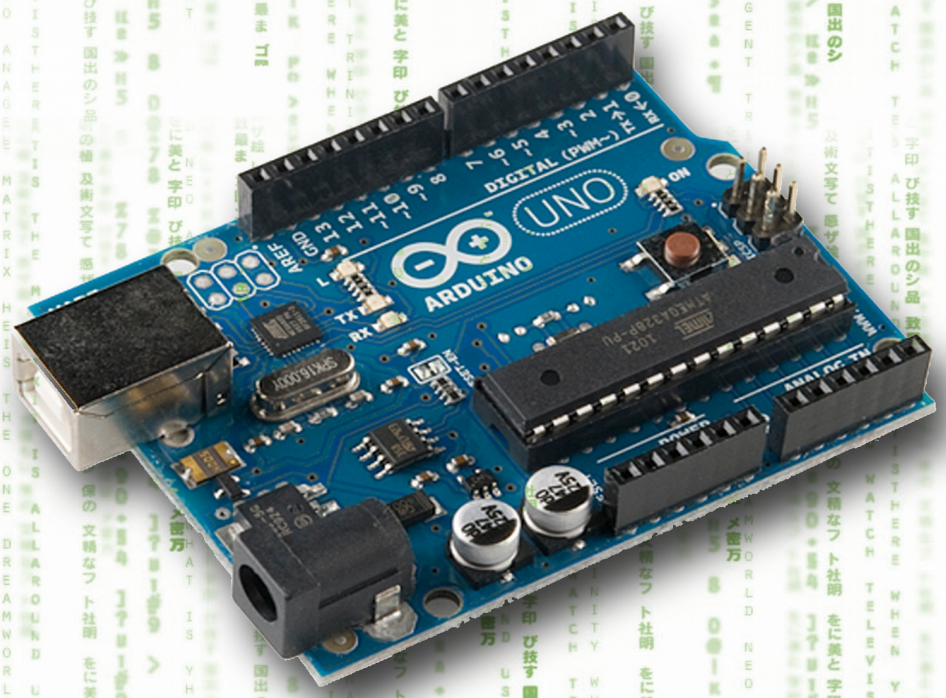
Mikrovezérlős Rendszerfejlesztés C/C++ nyelven II.

Arduino Platform



Írta: Ruzsinszki Gábor

v: 2013-08-15

ISBN: 978-963-08-7261-4

Tartalom

0. Dokumentum információk

Írta: Ruzsinszki Gábor. ISBN: 978-963-08-7261-4

Licenc

A könyv digitális változata, CD melléklete a http://webmaster442.hu címen szerezhető be.

Változások listája

2013-04-05 változathoz képest:

- *MCU Tools programot bemutató fejezet*
- *Fekvő melléklet oldalakon az élőláb szövege javítva*
- *Új projektek:*
  - *RGB Led-ek kezelése*
  - *Több, egyidejűleg futtatott feladatkezelést bemutató projekt*
  - *Firmata kommunikációs protokoll*
  - *Szoftveres pergésmentesítés*
  - *Nyúlásmérő bélyegek alkalmazása*
  - *Fejlesztés Visual/Atmel Studio-val*
  - *Arduino készítése házilag*
- *Arduino Uno, Mega, Leonardo lábkiosztás hozzáadva a mellékletekhez.*
- *Hálózatkezelési alapismeretek kibővítve az ENC26J60 típusú vezérlő leírásával, valamint mellékletekben ENC26J60 modul kapcsolási rajz*
- *A mű kapott ISBN számot.*
- *Tartalomjegyzék stílusa módosítva*

1. Bevezetés

Az Arduino platform megjelenésével forradalmasította a mikrovezérlős rendszerfejlesztést. Korábban viszonylag igen sok kiegészítő ismeret kellett ahhoz, hogy egyáltalán az ember tudjon valamit kezdeni egy mikrovezérlővel. Az Arduino fejlesztői rendszerük tervezésekor arra törekedtek, hogy minél felhasználóbarátabb és minél egyszerűbb legyen a kész termék használata.

A platform egyszerűsége és nyíltsága miatt hatalmas sikernek örvend a rendszer. Ezen jelenséget figyelmen kívül hagyni hatalmas vétség lenne, ezért úgy döntöttem, hogy a Mikrovezérlős rendszerfejlesztés C nyelven második kötete az Arduino platformmal fog foglalkozni.

Ez nem egy különálló könyv. Erősen épít az első kötetre, mert ott a PIC programozás rejtelmei mellett igen sok szó esett a C programozás rejtelmeiről, valamint digitális technikáról és a mikrovezérlőkhöz kapcsolódó kiegészítő áramkörökről is. Ezen témakörök mikrovezérlőtől függetlenül alkalmazhatóak. Ezért ebben a könyvben csak az Arduino platformról lesz szó elsődlegesen.

A könyv készítésekor konkrétan egy Uno lapot használtam. Így a programozás rész is főleg ezen lapra vonatkozik. A többi modell esetén az analóg és digitális lábak száma között van leginkább különbség.

A projektek szekcióban jelen könyv esetén nem csak konkrét projektek és hozzájuk tartozó kódok találhatóak meg. Egy pár technológia leírása is helyet kapott ezen szekción belül, mivel nem igen fért volna be a könyv más részébe.

Köszönetnyilvánítás

Köszönettel tartozom az alábbi embereknek, akik támogatták ezen mű létrejöttét munkájukkal, biztatásukkal. Nélkülük biztosan nem lenne ilyen formában olvasható ez a könyv:

- *SZISZSZI Déri Miksa Tagintézmény vezetősége, kiemelten Vargáné Szőke Márta és Németh Géza*
- *Kollégám, Bicskei Róbert*
- *Máté Margit*
- *Mr. Barna, Vilmos és a „Szakszervezet” (Mondtam, hogy beleírom :))*
- *Diákjaim, akik hatalmas lelkesedéssel biztattak, valamint kérdések és ötletek millióival segítették a könyv fejlesztését. Biztatásotok, ötleteitek, kérdéseitek nélkül ez a könyv nem született volna meg. A nevek olyan sorrendben szerepelnek, ahogy eszembe jutottak. Jelentősége nincs.*

  Csányi Dániel, Fraknóy Attila, Garai Gábor, Molnár Zsolt Roland, Simon Gábor, Varga Lajos, Gál Gergő.

2. Az Arduino platform

Az Arduino platform története 2003-ig nyúlik vissza. Kolumbiában Hernando Barragán létrehozta a Wiring platformot diplomamunkájaként, majd ezt nyílt forráskód alatt közzé tette. A platform azóta is aktív, bár nem örvend akkora sikernek, mint az Arduino. Maga az Arduino platform 2005-ben született meg Massimo Banzi és Casey Reas munkájának gyümölcseként. A platform a nevét az Olaszországi Ivrea városának történelmi alakjáról Arduin of Ivrea-ról kapta. Az Arduino szó magyarul "bátor barát"-ot jelent.

Az évek alatt több, különböző modell jelent meg a platformból, ezek közös jellemzője (ami a hivatalosakat illeti) az, hogy Atmel mikrovezérlőket alkalmaznak, valamint egy egységes programozó környezetből használhatóak. Az összes modell kapcsolási rajza és a szoftverek forráskódja elérhető a projekt weboldalán, ami a http://arduino.cc oldalon található.

A platform nyíltságából adódóan léteznek klón lapok is, amelyek tudásban, minőségben igencsak eltérhetnek az eredeti lapoktól. Magyarországon a TávIR gyárt az Arduino-hoz kompatibilis lapokat és csomagokat. A weblapjuk a http://avr.tavir.hu címen érhető el.

Sajnos az Arduino hátránya az, hogy önmagában „csak" egy mikrovezérlő. Ezzel az a gond, hogy a fejlesztéshez mindenképpen szükséges más elektronikai komponenseket máshonnan kell beszereznie az embernek. A készítők egyelőre csak egy hivatalos kezdőcsomagot kínálnak, de korábban még ez sem volt.

A nem hivatalos, más gyártóktól származó kezdő csomagokkal a probléma az, hogy a gyártóik ráteszik a „kis" hasznukat ezen csomagokra, így nem igazán olcsóak szoktak lenni. Sokat tud spórolni az ember, ha saját maga válogatja össze az alkatrészeket, amik a fejlesztéshez kellenek. Beszerzés előtt érdemes tájékozódni az elektronikai boltok kínálatában és áraiban, mivel bolt és bolt között lehet árban nem is kevés különbség.

Fejlesztéshez szükséges eszközök

Két eszközre mindenképpen szüksége van az embernek, ha Arduino-val akar foglalkozni. Először is kell egy PC, ami futtathat Windows, Linux de akár Mac OS X-et is, mivel a szükséges fejlesztőkörnyezet mind a három operációs rendszerre elérhető. Ezen kívül természetesen szükséges még egy Arduino lap is. Továbbá a fejlesztéshez én még ajánlanék egy-két eszközt, amelyek beszerzése opcionális, nem feltétlenül szükséges.

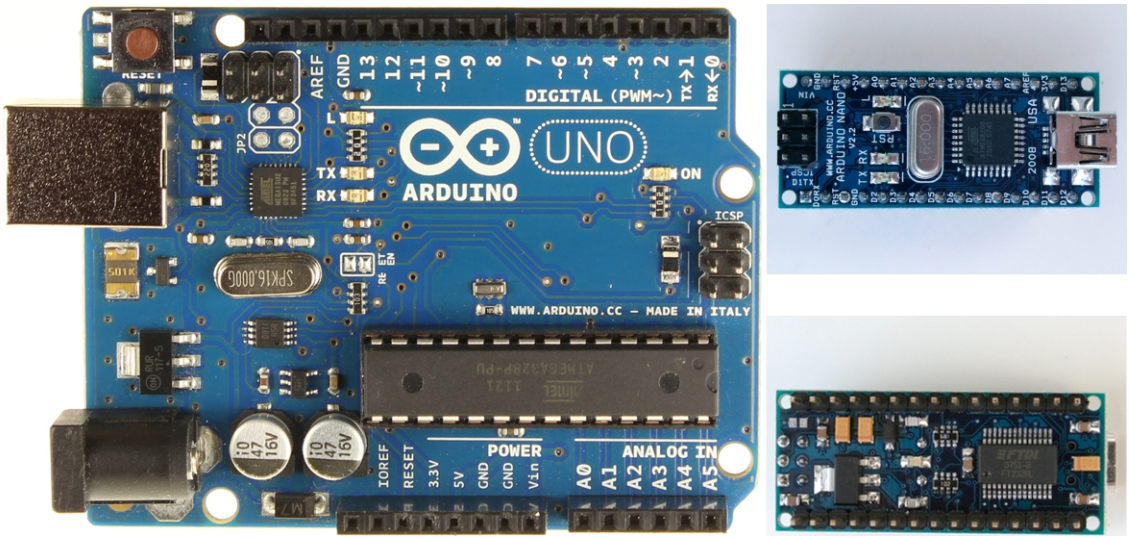
- *Egy Breadboard kísérletezéshez*
- *USB A-USB B kábel, vagy egyes Arduino modellek esetén USB A-Micro USB B kábel.*
- *Egy pár ellenállás,220 Ω, 1 KΩ, 10 KΩ*
- *Pár darab LED*
- *Eagle rajzolóprogram tervezéshez és kapcsolási rajz készítéséhez.*
- *Forrszemes Nyák lemez tartósabb prototípusok építéséhez*
- *Forrasztó és forrasztóón.*

Arduino típusok

A típusok listája viszonylag sűrűn változik, mivel egyes modellek népszerűbbek, mint mások, valamint a fejlesztések is folyamatosak. Így felesleges lenne egy teljes típuslistát ide illesztenem, mert elképzelhető, hogy 2-3 héten belül elavulttá válna a lista. Ezért itt csak a népszerűbb modellek ismertetése található meg.

Az aktuális lista egyébként kapcsolási rajzokkal együtt a http://arduino.cc/en/Main/Products címen érhető el. Ezen listában nem csak a fő fejlesztő lapok vannak felsorolva, hanem a kiegészítő lapok is. A kiegészítő lapok plusz tudással vértezik fel az alapmodelleket.

Arduino Uno / Arduino Nano



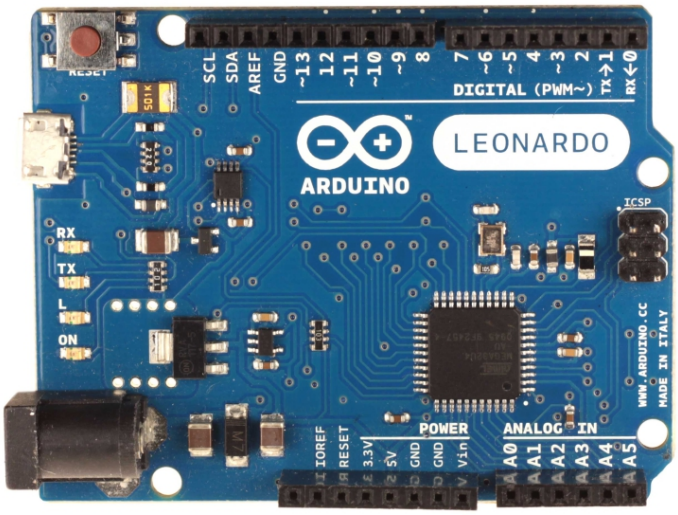
1. ábra: Arduino Uno és Arduino Nano

Digitális ki/bemenetek száma	*14, ebből 6db PWM képes*
Analóg bemenetek száma	*6*
Maximális áramerősség I/O lábanként	*40mA*
Programmemória / Adatmemória	*32KB, ebből 0,5KB foglalt / 2KB*
EEPROM memória mérete	*1KB*
Használt mikrovezérlő	*ATmega328 16MHz órajellel.*

1. táblázat: Az Arduino Uno és Nano technikai paraméterei

A legnépszerűbb Arduino modell, mondhatni az Arduino, mivel ezen szó hallatán a legtöbb embernek az UNO modell jut eszébe. Valódi USB kommunikációra nem képes, a lapon elhelyezett kicsi SMD mikrovezérlő USB soros kommunikációs átalakító feladatokat lát el. A Nano technikai paraméterei azonosak az Uno modellével, azonban a Nano kisebb méretű és csak USB-n keresztül táplálható. Külső tápegység fogadására nem képes.

Arduino Leonardo



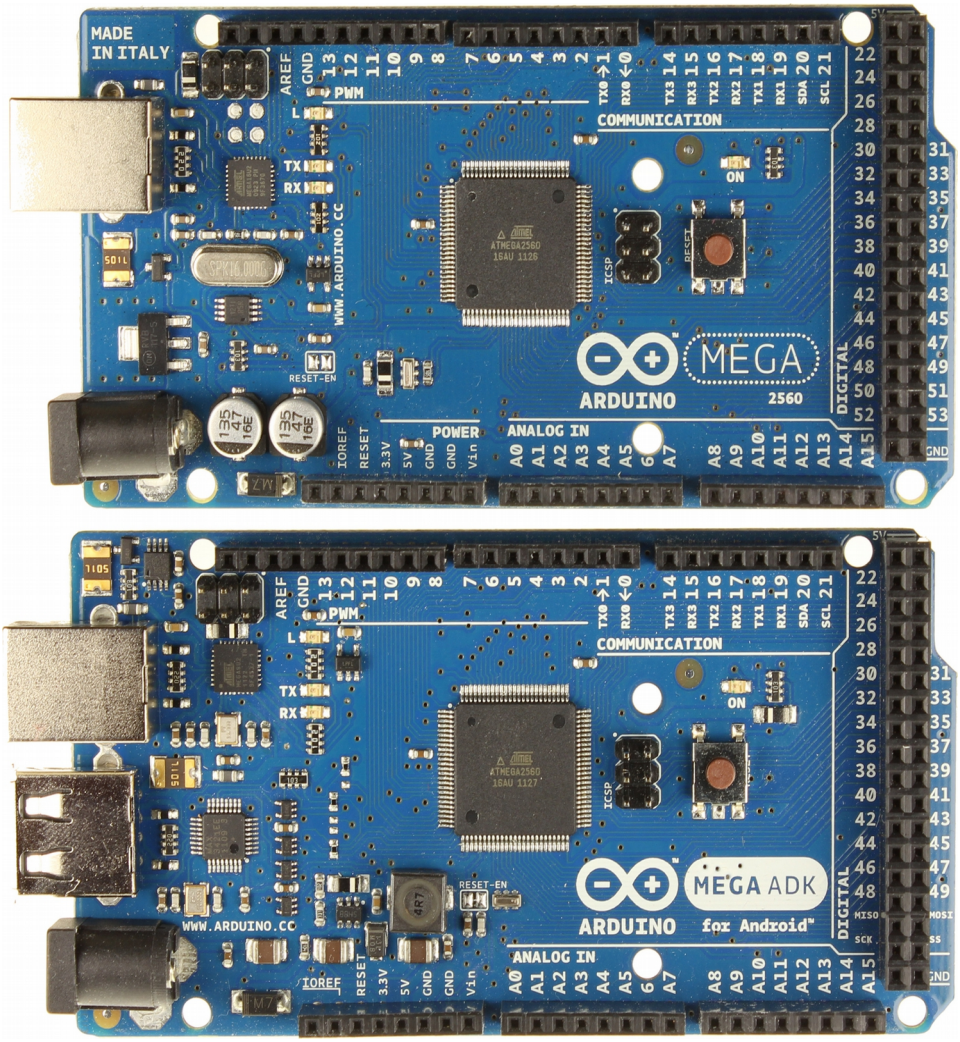
2. ábra: Arduino Leonardo

Digitális ki/bemenetek száma	*20, ebből 7db PWM képes*
Analóg bemenetek száma	*12*
Maximális áramerősség I/O lábanként	*40mA*
Programmemória / Adatmemória	*32KB, ebből 4KB foglalt / 2,5KB*
EEPROM memória mérete	*1KB*
Használt mikrovezérlő	*ATmega32u4 16MHz órajellel.*

2. táblázat: Az Arduino Leonardo technikai paraméterei

A Leonardo az Uno-val szemben valódi USB kommunikációs képességekkel rendelkezik. Tud emulálni billentyűzetet és egeret a PC számára, továbbá az USB-n keresztül soros portot is biztosít, méghozzá kettőt is.

Arduino Mega 2560 / Mega ADK



3. ábra: Arduino Mega 2560 és Mega ADK

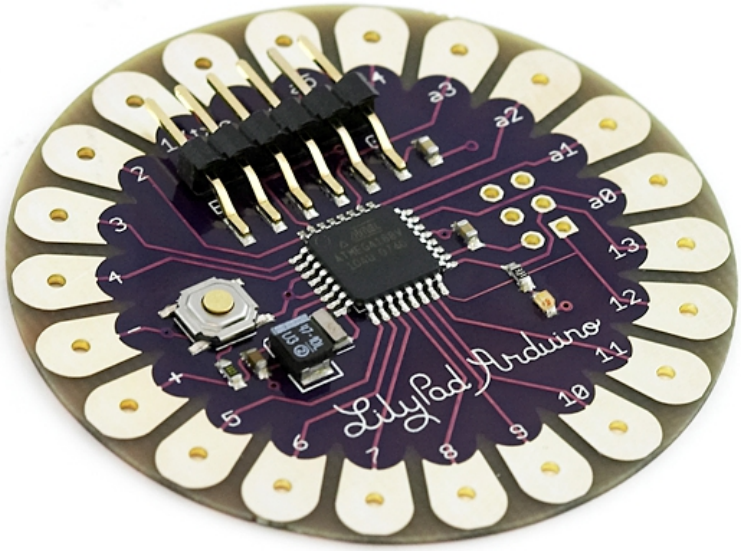
Digitális ki/bemenetek száma	*54, ebből 15db PWM képes*
Analóg bemenetek száma	*16*
Maximális áramerősség I/O lábanként	*40mA*
Programmemória / Adatmemória	*256KB, ebből 8KB foglalt / 8KB*
EEPROM memória mérete	*4KB*
Használt mikrovezérlő	*ATmega2560 16MHz órajellel.*

3. táblázat: Az Arduino Mega 2560 és Mega ADK technikai paraméterei

Az egyik legnagyobb tudású Arduino hardveres I2C és SPI buszrendszerrel rendelkezik, valamint 54 digitális ki-és bemenetet és 16 analóg bemenetet is biztosít. Az ADK változat annyiban tér el, hogy USB host funkciókra is képes, amit eredetileg arra terveztek, hogy képes legyen kommunikálni Android alapú telefonokkal. Mivel ezen lap USB host funkciókra is képes, ezért nagyon nem mindegy a

tápellátása. USB betáplálás esetén 500mA-t tud felhasználni, ami nem biztos, hogy elegendő mindenre. Ezért ajánlatos lenne külső tápegységről táplálni. Ebben az esetben olyan tápegység szükséges, ami legalább 1,5A áramot le tud adni.

Arduino LilyPad



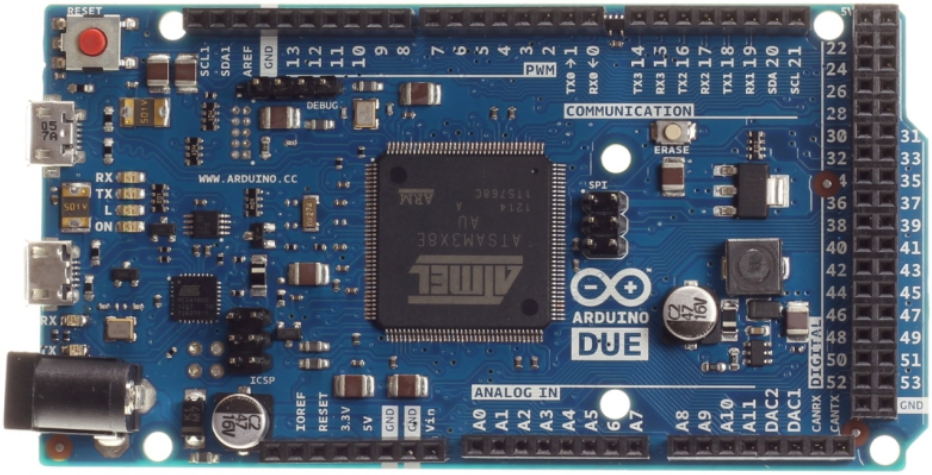
4. ábra: LilyPad Arduino

Digitális ki/bemenetek száma	*14, ebből 6db PWM képes*
Analóg bemenetek száma	*6*
Maximális áramerősség I/O lábanként	*40mA*
Programmemória / Adatmemória	*16KB, ebből 2KB foglalt / 1KB*
EEPROM memória mérete	*512b*
Használt mikrovezérlő	*ATmega128V/ATmega328V 8 MHz órajellel.*

4. táblázat: A LilyPad Arduino technikai paraméterei

Ezt az Arduino változatot direkt ruhák számára tervezték. Ezen mikrovezérlő előnye, hogy flexibilis és ezáltal jól használható fel intelligens ruházat készítéséhez. A programozásához külön átalakító adapter szükséges.

Arduino Due



5. ábra: Arduino Due

Digitális ki/bemenetek száma	*54, ebből 14db PWM képes*
Analóg bemenetek száma	*12*
Analóg kimenetek száma (DAC)	*2*
Maximális áramerősség I/O lábanként	*130mA, az összes lábbon együtt*
Programmemória / Adatmemória	*512KB / 96KB*
EEPROM memória mérete	*nincs*
Használt mikrovezérlő	*Atmel SAM3X8E 84 MHz órajellel.*

5. táblázat: Arduino Due technikai paraméterei

Az Arduino Due a legújabb Arduino változat. Ezen változat már 32 bites processzort használ, ami jóval nagyobb számítási teljesítményt kínál. Hátránya, hogy az eszköz 5V helyett 3,3V-os logikai jelekkel dolgozik. Ezért 5V közvetlenül nem kapcsolható rá, mert a processzor károsodásához vezethet. A nagyobb számítási teljesítmény előnye, hogy analóg/digitális átalakítóval is rendelkezik, méghozzá kettővel is, így készíthető belőle zenelejátszó is akár. Az Arduino ADK-hoz hasonlóan USB host funkciók ellátására is képes.

3. Beüzemelés

A beüzemelés első lépéseként le kell tölteni a fejlesztőkörnyezetet, majd telepíteni kell azt. Letölteni a http://arduino.cc/en/Main/Software oldalról lehet. Windows esetén a telepítés egy egyszerű kicsomagolásból áll csupán. Mac OS alatt is hasonlóan egyszerű műveletről van szó. Ubuntu/Debian alapú disztribúciók esetén a program telepíthető a szoftverközpontból is, vagy parancssorból az alábbi parancs kiadásával:

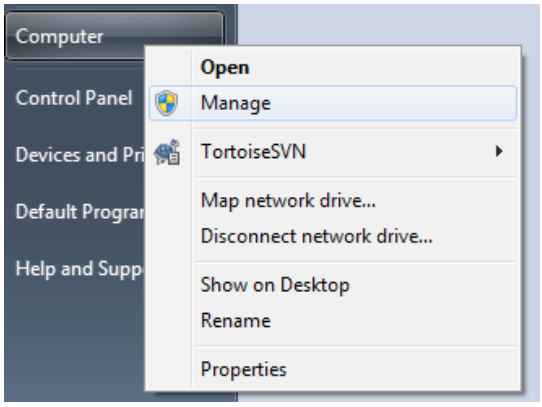
```
sudo apt-get install arduino
```

Ubuntu alapú disztribúciók esetén érdemes megjegyezni, hogy a szoftverközpont által letöltésre kínált verzió nem mindig a legfrissebb. Ezért érdemes figyelni a szoftver telepítésére vonatkozó oldalt az Arduino oldalon belül, melynek elérési címe: http://www.arduino.cc/playground/Learning/Linux

Amennyiben a letöltési vagy telepítési folyamat nem teljesen tiszta, a weblap kínál egy kezdő lépések leírást is a http://arduino.cc/en/Guide/HomePage címen.

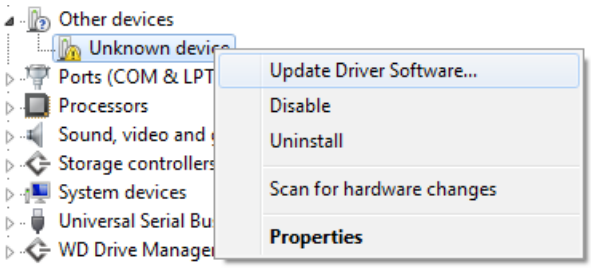
A szoftver letöltése és kicsomagolása után jöhet az illesztőprogram telepítése. Ehhez első lépésben csatlakoztatni kell a gépünk egy szabad USB portjára az eszközt. A csatlakoztatás után a Windows megpróbálja majd telepíteni a hardvert, de ez nem fog neki sikerülni, mivel a Windows nem tartalmaz beépítetten illesztőprogramot az Arduino számára.

Ezért manuálisan kell telepítenünk az illesztőprogramot. Első lépésben meg kell nyitni az eszközkezelőt. Ezt gyorsan és hatékonyan úgy tehetjük meg, hogy a Számítógép ikonon jobb kattintunk, majd a kezelés opciót választjuk. A megjelenő ablakban pedig bal oldalt kiválasztjuk az eszközkezelőt.



6. ábra: Számítógép-kezelés kiválasztása

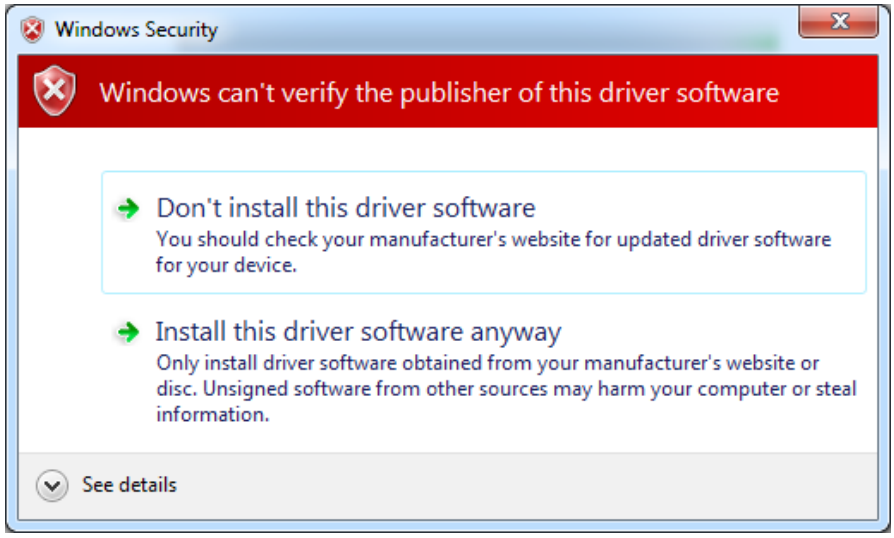
Az eszközkezelőben lesz egy kategória, amiben a fel nem ismert eszközök szerepelnek. Itt jó esetben egy eszköz található csak. Az ismeretlen eszközön kattintsunk jobb gombbal, majd válasszuk az Illesztőprogram frissítése opciót.



7. ábra: Illesztőprogram frissítése

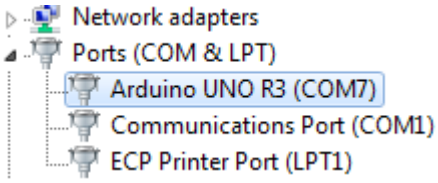
A megjelenő Illesztőprogram telepítő ablak két lehetőséget kínál. Vagy automatikusan kerestetünk a géppel illesztőprogramot, vagy manuálisan meghatározzuk a keresés helyét. Az utóbbit választva a következő lépésben meg kell adni az illesztőprogram helyét. Ez az Arduino mappán belül található Drivers mappa lesz.

Az illesztőprogram telepítése előtt meg kell erősíteni a felbukkanó figyelmeztetésben, hogy valóban telepíteni szeretnénk. A figyelmeztetést azért dobja fel a rendszer, mivel az illesztő nem rendelkezik digitális aláírással. Ezt most nyugodtan figyelmen kívül hagyhatjuk.



8. ábra: Illesztőprogram aláírás hiány miatti figyelmeztetés

Az illesztő telepítése után az eszközkezelőben a Portok kategóriában meg kell jelennie az Arduino lapunknak. Mellette egy COM port szám lesz olvasható. Erre a portszámra szükségünk lesz a programozó beüzemelésekor.

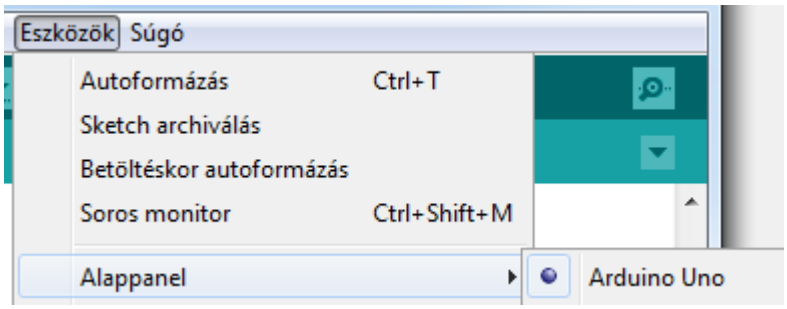


9. ábra: Telepített Arduino Uno az eszközkezelőben

Lap tesztelése, kész programok feltöltése

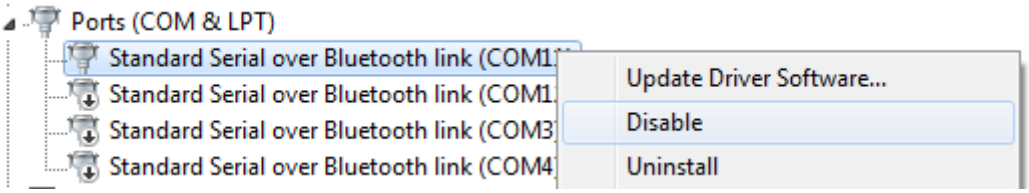
A folyamat végeztével használatba vehető a lapunk. Új, sosem használt lapok esetén érdemes leteszteltelni a működést egy egyszerű programmal. Erre a legmegfelelőbb az egyszerű LED villogtató alkalmazás. Azonban ehhez kell egy LED és egy ellenállás is. Azonban ha nincs kéznél ilyen, akkor tesztelhetjük a lap működőképességét egy soros kommunikációra épülő programmal is.

A teszteléshez meg kell nyitni az Arduino fejlesztőkörnyezet, majd az Eszközök menü alappanel menüpontjából ki kell választani a használt eszközt.

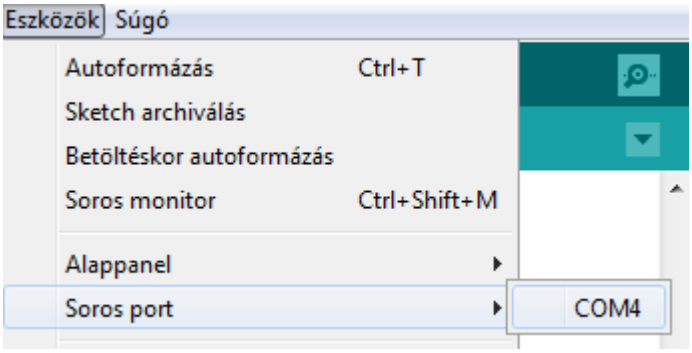


10. ábra: Alappanel kiválasztása

Az alappanel kiválasztása után ellenőrizzük a használt COM port számát. Ennek meg kell egyeznie az eszközkezelőben feltüntetett számmal. Amennyiben a gépen nem található integráltan soros port, akkor csak az Arduino lapot fogja látni a fejlesztőeszköz, mint ahogy az az alábbi képen is látható. Az eszközök menü megnyitása előfordulhat, hogy több másodpercig is eltart. Ez egy ismert hiba, amit olyan soros portok okoznak, amik Bluetooth eszközökhöz rendeltek. A probléma megoldható úgy, hogy ezen portokat ha nem használjuk letiltjuk. Ezt szintén az eszközkezelőben tudjuk megtenni. A kiválasztott porton jobb kattintás után a letiltás lehetőséget kell választani. A problémát okozó portok letiltása után az Eszközök menü megnyitására nem kell több másodpercet várnunk.

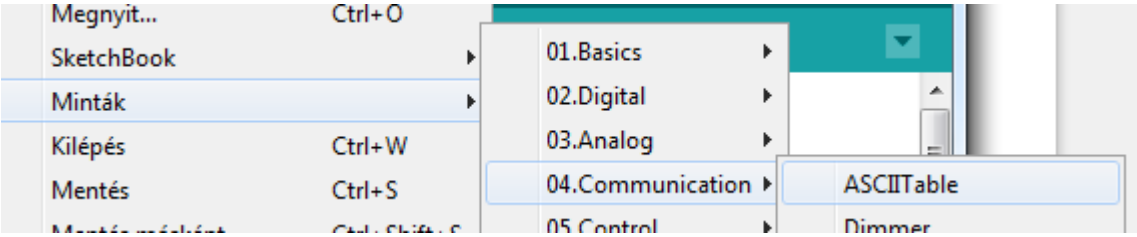


11. ábra: Nem használt Bluetooth soros port letiltása



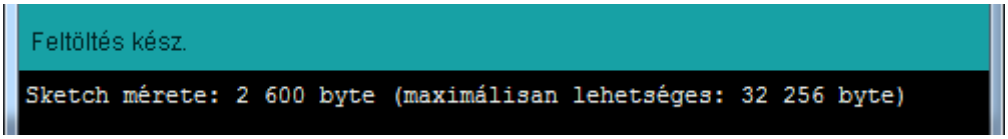
12. ábra: COM port kiválasztása

A port kiválasztása után jöhet a feltöltendő program kiválasztása. Tesztelésképpen most egy ASCII tábla programot töltünk fel az eszközre. Ez a program a beépített mintaprogramok között található meg. Megnyitni a Fájl/Minták/0.4 Communication/ASCIITable menüpont segítségével tudjuk.



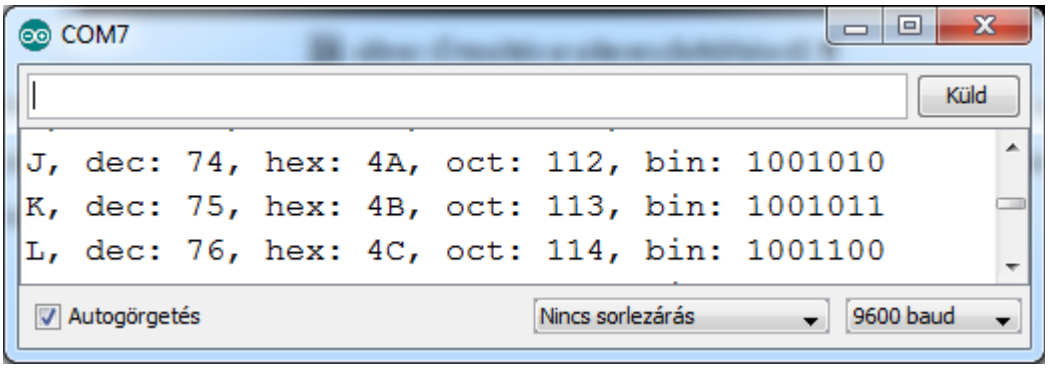
13. ábra: Teszt mintaprogram betöltése

Ennek hatására betöltődik a mintakód. Ezt az eszközre a Fájl menü Feltöltés parancsával tudjuk felvinni. A program fordítása és tényleges feltöltése az eszközre eltarthat egy ideig. A státusz sorban a folyamat végeztével kapnunk kellene egy értesítést.



14. ábra: Értesítés a sikeres feltöltésről.

Tesztelni a lap működőképességét az Eszközök menü alatt megbúvó soros monitor alkalmazással tudjuk. Ezt megnyitva a lapnak egy ASCII táblát kellene visszaküldenie a PC felé. A küldési folyamat a lapon található Reset gomb megnyomásával újraindítható.



15. ábra: Soros terminálban a program kimenete

4. Programozás, alapismeretek

Az Arduino fejlesztőkörnyezete C++ programozási nyelvet alkalmaz, méghozzá egyszerűsített formában. A fejlesztőkörnyezet nagy része a C és C++ terhelést leveszi a felhasználó válláról, mivel csupán két függvényt kell megírnunk ahhoz, hogy valamit kezdeni tudjunk a lapunkkal, így nem is feltétlenül kell ismerni a C++ sajátosságait, mert sima C tudással is használható az eszköz.

A könyv ezen fejezetében az Arduino alap függvénykönyvtárát ismertetem. Pontosabban a függvénykönyvtár azon részét, ami csak függvényekre támaszkodik.

Ahhoz, hogy fordítható programot készítsünk az Arduino környezetben, két függvényt kell megírnunk, ami 100%-al több, mint egy C program esetén (hiszen ott elég egy fő függvényt megírni), de cserébe legalább 80%-al könnyebb az eszköz programozása a PIC mikrovezérlőkhöz hasonlítva.

Az első ilyen függvény, amit meg kell írnunk, a setup névre hallgat. Definíciója a következő:

```
void setup()
{
}
```

Ez a függvény az eszköz indítása után fog lefutni. Itt kell megadnunk a használt lábak konfigurációját, vagyis, hogy melyik lábát akarjuk az eszköznek bemenetként vagy kimenetként használni. A nem konfigurált lábak nem használhatóak, illetve nem fognak megfelelően működni.

A másik függvény, amire mindenképpen szükségünk lesz, az a loop nevet viseli. Ezen függvény tartalma végtelenített ciklusban fog ismétlődni az eszközön. PIC mikrovezérlők esetén ezen feladatot a minden main függvényben előkerülő végtelenített ciklus látta el. A függvény definíciója:

```
void loop()
{
}
```

Ha fordítható programot szeretnénk kapni, akkor ezt a két függvényt mindenképpen meg kell valósítanunk. A programjaink, mivel nem teljesen felelnek meg a C++ követelményeinek .ino kiterjesztéssel mentődnek. Egy mappába csak egy ilyen programot menthetünk. Ezen programok Arduino terminológiában Sketch névre hallgatnak, ami magyarra fordítva vázlatot jelent.

A platform nagy előnye, hogy vázlatokat bőségesen találunk a fejlesztőkörnyezetben és az interneten is szétszórva. Szóval nem feltétlenül szükséges programozni tudni ahhoz, hogy látványos dolgokat műveljünk az eszközzel. Persze, ha valamilyen szinten tud programozni az ember, akkor könnyebb helyzetben van.

Az Arduino platform kódja az AVR Libc kódjaira támaszkodik. Ezen könyvtár használatával nem foglalkozok, mivel a szolgáltatásainak nagy része elérhető az Arduino platform egyszerűsített függvénykönyvtárain keresztül. Az AVR Libc dokumentációja a következő címen érhető el: http://www.nongnu.org/avr-libc/user-manual/modules.html

Be – és kimenetek konfigurálása

A setup függvényben a be-és kimeneteinket konfigurálnunk. A kimenetek konfigurálására szolgáló függvény a pinMode nevet kapta.

```
pinMode(pin, mode);
```

Első paramétere a láb, amit konfigurálni akarunk. Ez egy 0 és 13 közötti tetszőleges szám. Ez a digitális lábak konfigurálását fogja jelenteni. Ha az első paraméterben meghatározott számot kiegészítjük egy A betűvel, akkor a konfiguráció egy analóg lábra vonatkozik. Az analóg lábak extra konfigurálás nélkül analóg üzemmódban fognak működni. Konfigurálás esetén azonban digitális módba váltanak.

Második paraméter, amit meg kell határozni, az a láb működési módja. Itt három működési típust határozhatunk meg konstansokkal. Kimenet konfigurációhoz a paraméternek OUTPUT értéket kell adni, bemenethez pedig INPUT értéket. Egy speciális mód az INPUT_PULLUP, ami bemenetként konfigurálja a lábat és automatikusan 5V feszültségre húzza belső ellenálláson keresztül. Ez gyakorlatban a bemeneti logika váltására szolgál. Ebben az esetben, ha 5V érkezik a lábra, azt jelenti, hogy a bemenet hamis, ha viszont 0V érkezik a lábra, akkor azt jelenti, hogy a bemenet igaz. Néhány konfigurációs példa:

```
void setup()
{
   pinMode(1, OUTPUT); //digit. 1-es láb kimenet
   pinMode(2, INPUT); //digit. 2-es láb bemenet
   pinMode(A0, INPUT_PULLUP); //A0 digit. bem. Ford. logikával.
}
```

Digitális kimenetek olvasása és írása

A digitális kimenetek írására a digitalWrite funkció használható.

```
digitalWrite(pin, value);
```

Első paramétere az írandó láb. Ezen paraméter használata megegyezik a pinMode függvény első paraméterével. A második paraméter az érték, amit a lábra szeretnénk írni. Ez lehet 0 vagy 1. Ezen számokhoz biztosít a rendszer nekünk két konstanst is. A LOW konstans értéke 0, míg a HIGH konstans értéke 1.

Digitális bemenetet a digitalRead funkcióval tudunk olvasni.

```
digitalRead(pin);
```

Ennek a funkciónak csak egy paraméter kell. Ez a paraméter az olvasott láb azonosítója. Ez szintén megegyezik a pinMode esetén tárgyalt azonosítással. Ez a függvény értékként vagy 0-t vagy 1-et ad vissza, a bemenet állapotától függően. A bemenetek és kimenetek TTL kompatibilisek.

Analóg értékek olvasása

Analóg bemenet olvasásra az analogRead függvény szolgál.

```
analogRead(pin);
```

A függvény hasonlóan működik a digitális bemenet olvasáshoz. A paraméter szintén egy szám. Ez a szám az analóg bemenet számát adja meg. A függvény visszatérési értéke egy 0 és 1023 közötti szám, ami azt jelenti, hogy a belső A/D átalakító pontossága 10 bites.

Az analóg bemenetek esetén a maximális értékhez társított feszültség felülbírálható, de maximum 5V lehet. A felülbírálásra az analogReference függvény használható.

```
analogReference(type);
```

A függvény paramétere az analóg referencia típusát határozza meg. Itt két konstans érték közül választhatunk. A DEFAULT konstans 5V-ra (vagy 3,3V-ra modelltől függően) állítja a referencia maximumot, az EXTERNAL konstans pedig külső forrásra. A külső referencia feszültséget az AREF lábra kell kötni.

Arduino-tól függően lehetőségünk van még más, belső referencia használatára is. Erről érdemes tájékozódni az analogReference függvény leírásában, ami itt található meg:
http://arduino.cc/en/Reference/AnalogReference

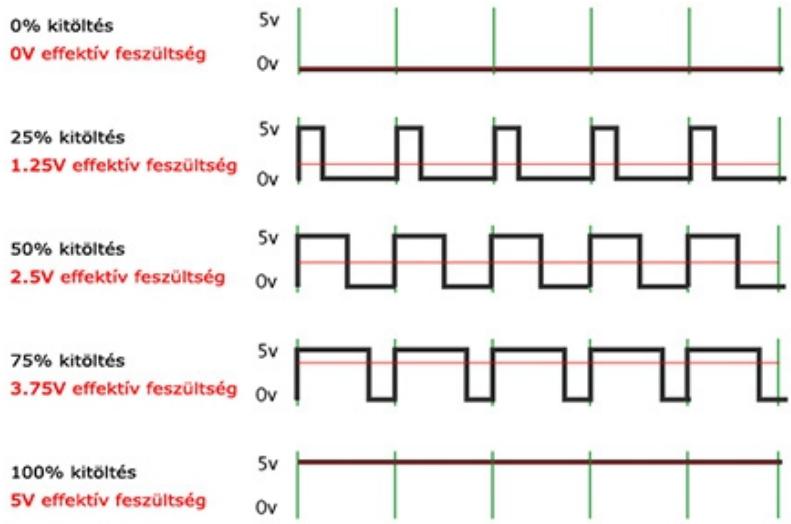
Az analóg referencia feszültség nem lehet negatív és nem haladhatja meg a lap által használt logikai egyes feszültség szintjét. Az analóg referencia feszültség változtatása a maximális érték lefelé tolására szolgál, így növeli a felbontóképességet.

Vegyük példának az Uno lapot, amely 10 bites A/D átalakítóval rendelkezik. Analóg referencia állítása nélkül 4,88mV / bit érzékenységre képes lap 0V és 5V közötti tartományban. Azonban, ha az analóg referencia lábra 3,3V-ot kötünk, akkor 5V helyett 3,3V lesz az olvasás maximuma, így az érzékenység 3,22mV / bit értékre csökken.

Analóg érték írása PWM modulációval

PWM (angolul a Pulse Width Modulation rövidítése, magyarul ez annyit jelent, hogy impulzus szélesség moduláció) modulációval egy olyan négyszögjelet állítunk elő, amelynek nem a frekvencia és amplitúdó paraméterét változtatjuk, hanem a kitöltési tényezőjét, vagyis, hogy mennyi ideig van magas szinten és alacsony szinten a kimenet. Ezt a tényezőt százalék értékben szokás megadni. Egy 50%-os PWM jel azt jelenti, hogy a jel az idő felében be, míg a másik felében ki van kapcsolva.

A kimenet kapcsolgatásából adódóan a rajta mérhető effektív feszültség a kitöltési tényezővel arányos lesz. Az előállított jel frekvenciájának akkorának kell lennie, hogy az ne befolyásolja a rákötött terhelést. Ez alkalmazástól függően pár száz Hz és pár kHz közötti frekvenciatartományt is jelenthet.



16. ábra: PWM moduláció magyarázat

Az Arduino nem minden digitális kimenete alkalmas ilyen modulált jel előállítására. A PWM-képes kimenetek mellett a panelen egy kis hullámvonal (~) látható.

Szoftverből a PWM-képes lábak vezérlésére a analogWrite függvény használható. Használatának feltétele, hogy a használni kívánt PWM-képes láb kimenetként legyen konfigurálva.

```
analogWrite(pin, value)
```

A függvény első paramétere szintén egy láb, ami megegyezik a pinMode függvényben használt elnevezéssel. A második paramétere pedig egy 8 bites szám, ami 0 és 255 közötti értéket vehet fel. Ennek segítségével ~0,39% lépésközzel tudjuk módosítani a kitöltési tényezőt. A PWM moduláció frekvenciája lábanként eltérő, mivel nem egy időzítő vezérli őket. A 3-as, 9-es, 10-es, és 11-es lábakon az alap frekvencia 31250Hz. Az 5-ös és 6-os láb esetén pedig 62500Hz.

A PWM frekvencia megváltoztatható, de erre alapértelmezetten nincs függvény a fejlesztőkörnyezetben. Ennek oka az, hogy a frekvencia módosítása csak a belső időzítők módosításával lehetséges, ami a késleltetések belső lelki világát igencsak összekavarja. Az Uno modellekhez az alábbi kódrészletet segítségével módosítható a PWM frekvencia.

```
//http://www.arduino.cc/playground/Code/PwmFrequency
void setPwmFrequency(int pin, int divisor)
{
    byte mode;
    if(pin == 5 || pin == 6 || pin == 9 || pin == 10)
    {
          switch(divisor)
          {
               case 1: mode = 0x01; break;
               case 8: mode = 0x02; break;
               case 64: mode = 0x03; break;
               case 256: mode = 0x04; break;
               case 1024: mode = 0x05; break;
               default: return;
          }
          if(pin == 5 || pin == 6)
               TCCR0B = TCCR0B & 0b11111000 | mode;
          else
               TCCR1B = TCCR1B & 0b11111000 | mode;
    }
    else if(pin == 3 || pin == 11)
    {
          switch(divisor)
          {
               case 1: mode = 0x01; break;
               case 8: mode = 0x02; break;
               case 32: mode = 0x03; break;
               case 64: mode = 0x04; break;
               case 128: mode = 0x05; break;
               case 256: mode = 0x06; break;
               case 1024: mode = 0x7; break;
               default: return;
          }
          TCCR2B = TCCR2B & 0b11111000 | mode;
    }
}
```

Lábak	***Osztószám / Frekvencia (Hz)***						
5, 6, 9, 10	*1 / 62500*	*8 / 7812*	*64 / 976*	*256 / 244*	*1024 / 61*	-	
3, 11	*1 / 31250*	*8 / 3906*	*32 / 976*	*64 / 488*	*128 / 244*	*256 / 122*	*1024 / 30*

6. táblázat: Lehetséges PWM frekvenciák lábanként a használható osztószámokkal

Adattípusok és kezelésük

Az Arduino környezetben használható alap adattípusokat az alábbi táblázat foglalja össze. Az eszköz nem rendelkezik alapértelmezetten 64 bites egész szám típussal és a double megegyezik a float típussal. Ennek oka a számítási erő hiánya.

Típus	Méret byte-ban	Tárolható értékek	
		Minimum	Maximum
boolean	1	false (0)	true (1)
char	1	− 128	+128
byte	1	0	255
int	2	− 32 768	+32 767
unsigned int / word	2	0	65 536
long	4	− 2 147 483 648	2 147 483 647
unsigned long	4	0	4 294 967 295
float	4	− 3,4028235E+38	3,4028235E+38
double	4	Azonos a float típussal számítási erő hiányában	

7. táblázat: Arduino környezetben használható adattípusok

Mivel a fejlesztőkörnyezet sima C helyett C++ nyelvet használ, ezért a fordítóprogram a típusok tekintetében intelligensebb, mint a MikroC. Így változókat definiálhatunk függvényeken belül is. Ezen szabadság miatt kétféleképpen működik a cost típus módosító.

Ha függvényen belül definiált változót a const kulcsszóval látunk el, akkor normál, a PC-k esetén is megszokott működést tapasztalunk. Vagyis a változó csak olvasható, de ebben az esetben az adatmemóriában fog tárolódni.

Ha függvényeken kívül, globálisan definiálhatunk egy változót const kulcsszóval ellátva, akkor szintén csak olvasható lesz, mégpedig hardveresen is, mivel ekkor a program a programmemóriában fog tárolódni.

Matematikai függvények

```
abs(x);
```

Egy szám abszolút értékét adja vissza

```
constrain(x, a, b);
```

Egy adott keret közé szorít be egy számot. Első paramétere a beszorítandó szám, a második paraméter a keret minimuma, a harmadik pedig a keret maximuma.

```
map(value, fromLow, fromHigh, toLow, toHigh);
```

Leképez egy adott halmazbeli számot egy másik halmazba. Első paramétere a leképezendő szám, a második a forrás halmaz minimuma, harmadik a forrás halmaz maximuma, negyedik a cél halmaz minimuma, ötödik a cél halmaz maximuma.

```
max(x, y);
```

Két szám közül a nagyobbat adja vissza.

```
min(x, y);
```

Két szám közül a kisebbet adja vissza.

```
pow(base, exponent);
```

Az első paraméter számot emeli a második paraméter által meghatározott kitevőre.

```
sqrt(x);
```

A paraméterként megadott szám négyzetgyökét adja vissza.

```
sin(rad);
cos(rad);
tan(rad);
```

Trigonometrikus függvények, a paramétert és a visszatérési értéket radiánban értelmezi.

```
random(max);
random(min, max);
```

Véletlenszerűen generál egy egész számot. Két változata is létezik. Egy megadott paraméterrel 0 és a paraméter által meghatározott érték közötti számot generál. Két paraméterrel használva a két paraméter által meghatározott tartományból generál számot. Első paraméter ebben az esetben a tartomány minimuma, a második paraméter a tartomány maximuma.

```
randomSeed(seed);
```

Beállítja a véletlenszám generátor szórását a paraméterben megadott szám alapján.

Bitekkel és byte-okkal kapcsolatos műveletek

Sok esetben előfordulhat, hogy egy változóban egy adott bitre vagyunk kíváncsiak, vagy mondjuk egy adott 2 byte-os szám felső vagy alsó byte-jára vagyunk kíváncsiak. Ekkor alkalmazhatnánk a C bitenkénti operátorait a célunk elérésére, azonban szerencsére a környezet beépítetten tartalmaz függvényeket ezen célokra. A bitműveletek csak egész típusokra alkalmazhatóak, míg a lowByte és highByte függvény bármely típusra.

```
lowByte(x);
```

Egy 2 byte-os típus alsó byte-ját adja vissza.

```
highByte(x);
```

Egy 2 byte-os típus felső byte-ját adja vissza.

```
bitRead(x, y);
```

Az első paraméterben meghatározott szám adott bitjének értékét adja vissza. A bit számát a második paraméter határozza meg.

```
bitWrite(x, y, z)
```

Az első paraméterben meghatározott szám adott bitjének értékét állítja be adott értékre. A bit számát a második paraméter határozza meg, míg a bit értékét a harmadik paraméter.

```
bitSet(x, y)
```

Az első paraméterben meghatározott szám adott bitjének értékét állítja 1 – re. A bit számát a második paraméter határozza meg.

```
bitClear(x, y)
```

Az első paraméterben meghatározott szám adott bitjének értékét állítja 0 – ra. A bit számát a második paraméter határozza meg.

```
bit(x)
```

Adott bit numerikus reprezentációját számítja ki, lényegében kettő hatványait. A bitet az első paraméter határozza meg. A 0. bit értéke 1, 1. bit értéke 2, 2. bit értéke 4, 3. bit értéke 8, 4. bit értéke 16, stb...

Különleges be/kimenet kezelő függvények

```
tone(pin, frequency);
tone(pin, frequency, duration);
```

Első paramétereként megadott lábon generál egy négyszögjelet (hang), amely frekvenciáját a második paraméter határozza meg. Opcionálisan 3 paraméterrel is meghívható, ekkor a 3. paraméter a hang lejátszásának hosszát állítja be. A 3. paraméter milliszekundumban értendő.

```
noTone(pin);
```

Az első paraméterben megadott lábon felfüggeszti a négyszögjel generálását.

```
shiftOut(dataPin, clockPin, bitOrder, value)
```

*8 bit adat továbbítására szolgál két lábon egy Shift regiszter felé. Az első paramétere az adatátvitelre használt láb, második paramétere az órajel közlésére szolgáló láb, harmadik paramétere a bitsorrend. Itt két konstans értéke közül választhatunk. A **MSBFIRST** konstans a legnagyobb helyi értéken lévő bittel kezdi az átvitelt, míg a **LSBFIRST** konstans a legkisebb helyi értéken lévővel. Utolsó paramétere pedig az átvinni kívánt adat (byte típus).*

```
shiftIn(dataPin, clockPin, bitOrder);
```

Adatot fogad két lábon egy 8 bites Shift regiszterből. Az első paramétere az adatátvitelre szolgáló láb, második paramétere az órajel láb, harmadik paramétere a bitsorrend. Bitsorrend esetén a MSBFIRST és LSBFIRST konstansok használhatóak. A függvény visszatérési értéke a fogadott 8 bites adat (byte típus).

```
pulseIn(pin, value);
pulseIn(pin, value, timeout);
```

Impulzust olvas a megadott lábon. Visszatérési értéke az impulzus hossza mikroszekundumban. Első paramétere a láb, amelyen olvassa az impulzust, második paramétere az impulzus olvasás indító feltétele. Itt két konstans közül lehet választani: HIGH vagy LOW értéket. A harmadik opcionális paraméter az, hogy mennyi ideig próbálkozzon az impulzus hosszának megállapításával. Ez a paraméter mikroszekundumban értendő. Amennyiben nem adjuk meg, akkor alapértelmezetten egy másodpercet vár. A függvény működése egy példán keresztül jobban megérthető.

Ha a függvényt HIGH konstanssal hívjuk meg, mondjuk a 7-es lábon, akkor a mikrovezérlő vár addig, amíg a 7-es láb logikai magas szintre kerül, majd elkezdi mérni az időt. Az időmérést akkor fejezi be, mikor a láb logikai alacsony szintre kerül. Ekkor visszatérési értékként a logikai magas szint időtartamát adja vissza.

Megszakítások

Megszakításokból két félét használhatunk mikrovezérlők esetén: időzítő vezéreltet vagy bemenet vezéreltet.

Bemenet alapú megszakítások

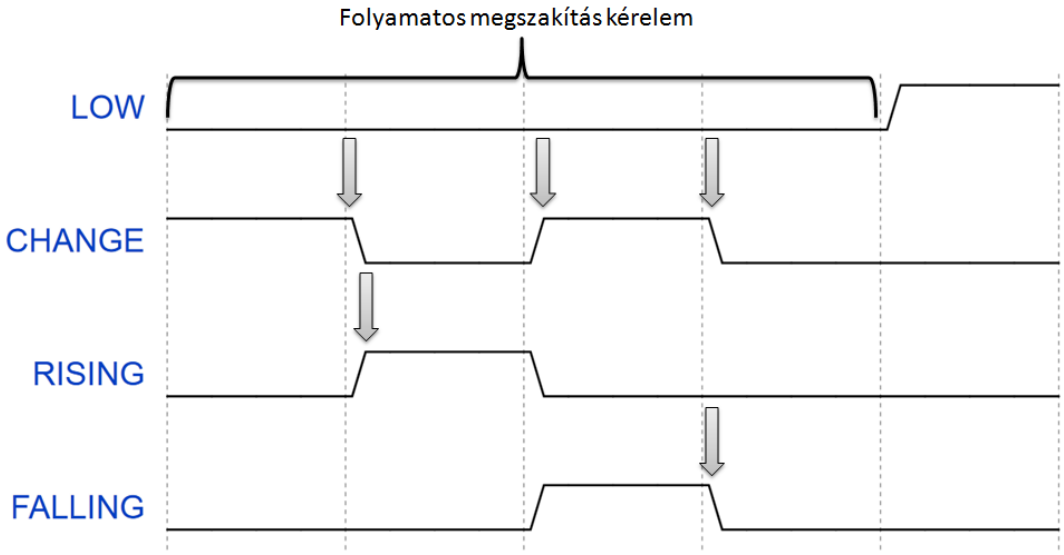
A bemenet alapú megszakítás vezérlés nem minden digitális lábon lehetséges, csak olyanok esetén, amelyek támogatják ezen funkciót. Az egyes Arduino modellek megszakítás fogadására képes lábait az alábbi táblázat foglalja össze.

	0. Megszakítás	1. Megszakítás	2. Megszakítás	3. Megszakítás	4. Megszakítás	5. Megszakítás
Uno, Ethernet	2. láb	3. láb	-	-	-	-
Mega2560	2. láb	3. láb	21. láb	20. láb	19. láb	18. láb
Leonardo	3. láb	2. láb	0. láb	1. láb	-	-

8. táblázat: Arduino modellek megszakítás kezelésére képes digitális bemenetei

A megszakítás képes lábak esetén négyféle megszakítás indítási feltétel közül választhatunk. Ezen módok konstansokként vannak definiálva a szoftverben. Ezek a következők:

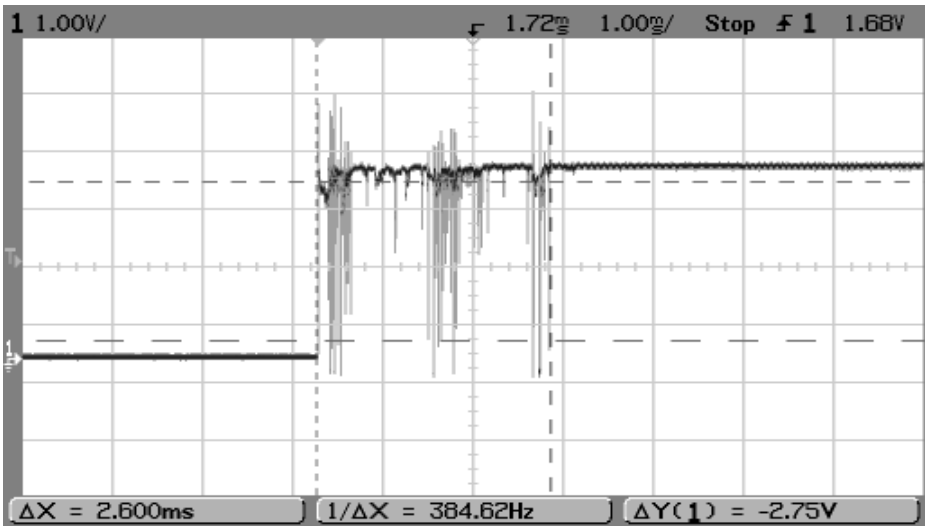
- ***LOW:** A megszakítás-kérelem érvényesül, ha a láb alacsony logikai szinten van. A megszakítás-kérelem csak akkor szűnik meg, ha a láb magas logikai szintre kerül.*
- ***CHANGE:** A megszakítás minden logikai szint váltáskor érvényesül.*
- ***RISING:** A megszakítás csak felfutó él esetén hajtódik végre.*
- ***FALLING:** A megszakítás csak lefutó él esetén hajtódik végre.*



17. ábra: Megszakítási módok és kód futtatások gyakorisága

Ha a megszakítási kérelmet egy nyomógombról szeretnénk levenni, akkor hardveresen kell tennünk róla, hogy a bemenetünk pergés mentes legyen. Szoftveresen késleltetés beiktatásával tudjuk csak megoldani a pergésmentesítést. Ha ezt a megoldást választanánk megszakítás esetén, akkor a megszakítás működésképtelenné válna, mivel a késleltetések belsőleg időzítő megszakításokra támaszkodnak.

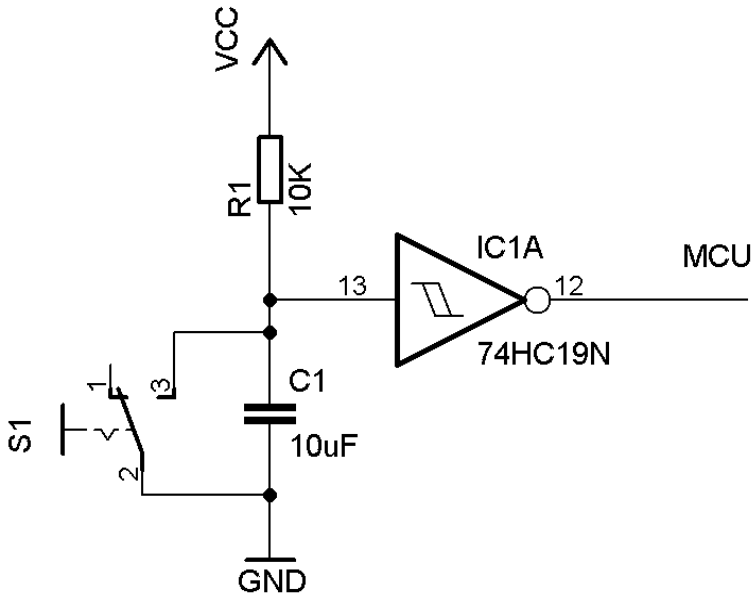
A pergés jelensége a kapcsoló mechanikus felépítéséből adódik. Az egymáson elmozduló fém érintkezők között az ellenállás-változás nulláról végtelenre és fordítva nem azonnal következik be, némi időátmenet alatt. Ezen rövid időátmenet alatt a kapcsoló bizonytalan állapotban van. Nagy sebességű áramkörök esetén ez azt eredményezheti, hogy a gombhoz rendelt funkció többször is végrehajtódhat egy gombnyomás hatására.



18. ábra: Kapcsoló pergés jelensége oszcilloszkópon

Hardveresen a pergés jelenség egy Schmitt triggerrel és egy soros RC rezgőkörrel könnyen kiiktatható.

Az alábbi kapcsolásban szereplő soros rezgőkör időkonstansa a $\tau = R \cdot C$ *alapján 0,1 másodperc, ami bőven elég a pergés jelenség lezajlásához. Az áramkörben használt 7419-es Schmitt triggeres inverter az RC kapcsolás kimeneti jelét simítja, majd invertálja. Ezáltal csak egy felfutó és lefutó élt kapunk a gombnyomás következtében.*



19. ábra: Pergésmentesítés hardveresen Schmitt triggeres inverter segítségével

A megszakítások kezelésére négy függvény áll rendelkezésünkre:

```
interrupts();
```

Engedélyezi a megszakítások működését. Alapértelmezetten engedélyezve vannak, csak akkor kell meghívni ezt a függvényt, ha a megszakítások letiltása után újra engedélyezni akarjuk azokat.

```
NoInterrupts();
```

Letiltja az összes megszakítást, de nem törli azokat (nem fog velük foglalkozni a processzor). Akkor hasznos, ha kritikus időzítést igénylő kódrészletet futtatunk.

```
attachInterrupt(interrupt, function, mode);
```

Megszakítási funkciót csatlakoztat egy megadott számú megszakításhoz (A megszakítás száma nem azonos a bemeneti láb számával! Lásd: korábbi táblázat). Első paramétere a megszakítás száma, második paramétere a megszakításhoz rendelendő függvény neve idézőjelek és zárójelek nélkül. Utolsó paramétere a megszakítás módja. Négy konstans választható: ***LOW***, ***CHANGE***, ***RISING***, ***FALLING.***

```
detachInterrupt(interupt);
```

Megadott számú megszakítás törlésére szolgál.

Időzítő alapú megszakítások

Az időzítő alapú megszakítások kihasználásához az alap függvénykönyvtár nem biztosít függvényeket, mivel az időzítők módosítása megváltoztatná a késleltető függvények és a PWM moduláció működését. Külső függvénykönyvtárral azonban ki lehet használni az időzítő alapú megszakításokat is.

Az időzítő alapú megszakítások úgy működnek, hogy a mikrovezérlőben található egy időzítő, ami nem más mint egy egyszerű számláló adott bit pontossággal. Ennek a számlálónak adandó órajelet beállítjuk az időzítőtől elvárt periódusidő alapján, majd mikor a számláló túlcsordul, akkor meghívja a megszakításhoz rendelt függvényt.

Az időzítő alapú megszakítások kezeléséhez szükséges könyvtár erről a címről szerezhető be: http://www.arduino.cc/playground/code/timer1

Késleltetések és időkezelés

Késleltetésekre az Arduino platform két függvényt biztosít. A függvények a MikroC-vel ellentétben nem a processzor fix utasítás végrehajtási idejéből számolódnak, hanem időzítő alapúak.

```
delay(ms);
```

Milliszekundumban meghatározott késleltetés (1s = 1000ms). A késleltetési idő nem lehet negatív, és 32 bites tartományon ábrázolhatónak lennie kell.

```
DelayMicroseconds(us);
```

Mikroszekundumban meghatározott késleltetés (1s = 1 000 000us). A késleltetési idő nem lehet negatív, és maximálisan 16 383 mikroszekundum adható meg modelltől függetlenül. A megadható maximális érték a pontosság miatt limitálva. Ezen érték feletti késleltetések nem lennének pontosak.

Egy Arduino-ban sincs valós idejű óra, viszont a beépített időzítőknek köszönhetően a bekapcsolás után eltelt idő kinyerhető. Erre a függvénykönyvtár két függvényt biztosít:

```
Micros();
```

Mikroszekundumban adja vissza a bekapcsolás óta eltelt időt 4 mikroszekundum pontossággal 16MHz-es Arduino modellek esetén. A 8MHz-es modellek esetén a pontosság 8 mikroszekundum. A függvény által visszaadott érték (előjel nélküli 32 bites egész szám) nagyjából 70 perc után túlcsordulást eredményez, ami után a mikrovezérlő újraindítja a számlálást.

```
Millis();
```

Milliszekundumban adja vissza a bekapcsolás óta eltelt időt. Előjel nélküli egész számot ad vissza, ami nagyjából 50 nap után túlcsordulást okoz. Túlcsordulás esetén szintén újraindul az eltelt idő számlálása.

Típuskonverziós függvények

A típuskonverziós függvények közös jellemzője, hogy ha a bemeneti változó nagyobb/kisebb értékkel rendelkezik, mint a céltípus által ábrázolható legnagyobb/legkisebb szám, akkor a konverziós függvény által visszaadott érték a céltípus legnagyobb ábrázolható értéke, vagy a legkisebb ábrázolható értéke.

Amennyiben lebegőpontos számot próbálunk meg konvertálni egész számmá, akkor a lebegőpontos szám egész része lesz a kimeneti érték. Tehát a lebegőpontos rész figyelmen kívül lesz hagyva, kerekítés nem fog történni.

A szöveg konverzió a szövegkezelés fejezetben van tárgyalva.

```
char(x);
```

Karakterré konvertálja a beadott x változót.

```
byte(x);
```

Byte típusúvá konvertálja a bemeneti x változót.

```
int(x);
```

Egész szám típusúvá konvertálja a bemeneti x változót.

```
word(x);
word(h, l);
```

Előjel nélküli egész számmá konvertálja a bemeneti x változót. Kétparaméteres változata esetén az első paraméter a felső byte értékét határozza meg, a második paraméter pedig az alsó byte értékét.

```
long(x);
```

Hosszú egész számmá konvertálja a bemeneti x változót.

```
float(x);
```

Lebegőpontos számmá konvertálja a bemeneti x változót.

5. C++ alapismeretek

Az előző fejezetben ismertetett függvénykönyvtár a teljes könyvtárnak csak egy része. Azonban a további függvények és osztályok ismertetése előtt egy kicsit beszélnünk kell a C++ sajátosságairól. Ezek ismerete nélkül is használhatóak a beépített függvények és objektumok, viszont frusztráló érzés lenne, ha nem tudnánk, mitől működik a rendszer úgy, ahogy.

Az itt ismertetett C++ tudás a teljes C++ egy része. Ennek oka az, hogy a C++ nyelv rejtelmeiről több ezer oldalt lehetne írni több-kevesebb sikerrel. Mivel mások már írtak ebben a témában könyveket szép számmal, ezért nem fog itt minden előkerülni a C++ nyelvről. Csak azon dolgok lesznek ebben a fejezetben, amelyek mindenképpen szükségesek ahhoz, hogy megértsük az objektumorientált programozási szemléletet és esetlegesen saját osztály/függvény könyvtárakkal bővítsük a beépített tudást.

A C++ annyiban másabb, mint a C, hogy lehetővé teszi az objektumorientált programozást, de ezt nem kényszeríti rá a felhasználóra minden áron. Ennek vannak előnyei és hátrányai is. Előny az, hogy sima C tudással is tudunk programozni, hátránynak azt lehet felróni, hogy igazán átláthatatlan és csúnya kódokat lehet készíteni, ha a két paradigmát (szemléletmódot) együtt, vegyesen alkalmazzuk.

Emiatt születtek meg a csak objektumorientált nyelvek, amelyek kikényszerítik a programozóból az objektumorientált gondolkodásmód alkalmazását. Ilyen nyelvek az ismertebbek közül a Java és a C#.

A C (és nagyjából az összes csak strukturált nyelv) legnagyobb problémája, hogy ha hosszú programokat írunk, akkor előbb-utóbb átláthatatlan kódot fogunk kapni a függvények sokasága miatt. Itt jön be az objektumorientált szemléletmód, amely lényege, hogy megpróbáljuk uralni a káoszt. Ehhez alapvetően két eszközt tudunk igénybe venni: az absztrakciót és a dekompozíciót.

Az absztrakció lényege, hogy a dolgok számunkra fontos jellemzőit elvonatkoztatjuk az általános, kevésbé fontosaktól, vagyis kiemeljük az egyedi tulajdonságokat az általánosak közül.

A dekompozíció lényege pedig az, hogy egy nagy rendszert egymással együttműködő kisebb rendszerek sokaságára bontunk úgy, hogy a rendszerek sokasága az eredeti nagy rendszernek megfelelően működjön.

Objektum és osztály

Objektumorientált nyelvek alapfogalma az objektum. Az objektum nem más, mint egy változó, amely egy olyan komplex típust reprezentál, ami adattagok mellett függvényeket is tartalmazhat. Továbbá ebben a típusban szabályozható az adattagok és függvények elérhetősége.

Az ilyen típusokat osztályoknak szokás nevezni. Tehát az objektum nem más, mint egy adott osztályú változó.

C++ nyelven egy osztály definíciója következőképpen néz ki:

```
class osztálynév
{
   private:
        //privát adattagok és függvények
   protected:
        //védett adattagok és függvények
   public:
        //publikus adattagok és függvények
};
```

Az osztálydefiníciók hasonlítanak az előre definiált függvényekre. Tehát az osztályban lévő adattagok és függvények prototípusait szokás itt megadni, a tényleges függvény implementációkat később szokás megadni.

Minden osztályban három hozzáférési szintet adhatunk meg. A privát adattagok és függvények csak az adott osztályon belül érhetőek el, a külvilág számára nem. Tervezés során olyan változókat és függvényeket érdemes ilyen minősítéssel megjelölni, amelyek szükségesek az objektum belső működéséhez, de nem kell, hogy a külvilág tudjon ezekről. Autós hasonlattal élve, ha az autót egy objektumnak tekintjük, melyben a motor szintén egy objektum, akkor a hengerfej például a motor privát adattagja a vezető szempontjából, mivel a vezetőnek a motorhoz csak annyi köze van, hogy megy-e, vagy nem.

A publikus adattagok és függvények az osztályon belül és az osztályon kívül is elérhetőek bárki számára megkötés nélkül. Ide olyan adattagokat és függvényeket szokás tenni, amiket a külvilág számára is használhatóvá szeretnénk tenni. Maradva az autós hasonlatnál, a kormánykerék egy publikus adattag az autóban, hiszen ahhoz hozzáférhet a vezető és a szerelő is egyaránt.

A védett adattagoknak öröklés során van szerepük, úgyhogy erről az öröklés témakörnél lesz szó, hogy konkrétan miért is jó ez.

Ha nincs olyan adattagunk vagy függvényünk, amit szeretnénk egy adott hozzáféréssel megjelölni, akkor a hozzáférési szint definíciója elhagyható. Továbbá, ha az osztályban nem adjuk meg egy adattag vagy tagfüggvény elérési szintjét, akkor alapértelmezetten privát típust fog alkalmazni a fordító.

Egy osztály nem tartalmazhat saját típusával megegyező adattagot, csak mutatóként. Továbbá az adattagoknak deklarálásukkor nem adhatunk kezdőértéket.

*Feltűnhetett az osztály deklarációkor, hogy nem lett használva a **typedef** kulcsszó. C++ esetén struktúrák és osztályok esetén nem szükséges ezen kulcsszó használata, mivel a fordító automatikusan*

típusként definiálja őket, nem pedig változóként.

További érdekesség, hogy C++ esetén (és egyébként egy-két érdekes C fordító esetén is) a struktúrák tartalmazhatnak függvényeket is. Tehát hasonló szerepkörrel bírnak, mint az objektumok. Azonban struktúra esetén nem adható meg adattag és függvény hozzáférési szint. Struktúrákban minden adattag és függvény publikus hozzáférési szinttel rendelkezik.

Konstruktor és destruktor

A konstruktor és destruktor az osztályok esetén két kiemelt jelentőségű függvény. A konstruktor szerepe az osztályon belüli változók kezdőértékének beállítása. Minden osztály legalább egy konstruktor függvényt tartalmaz. Ha nem definiáljuk ezt a függvényt, akkor is létrejön, csak éppen semmi hasznosat nem csinál. A konstruktor az osztály példányosításakor automatikusan le fog futni. A konstruktorok jellemzői:

Nincs visszatérési értékük (void kulcsszó sem kell eléjük); nevük megegyezik az osztály nevével; több is lehet belőlük, viszont paraméterek számában és típusában egyértelműen megkülönböztethetőnek kell lennie két konstruktornak; fogadhatnak paramétereket.

A hozzáférési szintek szintén szabályozhatóak konstruktorok esetén, viszont, ha csak egy privát vagy védett elérésű konstruktorral rendelkezik az osztályunk, akkor nem lesz példányosítható. Ennek absztrakt osztályok esetén lehet értelme. Absztrakt osztályokról későbbiekben lesz szó.

A destruktor függvény a konstruktor elletettje. Ez a függvény az objektum megszűnésekor fut le automatikusan. Feladata az objektum által lefoglalt erőforrások felszabadítása. Egy osztály csak egy destruktor függvényt tartalmazhat. A destruktor jellemzői:

*Neve megegyezik az osztály nevével az elején kiegészítve egy hullámvonallal (~), ami utal a negált konstruktorszerepkörre; szintén nincs visszatérési értéke (**void** se); nem lehetnek paraméterei.*

A destruktor kiemelten fontos szerepet lát el olyan osztályok esetén, amelyek dinamikusan foglalnak memóriát egyes adattagjaiknak. Ha egy ilyen osztályban a destruktor nem szabadítja fel az erőforrásokat, miután az objektum végzett, az úgynevezett memória szivárgásos hibát hoz létre. Ezen hibáról a dinamikus memória kezelés témakörnél lesz szó.

Tagfüggvények típusai

Az inicializáló és lebontó függvényeken (konstruktor és destruktor) kívül további négy szerepkört tudunk megkülönböztetni tagfüggvények esetén.

Lekérdező függvények

Mivel a C++ nem tartalmaz tulajdonság implementációt (legalábbis a C++ 98 szabvány nem, C++ 11 elképzelhető, hogy tartalmazza, viszont még nincs elterjedve), ezért a változók csak olvashatóvá tétele lekérdező függvényeken keresztül oldható meg. Ezen függvények privát vagy védett adattagok értékeit adják át publikus felületen a külvilágnak.

Beállító függvények

Egyetlen egy osztály esetén sem célszerű, hogy a felhasználó ellenőrizetlenül adjon értéket, még a publikus adattagoknak sem. Privát adattagok esetében erre nincs is lehetősége. Itt jönnek képbe a beállító függvények, amelyek adattagok értékeit állítják be ellenőrzött formában. Fejlettebb nyelvek esetén ezt a szerepkört szintén kiváltják a tulajdonságok.

Munkavégző függvények

Az osztály lényegi funkcióit valósítják meg.

Segítő függvények

Az osztály lényegi funkcióit megvalósító függvények munkáját segítik. Általában privát vagy védett hozzáférési szinttel rendelkeznek. Ilyen függvények segítségével a munkavégző függvények feldarabolhatóak kisebb, logikailag összetartozó részekre.

Osztályok megvalósítása és használatba vétele

Osztályok esetén maga az osztály definíció csak elődefiniálási szerepet lát el. Ezért szokás szétbontani az osztály definíciót és implementációt két külön fájlra. A definíció általában egy header fájlba kerül, míg az implementáció egy .ccp kiterjesztéssel ellátott fájlba. Általában az implementáció fájl és a header fájl neve ugyanaz, kiterjesztésükben van csak eltérés. Ezen lépés megtétele kis méretű osztályok esetén nem szükséges.

Az adattagok implementálása nem szükséges, mivel azok a definícióban szerepléssel implementálódnak. A tagfüggvényeket az alábbi szintaxis szerint lehetséges implementálni:

```
visszatérési_érték osztálynév::tagfüggvény_neve(paraméterek)
{
   //függvénytörzs
}
```

Implementációban kulcsfontosságú szerepet játszik a dupla kettőspont operátor, amit a szakirodalom hatókör feloldó operátornak vagy érvényességi kör operátornak nevez. A jobb érthetőség kedvéért íme egy teljes osztály implementáció:

```
class szamol
{
   private:
        double sz1, sz2, eredmeny; //változók
   public:
        szamol(double szam1, double szam2); //kontstuktor
        ~szamol(); //destruktor
        void MuveletVegez(char muvjel); //művelet végző fv.
        double Eredmeny(); //Eredmény lekérdező függvény
};

szamol::szamol(double szam1, double szam2)
{
   sz1 = szam1;
   sz2 = szam2;
   eredmeny = 0.0;
}

szamol::~szamol()
{
   sz1 = 0;
   sz2 = 0;
   eredmeny = 0;
}

void szamol::MuveletVegez(char muvjel)
{
   switch (muvjel)
   {
        case '/':
             eredmeny = sz1 / sz2;
```

```
            break;
        case '*':
            eredmeny = sz1 * sz2;
            break;
        case '+':
            eredmeny = sz1 + sz2;
            break;
        case '-':
            eredmeny = sz1 - sz2;
            break;
    }
}

double szamol::Eredmeny()
{
    return eredmeny;
}
```

A fenti példaprogramban egy egyszerű, négy alapművelet elvégzésére képes számológép osztály definíciója és implementációja látható.

A létrehozott osztályainkból objektumot úgy csinálunk, hogy egy, az osztály típusú változót létrehozunk. Amennyiben az osztályunk konstruktora rendelkezik paraméterekkel, akkor zárójelben meg kell adni paramétereket is.

Ezután az objektum publikusan elérhető részeire a pont operátor segítségével a következőképpen tudunk hivatkozni:

```
változónév.tagfügvény_neve()
```

A fentebb definiált számológép osztályunk használatára egy példa:

```
void main()
{
    szamol szamologep(4, 6);
    szamologep.MuveletVegez('*');
    double eredmeny = szamologep.Eredmeny(); //24
}
```

Statikus függvények és adattagok

A statikus adattagok és függvények jellemezője, hogy nem egy objektum példányhoz kötődnek, hanem konkrétan a megvalósító osztályaikhoz.

Ez adattagok esetén azt jelenti, hogy a statikus adattagok osztályonként egyszer tárolódnak a memóriában, míg a nem statikus társaik objektum példányonként. Ahhoz, hogy egy adattag statikus legyen, az adattag típusának meghatározása elé be kell írni a ***static*** *kulcsszót. Továbbá a fordító számára jelzésként globálisan is létre kell hozni a változót. Ez nem befolyásolja a statikus tag elérhetőségét, mivel ilyenkor is az osztályban megadott elérhetőségi szint lesz érvényes az adattagra. A következő mintakódrészletben egy statikus adattag létrehozás látható:*

```
class statikuspelda
{
   public:
        static int szam;
};


int statikuspelda::szam; //globális jelzés
```

A statikus tagfüggvények létrehozása hasonlómód a ***static*** *kulcsszóval történik:*

```
class statikusfv
{
   public:
        static int pelda();
};

int statikusfv::pelda()
{
   return 42;
}
```

A statikus tagok elérése az osztályon keresztül lehetséges, méghozzá az érvényességi kör operátor segítségével. Objektumokon keresztül a statikus tagok elérése nem lehetséges. Az alábbi példa részlet a statikus adattag és függvény használatot mutatja be:

```
statikuspelda::szam = 44; //addattag érték adás
statikusfv::pelda(); //függvényhívás
statikuspelda::szam = statikusfv::pelda();
```

Dinamikus memóriakezelés

A dinamikus memóriakezelés a C++ hatalmas előnye a C nyelvvel szemben. C esetén is lehetséges dinamikusan kezelni a memóriát, de jóval nehezebben érthető és kevésbé kézenfekvő, mint C esetén.

A dinamikus memóriakezelést tipikusan akkor szokták használni, ha a felhasználótól (vagy fájlból) kell bekérni változó mennyiségű adatot. Ha nem dinamikusan kezelnénk a memóriát, akkor mondjuk hasraütés-szerűen csinálnánk egy fix méretű tömböt. Ezzel az a probléma, ha az adatunk kisebb, mint a tömb mérete, akkor feleslegesen foglalunk erőforrást, ha pedig a tömb kisebb, akkor nem tudjuk kezelni megfelelően az adatmennyiséget.

A felesleges memóriakezelést elkerülhetjük dinamikus tömbök alkalmazásával. Ezen tömbök mérete a program futása alatt is megváltozhat. Azonban a méretváltoztatás a meglévő tömböt teljes egészében felülírja és nem hozzáfűzi az újonnan létrehozott elemeket. Dinamikus tömböt az alábbi módon tudunk létrehozni:

```
char *dinamikus = new char[12];
```

*Lényegében a dinamikus tömb nem más, mint egy mutató, ami egy adott memóriarészletre mutat. A **new** operátor (ami megvalósításában egy makró) foglalja le a megadott számú elemnek a memóriát.*

*Mivel a tömb nem más, mint egy mutató, azt hihetnénk, hogy ha simán az értékét NULL-ra állítjuk, akkor megszűnik létezni. A helyzet azonban az, hogy nem. A korábbi memóriafoglalás érvényben marad, de a mutató már nem lesz jó semmire. A lefoglalt memóriát a **delete** operátorral tudjuk felszabadítani. Tehát a korábban lefoglalt tömbünk helyes felszabadítása a következőképpen néz ki:*

```
delete [] dinamikus;
dinamikus = NULL;
```

A tömb zárójelek jelzik a delete után a fordítónak, hogy tömbtörlést hajtson végre. Ha itt nem tesszük ki a tömb zárójeleket, akkor csak az első elemet törli a memóriából és nem az összeset. Könnyen el lehet követni a nem törlést és a nem megfelelő törlés hibáját, főleg egy igen nagyméretű program esetén. Ezt a hibát szokás memóriaszivárgásnak nevezni. Igen nehezen felderíthető hiba. Többek között ez vezetett az úgynevezett menedzselt nyelvek kialakulásához, mint a C# vagy a Java.

Az osztályainkat is kezelhetjük dinamikusan. Az egyetlen változás a tagfüggvények és adattagok elérésében lesz. Ekkor ugyanis nem használhatjuk a tagkiválasztó pont operátort, helyette a C++ erre egy új mutatót vezet be, ami tagkiválasztásra szolgál dinamikus objektumokon. Ez az operátor az úgynevezett „mutató" operátor, ami egy kötőjelből áll és egy > jelből: ->

Az alábbi példa a korábban létrehozott számológép objektum használatát mutatja be dinamikus memóriakezeléssel:

```
void main()
{
    szamol *szamologep = new szamologep(4, 6);
    szamologep->MuveletVegez('*');
    double eredmeny = szamologep->Eredmeny(); //24
}
```

Az objektum használata után a memóriaterületét szintén a delete operátorral tudjuk felszabadítani:

```
delete szamologep;
szamologep = NULL;
```

Objektumok dinamikus kezelése esetén nem lehet eléggé kihangsúlyozni, hogy ha az objektumunk további dinamikus adattagokkal dolgozik, és a destruktor nem megfelelően szabadítja ezeket fel, vagy egyáltalán nem is szabadítja fel őket, akkor szintén memóriaszivárgást kapunk. A memóriaszivárgás kártékonyságát az alábbi mintaprogrammal lehet szemléltetni:

```
void main()
{
    while (1)
    {
          char *dinamikus = new char[1024];
    }
}
```

A fenti mintaprogram nem csinál semmi értelmeset, azonban a while ciklus minden futása esetén 1Kb memóriát lefoglal feleslegesen. Gépsebességtől függő a ciklus végrehajtásának gyorsasága, de igen rövid időn belül le fogja foglalni a programunk magának az összes szabad memóriát, kiszorítva ezáltal az operációs rendszert a memóriából. Az operációs rendszer próbálkozni fog memórialapozással védekezni, azonban egy idő után ez haszontalanná válik és előbb-utóbb összeomlik a rendszer, majd újraindul a gép. Mikrovezérlő esetén, mivel az eszköz kifogy a szabad memóriából, újra fog indulni.

A menedzselt nyelvek rendelkeznek egy szemétgyűjtővel, ami időnként lefut és felszabadítja azt a memóriaterületet, ami már nincs használatban.

Objektum saját magára mutatása

A C++ nyelvben a this kulcsszó kiemelten fontos szereppel rendelkezik. Egy osztályon belüli tagfüggvényben használva mindig egy, a létrejövő objektumra mutató mutatót jelöl. A this belső működésében csak egy szimbólum, így ténylegesen memóriát nem is foglal. Használatának olyan esetekben van értelme, ha az osztálynak és a függvénynek is van megegyező nevű változója. Ekkor a this mutatón keresztül hivatkozhatunk az osztály adott nevű adattagjára. Nézzünk egy egyszerű kódrészletet erre:

```
class pelda
{
   private:
        int ar;
   public:
        void szamolAr(int darab);
};

void pelda::szamolAr(int darab)
{
   int ar = 130 * darab; //lokális ar változó
   this->ar = ar; //osztály ar adattagja
}
```

Operátor átdefiniálás

Programozás során előbb-utóbb szembesülni fogunk azzal, hogy ha a nyelv operátorait fel tudnánk ruházni kiegészítő jelentéssel az osztályaink kezelésére, akkor jóval tömörebb, egyszerűbb, jobban olvasható kódot kapnánk. A programozási nyelvek ezen szolgáltatását, amivel a nyelv műveleti jeleinek értelmezését kiterjesztjük a saját osztálytípusainkra, operátor átdefiniálásnak nevezzük.

C++ esetén csak már a nyelvben létező operátorokat definiálhatunk át, új operátorokat nem alkothatunk. Az operátorok közül majdnem mindegyik jelentése átdefiniálható, kivéve három operátort. A nem átdefiniálható operátorok a tagkiválasztó operátor (.), az érvényességi kör operátor (::) és a háromoperandusú feltételes kifejezés operátora (? :).

Két operátornak átdefiniálás nélkül is értelmezhető jelentése van minden osztály esetén. Ez a címoperátor (&) és az értékadó operátor (=).

*Átdefiniálás során az operátorok precedencia szintje nem fog megváltozni, valamint az átdefiniált operátornak is pontosan ugyanannyi operandussal kell rendelkeznie, mint az eredeti operátornak. Amennyiben az operátornak létezik egy-és kétoperandusú változata, akkor mindkét változatát átdefiniálhatjuk. Továbbá az összetett értékadó operátorok jelentését is külön kell definiálnunk. Tehát a * és az = jel átdefiniálása nem teszi automatikusan értelmezhetővé a *= operátor jelentését a fordító számára.*

Az átdefiniálás megvalósítható globális függvényként, vagy osztály tagfüggvényeként. Ajánlott megoldás lenne az osztály tagfüggvényeként való átdefiniálás, mivel így az átdefiniáló függvények már a forráskódban is kapcsolódnak az osztályhoz. A kiíró és beolvasó operátorok csak globális függvénnyel definiálhatóak át. Ezen operátorok átdefiniálásáról itt nem lesz szó.

Az operátor átdefiniálás következőképpen valósítható meg C++ esetén:

```
visszatérési_típus operator műveleti_jel (típus változó, típus →
változó);
```

*Ha olyan operátort szeretnénk átdefiniálni, amit értékadás bal oldalán is használni szeretnénk, akkor az átdefiniáló függvénynek referenciát kell visszaadni, vagyis a visszatérési típus és az **operator** kulcsszó közé be kell tenni egy memóriacím operátort. (&) Az alábbi kódrészlet az összeadás jel átdefiniálását mutatja be:*

```
class komplex
{
   public:
        int i, r;
        int & operator + (komplex elso, komplex masodik);
}

int & komplex::operator + (komplex elso, komplex masodik)
{
   komplex ret;
   ret.i = elso.i + masodik.i;
   ret.r = elso.r + masodik.r;
```

```
    return ret;
}
```

Operandusok esetén a referencia átadás olyan operátorok esetén kötelező, amelyek módosítják valamely paraméter tartalmát. Ha nem vagyunk benne biztosak, hogy mikor szükséges ezt alkalmaznunk, akkor abból különösebb bajunk nem származhat, ha minden esetben a paramétereket is referencia átadással tesszük meg.

Fejlettebb nyelvek, mint a C#, kevesebb operátor átdefiniálását teszik lehetővé. Ennek oka az, hogy ha nem eléggé körültekintően járunk el, akkor nehezebben érthető programot fogunk kapni. Például vegyünk egy olyan esetet, ahol a programozó megvalósít egy Descartes koordináta-rendszerbeli vektor osztályt. Ezen osztályhoz átdefiniálja az egyenlőségjelet, mégpedig arra, hogy megvizsgálja, párhuzamos-e két vektor egymással.

Ezen alkalmazás szabályos, a nyelv megengedi, de nem biztos, hogy jó, mert ha nem nézi meg a kódot használatba vevő vagy továbbfejlesztő ember, hogy az egyenlőségjel mit is jelent a kódban, akkor azt hihetné, értékadásról van szó.

Ezért az operátorok alapértelmezett jelentésétől radikálisan eltérő célra történő átdefiniálása nem ajánlott. Továbbá a new, delete, =. -> operátorok átdefiniálását nem tiltja meg ugyan a fordító, de nem érdemes őket más jelentéssel felruházni.

Barát függvények és barát osztályok

Ha egy függvényt egy osztály tagjaként adunk meg, akkor a következőket jelezzük:

– A függvény hozzáférhet az osztály védett és privát részeihez.

– A függvény az osztályhoz (statikus esetben) vagy objektumhoz tartozik.

A barát függvények jellemzője az, hogy nem tartoznak az osztályhoz vagy objektumhoz, de hozzáférhetnek az osztály/objektum privát és védett részeihez. Leggyakrabban külső függvényes operátor átdefiniáláskor használatosak, mivel másképpen az operátor nem férhetne hozzá az osztály védett részeihez.

Ahhoz, hogy egy függvény barátként legyen megjelölve, szerepelnie kell annak az osztálynak a definíciójában, amihez barátként rendelni szeretnénk, mégpedig úgy, hogy a ***friend*** *kulcsszóval látjuk el. Az alábbi példában egy olyan osztály definíciója látható, amely rendelkezik egy barát osztállyal.*

```
class baratpelda
{
   friend void fv(baratpelda &b); //barát függvény definíció
   private:
        int x, y;
};

void fv(baratpelda &b) //referencia átadás, mivel értéket módosít
{
   --b.x;
   ++b.y;
}
```

Amennyiben barátként jelölünk meg egy függvényt, akkor az nem csak a privát adattagokhoz fog hozzáférni, hanem a publikus és védett adattagokhoz és függvényekhez is.

A barátnak jelölt függvények definíciója az osztály definícióban bárhol elhelyezhető. Átláthatóság szempontjából azonban érdemes az osztály definíció elején megadni ezeket.

A barát függvény lehet más osztályból származó is:

```
class A
{
   public:
        int pelda();
};

class B
{
   friend int A::pelda();
};
```

Függvényeken kívül egész osztályok, struktúrák megjelölhetőek barátként. Ebben az esetben csak az osztály nevét kell megadni, nem kell a teljes osztály definíciót megadni.

```
class A
```

```
{
    friend class B;
    private:
        int a, b, c;
};

class B
{
    private:
        void titkos() { }
    public:
        int d, e;
        A barat;
        void BaratModosito();
};

void B::BaratModosito()
{
    barat.a = 5;
}
```

Osztályok barát viszonya esetén érdemes megjegyezni, hogy ez a viszony nem kölcsönös az osztályok között. Tehát attól, hogy A osztály megengedi B osztálynak, hogy hozzáférjen az adattagjaihoz és függvényeihez, attól még a B osztály nem fogja megengedni A osztálynak ugyanezt.

Pontosabban automatikusan nem fog ez történni. Abban az esetben, ha B osztályban barátként jelölném meg A osztályt, akkor A osztály is képes lenne hozzáférni és módosítani B osztály privát részeit.

A barát viszony hasznos eszköz tud lenni, azonban használata nem igen ajánlott, csak akkor használjuk, ha mindenképpen szükséges, mivel súlyosan megsérti az egységbe zárás alapelvét. Éppen emiatt a fejlettebb objektumorientált nyelvekben (C# és Java) ez a viszony nem is létezik.

Öröklődés, osztályok származtatása

Az öröklődés az objektumorientált nyelvek sajátossága. Kettő vagy több osztály között fennálló kapcsolat. Ebben a kapcsolatban az egyik osztály elnevezése bázis (ős, szülő) osztály, a másik osztály elnevezése pedig leszármazott (utód) osztály. A leszármazott osztály a bázis osztály tulajdonságait örökli, de emellett rendelkezhet olyan tulajdonságokkal, amik az eredeti bázis osztályban nem találhatóak meg. Továbbá az öröklődés segítségével az egymással kapcsolatban álló osztályainkat hierarchikusan rendezni tudjuk.

A leszármazott osztályokról elmondható, hogy felülről kompatibilisek a bázis osztályaikkal, vagyis ha B osztály A osztály leszármazottja, akkor B osztály valójában nem más, mint egy A osztály néhány kiegészítő tulajdonsággal.

Az öröklésnek C++ esetén két fajtája van, analitikus és korlátozó. Analitikus öröklődés esetén a leszármazott osztály úgy bővíti tulajdonságokkal a bázis osztályt, hogy a bázis osztály tulajdonságit meghagyja, nem csorbítja és nem korlátozza.

Korlátozó öröklődés esetén viszont a származtatott osztályban nem lesznek elérhetőek a bázis osztály bizonyos tulajdonságai, tagfüggvényei.

Az öröklés szintaktikája:

```
class leszármazott_osztály : elérés_módosító Ősosztály
```

*Elérés módosítóként használható a **public**, **protected** és **private**. Amennyiben **public** kulcsszót használunk, akkor analitikus öröklés fog történni, **protected** és **private** esetén korlátozó öröklődés.*

A különböző öröklési módok eltérően hatnak a leszármazott osztályokra. A három öröklődési fajta hatását az örökölt adattagokra és függvényekre az alábbi táblázatokban foglaltam össze:

Bázisosztály tagjainak elérési szintje:	*Leszármazott osztályban az örökölt tagok elérési szintje:*
private	*Nem lesz elérhető*
protected	***protected***
public	***public***

9. táblázat: Publikus, analitikus öröklés hatása a leszármazott osztályra

Bázisosztály tagjainak elérési szintje:	*Leszármazott osztályban az örökölt tagok elérési szintje:*
private	*Nem lesz elérhető*
protected	***protected***
public	***protected***

10. táblázat: Védett, korlátozó öröklés hatása a leszármazott osztályra

Bázisosztály tagjainak elérési szintje:	*Leszármazott osztályban az örökölt tagok elérési szintje:*
private	*Nem lesz elérhető*
protected	***private***
public	***private***

11. táblázat: Privát, korlátozó öröklés hatása a leszármazott osztályra

Amennyiben az öröklés módját nem adom meg a leszármazott osztályomban, akkor privát öröklés fog történni. A privát és védett öröklés kívülről nem különböztethető meg igazán a privát vagy védett adattagok és függvények esetén, mivel ezek csak az osztályon belül érhetőek el. Ezen öröklési módok hatása leginkább a belőlük származtatott osztályok esetén érezhető.

Egy osztály több bázis osztállyal is rendelkezhet. Ezt nevezzük többszörös öröklésnek. A származtatás során a leszármazott osztály örökli az összes bázis osztály publikus és védett tagját és függvényét. A többszörös öröklés elsőre igen hasznos szolgáltatásnak tűnhet, azonban számos ponton feleslegesen túlbonyolíthatja a programunk logikai szerkezetét. A legtöbb dolgot egy ősosztályból származtatással is meg lehet oldani. Éppen ezért fejlettebb objektumorientált nyelvek (Pl: C#) nem támogatják a többszörös öröklést. C++ esetén is csak akkor használjuk, ha nagyon szükséges.

A többszörös öröklés szintaxisa hasonló az egyszeres öröklésheZ:

```
class osztálynév: elérés_módosító Ősosztály1, elérés_módosító
Ősosztály2, ... elérés_módosító ŐsosztályN
```

Örökölt komponensek kezelése

Az ősosztályból származó komponensek kezelése ugyanúgy történik, mint az osztály saját komponenseinek kezelése. Az örökölt komponensekre hivatkozhatunk az érvényességi kör operátorral is. Erre akkor lehet szükségünk, ha a származtatott osztályban létrehozunk egy függvényt, amely neve és paraméterlistája megegyezik egy, a bázis osztályban található függvénnyel.

Ezt az eljárást szokás függvény átdefiniálásnak is nevezni, mivel a leszármazott függvény az eredeti bázis osztályban szereplő függvény viselkedését megváltoztatja.

Az alábbi rövid példa egy ilyen esetet mutat be:

```
class A
{
   public:
        int fv();
}

int A::fv()
{
   return 3;
}

class B : public A
{
```

```
    public:
        int fv(); //ilyen az A osztályban is van;
}

int B::fv()
{
    int eredeti = A::fv(); //bázis osztály fv() hívása
    //új érték a bázis osztály értékének felhasználásával
    return eredeti + 12;
}
```

Konstruktorok a leszármazott osztályokban

A konstruktorok és az átdefiniált értékadó operátorok nem öröklődnek, így ezeket a leszármazott osztályban is el kell készíteni.

Amennyiben a bázis osztály csak paraméteres konstruktorral rendelkezik, akkor szükséges ennek a meghívása általában ahhoz, hogy használni tudjuk megfelelően az osztályt. Öröklés esetén sincs ez másképpen, úgynevezett explicit módon lehetséges a leszármazott osztályból meghívni a bázis osztály konstruktorát.

```
class os
{
    private:
        int x, y;
    public:
        os(int x, int y);
        int osszeg();
}

os::os(int x, int y)
{
    this.x = x;
    this.y = y;
}

int os::osszeg()
{
    return x+y;
}

//az osszeg függvény meghívható innen is,
//de értelmetlen hülyeséget adna vissza. Ezért a konstruktorral
//explicit módon meg kell hívni a bázis osztály konstruktorát
class uj: public os
{
    public:
        uj();
}
//ősosztály konstruktorának meghívása 4 és 3 értékkel, ezután
//az osszeg függvény már nem ad hülyeséget vissza.
uj::uj() : os(4, 3)
{
```

```
}
```

Virtuális függvények

Az örökölt komponensek kezelése részben volt szó a függvény átdefiniálásról. Az ott ismertetett módszerrel az a baj, hogy ha mutatón keresztül szeretnénk elérni a származtatott osztályunk átdefiniált függvényét, akkor a fordító mindig a bázis osztály függvényét hívja meg, mivel fordítás közben nem fogja tudni eldönteni nekem a fordító a mutató típusát. A problémát szemlélteti az alábbi példaprogram:

```
class Egyik
{
   public:
        int Valami();
}

int Egyik::Valami()
{
   return 33;
}

class Masik : public Egyik
{
   public:
        int Valami();
}

int Masik::Valami()
{
   return 44;
}

int main()
{
   Egyik e;
   Masik m;
   int hivas1 = e.Valami(); //33
   int hivas2 = m.Valami(); //44
   Egyik *mutato = new m();
   int hivas3 = mutato->Valami();//33-at fog adni
   return 0;
}
```

*Ezen probléma elkerülésére vezették be a virtuális függvényeket. Segítségükkel kiküszöbölhető ez a jelenség. A módosítást elég elvégezni a bázisosztályban. A módosítás abból fog állni, hogy a függvényünk neve előtt elhelyezzük a **virtual** kulcsszót. A származtatott osztályok esetén nem kell elhelyezni ezt a kulcsszót, mivel a virtuális függvény öröklés után is virtuális marad. A virtuális függvények által megvalósított megoldást késői vagy dinamikus kötésnek nevezik, mivel a programunk futás közben fogja eldönteni, hogy konkrétan melyik függvényt is kell neki meghívnia a mutatón keresztül.*

Absztrakt osztályok

A programunk felépítése során előfordulhat, hogy készítünk egy ősosztályt, ami általános tulajdonságokat és függvényeket tartalmaz a többi osztályunk leírásához. Ez az osztály önmagában nem alkalmas semmire, ezért jó lenne valahogy elérni azt, hogy csak öröklési láncban használható legyen, és ne lehessen objektumot létrehozni belőle.

Az ilyen osztályokat, amelyek leszármazottjai példányosíthatóak, de maga az ősosztály nem, absztrakt osztályoknak nevezzük.

C++ esetén egy osztály akkor lesz absztrakt, ha tartalmaz legalább egy tisztán virtuális függvényt. Egy függvény akkor tisztán virtuális, ha virtuális és nem csinál semmit. Ezt a fordító számára 0 értékkel jelezzük.

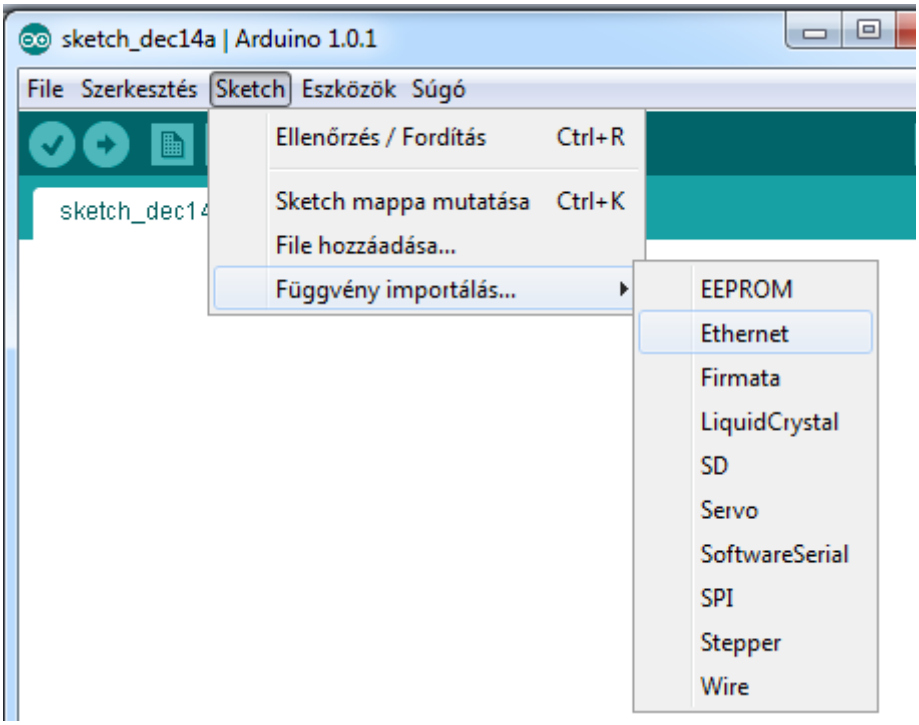
```
class absztraktpelda
{
   protected:
        int x0, y0;
   public:
        absztraktpelda(x, y);
        virtual int szamol() = 0; //tisztán virtuális függvény
}

absztraktpelda::absztraktpelda(x, y)
{
   x0 = x;
   y0 = y;
}

int main()
{
   absztraktpelda a(2, 65); //hibát fog kiváltani
   return 0;
}
```

Amennyiben a származtatott osztályunk az örökölt tisztán virtuális függvényt nem definiálja át, akkor a származtatott osztály is absztrakt osztály lesz.

6. Az Arduino osztálykönyvtárai

A legtöbb osztálykönyvtár az Arduino esetén külön header fájlban van elhelyezve azért, hogy amikor nincs rájuk szükség, feleslegesen ne növeljék a kód méretét. A szövegkezelést és a soros kommunikációt biztosító könyvtárat nem kell külön importálni, ezek minden esetben elérhetőek.

Azért, hogy kellően egyszerű legyen a függvénykönyvtárak használata, az Arduino környezet rendelkezik egy importáló felülettel, ami a szükséges #Include utasításokat elhelyezi a programunk elején. Az elérhető és importálható könyvtárak listája a Sketch->Függvény importálás menü alatt található meg.



20. ábra: Az Arduino osztálykönyvtár kezelő rendszere

Szövegkezelés

C és C++ nyelvek esetén az alap osztály és függvény könyvtár nem tartalmaz dedikáltan szöveg típust. A szövegek karakter tömbökként vannak megvalósítva. Ez egészen addig nem is okoz problémát, míg nem kell komolyabban szövegkezeléssel foglalkoznunk.

A „sima" C stílusú szövegkezelés nehézségeit bemutatóan nézzünk meg egy példát. Tételezzük fel, hogy van egy karakter tömbünk, aminek egy bizonyos részére lenne szükségünk további műveletvégzési szempontból. Ekkor lényegében csinálnunk kell egy karakter típusú mutatót, ami majd tárolni fogja a cél szövegrészletet. Ennek dinamikusan memóriát kell foglalnunk, hogy feleslegesen ne pazaroljunk memóriát, majd a megadott karaktereket átmásolni az egyik tömbből a másikba.

Kicsit macerás műveletnek hangzik, és valójában az is. Ezért van, hogy a fejlett objektumorientált nyelvek a szövegeket objektumokként kezelik. Belsőleg ezen objektumok az esetek többségében ugyanúgy karakter tömbökként kezelik a szöveget, viszont sokkal könnyebb őket így használni, mint minden esetben szórakozni a szövegkezelés megvalósításával.

A C++ ugyan objektumorientált, de valamiért nem került bele egy szöveg típus. Éppen ezért

körülbelül minden komolyabb C++ osztálykönyvtár rendelkezik a sajátos szöveg implementációjával.

Az Arduino osztálykönyvtárba is bekerült egy szöveg típus. Nem meglepő módon a megvalósító osztály neve String.

A típus több konstruktort tartalmaz. A konstruktorok működési leírása és példák a használatukra az alábbi táblázatban láthatóak:

Leírás	***Példa***

Létrehozás karakterek idézőjel közötti megadásával / karakter tömbökből	`String s = String(”Példa”);`
	`String z = ”Ez is helyes”;`
	`char t[] = ”Pelda 2”;`
	`String s = String(t);`

Létrehozás karakter megadásával.	`char c = 'a';`
	`String s = String(c);`
	`String k = String('k');`

Létrehozás egy másik szöveg típus felhasználásával / szövegek összefűzése	`String elso = ”Első szöveg”;` `String masodik = String(elso + ” volt”);`

Létrehozás egész szám típusokból, szöveggé konvertálja a számot.	`int valami = 3321;`
	`String s = String(valami);`
	`String t = String(2343);`

Létrehozás egész szám típusból számrendszer megadásával.	`int szam = 421;`
	`String s = Sting(szam, BIN);`
	`String t = String(2242, OCT);`

12. táblázat: A String típus konstruktorának használata

Tagfüggvények

A tagfüggvények ismertetése a string nevű változón keresztül történik. Saját programunkban a szöveg változónk nevén keresztül érhetjük el ezeket a függvényeket.

```
string.charAt(n);
```

A szöveg n-edik karakterét adja vissza. Opcionálisan a szöveg típuson alkalmazhatóak a tömb zárójelek ([ ]) is, melyek funkciója megegyezik ezen függvénnyel.

```
string.concat(string, string2);
```

A paraméterként kapott két szöveget összefűzve készít egy új szöveget. A függvény helyett opcionálisan alkalmazható szöveg típusokon az összeadás jel (+), ami azonos funkciót lát el.

```
string.endsWith(string2);
```

Megvizsgálja, hogy a szöveg típusunk a paraméterben szereplő szöveggel végződik-e. A visszatérési értéke igaz, ha a szöveg a paraméterrel végződik, hamis pedig akkor, ha nem.

```
string.equals(string2);
```

Megvizsgálja, hogy a szöveg típusunk azonos – e a paraméterben megadott szöveggel. A függvény visszatérési értéke igaz, ha a két szöveg azonos, hamis pedig akkor, ha nem. A kis-és nagybetűk jelentése nem azonos. Opcionálisan használható a függvény helyett az egyenlőség összehasonlító (==) operátor is szövegek között erre a célra.

```
string.equalsIgnoreCase(string2);
```

Működése megegyezik az equals függvényével azzal a lényeges különbséggel, hogy ezen függvény számára a kis – és nagybetűk jelentése azonos.

```
string.getBytes(buf, len);
```

A szöveg karaktereiből megadott számút egy byte tömbbe másol. Az első paraméter a buffer változó neve. Ezt referenciaként kell átadni, a második paraméter a másolandó karakterek száma.

```
string.toCharArray(buf, len);
```

Működése megegyezik a getBytes függvénnyel, azonban itt a buffer változó típusának karakter típusúnak kell lennie.

```
string.indexOf(val);
string.indexOf(val, from);
```

Megadott karakter vagy szövegrészlet első előfordulási helyének meghatározása a szövegben. A kétparaméteres változatában a második paraméter a kezdőkarakter helyét határozza meg, ahonnan a keresés indul.

```
string.lastIndexOf(val);
string.lastIndexOf(val, from);
```

Megadott karakter vagy szövegrészlet utolsó előfordulási helyének meghatározása a szövegben. A kétparaméteres változatában a második paraméter a kezdőkarakter helyét határozza meg, ahonnan a keresés indul.

```
string.length();
```

A szöveg típusban tárolt karakterek számát adja vissza, vagyis a szöveg hosszát.

```
string.replace(substring1, substring2);
```

Egy szöveg típusban cserél egy szövegrészletet egy másik szövegrészletre. Az első paraméter a cserélendő szövegrészletet határozza meg, a második paraméter pedig a csere szövegrészletet állítja be.

```
string.setCharAt(index, c);
```

Adott indexű karakter cseréje egy másik karakterre egy szöveg típusban. Az első paraméter a cserélendő karakter indexe a szövegben, a második paraméter a csere karakter.

```
string.startsWith(string2);
```

Megvizsgálja, hogy a szöveg a paraméterként megadott szövegrészlettel kezdődik e.

```
string.substring(from);
string.substring(from, to);
```

Egy szövegből szövegrészlet előállítása. Egyparaméteres változatában a szövegrészlet a paraméter által megadott indextől fog tartani a szöveg végéig. Kétparaméteres változatában a második paraméter a szövegrészlet vége indexet állítja be. Értelemszerűen ennek nagyobbnak kell lennie a kezdőindexnél.

```
string.toLowerCase();
```

A szöveg karaktereinek kisbetűsre konvertálása.

```
string.toUpperCase();
```

A szöveg karaktereinek nagybetűsre konvertálása.

```
string.trim();
```

A szöveg elején és végén található felesleges üres és szóköz karaktereket távolítja el.

Soros kommunikáció

A soros kommunikációs osztálykönyvtár minden esetben importálva van a projektünkbe, így külön nem kell azt importálnunk. Arduino Uno esetén csak egy Serial objektumunk van, de olyan eszközök esetén, amelyek több soros porttal rendelkeznek, mint a Mega, négy darab is rendelkezésre áll. Mega esetén a Serial objektum az USB-soros átalakítót reprezentálja, a Serial1, Serial2 és Serial3 objektumok meg a hardveres soros portokat.

Az összes Serial objektum ugyanazokat a függvényeket kínálja:

```
Serial.begin(speed);
```

Setup függvényben kell egyszer meghívni, engedélyezni a soros kommunikációt. Paramétere a sebességet határozza meg bit/s, vagy más néven Baud mértékegységben. Itt szabványos értékek közül választhatunk. Sok esetben 9600-at szokás megadni, mivel ez egy jó középérték sebesség és kábelhossz tekintetében.

```
Serial.end();
```

Kommunikáció lezárására szolgál. Ha újra kommunikálni szeretnénk az eszközzel, akkor ismét a Begin függvényt meg kell hívnunk.

```
Serial.available();
```

Megnézi, hogy a bejövő adat bufferban, hány byte adat várakozik feldolgozásra. A várakozó byte-ok számát adja vissza. USB soros átalakítók esetén a buffer 64 byte méretű.

```
Serial.flush();
```

Várakozás addig, amíg a kimenő adat el nincs küldve.

```
Serial.parseFloat();
```

A bejövő adat buffer karaktereit próbálja meg értelmezni lebegő pontos számnak. Előtte és utána is lehetnek betű karakterek, mivel csak addig próbálkozik a feldolgozással, amíg nem találta meg az első lebegőpontos számként is értelmezhető szöveg részletet. Visszatérési értéke a feldogozott szám float típusban.

```
Serial.parseInt();
```

Működése hasonló a parseFloat függvényhez, azonban ez a függvény egész szám típusra próbálja meg konvertálni a bejövő szöveget. Visszatérési értéke a feldolgozott szám int típusban.

```
Serial.peek();
```

Soros bufferból olvas egy byte-ot, azonban a bufferből nem távolítja el az adatot.

```
Serial.print(val);
Serial.print(val, format);
```

A paraméterként megadott értéket továbbítja sorosan szövegként. Az érték lehet karakter, szöveg, egész és lebegőpontos típusú is. A két paraméteres változat második paramétere egy formátum kód, ami egész számok esetén a számrendszer meghatározására szolgál, lebegőpontos számok esetén meg a tizedes

jegyek számát határozza meg. Egész számok esetén a következő számrendszer konstansok adhatóak meg:

- *BIN – Bináris, kettes számrendszer*
- *OCT – Oktális, nyolcas számrendszer*
- *DEC – Decimális, tízes számrendszer*
- *HEX – Hexadecimális, tizenhatos számrendszer*

```
Serial.println(val);
Serial.println(val, format);
```

Működése megegyezik a print függvénnyel, azonban ezen függvényhívás automatikusan a küldendő szöveg végére beiktatja a \n soremelés karaktert.

```
Serial.read();
```

Egy byte adatot olvas be a bejövő bufferből és ezt adja vissza int típusban. Ha nem áll rendelkezésre beolvasható adat, akkor –1-et ad vissza értékként.

```
Serial.readBytes(&buffer, length);
```

*Buffer változóba megadott mennyiségű adat beolvasása. Az első paraméter a buffer változó neve, ami **char** vagy **byte** típusú tömb lehet. A második paramétere a beolvasandó byte-ok száma. A függvény visszatérési értéke a sikeresen beolvasott byte-ok száma. Ha ez az érték 0, akkor nem volt beolvasandó adat.*

```
Serial.readBytesUntil(character, &buffer, length);
```

Hasonló a readBytes függvényhez. A függvény első paramétere egy karakter, ami ha szerepel a beolvasott adatok között, akkor a beolvasást megállítja. Ha a karakter nem szerepelt az adatok között, akkor a harmadik paraméterként megadott olvasandó byte-ok számával megegyező mennyiségű adat kerül bele a második paraméter által meghatározott bufferba.

```
Serial.setTimeout(time);
```

Időtúllépési korlát határozható meg vele a readBytes és a readBytesUntil függvényekhez. A paramétere milliszekundumban értendő. Ha be van állítva, akkor megadott ideig próbálkozik csak a mikrovezérlő az adatfogadással, így elkerülhető, hogy végtelen ciklusban ragadjon adatolvasás közben. Alapértelmezetten a beolvasó függvények időkorlátja 1mp, amennyiben a setTimeout függvénnyel nem állítunk be mást.

```
Serial.write(val)
Serial.write(str)
Serial.write(buf, len);
```

*Adat írása soros porton keresztül. Három változata is létezik, mindegyik változat visszatérési értéke a sikeresen írt byte-ok száma. Egyparaméteres változatában a paraméter lehet **char**, **byte** vagy szöveg típusú. Kétparaméteres változatában az első paraméter egy **char** vagy **byte** típusú buffer, második paramétere pedig az írandó byte-ok száma.*

Szoftveres soros kommunikáció

Szoftveres soros kommunikáció is kialakítható az Arduino bármely két digitális kimenetén, erre szolgál a SoftwareSerial osztály. Ezen osztály ugyanazokat a függvényeket biztosítja, mint a hardveres soros kommunikációs Serial objektum.

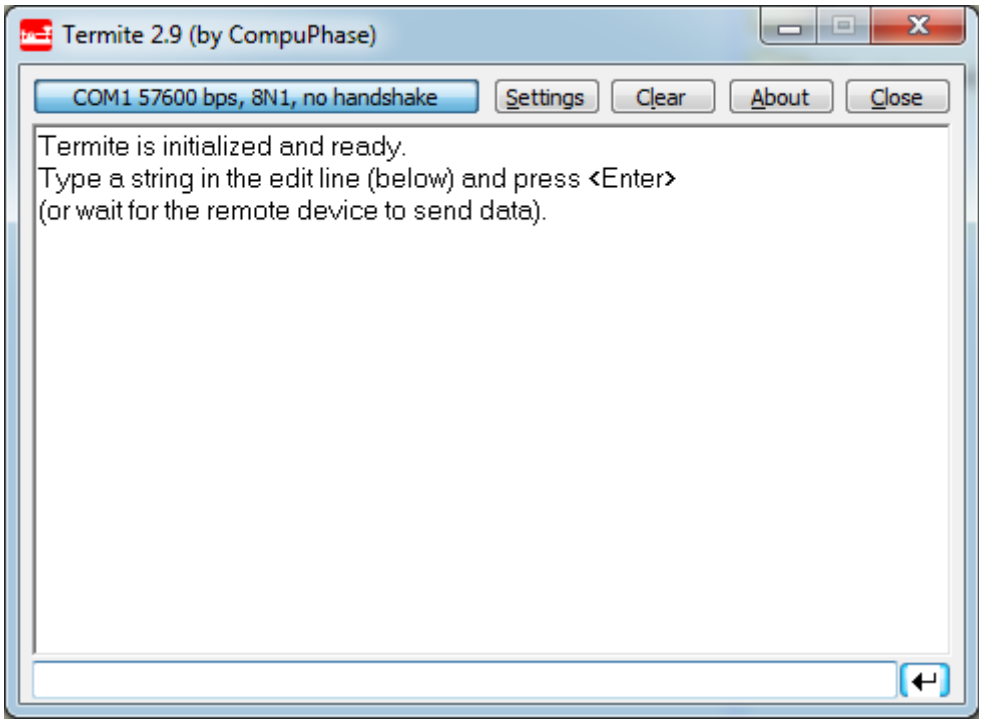
Az osztály példányosításakor a konstruktornak meg kell határozni az Rx (adatfogadás) és a Tx (adattovábbítás) lábakat.

```
SoftwareSerial Szoftversoros = SoftwareSerial(rxPin, txPin);
```

Ezután a létrehozott objektum ugyanúgy kezelhető, mint egy hardveres soros port, persze a létrehozott objektum változónevén keresztül. Használata előtt szintén meg kell hívni a Begin függvényt, ami beállítja a sebességet.

Soros kommunikáció tesztelése

Az Arduino fejlesztőkörnyezet tartalmaz egy soros kommunikáció tesztelésére alkalmas terminált, azonban ez tudás tekintetében igen szegényes. Ezért ezen alkalmazás helyett a Termite nevezetű terminál programot ajánlom. A program beszerezhető a http://www.compuphase.com/software_termite.htm címről.



21. ábra: Termite terminál program

EEPROM kezelés

Az Arduino DUE kivételével az összes Arduino modellben használt Atmel mikrovezérlő rendelkezik beépített EEPROM memóriával. Ennek kezelésére szolgál az EEPROM könyvtár, aminek segítségével byte-onként írhatunk és olvashatunk adatot a beépített EEPROM memóriából.

A beépített EEPROM memória nagyjából 100 000 írási/olvasási ciklust bír ki, továbbá az írási sebessége nem túl nagy. Egy byte beírása 3,3ms időt vesz igénybe, így az írási sebesség csupán 303byte/s körüli.

A könyvtár igen egyszerű felépítésű, csupán két függvényt biztosít.

```
EEPROM.read(address);
```

Egy byte adatot olvas ki az EEPROM memóriából. Paramétere a memóriacímet határozza meg. A függvény visszatérési értéke a memóriacímen tárolt adat. A memóriacím nem lehet negatív.

```
EEPROM.write(address, value);
```

Egy byte adat írása az EEPROM memóriába. Első paramétere a memóriacímet határozza meg, második paramétere meg a beírandó értéket.

Mivel az osztálykönyvtár EEPROM kezelő függvényei igencsak szegényes tudást biztosítanak, a felhasználónak kell azt megoldania, hogy hogyan ír be a memóriába egy 2 byte méretű egész számot, vagy mondjuk egy lebegőpontos számot.

Erre számos megoldás létezik készen az interneten is. A saját osztálykönyvtárak fejlesztése részben is szerepel egy bővített tudással rendelkező EEPROM kezelőkönyvtár, ami lehetőséget ad az összes Arduino típus EEPROM memóriában tárolására.

I²C buszrendszer kezelés

Az Arduino Due kivételével az összes Arduino modell egy I²C buszrendszerrel rendelkezik. Ez a modellben használt mikrovezérlőtől függően vagy szoftveres, vagy hardveres megoldásra támaszkodik, azonban a megvalósítástól függetlenül ugyanazon függvényekkel kezelhető a buszrendszer. A buszt megvalósító lábak az alábbi táblázatban láthatóak Arduino modellenként:

Arduino Modell	***I²C lábak***
Uno	*A4 (SDA), A5 (SCL)*
Mega2560	*20 (SDA), 21 (SCL)*
Leonardo	*2 (SDA), 3 (SCL)*
Due	*20 (SDA), 21 (SCL), SDA1, SCL1*

13. táblázat: I²C buszrendszer lábai Arduino modellenként

Az I²C buszrendszer 7 és 8 bites címekkel is működőképes, a felső 7 bit az eszköz címét határozza meg, az utolsó bit pedig azt, hogy írunk az eszközre, vagy olvasunk róla. Az Arduino osztálykönyvtára csak a felső 7 bitet használja, így ha egy mintakódban/adatlapban 8 bites címeket alkalmaznak, akkor a legalsó bitet el kell hagyni, így a cím minden esetben 0 és 127 közötti érték lesz, amit alkalmazni kell.

```
Wire.begin(address);
```

Inicializálja a buszrendszert, egyszer kell csak meghívni. A paraméter az eszköz címét határozza meg. Opcionális a megadása, amennyiben nincs megadva, akkor az eszköz a buszra master egységként csatlakozik, ha a címet meghatározzuk, akkor meg slave eszközként.

```
Wire.requestFrom(address, quantity);
Wire.requestFrom(address, quantity, stop);
```

Adat kérése a buszrendszer adott címmel rendelkező eszközéről. Az első paraméter a címet határozza meg, a második paraméter a kért byte-ok száma. Három paraméteres változatában, ha a harmadik paraméter igaz értéket vesz fel, akkor a buszrendszert felszabadítja a kérés után. Amennyiben ez hamis, akkor a buszrendszert folyamatosan foglalva tartja. Ha a harmadik paraméter nincs megadva, akkor az értékét igaznak tekinti a fordító.

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

Slave eszköz címének beállítása írási és olvasási művelet előtt. A paramétere a slave eszköz címét határozza meg.

```
Wire.endTransmission();
Wire.endTransmission(stop);
```

A beginTransmission függvény ellentettje, befejezi a kommunikációt a slave eszközzel. A paramétere megadása nem kötelező, amennyiben a paraméter igaz értékű, akkor a buszrendszert felszabadítja a függvény után. Ha meg hamis, akkor lefoglalva tartja. Ha a paramétere nem megadott, akkor az értékét igaznak veszi.

```
Wire.write(value);
Wire.write(string);
Wire.write(data, length);
```

Adat írása a slave eszköznek. A paramétere a küldendő adatot határozza meg. A paramétere lehet egy byte adat, vagy egy szöveg, ami bytesorozatként továbbítódik majd. Kétparaméteres változatában az első paraméter egy byte tömb, a második paraméter a tömbből a küldendő byte-ok száma.

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

A belső bufferben olvasásra elérhető byte-ok számát adja vissza.

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

Egy byte olvasása a buszrendszerről.

SPI buszrendszer kezelés

Az összes Arduino hardveres SPI implementációval rendelkezik, mivel a használt AVR mikrovezérlők programozó felülete is lényegében egy SPI busz. Az I²C buszrendszerhez hasonlóan a buszrendszer lábai eltérnek modellenként. A lábak elhelyezkedését az alábbi táblázat foglalja össze:

Arduino modell	MOSI	MISO	SCK	SS (szolga)	SS (mester)
Uno	11 vagy ICSP-4	12 vagy ICSP-1	13 vagy ICSP-3	10	-
Mega2560	51 vagy ICSP-4	50 vagy ICSP-1	52 vagy ICSP-3	53	-
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-
Due	ICSP-4	ICSP-1	ICSP-3	-	4, 10, 52

14. táblázat: SPI buszrendszer lábai Arduino modellenként

Az SS láb a szolga kiválasztásra szolgál, az Arduino osztálykönyvtára csak a mester üzemmódot támogatja, így a táblázatban SS címszó alatt feltüntetett lábnak minden esetben kimenetnek kell lennie, máskülönben bezavarhat a busz belső működésébe.

A táblázatban két SS oszlop található, mivel egyes Arduino modellek az SPI buszt hardveresen vagy csak szolga, vagy csak mester üzemmódban implementálják. Mivel a SS láb gyakorlatilag csak egy engedélyező jel, így bármelyik láb tetszőlegesen alkalmazható vezérlésre.

Az Arduino DUE kibővített SPI buszrendszer kezeléssel rendelkezik, ezért az itt ismertetett függvények működése némiképpen eltér. Ezért Due modell esetén érdemes a hivatalos dokumentációt áttekinteni. Az osztálykönyvtár által biztosított függvények:

```
SPI.begin();
```

Inicializálja a buszrendszert, a kommunikáció megkezdése előtt mindenképpen meg kell hívni.

```
SPI.setBitOrder(order);
```

Bitsorrend beállítása a buszrendszeren. Paramétere az LSBFIRST vagy MSBFIRST konstans lehet, ami legkisebb helyi értéken lévő bitet vagy a legmagasabb helyi értéken lévő bitet küldi el először.

```
SPI.setDataMode(mode);
```

Az SPI buszrendszer szabadságából adódóan nincs szabványosítva, hogy az órajel lefutó vagy felfutó élére kellene reagálniuk az eszközöknek, de az sincs szabványosítva, hogy az órajel akkor 0 értékű, ha 5V-on van, vagy akkor, ha 0V-on van. Ezért a megfelelő működéshez ezt be kell állítani. Erre szolgál ez a függvény. E két paraméter kombinációjából négyféle működési mód adódhat. Ezek az alábbi táblázatban láthatóak:

Konstans	Órajel nyugalmi állapota (CPOL)	Órajel él reagálás (CPHA)
SPI_MODE0	0	0 (lefutó)
SPI_MODE1	0	1 (felfutó)
SPI_MODE2	1	0 (lefutó)
SPI_MODE3	1	1 (felfutó)

15. táblázat: SPI busz működési módok

A függvény egyetlen egy paraméterébe a táblázat konstans oszlopában szereplő működési módot kell írni. Ez a használt eszköztől függ.

```
SPI.setClockDivider(divider);
```

A busz órajel sebességét határozza meg a rendszer órajel függvényében. A paraméter egy osztószám. Ezzel lesz osztva a rendszer órajel, hogy a busz a kívánt sebességen dolgozzon. Az osztó számok rögzített konstansok. A konstansokat és a buszrendszer sebességét az alábbi táblázat foglalja össze. A számított busz órajel 16Mhz-es rendszer rezgőkvarc függvényében van megadva.

Konstans	SPI Buszfrekvencia
SPI_CLOCK_DIV2	8 Mhz
SPI_CLOCK_DIV4	4 Mhz
SPI_CLOCK_DIV8	2 Mhz
SPI_CLOCK_DIV16	1 Mhz
SPI_CLOCK_DIV32	500 Khz
SPI_CLOCK_DIV64	250 Khz
SPI_CLOCK_DIV128	125 Khz

16. táblázat: SPI busz órajel konstansok és a busz sebessége

```
SPI.transfer(val);
```

Egy byte átvitele a buszon. A paramétere a küldendő byte-ot határozza meg. A függvény visszatérési értéke a buszrendszeren fogadott byte.

```
SPI.end();
```

Kommunikáció lezárása az SPI buszrendszeren. A függvény hívása után a busz csak egy új Begin függvényhívás után használható.

Karakteres LCD-k kezelése

Az Arduino osztálykönyvtára is támogatja a karakteres LCD-k kezelését, mint amilyen a hd47780-as vezérlő.

A kezelőkönyvtár a LiquidCrystal nevet kapta, a függvényei segítségével kezelhető a kijelző 4, illetve 8 bites üzemmódban is.

A korábban bemutatott hardverkezelő Arduino könyvtárakkal ellentétben ez a könyvtár behívása után automatikusan nem példányosítja magát, mivel egy vezérlőn több kijelzőt is elhelyezhetünk. A kezelés üzemmódját a kezelő osztály konstruktora állítja be. Ebből 4db-is rendelkezésre áll:

```
LiquidCrystal(rs, enable, d4, d5, d6, d7);
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7);
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7);
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7);
```

Az első konstruktor esetén az RS, Enable és a D4, D5, D6, D7 láb megadásával 4 bites, csak írható üzemmódban inicializáljuk a kijelzőt.

A második konstruktor tartalmaz egy RW paramétert is, ami az RW lábat határozza meg. Ezen konstruktor használatával írható/olvasható üzemmódban kezelhetjük a kijelzőt.

A harmadik konstruktor 8 bites csak írható üzemmódban inicializálja a kijelzőt, a negyedik pedig szintén 8 bites, de írható/olvasható üzemmódban inicializálja a kijelzőt.

Tagfüggvények

A szöveg osztályhoz hasonlóan itt is az osztály tagfüggvényei az objektum változónevén keresztül érhetőek el. A példában szereplő LCD osztályt kell lecserélni a saját változónevünkre.

```
lcd.begin(cols, rows);
```

Beállítja a kezelőkönyvtárnak a kijelző méretét. Az első paraméter a soronkénti karakterek számát, a második paraméter pedig a sorok számát határozza meg. Meghívása szükséges a helyes működéshez, mivel a kijelző eltolás és egyéb funkciók máskülönben nem működnek jól.

```
lcd.clear();
```

Törli a kijelző tartalmát, valamint a kurzort áthelyezi az első sor első karakterére.

```
lcd.home();
```

A kurzort áthelyezi az első sor első karakterére, de a kijelző tartalmát nem törli.

```
lcd.setCursor(col, row);
```

A kurzor áthelyezése adott pozícióba. Az első paraméter a soron belüli karakterszámot adja meg, a második paraméter pedig a sort.

```
lcd.write(data);
```

Egy karaktert ír ki a kijelzőre, az aktuális kurzorpozíción.

```
lcd.print(data);
lcd.print(data, BASE);
```

Adat kiírása a kijelzőre. A paraméter az adatot határozza meg, ami lehet karakter, egész szám, vagy egy szöveg. Kétparaméteres változatában a második paraméter a számrendszert határozza meg. Ekkor az első paraméternek egész számnak kell lennie. A számrendszer helyére a BIN, OCT, DEC, HEX konstansok kerülhetnek.

```
lcd.cursor();
```

A kurzor megjelenítése a kijelzőn. Mindig a következő karakter helyét jelöli alulvonással.

```
lcd.noCursor();
```

Kurzormegjelenítés kikapcsolása.

```
lcd.blink();
```

Alulvonásos kurzor helyett villogó kurzor használata. Amennyiben együtt van használva a cursor függvénnyel, akkor a függvény hatása kijelzőfüggő.

```
lcd.noBlink();
```

Villogó LCD kurzor kikapcsolása.

```
lcd.display();
```

LCD kijelző bekapcsolása. A konstruktor automatikusan bekapcsolja. Kikapcsolt állapotból való visszakapcsolásra szolgál.

```
lcd.noDisplay();
```

Kijelző kikapcsolása. Kikapcsolt üzemmódban is írni lehet a kijelzőre, de a karakterek csak bekapcsolás után fognak megjelenni.

```
lcd.scrollDisplayLeft();
```

Egy karakterrel balra tolja el a kijelző tartalmát.

```
lcd.scrollDisplayRight();
```

Egy karakterrel jobbra tolja el a kijelző tartalmát.

```
lcd.autoscroll();
```

Automatikus kijelző tartalom eltolás engedélyezése.

```
lcd.noAutoscroll();
```

Automatikus kijelző tartalom eltolás kikapcsolása.

```
lcd.createChar(num, data);
```

Egyéni, 5×8 képpontból álló karakter létrehozása. Az első paraméter a karakter számát határozza meg. A kezelőkönyvtár 8db egyedi karakter létrehozását támogatja, így az első paraméter egy 0 és 7 közötti szám kell, hogy legyen. A második paramétere egy 8 byte-ból álló tömb, ami a karaktert határozza meg soronként. Minden byte alsó 5 bitje van csak használatban.

Az egyedi karakterek a Write függvénnyel használhatóak, méghozzá úgy, hogy megadjuk paraméternek az egyedi karakterünk számát.

7. Saját osztálykönyvtárak készítése

Előbb-utóbb adódni fog, hogy saját osztálykönyvtárra lesz szükségünk az Arduino projektjeinkhez. Az osztálykönyvtárak készítése igen egyszerű feladat. Csak egy szövegszerkesztőre van szükségünk, és egy kis C++ tudásra.

Szövegszerkesztőből érdemes egy olyan programot alkalmazni, ami támogatja a szintaxis kiemelést. Windows esetén ilyen program például a Notepad++. Linux esetén a Gedit támogat szintaxis kiemelést, viszont jobban megfelel a szerkesztési célokra a Geany nevű szerkesztő.

Minden osztálykönyvtár legalább 3db fájlból áll. Kell egy .h kiterjesztésű fájl, ami a könyvtár által biztosított funkciók, osztályok definícióját tartalmazza. Ezután kell egy .cpp fájl, ami a függvények és osztályok implementációját tartalmazza. Végezetül szükségünk van egy keywords.txt nevezetű fájlra, ami az osztálykönyvtár által biztosított függvények szintaxis kiemeléséért felelős.

Az osztálykönyvtárak felépülhetnek csak függvényekből is, bár ekkor nem nevezhetőek osztálykönyvtáraknak. Egyéni használatra tökéletesek ezen könyvtárak is, azonban ha közzé szeretnénk tenni a munkánkat, akkor érdemes osztályokba szervezni a kódunkat, mert így később is átlátható marad.

Ebben a fejezetben egy egyszerű osztálykönyvtár készítését mutatom be. Az osztálykönyvtár egy virtuális portot fog megvalósítani adott lábak felett. A legtöbb mikrovezérlő lábai hardveresen portokba vannak szervezve. Nincs ez másképpen az Atmega328-as chip esetén sem (Arduino Uno és Nano vezérlője), azonban egy ilyen kevés lábú mikrovezérlő esetén ritkán fog előfordulni, hogy hardveresen bármelyik portját használjuk, mivel ezzel belepiszkálnánk az Arduino rendszer működésébe, valamint bizonyos funkciókat akkor nem tudnánk használni. Ezért a könyvtár implementál egy Port osztályt, amely segítségével adott lábakra tudunk írni 8 bit adatot egyszerre, ezáltal egyszerűbben megvalósíthatunk dolgokat.

Kezdetek

Minden osztálykönyvtárnak egy mappán belül kell elhelyezkednie. A mappa neve fogja adni az osztálykönyvtár nevét. A .h és a .cpp kiterjesztésű fájl nevének meg kell egyeznie a mappa nevével. Szóval, ha az osztálykönyvtárunk neve PortLib, akkor kell egy PortLib mappa, ami tartalmaz egy PortLib.h, PortLib.cpp és egy Keywords.txt fájlt.

A Header fájl

A header fájl tartalmazza az osztály definícióját. C++ esetén a header fájlok nem térnek el a C esetén ismertetett header felépítéstől. Ahhoz, hogy az osztálykönyvtárunk működőképes legyen, importálnunk kell bele az Arduino.h fájlt. Ez a fájl tartalmazza a korábban ismertetett függvények és típusok definícióját. A header fájl tartalma a következő lesz:

```
#ifndef PortLib_h
#define PortLib_h

#include <Arduino.h>
```

```
class Port8
{
   private:
        byte _pins[8];
        int _PORTMODE;
   public:
        Port8 (int p1, int p2, int p3, int p4, int p5, int p6, →
int p7, int p8, int PORTMODE);
        void Write(byte Data);
        byte Read();
};
#endif
```

A fájl a Port8 osztály definícióját tartalmazza. A privát változók nevét én egy alulvonás karakterrel szoktam kezdeni, így a későbbiekben is egyértelmű, hogy ez egy privát változó. A _pins tömb tárolja a porthoz tartozó fizikai lábakat, a _PORTMODE változó pedig a port irányát.

A konstruktor átvesz paraméterként 8 lábat és egy iránybeállító konstanst későbbi használatra. A Write tagfüggvény lehetővé teszi egy 8 bites változó kiírását a portra, a Read függvény pedig 8 bites adat olvasására ad lehetőséget.

Az implementáció

C++ esetén a forráskódok kiterjesztése .cpp, ahol a pp a ++ jelekre utal. Az implementációs forrásfájlunkban szintén importálnunk kell az Arduino.h fájlt a függvények végett, valamint importálnunk kell az előbb létrehozott .h fájlunkat. Továbbá érdemes importálni a pins_arduino.h fájlt is. Ez biztosítja azt, hogy minden Arduino modellen működőképes legyen a kódunk. Ennél nagyobb szerepe viszont az, hogy ha nem importáljuk be, akkor a környezet nem fogja felismerni az A0, A1, stb... analóg bemenetekre vonatkozó konstansokat, így nem fogjuk tudni használni őket digitális ki-és bemenetként.

Ezek után az osztály függvényeit megvalósíthatjuk a korábban tárgyalt osztály implementációs módszer szerint. Így a .cpp fájl tartalma a következő lesz:

```
#include "Arduino.h"
#include "PortLib.h"
#include "pins_arduino.h"

Port8::Port8(int p1, int p2, int p3, int p4, int p5, int p6, int
p7, int p8, int PORTMODE)
{
   _pins[0] = p1;
   _pins[1] = p2;
   _pins[2] = p3;
   _pins[3] = p4;
   _pins[4] = p5;
   _pins[5] = p6;
   _pins[6] = p7;
   _pins[7] = p8;
   for (int i=0; i<8; i++)
   {
        if (_pins[i] < 0) continue;
```

```
            pinMode(_pins[i], PORTMODE);
        }
        _PORTMODE = PORTMODE;
    }

    void Port8::Write(byte Data)
    {
        if (_PORTMODE == INPUT || _PORTMODE == INPUT_PULLUP) return;
        for (int i=0; i<8; i++)
        {
            if (_pins[i] < 0) continue;
            digitalWrite(_pins[i], bitRead(Data, i));
        }
    }

    byte Port8::Read()
    {
        byte ret = 0;
        if (_PORTMODE == OUTPUT) return ret;
        for (int i=0; i<8; i++)
        {
            if (_pins[i] < 0) continue;
            bitWrite(ret, i, digitalRead(_pins[i]));
        }
        return ret;
    }
```

Az osztály implementáció az Arduino könyvtár függvényeire támaszkodik. A konstruktor a lábak átvétele és tárolása után beállítja őket a megadott irányba (kimenet vagy bemenet).

A Write függvény bitenként felbontja a 8 bites bemenetet, majd a megfelelő lábakra küldi őket. Ha a port bemenetként lett konfigurálva, akkor nincs értelme adatot írni rá. Ezért még az adatküldés előtt ellenőrizve van a port iránya. Ha nem kimenet, akkor nem fog semmit csinálni a függvény.

A Read függvény a lábak állapota alapján állítja be a 8 bites visszatérési értek egyes bitjeit. Itt is ellenőrizve van a port iránya. Ha kimenet, akkor nincs értelme az olvasásnak. Ebben az esetben a függvény 0-t ad vissza értéknek.

Amennyiben egy lábat nem szeretnénk használni, akkor a konstruktorban egy nullánál kisebb számot kell írni az adott láb helyére, például –1-et.

Kulcsszavak

A keywords.txt tartalma alapján fogja az Arduino fejlesztőkörnyezet színezni a saját kódunkat. Ez egy egyszerű szöveges fájl, amiben a kettőskereszttel kezdődő sorok megjegyzést jelentenek. A fájlban soronként fel kell sorolni az osztálykönyvtár által biztosított típusokat, függvényeket és konstansok neveit.

A típusnév után tabulátorral elválasztva a KEYWORD1 kulcsszónak kell állnia. Ez jelzi a fejlesztőeszköz számára, hogy az adott szó típus. A függvényeket KEYWORD2 kulcsszóval kell ellátni, a konstansokat pedig LITERAL1 kulcsszóval.

A kulcsszavakat érdemes típus szerint szétválasztani. Amennyiben hiányzik a keywords.txt fájl, nem probléma, viszont a kód nem lesz színezve.

Az osztálykönyvtárhoz tartozó keywords.txt tartalma:

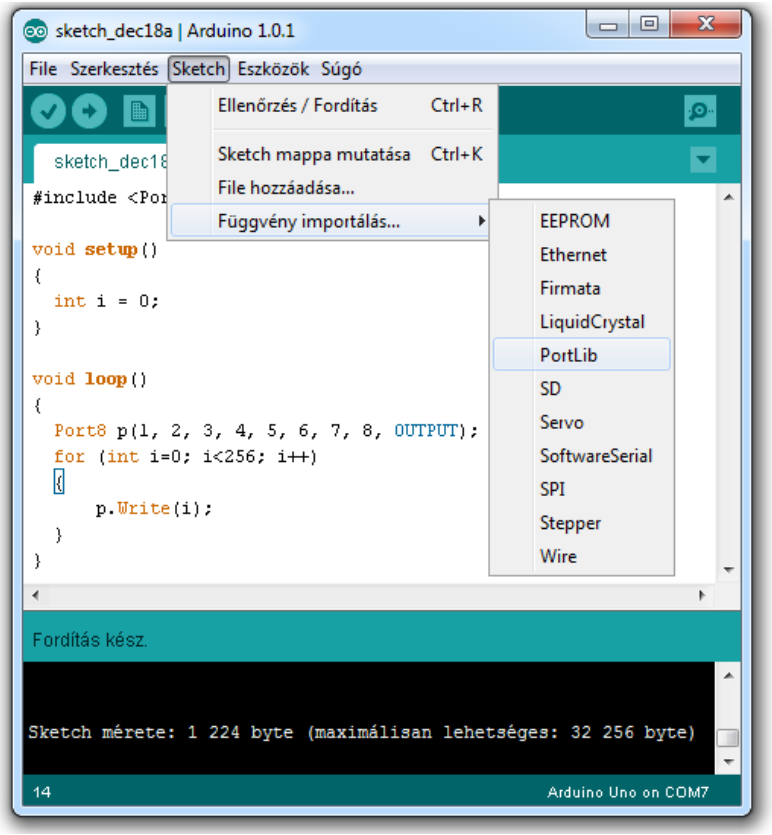
```
# Datatypes (KEYWORD1)
#######################################
Port8   KEYWORD1

# Methods and Functions (KEYWORD2)
#######################################
Port8   KEYWORD2
Write   KEYWORD2
Read    KEYWORD2

# Constants (LITERAL1)
#######################################
```

Használatba vétel

A kész osztálykönyvtárunkat két helyre is telepíthetjük. Először is az Arduino fejlesztőkörnyezet Libraries mappájába, vagy a felhasználói profilunkban található Arduino mappa Libararies almappájába. Ez Windows alatt a Dokumentumok mappában található meg.

Hivatalos ajánlás az lenne, hogy az osztálykönyvtárainkat a felhasználói profilunkba másoljuk. Azonban ha több ember is dolgozik egy adott gépen és mindenki számára elérhetővé szeretnénk tenni a könyvtárat, akkor a fejlesztőkörnyezet megfelelő mappájába másoljuk.

Mindkét esetben a kész könyvtár ugyanúgy importálható, mint egy „gyári" osztálykönyvtár.



22. ábra: A saját osztálykönyvtárunk az osztálykönyvtárak között

8. Kiegészítő szoftver Arduino fejlesztéshez: MCU Tools

Az Arduino fejlesztőkörnyezetét úgy alakították ki, hogy hobbi szintű felhasználók által is kényelmesen kezelhető legyen. Emiatt nem nevezhető a fejlesztőkörnyezet professzionális programnak.

Ezen limitáció leküzdése érdekében kezdtem el fejleszteni az MCU Tools névre keresztelt programomat. Nevezhetjük egy „univerzális svájci bicskának", amely nagymértékben megkönnyíti a mikrovezérlős programok fejlesztését, különösen akkor, ha az ember Arduino-t használ. Azonban ez nem jelenti azt, hogy más mikrovezérlős platformokhoz nem használható.

23. Ábra: Az MCU Tools program logója

A program számos eszközt/számológépet tartalmaz, felesleges lappazarlás lenne mindegyiket külön bemutatni, mivel a program jelenleg is fejlesztés alatt áll, tudása szinte naponta bővül, így ebben a fejezetben csak és kizárólag a legfontosabb szolgáltatásait, képességeit mutatom be, de mielőtt ebbe belevágnánk, tisztázzuk a minimális rendszerkövetlémenyeket

Minimális rendszerkövetelmények

- ***Windows Vista-t, Windows 7-et vagy Windows 8 futtató számítógép***
  A Windows XP nem támogatott rendszer, mivel a .NET Framework 4.5 nem támogatja.
- ***.NET Framework 4.5***
  Erre ezért van szükség, mivel a program C#-ban íródott. Amennyiben hiányzik, akkor a telepítő automatikusan feltelepíti. A telepítéshez ebben az esetben adminisztrátori/rendszergazdai jogok szükségesek.
- ***100 MB szabad hely a Windows meghajtón***
  A program a felhasználó saját könyvtárába települ, így a telepítéshez és a frissítéshez a legtöbb esetben nem szükségesek adminisztrátori/rendszergazdai jogok.

Ajánlott rendszerkövetelmények

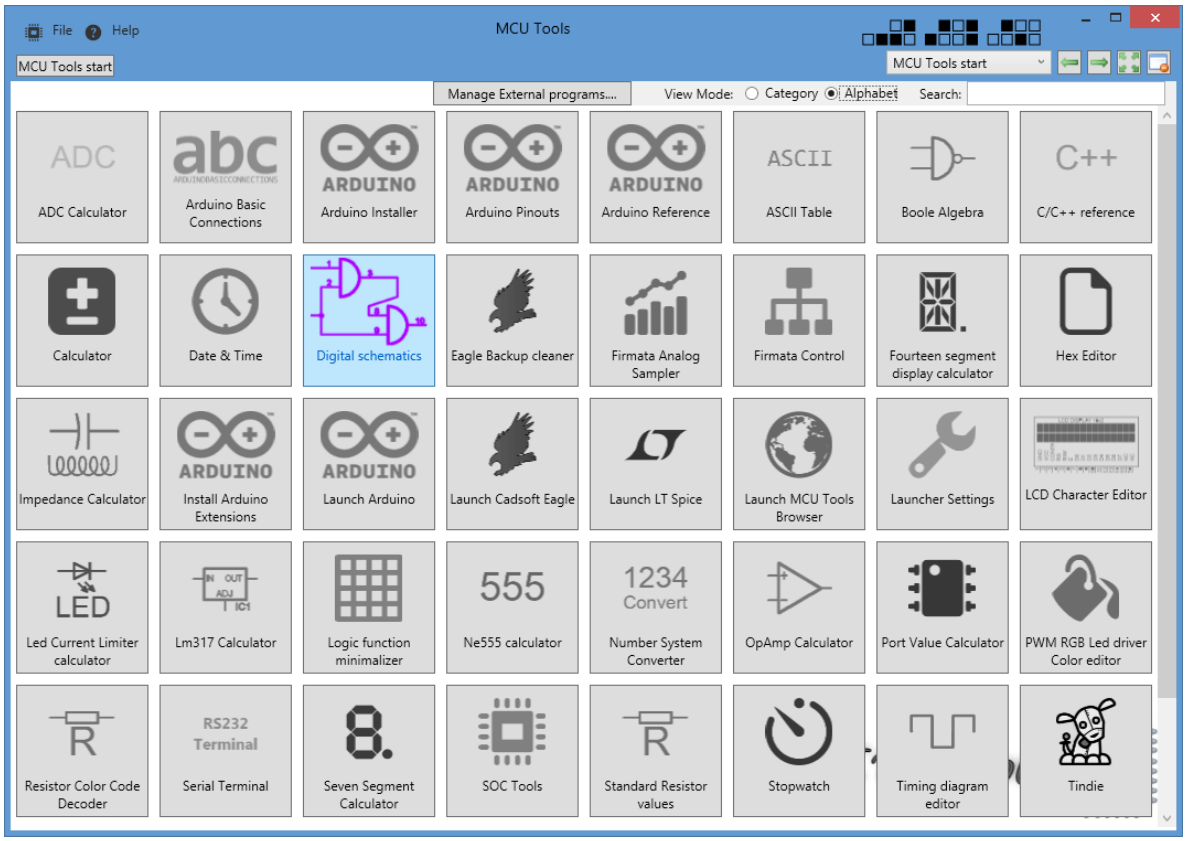
A minimális rendszerkövetelmények teljesülésével a program futni fog, azonban a program teljes értékű és zökkenőmentes működéséhez a további hardverek megléte ajánlott:

- ***Intel i szériába tartozó 2 magos processzor, AMD esetén Fx szériába tartozó processzor***
- ***DirectX 10.0-t hardveresen is támogató videovezérlő legalább 64Mb grafikus memóriával***

A programhoz mérten nagy rendszerkövetelmények alapvető oka az, hogy a legtöbb hasonló programmal szemben az MCU Tools a DirectX alapú WPF képernyőre rajzoló könyvtárakra épül. Ennek következtében a program fejlesztése sokkal gyorsabb tud lenni, illetve a modern hardverelemeket jobban kihasználja. Ezen felül, egyedi és igen sokszínű grafikus felület alkotása lehetséges a rendszerben.

Beszerezhetőség, licenc

A program nyílt forráskódú a GNU GPL v3 szerint, ami azt jelenti, hogy nyugodtan módosíthatod, terjesztheted és használhatod bármelyik gépeden. A forráskód a https://code.google.com/p/mcu-tools/source/checkout címen lelhető fel. A bináris, telepíthető változatok pedig a https://googledrive.com/host/0BzRxNHSnXbB5QjJZRG1iQWwyR00/ címről szerezhetőek be. A programmal kapcsolatos hírek, bejelentések a weboldalamon létrehozott aloldalon találhatóak meg, melynek címe: http://www.webmaster442.hu/programjaim/mcu-tools/



24. Ábra: A program főképernyője, az eszközválasztó

Eszközök

A főképernyőn az eszközök megjelenítése között alapértelmezetten az ABC sorrend van bekapcsolva, azonban választható kategória szerinti bontás is. Az eszközök közötti választásban egy beépített kereső segít. A kategóriára bontás az átláthatóságot hivatott szolgálni. Az eszközök az alábbi kategóriákra vannak bontva:

Analóg eszközök

Ebbe a kategóriába analóg elektronikával kapcsolatos számológépek kerültek. A kategória fontosabb eszközei a teljesség igénye nélkül:

- ***Feszültség és áramosztó számológép***
- ***NE555 frekvencia és kitöltési tényező számológép***
  A klasszikus 555-ös áramkör kimeneti frekvenciáját és kitöltési tényezőjét határozza meg két ellenállás és egy kondenzátor függvényében.
- ***LED előtét ellenállás méretező***
  Soros és párhuzamos kapcsolású n darab LED feszültség és áramkorlátozó ellenállását kiszámoló eszköz.
- ***Műveleti erősítős alapkapcsolások számológépe***
  A legfontosabb műveleti erősítős alapkapcsolások kimeneti feszültségét képes meghatározni a bementi feszültség és az ellenállások ismeretében.
- ***Ellenállás színkód visszafejtő***
- ***4 oszcillátoros hangfrekvenciás jelgenerátor***
  Szomszéd idegesítésére, valamint hangfrekvenciás kapcsolások teszteléséhez. A kimeneti jelalak pontosságát, minőségét nagyban befolyásolja a hangkártya minősége.

Digitális eszközök

Ezen kategóriába tartozó eszközök alkotják a program fő profilját. Itt az egyszerű kombinációs hálózatokra vonatkozó eszközöktől az egy chip-en megtalálható számítógépekig mindenféle eszköz szerepel. Szintén a teljesség igénye nélkül:

- ***ADC számológép***
  Kiszámolja, hogy az analóg-digitális átalakító bemenetére adott feszültség mekkora digitális jelnek fog megfelelni, vagy a jel szintből számol vissza feszültségértéket.
- ***Logikai függvény egyszerűsítő***
  Quine-McCluskey algoritmuson alapuló logikai függvény egyszerűsítő eszköz nyolc változóig. Négy változóig a bemenetek megadása lehetséges minterm táblázatokkal is. Öt változó felett a függvény igazságtáblázatát kell megadni. Az eszköz képes kiolvasni a legegyszerűbb függvényalakot és a legegyszerűbb hazárdmentes alakot is.
- ***Soros terminál***
  Az Arduino környezet beépített soros termináljához képest igencsak kibővített terminálfelület, amely szöveges adatok küldésén kívül lehetővé teszi a bináris adatátvitelt is. Ebben a módban a felhasználó egy hexadecimális szerkesztőfelületen viheti be a küldendő adatot. Ezen felül a beállítási lehetőségek is bővítettek.
- ***7 és 14 szegmenses kijelző érték számológépek***
  7 vagy 14 szegmenses kijelzők esetén kiszámolja a vezérlő portra küldendő értéket a szerkesztőn

bekapcsolt szegmensek függvényében. Közös anódos és közös katódos kijelzőket is támogat.

- ***Arduino lábkiosztások és alapvető kapcsolások.***
  Beépített „puska” az Arduino modellek lábkiosztásával, valamint egy adag kapcsolási rajz, amely segítségével a hardver fejlesztés könnyebbé válik. A kapcsolások és a lábkiosztások a http://www.pighixxx.com/ oldalról származnak.
- ***Port érték számológép***
  8 vagy 16 bites port esetén kiszámolja a port értékét a bekapcsolt lábak függvényében.
- ***RGB LED színkeverő***
  Három színű LED vezérlést megkönnyítő eszköz. A PWM vezérlő bitfelbontásának ismeretében kiszámolja, hogy mekkora PWM értékeket kell küldeni a piros, kék és zöld LED-ekre ahhoz, hogy a képernyőn megjelenő színt kapjuk.
- ***Időzítés diagram szerkesztő***
  Szinkron és aszinkron hálózatok tervezésénél, dokumentálásánál igen hasznos eszköz. Segítségével könnyen, gyorsan készíthetőek időzítési ábrák.

Egyéb eszközök

Ezen kategóriába azok a modulok kerültek, amelyek másik kategóriákba nem fértek be. Szintén, a teljesség igénye nélkül, az eszközök:

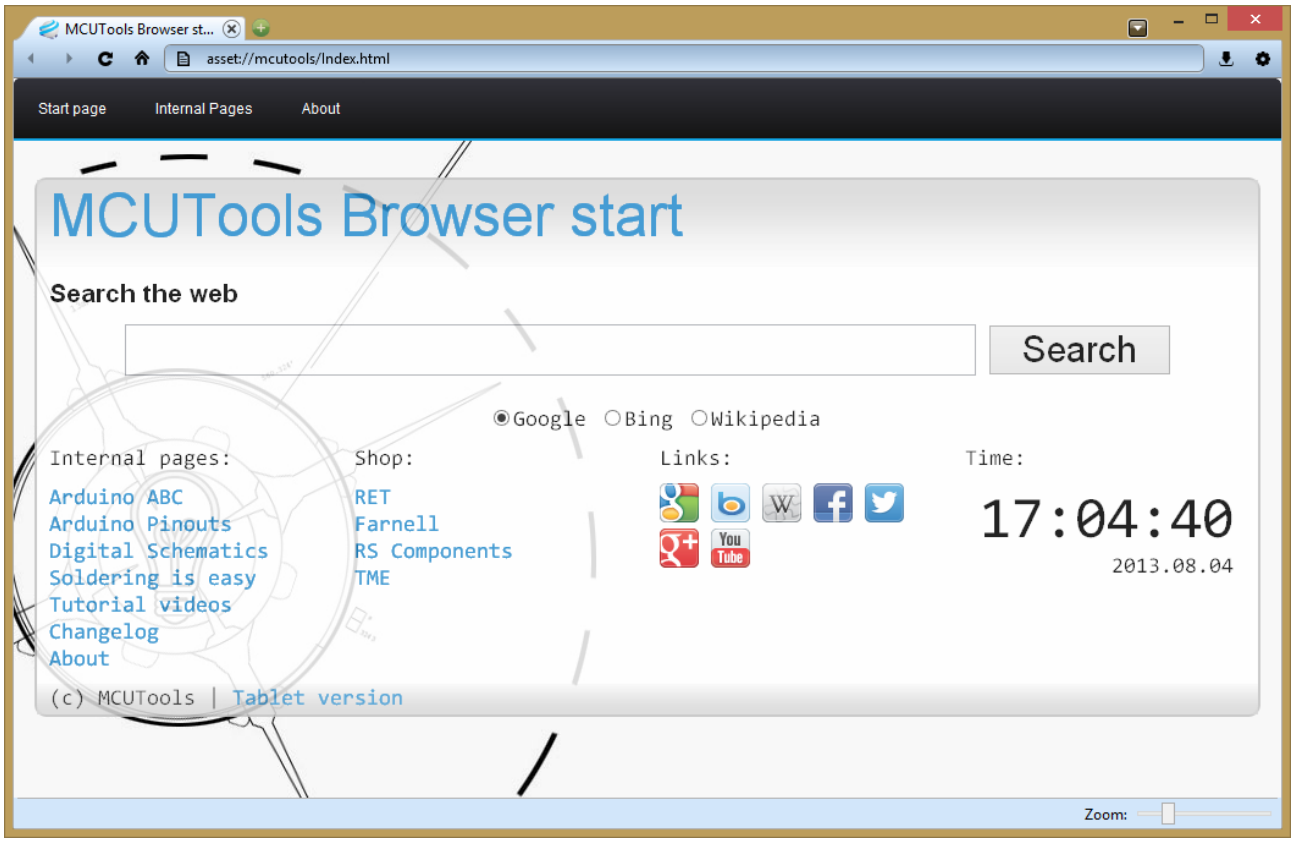
- ***Arduino környezet letöltő és telepítő***
  A hivatalos kiadási csatornát felhasználva letölti a kiválasztott verziót, majd telepíti a megadott könyvtárba a programot.
- ***Arduino kiegészítések***
  Saját készítésű gyűjtemény az interneten fellelhető legfontosabb, alapértelmezetten az Arduino környezetben nem megtalálható mikrovezérlők és egyéb hardverek támogatásával. Professzionális fejlesztéshez mindenképpen ajánlott.
- ***Eagle biztonsági másolat eltávolító***
  Az Eagle program minden egyes mentéskor készít a munkánkról egy biztonsági másolatot, ami azért jó, mert így a munka bármelyik fázisa helyreállítható. Azonban ez egy idő után igen kellemetlenné tud válni, főleg ha az ember sűrűn ment. Ez az eszköz megkeresi egy adott mappán/meghajtón belül ezeket a biztonsági másolatokat, majd a kiválasztottakat törli.
- ***Számológép***
  Egy mérnök sem élhet egy jó számológép nélkül. Sajnos a Windows beépített számológépe minden, csak nem éppen jó, ha professzionális felhasználásról beszélünk. Ezért az MCU Tools beépítetten tartalmaz egy tudományos számológépet. A számológép az IronPyhton programozási nyelv feldolgozóján alapszik, így egyéni programokkal, modulokkal bővíthető tudása.
- ***Stopperóra és óra***
  Időmérési feladatokhoz, szerintem nem kell bemutatni a funkcionalitásukat.

- ***USB videoeszköz képmegjelenítő***

  Joggal merülhet fel a kérdés, hogy mi szükség egy elektronikai programban USB videoeszközök képének megjelenítésére. A válasz egyszerű. A legtöbb USB mikroszkóp nem más, mint egy webkamera mikroszkóp optikával. A legtöbb esetben adnak ezekhez illesztő-és kezelő programot is, azonban a kezelő programok az esetek többségében nem mikroszkópok számára lettek fejlesztve. Ez a kezelő modul az egyszerű képmegjelenítésen kívül olyan szűrőkkel is rendelkezik, amelyek kifejezetten mikroszkópok számára lettek fejlesztve.

Web linkek

A web linkek kategória a programba azért került be, mert egy pár eszköz webes alapú a programban. Ez nem jelenti azonban azt, hogy használatukhoz működő internet kapcsolat szükséges. Azonban ezen eszközökhöz kellett egy böngésző. A választás a Chrome motorján alapuló Awesomium-ra esett, mivel ez egy picivel jobb kompatibilitást kínál, mintha a felhasználó által telepített „isten tudja mikor frissített" böngészőket használná a program. És ha már úgyis van böngésző a programban, akkor gondoltam teszek bele egy pár hasznos web linket.

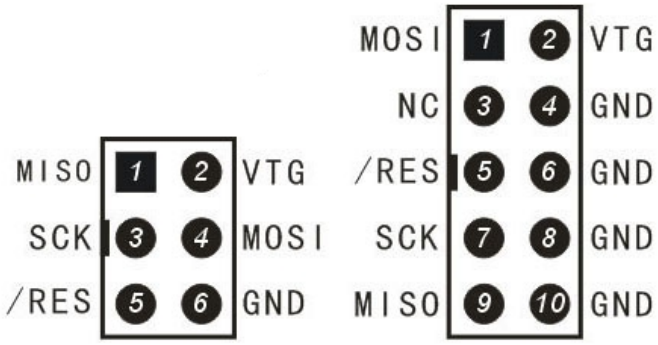


25. Ábra: A beépített böngésző kezdőlapja, amiről könnyen elérhetőek a böngészőt igénylő eszközök, illetve más webes eszközök is.

9. Projektek

1. Arduino, mint univerzális ISP programozó

Az összes Atmel mikrovezérlő AVR-ISP felületen keresztül programozható. Ez lényegében egy SPI busz plusz vezetékekkel ellátva a programozáshoz. Ez egy 2x3 lábas vagy egy 2x5 lábas csatlakozó szokott lenni. A 2x5 lábas változatot ritkán alkalmazzák. Az AVR-ISP programozási módja hasonló tulajdonságokkal rendelkezik, mint a PicKit. Úgy teszi lehetővé az eszköz programozását, hogy azt nem kell eltávolítani a céláramkörből.



26. ábra: 6 és 10 lábas AVR-ISP csatlakozók

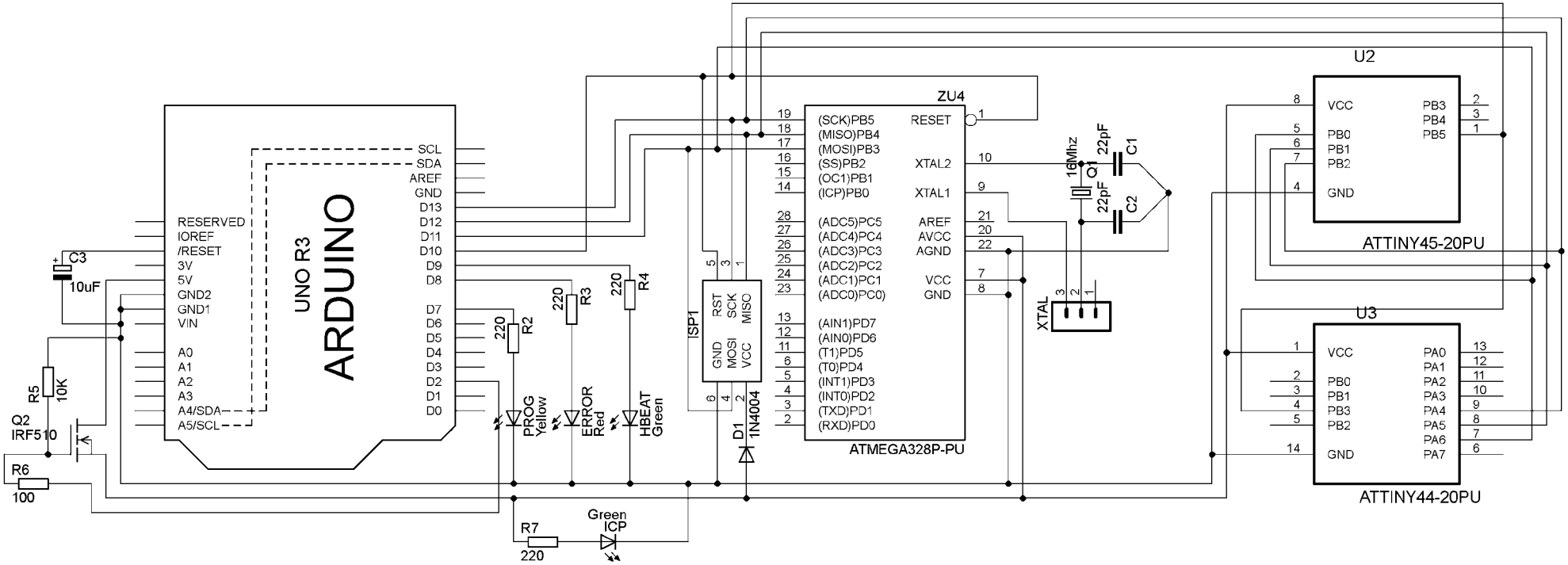
Az Arduino-ban használt mikrovezérlő a bootloader nélkül nem lenne használható az Arduino környezetben, így ha egy projektünket meg akarjuk valósítani nyomtatott áramkörben, akkor vennünk kell vezérlő chip-et (mondjuk egy Atmega328-at, amit az Uno-ban használnak). Erre először fel kell programoznunk a bootloadert, hogy használni tudjuk. A bootloader felprogramozásához szükségünk van egy ISP programozóra. ISP programozót igen sok helyen beszerezhetünk, de alkalmazhatjuk a meglévő Arduino modellünket is erre a célra.

A programozáshoz szükségünk lesz egy üres chip-re, amit fel akarunk programozni, 2db 22pF-os kondenzátorra és egy 16Mhz-es rezgőkvarcra.

A meglévő Arduino modellünk programozóvá alakításának első lépéseként fel kell programoznunk programozónak. A szoftvert nem nekünk kell megírni, mivel a környezet beépítve tartalmazza a szükséges kódot, amit az eszközre fel kell töltenünk. A szükséges programot a minták között találjuk ArduinoISP néven.

A szoftver feltöltése után a lap használható Atmel programozó eszközként bizonyos mikrovezérlőkhöz. A programozó nem támogat minden mikrovezérlőt, de az Atmega328 és pár Attiny vezérlőt igen. Ezekkel nagyon sok mindent meg tudunk valósítani.

A következő oldalon a programozó bekötése látható Atmega328/168, Attiny44/45/84/85 mikrovezérlőkhöz.



27. ábra: Arduio ISP kapcsolás több mikrovezérlőhöz

A kapcsolásból a 3db LED kihagyható, azonban ha be vannak kötve, akkor vizuálisan információt adnak a programozás állapotáról. A HBEAT LED villogással jelzi, hogy ha a programozó működőképes. A PROG nevű LED programozás közben folyamatosan világít. Ha a programozás közben hiba történt, akkor az ERR nevű LED világítani kezd.

A XTAL jelölésű jumper segítségével a külső órajelforrás lekapcsolható az Atmega328 vezérlőről. A tápellátás egyszerű szoftverből kapcsoltathatósága miatt a +5V tápfeszültséget a Q2 elnevezésű FET tranzisztor kapcsolja. Ehhez egy kicsit módosítani kellett az alapszoftvert. Ezért az Arduino környezetben megtalálható szoftverrel csak akkor fog működni a fenti kapcsolási rajz, ha a Q2 jelű tranzisztor Gate lába fixen +5V-ra van kötve. A módosított szoftver megtalálható a könyv CD mellékletén.

A Bootloader beégetése a programozandó mikrovezérlőbe az Arduino környezet Eszközök → Bootloader beéget menüpontjával lehetséges. Ezelőtt azonban be kell állítani a programozót az Eszközök->Programozó menüben. Itt az Arduino as ISP opciót kell választani. További fontos beállítás a céllap típusa, amit az Eszközök->Alappanel menüből lehet kiválasztani. ATMEGA328 esetén Arduino Uno-t kell választani.

A Bootloader beégetése után a mikrovezérlő programozható, mégpedig úgy, hogy a programozó Arduino lapból eltávolítjuk a mikrovezérlőt, majd a lap RX és TX lábát átkötjük a cél mikrovezérlő megfelelő lábaira. Mivel ez a megoldás macerás, illetve nem biztos, hogy a mikrovezérlő eltávolítható, ezért a szoftverbe bekerült egy másféle szoftverfeltöltési mód is. A kész kódunkat feltölthetjük az eszközre a megépített kapcsolással is. Csupán annyit kell tennünk, hogy a Fájl menüből a Feltöltés programozó segítségével menüpontot választjuk ki. Ezen üzemmód használatakor is kell bootloader-t a cél eszközünkbe égetni, mivel máskülönben nem működik megfelelően a kód. A programozóval való feltöltés használata után a Bootloader használhatatlanná válik, így ezután a soros feltöltés használata nem lehetséges. A Bootloader azonban nem csak ezért felelős. Felelős egy csomó konfigurációs regiszter beállításáért is, ami nélkül a lefordított programunk működésképtelen lenne.

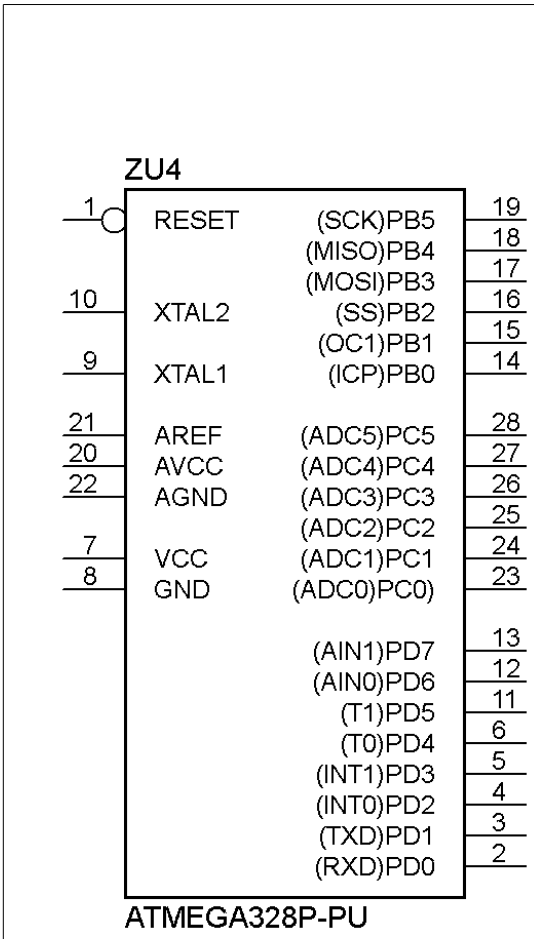
A kapcsolásból a rezgőkvarc elhagyható, programozható és használható az ATMEGA328 mikrovezérlő a beépített órajel forrásával is. Ebben az esetben az eszköz fele sebességen működik, 8Mhz-en. Ehhez külön szoftvert kell letöltenünk, ami nincs benne gyárilag az Arduino környezetben. A szükséges szoftver kiegészítés a http://arduino.cc/en/Tutorial/ArduinoToBreadboard címen szerezhető be.

A letöltött fájlok megfelelő helyre másolása után az Arduino szoftver Alappanel menüjében megjelenik egy ATmega328-on a breadboard (8 MHz internal clock) elnevezésű eszköz. Ezt kell kiválasztani, ha a belső oszcillátort szeretnénk használni.

ATmega328-as mikrovezérlő helyett alkalmazhatunk ATmega168-as mikrovezérlőt is. Ez lábkiosztásában és belső felépítésében megegyezik az ATmega328-al. A kettő között annyi különbség van, hogy a 168-as típus 16Kb programmemóriával rendelkezik. Korábbi Arduino változatokban használva volt, ezért az Arduino környezet beépített támogatást tartalmaz hozzá.

Érdemes megjegyezni, hogy az ATMega168/328 lábainak számozása nem azonos az Arduino panelen található lábszámokkal. Ez akkor lehet fontos, ha az Arduino panellel kifejlesztett rendszerünket véglegesen össze szeretnénk rakni egy saját nyomtatott áramkörön. Az alábbi táblázatban az ATMega328-

as mikrovezérlő tényleges lábkiosztása látható. Az egyes lábak mellett feltüntetésre kerültek, hogy az Arduino panelen ténylegesen melyik lábhoz vannak kivezetve.



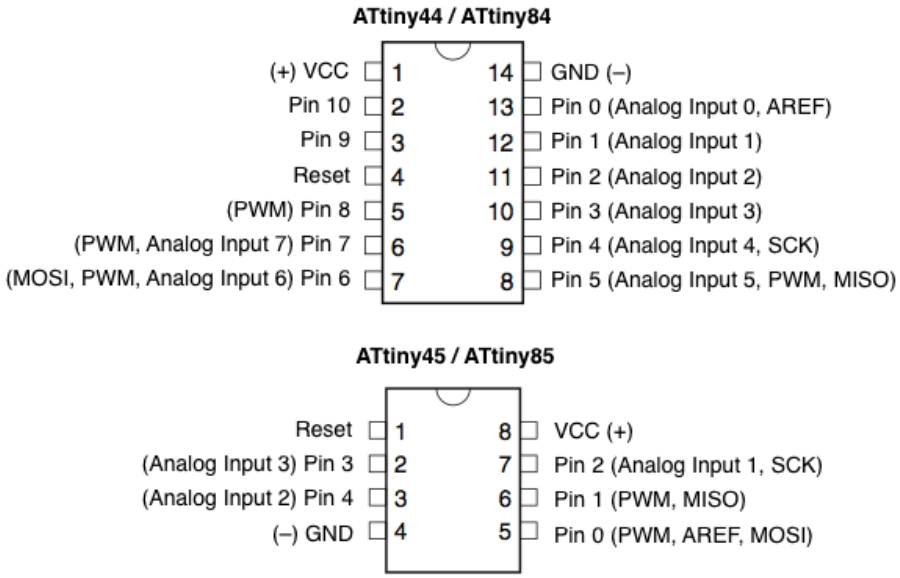
Atmega328/168 láb	*Arduino láb*
PD0	*0 (RX)*
PD1	*1 (TX)*
PD2	*2*
PD3	*3 (PWM)*
PD4	*4*
PD5	*5 (PWM)*
PD6	*6 (PWM)*
PD7	*7*
PB0	*8*
PB1	*9 (PWM)*
PB2	*10 (PWM)*
PB3	*11 (PWM)*
PB4	*12*
PB5	*13*
PC0	*A0*
PC1	*A1*
PC2	*A2*
PC3	*A3*
PC4	*A4*
PC5	*A5*

17. táblázat: ATMega328/168 fizikai lábkiosztása és az Arduino környezet lábkiosztása

Attiny vezérlők programozása

A szoftverkörnyezet gyárilag nem rendelkezik beépített Attiny támogatással, viszont felokosítható, hogy támogassa. Érdemes megemlíteni, hogy az Attiny vezérlők igen kevés memóriával rendelkeznek, így nem minden függvény és könyvtár működőképes. Továbbá a támogatás nem hivatalos Arduino fejlesztés. Ezért előfordulhat, hogy a használt szoftverkörnyezettel éppen nem lesz működőképes a kiegészítés. Az projekt weboldalán szerezhető be a szükséges szoftver, amit telepíteni kell az Arduino könyvtár mellé. A projekt weboldala a http://hlt.media.mit.edu/?p=1695 címen található.

Ezen mikrovezérlők esetén szintén eltérés van a lábak szoftveres elnevezése és fizikai elnevezésük között. Az alábbi ábrán a fizikai lábkiosztás és a szoftver által használt jelölés látható:



28. ábra: Attiny44/84/45/85 fizikai és szoftveres lábkiosztása

A fent említett projekten kívül létezik még egy projekt, melynek célja támogatás építése az Arduino környezetbe. Ez a projekt a http://code.google.com/p/arduino-tiny/ címen érhető el.

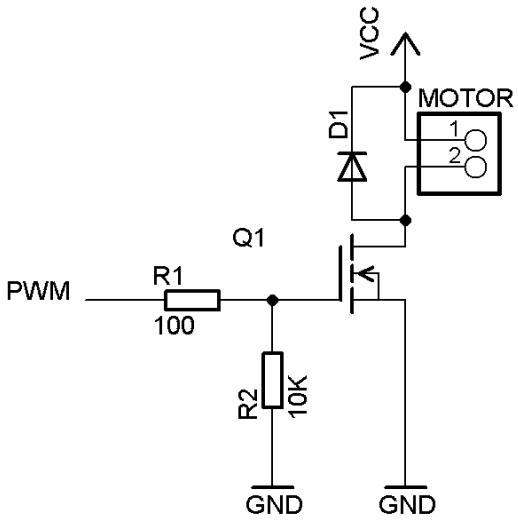
A két projekt közös jellemzője, hogy a limitált memóriaméret (4Kb vagy 8Kb) miatt nem minden Arduino könyvtár működik ezen mikrovezérlőkkel. Az alap analóg és digitális ki/bemenet kezelő függvények és a késleltetések biztosan működnek, más azonban kétséges. Ezen függvényekkel is igen sok minden megvalósítható egyébként.

2. Motorvezérlések

Egyenáramú kefés motor vezérlése

A villamos gépek közül a legegyszerűbb vezérlési szempontból az egyenáramú szénkefés motor. Ez a típusú motor a forgómozgást úgy hozza létre, hogy egy állandó mágneses térben elhelyezett tekercsen keresztül folyik át az áram. Mivel az áram mindig egy irányba folyik egyenfeszültség esetén, ezért egy mechanikus szerkezet, a kommutátor fordítja meg az áram irányát a tekercsekben az elfordulástól függően, így a forgómozgás folyamatos tud lenni. A kommutátorra az áram átvitele csúszó érintkezőkön, úgynevezett keféken keresztül történik.

A vezérlése ezen motoroknak igen egyszerű, mivel a két kivezetés polaritása határozza meg a motor forgásirányát. A sebesség szabályozása PWM moduláció segítségével lehetséges. Az alábbi ábrán egy PWM modulációt használó sebesség szabályozó kapcsolás látható.

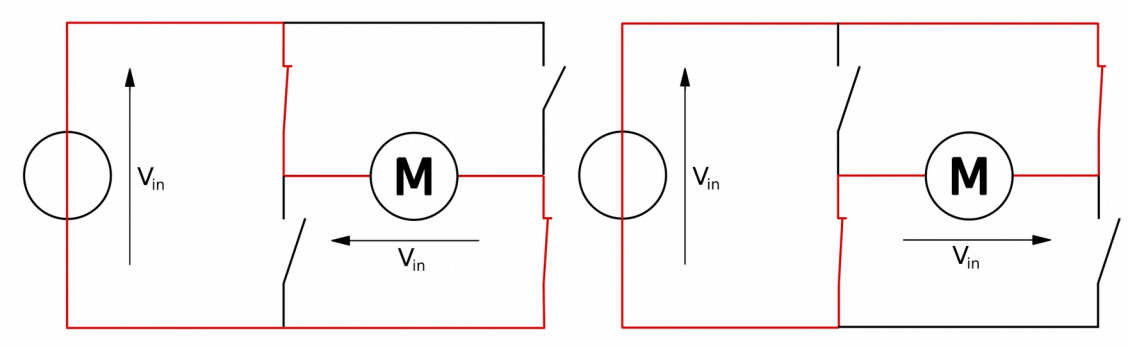


29. ábra: Egyenáramú kefés motor sebesség szabályozása PWM modulációval

A mikrovezérlő által szolgáltatott PWM jel áramerőssége szinte sosem elég egy motor meghajtásához. Ezért a PWM jelet egy tranzisztorra/FET-re kell vezetni, ami a motort fogja közvetlenül kapcsolgatni. Tranzisztor helyett érdemes FET-et alkalmazni, mivel FET-ek segítségével jobb hatásfok érhető el. A motorral párhuzamosan között, záró irányban betett dióda az induktív visszahatástól védi kapcsolást. Az induktív visszahatás a mágneses térben forgó tekercs miatt adódik. Ha a motort kikapcsoljuk, akkor nem azonnal áll meg, köszönhetően a tehetetlenségének. Ezen rövid idő alatt egyfajta generátorként működik, ami a normális áramiránnyal ellentétes irányú áramot indít meg a kapcsolásban. A védő dióda ezt az áramot vezeti vissza a motorra, ami fel fogja azt használni. A védő dióda kihagyása nem javasolt, mivel nélküle nagyon gyorsan tönkremehet az elektronika.

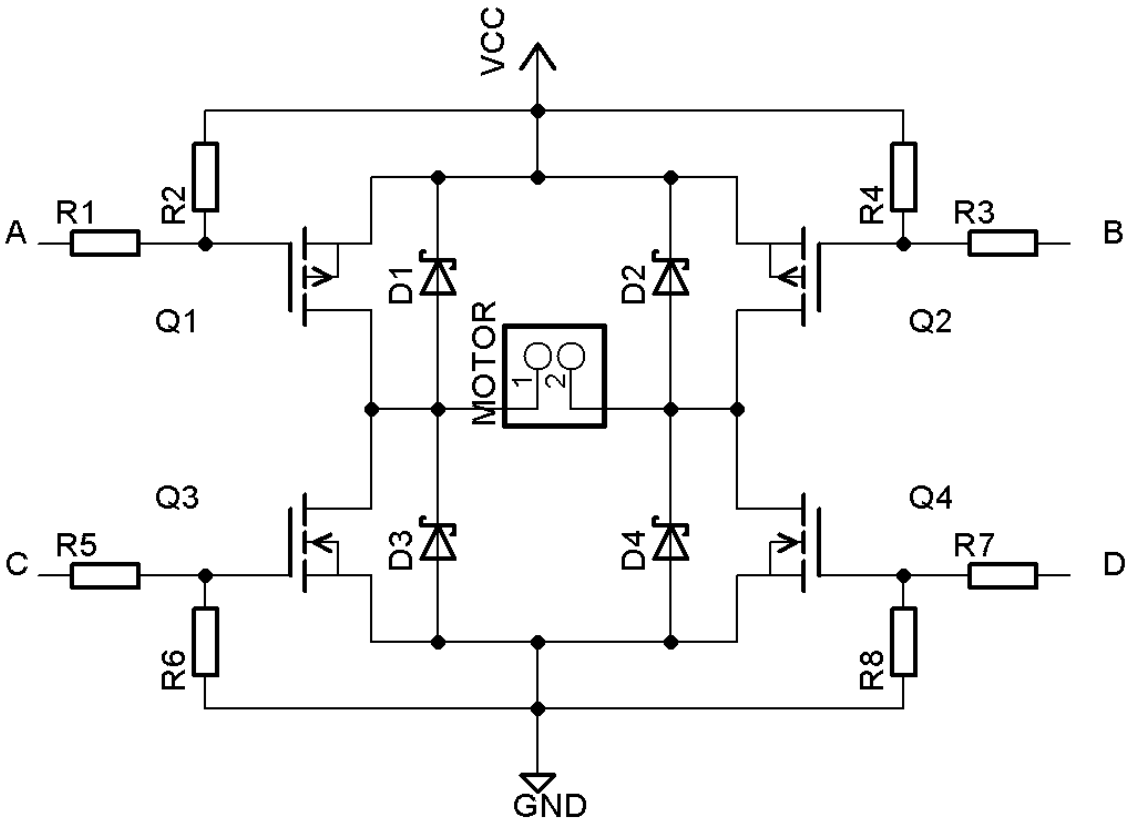
A fentebb említett vezérlés segítségével a motor egy irányba sebességszabályozható, azonban akadnak esetek, amikor a motort mindkét irányba forgatni szeretnénk, valamint sebességét is szabályozni akarunk. Ehhez egy kicsivel bonyolultabb elektronika kell, egy úgynevezett H híd.

A H híd 4db kapcsolóból áll, attól függően, hogy melyik két kapcsolópár aktív, a motoron keresztülfolyó áram iránya megváltoztatható, ami a forgásirány változását jelenti. Az alábbi ábrán a H híd két alapállapota látható.



30. ábra: A H híd két alapállapota

A H hídban szereplő kapcsolókat oldalanként nem szabad együtt működtetni, mivel akkor rövidzár keletkezik. Amennyiben a motor mindkét pontja azonos potenciálra kerül, akkor a motor rövidre lesz zárva. Ez fékezni fogja a motor forgómozgását. A kapcsolás tranzisztorok/FET-ek segítségével is megépíthető, bár itt is ajánlott a FET-ek alkalmazása a jobb hatásfok végett. Az alábbi ábrán egy négy vezérlőjeles H híd kapcsolása látható.



31. ábra: Négy vezérlőjeles H híd

A vezérlőjelek száma ebben a kapcsolásban leredukálható kettőre, ha sebességet nem kívánunk szabályozni. Ehhez az A és C vezérlő bemeneteket össze kell kötni, valamint a B és D bemeneteket is. Ebben az esetben a kapcsolás az alábbi táblázatban leírtak szerint fog működni

AC	BD	Hatás
0	0	Kikapcsolt, fékezett állapot
0	1	Forgás jobbra
1	0	Forgás balra
1	1	Kikapcsolt, fékezett állapot

18. Táblázat: Négy vezérlőjeles H híd vezérlése két vezérlőjellel

Amennyiben sebesség szabályozásra is szükségünk van, akkor négy vezérlőjelet kell alkalmaznunk és 2 vezérlőjelnek PWM jelnek kell lennie. A PWM jeleket a C és D vezérlőpontra kell kötni, míg az A és B bemenetet digitális kimenetre.

A kapcsolásban az A és B vezérlőponton P csatornás MOSFET-ek találhatóak, amelyek vezérlése inverz logikát kíván. Ezen MOSFET-ek pozitív feszültség hatására kerülnek zárt állapotba. (Ha logikai szinttel vezérelhető FET-eket alkalmazunk, akkor 5V hatására) Ezért a kapcsolásban tápfeszültségre vannak húzva alapértelmezetten. A négy vezérlőjeles működést az alábbi táblázat foglalja össze:

A	B	C	D	Hatás
0	0	0	0	Fékezett állapot
0	0	1	1	**Rövidzárlat, tiltott állapot!**
0	1	0	PWM	Forgás jobbra
1	0	PWM	0	Forgás balra
1	1	0	0	Kikapcsolt állapot

19. Táblázat: Négy vezérlőjeles H híd vezérlése négy vezérlőjellel

Léptetőmotorok

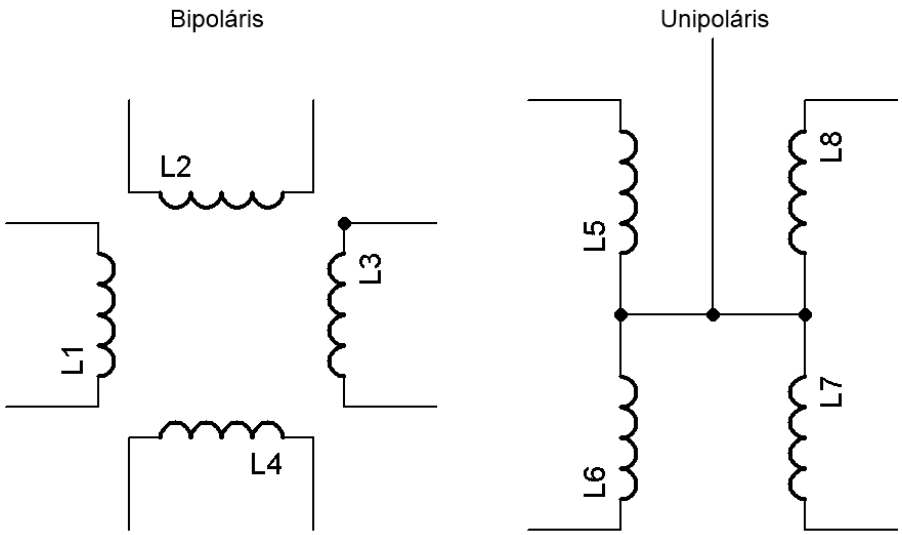
A léptetőmotorok felépítésükben igencsak különböznek a hagyományos egyenáramú motoroktól. Ezen motorok digitális vezérlésűek, jel hatására egy előre meghatározott pozícióba fordul a motor forgórésze. Olyan területen alkalmazzák ezen motorokat, ahol lényeges az, hogy pontos pozícióban álljon meg a motor. Legtöbbször nyomtatókban, és egyéb számítógép perifériákban találhatóak meg. Előnye a léptetőmotoroknak a többi hasonló megoldáshoz képest, hogy a pozícióba álláshoz nincs szükségük bonyolult vezérlési visszacsatolásra és a pozícióba állás pontossága is igen nagy, 95-99% körüli, motorfüggően.

Az előnyök mellett természetesen megvannak ezen motoroknak is a hátrányaik, ami miatt nem fogják kiváltani a hagyományos egyenáramú motorokat minden területen. Ilyen hátrány például az alacsony maximális fordulatszám, a drágaság (a hagyományos motorokhoz viszonyítva), továbbá, ha egy léptetőmotort összehasonlítunk egy azonos teljesítményű egyenáramú motorral, akkor a léptetőmotor biztos nagyobb és nehezebb lesz.

A léptetőmotor forgórésze egy fogaskerékre hasonlító állandó mágnes. A léptetőmotor egyik tekercsén áthaladó áram a legközelebbi fogat fogja magához húzni, így a motor lép egyet. Tehát vezérléshez a tekercseket sorban kapcsolgatni kell, hogy a forgó rész forogjon. Az egy teljes körbeforduláshoz szükséges lépések számát a tekercsek száma és a forgórész fogainak száma határozza meg. Általában a tekercsek száma páros szokott lenni, míg a fogak száma páratlan. Ha egy motor 4db tekerccsel rendelkezik és forgórész 25 fogból áll, akkor összesen 100 lépés kell egy teljes körbeforduláshoz. Ez 3,6°-os elfordulást jelent lépésenként.

A motorokon általában fel szokták tüntetni a lépésenkénti elfordulás vagy az egy teljes fordulat megtételéhez szükséges lépések száma.

A motorok kivezetéseitől függően kétfajta léptetőmotor létezik: unipoláris és bipoláris. Bipoláris elrendezésnél az egyes tekercsek mindkét kivezetése ki van vezetve a motorból, unipoláris elrendezés esetén pedig a tekercsek egyik vége ki van vezetve egy közös pontra. A bipoláris motorok esetén az egyes tekercsek közötti közös pont kialakítása a felhasználó feladata. Az alábbi rajz jól szemlélteti a két elrendezés közötti különbségeket.

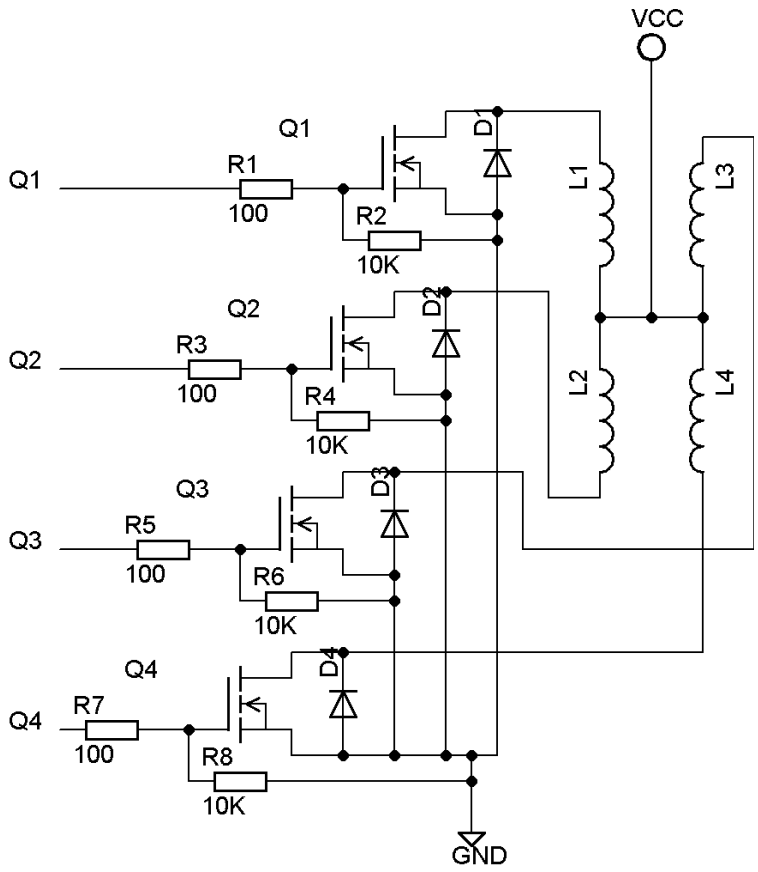


32. ábra: Léptetőmotorok tekercseinek elrendezése

A motor vezérlésénél három üzemmód is lehetséges: egyfázisú teljes léptetéses, kétfázisú teljes léptetéses és kétfázisú félléptetéses üzemmód.

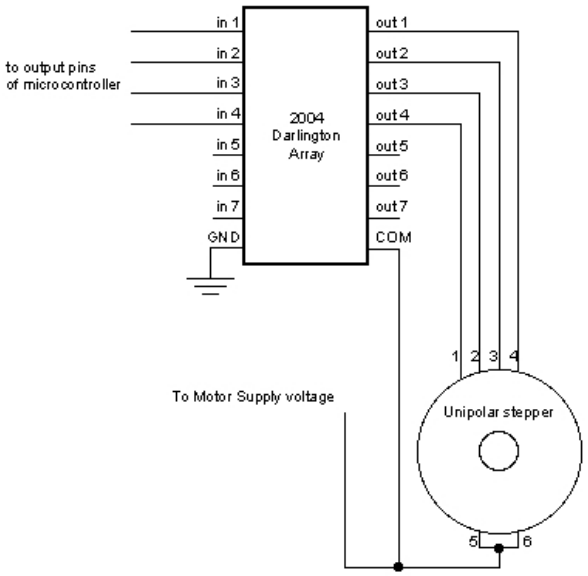
- ***Egyfázisú teljes léptetéses üzemmód***
  Egyfázisú teljes léptetéses üzemmód esetén mindig csak egy tekercsen halad át áram, a motorok általában erre a működésre vannak tervezve.
- ***Kétfázisú teljes léptetéses üzemmód***
  Ezen vezérléskor egyszerre két szomszédos tekercsen halad át áram, így a motor majdnem kétszer akkora teljesítmény leadására képes, de cserébe jobban is fog melegedni. Ezért ez a vezérlési mód csak nagyon rövid ideig ajánlott.
- ***Kétfázisú félléptetéses üzemmód***
  Ezen üzemmód az egyfázisú és a kétfázisú léptetés keveréke. Egy tekercs kapcsolása után az adott tekercsnek és a mellette lévő tekercsnek adunk feszültséget egyszerre. Így a lépésszám megduplázható, pontosabb lesz a motor, cserébe viszont egy teljes fordulat több időt fog igénybe venni.

A motor és a mikrovezérlő közötti illesztésre készíthetünk saját áramkört is, de használhatunk integrált áramkört is. A legtöbb kis teljesítményű motor vezérléséhez elég lesz egy integrált áramkör, de ha nagy teljesítményű motort szeretnénk vezérelni, akkor érdemes saját illesztő elektronikát építeni. Egy ilyen vezérlő elektronika kapcsolási rajza látható az alábbi ábrán.



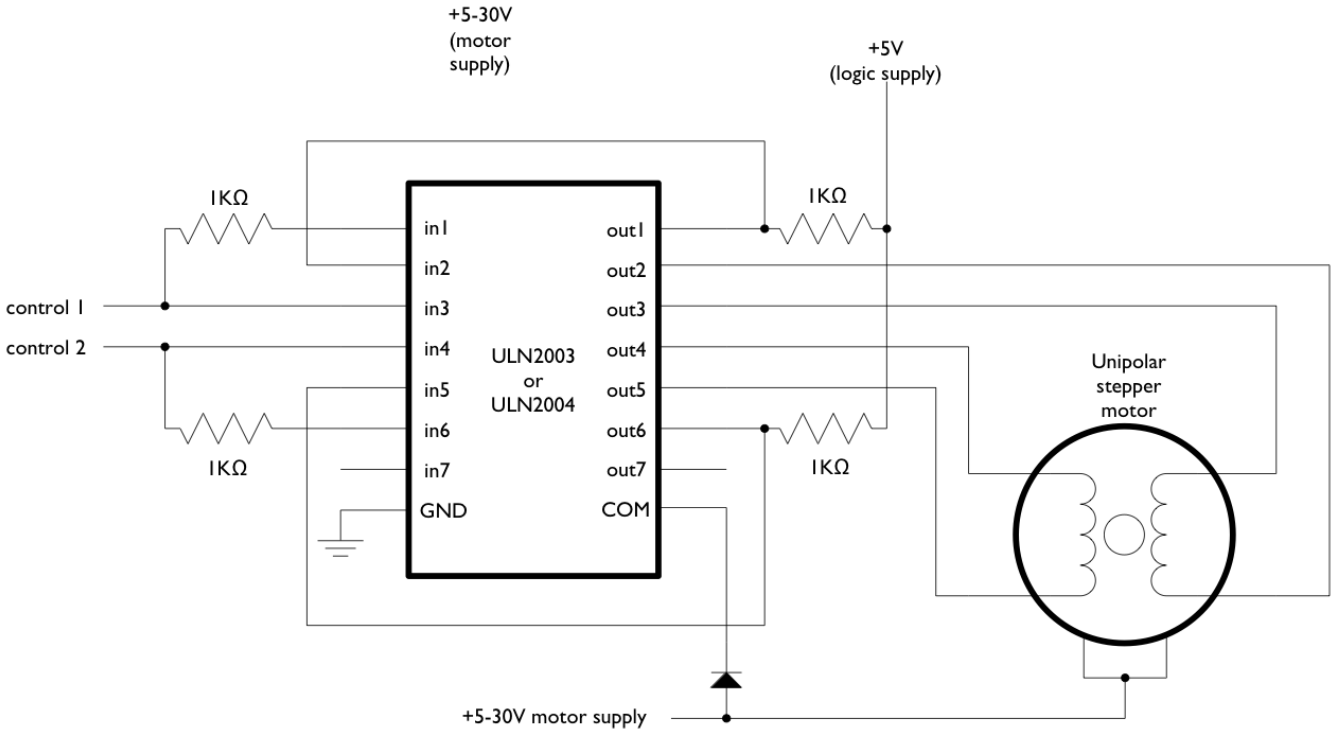
33. ábra: Léptetőmotor illesztése mikrovezérlőre

Kisebb motorokhoz készen is kapható egy ilyen áramkör, az ULN2004, ami tranzisztoros belső felépítésű, ezáltal rosszabb hatásfokot fog biztosítani. Az Arduino oldalon ezt az áramkört ajánlják unipoláris motorokhoz. Az alábbi ábrán látható egy unipoláris motor bekötési rajza. A rajz a hivatalos Arduino oldalról származik.



34. ábra: Unipoláris motor bekötése integrált áramkörrel négy kivezetéssel

Bipoláris motor esetén a tekercseket külsőleg unipoláris elrendezésre kell kötni, vagy használhatunk kifejezetten unipoláris motorokra tervezett meghajtó áramköröket is. A fejlesztőkörnyezettel szállított léptető motorvezérlő képes működni két kivezetéssel és négy kivezetéssel is. Négy kivezetés esetén a fentebb említett kapcsolásokat kell alkalmazni, két kivezetés esetén azonban a meghajtó áramkör némileg módosul. Az alábbi ábrán a hivatalos oldalon ajánlott kapcsolás látható két kivezetés esetén, unipoláris motorokhoz.



35. ábra: Unipoláris léptetőmotor vezérlése két kimenettel

Kezelőkönyvtár

A kezelőkönyvtár a Stepper nevet viseli, csak importálás után használható. Két konstruktora létezik. Ezek:

```
Stepper(steps, pin1, pin2);
Stepper(steps, pin1, pin2, pin3, pin4);
```

Mindkét konstruktor esetén az első paraméter a motor teljes körbefordításához szükséges lépések számát jelöli, az ezt követő paraméterek pedig a vezérléshez használt lábakat. A könyvtár két függvényt biztosít ezután a motor vezérléshez.

```
setSpeed(rpms);
```

Ez a függvény adott fordulatszám beállítására szolgál. Paramétere a kívánt percenkénti fordulatszám.

```
step(steps);
```

N darab lépéssel elforgatja a motor tengelyét, majd megállítja a motort.

Szervomotorok

A szervomotorok két fő részből állnak. Egy hagyományos szénkefés motorból (nagyobb teljesítményű motorok esetén kefe nélkülit szoktak alkalmazni), és egy vezérlőből. A vezérlő tartalmaz egy abszolút jeladót, ami minden esetben meg tudja mondani a tengely aktuális pozícióját, így ezen motorok elfordulása is igen pontosan szabályozható.

A szervomotorok belső felépítése bonyolultabb a léptetőmotoroknál, azonban sokkal nagyobb teljesítményre képesek, ezért főként olyan helyen alkalmazzák ezen motorokat, ahol fontos a teljesítmény.

Vezérlési szempontból egy léptetőmotornak három kivezetése szokott lenni. Egy pozitív tápfeszültség, egy földpont és egy vezérlő bemenet. A vezérlő bemeneten az elfordulás PWM jelek sorozatával adható meg. Az Uno nem minden lábán képes PWM jel előállításra hardveresen, így a beépített Szervo kezelő osztálykönyvtár szoftveresen oldja meg a vezérlést, némi hardveres támogatással. Ez azt jelenti, hogy a szervomotorunk bármelyik lábára csatlakoztatható, viszont a 9-es és 10-es lábon nem lesz használható az analogWrite függvény, amennyiben a szervokönyvtárat használjuk.

A legtöbb kereskedelmi forgalomban lévő hobbi szervomotor 5V feszültségről üzemel, azonban az Arduino +5V kimenete nem biztos, hogy tud akkora áramot szolgáltatni, ami kell a motor meghajtásához. Ezért érdemes ezen motorokat külön áramforrásról meghajtani. Külső áramforrás használata esetén nem szabad elfelejteni a közös földelés kialakítását, ami abból áll, hogy össze kell kötni a külső tápegység földpontját az Arduino földpontjával. Ha ez elmarad, akkor a vezérlő jeleknek nincs közös viszonyítási alapja, ami be fog zavarni a vezérlésbe. Rosszabb esetben nem is fog működni a vezérlés.

Érdemes megemlíteni, hogy a legtöbb kereskedelmi forgalomban kapható szervomotor csupán 0 és 180 fok közötti elfordulásra képes, ezért az Arduino kezelő könyvtár is 0 és 180 fok között elforduló motorok kezelését támogatja.

Kezelőkönyvtár

A szervomotor kezelőkönyvtára szintén nincs importálva alapértelmezetten. Importálás után a Servo típusú osztály használható objektumok létrehozására. Az osztálynak nincs paraméterrel ellátott konstruktora, így létrehozáskor nem kell megadni a vezérlő lábat. A vezérlő láb megadásra a következő függvények szolgálnak.

```
servo.attach(pin);
servo.attach(pin, min, max);
```

A függvény egyparaméteres változatában a paraméter a vezérlő láb. A három paraméteres változatban is az első paraméter a lábat határozza meg. Második paramétere a 0° elforduláshoz rendelt impulzus szélesség, a harmadik paraméter pedig a 180° elforduláshoz rendelt impulzus szélesség. Az impulzus szélességek mikroszekundumban értendőek. Az egyparaméteres függvényváltozat 544 mikroszekundumot rendel 0 fokhoz, és 2400 mikroszekundumot a 180 fokhoz. A három paraméteres változat használatára csak a szabványtól nagyon eltérő motorok esetén lesz szükség.

```
servo.attached();
```

Ellenőrzi, hogy az objektum hozzá lett-e rendelve egy lábhoz, vagy sem. Ha igen, akkor logikai igaz értékkel tér vissza a függvény, ellenkező esetben pedig logikai hamis értékkel.

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

A szervo objektumot lecsatolja a hozzárendelt lábról. A 9-es és 10-es kimenetek Uno esetén csak akkor lesznek használhatóak analóg érték írására, ha az összes szervo objektum le lett csatlakoztatva.

```
servo.read();
```

A szervomotor helyzetét olvassa ki. Visszatérési értéke a motor tengelyének elfordulása fokokban. Egy 0 és 180 közötti szám.

```
servo.write(angle);
```

Adott fokkal elfordítja a motor tengelyét. A paraméter fokban határozza meg az elfordulást.

```
servo.writeMicroseconds(uS);
```

Az elfordulást kiváltó vezérlőjel meghatározása mikroszekundumban. Nem éppen szabványos motorok esetén a 0 fokhoz 700 körüli érték tartozik, míg a 180 fokhoz 2300 körüli érték. Hasznos funkció a motor vezérlésének jobb kiismeréséhez és pontos motorvezérléshez.

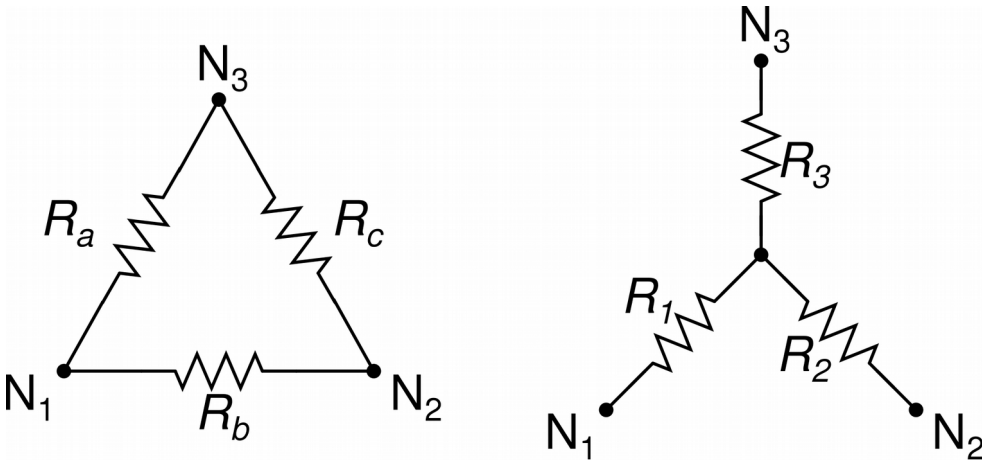
Kefe nélküli egyenáramú motorok (BDLC)

Ezen motorok esetén a motor forgórésze egy állandó mágnes, az állórész pedig a motor tekercselése. Ahhoz, hogy a motor tengelye forogjon a tengely körül, létre kell hozni egy forgó mágneses mezőt a megfelelő tekercsek időbeli kapcsolgatásával. Az ilyen motorok előnye a kefés motorokhoz képest az, hogy jóval nagyobb nyomaték leadására képesek, valamint az egyetlen kopó alkatrész a motorban csak a csapágyazás, míg a kefés motoroknál a kefék és a kommutátor is kopik. Tehát ezen motorok tartósabbak.

Hátrányuk viszont az, hogy vezérlő elektronika nélkül nem képesek működni, mivel a tekercseket a forgórész helyzetétől függően kapcsolgatni kell. A vezérlő elektronika egy mikrovezérlőt és egy teljesítmény fokozatot fog tartalmazni. Az elektronika konyultsága miatt ezzel részletesen ezen könyv keretein belül nem foglalkozom. Ezen motorokat főleg modellautókban és repülőkben alkalmazzák kiváló teljesítmény/súly arányuk végett, de ilyen motorokat találunk a merevlemezekben és az optikai meghajtókban is.

Mivel a modellezők körében ezen motorok igencsak elterjedtek, így bármelyik modellező boltban kapunk egy ilyen motorhoz vezérlő elektronikát. Az elektronika bonyolultságától függően az ára változhat.

Egy kefe nélküli motornak általában három kivezetése van, ritka esetben hat. Ennek oka az, hogy az állórész három tekercsszakaszra van felosztva. Ha három kivezetésünk van, akkor a motor belsejében a tekercsek csillag, vagy delta kapcsolás szerint vannak fixen bekötve. Ezen nem változtathatunk. A két tekercs kapcsolás az alábbi ábrán látható.



36. ábra: Delta és csillag tekercs kapcsolások.

A legtöbb kefe nélküli motor esetén a csillag kapcsolást szokták alkalmazni. A tekercsek alapértelmezett elrendezése a motor teljesítményétől függ leginkább. Csillag kapcsolású motorok esetén találkozhatunk négy kivezetéssel is. Ebben az esetben a negyedik kivezetés a csillagpont. Hat kivezetéssel rendelkező motorok esetén a motort köthetjük csillag kapcsolás szerint, vagy deltába is.

Csillag kapcsolás esetén a motor nyomatéka nagy, alacsony fordulatszám esetén is, viszont a maximálisan elérhető fordulatszám nem nagy a delta kapcsoláshoz viszonyítva. Delta kapcsolás esetén a motor kisebb nyomaték leadására képes alacsony fordulatszám mellett, de a maximális fordulatszáma

nagy. A két kapcsolás előnyeit háromfázisú motorok esetén egy csillag-delta váltókapcsolóval szokták kihasználni, méghozzá úgy, hogy a motort csillag kapcsolásban indítják el, majd mikor felpörgött, akkor váltják át delta kapcsolásba. Így a motor terhelten is el fog tudni indulni.

Kefe nélküli egyenáramú motorok esetén ezt nem igen szokták alkalmazni. Azonban delta kapcsolás esetén meg van az esélye annak, hogy terhelés mellett a motor szaggatottan, vagy nehezen indul el. Főként ezért alkalmazzák a csillag kapcsolást fixen.

A vezérlő elektronikának a motor felé minden esetben három kimenete lesz, amit a három tekercs kivezetésre kell bekötni. Bemeneti oldalról is szintén három kivezetése lesz a vezérlőnek. Egy pozitív tápfeszültség, egy földelés, és egy vezérlő bemenet. A vezérlő bemenet működése megegyezik a szervomotoroknál tárgyalt vezérléssel. Annyi a különbség, hogy ezen motorok teljesen körbeforduló szervomotorokként foghatók fel.

Ez vezérlésben annyit jelent, hogy a 0 fok egyik irányban fogja jelenteni a teljes sebességet, a 180 fok pedig másik irányban a teljes sebességet. 90 fok környékén pedig a motor fékezett állapotban lesz. Szintén vezérléstől függően a minimum és maximum pozícióhoz szükséges impulzusszélességgel el kell játszani a tökéletesen pontos vezérlés miatt.

Mivel a kefe nélküli egyenáramú motorok 5V-nál nagyobb feszültségről üzemelnek a legtöbb vezérlőáramkör tartalmaz egy úgynevezett elem kiiktató áramkört (Battery elliminator) is, ami arra szolgál, hogy a vezérlésnek, ami tipikusan 5V-ról működik, ne kelljen külön akkumulátort beszerelni.

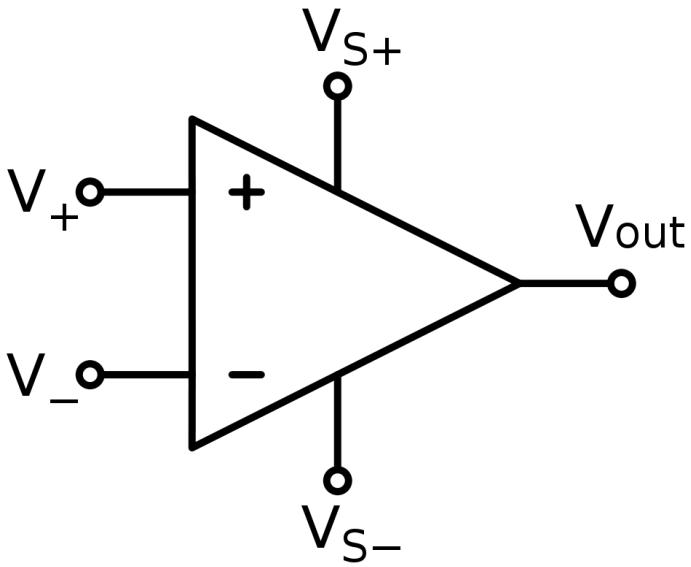
3. Műveleti erősítők és alapkapcsolásaik

A műveleti erősítők analóg áramkörök, azonban széles körben elterjedtek és mikrovezérlős rendszerfejles során bizonyos szenzortípusok illesztésénél alkalmazzák őket. De mi az a műveleti erősítő? A műveleti erősítő egy pár tucat tranzisztorból álló integrált áramkör. Széles körben elterjedt a rengeteg alkalmazási lehetősége és olcsósága miatt.

A műveleti erősítőt tartalmazó áramkör tulajdonságait elsősorban a hozzá épített külső alkatrészek határozzák meg. Leginkább feszültségerősítési célokra használják őket. Egy műveleti erősítő ideális esetben az alábbi tulajdonságokkal rendelkezik:

- *Bemeneti oldalán végtelen nagy impedanciával (váltakozó áramú ellenállás) rendelkezik, tehát a bemenet jelszintje nem tolja el a kimenet feszültségszintjét*
- *Kimeneti oldalán nem rendelkezik impedanciával*
- *A tápfeszültség zavarait teljes mértékben kiküszöböli*
- *Végtelen nagy frekvenciatartományban képes működni zajmentesen, a kimeneti jelalak torzítása nélkül.*

A valóságban persze ideális műveleti erősítő nem létezik, de a gyártók igyekeznek olyan erősítőket gyártani, amelyek az ideálishoz a leginkább hasonlítanak. Természetesen egy adott típusú műveleti erősítő nem lesz minden célra alkalmas, ezért az erősítők katalógus adatlapja alapján érdemes tájékozódnunk, hogy az adott célra meg fog-e nekünk felelni. Egy műveleti erősítő rajzjele az alábbi ábrán látható. Az egyes kivezetések jelentései az ábra utáni táblázatban találhatóak meg



37. ábra: Egy műveleti erősítő rajzjele

Kivezetés jele	Funkciója
V+	Nem invertáló bemenet
V-	Invertáló bemenet
Vs+	Pozitív tápfeszültség
Vs-	Negatív tápfeszültség
Vout	Kimeneti kivezetés

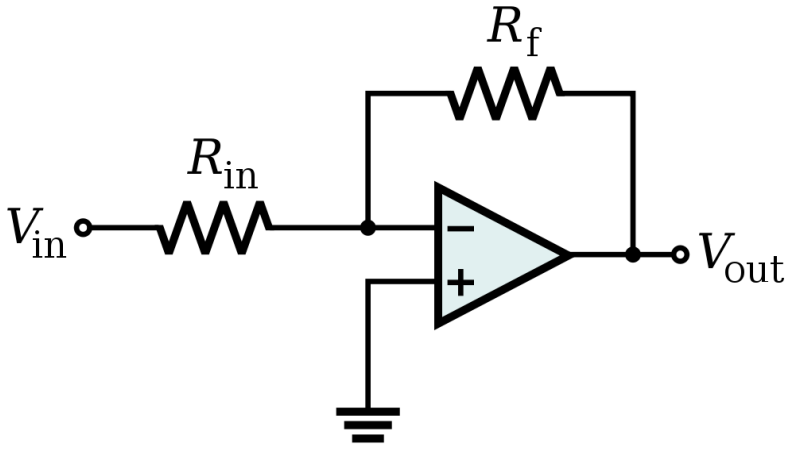
20. táblázat: Műveleti erősítő kivezetései

Minden műveleti erősítő két bemenettel rendelkezik. Egy invertáló és egy nem invertáló bemenettel. A nem invertáló bemenet tulajdonsága, hogy a kimeneten az ezen a bemeneten át érkező jel azonos polaritással és azonos fázishelyzetben jelenik meg. Az invertáló bemenet tulajdonsága, hogy erre a bemenetre érkező feszültség invertált polaritással és 180 fok fáziseltolással jelenik meg a kimeneten.

A legtöbb műveleti erősítőnek működéshez negatív és pozitív tápfeszültségre is szüksége van a tökéletes váltakozó áramú erősített jel előállításához. Amennyiben csak pozitív tápfeszültséget kapnak (úgy, hogy a negatív tápfeszültség lába földponton van), akkor a kimeneten a bemenő jel a pozitív feszültség irányába eltolva jelenik meg.

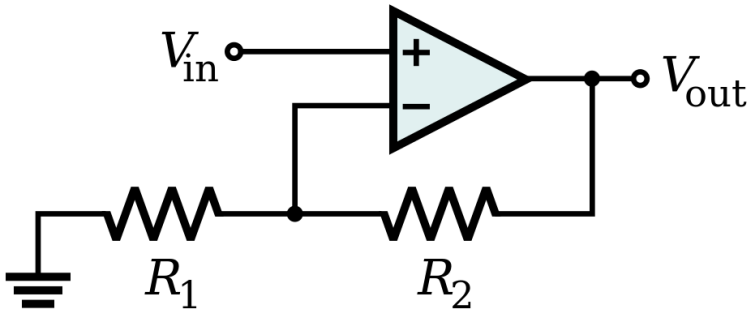
A műveleti erősítőknek számos kapcsolása létezik, az alábbiakban csak a legfontosabb alapkapcsolásokkal ismerkedünk meg. Ezeket igen gyakran alkalmazzák. A kapcsolások leírása során használt képletek feltételezik, hogy a használt erősítő erősítése végtelen nagy, és a feldolgozandó jel frekvenciájával elboldogul az erősítő. Az összes alapkapcsolás esetén a kimeneti feszültség van megadva a bemeneti feszültség és az áramköri elemek függvényében.

Invertáló alapkapcsolás



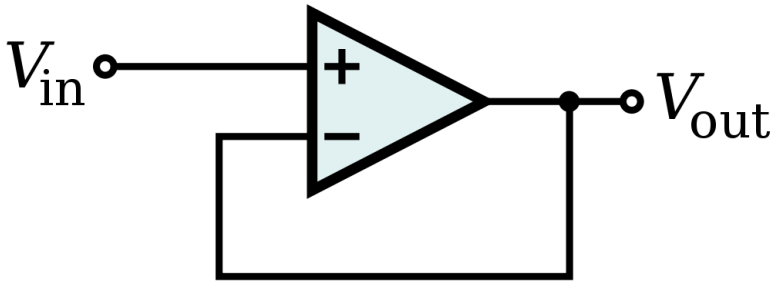
$$U_{ki} = -\frac{R_f}{R_{in}} \cdot U_{be}$$

Nem invertáló alapkapcsolás



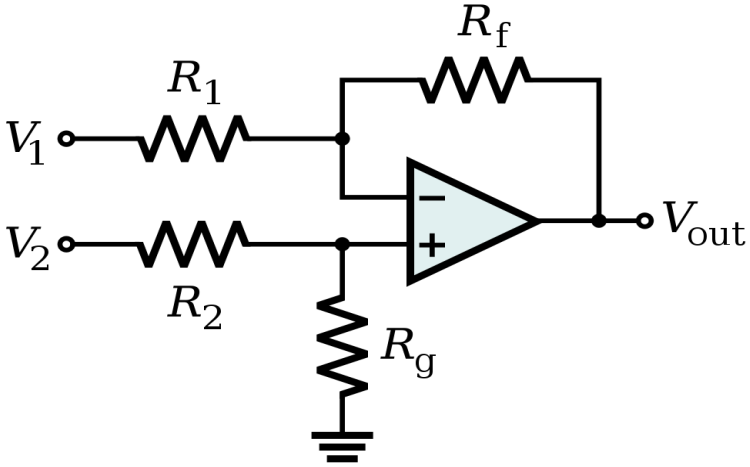
$$U_{ki} = U_{be} \cdot \left(1 + \frac{R_2}{R_1}\right)$$

Követő erősítő



$$U_{ki} = U_{be}$$

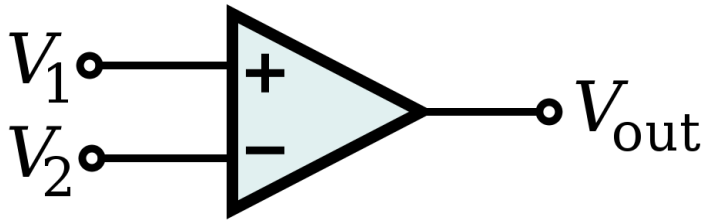
Differenciál erősítő, kivonó áramkör



$$U_{ki} = \frac{(R_f + R_1) \cdot R_g}{(R_g + R_2) \cdot R_1} \cdot U_2 - \frac{R_f}{R_1} \cdot U_1$$

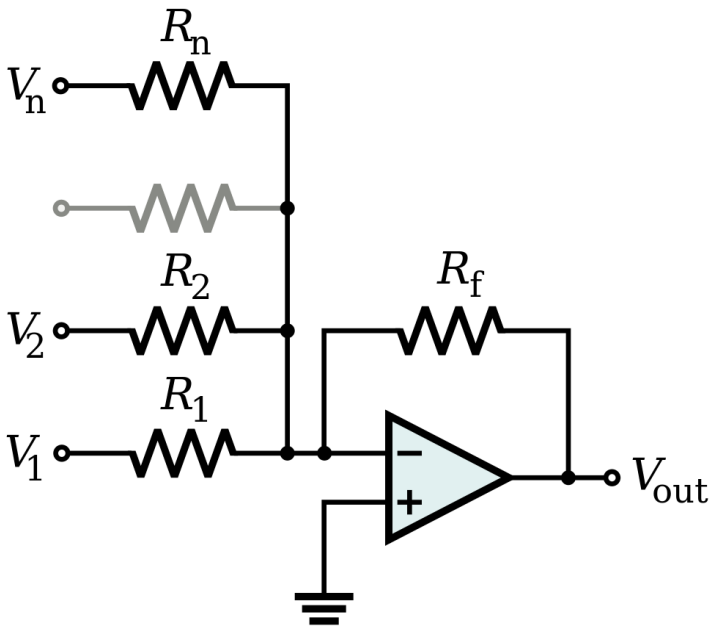
$$U_{ki} = \frac{R_1 + R_f}{R_1} \cdot \frac{R_g}{R_g + R_2} \cdot U2 - \frac{R_f}{R_1} \cdot U_1$$

Komparátor



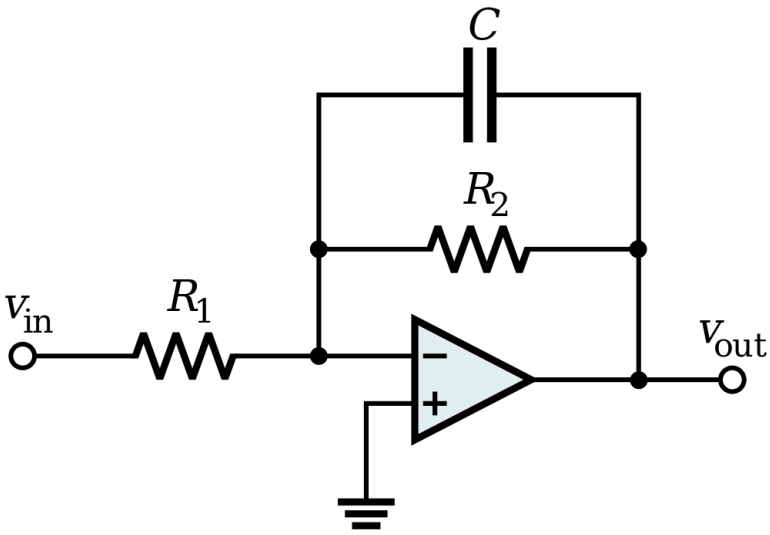
$$U_{ki} = \begin{Bmatrix} V_S + & ha & V_1 > V_{2,} \\ V_S - & ha & V_1 < V_{2,} \\ 0 & ha & V_1 = V_2 \end{Bmatrix}$$

Összegző erősítő



$$U_{ki} = -R_f \cdot \left(\frac{U_1}{R_1} + \frac{U_2}{R_2} + \ldots + \frac{U_n}{R_n} \right)$$

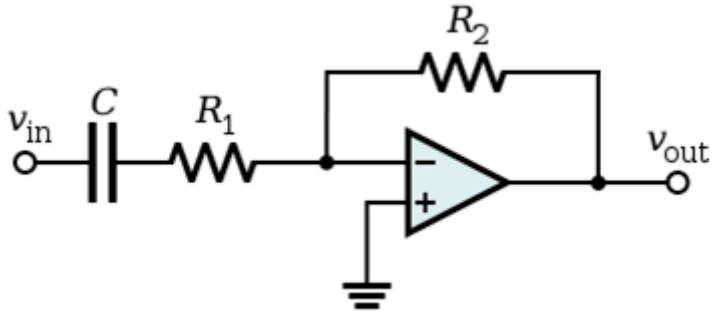
Aktív alul áteresztő szűrő



$$f_{levágás} = \frac{1}{2 \cdot \pi \cdot R_2 \cdot C}$$

$$\omega_{levágás} = \frac{1}{R_2 \cdot C}$$

Aktív felül áteresztő szűrő



$$f_{levágás} = \frac{1}{2 \cdot \pi \cdot R_1 \cdot C}$$

$$\omega_{levágás} = \frac{1}{R_1 \cdot C}$$

4. Hőmérséklet meghatározása termisztorral

A termisztor nem más, mint egy erősen hőmérsékletfüggő félvezetőből kialakított ellenállás. Olcsó előállítási költségei miatt széles körben alkalmazzák hőmérséklet mérésére és túlmelegedés elleni védelmekben. A termisztorok hőmérséklet-ellenállás függése nem lineáris, ebből kifolyólag nem triviális az alkalmazásuk mikrovezérlős környezetben.

A problémára több megoldás is létezik. A legegyszerűbb egy fordítótáblázaton alapul. A fordítótáblázat bizonyos pontosság mellett tartalmazza a mért ellenállás értékhez tartozó hőmérséklet értéket. Ezen megoldás több szempontból sem ideális. Elsősorban a pontossággal és a táblázat által elfoglalt memóriával vannak gondok. Másodsorban ezt minden egyes termisztor esetén saját magunknak kellene elkészítenünk, ami időigényes.

Ennél egyszerűbb és elegánsabb megoldás a Steinhart–Hart egyenlet alkalmazása. Ezt 1968-ban publikálta John S. Steinhart és Stanley R. Hart egy közös cikkben. Innen jött az egyenlet neve. A következőképpen néz ki az egyenlet:

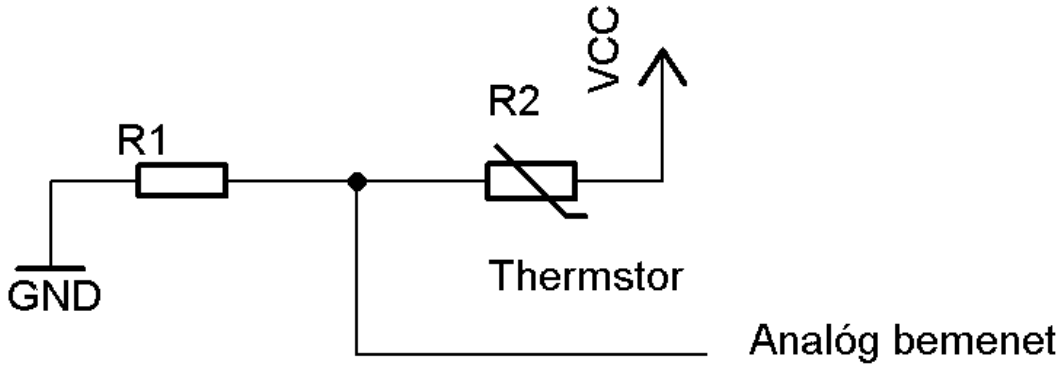
$$\frac{1}{T} = A + B \cdot \ln(R) + C \cdot (\ln(R))^3$$

Az egyenletben T betűvel jelölt hőmérséklet Kelvin fokban értendő. Az R ellenállás pedig a T hőmérsékleten mért termisztor ellenállás. A, B és C pedig Steinhart–Hart együtthatók. Ezek termisztoronként egyediek. Az egyenlet elterjedésének köszönhetően általában ez mára a termisztorok esetén katalógus adat. Amennyiben nincsenek megadva, akkor három különböző hőmérséklet mellett meg kell mérni a termisztor ellenállását. Ezáltal az alábbi háromismeretlenes egyenletrendszer segítségével kiszámolhatóak az együtthatók:

$$\left\{ \begin{array}{l} A + \ln(R_1) \cdot B + (\ln(R1))^3 \cdot C = \frac{1}{T_1} \\ A + \ln(R_2) \cdot B + (\ln(R2))^3 \cdot C = \frac{1}{T_1} \\ A + \ln(R_3) \cdot B + (\ln(R3))^3 \cdot C = \frac{1}{T_1} \end{array} \right\}$$

Az egyenlet által meghatározott hőmérséklet, ha az A, B és C együtthatók pontosan ismertek és a hőmérséklet nagyobb, mint 200 Kelvin, akkor kevesebb, mint 0,02 fokban tér el a valós hőmérséklettől.

A csatlakoztatáshoz szükséges kapcsolási rajz igen egyszerű. A termisztorból és egy ellenállásból feszültségosztót kell építeni, majd ennek a jelét kell rávezetni a mikrovezérlőre. Arra kell ügyelni csupán, hogy ha a termisztor névleges ellenállásával megegyező értékű legyen a másik ellenállás is a feszültség osztóban. A szükséges kapcsolási rajz a következő ábrán látható:



38. ábra: Termisztor bekötése mikrovezérlőhöz

A kezelő kód igen egyszerű, de használatához néhány konstanst be kell állítani. Ilyen konstans a feszültségosztó ellenállás ténylegesen mért értéke és az A, B és C együtthatók értéke. Amennyiben nem ismerjük az együtthatókat, az adott termisztor esetén jó közelítéssel alkalmazhatjuk a következő konstansokat:

$$A = 1{,}129148 \cdot 10^{-3} \quad B = 2{,}34125 \cdot 10^{-4} \quad C = 8{,}76741 \cdot 10^{-8}$$

Mivel a kód tartalmaz egy logaritmus függvényt, ezért szükség lesz a Math.h fájl importálására. Ez nincs dokumentálva az Arduino környezetben, mivel a C szabványos matematikai függvénykönyvtára. Ez tartalmazza többek között a logaritmus definícióját. A kód a hőmérséklet értéket Celsius fokban adja vissza. A mérés pontosítható többszöri mintavételezéssel és átlagolással esetlegesen.

```
#include <math.h>

//konstansok
const float Rnevleges = 9850;
const float A = 0.001129148;
const float B = 0.000234125;
const float C = 0.0000000876741;
//----------------------------

float Thermistor(int AnalogErtek)
{
    long Resistance;
    float Temp;
    Resistance=((1024 * Rnevleges / AnalogErtek) - Rnevleges);
    Temp = log(Resistance);
    Temp = 1 / (A + (B * Temp) + (C * Temp * Temp * Temp));
    Temp = Temp - 273.15;

    return Temp;
}
```

5. SD kártya kezelése

Az SD kártyák kezelése Arduino esetén az SPI buszrendszeren keresztül történik. A kártyák használata esetén nem kell állítani az SPI konfiguráción semmit, mivel ezt a könyvtár megteszi helyettünk. Ennek oka az, hogy a SD kártya szabvány igen részletesen leírja, hogy milyen beállítások mellett lehet kommunikálni a kártyával.

A kezelőkönyvtár egyaránt támogatja a FAT32 és FAT16 fájlrendszerrel formázott kártyákat, így akár SDHC kártyákat is alkalmazhatunk. Egyetlen megkötés csupán az, hogy a fájlneveknek 8.3 formátumban (DOS) kell lenniük. Ez azt takarja, hogy a fájlnév nem lehet több 8 karakternél és a kiterjesztés nem lehet több 3 karakternél. Ezen megkötés oka az, hogy a hosszú fájlnevek támogatása ezen fájlrendszereken a Microsoft szabadalma, amely használatáért fizetni kellene.

A hivatalos ajánlás fájlrendszer tekintetében FAT16, de ennek a használata csak 2Gb-nál kisebb méretű kártyák esetén lehetséges. 2Gb-nál nagyobb kártyákat mindenképpen muszáj FAT32 fájlrendszerre formázni, mivel a kezelőkönyvtár nem támogat több partíciót.

A kezelőkönyvtár természetesen támogatja a mappában elhelyezett fájlokat is. A mappa elválasztó karakter a sima perjel (/) és nem a Windows-on megszokott visszaperjel. (\ ) A mappákra is érvényes a fájlnév megkötés. A mappa neve nem lehet több 8 karakternél.

A kezelőkönyvtár két osztályból áll. Egy az SD kártya kezelésére szolgáló osztályból, és egy a fájlok kezelésére szolgáló osztályból. Ezek az osztályok az SD.h állományban vannak definiálva.

SD osztály

Ebben az osztályban olyan függvények vannak definiálva, amelyek hozzáférést biztosítanak az SD kártyához, valamint fájlokkal és könyvtárakkal kapcsolatos kezelőfunkciókat biztosít. Függvényei:

```
SD.Begin();
SD.Begin(cspin);
```

Inicializálja a könyvtárat és a kártyát az SPI buszrendszeren keresztül. A paramétere a chip select lábat határozza meg. Paraméter nélküli változatában a chip select láb megegyezik az SPI buszrendszerhez rendelt alapértelmezett Slave Select lábbal. Amennyiben nem ezt a lábat használjuk, akkor is a hardveres Slave Select lábnak kimenetnek kell lennie, máskülönben nem fog működni a kártya kezelése

```
SD.exists(filename);
```

Megnézi, hogy egy adott nevű fájl, vagy könyvtár létezik e. Visszatérési érteke igaz, amennyiben létezik a fájl, vagy mappa. A fájlnév természetesen teljes elérési útvonalat tartalmazhat.

```
SD.mkdir(filename);
```

Könyvtár létrehozása adott névvel. Több almappa megadása esetén (pl: a/b/c) minden olyan mappát létrehoz, ami még nem létezik az útvonalon. Visszatérési értéke igaz, ha nem keletkezett hiba a mappák létrehozása során.

```
SD.open(filepath);
SD.open(filepath, mode);
```

Megnyitja a paraméterben meghatározott fájlt. Visszatérése egy File objektum. Kétparaméteres változatában a második paraméter a megnyitás módját állítja be. Két konstans közül lehet itt választani:

- *FILE_READ*

  Csak olvasható módban megnyitás, az egyparaméteres változat ezt a megnyitási módot alkalmazza.

- *FILE_WRITE*

  A fájlt írható, olvasható módban nyitja meg. Amennyiben a fájl létezik már, akkor az írás a fájl végén fog kezdődni, vagyis nem írja felül a létező fájl tartalmát az új. Helyette hozzá lesz fűzve az új tartalom.

Amennyiben a fájl megnyitása sikertelen volt, akkor a fájl egy logikai vizsgálat során hamis értéket fog reprezentálni, így ellenőrizhető, hogy a megnyitás sikeres volt e.

```
SD.remove(filename);
```

Fájl törlése a kártyáról. Csak fájlok törlésére alkalmas. Visszatérési értéke igaz, ha a fájl törlése sikerült.

```
SD.rmdir(filename);
```

Könyvtár törlése a kártyáról. Csak könyvtárak törlésére alkalmas. Visszatérési értéke igaz, ha a könyvtár törlése sikeres volt. A törlés csak akkor lesz sikeres, ha a könyvtár nem tartalmaz fájlokat.

File osztály

```
file.available();
```

Visszaadja, hogy hány byte olvasása lehetséges még a fájlból.

```
file.close();
```

Bezárja a fájlt. Bezárás előtt minden kártyára írandó adat kiírásra kerül.

```
file.flush();
```

Kártyára írandó adatok kártyára írása a memóriából. Automatikusan meghívódik a fájl bezárásakor.

```
file.peek();
```

Egy byte olvasása a fájlból anélkül, hogy továbblépne a következő byte-ra a fájlban.

```
file.size();
```

A fájl méretét adja vissza unsigned long típusban. Ebből és a fájlrendszer (FAT32) limitációjából adódóan egy fájl mérete maximum 4Gb lehet.

```
file.read();
```

Egy byte olvasása a fájlból. A byte olvasása után automatikusan lépteti a fájl pozíciót, míg a peek nem.

```
file.write(data);
file.write(buf, len);
```

Egy byte kiírása a fájlba. Kétparaméteres változatában az első paraméter egy tömb, amely elemeit szeretnénk a fájlba írni. Második paramétere a tömbből a fájlba írandó byte-ok száma. A függvény visszatérési értéke a fájlba sikeresen írt byte-ok száma.

```
file.isDirectory();
```

Teszteli, hogy az adott fájl könyvtár-e, vagy egy fájl. Igaz értéket ad vissza, ha a megnyitott fájl könyvtár.

```
file.openNextFile();
```

Megnyitja a következő fájlt a könyvtárban.

```
file.rewindDirectory();
```

Az aktuális fájl könyvtárában az első fájlra ugrik.

Mappák tartalmának listázása

Mivel külön függvény nincs egy adott mappa tartalmának a kilistázására, a listázás az openNextFile és a rewindDirectory függvények segítségével oldható meg. Erre a hivatalos Arduino oldalról származó mintaprogram némi egyszerűsítés, és a megjegyzések magyarítása után a következő:

```
#include <SD.h>

File root;

void setup()
{
   Serial.begin(9600);
   pinMode(10, OUTPUT);
   SD.begin(10);

   root = SD.open("/");
   printDirectory(root);
   Serial.println("done!");
}

void loop()
{
   // setup lefutása után nem történik semmi.
}

void printDirectory(File dir)
{
   while(true)
   {
        File entry = dir.openNextFile();
        if (! entry)
        {
             // Nincs több fájl, vissza az elsőre
             dir.rewindDirectory();
             break;
        }
```

```
            for (unsigned int i=0; i<numTabs; i++)
            {
                Serial.print('\t');
            }
            Serial.print(entry.name());
            if (entry.isDirectory())
            {
                Serial.println("/");
                printDirectory(entry, numTabs+1);
            }
            else
            {
                // fájloknak van méretük, könyvtáraknak nincs.
                Serial.print("\t\t");
                Serial.println(entry.size(), DEC);
            }
        }
    }
```

További információk kinyerése a kártyáról

Az Arduino környezet beépítetten tartalmaz egy cardinfo nevezetű példaprogramot, amit eredetileg arra szántak, hogy az SD kártyákat alkalmazó projektek készítésekor könnyen tesztelni lehessen, hogy a kapcsolás működik-e, vagy nem. Ebben a példaprogramban számos nem dokumentált függvényt és konstanst találunk, amelyek segítségével információkat szerezhetünk az SD kártyánkról.

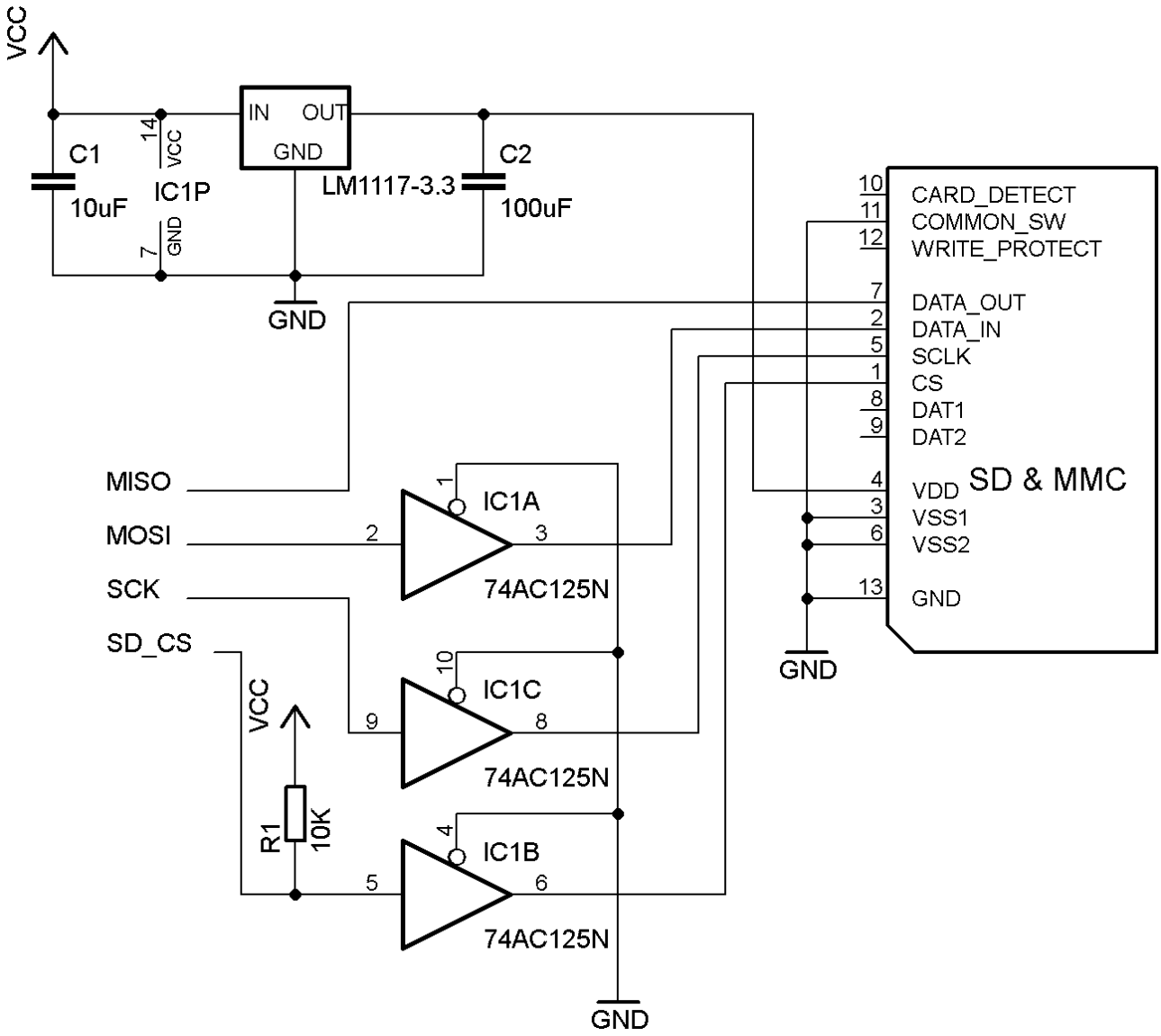
Ezen információk közé tartozik a kártya típusa és mérete. Mivel ezek nem olyan információk, amelyekre minden SD kártya kezelő alkalmazás esetén szükségünk van, ezért nincsenek külön dokumentálva.

Kapcsolási rajz

Az SD kártyák 3,3V feszültségről működnek és emiatt 3,3V-os logikai jelekkel kommunikálnak. Közvetlenül, jelszint konverzió nélkül nem köthetőek rá egy Arduino-ra sem, mert ez a kártya károsodásához vezetne. Ezért a jelszinteket illeszteni kell. Ezt megtehetjük passzív módon, ellenállásokból kialakított feszültségosztóval, vagy erre a célra kitalált logikai áramkörrel. Az ellenállásos illesztés nem túlzottan közkedvelt megoldás, mivel az ellenállások értéke hőmérséklettől és pontosságtól függően eltérhet a névleges értéktől. Ekkor a jelszintek is módosulnak.

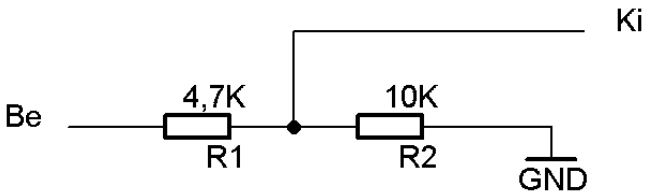
Jelszint illesztésre tökéletes áramkör a 74125 jelű logikai áramkör, ami 4db 3 állapotú buszrendszer meghajtót tartalmaz. Ez az egyik bemenetére érkező 5V logikai jelet átalakítja 3,3V-os logikai jellé.

A kártya tápfeszültség ellátása érdekében szükség lesz egy feszültség stabilizátorra is. Ebből gyakorlatilag bármilyen típus megteszi. Amennyiben gyári Arduino lappal dolgozunk, akkor a stabilizátorra nincs szükség, mert az Arduino-n van 3,3V-os feszültség kimenet is.



39. ábra: SD kártya bekötése az Arduino SPI buszrendszerére

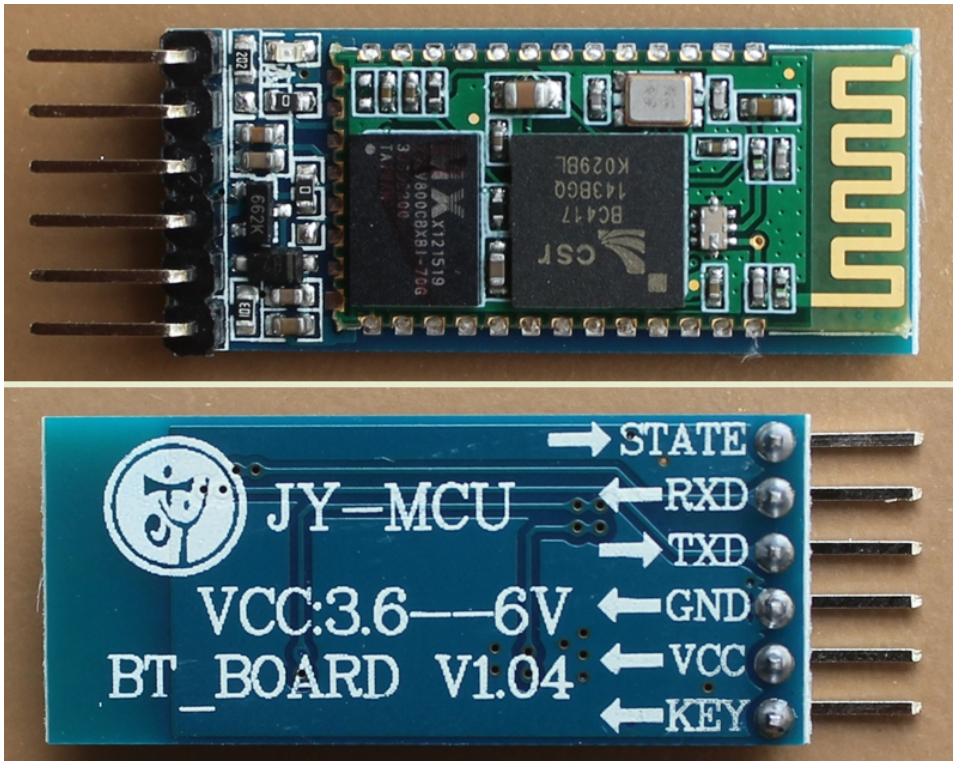
Az alábbi ábrán látható jelszint illesztés csak tesztelési célokra ajánlott:



40. ábra: Feszültségosztós jelszint illesztés

6. Bluetooth modulok használata

Bluetooth modulból igen sokféle kapható a piacon. Itt csak az olyan modulok kezeléséről lesz szó, amelyek soros portot valósítanak meg Bluetooth felületen. Az ilyen modulok relatíve olcsóak, igen sok helyről beszerezhetőek, viszont nem igen dokumentáltak. Az alábbi ábrán egy igen elterjedt számos helyen árult JY-MCU gyártmányú modul látható. Ennek a hatótávolsága nagyjából 10 méter.



41. ábra: Egy olcsó Bluetooth modul

Ezen modul dokumentációja forgalmazótól függően igen eltérő. Van, ahol azt írják, hogy 3,3 V feszültségről üzemel, van ahol azt, hogy 5. Továbbá az sem egyértelmű, hogy az eszköz 3,3V-os jelszinttel dolgozik, vagy 5V-os jelszintekkel. A modul, amihez volt szerencsém, 5V tápfeszültség mellett 5V-os logikai jelszintekkel dolgozott. Gyanítom, hogy 3,3V tápfeszültség mellett 3,3V jelszintekkel dolgozna, de ez csak egy tipp, mivel a felelhető dokumentációk igen ellentmondásosak. Ha modul vásárlásra szánjuk el magunkat, akkor azt tanácsolom, hogy először kisebb tápfeszültségről alacsonyabb jelszinttel próbálkozzunk, abból nem lehet baj, viszont ekkor illeszteni kell a jelszinteket, hogy működjön az Arduino-val a modul. Egy fellelt adatlapban azt írták, hogy a modul bemenetei nem 5V toleránsak. A helyes jelszint illesztéshez egy kapcsolási rajz a projekt leírásának a végén található.

A modul bekötése igen egyszerű. VCC lábra a tápfeszültséget kell csatlakoztatni, a GND lábat pedig földre. A modul RXD lábát a mikrovezérlő TXD lábára, a TXD lábát pedig a vezérlő RXD lábára. A STATE láb a modulon egy LED állapot kivezetés.

A modulokból három fajta lelhető fel a piacon: master, slave és univerzális. A slave és univerzális típusok a legelterjedtebbek. A modulok közötti különbség akkor mutatkozik meg, ha két mikrovezérlőt szeretnénk rávenni arra, hogy Bluetooth kapcsolaton keresztül kommunikáljanak. Ebben az esetben kell egy master eszköz, és egy slave eszköz. A master modul fogja minden esetben kezdeményezni a

kommunikációt a slave eszköz felé. Ebből adódóan két slave vagy két master eszköz nem tud egymással beszélgetni. PC-vel, vagy telefonnal való összekapcsoláshoz slave modult kell kötni a mikrovezérlőre. Az univerzális modulok esetén konfigurációfüggő, hogy master, vagy slave eszközként fog viselkedni a modul.

Az eszköz, ha nincs párosítva egy másik eszközhöz, akkor konfigurálható soros felületen keresztül. Ehhez szükségünk lesz egy USB-TTL soros átalakítóra. Továbbá kelleni fog egy terminálprogram, ami képes soros portot megnyitni. Erre több program képes, azonban célszerű olyat alkalmazni, amely direkt soros kommunikációhoz van kifejlesztve. Ilyen program például a termite nevű alkalmazás. Ez beszerezhető a http://www.compuphase.com/software_termite.htm címen.

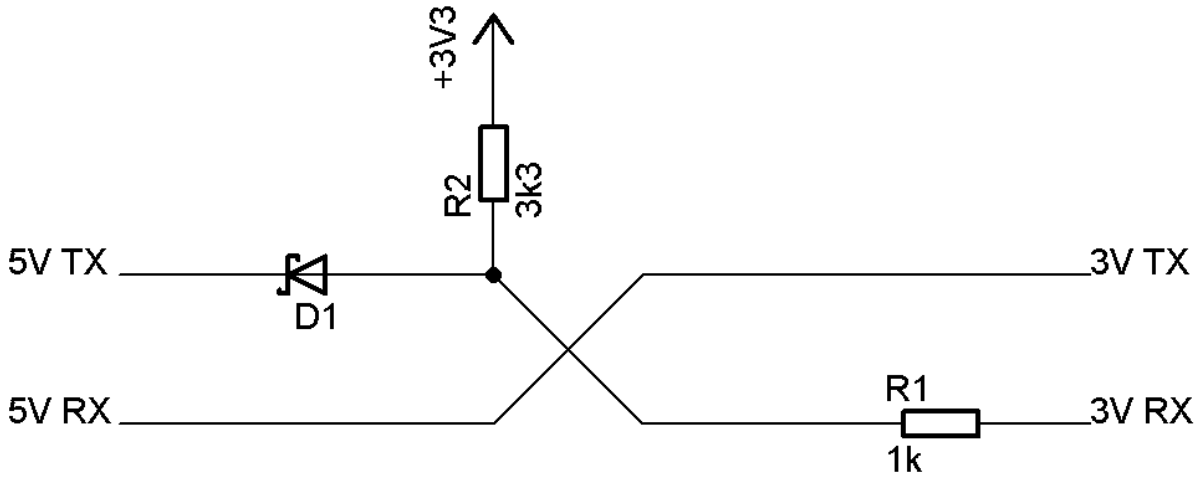
Az eszköz bekötése után a soros átalakítóra, majd megnyitva a megfelelő portot parancsszavak segítségével konfigurálhatjuk az eszközt. Válasz az eszköztől csak felismert parancs esetén fog érkezni. Az általam használt modul parancskészletét az alábbi táblázatban foglaltam össze.

<table>
<tr><th>Parancs</th><th>Hatása</th></tr>
<tr><td>AT</td><td>Kommunikáció tesztelése, ha az eszköz készen áll a kommunikációra, akkor egy OK üzenettel fog válaszolni.</td></tr>
<tr><td>AT+BAUDn</td><td>Baud ráta beállítása, az n az alábbi táblázatból egy szám, amely meghatározza a Baud rátát.
<table>
<tr><td>1</td><td>1200</td><td>7</td><td>57 600</td></tr>
<tr><td>2</td><td>2400</td><td>8</td><td>115 200</td></tr>
<tr><td>3</td><td>4800</td><td>9</td><td>230 400</td></tr>
<tr><td>4</td><td>9600</td><td>A</td><td>460 800</td></tr>
<tr><td>5</td><td>19 200</td><td>B</td><td>921 600</td></tr>
<tr><td>6</td><td>38 400</td><td>C</td><td>1 382 400</td></tr>
</table></td></tr>
<tr><td>AT+NAME=name</td><td>Az eszköz nevének beállítása, a nevet a name helyére kell írni, maximum 31 karakter lehet.</td></tr>
<tr><td>AT+NAME?</td><td>Az eszköz nevének lekérdezése.</td></tr>
<tr><td>AT+PIN=xxxx</td><td>Eszköz pin kód beállítása. A pin kód egy 4 karakteres szám lehet.</td></tr>
<tr><td>AT+PN</td><td>Paritás ellenőrzés kihagyása (alapértelmezés).</td></tr>
<tr><td>AT+PO</td><td>Páros paritás ellenőrzés bekapcsolása.</td></tr>
<tr><td>AT+PE</td><td>Páratlan paritás ellenőrzés bekapcsolása.</td></tr>
<tr><td>AT+ORGL</td><td>Gyári alapértelmezett beállítások visszaállítása.</td></tr>
<tr><td>AT+VERSION?</td><td>Verziószám lekérése.</td></tr>
</table>

21. táblázat: Bluetooth modul fontosabb konfigurációs parancsai

A legtöbb modul egyébként valamilyen EGBT Bluetooth modulon alapul. Ennek a teljes parancskészlete és dokumentációja fellelhető az interneten. Az egyszerűség kedvéért a modul parancskészletének dokumentációja megtalálható a könyv Mellékletek részében.

Az EGBT modulok dokumentációja alapján a 3.3V-os jelszintű modulok helyes illesztése 5V-os mikrovezérlőre a következő ábrán látható.

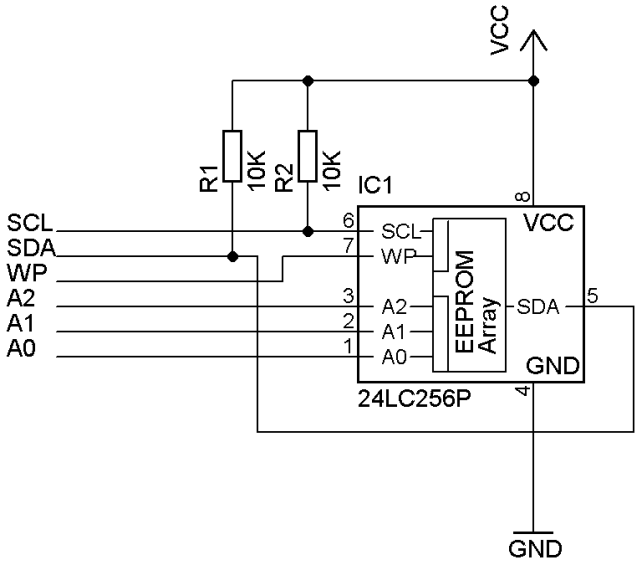


42. ábra: 3,3V-os Bluetooth modul illesztése 5V-os mikrovezérlőhöz.

Ezen illesztés esetén a modult 3,3V tápfeszültségről kell üzemeltetni. A D1 típusa mindegy, a lényeg annyi, hogy Schottky dióda legyen.

7. Soros EEPROM kezelése

A mikrovezérlőkben található EEPROM egy igen hasznos, de ez általában csak pár kilobyte méretű. Komplexebb Arduino projektjeink során előfordulhat, hogy ez kevés lesz. Kézenfekvő megoldás egy soros EEPROM illesztése ilyenkor. Ezen projekt írásakor a választás a Microchip 24LC256 típusú EEPROM memóriájára esett. Ez egy 256 Kbit kapacitású I2C buszra illeszkedő memória, amely 32Kb adat tárolását teszi lehetővé.



43. ábra: Soros EEPROM bekötése

A WP jelölésű láb írásvédelem bekapcsolására szolgál. Ha ez a láb földre van kötve, akkor a memória írható és olvasható is, ha pedig tápfeszültségre, akkor csak olvasható. Az A0, A1, A2 lábak a memória címének beállítására szolgálnak. Az eszköz 7 bites címének utolsó három bitjét határozzák meg, ebből adódóan 8db 24LC256 típusú memória lehet egyszerre a buszrendszerre kötve. A cím rögzített négy bitje 1010 bináris formában.

Memória száma	***Címbitek***			***Eszköz Cím hexadecimálisan***
	A2	***A1***	***A0***	
1	*0*	*0*	*0*	*0x50*
2	*0*	*0*	*1*	*0x51*
3	*0*	*1*	*0*	*0x52*
4	*0*	*1*	*1*	*0x53*
5	*1*	*0*	*0*	*0x54*
6	*1*	*0*	*1*	*0x55*
7	*1*	*1*	*0*	*0x56*
8	*1*	*1*	*1*	*0x57*

22. táblázat: A memória lehetséges címei

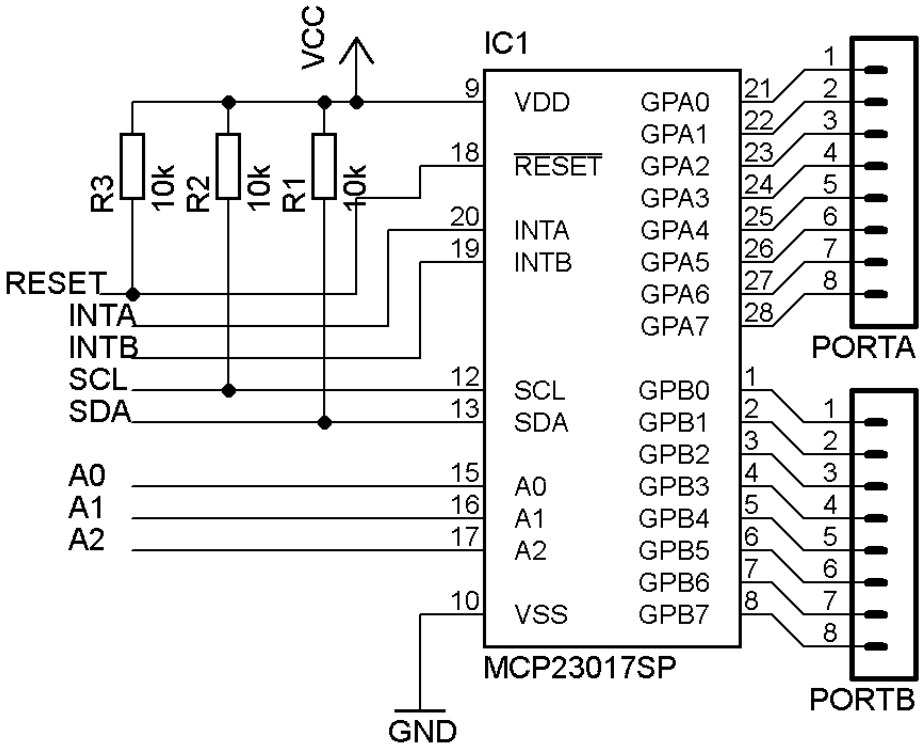
A memória kezelése úgy történik, hogy elküldjük a buszon keresztül a használni kívánt eszköz

címét. Ezután két byte-on elküldjük a kért memóriacímet, majd a címzési módtól függően (írási vagy olvasási kérés) vagy küldünk még egy byte-ot, ami az adat lesz, vagy az eszköz küld egy byte-ot, ami a memóriacímen található adat lesz. A kezelés egyszerűbbé tétele érdekében készítettem két függvényt, amivel egy byte adat írható, vagy olvasható a memóriából. A függvények használatához importálni kell az I²C buszrendszer kezelőfüggvényeket.

```
void writeEEPROM(int device, unsigned int eeaddress, byte data)
{
   Wire.beginTransmission(device);
   Wire.send((int)(eeaddress >> 8));       //MSB
   Wire.send((int)(eeaddress & 0xFF));     //LSB
   Wire.send(data);
   Wire.endTransmission();

   delay(5);
}

byte readEEPROM(int device, unsigned int eeaddress)
{
   byte rdata = 0x00;
   Wire.beginTransmission(device);
   Wire.send((int)(eeaddress >> 8));       //MSB
   Wire.send((int)(eeaddress & 0xFF));     //LSB
   Wire.endTransmission();
   Wire.requestFrom(device,1);

   if (Wire.available()) rdata = Wire.receive();
   return rdata;
}
```

8. Ki-és bemenetek bővítése

Kimenetek és bemenetek különálló bővítésére jó megoldást kínálhatnak a Shift regiszterek. Ezen megoldások hátránya az, hogy ha egyszerre szeretnénk ki-és bemenetet készíteni belőlük, akkor igen bonyolult elektronikát kapunk. Jobb megoldás erre a célra egy céláramkör használata. A Microchip MCP23017 pont ilyen áramkör. Ez I²C vagy SPI buszrendszerre kapcsolható (SPI rendszerre csak a MCP23S17) port bővítő. 2 db 8 bites portot tartalmaz. A portokhoz megszakítás is rendelhető.



44. ábra: MCP23017 I/O bővítő bekötése

Az áramkör címzése hasonló az előző projektben ismertetett memóriához. Az eszköz címének utolsó három bitje szintén szabadon megválasztható. A rögzített négy címbit binárisan 0100.

Bővítő száma	***Címbitek***			***Eszköz Cím hexadecimálisan***
	A2	***A1***	***A0***	
1	*0*	*0*	*0*	*0x20*
2	*0*	*0*	*1*	*0x21*
3	*0*	*1*	*0*	*0x22*
4	*0*	*1*	*1*	*0x23*
5	*1*	*0*	*0*	*0x24*
6	*1*	*0*	*1*	*0x25*
7	*1*	*1*	*0*	*0x26*
8	*1*	*1*	*1*	*0x27*

23. táblázat: Az MCP23017 I/O bővítő lehetséges címei

Az eszköz belső felépítésében hasonlít a PIC mikrovezérlőkre. Megadott regiszterekre értékeket kell írni, ezekkel konfiguráljuk. A fontosabb regisztereket az alábbi táblázat foglalja össze.

Regiszter neve	***Regiszter címe***	***Regiszter funkciója***
IODIRA	*0x00*	*A port lábainak szerepét határozza meg. Ahol egyes érték szerepel a 8 bites számban, az a láb bemenet lesz, ahol pedig nulla, ott kimenet lesz.*
IODIRB	*0x01*	*B port lábainak szerepét határozza meg, működése megegyezik az IODIRA regiszterrel.*
GPIOA	*0x12*	*A port kimeneteinek értékét határozza meg.*
GPIOB	*0x13*	*B port kimeneteinek értékeit határozza meg.*
OLATA	*0x14*	*A port értékét lehet belőle kiolvasni. Ezen regiszter olvasásakor nem a tényleges kimenetek olvasása történik, hanem a porthoz rendelt buffer olvasása.*
OLATB	*0x15*	*B port értékét lehet belőle kiolvasni. Ezen regiszter olvasásakor nem a tényleges kimenetek olvasása történik, hanem a porthoz rendelt buffer olvasása.*

24. táblázat: Az MCP23017 fontosabb regiszterei

Az egyszerűbb kezelhetőség miatt készítettem egy osztálykönyvtárt. A könyvtár az mcp2317 nevet viseli. Az alábbi függvényekkel rendelkezik:

```
Begin(Adress);
```

Inicializálja az osztályt. A függvény paramétere az eszköz címét határozza meg.

```
PortADirection(PORTMODE);
```

Beállítja az A port irányát. A paramétere két konstans lehet: OUTPUT vagy INPUT.

```
PortBDirection(PORTMODE);
```

Beállítja az B port irányát. A paramétere két konstans lehet: OUTPUT vagy INPUT.

```
WritePortA(value);
```

A függvény paramétere által meghatározott 8 bites számot küldi ki az A portra.

```
WritePortB(value);
```

A függvény paramétere által meghatározott 8 bites számot küldi ki a B portra.

```
ReadPortA();
```

Kiolvassa az A port értékét. Ez a függvény ténylegesen a portot olvassa, és nem a porthoz tartozó buffert. Visszatérési értéke egy 8 bites szám.

```
ReadPortB();
```

Kiolvassa a B port értékét. Ez a függvény ténylegesen a portot olvassa, és nem a porthoz tartozó buffert. Visszatérési értéke egy 8 bites szám.

```
pinMode(pin, MODE);
```

Arduino stílusú bővítő függvény. Egy megadott lábon állítja be, hogy kimenet legyen, vagy bemenet. A lábak számozása 0-tól 15-ig terjed. Az A port tartalmazza az első 8 kimenetet, míg a B port a második 8-at. A mód két konstans közül kerülhet ki: OUTPUT vagy INPUT.

```
digitalRead(pin);
```

Adott láb értékét olvassa ki ténylegesen a lábról, és nem a bufferből. Visszatérési értéke 0 vagy 1 lehet a láb állapotától függően.

```
digitalWrite(pin, value);
```

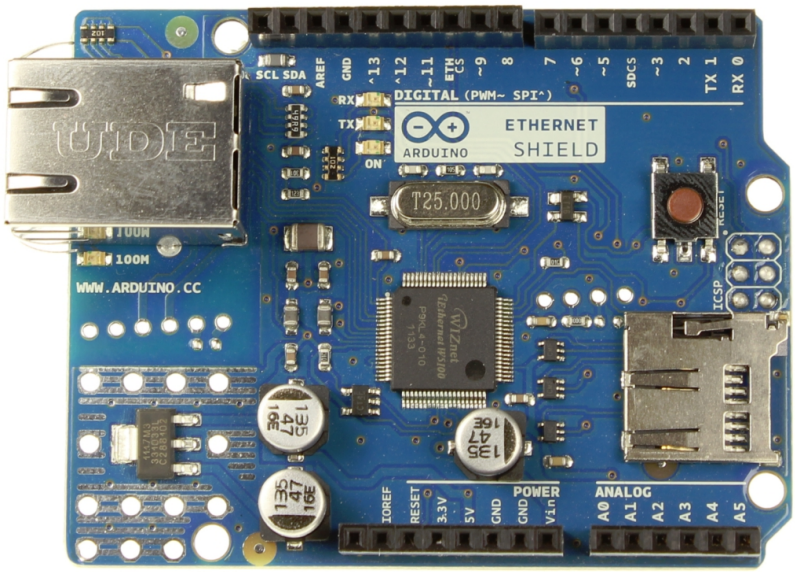
Adott láb értékének beállítása. A második paraméter két konstans lehet: HIGH vagy LOW.

A könyvtár használatához szükséges minden esetben az I^2C függvénykönyvtár importálása. A könyvtár használatát az alábbi rövid példa szemlélteti:

```
#include <Wire.h>
#include <mcp2317.h>

mcp2317 bovito;

void setup()
{
   bovito.Begin(0x20); //első eszközcím
   bovito.pinMode(0, OUTPUT);
}

void loop()
{
   bovito.digitalWrite(0, HIGH);
   delay(100);
   bovito.digitalWrite(0, LOW);
   delay(100);
}
```

9. Ethernet hálózatkezelés alapjai

Az Ethernet hálózatkialakítás a legnépszerűbb megoldás, ha számítógépes hálózatokról beszélünk. Előbb-utóbb a projektjeink során találkozni fogunk olyan igénnyel, hogy jó lenne hálózatra kapcsolni a projektünket. Erre van megoldás, mivel az Arduino platform fejlesztői készítettek egy kiegészítő panelt az Arduino számára, ami segítségével hálózatra kapcsolható. Ez az Ethernet Shield.



45. ábra: Arduino Ethernet Shield

A kiegészítő panel a Wiznet W5100 jelű integrált áramkörére épül, ami 10//100Mbit hálózati kapcsolat kezelésre képes. Az IC az Arduino-val SPI buszon keresztül kommunikál. A panel továbbá tartalmaz egy SD kártya foglalatot is, így akár egy minimális HTTP szerverként is tud működni az Arduino.

A kezeléshez a környezet tartalmaz egy függvénykönyvtárat. Az Ethernet Shield problémája, hogy az általa használt vezérlő chip viszonylag drága és otthoni tervekben nem igen használható, mivel a szükséges chip csak SMD típusban létezik. Otthoni alkalmazásra jobb a Microchip ENC26J60 típusú vezérlőjére épülő hálózati illesztő használata. Ennek a hátránya a hivatalos panelhez képest az, hogy csak 10Mbit maximális sebességre képes, de ez mikrovezérlős környezetben nem okozhat nagy gondot. A könyv mellékleti részében referenciaképpen megtalálható egy ENC26J60 vezérlőre épülő hálózati modul kapcsolási rajza.

Ehhez az illesztőhöz a környezet nem tartalmaz beépítetten kezelőfüggvényeket. A szükséges vezérlőkönyvtár a https://github.com/jcw/ethercard címről szerezhető be. Ennek használatával a könyv nem foglalkozik, mivel a könyvtár csak példákon keresztül dokumentált, valamint az írás pillanatában is erősen fejlesztés alatt állt, így megkérdőjelezhető a kód stabilitása és megbízhatósága.

A hivatalos Ethernet illesztő használatának megismerése előtt azonban a hálózatok alapjaival érdemes foglalkoznunk. Egyetlen egy mikrovezérlős könyvben sem olvastam a hálózatok alapjairól, mikor a hálózatkezelés szóba került. Pedig nem feltétlen kell az olvasónak értenie a témához. Természetesen ez is egy hatalmas téma (Andrew S. Tanenbaum egy több, mint 500 oldalas könyvet írt a témáról), így itt

csak a nagyon szükséges ismeretek lesznek közölve.

Az Ethernet

Az Ethernet egy 8db csavart vezetékpárt alkalmazó hálózatrendszer. Ebben minden állomás között pont-pont kapcsolat jön létre. A vezetékpárok galvanikusan le vannak választva mindkét oldalon egy-egy leválasztó transzformátorral. Ezt újabban már a hálózati kábel foglalatába beleépítik. Az Ethernet csatlakozója az RJ−45.

Két pont közötti távolság maximum 100 méter lehet, bár kábeltől függően ez lecsökkenhet 80 méterre is. Attól függően, hogy hány vezetékpárt közünk be, a hálózat sebessége változik. 10, 100 és 1000Mbps változatokat különböztetünk meg, melyek mindegyike visszafelé kompatibilis. Tehát egy 1000Mbps hálózaton használhatunk 10 és 100Mbps eszközöket egyaránt.

A használt vezetékek a hálózat sebességének függvényében eltérőek, felépítés és a csatlakozó bekötése szempontjából is. A vezetékek kategóriákra vannak osztva. Létezik CAT 3 (10Mbps), CAT 5 (100Mbps) és CAT 7 (1000Mbps) kábel.

A CAT 3 típusú kábelek a legegyszerűbbek. Ezekben a 8 darab vezeték (4 csavart érpár) semmilyen árnyékolással sem rendelkezik. A CAT 5 kábel egy külső árnyékolással rendelkezik, amelyen belül található meg a 4 csavart érpár. A CAT 7 kábelben minden egyes érpár külön árnyékolva van a külső árnyékoláson belül. A megfelelő kábeltípus kiválasztása a sebesség fenntartásában játszik nagy szerepet, például használhatunk CAT 5 kábelt is 1000Mbps hálózat építéséhez, azonban sosem lesz olyan gyors a hálózatunk, mint amilyen lehetne, ha CAT 7 típusú kábeleket használtunk volna.

A kábel bekötése sebesség-és eszközfüggő. Eszköz függés miatt minden bekötésből kétféle létezik. Az egyeneskötésű és a keresztkötésű kábeltípus. Keresztkötésű kábelt olyan helyen kell alkalmazni, ha két passzív eszközt kötünk össze valamilyen aktív eszköz nélkül (Pl. Két számítógép, vagy több számítógép hub-on keresztül.). Az egyeneskötésű kábelt pedig minden olyan esetben, ha aktív eszközön keresztül kapcsolunk össze két, vagy több eszközt. Az aktív eszközök többségének egyébként mindegy, hogy keresztkötésű, vagy egyeneskötésű kábellel van összekötve. A bekötések A és B típusra vannak bontva.

Egyenes kötésű akkor lesz egy kábelünk, ha mindkét végén a dugó A szabvány szerint van bekötve (Opcionálisan alkalmazhatunk 2db B bekötést is, de ez nem népszerű, mivel nem szabványos, de ettől függetlenül működőképes.). Keresztkötésű lesz a kábelünk akkor, ha az egyik oldalán a csatlakozó A szabvány szerint van kötve, míg a másik oldalán B szabvány szerint. Ezt a kábeltípust a gyárilag készített kábelek esetén crossover kábelnek nevezik.

Magasabb hálózati sebességre tervezett kábelt alkalmazhatunk alacsonyabb sebességű hálózatok esetén is abban az esetben, ha a hálózati kábelen csak hálózati jeleket továbbítunk. Alacsonyabb sebességű hálózatok esetén az átvitelre nem használt lábak szállíthatnak elektromos áramot is, amely alkalmas a hálózati eszköz működtetésére. Ezt nevezzük Power over Ethernet-nek (PoE). Először az ilyen megoldások gyártó specifikusak voltak, de mára már teljesen szabványos megoldás. Szabvány szerint maximum 48V egyenfeszültség továbbítható 350mA áramerősséggel, ami 16,5W teljesítmény átvitelére alkalmas.

A hivatalos Arduino Ethernet modulból létezik olyan változat, amely PoE kompatibilis. Pontosabban megfogalmazva, mindegyik hivatalos modul PoE kompatibilis, csak egyes modulokra nem szerelik rá azt a modult, ami a tápfeszültség levételéhez kellene.

A szabvány	Láb	B szabvány
Fehér-narancs	1	Fehér-zöld
Narancs	2	Zöld
Fehér-zöld	3	Fehér-narancs
Nincs bekötve	4	Nincs bekötve
Nincs bekötve	5	Nincs bekötve
Zöld	6	Narancs
Nincs bekötve	7	Nincs bekötve
Nincs bekötve	8	Nincs bekötve

25. táblázat: 10Mbps sebességű CAT 3 kábel szabványos színsorrendje

A szabvány	Láb	B szabvány
Fehér-narancs	1	Fehér-zöld
Narancs	2	Zöld
Fehér-zöld	3	Fehér-narancs
Kék	4	Kék
Fehér-kék	5	Fehér-kék
Zöld	6	Narancs
Fehér-barna	7	Fehér-barna
Barna	8	Barna

26. táblázat: 100Mbps sebességű CAT 5 kábel szabványos színsorrendje

A szabvány	Láb	B szabvány
Fehér-narancs	1	Fehér-zöld
Narancs	2	Zöld
Fehér-zöld	3	Fehér-narancs
Kék	4	Fehér-barna
Fehér-kék	5	Barna
Zöld	6	Narancs
Fehér-barna	7	Kék
Barna	8	Fehér-kék

27. táblázat: 1000Mbps sebességű CAT 7 kábel szabványos színsorrendje

Hálózati eszközök fajtái

Hálózati eszközökből megkülönböztetünk aktív, és passzív eszközöket. Azonban ez nem a tápfeszültség igényükre vonatkozik, hanem a hálózati működésükre. A passzív eszközök ilyen szempontból a legbutábbak, míg az aktív eszközök „úgymond" okosabbak. Passzív eszközök a repeater és a hub, aktívak pedig a switch és a router.

Repeater

A repeater nem más, mint egy ismétlő. Olyan helyeken alkalmazzák, ahol a vezeték hossza több, mint a maximálisan megengedett 100 méter. Ekkor 100 méterenként egy ismétlőt kell elhelyezni, ami a bemeneti portján érkező adatot átpakolja a kimeneti portjára.

Hub

A hub egy csillagponti elosztó. Minden pontja bemenet és kimenet is. A beérkező jelet szétosztja minden kimeneti pontján. A valóságban minden hub egyben egy ismétlő is.

Switch

A switch egy okosabb hub, mivel a beérkező jelet csak arra a vezetékre továbbítja, amelyiken a cél állomás található, így csökkenti a hálózat terhelését.

Bridge

A switch és a router közötti eszköz. Hasonlóan a router-hez, ez is hálózatok összekapcsolására alkalmas. Különbség azonban a megvalósításban van. A switch a logikai (IP) címek alapján végzi a hálózatok összekapcsolását, míg a bridge a hardveres (MAC) cím alapján, így nem alkalmas alhálózatra bontásra.

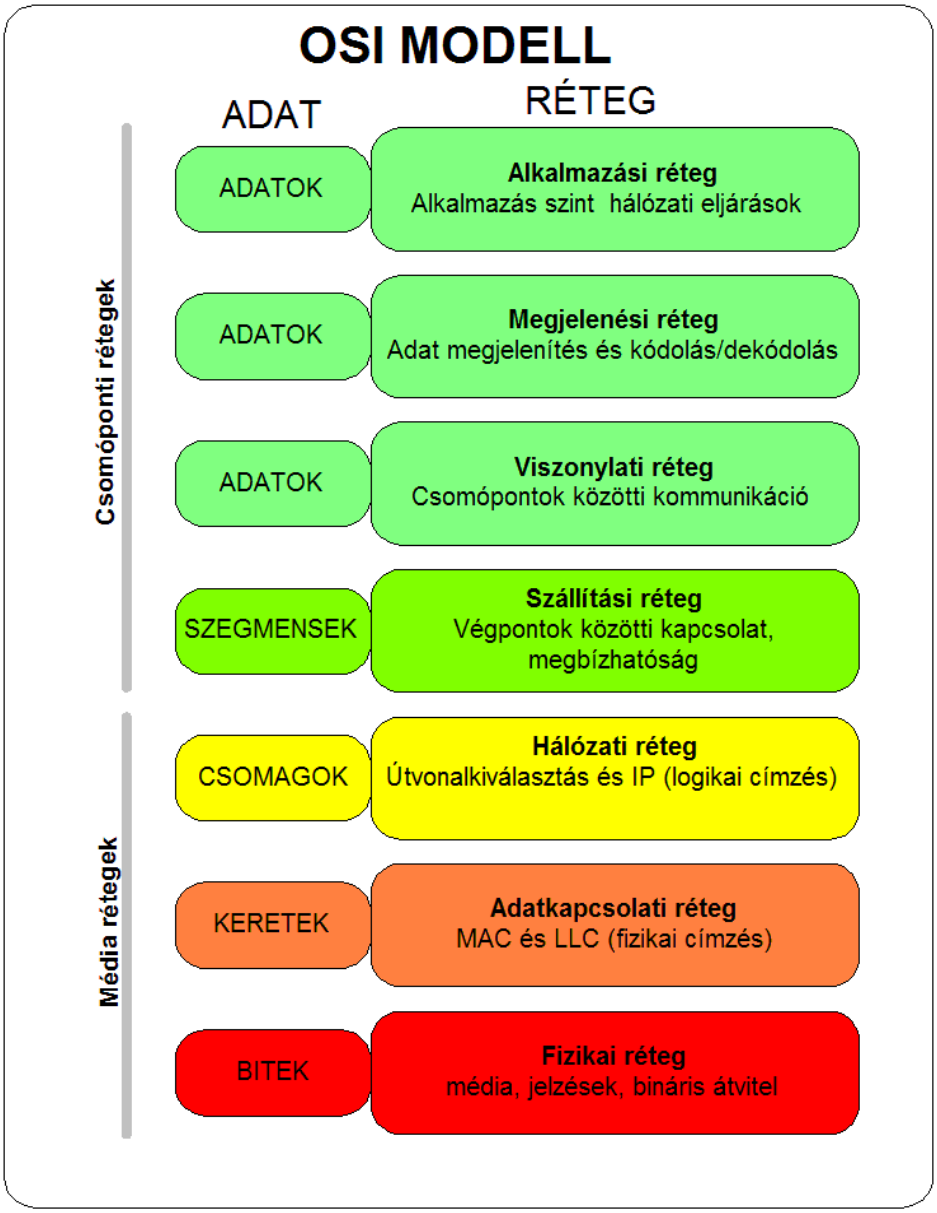
Router

A router hálózatok összekapcsolására alkalmas. Definíció szerint két portja van. Egy WAN port, ami egy másik hálózathoz kapcsolódik, és a LAN port, ami ahhoz a hálózathoz, amiben a router van. Valóságban azonban a legtöbb router egyben egy switch is, tehát több LAN portja van. Annyival fejlettebb a bridge-nél, hogy a router képes különbséget tenni az alhálózatok között.

Az OSI modell

Az OSI modell a nyílt rendszerek összekapcsolásának referenciamodellje. Azért alkották meg, hogy a számítógépes hálózatok összekapcsolhatóak legyenek implementációtól függetlenül. A valóságban csak papíron létezik, sosem alkalmazták teljes mértékben, csak egyes részhalmazait. Gyakorlatban a TCP/IP ötrétegű modellje van használatban az OSI 7 rétege helyett, mivel a teljes OSI megvalósítás feleslegesen bonyolult lenne.

A hét réteg közös tulajdonsága, hogy minden réteg csak az alatta lévő réteg szolgáltatásaira építhet, valamint a réteg a szolgáltatásait csak a felette álló réteg számára nyújthatja. Így gyakorlatilag két állomáson felépített modell egyes rétegei csak egymással kommunikálhatnak.



46. ábra: Az OSI modell rétegei

Fizikai réteg

A fizikai réteg definiálja az eszközökkel szemben támasztott fizikai-és villamos specifikációt, ami kell a hálózat működéséhez. Ez a réteg konkrétan bitek sorozatával dolgozik. Hálózati eszközökből a hub és a repeater dolgozik ezen a szinten.

Adatkapcsolati réteg

Azokat a funkciókat és eljárásokat biztosítja, amelyek lehetővé teszik az adatok átvitelét két hálózati elem között. Jelzi, illetve lehetőség szerint korrigálja a fizikai szinten történt hibákat is (megjegyzés: gyakorlatban mára ez a feladat az IP protokoll része, így nem igen használt ezen rétegben). A használt egyszerű címzési séma fizikai szintű, azaz a használt címek fizikai címek (MAC címek), amelyeket a gyártó fixen állított be hálózati kártya szinten.

Hálózati réteg

Biztosítja a változó hosszúságú adatsorozatoknak a küldőtől a címzetthez való továbbításához szükséges funkciókat és eljárásokat úgy, hogy az adatok továbbítása a szolgáltatási minőség függvényében akár egy, vagy több hálózaton keresztül is történhet. A hálózati réteg biztosítja a hálózati útvonalválasztást, az adatáramlás ellenőrzést, az adatok szegmentálását/deszegmentálását, és főként a hibaellenőrzési funkciókat. Az útvonalválasztók (router-ek) ezen a szinten működnek a hálózatban. Itt már logikai címzési sémát használ a modell-az értékeket a hálózat karbantartója adja meg egy hierarchikus szervezésű címzési séma használatával. Ez lesz az IP cím. Ebből jelenleg két változat van használatban. A 4-es és 6-os változat.

Szállítási réteg

A szállítási réteg biztosítja, hogy a felhasználók közötti adatátvitel transzparens legyen. A réteg biztosítja, és ellenőrzi egy adott kapcsolat megbízhatóságát. Néhány protokoll kapcsolatorientált. Ez azt jelenti, hogy a réteg nyomon követi az adatcsomagokat, és hiba esetén gondoskodik a csomag vagy csomagok újraküldéséről. A legismertebb 4. szintű protokoll a TCP, de ebbe a rétegbe tartozik az UDP protokoll is.

Viszony réteg

A viszony réteg a végfelhasználói alkalmazások közötti dialógus menedzselésére alkalmas mechanizmust valósít meg. Jellegzetes feladata a logikai kapcsolat felépítése és bontása, párbeszéd szervezése. Szinkronizációs feladatokat is ellát, ellenőrzési pontok beépítésével. Gyakran az együttműködési réteg elnevezéssel is illetik.

Megjelenítési réteg

A megjelenítési réteg felelős az információ formázásáért és eljuttatásáért az alkalmazási rétegnek, ahol a további feldolgozás, illetve megjelenítés történik. Gondoskodik róla, hogy az alkalmazási rétegnek már ne kelljen foglalkoznia a végfelhasználói rendszerek esetleg különböző adatértelmezési módszereiből

származó szintaktikai eltérésekkel.

Alkalmazási réteg

Az alkalmazási réteg szolgáltatásai támogatják a szoftver alkalmazások közötti kommunikációt, és az alsóbb szintű hálózati szolgáltatások képesek értelmezni alkalmazásoktól jövő igényeket, illetve, az alkalmazások képesek a hálózaton küldött adatok igényenkénti értelmezésére. Az alkalmazási réteg protokolljain keresztül az alkalmazások képesek egyeztetni formátumról, további eljárásról, biztonsági, szinkronizálási vagy egyéb hálózati igényekről.

A TCP/IP modell

A TCP/IP rövidítés az internetet is felépítő protokoll struktúrát jelenti. Nevét a TCP és IP protokollokból kapta, az OSI modell gyakorlatban is létező megvalósítása. A rétegek neve és feladata többnyire megegyezik az OSI modellben tárgyaltakkal, de némiképpen egyszerűsítve vannak.

Alkalmazási réteg

Az alkalmazási réteg a felhasználó által indított program és a szállítási réteg között teremt kapcsolatot. Ha egy program hálózaton keresztül adatot szeretne küldeni, az alkalmazási réteg továbbküldi azt a szállítási rétegnek.

Szállítási réteg

Az alkalmazási rétegtől kapott adat elejére egy úgynevezett fejlécet (angolul: header) csatol, mely jelzi, hogy melyik szállítási rétegbeli protokollal (leggyakrabban TCP vagy UDP) küldik az adatot.

Hálózati (Internet) réteg

A szállítási rétegtől kapott header-adat pároshoz hozzáteszi a saját fejlécét, amely arról tartalmaz információt, hogy az adatot melyik végpont kapja majd meg.

Adatkapcsolati réteg

Az adatkapcsolati réteg szintén hozzárakja a kapott adathoz a saját fejlécét, és az adatot keretekre bontja. Ha a kapott adat túl nagy ahhoz, hogy egy keretbe kerüljön, feldarabolja és az utolsó keret végére egy úgynevezett tail-t kapcsol, hogy a fogadó oldalon vissza lehessen állítani az eredeti adatot.

Fizikai réteg

A fizikai réteg továbbítja az adatkapcsolati rétegtől kapott kereteket a hálózaton. A fogadó oldalon ugyanez a folyamat játszódik le visszafelé, míg az adat a fogadó gép alkalmazásához nem ér.

Protokollok, technológiák

A modellek ismertetése önmagában nem elég a hálózat működésének megértéséhez, bár jó alapot biztosítanak hozzá. Ebben a szekcióban az egyes fontosabb technológiákkal és protokollokkal részletesebben foglalkozok, amik ismerete mindenképpen kell a hálózatok megértéséhez.

IP protokoll

Az IP protokoll felelős egy hálózaton belül a gépek logikai címzéséért. Két változata van jelenleg használatban: a négyes (IPv4) és hatos (IPv6). A négyes verzió 1981-ben jelent meg. Ez 32 bites címmel azonosítja az eszközöket. A 32 bites címet byte-onként adjuk meg decimális formában, pontokkal elválasztva. 32 bit segítségével picivel több, mint 4,2 milliárd eszköz címezhető meg egyszerre. A szabvány kidolgozói gondoltak arra, hogy az elérhető címek előbb-utóbb el fognak fogyni, ezért azt a megoldást találták ki, hogy az internetet felbontják alhálózatokra. Ez konkrétan azt jelenti, hogy az interneten címezhető eszközök (hostok) rendelkezhetnek két címmel. Egy külső, publikus címmel és egy belső, hálózati címmel. Így a belső hálózaton szintén 4,2 milliárd gép címezhető, ha a belső hálózatot nem bontja egy host további hálózatokra. A hálózatokra bontás eszköze a router, a belső címére érkező adatcsomagokat továbbítja a külső hálózat felé, és fordítva is.

Ez a címzési séma egy ideig bevált, azonban a hálózatok egy idő után elkezdtek bonyolódni. A protokoll kidolgozói felismerték, hogy hosszútávon nem megoldás ez a címzési séma. Ezért sok huzavona után elkészült az IP protokoll hatos verziója. Érdekességképpen megemlíthető, hogy bár volt a protokollnak ötös verziója is, de ezt sosem használták, helyette a javított hatos verzió terjedt el. Ezt mára a legtöbb operációs rendszer támogatja. A hálózati eszközökről ez már sajnos nem igen mondható el. Szerencsére tud a protokoll IPv4-es hálózati eszközökön keresztül is működni egy speciális módban, de ezt sem minden támogatja jelenleg. A hatos verzió 128 bites címeket használ. Így elméletileg $3{,}402 \cdot 10^{38}$ *darab host címezhető meg egyidejűleg. Ez egy hatalmas szám, olyan nagy, hogy ha kivitelezhető lenne, adhatnánk a levegőben található összes molekulának egyedi címet, és még akkor is maradni ki címünk. A hatos verzió felépítésével nem foglalkozok a könyv keretein belül, mivel a beágyazott eszközök többsége a négyes verziót alkalmazza.*

A kioszthatóak címeket három osztályba soroljuk. A címosztályok meghatározzák, hogy a teljes címtartomány hány darab hálózatra és hostra bontható fel. Ettől a címzési rendszertől el lehet térni.

Cím osztály	***Cím kezdőbitek***	***Hálózat azonosító hossza***	***Hálózatok száma***	***Host azonosító hossza***	***Címezhető eszközök száma***
A	*0*	*8 bit*	*126*	*24 bit*	*16 777 214*
B	*10*	*16 bit*	*16 382*	*16 bit*	*65 534*
C	*110*	*24 bit*	*2 097 150*	*8 bit*	*254*

28. táblázat: IPv4 címosztályok jellemzői

Minden cím mellé lesz egy alhálózati maszk. Ez határozza meg egy IP címből a hálózat címét. Ez a hálózatok további felbontása és összekötése érdekében kell. A maszk szintén 32 bites. Ahol binárisan egyes értéket képvisel, azon bitek határozzák meg a hálózat címét. A maszk és az IP cím között elvégzett bináris ÉS művelettel fogja tudni megállapítani a hálózati címet bármelyik eszköz. Egy C osztályú IP cím esetén a maszk első 24 bitje csupa egyes, utolsó nyolc bitje pedig 0. Ha ezt az IP címek esetén megszokott

módon leírjuk, akkor azt kapjuk, hogy a maszk értéke: 255.255.255.0

IP cím osztályon belül olyan host cím nem adható meg, amely csak nullákat és egyeseket tartalmaz. Ezek fenntartott címek. A csupa nullás host jelenti a hálózat címét, míg a csupa egyeseket tartalmazó az úgynevezett broadcast cím, amelyre ha üzenetet küldünk, akkor az adott hálózat összes gépe megkapja.

Amennyiben a hálózatunkat össze szeretnénk kapcsolni egy másik hálózattal, meg kell adnunk még egy címet, mégpedig az alapértelmezett átjáró címét. Ez egy olyan állomás címe lesz, amely kapcsolatban van a saját hálózatunkkal és egy másikkal is. Általában ez a legelső elérhető host címet kapja, vagy az utolsót. De ettől tetszőlegesen el lehet térni.

Szállítási protokollok: TCP és UDP

Az IP protokoll önmagában csak az állomások logikai címzését biztosítja. Azt már nem, hogy az üzenet elérjen a küldőtől a fogadóig, mivel ez a szállítási protokoll feladata. Szállítási protokollok közül a legelterjedtebbek a TCP és az UDP. Bár az UDP az elmúlt években igencsak eltűnt, mondhatni majdnem kihalt.

A TCP protokoll maximum 64Kb-os darabokra bontja a küldendő üzeneteket, majd ezeket a csomagokat továbbítja. A protokoll garantálja az üzenetek hibamentes célba juttatását. A címzett gép minden fogadott csomag után egy visszajelzést küld a feladónak. Amennyiben a csomag elveszne a küldés során, akkor a feladó újraküldi a csomagot.

Az UDP egy összeköttetés-mentes protokoll. Ez azt jelenti, hogy a feladó feladja a csomagot, de nem kér a fogadásról visszajelzést a vevőtől. Így nem garantált az üzenetek célba érkezése, de mivel ellenőrző összeg tartozik minden csomaghoz, a hibás kézbesítés felismerését lehetővé teszi. Leginkább olyan helyeken alkalmazott ezen szállítási protokoll, ahol az üzenet újbóli átvitele könnyen megoldható egy egyszerű ismételt lekéréssel. Például a DNS szolgáltatás UDP szállítással kommunikál.

MAC-cím

Minden hálózati eszköz rendelkezik egy fizikai címmel. Ez fogja egy hálózaton belül ténylegesen azonosítani az eszközt. Ez egy 48 bites szám, amelyet hexadecimális formában adnak meg, byte-onként kötőjellel elválasztva. A szám 24 bitje az eszköz gyártóját és/vagy forgalmazóját azonosítja. A számsorozat ezen részének kiosztását egy nemzetközi szervezet felügyeli, míg a maradék 24 bit kiosztásáért a gyártó felelős. Egyes operációs rendszerek és eszközök esetén ez szoftveresen felülbírálható különböző okok miatt. Ha egy hálózaton belül két azonos fizikai című gép van, az komoly problémákat okoz.

DNS

A DNS szolgáltatás teszi azt lehetővé, hogy ne kelljen számszerű IP címeket megjegyeznünk. Segítségével a számítógépeket nevük és tartományuk alapján címezni tudjuk. Egy hálózatot a DNS szolgáltatás egymásra épülő, szintekből álló, névvel ellátott tartományra bont.

Technikai oldalról a DNS nem más, mint egy osztott adatbázis. Az osztott kifejezés adatbázisok

esetén azt jelenti, hogy nem egy gépen tárolódnak az adatok.

Egy hálózati konfiguráció során meg kell adnunk minden esetben egy DNS kiszolgáló címét. Ezen kiszolgáló felé indulnak majd el a névfeloldási kérelmeink. Amennyiben az adott szerveren nem találhatóak meg a kért adatok, akkor a szerver átirányítja a kérést egy olyan szerver felé, amely rendelkezik információval a névhez tartozó címről.

Portok

Az IP címek az egyes gépek azonosítására szolgálnak. A portok pedig a hálózati kommunikációban résztvevő programok megjelölésére szolgálnak. A portok címzése 16 bites, vagyis összesen 65536db port létezik. A portok kiosztása az első 1024 port esetében közmegegyezés alapján előre meghatározott alkalmazások számára fenntartott. A további portok kiosztása dinamikus, azaz az alkalmazás addig foglalja le az adott portot, amíg szüksége van rá, majd felszabadítja. Ezzel a megoldással elkerülhető, hogy a népszerű és fontos hálózati szolgáltatások, programok problémamentesen tudjanak kommunikálni. A fontosabb szolgáltatások és programok által használt portokat az alábbi táblázat foglalja össze:

Szolgáltatás, program	***Használt port***	***Szállítási protokoll***
HTTP	*80*	*TCP*
HTTPS	*443*	*TCP*
FTP	*20, 21*	*TCP*
Telnet	*23*	*TCP/UDP*
SSH	*22*	*TCP*
DNS	*53*	*UDP*
DHCP	*68, 67*	*UDP*

29. táblázat: Fontosabb szolgáltatások által használt portok

10. Az Arduino Ethernet könyvtára

Az Arduino Ethernet könyvtára öt alaposztályból áll. A legtöbb osztály az ethernet.h fejléc állományban van definiálva, így a projektünkhöz csak ezt kell csatolni, valamint az spi.h állományt is, mivel a könyvtár chip-je SPI buszrendszeren keresztül kommunikál a mikrovezérlővel.

A kiegészítő panelen található egy micro SD kártya foglalat is. Ez a korábban már ismertetett SD kártya kezelő könyvtárral kezelhető.

A panel Uno esetén a 11, 12, 13 lábakat használja (SPI buszrendszer lábai), valamint a 10-es és 4-es lábat. A 10-es láb az Ethernet vezérlők kapcsolja a buszra, míg a 4-es az SD kártyát.

Ethernet osztály

Az osztály feladata a panel inicializációja, valamint a hálózati beállítások elvégzése. Három függvényt biztosít.

```
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);
```

Inicializálja a panelt és a hálózati beállításokat. A függvény első változata DHCP szerver segítségégével próbál konfigurációt létesíteni. A szükséges paraméter a MAC cím. Ez újabb gyártmányú kiegészítő panelek esetén megadott egy matrícán. Régebbi panelek esetén tetszőlegesen megválasztható. A megadása egy 6 elemű byte tömb segítségével történik.

A függvény többi változata manuális konfigurációt létesít. A szükséges címek megadhatóak 4 elemből álló byte tömbként, vagy egy IPAddress típusú változóval.

A begin függvény DHCP konfiguráció esetén 1-es értékkel tér vissza, ha a konfiguráció sikerült, 0 értékkel pedig akkor, ha nem sikerült a konfiguráció. A manuális konfigurációs változatoknak nincs visszatérési értéke, valamint a manuális konfigurációk kisebb méretű kódot eredményeznek.

```
Ethernet.localIP();
```

Lekérdezi az aktuális IP címet. Hasznos DHCP konfiguráció esetén. Visszatérési értéke az IP cím IPAddress típusként.

```
Ethernet.maintain();
```

DHCP konfiguráció esetén fenntartja a kapcsolatot, mivel minden egyes DHCP-n kiosztott címnek van egy érvényességi ideje. Ennek lejárta után a kapcsolat megszakad, ha csak közben nem érkezik megújítási kérelem. A függvény visszatérési értéke egész szám. 5 különböző visszatérési érték lehetséges. Ezek:

0	*Nem történt semmi*	*3*	*Új cím kérési hiba*
1	*Megújítás nem sikerült.*	*4*	*Új cím kérése sikeresen megtörtént.*
2	*Megújítás sikeres volt.*		

IPAddress osztály

IP címek tárolására szolgáló osztály. Tagfüggvényekkel nem rendelkezik. Egyetlen konstruktora négy darab byte-ot vár, ami meghatározza a címet.

```
IPAddress(address);
```

EthernetServer osztály

Az EthernetServer osztály egy TCP szállítási réteggel rendelkező általános szerverosztály, amely alkalmas kliensek kiszolgálására.

```
EthernetServer serv = EthernetServer(port);
```

Az osztály konstruktora egyetlen egy paramétert igényel, egy portszámot, amin a szerver dolgozhat.

```
serv.begin();
```

A függvény meghívása után a szerver ténylegesen készen áll kapcsolatok fogadására.

```
serv.available();
```

Megnézi, hogy van e kliens, akinek adatot lehet küldeni, ha van, akkor a függvény visszatérési értéke egy kliens osztály (definícióját lásd később). Amennyiben nincs kliens, akkor a függvény visszatérési értéke logikai hamis érték (false).

```
serv.write(data);
```

Egy byte, vagy egy karakter adatot küld át az összes szerverhez csatlakozott kliensnek.

```
serv.print(data);
serv.print(data, BASE);
```

Szöveges üzenetet továbbít a szerverhez csatlakozott összes kliensnek. A formázási beállításai és lehetőségei megegyeznek a soros port kezelésekor tárgyalt print függvénnyel.

```
server.println();
server.println(data);
server.println(data, BASE);
```

Szöveges üzenetet továbbít a szerverhez csatlakozott összes kliensnek, majd az üzenet végére illeszt egy sortörést. A formázási beállításai és lehetőségei megegyeznek a soros port kezelésekor tárgyalt println függvénnyel.

Client osztály

A client osztály reprezentálja a szerverhez kapcsolódni tudó klienseket, valamint lehetővé teszi a kapcsolódást szerverekhez.

```
EthernetClient client;
```

Létrehoz egy új kliens objektumot, a konstruktora nem igényel paramétereket.

```
client.connected();
```

Ellenőrzi, hogy a kliens csatlakozik-e szerverhez, vagy sem. Amennyiben a kapcsolat már le lett zárva, de a kliensnek még van fel nem dolgozott adatja, akkor a függvény még igaz értékkel fog visszatérni.

```
client.connect(ip, port);
client.connect(URL, port);
```

Csatlakozás szerverhez. Az első paraméter egy IP cím, ami megadható egy négy byte-ból álló tömb formájában, vagy az IP cím osztály segítségével, vagy egy domain név segítségével. A második paraméter a szerver portszáma.

```
client.write(data);
```

Egy byte, vagy egy karakter adatot továbbít a szerver felé, amelyikhez a kliens csatlakozik.

```
client.print(data);
client.print(data, BASE);
```

Adatot továbbít szöveges formában a szerver felé. A formázási beállításai és lehetőségei megegyeznek a soros port kezelésekor tárgyalt print függvénnyel.

```
client.println();
client.println(data);
client.print(data, BASE);
```

Adatot továbbít szöveges formában a szerver felé, az üzenet végére sortörést illeszt. A formázási beállításai és lehetőségei megegyeznek a soros port kezelésekor tárgyalt print függvénnyel.

```
client.available();
```

Visszaadja a beolvasható byte-ok számát, amit a szerver küldött.

```
client.read();
```

Egy byte adatot fogad a szervertől.

```
client.flush();
```

A szerver által küldött, de nem feldolgozott byte-ok törlése a pufferből.

```
client.stop();
```

Lekapcsolódás a szerverről, a kapcsolat bontása.

EthernetUDP osztály

UDP csomagok küldését és fogadását teszi lehetővé ez az osztály. Az osztály az EthernetUdp.h fájban van definiálva, így ha használni szeretnénk, akkor ezt is csatolnunk kell a programunkhoz.

```
EthernetUDP Udp;
```

Példányosítja az EthernetUDP osztályt. Az osztály konstruktora paramétereket nem igényel.

```
EthernetUDP.begin(localPort);
```

Inicializálja az objektumot, illetve felkészíti a megadott porton való kommunikációra.

```
UDP.parsePacket();
```

Feldolgoz egy fogadott csomagot. Előkészíti olvasásra és kiszámítja a csomag hosszát. Ezt a függvényt mindenképpen meg kell hívni, mielőtt olvasni próbáljuk a csomagot.

```
UDP.read();
UDP.read(packetBuffer, MaxSize);
```

Paraméter nélküli változata egy byte adatot olvas ki a fogadott csomagból. A paraméterezett változatában az első paraméter egy tömb, amibe az adat fog kerülni, a második paramétere pedig az, hogy ténylegesen hány bájt adatot tegyen bele a tömbbe.

```
UDP.beginPacket(remoteIP, remotePort);
```

Előkészít egy UDP csomagot adatátvitelre. Az első paraméter a cél IP cím, ami megadható egy négy byte hosszúságú tömbként, vagy az IP cím osztállyal. Második paramétere a cél port, amire az adatot küldjük.

```
UDP.write(message);
```

Egy karaktert ír bele az elküldendő csomagba.

```
UDP.endPacket();
```

Lezárja a küldendő csomagot. A függvény hívása után lesz elküldve az UDP csomag.

```
UDP.available();
```

Visszaadja, hogy hány byte adat olvasható még ki a fogadott csomagból, ezért használata előtt meg kell hívni a parsePacket(); függvényt.

```
UDP.remoteIP();
```

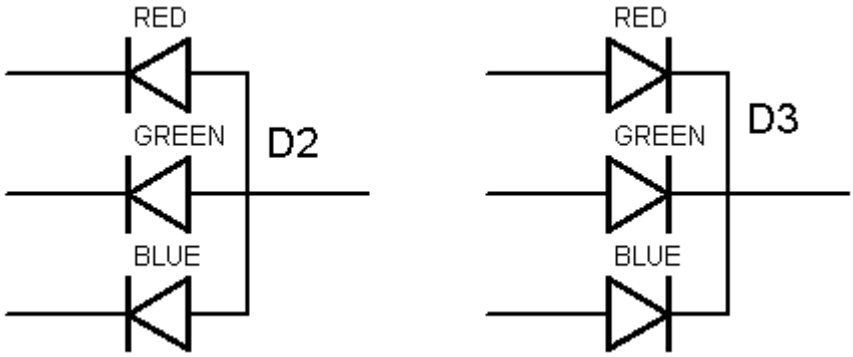
Egy csomagból olvassa ki a feladó IP címét, ezért használata előtt meg kell hívni a parsePacket(); függvényt. Az IP címet egy négy byte-ból álló tömb formájában adja vissza, vagy IP cím osztályként.

```
UDP.remotePort();
```

Egy csomagból olvassa ki a feladó által használt portot, ezért használata előtt meg kell hívni a parsePacket(); függvényt.

11. Színkeverés háromszínű LED dióda segítségével

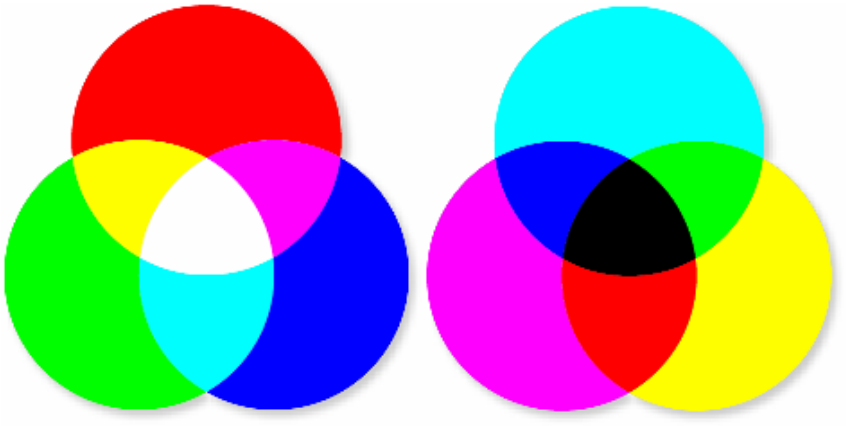
A háromszínű LED diódát a legegyszerűbb úgy értelmezni, mint három külön diódát, amit egy tokba tesznek és négy kivezetéssel látnak el, mivel vagy az anód, vagy katód oldaluk közösítve van, a legtöbbször a katód kivezetést szokták közösíteni.



47. Ábra: Közös anódos és közös katódos RGB LED dióda rajzjele

Színkeverési modellek

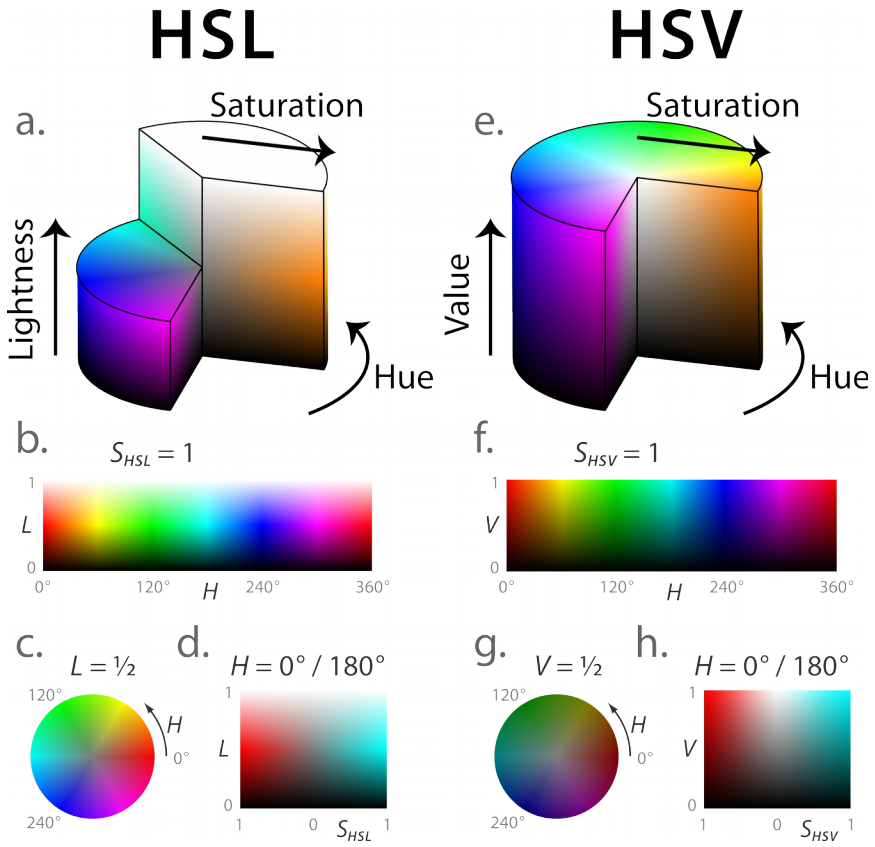
A három alapszín, amelyek kombinálásával mindenféle szín előállítható, a piros, zöld és a kék. Ezt a színkeverési modellt szokás RGB modellnek nevezni. Ezt alkalmazzák a monitorok és az LCD TV-k is.



48. Ábra: RGB és CMYK színkeverési modellek

Ezen kívül más fajta színkeverési modellek is léteznek, például a nyomdaiparban használt négyszínű modell, a CMYK. Ezen kívül a számítógépes rajzolóprogramokban találkozhatunk HSL és HSV színkeverési modellekkel is. Ezek hengeres koordináta-rendszerben ábrázolják az RGB tér színeit.

A modellek között természetesen algoritmusok segítségével konvertálni lehet. A különböző modellek közötti konvertálási algoritmusokról itt részletesen nem lesz szó, mivel csak erről a témáról egy külön fejezetet lehetne írni. Az érdeklődő olvasók figyelmébe így csak egy cikket ajánlanék, amely a matematikai hátteret igen jól elmagyarázza és még mintakódokat is tartalmaz. A cikk a http://www.codeproject.com/Articles/19045/Manipulating-colors-in-NET-Part-1 címen lelhető fel.



49. Ábra: HSL és HSV színalkotási módszerek

Az RGB modellből a bitek száma szerint többfélét megkülönböztetünk: létezik 16, 24 és 32 bites változat. Felépítésileg a 24 bites a legegyszerűbb. Ekkor minden színértéket 8 bit ábrázol, így 16 777 215 különböző szín előállítása lehetséges. A 32 bites modellben az extra 8 bit a szín fényességét szabályozza. Ebben a modellben már több, mint 4 milliárd különböző színt tudunk kikeverni, amely bőven több, mint amit az emberi szem meg tudna különböztetni.

Mielőtt technikailag lehetségessé vált volna ezen modellek széleskörű alkalmazása, kevesebb bitet alkalmaztak az egyes pixelek színinformációinak tárolására. Ezen megoldások közül csak a 16 bites ábrázolással foglalkozok itt részletesebben, hiszen a korábbi, kevesebb bitet alkalmazó megoldásoknak ma már nincs létjogosultsága.

A 16 bites modell esetén abba a problémába ütközünk, hogy a 16 nem osztható maradéktalanul hárommal. Ezért kétfajta 16 bites modell is létezik, attól függően, hogy a csatornákhoz hány bitet rendelünk: az 5R5G5B1X kódnévvel ellátott modell és az 5R6G5B modell. 5R5G5B1X modell esetén minden egyes komponens 5 biten van ábrázolva és egy bit nem használt. Ezt jelképezi az X. A nem használt bit helye lehet a színkód elején vagy a végén is. Az 5R6G5B modell előnye, hogy nincs benne nem használt bit, valamint a tudósok azt vették észre, hogy az emberi szem sokkal érzékenyebb a zöld szín meglétére, mint bármelyik más színre. Ezért kapott ebben a modellben a zöld egy extra bitet.

Megvalósítás

A projekt megvalósításában a 32 bites RGB modellt fogjuk implementálni. Ehhez, hogy a programunk egyszerűbb és később is könnyen felhasználható legyen, készíteni fogunk egy kezelő könyvtárt. Hardverileg 3db PWM képes kimenetet fogunk felhasználni az Arduino-n és egy közös katódos RGB LED diódát, vagy 3db külön LED-et, amelyek katódja közösítve van.

A három komponenssel nem is lesz gond, mivel az Arduino PWM kimenete 8 bites, így itt konverziót nem is kell végeznünk. A fényességet tároló érték pedig egyfajta százalékos szorzóként fog viselkedni minden egyes komponenshez. A kimeneti érték, a szín és a fényesség kapcsolatát az alábbi képlet jól szemlélteti:

$$Kimenet = \frac{fényesség}{255} \cdot színkomponens$$

A kezelőkönyvtár az RGBLed nevet kapta. Szintén három darab fájlból áll. A .h fájl tartalma a következő:

```
#ifndef RgbLedLib_h
#define RgbLedLib_h

#include "Arduino.h"
#include "pins_arduino.h"

#define COLOR-RED           0xFF0000
#define COLOR-BLUE          0x0000FF
#define COLOR-BLACK         0x000000
#define COLOR-WHITE         0xFFFFFF
#define COLOR-DARKRED       0x800000
#define COLOR-PINK          0xFF00FF
#define COLOR-TEAL          0x008080
#define COLOR-LIGHTGREEN    0x00FF00
#define COLOR-GREEN         0x008000
#define COLOR-TURQOISE      0x00FFFF
#define COLOR-DARKBLUE      0x000080
#define COLOR-VIOLET        0x800080
#define COLOR-GRAY25        0xC0C0C0
#define COLOR-GRAY50        0x808080
#define COLOR-DARKYELLOW    0x808000
#define COLOR-YELLOW        0xFFFF00

class RGBLed
{
   private:
        int _pinR, _pinG, _pinB;
        byte _r, _g, _b, _a;
        void HSLtoRGB(float H, float S, float L);
        void longtoRGB(long value);
        void OutputColor();
   public:
        RGBLed(int pinR, int pinG, int pinB);
        void SetAlpha(byte value);
```

```
        void SetRGBValue(byte R, byte G, byte B);
        void SetHSLValue(float H, float S, float L);
        void SetColor(long value);
        void FadeToColor(byte R, byte G, byte B, →
                         unsigned long int fadetime);
        void FadeToColor(byte R, byte G, byte B, byte A, →
                         unsigned long int fadetime);
        void FadeToColor(long color, →
                         unsigned long int fadetime);
};

#endif
```

A cpp tartalma:

```
#include "Arduino.h"
#include "RgbLedLib.h"
#include "pins_arduino.h"

void RGBLed::OutputColor()
{
   analogWrite(_pinR, _r * ((float)_a / 255.0));
   analogWrite(_pinG, _g * ((float)_a / 255.0));
   analogWrite(_pinB, _b * ((float)_a / 255.0));
}

void RGBLed::HSLtoRGB(float H, float S, float L)
{
   if (S == 0)
   {
        _r = L * 255;
        _g = L * 255;
        _b = L * 255;
        return;
   }

   float q = (L<0.5)?(L * (1.0+S)):(L+S - (L*S));
   float p = (2.0 * L) - q;
   float Hk = H/360.0;
   float T[3];
   T[0] = Hk + (1.0/3.0);    // Tr
   T[1] = Hk;                // Tb
   T[2] = Hk - (1.0/3.0);    // Tg

   for(int i=0; i<3; i++)
   {
        if(T[i] < 0) T[i] += 1.0;
        if(T[i] > 1) T[i] -= 1.0;

        if((T[i]*6) < 1) T[i] = p + ((q-p)*6.0*T[i]);
        else if((T[i]*2.0) < 1) T[i] = q;
        else T[i] = p;
   }
```

```
    _r = T[0] * 255;
    _g = T[1] * 255;
    _b = T[2] * 255;
}

void RGBLed::longtoRGB(long value)
{
    _r = (value & 0x00FF0000) >> 16;
    _g = (value & 0x0000FF00) >> 8;
    _b = (value & 0x000000FF);
}

RGBLed::RGBLed(int pinR, int pinG, int pinB)
{
    _pinR = pinR;
    _pinG = pinG;
    _pinB = pinB;
    pinMode(pinR, OUTPUT);
    pinMode(pinG, OUTPUT);
    pinMode(pinB, OUTPUT);
    _r = 255;
    _g = 255;
    _b = 255;
    _a = 255;
    OutputColor();
}

void RGBLed::SetRGBValue(byte R, byte G, byte B)
{
    _r = R;
    _g = G;
    _b = B;
    OutputColor();
}

void RGBLed::SetAlpha(byte value)
{
    _a = value;
    OutputColor(); //refresh color;
}

void RGBLed::SetHSLValue(float H, float S, float L)
{
    HSLtoRGB(H, S, L);
    OutputColor();
}

void RGBLed::SetColor(long value)
{
    longtoRGB(value);
    OutputColor();
}
```

```
void RGBLed::FadeToColor(byte R, byte G, byte B, byte A, →
unsigned long int fadetime)
{
   unsigned long int delayt = fadetime / 255; //255 step fade

   int difR, difG, difB, difA;
   difR = (_r - R) / 255;
   if (R < _r) difR *= -1;
   difG = (_g - G) / 255;
   if (G < _g) difG *= -1;
   difB = (_b - B) / 255;
   if (B < _b) difB *= -1;
   difA = (_a - A) / 255;
   if (A < _a) difA *= 1;

   for (int i=0; i<100; i++)
   {
        _r += difR;
        _g += difG;
        _b += difB;
        _a += difA;
        OutputColor();
        if (delayt != 0) delay(delayt);
   }
}

void RGBLed::FadeToColor(byte R, byte G, byte B, →
unsigned long int fadetime)
{
   FadeToColor(R, G, B, _a, fadetime);
}

void RGBLed::FadeToColor(long int color, →
unsigned long int fadetime)
{
   byte r = (color & 0x00FF0000) >> 16;
   byte g = (color & 0x0000FF00) >> 8;
   byte b = (color & 0x000000FF);
   FadeToColor(r, g, b, fadetime);
}
```

A keywords.txt tartalma:

```
#######################################
# Syntax Coloring Map For RgbLedlib
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################
RGBLed  KEYWORD1
```

```
#######################################
# Methods and Functions (KEYWORD2)
#######################################
RGBLed        KEYWORD2
SetAlpha      KEYWORD2
SetRGBValue   KEYWORD2
SetHSLValue   KEYWORD2
SetColor      KEYWORD2
FadeToColor   KEYWORD2

#######################################
# Constants (LITERAL1)
#######################################
COLOR-RED          LITERAL1
COLOR-BLUE         LITERAL1
COLOR-BLACK        LITERAL1
COLOR-WHITE        LITERAL1
COLOR-DARKRED      LITERAL1
COLOR-PINK         LITERAL1
COLOR-TEAL         LITERAL1
COLOR-LIGHTGREEN   LITERAL1
COLOR-GREEN        LITERAL1
COLOR-TURQOISE     LITERAL1
COLOR-DARKBLUE     LITERAL1
COLOR-VIOLET       LITERAL1
COLOR-GRAY25       LITERAL1
COLOR-GRAY50       LITERAL1
COLOR-DARKYELLOW   LITERAL1
COLOR-YELLOW       LITERAL1
```

A megvalósítás működése

Az RGBLed osztály egyetlen konstruktort definiál, ami három paramétert vár. Ez a három paraméter azon kimenetek azonosítója, amelyeket az RGB LED meghajtására használunk. A három kimenetnek PWM képesnek kell lennie.

Privát függvények

```
void HSLtoRGB(float H, float S, float L);
```

Ez a függvény egy HSL értéket konvertál RGB értékekké, belsőleg akkor van használva, ha a felhasználó meghívja a SetHSLValue függvényt.

```
void longtoRGB(long value);
```

Egy 32 bites egész számot bont fel RGB komponensekre. A szám legfelső byte-ja nincs használva, az a konverzió esetén kimarad. A függvény azért került be, mivel sok esetben a színeket hexadecimális formában adják meg, pl: #3f4c5b.

```
void OutputColor();
```

Frissíti a LED-et, ténylegesen kiteszi a diódára a beállított színt. Minden olyan függvény meghívja, amely módosítja a kimeneti színértéket.

Publikus függvények

```
void SetAlpha(byte value);
```

A szín intenzitását, vagy más szóval a LED globális fényerejét szabályozza. 0 – A LED nem világít, 255 – A LED teljes fényerőn világít.

```
void SetRGBValue(byte R, byte G, byte B);
```

Beállítja a LED színét R, G és B paraméterek alapján.

```
void SetHSLValue(float H, float S, float L);
```

Beállítja a LED színét H, S és L paraméterek alapján.

```
void SetColor(long value);
```

A kimenet színét beállítja egy 32 bites egész szám alapján. A számot érdemes hexadecimális formátumban átadni a függvénynek. Például a 0xc3c3c3 kód szürke színűre állítja be a kimenetet. A kezelőkönyvtár tartalmaz 16 alapszínt is, ezen alapszínek a függvénynek átadhatóak. Az alapszínek COLOR- névvel kezdőnek. Az elérhető színek nevei és értékei a header fájlban találhatóak meg.

```
void FadeToColor(byte R, byte G, byte B, →
                 unsigned long int fadetime);
```

A LED jelenlegi színe és az R, G, B paraméterek által meghatározott szín között átkeverési animációt hoz létre. Ez a függvény a fényerőt nem állítja az átmenet közben. A fadetime által meghatározott idő belsőleg 255 egységre lesz bontva, mivel az átkeverés 255 lépésben történik.

```
void FadeToColor(long color, →
                 unsigned long int fadetime);
```

Működése megegyezik az előzőleg említett függvényével, azzal a különbséggel, hogy a színt egy 32 bites egész szám határozza meg.

```
void FadeToColor(byte R, byte G, byte B, byte A, →
                 unsigned long int fadetime);
```

Működése megegyezik az előzőleg említett függvényekkel, azonban ezen függvény átkeverés közben képes a fényerőt is változtatni.

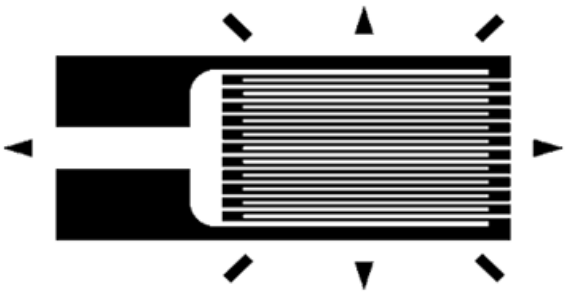
Példaprogram

```
#include <RgbLedLib.h>

RGBLed led(9, 10, 11); //RGB LED on PIN 9, 10, 11

void setup()
{
    led.SetRGBValue(255, 255, 255); //white color;
    delay(1000);
}

void loop()
{
    led.SetAlpha(255); //full opacity
    led.SetRGBValue(255, 0, 0); //red color;
    delay(1000);

    led.SetRGBValue(0, 255, 0); //green color;
    delay(1000);

    led.SetRGBValue(0, 0, 255); //blue color;
    delay(1000);

    led.SetAlpha(128); //half opacity

    led.SetRGBValue(0, 0, 255); //blue color;
    delay(1000);

    led.SetRGBValue(0, 255, 0); //green color;
    delay(1000);

    led.SetRGBValue(255, 0, 0); //red color;
    delay(1000);
}
```

12. Nyúlásmérő bélyegek alkalmazása

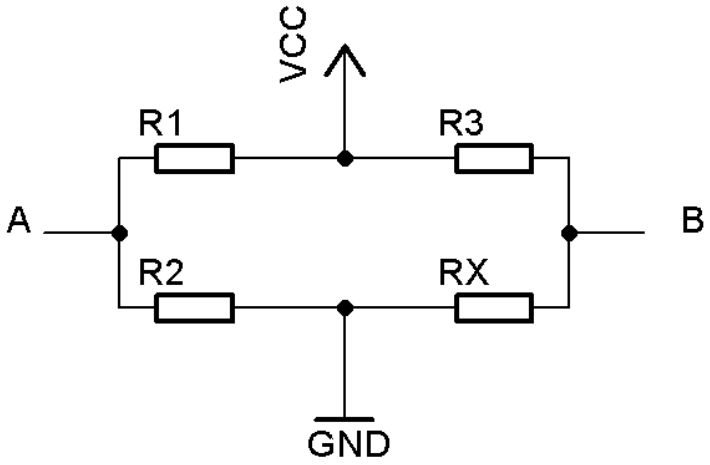
A nyúlásmérő bélyegek belső felépítésének az elve az, hogy a benne alkalmazott villamos vezető anyag ellenállása fizikai erőbehatás (nyúlás, nyomás) hatására megváltozik. Az ellenállás változása, nagyon minimális hibával lineáris az erőbehatáshoz viszonyítva. Feltéve persze, hogy az illesztő kapcsolás és a mérő mechanika helyes.

A mérőcellát illeszteni kell a mikrovezérlőhöz, mivel egy cella általában négy nyúlásmérő bélyeget tartalmaz Wheatstone-hídba kapcsolva. Az illesztés szükségességének másik oka az, hogy az erőbehatás következtében nem csak a cella ellenállása változik, hanem a cellát alkotó anyag fajlagos ellenállása is.



50. Ábra: Egy nyúlásmérő cella belső felépítése

A Wheatstone-híd egy mérőhíd, amelyet Sir Charles Wheatstone fejlesztett ki ellenállások mérésére. A kapcsolásban 4db ellenállás szerepel, amelyek közül egyik értéke nem ismert.



51. Ábra: A Wheatstone-híd

Amennyiben az ábrán jelölt A és B pont között egyik irányba sem folyik áram, akkor a híd kiegyenlített állapotban van. Ebben az esetben az RX ellenállás értéke a következő képlet alapján határozható meg:

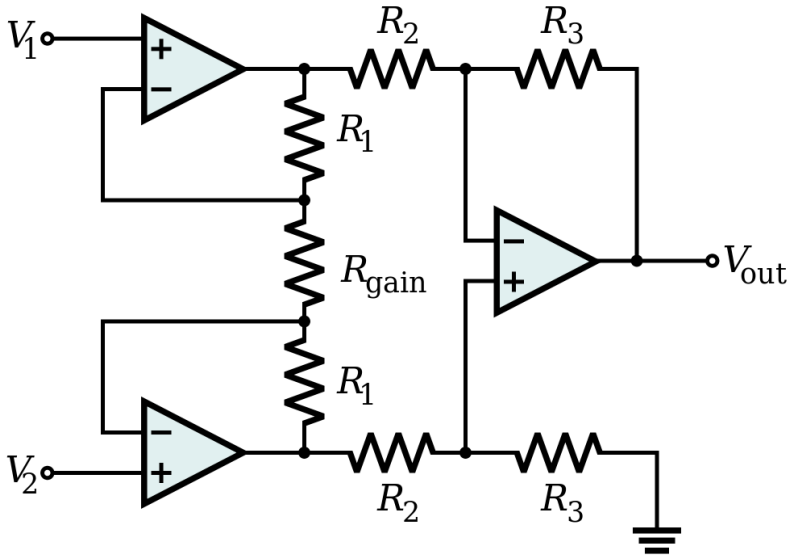
$$RX = \frac{R2}{R1} \cdot R3 = \frac{R3}{R1} \cdot R2$$

Amennyiben a Wheatstone-híd tápfeszültsége, és az R1, R2, R3 ellenállások értéke ismert, valamint az A és B pont között folyó áram kellően kicsi ahhoz, hogy az R1 és R3 ellenállásokon folyó áramokhoz képest elhanyagolható legyen, akkor az A és B pontok között mérhető feszültség az alábbi

képlet alapján határozható meg:

$$U_{AB} = \frac{RX}{R3 \cdot RX} \cdot U_T - \frac{R2}{R1 \cdot R2} \cdot U_T$$

A mikrovezérlőhöz illesztést az úgynevezett instrumention amplifier (mérő erősítő) kapcsolással szokták elvégezni. Ennek két bemenete kapcsolódik a mérőcella kimenetére, majd a föld ponthoz viszonyítva egy feszültségértéket ad ki, amely ezután jól digitalizálható.



52. Ábra: Instrumentation amplifier kapcsolás

A kapcsolás kimeneti feszültsége a bemeneti feszültségek függvényében az alábbi képlet segítségével számolható ki:

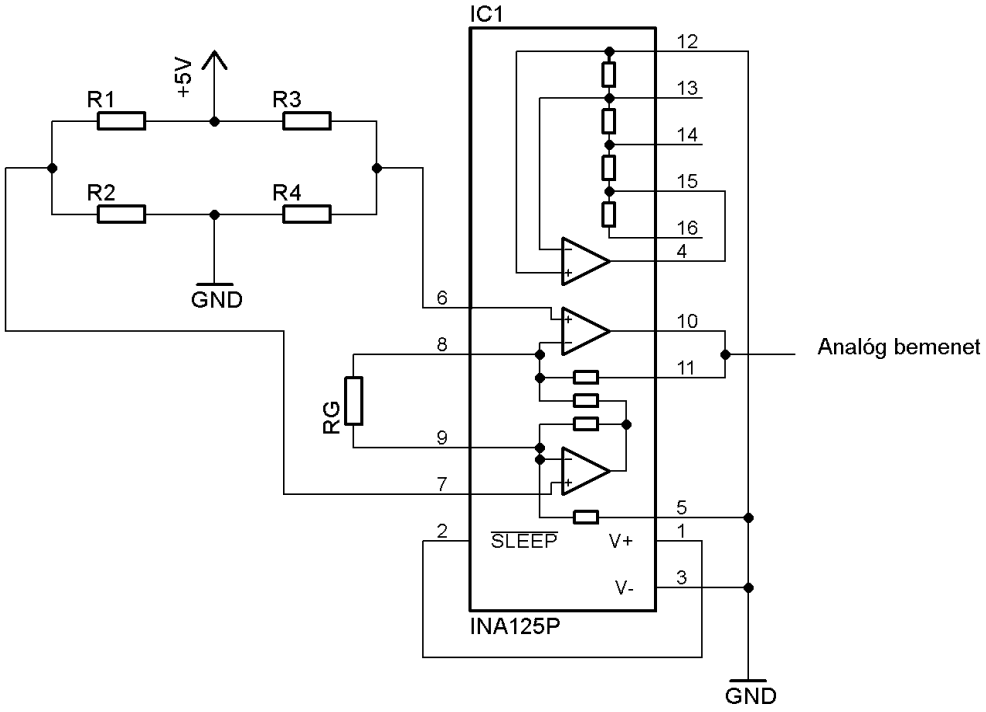
$$U_{Ki} = \left(1 + \frac{2 \cdot R_1}{R_{gain}}\right) \cdot \frac{R3}{R2} \cdot (U_2 - U_1)$$

Az áramkör összeállítható általános célú műveleti erősítőkből is, azonban ebben az esetben soha nem lesz pontos a mérés, az áramköri elemek gyártási hibájából adódóan. Célszerű egy eleve erre a célra integrált áramkört alkalmazni, amely a szükséges működést minimális külső áramköri elemek segítségével tudja biztosítani. Ilyen áramkör például az INA125-ös jelzésű áramkör, amely egyetlen külső ellenállást igényel, ami meghatározza az erősítési tényezőjét.

Az erősítési tényező nagyra állítása mellett is előfordulhat, hogy a kimeneti feszültségérték alacsony lesz. Végtelenségig nem erősíthető a jel, így célszerű egy 16 vagy 24 bites analóg-digitális átalakítót használni a mikrovezérlő beépített 10 bites átalakítója helyett.

Amennyiben a beépített analóg-digitális átalakítót alkalmazzuk, akkor érdemes az analóg referencia feszültséget a lehető legkisebb belső értékre állítani, hogy a pontosságot növeljük.

A mintavételezés sebessége is igen fontos tényező. Az analóg bemeneten az erősítő igencsak nagy impedanciát képvisel, ezért az első cellaérték olvasás a legtöbb esetben pontatlan lesz. Ez kiküszöbölhető a cella értékének kétszeri olvasásával. A két olvasás között 10ms várakozási időt célszerű beiktatni. A második olvasás eredménye jobban fog közelíteni a valósághoz, mint az első olvasásé.



53. Ábra: Mérőcella illesztése Ina125-el Arduino-ra

Az INA125 aszimmetrikus tápfeszültségről is képes működni hiba nélkül, köszönhetően a belső felépítésének.

A mérőcellák árai a maximális terhelhetőségtől és a gyártótól függően igen eltérőek lehetnek. Olcsón régi, használt mérlegekből lehet őket kiszerelni.

A mintaprogram igencsak egyszerű. Ismerni kell az alapértelmezetten a cellára eső súlyt, pontosabban az analóg jel nagyságát, valamint ehhez képest egy ismert súly által kiváltott analóg jel nagyságát. Ekkor a szenzor linearitása miatt minimális hibával meghatározható a súly.

```
float aReading = 0;
float aLoad = 0; //kezdeti súly ami a cellán van grammban
float bReading = 0;
float bLoad = 1000; //ismert súly amit ráteszünk grammban
float diffb = 450; //ismert súly ált. létrehozott analóg jelvált.
long time = 0;
int interval = 500;

int Read(int adc)
{
   analogRead(adc);
   delay(10);
   return analogRead(adc);
}

void setup()
{
   Serial.begin(9600);
   aReading = (float)Read(0);
   bReading = aReading + diffb;
}
```

```
void loop()
{
   float newReading = Read(0);

   float load = ((bLoad - aLoad)/(bReading - aReading)) * →
(newReading - aReading) + aLoad;

   if (millis() > time + interval)
   {
        Serial.print("analog: ");
        Serial.print(newReading,1);
        Serial.print("Suly: ");
        Serial.println(load,1);
        time = millis();
   }
}
```

A Read() függvény egy analóg bemenetet olvas az említett „olvasok, várok olvasok” algoritmus szerint. A külön függvénybe kerülését az indokolta, hogy a mintaprogram során többször előfordul. A setup() függvény inicializálja a soros kommunikációt, valamint kiolvassa a mérőcella jelenlegi értékét. A cella induláskori súlyértékét az aLoad változó tartalmazza.

Az indításkori nullapont felvétel oka abban keresendő, hogy az Arduino analóg-digitális átalakítója nem elég pontos komoly mérőcellás alkalmazásra. Ezért van úgy megírva a program, hogy a kezdeti értékhez viszonyítva vett valamekkora jelváltozást vegyen x grammnak. A jelváltozás értékét a diffb változó tárolja, a jelváltozáshoz társított súlyértékét pedig a bLoad változó.

A mérőcella által kiadott súlyérték csak az aLoad és bLoad változó által meghatározott tartományon belül tekinthető megközelítően pontosnak. Ennek oka az algoritmusban keresendő, amely egyszerű lineáris interpolációt végez. Amennyiben nagyobb súly kerül a cellára, mint a bLoad által meghatározott, akkor a program extrapolációt végez, amely definíciójából adódóan „csak” egy valószínűségi becslés.

13. Több feladat futtatása egyidejűleg

A mai modern operációs rendszerek több program futtatására képesek egy időben. Ezen rendszereket Multitask rendszereknek szokás nevezni. Az igazság azonban az, hogy a programok nem egy időben futnak, hanem időosztással, mivel a processzorok száma jóval kevesebb a futtatott programok számánál.

A processzor időosztás az egyik legmeghatározóbb dolog az operációs rendszeren belül. Ez szabja meg, hogy mennyire jól is képes kihasználni a rendszer a rendelkezésre álló erőforrásokat. Értelemszerűen egy jó ütemező mögött összetett, több ezer sorból álló kód áll.

Mikrovezérlők esetén klasszikus értelemben nem futtathatunk egy időben több feladatot. Ennek oka az, hogy a feladatok futtatását időzítők használatával szokták megoldani, ha szükséges. Arduino esetében az összes hardveres időzítőre jut valami feladat, így itt ezt nem alkalmazhatjuk.

Azonban ez nem jelenti azt, hogy lehetetlen több feladat egyidejű futtatása, csak némileg bonyolult, mivel ehhez egy minimális operációs rendszert kellene írnunk, ami kezeli a feladatainkat és időközönként váltogat köztük. Szerencsére ezt nem kell nulláról megírnunk, mivel valaki ezt már megtette helyettünk.

A projekt az AVR-OS nevet kapta. Ez egy Arduino programkönyvtár, ami egy minimális operációs rendszert implementál. Jelenleg Arduino Uno, Nano és Mega2560 vezérlőket támogat. A működéséhez egy belső időzítőt használ fel. A feladatok ütemezésére a pre-emptive multitasking algoritmust használja, ami azért tökéletes mikrovezérlők számára, mert lehetővé teszi egy feladatnak azt, hogy amíg I/O eszközzel dolgozik, exkluzívan zárolja magát. Ebben az esetben feladatok között váltás nem történik, a feladat addig fog futni, ameddig a zárolást fel nem engedte.

A telepítendő programkönyvtár forrása a https://github.com/chrismoos/avr-os címről szerezhető be. Ez nem egy klasszikus Arduino osztálykönyvtár, mert ez pusztán függvényekből áll, nem alkalmazza az objektumorientált C++ előnyeit.

A könyvtár használata

Első lépésként importálnunk kell a könyvtárat. Ez a művelet a forrásunk elejére beilleszti a következő sorokat:

```
#include <context.h>
#include <os.h>
#include <os_internal.h>
```

Mivel ez a könyvtár C nyelven íródott C++ helyett, ezért a fordítóprogramot utasítani kell arra, hogy a header fájlokat C forráskódnak kezelje. Ehhez a fent említett sorokat extern ”C” blokkba kell tennünk:

```
extern "C"
{
    #include <context.h>
    #include <os.h>
    #include <os_internal.h>
}
```

Ezután létre kell hoznunk egy spinlock_t típusú változót. Ez a változó a megfelelő I/O zárolási állapotot fogja tartalmazni.

Feladatok létrehozása

A feladatok void visszatérési típusú függvények, melyek egy void típusú mutató paramétert vesznek át. A void mutató előnye, hogy ez egy olyan típus, amely segítségével minden létező típust átadhatunk a függvénynek.

A feladatnak tartalmaznia kell egy végtelen ciklust, ami futtatja a feladat által ismétlődő kódot. Amennyiben a feladatunk I/O eszközt szeretne használni, akkor zárolnia kell magát a végrehajtásban, hogy az I/O elérés biztos sikeres legyen és még véletlenül se történjen a feladatok között váltás I/O elérés közben. Az I/O használat végén fel kell engednie a zárolást, hogy az ütemezés folytatódhasson.

Az alábbi függvénnyel tudjuk zárolni a futtatást. A paraméterében szereplő testLock változónév helyére a korábban létrehozott spinlock_t típusú változónk nevét kell írni. Fontos, hogy a változónevet paraméterként adjuk át és ne értékként. Erre szolgál a változónév előtti & jel.

```
spinlock_acquire(&testLock);
```

A zárolás feloldására egy ugyanilyen paraméterezésű függvény szolgál, csak más a neve. A használandó függvény:

```
spinlock_release(&testLock);
```

A feladatok várakoztatására (késleltetésére) a delay függvény nem alkalmas, mivel ez teljesen felborítaná az ütemezést. Helyette a következő függvény használható, amely milliszekundumokban mérhető késleltetést hoz létre:

```
os_sleep(ms);
```

Ütemezés és a feladatok futtatása

A feladatütemező működéséhez a Setup függvényünkben első függvényként meg kell hívnunk az os_init funkciót. Utána előkészíthetjük a programot a szokásos módon. A függvény használatához paraméterek nem kellenek. Definíciója a következő:

```
os_init();
```

A loop funkcióban inicializálni kell a zárolásért felelős változót, valamint el kell indítani a feladatok közötti váltást. Ez a következőképpen történik:

```
spinlock_init(&testLock); //zárolás inicializáció
os_schedule_task(Task1, NULL, 0);
os_loop();
```

A feladatok ütemezésére az os_schedule_task függvény használható. Ennek első paramétere a feladatot ellátó függvény neve, második paramétere a függvénynek átadandó paraméter, amennyiben ez nem lenne, akkor NULL értéket kell átadni neki. A harmadik paraméter pedig a feladat indításának késleltetése milliszekundumban.

Az os_loop függvény futtatja a feladatütemezőt ezután végtelen ciklusban. A loop függvénybe írt kódunk, amely ez után a függvény után szerepel, nem kerül végrehajtásra.

Mintaprogram

Az alábbi mintaprogram bemutatja a könyvtár használatát teljes egészében 2db LED eltérő időközönkénti villogtatásával.

```
extern "C"
{
    #include <context.h>
    #include <os.h>
    #include <os_internal.h>
}

spinlock_t testLock;
int state1, state2;

void setup()
{
    os_init();
    pinMode(10, OUTPUT);
    pinMode(13, OUTPUT);
    state1 = 0;
    state2 = 0;
}

void Task1(void *arg)
{
    while(1)
    {
         if (state1 == 0) state1 = 1;
         else state1 = 0;
         spinlock_acquire(&testLock);
         digitalWrite(10, state1);
         spinlock_release(&testLock);
         os_sleep(1000);
    }
}

void Task2(void *arg)
{
    while(1)
```

```
    {
        if (state2 == 0) state2 = 1;
        else state2 = 0;
        spinlock_acquire(&testLock);
        digitalWrite(13, state2);
        spinlock_release(&testLock);
        os_sleep(250);
    }
}

void loop()
{
    spinlock_init(&testLock);
    os_schedule_task(Task1, NULL, 0);
    os_schedule_task(Task2, NULL, 0);
    os_loop();
}
```

14. Fejlesztés Visual/Atmel Studio segítségével

Az Arduino fejlesztőkörnyezete, ahogy korábban is említettem, nem éppen nevezhető professzionális eszköznek. Szerencsére azonban léteznek más fejlesztőkörnyezetek is, mint például a VisualMicro.

A VisualMicro egy beépülő modul a Microsoft Visual Studio termékébe. Segítségével a Visual Studio összes előnyét élvezve tudunk Arduino programokat írni. Mielőtt a telepítést és a használatot bemutatnám, szólok pár szót a Visual Studio-ról, mivel nem feltétlen kell, hogy az olvasó ismerje ezt a környezetet.

A Visual Studio a Microsoft programozó eszköze. Segítségével az összes Microsoft által forgalmazott platformra tudunk programokat írni. Több programozási nyelvet is támogat az eszköz, ezek közül a legnépszerűbb a C# és talán a C++. Hatalmas előnye a környezetnek, hogy minden támogatott nyelv számára kínál automatikus kódkiegészítést, amelyet InteliSense-nek neveznek. Ez annyira jól működik, hogy aki egyszer hozzászokott, az (saját tapasztalataim alapján) mást nem is szeretne használni.

Visual Studio-ból természetesen több változat is létezik. Alapvetően két csoportra lehet bontani az eszközöket: ingyenes és fizetős változatok.

Az ingyenes (Express) változatok jellemzője, hogy tudásuk a nagy testvérhez képest korlátozott. Ez a korlátozás az esetek többségében olyan funkciók hiányát jelenti, amely funkciókat kezdő/hobbi szinten fejlesztőként nem is igen hiányol az ember. Ezzel azt szeretném kifejezni, hogy az ingyenes nem azért ingyenes, mert nincs benne semmi. Az ingyenesség oka leginkább az, hogy a hobbi fejlesztőkből lesznek később a komoly fejlesztők, akik potenciális vásárlói lesznek a terméknek.

A fizetős (Professional, Ultimate) változatokból többféle létezik, több-kevesebb tudással és természetesen eltérő árképzéssel. A legnagyobb és talán a legzavaróbb különbség az ingyenes és a fizetős változatok között az, hogy az ingyenes változatok tudása nem bővíthető modulokkal. Ennek oka az, hogy pár komoly bővítménnyel a program felokosítható lenne a fizetős változatok szintjére, ami rontaná az üzleti érdekeket.

Mivel a VisualMicro egy bővítmény, így sajnos nem használható az ingyenes Express változattal együtt. Maga a VisualMicro egy ingyenes bővítmény, ingyenesen beszerezhető a http://visualmicro.codeplex.com/ címről. 2008-as, 2010-es és a 2012-es Visual Studio-t is támogatja a modul.

Használatához két előfeltétel is van. A Visual Studio-ba telepítve kell lennie legalább egy alapszintű C++ támogatásnak. Erre azért van szükség, mert másképpen nem működőképes az InteliSense támogatás.

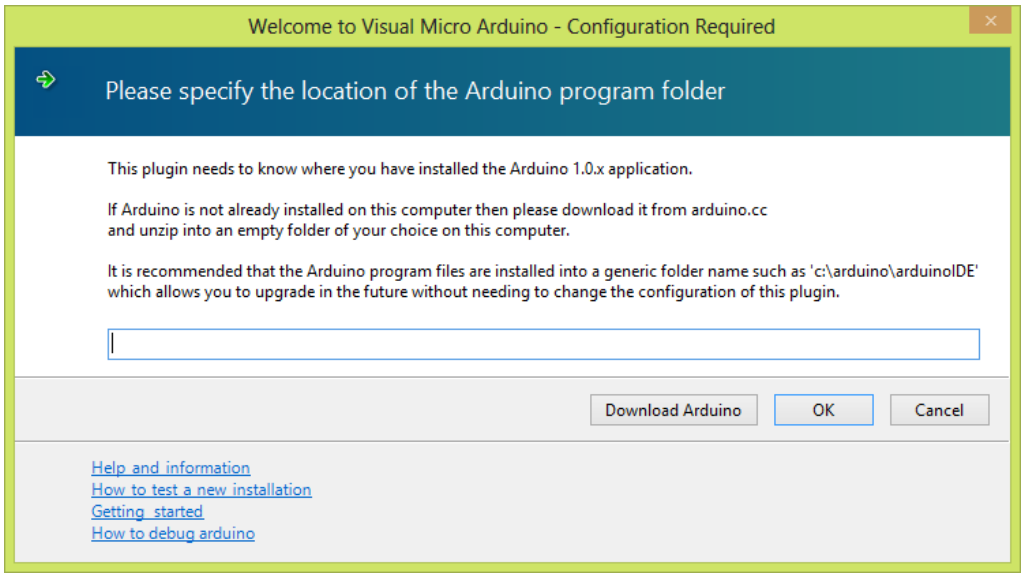
A második előfeltétel az, hogy az Arduino környezetnek is telepíve kell lennie. Ennek oka az, hogy a környezettel járó C++ fordítót és feltöltőket használja a modul.

A könyv írásának pillanatában a modul már támogatja az Atmel hivatalos fejlesztőkörnyezetét, az Atmel Studio-t is. A támogatás egyelőre béta állapotú, vagyis lehetnek benne bőven hibák. Az Atmel

Studio a Visual Studio-n alapul, teljesen ingyenes fejlesztőeszköz. Bővebb információ a termékről a http://www.atmel.com/microsite/atmel_studio6/ címen olvasható.

Telepítés és beüzemelés

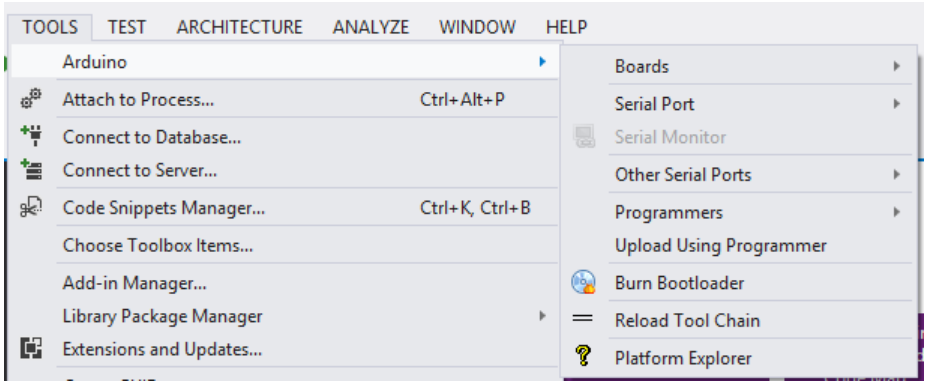
A modul telepítése különösebben nem nehéz, szokásos módon telepíthető. A telepítés után, ha minden jól ment és az előfeltételek megvannak, akkor a Visual Studio indítása után egy ablak fogad bennünket, ahol be kell állítanunk annak a mappának az elérési útvonalát, amelyben az arduino.exe található.



54. Ábra: Arduino telepítés helyének beírása.

Ezután egy ablakot kapunk, amelyben azt kérdezi a bővítmény, hogy aktiválja-e a debugger (hibakereső) funkciót. Ez a része a programnak fizetős, azonban 30 napig ingyenesen kipróbálható, később ez a funkció csak vásárlás után lesz használható. Itt választhatjuk a 30 napos próbát vagy egyszerűen kattinthatunk a mégsem gombra. Ekkor a funkció egyáltalán nem elérhető.

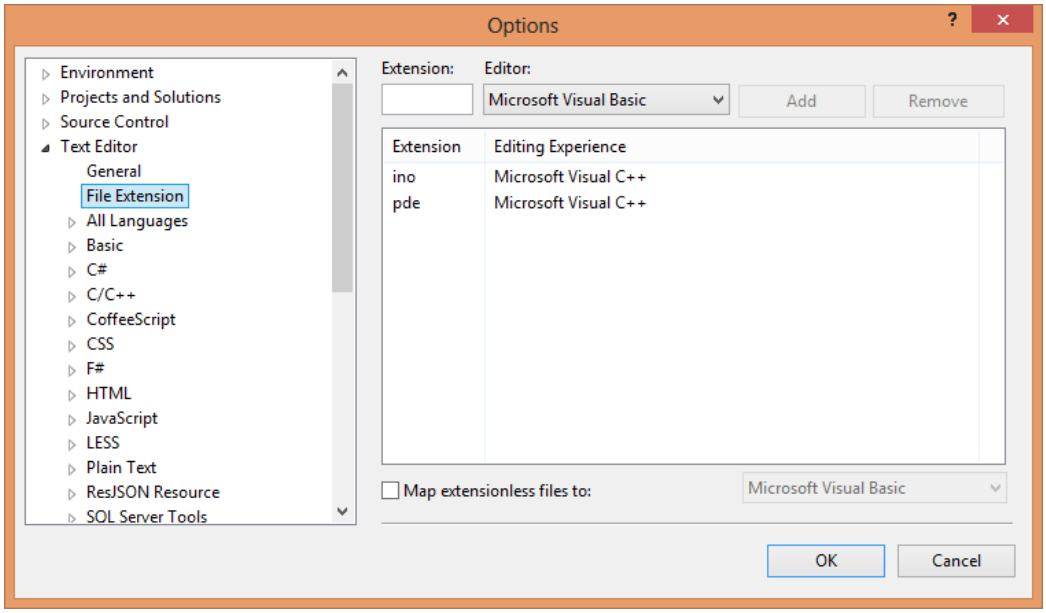
A program indulása után a TOOLS menüben láthatóvá válik egy Arduino menüpont. Innen érhetjük el az Arduino funkciókat.



55. Ábra: Arduino menüpont a Tools menün belül

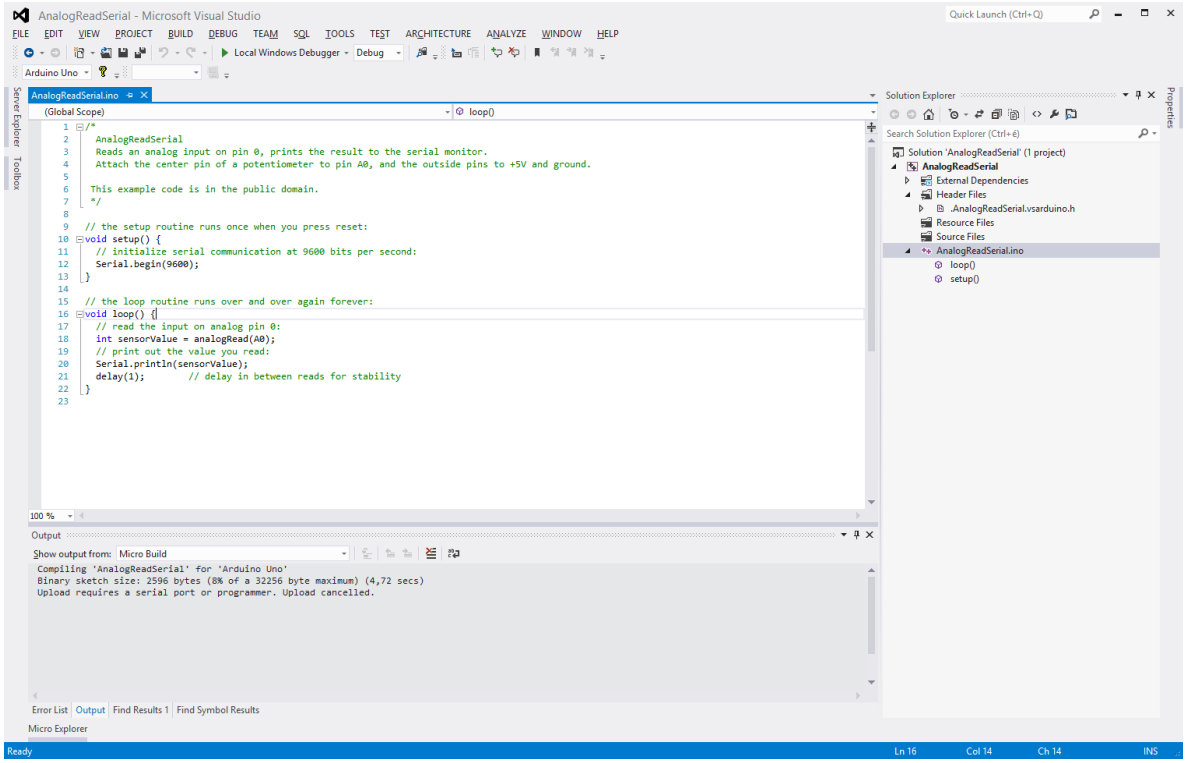
A 2012-es változatban külön nem kellett beállítani az INO és PDE kiterjesztések színezését, azonban a korábbi változatok esetén előfordulhat, hogy be kell állítani. A kiterjesztések a TOOLS →

Options menüpont alatt megjelenő beállítások ablak Text Editor → File Extension menüpontjában állíthatóak be.



56. Ábra: Beállított kiterjesztések

Az Arduino környezet mintaprogramjai a Platform explorer segítségével érhetőek el. A példák megnyitása előtt mindig megkérdezi a modul, hogy az eredetit nyissa-e meg, vagy az alapján hozzon létre egy új projektet. Fordítás közben a kód méretét természetesen megjeleníti a modul, illetve az Arduino környezetbe telepített könyvtárak és extra hardverek támogatása is megoldott.



57. Ábra: Visual Studio VisualMicro bővítménnyel

15. Kommunikáció mikrovezérlők között soros porton Firmata protokollal

A soros portos kommunikáció előnye, hogy nincs kötött protokollja, így kötöttségek nélkül használható adatok továbbítására. Ez az előny hátrányokkal is jár. A legnagyobb hátrány a szabvány hiánya, vagyis szinte biztos, hogy semmi sem fog egymással automatikusan működni.

Ezen probléma megoldására találták ki a Firmata protokollt. Ezt kifejezetten mikrovezérlők egymás közötti kommunikációjához és mikrovezérlők számítógépetekről történő vezérléséhez fejlesztették ki.

A protokollt megvalósító könyvtár az Arduino környezetben gyárilag telepítve van, így nem kell külön telepíteni, azonban mivel folyamatos fejlesztés alatt áll, érdemes a legfrissebb változatot beszerezni a http://firmata.org/wiki/Download címről. Itt az Arduino könyvtár mellett számos programnyelven elkészített kezelőkönyvtárat találunk, amelyek segítségével írhatunk olyan programot a kedvenc programozási nyelvünkön, amely tud beszélgetni a számítógéppel.

A kezelőkönyvtár üzenetek küldésére és üzenetek fogadására alkalmas függvényekből áll, valamint pár darab inicializációs függvényből. Használatukhoz importálni kell a Firmata függvénykönyvtárat.

Inicializáció

```
Firmata.Begin()
```

Inicializálja a könyvtárat.

```
Firmata.begin(long)
```

Inicializálja a könyvtárat az alapértelmezettől (9600) eltérő Baud rátán.

```
Firmata.PrintVersion()
```

Közli a használt protokollverziót a számítógéppel/másik eszközzel. Kompatibilitási célokra alkalmazható.

```
Firmata.blinkVersion()
```

A 13-as lábra kötött LED-en levillogja a protokoll verzióját.

```
Firmata.PrintFirmwareVersion()
```

A firmware nevének és verziójának elküldése a számítógép számára. Használata előtt be kell állítani a verziót. A firmware neve az Arduino környezetben használt név lesz.

```
Firmata.setFirmwareVersion(byte major, byte minor)
```

Beállítja a firmware verzióját. Első paramétere a fő verzió, a második paramétere pedig az alverzió.

Üzenetek küldése

```
Firmata.sendAnalog(byte pin, int value)
```

Egy analóg értéket küld el. Első paramétere az analóg jel által használt láb, második paramétere pedig a lábon olvasható érték.

```
Firmata.sendDigitalPortPair(byte pin, int value)
```

Digitális jel értékek továbbítása. Első paraméter egy port azonosítója, második paramétere pedig egy portérték. A digitális lábak 8 bites portokra vannak bontva. Az első 8 láb (0...7) az első port, míg a maradék 5 láb (8...13) Uno esetén a második port.

```
Firmata.sendSysex(byte command, byte bytec, byte* bytev)
```

Tetszőleges számú bájt továbbítása. Első paramétere egy parancs azonosító, második paramétere a küldendő byte-ok száma, harmadik paramétere pedig a küldendő byte-okat tartalmazó tömb.

```
Firmata.sendString(const char* string)
```

Tetszőleges szöveg továbbítása.

```
Firmata.sendString(byte command, const char* string)
```

Tetszőleges szöveg továbbítása, kiegészítve egy parancs byte-tal, ami az első paraméter.

Üzenetek fogadása

Az üzenetek fogadására eseménykezelők szolgálnak. Az események kezelésére pár függvény szolgál. Ezek:

```
Firmata.available()
```

Ellenőrzi, hogy van-e feldolgozható üzenet az eseménykezelők számára. Működése és visszatérési értéke megegyezik a soros port kezelésnél tárgyalt Serial..available() függvénnyel.

```
Firmata.ProcessInput();
```

Bejövő üzenetek feldolgozása az üzenetkezelők segítségével.

```
Firmata.attach(message, callback);
```

Létrehoz egy üzenetkezelőt. A függvény első paramétere az üzenet típusa, amihez hozzá szeretnénk rendelni az eseménykezelőt, második paramétere pedig az eseménykezelő függvény neve.

Az eseménykezelő függvényeket a Firmata.begin() függvény meghívása előtt kell beállítani.

Az eseménykezelő függvények paraméterezése eltér az üzenetek tekintetében. Az alábbi táblázat az üzenettípusokat foglalja össze a hozzájuk rendelhető függvények definíciójával és leírásával.

Üzenet típusa	Eseménykezelő függvény	Leírás
ANALOG_MESSAGE	void callbackFunction (byte pin, int value)	*Analóg üzenet, első paraméter a feladó láb, második paraméter az érték.*
DIGITAL_MESSAGE		*Digitális üzenet, első paraméter a feladó láb, második paraméter a láb értéke.*
REPORT_ANALOG		*Analóg olvasási kérelem. Első paramétere az olvasandó láb, második paramétere elhanyagolható. Válaszolni analóg üzenettel kell.*
REPORT_DIGITAL		*Digitális olvasási kérelem. Első paramétere az olvasandó láb, második paramétere elhanyagolható. Válaszképpen digitális üzenetet kell küldeni.*
SET_PIN_MODE		*Láb funkció beállítási kérelem. Első paramétere a beállítandó láb, második paramétere a láb funkciója. Az alábbi konstansok közül kerülhet ki az értéke: INPUT, OUTPUT, PWM, ANALOG.*
FIRMATA_STRING	void stringCallbackFunction (char *myString)	*Szöveg fogadására alkalmas üzenetkezelő. Paramétere a fogadott szöveg.*
SYSEX_START	void sysexCallback (byte command, byte argc, byte *argv)	*Általános parancsüzenetek fogadása, első paraméter a parancs byte, második paraméter a paraméterek száma, harmadik paraméter a fogadott byte-ok tömbként.*
SYSTEM_RESET	void systemResetCallback()	*Szoftveres reset kérelem, ha ez érkezik, akkor a firmware-nek szoftveresen újra kell indulnia.*

30. Táblázat: Események típusai és a hozzájuk rendelhető eseménykezelők

A SYSEX_START üzenetkezelő esetén pár üzenettípus alapértelmezetten definiált, ezen üzenetek száma protokoll verziónként eltérő. Az előre definiált üzenetek listája a http://firmata.org/wiki/Protocol címen található meg.

```
Firmata.detach(byte command);
```

Egy üzenettípushoz hozzárendelt eseménykezelő eltávolítása. A paramétere az esemény típusát határozza meg.

Kommunikáció PC és mikrovezérlő között

A Firmata könyvtár tartalmaz egy mintaprogramot, amely a StandardFirmata nevet kapta. Ez szintén megtalálható az Arduino környezetben is, méghozzá a minták között.

Ezt a programot feltöltve a mikrovezérlőre az Aruino lapunk egy univerzális, számítógép által távirányítható lappá válik a megfelelő szoftver használatával.

A korábban említett MCU Tools programba építettem egy Firmata vezérlőpultot, amely segítségével vezérelni tudjuk az Arduino lapunkat. Ezen felül a szoftver tartalmaz egy analóg-digitális átalakító értékének olvasására szolgáló Firmata programot is. Az Arduino modellek közül a vezérlő szoftver a Nano, Uno, Leonardo és Mega1280 modelleket támogatja.

Kommunikáció mikrovezérlők között

Két mikrovezérlőt soros porton keresztül is összeköthetünk. Ebben az esetben a kommunikáció lebonyolításához is érdemes alkalmazni a Firmata protokollt. A következő mintaprogram két mikrovezérlő kommunikációját mutatja be.

Első vezérlő programja:

```
#include <Firmata.h>

void setup()
{
   Firmata.setFirmwareVersion(0, 1);
   Firmata.begin();
}

void loop()
{
   while(Firmata.available()) Firmata.processInput();

   Firmata.sendDigitalPortPair(13, 0);
   delay(1000);
   Firmata.sendDigitalPortPair(13, 1);
   delay(1000);
}
```

Második vezérlő programja:

```
#include <Firmata.h>

void WriteCallback(byte pin, int value)
{
   pinMode(pin, OUTPUT);
   digitalWrite(pin, value);
}

void setup()
{
   Firmata.setFirmwareVersion(0, 1);
   Firmata.attach(DIGITAL_MESSAGE, WriteCallback);
   Firmata.begin();
}

void loop()
{
   while(Firmata.available()) Firmata.processInput();
}
```

A példaprogramban az egyszerűség kedvéért a digitális port értékek küldése nem a Firmata szabvány szerint portokra bontva történik, hanem egyesével van továbbítva minden láb állapota. Egy való élet beli példa esetén ez az átviteli mód nem ajánlott, mivel feleslegesen nagy kommunikációs forgalmat generál, továbbá nem lesz kompatibilis a többi Firmata implementációval.

Az első program igen egyszerű, inicializáció után a loop() függvényben megnézi, hogy van-e feldolgozandó üzenet, majd a második mikrovezérlő 13-as lábra küld egy be-és kikapcsolási kérelmet. A második mikrovezérlő programja fogadja a kéréseket és ennek megfelelően jár el.

16. Nyomógomb pergésmentesítése szoftverrel

A kapcsolók, nyomógombok hardveres pergésmentesítéséről a megszakítások kezelésénél már volt szó. Azonban, ha külön hardverelemet nem szeretnénk a tervünkbe beépíteni és a gombokat/kapcsolókat nem szeretnénk megszakításként használni, akkor a mikrovezérlő programjában is kiküszöbölhető ezen jelenség

Erre két módunk is van. Használhatunk ciklust és várakozást is. A ciklusos mentesítés lényege, hogy addig blokkolva tartjuk a programunkat, amíg a gomb/kapcsoló nyomva van. Erre egy példa:

```
void setup()
{
   pinMode(10, INPUT);
   pinMode(13, OUTPUT);
}

int allapot;

void loop()
{
   if (digitalRead(10) == 1)
   {
        allapot = 1;
        while (digitalRead(10) == 1) {}
   }
   else allapot = 0;
   digitalWrite(13, allapot);
}
```

A könyv első résszének olvasói számára a példa kísértetiesen ismerős lehet, mivel ezen kódot mutattam be a PIC mikrovezérlők esetén is.

Egy másik megoldás lehet az időzítő alapú várakozás. A fenti példa időzített várással működő alternatívája:

```
#define VARAKOZAS 50

int allapot, elozo, gomb;
long ido;

void setup()
{
   pinMode(10, INPUT);
   pinMode(13, OUTPUT);
   allapot = 0;
   elozo = 0;
   gomb = 0;
}

void loop()
{
   allapot == digitalRead(10);
```

```
    if (allapot != elozo) ido = millis();
    if ((millis() - ido) > VARAKOZAS) gomb = allapot;
    else gomb = 0;

    digitalWrite(13, gomb);

    elozo = allapot;
```

A fenti mintakód a hivatalos Arduino oldalon található http://arduino.cc/en/Tutorial/Debounce leírás alapján született. Azonban megjegyzem, hogy ezen kód nem túl hatékony memória és utasítások számának tekintetében sem. Ezért az első példában szereplő kód alkalmazását javaslom. Ahol pedig lehetőség van rá, ott hardveresen küszöböljük ki ezt a problémát.

17. Arduino Uno készítése házilag

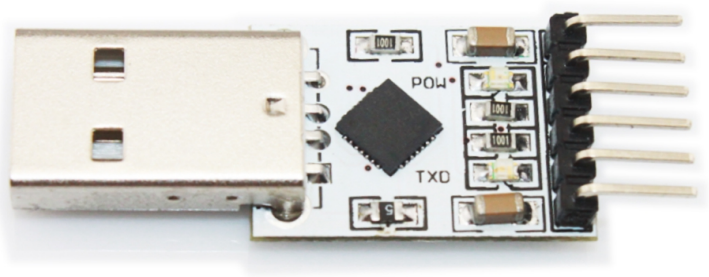
Az Arduino nyílt forrású hardver és szoftver. Ennek ellenére igencsak nehéz házilag a tervrajzok alapján építeni egy klónt bármelyik modellről. Ennek hardveres oka van.

A hardverrel házi gyártás szempontjából az a baj, hogy helytakarékosság és olcsó előállíthatóság miatt jó néhány felület szerelt alkatrészt tartalmaz egy kétoldalas nyomtatott áramkörön. A felület szerelt alkatrészek beültetése és forrasztása szakértelmet és speciális eszközöket igényel.

Az Uno tervrajza alapján terveztem egy saját, csak furat szerelt alkatrészeket tartalmazó Uno klónt. A tervezéskor a következő szempontokat tartottam szem előtt:

1. *A lehető legkevesebb alkatrészt tartalmazza.*
2. *A nyomtatott áramköri terv egy oldalas legyen, átkötésekkel.*
3. *A lábak kiosztása és a tüskesorok pozicionálása megegyezzen az eredeti modellel.*

Az USB kapcsolat és programozás lehetőségét úgy valósítottam meg, hogy egy tüskesort tettem az USB csatlakozó helyére. Erre egy FTDI chip-et alkalmazó olcsó USB átalakító köthető be. Ezen átalakítók igen változatos lábkiosztással vannak felszerelve és különböző méretben, kivitelben szerezhetőek be. A külső átalakítóra támaszkodás oka abban keresendő, hogy furat szerelt kivitelben nem lehet találni olyan integrált áramkört, ami USB – RS232 átalakítást végezne.



58. Ábra: FTDI Chip alapú USB RS232 (TTL) átalakító

Persze lehetett volna erre a célra egy általános mikrovezérlőt beprogramozni, de az nem biztos, hogy képes lenne külső órajel nélkül a programozási sebességre. Ha külső órajel is kell, akkor az még több alkatrészt jelent. Ezen felül Windows-ra illesztőprogramot is kell írni, ami nem egyszerű feladat, így az FTDI chip ebből a szempontból is jobb választás, mivel az Arduino környezet tartalmaz hozzá illesztőt, mert az Uno előtti változatok szintén FTDI chip-et alkalmaztak USB – soros konverzióra

A lábkiosztás kompatibilitása részleges, mivel a programozó ISP port nem ugyanott van, mint a gyári modellen, valamint a digitális oldalon található I^2C busz vezetékek nincsenek bekötve.

Ezen okokból, és mert a terv nyílt forráskódú, nem tudok rá vállalni semmilyen felelősséget. A kapcsolási rajz igen egyszerű felépítésű, megtalálható a mellékletek fejezetben és a CD mellékleten is a nyomtatott áramköri tervvel együtt.

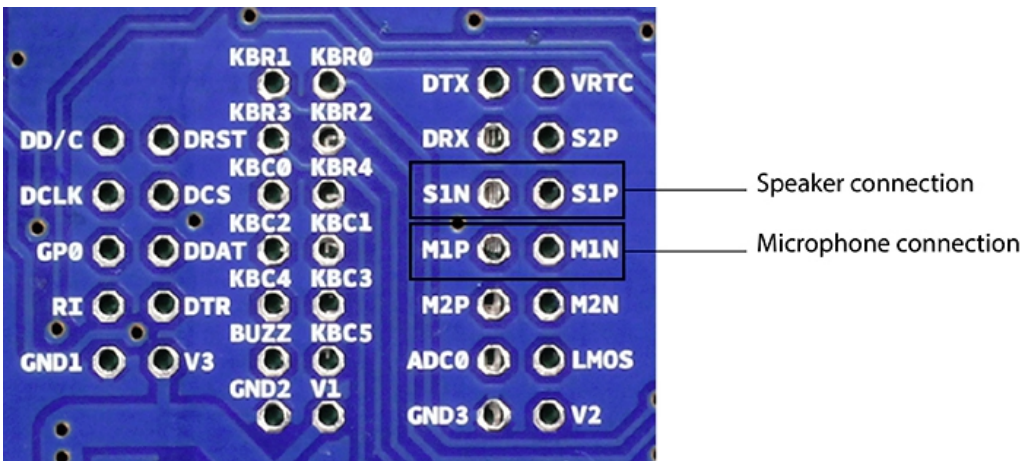
18. GSM modul kezelése

Az Arduino platform nem sokkal a könyv első változatának megjelenése után kapott egy GSM modult, amely segítségével teljes értékű mobiltelefonná változtathatjuk. Küldhetünk vele SMS-t, hívást kezdeményezhetünk és fogadhatunk, valamint az internetre is kapcsolódhatunk, mivel a modul képes GPRS kommunikációra is, amely 56-114Kbit/s adatátvitelt tesz lehetővé.

59. Ábra: Arduino és a GSM modul

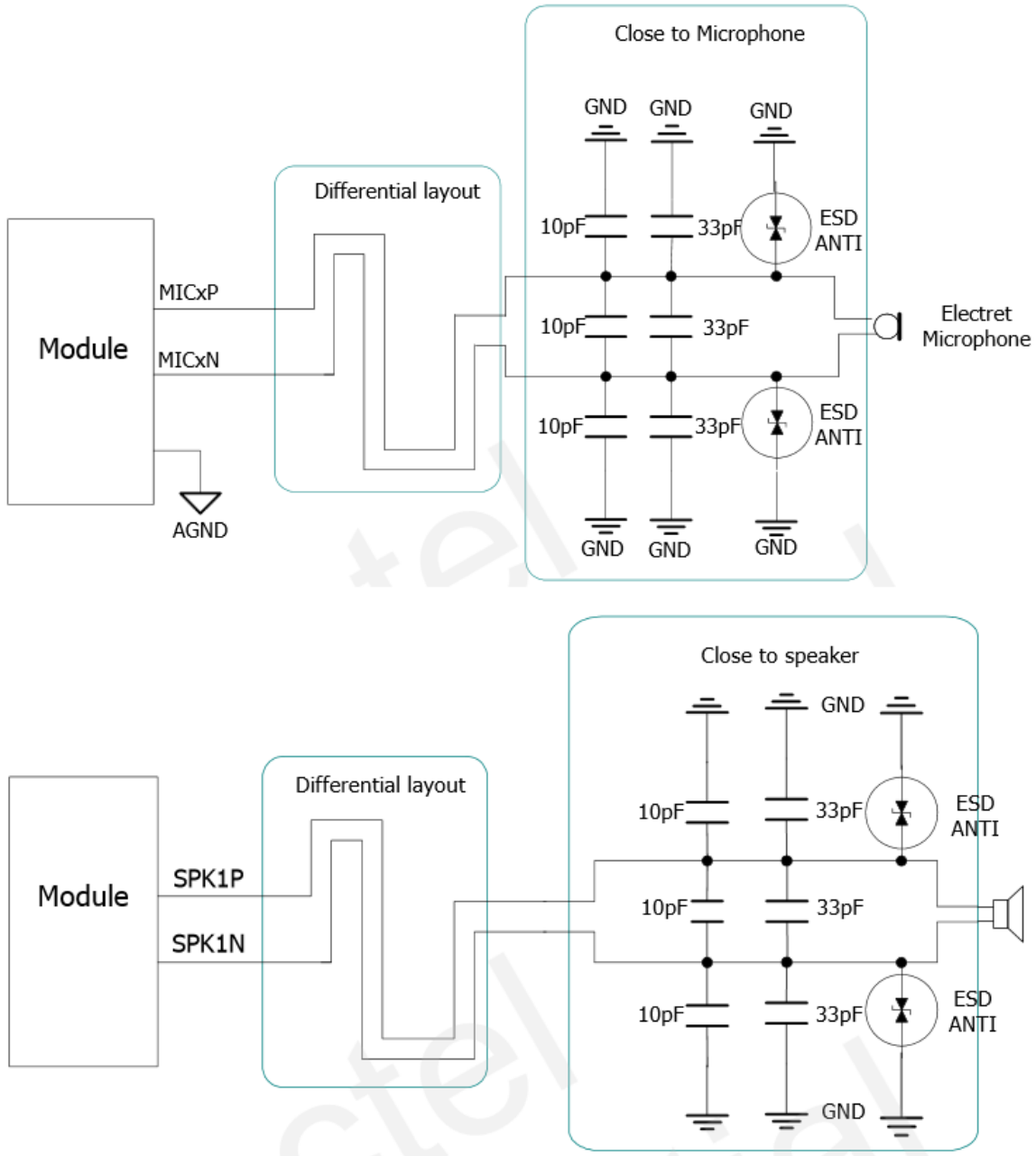
Amennyiben a modul csatlakozik az Arduino-hoz, akkor nem ajánlott USB-ről táplálni, mivel az USB csatlakozó szabvány szerint 500mA áramot tud szolgáltatni. Terhelt állapotban a modulon található modem fogyasztása elérheti a 800mA-t is, amely azt eredményezi, hogy a számítógép bontani fogja a kapcsolatot az Arduino-val, így igen instabillá válhat a működés.

A modullal hanghívás kezdeményezése és fogadása is lehetséges, azonban ehhez csatlakoztatni kell a modulhoz hangszórót és mikrofont. Ezek kivezetései a modul hátlapján lévő forrasztási pontokra vannak kivezetve. Az alábbi ábrán a szükséges csatlakozások láthatóak.



60. Ábra: A szükséges csatlakozási pontok

A modulon található modem dokumentációja alapján a mikrofon és a hangszóró helyes bekötése az alábbi ábrán látható.



61. Ábra: Hangszóró és mikrofon kapcsolása a modulhoz

A modul soros üzenetekkel kommunikál az Arduino-val, azonban ezt nem a hardveres soros kommunikációt biztosító 0 és 1 lábakon teszi meg, hanem szoftveresen, a 2-es és a 3-as lábakon. Továbbá a 7-es lábat arra használja fel a modul, hogy alaphelyzetbe állítsa a modemet.

A kezelőkönyvtárról

A kezelőkönyvtár a legtöbb osztályból álló Arduino könyvtár. Az 1.04-es kiadás óta részét képezi a hivatalos Arduino kiadásnak.

GSM osztály

A GSM funkciók alaposztálya. Inicializálja a modemet és előkészíti használatra. Az osztály két konstruktort definiál, egy paraméterest és egy paraméter nélkülit.

```
GSM gsm
GSM gsm(debug)
```

A paraméteres konstruktor hibakeresési funkciókat lát el. Amennyiben a paraméter értéke igaz (true), akkor a soros porton keresztül kiírja a függvények által végrehajtott AT parancsokat is.

Tagfüggvények

```
gsm.begin();
gsm.begin(pin);
gsm.begin(pin, restart);
gsm.begin(pin, restart, sync);
```

A modemet hálózatra kapcsolja. Leggyakrabban a függvény egyparaméteres változata használt. A több paraméteres változatok paraméterkiosztása függvényenként megegyezik. Az első paraméter a PIN kódot határozza meg. A PIN kód szövegként van tárolva. A második és a harmadik paraméter a modem újraindítását és a szinkron vagy aszinkron kapcsolatot határozzák meg logikai értékekként. Amennyiben ezeket a paramétereket nem adjuk meg, akkor ezek értékét igaznak tekinti, vagyis kapcsolódás után a modem újraindul és szinkron adatátviteli módba kapcsol.

A függvény visszatérési értéke 0, ha az átviteli mód aszinkron. Szinkron módban az alábbi konstansok értéke közül vehet fel egyet: ERROR, IDLE, CONNECTING, GSM_READY, GPRS_READY, TRANSPARENT_CONNECTED. Amennyiben a kapcsolódás közben nem volt hiba, GSM_READY üzenetet kell visszakapnunk.

```
gsm.shutdown();
```

Bontja a GSM kapcsolatot.

GSMVoiceCall osztály

Hanghívások lebonyolítására alkalmas osztály, konstruktora paraméter nélküli.

```
GSMVoiceCall voice;
```

Tagfüggvények

```
voice.getVoiceCallStatus();
```

A hívás állapotáról ad vissza értéket konstansok formájában. Visszatérési értéke az alábbi konstansok közül kerülhet ki:

Konstans	Leírás
IDLE_CALL	*Nincs hívási esemény, nem történik semmi.*
CALLING	*Hívás kezdeményezése folyamatban.*
RECEIVINGCALL	*Beérkező hívás.*
TALKING	*Hanghívás folyamatban.*

31. Táblázat: Hívás állapot konstansok és leírásuk

```
voice.ready();
```

Visszatérési értéke igaz, ha az előző hívással kapcsolatos függvény végrehajtása befejeződött, hamis, ha még folyamatban van.

```
voice.voiceCall(number);
```

Hívás indítása. A paraméter a telefonszámot határozza meg. A telefonszámot szövegként kell átadni a függvénynek és nemzetközi formátumban javasolt megadni.

```
voice.answerCall();
```

Bejövő hívás fogadására szolgál.

```
voice.hangCall();
```

Folyamatban lévő hívás megszakítása.

```
voice.retrieveCallingNumber(number, size);
```

Hívó fél számának lekérdezésére szolgál. Az első paramétere egy karaktertömböt határoz meg, amelybe beleírja a hívó fél telefonszámát, második paramétere pedig a karaktertömb méretét állítja be.

GSM_SMS osztály

SMS üzenetek küldésére és fogadására alkalmas osztály, konstruktora paraméter nélküli. Egy SMS-ben 160 karakter továbbítható.

```
GSM_SMS SMS;
```

Tagfüggvények

```
SMS.beginSMS(number);
```

Megkezdi egy SMS üzenet létrehozását. Paramétere a címzett telefonszámát határozza meg. A telefonszámot szövegként kell átadni a függvénynek és nemzetközi formátumban javasolt megadni.

```
SMS.ready();
```

Visszatérési értéke igaz, ha az előző SMS küldéssel kapcsolatos függvény végrehajtása befejeződött, hamis, ha még folyamatban van.

```
SMS.endSMS();
```

A modem számára azt jelzi, hogy az SMS üzenet készen áll a küldésre. A függvény hívása után a modem elküldi az üzenetet.

```
SMS.available();
```

Ellenőrzi, hogy van-e bejövő, feldolgozatlan SMS üzenet a modemen. Visszatérési értéke a feldolgozatlan SMS üzenet karaktereinek száma.

```
SMS.remoteNumber(number, size);
```

Az SMS feladó számának a lekérdezésére szolgál. Az első paramétere egy karaktertömböt határoz meg, amelybe beleírja a feladó telefonszámát, második paramétere pedig a karaktertömb méretét állítja be.

```
SMS.read();
```

Egy byte adatot olvas ki az aktuális SMS-ből.

```
SMS.write(val);
```

Egy byte írása az aktuális SMS-be

```
SMS.peek();
```

Az SMS üzenetből beolvas egy byte-ot, azonban a bufferből nem távolítja el az adatot.

```
SMS.print(message);
```

Egy karaktertömböt (szöveget) ír az SMS-be. Paramétere az üzenetbe írandó karaktertömböt határozza meg.

```
SMS.flush();
```

A modemből eltávolítja az üzeneteket. Az éppen küldés alatt lévő üzeneteket csak akkor távolítja el, ha a küldés befejeződött.

GPRS osztály

GPRS alapú mobilinternet használatát megvalósító osztály. Konstruktora paraméter nélküli.

```
GPRS gprs;
```

Az osztály csak egyetlen tagfüggvénnyel rendelkezik.

```
grps.attachGPRS(APN, user, password);
```

Csatlakozik a megadott hozzáférési ponthoz és aktiválja a GPRS adatátvitelt. Az első paramétere a hozzáférési pont címét határozza meg, második paramétere a felhasználónév, harmadik paramétere pedig a szükséges jelszó. A paramétereket szövegként kell átadni. A szükséges adatokat a szolgáltatónak kell biztosítania. Az alábbi táblázat a Magyarországi szolgáltatók adatait tartalmazza tájékoztató jelleggel.

Szolgáltató	***APN név***	***Felhasználónév***	***Jelszó***
T-Mobile	*internet*	*wap*	*Wap*
Vodafone	*wap.vodafone.net*	*vodawap*	*vodawap*
Telenor	*online*		

32. Táblázat: Legnagyobb magyarországi mobilszolgáltatók beállításai

GSMClient osztály

GPRS hálózaton történő adatátvitelhez TCP kliens osztály. Felépítésében és működésében hasonló a TCP protokoll klienshez. Konstruktora paraméter nélküli.

```
GSMClient client;
```

Tagfüggvények

```
client.ready();
```

Visszatérési értéke igaz, ha az előző küldéssel/fogadással kapcsolatos függvény végrehajtása befejeződött, hamis, ha még folyamatban van.

```
client.connect(ip, port);
```

Csatlakozik egy szerverhez. Első paramétere a szerver címét határozza meg, második paramétere a kommunikációra használt port száma. A szerver IP címét szövegként adjuk át, míg a port egész szám.

```
client.beginWrite();
```

Jelzi a kliens a szervernek, hogy üzenetet szeretne küldeni számára.

```
client.write(data);
client.write(buffer);
client.write(buffer, size);
```

A szerver felé küldendő üzenetbe ír. Egyparaméteres változatában a paraméter egy byte adatot határoz meg, vagy egy buffer tömböt. Kétparaméteres változatában az első paraméter a buffer tömb neve, a második pedig a küldendő byte-ok száma.

```
client.endWrite();
```

Befejezi a szervernek szánt üzenet összeállítását és elküldi az üzenetet.

```
client.connected();
```

Ellenőrzi, hogy a kliensnek sikerült-e csatlakoznia.

```
client.read();
```

Egy byte adatot olvas a szerver által küldött üzenetből.

```
client.available();
```

A szerver által küldött üzenet hosszát adja vissza byte-ban.

```
client.peek();
```

Az üzenetből beolvas egy byte-ot, azonban a bufferből nem távolítja el az adatot.

```
client.stop();
```

Bontja a kapcsolatot a szerverrel.

```
client.flush();
```

Eltávolítja a bejövő üzenet bufferből az összes fel nem dolgozott adatot.

GSMServer osztály

GPRS hálózaton történő adatátvitelhez TCP szerver osztály. Felépítésében és működésében hasonló a TCP protokoll szerverhez. Nem minden mobilinternet szolgáltató engedélyez hozzáférést a hálózatán működő szerverhez a publikus internet számára. Nagy eséllyel a szerverhez csak akkor lehet csatlakozni, ha a kliens is ugyanazon a mobilhálózaton van. Konstruktorának paramétere a kommunikációban használt portot határozza meg.

```
GSMServer server(port);
```

Tagfüggvények

```
server.ready();
```

Visszatérési értéke igaz, ha az előző küldéssel/fogadással kapcsolatos függvény végrehajtása befejeződött, hamis, ha még folyamatban van.

```
server.beginWrite();
```

Jelzi a csatlakozott klienseknek, hogy a szerver üzenetet szeretne küldeni számukra.

```
server.write(data);
server.write(buffer);
server.write(buffer, size);
```

A klienseknek küldendő üzenetbe ír. Egyparaméteres változatában a paraméter egy byte adatot határoz meg, vagy egy buffer tömböt. Kétparaméteres változatában az első paraméter a buffer tömb neve, a második pedig a küldendő byte-ok száma.

```
server.endWrite();
```

Befejezi a klienseknek szánt üzenet összeállítását és elküldi az üzenetet minden csatlakozott kliensnek.

```
server.read();
```

Egy bájt adatot olvas a kliens által küldött üzenetből.

```
server.available();
```

Bejövő kapcsolatokat figyeli. Visszatérési értéke a csatlakozott kliensek száma.

```
server.stop();
```

Leállítja a szervert, bontja a kapcsolatot az összes csatlakozott klienssel.

GSMModem osztály

A GSM modemhez kapcsolódó diagnosztikai függvényeket biztosít. Konstruktora paraméter nélküli.

```
GSMModem modem
```

Az osztály csupán két függvényt tartalmaz.

```
modem.begin();
```

Ellenőrzi a modem állapotát és újraindítja. Visszatérési értéke 1, ha minden rendben van a modemmel. Ettől eltérő szám hibára utal. A függvényt meg kell hívni, mielőtt az osztály másik függvénye használható lenne.

```
modem.getIMEI();
```

Szövegként visszaadja a modem egyedi IMEI számát.

GSMScanner osztály

A hálózathoz kapcsolódó diagnosztikai függvényeket biztosít. Konstruktora paraméter nélküli.

```
GSMScanner scanner;
```

Tagfüggvények

```
scanner.begin();
```

Alapállapotba helyezi (újraindítja) a modemet. Meghívása kötelező az osztály többi függvényének használata előtt. Visszatérési értéke 1, ha minden rendben van a modemmel. Ettől eltérő szám hibára utal.

```
scanner.getCurrentCarrier();
```

Szövegként visszaadja a jelenleg használt hálózat nevét.

```
scanner.getSignalStrength();
```

Térerő lekérdezésére szolgál. Visszatérési értéke egy 0 és 31 közötti szám, 31 jelzi azt, hogy a térerő 51dBm felett van, 0 pedig azt, hogy a térerő igen rossz. Amennyiben a függvény visszatérési értéke 99, az azt jelzi, hogy nincs térerő.

```
scanner.readNetworks();
```

Elérhető hálózatok listáját adja vissza szövegként.

GSMBand osztály

A modem által használt hálózati frekvenciákhoz biztosít diagnosztikai funkciókat. Konstruktora paraméter nélküli. A használható mobil frekvenciákról a http://www.worldtimezone.com/gsm.html oldal ad részletesebb felvilágosítást.

```
GSMBand band;
```

Tagfüggvények

```
band.begin();
```

Alapállapotba helyezi (újraindítja) a modemet. Meghívása kötelező az osztály többi függvényének használata előtt. Visszatérési értéke 1, ha minden rendben van a modemmel. Ettől eltérő szám hibára utal.

```
band.getBand();
```

Lekérdezi az aktuálisan használt frekvenciát. Visszatérési értéke az alábbi konstansok közül kerülhet ki: GSM_MODE_UNDEFINED, GSM_MODE_EGSM, GSM_MODE_DCS, GSM_MODE_PCS, GSM_MODE_EGSM_DCS, GSM_MODE_GSM850_PCS, GSM_MODE_GSM850_EGSM_DCS_PCS.

```
band.setBand(type);
```

Beállítja a modem által használt frekvenciát. Visszatérési értéke igaz, ha a beállítás sikerült, hamis, ha nem. Paramétere a getBand függvénynél felsorolt konstansok közül kerülhet ki.

GSMPIN osztály

A SIM kártya PIN és PUK kódjának kezeléséhez biztosít függvényeket. Konstruktora paraméter nélküli.

```
GSMPIN pin;
```

Tagfüggvények

```
pin.begin();
```

Alapállapotba helyezi (újraindítja) a modemet. Meghívása kötelező az osztály többi függvényének használata előtt. Visszatérési értéke 1, ha minden rendben van a modemmel. Ettől eltérő szám hibára utal.

```
pin.isPIN();
```

Ellenőrzi, hogy a kártya használatához szükséges-e PIN kód. Visszatérési értéke 1, ha kell PIN kód, 0, ha nem kell, –1, ha a kártya PUK lezárási módban van, –2, ha valami hiba van a kártyával.

```
pin.checkPIN(PIN);
```

Ellenőrzi, hogy a paraméterként megadott PIN kód helyességét. Visszatérési értéke 0, ha a kód helyes, –1, ha nem helyes. A legtöbb SIM kártya 3 helytelen kód próbálkozás után PUK kóddal lezárt állapotba kerül. A paraméter szövegként adandó át.

```
pin.checkPUK(puk, pin);
```

Ellenőrzi az első paraméterként megadott PUK kód helyességét. Amennyiben a kód helyes, akkor a második paraméterként megadott PIN kód lesz az új PIN kód. Visszatérési értéke 0, ha minden rendben volt, –1, ha nem. A paraméterek szövegként adandóak át.

```
pin.changePIN(oldPIN, newPIN);
```

PIN kód megváltoztatása. Első paramétere a jelenlegi PIN kód, második paramétere az új PIN kód. A paraméterek szövegként adandóak át.

```
pin.switchPIN(pin);
```

PIN kód lezárás feloldása. Paramétere a PIN kódot határozza meg és szövegként kell átadni a függvénynek.

```
pin.checkReg();
```

Ellenőrzi, hogy a modemnek sikerült-e csatlakoznia a hálózatra. Visszatérési értéke 0, ha igen, 1, ha roaming-al csatlakozott a kártya, –1, ha hiba történt.

```
pin.getPinUsed();
```

Ellenőrzi, hogy a kártya használatához kell-e PIN kód. Visszatérési értéke igaz, ha kell, hamis, ha nem.

```
pin.setPinUsed(used);
```

PIN kód használatának ki-és bekapcsolása paramétertől függően. Ha a paramétere igaz, akkor kell PIN kód, ha hamis, akkor nem kell.

19. Billentyűzet, egér emulálása

Az Arduino Leonardo-n és Due-n használt vezérlő hardveres USB támogatással rendelkezik, ami lehetővé teszi azt, hogy billentyűzetként vagy egérként is viselkedjen. Ez azért lehetséges, mivel a billentyűzet és az egér HID USB kategóriába esik. Ez az emberi beviteli eszközök kategóriája az USB szabványban. Ide tartoznak többek között a játékvezérlők és a különböző érintő felületek is.

Viszonylag szabványos a protokoll felépítése billentyűzet és egér esetén, mivel ezen eszközök nem annyira egyediek, mint mondjuk egy játékvezérlő. Játékvezérlők és egyéb HID kategóriás eszközök esetén a működéshez létre kellene hozni egy protokoll definíciót, ami alapján a kommunikáció lehetséges lenne. Ezt egyelőre az Arduino környezet nem támogatja, így „csak" billentyűzet és egér emulálására van lehetőségünk. Azonban ezzel is igen sok mindent meg tudunk tenni.

Egér emuláció

Az egér emuláció megvalósításáért a Mouse osztály felelős. Ezen osztály használatához nem kell importálni könyvtárat, példányosítani sem kell. Tagfüggvényein keresztül használható.

Tagfüggvények

```
Mouse.begin();
```

Megkezdi az egér emulációt. A többi emulációs függvény használata előtt meg kell hívni.

```
Mouse.click();
Mouse.click(button);
```

Egérkattintás küldése a számítógépnek. Paraméter nélküli változata a bal gomb lenyomását szimulálja. Paraméteres változatában a kattintás gombja meghatározható. A paraméter értéke az alábbi konstansok közül kerülhet ki: MOUSE_LEFT (bal gomb, alapértelmezett), MOUSE_RIGHT (jobb gomb) MOUSE_MIDDLE (középső gomb)

```
Mouse.move(xVal, yPos, wheel);
```

Egér mozgás szimulálása. A paraméterek előjeles egész számok, amelyek a mozgást határozzák meg pixelben, az egér aktuális pozíciójához viszonyítva. Első paramétere az x tengely menti elmozdulás, második paramétere az y tengely menti elmozdulás, harmadik paramétere a görgő elmozdulása.

```
Mouse.press();
Mouse.press(button);
```

Egér gomb lenyomásának szimulálása. A függvény hívása a számítógépnek azt jelzi, hogy egy gomb folyamatosan nyomva van. Paraméter nélküli változata a bal gomb lenyomását szimulálja. Paraméteres változatában a paraméter a gombot határozza meg. A gomb meghatározására a click függvény esetén tárgyalt konstansok használhatóak.

```
Mouse.release();
Mouse.release(button);
```

Egér gomb felengedésének szimulálása. A függvény hívása a számítógépnek azt jelzi, hogy egy

lenyomott gomb felengedésre került. Paraméter nélküli változata a bal gomb felengedését szimulálja. Paraméteres változatában a paraméter a gombot határozza meg. A gomb meghatározására a click függvény esetén tárgyalt konstansok használhatóak.

```
Mouse.isPressed();
Mouse.isPressed(button);
```

Ellenőrzi, hogy egy gomb lenyomott állapotban van-e, vagy sem. Visszatérési értéke igaz, ha a gomb lenyomva van, hamis, ha nem. Paraméter nélküli változata a bal gomb lenyomását teszteli. Paraméteres változatában a paraméter a gombot határozza meg. A gomb meghatározására a click függvény esetén tárgyalt konstansok használhatóak.

```
Mouse.end();
```

Egér emuláció kikapcsolása.

Billentyűzet emulálása

A billentyűzet emuláció megvalósításáért a Keyboard osztály felelős. Ezen osztály használatához nem kell importálni könyvtárat, példányosítani sem kell. Tagfüggvényein keresztül használható.

Tagfüggvények

```
Keyboard.begin();
```

Billentyűzet emuláció bekapcsolása. A többi emulációs függvény használata előtt meg kell hívni.

```
Keyboard.print(character);
Keyboard.print(characters);
```

Gomb leütésének vagy szöveg gépelésének szimulálása. A paramétere a leütendő karaktert határozza meg, vagy a begépelendő szöveget. Visszatérési értéke az elküldött karakterek száma. Nem nyomtatható ASCII karakterek átvitele nem javasolt, mivel kiszámíthatatlan működést eredményezhet.

```
Keyboard.println();
Keyboard.println(character);
Keyboard.println(characters);
```

Hasonló a print függvényhez, azonban a gomb leütés / szöveg gépelés után egy enter, sortörés karaktert is küld. Nem nyomtatható ASCII karakterek átvitele nem javasolt, mivel kiszámíthatatlan működést eredményezhet.

```
Keyboard.write(key);
```

Karakter kód küldése a számítógépnek. Paramétere a karakter ASCII kódját határozza meg.

```
Keyboard.press(char);
```

Gomb lenyomva tartásának szimulálása. Paramétere a lenyomásra kerülő gombot határozza meg karakter formában. Különösen hasznos módosító gombok beviteléhez, mivel a nem nyomtatható ASCII karakterek átvitelét nem támogatja az osztály. Az alábbi táblázat a lehetséges módosító gombokat tartalmazza.

Konstans	Hexadecimális érték	Decimális érték
KEY_LEFT_CTRL	0x80	128
KEY_LEFT_SHIFT	0x81	129
KEY_LEFT_ALT	0x82	130
KEY_LEFT_GUI	0x83	131
KEY_RIGHT_CTRL	0x84	132
KEY_RIGHT_SHIFT	0x85	133
KEY_RIGHT_ALT	0x86	134
KEY_RIGHT_GUI	0x87	135
KEY_UP_ARROW	0xDA	218
KEY_DOWN_ARROW	0xD9	217
KEY_LEFT_ARROW	0xD8	216
KEY_RIGHT_ARROW	0xD7	215
KEY_BACKSPACE	0xB2	178
KEY_TAB	0xB3	179
KEY_RETURN	0xB0	176
KEY_ESC	0xB1	177
KEY_INSERT	0xD1	209
KEY_DELETE	0xD4	212
KEY_PAGE_UP	0xD3	211
KEY_PAGE_DOWN	0xD6	214
KEY_HOME	0xD2	210
KEY_END	0xD5	213
KEY_CAPS_LOCK	0xC1	193
KEY_F1	0xC2	194
KEY_F2	0xC3	195
KEY_F3	0xC4	196
KEY_F4	0xC5	197
KEY_F5	0xC6	198
KEY_F6	0xC7	199
KEY_F7	0xC8	200
KEY_F8	0xC9	201
KEY_F9	0xCA	202
KEY_F10	0xCB	203
KEY_F11	0xCC	204
KEY_F12	0xCD	205

33. Táblázat: Támogatott módosító gombok konstansai

```
Keyboard.release(key);
```

Lenyomva tartott gomb felengedése. Paramétere a gombot határozza meg karakter formában.

```
Keyboard.releaseAll();
```

Összes lenyomva tartott gomb felengedése.

```
Keyboard.end();
```

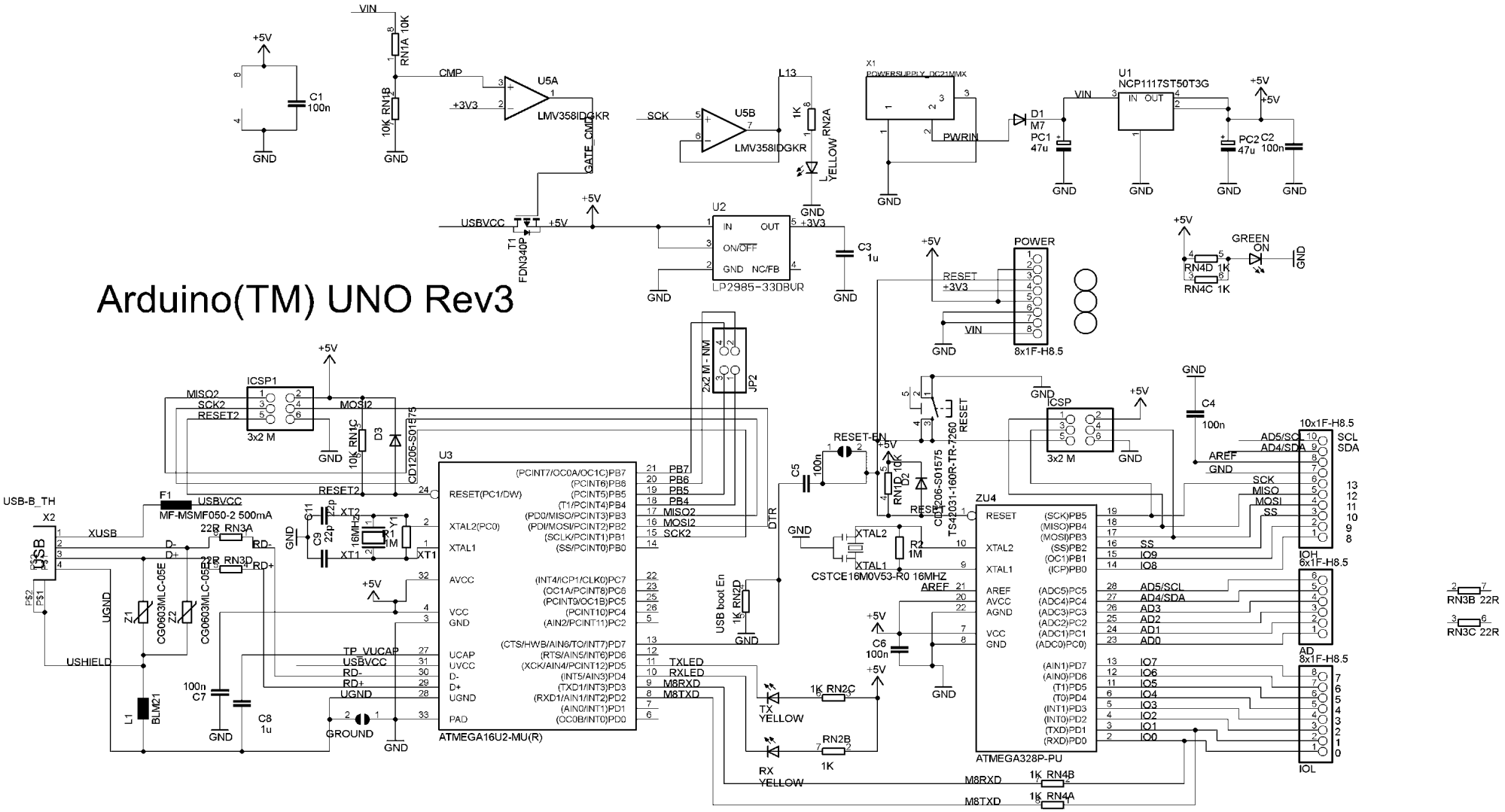
Billentyűzet emuláció kikapcsolása.

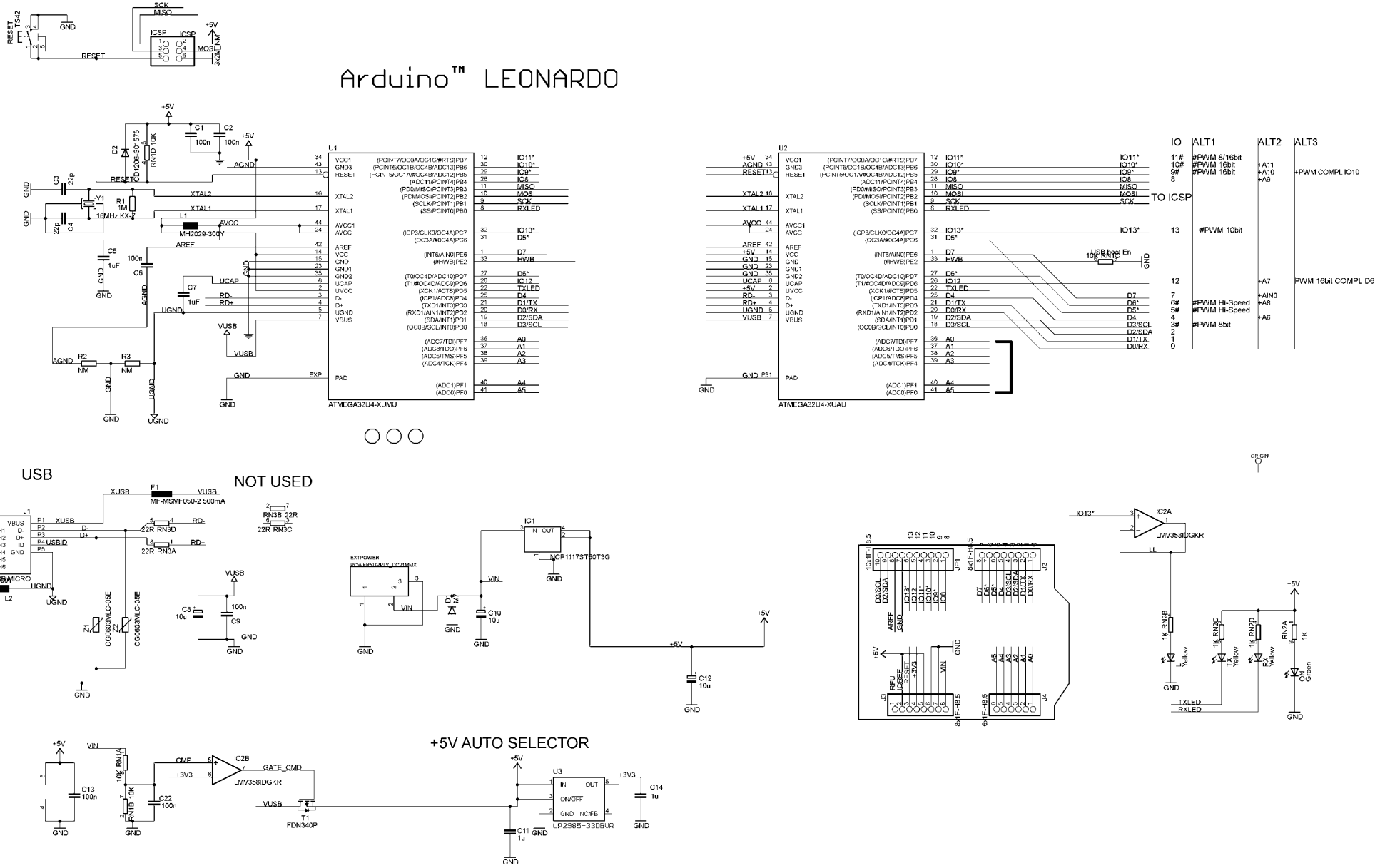
10. Mellékletek

Arduino kapcsolási rajzok

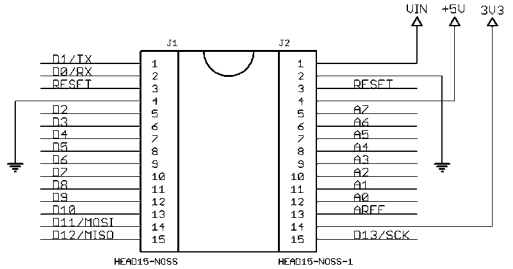
A könyv ezen részében pár Arduino modell referencia terve található meg. A referencia tervek alapján készülnek el a hivatalos Arduino modellek is, azonban a tervek alapján készült Arduino modellek után a készítők semmilyen felelősséget nem vállalnak. A tervek többsége Creative Commons Nevezd meg! - Így add tovább! 2.5 Licenc alatt érhető el, amely magyar fordítása az alábbi címen található meg: http://creativecommons.org/licenses/by-sa/2.5/hu/

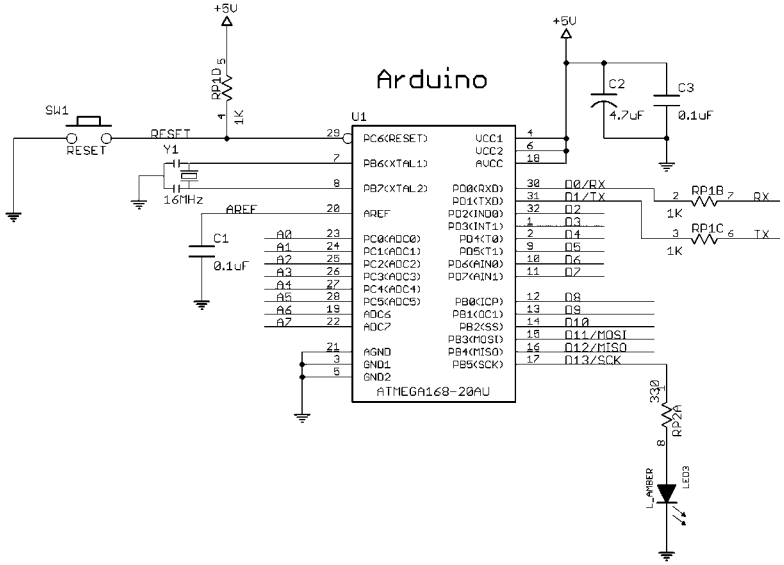
Az egyes modellek tervei időközönként frissülnek. Az egyes változatok naprakész tervei a http://arduino.cc/en/Main/Products oldalon a modell kiválasztása után tekinthetőek meg.

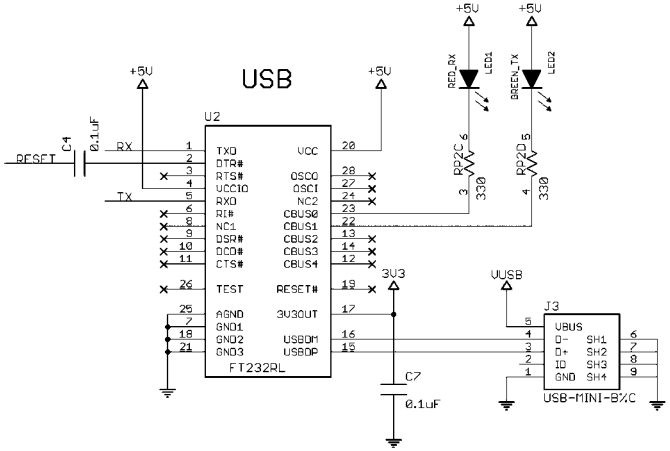

Arduino(TM) UNO Rev3
ATMEGA16U2-MU(R)
ATMEGA328P-PU
LP2985-33DBVR
LMV358IDGKR
NCP1117ST50T3G
TS4231-160R-TR-7260
CSTCE16M0V53-R0 16MHZ
CG0603MLC-05E
CD1206-S01575
MF-MSMF050-2 500mA
FDN340P
8x1F-H8.5
10x1F-H8.5
3x2 M
2x2 M - NM
USB-B_TH
POWERSUPPLY_DC21MMX
VIN
+5V
GND
USBVCC
USHIELD
UGND
RESET
RESET-EN
XTAL1
XTAL2
AREF
SCK
MISO
MOSI
SS
SCL
SDA
TXLED
RXLED
M8RXD
M8TXD
DTR
USB boot En
TX YELLOW
RX YELLOW
GREEN ON
POWER
ICSP
ICSP1
JP2
GROUND

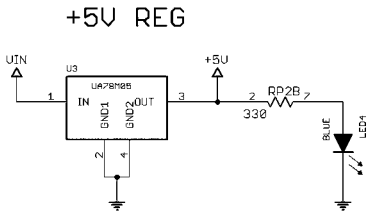

Arduino™ LEONARDO
ATMEGA32U4-XUMU
ATMEGA32U4-XUAU
TO ICSP
USB
NOT USED
+5V AUTO SELECTOR
LMV358IDGKR
NCP1117ST50T3G
LP2985-330BUR
FDN340P
MF-MSMF050-2 500mA

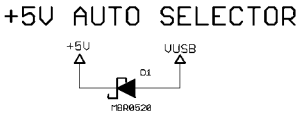
Arduino Nano

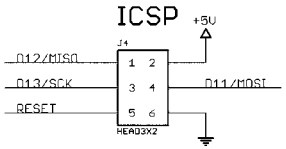


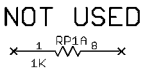


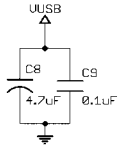


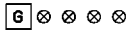






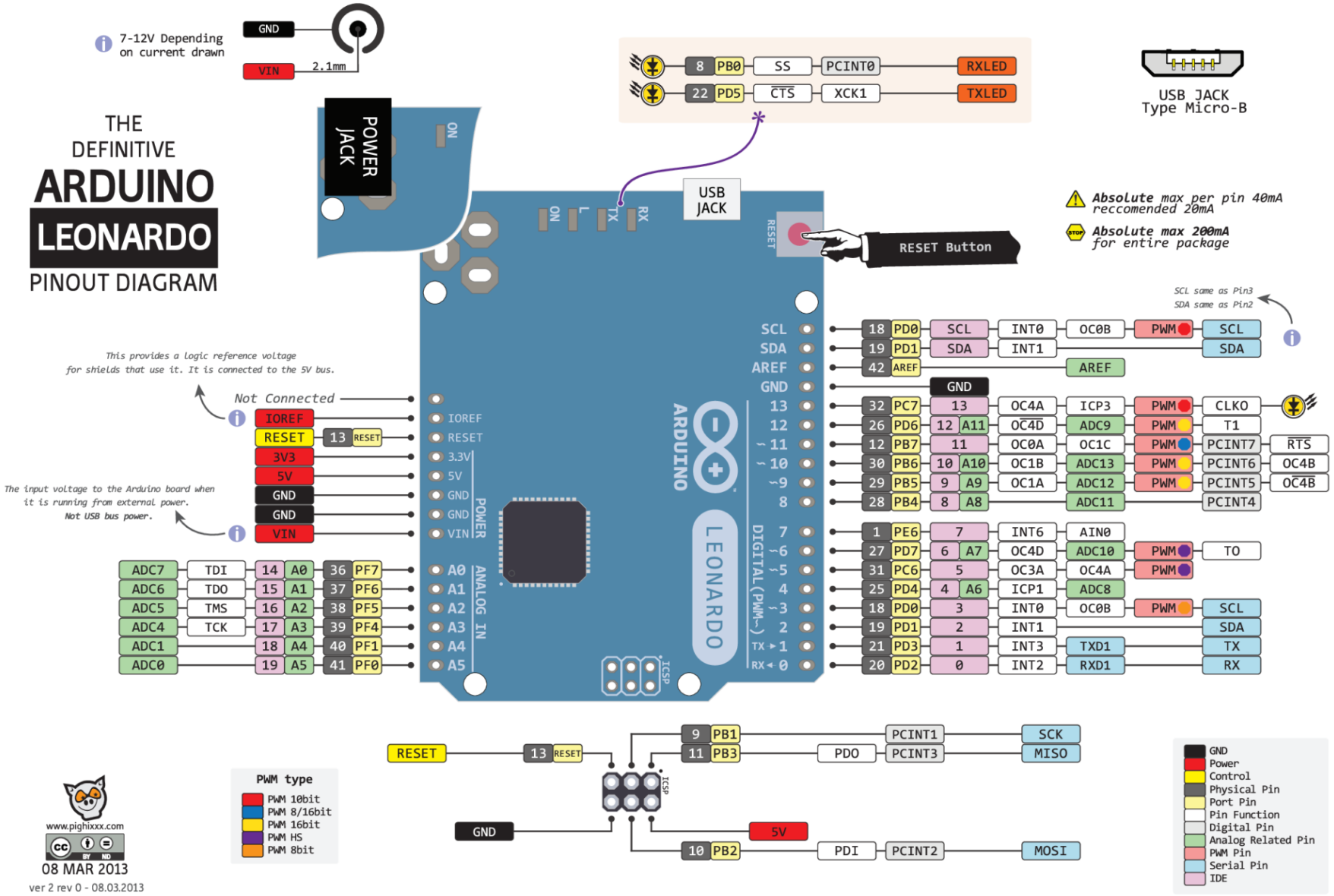


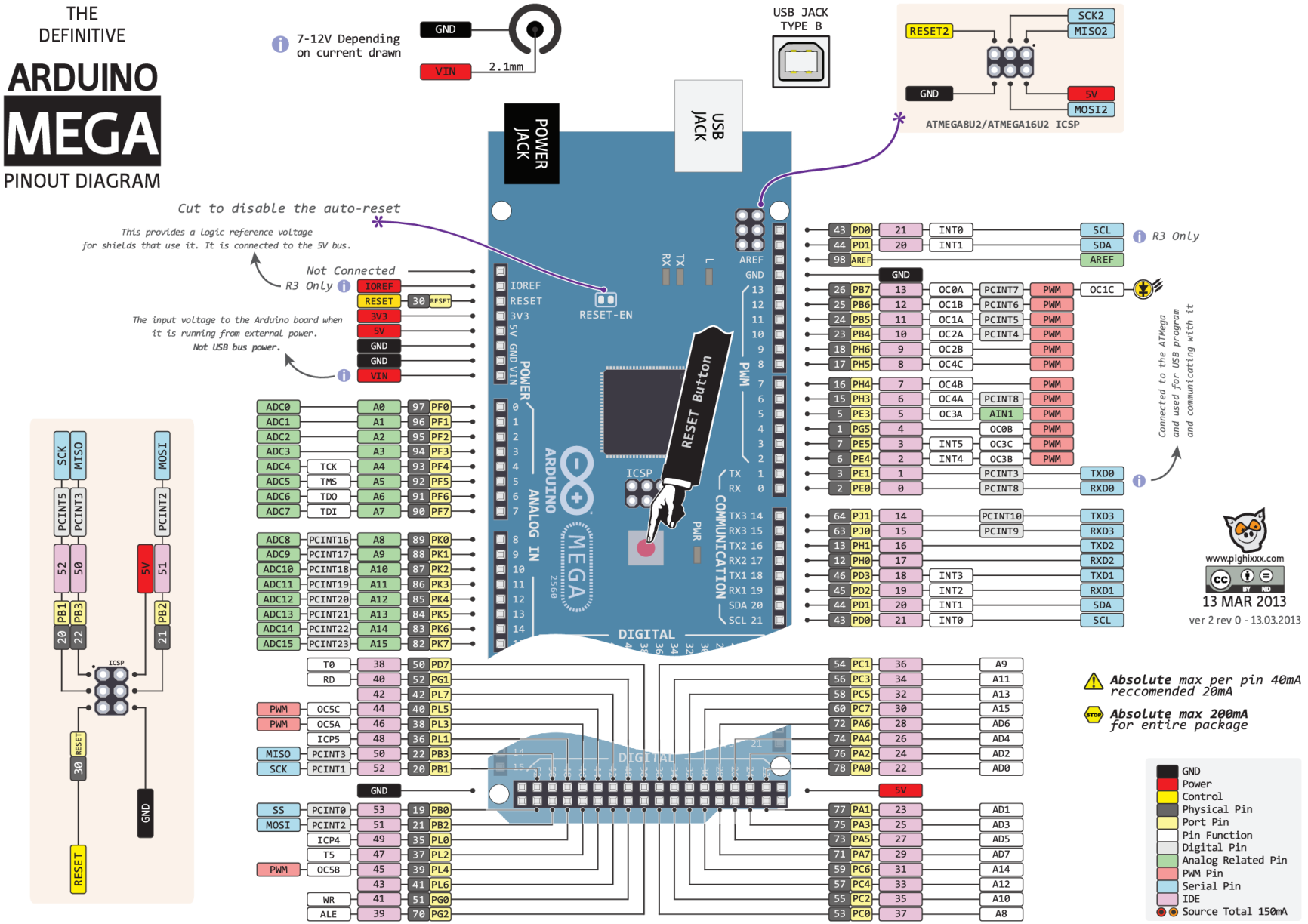


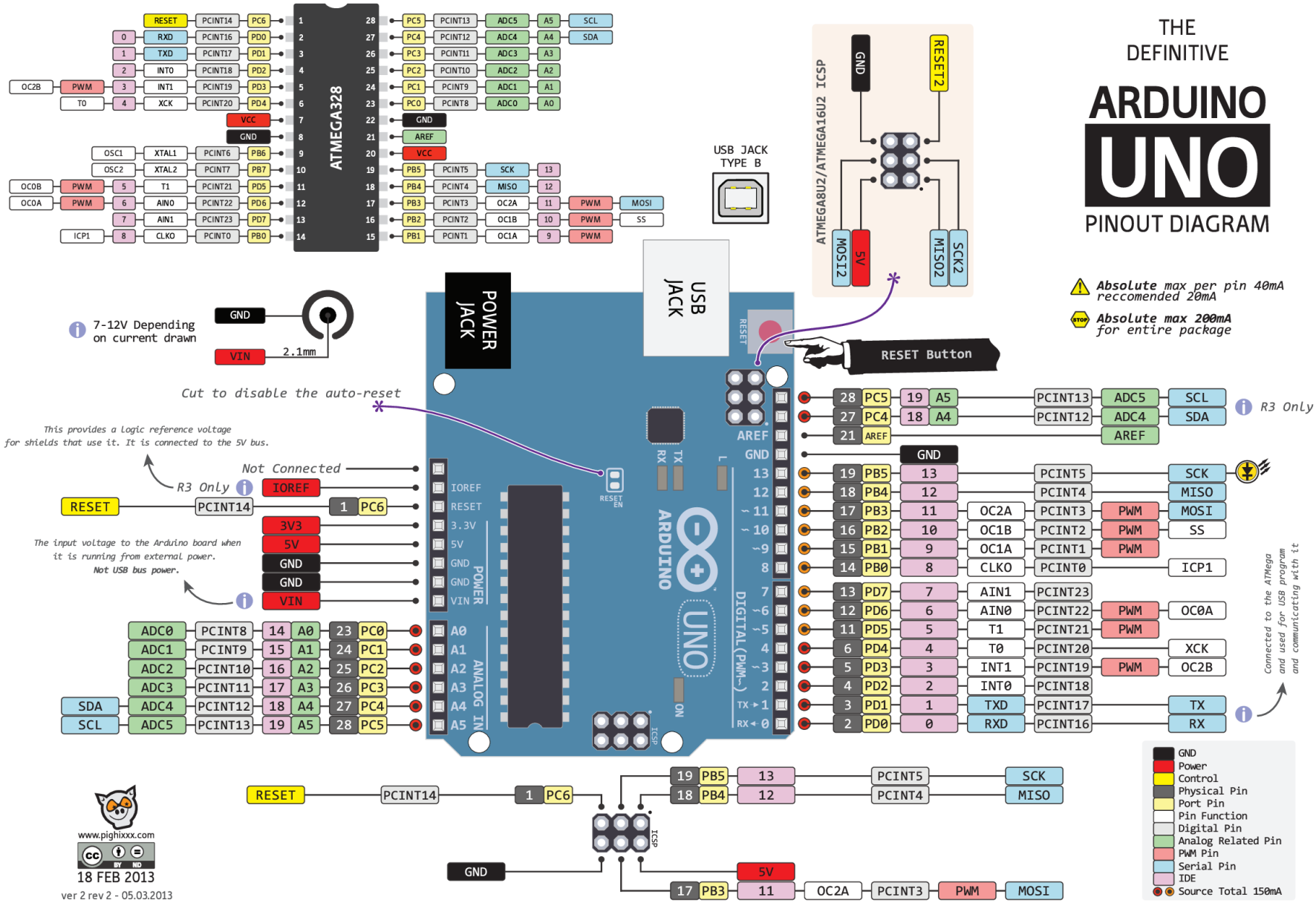


Arduino lábkiosztások

Az alábbi lábkiosztás diagramok a http://www.pighixxx.com/ címről származnak.

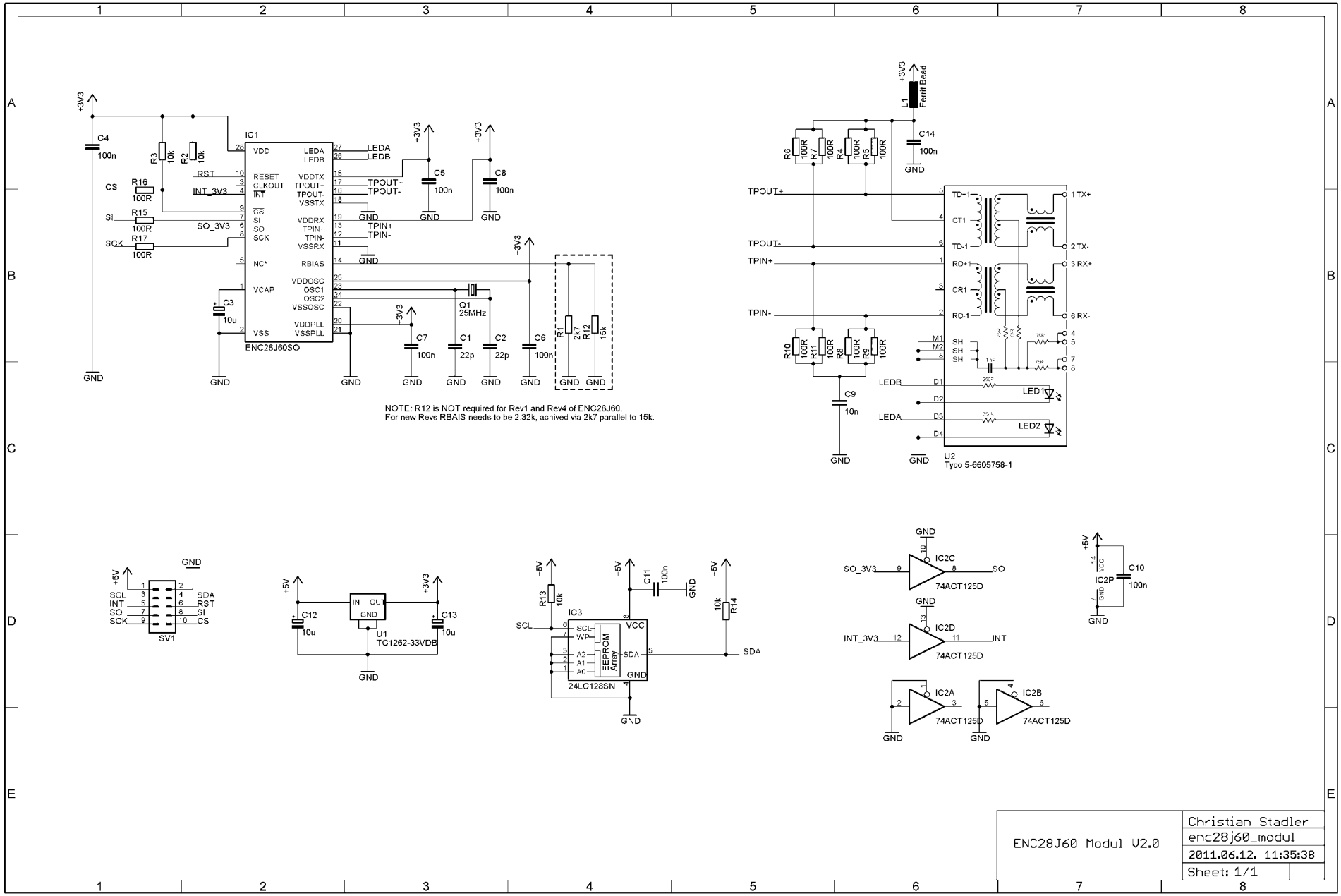

THE DEFINITIVE
ARDUINO
LEONARDO
PINOUT DIAGRAM
7-12V Depending on current drawn
GND
VIN
2.1mm
POWER JACK
USB JACK
RESET Button
USB JACK Type Micro-B
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
SCL same as Pin3
SDA same as Pin2
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
The input voltage to the Arduino board when it is running from external power.
Not USB bus power.
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
ANALOG IN
DIGITAL(PWM~)
ARDUINO
LEONARDO
ICSP
PWM type
PWM 10bit
PWM 8/16bit
PWM 16bit
PWM HS
PWM 8bit
www.pighixxx.com
08 MAR 2013
ver 2 rev 0 - 08.03.2013
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE


THE
DEFINITIVE
ARDUINO
MEGA
PINOUT DIAGRAM
7-12V Depending on current drawn
GND
VIN
2.1mm
USB JACK TYPE B
RESET2
SCK2
MISO2
GND
5V
MOSI2
ATMEGA8U2/ATMEGA16U2 ICSP
POWER JACK
USB JACK
Cut to disable the auto-reset
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
The input voltage to the Arduino board when it is running from external power.
Not USB bus power.
RESET-EN
RESET Button
ICSP
PWR
POWER
ANALOG IN
COMMUNICATION
DIGITAL
PWM
ARDUINO MEGA 2560
R3 Only
Connected to the ATMega and used for USB program and communicating with it
www.pighixxx.com
13 MAR 2013
ver 2 rev 0 - 13.03.2013
Absolute max per pin 40mA
reccomended 20mA
Absolute max 200mA
for entire package
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA


THE DEFINITIVE
ARDUINO UNO
PINOUT DIAGRAM
ATMEGA328
ATMEGA8U2/ATMEGA16U2 ICSP
USB JACK TYPE B
POWER JACK
USB JACK
RESET Button
Cut to disable the auto-reset
7-12V Depending on current drawn
2.1mm
This provides a Logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
The input voltage to the Arduino board when it is running from external power.
Not USB bus power.
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
Connected to the ATMega and used for USB program and communicating with it
www.pighixxx.com
18 FEB 2013
ver 2 rev 2 - 05.03.2013
GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE
Source Total 150mA

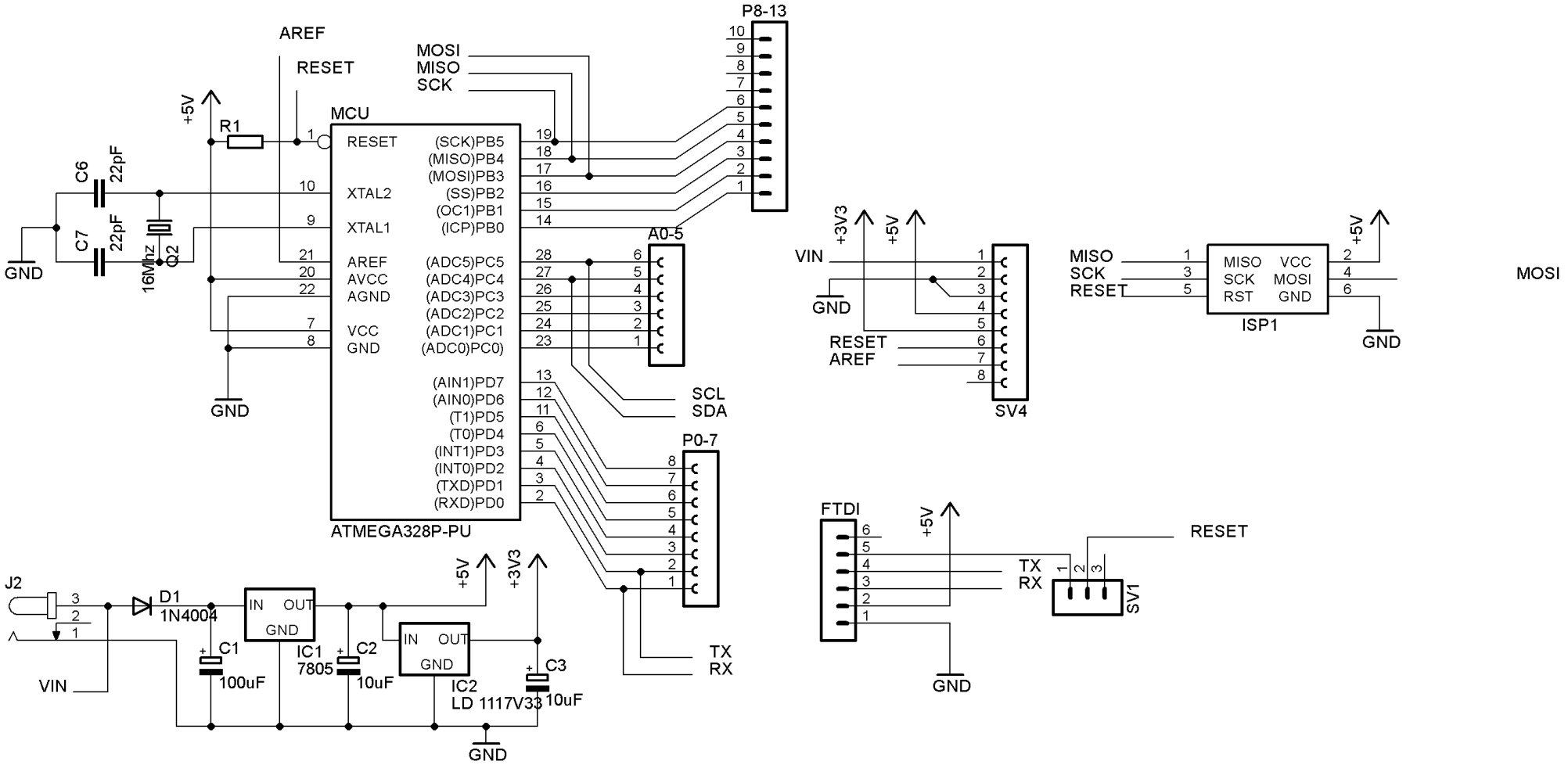
ENC26J60 vezérlőn alapuló Ethernet illesztő

Az Ethernet hálózatkezelés alapjai projektben említett modul kapcsolási rajza, amely a http://www.picprojects.net/enc28j60_modul/ címről származik.


IC1
ENC28J60SO
NOTE: R12 is NOT required for Rev1 and Rev4 of ENC28J60.
For new Revs RBAIS needs to be 2.32k, achived via 2k7 parallel to 15k.
U2
Tyco 5-6605758-1
SV1
U1
TC1262-33VDB
IC3
24LC128SN
74ACT125D
ENC28J60 Modul V2.0
Christian Stadler
enc28j60_modul
2011.06.12. 11:35:38
Sheet: 1/1

Házilag építhető Arduino Uno klón

A klón a legkevesebb szükséges alkatrészt tartalmazza. A 3,3V-os feszültség kimenet opcionálisan elhagyható a tervből, ha nem szükséges.



Bluetooth modul parancskészlete

AT COMMAND LISTING

	COMMAND	FUNCTION
1	AT	Test UART Connection
2	AT+RESET	Reset Device
3	AT+VERSION	Querry firmware version
4	AT+ORGL	Restore settings to Factory Defaults
5	AT+ADDR	Query Device Bluetooth Address
6	AT+NAME	Query/Set Device Name
7	AT+RNAME	Query Remote Bluetooth Device's Name
8	AT+ROLE	Query/Set Device Role
9	AT+CLASS	Query/Set Class of Device CoD
10	AT+IAC	Query/Set Inquire Access Code
11	AT+INQM	Query/Set Inquire Access Mode
12	AT+PSWD	Query/Set Pairing Passkey
13	AT+UART	Query/Set UART parameter
14	AT+CMODE	Query/Set Connection Mode
15	AT+BIND	Query/Set Binding Bluetooth Address
16	AT+POLAR	Query/Set LED Output Polarity
17	AT+PIO	Set/Reset a User I/O pin
18	AT+MPIO	Set/Reset multiple User I/O pin
19	AT+MPIO?	Query User I/O pin
20	AT+IPSCAN	Query/Set Scanning Parameters
21	AT+SNIFF	Query/Set SNIFF Energy Savings Parameters
22	AT+SENM	Query/Set Security & Encryption Modes
23	AT+RMSAD	Delete Authenticated Device from List
24	AT+FSAD	Find Device from Authenticated Device List
25	AT+ADCN	Query Total Number of Device from Authenticated Device List
26	AT+MRAD	Query Most Recently Used Authenticated Device
27	AT+STATE	Query Current Status of the Device
28	AT+INIT	Initialize SPP Profile
29	AT+INQ	Query Nearby Discoverable Devices
30	AT+INQC	Cancel Search for Discoverable Devices
31	AT+PAIR	Device Pairing
32	AT+LINK	Connect to a Remote Device
33	AT+DISC	Disconnect from a Remote Device
34	AT+ENSNIFF	Enter Energy Saving mode
35	AT+EXSNIFF	Exit Energy Saving mode

ERROR CODES

ERROR CODE	VERBOSE
0	Command Error/Invalid Command
1	Results in default value
2	PSKEY write error
3	Device name is too long (>32 characters)
4	No device name specified (0 lenght)
5	Bluetooth address NAP is too long
6	Bluetooth address UAP is too long
7	Bluetooth address LAP is too long
8	PIO map not specified (0 lenght)
9	Invalid PIO port Number entered
A	Device Class not specified (0 lenght)
B	Device Class too long
C	Inquire Access Code not Specified (0 lenght)
D	Inquire Access Code too long
E	Invalid Iquire Access Code entered
F	Pairing Password not specified (0 lenght)
10	Pairing Password too long (> 16 characters)
11	Invalid Role entered
12	Invalid Baud Rate entered
13	Invalid Stop Bit entered
14	Invalid Parity Bit entered
15	No device in the Pairing List
16	SPP not initialized
17	SPP already initialized
18	Invalid Inquiry Mode
19	Inquiry Timeout occured
1A	Invalid/zero lenght address entered
1B	Invalid Security Mode entered
1C	Invalid Encryption Mode entered

1. Test UART connection

COMMAND	RESPONSE
AT	OK

2. Reset Device

COMMAND	RESPONSE
AT+RESET	OK

3. Querry firmware version

COMMAND	RESPONSE
AT+VERSION?	+VERSION:<VER> OK

where <VER> = Version Number

4. Restore settings to Factory Defaults

COMMAND	RESPONSE
AT+ORGL	OK

Restore to the following settings:
Device Class: 0
Inquiry Code: 0x009e8b33
Device Mode: Slave
Binding Mode: SPP
UART: 38400bps, 8 bit, 1 stop bit, no parity
Pairing Code: 1234
Device Name: H-C-2010-06-01

5. Query EGBT-045MS Bluetooth Address

COMMAND	RESPONSE
AT+ADDR?	+ADDR:nn:uu:ll OK

Bluetooth Address Format:
nn - NAP (16 bit Non-significant Address Portion)
uu - UAP (8 bit Upper Address Portion)
ll - LAP (24 bit Lower Address Portion)
Returned bluetooth address

Example: Querry EGBT-045MS Bluetooth Address

From Host controller:
AT+ADDR?
EGBT-045MS Response
+ADDR:11:6:230154
OK

Bluetooth Address is 11:06:23:01:54

6. Query/Set Device Name

COMMAND	RESPONSE
AT+NAME?	+NAME:<name> OK
AT+NAME=<name>	OK

where <name> = Device Name (31 characters max.)

Example: Query device name

From Host controller:
AT+NAME?
EGBT-045MS Response
+NAME:HC-05
OK

Example: Set device name to “e-Gizmo”

From Host controller:
AT+NAME=e-Gizmo
EGBT-045MS Response
OK

Example: Set device name to “supercalifragilisticexpialidocious”

From Host controller:
AT+NAME=supercalifragilisticexpialidocious
EGBT-045MS Response
ERROR:(3) *name too long (>31 characters)*

7. Query Remote Bluetooth Device’s Name

COMMAND	RESPONSE
AT+RNAME?<addr>	+NAME:<name> OK

where <name> = Device name
<addr> = 48 bit bluetooth address
in NAP,UAP,LAP format

Example: Query remote Bluetooth device having address = 00:02:72:0A:3C:7F

Bluetooth address in NA:UAP:LAP format = 0002:72:0A3C7F

From Host controller:
AT+RNAME?0002,72,0A3C7F
EGBT-045MS response if remote device name is “HC-05”
+NAME:HC-05
OK
EGBT-045MS response if remote device name is unresolved
FAIL

8. Query/Set Device Role

COMMAND	RESPONSE
AT+ROLE?	+ROLE:<role> OK
AT+ROLE=<role>	OK

where <role>
0 - Slave (default)
1 - Master
2 - Slave-Loop

Slave - EGBT-045MS acts as discoverable wireless UART device ready for transparent data exchange.
Master - Scans for a remote bluetooth (slave) device, pairs, and setup connection for a transparent data exchange detween devices
Slave-Loop - Data loop-back Rx-Tx. Used mainly for testing.

Example: Set EGBT-045MS in master role

Bluetooth address in NA:UAP:LAP format = 0002:72:0A3C7F

From Host controller:
AT+ROLE=1
EGBT-045MS response
+ROLE:1
OK

9. Query/Set Class of Device CoD

COMMAND	RESPONSE
AT+CLASS?	+CLASS:<class> OK
AT+CLASS=<class>	OK

where <class>
0 - (default)

The Class of Device identifier. For more info, see the Bluetooth_Code_Definition.pdf file included with the product documentation of this kit.

10. Query/Set Inquire Access Code

COMMAND	RESPONSE
AT+IAC?	+IAC:<iac> OK
AT+IAC=<iac>	OK / FAIL

where <iac> = Inquire Access Code
9e8b33 - default value

11. Query/Set Inquire Access Mode

COMMAND	RESPONSE
AT+INQM?	+INQM:<inq1>,<inq2>,<inq3> OK
AT+INQM=<inq1>,<inq2>,<inq3>	OK / FAIL

where <inq1> Inquire Access Mode
0 - standard
1 - rssi (default)

<inq2> Maximum number of devices response
0 to 32000
1 (default)

<inq3> Inquire timeout
1 to 48
48 (default)

Maximum number of devices response - EGBT-045MS will stop inquiring once the number of devices that responded reaches this value.
Inquire Timeout - Multiply this number by 1.28 to get the maximum time in seconds the EGBT-045MS will wait for a respond to an inquiry call.

Example: Set EGBT-045MS Inquire Access Mode at
Inquire access mode - 1 (rssi)
Number of devices - 3
Timeout - 10 (10*1.28 = 12.8 seconds)

Bluetooth address in NA:UAP:LAP format = 0002:72:0A3C7F

From Host controller:
AT+INQM=1,3,10
EGBT-045MS response
OK

12. Query/Set Pairing Passkey

COMMAND	RESPONSE
AT+PSWD?	+PSWD:<password> OK
AT+PWSD=<password>	OK

where <password> = Alphanumeric password 16 characters max.
1234 - (default)

Example: Set EGBT-045MS Password to “e-Gizmo”

From Host controller:
AT+PSWD=e-Gizmo
EGBT-045MS response
OK

13. Query/Set UART parameter

COMMAND	RESPONSE
AT+UART?	+UART:<baud>,<stop>,<parity> OK
AT+UART=<baud>,<stop>,<parity>	OK

where <baud> =baud rate, any one of the following
4800
9600 (default)
19200
38400
57600
115200
234000
460800
921600
1382400

<stop> = number of stop bits
0 - 1 bit (default)
1 - 2 bits

<parity> = Parity bit
0 - None (default)
1 - Odd parity
2 - Even Parity

Example: Set EGBT-045MS UART parameter to 115200bps, 2 stop bits, even parity

From Host controller:
AT+UART=115200,1,2
EGBT-045MS response
OK

14. Query/Set Connection Mode

COMMAND	RESPONSE
AT+CMODE?	+CMODE:<mode> OK
AT+CMODE=<mode>	OK

where <mode>
0 - Connect to a specified Bluetooth device only (default). See related command Command 15.
1 - Can connect with any other Bluetooth device.
2 - Test mode

15. Query/Set Binding Bluetooth Address

COMMAND	RESPONSE
AT+BIND?	+BIND:<addr> OK
AT+BIND=<addr>	OK

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

Example: Bind with Bluetooth device having address = 00:02:72:0A:3C:7F

Bluetooth address in NA,UAP,LAP format = 0002,72,0A3C7F

From Host controller:
AT+BIND=0002,72,0A3C7F
EGBT-045MS response
OK

16. Query/Set LED Output Polarity

COMMAND	RESPONSE
AT+POLAR?	+POLAR:<led1>,<led2> OK
AT+POLAR=<led1>,<led2>	OK

where <led1> = LED1 (pin 31) Polarity
0 - LED1 output active low
1 - LED1 output active high (default)

<led2> = LED2 (pin 32) Polarity
0 - LED2 output active low
1 - LED2 output active high (default)

LED 1
Flashes once each seconds to indicate EGBT-045MS is in Command Mode. Flashes two times per second when EGBT-045MS is in data mode.

LED2
Turns ON when EGBT-045MS remote connection is successfuly opened.

17. Set/Reset a User I/O pin

COMMAND	RESPONSE
AT+PIO=<pn>,<value>	OK

where <pn> = port number. Available port are as follows
2 - PIO2
3 - PIO3
4 - PIO4
5 - PIO5
6 - PIO6
7 - PIO7
10 - PIO10

<value>
0 - Logic Low
1 - Logic High

Example: Set PIO2 to logic High

From Host controller:
AT+PIO=2,1
EGBT-045MS response
OK

18. Set/Reset multiple User I/O pin

COMMAND	RESPONSE
AT+MPIO=<iomap>	OK

where
<iomap> =12-bit I/O map presented in hexadecimal

x	PIO10	X	X	PIO7	PIO6	PIO5	PIO4	PIO3	PIO2	x	x

x - don't care/reserved

Example:
Set PIO2 and PIO6 to logic High, all others to logic 0

Bit pattern is 0000 0100 0100 = 44 hexadecimal

From Host controller:
AT+MPIO=44
EGBT-045MS response
OK

19. Query User I/O pin

COMMAND	RESPONSE
AT+MPIO?	+MPIO:<iomap> OK

where
<iomap> =12-bit I/O map presented in hexadecimal

x	PIO10	X	X	PIO7	PIO6	PIO5	PIO4	PIO3	PIO2	x	x

x - reserved -used by system, may assume any values

Example:
Read PIO inputs

From Host controller:
AT+MPIO?
EGBT-045MS response
+MPIO:944
OK
Returned value in binary: 1001 0100 0100

In this example, the PIO are previously set in command 18 with PIO2 and PIO6 set. The returned value also shows reserved bits 11 and 8 set by the system.

20. Query/Set Scanning Parameters

COMMAND	RESPONSE
AT+IPSCAN?	+IPSCAN:<int>,<dur>,<pint>,<pdur> OK
AT+IPSCAN=<int>,<dur>,<pint>,<pdur>	OK

where <int> = inquire scan time interval
1024 - default
<dur> = inquire scan time duration
512 - default
<pint> = page scan time interval
1024 - default
<pdur> = page scan time duration
512 - default

All parameters must be represented with decimal integer value.

21. Query/Set SNIFF Energy Savings Parameters

COMMAND	RESPONSE
AT+SNIFF?	+SNIFF:<tmax>,<tmin>,<retry>,<timeout> OK
AT+SNIFF=<tmax>,<tmin>,<retry>,<timeout>	OK

where <tmax> = maximum time
0 - default
<tmin> = minimum time
0 - default
<retry> = retry time
0 - default
<timeout> = timeout
0 - default

All parameters must be represented with decimal integer value.

22. Query/Set Security & Encryption Modes

COMMAND	RESPONSE
AT+SENM?	+SENM:<mode>,<encrypt> OK
AT+SENM=<mode>,<encrypt>	OK

where <mode> = Security Mode
0 - sec_mode_off (default)
1 - sec_mode1_non-secure
2 - sec_mode2-service
3 - sec_mode3_link
4 - sec_mode_unknown

<encrypt> = encryption mode
0 - hci_enc_mode_off (default)
1 - hci_enc_mode_pt_to_pt
2 - hci_enc_mode_pt_to_pt_and_bcast

23. Delete Authenticated Device from List

COMMAND	RESPONSE
AT+RMSAD=<addr>	OK

where <addr> = 48 bit bluetooth address
in NAP,UAP,LAP format

Example: Remove from Authenticated Device list a Bluetooth device having address = 00:02:72:0A:3C:7F

Bluetooth address in NA,UAP,LAP format = 0002,72,0A3C7F

From Host controller:
AT+RMASD=0002,72,0A3C7F
EGBT-045MS response if deletion is successful
OK
EGBT-045MS response if remote device address is not in the list
ERROR(15)

Caution:

Entering
AT+RMASD
will delete ALL authenticated device from the list!

24. Find Device from Authenticated Device List

COMMAND	RESPONSE
AT+FSAD=<addr>	OK

where <addr> = 48 bit bluetooth address
in NAP,UAP,LAP format

Note: AT+FSAD returns a FAIL response if device is not in the authenticated list

25. Query Total Number of Device from Authenticated Device List

COMMAND	RESPONSE
AT+ADCN?	+ADCN:<total> OK

where
<total> = total number of devices in the authenticated device list

26. Query Most Recently Used Authenticated Device

COMMAND	RESPONSE
AT+MRAD?	+MRAD:<addr> OK

where <addr> = 48 bit bluetooth address
in NAP:UAP:LAP format

27. Query Current Status of the Device

COMMAND	RESPONSE
AT+STATE?	+STATE:<stat> OK

where
<stat> = Current Status, any one of the following:

INITIALIZED
READY
PAIRABLE
PAIRED
INQUIRING
CONNECTING
CONNECTED
DISCONNECTED
UNKNOWN

28. Initialize SPP Profile

COMMAND	RESPONSE
AT+INIT	OK / FAIL

29. Query Nearby Discoverable Devices

COMMAND	RESPONSE
AT+INQ	+INQ: <addr>,<class>,>rssi> OK

where <addr> = 48 bit bluetooth address
in NAP:UAP:LAP format
<class> = Device Class
<rssi> = RSSI

This command will scan and report all nearby dicoverable Bluetooth devices. The same device may be reported more than once.

This command will work only if EGBT-045MS is set to work as a master device, i.e. AT+ROLE=1, and after AT+INIT is executed.

Example: Discover nearby devices

From Host controller: Set device role as master

AT+ROLE=1

EGBT-045MS response

OK

From Host controller: Initialize SPP

AT+INIT

EGBT-045MS response

OK

From Host controller: Set inquire mode as RSSI, look for 9 devices, and 48 as timeout

AT+INQM=1,9,48

EGBT-045MS response

OK

From Host controller: Start Device Discovery

AT+INQ

EGBT-045MS response (sample only, actual report will vary)

+INQ:101D:C0:2E7B54,5A0204,7FFF

+INQ:25:48:21AD1A,5A020C,7FFF

OK

In this example, EGBT-045MS found only two discoverable devices. It will quit searching after the timeout time specified by the AT+INQM command or if the number of discovered devices equals the number of specified devices.

30. Cancel Search for Discoverable Devices

COMMAND	RESPONSE
AT+INQC	OK

AT+INQ can be stopped at anytime by executing this command.

31. Device Pairing

COMMAND	RESPONSE
AT+PAIR=<addr>,<timeout>	OK / FAIL

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

<timeout> = Timeout time in sec

32. Connect to a Remote Device

COMMAND	RESPONSE
AT+LINK=<addr>	OK / FAIL

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

33. Disconnect from a Remote Device

COMMAND	RESPONSE
AT+DISC	+DISC:<results>

where

<results> = Disconnection results, any one of the following:

SUCCESS
LINK_LOSS
NO_SLC
TIMEOUT
ERROR

34. Enter Energy Saving mode

COMMAND	RESPONSE
AT+ENSNIFF=<addr>	OK

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

35. Exit Energy Saving mode

COMMAND	RESPONSE
AT+EXSNIFF=<addr>	OK

where <addr> = 48 bit bluetooth address in NAP,UAP,LAP format

Felhasznált irodalom

- ***Arduino weblap***
  http://arduino.cc/en/
- ***Steinhart–Hart equation***
  http://en.wikipedia.org/wiki/Steinhart%E2%80%93Hart_equation
- ***Operational amplifier applications***
  http://en.wikipedia.org/wiki/Operational_amplifier_applications
- ***OSI modell***
  http://hu.wikipedia.org/wiki/TCPIP
- ***TCPIP***
  http://hu.wikipedia.org/wiki/OSI_modell
- ***MCP23017 adatlap***
- ***EGBT-045MS adatlap***
- ***Microchip 24LC256 adatlap***
- ***Simon Monk – Programing Arduino, Getting Started with Sketches***
  ISBN: 978-0-07-178422-1
- ***Massimo Banzi – Getting Started with Arduino***
  ISBN: 978-1-449-30987-9

Ábrajegyzék

Táblázatjegyzék