
*,      dec:      42,      hex:      2A,      oct:      52,      bin:      101010
+,      dec:      43,      hex:      2B,      oct:      53,      bin:      101011
,,      dec:      44,      hex:      2C,      oct:      54,      bin:      101100
-,      dec:      45,      hex:      2D,      oct:      55,      bin:      101101
.,      dec:      46,      hex:      2E,      oct:      56,      bin:      101110
/,      dec:      47,      hex:      2F,      oct:      57,      bin:      101111
0,      dec:      48,      hex:      30,      oct:      60,      bin:      110000
1,      dec:      49,      hex:      31,      oct:      61,      bin:      110001
2,      dec:      50,      hex:      32,      oct:      62,      bin:      110010
3,      dec:      51,      hex:      33,      oct:      63,      bin:      110011

stb.
```

9.3. ábra: A program kimenete (ezt látjuk a soros terminálban)

9.1. Adatátvitel PC-ről Arduinora

Azon túl, hogy a soros terminálban képesek vagyunk adatok megjelenítésére – ami nagyon hasznos lehet az Arduino programjának írása közben hibák felderítésénél (debug), a hozzákapcsolt szenzorok érzékenységének beállításakor, a kiváltani kívánt hatások ellenőrzésekor –, használhatjuk az adatkapcsolatot arra is, hogy a számítógépünkön megírt komplex programok a fizikai környezetet az Arduinon, mint periférián keresztül érjék el.

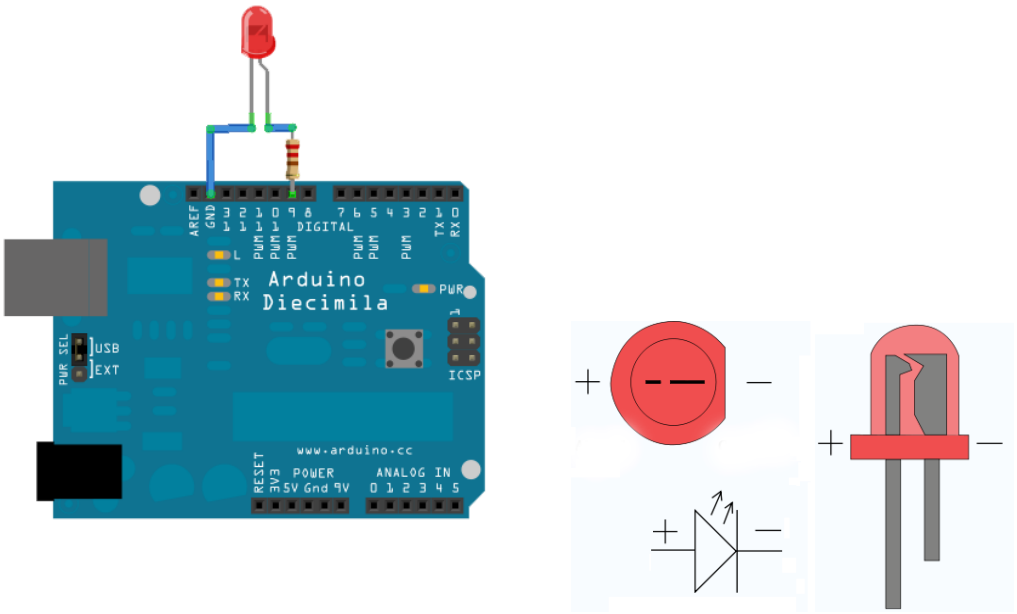
A következő példában egy Arduinohoz kapcsolt LED fényerejét fogjuk változtatni számítógépről küldött adatokkal. Ahogy a fentiekben, most is byte-okat fogunk küldeni, melyek értéke 0-tól 255-ig változhat. Ezeket beolvassuk az Arduinoval és hozzárendeljük a LED fényerejét meghatározó értékhez.

Bármilyen programot használhatunk a kommunikációhoz, ami képes gépünk soros portját megnyitni. Nézzük meg, hogyan lehetséges ez Processing és Max/MSP5 környezetekből!

Amire szükségünk lesz:

- Arduino,
- LED,
- 220 – 1 kiloohmos ellenállás,
- Processing vagy,
- Max/MSP v. 5.

Kapcsolás:



9.4. ábra: LED polaritása és bekötése az Arduinoba

Kössük a LED-ünk (10.4. ábrán) *negatívval* jelölt lábát az Arduino földjéhez (GND), a *pozitívval* jelölt lábát pedig egy soros ellenálláson keresztül a 9-es I/O csatornájához – amit most impulzusmodulációra (PWM) képes kimenetként fogunk használni.

Példakód

```
const int ledPin = 9;      // a változó, amivel a LED-et jelöljük
                           // és a hozzárendelt kimenet értékét tároljuk

void setup()
{
   // megnyitjuk a soros portot,
   Serial.begin(9600);
   // beállítjuk az Arduino 9-es csatornáját kimenetnek.
   pinMode(ledPin, OUTPUT);
```

```
}

void loop() {
   //brightness változóban fogjuk tárolni a fényerő értékét
   byte brightness;

   //nézzük meg, hogy kaptunk-e már adatot a számítógéptől
   if (Serial.available()) {
      //ha igen, legyen a brightness értéke, amit a soros portról beolvasunk
      brightness = Serial.read();
      // mivel ez az érték csak 0-255 között változhat, írhatjuk is a kimenetre
    analogWrite(ledPin, brightness);
  }
}
```

9.1.1. Soros adat küldése Processing segítségével

Hozzunk létre egy 256 pixel széles képernyőt, amin az egér aktuális vízszintes koordinátája jelenti majd az Arduinohoz kötött LED fényerejét (0–255). Vizualizáljuk az elvárt kimenetet függőleges vonalakból álló, feketéből fehérbe változó gradienssel. Küldjük ki az egér pozícióját sorosan byte-onként!

```
// by David A. Mellis

import processing.serial.*;
// használjuk a beépített soros kommunikációs könyvtárat

Serial port; // a soros portunkat nevezzük portnak

void setup() {
   size(256, 150); // legyen a képernyőnk praktikusan 256 pixel széles

   println("Elérhető soros portok:");
   println(Serial.list());
   //nyomtassuk ki az összes elérhető portot
   // A listában látni fogjuk, hogy az Arduino portja úgy jelenik meg, mint a
   // saját fejlesztőkörnyezetében.

   // A következő függvénnyel megnyitjuk a portot - adjuk meg a listában az
   // Arduinonak megfelelő sorszámot (itt 0) és azt az átviteli sebességet,
   // amit, az Arduino kódjában meghatároztunk (itt 9600).

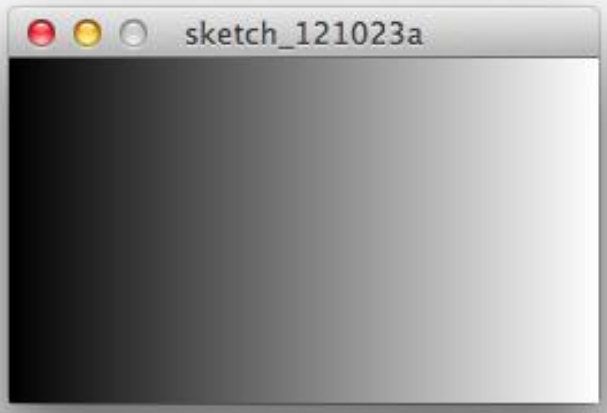
   port = new Serial(this, Serial.list()[0], 9600);

   // ha tudjuk a port nevét, akkor megadhatjuk a következő formában is
```

```
   //port = new Serial(this, "COM1", 9600);
}

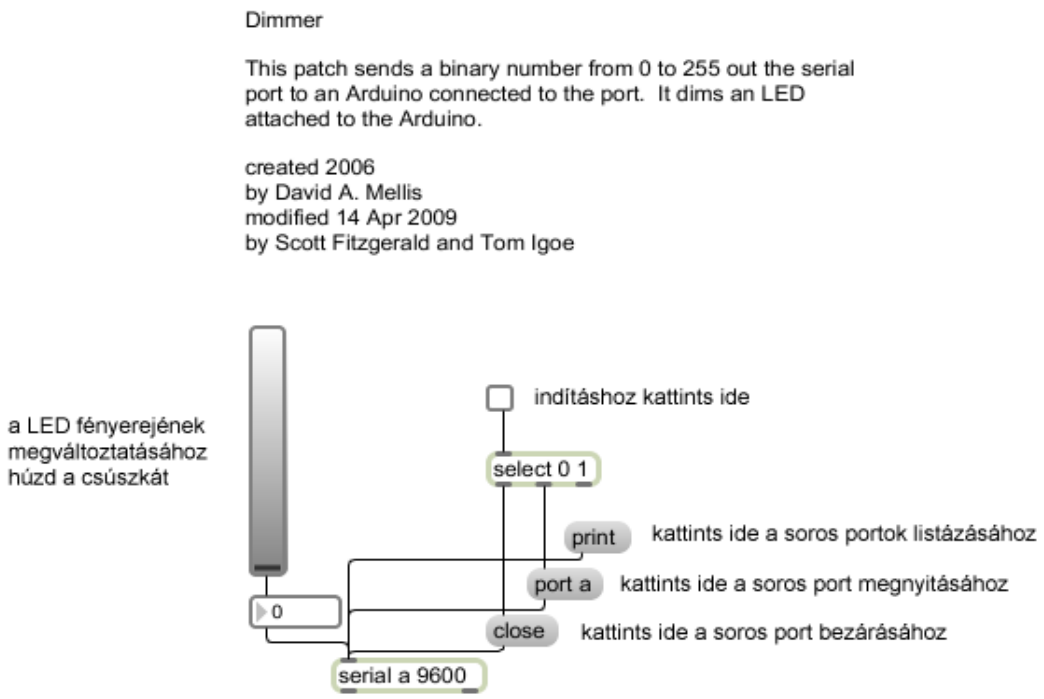
void draw() {
   // rajzoljunk egy gradienst függőleges vonalakból feketétől fehérig
   for (int i = 0; i < 256; i++) {
    stroke(i);
    line(i, 0, i, 150);
    }

   // mivel a képernyőnk 256 széles, kiírhatjuk az egér aktuális vízszintes
   // koordinátáját egy az egyben a soros portra.
   // A port.write() byte-onként küld.
   port.write(mouseX);
 }
```



9.5. ábra: Processing kód futásának eredménye

9.1.2. Soros adat küldése Max/MSP segítségével



9.6. ábra: Byte küldése Max/MSP-vel

9.1.3. Vezérlés karakterekkel

Nézzünk az előző kapcsolással még egy példát! Az Arduinohoz csatlakoztatott LED-et fogjuk ki- és bekapcsolni általunk meghatározott karakterekkel. Bármely soros kommunikációra alkalmas programból elküldhetjük a karaktereket, az Arduino fejlesztőkörnyezetének saját soros termináljából is.

Példakód

```
 /* created 2006
by David A. Mellis
modified 30 Aug 2011
by Tom Igoe and Scott Fitzgerald */


const int ledPin = 9; // a LED az Arduino 9-es lábán van
int incomingByte;      // a bejövő byte-okat tároló változónk

void setup() {
```

```
    // soros port megnyitása
    Serial.begin(9600);
    // 9-es csatorna beállítása kimenetként
    pinMode(ledPin, OUTPUT);
}

void loop() {
    // amennyiben van bejövő adat:
    if (Serial.available() > 0) {
        // olvassuk ki az utolsó byte-ot a tárolóból
        incomingByte = Serial.read();
        // ha ez egy nagy H (ASCII 72), kapcsoljuk be a LED-et
        if (incomingByte == 'H') {
            digitalWrite(ledPin, HIGH);
        }
        // ha ez egy nagy L (ASCII 76), kapcsoljuk ki a LED-et
        if (incomingByte == 'L') {
            digitalWrite(ledPin, LOW);
        }
    }
}
```

9.1.3.1. Karakterek küldése Processing segítségével

Az alábbi kód egyszerre példa meghatározott karakterek soros küldésére és egy egyszerű egérre reagáló gombra. Amikor egerünket a gombként működő négyzet fölé visszük, az Arduinohoz kapcsolt LED bekapcsolódik.

```
// based on examples by Casey Reas and Hernando Barragan

import processing.serial.*; // használjuk a beépített soros könyvtárat

float boxX; // gombunk vízszintes pozíciója
float boxY; // gombunk függőleges pozíciója
int boxSize = 20; // gomb mérete
boolean mouseOverBox = false; // „ha az egér a gomb felett van” bináris változó,
                              // alapállapota hamis

Serial port; // nevezzük soros portunkat portnak

void setup()  {
    size(200, 200);
    boxX = width/2.0; //width és height beépített változók, az aktuális
                      // képernyőméret kiterjedéseit tartalmazzák
    boxY = height/2.0;
    rectMode(RADIUS); // a négyszöget a középpontjával fogjuk pozícionálni
```

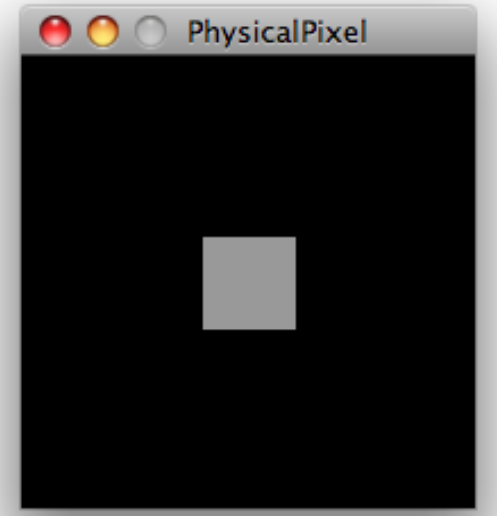
```
    // listázzuk az összes elérhető soros portot
    println(Serial.list());
    // nyissuk meg a portot, ahogy az előző példában
    port = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
    background(0);

    // megvizsgáljuk, hogy az egérkurzor a gomb felett van-e
    if (mouseX > boxX-boxSize && mouseX < boxX+boxSize &&
    mouseY > boxY-boxSize && mouseY < boxY+boxSize) {
        mouseOverBox = true;
        // ha felette van, legyen a négyzet körvonala fehér
        stroke(255);
        fill(153);
        // küldjünk a soros porton egy nagy H betűt
        port.write('H');
    } else {
    // máskülönben legyen a négyzet körvonala szürke
        stroke(153);
        fill(153);
        // és küldjünk L betűket, hogy a LED kikapcsolva legyen
        port.write('L');
        mouseOverBox = false;
    }

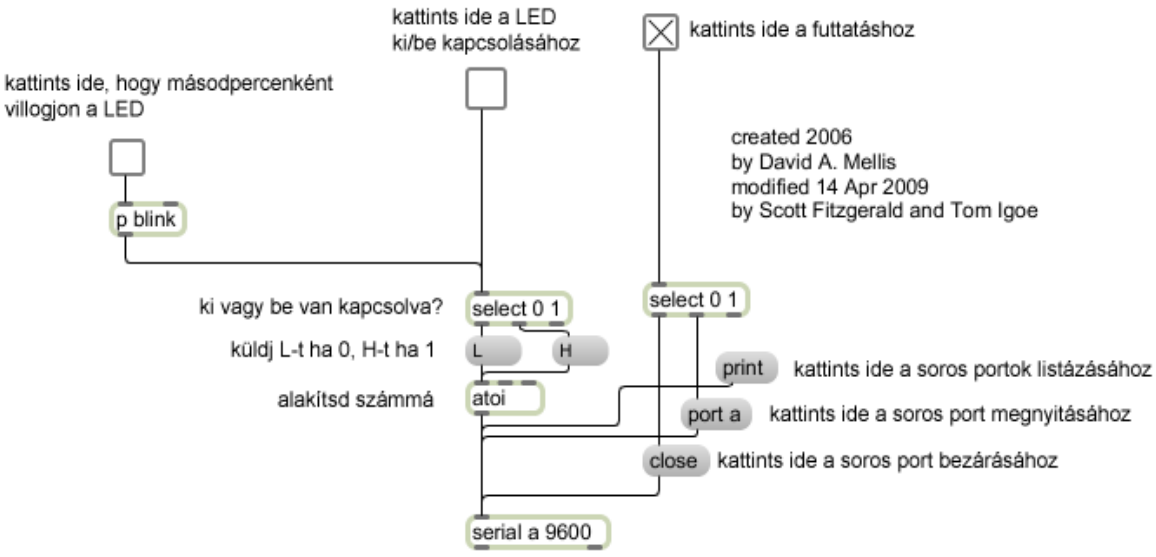
    // itt rajzoljuk a négyzetet
    rect(boxX, boxY, boxSize, boxSize);
}
```

 © Harsányi Réka, Juhász Márton András



9.7. ábra: A processing kódunk kimenete

9.1.3.2. Karakterek küldése Max/MSP segítségével



9.8. ábra: Vezérlés karakterekkel Max/MSP-vel

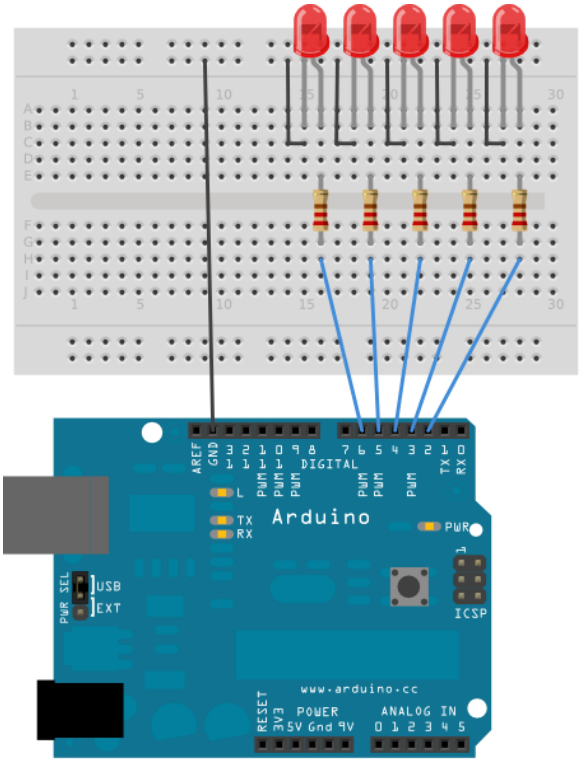
9.1.4. Switch feltétel használata soros kommunikációban

Itt még érdemes kitérnünk az Arduino nyelv egy funkciójára, amelynek segítségével diszkrét értékekhez rendelhetünk végrehajtandó kódrészeket. Ahogy az *if* esetében két – a feltételnek megfelelő és az annak nem megfelelő – állapot közül választhatunk, addig a *Switch (case)* feltétel esetében egy változó minden értékével különálló feltételt tudunk megvalósítani.

Amire szükségünk lesz:

- Arduino,
- 5 db LED,
- 5 db 220 – 1 kiloohmos ellenállás,
- próbapanel,
- vezetékek,
- Processing vagy,
- Max/MSP v. 5.

Kapcsolás



9.9. ábra: Kapcsolás a switch() feltétel megértéséhez

 © Harsányi Réka, Juhász Márton András

Csatlakoztassunk az Arduino 2, 3, 4, 5, 6-os csatornáihoz egy-egy LED-et soros ellenállással, az eddigi példáknak megfelelően. Az, hogy melyik LED fog világítani, az a sorosan beolvasott karaktertől függ, ami az alábbi kód szerint a, b, c, d vagy e lehet.

Példakód

```
/* created 1 Jul 2009 by Tom Igoe  */

void setup() {
   // nyissuk meg a soros portot
   Serial.begin(9600);

   // 2-6 csatornák legyenek mind kimenetek
   for (int thisPin = 2; thisPin < 7; thisPin++) {
      pinMode(thisPin, OUTPUT);
   }
}

void loop() {
   if (Serial.available() > 0) {
      int inByte = Serial.read();
      /* ha él a soros kapcsolat, olvassuk ki a kapott byte értékét és töltsük
   az inByte nevű változóba ASCII táblának megfelelően a karakterek értékeit is
   használhatnánk pl. 'a' = 97, 'b' = 98 */

      switch (inByte) {
         case 'a':
            // abban az esetben, ha az inByte egy ‘a’ karaktert tartalmaz
            digitalWrite(2, HIGH);
            // kapcsoljuk be a 2-es csatornára kötött LED-et
            break;
         case 'b':
            digitalWrite(3, HIGH);
            break;
         case 'c':
            digitalWrite(4, HIGH);
            break;
         case 'd':
            digitalWrite(5, HIGH);
            break;
         case 'e':
            digitalWrite(6, HIGH);
            break;
```

```
            default: /* a switch() feltétel egyik tulajdonsága, hogy egy
          alapállapotot is be tudunk állítani, ami akkor működik, ha semelyik
          másik feltétel nem áll fenn. Az alapállapotban most minden LED-et
          kikapcsolunk. */
                for (int thisPin = 2; thisPin < 7; thisPin++) {
                  digitalWrite(thisPin, LOW);
                }
            }
    }
}
```

A processing kódot és a MAX patchet az előzőek alapján át tudjuk alakítani. A fent meghatározott L és H karaktereket felválthatjuk a, b, c, d, e bármelyikére és kibővíthetjük a programunkat többgombosra is, az Arduino program kipróbálásához azonban elegendő a soros terminál is.

9.2. Adatátvitel Arduinoról PC-re

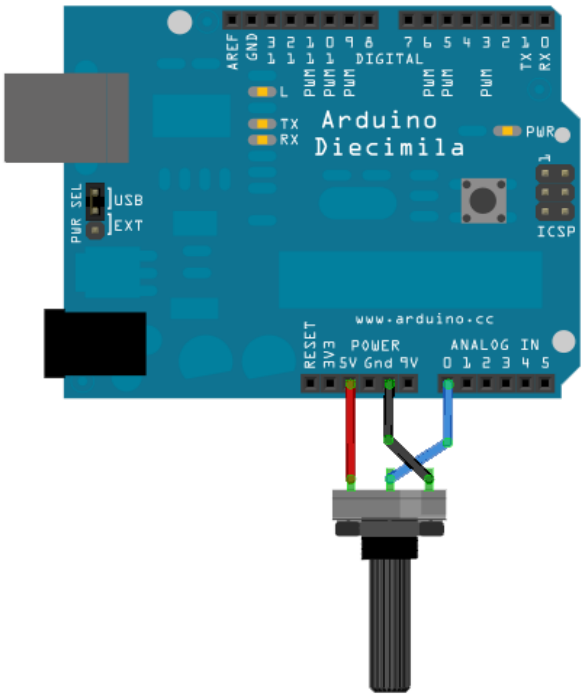
Fordítsuk meg az eddig taglalt folyamatot, és nézzük meg, hogyan lehet az Arduino által gyűjtött fizikai jeleket a számítógépbe eljuttatni. Legyen a szenzorunk az egyik analóg (ADC) bemenetre kötött potenciométer, amivel a tápfeszültséget leosztjuk annál kisebb feszültségekre, hogy a beolvasott 1024 lehetséges állapot közül a potméter elfordításával tudjunk választani. A beolvasott értéket aztán átadjuk a soros porton keresztül a számítógépnek, ahol grafikusan feldolgozzuk.

Amire szükségünk lesz:

- Arduino
- 10–100 kiloohmos potenciométer (3 db)
- Processing vagy
- Max/MSP v. 5

Kapcsolás:

 © Harsányi Réka, Juhász Márton András



9.10. ábra: Potenciométer, mint analóg szenzor bekötése

Kössük a potméterünk egyik szélső lábát az Arduino föld- (GND), a másikat az 5 V-os tápcsatlakozójába. Ha feszültségmérővel a föld és a potméter középső lába között mérünk, azt fogjuk látni, hogy a potméter elforgatásával a feszültség 0–5 V között változik. Ezzel a módszerrel bármikor lemodellezhetjük az összes analóg módon működő szenzort. Az ilyen típusú érzékelők a mért fizikai paramétereket az általuk kibocsátott feszültség szintjével állítják valamilyen arányú összefüggésbe. Kössük a potméter középső lábát az Arduino A0 analóg bemenetére, a digitalizált érték 0–1023 között fog változni.

Példakód

```
   /*
    created 2006 by David A. Mellis
 modified 9 Apr 2012
 by Tom Igoe and Scott Fitzgerald
    */

void setup() {
   // nyissuk meg a soros portot
   Serial.begin(9600);
}
void loop() {
```

```
/* nyomtassuk ki az A0 analóg bemenet aktuális értékét a portra figyeljük meg,
hogy ha a println-t használjuk, szemben a printtel, az adat elküldése után az
„új sor” kódot is elküldi */

  Serial.println(analogRead(A0));
  // várjunk egy kicsit, hogy az analóg-digitális átalakító stabilizálódjon
  delay(2);
}
```

9.2.1. Adatok grafikus ábrázolása Processing segítségével

Az alábbi példakód beolvassa az Arduino által sorosan elküldött adatokat, és minden új adatot egy olyan grafikonon ábrázol, amelyen a függőleges vonalak hossza arányos a potméterünkkel beállított feszültséggel.

```
// Graphing sketch

// This program takes ASCII-encoded strings
// from the serial port at 9600 baud and graphs them. It expects values in the
// range 0 to 1023, followed by a newline, or newline and carriage return

// Created 20 Apr 2005
// Updated 18 Jan 2008
// by Tom Igoe
// This example code is in the public domain.


 // használjuk a beépített soros kommunikációs könyvtárat
 import processing.serial.*;
 Serial myPort;        // nevezzük a soros portunkat myPortnak
 int xPos = 1;         // a grafikonunk aktuális vízszintes koordinátája

 void setup () {
    size(400, 300); // a képernyő mérete
    println(Serial.list()); // nyomtassuk ki az összes elérhető portot

     // nyissuk meg a portot, ahogy az előző példában
    myPort = new Serial(this, Serial.list()[0], 9600);

   /* A serialEvent() funkciót fogjuk használni – ami nagyon hasonló pl. a
   mousePressed()-hez: akkor aktív, amikor egy előre definiált folyamat
   lezajlik. Itt nem az egér megnyomására, hanem egy soros porton beérkező
   vezérlőkarakterre fogunk várni. A két aposztróf közötti karakter (char)
   változóként megadott érték ‘\n’, decimálisan 10-ként is megadható (lásd
```

```
  ASCII tábla) a „New Line” vezérlőkarakter kódja. A bufferUntil addig gyűjti
  egy tárolóba a sorosan kapott byte-okat, amíg meg nem érkezik az a karakter,
  amire vár. Ebben a formában a soros portunk bejövő adatainak átmeneti
  tárolójába mindig egyetlen byte fog kerülni, hiszen az Arduino programunk
  minden adat után elküldi az „új sor” kódját is (println). /
  myPort.bufferUntil('\n');

  background(0); // fekete háttér
}

void draw () {
// minden a serialEvent()-en belül fog történni
}

void serialEvent (Serial myPort) {

  /* az adatot string változóként olvassuk be, ami egy karaktereket tároló
  tömb. Az inString fogja tartalmazni az összes beolvasott karakterünket
  readStringUntil-el megadjuk a fent ismertetett szeparáló karaktert, minden
  „új sor”-ig olvassuk a portot */
   String inString = myPort.readStringUntil('\n');

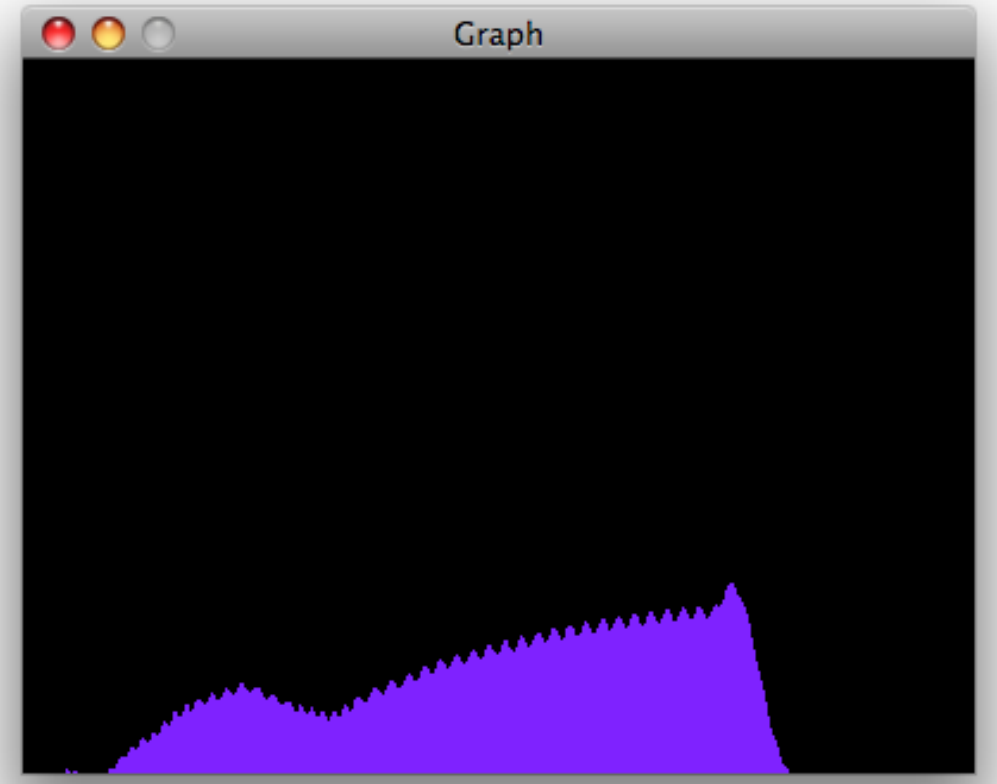
  if (inString != null) {
     // ha a karakterláncunk nem üres, akkor vágjuk le a felesleges részeket
     inString = trim(inString);
     // alakítsuk át a beolvasott karaktert szám formátumúra
     float inByte = float(inString);

     /* változó = map (változó, változó minimuma, változó maximuma, eredmény
     minimuma, eredmény maximuma) ezzel a függvénnyel a beolvasott adat
     értékét a képernyő magasságával arányossá tesszük*/
     inByte = map(inByte, 0, 1023, 0, height);

     stroke(127,34,255);
      // az adat a képernyő magasságához mért vonalként jelenik meg
     line(xPos, height, xPos, height - inByte);

     // a képernyő szélére érve a rajzolás vízszintes pozícióját
     // visszaállítjuk a képernyő elejére
     if (xPos >= width) {
        xPos = 0;
        background(0);
     } else {
     // növeljük a vízszintes pozíció értékét,
```

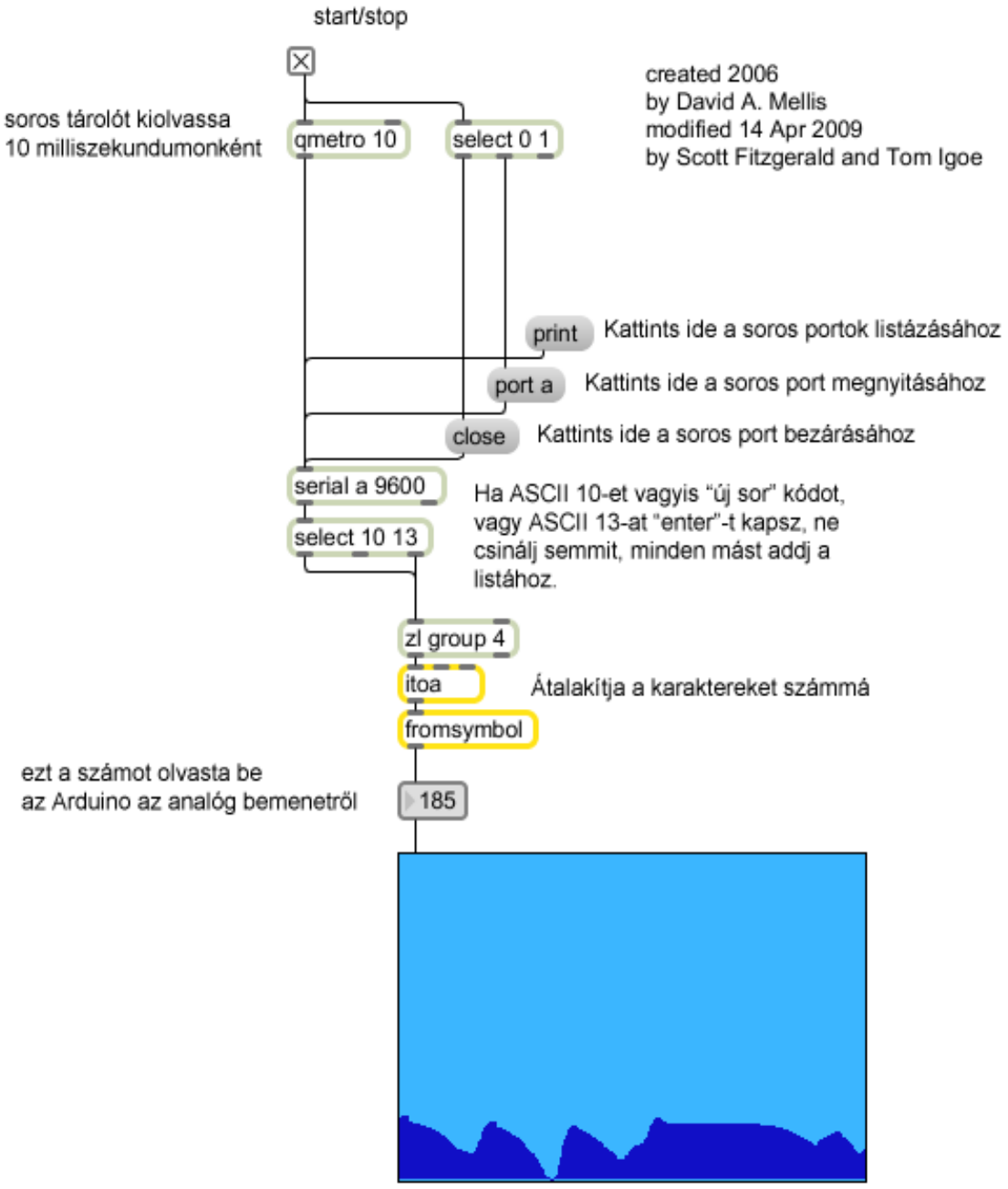
```
      // amíg el nem érjük a képernyő szélét
         xPos++;
      }
   }
}
```



9.11. ábra: Processinggel vizualizált grafikon

 © Harsányi Réka, Juhász Márton András

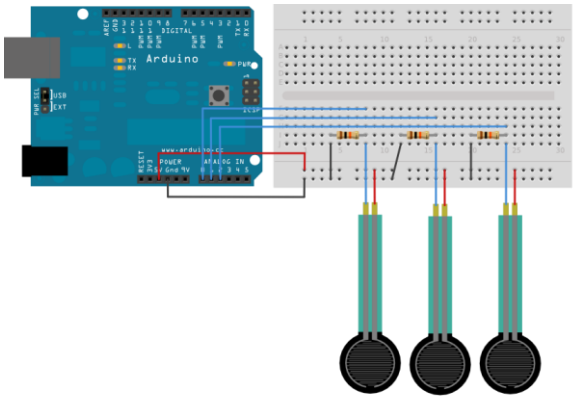
9.2.2. Adatok grafikus ábrázolása Max/MSP segítségével



9.12. ábra: Max/MSP patch Arduinoból sorosan olvasott adatok vizualizációjára

9.2.3. Több független adat átvitele

Bővítsük ki az előző példát, és nézzük meg, hogyan lehet egyszerre több bemenet értékét eljuttatni a számítógépbe. A kapcsolás lényegében lehet ugyanaz, mint eddig, azzal a különbséggel, hogy 3 db potenciométer kapcsolódik három különálló analóg bemenethez, A0-hoz, A1-hez és A2-höz. Ha rendelkezünk más jellegű érzékelőkkel, például fényérzékeny ellenállásokkal, akkor az alábbi feszültségosztó kapcsolást is használhatjuk.



9.13. ábra: Több analóg érzékelő bekötése Arduinoba

Példakód

Az analóg bemenetek értékeit továbbra is decimális kódolású ASCII karakterekként küldjük el, vagyis egy háromjegyű szám átvitelekor három egymást követő byte-ot továbbítunk. ASCII-ben az 1-es számértéke 49, a 2-es számértéke 50 és így tovább. (Lásd ASCII tábla.) A három szenzor értékét a számítógépes példakódokban három színcsatorna (piros, zöld, kék) intenzitásának változtatására fogjuk felhasználni.

```
   /* created 2 Dec 2006
 by David A. Mellis
 modified 30 Aug 2011
 by Tom Igoe and Scott Fitzgerald */

const int redPin = A0;      // piros színhez tartozó analóg bemenet
const int greenPin = A1;    // zöld színhez tartozó analóg bemenet
const int bluePin = A2;     // kék színhez tartozó analóg bemenet

void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  Serial.print(analogRead(redPin)); // küldjük el a piros szenzor értékét
  Serial.print(","); // aztán egy vesszőt
  Serial.print(analogRead(greenPin)); // a zöld értékét
  Serial.print(","); //még egy vesszőt
  Serial.println(analogRead(bluePin)); //végül a kéket és az „új sor” kódját
}
```

9.2.3.1. Virtuális színkeverő Processingben

```
import processing.serial.*;

float redValue = 0;         // a piros értékét lebegőpontos változóban tároljuk
float greenValue = 0;       // zöld
float blueValue = 0;        // kék

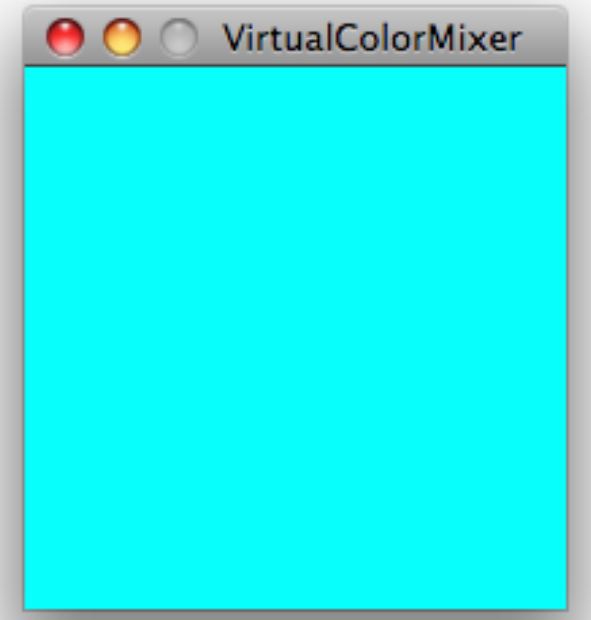
Serial myPort; // soros portunkat myPort-nak nevezzük

void setup() {
  size(200, 200);
  println(Serial.list()); // az összes elérhető soros port listája
  // megnyitjuk a portot, ahogy eddig
  myPort = new Serial(this, Serial.list()[0], 9600);
  // az új sor karakterig töltjük fel az átmeneti tárolót
  myPort.bufferUntil('\n');
}

void draw() {
  // legyen a háttér színe a beolvasott értékekből kikeverve
  background(redValue, greenValue, blueValue);
}

void serialEvent(Serial myPort) {
  // a karaktertömbünkbe töltsük be, amit beolvastunk
  String inString = myPort.readStringUntil('\n');
  if (inString != null) {
    // vágjuk le a felesleges részeket, mint amilyen az „új sor”
    inString = trim(inString);
    /* split() meghatározott karakter mentén hasítja a karaktertömböt, az
elhasított részeket számmá alakítva egyenként egy colors nevű, számokat
tartalmazó tömbbe töltjük. */
    float[] colors = float(split(inString, ","));
    // ha a tömb legalább három elemet tartalmaz, akkor mind a
    // három szenzor értékét megkaptuk
```

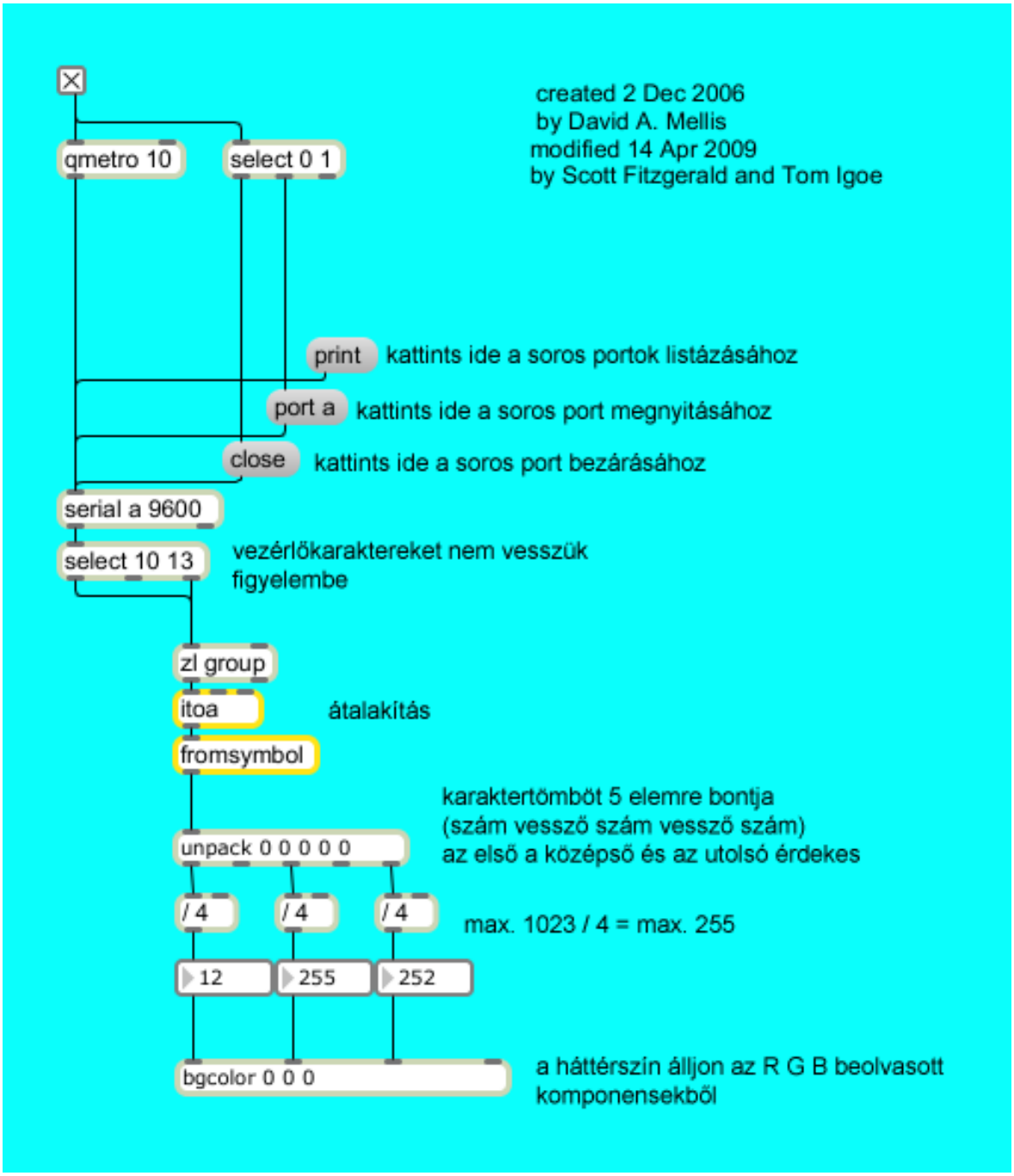
```
      if (colors.length >=3) {
         // a kapott szám 0-tól 1023-ig változik, viszont ez nem igaz
         // a szín csatornáira, amik csak 0-tól 255-ig képesek értéket
         // felvenni, ezért a map() függvénnyel arányosítjuk
         redValue = map(colors[0], 0, 1023, 0, 255);
         greenValue = map(colors[1], 0, 1023, 0, 255);
         blueValue = map(colors[2], 0, 1023, 0, 255);
      }
   }
}
```



9.14. ábra: A potméterek elforgatásával virtuálisan színeket keverhetünk.

© Harsányi Réka, Juhász Márton András

9.2.3.2. Virtuális színkeverő Max/MSP-ben



9.15. ábra: A potméterek elforgatásával virtuálisan színeket keverhetünk.

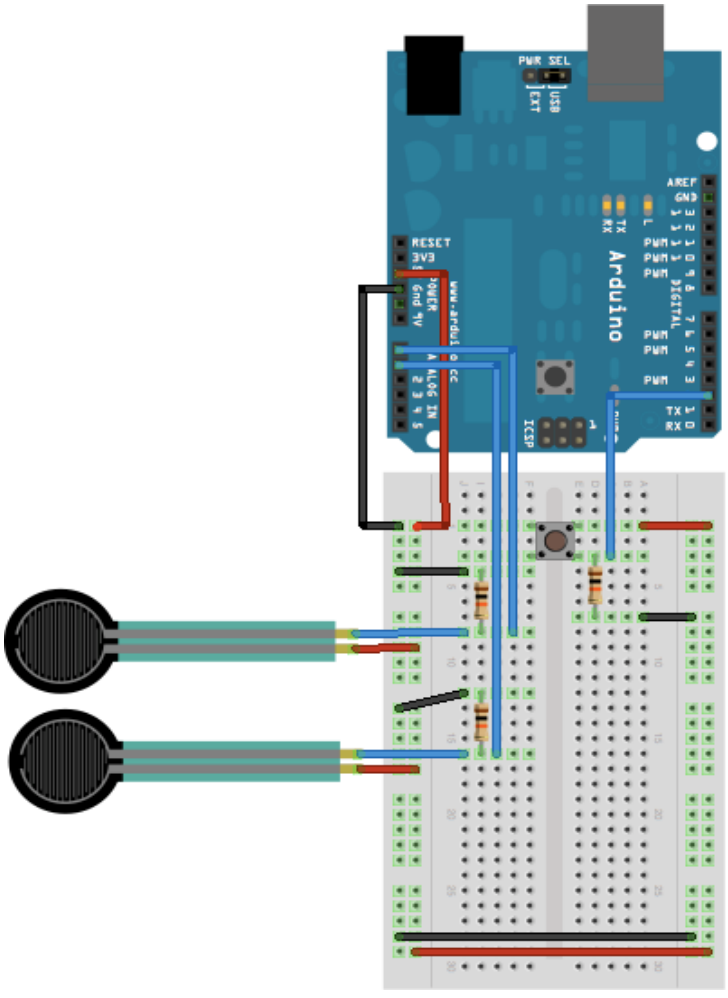
9.3. Kétirányú kapcsolat – meghívás és válasz

Előfordulhat, hogy a szenzorértékek beolvasása vagy a kimenetek kapcsolása önmagában nem elegendő, és kétirányú kapcsolatot kell létrehozni az Arduino és a számítógép között. Erre léteznek különféle protokollok, melyek közül a legismertebb a Firmata – aminek működésére később kitérünk. Azonban most nézzünk egy jóval egyszerűbb megoldást, az úgynevezett Handshaking módszert. A példában az Arduino indulásakor egy A karaktert küld a számítógépnek, ezt követően a hozzá kapcsolt három szenzor értékét is továbbítja. Ha a számítógép készen áll és megkapta a „Start jelet” (A), beolvassa a szenzorok értékét és visszaküld egy A karaktert az Arduinonak, hogy folytassa az adatok küldését.

Amire szükségünk lesz:

- Arduino
- 2 db analóg szenzor,
- 1 db nyomógomb,
- 3 db 10 kiloohmos ellenállás,
- próbapanel,
- vezetékek,
- Processing, vagy
- Max/MSP v. 5.

Kapcsolás:



9.16. ábra: Két érzékelő és egy gomb bekötése

 © Harsányi Réka, Juhász Márton András

Kössünk be két analóg érzékelőt az A0 és A1 analóg bemenetekre, illetve egy nyomógombot a 2-es digitális csatornára. A két analóg érzékelő működését modellezhetjük potméterekkel, lásd a virtuális színkeveréskor használt kapcsolást.

Példakód

```
/*
 Serial Call and Response

 Thanks to Greg Shakar and Scott Fitzgerald for the improvements
 Created 26 Sept. 2005 by Tom Igoe
 modified 24 April 2012 by Tom Igoe and Scott Fitzgerald
 */

int firstSensor = 0;    // egyik analóg érzékelő
int secondSensor = 0;   // másik analóg érzékelő
int thirdSensor = 0;    // nyomógomb
int inByte = 0;         // beérkező soros byte

void setup() {
  // megnyitjuk a soros portot
  Serial.begin(9600);
  pinMode(2, INPUT);   // a nyomógomb a 2-es csatornán bemenet
  // A karakter elküldése, lásd lent a függvény definíciójánál
  establishContact();
}

void loop() {
  // ha kapunk a számítógéptől adatot (A)
  if (Serial.available() > 0) {
    // olvassuk be, nem csinálunk vele semmit, de ne maradjon a tárolóban
    inByte = Serial.read();
    // A0-ról kapott értéket osszuk el néggyel,
    // hogy 0 és 255 közé essen az értéke.
    firstSensor = analogRead(A0)/4;
    // várjunk egy kicsit, hogy az analóg-digitális átalakító stabilizálódjon
    delay(10);
    // következő analóg érzékelő
    secondSensor = analogRead(A1)/4;
    // olvassuk be a digitális gomb állapotát, ami 0 vagy 1 lehet,
    // ha 0, thirdSensor értéke legyen 0, ha 1, akkor legyen 255
    thirdSensor = map(digitalRead(2), 0, 1, 0, 255);
```

```
    // küldjük el a három értéket byte-ként
    Serial.write(firstSensor);
    Serial.write(secondSensor);
    Serial.write(thirdSensor);
  }
}

void establishContact() {
  while (Serial.available() <= 0) {
    // ha még nem jött létre a kapcsolat, küldjünk egy A karaktert
    Serial.print('A');
    delay(300);
  }
}
```

9.3.1. Handshaking módszer Processing oldalról

Az analóg szenzorokkal egy körlapot mozgatunk a képernyőn, a nyomógombbal a körlap színét változtatjuk feketére.

```
import processing.serial.*;

int bgcolor;   // háttérszín
int fgcolor;   // körlap színe
Serial myPort;    // soros port
// serialInArray nevű tömbben tároljuk a három adatot, amit kapunk
int[] serialInArray = new int[3];
int serialCount = 0;            // számoljuk a beérkező byte-okat
int xpos, ypos;                // a körlap koordinátái
// firstContact bináris változó: bejelentkezett-e már legalább egyszer az
Arduino boolean firstContact = false; // alapállapotban hamis

void setup() {
  size(256, 256);  // képernyő mérete
  noStroke();      // nem rajzolunk körvonalakat

  // a körlap kiinduláskor a képernyő közepén lesz
  xpos = width/2;
  ypos = height/2;

  // listázzuk az összes soros portot
  println(Serial.list());

  // megnyitjuk a soros portot
```

```
  String portName = Serial.list()[0];
  myPort = new Serial(this, portName, 9600);
}

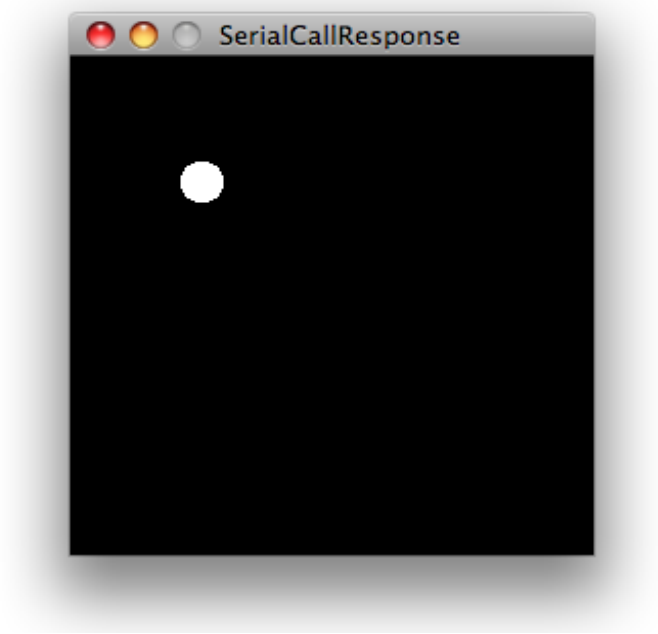
void draw() {
  background(bgcolor);
  fill(fgcolor);
  // körlap kirajzolása
  ellipse(xpos, ypos, 20, 20);
}

void serialEvent(Serial myPort) {
  // olvassunk egy byte-ot a soros portról
  int inByte = myPort.read();
  // ha még nem jött létre a kapcsolat egyszer sem és az olvasott byte
  // egy A karakter, akkor állítsuk a firstContact változót IGAZ-ra
    if (firstContact == false) {
      if (inByte == 'A') {
         myPort.clear();
         /* töröljük az átmeneti soros tárolót (az első megkapott szenzor
      értékeket, mert csak a következő ciklustól fogjuk tudni őket használni a
      jelenlegi feltétel miatt). */
         firstContact = true;     // állítsuk a firstContact változót igazra
         myPort.write('A');       // küldjünk egy karaktert az arduinonak, hogy
                                  // folytassa
      }
    } else {
      // ha nem ez az első alkalom, hogy a kapcsolat létrejött
      // adjuk hozzá az adatokat tartalmazó tömbhöz, amit kaptunk
      serialInArray[serialCount] = inByte;
      serialCount++; // növeljük a tömb számlálójának értékét eggyel

      // ha megvan a 3 byte, amire vártunk,
      if (serialCount > 2 ) {
          // az első legyen a körlap vízszintes koordinátája,
         xpos = serialInArray[0];
         // a második a függőleges,
         ypos = serialInArray[1];
         // a harmadik pedig a nyomógombbal állítsa be a körlap színét.
         fgcolor = serialInArray[2];

         //println() függvénnyel ki is irathatjuk a kapott értékeket
         // („\t” a tabulátor)
        println(xpos + "\t" + ypos + "\t" + fgcolor);
```

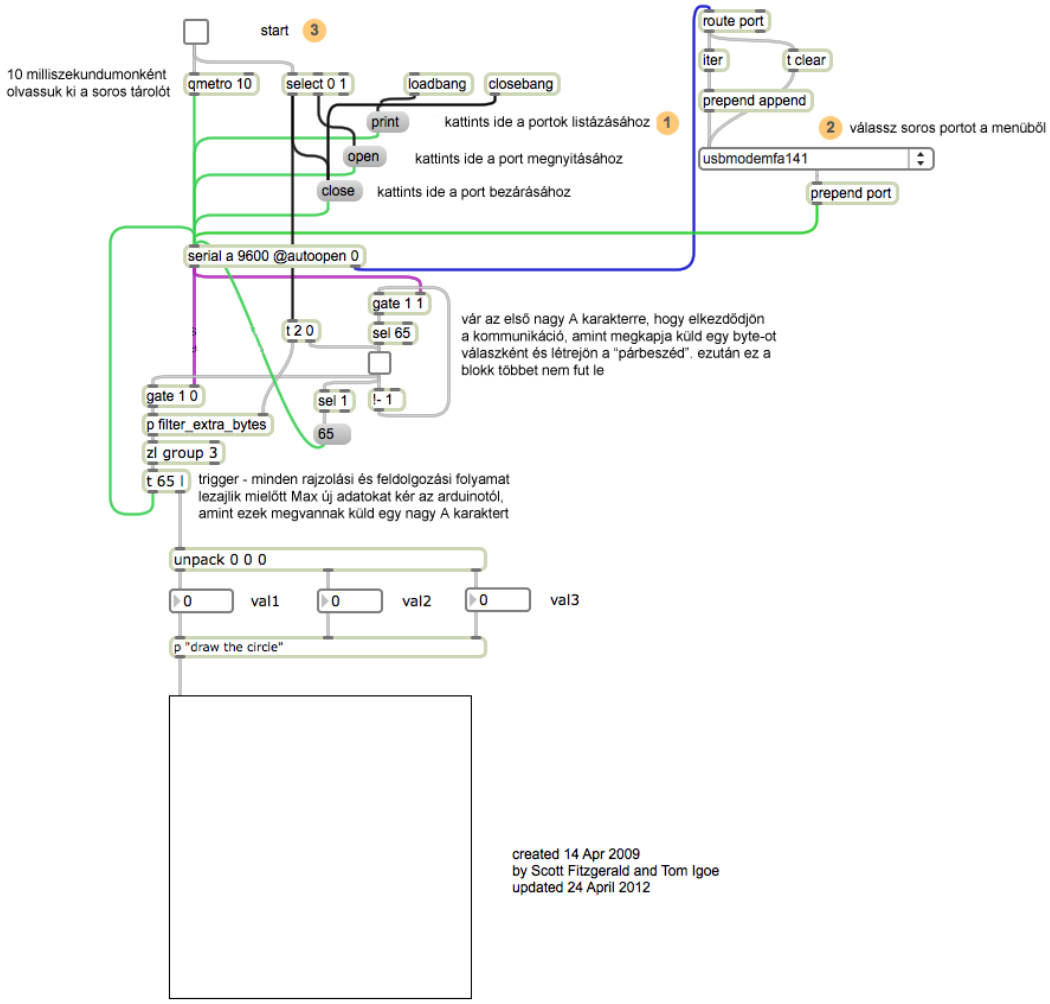
```
        // küldjünk egy 'A' karaktert az Arduinonak, hogy folytassa a küldést
        myPort.write('A');
        serialCount = 0; // állítsuk vissza a tömb számlálóját
       }
   }
}
```



9.17. ábra: Körlap mozgatása két analóg szenzorral

 © Harsányi Réka, Juhász Márton András

9.3.2. Handshaking módszer Max/MSP oldalról



9.18. ábra: Max/MSP handshaking patch

10. Vezérlő struktúrák

10.1. If állítás

Az *if()* állítás a legalapvetőbb vezérlő struktúra. Lehetővé teszi, hogy valami megtörténjen, vagy ne történjen meg, attól függően, hogy a feltétel igaz vagy hamis. Így néz ki:

```
if (someCondition) {
   // csinálj dolgokat, ha a feltétel igaz
}
```

Van egy gyakoribb verziója is, amit *if-else*-nek neveznek:

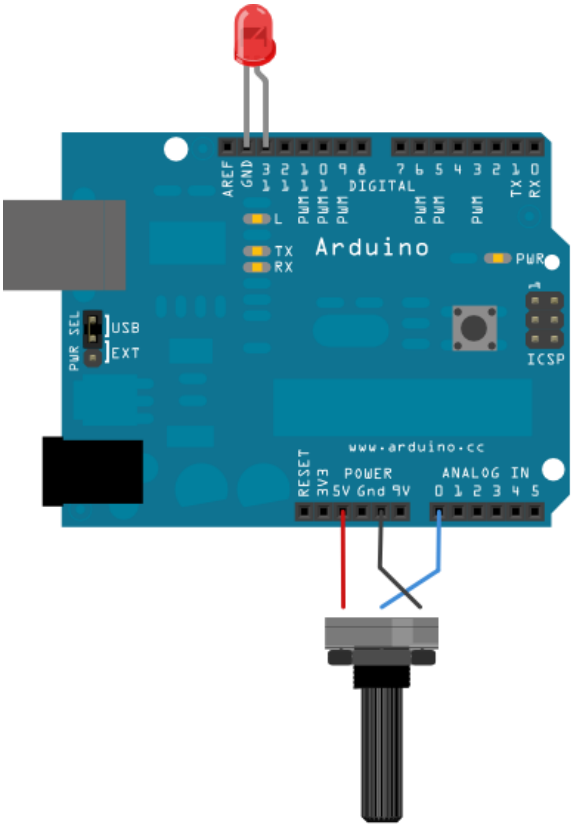
```
if (someCondition) {
   // csinálj dolgokat, ha a feltétel igaz
} else {
   // csinálj dolgokat, ha a feltétel hamis
}
```

Létezik az *else-if* verzió is, ebben az esetben, ha az első feltétel hamis, akkor a másodikat is megvizsgálja:

```
if (someCondition) {
   // csinálj dolgokat, ha a feltétel igaz
} else if (anotherCondition) {
   // csinálj dolgokat, ha a feltétel hamis
   // és a második feltétel igaz
}
```

Az alábbi példa bekapcsolja a LED-et, ha az analóg bemeneten kiolvasott érték a küszöbérték alá megy.

Bekötése: A LED pozitív lába elé kössünk be egy 220 ohmos ellenállást is, a potméter középső lábát az A0-ba, a két szélsőt az 5 V-ba és a feszültségbe.



10.1. ábra: Potméter LED-del

Az analogValue nevű változóban fogjuk eltárolni a potméterből kiolvasott adatokat. Ezeket hasonlítjuk majd össze a küszöbértékkel. Ha a felett van az érték, bekapcsol a LED, ha alatta van, akkor kikapcsol.

```
/* -
Feltételek – if állítás
Az analóg bemenetből kiolvasott potméter adatait hasonlítja össze a
küszöbértékkel, ahhoz képest be- vagy kikapcsolja a LED-et.
A beolvasott értéket kiírja, függetlenül a küszöbértéktől.

 Áramkör:
 * A potméter középső lába az analóg 0-ba, a szélsők 5 V-ba és földelésbe
kötve,
 * LED digitális 13-ba és földelésbe kötve (elhagyható)

 created 17 Jan 2009, modified 9 Apr 2012 by Tom Igoe
 */

// Állandókkal meghatározzuk a pineket:
const int analogPin = A0;    // potméter
```

```
const int ledPin = 13;       // LED
const int threshold = 400;   // a tetszőleges küszöbérték,
// az analóg szenzor értéktartományán belül .

void setup() {
   // inicializáljuk a LED pint kimenetként:
   pinMode(ledPin, OUTPUT);
   // inicializáljuk a soros kommunikációt:
   Serial.begin(9600);
}

void loop() {
   // kiolvassuk a potmétert:
   int analogValue = analogRead(analogPin);

   //ha a bejövő analóg érték elég magas, bekapcsoljuk a LED-et:
   if (analogValue > threshold) {
      digitalWrite(ledPin, HIGH);
   } else {
      digitalWrite(ledPin,LOW);
   }

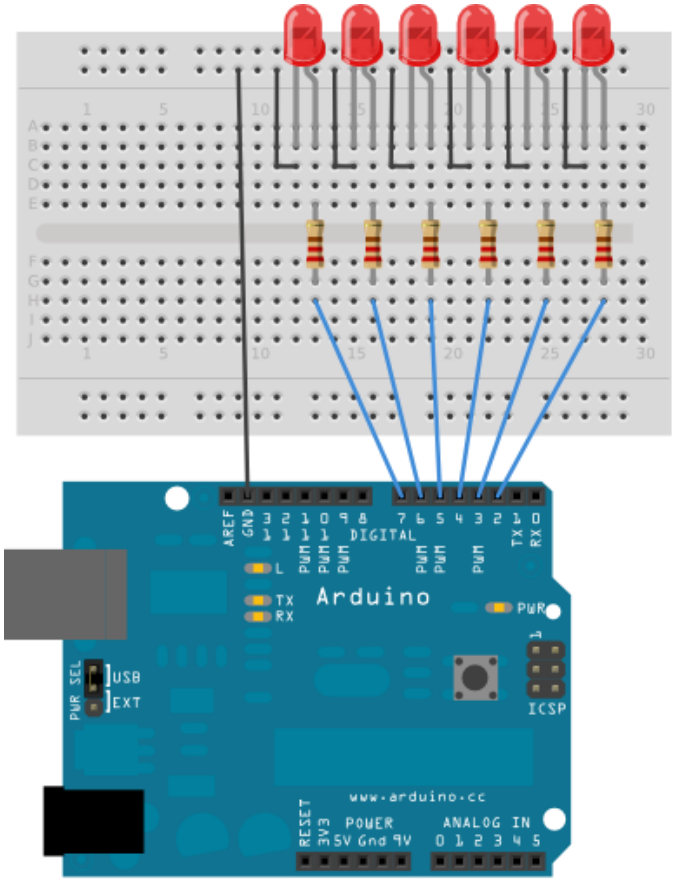
   // kiírjuk a bejövő értéket:
   Serial.println(analogValue);
   delay(1);
// késleltetés a kiolvasások között a stabilitáshoz
}
```

10.2. For ciklus

Néha arra van szükség, hogy ismétlődjön a folyamat, pl. pinek sorozatán ugyanaz a parancs fusson le, vagy növelni, illetve csökkenteni akarunk egy változót és újra elvégezni rajtuk az utasításokat. Ilyenkor a for() ciklus a legmegfelelőbb megoldás.

A következő példában digitális 2–7 pinbe egy-egy LED-et kötünk és egymás utáni sorban be- és kikapcsoljuk őket egyik irányba, majd visszafelé is. A cikluson belül a *digitalWrite()* és a *delay()* függvényeket fogjuk használni.

Bekötése: Mindegyik piros 5 mm-es LED-ünk pozitív lába elé bekötünk egy-egy 220 ohmos ellenállást, ezeket a digitális 2, 3, 4, 5, 6, 7 pinekbe kötjük, a negatív lábakat pedig leföldeljük.



10.2 ábra: 6 LED bekötése

Arduino kód:

A loopban két **for()** ciklus szerepel, az elsőben a 2. pintől indulva lépkedünk a 7-ig, a másodikban fordítva, a 7-estől indulunk, és mindig eggyel csökkentve az értéket eljutunk a 2-ig. Illetve a pineket is for ciklussal inicializáljuk. A for ciklus addig ismétlődik, amíg a kerek zárójelben lévő feltétel teljesül, utána továbblép.

```
/*
 For ciklus ismétlés.

 A 2-7 digitális pinbe kötött LED-eket egyenként sorban fel- és lekapcsolja
mindkét irányban, folyamatosan ismételve.

 created 2006 by David A. Mellis
 modified 30 Aug 2011 by Tom Igoe
```

```
 */

int timer = 100;
// Minél nagyobb a szám, annál lassabb az időzítés.

void setup() {
  // a for ciklust használjuk a pinek inicializálására,
  // kimenetként határozzuk meg őket:
  for (int thisPin = 2; thisPin < 8; thisPin++)  {
    pinMode(thisPin, OUTPUT);
  }
}

void loop() {
  // A legkisebb pintől indulunk felfelé,
  // egyesével fut le a parancs (looponként):
  for (int thisPin = 2; thisPin < 8; thisPin++) {
    // bekapcsoljuk a pint:
    digitalWrite(thisPin, HIGH);
    // várunk a fent meghatározott ideig
    delay(timer);

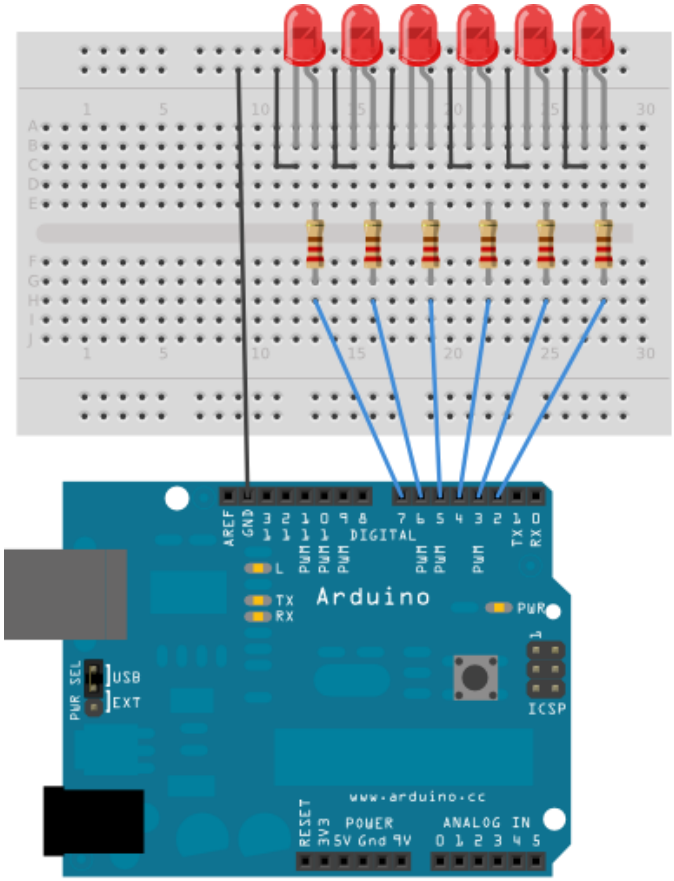
    // kikapcsoljuk a pint:
    digitalWrite(thisPin, LOW);
  }

  // A legnagyobb pintől számolunk egyesével lefelé:
  for (int thisPin = 7; thisPin >= 2; thisPin--) {
    // bekapcsoljuk a pint:
    digitalWrite(thisPin, HIGH);
    // várunk a fent meghatározott ideig
    delay(timer);
    // kikapcsoljuk a pint:
    digitalWrite(thisPin, LOW);
  }
}
```

10.3. Tömbök

A for ciklus példáját alakítjuk át úgy, hogy a pinek értékeit tömbben tároljuk el, és innen olvassuk ki őket mind az inicializálásnál, mind a loopban szereplő két for ciklusban.

Bekötése: Mindegyik piros 5 mm-es LED-ünk pozitív lába elé bekötünk egy-egy 220 ohmos ellenállást, ezek kerülnek bekötésre a digitális 2, 3, 4, 5, 6, 7 pinekbe, a negatív lábakat pedig leföldeltük.



10.3 ábra: 6 LED bekötése

Arduino kód:

```
/*
 Tömbök

Egy tömböt használunk, amik sorrendben tartalmazzák a kimenetkét használt pinek
számait. Ez a sorrend egy tetszőleges, általunk megadott sorrendet jelent. A
for ciklusban a tömbből sorban olvassuk ki a számokat, amik a pinekre
vonatkoznak, de ezeket most csak az inicializáláshoz használjuk. A két for
loopban egyesével számolunk felfelé, majd lefelé, így határozzuk meg, melyik
pin legyen bekapcsolva.

 created 2006 by David A. Mellis
 modified 30 Aug 2011 by Tom Igoe
 */

int timer = 100;
```

```
// Minél nagyobb a szám, annál lassabb az időzítés.

int ledPins[] = {
  2, 7, 4, 6, 5, 3 };       // pin számok, sorrendben
int pinCount = 6;           // az összes pin darabszáma,
      // megegyezik a tömb elemeinek darabszámával

void setup() {
  int thisPin;
  // A tömb elemeit 0-tól számoljuk!
  //  a pineket meghatározzuk kimenetként:
  for (int thisPin = 0; thisPin < pinCount; thisPin++)  {
    pinMode(ledPins[thisPin], OUTPUT);
  }
}

void loop() {
  // A legkisebb pintől számolunk egyesével felfelé:
  for (int thisPin = 0; thisPin < pinCount; thisPin++) {
    // bekapcsoljuk a pint:
    digitalWrite(ledPins[thisPin], HIGH);
    // várunk a fent meghatározott ideig
    delay(timer);

    // kikapcsoljuk a pint:
    digitalWrite(ledPins[thisPin], LOW);
  }

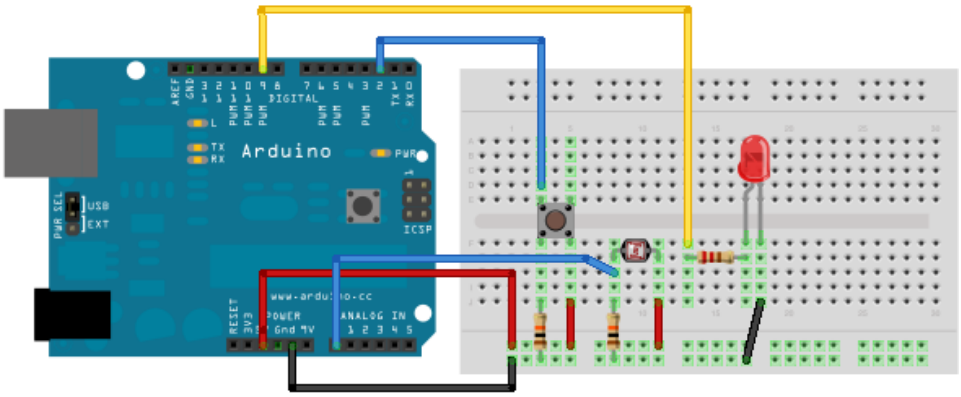
  // a legnagyobb pintől számolunk egyesével lefelé
  for (int thisPin = pinCount - 1; thisPin >= 0; thisPin--) {
    // bekapcsoljuk a pint:
    digitalWrite(ledPins[thisPin], HIGH);
    // várunk a fent meghatározott ideig
    delay(timer);

    // kikapcsoljuk a pint:
    digitalWrite(ledPins[thisPin], LOW);
  }
}
```

10.4. While ciklus

A while ciklus addig fut, amíg a feltétele teljesül. Ebben az esetben, amikor megnyomjuk a gombot, lefut a fotóellenállás kalibrálására vonatkozó parancssor. Ezután a fotóellenállásból kiolvasott analóg értékeket skálázás után kiküldjük a LED fényerejét vezérlő PWM pinre. Minél nagyobb a beérkező érték, annál erősebben világít a LED.

Bekötése: A LED pozitív lábát a 220 ohmos ellenálláson át a 9 PWM pinbe kötjük, a negatívat a földelésbe. A kapcsolóhoz 10 kiloohmos ellenállást használunk, ezen keresztül a bal alsó lábát bekötjük a földelésbe, a jobb alsót az 5 V pinbe, a bal felsőt pedig a 2 pinbe. A fotóellenállás bármilyen irányban beköthető, egyik oldala az 5 V pinbe, a másik szintén 10 kiloohmos át a földelésbe és az analóg 0 pinbe is megy.



10.4 ábra: Nyomógomb, fotóellenállás és LED bekötése

Arduino kód:

```
/*
 Feltételek - while ciklus
 Ameddig nyomjuk a gombot, fut a kalibrációs rutin. Ezalatt megkapjuk a
szenzorból kiolvasott minimum és maximum értékeket.

 Áramkör:
 * fotóellenállás – 10 kiloohm ellenállással, nyomógomb – 10 kiloohm
ellenállással, LED – 220 Ohm ellenállással

 created 17 Jan 2009
 modified 30 Aug 2011 by Tom Igoe
 */

// Állandókkal meghatározzuk a pineket:
const int sensorPin = A0;       // fotóellenállás
const int ledPin = 9;           // LED
```

```
const int indicatorLedPin = 13; // beépített LED
const int buttonPin = 2;        // nyomógomb

// Deklaráljuk a változókat:
int sensorMin = 1023;  // minimum szenzor érték
int sensorMax = 0;     // maximum szenzor érték
int sensorValue = 0;     // kiolvasott szenzor érték


void setup() {
  // meghatározzuk a LED pineket kimenetnek,
  // a gombot bemenetnek
  pinMode(indicatorLedPin, OUTPUT);
  pinMode (ledPin, OUTPUT);
  pinMode (buttonPin, INPUT);
}

void loop() {
  // Ha a gombot megnyomjuk, lefut a kalibrálás:
  while (digitalRead(buttonPin) == HIGH) {
    calibrate();
  }
  // A kalibrálás végén a „jelző” beépített LED kikapcsol:
  digitalWrite(indicatorLedPin, LOW);

  // kiolvassuk a szenzort:
  sensorValue = analogRead(sensorPin);

  // az aktuális beolvasott szenzorértéket
  // meghatározzuk: skálázzuk PWM-hez:
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 255);

  // abban az esetben, ha a beolvasott érték a kalibrációs
  // értékeken kívülre esne, constrainnel
  // bekényszerítjük 0-255 közé:
  sensorValue = constrain(sensorValue, 0, 255);

  // az értéket a PWM LED pinre küldjük:
  analogWrite(ledPin, sensorValue);
}

void calibrate() {
  // bekapcsoljuk a beépített LED-et, amikor a
  // kalibrálás megkezdődik:
  digitalWrite(indicatorLedPin, HIGH);
  // kiolvassuk a szenzort:
  sensorValue = analogRead(sensorPin);

  // rögzítsük a maximum szenzor értéket
  if (sensorValue > sensorMax) {
    sensorMax = sensorValue;
  }
```

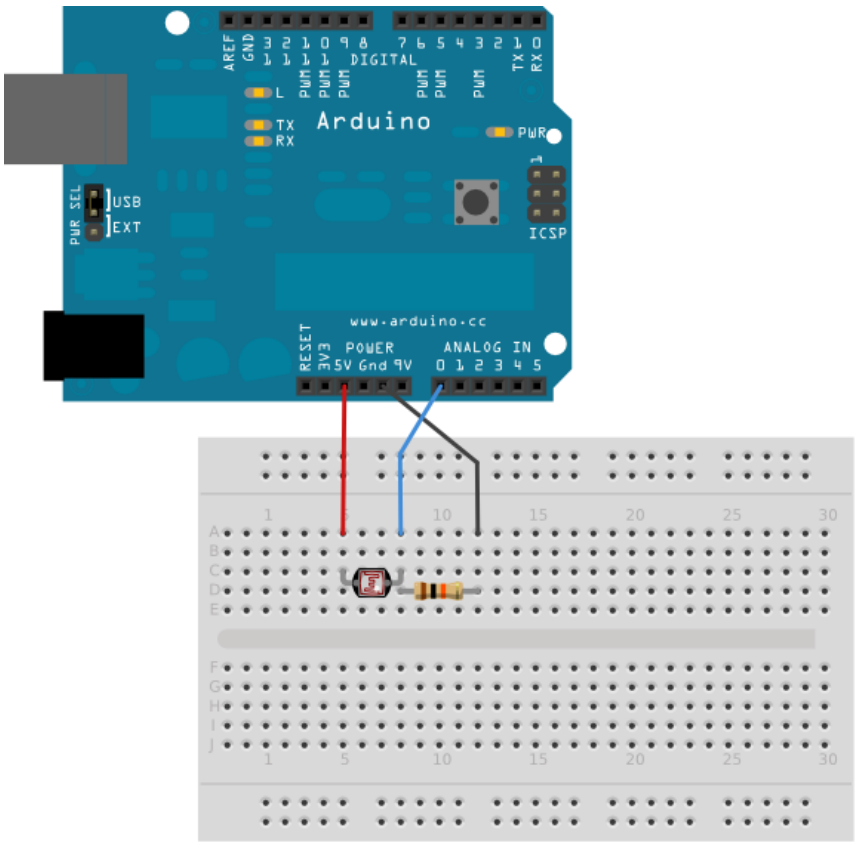
 © Harsányi Réka, Juhász Márton András

```
  // rögzítsük a minimum szenzor értéket
  if (sensorValue < sensorMin) {
    sensorMin = sensorValue;
  }
}
```

10.5. Switch utasítás szenzor kiolvasással

Az if feltételnek két kimenetele lehet, igaz vagy hamis, ettől függően lefut vagy nem fut le a parancs. A switch kifejezésben egyszerre több feltételt vizsgálhatunk, és az fut le közülük, amelyik épp teljesül. Ebben a példában a fotóellenállásból kiolvasott értékekhez rendelünk a fényre vonatkozó definíciókat: sötét (dark), gyenge (dim), közepes (medium), világos (bright). A szenzor érzékelési tartományát a kód elején adjuk meg, most 0–600 közötti, de ennek megváltoztatásával kísérletezhetünk. A beolvasott értéket a map() utasítással 0–3 közötti értékre skálázzuk, ez összesen négy darab lehetőséget eredményez (0, 1, 2, 3), mindegyikhez tartozik egy switch utasítás.

Bekötése: A fotóellenállás egyik lábát az 5 V pinbe, a másikat az analóg 0-ba és egy 10 kiloohmos ellenálláson át a földelésbe is bekötjük.



10.5. ábra: Fotóellenállás bekötése

Arduino kód:

```
/*
 Switch kifejezés
 A switch kifejezés lehetővé teszi, hogy több különböző érték közül válasszunk.
Olyan, mintha több if feltételünk lenne egyben. Serial monitoron kiírja a
szenzorból beolvasott értékekre vonatkozó definíciókat.

 Áramkör:
 A fotóellenállás egyik lábát az 5 V pinbe, a másikat az analóg 0-ba és egy 10
kiloohmos ellenálláson át a földelésbe is bekötjük.

 created 1 Jul 2009
 modified 9 Apr 2012 by Tom Igoe
 */

// Állandókkal meghatározzuk a szenzor szerintünk használható tartományát:
```

 © Harsányi Réka, Juhász Márton András

```
// kísérletezzünk ezek változtatásával:
const int sensorMin = 0;      // minimum érték, const int sensorMax = 600;
 // maximum érték

void setup() {
  // inicializáljuk a soros kommunikációt:
  Serial.begin(9600);
}

void loop() {
  // kiolvassuk a szenzor értékét:
  int sensorReading = analogRead(A0);
  // a szenzor értékét skálázzuk 0-3 közé - egy range nevű
  // változóban tároljuk,
  // ezeket használjuk a switch-ben:
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3);

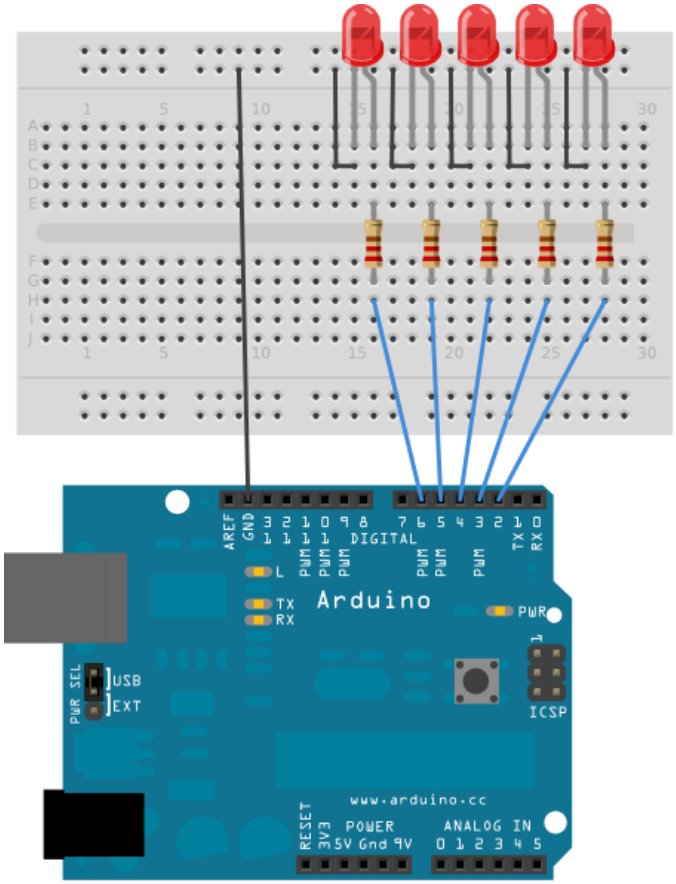
  // az a case parancs fut le, amelyik megegyezik a
  // „range” változó aktuális értékének:

  switch (range) {
  case 0:    // a kezed a szenzoron van
    Serial.println("dark");
    break;
  case 1:    // a kezed a szenzor közelében van
    Serial.println("dim");
    break;
  case 2:    // a kezed pár cm-re van a szenzortól
    Serial.println("medium");
    break;
  case 3:    // nem takarod el a szenzort
    Serial.println("bright");
    break;
  }
  delay(1);
  // késleltetés a kiolvasások között a stabilitás miatt
}
```

10.6. Switch utasítás soros bemenettel

Most a switch case-t LED-ek bekapcsolására fogjuk használni. Attól függően kapcsol be egyik vagy másik LED, hogy a soros porton a, b, c, d vagy e érték érkezik be. Ha megnyitjuk a serial monitort, le tudjuk őket követni. ASCII táblázatban meg tudjuk nézni, melyik karakternek mennyi a számértéke, pl. „a” = 97, „b” = 98 stb.

Bekötése: Mindegyik piros 5 mm-es LED-ünk pozitív lába elé bekötünk egy-egy 220 ohmos ellenállást, ezek mennek a digitális 2, 3, 4, 5, 6, 7 pinekbe, a negatív lábakat pedig leföldeljük.



10.6. ábra: 6 LED bekötése

Arduino kód:

```
/*
Switch utasítás soros bemenettel

Több feltétel közül az az utasítás fut le, amelyikre épp igaz az állítás.
Olyan, mintha sok if feltételünk lenne egyben. Serial monitoron láthatjuk,
milyen karakterértéket küldtünk soros porton. Az a, b, c, d, e karakterek
felkapcsolnak egy LED-et.

Áramkör:
 * 5 mm-es LED pozitív labai a 2-6 pinekbe mennek egy-egy 220 ohmos
ellenálláson át.

 created 1 Jul 2009 by Tom Igoe
 */

void setup() {
```

```
  // inicializáljuk a soros kommunikációt:
  Serial.begin(9600);
  // inicializáljuk a LED pint
  for (int thisPin = 2; thisPin < 7; thisPin++) {
        pinMode(thisPin, OUTPUT);
  }
}

void loop() {
  // kiolvassuk a szenzort:
  if (Serial.available() > 0) {
    int inByte = Serial.read();
    // A beérkező karakterektől függően különböző dolgok
    // történnek. A case feltétel egy karakterértéket vár el,
    // minden karakternek van egy számértéke,
    // ASCII táblázatban megkereshető, pl.
    //'a' = 97, 'b' = 98 stb.

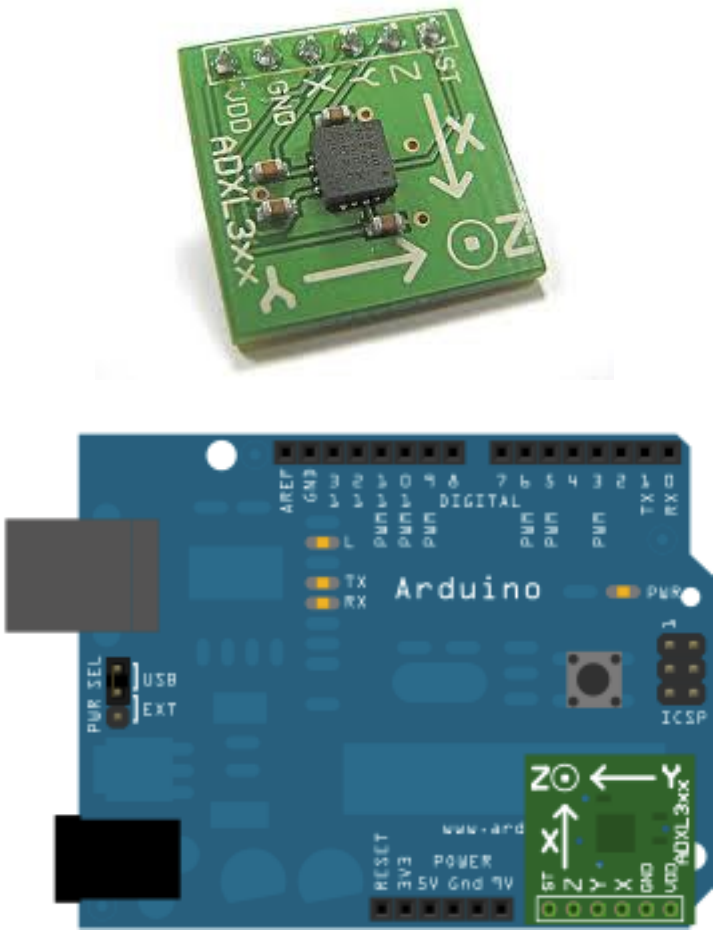
    switch (inByte) {
    case 'a':
      digitalWrite(2, HIGH);
      break;
    case 'b':
      digitalWrite(3, HIGH);
      break;
    case 'c':
      digitalWrite(4, HIGH);
      break;
    case 'd':
      digitalWrite(5, HIGH);
      break;
    case 'e':
      digitalWrite(6, HIGH);
      break;


    default:
      // minden LED-et kikapcsol:
      for (int thisPin = 2; thisPin < 7; thisPin++) {
        digitalWrite(thisPin, LOW);
      }
    }
  }
}
```

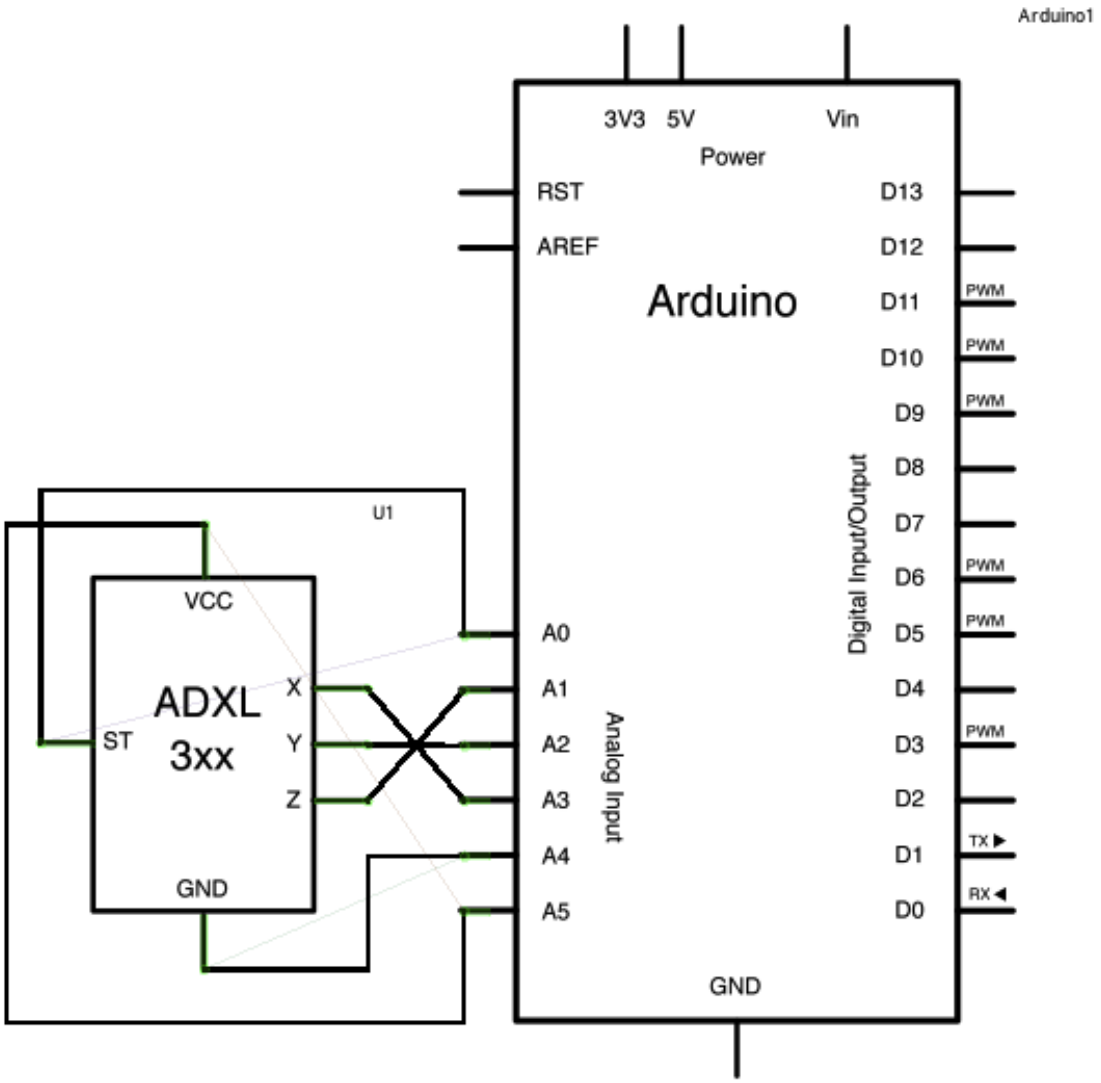
11. Szenzorok, motorok

11.1. ADXL3xx gyorsulásmérő

Megnézzük, hogyan lehet adatot kinyerni az ADXL3xx sorozat gyorsulásmérőiből. 1, 2 vagy 3 tengely irányában működnek, a rászerelt tárgy Földhöz viszonyított helyzetét mérik, tehát a sebesség változását. Analóg és digitális kimenettel rendelkezőek egyaránt vannak. Ebben a példában az x, y, z tengelyből kinyert adatokat soros portra küldjük. Egyéb gyorsulásmérőt is használhatunk, a datasheetben keressük vissza, melyik lába miként funkcionál.

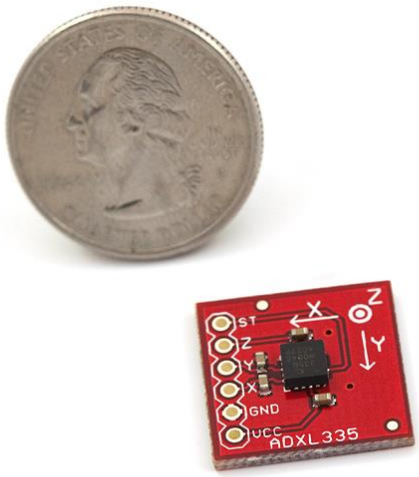


11.1.1. ábra: ADXL3xx gyorsulásmérő és az Arduino panelbe kötött ábrája. A lábak kiosztás olyan sorrendű, hogy elég rádugni az analóg pinsorra.



11.1.2. ábra: ADXL3xx gyorsulásmérő bekötése, áramköri rajz

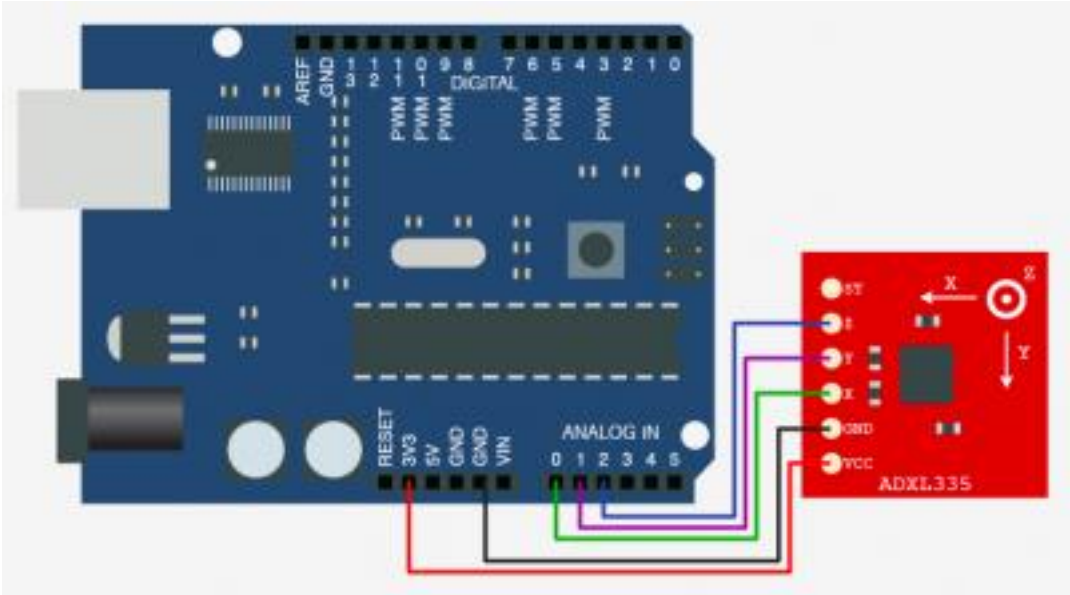
Több gyártó létezik, közülük most két szenzornak a bekötését nézzük meg.
Az első a **Spurkfun** által gyártott:



11.1.3. ábra: Spurkfun ADXL335 gyorsulásmérő

11.1.4. táblázat: Spurkfun gyorsulásmérő bekötése Arduinoba:

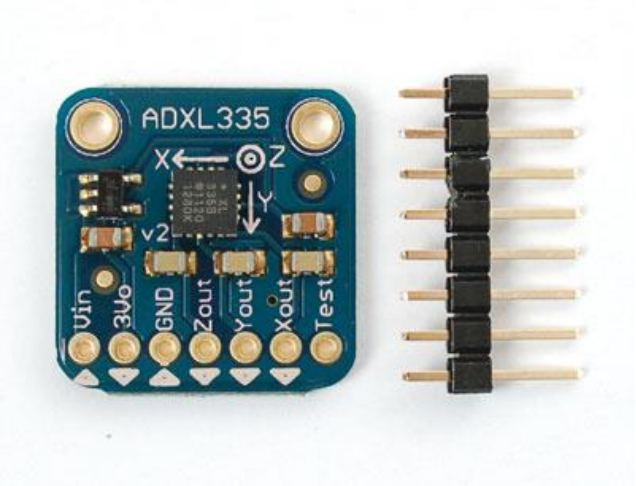
Spurkfun	Self-Test	Z-Axis	Y-Axis	X-Axis	Ground	VCC
Arduino	A0	A1	A2	A3	A4	A5



11.1.5. ábra: Spurkfun gyorsulásmérő Arduinoval

 © Harsányi Réka, Juhász Márton András

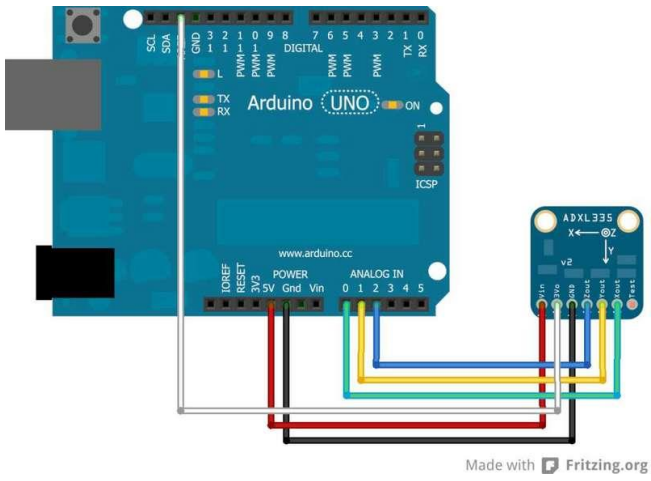
A második az **Adafruit** által gyártott:



11.1.6. ábra: Adafruit ADXL335 gyorsulásmérő

11.1.7. táblázat: Adafruit gyorsulásmérő bekötése Arduinoba:

Adafruit	Self-Test	Z-Out	Y-Out	X-Out	Ground	Vin
Arduino	sehova	A1	A2	A3	GND	5V



11.1.8. ábra: Adafruit gyorsulásmérő Arduinoval

Az adafruit.com oldalon megtalálható, hogyan lehet kalibrálni ezt a szenzort: http://learn.adafruit.com/adafruit-analog-accelerometer-breakouts/programming

A kódban az analóg 4 és 5 pin speciális funkcióját használjuk. Az I2C kommunikációra a 13. fejezetben a Wire Könyvtár használata kapcsán részletesen kitérünk.

Arduino kód:

```
/*
 ADXL3xx
 Kiolvassuk a gyorsulásmérőből az x, y, z tengelyen mért sebességet és soros
portra küldjük az adatokat. // Analóg gyorsulásmérő ADXL3xx Sparkfun.

 Áramkör:
  Arduino analóg 0: gyorsulásmérő self test pinje
  Arduino analóg 1: gyorsulásmérő z-axis
  Arduino analóg 2: gyorsulásmérő y-axis
  Arduino analóg 3: gyorsulásmérő x-axis
  Arduino analóg 4: gyorsulásmérő ground
  Arduino analóg 5: gyorsulásmérő vcc

 created 2 Jul 2008 by David A. Mellis
 modified 30 Aug 2011 by Tom Igoe
*/

// állandókkal meghatározzuk a pineket:
const int groundpin = 18;       // A4-szenzor: ground
const int powerpin = 19;       // A5-szenzor: voltage
const int xpin = A3;           // szenzor: x-axis
const int ypin = A2;           // szenzor: y-axis
const int zpin = A1;           // szenzor: z-axis

void setup() {
  // inicializáljuk a soros kommunikációt:
  Serial.begin(9600);

  // A földelés pint és a feszültség pint az Arduino analóg
  // pinjeibe kötjük. Ha az Arduino 5 V és GND pinjeit
  // használjuk, akkor az alábbi kód eltávolítható:
  pinMode(groundpin, OUTPUT);
  pinMode(powerpin, OUTPUT);
  digitalWrite(groundpin, LOW);
  digitalWrite(powerpin, HIGH);
}

void loop() {
  // kiolvassuk és kiírjuk az x-tengely értékét:
  Serial.print(analogRead(xpin));
```

```
  // az értékek közé kiír egy elválasztó fület
  Serial.print("\t");
  // kiolvassuk és kiírjuk az y-tengely értékét:
  Serial.print(analogRead(ypin));
  // az értékek közé kiír egy elválasztó fület
  Serial.print("\t");
  // kiolvassuk és kiírjuk a z-tengely értékét:
  Serial.print(analogRead(zpin));
  // serial monitorra küldi:
  Serial.println();
  // késleltetés a kiolvasások között:
  delay(100);
}
```

A kiolvasott érték szenzoronként változó, kb. 300–700 közé esik. Az ADXL322 2g gyorsulásmérőnél a vízszintes (amikor a szenzor párhuzamos a földdel) értéke 512 körüli.

Az alábbi táblázat csak nagyságrendi adatokat mutat, és egy tengelyre vonatkozik:

11.1.7. táblázat: Gyorsulásmérő dőlésszöge és kiolvasott értéke (minta, szenzoronként eltérő)

fok	-90	-80	-70	-60	-50	-40	-30	-20	-10	0
Szenzor	662	660	654	642	628	610	589	563	537	510
fok	10	20	30	40	50	60	70	80	90	
Szenzor	485	455	433	408	390	374	363	357	355	

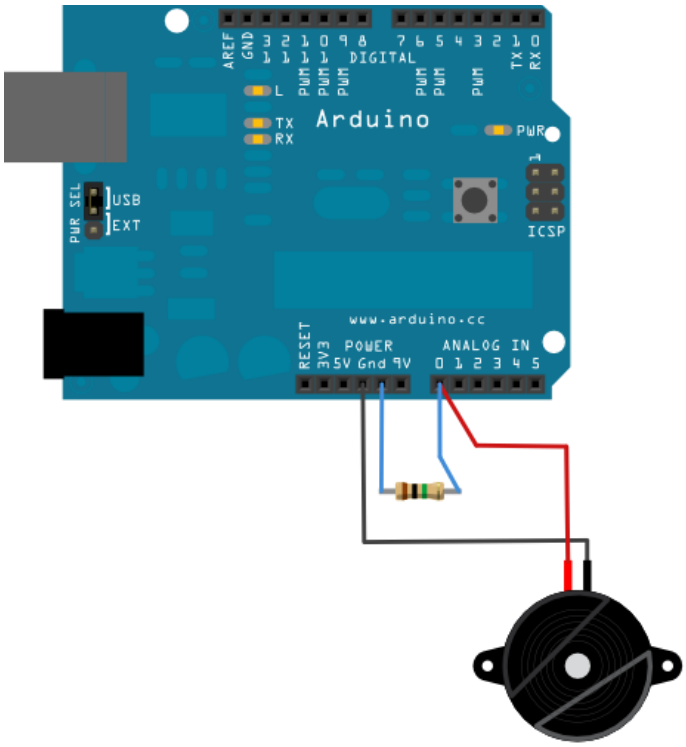
11.2. Piezo szenzor

A piezo elektromos kristályszerkezetben összenyomás hatására feszültség keletkezik. A piezo hangszórók csak megfelelő polaritással működnek, mely azt jelenti, hogy meghatározott irányban kell, hogy átmenjen rajta az áram, ezért a fekete vagy kék vezetékét a földbe, míg a pirosat az analóg pinbe kell kötni. Ezt az analóg pint szintén le kell földelni úgy, hogy az 1 megaohmos ellenálláson keresztül vezetve átkötjük a GND pinbe, ahogy a 11.2.2. ábrán látható. A példában használt típus: Piezo hangszóró 12 V DC gerjesztő áramkörrel.

Műanyag borítás nélkül is beszerezhető a piezo, ilyenkor fényes lemezeknek látszanak és könnyebb őket szenzorként használni, pl. lehet nyomogatni. Akkor működnek a legjobban, ha az érzékelő felületük erősen neki van támasztva valaminek, vagy fel van ragasztva.



11.2.1. ábra: Piezo érzékelő



11.2.2. ábra: Piezo érzékelő bekötése

 © Harsányi Réka, Juhász Márton András

A lejjebb található kódban a piezóból érkező adatok egy általunk megadott küszöbértékhez vannak viszonyítva. Próbáljuk meg növelni vagy csökkenteni ezt az értéket, hogy kihasználjuk a szenzor teljes érzékenységi tartományát.

Arduino kód:

```
/* Knock Sensor / Kopogás szenzor
   Piezo szenzort használunk a kopogás érzékelésére. Analóg bemeneten át
olvassuk ki az értéket, amit a küszöbértékhez viszonyítunk. Ha a bejövő érték
nagyobb annál, akkor serial monitoron kiírjuk a „knock” szót és felkapcsoljuk a
beépített LED-et.

Áramkör:
    * + a piezo pozítív szára megy A0 pinbe,
    * - a piezo negatív lába a földelésbe,
    * 1 megaohm ellenállás az A0 és a földelés között.

   created 25 Mar 2007 by David Cuartielles
   modified 30 Aug 2011 by Tom Igoe
   */


// állandókkal meghatározzuk a pineket
const int ledPin = 13;      // beépített LED
const int knockSensor = A0; // piezo + szára const int threshold = 100;  //
küszöbérték


// változók:
int sensorReading = 0;      // A0-ból kiolvasott értékhez int ledState = LOW;
      // LED állapota

void setup() {
 pinMode(ledPin, OUTPUT); // LED pin kimenetként funkcionál
 Serial.begin(9600);       // bekapcsoljuk a soros portot
}

void loop() {
  // kiolvassuk a szenzort, értékét a
  //  sensorReading változóba tesszük:
  sensorReading = analogRead(knockSensor);

  // ha a kiolvasott érték nagyobb a küszöbértéknél
  if (sensorReading >= threshold) {
    // átkapcsoljuk a LED állapotát ellentétesre:
    ledState = !ledState;
    // és kiküldjük a pinre:
    digitalWrite(ledPin, ledState);
    // a „Knock!" karakterstinget a számítógépre küldjük
```

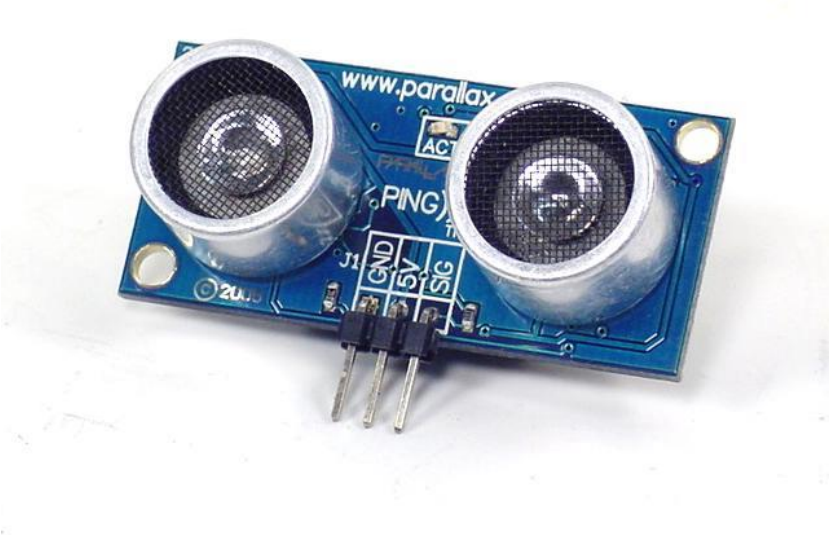
```
    // mindig új sorba írja
    Serial.println("Knock!");
  }
  delay(100);
  // késleltetés, hogy elkerüljük a
  // soros port puffer túltöltését
}
```

11.3. Ultrahangos távolságszenzor

Ping))) ultrahangos távolságszenzor, gyártó: Parallax

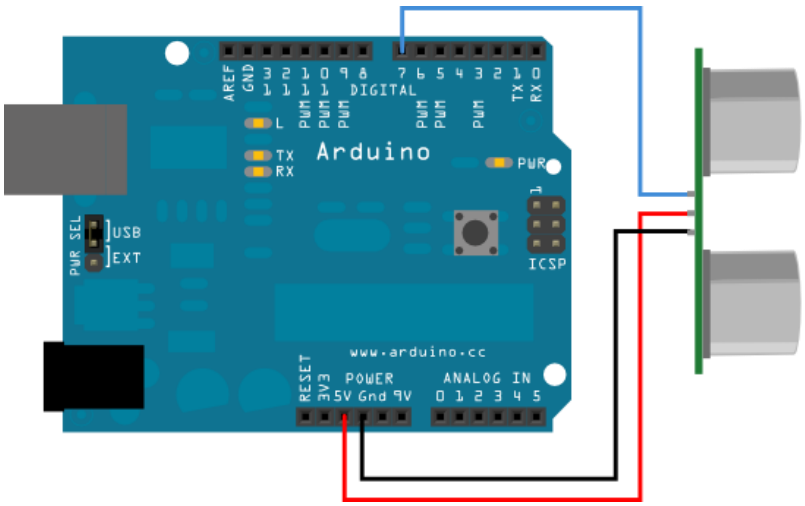
A tőle 2–3 cm-re lévő tárgyakat érzékeli.

Az alábbiak szerint működik: Kiküld egy ultrahang jelet és a visszhangot figyeli, ami akkor keletkezik, ha az ultrahang visszaverődik a tárgyról. Az Arduino egy rövid impulzust küld, hogy előidézze az érzékelést, majd ugyanazon a pinen várja vissza az impulzust (ehhez a **pulseIn()** függvényt használja). A második impulzus időtartama megegyezik azzal az idővel, amit az ultrahang tett meg a szenzorból a tárgyig, majd arról visszaverődve a szenzorig – ezt számítja át a távolság meghatározásához.



11.3.1. ábra: Ping))) ultrahangos távolságszenzor

Bekötése: A földelés pineket és a feszültség pineket összekötjük az Arduinoval, a SIG (signal) pint pedig az Arduinon keresztül a digitális 7 pinbe vezetjük.



11.3.2. ábra: Ping))) ultrahangos távolságszenzor bekötése

Arduino kód:

```
/* Ping))) Sensor / Ultrahangos távolságérzékelő szenzor (Parallax)

   Áramkör
    * +V PING))) bekötve Arduino +5 V-ba
    * GND PING))) bekötve Arduino GND-be
    * SIG PING))) bekötve Arduino digitális 7 pinbe

   created 3 Nov 2008 by David A. Mellis
   modified 30 Aug 2011 by Tom Igoe
  */

// állandóval meghatározzuk a szenzor pint:
const int pingPin = 7;

void setup() {
  // inicializáljuk a soros kommunikációt:
  Serial.begin(9600);
}


void loop() {
  // long változóban tároljuk a ping idejét
  // a távolság eredményét pedig inch-ben és cm-ben
  long duration, inches, cm;

  // Előidézzük a PING)))-et egy minimum
  // 2 mikroszekundumos HIGH impulzussal.
  // Egy rövid LOW impulzus előzi meg
  // a HIGH impulzust, hogy tisztán kapjuk meg:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
```

```
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  // Ugyanazt a pint használjuk a visszajövő
  // impulzus kiolvasásához, ahol a HIGH impulzus
  // időtartama a kiküldéstől a visszaérkezésig tart:
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);

  // az időtartamot távolsággá alakítjuk:
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);

  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);
}


long microsecondsToInches(long microseconds) {
/* Parallax's használati utasítása szerint, 73.746 mikroszekundum egy inchnek
felel meg. Mivel az impulzus oda-vissza megy, a kapott adatot elosztjuk 2-vel.
Lásd: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf */

   return microseconds / 74 / 2;
}


long microsecondsToCentimeters(long microseconds) {
/* A hangunk sebessége 340 m/s vagy 29 centiméterenként. Mivel oda-vissza
mértük az impulzust, a valós távolsághoz el kell osztanunk 2-vel: */
    return microseconds / 29 / 2;
}

--
```

11.4. Sharp infravörös távolságszenzor

A távolságszenzorok analóg vagy digitális kivitelben léteznek. Az egyes típusok különböző távolságokban mérnek. Háromszögeléses módszerrel működik, azaz kibocsát egy infravörös fényt, ami a tárgyakról bizonyos szögben visszaverődve a szenzor optikáján át a CCD-re érkezik (hogy azon belül hova, azt a távolság határozza meg).

A kódban kiolvassuk az értéket és serial monitoron kiírjuk. A példában a GP2D12 Sharp szenzort használjuk. Hogy a szenzornak melyik lába a feszültség (Vcc), a föld (Gnd) és az adatkimenet (Vo), azt a datasheetben megtaláljuk. Értelem szerűen a Vcc pint az Arduino 5V pinjébe, a Vo pint az analóg bementi pinbe, a Gnd-t pedig a földbe kötjük.

SHARP

GP2D12

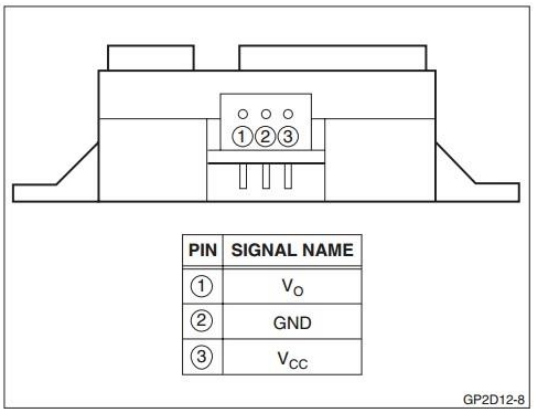
Optoelectronic Device

FEATURES

- Analog output
- Effective Range: 10 to 80 cm
- LED pulse cycle duration: 32 ms
- Typical response time: 39 ms
- Typical start up delay: 44 ms
- Average current consumption: 33 mA
- Detection area diameter @ 80 cm: 6 cm

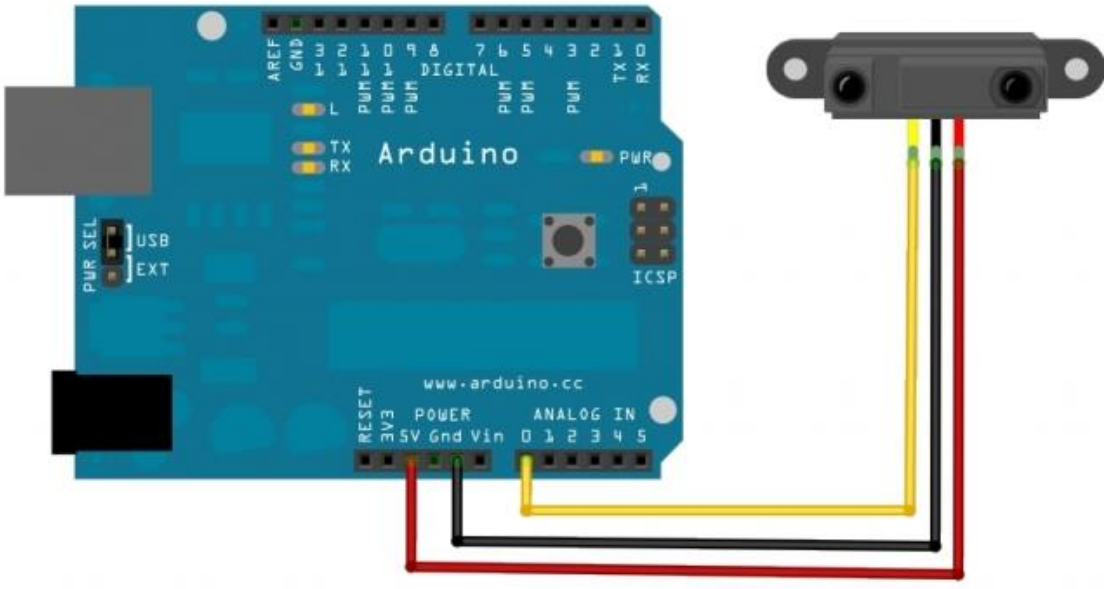
DESCRIPTION

The GP2D12 is a distance measuring sensor with integrated signal processing and analog voltage output.



PIN	SIGNAL NAME
①	V_O
②	GND
③	V_{CC}

Figure 1. Pinout



11.4.1. ábra: Sharp GP2D12 adatlapja és bekötése az Arduinoba

Arduino kód:

```
int sensorpin = 0;
// a Sharp szenzor Arduino pinje
int val = 0;
//inicializáljuk a változót a szenzor értékhez

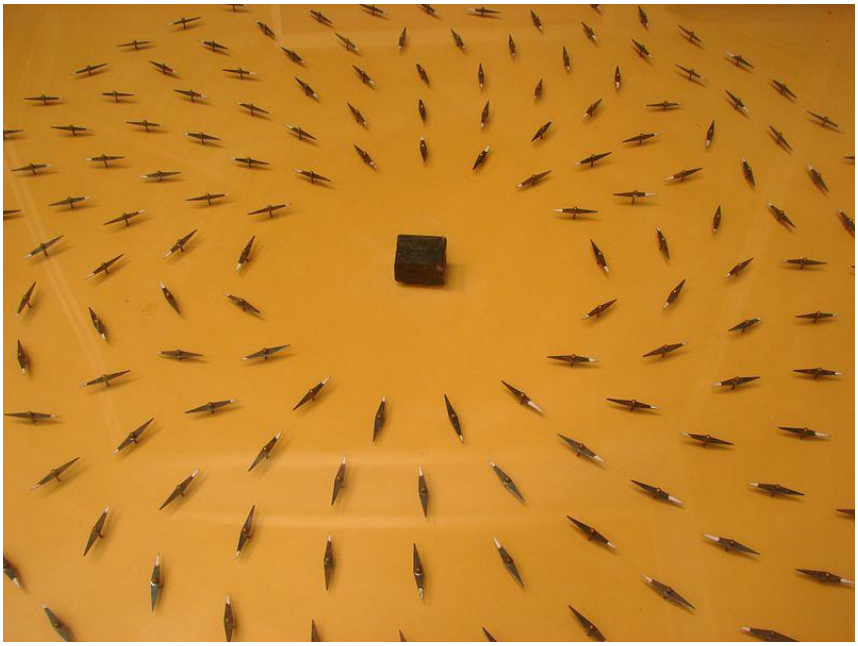
void setup() {
        Serial.begin(9600);
        // soros port bekapcsolása
}

void loop() {
        val = analogRead(sensorpin);
        // kiolvassuk a szenzort
        Serial.println(val);
        // az értéket serial monitoron kiírjuk
        delay(100);
        // késleltetés a kiolvasások között
}
```

11.5. Motorok fajtái és működése

11.5.1. Az elektromágneses mező

Az elektromágneses teret az elektromos mező változása, azaz az elektronok áramlása idézi elő. Az elektromos mező a töltéssel rendelkező részecskékből származik, a mágneses mező pedig ezek mozgásából, a kettő együtt eredményezi az elektromágneses teret. Általában a mágneses mező irányát és nagyságát az úgynevezett erővonalakkal jelezzük, ezek a mágnes északi sarkától a déli felé haladnak. Szemléltetni úgy lehet, hogy vasreszelék fölé helyezett mágnes hatására a reszelék összefüggő vonalakká rendeződik.



11.5.1. ábra: Mágneses tér iránytűkkel (© Nevit Dilmen found at Wikimedia commons)

Minden vezetőnek, amiben áram folyik, van mágneses tere, melynek erővonalai koncentrikus körök formájában körülveszik a vezetőt

Az elektromágnes esetén egy tekercsbe helyezett lágyvas hozza létre a mágneses teret addig, amíg a tekercsen áramot vezetünk át. Formájukat tekintve lehetnek rúd, patkó vagy gyűrű alakúak, ezeknek eltérő a mágneses áramlásuk. A legjobb a teljesen zárt vasmaggal rendelkező elektromágnesé.

11.5.2. Az elektromágneses indukció

Mágneses térben mozgó vezetőben elektromos feszültség keletkezik. Ez teszi lehetővé a dinamók, a villanymotorok és a transzformátorok működését. Két csoportra lehet osztani ezt a jelenséget, a nyugalmi és a mozgási indukciókra. Nyugalmiba pl. a transzformátorok, a mozgásiba pl. a motorok tartoznak. Nyugalmi esetén a fluxus (egy erőtérnek egy adott felületen való áthatolását jellemző mennyiség) hozza létre az indukciót, mivel sem a vezető, sem a mágneses mező nem mozog.

Számunkra a mozgási indukció megértése fontos, mivel villanymotorokat fogunk használni. Ebben az esetben vagy a vezető, vagy a mágneses mező, vagy mindkettő elmozdul egymáshoz viszonyítva. Legelterjedtebb formája a forgómozgás. A villanymotorok az elektromos energiából nyerik mechanikus energiájukat.

Három motortípusra térünk ki – mivel ezekből létezik kisfeszültséggel működő változatban is –, ezek a villanymotorok, a léptetőmotorok és a szervomotorok.

11.5.3. Egyenáramú villanymotor

Angolul DC motor. Az elektromágneses indukció elvén működik, az elektromos áramból nyert energiát alakítja át mechanikus energiává, konkrétabban forgó mozgássá.

> *„**Kommutátor**: áramfordító, egyenáramú dinamók és motorok, valamint bizonyos fajtájú váltóáramú motorok tengelyére szerelt, vele együtt forgó, egymástól elszigetelt vörösréz-szegmensekből álló henger. Minden szegmenshez egy-egy tekercsvég kivezetés van forrasztva. A kommutátorra rugók szorítják rá a réz- vagy szénkeféket, melyek az áramot be, illetve elvezetik. A forgórész valamely vezetőjében az állórész két pólusa között való elhaladásakor az indukált áram iránya megváltozik, azonban a keféken keresztül csak egyirányú áram folyik, mert a tekercsvéghez forrasztott kommutátor szegmens addigra már elhaladt a kefe alatt és a következő tekercs szegmensével nyer érintkezést, melyben még nem változott meg az áramirány.” (hu.metapedia.org)*

> *„**Mágneses fluxusnak** hívjuk, ami szemléletesen a felületet metsző mágneses indukcióvonalak száma” (hu.wikipedia.org/fluxus)*

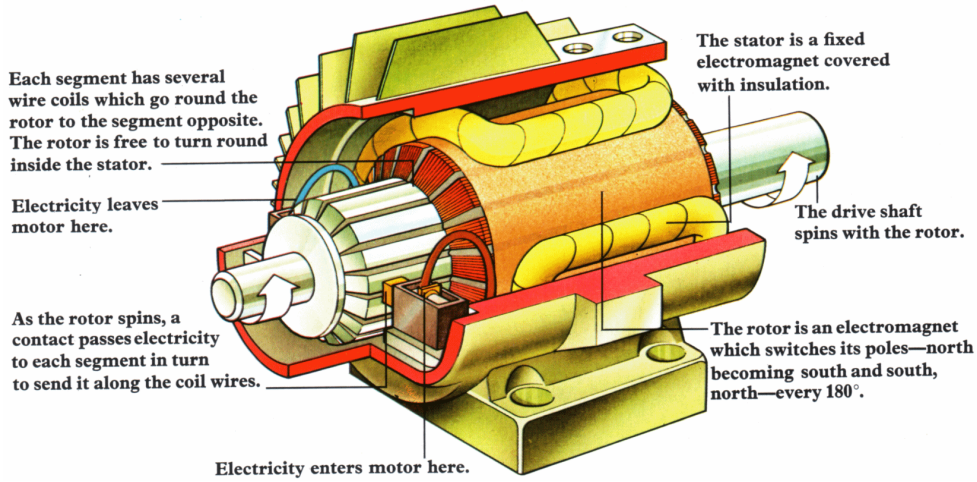
> *„**Armatúra:** elektromos gépeknek az az alkatrésze, amelyben a feszültség indukálódik.” (kislexikon.hu)*

Az egyenáramú motor működése a mágneses pólusok vonzásán és taszításán alapul. A motor két kivezetésére rákapcsoljuk az áramforrás pozítív és negatív kivezetését. Egy mágneses térben elhelyezett tekercsbe ha áramot vezetünk, akkor az indukálódott elektromágneses mező kölcsönhatásba lép az állandó mágnesek erőterével. Ezért az amatúra (a tekercs, ami a motor forgó része) a pólusok vonzó, illetve taszító hatása következtében elmozdul. Ha az álló és forgórész pólusa azonos akkor vonzzák, ha ellentétes akkor taszítják egymást. Az áramváltást, azaz a pólusok felcserélését a kommutátor biztosítja, abban a pillanatban amikor a forgó és állórész pólusa megegyezik, átkapcsolja ellentétesre, így ismét taszítják egymást ezek az alkatrészek, ezért folytatódik a forgás. Az erőtér és az armatúra különféle összekapcsolása különböző belső fordulatszámot eredményez. A villanymotor sebességét befolyásolhatjuk az armatúra feszültségének vagy a gerjesztő áramnak a változtatásával. Villanymotor meghajtó mikrochippel is vezérelhetjük őket, úgynevezett H-híddal, melyre később térünk ki. A villanymotor jellemző adatai a feszültség és az áramerősség.

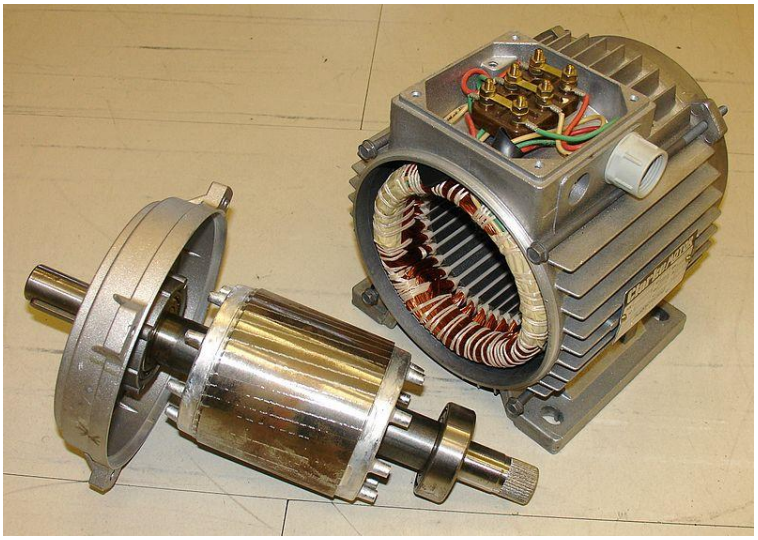
 © Harsányi Réka, Juhász Márton András



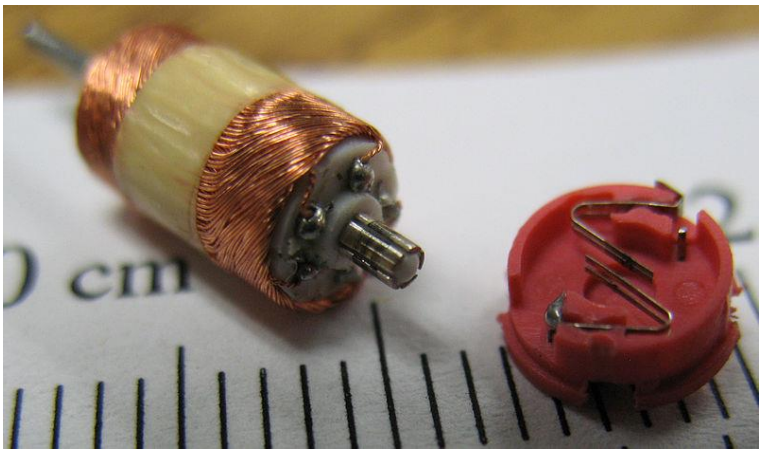
11.5.2. ábra: Villanymotorok



11.5.3. ábra: Villanymotor szerkezete



11.5.4. ábra: Villanymotor álló- és forgórésze

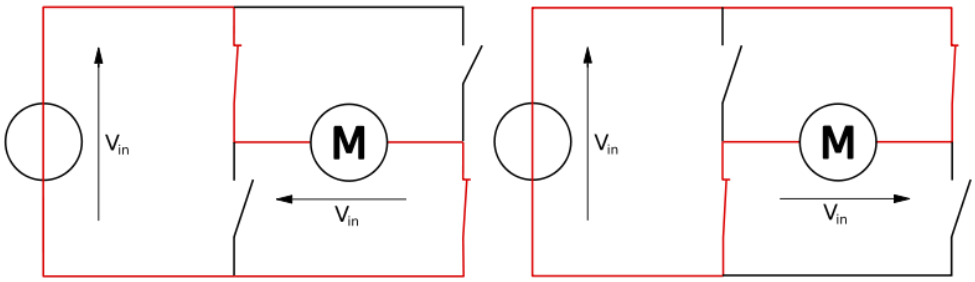


11.5.5. ábra: Nagyon kis méretben is léteznek, általában játékokban használják őket.

11.6. Egyenáramú villanymotor vezérlése H-híddal

Mivel a villanymotorok – ha szimplán rákötjük őket az áramra – egy irányba és adott sebességgel forognak, ezért ha irányítani akarjuk, egy motorvezérlő áramkörre, úgynevezett H-hídra van szükségünk. Ez egy elektromos áramkör, amit házilag is megépíthetünk, de mikrochip formájában meg is vásárolhatunk. A megfelelő lábakra bekötve a motorunkat, programozással vezérelhetjük a forgás irányát és sebességét. Arduinoval 3 voltos DC motorokat fogunk meghajtani L293N integrált áramkör használatával.

 © Harsányi Réka, Juhász Márton András

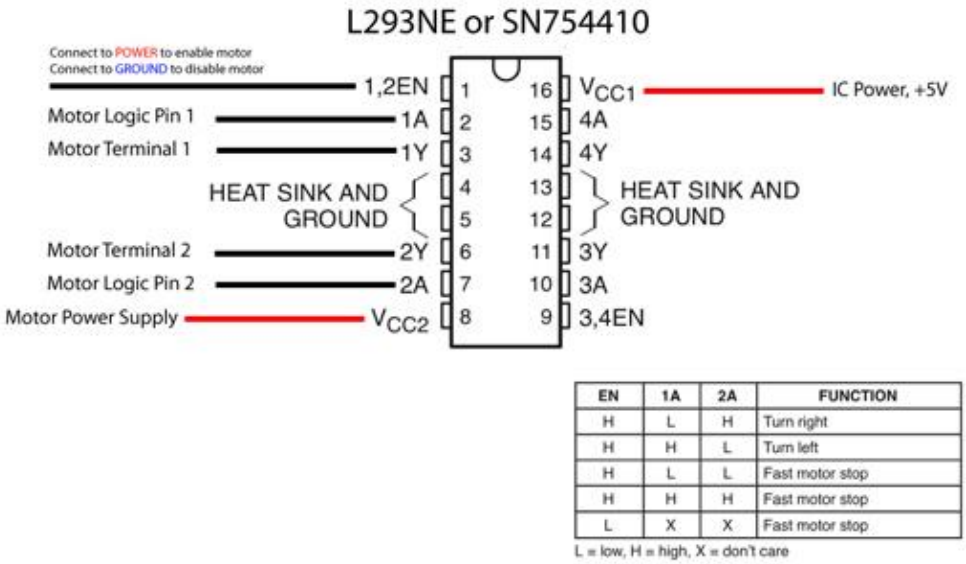


11.5.11. ábra: A H-híd két alapállapotának kapcsolási rajza



11.5.12. ábra: L293NE H-híd

Szükségünk van egy 10 kiloohmos ellenállásra, egy kétállású SPDT kapcsolóra (on-on), 1 DC motorra (3 V elég, az Arduino csak 5 V-ot bír meghajtani) és egy H-hídra. A H-hídnak mindkét oldalát be kell kötni a feszültségbe és a földelésbe is, de csak azt az oldalát kell bekapcsolni, amelyiket használjuk. (EN lábra kell HIGH jelet küldeni.)

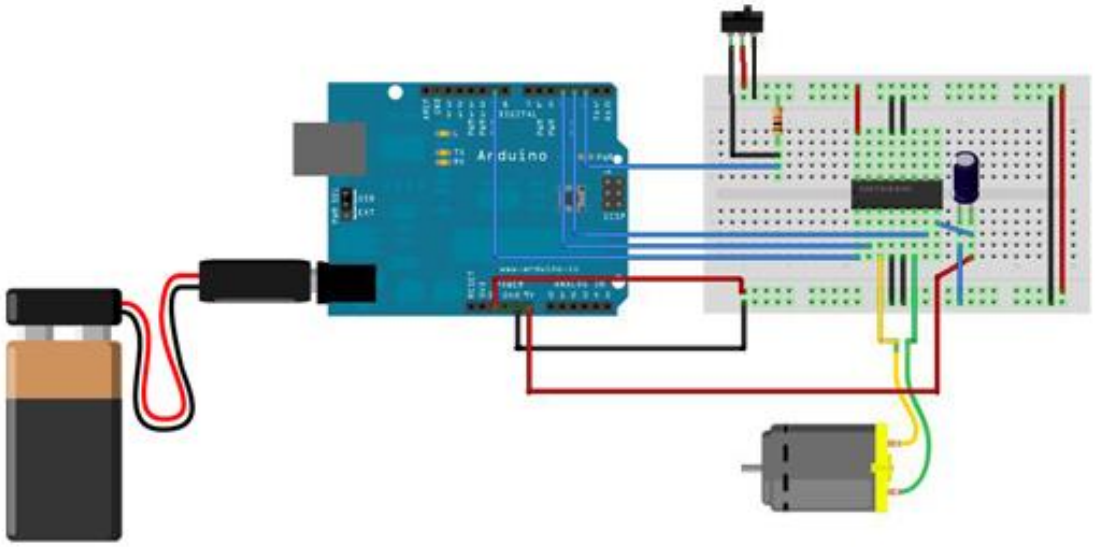


EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

11.5.13. ábra: A H-híd lábainak kiosztása

Összeépítés: próbapanelen, egy motorral, de most a 9 V elemet ne kössük rá, mert az USB kábel is elég ahhoz, hogy forogjon egy 3 V-os motor.



11.5.14. ábra: DC motor és L293NE h-híd bekötése próbapanelen

Arduino kód:

```
// kód forrása:  http://itp.nyu.edu/physcomp/Labs/DCMotorControl

const int switchPin = 2;    // kapcsoló bemenete
const int motor1Pin = 3;    // H-híd 1. lába
const int motor2Pin = 4;    // H-híd 2. lába
const int enablePin = 9;    // H-híd engedélyezése pin
const int ledPin = 13;      // LED

void setup() {
    // a kapcsoló pint bemenetként használjuk:
    pinMode(switchPin, INPUT);

    // a többit kimenetként:
    pinMode(motor1Pin, OUTPUT);
    pinMode(motor2Pin, OUTPUT);
    pinMode(enablePin, OUTPUT);
    pinMode(ledPin, OUTPUT);

    //engedélyezni kell a H-hidat, hogy működjön,
    //azaz be kell kapcsolni

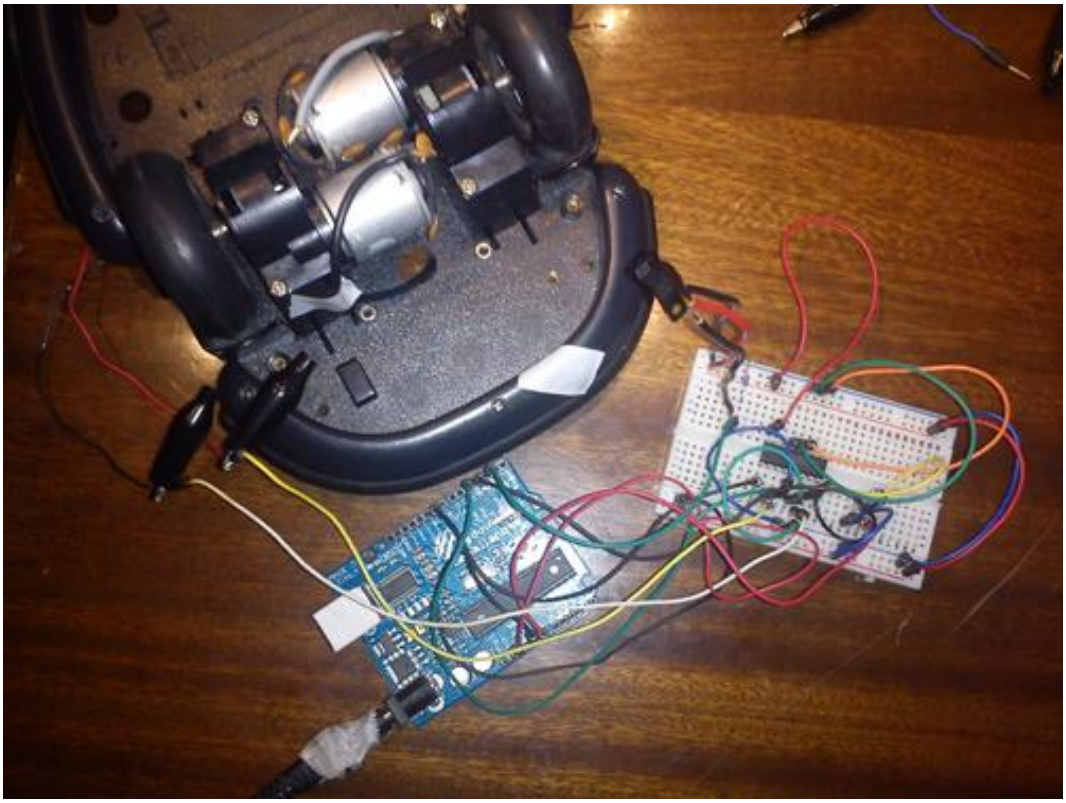
    digitalWrite(enablePin, HIGH);

    // Háromszor felvillan a LED. Ez egyszer történik meg.
```

```
    blink(ledPin, 3, 100);
}

void loop() {
    // ha a kapcsoló be van kapcsolva, akkor ez egyik irányba forog a motor
    if (digitalRead(switchPin) == HIGH) {
       digitalWrite(motor1Pin, LOW);
       // a H-híd 1. lábát kikapcsolja
       digitalWrite(motor2Pin, HIGH);
       // a H-híd 2. lábát bekapcsolja
    }
    // ha ki van kapcsolva, akkor a másik irányba:
    else {
       digitalWrite(motor1Pin, HIGH);
       // a H-híd 1. lábát bekapcsolja
       digitalWrite(motor2Pin, LOW);
       // a H-híd 2. lábát kikapcsolja
    }
}

/*
 LED villogtatása
 */
void blink(int whatPin, int howManyTimes, int milliSecs) {
    int i = 0;
    for ( i = 0; i < howManyTimes; i++) {
       digitalWrite(whatPin, HIGH);
       delay(milliSecs/2);
       digitalWrite(whatPin, LOW);
       delay(milliSecs/2);
    }
 }
```



11.5.15. ábra: A H-hídra kötött DC motor

 © Harsányi Réka, Juhász Márton András

12. Műveletek karakterláncokkal

Mint azt a soros kommunikációról szóló fejezetben már érintettük, a *char* olyan adattípus, amely 1 byte memóriát foglal és az ASCII táblázatnak megfelelően egy karakter értékét tárolja. Mivel a karakterek mindig számként tárolódnak, ezért műveleteket is végrehajthatunk rajtuk, például „A” + 1 = 66 – mert az A értéke a táblázatban 65.

Az egyszerű karakterláncokat, vagyis a karaktereket tartalmazó tömböket is (kisbetűvel) stringeknek nevezzük, és a string osztály példányait is (nagybetűvel) stringeknek hívjuk. Tehát karakterláncot kétféleképpen tudunk definiálni, string adattípusként – amin a string osztály függvényeivel komplexebb műveleteket is végezhetünk –, vagy egyszerű karaktertömbként.

12.1. Karaktertömbök

Fontos megjegyeznünk, hogy a fordítóprogram a karakterlánc végét egy ASCII 0 *(null)* karakterről ismeri meg, így például serial.print() függvény a string nyomtatásakor nem szalad tovább oda nem illő memóriaterületekre. Ez azt is jelenti, hogy mindig egy karakterrel hosszabbnak kell lennie a tömbnek, mint a tárolni kívánt szöveg. A karaktereket tartalmazó tömböket az alábbi példák szerint tudjuk deklarálni:

- Iniciáló értékek megadása nélkül, csak az elemek számával – a fordító lezárja *null* karakterrel:

  char Str1[15];

- **Karakterenként aposztrófokkal** – az utolsó üres helyet a fordító feltölti *null* karakterrel, de mi is megadhatjuk a sort lezáró karaktert:

```
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
```

- **Szövegként idézőjelekben** – ha nem adjuk meg az elemek számát, a fordító megszámolja a karaktereket és hozzáadja a végéhez a lezáró *null* karaktert, ahogy az üres helyet is feltölti:

```
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

 © Harsányi Réka, Juhász Márton András

Bár lehetséges karaktertömb definiálása sorzáró *null* karakter nélkül – hacsak nem ez a célunk –, kerüljük el, mivel a legtöbb függvény, amit karakterláncokon alkalmazunk, nincs erre felkészítve. Ha a programunk futása közben furcsa, oda nem illő karakterek jelennek meg, akkor valószínűleg ezt a hibát vétettük.

Hosszú szövegeket így is megadhatunk:

```
char myString[] = "ez az első sor"
" ez a második sor"
" stb…";
```

Amikor sokféle szöveggel dolgozunk, például LCD kijelző vagy soros terminál menüjének szövegeit tároljuk, sokkal kézenfekvőbb a stringeket is egy tömbbe gyűjteni. Mivel a stringek önmagukban is tömbök, ezért kétdimenziós tömböt hozunk létre:

```
char* myStrings[]={"This is string 1", "This is string 2", "This is
string 3", "This is string 4", "This is string 5","This is string 6"};
```

Az adattípus után a csillag (char*) azt jelöli, hogy ez egy mutatókból (pointer) álló tömb. Valójában minden tömb neve pointer, ezért szükséges a megkülönböztető csillag jelölés a tömböket tartalmazó tömbök létrehozásakor.

Példakód

```
char* myStrings[]={"This is string 1", "This is string 2", "This is string 3",
"This is string 4", "This is string 5","This is string 6"};

void setup(){
   // megnyitjuk a soros portot
   Serial.begin(9600);
}

void loop(){
   for (int i = 0; i < 6; i++){
      // kinyomtatjuk a karakterláncokat a kétdimenziós tömbből elemenként
      Serial.println(myStrings[i]);
      delay(500);
   }
}
```

12.2. A string osztály példányai

A string osztály kifejezetten szöveg alapú adattípusok manipulálására jött létre. Az egyszerű karaktertömböktől eltérően az osztály példányain olyan függvényekkel is

végezhetünk műveleteket, mint a vágás, az összefűzés, szövegrészletek keresése és cseréje. Ugyan a string adattípus használata ebből kifolyólag több memóriát igényel, mint a karakterláncé használatuk mégis sokszor megtérül.

String osztály példányait a következő módokon hozhatjuk létre:

- Konstansként idézőjelek közé zárva:

```
String stringOne = "Hello String";
```

- **String() függvény** hívásával konstans karaktert átkonvertálva:

```
String stringOne =  String('a');
```

- Konstansból objektummá alakítva:

```
String stringOne =  String("This is a string");
```

- Több stringből létrehozva:

```
String stringOne =  String(stringTwo + "with more");
```

- Szám típusú változóból:

```
String stringOne =  String(13);
```

- Szám típusú változóból, megadva a számrendszer alapját:

```
String stringOne =  String(analogRead(0), DEC);
String stringOne =  String(45, HEX);
String stringOne =  String(255, BIN);
String stringOne =  String(millis(), DEC);
```

12.2.1. String osztály további függvényei

12.2.1.1. charAt()

Visszaadja a string típusú változó n-edik karakterét.

Szintaxisa
string.charAt(n).

Paraméterei
string: az a string típusú változó, amiben;
n: az n-edik karaktert szeretnénk elérni.

12.2.1.2. setCharAt()

Lecserél egy adott karaktert a string változón belül. Csak a string hosszán belül érvényes.

Szintaxisa
setCharAt(index,c).

Paraméterei
string: string típusú változó neve;
index: a lecserélendő karakter sorszáma;
c: a lecserélendő karakter – char típusú változó.

12.2.1.3. compareTo()

Két stringet hasonlít össze, az egyenlőségüket vizsgálja, egyenlőtlenség esetén pedig az ABC sorrendben betöltött sorrendjüket.

Szintaxisa
string.compareTo(string2).

Paraméterei
string: az egyik összehasonlítandó string;
string2: a másik összehasonlítandó string.

Kimenete
negatív érték: string előrébb való string2-nél;
0: a két string megegyezik;
pozitív érték: string2 előrébb való string-nél.

Példakód

Stringek egybevetésekor használhatunk összehasonlító műveleti jeleket is, mint az egyenlő == , nem egyenlő **!=** , kisebb/nagyobb < > , nagyobb egyenlő >= , kisebb egyenlő <= , az *equals()* és *equalsIgnoreCase()* függvényeket is. A két függvény közül az előbbi megegyezik a == jellel, az utóbbi csak abban tér el, hogy figyelmen kívül hagyja a kis- és nagybetűk közötti különbségeket. Számokat is tartalmazó szövegek összehasonlításakor vegyük figyelembe, hogy a fenti függvények a stringeket karakterenként vetik egybe, így például könnyen 999 > 1000 eredményre jutnak, mivel a 9 később következik a számok sorában, mint az 1.

```
  /*created 27 July 2010
modified 2 Apr 2012
```

```
 by Tom Igoe */

String stringOne, stringTwo;

void setup() {
  Serial.begin(9600);
  stringOne = String("this");
  stringTwo = String("that");
  }

void loop() {
  // a két string megegyezik
  if (stringOne == "this") {
    Serial.println("StringOne == \"this\"");
  }
  // a két string különbözik
  if (stringOne != stringTwo) {
    Serial.println(stringOne + " =! " + stringTwo);
  }

  // a két string nem egyezik (kis- és nagybetűk különbsége is számít):
  stringOne = "This";
  stringTwo = "this";
  if (stringOne != stringTwo) {
    Serial.println(stringOne + " =! " + stringTwo);
  }
  // az equals() függvényt is használhatjuk az egyenlőség vizsgálatára
  if (stringOne.equals(stringTwo)) {
    Serial.println(stringOne + " equals " + stringTwo);
  }
  else {
    Serial.println(stringOne + " does not equal " + stringTwo);
  }

  // ha nem fontos a kis- és nagybetűk közötti különbség
  if (stringOne.equalsIgnoreCase(stringTwo)) {
    Serial.println(stringOne + " equals (ignoring case) " + stringTwo);
  }
  else {
    Serial.println(stringOne + " does not equal (ignoring case) " + stringTwo);
  }

  // számot tartalmazó stringet hasonlítunk számértékhez
  stringOne = "1";
  int numberOne = 1;
```

```
  if (stringOne.toInt() == numberOne) {
    Serial.println(stringOne + " = " + numberOne);
  }


  // két számot tartalmazó string összehasonlítása
  stringOne = "2";
  stringTwo = "1";
  if (stringOne >= stringTwo) {
    Serial.println(stringOne + " >= " + stringTwo);
  }

  // példa az ABC sorba rendezésre
  stringOne = String("Brown");
  if (stringOne < "Charles") {
    Serial.println(stringOne + " < Charles");
  }

  if (stringOne > "Adams") {
    Serial.println(stringOne + " > Adams");
  }

  if (stringOne <= "Browne") {
    Serial.println(stringOne + " <= Browne");
  }


  if (stringOne >= "Brow") {
    Serial.println(stringOne + " >= Brow");
  }

  // compareTo() használata
  stringOne = "Cucumber";
  stringTwo = "Cucuracha";
  if (stringOne.compareTo(stringTwo) < 0 ) {
    Serial.println(stringOne + " comes before " + stringTwo);
  }
  else {
    Serial.println(stringOne + " comes after " + stringTwo);
  }

  // compareTo() akkor is hasznos, ha a stringek számokat tartalmaznak

  while (true) {
    stringOne = "Sensor: ";
```

```
    stringTwo= "Sensor: ";

    stringOne += analogRead(A0);
    stringTwo += analogRead(A5);

    if (stringOne.compareTo(stringTwo) < 0 ) {
      Serial.println(stringOne + " comes before " + stringTwo);
    }
    else {
      Serial.println(stringOne + " comes after " + stringTwo);
   }
  }
}
```

12.2.1.4. concat()

Összekapcsol két stringet úgy, hogy a másodikat folytatólagosan az elsőhöz ragasztja és a produktumot egy új stringbe tölti. Szám típusú változókat is hozzáadhatunk vele egy szöveghez.

Szintaxisa

string.concat(string,string2);
string.concat(string2);
string += string2.

Paraméterei

string: első változó;
string2: második változó.

Példakód

```
stringOne = "Egy nagy szám: ";
stringTwo = "Program ideje: ";

stringOne += 123456789;
Serial.println(stringOne);
// kimenete "Egy nagy szám: 123456789"

// long típusú változó hozzáadása concat() függvénnyel
stringTwo.concat(millis());
Serial.println(stringTwo);
// kimenete "Program ideje: 43534", vagy épp, ahol a millis()
// számlálója tart
```

12.2.1.5. startsWith()

Megvizsgálja, hogy egy string azzal a karakterlánccal kezdődik-e, mint amit egy másik tartalmaz, és igaz/hamis választ ad vissza.

Szintaxisa
string.startsWith(string2).

Paraméterei
string: string, aminek az elején keresünk;
string2: string, amit keresünk.

Példakód

```
stringOne = "HTTP/1.1 200 OK";
if (stringOne.startsWith("200 OK")) {
  Serial.println("Got an OK from the server");
}
```

12.2.1.6. endsWith()

Megvizsgálja, hogy egy string azzal a karakterlánccal végződik-e, mint amit egy másik tartalmaz, és igaz/hamis választ ad vissza.

Szintaxisa
string.endsWith(string2).

Paraméterei
string: string, aminek az elején keresünk;
string2: string, amit keresünk.

Példakód

```
String sensorReading = "sensor = ";
sensorReading += analogRead(A0);
Serial.print (sensorReading);
if (sensorReading.endsWith(0)) {
  Serial.println(". This reading is divisible by ten");
```

```
}
else {
  Serial.println(". This reading is not divisible by ten");
}
```

12.2.1.7. indexOf() és lastIndexOf()

Szövegen belüli keresést hajthatunk végre a segítségükkel, mely során a keresendő minta lehet karakter (char) vagy string típusú is. Alapértelmezésként az indexOf() a string elejétől kezdi a keresést, de megadhatunk kezdő pozíciót is. A lastIndexOf() abban különbözik, hogy képes a string végéről visszafelé keresni. Mindkét függvény a találat tömbön belüli indexét adja vissza és –1-et, ha nem talál egyezést.

Szintaxisuk
string.indexOf(val);
string.indexOf(val, from);
string.lastIndexOf(val);
string.lastIndexOf(val, from).

Paramétereik
string: amiben keresünk;
val: amit keresünk, lehet *char* vagy *string;*
from: a karakter pozíciója a tömbben, ahonnan a keresés indul.

Példakód

```
  String stringOne = "<HTML><HEAD><BODY>";
int firstClosingBracket = stringOne.indexOf('>');
```

12.2.1.8. length()

Visszaadja a string karaktereinek számát a lezáró *null* karakter nélkül.

Szintaxisa
string.length().

Paramétere
string: aminek a hosszára kíváncsiak vagyunk.

12.2.1.9. trim()

Levágja a szöveget megelőző vagy követő üres helyeket és az eredményt visszatölti az eredeti változóba. Az szünetnek látszó üres karakterek például: SPACE (ASCII 32), tab (ASCII 9), függőleges tab (ASCII 11), sor elejére ugrás (ASCII 13), új sor (ASCII 11).

Szintaxisa

string.trim().

Paramétere

string: amit szeretnénk megváltoztatni.

Példakód

```
void setup() {
   Serial.begin(9600);
}

void loop() {
  // ez itt egy string sok üres karakterrel
  String stringOne = "Hello!       ";
  Serial.print(stringOne);
  Serial.print("<--- end of string. Length: ");
  // kiírja a string hosszát: 13
  Serial.println(stringOne.length());

  // vágjuk le az üres karaktereket
  stringOne.trim();
  Serial.print(stringOne);
  Serial.print("<--- end of trimmed string. Length: ");
  // kiírja a string hosszát, ami vágás után: 6
  Serial.println(stringOne.length());

  // nem csinál semmit, ha az előzőek lefutottak
  while(true);
}
```

12.2.1.10. substring()

Egy string tetszőleges részletét adja vissza, amit a tömb indexeivel határolhatunk. A kezdőpozíción található karakter még bele fog esni a kimásolt karakterláncba, míg a végpozíción található karakter már nem. Ha nem adunk meg végpozíciót, akkor az automatikusan a karakterlánc vége lesz.

Szintaxisa
string.substring(from);
string.substring(from, to).

Paraméterei
string: amiből karakterláncot másolunk;
from: a tömb indexe, ami a kezdőpozíciót jelöli;
to: a tömb indexe, ami a végpozíciót jelöli.

Példakód

```
String stringOne = "Content-Type: text/html";
  if (stringOne.substring(19) == "html") {
}
```

12.2.1.11. replace()

Kicserél két karakterláncot egy stringen belül, de csak akkor, ha amire cserélünk, nem hosszabb, mint az egész string. Máskülönben a fordító hiba nélkül továbblép és az eredeti string változatlan marad.

Szintaxisa
string.replace(substring1, substring2).

Paraméterei
string: a karakterlánc, aminek egy részét cseréljük;
substring1: amit ki szeretnénk cserélni;
substring2: amire ki szeretnénk cserélni.

Példakód

```
void setup() {
     Serial.begin(9600);
     }
}

void loop() {
  String stringOne = "<html><head><body>";
  Serial.println(stringOne);
```

```
    // először csináljunk másolatot az eredeti stringről
  String stringTwo = stringOne;
  // hajtsuk végre a cserét
  stringTwo.replace("<", "</");
  // nyomtassuk ki az eredetit
  Serial.println("Original string: " + stringOne);
  // aztán a módosított stringet is.
     Serial.println("Modified string: " + stringTwo);

  // ezt karakterekkel is megtehetjük
  String normalString = "bookkeeper";
  Serial.println("normal: " + normalString);
  String leetString = normalString;
  leetString.replace('o', '0');
  leetString.replace('e', '3');
  Serial.println("l33tspeak: " + leetString);

  // állj, ha vége
     while(true);
}
```

12.2.1.12. toLowerCase()

Átalakítja egy stringben a nagybetűket kisbetűkké és visszatölti az eredményt az eredeti változóba. Ezzel szemben a *toUppercase()* a kisbetűket alakítja nagybetűkké.

Szintaxisa
string.toLowerCase();
string.toUpperCase().

Paramétere
string: amin a változtatást végrehajtjuk.

12.2.1.13. getBytes()

Előfordulhat, hogy szeretnénk a string típusát átalakítani. Ha byte-okat tartalmazó tömbként szeretnénk a karakterláncot kezelni, akkor ez a függvény kimásolja a string tartalmát byte-onként egy tömbbe.

Szintaxisa
string.getBytes(buf, len).

Paraméterei
string: string változónk, aminek a tartalmát másoljuk;
buf: byte[] (byte tömb) típusú változónk, ahová másolunk;
len: a byte tömbünk hossza (unsigned int típusú).

12.2.1.14. toCharArray()

Hasonlóan az előzőhöz, kimásolja a string tartalmát és egyszerű karaktereket tartalmazó tömbbe tölti.

Szintaxisa
string.toCharArray(buf, len).

Paraméterei
string: string változónk, aminek a tartalmát másoljuk;
buf: char[] (karakter tömb) típusú változónk, ahová másolunk;
len: a karakterlánc hossza (unsigned int típusú).

13. Kommunikációs könyvtárak

Számítástudományban a *könyvtár* olyan előre megírt kódrészeket jelent, amik meghatározott komplex funkciók elvégzésére képesek. Osztályaik és függvényeik felhasználásával a független programokban nincs szükség ezen feladatok ismételt megoldására. Általánosságban olyan csomagokként értelmezzük őket, amik szolgáltatásként teszik elérhetővé a konfigurációs adatokat, konstansokat és kódokat. Például egy matematikai könyvtár (Arduino esetében a math.h) a programnyelvből közvetlenül nem elérhető függvényeket írja le, mint a tangens, kotangens, logaritmus, gyökvonás. A rendkívül hasznos függvényeken túl sok esetben bizonyos típusú hardware elemek használatát könnyítik meg – operációs rendszer hiányában ezeket a könyvtárakat felfoghatjuk úgy is, mint eszközillesztő programokat (device driver). Mivel az Arduino egy nyílt forrású, együttműködő közösség által is fejlesztett rendszer, megkülönböztetünk hivatalos és kontribúciós könyvtárakat. Az előbbieket standardként is szokás nevezni, mivel kiforrottságuk miatt a fejlesztőkörnyezet programcsomagjának részét képezik, és olyan általános funkciókat írnak le, amik végrehajtására minden Arduino-típus képes.

13.1. Könyvtárak szerkezete

Amikor a két fő rutin mellett (setup és loop) szubrutinokat szeretnénk létrehozni, az a **void** *szubrutin_neve*() összefüggés segítségével tehető meg:

```
int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
}

void loop() // az alrutinok végrehajtódnak a fő ciklusban
   {
  dot(); dot(); dot(); // . . . morze S
  dash(); dash(); dash(); // - - - morze O
  dot(); dot(); dot(); //  . . . morze S
  delay(3000); // vár 3 másodpercet majd újrakezdi
}

void dot() // rövid villanást előidéző szubrutin
{
```

```
  digitalWrite(pin, HIGH);
  delay(250);
  digitalWrite(pin, LOW);
  delay(250);
}

void dash() // hosszú villanást előidéző szubrutin
{
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(250);
}
```

Ha a fenti kódból könyvtárat szeretnénk létrehozni, akkor legalább két fájlra lesz szükségünk, az egyik az úgynevezett *fejléc* vagy *header* fájl (kiterjesztése .h), a másik pedig a *forrás* vagy *source* fájl (kiterjesztése .cpp). A fejléc definiálja a könyvtárat – minden kifejezésnek szerepelnie kell benne, amire hivatkozni szeretnénk –, míg a forrásfájl tartalmazza a kódot.

A fejléc már a fájl nevében tartalmazza a könyvtárunk nevét (**Morse.h**):

```
   #ifndef Morse_h // a fájl végén található #endif párja, elkerülhetjük az
   // esetleges hibákat, ha véletlenül kétszer töltenénk be a könyvtárat

   #define Morse_h

   #include "Arduino.h" // #include-dal tudunk könyvtárat betölteni, az
   Arduino.h
   // az alapkönyvtár

class Morse // az osztály olyan változók és függvények gyűjteménye, amik
   // összetartoznak
{
    public: // globális változók, amik a független programból is elérhetők
    Morse(int pin);
    void dot(); // szubrutinjaink, amik kódjai a forrásfájlban vannak
    void dash();
    private: // belső változók, csak a könyvtár használja
    int _pin;
};
#endif
```

A forrásfájlba már betöltjük a fejlécet, hiszen a definiált osztályt és szubrutinjait töltjük meg tartalommal (**Morse.cpp**):

```
#include "Arduino.h"
#include "Morse.h"

Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin; // Morse osztály egyetlen változóját átadjuk belső használatra
}

// hasonlítsuk össze a szubrutinokat az eredeti kódban találhatókkal
void Morse::dot(){
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

Végül a kész könyvtárunkkal a fenti példakódunk lerövidült és univerzálisabbá vált – használata pedig így néz ki:

```
#include <Morse.h> // importáljuk a könyvtárunkat

// morse néven létrehozunk egy Morse osztályt a 13-as kimenetet használva
Morse morse(13);

void setup()
{
   // itt már nem kell tennünk semmit, a pinMode() lefut a forrásfájlunkból
   }

void loop() // a ciklusban lefutnak a szubrutinok
{
  morse.dot(); morse.dot(); morse.dot();
```

```
  morse.dash(); morse.dash(); morse.dash();
  morse.dot(); morse.dot(); morse.dot();
  delay(3000);
}
```

A fentiek alapján láthatjuk, hogyan épülnek fel a könyvtárak és hogyan kell őket használni. Akár saját könyvtár készítéséhez is hozzáfoghatunk, azonban mielőtt ezt megtennénk, érdemes kutatómunkát végezni, hiszen mérhetetlenül sokféle funkciót lefed a már létező könyvtárak összessége.

13.2. SoftwareSerial könyvtár

Az Arduino egyetlen fizikai soros porttal rendelkezik, ami a beépített UART (Universal Asynchronous Receiver Transmitter) terhére mentesíti a feldolgozóegység erőforrásait, amíg a 64 bájtnyi soros átmeneti tárolóban van elég hely. Az integrált port a 0-ás és 1-es digitális csatornákon kommunikál (lásd 9. fejezet), ahol a számítógéphez USB átalakítóval is csatlakozik.

A SoftwareSerial könyvtárat – ahogy a neve is sugallja – azért fejlesztették ki, hogy a feldolgozóegység terhére a többi digitális csatornán is létre lehessen hozni soros kommunikációs felületeket. Segítségével egyszerre kapcsolódhatunk a számítógéphez és más soros eszközökhöz, kijelzőkhöz, digitális szenzorokhoz, vagy kialakíthatunk soros hálózatot több Arduino között. A szoftveres portok maximális sebessége is 115200 baud lehet, ráadásul megváltoztatható a jel polaritása, ami szélesíti a kompatibilis egységek spektrumát, azonban több port létrehozása esetén az adatok egyidejű fogadása nem lehetséges.

Példakód:

```
#include <SoftwareSerial.h> // importáljuk a könyvtárat

SoftwareSerial mySerial(10, 11); // mySerial objektum létrehozása 10-Rx, 11-Tx

void setup()
{
   // mySerial nevű szoftveres soros port inicializálása, sebesség
   // beállításával pl. 4800 baud
   mySerial.begin(4800);
   // nyomtatás, hasonló, mint a fizikai soros port esetén
  mySerial.println("Hello, world!");
}
```

```
void loop() {}
```

A könyvtár függvényei:

- available(),
- begin(),
- isListening(),
- overflow(),
- peek(),
- read(),
- print(),
- println(),
- listen(),
- write().

13.2.1. SoftwareSerial()

Hívásakor létrehoz egy SoftwareSerial objektumot az általunk megadott névvel. Ahhoz, hogy a kommunikációt engedélyezzük, meg kell hívnunk a SoftwareSerial.begin() parancsot is.

Szintaxisa:
SoftwareSerial(Rx,Tx).

Paraméterei:
Rx: a szoftveres soros portunk bejövő csatornája,
Tx: a szoftveres soros portunk kimenő csatornája.

Példakód:
```
SoftwareSerial mySerial =  SoftwareSerial(rxPin, txPin);
```

13.2.2. available()

Visszaadja a szoftveres átmeneti tárolóban elérhető, beérkezett bájtok számát.

Szintaxisa:
mySerial.available().

Példakód:

```
void loop() {
  if (mySerial.available()>0){ // ha már van valami a tárolóban
  mySerial.read(); // olvassuk ki
 }
}
```

13.2.3. begin()

Beállítja a soros kommunikáció sebességét (baud rate). A szabvány átviteli sebességek: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 és 115200.

Szintaxisa:
mySerial.begin(baudrate).

Paramétere:
baudrate: long típusú változó a fenti szabványértékek valamelyike.

Példakód:

```
void setup()  {
  // beállítjuk a vevő és adó csatornákat bemenetnek és kimenetnek
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  // meghatározzuk az átviteli sebességet és megnyitjuk a portot
  mySerial.begin(9600);
}
```

13.2.4. isListening()

Megnézhetjük vele, hogy a kiválasztott szoftveres soros port valóban aktív-e. Boole típusú választ ad vissza.

Szintaxisa:
mySerial.isListening().

Példakód:

```
void loop()
{
  if (portOne.isListening()) { // ha a port aktív, akkor
   Serial.println("A port aktív!");
}
```

13.2.5. overflow()

A szoftveres soros port átmeneti tárolója 64 bájtot képes elraktározni. Amennyiben ezek nem kerülnek időben kiolvasásra, a tároló túlcsordul – a régi adatok helyét nem veszik át az újak és lemaradunk a közölt információ egy részéről. Ezzel a függvénnyel ellenőrizhetjük, hogy történt-e túlcsordulás. Az ellenőrzést követően a függvény válaszát képező Bool változó hamis állapotba kerül.

Szintaxisa:
mySerial.overflow().

Példakód:

```
void loop()
{
  if (portOne.overflow()) { // ha túlcsordult a tároló, akkor
   Serial.println("A tároló túlcsordult!");
}
```

13.2.6. read()

Karakterenként kiolvassa a soros tárolót.

Szintaxisa:
mySerial.read().

Példakód:

```
void loop()
{
  char c = mySerial.read();
}
```

13.2.7. peek()

Hasonlóan a read()-hez, az utolsó a porton beolvasott karaktert adja vissza, ha nem jött semmi –1-et.

Szintaxisa:
mySerial.peek().

13.2.8. print()

Adatokat küld a szoftveres soros port Tx csatornájára. A kiírt bájtok számát adja vissza.

Példakód:

```
SoftwareSerial serial(10,11);

int analogValue;

void setup()
   {
   serial.begin(9600);
   }

void loop()
{
   // analóg bemenet értékét fogjuk különféleképpen elküldeni
   analogValue = analogRead(A0);

   serial.print(analogValue);         // ASCII kódolású decimális formátum
   serial.print("\t");                // tab
   serial.print(analogValue, HEX);    // ASCII kódolású hexadecimális formátum
   serial.print("\t");                // tab
   serial.print(analogValue, OCT);    // ASCII kódolású oktális formátum
   serial.print("\t");                // tab
   serial.print(analogValue, BIN);    // ASCII kódolású bináris formátum
   serial.print("\t");                // tab
   serial.print(analogValue/4, BYTE); // a negyedét (= max. 255) nyers bájtként
   serial.print("\t");                // tab
   serial.println();                  // soremelés

   // várakozás a következő érték felvétele előtt
   delay(10);
}
```

13.2.9. println()

Ahogy az a fenti példából is látszik, a küldendő adatot automatikusan kiegészíti a „soremelés” kódjával – ugyanúgy, mint azt a hardware-es soros portnál már láttuk.

13.2.10. listen()

Több szoftveres soros port használata esetén egyszerre csak egy port kaphatja meg az olvasás jogát. Ezzel a függvénnyel beállíthatjuk, melyik port legyen az, amelyik olvashat. Az összes többi portra érkező adatok elvesznek.

Szintaxisa:
mySerial.listen().

Példakód:

```
#include <SoftwareSerial.h>

// egyik portunk: TX = digitális 10, RX = digitális 11 csatornákon
SoftwareSerial portOne(10, 11);

// másik portunk: TX = digitális 8, RX = digitális 9 csatornákon
SoftwareSerial portTwo(8, 9);

void setup()
{
   // nyissuk meg a hardware-es soros portot
   Serial.begin(9600);

   // nyissuk meg a szoftveres portokat is
   portOne.begin(9600);
   portTwo.begin(9600);

}

void loop()
{
   // az első szoftveres port olvashat
   portOne.listen();

   if (portOne.isListening()) { // ha az első olvashat
      Serial.println("Az első port aktív!");
   }else{ // ha nem olvashat
```

```
        Serial.println("Az első port nem aktív!");
    }

    if (portTwo.isListening()) { // ha a második olvashat
        Serial.println("A második port aktív!");
    }else{ // ha nem olvashat
        Serial.println("A második port nem aktív!");
    }
}
```

13.2.11. write()

Az adatokat nyers bájt formátumban küldi ki a portra és visszaadja a kiírt bájtok számát.

Példakód:

```
//elküldi a "hello" karaktersort és visszaadja a kiküldött karakterek számát
int bytesSent = mySerial.write("hello");
```

13.3. Firmata könyvtár

A Firmata egy kommunikációs protokoll, aminek célja a számítógépes környezetekkel való adatcsere megkönnyítése.

Sok olyan gyors prototipizálásra alkalmas környezet létezik, amelyiken a firmata protokoll használata már megoldott. Ilyenek például a VVVV, Max/MSP, Processing és Flash. Létezik olyan firmata alapú program (firmware), ami az Arduino minden funkcióját elérhetővé teszi soros parancsokon keresztül, az analóg olvasástól a digitális csatornák kapcsolgatásáig, és ezeket saját számítógépen futó fejlesztőkörnyezetünkből el is tudjuk érni. A könyvtár használatával anélkül hozhatunk létre célfeladatokra kétirányú soros kommunikációs programot, hogy saját protokollt (pl. handshaking) vagy objektumokat fejlesztenénk.

Metódusai:

13.3.1. begin()

Elindítja a könyvtárat.

13.3.2. begin(long)

Elindítja a könyvtárat és felülírja az előzőleg beállított adatátviteli sebességet arra, amit megadunk.

13.3.3. printVersion()

Elküldi a protokoll verziószámát a számítógépnek.

13.3.4. blinkVersion()

Elvillogja a protokoll verziószámát a 13-as csatornára kötött LED-en keresztül.

13.3.5. printFirmwareVersion()

Elküldi a firmware nevét és verziószámát a számítógépnek.

13.3.6. setFirmwareVersion(byte major, byte minor)

Beállíthatjuk a firmware nevét és verziószámát. A firmware neve a programunk nevével fog megegyezni a (.pde) kiterjesztés nélkül.

Üzenetek küldése:

13.3.7. sendAnalog(byte pin, int value)

Elküldi az analóg bemenet csatornájának számát és értékét.

13.3.8. sendDigitalPorts(byte pin, byte firstPort, byte secondPort)

Elküldi a két digitális port állapotát különálló bájtokként.

13.3.9. sendDigitalPortPair(byte pin, int value)

Elküldi a két digitális port állapotát egy integer változóként.

13.3.10. sendSysex(byte command, byte bytec, byte* bytev)

Önkényesen választott bájtok tömbjeként küld parancsot.

13.3.11. sendString(const char* string)

Stringet küld a számítógépnek.

13.3.12. sendString(byte command, const char* string)

Parancsként is definiálhatunk stringet, amit elküld.

Üzenetek fogadása:

13.3.13. available()

Leellenőrzi, hogy van-e bejövő adat a tárolóban.

13.3.14. processInput()

Feldolgozza a tárolóban található bejövő üzeneteket és – ha vannak – átvezeti őket a megfelelő visszahívási funkciókhoz.

13.3.15. attach(byte command, callbackFunction myFunction)

Funkciót köt bizonyos típusú bejövő üzenetekhez.

13.3.16. detach(byte command)

Törli az üzenet típusához kapcsolt funkciót.

Visszahívási funkciók:

Ahhoz, hogy egy üzenettípushoz funkciót tudjunk társítani, annak valamelyik szabvány visszahívási funkcióval egyeznie kell. Három különböző visszahívási funkció létezik:

generic

13.3.17. void callbackFunction(byte pin, int value);

Általános visszahívási funkció, pl. csatornák állapotának lekérdezésére.

system_reset

13.3.18. void systemResetCallbackFunction(void);

Visszaállítja a firmware-t az alapállapotba.

string

*13.3.19. void stringCallbackFunction(char *myString);*

Karakterláncok visszaadásához.

sysex

*13.3.20. void sysexCallbackFunction(byte pin, byte byteCount, byte *arrayPointer);*

pl. MIDI vezérlőjelek visszaadásához.

Üzenettípusok:

11.3.21. ANALOG_MESSAGE

Egy analóg bemenethez tartozó aktuális érték.

13.3.22. DIGITAL_MESSAGE

Egy digitális port 8 bitjének értéke.

13.3.23. REPORT_ANALOG

Ki-/bekapcsolja az analóg üzeneteket.

13.3.24. REPORT_DIGITAL

Ki-/bekapcsolja a digitális üzeneteket.

 © Harsányi Réka, Juhász Márton András

13.3.25. SET_PIN_MODE

Beállít egy csatornát: INPUT/OUTPUT/PWM.

13.3.26. FIRMATA_STRING

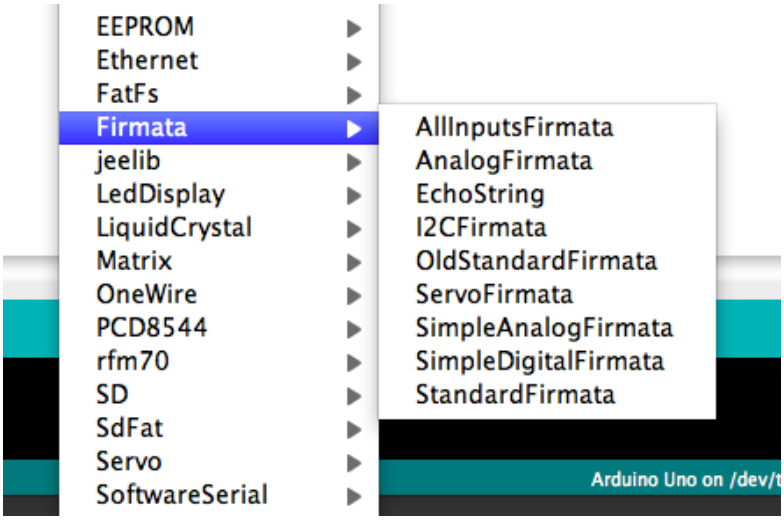
stringCallbackFunction-nel társítható string alapú üzenettípus.

13.3.27. SYSEX_START

Általános, változó hosszúságú üzenetek (pl. MIDI SysEx protokoll) sysexCallbackFunction-nel társítható üzenettípus.

13.3.28. SYSTEM_RESET

Visszaállítja a firmware-t az alapállapotába, *systemResetCallbackFunction*-nel társítható üzenettípus.



13.1. ábra: Előre elkészített firmata alapú firmware-ek

Miután firmatára támaszkodó, előre elkészített firmware-ből többféle is létezik a programcsomagban, nézzük inkább ezek gyakorlati hasznát!

AllInputsFirmata

Az összes analóg és digitális bemenet aktuális állapotát a lehető leggyorsabban továbbítja a számítógépnek. Létrehozását az Arduino2Max program inspirálta.

AnalogFirmata

A lehető legtöbb analóg portot hasznosítja, beleértve az összes analóg bemenetet, 3, 5, 6, 9, 10, 11 PWM kimeneteket, melyek közül a 9 és 10 szervo meghajtóként is használható.

ServoFirmata

Az összes digitális csatornát szervo meghajtóként érhetjük el.

I2CFirmata

I2C protokollt használó eszközökkel kommunikálhatunk az Arduinon, mint átalakítón keresztül.

StandardFirmata

Minden felsorolt funkciót támogat, elérhetjük a digitális és analóg ki-/bemeneteket, I2C kommunikációt, használhatunk digitális csatornákat szervo meghajtónak.

13.3.1. StandardFirmata használata Processingből

```
import processing.serial.*; // soros port eléréséhez
import cc.arduino.*; // firmata eléréséhez

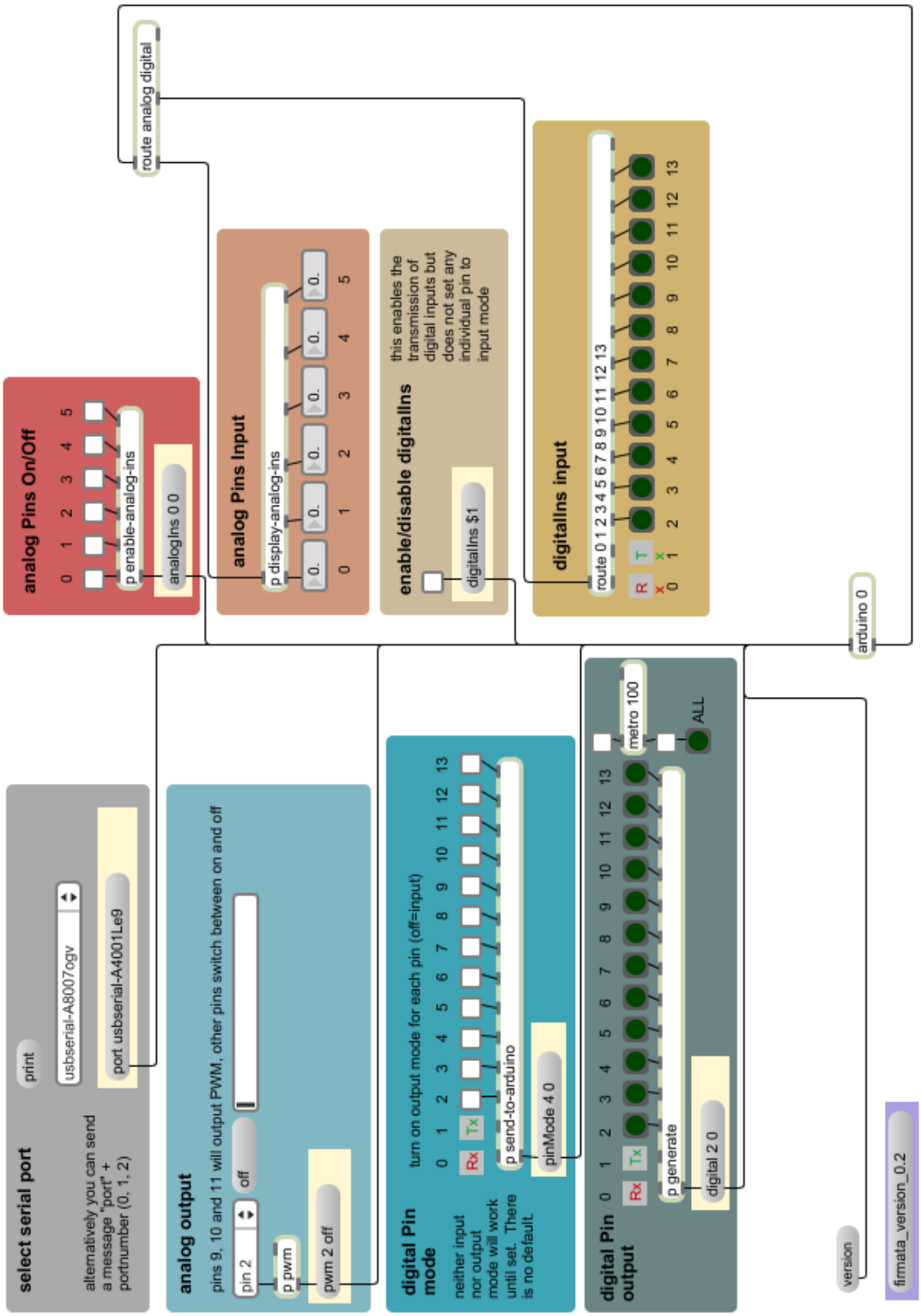
Arduino arduino; // Arduino, mint objektum

int ledPin = 13;

void setup()
{
  println(Arduino.list()); // Arduinonak tűnő eszközök listája
  arduino = new Arduino(this, Arduino.list()[0], 57600);
  // megnyitjuk a portot
  arduino.pinMode(ledPin, Arduino.OUTPUT); // az Arduino környezetből ismert
  // parancsokat használhatjuk a 13-as csatorna kimenetként való kezelésére
  // bemenetként
  background(0);
}

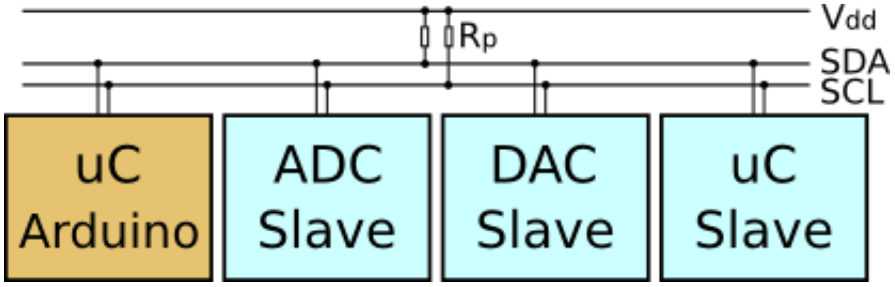
void draw()
{
  arduino.digitalWrite(ledPin, Arduino.HIGH);
  //kimenet állapotának változtatása
  delay(1000);
  arduino.digitalWrite(ledPin, Arduino.LOW);
  delay(1000);
}
```

13.3.2. StandardFirmata használata Max/MSP-vel



13. 4. Wire könyvtár

Az I2C, vagy más néven TWI (Two Wire Interface) egy soros buszrendszer, amit a Philips fejlesztett ki, hogy mikrokontrollerekhez alacsony sebességű perifériákat két vezeték használatával lehessen illeszteni. Ilyen eszközök például a Maxim által gyártott csatornabővítők (MAX6956 I2C-t 20 párhuzamos I/O csatornává alakít), LED meghajtók, adatátalakítók (pl. digitalizálók), Microchip által gyártott soros EEPROM-ok (legnagyobb 1Mbyte-os), vagy az NXP termékei: órák, memóriák, audio és video jelfeldolgozók, szintátalakítók.



13.2. ábra: A buszon keresztül sok eszközt csatlakoztathatunk

A *buszokra* általánosságban az a jellemző, hogy az eszközök megosztoznak az adatvezetékeken, ezért minden eszköznek rendelkeznie kell egy *címmel*, ami megkülönbözteti a többi buszhoz kapcsolt egységtől. Az I2C esetén a címzés eszköztől függően kétféle lehet, 7 vagy 8 bites. Az utóbbi esetben 7 bit a tényleges címet kódolja, míg a 8. bit arról ad információt, hogy a címen írás vagy olvasás folyik. A Wire könyvtár mindkét típust támogatja. Az adatvezetékeket rendre **SDA** (Serial Data) és **SCL** (Serial Clock) neveket viselik, míg az egyiken a szinkronitást fenntartó órajel, a másikon a szekvenciális adat jelenik meg bitenként. Az Arduino mikrokontrollere fizikailag tartalmaz csatolófelületet a buszhoz, aminek kivezetései UNO – és megelőző típusok – esetében az (analóg bemenetként is használható) **A4** (SDA) és **A5** (SCL).

Ezen a busztípuson kétféle eszköztípus, egy mester (**Master**) és a címek elfogyásáig tetszőleges mennyiségű szolga (**slave**) eszköz kommunikálhat. A mester eszköz az, aki adott címen végrehajthat írást vagy olvasást, a szolgák pedig, akik a mester parancsait követik. A könyvtár alapbeállítása szerint a mester az Arduino, de ha könyvtár inicializálásakor címet adunk neki, akkor szolgaként is tudjuk használni – ez akkor lehet hasznunkra, ha olyan hálózatot szeretnénk építeni, amihez I2C eszközök is és több Arduino is csatlakozik (lásd 13.2. ábra). Ha képesek vagyunk a buszon a konfliktusokat kezelni, több mesteregység használata is elképzelhető.

A könyvtár függvényei:

13.4.1. begin()

Inicializálja az I2C kommunikációt. Ha nem adunk meg neki külön címet, automatikusan az Arduino lesz a mester, ellenkező esetben a megadott címen szolgaként fog működni.

Szintaxisa:
Wire.begin(address).

Paramétere:
address: 7 bites cím.

13.4.2. requestFrom()

A mester által használt parancs, hogy bájtokat kérjen valamelyik szolgaeszköztől. Az eszközök válaszait az available() vagy read() függvényekkel kezelhetjük. Az I2C eszközök közötti működésbeli különbségek áthidalására ez a függvény elfogad bemenetként bool típusú változót. Ami ha **igaz** (alapállapot), akkor a lekérés után STOP jelet küld, és nem kommunikál a buszon, ha **hamis**, akkor RESTART jelet küld, és nem szabadítja fel a buszt.

Szintaxisa:
Wire.requestFrom(address, quantity),
Wire.requestFrom(address, quantity, stop).

Paraméterei:
address: 7 bites cím,
quantity: lekérdezendő bájtok száma,
stop: bool változó.

13.4.3. available()

Visszaadja az olvasásra kész bájtok számát. Mestereszközön requestFrom() után kell meghívni, szolgaeszközön pedig az onReceive() kezelőben.

Szintaxisa:
Wire.available().

13.4.4. read()

Beolvassa a soron következő bájtot.

Szintaxisa:
Wire.read().

Példakód:

```
#include <Wire.h> // importáljuk a Wire könyvtárat

void setup()
{
  Wire.begin();        // csatlakozás a buszhoz
  Serial.begin(9600);  // soros port inicializálása
}

void loop()
{
  Wire.requestFrom(2, 6);    // lekérünk 6 bájtot a 2-es eszközről

  while(Wire.available())    // szolga küldhet kevesebbet, mint kértünk
  {
    char c = Wire.read();    // olvassuk be karakterként a bájtot
    Serial.print(c);         // írjuk ki a soros portra, mit kaptunk
  }

  delay(500);
}
```

13.4.5. write()

Küld egy bájtot a szolgaeszközről a mestertől kapott kérésre válaszul – vagy a beginTransmission() és endTransmission között használva a mestereszközről küld egy bájtot valamelyik szolgának.

Szintaxisa:
Wire.write(value),
Wire.write(string),
Wire.write(data, length).

 © Harsányi Réka, Juhász Márton András

Paraméterei:
value: bájtként küldendő érték,
string: bájtok sorozata,
data: length hosszúságú bájtok tömbje.

Példakód:

```
#include <Wire.h>

byte val = 0;

void setup()
{
  Wire.begin(); // csatlakozás a buszhoz mesterként
}

void loop()
{
  Wire.beginTransmission(44); // 44-es eszközhöz szól
                              // a cím általában az eszköz adatlapján van
  Wire.write(val);            // egy bájtot küld
  Wire.endTransmission();     // felszabadítja a buszt

  val++;        // növeljük eggyel a bájt értékét
  if(val == 64) // ha elértük a maximumot
  {
    val = 0;    // kezdjük újra a minimumtól
  }
  delay(500);
}
```

13.4.6. beginTransmission()

Adatküldés inicializálása szolgaeszköz felé, a bájt értékének megadása a write() függvénnyel, endTransmission() küldi végül el.

13.4.7. endTransmission()

Elküldi a beginTransmission()-nel hívott címre a write()-ban megadott értékeket. Egy bájtot ad vissza, ami ötféle érték lehet:

0: sikeres átvitel,
1: az adat nem fér bele az átviteli tárolóba,
2: a címről nem érkezett válasz,
3: az adatot nem fogadták,
4: egyéb hiba.

Szintaxisa:
Wire.endTransmission(),
Wire.endTransmission(stop) (stop – lásd feljebb).

13.4.8. onReceive()

Szolgaeszközön regisztrál egy függvényt, amit végre kell hajtania, ha a mestertől *bármi* érkezik.

Szintaxisa:
onReceive(handler).

Paramétere:
handler: a függvény, amit végre kell hajtania.

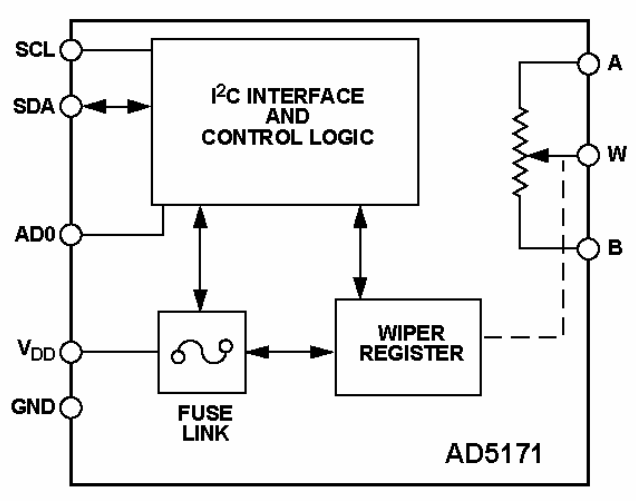
13.4.9. onRequest()

Szolgaeszközön regisztrál egy függvényt, amit végre kell hajtania, ha a mestereszköz *adatot* szeretne *lekérdezni*.

Szintaxisa:
onReceive(handler).

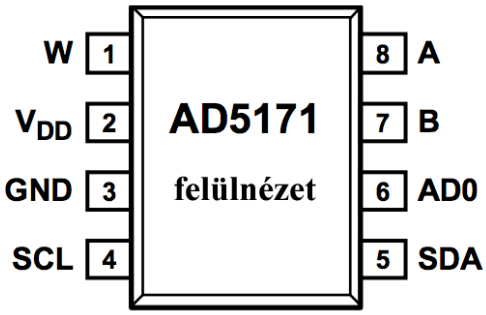
Paramétere:
handler: a függvény, amit végre kell hajtania.

A gyakorlatban például nagyon hasznos I2C eszköznek bizonyul a *digitális potméter*, amivel kiterjeszthetjük az Arduino analóg kimeneteinek számát, vagy programozhatóvá alakíthatunk olyan analóg áramköröket, mint egy erősítő vagy rádió.



13.3. ábra: Digitális potméter funkcionális rajza

A digitális potméter egy integrált áramkör (IC – Integrated Circuit), amiben az I2C protokollt értelmező vezérlő, elektronikus kapcsolók és számtalan különféle értékű ellenállás kapott helyet (13.3. ábra AD5171 típusú áramkör).



13.4. ábra: Az AD5171 lábkiosztása

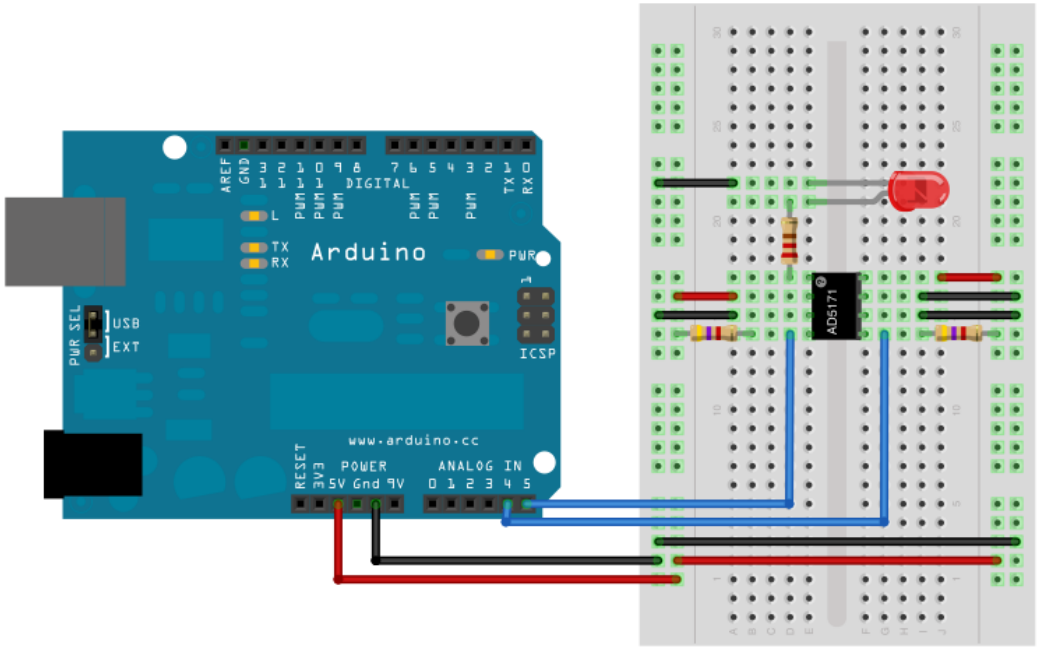
A 13.3. és 13.4. ábrák összevetésével el tudjuk képzelni, hogy az áramkör fizikai lábaihoz mely funkciók társulnak – ezeket a rajzokat a gyártó biztosítja. **GND** (ground v. föld) a potenciál alapja, **Vdd** a tápfeszültség, **SCL** (Serial Clock) az órajel, **SDA** (Serial Data) az adatvonal, **AD0** (Address), amivel beállíthatjuk az eszköz címét a buszon, **A** és **B** a potméter két „szélső” kivezetése, míg **W** a közepe. Változtassuk egy LED fényerejét I2C buszon kommunikáló digitális potméter segítségével!

Amire szükségünk lesz:
- Arduino,
- AD5171 (DIP tokozású),
- LED,

- 220 ohmos ellenállás,
- 2 db 4,7 kiloohmos ellenállás,
- próbapanel,
- vezetékek.

Kapcsolás:

Kössük az IC tápvezetékeit értelemszerűen az Arduino földjéhez és + 5 V jelölésű kivezetéséhez! Az A4 és A5 csatornákat 4,7 kiloohmos ellenálláson keresztül szintén a + 5 V-hoz. Ezek az úgynevezett *felhúzó ellenállások,* amik stabilizálják a busz vonalait. Ezután illesszük A4-et az IC SDA (5.) lábához, A5-öt az SCL (4.) lábához. A potmétert használjuk feszültségosztónak és a közepéről hajtsuk meg a LED-et a 220 ohmos soros ellenálláson keresztül. AD0-t, ha földre kötjük, a címet 44-re, ha + 5 V-ra kötjük, 45-re állítja.



13.5. ábra: AD5171 bekötése az Arduinoba

 © Harsányi Réka, Juhász Márton András

Példakód:

```
#include <Wire.h> // importáljuk a wire könyvtárat

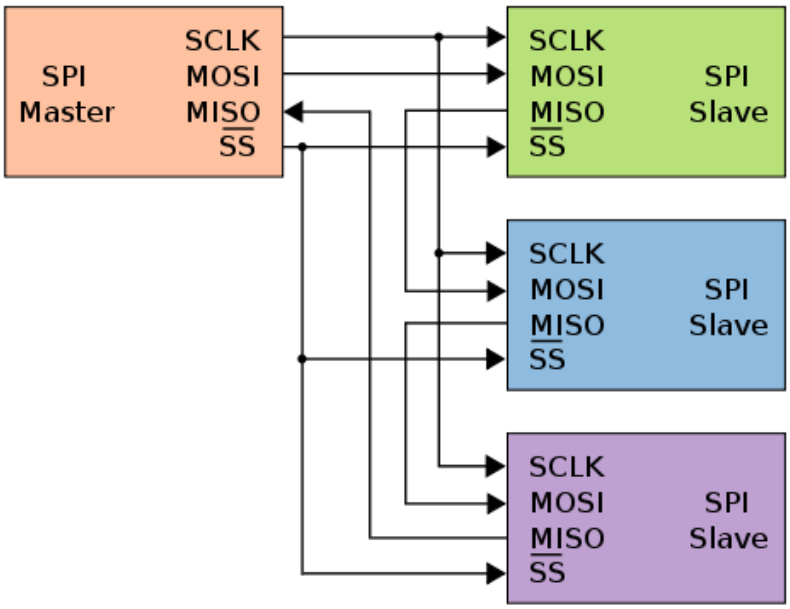
void setup()
{
  Wire.begin(); // mesterként csatlakozunk a buszhoz
}

byte val = 0; // ezt az értéket fogjuk elküldeni a potméternek

void loop()
{
  Wire.beginTransmission(44); // 44-es csatorna, ha AD0 földre van kötve
  Wire.write(byte(0x00));  // instrukciós bájt (adatlap alapján)
  Wire.write(val);        // a potmétert erre az értékre állítjuk
  Wire.endTransmission();     // itt küldjük el az üzenetet
  val++;        // növeljük eggyel az elküldendő értéket
  if(val == 64) // az AD5171 64 különböző állapotot tud
  {
    val = 0;    // ha a végére értünk, kezdjük elölről
  }
  delay(500);
}
```

13. 5. SPI könyvtár

Az SPI (Serial Peripherial Interface) egy szinkron soros protokoll. Alapvető különbség az I2C-hez képest, hogy külön adatvonalat tart fenn a mestertől a szolga felé és az ellenkező irányú kommunikáció céljára, továbbá nem használ címzést, az egyetlen mester az eszközöket egyenként külön vonalakon engedélyezi. Mindezek együttesen egy gyorsabb soros adatátvitelt tesznek lehetővé a mikrokontroller és perifériái között. A közös adatvonalak a MISO (Master In Slave Out), amin keresztül a mester fogad adatokat, a MOSI (Master Out Slave In), amin a szolgák fogadnak adatokat és az SCK (Serial Clock) órajel a szinkronizáláshoz. Az eszközönként specifikus vonal az engedélyezéshez az SS (Slave Select). Amikor egy szolga SS vonala logikai alacsony szintre kerül, létrejön a kapcsolata a mesterrel, ha magas szinten tartjuk, akkor teljesen kimarad a buszon való kommunikációból, így nem zavarhatja a többi perifériát.



13.6. ábra: SPI busz funkcionális rajza több szolgaeszköz esetén

A nyílt fejlesztésű rendszernek és a közösségben együttműködő programozóknak köszönhetően az Arduinoval számtalan SPI alkatrész került implementálásra, és a könyvtár ebből a rutinból fakadóan rendelkezik olyan függvényekkel, amik megkönnyítik az új eszközök használatát. Mindazonáltal az SPI protokoll meglehetősen szabadon értelmezett az eszközgyártók körében, ezért, ha egy új alkatrészhez szeretnénk programot írni, akkor nézzük meg alaposan az adatlapját.

Eltérések a következő paraméterekben lehetnek:

- **MSB** (Most Significant Bit): legnagyobb helyi értékű bitet küldi a szolga először.
- **LSB**: (Less Significant Bit): legkisebb helyi értékű bitet küldi a szolga először.

A két változat között az *SPI.setBitOrder()* függvény biztosítja az átjárást.

- **Órajel frekvenciája**: Az órajel sebességét az *SPI.setClockDivider()*-rel adhatjuk meg.
- **Órajel fázisa (CPHA)**: Az adatok az órajel negatív vagy pozitív élével vannak szinkronban.
- **Órajel polaritása (CPOL)**: A várakozás logikailag magas vagy alacsony szint.

Erről az *SPI.setDataMode()* függvény segítségével dönthetünk:

Mód	CPOL	CPHA
SPI_MODE0	0	0
SPI_MODE1	0	1
SPI_MODE2	1	0
SPI_MODE3	1	1

13.1. táblázat: setDataMode függvény állapotai

Az SPI kivezetéseit megtalálhatjuk az Arduino alaplapján egy külön csatlakozón is, amit ICSP-nek (In Circuit Serial Programming) hívunk.



13.7. ábra: ICSP csatlakozó kiosztása

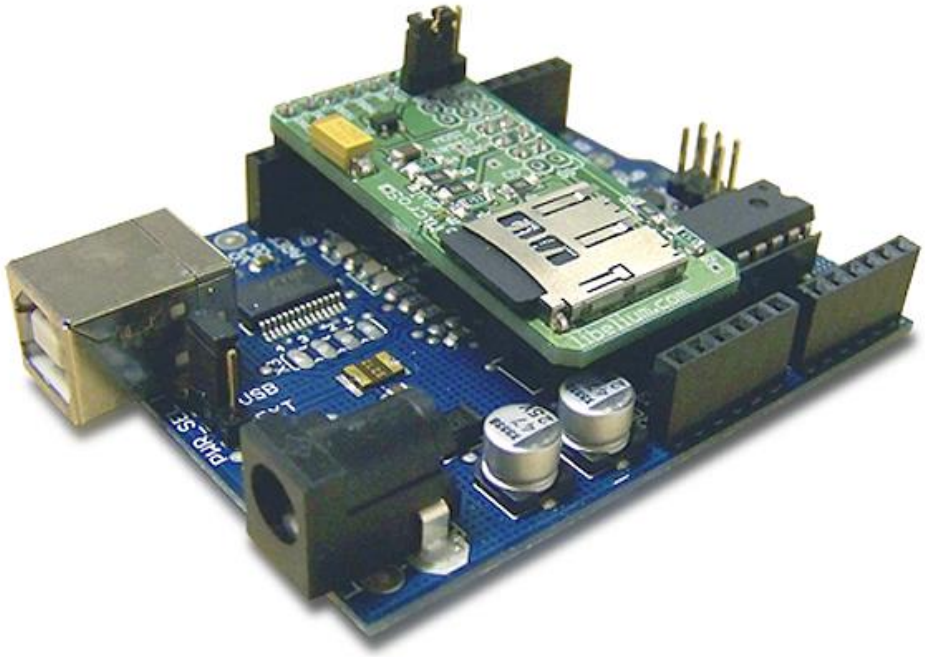
Mivel ezen a csatlakozón keresztül tölthetjük fel az Arduino alapprogramját, a bootloadert is, ezért megtalálható minden típusú alaplapon ugyanebben a formában. Mint azt láttuk, a kivezetések sokszor több különböző funkciót is el tudnak látni, így nem meglepő, hogy az ICSP vonalait az eddig tárgyalt csatlakozók valamelyikén is megtaláljuk:

Típus	MOSI	MISO	SCK	SS (szolga)	SS (mester)
UNO	13	11	12	10	-
Mega	51	50	52	53	-
Leonardo	ICSP4	ICSP1	ICSP3	-	-
DUE	ICSP4	ICSP1	ICSP3	-	4, 10, 52

13.2. táblázat: SPI csatornák kivezetései a különböző típusú Arduino alaplapokon

Az eltérések nem kizárólag az alapvető kommunikációban mutatkoznak meg, minden alkatrész rendelkezik saját utasításkészlettel és regiszterekkel.

Az SD kártyák is SPI buszon kommunikálnak, amiket viszonylag könnyen és nagyon sokféleképpen hasznosíthatunk arduinos projektjeinkben (háttértár, hangfelvétel, lejátszás, adatrögzítés, GPS). Példánkban a Libelium által gyártott microSD panelt fogjuk alkalmazni, aminek kialakításakor figyelembe vették mind a kétféle csatlakoztatási módot.



13.8. ábra: Arduino Libelium microSD modullal

Az SD kártyák csatlakoztatása a modul megvásárlása nélkül is igen egyszerű, mindössze pár ellenállásra van szükség, amikkel az Arduino 5 V-os rendszerű SPI buszának vonalait a memóriakártyák által elfogadható 3,3 V-os szintűvé alakítjuk. Tulajdonképpen a bemutatásra kerülő modul sem tesz ennél többet. A kártya eléréséhez az SPI könyvtár egy módosított változatát fogjuk használni, ami képes FAT partíción fájlstruktúrák kezelésére.

Amire szükségünk lesz:

- Arduino,
- Libelium microSD modul, vagy
- 5 V – 3,3 V szintillesztő.

Példakód:

```
   /*
 * SD kártya lábait a következő Arduino csatlakozókhoz kell kötni:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 10
    */

#include < SD.h > // ezt az alap programcsomag tartalmazza

// ezen a csatornán keresztül fog áramot kapni a kártya
#define MEM_PW 8

File myFile;

void setup()
{

  Serial.begin(9600);
  Serial.print("SD kartya inicializálás...");

  // kapcsoljuk be a kártyát
  pinMode(MEM_PW, OUTPUT);
  digitalWrite(MEM_PW, HIGH);

  if (!SD.begin(10)) {
    Serial.println("sikertelen inicializálás!");
    return;
  }
  Serial.println("sikeres inicializálás.");

  // nézzük meg, létezik-e az example.txt fájl:
  if (SD.exists("example.txt")) {
    Serial.println("example.txt létezik.");
    } else {
    Serial.println("example.txt nem létezik.");
  }

  // megnyitás vagy létrehozás, egyszerre csak egy fájlt tudunk kezelni
  myFile = SD.open("example.txt", FILE_WRITE);

  // ha megnyílt a fájl, írjunk bele:
  if (myFile) {
```

```
    Serial.print("Írás az example.txt-be…");
    myFile.println("próba 1, 2, 3.");
    // fájl bezárása / mentése:
    myFile.close();
    Serial.println("kész.");
  } else {
    // ha nem nyílt meg:
    Serial.println("nem tudom megnyitni az example.txt-t!");
  }

  // nyissuk meg újra, most olvasásra
  myFile = SD.open("example.txt");
  if (myFile) {
    Serial.println("example.txt:");

    // olvassunk belőle, amíg tart a fájl:
    while (myFile.available()) {
        Serial.write(myFile.read());
    }
    // fájl lezárása:
    myFile.close();

   } else {
      // ha nem nyílik meg:
    Serial.println("nem tudom megnyitni az example.txt-t!");
  }

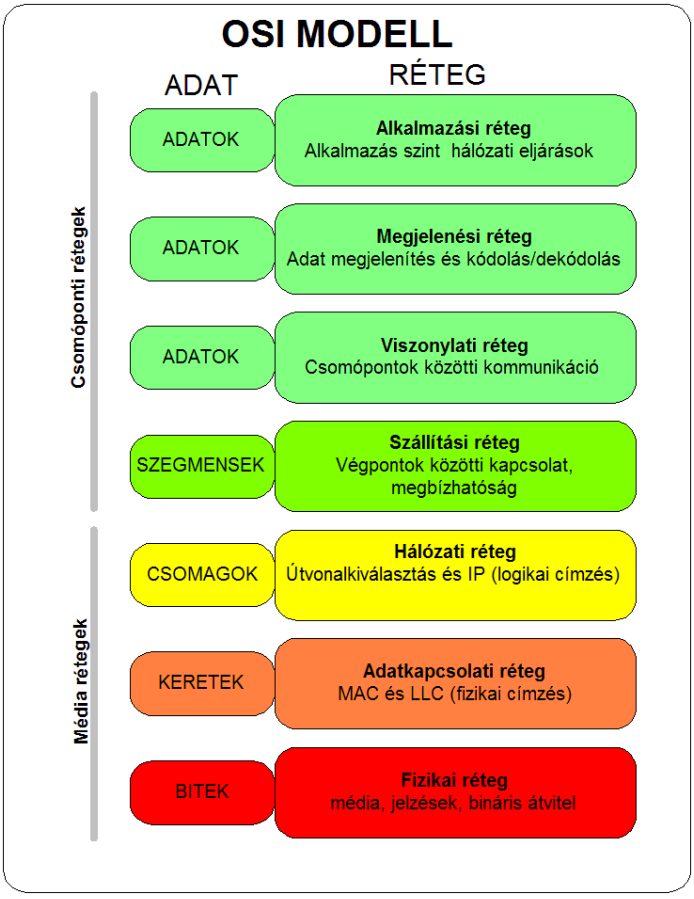
  // töröljük le az example.txt-t:
  Serial.println("example.txt törlése…");
  SD.remove("example.txt");

  if (SD.exists("example.txt")){
    Serial.println("törlés sikertelen.");
  } else {
    Serial.println("törlés sikeres.");
  }
}

void loop()
{
    // a fentit csak egyszer futtatjuk
   }
```

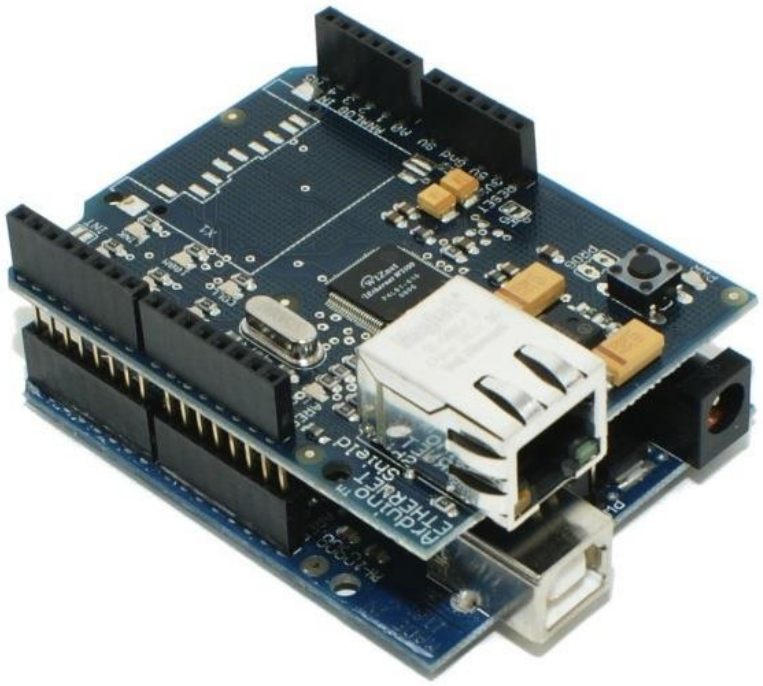
13. 6. Ethernet könyvtár

Az OSI (Open Systems Interconnection) modell szerint – mely hét absztrakt rétegre osztja a kommunikációs technológiák funkcionális szintjeit – az Ethernet 1985 óta a számítógépeket összekötő hálózatok fő kommunikációs technológiai szabványa és fizikai rétegtől az adatkapcsolati rétegig nyújt szolgáltatásokat. A fizikai réteg magát a hardware-t jelenti, az adatok bitenkénti csatornákra juttatását. Az adatátviteli réteg pedig két hálózati elem közötti átvitelt biztosító eljárásokért felelős.

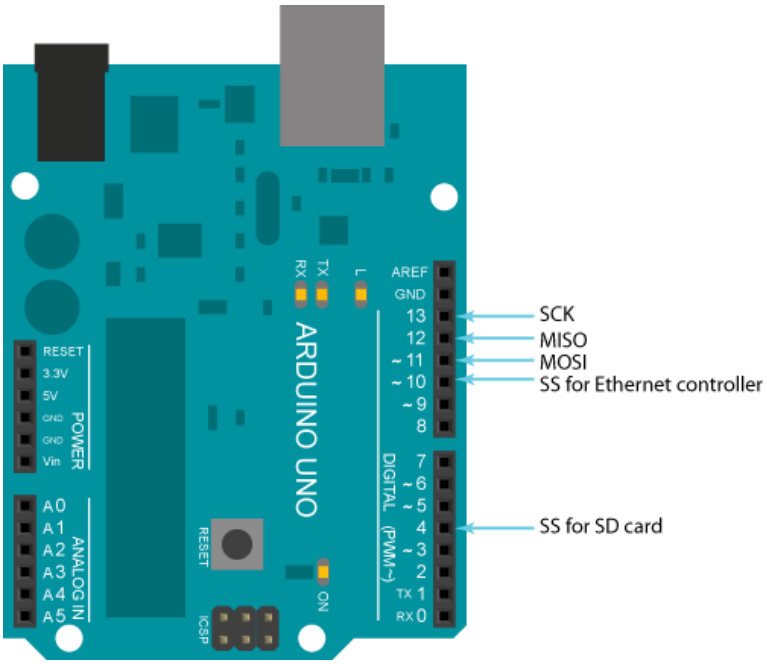


13.9. ábra: Open Systems Interconnection modell

Az Ethernet kiegészítő modullal (shield) és a hozzá tartozó könyvtár segítségével eljuthatunk a hálózati rétegig és internetképessé tehetjük az Arduinot. A shield a Wiznet, SPI kommunikáción alapuló W5100-as integrált áramkörére épül és fenntartja az SD kártya csatlakoztatásának lehetőségét is. A kiegészítőt elég csatlakoztatni. A következő példákban megnézzük, hogyan lehet bejövő kapcsolatokat kezelő szerverként és kimenőket intéző kliensként egyaránt működtetni.



13.10. ábra: Arduino Ethernet Shielddel



13.11. ábra: Csatornák kiosztása az Eethernet Shieldhez

 © Harsányi Réka, Juhász Márton András

13.6.1. Ethernet osztály

13.6.1.1. Ethernet.begin()

Az ethernet könyvtárat és a hálózati beállításokat inicializálja a parancs. A begin() függvénnyel megadhatjuk az eszközünk MAC (Media Access Control – fizikai) és az IP (Internet Protocol) címét. A könyvtár támogatja a DHCP (Dynamic Host Configuration Protocol) alapú IP-cím igénylést, ezért ha csak a fizikai címet adjuk meg, ebbe az üzemmódba kerül. Az automatikus címigényléshez a könyvtár nagyobb hányada töltődik az Arduinoba, ezért ez a mód sokkal több memóriát igényel, mintha csak statikus címet adnánk meg. Ethernet.begin(mac) sikeres cím igénylés esetén 1-et, sikertelen esetén 0-át ad vissza.

Szintaxisa:
Ethernet.begin(mac),
Ethernet.begin(mac, ip),
Ethernet.begin(mac, ip, dns),
Ethernet.begin(mac, ip, dns, gateway),
Ethernet.begin(mac, ip, dns, gateway, subnet).

Paraméterei:
mac: 6 bájtos tömb, az eszköz fizikai címe,
ip: 4 bájtos tömb, az eszköz IP-címe,
dns: 4 bájtos tömb, DNS (Domain Name Server) IP-címe,
gateway: 4 bájtos tömb, hálózati átjáró címe,
subnet: 4 bájtos tömb, a hálózat alhálózati maszkja, alapállapotban 255.255.255.0.

13.6.1.2. Ethernet.localIP()

Visszaadja az eszköz aktuális IP-címét. Ez akkor hasznos, ha automatikusan igényeltük.

13.6.1.3. Ethernet.maintain()

DHCP használata esetén egy meghatározott időre kap az eszköz IP-címet. Amikor ez lejár, meg kell újítani. Ezzel a függvénnyel a lejárt cím helyett újat igényelhetünk, ami a DHCP szervertől függően lehet ugyanaz, egy teljesen más, vagy semmi is. Egy számot ad vissza, ami ha:

0: nem történt semmi,
1: cím megújítása sikertelen,
2: cím megújítása sikeres,
3: újrakapcsolódás sikertelen,
4: újrakapcsolódás sikeres.

13.6.2. IPAddress osztály

13.6.2.1. IPAddress()

Meghatározza az IP-címet, ami lehet helyi és távoli cím is.

Szintaxisa:
IPAddress(address)

Paramétere:
address: vesszőkkel elválasztott 4 bájt (pl.: 192,168,1,1).

Példakód:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte gateway[] = { 10, 0, 0, 1 };
byte subnet[] = { 255, 255, 0, 0 };

   IPAddress ip(192,168,1,1);

void setup()
{  Ethernet.begin(mac, ip, dns, gateway, subnet); }
```

Példakód:

```
#include <SPI.h>
#include <Ethernet.h>

// a fizikai cím. Újabb ethernet shieldekre rá van írva, a régebbieknél
// szabadon választott
byte mac[] = {
  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };

void setup() {
    Serial.begin(9600);
    // ethernet inicializálása DHCP-vel
    if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // ha nem tudtunk kapcsolódni, akkor nem csinálunk semmit
    for(;;)
      ;
  }
  // írjuk ki soros terminálra az eszköz IP-címét
  Serial.println(Ethernet.localIP());
```

 © Harsányi Réka, Juhász Márton András

```
}

void loop() {}
```

13.6.3. Server osztály

13.6.3.1. EthernetServer()

Létrehoz egy szervert, ami a bejövő kapcsolatokat figyeli egy előre meghatározott porton.

Szintaxisa:
Server(port).

Paramétere:
port: a port száma, amit a szerver figyel.

Példakód:
```
EthernetServer server = EthernetServer(23);
```

13.6.3.2. server.begin()

Szól a szervernek, hogy kezdjen figyelni.

server.available()
Visszaadja egy csatlakozott kliens címét, ha rendelkezésre állnak olvasható adatok.

Példakód:
```
EthernetClient client = server.available();
```

13.6.3.3. server.write()

Adatokat küld a szerverhez kapcsolódott klienseknek. Az adat bájt vagy karakter típusú lehet.

13.6.3.4. server.print()

ASCII kódolású adatokat küld a szerverhez kapcsolódott klienseknek. A *server.println()* az adatot kiegészíti a soremelés kódjával.

Szintaxisa:
server.print(data),
server.println(data, BASE).

Paraméterei:
data: char, byte, int, long, string, amit a szervernek ki kell küldenie,
BASE: a számrendszer alapja BIN, HEX, OCT, DEC.

Példakód:

```
#include <SPI.h>
#include <Ethernet.h>

// hálózati konfiguráció, átjáró és alhálózat opcionális

 // fizikai cím
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
// IP-cím
byte ip[] = { 10, 0, 0, 177 };
// általában a router címe
byte gateway[] = { 10, 0, 0, 1 };
// alhálózat, ha nem az alapállapot
byte subnet[] = { 255, 255, 0, 0 };

// telnet portja a 23
EthernetServer server = EthernetServer(23);

void setup()
{
  // ethernet inicializálása
  Ethernet.begin(mac, ip, gateway, subnet);

  // szóljunk a szervernek, hogy figyeljen
  server.begin();
}

void loop()
{
  // ha egy kliens kapcsolódik, akkor lesznek elérhető bájtok:
  EthernetClient client = server.available();
  if (client == true) {
    // olvassuk ki ezeket a bájtokat és küldjük őket vissza
    // minden kliensnek, aki kapcsolódott
    server.write(client.read());
  }
}
```

13.6.4. Client osztály

13.6.4.1. EthernetClient()

Létrehoz egy klienst, ami egy IP-címen meghatározott porthoz képes csatlakozni.

13.6.4.2. If(ethernetClient)

Megvizsgálja, hogy a kliens készen áll-e. Bool típusú változóban adja vissza a választ.

13.6.4.3. client.connected()

Megvizsgálja, hogy a kliens kapcsolatban van-e. Amíg van olvasatlan adat, addig igen.

13.6.4.4. client.connect()

A meghatározott címre és portra kapcsolja a klienst, ha sikerült, IGAZ választ ad vissza.

13.6.4.5. client.write()

Adatot küld a szervernek, amelyikhez a kliens kapcsolódott.

13.6.4.6. client.print() és client.println()

ASCII kódolású adatokat küld a szervernek.

13.6.4.7. client.available()

Visszaadja a szerver által küldött, olvasásra kész bájtok számát.

13.6.4.8. client.read()

Beolvassa a szerver által küldött utolsó bájtot.

13.6.4.9. client.flush()

Kiüríti az átmeneti tárolót, minden olvasatlan bájtot töröl.

13.6.4.10. client.stop()

Megszakítja a kapcsolatot a szerverrel.

Példakód:

```
#include <Ethernet.h>
#include <SPI.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 10, 0, 0, 177 }; // az eszközünk IP-címe
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client; // létrehozzuk a klienst

void setup()
{
  Ethernet.begin(mac, ip);
  Serial.begin(9600);

  delay(1000);

  Serial.println("connecting...");
    // 80-as port a HTTP
   if (client.connect(server, 80)) {
     Serial.println("connected");
     // ha sikerül csatlakozni, keresés Arduinora a google-n
     client.println("GET /search?q=arduino HTTP/1.0");
     client.println();
  } else {
    Serial.println("connection failed");
   }
}

   void loop()
{
   // ha van elérhető adat a szervertől, nyomtassuk ki a soros terminálra
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

   // ha nincs több adat, szakítsuk meg a kapcsolatot
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for(;;);
```

 © Harsányi Réka, Juhász Márton András

```
  }
}
```

13.6.5. EthernetUDP osztály

Az UDP (User Datagram Protocoll) az internetet alkotó protokollok egyike, amivel számítógép-alkalmazások cserélnek úgynevezett datagrammokat egymás között. Leginkább olyan esetekben érdemes használni, amikor nem fontos a hibaellenőrzés és -javítás a csomagok átvitelekor. Ez általában valós idejű adatátviteli rendszereknél fontos, ahol egy csomag elvesztése kisebb probléma, mint az ellenőrzésekből fakadó késés.

13.6.5.1. UDP.begin()

Inicializálja az UDP könyvtárat és hálózati beállításait.

Szintaxisa:

EthernetUDP.begin(localPort);

Paraméterei:
localPort: helyi port száma

13.6.5.2. UDP.read()

Datagrammot olvas egy meghatározott tárolóból. Argumentumok nélkül a soron következő karaktert adja vissza.

Szintaxisa:
UDP.read(packetBuffer, MaxSize).

Paraméterei:
packetBuffer: az átmeneti tároló, ami a bejövő karaktereket tárolja,
MaxSize: az átmeneti tároló maximális mérete.

13.6.5.3. UDP.write()

Datagrammot ír egy távoli kapcsolatnak. beginPacket() és endPacket() között kell használni. Az elküldött karakterek számát adja vissza.

Szintaxisa:

UDP.write(message);

Paraméterei:
message: kimenő üzenet (char adattípus)

13.6.5.4. UDP.beginPacket()

Inicializálja a datagramm elküldését. A küldendő adatokat a write()-tal határozzuk meg.

Szintaxisa:
UDP.beginPacket(remoteIP, remotePort);

Paraméterei:
remoteIP: a távoli kapcsolat IP címe (4 byte)
remotePort: a távoli kapcsolat port száma (int)

13.6.5.5. UDP.endPacket()

Jóváhagyja a datagramm továbbítását.

Szintaxisa:
UDP.endPacket();

Paraméterei:
nincs

13.6.5.6. UDP.parsePacket()

Megvizsgálja, van-e elérhető datagramm, és ha van, visszaadja a méretét. Read() előtt kell használni.

Szintaxisa:
UDP.parsePacket();

Paraméterei:
nincs

13.6.5.7. UDP.available()

Az átmeneti tárolóba már megérkezett bájtok (karakterek) számát adja vissza. A parsePacket() után érdemes használni.

Szintaxisa:
UDP.available();

Paraméterei:
nincs

13.6.5.8. UDP.remoteIP()

Visszaadja a távoli kapcsolat IP-címét. A parsePacket() után kell használni.

Szintaxisa:
UDP.remoteIP();

Paraméterei:
nincs

13.6.5.9. UDP.remotePort()

Visszaadja a távoli kapcsolat portjának számát. A parsePacket() után kell használni.

Szintaxisa:
UDP.remotePort();

Paraméterei:
nincs

Példakód:

```
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);
```

```
unsigned int localPort = 8888; // helyi port az UDP-nek

EthernetUDP Udp; // UDP példányosítás

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; // átmeneti tároló

void setup() {
  // ethernet és UDP indítása
  Ethernet.begin(mac,ip);
  Udp.begin(localPort);
}

void loop() {

  // lekérjük az elérhető datagram méretét
  int packetSize = Udp.parsePacket();
  if(packetSize) // ha nem 0
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
    Serial.print("From ");
    // lekérjük a távoli kapcsolat IP-címét
    IPAddress remote = Udp.remoteIP();
    for (int i =0; i < 4; i++) // 4 bájtból áll
    {
      //kinyomtatjuk bájtonként decimálisan
      Serial.print(remote[i], DEC);
      if (i < 3)
      {
        //pontokkal elválasztva
       Serial.print(".");
      }
    }
    Serial.print(", port ");
    // lekérjük a távoli kapcsolat portjának számát
    Serial.println(Udp.remotePort());

    // beolvassuk a datagrammot az átmeneti tárolóba
    Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
    Serial.println("Contents:");
     // kiírjuk soros terminálra a tároló tartalmát
    Serial.println(packetBuffer);
 }
}
```

 © Harsányi Réka, Juhász Márton András

14. Könyvtárak hasznos eszközökhöz (háttértár, kijelző, motorok, programozó)

14.1. EEPROM könyvtár

Az Arduino három különböző típusú memóriával rendelkezik, az egyes típusok kapacitása a mikrokontroller fajtájától függ:

- *Flash*: Többszörösen újraírható memória, tartalma az eszköz kikapcsolását követően is megmarad. A fejlesztőkörnyezet által lefordított kódunk itt tárolódik gépi utasítások formájában.
- *RAM*: Gyors hozzáférésű memória s programunk futtatása közben ideiglenesen tárolt adatok számára. Tartalma kikapcsoláskor elvész.
- *EEPROM*: Flashnél régebbi technológia a kikapcsolást követően is megőrzendő adatok, konfigurációs beállítások, kezdeti állapotok, hosszabb szövegek eltárolásához. Nagyságrendileg 100.000 törlés az életideje.

Az EEPROM könyvtár azokat a funkciókat fogja össze, amikkel bájtonként tudjuk írni és olvasni a beépített memória ezen típusát. Az Arduino Mega kontrollerének (ATmega1280 és ATmega2560) **4096**, az UNO és Duemilanove (ATmega328) **1024**, a Diecimila, NG (ATmega168) és Digital (ATmega8) **512** bájt a kapacitása.

A könyvtár függvényei:

- read(),
- write().

14.1.1. EEPROM.read()

Visszaadja az EEPROM adott címén található bájt értékét.

Szintaxisa:
EEPROM.read(address).

Paraméterei:
address: a memóriacím, ahonnan az olvasás kezdődik (0-tól számozva).

Példakód:

```
#include <EEPROM.h> // importáljuk az EEPROM könyvtárat
```

```
// legyen az olvasás kezdő címe 0
int address = 0;
byte value;


void setup()
{ Serial.begin(9600); }

void loop()
{
  // value változóba töltsük be a címen található bájtot
  value = EEPROM.read(address);

  Serial.print(address); // nyomtassuk ki a címet
  Serial.print("\t"); // tabulátor
  Serial.print(value, DEC); // a kiolvasott bájt decimálisan kifejezve
  Serial.println(); // soremelés

  // növeljük eggyel a cím értékét
  address = address + 1;

  // az első 512 bájtot olvassuk, aztán ugorjunk az elejére
  if (address == 512)
    address = 0;

  delay(500);
}
```

14.1.2. EEPROM.write()

Beállítja az EEPROM adott címén a megadott bájt értékét. Egy bájt kiírása 3,3 ms-ot vesz igénybe, ami a többi memóriatípus sebességét tekintve igen lassú folyamatnak számít, használata kihathat a teljes kódunk futására. Ne felejtsük el, hogy az EEPROM 100.000 írás után használhatatlanná válik!

Szintaxisa:
EEPROM.write(address, value).

Paraméterei:
address: a memóriacím, ahonnan az olvasás kezdődik (0-tól számozva),
value: a bájt értéke, amit szeretnénk beírni.

Példakód:

```
#include <EEPROM.h>  // importáljuk az EEPROM könyvtárat
void setup() {
// írjuk bele az első 512 bájtba a saját címének értékét
for (int i = 0; i < 512; i++)
EEPROM.write(i, i);
}

void loop() { }
```

14.2. LiquidCrystal könyvtár

A Hitachi HD44780 folyadékkristályos kijelző meghajtó IC hosszú időn keresztül uralta a piacot, mint a rendkívül populáris, kizárólag monokróm szöveg megjelenítésére alkalmas faxok, telefonok, lézernyomtatók, fénymásoló gépek kijelzőjének vezérlője. Mostanra hobbisták és gyors prototipizálók közkedvelt megoldásává vált a mikrokontrollerek alfanumerikus üzeneteinek megjelenítésére. Az IC által meghajtott kijelzők pár általános szabálynak felelnek meg. Egy karakter mindig 5 vízszintes és 8 függőleges pixelből áll. A leggyakoribb típus egy sorban 8 karakter megjelenítésére alkalmas (8 x 1), de létezik 16 x 2, 20 x 2 és 20 x 4-es is. A kijelzők készülhetnek LED-es, fluoreszcens, elektrolumineszcens háttérvilágítással, vagy anélkül. Az IC maga is egy mikrokontroller, ROM-jában 208 karakter térképét tárolja – japán verziója a kana, európai verziója nyugati és cirill ABC betűiből áll. A LiquidCrystal könyvtár a vezérlő IC instrukciós kódjait és kommunikációs protokollját implementálta pár könnyen használható függvény formájában.



14.1 ábra: 16 x 2-es alfanumerikus LCD HD44780-as vezérlővel

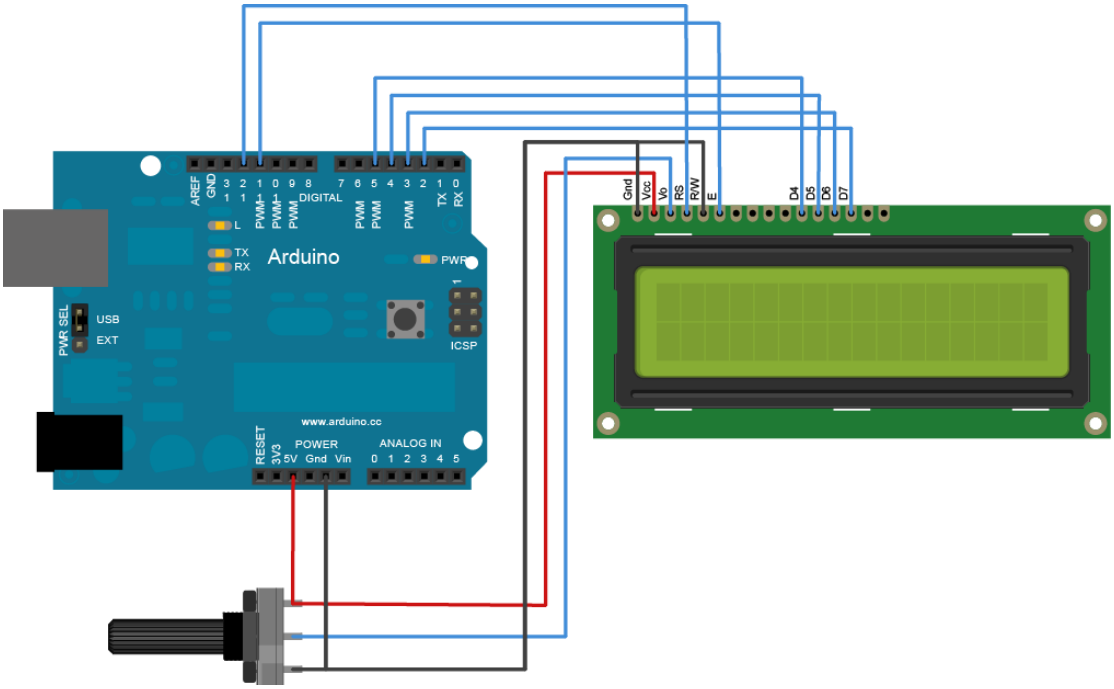
A kijelző csatolófelülete (interface), amihez kapcsolódhatunk, egy párhuzamos interfész – a 8 bitnyi adatot minden órajelre egyszerre kell megadni (**D0-D7** lábak). Szerencsére szöveges üzenetekhez 4 bites üzemmódot is választhatunk, ami szabadon hagy 4 Arduino csatornát más célokra. Az **RS** (Register Select) láb logikai alacsonyra vagy magasra állításával választhatjuk meg, hogy az adatvonalak a kijelző mely memóriaterületére írjanak. Az írást az **EN** (Enable) lábbal engedélyezhetjük. Az adatregiszter tárolja az ASCII kódolású bájtokat, amik karakterként kerülnek megjelenítésre, az instrukciós regiszterben pedig utasítások tárolódnak a meghajtó számára. Ezek az utasítások a kijelző törlésére, a kurzor mozgatására és egyéb funkciókra vonatkoznak. A regiszterek állapotát vissza is olvashatjuk – az **RW** láb állapotának változtatásával megfordítható a D0-D7 adatvonalak iránya. A kijelző kontrasztját analóg módon a tápfeszültség leosztásával **Vo** lábon tudjuk beállítani (vagy digitális potméterrel). A háttérvilágítást annak típusától függően a kontraszthoz hasonló módon kezelhetjük. Megfelelő illesztéssel LED-es háttérvilágítást az Arduino egy PWM képes kimenetére is köthetjük, így programozhatóvá válik a fényerő állítása is.

Amire szükség lesz:

- Arduino,
- LCD kijelző HD44780-as meghajtóval,
- 10 kiloohmos potméter,
- próbapanel,
- vezetékek.

Kapcsolás:

Kössük az LCD RS lábát az Arduino digitális 12-es csatornájára, az EN lábát pedig a digitális 11-re. Az LCD 4-es, 5-ös, 6-os, 7-es adatvonalait pedig rendre a digitális 5, 4, 3, 2 csatornákra. A 10 kiloohmos potméterrel készítsünk feszültségosztót az 5 V-os tápvezetékről és kössük be a potméter közepét az LCD kontrasztját szabályozó Vo lábra. Az LCD 5 V-os tápfeszültségről működik (GND, Vcc). RW-t kössük a földre!



14.2. ábra: LCD kijelző Arduinohoz kapcsolása

Példakód:

```
#include <LiquidCrystal.h>
// LCD nevű kijelző példány lábainak megadása RS, EN, D4, D5, D6, D7 sorrendben
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
    lcd.begin(16, 2);   // állítsuk be az oszlopok és sorok számát
    lcd.setCursor(0, 1); // első sor (0), első karakterére állítja a kurzort
    lcd.print("hello, world!");   // a szöveg kiírása
}

void loop() {}
```

14.2.1. LiquidCrystal()

Létrehozza a kijelző egy példányát, ami 4 vagy 8 bites módban fog működni, attól függően, hogy a híváskor hány adatvonalat adunk meg.

Szintaxisa:
LiquidCrystal(RS, EN, D4, D5, D6, D7),
LiquidCrystal(RS, RW, EN, D4, D5, D6, D7),
LiquidCrystal(RS, EN, D0, D1, D2, D3, D4, D5, D6, D7),
LiquidCrystal(RS, RW, EN, D0, D1, D2, D3, D4, D5, D6, D7).

Paraméterei:
RS: Annak a digitális csatornának a száma, amihez a REGISTER SELECT van kötve.
EN: Annak a digitális csatornának a száma, amihez az ENABLE van kötve.
RW: Annak a digitális csatornának a száma, amihez a READ/WRITE van kötve.
D0-D7: Azok a digitális csatornák, amikhez az adatvonalak kapcsolódnak.

14.2.2. begin()

A kijelző méreteit, sorok és oszlopok számát határozza meg a könyvtár számára.

Szintaxisa:
lcd.begin(cols, rows).

Paraméterei:
lcd: a LiquidCrystal példány neve,
cols: a kijelző oszlopainak száma,
rows: a kijelző sorainak száma.

14.1.2.3. clear()

Üresre törli az megjelenítendő karaktereket tartalmazó adatregisztert és a kijelzőt.

Szintaxisa:
lcd.clear().

Paramétere:
lcd: a LiquidCrystal példány neve.

14.2.4. home()

A kurzort az első sor első karakterére – kijelző bal felső sarkába – mozgatja.

Szintaxisa:
lcd.home().

Paramétere:
lcd: a LiquidCrystal példány neve.

14.2.5. setCursor()

A kurzort a megadott helyre mozgatja.

Szintaxisa:
lcd.setCursor(cols,rows).

Paraméterei:
lcd: a LiquidCrystal példány neve,
cols: az oszlop, ahová a kurzort mozgatjuk,
rows: a sor, ahová a kurzort mozgatjuk.

14.2.6. write()

Egy bájtot, vagyis karaktert ír ki a kijelzőre.

Szintaxisa:
lcd.write(data).

Paraméterei:
lcd: a LiquidCrystal példány neve,
data: a megjelenítendő karakter ASCII-kódja.

Példakód:

```
   void loop()
{
  if (Serial.available()) {
    lcd.write(Serial.read());
  }
}
```

14.2.7. print()

Szöveget jelenít meg a kijelzőn.

Szintaxisa:
lcd.print(data),
lcd.print(data, BASE).

Paraméterei:
lcd: a LiquidCrystal példány neve,
data: char, byte, long, int, string – amit megjelenítünk,
BASE: számrendszer alapja (opcionális) BIN, HEX, OCT, DEC.

14.2.8. cursor()

Hívását követően egy alulvonásként megjeleníti a kurzort.

14.2.9. noCursor()

Hívását követően kikapcsolja a kurzor megjelenítését.

14.2.10. blink()

Ha a kurzor látható, akkor bekapcsolja a villogtatását.

14.2.11. noBlink()

Ha a kurzor látható, akkor kikapcsolja a villogtatását.

14.2.12. noDisplay()

Kikapcsolja a kijelzőt.

14.2.13. display()

Visszakapcsolja a kijelzőt, ha előtte a noDisplay() hívással ki volt kapcsolva, és újra megjeleníti a szöveget, ami a kikapcsolás pillanatában volt látható.

 © Harsányi Réka, Juhász Márton András

14.2.14. scrollDisplayLeft()

Hívásakor egy karakterrel eltolja a kijelző teljes tartalmát balra.

14.2.15. scrollDisplayRight()

Hívásakor egy karakterrel eltolja a kijelző teljes tartalmát jobbra.

Példakód:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
     lcd.begin(16, 2); // 16 karakter egy sorban
     lcd.print("hello, world!"); // 13 karakterből álló szöveg
     delay(1000);
}

void loop() {

   // "letolja" a szöveget a kijelzőről balra
     for (int positionCounter = 0; positionCounter < 13; positionCounter++) {
       lcd.scrollDisplayLeft();
       delay(150);
  }

   // 13 + 16 "letolja" a szöveget a kijelzőről jobbra
  for (int positionCounter = 0; positionCounter < 29; positionCounter++) {
       lcd.scrollDisplayRight();
       delay(150);
  }

   delay(1000);
}
```

14.2.16. autoScroll()

Bekapcsolja a karakterek automatikus eltolását. Amikor egy új karaktert jelenítünk meg, a beállított iránynak megfelelően a már meglévő szöveget egy pozícióval elmozgatja.

Példakód:

```
#include <LiquidCrystal.h>

// kijelző inicializálása
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // oszlopok és sorok megadása
  lcd.begin(16,2);
}

void loop() {
  // kurzor a kezdő állapotban (0,0) - home() ugyanezt teszi
  lcd.setCursor(0, 0);
  // kiírja a számokat 0-tól 9-ig
  for (int thisChar = 0; thisChar < 10; thisChar++) {
     lcd.print(thisChar);
     delay(500);
  }

  // kurzort a második sor végére állítja (16,1)
  lcd.setCursor(16,1);
  // bekapcsolja az automatikus eltolást
  lcd.autoscroll();
  // újra kinyomtatja a számokat 0-tól 9-ig
  for (int thisChar = 0; thisChar < 10; thisChar++) {
    lcd.print(thisChar);
    delay(500);
  }
  // kikapcsolja az automatikus eltolást
  lcd.noAutoscroll();

  // letörli a kijelzőt
  lcd.clear();
}
```

14.2.17. noAutoscroll()

Kikapcsolja a karakterek automatikus eltolását.

14.2.18. leftToRight()

Az automatikus karaktereltolás irányának beállítása balról jobbra.

 © Harsányi Réka, Juhász Márton András

14.2.19. rightToLeft()

Az automatikus karaktereltolás irányának beállítása jobbról balra.

14.2.20. createChar()

Segítségével létrehozhatunk olyan karaktereket, amit nem tartalmaz a ROM. Ezeket az 5 x 8 pontból álló jeleket glyph-nek is szokás nevezni. A vezérlő 8 különféle glyph-et tud eltárolni a létrehozása után.

Szintaxisa:
lcd.createChar(num, data).

Paraméterei:
lcd: a LiquidCrystal példány neve,
num: 0–7 a glyph száma, amelyik pozíción tárolni szeretnénk,
data: a karakter pixeleit tartalmazó tömb.

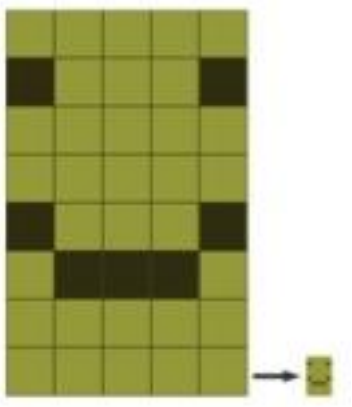
Példakód:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte smiley[8] = {
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
};

void setup() {
  lcd.createChar(0, smiley);
  lcd.begin(16, 2);
  lcd.write(0);
}

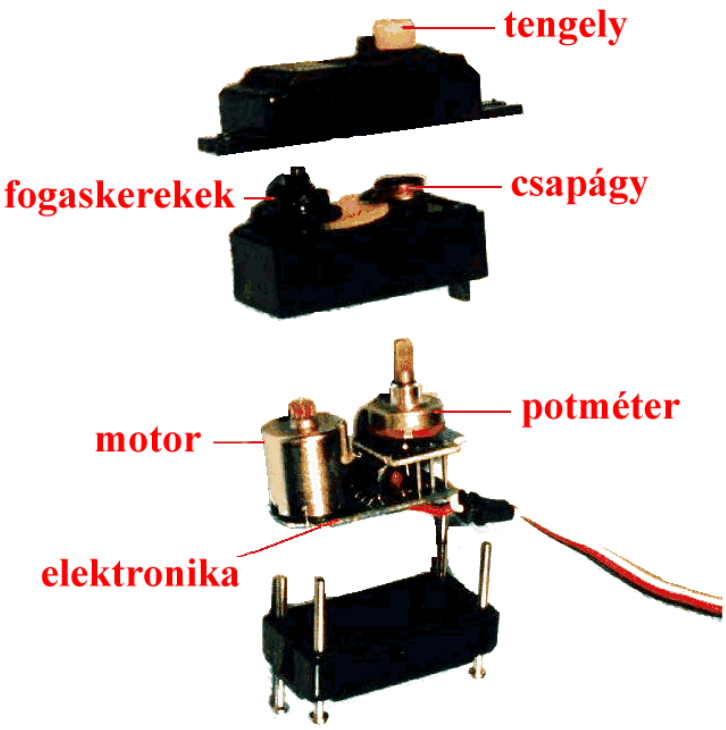
void loop() {}
```



14.3. ábra: Smiley glyph

14.3. Servo könyvtár

A szervomotorok olyan aktuátorok, melyek a visszacsatolás elvén működnek. A legegyszerűbb esetben a szervo tengelyét elfordító motor egy áttételen keresztül egy potmétert is elforgat, ami egy komparátor áramkör egyik bemenetéhez kapcsolódik. A komparátor egyszerű analóg integrált áramkör, amit legegyszerűbben úgy képzelhetünk el, mint egy mérleget. A szervo esetében a „mérleg” egyik oldalán a potméter – és így a tengely – állapota, a másik oldalán a szervo bemenetére adott feszültségszint a „súly”, a „mérleg nyelve” pedig a motor elfordulási irányát és erejét reprezentálja. Ily módon meghatározott feszültségszinteknek megfeleltethetők a motortengely állásának szögei. A Servo könyvtárral az Arduino PWM képes kimeneteit felhasználva, a csatlakoztatott szervomotorokat az elfordulás szögének megadásával irányíthatjuk. Miután a potméter nem képes a körbeforgásra, a megadható szögek 0 és 180 fok közé esnek és mindig egész számok.



14.4. ábra: Szervomotor szerkezete

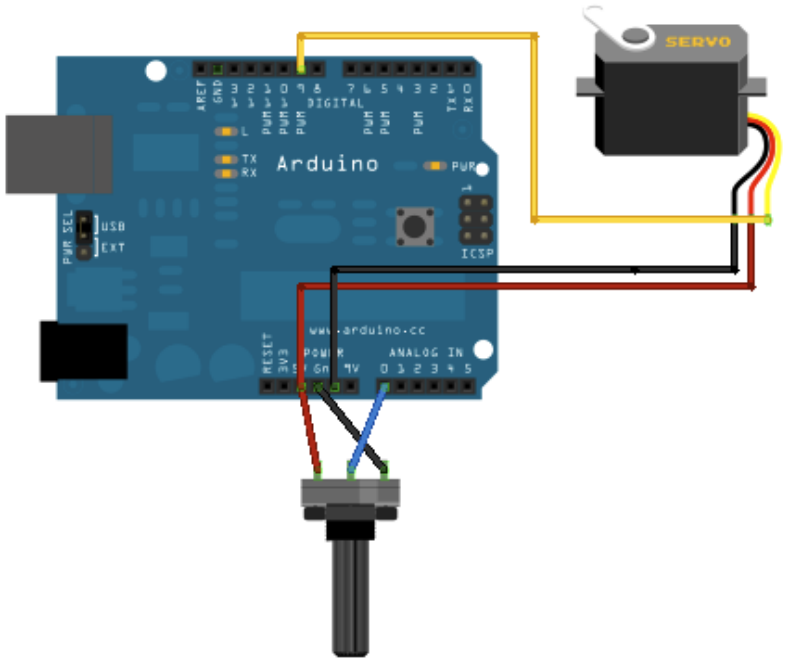
(A bonyolultabb felépítésű szervomotorok komparátor helyett digitális jelfeldolgozást is végeznek. A potméter szerepét optikai vagy mágneses kódtárcsa váltja fel és a kommunikáció a motorral digitális buszon zajlik.) Hobbi és modellezési célra három méretben szabványosították a szervokat. Alaptípusaikat tekintve megkülönböztetünk nagy nyomatékú és nagy sebességű motorokat. Mindegyikre jellemző, hogy három vezeték tartozik hozzájuk: föld (barna/fekete), táp (piros) és vezérlés (narancs/fehér).

Amire szükség lesz:

- Arduino,
- hobbi szervomotor,
- 10 kiloohmos potméter.

Kapcsolás:

Kössük be a potméterünket feszültségosztóként, a közepét pedig az A0 analóg bemenetre. A szervomotor tápkábeleit az Arduino 5 V-os tápvezetékeihez, vezérlőkábelét a 9-es PWM kimenethez csatlakoztassuk.



14.5. ábra: Szervomotor és potméter bekötése

Példakód:

```
#include <Servo.h>

Servo myservo;  // Servo objektum létrehozása myservo néven
int val;    // a potméter állapotát fogja tárolni

void setup()
{ myservo.attach(9); }  // myservo objektumhoz a 9-es lábat társítjuk

void loop()
{
  val = analogRead(0);  // potméter állapota 0-1023
  val = map(val, 0, 1023, 0, 179);     // átalakítjuk 0-179
  myservo.write(val);                  // beállítjuk a szervot pozícióba
  delay(15);                           // várunk, amíg a szervo odaért
}
```

14.3.1. attach()

Szervo objektumhoz fizikai csatlakozó címét köti.

Szintaxisa:
Servo.attach(pin),
Servo.attach(pin, min, max).

Paraméterei:
Servo: Servo típusú változó,
pin: a PWM képes kimenet, amire a motor vezérlését kötöttük,
min: a vezérlőimpulzus szélességének minimuma mikroszekundumokban kifejezve, alapállapotban 544 és ez feleltethető meg a motoron a 0 foknak. Opcionális finomhangolásra ad lehetőséget.
max: a vezérlőimpulzus szélességének maximuma, alapállapotban 2400 mikroszekundum.

14.3.2. write()

A megadott pozícióba mozgatja a szervomotort. Szabvány szervo esetén a megadott szögnek megfelelően mozdul el a motor tengelye. (Ha szétszedjük a szervomotort és a potméter helyére két egyforma értékű ellenállást teszünk, mintha a potméter mindig középen állna, akkor folyamatosan forgó mozgást végző motort fogunk kapni, aminek ez a parancs a sebességét fogja állítani. 180 esetén az egyik irányba forog teljes sebességgel, 0 esetén a másikba és 90-nél megáll.)

Szintaxisa:
Servo.write(val).

Paramétere:
Servo: Servo típusú változó,
val: a szög, ahová a motort fordítani szeretnénk.

14.3.3. writeMicroseconds()

Abban az esetben, ha precízebb mozgatásra van szükségünk, célszerű nem szögértékben megadni az elfordulás mértékét, hanem a vezérlő impulzus szélességét közvetlenül mikroszekundumokban. Ideális esetben 1000 körül az óramutató járásának megfelelő irányú végállapotot, 2000 körül az ellenkező végállapotot és 1500 körül a középső állapotot kellene jelentenie. Ez a gyakorlatban nem jellemző, valahol 700 és 2300 körüli értékek az irányadók.

Szintaxisa:
Servo.writeMicroseconds(val).

Paramétere:
Servo: Servo típusú változó,
val: impulzus szélessége mikroszekundumokban.

14.3.4. read()

Visszaadja a write() utasításnak legutoljára megadott értéket.

Szintaxisa:
Servo.read().

14.3.4. attached()

Megvizsgálja, hogy egy Servo objektumhoz kötöttünk-e fizikai csatlakozót. Bool típusú változót ad vissza.

Szintaxisa:
Servo.attached().

14.3.5. detach()

Megszünteti a Servo objektum fizikai címhez kötését.

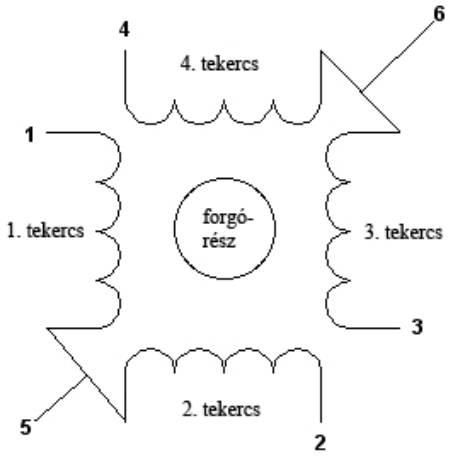
Szintaxisa:
Servo.detach(pin).

Paraméterei:
Servo: Servo típusú változó,
pin: a PWM képes kimenet, amire a motor vezérlését kötöttük.

14.4. Stepper Motor könyvtár

A léptetőmotorok olyan egyenáramú motorok, amiknek az állórésze több elektromágnest tartalmaz. A tengelyen elhelyezett aktív mágnest (forgórész) az egymás után szekvenciálisan be-/kikapcsolt elektromágnesek kis lépésekben elfordítják. Jelentőségük az, hogy visszacsatolás nélkül nagy pontosságú és apró lépésekből álló forgómozgást lehet

velük elérni. Két alaptípust különböztetünk meg az elektromágnesek tekercseinek kivezetései alapján. Az *unipoláris* motorok esetén minden tekercs egyik vége közös vezetékként hagyja el a motor burkolatát, míg a *bipoláris* motorok esetén a tekercsek kivezetései szabadon állnak.



14.6. ábra: Unipoláris léptetőmotor vázlatrajza

A tekercseket meghatározott rendben kell működtetni, hogy a felépülő mágneses tér négy lépésben fordítson a tengelyen:

Lépés	1. tekercs	2. tekercs	3. tekercs	4. tekercs
1	MAGAS	ALACSONY	MAGAS	ALACSONY
2	ALACSONY	MAGAS	MAGAS	ALACSONY
3	ALACSONY	MAGAS	ALACSONY	MAGAS
4	MAGAS	ALACSONY	MAGAS	ALACSONY

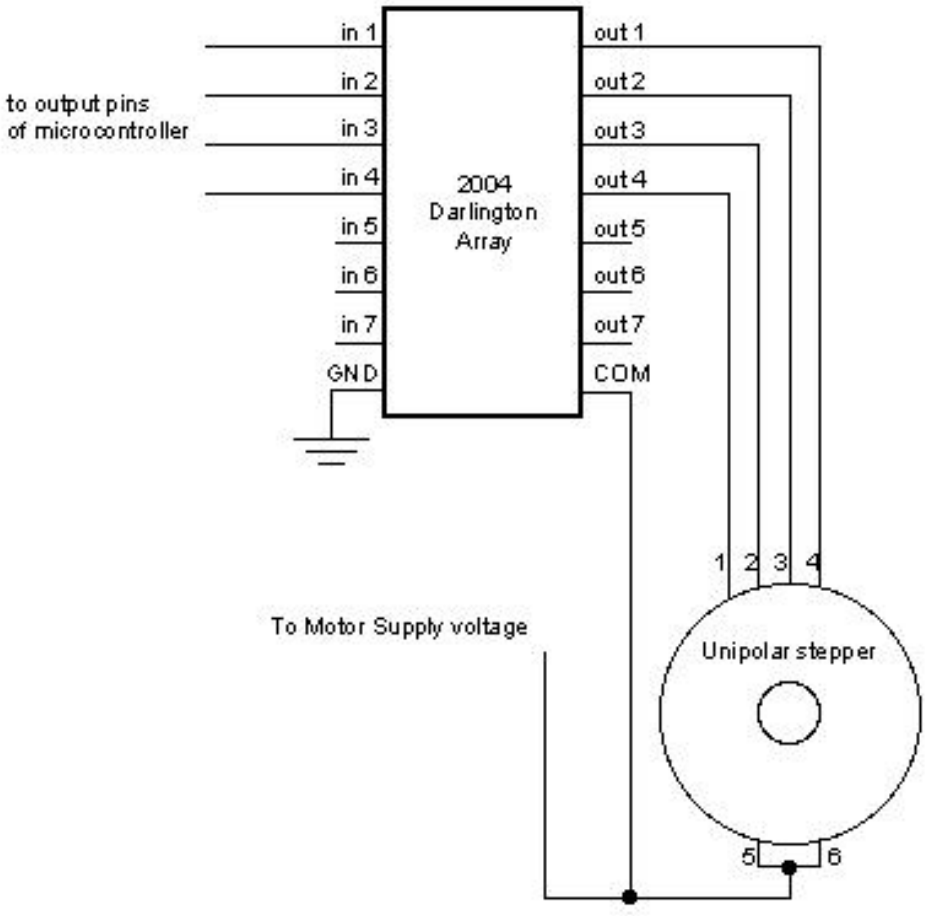
14.1. táblázat: Logikai jelek a negyed lépésekhez

A léptetőmotorokat leíró legfontosabb adat, hogy egy lépés alatt hány fokot fordulnak el. Egy 1,8 fok/lépés precizitású motor 200 lépésben fordul teljesen körbe, tehát a 14.1 táblázatban látható lépéseket fordulatonként 200-szor kell megismételni.

14.4.1. Unipoláris léptetőmotorok

A léptetőmotorok általában több áramot vesznek fel működésükhöz, mint amennyit az Arduino digitális csatornáin keresztül kezelni képes, és sok esetben a működésükhöz

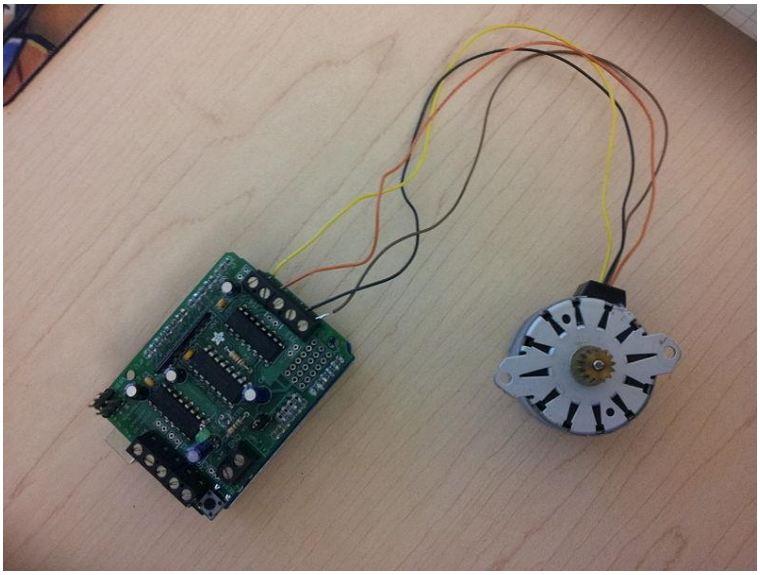
szükséges feszültség sem 5 V, hanem magasabb. Az alábbi kapcsolásban szereplő integrált áramkör az ULN2004 egy teljesítmény-tranzisztorokból álló úgynevezett Darlington sor. Univerzálisan felhasználhatjuk az Arduino kimeneteinek erősítésére, csatornánként 50 V-ig 500 mA-t képes kapcsolni. Minden bemenetre adott magas szintű jel a bemenethez tartozó kimenet és a föld közé kötött fogyasztót üzembe helyezi. A COM jelzésű lábon kell, legyen az a közös tápfeszültség, amit az IC-vel kapcsoltatni szeretnénk.



14.7. ábra: ULN2804, mint léptetőmotor meghajtó

Az unipoláris motorok általában 5 vagy 6 kivezetéssel rendelkeznek. A vezetékek közül 4 a forgórészt körülfogó tekercsek egy-egy vége, az ötödik pedig az összes tekercs másik vége egy vezetéken. Amennyiben 6 vezetéket látunk, akkor a motorban két tekercs van, amik két-két vége az első négy kivezetés, az ötödik és hatodik elvezetések pedig a tekercsek közepére csatlakoznak. A tekercsek ellenállásának megmérésével – egy műszer

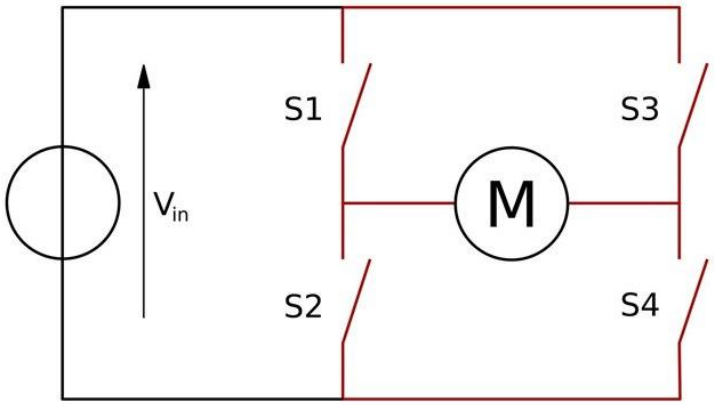
segítségével – kikövetkeztethetjük, hogy melyik kivezetés hová tartozik. A közös elvezetés egyben azt is jelenti, hogy a tekercsek és a mágneses tér polaritása mindig ugyanaz lesz.



Arduino motor-shield és léptető motor

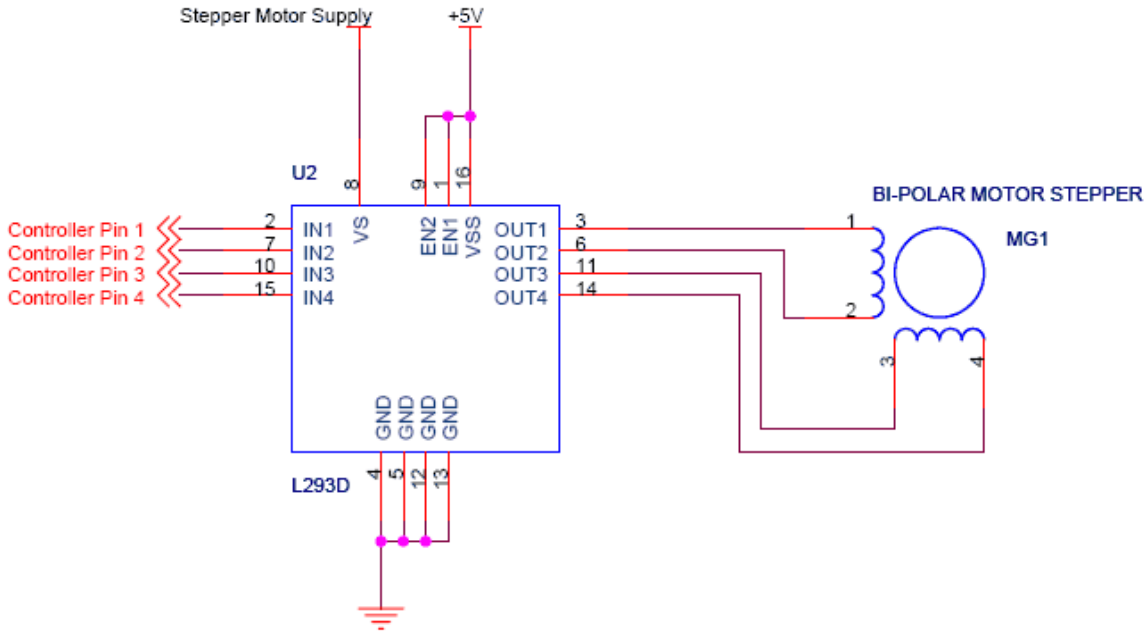
14.4.2. Bipoláris léptetőmotorok

Bipoláris motorok esetén a tengely forgatásához az elektromágnesek polaritását is meg kell változtatni, ezért a két tekercs mindkét kivezetése szabadon áll. A polaritás megváltoztatásának legegyszerűbb módja H-híd kapcsolással valósítható meg, ami négy tranzisztorból álló áramkör. Képzeljük el úgy, mint négy kapcsolót, amik közül kettő a tápvezetékek pozitív, kettő pedig a negatív ágát tudja kapcsolni. Amikor (14.8. ábra) S1 és S4 megszakítja, S2 és S3 zárja az áramkört, az áram az M-mel jelölt motorban az egyik irányba folyik. Megfordítva a kapcsolók állását, az áram iránya is megfordul, csakúgy, mint a tekercs által keltett mágneses tér polaritása.



14.8. ábra: H-híd kapcsolás funkcionális rajza

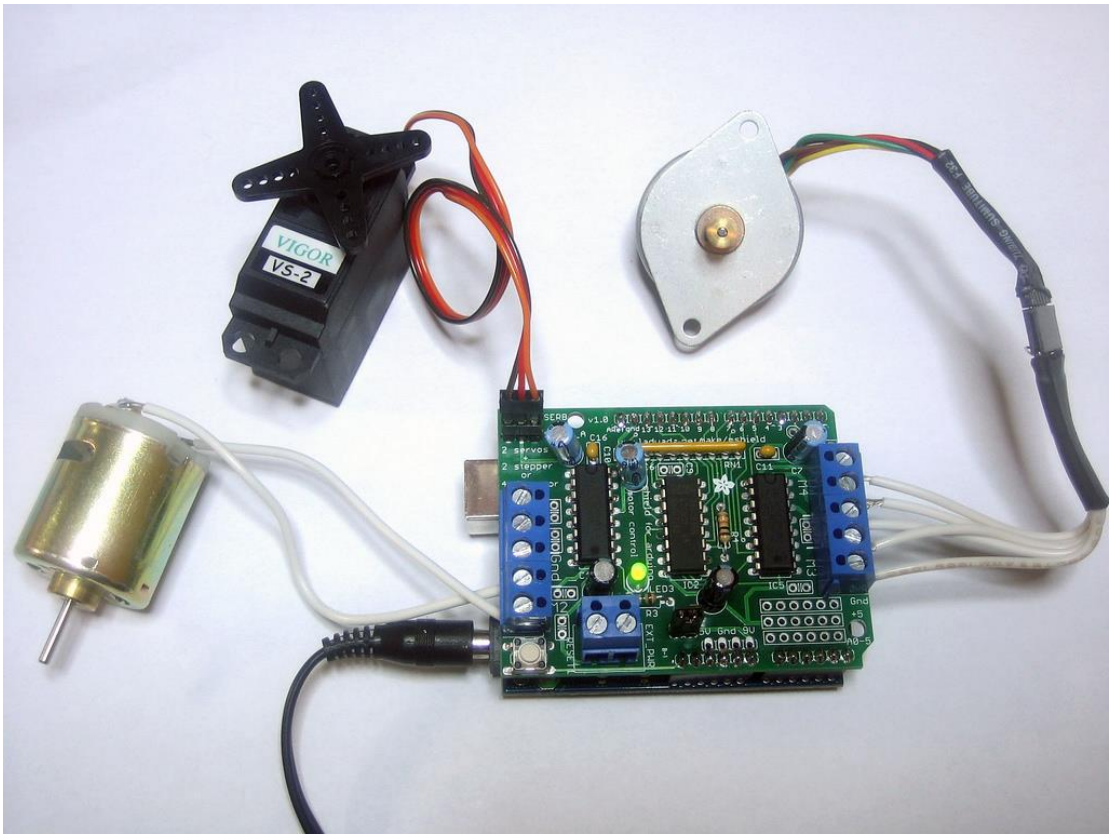
A gyakorlatban használhatunk integrált H-híd áramkört (pl. L293D) két egyenáramú motorok forgásirányának váltására vagy egy bipoláris léptetőmotor meghajtására.



14.9. ábra: L293D integrált H-híd bekötése bipoláris léptetőmotorral

Amennyiben nem szeretnénk az áramkörök megépítésével fáradozni, különféle motorok meghajtására a legkézenfekvőbb az Adafruit által gyártott MotorShield. A kiegészítő modul tartalmaz két darab L293D-t és a vezérlésüket leegyszerűsítő logikai áramkört. Alkalmazásával egyidejűleg meg tudunk hajtani 4 db egyenáramú motort, vagy két léptetőmotort és mindezektől függetlenül két szervomotort is.

 © Harsányi Réka, Juhász Márton András



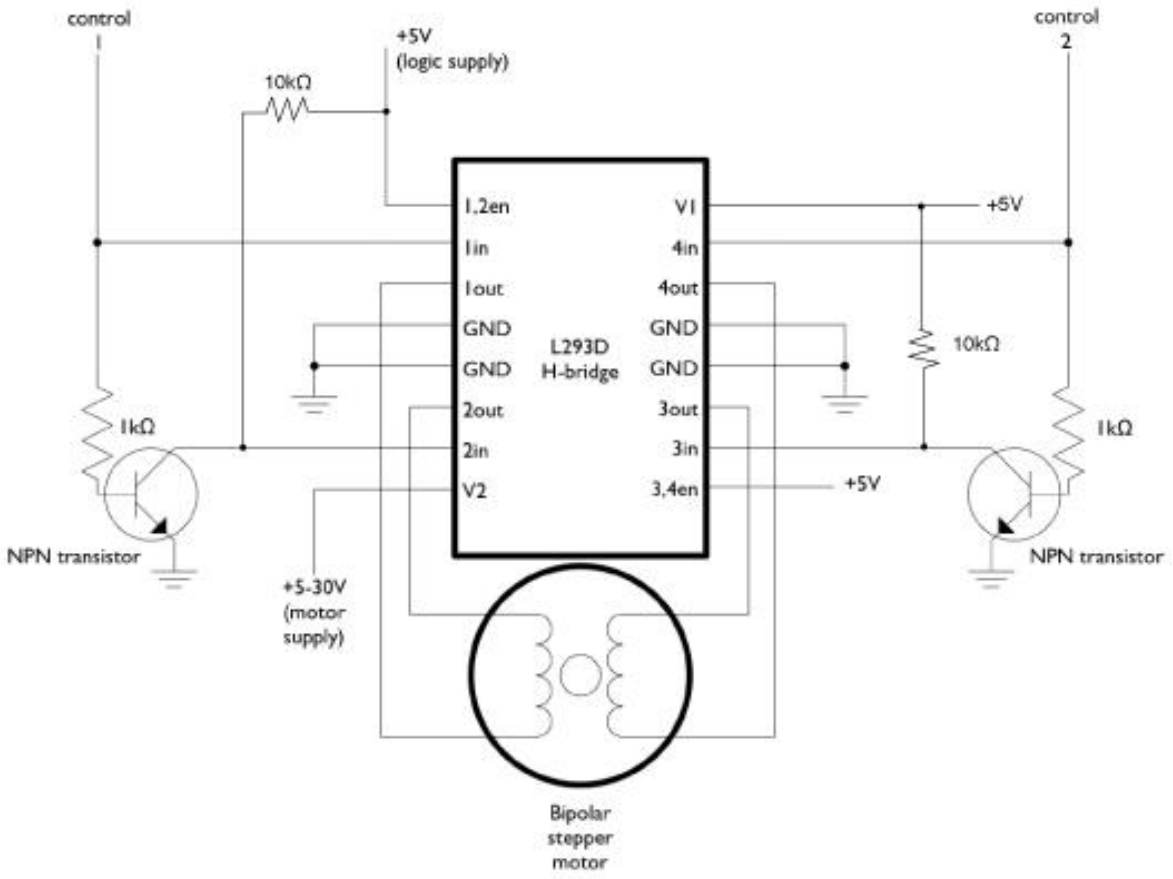
14.10. ábra: Adafruit MotorShield

14.4.3. A stepper motor könyvtár használata

Amire szükség lesz:

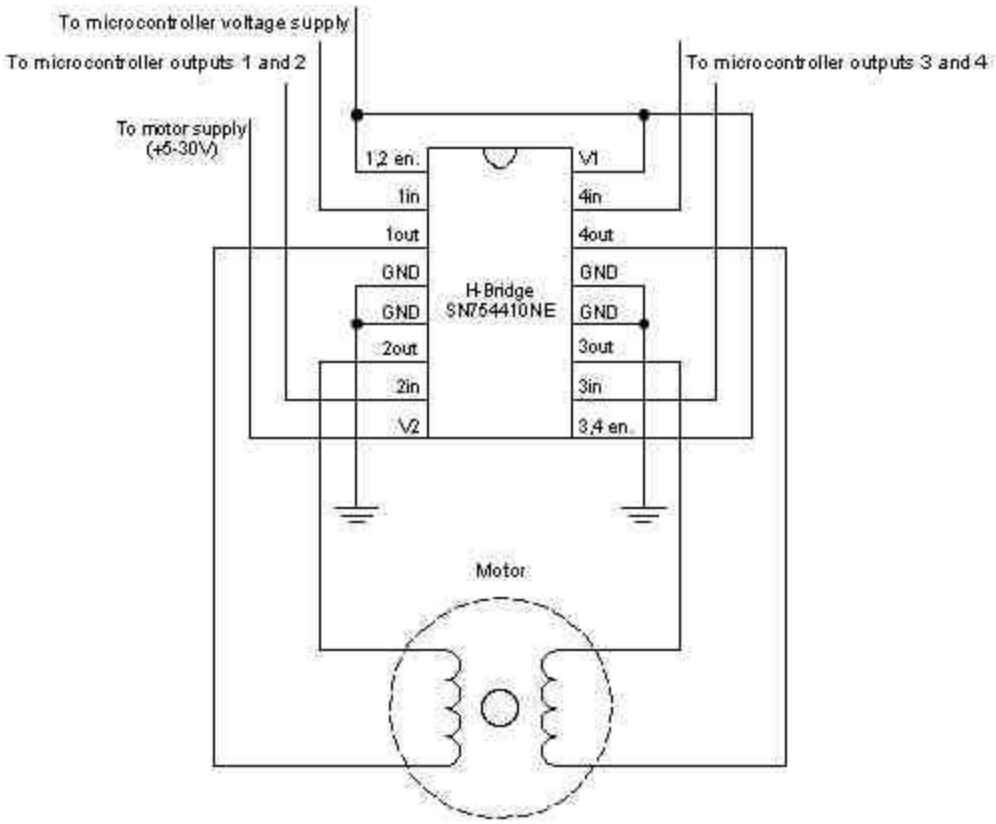
- Arduino,
- unipoláris vagy bipoláris léptetőmotor,
- ULN2804 vagy L293D vagy Adafruit MotorShield,
- 10 kiloohmos potméter.

Kapcsolások:



2 pines bipoláris stepper motor L293D H-híddal

 © Harsányi Réka, Juhász Márton András



4 pines bipoláris stepper motor L293D H-híddal

A fentieknek megfelelően csatlakoztassunk egy léptetőmotort, és jegyezzük meg, melyik tekercs melyik Arduino kivezetéssel hozható működésbe. A potmétert feszültségosztóként a középső kivezetésével az A0 analóg bemenetre kössük.

Példakód:

```
#include <Stepper.h>

// a motorunk körbefordulásához szükséges lépések száma
#define STEPS 100

// Stepper osztály példányosítása a lépések számával és a tekercseknek
// megfelelő Arduino csatornák sorrendjével
Stepper stepper(STEPS, 8, 9, 10, 11);
```

```
// az analóg bemeneten olvasott előző érték
int previous = 0;

void setup()
{
  // a motor sebességét 30 fordulat/percre állítjuk
  stepper.setSpeed(30);
}

void loop()
{
  // olvassuk be a potméter állását
  int val = analogRead(0);

  // léptessük a motort a potméter elfordításának különbségével
  stepper.step(val - previous);

  // tároljuk el a potméter állapotát
  previous = val;
}
```

14.4.4. A stepper motor könyvtár függvényei

14.4.4.1. Stepper()

Létrehozza a Stepper osztály egy példányát a megadott lépésszámmal és tekercsekhez tartozó Arduino csatornákkal.

Szintaxisa:
Stepper(steps, pin1, pin2),
Stepper(steps, pin1, pin2, pin3, pin4).

Paraméterei:
steps: az egy körülforduláshoz szükséges lépések száma (motortól függ),
pin1, pin2: kétvezetékes meghajtás esetén (kiegészítő logikai áramkör szükséges) a meghajtó vezérléséhez használt csatornák száma,
pin1, pin2, pin3, pin4: négyvezetékes meghajtás esetén a tekercsekhez tartozó csatornák száma.

Példakód:
Stepper myStepper = Stepper(200,4,5,6,7);

14.4.4.2. setSpeed()

Beállítja a motor forgási sebességét a megadott fordulat/perc alapján.

Szintaxisa:
Stepper.setSpeed(rpm).

Paraméterei:
Stepper: Stepper osztály példánya,
rpm: forgási sebesség fordulat/percben.

14.4.4.3. step()

Adott számú lépést hajtat végre a motorral. Ha a megadott érték pozitív, akkor egyik, ha negatív, akkor a másik irányba forgat. Ez a függvény megvárja, amíg a motor megteszi a lépéseket.

Szintaxisa:
Stepper.step(steps).

Paraméterei:
Stepper: Stepper osztály példánya,
steps: lépések száma, polaritása a forgás irányát határozza meg.

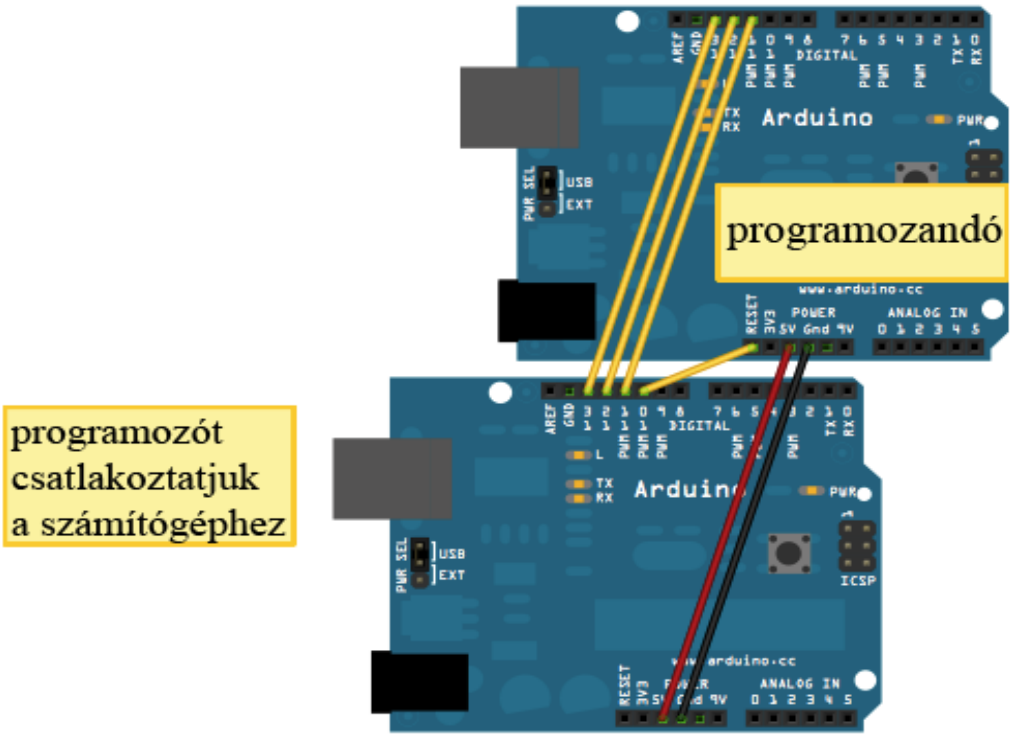
14.5. Arduino, mint ISP

Az Arduino legnagyobb előnye és egyszerű használatának kulcsa, hogy a fejlesztő-környezetből – minden egyéb eszköz nélkül – az USB porton keresztül feltölthetjük a megírt programot, ami ezt követően futtatásra kerül. Az állandó program, ami ezt lehetővé teszi, az Arduino firmware-e, az úgynevezett *bootloader*. (Az ICSP csatlakozó SPI kommunikációs funkciójának tárgyalásakor már említettük (14.1.5.), hogy a bootloader feltöltése miatt ez a felület az összes típusú lapkán megtalálható.) Az ISP (In System Programmer) egy olyan eszköz, amivel bootloader nélküli mikrokontrollerbe tölthetünk fel kódot. A környezet példakódjai között megtaláljuk az ArduinoISP firmware-t, ami AVR ISP eszközzé alakítja az Arduinot. Lehetséges alkalmazások közül az egyik leghasznosabb a tönkrement kontrollerű Arduinok javítása – elegendő kivenni a tokjából a rossz IC-t és a helyére került újba beégetni a bootloadert egy másik Arduino segítségével. Készíthetünk egyetlen kontrollerből is költséghatékony Arduinot ugyanezzel a módszerrel.

Instrukciók bootloader beégetéséhez:

1. Töltsük fel a **programozóként** használni kívánt Arduinora az ArduinoISP kódot.
2. Tegyünk egy 10 uF-os kondenzátort a programozó RESET lába és a föld közé.
3. Kössük be a programozandó eszközt az alábbi példák egyike szerint.
4. Válasszuk ki a Board menüből a **programozandó** eszköz típusát.
5. Burn bootloader menüből válasszuk az Arduino és ISP parancsot.

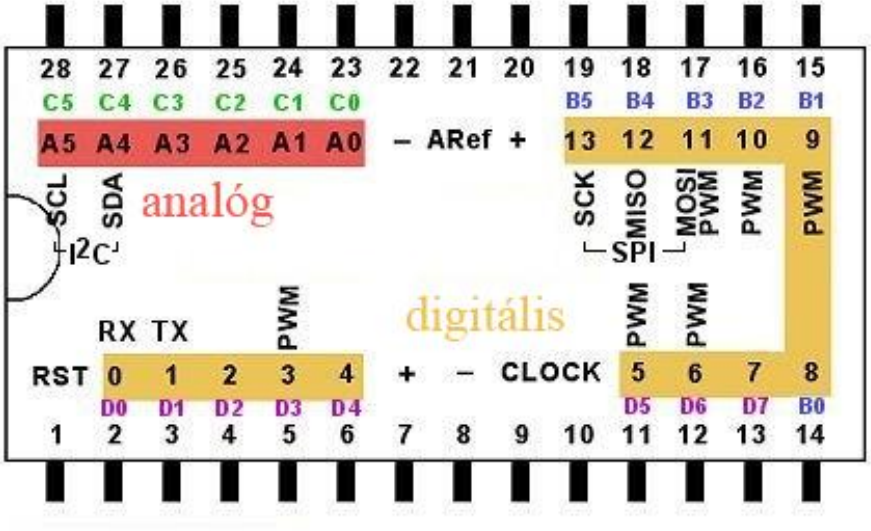
14.5.1. Arduino programozása Arduinoval



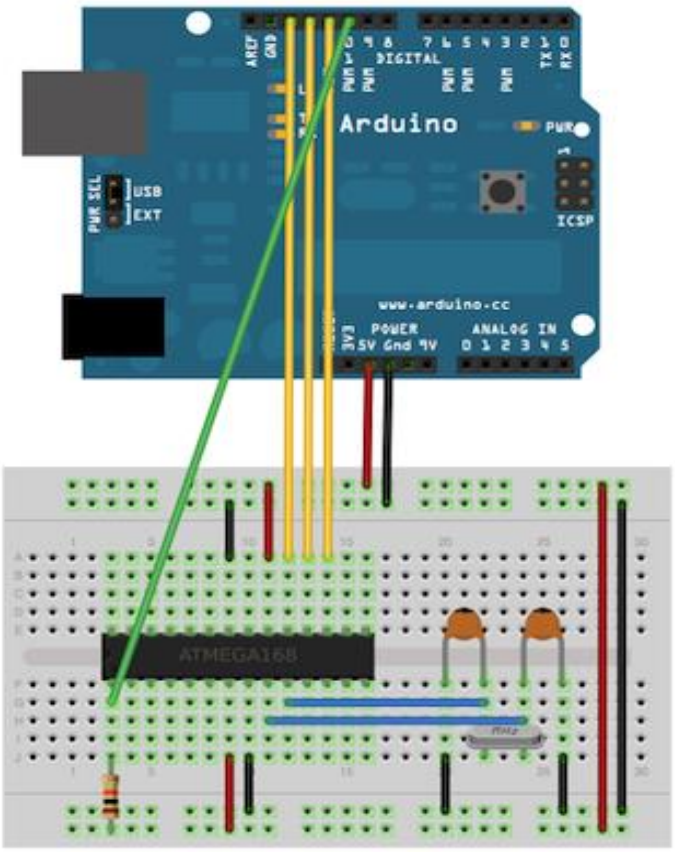
14.11. ábra: UNO, Duemilanove, Diecimila programozása

14.5.2. ATmega mikrokontroller programozása Arduinoval

Abban az esetben, ha egy feladat ellátásához nincs szükség az Arduino USB csatlakozójára és már megírtunk egy jól működő programot, feltölthetjük egy szabadon álló mikrokontrollerre, ami ugyanúgy fogja végrehajtani azt, mint az Arduinonk.



14.12. ábra: Atmega168 és Atmega328 lábkiosztása és az Arduino megfelelő csatornái



14.13. ábra: Atmega programozása

Szintaxis Gyűjtemény

A könyvtárakhoz tartozó szintaxisokat nem tartalmazza, azok részletesen a 13. fejezetben találhatóak.

SZERKEZETEK

Vezérlési szerkezetek

void setup(){}

A kód legelején definiáljuk benne a változókat (inicializálás), pin módokat, könyvtár használatát stb. Csak egyszer fut le.

Példa:

```
void setup() {
    Serial.begin(9600);
    pinMode(buttonPin, INPUT);
  }
```

forrás: (*http://arduino.cc/en/Reference/Setup*)

void loop(){}

A setup után jön, folyamatosan, azaz többször lefut, lehetővé teszi, hogy a program változzon és reagáljon.

Példa:

```
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');
    delay(1000);
}
```

forrás: (*http://arduino.cc/en/Reference/Loop*)

© Harsányi Réka, Juhász Márton András

if(feltétel**){}**

Feltétel és összehasonlító operátorok (mint nevük is sugallja, két érték összehasonlítására szolgálnak). Megvizsgálja, hogy az egyes feltételek teljesülnek-e, mint pl., hogy a bemeneti szám a megadott számérték felett marad-e. Ha a feltétel teljesül, lefut a kód, ha nem, akkor átugorja a program.

Példa:

```
if (Változó > 50) {
   // csináld ezt
 }
```

A kapcsos zárójel elhagyható, a következő szintaxisok mind helyesek:

1. if (x > 120) digitalWrite(LEDpin, HIGH);
2. if (x > 120) digitalWrite(LEDpin, HIGH);
3. if (x > 120){ digitalWrite(LEDpin, HIGH); }
4. if (x > 120){
   digitalWrite(LEDpin1, HIGH);
   digitalWrite(LEDpin2, HIGH);
   }

forrás: (*http://arduino.cc/en/Reference/If*)

if (feltétel**){} else if(**feltétel**){} else{}**

Ha az if-feltétel nem teljesül, akkor az else-feltétel fut le. Az else folytatódhat egy másik if vizsgálattal, így összetettebb vizsgálatokat futtathatunk ugyanazon időben végtelen számú else if-feltételek vizsgálatával.

Példa:

```
if (pin5Bemenet < 500) {
    // csináld A-t
  } else if (pin5Bemenet >= 1000) {
    // csináld B-t
  } else {
    // csináld C-t
  }
```

forrás: (*http://arduino.cc/en/Reference/Else*)

for(utasítás1; feltétel; utasítás2**){}**

Amíg a feltétel igaz, addig folyamatosan ismétlődve futnak le a kapcsos zárójelben lévő állítások. Az inicializálás csak egyszer fut le, onnantól mindig az aktuális állapotra vizsgálja meg a program, hogy igaz-e a feltétel. Ha igen, elvégzi az érték növelését vagy csökkentését, lefutnak az állítások. A folyamat akkor ér véget, amikor hamis lesz a feltétel. Gyakran array-jel együtt használjuk.

Példa:

```
for (int i=0; i <= 255; i++){
       analogWrite(PWMpin, i);
       delay(10);
    }
```

forrás: (http://arduino.cc/en/Reference/For)

switch (kifejezés**){case** konstans kifejezés**:** utasítás; **break; default:** utasítás**}**

Az if-hez hasonlóan különböző feltételek végrehajtásához használjuk. A switch állítás változó értékeit hasonlítja össze a case állításban részletezett értékekkel. Ha a két érték megegyezik, akkor lefut a case állításban szereplő kód. Minden case állítás végét a break kulcsszóval adjuk meg, e nélkül a következő kifejezés végrehajtása következne.

Példa:

```
switch (var) {
    case címke1:
      //csináld ezt, ha a változó 1
      break;
    case címke2:
      //csináld ezt, ha a változó értéke egyenlő 2-vel
      break;
    default:
      // ha semelyik nem egyezik, csináld az alapértelmezett
      // állítást
      // az alapértelmezett állítás opcionális
  }
```

A var a változó értéke, amit a különböző case állításokhoz hasonlítunk. A címke az az érték, amivel a változót összehasonlítjuk.

forrás: (*http://arduino.cc/en/Reference/SwitchCase*)

 © Harsányi Réka, Juhász Márton András

while(kifejezés**) {}**

Mindaddig folyamatosan fut le, amíg a kerek zárójelben lévő kifejezés hamissá nem válik. Valaminek változnia kell a vizsgált változóban, különben a folyamat soha nem ér véget. Ez történhet változó bevezetésével.

Példa:

```
var = 0;
while(var < 200){
  // 200-szor lefutnak a ciklusban lévő utasítások
  var++;
}
```

forrás (*http://arduino.cc/en/Reference/While*)

do{}while (feltétel**){}**

A do a while-hoz hasonlóan működik, de a ciklus végén lévő feltételt vizsgálja, ezért a do ciklus mindig lefut legalább egyszer.

Példa:

```
do
{
  delay(50);          // várakozás a szenzorra
  x = readSensors();  // kiolvassa a szenzorból az értéket
} while (x < 100); // amit összehasonlít
```

forrás: (*http://arduino.cc/en/Reference/DoWhile*)

break:

A do, for, while ciklusokból és a switch állításból való kilépésre használjuk.

Példa:

```
for (x = 0; x < 255; x ++)
{
    digitalWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){      // bail out on sensor detect
       x = 0;
```

```
            break;
        }
        delay(50);
    }
```

forrás: (*http://arduino.cc/en/Reference/Break*)

continue:

Átugorja az ismétlések lefuttatását a ciklusokban (do, for, while), majd a feltétel vizsgálatával és a további ismétlésekkel (a példában for ciklus) folytatja.

Példa:

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){      // ugrik az érték
        continue;
    }
    digitalWrite(PWMpin, x);
    delay(50);
}
```

forrás: (http://arduino.cc/en/Reference/Continue)

return érték**:**

return.

Befejezi a függvényt és visszaállítja a függvényből származó értéket a meghívott értékre, ha szükséges. Ebben az esetben az érték származhat bármilyen változóból vagy állandóból.

Példa: A következő függvény a szenzorból kiolvasott értéket hasonlítja össze a küszöbértékkel:

```
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;   //érték nagyobb 400-nál, 1-re állítja (be)
    else{
        return 0;   //kisebb, 0-ra állítja az értéket (ki)
    }
}
```

 © Harsányi Réka, Juhász Márton András

forrás: (http://arduino.cc/en/Reference/Return)

goto címke**:**
címke: utasítás;
A program lefolyását áthelyezi a programban az általunk elnevezett címkével megjelölt ponthoz a következő paranccsal: **goto** címke**.** Ezután visszatér és tovább futtatja a többit.

Példa:

```
for(byte r = 0; r < 255; r++){
    for(byte g = 255; g > -1; g--){
        for(byte b = 0; b < 255; b++){
            if (analogRead(0) > 250){ goto kisegítés;}
            // kisegítés: általunk elnevezett
            // többi állítás ...
        }
    }
}
kisegítés:
csináld ezt;
```

forrás: (http://arduino.cc/en/Reference/Goto)

További szintaxisok

; pontosvessző
Az állítások végén használjuk, lezárjuk vele.

Példa:

```
int a = 13;
```

forrás: http://arduino.cc/en/Reference/SemiColon

{ } kapcsos zárójel
A C programnyelv legfőbb eleme, különböző konstrukciókban használjuk.
Ha van egy nyitó kapcsos zárójel, akkor mindig kell, hogy legyen egy hozzá tartozó záró is. Az Arduino fejlesztőkörnyezetébe be van építve, hogy megmutassa, melyik zárójelek tartoznak össze, amikor az egyiket kijelöljük. Az egyszerűség kedvéért, hogy ne

felejtsük el bezárni a zárójelet, érdemes kitenni egyből mind a kettőt és utána beírni közéjük a szükséges szöveget.

Függvényeknél, ciklusoknál és feltételeknél használjuk.

forrás: *http://arduino.cc/en/Reference/Braces*

// megjegyzés

Két darab perjel mögé beírhatunk magunknak megjegyzéseket a programsorba, melyeket a fordítóprogram figyelmen kívül hagy, nem tölti fel őket az Atmega chip-re.

Ez a kommentezés csak egy sorra vonatkozik. Ha több sort akarunk írni, a csillagos perjeleket használhatjuk.

```
/* ide jön
több sor
megjegyzés */
```

forrás: *http://arduino.cc/en/Reference/Comments*

#define Állandó Neve Érték

Egy állandó értékéhez hozzárendelhetünk egy nevet, mely a könnyebb átláthatóságot segíti. A fordítóprogram ezeket a háttérben behelyettesíti az állandó értékével. (Nincs a végén pontosvessző.)

Példa:

```
#define ledPin 3
```

forrás: http://arduino.cc/en/Reference/Define

#include

Ezzel hívunk meg külső könyvtárakat (library).

Így néz ki a programban:

```
#include <avr/pgmspace.h>
```

forrás: http://arduino.cc/en/Reference/Include

Összehasonlító operátorok

Az if, if-else feltételekkel együtt használjuk őket.

x == y (x egyenlő y-nal);
x != y (x nem egyenlő y-nal);
x < y (x kisebb, mint y);
x > y (x nagyobb, mint y);
x <= y (x kisebb vagy egyenlő y-nal);
x >= y (x nagyobb vagy egyenlő y-nal).

forrás: http://arduino.cc/en/Reference/If

Aritmetikai operátorok

= kijelölő operátor

Az egyenlőségjel jobb oldalán található értéket eltárolja az egyenlőségjel bal oldalán lévő változóban. Kezdőérték megadásánál használjuk. Nem összetévesztendő a matematikai egyenlőség megadásával, ami a dupla egyenlőségjel: ==

Példa:

```
int sensVal;
senVal = analogRead(0);
```

Forrás: http://arduino.cc/en/Reference/Assignment

+ összeadás
- kivonás
*** szorzás**
/ osztás

szintaxisok:
eredmény = érték1 + érték2;
eredmény = érték1 - érték2;
eredmény = érték1 * érték2;
eredmény = érték1 / érték2;

Az érték bármilyen állandó vagy változó lehet: int, float, double.

Példa:

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5.
```

forrás: http://arduino.cc/en/Reference/Arithmetic

% moduló

Amikor egy egész számot elosztunk egy másikkal, megmondja a maradék értékét. Csak egész számmal működik.

Szintaxis: eredmény = osztandó % osztó

Példa:

```
x = 7 % 5;   // x 2-t tartalmaz;
x = 9 % 5;   // x 4-t tartalmaz;
x = 5 % 5;   // x 0-t tartalmaz;
x = 4 % 5;   // x 4-t tartalmaz.
```

forrás: http://arduino.cc/en/Reference/Modulo

Boolean / logikai operátorok

Igazságértéket fejez ki. Lehet **TRUE** (igaz) vagy **FALSE** (hamis). IF feltételen belül használhatjuk őket.

&& (logikai és)

Akkor teljesül, ha mindkét állítás igaz.

Példa:

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) {
 // csináld ezt
}
```

|| (logikai vagy)

Akkor igaz, ha valamelyik állítás igaz.

 © Harsányi Réka, Juhász Márton András

Példa:

```
if (x > 0 || y > 0) {
// csináld ezt
}
```

! (nem)

Akkor igaz, ha az állítás hamis. Tagadás.

Példa:

```
if (!x) {    /
// csináld ezt
}
```

forrás: http://arduino.cc/en/Reference/Boolean

Pointerek

& (objektum címe) és *** (dereferálás)**

C++-ban használt operátorok. A pointer (mutató) típusú változó egy olyan változó, amely egy memóriacímre mutat.

Pointerek: http://hu.wikipedia.org/wiki/C%2B%2B#Pointerek

forrás: http://arduino.cc/en/Reference/Pointer, http://hu.wikipedia.org/wiki/Oper%C3%A1torok_(C%2B%2B)

Bitenkénti operátorok

A változók bit szintjén végzik el a számítást. Arduinoban az integer 16 bites.
Bináris / kettes számrendszer: http://hu.wikipedia.org/wiki/Kettes_sz%C3%A1mrendszer

& (bitenkénti és)

Ha mindkét bemeneti bit értéke 1, akkor a kapott kimeneti eredmény 1 bit. Ha az egyik vagy mindkettő bemeneti bit 0, akkor a kapott kimeneti eredmény 0.

A	B	A&B
0	0	0
1	0	0

0	1	0
1	1	1

Példa:

```
int a =  92;    // binárisan: 0000000001011100
int b = 101;    // binárisan: 0000000001100101
int c = a & b;  // eredmény: 0000000001000100,
                // vagy 68 a tízes számrendszerben.
```

| (bitenkénti megengedő vagy)

Ha legalább az egyik bit 1, akkor az eredmény is 1. Ha mindkettő bemeneti bit 0, akkor a kimeneti bit 0.

A	B	A|B
0	0	0
1	0	1
0	1	1
1	1	1

Példa:

```
int a =  92;    // binárisan: 0000000001011100
int b = 101;    // binárisan: 0000000001100101
int c = a | b;  // eredmény:    0000000001111101,
                // vagy 125 a tízes számrendszerben.
```

^ (bitenkénti kizáró vagy)

Hasonlóan működik, mint a bitenkénti megengedő vagy operátor, azzal a különbséggel, hogy ha mindkét bemeneti bit 1, akkor a kapott eredmény 0.

A	B	A^B

0	0	0
1	0	1
0	1	1
1	1	0

Példa:

```
int x = 12;     // binárisan: 1100
int y = 10;     // binárisan: 1010
int z = x ^ y;  // eredmény: 0110,
                // vagy 6 a tízes számrendszerben.
```

~ (bitenkénti negálás)

Egy bitre alkalmazzuk. Ellenkező értéket vesz fel: 0-ból 1, 1-ből 0 lesz.

A	~A
0	1
1	0

Példa:

```
int a = 103;    // binárisan:  0000000001100111
int b = ~a;     // eredmény:  1111111110011000 = -104
                // tehát -x-1
```

<< (bitenkénti eltolás balra)

A szám bitjeit tolja el balra a megadott értékkel.

1111111111111000

<< 2

1111111111100000

Arduino szintaxis:
változó << bitek száma

>> (bitenkénti eltolás jobbra)

A szám bitjeit tolja el jobbra a megadott értékkel.

0000000000001000

>> 2

0000000000000010

Arduino szintaxis:
változó >> bitek száma

Példa:

```
int a = 5;        // bináris: 0000000000000101
int b = a << 3;   // bináris: 0000000000101000,
                  // vagy 40 tízes számrendszerben
int c = b >> 3;   // bináris: 0000000000000101,
                  // vagy 5-tel vissza, a kezdőértékhez jutunk.
```

forrás: http://arduino.cc/en/Reference/BitwiseAnd, http://www.arduino.cc/playground/Code/BitMath, http://arduino.cc/en/Reference/BitwiseXorNot, http://arduino.cc/en/Reference/Bitshift, http://hu.wikipedia.org/wiki/Oper%C3%A1torok_(C%2B%2B)

Léptető és bitenkénti operátorok

++ (növelés)
- - (csökkentés)

Szintaxis:
x++;
//x értékét növeli eggyel és visszatér az x régi értékéhez.
++x;
//x értékét növeli eggyel és visszatér az x új értékéhez.
x-- ;
//x értékét csökkenti eggyel és visszatér az x régi értékéhez.
--x ;
//x értékét csökkenti eggyel és visszatér az x új értékéhez.
/// x egy integer vagy egy long.

Példa:

```
x = 2; y = ++x;      // x most 3, y 3-t tartalmaz,
y = x--;      // x 2 ismét, y még mindig 3.
```

+= (összeadás és értékadás)
- = (kivonás és értékadás)
***= (szorzás és értékadás)**
/= (osztás és értékadás)

Elvégez egy matematikai műveletet egy változón egy másik változóval vagy állandóval.

Szintaxis:

x += y; // egyenértékű a következő kifejezéssel: x = x + y;
x -= y; // egyenértékű a következő kifejezéssel: x = x - y;
x *= y; // egyenértékű a következő kifejezéssel: x = x * y;
x /= y; // egyenértékű a következő kifejezéssel: x = x / y;
//// x egy változó, y egy állandó vagy egy változó

Példa:

```
x = 2;
x += 4;      // x = 6
x -= 3;      // x = 3
x *= 10;     // x = 30
x /= 2;      // x = 15
```

A **bitenkénti operátorok** úgy kezelik az operandusaikat, mint bitek halmazát, nem úgy, mint decimális, hexadecimális vagy oktális számokat.

&= (bitenkénti ÉS és értékadás)

Akkor tér vissza eggyel, ha mindkét operandusa egyes:

```
0 0 1 1   operandus 1
0 1 0 1   operandus 2
----------
0 0 0 1   (operandus 1 & operandus 2) - visszatérő érték
```

Szintaxis:

x &= y; // egyenértékű ezzel: x = x & y;
// x egy char, int vagy long változó;
// y egy char, int vagy long integer.

Példa:

```
myByte =  10101010;
myByte &= B1111100 == B10101000;
```

|= (bitenkénti megengedő VAGY és értékadás)
Akkor tér vissza eggyel, ha valamelyik operandusa egyes.
0 0 1 1 operandus 1
0 1 0 1 operandus 2
---
0 1 1 1 (operandus 1 | operandus 2) - visszatérő érték

Szintaxis:
x |= y; // egyenértékű ezzel: x = x | y;
// x egy char, int vagy long változó;
// y egy char, int vagy long integer.

Példa:

```
myByte =  B10101010;
myByte |= B00000011 == B10101011;
```

forrás: http://arduino.cc/en/Reference/Increment,
http://arduino.cc/en/Reference/IncrementCompound,
http://arduino.cc/en/Reference/BitwiseCompoundAnd,
http://arduino.cc/en/Reference/BitwiseCompoundOr,
http://hu.wikipedia.org/wiki/C%2B%2B#Az_oper.C3.A1torok_.C3.B6sszefoglal.C3.A1sa

ADAT TÍPUSOK

Állandók

Az Arduino programnyelvben előre definiálhatunk változókat, melyeket csoportonként osztályozunk.

Logikai szint meghatározás, boolean:

Két állandót használunk igazság vagy hamisság kifejezésére:
true | false (kisbetűkkel írva!)

true

Gyakran 1-ként határozzuk meg, ami igaz is, de ennél szélesebb körű a használata. Bármely integer, ami ***nem-nulla,*** az igaz egy boolean kifejezésben. Tehát a –1, 2 vagy –200 is igazként lesz definiálva bennük.

false

Amit 0-ként határoztunk meg.

Pin szint meghatározás:

Amikor egy pinből kiolvasunk vagy egy pinre kiküldünk jelet, akkor annak két állapota lehet: HIGH vagy LOW (nagybetűkkel írva!)

HIGH

A HIGH ebben az esetben összefügg azzal, hogy a pint inputként vagy outputként definiáljuk. Első esetben a mikrokontroller akkor ad HIGH jelet, ha a pinbe minimum 3 volt érkezik. Második esetben, ha az outputként definiált pinre HIGH jelet küldünk, akkor az 5 voltos lesz.

LOW

Inputként definiálva a pint a mikrokontroller akkor kap LOW jelet, ha a beérkező feszültség 2 volt vagy kevesebb. Output pinbe küldött LOW jel 0 voltot jelent.

Digitális pinek meghatározása:

INPUT, OUTPUT, INPUT_PULLUP – a pinek elektromos viselkedését állítjuk be velük a pinMode() parancs segítségével.

INPUT

A pin magas impedanciájú állapotba kerül (pl. szenzorok adatainak kiolvasásához jó digitalRead-del.) A pint sorba kötött ellenállások védik a sérüléstől.

OUTPUT

A pin alacsony impedanciájú állapotba kerül, így jelentős mennyiségű árammal tudja ellátni a rákötött alkatrészt (maximum 40 mA).

INPUT_PULLUP

Az Arduinon az ATMega chipnek hozzáférhető, programozható belső felhúzó ellenállásai vannak. (Egy áramkör bemenete, ami pozitív feszültségre van kötve egy ellenállással a helyes működéshez.) Ha inkább ezeket akarjuk használni a külső ellenállások helyett, akkor kell az INPUT_PULLUP-ot alkalmazni (alapállapotban le vannak választva). Ami ténylegesen megfordítja a pin viselkedését: HIGH-nál a szenzor ki van kapcsolva, LOW-nál a szenzor be van kapcsolva. Az Arduino maximum áramerőssége 40 mA. A duemilanove minden pinje rendelkezik 20–50 kiloohmos felhúzó ellenállással.

Integer állandó:

itn

Integer: előjeles egész számok (tehát negatív is) lehetnek. Alapvetően a 10-es számrendszerben működnek, de egy speciális jelöléssel átalakíthatóak 2-es, 8-as vagy 16-os számrendszerbe.

ALAP	PÉLDA	FORMÁZÓ
10 (tízes)	123	NINCS

///csak 8 bites értékkel működik (0–255 között)

2 (kettes)	B1111011	„B” legelöl

///0 és 1 karakterekkel működik

8 (nyolcas)	0173	"0" legelöl

///érvényes karakterek 0–7 között

16 (tizenhatos)	0x7B	"0x" legelöl

///érvényes karakterek 0–9, A–F, a–f

Az integer után írt megfelelő betűvel, úgynevezett formázókarakterrel átalakítható egyéb adattípussá:

U = unsigned

L = long

UL = unsigned long

 © Harsányi Réka, Juhász Márton András

Floating point állandó:

foat

Lebegőpontos érték, tehát a tizedespont a számjegyeken belül bárhová átkerülhet, így szélesebb tartományban képes felvenni értéket, mint a fixpontos számok. Normalizált formában is megadhatjuk ezeket a számokat, ami azt jelenti, hogy a tizedesvesszőt előre toljuk, hogy 0-val kezdődjön a szám, és megszorozzuk a 10 megfelelő hatványával; pl. $135.7896 = 0.1357896 \times 10^3$.

Ezek a számok rögzített számjegyekkel rendelkeznek és egy kitevő (exponens) segítségével lehet skálázni őket, aminek jele „e” vagy „E”, tehát azt mutatja meg, hogy a 10-nek hányadik hatványával kell beszorozni az értéket: ***értékes számjegy* × *alapexponens***

2.34E5 2.34 * 10^5 234000 67e-12 67.0 * 10^-12 .000000000067

Az előző példában a három oszlopban lévő érték soronként megegyezik.

Forrás:
http://arduino.cc/en/Reference/Constants, http://arduino.cc/en/Reference/IntegerConstants
http://arduino.cc/en/Reference/Fpconstants, http://hu.wikipedia.org/wiki/C%2B%2B
http://hu.wikipedia.org/wiki/Lebeg%C5%91pontos_sz%C3%A1m%C3%A1br%C3%A1zol%C3%A1s

Adattípusok

void()

A függvények deklarálásához használt kulcsszó. Saját függvényt írhatunk vele, amit csak meghívunk egy másik helyen a kódban.

Forrás: http://arduino.cc/en/Reference/Void

boolean

Két értéke lehet: igaz vagy hamis. Minden boolean változó egy byte-ot foglal el a memóriában.

Példa:
boolean running = false.

Forrás: http://arduino.cc/en/Reference/BooleanVariables

char

Ez az adattípus egy byte memóriát foglal el, karakterérték tárolására alkalmas, tehát 1 karakter értékét tárolja. A betűket szimpla idézőjelbe, a karakter stringeket dupla idézőjelbe írjuk: 'A' „ABC” típus-átalakításhoz használatos kulcsszó. Előjeles adattípus, mely azt jelenti, hogy –128 és 127 közötti értékeket tárolhat.

A karaktereket számként tároljuk, a betűk decimális értéke visszakereshető az ASCII karakterszabványban: http://arduino.cc/en/Reference/ASCIIchart De pl. Serial.println parancs segítségével kiírattathatjuk a többi számrendszerben is serial monitor segítségével, így megnézhetjük azok értékeit is.

Példa:

```
char myChar = 'A';
char myChar = 65; //a két sor ugyanazt jelenti.
```

Forrás: http://arduino.cc/en/Reference/Char

unsigned char

Ugyanaz, mint a *byte*. 0 és 255 közötti számokat lehet belekódolni. A következetesség kedvéért használjuk a *byte*-ot.

Példa:

```
unsigned char myChar = 240.
```

Forrás: http://arduino.cc/en/Reference/UnsignedChar

byte

8 biten (= 1 byte) tárol nem előjeles számokat, 0 és 255 között.

Példa:

```
byte b = B10010;  // „B” bináris formázó (B10010 = 18 tízes számrendszerben)
```

Forrás: http://arduino.cc/en/Reference/Byte

 © Harsányi Réka, Juhász Márton András

int

Elsődleges adattípus számok tárolásához. Egész szám értéket tárol. Az Arduino Unon az integer 16 bit értéket tárol –32768 és 32767 között. Az Arduino Due 32 biten működik, így 2147483648 és 2147483647 közötti értéket tárol. Aritmetikai operátorokkal a hagyományos módon működnek. Bitenkénti operátorokat is használhatunk velük, viszont ilyenkor figyelni kell, mert azok kettes számrendszerben tolják arrébb a bitértékeket.

Szintaxis:
int var = val; // integer változóneve = értékmegadás

- **var:** az általunk elnevezett integer változó neve,
- **val:** az érték, amit ehhez a változóhoz hozzárendelünk,

speciális esete, ha folyamatosan növekvő vagy csökkenő változót akarunk:
(int x; x= x+1; vagy x= x-1;).

Példa:

```
int a = 5;         // binary: 0000000000000101
int b = a << 14;   // binary: 0100000000000000
```

forrás: http://arduino.cc/en/Reference/Int

unsigned int

Előjel nélküli egész számok tárolására alkalmas. Mivel ugyanúgy 2 byte-on működik, mint az integer, nagyobb pozitív értéket tud tárolni 0 és 65535 között az ATMega alapú Arduinoknál. Due típuson 4 byte-tel működik, így 0 és 4294967295 között használható.

Szintaxis:
unsigned int var = val; // előjeles integer változóneve = értékmegadás.

Ha a változó eléri az egyik szélsőértéket, akkor átugrik a másik végére és ott folytatódik a számítás, tehát:

```
unsigned int x
x = 0;
x = x - 1;        // x most 65535 – negatív irányba ugrott érték;
x = x + 1;        // x most 0 – pozitív irányba ugrott érték.
```

Forrás: http://arduino.cc/en/Reference/UnsignedInt

word

Ugyanaz, mint az unsigned int, tehát 16 biten tárol előjel nélkül számokat, 0-tól 65535-ig.

Példa:

```
word w = 10000;
```

Forrás: http://arduino.cc/en/Reference/Word

long

32 biten tárol számértéket -2 147 483 648 és 2 147 483 647 között.

Példa:

```
long speedOfLight = 186000L;      // 'L' kifejezést lásd az Integer Állandóknál.
```

Szintaxis:

long var = val; // long változó Neve = Érték

Forrás: http://arduino.cc/en/Reference/Long

unsigned long

Ugyanúgy 32 biten tárol számértéket, mint a long, de nincs előjele, így 0-tól 4294967295 (2^32 – 1)-ig terjedhet az értéke.

Példa:

unsigned long time;

```
void setup() {
   Serial.begin(9600);
}

void loop() {
   Serial.print(„Time:”);
   time = millis();   //prints time since program started
```

```
    Serial.println(time);     // wait a second so as not to send massive
    amounts of data   delay(1000);
}
```

Szintaxis:
unsigned long var = val; // unsigned long változó Neve = Érték

Forrás: http://arduino.cc/en/Reference/UnsignedLong

short

16 bites adattípus ATMega és ARM alapú Arduinokhoz. Tárolt érték –32768 és 32767 között.

Példa:
```
short ledPin = 13;
```

Szintaxis:
short var = val; // short változó Neve = Érték

Forrás: http://arduino.cc/en/Reference/Short

float

Tizedesponttal rendelkező számok. Lebegőpontos érték tárolására alkalmasak, 32 biten. Maximális értéke 3.4028235E+38 és minimális értéke –3.4028235E+38.

A float kifejezésben csak 6–7 tizedes számjegy van. Ez az összes számjegyre értendő, nem a tizedespont jobb oldalára.

A lebegőpontos számok nem precízek, összehasonlításnál furcsa eredményt adhatnak. Pl. 6.0 / 3.0 nem egyenlő 2.0-val, mert a számok abszolút értékének különbsége kevesebb lehet néhány kicsi számnál.

Példák:
```
float myfloat;
float sensorCalbrate = 1.117;
```

Szintaxis:
float var = val; // float változó Neve = Érték

Forrás: http://arduino.cc/en/Reference/Float

double

Lebegőpontos számérték. Unon és ATMega alapú boardokon 4 byte-os,
Due-n 8byte-os. Ugyanaz, mint a float.

Forrás: http://arduino.cc/en/Reference/Double

string – karaktertömb

Kétféleképpen ábrázolható, egyrészt használhatjuk, mint a 0019 verzióba beépített String adattípust, másrészt létrehozhatunk stringet egy karakterlánc tömbbel.

Ezeket a következő példák szerint lehet deklarálni, mindegyik érvényes:
Inicializáló érték megadása nélkül:
char Str1[15];

Karakterenként: A karaktertömbök utolsó eleme a 0 (nulla), innen ismeri meg a fordítóprogram a string végét.

```
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
```

Karakterstring idézőjelek között, szövegként értelmezve:

```
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Forrás: http://arduino.cc/en/Reference/String

String - object

Szöveg alapú stringeket manipulálhatunk a következő függvények által:

charAt()

Visszaadja a String típusú változó n-edik karakterét.

Szintaxisa:
string.charAt(n)

 © Harsányi Réka, Juhász Márton András

Paraméterei:

- **string**: az a String típusú változó, amiben
- **n**: az n-edik karaktert szeretnénk elérni.

setCharAt()

Lecserél egy adott karaktert a String változón belül. Csak a String hosszán belül érvényes.

Szintaxisa:
setCharAt(index,c)

Paraméterei:

- **string**: String típusú változó neve,
- **index**: a lecserélendő karakter sorszáma,
- **c**: a lecserélendő karakter – char típusú változó.

compareTo()

Két Stringet hasonlít össze, vizsgálja az egyenlőségüket – egyenlőtlenség esetén pedig az ABC sorrendben betöltött sorrendjüket.

Szintaxisa:
string.compareTo(string2)

Paraméterei:

- **string**: az egyik összehasonlítandó String,
- **string2**: a másik összehasonlítandó String.

Kimenete:
Negatív érték: string előrébb való string2-nél,
0: a két String megegyezik,
Pozitív érték: string2 előrébb való stringnél.

concat()

Összekapcsol két stringet, a másodikat folytatólagosan az elsőhöz ragasztja és a produktumot egy új stringbe tölti. Szám típusú változókat is hozzáadhatunk vele egy szöveghez.

Szintaxisa:
string.concat(string,string2),
string.concat(string2),
string += string2.

Paraméterei:

- **string**: első változó,
- **string2**: második változó.

startsWith()

Megvizsgálja, hogy egy string azzal a karakterlánccal kezdődik-e, mint amit egy másik tartalmaz, és igaz/hamis választ ad vissza.

Szintaxisa:
string.startsWith(string2).

Paraméterei:

- **string**: string, aminek az elején keresünk,
- **string2**: string, amit keresünk.

endsWith()

Megvizsgálja, hogy egy string azzal a karakterlánccal végződik-e, mint amit egy másik tartalmaz, és igaz/hamis választ ad vissza.

Szintaxisa:
string.endsWith(string2).

Paraméterei:

- **string**: string, aminek az elején keresünk,
- **string2**: string, amit keresünk.

indexOf() és **lastIndexOf()**

Szövegen belüli keresést hajthatunk végre a segítségükkel, a keresendő minta lehet karakter (char) vagy string típusú is. Alapértelmezésként az indexOf() a string elejétől kezdi a keresést, de megadhatunk kezdő pozíciót is. A lastIndexOf() abban különbözik, hogy képes a string végéről visszafelé keresni. Mindkét függvény a találat tömbön belüli indexét adja vissza és –1-et, ha nem talál egyezést.

Szintaxisuk:
string.indexOf(val),
string.indexOf(val, from),
string.lastIndexOf(val),
string.lastIndexOf(val, from).

Paramétereik:

- **string**: amiben keresünk,
- **val**: amit keresünk, lehet *char* vagy *String*,
- **from**: a karakter pozíciója a tömbben, ahonnan a keresés indul.

length()
Visszaadja a string karaktereinek számát a lezáró *null* karakter nélkül.

Szintaxisa:
string.length().

Paramétere:
string: aminek a hosszára kíváncsiak vagyunk.

trim()
Levágja a szöveget megelőző vagy követő üres helyeket és az eredményt visszatölti az eredeti változóba. A szünetnek látszó üres karakterek például: SPACE (ASCII 32), tab (ASCII 9), függőleges tab (ASCII 11), sor elejére ugrás (ASCII 13), új sor (ASCII 11).

Szintaxisa:
string.trim().

Paramétere:

- **string**: amit szeretnénk megváltoztatni.

substring()
Egy string tetszőleges részletét adja vissza, amit a tömb indexeivel határolhatunk. A kezdő pozíción található karakter még bele fog esni a kimásolt karakterláncba, míg a végpozíción található karakter már nem. Ha nem adunk meg végpozíciót, akkor az automatikusan a karakterlánc vége lesz.

Szintaxisa:

string.substring(from),
string.substring(from, to).

Paraméterei:

- **string**: amiből karakterláncot másolunk,
- **from**: a tömb indexe, ami a kezdőpozíciót jelöli,
- **to**: a tömb indexe, ami a végpozíciót jelöli.

replace()

Kicserél két karakterláncot egy stringen belül, de csak akkor, ha amire cserélünk, nem hosszabb, mint az egész string. Máskülönben a fordító hiba nélkül továbblép és az eredeti string változatlan marad.

Szintaxisa:
string.replace(substring1, substring2).

Paraméterei:

- **string**: a karakterlánc, aminek egy részét cseréljük,
- **substring1**: amit ki szeretnénk cserélni,
- **substring2**: amire ki szeretnénk cserélni.

toLowerCase()

Átalakítja egy stringben a nagybetűket kisbetűkké és visszatölti az eredményt az eredeti változóba. Ezzel szemben a *toUppercase()* a kisbetűket alakítja nagybetűkké.

Szintaxisa:
string.toLowerCase(),
string.toUpperCase().

Paramétere:

- **string**: amin a változtatást végrehajtjuk.

getBytes()

Előfordulhat, hogy szeretnénk a string típusát átalakítani. Ha byte-okat tartalmazó tömbként szeretnénk a karakterláncot kezelni, akkor ez a függvény kimásolja a string tartalmát byte-onként egy tömbbe.

Szintaxisa:

string.getBytes(buf, len).

Paraméterei:

- **string**: string változónk, aminek a tartalmát másoljuk,
- **buf**: byte[] (byte tömb) típusú változónk, ahová másolunk,
- **len**: a byte tömbünk hossza (unsigned int típusú).

toCharArray()

Hasonlóan az előzőhöz, kimásolja a string tartalmát, és egyszerű karaktereket tartalmazó tömbbe tölti.

Szintaxisa:
string.toCharArray(buf, len).

Paraméterei:

- **string**: string változónk, aminek a tartalmát másoljuk,
- **buf**: char[] (karakter tömb) típusú változónk, ahová másolunk,
- **len**: a karakterlánc hossza (unsigned int típusú).

Forrás: http://arduino.cc/en/Reference/StringObject

array

Tömb. Adathalmazt tárolhatunk memóriában és műveleteket végezhetünk el rajtuk. A tömbök elemeire mindig az indexelés operátorával ([]) hivatkozunk (pl. x[3]).

Szintaxisa:
típus tömbnév[elemszám].

Tömb deklarálása a következő módokon történhet:

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

Érték hozzárendelése a tömb egy eleméhez:

```
mySensVals[0] = 10;
```

Érték kiolvasása a tömb egy eleméből:

```
x = mySensVals[4];
```

Forrás: http://arduino.cc/en/Reference/Array

Típusmódosítók

char(x)

Az értéket char adattípussá alakítja.

Paramétere:

- **x:** bármilyen típusú érték lehet.

Kimenete: Char adattípus.

Forrás: http://arduino.cc/en/Reference/CharCast

byte(x)

Az értéket byte adattípussá alakítja.

Paramétere:

- **x:** bármilyen típusú érték lehet.

Kimenete: Byte adattípus.

Forrás: http://arduino.cc/en/Reference/ByteCast

int(x)

Az értéket int adattípussá alakítja.

Paramétere:

- **x:** bármilyen típusú érték lehet.

Kimenete: Int adattípus.

Forrás: http://arduino.cc/en/Reference/IntCast

word(x)

word(h,l)
Az értéket vagy két byte-ot word adattípussá alakítja.

Paraméterei:

- **x:** bármilyen típusú érték lehet;
- **h:** a word magas sorrendű byte-ja, balról az első;
- **l:** a word alacsony sorrendű byte-ja, jobbról az első.

Kimenete: Word adattípus.

Forrás: http://arduino.cc/en/Reference/WordCast

long(x)
Az értéket long adattípussá alakítja.

Paramétere:

- **x:** bármilyen típusú érték lehet.

Kimenete: Long adattípus.

Forrás: http://arduino.cc/en/Reference/LongCast

float(x)
Az értéket float adattípussá alakítja.

Paramétere:

- **x:** bármilyen típusú érték lehet.

Kimenete: Float adattípus.

Forrás: http://arduino.cc/en/Reference/FloatCast

Típusminősítők

Változók hatóköre

Az úgynevezett globális változók, amiket a setup() függvényen kívül, azt megelőzően deklarálunk, bárhol meghívhatóak a kódban. De létrehozhatunk úgynevezett helyi változókat függvényeken belül is, ilyenkor csak ezen belül tudjuk használni őket, ilyen

például a for ciklusban használatos i változó is. Hosszabb kódokban a helyi változók segítenek elkerülni a hibákat.

Forrás: http://arduino.cc/en/Reference/Scope

static

A static parancsszóval olyan változókat hozhatunk létre, amik csak egy függvény számára láthatóak. Ellentétben a helyi változókkal, amiket a függvényen belül hozunk létre, és azon kívül nem tudjuk használni, a statikus változók az adott függvényen kívül is meghívhatóak és megtartják a bennük tárolt értéket. A statikusként deklarált változókat csak egyszer kell, a meghívott függvény elején létrehozni és inicializálni (példa: static int place;).

Forrás: http://arduino.cc/en/Reference/Static

volatile

A változó minősítésére szolgáló kulcsszó. A változó adattípusát meghatározó kulcsszó elé kell írni (példa: volatile int state = LOW;). Módosítja azt, hogyan kezelje a fordítóprogram és a programkód a változót. A fordítóprogram ilyenkor a RAM-ból tölti be a változót és nem az ideiglenesen használható tároló regiszterből (ami bizonyos körülmények között pontatlan értéket is adhat). Akkor érdemes használni, ha az értékét az ezt tartalmazó kódrészen kívül valami más változtatja meg, mint pl. egy egyidejűleg végrehajtott parancs. Arduinoban az egyetlen hely, ahol ez valószínűleg előfordul, az úgynevezett megszakítás-kiszolgáló rutinban van (attachInterrupt();).

Forrás: http://arduino.cc/en/Reference/Volatile

const

A const a változók minősítésére szolgáló parancsszó, ami által a változó a későbbiekben nem módosítható, hanem csak olvasható. Hibaüzenetet kapunk, ha később megpróbálunk hozzárendelni egy másik értéket, tehát a változók így állandóvá válnak (példa: const float pi = 3.14;).

Forrás: http://arduino.cc/en/Reference/Const

 © Harsányi Réka, Juhász Márton András

Alaptípus

sizeof(variable**)**

Bármilyen típusú változó vagy tömb lehet a paramétere. Ez az operátor a bájtok számát vagy egy tömbben szereplő bájtok értékét adja vissza az adott változótípusban.

Forrás: http://arduino.cc/en/Reference/Sizeof

FÜGGVÉNYEK

Digitális be-/kimenetek

pinMode(pin, mode):

Itt adjuk meg, hogy melyik pint mire használjuk.

Paraméterei:

- **pin:** az általunk használt pin száma, digitálisokat számmal 0, 1, 2 stb.), az analógokat az A betű mellé írott számmal (A0, A1 stb.) jelöljük;
- **mode:** bemenetként vagy kimenetként akarjuk használni: INPUT, OUTPUT.

Forrás: http://arduino.cc/en/Reference/PinMode

digitalWrite(pin, value):

Kimenetként használt digitális pint tudjuk vele be- és kikapcsolni.

Paraméterei:

- **pin:** az általunk használt pin száma, amit a pinMode-nál is megadtunk;
- **value:** értéke lehet ki- (0 V) vagy bekapcsolt (5 V): LOW, HIGH.

Forrás: http://arduino.cc/en/Reference/DigitalWrite

digitalRead(pin);

Kiolvassa az értéket az adott pinből.

Paramétere:

- **pin:** annak a pinnek a száma, amiből ki akarjuk olvasni az adatot, integer = egész szám. A kapott érték lehet HIGH (1, be) vagy LOW (0, ki).

Forrás: http://arduino.cc/en/Reference/DigitalRead

Analóg be-/kimenetek

analogReference(type) ;

Az analóg input által használt referencia-feszültséget állítja be. Az átállítást követően az első AnalogRead() kiolvasás nem lesz pontos!

Típusai (*type*) a következőek lehetnek:

- **DEFAULT:** az alapbeállítás az Arduino típusától függően 5 V vagy 3,3 V lehet.
- **INTERNAL:** a beépített referenciaérték, Atmega168 vagy 328 chip esetén 1,1 V, Atmega8-nál 2,56 V (Megán nem elérhető).
- **INTERNAL1V1:** csak Megán, a beépített 1,1 V referencia.
- **INTERNAL2V56:** csak Megán, a beépített 2,56 V referencia.
- **EXTERNAL:** az AREF pin feszültsége (0–5 V), amit referenciaértéknek használhatunk. NE HASZNÁLJUNK 0 VOLTNÁL KISEBB VAGY 5 VOLTNÁL NAGYOBB KÜLSŐ FESZÜLTSÉGFORRÁST! Az AnalogRead() meghívása előtt ezt állítsuk be!

Forrás: http://arduino.cc/en/Reference/AnalogReference

analogRead(pin);

Ezzel a paranccsal tudjuk kiolvasni az analóg pinbe érkező értékeket. Egy 0–1023 közötti értéket ad vissza.

Paramétere:

- **pin:** annak az analóg pinnek a száma, amiből ki akarunk olvasni.

Forrás: http://arduino.cc/en/Reference/AnalogRead

 © Harsányi Réka, Juhász Márton András

analogWrite(pin, value);

PWM pinre küldhetünk vele értéket, hogy analóg tulajdonsággal ruházzuk fel a digitális pint.

Paraméterei:

- **pin:** amelyik pwm pint kimenetként használjuk,
- **value:** értéke 0–255 között.

Forrás: http://arduino.cc/en/Reference/AnalogWrite

Haladó szintű be-/kimenetek

tone(pin, frequency, duration)

Előre megadott frekvenciájú hullámot generál az adott pinen, így egy hang szólal meg a hangszórónkon vagy a piezo buzzeren. A hullám időtartamát is meg kell határozni, különben addig szól a hang, amíg a noTone() függvény meghívásával meg nem szakítjuk.

Paraméterei:

- **pin:** ahová bekötöttük a hangszórót vagy a piezo berregőt;
- **frequency**: a hangjegy frekvenciája Hz-ben megadva (unsigned int);
- **duration:** a hangjegy hossza milliszekundumban megadva (unsigned long).

Forrás: http://arduino.cc/en/Reference/Tone

noTone(pin)

Megállítja a tone() függvény által generált hangot. Amennyiben több pinen akarunk lejátszani dallamot, meg kell hívni a noTone()-t, mielőtt a másik pinen meghívjuk a tone()-t.

Paramétere:

- **pin:** ugyanaz a pin, amelyikhez a tone()-t meghívtuk.

Forrás: http://arduino.cc/en/Reference/NoTone

shiftOut(dataPin, clockPin, bitOrder, value)

Egyenként kitolja a bájtban szereplő bit adatokat. Indulhat mindkét oldalról, az elejéről vagy a végéről is. A bitek után a bemeneti pinen egy órajel adat (be- és kikapcsol) jelzi, hogy a következő jön. A parancs meghívása előtt az órajelet ki kell kapcsolni: digitalWrite(clockPin, LOW).

Paraméterei:

- **dataPin:** a bitek kimenetének szánt pin (integer);
- **clockPin:** kapcsoló pin, amikor a dataPin értéke megfelelő, be- és kikapcsol;
- **bitOrder:** a kitolás indulásának helyét adja meg: MSBFIRST vagy LSBFIRST paranccsal működik. (Most Significant Bit First = legmagasabb helyi értékű bit először, Least Significant Bit First = legalacsonyabb helyi értékű bit először);
- **value:** az adat, amit kitol (byte).

Az adat pint és az órajel pint pinMode()-dal be kell állítani kimenetként! 1 byte-tal, azaz 8 bittel működik.

Forrás: http://arduino.cc/en/Reference/ShiftOut

shiftIn()

Szintaxis: byte incoming = shiftIn(dataPin, clockPin, bitOrder)

Egyesével behúzza a bájtban szereplő bit adatokat, kezdődhet a legmagasabb vagy a legalacsonyabb helyi értékű bitnél is. Minden egyes bit előtt az órajel pin bekapcsol, az adat pin kiolvassa a soron következő bitet, majd az órajel pin kikapcsol.

Paraméterei:

- **dataPin:** a bitek kimenetének szánt pin (integer);
- **clockPin:** kapcsoló pin, amikor a dataPin értéke megfelelő, be- és kikapcsol;
- **bitOrder:** az eltolás indulásának helyét adja meg: MSBFIRST vagy LSBFIRST paranccsal működik. (Most Significant Bit First = legmagasabb helyi értékű bit először, Least Significant Bit First = legalacsonyabb helyi értékű bit először).

Visszatér: a kiolvasott érték (byte).

Forrás: http://arduino.cc/en/Reference/ShiftIn

pulseIn (pin, value, timeout)

Kiolvassa az impulzust az adott pinen, mely lehet HIGH vagy LOW. Például, ha az értéke HIGH, akkor a pulseIn() megvárja, míg a pin értéke is HIGH lesz és elkezd számolni. Ez addig tart, míg a pin LOW értéket kap, ekkor abbahagyja a számolást. Az impulzus hosszának értékét mikroszekundumban kapjuk vissza. Ha a meghatározott idő alatt (timeout) nem érkezik jel, akkor 0-t kapunk vissza.

Paraméterei:

- **pin:** amelyik pinen kiolvassuk az impulzust (integer);
- **value:** az impulzus típusa, HIGH vagy LOW (integer);
- **timeout:** időtúllépés (opcionális), ha nincs megadva, akkor alapból 1 másodperc, vagy mikroszekundumban kell megadni, várakozási időtartam az impulzus kezdete előtt (unsigned long).

Tehát visszatérhet egy mikroszekundum értéket takaró unsigned long, vagy pedig 0.

Forrás: http://arduino.cc/en/Reference/PulseIn

Idő

millis();

Milliszekundumban adja vissza a program futtatása óta eltelt értéket, kb. 50 nap után a számérték túlcsordul és visszaugrik nullára. Paramétere nincs, unsigned long változót ad vissza.

Forrás: http://arduino.cc/en/Reference/Millis

micros();

A programfuttatás megkezdésétől eltelt időt adja vissza mikroszekundumban. Kb. 70 perc után túlfordul és 0-ról kezdi újra a számolást. Paramétere nincs. Unsigned long változót ad vissza.

Forrás: http://arduino.cc/en/Reference/Micros

delay(ms);

Szünetet tart a futtatásban. Visszatérő értéke nincs.

Paramétere:

- **ms:** millisecundumban kell megadni az értékét, 1000 ms = 1 s (másodperc).

Forrás: http://arduino.cc/en/Reference/Delay

delayMicroseconds(us)

Szünetet tart a program futtatásában a paraméterként *(us)* mikroszekundumban megadott ideig. Visszatérő értéke nincs.

Forrás: http://arduino.cc/en/Reference/DelayMicroseconds

Matematika

min(x,y)

Kiszámítja a paraméterként megadott, bármilyen adattípusú számok *(x és y)* minimumát. A két szám közül a kisebbik érték tér vissza.

Forrás: http://arduino.cc/en/Reference/Min

max(x,y)

Kiszámítja a paraméterként megadott, bármilyen adattípusú számok *(x és y)* maximumát. A két szám közül a nagyobbik érték tér vissza.

Forrás: http://arduino.cc/en/Reference/Max

abs(x)

Kiszámítja egy paraméterként megadott szám *(x)* abszolútértékét. Ha a szám nagyobb vagy egyenlő nullával, akkor a szám értéke tér vissza, ha kisebb, mint nulla, akkor a negatív érték tér vissza. Hiba elkerülése végett paraméterként csak értéket adjunk meg (példa: abs(a);) és egyéb parancsokat ne (példa: abs(a++);)!

Forrás: http://arduino.cc/en/Reference/Abs

constrain(x,a,b)

Egy adott tartományon belülire kényszeríti az értéket, azaz limitálja. Bármely adattípuson alkalmazható.

Paraméterei:

- **x:** a szám, amit limitálni akarunk, ha a és b közé esik, akkor megtartja az értékét;
- **a:** alsó limit, ha x ez alá esik, átugrik az értéke a-ra;
- **b:** felső limit, ha x e felé megy, átugrik az értéke b-re.

Forrás: http://arduino.cc/en/Reference/Constrain

map(value, fromLow, fromHigh, toLow, toHigh)

Arányosan szétosztja a számokat egyik tartományról a másikra Az alsó és a felső értékeket rendeli egymáshoz, a többit ehhez igazítja. Ez az utasítás egész számokkal működik.

Paraméterei:

- **value:** az érték, amit arányosítani akarunk (gyakran a változónk neve);
- **fromLow:** az aktuális értékünk legalsó határa;
- **fromHigh:** az aktuális értékünk legfelső határa;
- **toLow:** a célérték legalsó határa;
- **toHigh:** a célérték legfelső határa.

Forrás: http://arduino.cc/en/Reference/Map

pow(base, exponent)

Egy megadott szám hatványértékét számolja ki. Értékek vagy görbék exponenciális leképezésénél hasznos.

Paraméterei:

- **base:** a szám (float);
- **exponent:** kitevő, a hatvány, amivel megnöveljük az alap számot (float).

Forrás: http://arduino.cc/en/Reference/Pow

sqrt(x)

Egy paraméterként megadott bármilyen típusú szám *(x)* négyzetgyökét adja vissza double típusban.

Forrás: http://arduino.cc/en/Reference/Sqrt

Trigonometria

sin(rad)

Egy radiánsban megadott szög szinuszát számolja ki. Az eredmény egy –1 és 1 közötti érték, ami double típusban tér vissza.

Paramétere:

- **rad:** a radiánsban megadott szög (float).

Forrás: http://arduino.cc/en/Reference/Sin

cos(rad)

Egy radiánsban megadott szög koszinuszát számolja ki. Az eredmény egy –1 és 1 közötti érték, ami double típusban tér vissza.

Paramétere:

- **rad:** a radiánsban megadott szög (float).

Forrás: http://arduino.cc/en/Reference/Cos

tan(rad)

Egy radiánsban megadott szög tangensét számolja ki. Az eredmény egy a negatív végtelen és a végtelen közötti érték, ami double típusban tér vissza.

Paramétere:

- **rad:** a radiánsban megadott szög (float).

Forrás: http://arduino.cc/en/Reference/Tan

Random számok

randomSeed(seed)

Ezzel a paranccsal egy pszeudo-random szám generátort inicializálunk. Paramétere egy long vagy egy integer lehet. Nincs visszatérő értéke.

Forrás: http://arduino.cc/en/Reference/RandomSeed

random(max)
random(min,max)

Pszeudo random számokat generál. Egy random szám értéket ad vissza a két szélsőérték között long formában.

Paraméterei:

- **min:** a random érték alsó határa, nem kötelező megadni – ilyenkor 0;
- **max:** a random érték felső határa (ami már nem tér vissza, csak az eggyel kisebb szám).

Forrás: http://arduino.cc/en/Reference/Random

Bitek és byte-ok

lowByte(x)

Kihúzza a legalacsonyabb helyi értékű (jobbról az első) bájtot a változóból. Visszatérő értéke a bájt.

Paramétere:

- **x:** bármilyen típusú érték.

Forrás: http://arduino.cc/en/Reference/LowByte

highByte(x)

Kihúzza a legmagasabb helyi értékű (balról az első) bájtot a wordből (vagy nagyobb adattípus esetén a második legalacsonyabbat). Visszatérő értéke a bájt.

Paramétere:

- **x:** bármilyen típusú érték.

Forrás: http://arduino.cc/en/Reference/HighByte

bitRead(x,n)
Egy szám bitjeit olvassa ki. Ez lesz a visszatérő értéke, ami lehet 0 vagy 1.

Paramáterei:

- **x:** a szám, amiből kiolvasunk;
- **n:** amelyik bitet ki akarjuk olvasni, jobbról és 0-tól kell számolni.

Forrás: http://arduino.cc/en/Reference/BitRead

bitWrite(x,n,b)
Átír egy bitet egy szám változóban. Visszatérő értéke nincs.

Paraméterei:

- x: a szám változó, amit át akarunk írni;
- **n:** amelyik bájtot ki akarjuk cserélni, jobbról és 0-tól kell számolni;
- **b:** az érték, amivel átírjuk a bitet (0 vagy 1 lehet).

Forrás: http://arduino.cc/en/Reference/BitWrite

bitSet(x,n)
Egy számváltozóban beállít egy bitet, azaz átírja 1-re. Visszatérő értéke nincs.

Paraméterei:

- **x:** a változó, amiben átírjuk a bitet;
- **n:** ahányadik bitet átírjuk 1-re (jobbról és 0-tól kell számolni).

Forrás: http://arduino.cc/en/Reference/BitSet

bitClear(x,n)
Egy számváltozóban kitöröl egy bitet, azaz átírja 0-ra. Visszatérő értéke nincs.

Paraméterei:

- **x:** a változó, amiben átírjuk a bitet;
- **n:** ahányadik bitet átírjuk 1-re (jobbról és 0-tól kell számolni).

Forrás: http://arduino.cc/en/Reference/BitClear

bit(n)

Kiszámítja a megadott bit értékét (a 0. bit lesz az első, az 1. bit lesz a második stb.). Visszatérő értéke a bit értéke.

Paramétere:

- **n:** amelyik bitnek az értékét ki akarjuk számolni.

Forrás: http://arduino.cc/en/Reference/Bit

Soros port kommunikáció

Serial.begin(9600);
Serial.println(val, format);

Az elsővel bekapcsoljuk a soros kommunikációt 9600 bit/sec-on, a másodikkal kiíratjuk a kapott adatokat a serial monitoron.

Paraméterei:

- **val:** az érték, amit ki akarunk íratni – bármilyen adat lehet;
- **format:** elhagyható, ilyenkor decimális számrendszert használ (ha oda akarjuk írni: DEC a tízes-, HEX a tizenhatos-, OCT a nyolcas-, BIN a kettes számrendszer jele).

A könyvtárhoz tartozó további szintaxisok részletesen a 13. fejezetben találhatóak.

Forrás: http://arduino.cc/en/Reference/Serial