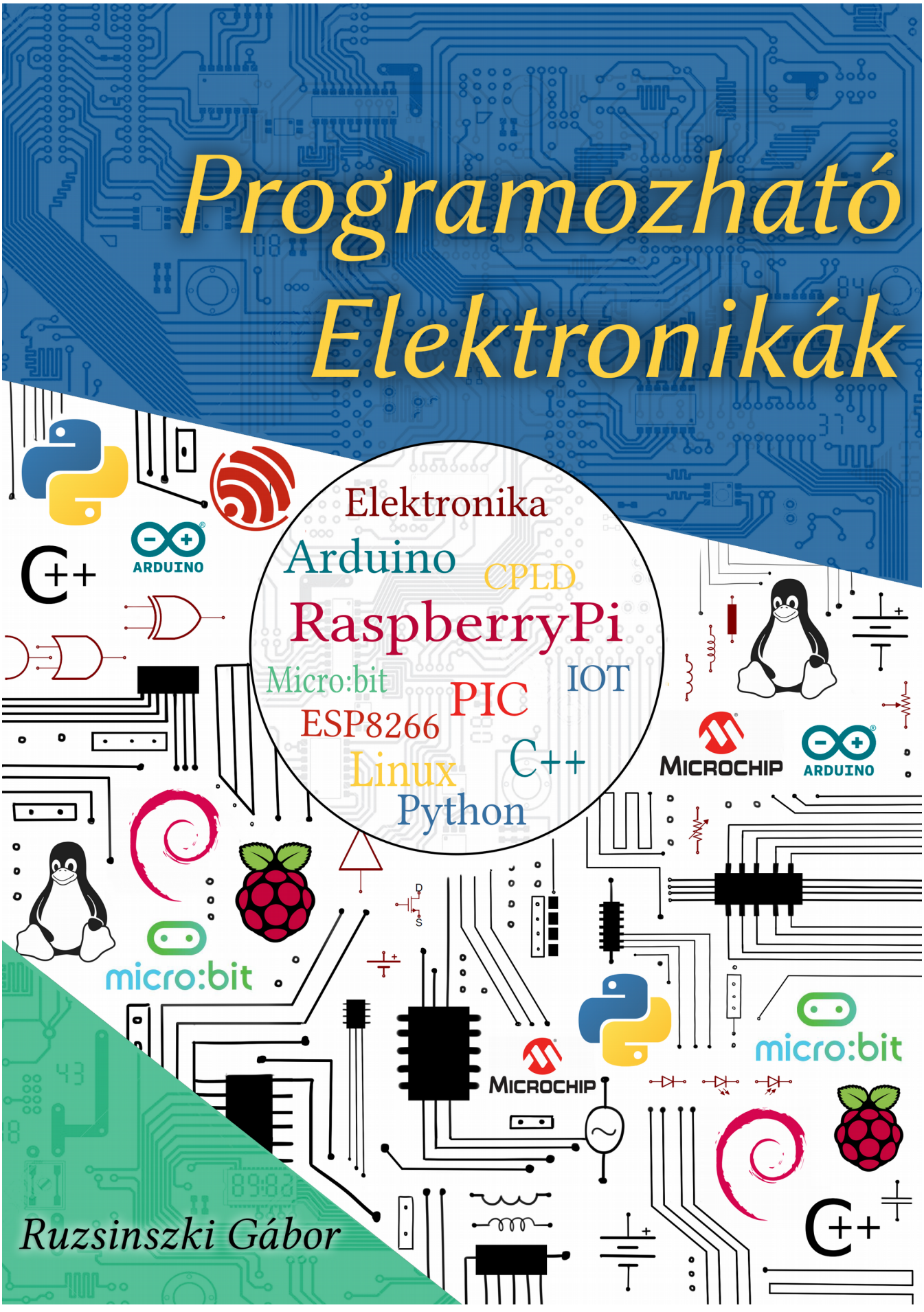
Programozható Elektronikák



Ruzsinszki Gábor

Szándékosan üresen hagyott oldal

Szándékosan üresen hagyott oldal

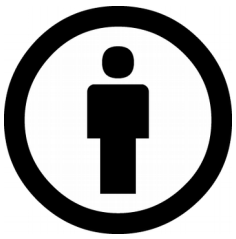
Tartalom

Licenc

Nevezd meg! – Ne add el! – Így add tovább! 4.0 Nemzetközi

Ruszinszki Gábor Programozható Elektronikák című műve Creative Commons Nevezd meg! - Ne add el! - Így add tovább! 4.0 Nemzetközi Licenc alatt van. A licenc teljes szövege a http://creativecommons.org/licenses/by-nc-sa/4.0/legalcode címen érhető el.

Emberek által is érthető formában a licencszerződés fontosabb feltételei:

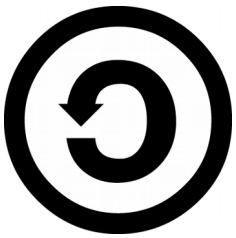


Nevezd meg!

A mű felhasználása, átdolgozása esetén meg kell jelölnöd az eredeti szerzőt és a művet.

Ne add el!

Kereskedelmi haszonszerzés céljából nem használhatod fel a művet. Ez alatt értendő a nyomtatott változatok kereskedelmi értékesítése.



Így add tovább!

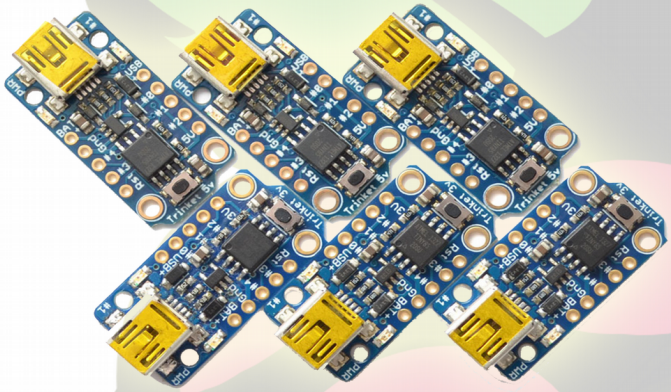
A származékos műveket az eredetivel megegyező féltelek mellett adhatod tovább.

Borítóképre vonatkozó információ:

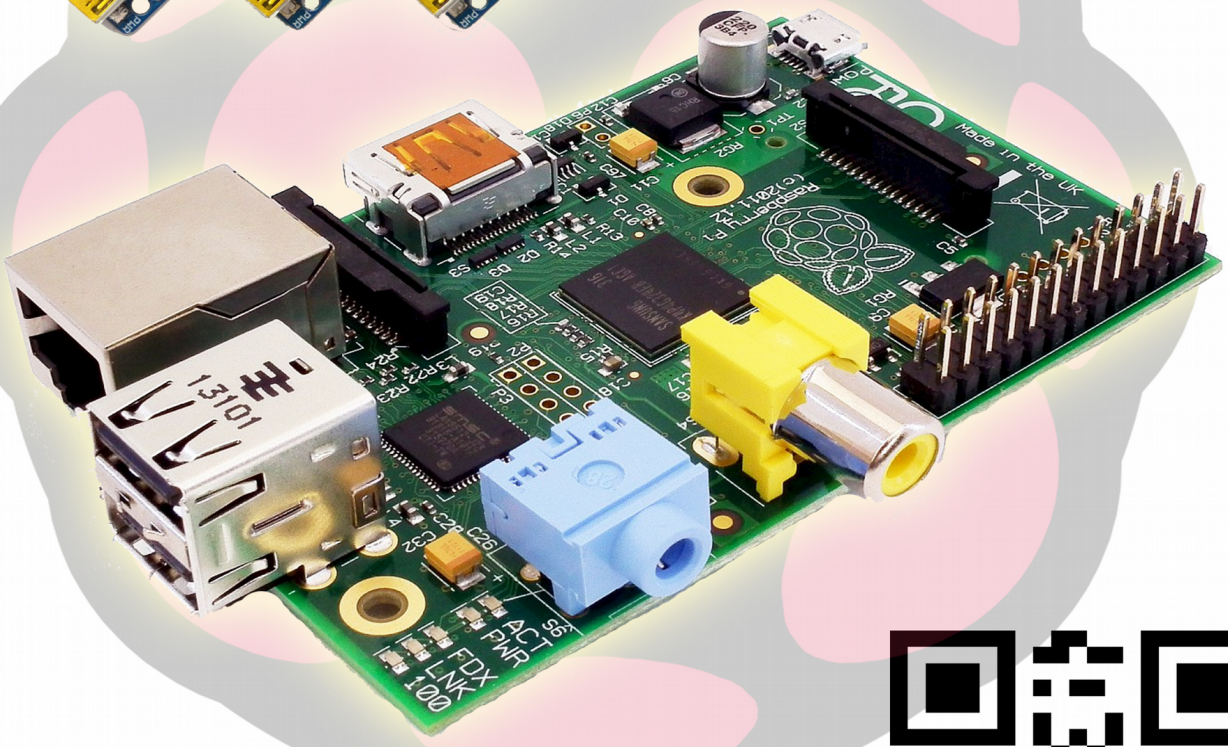
A borítókép a mű többi részével ellentétben nem Creative Commons licenc alatt áll, azaz szabadon a szerzők írásos engedélye nélkül nem használható fel származékos művek készítésre. A képet tervezte és készítette: Pászti-Tóth Enikő aka. EeNii - http://eenii.deviantart.com/

RaspberryPi és Adafruit termékek

a **Magyarországi** hivatalos forgalmazótól,
raktárról, 12 hónapos garanciával



http://malnapc.hu/



adafruit
INDUSTRIES

1. Bevezetés

Manapság minden érdekes kapcsolás tartalmaz egy programozott elektronikai alkatrészt.. Ez az elem lehet egy mikrovezérlő, vagy egy CPLD, vagy ha bonyolultabb kapcsolásról beszélünk, akkor SOC vagy FPGA.

Ezen eszközök az elmúlt években igen sokat fejlődtek, valamint az áruk arányosan csökkent, így manapság már kellően olcsóak ahhoz, hogy hobbi célra is alkalmazhatóak legyenek. A használatukra vonatkozó ismeretekről számos angol nyelvű szakirodalom érhető el, azonban ezek hátránya, hogy az ismeretek tanítását nem igazán az alapoktól kezdik, vagy csak a programozással foglalkoznak, kiegészítő ismeretekkel nem. Így az alapok megtanulása eszközfüggetlenül bonyolult folyamat is lehet.

A könyv létrehozásakor a fő motivációm egy átfogó magyar nyelvű szakirodalom megalkotása volt, amely az alapoktól kezdi az oktatást. Azt, hogy ez mennyire sikerült, csak az olvasó döntheti el.

A könyv igencsak nagy méretű, ennek ellenére nem ad teljes képet a tárgyalt témákról, csupán a kellő alapismereteket adja meg, ami alapján a kedves olvasó egy-egy altémába igazán bele tudja magát ásni, ha szeretné.

A mű alkotása közben felhasználtam két korábbi könyvemet is, amelyek a PIC mikrovezérlőkről és az Arduino platformról szóltak. Azonban ez nem jelenti azt, hogy a könyv a két könyv összeollózásából született meg, mivel egyes részek igencsak ki lettek bővítve és át lettek dolgozva, az olvasói visszajelzések alapján.

Az SOC eszközök közül a könyv részletesen a RaspberryPi-vel foglakozik, annak ellenére, hogy léteznek nála jobb megoldások vezérlési feladatokra. A választás mégis azért esett erre a lapra, mivel olcsón és könnyen beszerezhető, valamint igencsak ismert, ebből adódóan részletesen dokumentált és igencsak karbantartott szoftverekkel rendelkezik. Ez sajnos a vetélytársairól nem igen mondható el. A szükséges Linux alapismeretek vezérlőtől függetlenül alkalmazhatóak.

PIC mikrovezérlőkből a 16F887-es mikrovezérlővel foglalkozik a könyv, mivel a munkahelyemen ezzel oktatunk. A 16F széria helyett azonban manapság már a 18F széria alkalmazása ajánlott, komolyabb célokra pedig egy PIC24 vagy egy PIC32 alkalmazása lenne célszerű. A könyvben ismertetett MikroC program ezen vezérlőket is támogatja, így minimális módosítással a könyvben szereplő kódok PIC mikrovezérlőtől függetlenül alkalmazhatóak.

A könyv a PIC mikrovezérlőknél részletesebben foglalkozik az Atmel mikrovezérlőket alkalmazó Arduino platformmal. Ennek az oka, hogy az Arduino környezet az elmúlt pár évben akkora népszerűségre tett szert, hogy szó nélkül elmenni a jelenség mellett főben járó bűn lenne.

A népszerűség oka az egyszerűség, a részletes dokumentáltság és a bődületes mennyiségű kész programkód. Részletesen az Uno modell kerül tárgyalásra. Ennek oka az, hogy ez volt az első modell és ebből adódóan az Arduino szó sok esetben az Arduino Uno-t jelenti. De mivel a lapok egymással kompatibilisek, így a kódrészletek modelltől függetlenül hordozhatóak, amennyiben csak a beépített könyvtárakra támaszkodunk. Külső könyvtárak esetén a hordozhatóság nem feltétlen biztosított, az ilyen eseteket igyekeztem jelezni a könyvben.

Használt jelölések

A könyvben számos parancssori példa, programkód található, ezek eltérő formázással szerepelnek a könyvben.

```
C/C++ programkód
```

```
Unix shell parancs vagy parancsfájl
```

```
Unix Shell Parancs kimenete vagy konfigurációs fájl tartalma
```

```
Python forráskód
```

```
Verilog forráskód
```

A könyvben a byte mértékegység prefixumai kettő hatványai alapján vannak jelölve. Tehát 1 Kilo Byte 1024 bytnak felel meg, 1 MB pedig 1 048 576 Byte-nak.

NCore kiadás

A könyv ezen változata szándékosan az Ncore torrent tracker látogatóinak készült. Időről-időre felbukkant itt a

könyvem valamelyik változata, amiért más kapott pontokat és nem utolsó sorban az engedélyem nélkül kerültek ezen változatok fel.

Ezért úgy döntöttem itt az ideje, hogy saját magam tegyem közzé a könyvet, ha már igény mutatkozott rá.

Az itt közzétett változat szándékosan nem a legfrissebb változat a könyvből, mert ugye annak elkészítésével munka volt. Viszont ebből, a kb 2015-ből származó verzióban is sok hasznos információt találsz.

A legfrissebb változatot a https://www.webmaster442.hu/letoltheto-irasok/programozhato-elektronikak/ címről tudjátok beszerezni, amihez egy 40%-os kedvezményre jogosító kupont is mellékelnék: „ncore”

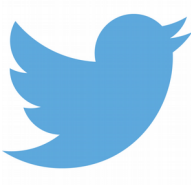
A fenti kuponkódot a vásárlási folyamat közben az offer code mezőbe kell beírni, illetve ha a C# programozási nyelv iránt érdeklődnél, akkor ajánlom figyelmedbe a https://csharptutorial.hu/ oldalamat is.

Frissítések, hírek kapcsolat.

A könyveim időről időre frissülnek, ahogy a technika fejlődik. Ebből adódóan preferálom az E-book formátumokat, mivel így papírfelhasználás nélkül, viszonylag kevés problémával lehet eljuttatni az olvasókhoz a kész tartalmat. A frissítésekről a weblapomon keresztül tájékozódhat az olvasó. Ennek a címe: http://webmaster442.hu. Továbbá ezen a címen található a könyvhöz tartozó DVD melléklet is, amelyen a használt programok telepítője és a forráskódok többsége található meg.

Kisebb frissítések a Twitter fiókomon keresztül érhetőek el vagy az oldalamhoz tartozó Facebook oldalon. Ezek címei:

https://www.facebook.com/pages/Webmaster442hu/197381450300096

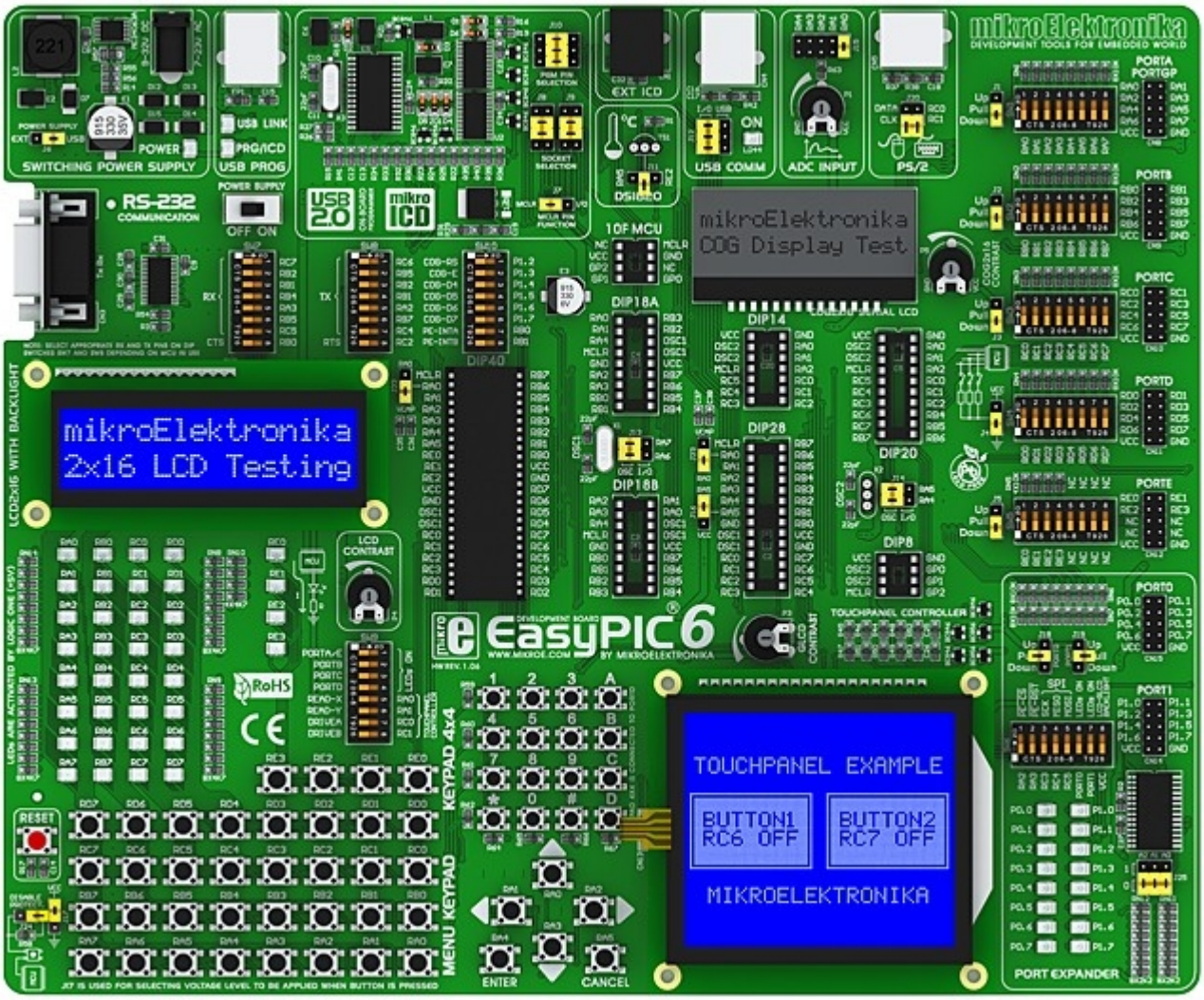


https://twitter.com/webmaster442

2. Fejlesztéshez szükséges eszközök

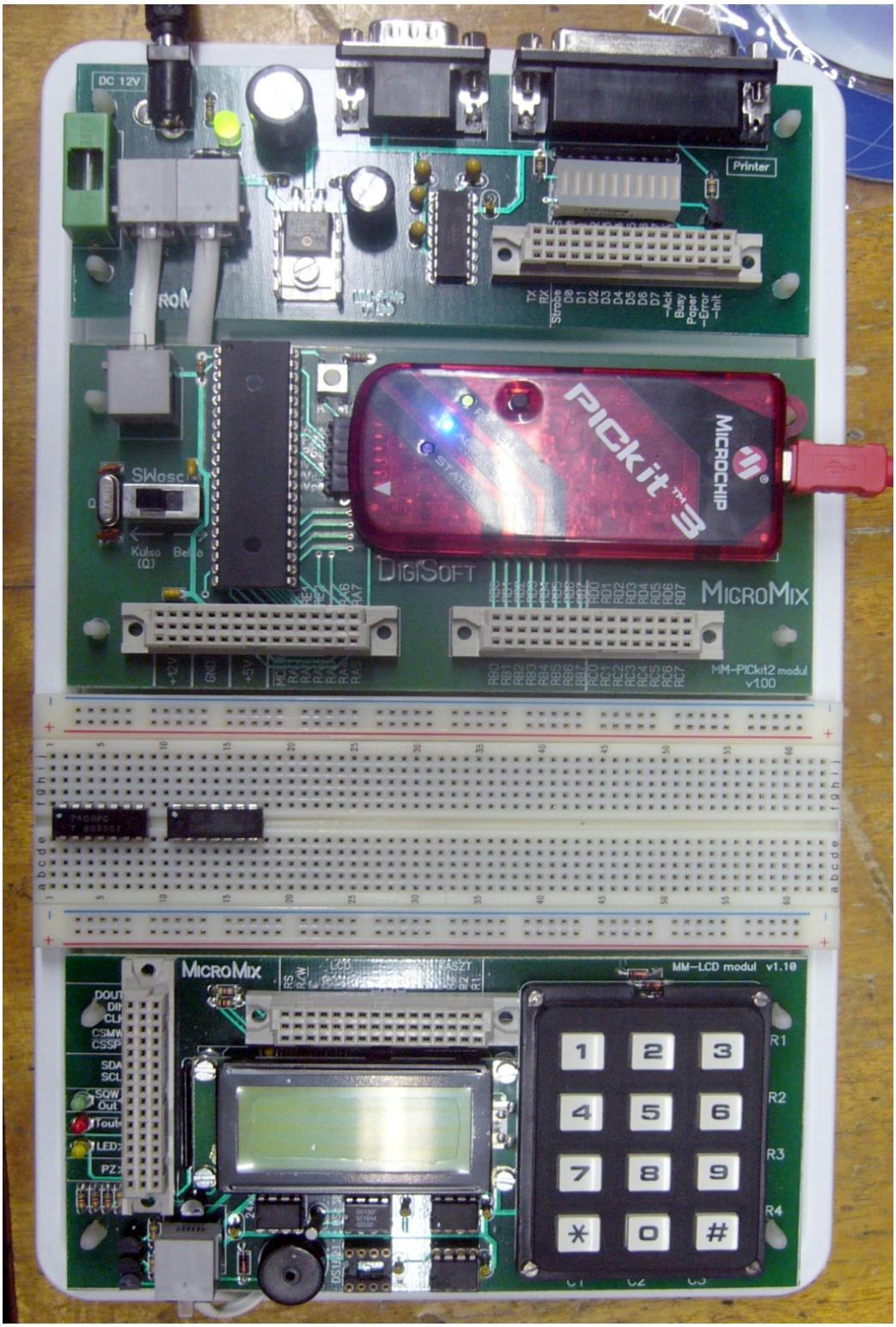
PIC fejlesztéshez szükséges eszközök

PIC fejlesztéshez használhatóak előre elkészített fejlesztőkörnyezetek, mint pl. a Mikroelektronika által gyártott EasyPIC és BigPIC fejlesztőkészletek is. Ezen eszközök előnye, hogy a lapra van integrálva a programozó, valamint a MikroC fordító példaprogramjai is ezen eszközökre készültek.



EasyPIC és BigPIC fejlesztőpanelek esetén nincs szükség külön programozó eszközre, mivel ezen fejlesztő lapok integrált programozót is tartalmaznak. A programozó eszköz programja a MikroC fordítóval együtt települ.

PIC oktatásra a MicroMIX oktatókészletet használom, amelyet a DigiSoft gyárt itthon, Magyarországon. Ezen oktatókészlet előnye az, hogy szabadabb felhasználást biztosít a többi készlettel szemben, mivel itt a felhasználónak lehetősége van saját magának elvégeznie a bekötéseket. Programozó eszköznek ezen készlet a PICKit 2-t vagy PICKit 3-at tudja fogadni. Korábbi változatai egy egyedi, soros porton illeszkedő programozót használtak.


DC 12V
Printer
PICkit 3
MICROCHIP
DIGISOFT
MICROMIX
MM-PICkit2 modul
v1.00
MICROMIX
MM-LCD modul v1.10
1
2
3
4
5
6
7
8
9
*
0
#
R1
R2
R3
R4

Programozó eszköz: PICKit

Minden mikrovezérlőbe a programot egy speciális eszközzel, a programozóval lehet betölteni. A PICKit programozó eszközt maga a PIC vezérlők gyártója, a Microchip fejleszti. Jelenleg forgalomban lévő változata a 3-as, amely a népszerű 2-es széria javított változata. Ezen eszköz az összes létező PIC vezérlő programozására használható.

Beszerzési helytől függően egy ilyen programozó ára 9-12 ezer forint körül mozog. De mivel a PICKit 2 kapcsolási rajza nyilvános, ezért a vállalkozó kedvű olvasó akár saját maga is készíthet egyet. Sőt, az interneten számos PICKit klón projektet lehet találni. Azonban, ha saját programozó építése mellett dönt a kedves olvasó, akkor tisztában kell lennie azzal, hogy ezen eszközt is fel kell programozni, vagyis szüksége lesz még egy programozóra.

Programozás mellett ezen eszközök MPLAB (Mikrochip által fejlesztett PIC fejlesztő környezet) segítségével használhatóak hardveres debugger eszköznek is azon mikrovezérlőkkel, amelyek támogatják ezt az opciót.

Továbbá MPLAB segítségével lehet kihasználni az eszköz úgynevezett egy gombnyomásos programozási funkcióját. Ennek lényege az, hogy a programozó rendelkezik beépített memóriával, amelybe egy .hex fájl tölthető be. Ezután a programozónak csupán tápfeszültségre van szüksége a program feltöltéséhez a céleszközre. A programozási folyamat ebben az esetben az eszközön elhelyezett gomb segítségével indítható.

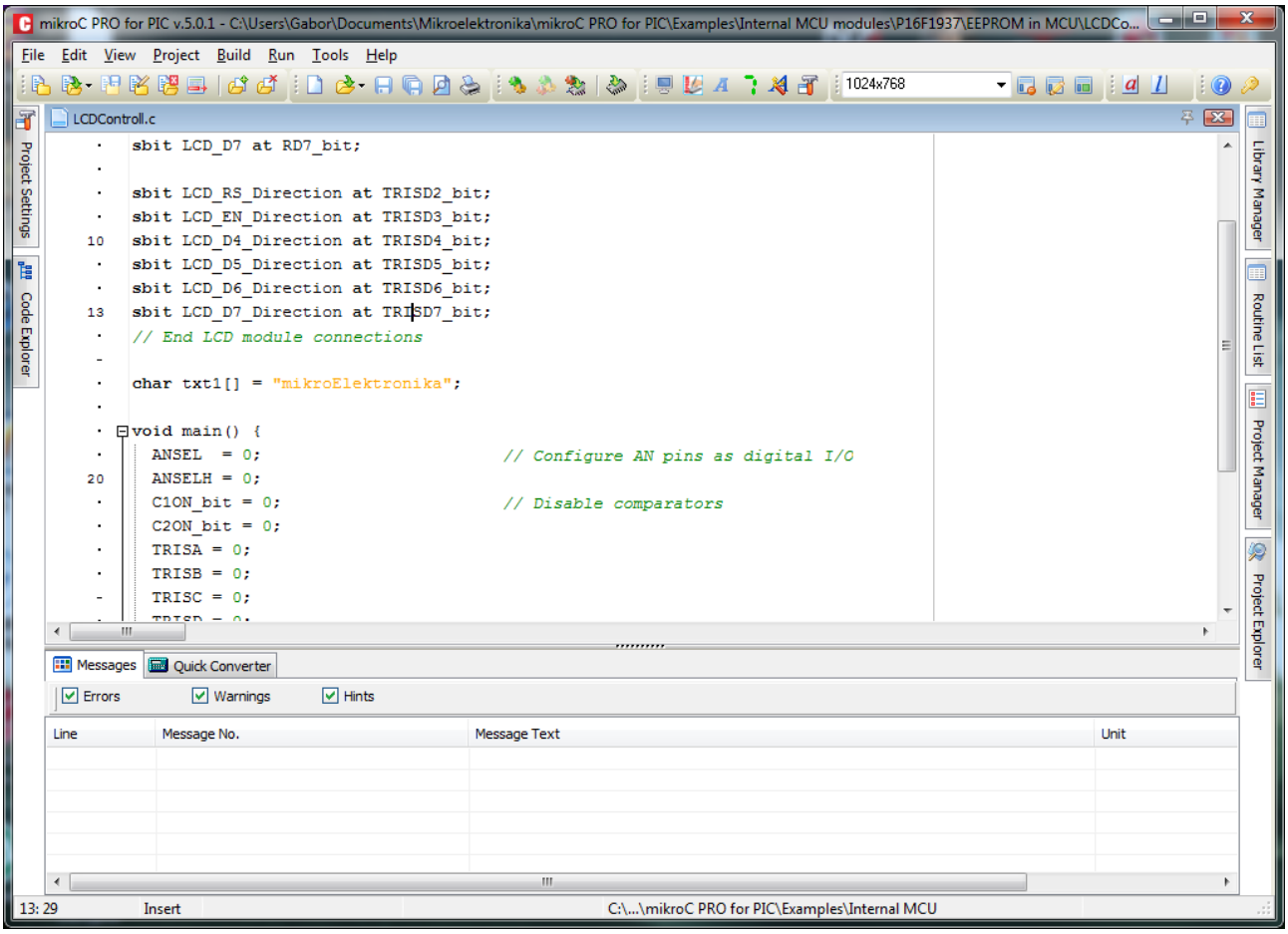
MPLAB mellett van ezekhez az eszközökhöz egy különálló programozó szoftver is. Így, ha csak égetési célokra van használva az eszköz, akkor nem muszáj telepíteni egy több száz megabyte-ot foglaló teljes értékű fejlesztőkörnyezetet. A 3-as sorozat esetén sehol sincs megemlítve a dokumentációban, de a különálló programozó szoftver működéséhez telepíteni kell a .NET Framework 3.5-ös változatát Windows 7 előtti rendszereken.

Fordítóprogram: MikroC

A MikroC a Mikroelektronika által fejlesztett C fordító program PIC eszközök számára. Néhány kisebb, a mikrovezérlők sajátosságából adódó ok miatt teljesen szabványos C fordító program. C nyelv esetén itt ténylegesen C értendő, objektumorientáltság nélkül.

A Mikroelektronika cég fejleszti, eredetileg az EasyPIC és BigPIC fejlesztőkészleteikhez lett kifejlesztve, ezzel megkönnyítve a PIC mikrovezérlők alkalmazásának megtanulását és felhasználását. Természetesen a program használható más fejlesztőkészletekkel is.

Szintén nem ingyenes program, azonban ingyenes üzemmódban is használható, azzal a megkötéssel, hogy a lefordított kód mérete nem érhet el egy bizonyos méretkorlátot. Szerencsére a méretkorlát relatíve igen nagy, így kisebb projektjeinkhez gond nélkül alkalmazható az ingyenes változat is. Komolyabb programokhoz természetesen ajánlott a teljes verzió beszerzése, amely nem rendelkezik méretkorláttal a lefordítandó program méretét illetően.



Arduino 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, de nagymértékben megkönnyíti a fejlesztést:

- *Egy breadboard kísérletezéshez*
- *Kábelek a breadboard részeinek összekötéséhez*
- *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.*

Amennyiben nem kívánunk foglalkozni az egyes elemek egyedi beszerzésével, akkor több gyártó is forgalmaz Arduino kezdő készleteket, amelyek nagyjából a fenti elemeket és más kiegészítőket is tartalmaznak még. Ez forgalmazótól és gyártótól függően igen drága is lehet, így vásárlás előtt érdemes tájékozódni az árakról.

Az Arduino készítők is forgalmaznak egy hivatalos kezdő csomagot, ami egy Uno lapot is tartalmaz, valamint egy könyvet is, amiben pár projekt dokumentálva van.

SOC fejlesztéshez szükséges eszközök

A SOC lapokra való szoftver fejlesztéshez szükséges maga a SOC lap. Ezen felül, hogy mire van szükség még azt maga a lap és a megvalósítandó projekt határozza meg. Mivel a könyvben a RaspberryPi van részletesen tárgyalva, ezért a RaspberryPi beüzemeléséről szóló szekcióban részletesen le van írva, hogy mi kell ahhoz, hogy a Pi-t működésre bírjuk.

Konkrét laptól függetlenül szükségünk lehet tartalék SD kártyára, ebből egy 4GB méretű beszerzése javasolt, microSD formátumban, mivel egyes lapok micro kártyát fogadnak, mások pedig normál méretűt. A microSD kártyákhoz pedig szoktak átalakító kártyát mellékelni.

Ezen felül szükségünk lehet perifériákra: egér, billentyűzet, monitor. Billentyűzet és egér esetén adódik az USB csatlakozás. A monitor esetén nem ennyire egyértelmű a helyzet. A Pi rendelkezik HDMI kimenettel, ami segítségével könnyen csatlakoztatható HDMI vagy DVI bemenettel rendelkező monitorokhoz. A HDMI csatlakozó passzívan, elektronika nélkül konvertálható DVI csatlakozásra, mivel a kettő egymással kompatibilis.

Analóg VGA jellé azonban a HDMI-n keresztül érkező jel nem konvertálható, csak aktív átalakítóval, amelyek igencsak megbízhatatlanok tudnak lenni és márkától és gyártótól függően sokba is kerülhetnek.

Egyes lapok, mint a Pi rendelkeznek kompozit videó kimenettel is, amely segítségével analóg TV-hez vagy kompozit bemenetes TV készülékhez lehet őket csatlakoztatni. Lehet kapni olyan VGA átalakító kábeleket is, amelyek kompozit videó jelből VGA-t konvertálnak. Ez egy passzív szűrős átalakítási módszer, ebből adódóan olcsó. A megoldás problémája az, hogy nem plug & play. Tehát a monitor frissítési frekvenciáját és a kép méretet ismernünk kell és manuálisan kell konfigurálnunk.

Komolyabb szoftverek fejlesztése esetén célszerű lehet egy Linux rendszert telepíteni a fejlesztő gépre és ott elvégezni a fejlesztéseket. Ennek előnye, hogy egy nagyobb teljesítményű gép hamarább végez a szoftver fordításával, mint egy kis teljesítményű SOC lap.

A Linux rendszert nem muszáj fizikai gépre telepíteni, virtuális gépet is alkalmazhatunk. Virtuális gépet akkor érdemes használni, ha legalább 4GB memóriánk van és a számítógép processzora támogatja az Intel VT-X vagy AMD-V utasításkészleteket. AMD processzorok esetén az összes Athlon64 után megjelent processzor támogatja ezen funkciót, Intel processzorok esetén az összes i szériás (i3, i5, i7 generációtól függetlenül) modell és a Quad Core Core 2 processzorok támogatják.

CPLD / FPGA fejlesztéshez szükséges eszközök

CPLD vagy FPGA fejlesztéshez szükségünk lesz egy program feltöltőre és egy CPLD vagy FPGA eszközre. FPGA eszközök esetén a tanulást, programozást érdemes egy komplett fejlesztő vagy bemutató lappal kezdeni, mivel a legtöbb igen jó teljesítményű FPGA csak BGA tokozásban érhető el, ami nem éppen arra lett kitalálva, hogy otthoni körülmények között szerelhető legyen.

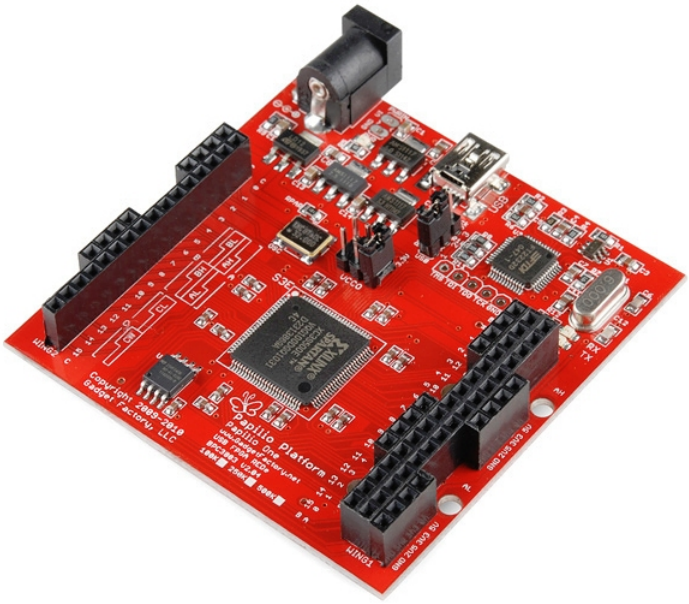
CPLD lapok esetén is érdemes egy fejlesztő vagy bemutató lappal kezdeni, mivel ezen eszközök is érdekes tokozásokban lelhetőek fel, valamint ahány logikai eszköz, annyi felépítés. Egy „mezei" CPLD esetén is előfordulhat, hogy 2-3 tápfeszültség szükséges a meghajtásához, ami tovább bonyolítja a terveket.

Két fő CPLD és FPGA gyártó van. Az Altera és a Xilinx. Mindkét gyártó rendelkezik saját tervező szoftverrel és program feltöltővel, amelyek természetesen nem kompatibilisek egymással.

A könyv logikai eszközökről szóló szekciójában egy Altera MAX II családba tartozó CPLD lapon keresztül van bemutatva a Verilog programozás. A verilog kódok gyártótól függetlenül hordozhatóak logikai eszközök között, eltérés csak a szoftver használatában és a programfeltöltőben van.

Azt érdemes megjegyezni, hogy az FPGA fejlesztő lapok nem igen olcsóak, akár több száz dollárba is kerülhetnek, így kezdésnek én egy CPLD lapot ajánlok, aztán ha annak a képességeit sikerült kinőni, akkor érdemes elgondolkodni egy FPGA lap beszerzésén.

FPGA lapokból rengeteg létezik, talán a legérdekesebb a Papilo nevezetű lap, amely egy Xilinx FPGA chip-en alapul és szoftveresen egy arduino környezet kompatibilis mikrovezérlőt valósít meg, 96Mhz órajellel, 32 bites architektúrával és 8Mb kód tárolási lehetőséggel. A lap a http://papilio.cc/ oldalon keresztül vásárolható meg.



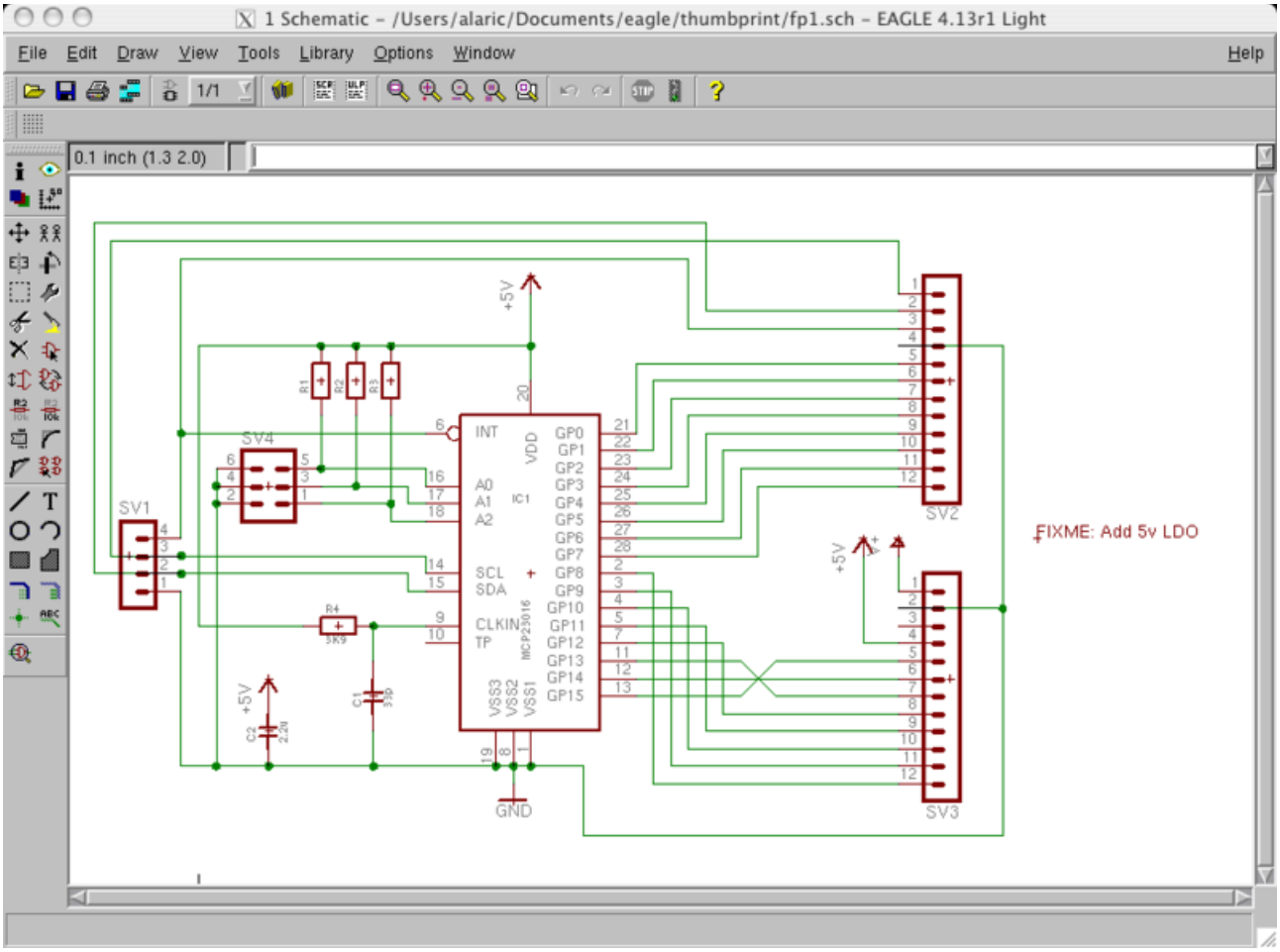
Környezettől független eszközök

Rajzoló program: EAGLE

Az EAGLE kapcsolási rajz szerkesztő szoftver az elmúlt években gyakorlatilag ipari szabvánnyá vált. Ezt elsősorban annak köszönheti, hogy kész kapcsolási rajzból automatikusan tud nyomtatott áramköri tervet készíteni, egy-két paraméter megadása után. Széles körű és jól karbantartott alkatrész könyvtárral rendelkezik, valamint akár 16 réteges nyomtatott áramkört is képes tervezni.

Nem olcsó program, azonban pár megszorítással ingyenesen is használható. Ezen megszorítások közé tartozik az, hogy limitálva van a nyomtatott áramköri terv mérete, valamint az automatikus tervezés le van tiltva. A felhasználási feltételek és a korlátozások verziótól függően változhatnak, ezért érdemes tájékozódni róla.

Használatáról nem lesz szó a könyvben. Azonban, ha komplett projektet készítünk, amit közzé akarunk tenni akár interneten is, akkor érdemes ezen program segítségével elkészíteni a rajzot.



Verziókezelő szoftver

Profi fejlesztési munkák előtt érdemes egy verziókezelő rendszert beállítani, hogy a módosításainkról legyen egy részletes másolat.

Egy verziókezelő rendszer azt a feladatot látja el, hogy minden egyes fontosabb módosításról tárol egy biztonsági másolatot. Ezen biztonsági másolat készítése lehet automatikus, vagy a felhasználó által kezdeményezett. A másolatok általában inkrementális biztonsági mentéssel készülnek, ami azt jelenti, hogy az eredeti kiinduló állapothoz (első biztonsági mentés) képest csak a változások lesznek tárolva. Ezen felül egy verziókezelő rendszer használata lehetővé teszi azt is, hogy több ember dolgozzon egyszerre egy projekten hatékonyan, egymás munkáját nem zavarva.

Mivel verziókezelő rendszerekből annyi van, mint égen a csillag, ezért érdemes olyat választani, ami a fejlesztőknek kezükre áll. Mivel Windows lesz az alapértelmezett fejlesztési platformunk, ezért kézenfekvő olyat választani, ami rendelkezik megfelelő Windows támogatással. Néhány rendszer, amivel érdemes foglalkozni, és ezek főbb jellemzői:

- ***CVS***

  Talán a legrégebbi nyílt forráskódú verziókezelő rendszer. Számos alkalmazás és grafikus kliensprogram támogatja. A gond vele az, hogy egy picit elavult, valamint szolgáltatások tekintetében nem igazán fejlesztett.
- ***SVN***

  A CVS verziókezelő utódja. Létrehozásakor a fejlesztők célja a CVS hiányosságainak és hibáinak pótlása volt. Hasonlóképpen működik a CVS-hez. Apache HTTP szerverhez modulként telepíthető vagy különállóan futtatható. Néhanapján igencsak lassú tud lenni, főleg, ha Apache modulként van telepítve. Nagyon jó a támogatottsága.
- ***Bazaar***

  A Canonical INC. fejlesztette ki, eredetileg az Ubuntu disztribúció fejlesztésének megkönnyítéséhez. Jelenleg is ezt használják. Számos eszköz érhető el hozzá, de ennek ellenére nem túlzottan elterjedt.
- ***GIT***

  A Linux kernel fő fejlesztője, Linus Torvalds találta ki és hozta létre, hogy megkönnyítse munkáját a Linux kernel fejlesztésén. Célja a gyorsaság elérése volt, mivel a sebességet illetően egyik verziókezelő sem nyűgözte le. Az utóbbi időben egyre szélesebb körben elterjedt, azonban Windows alá nincs sok eszköz a használatához.

Mérőeszközök

Előfordulhat, hogy az összeállított, tervezett kapcsolás nem úgy működik, mint ahogy mi azt szeretnénk. A hibák elhárítása lehetetlen mérőeszközök nélkül. A legalapvetőbb mérőeszköz egy multiméter. Ennek használatáról és arról, hogy vásárláskor mire kell figyelni az elektronikai alapismeretek szekcióban külön írtam.

Komolyabb kapcsolások teszteléséhez azonban a multiméter nem lesz elég, ehhez más műszer kell. Méghozzá egy oszcilloszkóp. Az oszcilloszkóp képes megmutatni a jel pillanatnyi változásait, míg a multiméter csak az effektív értéket mutatja egy jelből.

Az oszcilloszkóp tipikusan nem olcsó mérőeszköz. Régi, analóg elven működő oszcilloszkópok viszonylag olcsón beszerezhetőek, viszont ezek nem igen alkalmasak digitális elektronikai áramkörök mérésére, mivel általában nem rendelkeznek tároló funkcióval, így a gyors változások nem követhetőek le szépen. A digitális és az analóg oszcilloszkópok esetén az árt leginkább a bemeneti sávszélesség határozza meg. Amennyiben a szkópot csak mikrovezérlős környezetben szeretnénk használni, akkor bőven elég egy 24MHz-es sávszélességű modell. Azonban ha SOC és CPLD/FPGA eszközök tesztelésre is alkalmazni szeretnénk a szkópot, akkor legalább egy 50 vagy 75MHz sávszélességű változatra van szükségünk. A legtöbb oszcilloszkóp legalább két független bemeneti csatornával rendelkezik. Ez azért jó, mert egy áramkör esetén egy időben láthatjuk a bemeneti és kimeneti jelet. Szkóp vásárlás esetén ügyeljünk arra, hogy legalább két csatornája legyen.

Oszcilloszkópokból léteznek USB változatok is. Ezek belépő, kezdő szintű eszköznek bőven megteszik, mivel egy új digitális szkóp márkától és tudástól függően akár több százezer forint is lehet.

Logikai jelek analizálásához vizsgálatához nem árt egy logikai analizátor sem. Ezen eszközök a digitális jelek változásait tárolják le és rajzolják időfüggvényre. Ezen eszközöknek két fő változata létezik. Az olcsóbb változatok nem rendelkeznek beépített memóriával, ezért a sávszélességük korlátolt az USB átviteli sebességéhez. (mivel tipikusan ezek USB eszközök) Az ilyen analizátorok előnye, hogy végtelen sok memória áll rendelkezésre a számítógépen mintavételezésre és hibakeresésre, de limitált a sávszélesség. A limit általában 24-32MHz környékén van.

A másik fő típusa a logikai analizátoroknak azok amelyek rendelkeznek beépített memóriával. Ezen eszközök esetén fő jellemző a memória nagysága. Értelem szerűen ez minél több, annál jobb. Felépítésükből adódóan ezen típusok igen nagy frekvencián is működőképesek.

Logikai analizátorból csak olyan típust szabad megvásárolni, amely rendelkezik protokoll dekódolással. A protokoll dekódolás lehetővé teszi, hogy a $0 \longrightarrow 1$ és $1 \longrightarrow 0$ átmenetek helyett a küldött adatot lássuk egy komplexebb protokoll, mint SPI vagy I2C esetén.

A logikai analizátorok esetén fontos jellemző még a bemeneti csatornák száma. A legelterjedtebb modellek 8 csatornával rendelkeznek, amely segítségével egy 8 bites párhuzamos busz analizálható. A drágább, komolyabb változatok 16 csatornával rendelkeznek.

A komolyabb oszcilloszkópok rendelkeznek beépített Logikai analizátorral is.

Szoftverek beszerzési helyei:

Mplab Assembly fordító PICKit programozó és debug alkalmazás:

- *http://www.microchip.com/en_us/family/mplabx/index.html*

Mikrochip PICKit 2 weboldala:

- *http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805*

Mikrochip PICKit 3 weboldala:

- *http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en538340&redirects=pickit3*

Mikroelektronika MikroC weboldala:

- *http://www.mikroe.com/eng/products/view/7/mikroc-pro-for-pic/*

EAGLE rajzolóprogram:

- *http://www.cadsoftusa.com/eagle-pcb-design-software/?language=en*

Raspberry Pi weblap:

- *http://www.raspberrypi.org/*

Arduino weblap:

- *http://arduino.cc/*

Altera Quatrus:

- *http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html*

CVS verziókezelő:

- *http://cvs.nongnu.org/*

SVN verziókezelő:

- *http://subversion.apache.org/*

Bazaar verziókezelő:

- *http://bazaar.canonical.com/en/*

GIT verziókezelő:

- *http://git-scm.com*

3. C nyelv alapok

A C programozási nyelv részletes megtanulásához, komolyabb trükkök elsajátításához ezen könyv anyaga igencsak kevés. Itt csak a mikrovezérlők programozásához feltétlenül szükséges alapismeretek lesznek közölve. Így, ha a kedves olvasó többet szeretne tudni a C programozási nyelv rejtelmeiről, akkor érdemes egy másik könyvet keresnie. Kiváló könyvek kaphatóak kereskedelmi forgalomban akár magyar, akár angol nyelven is.

A C nyelvet Dennis Ritchie fejlesztette ki 1969 és 1974 között, tehát nem egy friss programozási nyelv. Ez idő alatt igencsak ipari szabvánnyá vált a programozási nyelvek között.

Célszerű assembler helyett C-ben fejleszteni programokat mikrovezérlőkre, hiszen szinte minden eszközhöz található C fordító, valamint így elég egy nyelvet megtanulni a fejlesztéshez, nem kell fejben tartania a fejlesztőnek az adott eszköz utasításainak listáját. További előny az eszközök közötti hordozhatóság. Szerencsés esetben egyáltalán nem kell módosítani a programot. Rosszabb esetben is könnyebb átírni azt, mintha assembly nyelven lenne írva.

Szükséges alapismeretek

C programozási nyelv esetén a legfontosabb tudnivaló talán az, hogy a kis-és nagybetűs nevek különbözőek. Tehát az Alma és alma változónevek nem ugyanazt jelentik a fordítóprogram számára.

Alapértelmezetten nincs logikai bool típus definiálva, amely a kifejezések igaz vagy hamis voltát fejezné ki. Ehelyett egész számokat használ logikai értékek jelölésére. Ha egy kifejezés értéke 1, akkor azt igaznak tekinti a program, ha pedig nem 1, akkor hamisnak.

Az utasítások után egy pontosvessző karaktert kell tenni. A fordítóprogram számára ez jelzi egy utasítás végét.

A // karaktersorozattal kezdődő utasításokat a fordítóprogram nem veszi figyelembe, megjegyzésnek tekinti ezen szövegrészleteket. Ez egy C++ újítás, amit nem minden fordítóprogram támogat. Ezért célszerű a C szabványban definiált / és */ jelsorozat között elhelyeznünk megjegyzéseinket.*

Alap adattípusok

- **char**
  - *Egész szám, a legkisebb, a gép által címezhető adatmennyiség. 8 bit hosszú típus. Alapértelmezetten előjel nélküli, így 0 és 255 közötti egész szám értéket képes kifejezni, vagy egy karakter tárolására alkalmas az ASCII kódtábla szerint.*
- **short int**
  - *Rövid egész szám, a gép által címezhető alapegység fele. MikroC esetén 8 bit, 32 bites rendszerek esetén 16 bit hosszú, alapértelmezetten előjeles.*
- **int**
  - *Egész szám, a gép által címezhető alapegység. MikroC esetén 16 bit hosszúságú, 32 bites rendszerek esetén 32 bit hosszú, alapértelmezetten előjeles.*
- **long int**
  - *Egész szám, a gép által címezhető alapegység kétszerese. MikroC esetén 32 bit hosszúságú, 32 bites rendszerek esetén 64 bit hosszú, alapértelmezetten előjeles.*
- **float**
  - *32 bites lebegőpontos szám, IEEE 754 szabvány szerinti egyszeres pontosság.*
- **double**
  - *64 bites lebegőpontos szám, IEEE 754 szabvány szerinti dupla pontosság. MikroC fordítás közben átírja float-ra, mivel a 8 bites mikrovezérlőkben nincs elég számítási teljesítmény 64 biten számoláshoz.*
- **long double**
  - *128 vagy 80 bites lebegőpontos szám. Fordítótól függően az IEEE 754 szabvány szerinti négyszeres pontosságot jelöli vagy a 80 bites kibővített pontosságot. MikroC esetén szintén float lesz.*

	Változó típus	***Hossz 32 biten***	***Minimum érték 10-es számrendszerben***	***Maximum érték 10-es számrendszerben***
Előjeles	*char*	*8 bit*	*–128*	*127*
Előjel nélküli			*0*	*255*
Előjeles	*short int*	*16 bit*	*–32 768*	*32 767*
Előjel nélküli			*0*	*65 535*
Előjeles	*int*	*32 bit*	*–2 147 483 648*	*2 147 483 647*
Előjel nélküli			*0*	*4 294 967 295*
Előjeles	*long int*	*64 bit*	*–9 223 372 036 854 775 808*	*9 223 372 036 854 770 000*
Előjel nélküli			*0*	*1 844 674 407 3 709 500 000*

1. táblázat. Változó típusok bithosszúsága és értékeik 32 bites C esetén

Típusmódosító szók

- **auto**

  A változót csak abban a blokkban teszi láthatóvá, ahol változó definiálva van. Alapértelmezetten nincs értéke, definiáláskor meg kell adni egy kezdőértéket. Nem kötelező a használata, hiszen az összes változó e módon jön létre.
- **const**

  Hatására a változó értéke konstanssá válik, egyszer lehet neki értéket adni a változó definiálásakor, később nem.
- **extern**

  Azt jelzi a fordító számára, hogy az extern módosítóval megjelölt változó egy másik forrásfájlban van definiálva. Ezzel a módszerrel egy másik forráskód fájljában definiált változó értékéhez hozzáférhetünk.

Továbbá az extern kulcsszó használható függvények előtt is.

- **register**
  Hatására a változó egy CPU regiszterben tárolódik a memória helyett. Ez nem biztosítható minden esetben, úgyhogy a fordító csak egy ajánlásnak veszi fordítás közben, ha nincs akadálya a register tárolási módnak, akkor használni fogja.
- **signed**
  Kikényszerített előjelesség. Nem használható az alábbi típusok esetén: float, double, long double. Használatának a char típus esetén lehet kiemelten értelme, mivel fordítóprogramtól függő, hogy a char előjeles, vagy előjel nélküli.
- **static**
  Hatására a változó statikus lesz. Amennyiben egy változó statikusnak van jelölve egy függvényben, akkor értékét megőrzi a funkcióhívások közben is. Mivel az értéke futás közben inicializálódik, ezért a kezdőértéknek mindenképpen konstansnak kell lennie. A globális (függvényeken kívüli, minden függvény számára elérhető) változók is statikusként definiálódnak külön jelzés nélkül is.
- **unsigned**
  Kikényszerített előjel nélküliség. Nem használható az alábbi típusok esetén: float, double, long double. Használatának a char típus esetén lehet kiemelten értelme, mivel fordítóprogramtól függő, hogy a char előjeles, vagy előjel nélküli.
- **volatile**
  Hatására a változó hozzáférhető lesz egy másik program számára. Ezen változók minden esetben a memóriában tárolódnak. Mikrovezérlők esetén ezt a típust nem fogjuk használni, mivel a fő programon kívül másik program nem fut az eszközön.

Operátorok

A C nyelvben használható operátorok a következőek:

"" '' + - * / % ! = () | ^ []

< > & {} ?: -> != += -= /= *= ~

|| . && == ++ – sizeof

Alap műveleti operátorok:

- + összeadás
- - kivonás
- * szorzás
- / osztás
- % maradékos osztás. Eredményül az osztáskor keletkező maradékot adja vissza.

Bitenkénti logikai operátorok

- ~ bitenkénti negáció. Előjeles számtípusok esetén, ha a szám pozitív, akkor érdekes eredményt ad vissza negálás után, mivel az előjelbitet is negálja. A negatív előjelű számokat pedig a program kettes komplemens alaknak érzékeli.
- | bitenkénti vagy
- & bitenkénti és
- ^ bitenkénti kizáró vagy

Logikai operátorok

- ! logikai nem
- == logikai egyenlőség
- != logikai nem egyenlőség
- || logikai vagy
- && logikai és
- < logikai kisebb, mint
- > logikai nagyobb, mint

Értékadó operátorok

C esetén a sima egyenlőségjeles értékadás mellett vannak speciális értékadó operátorok, melyek segítségével időt spórolhatunk meg. Ezek az operátorok:

+= -= *= /=

Ezen operátorok azt teszik, hogy az előttük lévő művelet elvégzésével értéket adnak a változónak. Példa:

```
int x = 1;
x += 3;
```

A fenti kódrészlet azt eredményezi, hogy az eredetileg 1 kezdőértékkel ellátott x változó értékét megnöveli 3-mal.

Értéknövelő-és csökkentő operátorok

Más szóval inkrementáló és dekrementáló operátorok. A ++ operátor eggyel növeli, míg a -- operátor eggyel csökkenti a kifejezés értékét. Állhat a kifejezés előtt és után is, azonban így módosulnak a hatásaik. Amennyiben az operátorok valamelyike egy kifejezés előtt áll, az azt jelenti, hogy a további műveletek elvégzése előtt fog megtörténni a növelés vagy a csökkenés. Amennyiben kifejezés után áll, akkor a csökkentés vagy növelés lesz utoljára végrehajtva.

Blokk zárójelek

A kapcsos { } zárójeleket C esetén blokk zárójeleknek nevezik, mivel minden egynél több utasítást tartalmazó vezérlési blokk esetén a használatuk kötelező.

Tömb zárójelek

A tömb adattípusok jelölésére a szögletes [ ] zárójelek használatosak.

Sima zárójelek

A sima zárójelek ( ) C esetén kifejezések zárójelezésén kívül vezérlési ciklusokban is használtak.

Idézőjelek

Az aposztróf ' ' jelek karakterek megadására használtak, míg az idéző ” ” jelek szövegek megadására szolgálatosak.

A kérdőjel kettőspont operátor:

A C nyelv egyetlenegy 3 operandusú operátora. Működését tekintve leginkább egy egyutasításos if elágazásra hasonlít.(lásd feltételes vezérlési szerkezetek → if szerkezet).

Ez az operátor azt csinálja, hogy amennyiben a kérdőjel előtt meghatározott kifejezés értéke igaz, akkor a kérdőjel után lévő utasítás fog végrehajtódni. Amennyiben hamis, akkor a kettőspont után lévő rész fog végrehajtódni.

Sizeof operátor:

Megjelenését tekintve a sizeof operátor könnyen összekeverhető egy függvénnyel, azonban operátorról van szó. Ezen operátor a paraméterként kapott típusnak a méretét adja vissza byte-ban.

```
int meret = sizeof(long);
```

A fenti példában szereplő méret változó értéke 8 lesz. Ezen operátor használata különösen hasznos lesz majd struktúrák és unió típusok esetén.

A C nyelv kulcsszavai

auto break case char const continue default do double else enum extern float for goto if int inline long register return short signed sizeof static struct switch typedef union unsigned void volatile while

Egész számok megadása

C nyelvben az egész számok megadhatóak tízes, tizenhatos (hexa), nyolcas (oktális), és kettes (bináris) számrendszerben. Alapértelmezett formátum a tízes számrendszer. Ekkor sem a szám elé, sem a szám után nem kell módosítót tenni. Azonban vannak esetek (főleg mikrovezérlők esetén), amikor más számrendszer formátumok jól jönnek.

- *A fordító bináris számként értelmez egy számot, ha az 0b karakterekkel kezdődik.*
- *A fordító oktális számként értelmez egy számot, ha az 0 karakterrel kezdődik.*
- *A fordító hexadecimális számként értelmezi a számot, ha az 0x karakterekkel kezdődik.*

```
int tízes = 15;
int binaris = 0b1010;
int oktalis = 012;
int hexa = 0xa
```

Decimális	*Bináris*	*Oktális*	*Hexadecimális*
0	*0000*	*0*	*0*
1	*0001*	*1*	*1*
2	*0010*	*2*	*2*
3	*0011*	*3*	*3*
4	*0100*	*4*	*4*
5	*0101*	*5*	*5*
6	*0110*	*6*	*6*
7	*0111*	*7*	*7*
8	*1000*	*10*	*8*
9	*1001*	*11*	*9*
10	*1010*	*12*	*A*
11	*1011*	*13*	*B*
12	*1100*	*14*	*C*
13	*1101*	*15*	*D*
14	*1110*	*16*	*E*
15	*1111*	*17*	*F*

2. táblázat. Egész számok decimális, bináris, oktális és hexadecimális számrendszerben 0-tól 15-ig

Feltétel nélküli vezérlési szerkezetetek

goto szerkezet

A C nyelv egyetlenegy, feltétel nélküli vezérlési szerkezete. Arra jó, hogy a program futtatását egy másik helyre irányítsa át. A helyek megadása címkék segítségével történik. Példa:

```
int x = 0;

cimke:
     x++;
     goto címke;
```

A fenti kódrészlet azt teszi, hogy x értékének növelése után a vezérlés a címke nevezetű résznél folytatódik, ami x értékét növeli. Könnyen belátható, hogy ez egy végtelenségig ismétlődő folyamat, amit programozásban **végtelen ciklus***nak neveznek. Mivel könnyen bele lehet kavarodni a használatába egy bizonyos szám felett, ezért nem túlzottan ajánlják az alkalmazását. Azonban egy régi mondás szerint „Egy igazi programozó csak goto-t használ”.*

Feltételes vezérlési szerkezetek

if szerkezet

A legalapvetőbb feltételes elágazásos vezérlési módszer. Amennyiben a paraméterként megadott kifejezés igaz, akkor a megadott utasítások végrehajtódnak. Megadási formátuma a következő:

```
if ( feltétel )
{
     utasítás1;
     utasítás2;
}
```

Amennyiben csak egyetlenegy utasítást szeretnénk végrehajtatni, akkor nem kötelező a blokk zárójelek megadása:

```
if ( feltétel ) utasítás;
```

if – else szerkezet

Az if szerkezet opcionális kibővítése, amely segítségével a megadott kifejezés igaz és hamis volta esetén is tudunk utasításokat végrehajtani. Megadási formája:

```
if ( feltétel ) utasítás; /* feltétel igaz volt */
else utasítás; //a feltétel hamis volt
```

Többszörösen egymásba ágyazható több if ág megadásával:

```
if (x == 0) y = 0;
else if (x == 4) y = 1;
else if (x < 4) y = 2;
else y = 3;
```

switch – case szerkezet

Ezen szerkezet akkor hasznos, ha egy változó konstans előre tudott értékeire szeretnénk reagálni megadott módon. Ezen esetben sokkal átláthatóbb és némileg gyorsabb futást eredményez a programunkban a switch-case vezérlési szerkezet alkalmazása. Valamint tisztább kódot eredményez, mint rengeteg if-else ág. Megadási formája:

```
switch ( változó )
{
     case konstans kifejezés:
          utasítás;
```

```
            break;
      default:
            utasítás;
            break;
}
```

A case szerkezetek végén lévő brake utasítás azért kell, mert ha nem lenne ott, akkor a vezérlés folytatódna a következő case ágban is. Hiányáért nem panaszkodik a fordítóprogram, ám amennyiben nincs ott, akkor a program nem fog megfelelően működni. Ezen viselkedése abból következik, hogy fordítás közben a program goto szerkezetesre alakítja át a kódot.

A default ágba a vezérlés akkor kerül, ha a fentebb megadott egyik case vezérlési eset sem teljesül. Megadása nem kötelező.

Vezérléshez egész szám típusú változókat és karaktert használhatunk fel, valamint felsorolást is, mivel ez is egész szám alapú típus.

Ciklusok

Ciklus alatt n alkalommal ismétlődő utasítássorozat értendő. Speciális ciklus a végtelen ciklus, amely sosem ér véget.

For ciklus

Ezen ciklusfajta főként tömbök elemeinek bejárására használatos. Megadási formája:

```
for ( ciklus változó kezdőérték beállítás; ciklus vége feltétel; →
     ciklus változó módosítás )
{
     utasítások;
}
```

Ciklus vezérlő változónak szintén egész szám típusok használhatóak. Ciklus vége feltétel általában a vezérlő változó egy bizonyos értékének elérése, míg a módosító utasítás általában egy inkrementálás vagy dekrementálás szokott lenni. Kezdőérték megadáskor is definiálható a vezérlő változó. A vezérlő változó a ciklus magjában szereplő utasításokban használható.

1-től 10-ig számoló ciklus:

```
for (int i=1; i<11; i++) { }
```

10-től 1-ig számoló ciklus:

```
for (int i=10; i>0; i--) { }
```

Kettesével felfelé számláló ciklus 100-ig:

```
for (int i=1; i<101; i+=2) { }
```

A ciklus végtelen lesz, amennyiben a cikluslimit, vagy a ciklusváltozó módosítás nincs megadva. Továbbá végtelen ciklust eredményez az az eset is, ahol a feltételek nem teljesülnek sosem. Példa:

```
for (int i=1; i<11; i--) { }
```

Szándékosan végtelenített for ciklusokat a következő formában szokták definiálni:

```
for ( ; ; ) { }
```

while ciklus

Az elöltesztelős ciklus megfelelője C-ben. A magjában elhelyezett utasításokat addig ismétli, amíg a vezérlési feltétele igaz. A feltétel igazságát minden egyes ciklus futásának megkezdése előtt ellenőrzi. Tehát a feltétel a ciklusban maradás feltétele. Megadási módja:

```
while ( ciklusban maradás feltétele )
{
     utasítások;
}
```

Amennyiben a feltétel mindig igaz, végtelen ciklust kapunk:

```
while ( 1 )
{
     utasítok;
}
```

A while ciklus kapcsán érdemes megemlíteni a „goes toward” (megfelelő magyar nevét nem ismerem) operátort, amely igazából nem egy szabványosított operátor. Használatára a nyelv biztosít lehetőséget. Az operátor a következőképpen néz ki: -->

Használatáról íme egy példa:

```
int mennyiszer = 12;
while (mennyiszer --> 0)
{
     //12 alkalommal ismétlődő kódrészlet.
     //mennyiszer értéke 1-el csökken 0-ig
}
```

Az operátor az értékcsökkentő és nagyobb, mint operátor összekombinálásából keletkezik. Mivel a szabvány részét nem képezi, így használata kód olvasási aggályokat vethet fel, ezért a jobb olvashatóság kedvéért ajánlatos lehet megfelelően zárójelezni a kifejezést:

```
while ((mennyiszer --) > 0)
```

do-while ciklus

Hátul tesztelős ciklus, szintén a ciklusban maradás feltételét kell megadni számára. A feltétel igaz vagy hamis voltát a ciklusmag végrehajtása után teszteli. Megadási formája:

```
do
{
     utasítások;
}
while ( ciklusban maradás feltétele );
```

Ciklusok megszakítása és folytatása

Egy ciklus bármikor megszakítható a break paranccsal. Ellentettje, a continue parancs a vezérlést a ciklus elejére teszi, ahol a ciklusban maradás feltétele tesztelve lesz.

Példa:

```
int i = 0;
while ( 1 )
{
     if ( i == 100 ) break;
     else continue;
     i++;
}
```

A fenti kódrészlet i változó értékét végtelenségig növelné, ha nem lenne benne a break utasítás, amely akkor kerül végrehajtásra, ha i értéke elérte a 100-at. Ellenkező esetben a ciklus végrehajtása az elejétől folytatódik. A continue utasítás megadása jelen esetben felesleges, mivel a vezérlés amúgy is a ciklus elejéről folytatódna, amennyiben i értéke még nem érte el a 100-at.

Bővebb típuskezelés

A típus nélküli típus

*C esetén egy speciális típus a típus nélküli típus, amit a **void** (magyarra fordítva üresség, semmi) kulcsszó jelöl. Leggyakrabban olyan függvények esetén fogunk vele találkozni, amik nem adnak vissza értéket. Ezen kívül még olyan mutatók definiálására (ezen könyv nem foglalkozik velük, mivel mikrovezérlők esetén a RAM limitált mérete miatt nincs értelme a használatuknak) szokták használni, ami bármi lehet. De ez inkább már C++ esetén jellemző.*

Típuskonvertálódási sorrend

1. *ha x long double, akkor y is long double lesz*
2. *ha x double, akkor y is double lesz.*
3. *ha x float, akkor y float*
4. *minden char és short típusú változó int típusra konvertálódik, ha az int képes kifejezni az eredeti értéket, ellenkező esetben unsigned int típusúra konvertálódnak*
5. *ha x unsigned long, akkor y is unsigned long típusúra konvertálódik*
6. *ha x unsigned int és y long, akkor, ha az unsigned int képes kifejezni a long változó értékét, akkor a változók unsigned int típusra konvertálódnak. Ellenkező esetben pedig long típusra.*
7. *ha x long, akkor y is long típusúra konvertálódik*
8. *ha x unsigned int, akkor y is unsigned int típusra konvertálódik*
9. *ha x int, akkor y is int lesz*

Típusok explicit konvertálódása

Explicit konverziónak nevezzük azon konverziót, amikor megadjuk világosan, hogy az adott típus mibe konvertálódjon, és nem hagyatkozunk a nyelv szabályaiban meghatározott konvertálódási sorrendre. Az explicit kifejezés ellentettje az implicit.

Explicit konverzió használata azon típusok és konvertálódási esetek alkalmával kötelező, amelyekről nem rendelkezik a C nyelvtana. Továbbá használható ezen típus egyértelműsítésre is. A konverzió szintaxisa:

```
uj típusú változónév = ( új típus ) régi típusú változónév;
```

Az alábbi kódrészlet simán fordul minden egyes fordító alatt, azonban sosem fog lebegőpontos eredményt produkálni az implicit rögzített szabályok miatt:

```
float lebeg = 3 / 2;
```

Explicit konverzióval azonban fog:

```
float lebeg = (float)3 / 2;
```

Saját típusok definiálása

C - ben lehetőségünk van saját típus definiálására meglévő alap típusok felhasználásával. Ehhez a typedef kulcsszót kell használni a következő formában:

```
typedef új típus név meglévő típus;
```

Az alábbi példában egy byte típus kerül definiálásra, amely valójában egy előjel nélküli karaktertípus:

```
typedef byte unsigned char;
```

A felhasználói típusok a programon belül bárhol használhatóak úgy, mint a beépített típusok.

Felsorolások

A felsorolás a programozásban egy olyan neveket tartalmazó lista, amely minden eleméhez egy numerikus érték tartozik. Ezek a numerikus értékek szabadon megválaszthatóak, vagy az értékek eldöntése a fordítóprogramra is bízható. Azonban, ha saját magunk adunk értékeket a felsorolásban szereplő elemeknek, akkor az hasznosabb. A felsorolások megadási formátuma:

```
enum felsorolás neve
{
     elem1 = 0,
     elem2 = 1,
     elem3 = 2
};
```

A felsorolást pontosvessző zárja. Az elemek nevei után lévő egyenlőségjel és számérték opcionális. Amennyiben nincs számérték az elemekhez rendelve, akkor a fordító generál hozzájuk egy számot. Maga a felsorolás neve nem hoz létre egy új típust, hanem egy változót definiál ebben a formában. A felsorolás a fentebbi formában csupán számokat (int) tartalmaz nevekkel. A fentebb említett felsorolás elemeire elem1, elem2 és elem3 néven hivatkozhatunk. Sőt, megtehetjük a következőt:

```
int ertek = elem1;
```

Amennyiben a felsorolásunkat típusként szeretnénk használni, akkor a következő formát kell használni:

```
typedef enum
{
     elem1,
     elem2
} felsorolás neve;
```

Amennyiben a programunkat egy C++ kompatibilis fordítóprogrammal fordítjuk le, akkor az első példában megadott felsorolás hasonló módon lesz lefordítva, mint az utóbbi példa. Ellenkező esetben az alábbi mintaprogram simán lefordítható, ám működése nem lesz helyes:

```
enum gyumolcs { alma, korte, szilva, barack };
enum iranyok { fel, le, jobbra, balra, jobbraatlo, balraatlo };

/*valahol később a programban*/
enum gyumolcs = balra; /*simán lefordul, mivel nem lett →
                         typedef használva*/
```

Struktúrák

A struktúrák olyan összetett adattípusok, amelyek egy, vagy több különböző típust tartalmaznak strukturált formában. Definiálásuk előtt itt is ajánlott a typedef kulcsszó, amennyiben nem változóként akarunk egyetlenegy struktúrát definiálni. Létrehozási formája:

```
typedef struct
```

```
{
     típus változó név1,
     típus változó név2
} struktúra név;
```

A struktúrában szereplő egyes változók nevére a struktúra nevén keresztül a pont operátor segítségével hivatkozhatunk. Ezt az alábbi példa szemlélteti:

```
typedef struct
{
     int x,
     int y
} pont;

pont xy;

xy.x = 10;
xy.y = -1;
```

Uniók

A C programozási nyelvben az unió típus egy igencsak érdekes struktúrát takar, mivel az unió elemei ugyanazt a memóriaterületet osztják meg. Az unió típus mérete minden esetben a legnagyobb belehelyezett elem méretével lesz egyenlő. Amiatt, hogy az elemek közös memóriaterületen helyezkednek el, az egyik elem módosítása felülírja a másik elemet. Szintén typedef kulcsszóval érdemes használni. Megadási formátuma:

```
typedef union
{
     típus változó név1,
     típus változó név2
} unio név;
```

Tömbök

A tömbök arra jók, hogy azonos típusú elemekből egy halmazt tároljunk a memóriában. Megadási forma:

```
típus változónév[méret];
```

A méret a tömb elemeinek a számát definiálja. Az elemek számozása minden esetben 0-tól indul, és n-1-ig tart, ahol n a tömb elemeinek száma.

A tömbök indexelésére csak egész számok használhatóak. Egy 15db-os egész számokat (int) tartalmazó tömb mérete 15x4 bájt lesz, vagyis 60 bájt. A tömbök méretezését különösen érdemes figyelembe venni a mikrovezérlők limitált memóriája miatt, mivel könnyen előfordulhat, hogy elfogy a szabad memória. Nagy méretű tömbök esetén ajánlott a static jelölés, ami azt eredményezi, hogy a tömb a programmemóriában fog tárolódni és nem az adatmemóriában. Ennek megfelelően a statikusnak jelölt tömbök mikrovezérlők esetén csak olvashatóak.

A tömb egyes elemeire a [] operátor segítségével hivatkozunk. A [] jelek közé a kívánt elemszámot kell írni. Egymás utáni elemek kinyerésére, vagy az elemeken való gyors végigszaladásra általában a for ciklust szokták használni. Példa:

```
int tmb[15];
int maximum = tmb[0];
for (int i=0; i<15; i++)
{
     if (tmb[i] > maximum) maximum = tmb[i];
}
```

A fenti kódrészlet annyit csinál, hogy a tmb elnevezésű tömb elemein végighaladva kikeresi a tömbben található

legnagyobb elemet.

A tömb elemeit vagy egy for ciklussal töltjük fel, vagy a tömb létrehozásakor. Létrehozáskor a következő szintaxist kell használni az elemek megadásához:

```
tömb típus tmb[] = { elem1, elem2, elem3, elem4, elemN };
```

Ezen formában nem kell a tömb elemeinek számát megadni, mivel a fordítóprogram ki tudja számolni a tömb méretét. Azonban ha meg van adva, az sem jelent gondot. Ebben az esetben azonban, ha az elemszám, amit megadunk és a tényleges elem eírás eltér, akkor fordítási hibát kapunk.

Kétdimenziós tömbök

Az egydimenziós tömböt (egy elemmel indexelt tömb) vektornak nevezik, míg a kettő dimenziós tömböt mátrixnak. Kétdimenziós tömb definiálása:

```
típus változónév[méret1][méret2];
```

Kétdimenziós tömbök használata memória szervezésük miatt nem éppen a legjobb megoldás, mivel lassul a program futása tőlük. Helyettük érdemes alkalmazni az úgynevezett sűrű tárolási algoritmust, amely segítségével egy egydimenziós tömbben tudunk tárolni elemeket úgy, mintha kétdimenziós tömböt használnánk.

Először is az elem számot kell összeszorzunk. Vagyis, ha egy 12x12 elemből álló tömböt akarunk létrehozni, az végeredményben 144 elemet fog tartalmazni. Ezért létrehozunk egy 144 elemből álló tömböt:

```
int surutomb[144];
```

Ezután a tömb egyes elemeire az alábbi képlettel tudunk hivatkozni:

```
[Oszlop * Oszlopok + Sor];
```

Ezt célszerű kettő darab egymásba ágyazott for ciklusban megtenni, így sor és oszlop szerint végighaladhatunk a tömbünk egyes elemein úgy, mintha valóban kétdimenziós lenne:

```
int elem = 0;
for (int i=0; i<12; i++)
{
     for (int j=0; j<12; j++)
     {
          elem = surutomb[j * 12 + i];
     }
}
```

Karaktertömbök

*C esetén és gyakorlatilag majdhogynem minden programozási nyelv esetén speciális tömb a karaktertömb, vagyis a szöveg típus. Egyes programozási nyelvek esetén létezik külön **string** (szöveg) típus. Azonban belső működésükben ezen típusok is lényegében karaktertömbök. C esetén ezért nincs is külön **string** típus.*

Karaktertömb definiálására használható az előzőleg ismertetett forma is, valamint egy egyszerűsített megadási mód is. Példaként definiáljuk a „Margit” nevet tartalmazó szöveget mindkét szabályos formátumban:

```
char forma1[] = { 'M', 'a', 'r', 'g', 'i', 't' };
char forma2[] = ”Margit”;
```

Ezen megadási formában nem szükséges megadni a tömb méretét. Amennyiben mégis megadásra kerül, akkor a tömb elemeinek számát eggyel növelni kell, mivel a C fordító a szöveg végére illeszti a szöveg vége jelzőkaraktert. Tehát egy 4 karakteres szövegnek 5 karakter hosszú tömböt kell definiálni.

C esetén, mint említettem, a karakter 8 bites számként ábrázolódik ASCII kódolás és kiegészítő kódtáblázatai révén. Ennek hátránya az, hogy pl. a japán vagy magyar kódlappal készült szövegek nem úgy jelennek meg egy másik kódlapon (pl. orosz), mint ahogy kellene nekik. Ezért egyes (egyre több) C fordító és könyvtárkészlet lehetőséget biztosít UTF – 16, vagy UTF – 8 formátumú szövegek kezelésére.

Az UTF – 16 kódolás 16 biten tárolja a karaktereket, ezáltal több karaktert képes ábrázolni. Visszafele kompatibilis az ASCII kódrendszerrel. Az UTF – 8 kódolás a 16 bites UTF olyan variánsa, amelyben a nem speciális karakterek (pl az A betű) egy byte-on ASCII szerint vannak tárolva, a speciális ASCII kódtáblázaton el nem férő kódok pedig 2 byte-on

Ezen speciális típusokról szintén nem esik szó ebben a könyvben, hiszen mikrovezérlők esetén nem lesz használatukra szükségünk. Továbbá programkönyvtáranként és fordítónként más ezen típusok neve és használatuk. Ezért e típusok használatáról érdemes a felhasználói kézikönyvben tájékozódni.

Szövegek és karakter változók esetén használható speciális formázó karakterek:

- *\' - aposztróf*
- *\" - idézőjel*
- *\\ - Backslash (\)*
- *\a – Sípolás*
- *\b – Backspace*
- *\n – Új sor*
- *\r – Sor elejére*
- *\t – Horizontális tabulátor*
- *\v – Vertikális tabulátor*
- *\0 – Szöveg vége jel*

Mutatók

A C nyelv legnehezebben tanulható és megérthető része a mutatók kezelése. A mutatók, más néven pointerek segítségével dinamikus memóriakezelést tudunk megvalósítani a programunkban. A dinamikus memóriakezelés azért hasznos, mert így egy adott művelethez annyi memóriát tud foglalni a programunk, amennyire szüksége lesz. Egy egyszerű gyakorlati példa erre egy olyan program, amely 2db, maximum 256 karakter méretű szöveget fűz össze. Eddig tanult statikus memóriakezeléses megoldással csak a bemenetek és kimenetek tárolására használt tömbök 1Kb memóriát foglalnának el.

Mondhatnánk, hogy manapság a számítógépeknek több gigabyte memóriája van, ez nem akkora probléma. Azonban komolyabb szoftvereknél ez a megközelítés használhatatlan, ezért mindenképpen kell dinamikusan memóriát kezelni.

*C esetén egy változó akkor lesz mutató, ha a neve elé egy * karaktert írunk. Egy egyszerű mutatót létrehozó példa:*

```
int i, *p;
p = &i;
```

A példában szereplő egyszerű kódrészlet azt teszi, hogy lefoglal i-nek és p-nek egy memóriaterületet, majd p-nek i kezdő memóriacím értékét adja. Tehát a p változó i memóriacímére mutat.

*A dinamikus memóriakezelésről itt nem lesz szó, mivel mikrovezérlők esetén nincs rá szükségünk, C esetén igencsak kényelmetlen a használatuk. (C++ nyelvben ezen probléma orvoslására került bevezetésre a **new** operátor, amely valójában a legtöbb implementációban makró a malloc() függvényre) Mutatókkal mikrovezérlős témában, hacsak nem valamilyen komoly mikrovezérlővel dolgozunk, akkor maximum szövegek megadásakor fogunk találkozni. Egy szöveget megadhatunk mutató segítségével az alábbi módon is:*

```
char *szoveg = "Ez egy minta szöveg";
```

Ez a megadási módszer nagyon hasonlít a nem megadott elemszámmal rendelkező karaktertömbös megadási formához, amely ismétlésképpen így néz ki:

```
char szoveg[] = "Ez egy minta szöveg";
```

Ebből sejtheti az olvasó, hogy lényegében a tömbök is egyfajta mutatóként működnek belső felépítésükben. Egy

tömböt kezelhetünk mutató segítségével is. Ezt az alábbi kódrészlet mutatja be:

```
int tmb[] = { 2, -1, 20, 40, 3, 4 33 };
int *ptr = tmb;
int ertek, i;

for (i=0; i<6; i++)
{
     ertek = *(ptr+i); //megegyezik az ertek = tmb[i]; értékkel
}
```

Ebben a kódrészletben a ptr mutató kezdőértékét a tömb neve határozza meg, ami lényegében egy mutató a tömb első elemének címére. A for ciklusban az értékváltozó ezután a ptr és i változót felhasználva kapja meg a helyes értéket.

Amennyiben a mutató konverzió el lenne hagyva, akkor a ptr+i értéke a tmb[0]+i értéket adná eredményül.

Függvények

A függvények programozásban olyan kódrészletek, amelyek névvel hivatkozhatóak, tetszőlegesen paramétereket is kapnak és értékkel térnek vissza. C esetén minden függvény értéket ad vissza, még azok is, amelyek nem. Ezen függvények visszatérési értéke a típus nélküli típus.

Függvények megadási formátuma:

```
visszatérési típus függvény név(vált. Típus vált. Név)
{
      return visszatérési érték.
}
```

Függvény számára átadandó változónevek közül több is megadható. Amennyiben egy függvényt úgy definiálunk, hogy 3db paramétert várjon, akkor nem hívhatjuk meg csupán két paraméterrel. Ellenkező esetben a fordító hibát fog jelezni. Természetesen a hiba a kellőnél több paraméter megadásakor is fellép.

A void típussal visszatérő függvények esetén nem kell az érték visszaadás.

*Az érték visszaadás a **return** kulcsszóval történik. A függvényblokkban ezen utasítás bekövetkezte esetén megszakad a függvény futása. Ezt felhasználva, amennyiben elágazás van a függvényben, és mondjuk az egyik elágazás esetén csak egy utasítás van, a másik esetén meg több, akkor érdemes úgy szervezni a függvényt, hogy először a kevesebb utasítást tartalmazó kódrészlet következzen be először, ugyanis ezzel gyorsítható a program működése. Ez kicsit zavaros lehet gondolom, de íme egy példa:*

```
int primteszt(int szam)
{
      if (szam != 2 && szam % 2 != 0)
      {
            for (int i=3; i<szam; i++)
            {
                  if (szam % i == 0) return 1;
            }
      }
      else return 0;
}
```

Ez némiképpen optimalizálva:

```
int primteszt(int szam)
{
      if (szam % 2 == 0) return 0;
      else if (szam == 2 || szam == 3) return 1;

      for (int i=3; i<szam; i++)
      {
            if (szam % i == 0) return 0;
      }
      return 1;
}
```

Az utóbbi kódrészlet sokkal gyorsabban fog futni, mint az előtte szereplő. Továbbá egyszerűbb is ezen kódrészlet olvasása, mert kevesebb sorból áll.

A fő függvény

C esetén kiemelt függvény a fő függvény, angolul a main függvény. Ezen függvényben kezdődik a program végrehajtása. Szokták az alkalmazás belépési pontjának is nevezni. Csak egyetlenegy lehet belőle egy programban. Fordítótól függően az alakjai:

```
void main()
{
}
```

vagy

```
int main()
{
	return 0;
}
```

Az utóbbi eset Unix-szerű rendszereken bejáratott. Ha a program futása közben nem történt semmi hiba, akkor a program 0 kóddal lép ki. Hiba esetén 0-tól eltérő kóddal.

Érték és referencia paraméter átadás

A függvények számára a paraméterek értékei alapértelmezetten érték szerint lesznek átadva. Ez azt jelenti, hogy mikor hívunk egy függvényt, akkor a paraméter listába helyettesített változók tartalma lesz átadva a tényleges változó helyett. Mikor a tényleges változót kapja meg a függvény, akkor az a referencia szerinti átadás.

Referencia átadás használatával lehetősége van a hívott függvénynek módosítania az átadott változó tartalmát úgy, hogy az a függvény lefutása után is megőrizze a módosított tartalmat.

Referencia átadást használó függvény definíciója:

```
visszatérési típus függvény név(vált. Típus &vált. Név)
{
}
```

Az egyetlen eltérés az érték átadást használó függvényekhez képest, hogy a változó név elé bejön egy ÉS jel (&), ami a referencia átadást jelöli. Az ilyen függvények hívásakor az átadott változónevek elé is kell ÉS jelet tenni. Példa:

```
void szoroz(int szorzo, int &mit)
{
	mit *= szorzo;
}
```

Ezen kódrészlet későbbi használata:

```
int kimenet = 2;
szoroz(2, &kimenet);
```

Értelemszerűen az alábbi függvényhívás hibát fog eredményezni, mivel nem változóval lett meghívva a függvény:

```
szoroz(2, 2);
```

Beágyazott függvények

A beágyazott függvények olyan függvények, amelyek tartalmát a fordítóprogram bemásolja a hívás helyére, így az adott függvény lényegében nem is valódi függvény, mivel függvényhívás nem történik. Ahhoz, hogy egy függvény beágyazott legyen, csupán elébe kell írni az **inline** *kulcsszót. Ezen kulcsszó, hasonlóan a* **register***-hez, ajánlás a fordítóprogram számára, mivel a legtöbb C fordító van annyira intelligens, hogy eldöntse, lesz-e gyakorlati haszna (gyorsabb lesz e a program) a beágyazásnak vagy nem. Ezen viselkedés az* **__inline** *módosítószóval kerülhető el Visual C esetén. Ennek hatására ugyanis mindenképpen beágyazott lesz a függvény. GCC fordító esetén az* ***__inline__*** *utasítás teszi ugyan ezt.*

Érdemes megemlíteni, hogy az inline függvények használata gyorsítja a program működését, mivel nem kell függvényhívással macerálnia a programnak. De ennek ára van, mégpedig programméretben lesz neki ára, mivel minden hívás helyére be kell másolódnia a tényleges függvénytartalomnak. Így, ha 6x hívunk meg egy inline függvényt, akkor a

kész kódom 6x fogja tartalmazni a függvényem teljes másolatát. A mikrovezérlők limitált programmemóriája miatt érdemes átgondolni, hogy az inline függvényeknek van-e értelme az adott szituációban.

Tipikusan olyan függvényeket szokás inline kulcsszóval megjelölni, amelyek:

- *Viszonylag kevés utasítást tartalmaznak*
- *Logikailag különálló feladatrészt végeznek (valószínűleg ezért lettek függvények)*
- *Viszonylag kevés helyen vannak meghívva cikluson kívül vagy ciklusmagban szerepelnek.*

Igazi értelme az inline függvényeknek akkor van, ha ciklusban kerülnek felhasználásra. Ebben az esetben a normál függvények drasztikus sebességromlást tudnak produkálni beágyazott, kevés memóriával rendelkező eszközökön.

Assembler utasítások

Mindegyik C fordító lehetőséget biztosít Assembler utasítások beszúrására a C programunkban. Ennek előnye az, hogy ezen kódrészletek sokkal gyorsabban futnak, mint a C kód, főleg, ha jól van optimalizálva. Ezért legfőképpen optimalizációs célokra szokták használni, illetve speciálisan olyan hardver közeli dolgok kivitelezésére, ami másképp nem lenne megoldható.

Fordítófüggő az utasítás neve, amivel Assembly nyelvre válthatunk. Ez Visual Studio alatt a következőképpen történik:

```
__asm
{
      //Assembler utasítások
}
```

MikroC és egyéb C fordítók esetén:

```
asm
{
      //Assembler utasítások
}
```

Az Assembler utasítások szintaxisa Intel vagy AT&T stílusú lehet fordítótól függően. Ezért mielőtt Assembler utasításokat kezdünk használni, érdemes tájékozódni a kézikönyvben a szintaxis felől, mivel nem kompatibilis egymással a kettő. Az Intel által definiált Assembler szintaxis a következő:

```
művelet regiszter, érték
```

Az AT&T által definiált szintaxis:

```
művelet $érték, %cím
```

A MikroC fordítója és a Microchip is az Intel szintaxist használja az utasítások leírására. Konkrétan a PIC16F887 típusú mikrovezérlő által támogatott Assembler utasítások listája és részletes leírása a könyv 5. fejezetében megtalálható. Más típusú mikrovezérlők esetén a gyártó által kiadott leírásban és dokumentációban szerepel az utasításkészlet teljes leírása. Ezen leírások elektronikus formátumban mindenki számára ingyenesen hozzáférhetőek a gyártó weblapján.

Az Előfeldolgozó

A C és C++ nyelv rendelkezik egy előfeldolgozóval, amely a tényleges fordítás előtt még munkálkodik a forrásfájlokon. Ezen előfeldolgozónak saját nyelvtana van és saját utasításai. Az előfeldolgozó utasításai kettőskereszttel kezdődnek.

Az előfeldolgozó egyszerű mintabehelyettesítést végez. Használható konstansok és makró függvények definiálására, valamint header fájlok beimportálására.

Konstansok definiálása

Konstansok definiálása a #define utasítással történik, az alábbi formában:

```
#define AZONOSITO érték
```

Az azonosító nevet nagybetűvel szokás megadni, hogy elkülönüljön a név a változók neveitől. Ez nem kötelező, de erősen ajánlott.

Példák:

```
#define STRING1                 "Szöveg\n"
#define STRING2                 "Egy sorba kell férnie\n"
#define KONSTANS                1 + 2 + 3 + 4
```

Feltételes fordítás

A #if, #ifdef, #ifndef, #else, #elif és #endif utasításokkal ellenőrizhető egy azonosító definiáltsága vagy nem definiáltsága. Ezzel megoldható az, hogy bizonyos utasítások csak akkor kerüljenek bele egy programba, ha az azonosítót definiáltuk vagy nem. Ezt nevezik feltételes fordításnak.

Példa:

```
#define AZONOSITO
#ifdef AZONOSITIO
     //ez a blokk belefordítódik a programba, mert az
     //AZONOSITO definiálva lett.
#endif

#ifndef BELA
     //ez az utasítás is belefordul, mivel BELA nincs definiálva.
#endif
```

Makró definiálása

A makrók nem valódi C függvények, mivel a hívás helyére az előfeldolgozó egyszerű mintaillesztéssel másolja be a makró függvény utasításait. Így a valóságban nem történik függvényhívás, gyorsabb lesz a program, ám a kimeneti kód mérete növekszik. (valamit valamiért)

Makrók definiálásakor ügyelni kell a paraméterek zárójelezésére a feldolgozó szöveg illesztéses mivolta miatt. Ez gyakorlatban azt jelenti, hogy az alábbi példarészlet által definiált két makró függvény nem ugyanazt az eredményt fogja visszaadni:

```
#define MAX(a,b)                (a < b ? b : a)

#define MAX(a,b)                ((a < b) ? (b) : (a))
```

A két definíció közül az utóbbi fog helyesen működni.

További példa:

```
#define RADTODEG(x)                 ((x) * 57.29578)
#define ABS(x)                      (((x) < 0) ? -(x) : (x))
```

Fejlécfájlok és importálásuk

C esetén lehetőségünk van fejlécfájlokat definiálni. A fejlécfájl akkor jön jól, ha olyan kódrészletet szeretnénk megosztani a több forráskódfájlból álló projektünkben, amelyre mindegyik állomány hivatkozik. Lehetnek ezek például struktúrák, felsorolások, és még folytatható lenne a lista.

A fejléc fájlok .h kiterjesztéssel végződnek, és nem .c kiterjesztéssel. Továbbá a fájlnak az alábbi elrendezést kell követnie:

```
#ifndef HeaderFileNeve_H
#define HeaderFileNeve_H

    //Header tartalma

#endif
```

Ez a szerkezet biztosítja, hogy a header állományban definiált függvények és adattípusok csak egyetlenegyszer legyenek definiálva a fordítás során. A szerkezet elhagyása fordítási hibát eredményez.

A header fájlok később bármelyik C fájl elején a #include parancs segítségével csatolhatóak. Ennek a szintaxisa a következő:

```
#include "header utvonal.h"
```

Azon header fájlok, amelyek a fordító alapértelmezett header könyvtárában helyezkednek el, azok <> jelek között vannak megadva. Pl:

```
#include <math.h>
```

Fordító függő szolgáltatások

Az egyes fordítóprogramok előfeldolgozói további egyedi utasításokat is alkalmazhatnak. Ezeket a #pragma utasítással kell megadni. Ezen utasítások fordító specifikusságából következik, hogy az így kapott kód nem lesz hordozható egyes fordító programok között. Ez azonban kijátszható úgy, hogy a fordításhoz feltételeket definiálunk.

GCC esetén a támogatott pragma utasítások a https://gcc.gnu.org/onlinedocs/gcc-4.9.0/gcc/Pragmas.html#Pragmas címen lelhetőek fel. Visual C++ esetén a támogatott pragma utasítások listája: http://msdn.microsoft.com/en-us/library/d9x1s805.aspx

4. C++ alapismeretek

C++ használatára MikroC esetén nem lesz lehetőségünk, mivel ez a fordító csak C-t támogat, ami a legtöbb mikrovezérlős programunkhoz elegendő lesz. Azonban ha Arduino esetén osztálykönyvtárat szeretnénk létrehozni, vagy egy bonyolultabb programot, akkor ismernünk kell a C++ rejtelmeit.

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 az Arduino könyvtárt.

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 ellentettje. 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***

3. 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***

4. 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***

5. 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 öröklésheż:

```
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 Masik();
	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.

SOC eszközökhöz kapcsolódó c++ ismeretek

Képernyő és billentyűzet kezelése

C++ esetén a képernyő és a billentyűzet kezelése is objektum orientáltan van megvalósítva. A képernyőt a cout objektum reprezentálja, a beviteli billentyűzet ezközt pedig a cin objektum. Ezen kezelési mód fényévekkel jobb, mint a hagyományos c függvényekkel megvalósított rendszere véleményem szerint. Ezen osztályokat leginkább SOC eszközök esetén fogjuk használni, de előkerülhetnek nagyon egzotikus esetben mikrovezérlők kapcsán is.

A szükséges függvények az iostream.h fájlban vannak implementálva, de mivel ez a fájl része a szabványos c++ osztálykönyvtárnak használatba vételekor nem kell kitenni a végére a .h fájlt. Cserébe használatba kell vennünk a szabványos névteret.

A Névtér C++ esetén egyfajta logikai csoportosítást valósít meg. Egy névtér tartalmazhat objektumokat, struktúrákat, függvényeket, változókat. A névtérben megvalósított elemeket csak a névtér nevére hivatkozás után érhetjük el a :: operátorral. példaképpen ha az pelda névtér definiál egy proba nevezetű függvényt, akkor azt a következőképpen érhetjük el a programunkból:

```
pelda::proba();
```

Mivel ez a használati módszer komplikált és nem kényelmes nagy programok esetén bevezették a using direktívát, amivel egy teljes névtér használata megoldható. A C++ nem csak egy nyelv. Tartozik hozzá egy osztálykönyvtár is, amit szabványosítottak. Innen a szabványos névtér neve. Az alábbi példa a Hello World mintaprogramot mutatja be.

```
#include <iostream>

using namespace std;

int main()
{
     cout ”Hello World” << endl;
     return 0;
}
```

A cout (képernyő) objektumra kiírni a << operátorral tudunk, amely jelen esetben nem bitforgatást valósít meg. A jelentése az objektum hatáskörzetében át lett definiálva adat kiküldésre. A beleirányított adatokat automatikusan formázza, sőt a kiírások láncolhatóak is.

```
cout << ”ez egy ” << ”pelda” << ”a lancolasra” << endl;
```

az endl egy úgynevezett manipulátor. Ezen manipulátor használatához nem kell extra fejlécet importálnunk. További manipulátorokat az iomanip fejléc állomány tartalmaz, melyek a következőek:

Manipulátor	*Hatása*
`dec`	*Egés számok esetén a kiírandó számot átváltja a megadott számrendszerbe és úgy írja ki. Hatása megmarad egészen addig, amíg felül nem bíráljuk.*
`hex`	
`oct`	
`ends`	*Szöveg végét jelző \0 karakter*
`flush`	*Puffer ürítés*
`showbase`	*A számrendszert jelző karaktereket is kiírja.*
`ws`	*Megjelenítés előtt törli a szóköz és tab karaktereket.*
`setw(int w)`	*Mezőszélesség megadása. Hatása csak egy kiírásra vonatkozik.*
`setprecision(int n)`	*Tizedesjegyek számának meghatározása*
`setfill(char c)`	*Kitöltőkarakter meghatározása*

6. Táblázat: C++ kimeneti manipulátorok

A manipulátorokon kívül megadhatunk még kiíratási módokat jelzőbitek beállításával. A jelzőbiteket a cout objektum setf() függvényén keresztül tudjuk beállítani. A jelzőbitek az alábbiak lehetnek:

Jelzőbit	*Hatása*
`ios::fixed`	*A számok kiíratása fixpontosan történik*
`ios::scientific`	*A számok kiíratása normál alakban történik*
`ios::showpoint`	*A tizedespont utáni vezető nullák is megjelennek.*
`ios::showpos`	*Pozitív számok előtt megjeleníti a + jelet.*
`ios::right`	*A megadott mezőszélességen belül jobbra igazítva jeleníti meg a szöveget.*
`ios::left`	*A megadott mezőszélességen belül balra igazítva jeleníti meg a szöveget.*

7. Táblázat: Kiíratás közben használható jelzőbitek

Adatok beolvasása a cin objektum használatával lehetséges az alábbi formában:

```
cin >> valtozó neve;
```

A beolvasás esetén szükséges típus konverziót a cin objektum automatikusan elvégzi. A >> operátor jelentése itt is átdefiniált.

A bekérésnél további függvényeket is használhatunk, melyeket a cin osztály bisztosít:

```
cin.get();
```

Egy karaktert kér be, majd ad vissza.

```
cin.ignore();
```

Egy karaktert kér be, amit a bekérés után eldob.

```
cin.getline(*char sor, int hossz);
```

Egy sort olvas be az első paraméter által megjelölt karaktertömbbe. A szöveg maximális hosszát a második paraméter limitálja. A szöveget az olvasás végén lezárja a szöveg vége jellel. A sor a megadott hosszig fog tartani, ha hamarább nem talál a szövegben új sor jelet. (\n karakter) Használata javasolt olyan esetben, ha több szavat szeretnénk beolvasni, mivel a beolvasó operátor (>>) csak az első szóközig olvas.

```
cin.read(char *tomb, int hossz);
```

Hasonló a getline függvényhez, azonban a beolvasott szöveg végére nem illeszti oda a szöveg vége jelet.

Fájlkezelés

A fájlkezelés szintén objektum orientáltan történik. A fájlkezeléshez szükséges osztályok a fstream header fájlban vannak megvalósítva. Alapvetően három darab osztály van ebben a fejlécben definiálva. Az általános fstream, ami reprezentálhat egy bemeneti vagy kimeneti fájlt megnyitási módtól függően. Az ofstream osztály egy kimeneti fájlt reprezentál, amibe adatokat írhatunk. Az istream osztály pedig egy bemeneti fájlt reprezentál, amiből csak olvashatunk.

Az osztályok két konstruktorral rendelkeznek. Egy paraméter nélkülivel és egy paraméteressel. A paraméteres konstruktor használata megegyezik az open tagfüggvény használatával, ami a fájlt megnyitja használatra.

Fájlból olvasni a >> operátor segítségével tudunk, míg fájlba írni a << operátor segítségével. Megnyitás esetén különböző opciók állnak rendelkezésünkre, amellyel a fájl kezelésének módját befolyásolhatjuk. A támogatott megnyitási módok a következőek:

Megnyitási mód	***Hatása***
`ios:in`	*Megnyitás csak olvasásra. Az ifsrtream esetén alapértelmezett, itt külön nem kell megadni.*
`ios::out`	*Megnyitás csak írásra. Az ostream esetén alapértelmezett, itt külön nem kell megadni.*
`ios::app`	*Megnyitás hozzáfűzésre, az új tartlmat a fájl végére fogja írni.*
`ios::nocreate`	*Ha a megadott fájl nem létezik, akkor nem hozza létre, helyette hibát jelez.*
`ios::trunc`	*Ha a fájl már létezik, akkor felülírja a tartalmát.*
`ios::noreplace`	*Ha a fájl már létezik, akkor nem engedi meg, hogy létező fájlt írjunk felül.*
`ios::binary`	*A fájl megnyitása bináris módban*

8. Táblázat: Megnyitási módok

Az egyes megnyitási módok a bitenkénti vagy operátorral fűzhetőek össze. Az alábbi péda ezt mutatja be:

```
istream be;
be.open(”teszt.txt”, ios::noreplace | ios::app);
```

Tagfüggvények

A kezelő folyamok számos tagfüggvénnyel rendelkeznek, amelyek az állományok kezelését megkönnyítik. Ezek közül talán a legfontosabbak a hibák kezelésére és észlelésére szolgálóak.

```
good();
```

A függvény hívása előtti fájl művelet sikerességét ellenőrzi. Amennyiben a művelet sikeres volt, akkor az ezt követő művelet sikerességére is van esély. Ha a visszatérési értéke hamis, akkor az ezt követő művelet biztos nem lesz sikeres. Ha egy olyan adatfolyamon próbálunk műveletet végezni, amelyre a függvény hamis értéket ad vissza, akkor nem fogunk hibaüzenet kapni, de a művelet ettől függetlenül sikertelen lesz.

```
eof();
```

Ellenőrzi, hogy az olvasandó vagy írandó fájl végét elértük e.

```
fail();
```

Hasonló a good függvényhez, a művelet sikertelenségét ellenőrzi. Igaz visszatérési értéke lesz, ha a művelet sikertelen volt.

```
bad();
```

Az adatfolyam sérültségét ellenőrzi. A fail és bad állapot között nagyon kicsi a különbség. Ha a fail állapot fentáll, de a bad nem, akkor az adatfolyam érvénytelen lett, de nem történt adatvesztés. Amennyiben a bad állapot is fellép, akkor adatvesztés történt.

```
Clear();
```

Hiba állapot törlése, megszüntetése.

```
put(char c);
```

Egy karaktert ír az adatfolyamba.

```
get(char c);
```

Egy karaktert olvas az adatfolyamból. A változót referenciaként kell neki átadni.

```
getline(char *tomb, int hossz);
```

Egy sort olvas be az első paraméter által megjelölt karaktertömbbe. A szöveg maximális hosszát a második paraméter limitálja. A szöveget az olvasás végén lezárja a szöveg vége jellel. A sor a megadott hosszig fog tartani, ha hamarább nem talál a szövegben új sor jelet. (\n karakter) Használata javasolt olyan esetben, ha több szavat szeretnénk beolvasni, mivel a beolvasó operátor (>>) csak az első szóközig olvas.

```
seekg(int pos);
seekg(int pos, int seek_dir);
```

Az állomány megadott számú byte-jára pozicionál olvasáskor. Egyparaméteres változatában a pozíciót a fájl elejétől számolja. Kétparaméteres változatában a második paraméter azt határozza meg, hogy honnan pozicionáljon. Itt három konstans közül választhatunk. Ezek:

- *ios::beg*

  A fájl elejétől számolja a pozicionálást
- *ios::cut*

  Jelenlegi pozíciótól fogja számítani a pozicionálást.
- *ios::end*

  A fájl végétől fogja számítani a pozicionálást.

```
long int tellg();
```

Aktuális olvasási pozíciót adja vissza.

```
seekp(int pos);
seekp(int pos, int seek_dir);
```

```
long int tellp();
```

Ezen függvények használata megegyezik a g betűre végződő társaikkal azzal a különbséggel, hogy ezen függvények nem az olvasási pozíciót módosítják és adják vissza, hanem az írási pozíciót módosítják és adják vissza.

```
read(char *buffer, int length);
```

Az első paraméter által meghatározott bufferba olvas be a második paraméter által meghatározott mennyiségű adatot. Bináris fájlkezelés esetén is használható típus konverzióval.

```
write(char *buffer, int length);
```

Az első paraméter által meghatározott bufferből ír a fájlba a második paraméter által meghatározott mennyiségű adatot. Bináris fájlkezelés esetén is használható típus konverzióval.

Példák

Szövegfájl beolvasása, majd képernyőre írása memória hatékonyan:

```
#inclde <iostream>
#include <fstream>

using namespace std;

int main()
{
     istream fajl(”szoveg.txt”);
     if (fajl.fail())
     {
          cout << ”A fajl megynitasa nem sikerult” << endl;
          return 1;
     }

     char sor[255];

     while (!fajl.eof())
     {
          fajl.getline(sor, 255);
          cout << sor;
     }

     return 0;
}
```

Struktúra fájlba írása:

```
#inclde <iostream>
#include <fstream>
```

```
using namespace std;

typedef struct adat
{
     int szam1;
     int szam2;
};

int main()
{
     ofstream fajl(”kimenet.dat”, ios::binary | ios::trunc);
     adat teszt;
     teszt.szam1 = 55;
     teszt.sazm2 = 33;

     if (fajl.fail())
     {
          cout << ”A fajl letrehozasa nem sikerult” << endl;
          return 1;
     }

     fajl.write((char*)teszt, sizeof(adat));
     fajl.close();

     return 0;
}
```

Sablonok

A C++ sablonjai lehetővé teszik azt, hogy a programunk logikai szerkezetét jelentősen egyszerűsítsük. C esetén, ha kellett egy függvény, ami egy egész számtömb átlagát adja meg, akkor írtunk rá egy függvényt. Ha ugyan ez a függvény kellett lebegőpontos számokhoz is, akkor a függvényt lemásoltuk és a paramétereket átírtuk más típusra. Ez felesleges munkát jelent, valamint indokolatlanul növeli a program méretét, hiszen az algoritmus ugyan az mindkét esetben.

Ez a probléma kiküszöbölhető C++ esetén sablonok használatával. A sablonok használata lehetővé teszi az általánosított programozást, ahol a hangsúly nem a típusokon van, hanem a megvalósításon.

Az alábbi példa egy egyszerű sablon készítését mutatja be.

```
#include <iostream>

using namespace std;

template <typename T> T osszead(T param1, T param2)
{
	return param1 + param2;
}

int main()
{
	cout << "3.14 + 2.12 = " << osszead(3.14, 2.12) << endl;
	cout << "3 + 2 = " << osszead(3, 2) << endl;
}
```

A template kulcsszó jelöli a fordító számára, hogy egy általánosított függvényről lesz szó. A typename kulcsszó a típus leírására szolgáló azonosító bevezetését szolgálja, amelyet a példában egyszerűen csak T-vel jelöltem. Íratlan szabály, hogy a sablonokban a típusok azonosítóit csupa nagybetűvel jelöljük. A sablonokban a típusok lehetnek referenciák és mutatók is, ugyan úgy, mint minden más típus esetén.

A main függvényben a sablon meghívásakor illeszti be a megfelelő típust a sablonba a fordító program.

Több, eltérő típus is alkalmazható egy sablon létrehozásakor, ekkor minden egyes típus név előtt szerepeltetni kell a typename kulcsozót. Az alábbi példa az összead függvényt mutatja be két különböző típusra definiálva.

```
template <typename T, typename T1> T osszead(T param1, T1 param2)
{
	return param1 + param2;
}
```

Függvény sablonokon kívül létrehoztatunk osztály sablonokat is. Osztálysablonok esetén a kezelt típusokat class vagy typename jelzővel is bevezethetjük. Az alábbi példa egy egyszerű verem adatszerkezet létrehozását mutatja be sablonok segítésével.

```
template <class T, int Meret>
class Verem
{
	public:
		Verem();
		~Verem();
		void Push(T& elem);
		void Pop(T& elem);
		bool WasError();
		bool isEmpty();
	private:
		T elems[Meret];
		int iTop;
```

```
        bool hiba;
};

template <class T, int Meret>
Verem<T, Meret>::Verem()
{
     iTop = 0;
     hiba = false;
}

template <class T, int Meret>
Verem<T, Meret>::~Verem()
{
}

template <class T, int Meret>
void Verem< T, Meret >::push(T& elem)
{
     hiba = (iTop == Meret);
     if (!hiba) elems[iTop++] = elem;
}

template <class T, int Meret>
void Verem<T, Meret>::pop(T& elem)
{
     bErrorOccd = (iTop == 0);
     if (!hiba) elem = elems[--iTop];
}

template <class T, int Meret>
bool Verem<T, Meret>::wasError()
{
     return hiba;
}

template <class T, int Meret>
bool Verem<T, Meret>::isEmpty()
{
     return (iTop==0);
}
```

Amint a példában látható osztálysablonok létrehozásakor a sablon utasítással ki kell egészíteni az osztály definíciót, amit aztán a tagfüggvények implementálásakor is ki kell tenni. A sablonok tartalmazhatnak fixált típusokat is, mint a fenti példa.

A tagfüggvények csak fix vagy olyan típusokat adhatnak visszatérési értékként, amelyek szerepeltetve vannak az osztály definícióban is. Továbbá a sablon osztályok példányosításakor meg kell adni a típusokat.

Az alábbi példa a fentebb definiált osztály használatbavételét mutatja be.

```
int main()
{
     Verem<float, 3) verem;
     verem.Push(3.14);
     verem.Push(22.14);

     float val;
     verem.Pop(&val);
```

```
    cout << val << endl;
    verem.Pop(&val);
    cout << val << endl;
}
```

A typename és class szavak azonos jelentéssel bírnak C++ sablonok esetén, működésükben nincs különbség.

Az ok, amiért két kulcsszó is van ugyan arra célra történelmi eredetű. Eredetileg a C++ nyelv megalkotásakor lustaságból, hogy ne kelljen egy újabb kulcsszót bevezetni a készítő class szót definiálta felül a sablonok kontextusában.

Azonban szabványosításkor a szabványosító csoport úgy gondolta, hogy megtévesztő lehet a programozók számára az átdefiniált jelentés, így bevezették a typename kulcsozót, azonban a class használhatóságát kompatibilitási okokból meghagyták.

Szabvány szerint tehát a typename használata lenne helyes. Azonban én programozás közben függvény sablonok esetén a typename használatát kedvelem, a class használatát pedig sablonok esetén. Egy másik elterjedt kódolási stílus szerint a class használata olyan sablonok esetén célszerű, ahol a típus egy osztály vagy struktúra lesz.

C++ osztálykönyvtár

A C++ nyelv két részre bontható. Első fontos része maga a nyelv és szolgáltatásai, másik része pedig a nyelvhez tartozó osztálykönyvtár.

C++ esetén az osztálykönyvtárban alapvető adattárolási szerkezetek, objektum orientált ki és bemenet kezelés, valamint általános algoritmusok is helyet kapnak. Maga a nyelv és az osztálykönyvtár is számos változáson esett át az elmúlt években. Jelenleg, a legfrissebb még kidolgozási fázisban lévő C++ szabvány a C++14, ami 2014-ben kerül elfogadásra remélhetőleg.

A szabvány jelenleg stabilnak mondott változata a C++11, amit 2011-ben fogadtak el. A legtöbb fordító program és C++ osztálykönyvtár implementáció ezt a szabványt célozza meg támogatásban, azonban fordító és osztály könyvtár függő, hogy mi is támogatott a szabványból.

Ezen okból általános osztálykönyvtár leírást nem lehet készíteni, pontosabban lehetséges, de az nem lenne naprakész. Továbbá a több száz függvény leírása mondhatni egy külön könyvet igényelne.

A http://en.cppreference.com/w/ oldalon megtalálható a C++11 szabvány osztálykönyvtárának leírása, valamint a C függvénykönyvtárának leírása is, mivel C++ esetén alkalmaztatóak a C függvényei is.

Ezek támogatottsága is fordító és verzió függő, így mindkét nyelv esetén a használt fordító dokumentációjában érdemes tájékozódni a támogatott funkciókról és függvényekről. Az alábbi táblázat a könyvben használt vezérlők C/C++ könyvtárainak dokumentációjának beszerzési helyét tartalmazza.

Platform/eszköz	**C/C++ könyvtár dokumentációja**
Arduino	http://www.nongnu.org/avr-libc/
MikroC	http://www.mikroe.com/mikroc/pic/libraries/
GCC	http://www.gnu.org/software/libc/manual/

9. Táblázat: C/C++ könyvtár dokumentáció fordítókra bontva

Jelen pillanatban az egyetlen C/C++ fordító amely teljes C++11 támogatással rendelkezik, az a clang.

Argumentumok fogadása

Argumentumok fogadásával a programunk működését befolyásolhatjuk, segítségükkel a programjainkat univerzálisabbá tehetjük. Az argumentumok parancssori kapcsolók, beállítások, amelyeket a programunknak az indításakor adhatunk meg.

Ezek feldolgozása és kezelése igen egyszerű. A main függvény definícióját kell módosítanunk, hogy alkalmassá tegyük argumentumok fogadására. A módosítás abból áll, hogy a fő függvényt két paraméterrel kell ellátni a következő módon:

```
int main(int count, char *args[]);
```

Az első paraméter a függvény hívásakor (a program indulása után) a kapott argumentumok számát fogja tartalmazni, míg a második paraméter a ténylegesen kapott argumentumok lekérdezésére szolgál. A változóknak bármilyen nevet adhatunk a fejlécben, a típusuk a lényeges.

Minden esetben a 0. argumentum a futtatott program neve és/vagy elérési útvonala lesz operációs rendszertől függően. Ezért a paraméterek száma a ténylegesen kapottaknál mindig eggyel több lesz. Az alábbi példaprogram az argumentumok egyszerű feldolgozását mutatja be:

```
#include <iostream>

using namespace std;

int main(int count, char *args[])
{
	cout << "Kapott argumentumok szama: " << count << endl;
	for (int i=0; i<count; i++)
	{
		cout << i << ". Argumentum: " << args[i] << endl;
	}

	return 0;
}
```

Programok fordítása

A komolyabb programok ritkán állnak csak egy fájlból. Ennek az oka, hogy egy program működése könnyebben átlátható, ha részekre bontjuk és ezeket a részeket külön fájlban tároljuk. A külön fájlokból programot készíteni manuálisan parancssor segítségével is lehetséges, azonban nem éppen a legkényelmesebb.

A probléma megoldására alkották meg a make eszközt, amely kifejezetten több fájlokból álló programok fordítását könnyíti meg. Ehhez egy speciális fájlt kell létrehoznunk, amely a make program számára megmondja, hogy hogyan kell előállítani a programot. Ezen fájlt make fájlnak szokták nevezni. A fájlt abban a mappában kell létrehozni ahol a program forráskódja található. A fájl neve: Makefile

A make használata talán egy példa segítségével jobban érthető. Ezért tételezzük fel, hogy van egy programunk, amely a következő fájlokból áll:

```
main.c
forras1.c
header.h
header.c
```

A program lefordítását elvégző Makefile fájl tartalma a következő:

```
#Make konfigurációs fájl

program: main.o forras1.o header.o
	gcc -o program main.o forras1.o header.o

main.o:
	gcc -c main.c

forras1.o:
	gcc -c forras1.c

header.o
	gcc -o header.c

clean:
	rm -f *.o
```

A kettős kereszttel kezdődő sorok megjegyzések, a program figyelmen kívül hagyja ezen sorokat. A kettősponttal végződő szavak fordítási célokat jelentenek. Ezekre hivatkozhatunk a make parancs kiadása után. Például, ha a clean utasítást szeretnénk lefuttatni, ami törli a .o végződésű fájlokat a mappából, akkor ezt a következőképpen tudjuk megtenni:

```
make clean
```

Ha paraméterek nélkül futtatjuk a make parancsot a mappában, akkor az első fordítási célt fogja végrehajtani, ami jelen fájl esetében a program. Ez a cél hivatkozik további célokra, méghozzá a kettőspont után felsoroltakra. Így ezen cél fordításkor a hivatkozott fordítási célokat készíti el, majd ezek elkészülte után lát neki ténylegesen a cél előállításának.

A gcc C fájlok fordításához használható program, míg a g++ C++ fájlokhoz használható. Mindkét program esetén a -c kapcsoló fordítást jelent. Ez egy .o végződésű fájlt fog előállítani, ami konkrétan még nem futtatható, de már gépi utasításokat tartalmaz.

Ha a gcc/g++ programnak ilyen .o fájlokat adunk bemenetként, akkor az ezekből képes előállítani az operációs rendszerhez linkelt futtatható programot. A -o kapcsoló a kimeneti fájlnevet határozza meg.

Ha a programunk tartalmaz header fájlokat, amelyek függvények, osztályok definícióját tartalmazzák, de ezen

függvények/osztályok tényleges megvalósítása egy ugyan ilyen nevű c vagy cpp kiterjesztésű fájlban vannak, akkor csak a .c vagy .cpp kiterjesztésű fájl fordítására kell utasítást adnunk, mivel a definíciókat tartalmazó header fájlok minden esetben automatikusan a programmal együtt fordulnak le.

A Makefile tartalma makró parancsok segítségével tovább egyszerűsíthető, automatizálható. A fentebb említett makefile-al azonos kimenetet produkáló, de makrókkal ellátott fájl tartalma a következő:

```
#Make konfigurációs fájl

program: main.o forras1.o header.o
	gcc -o $@ @^

%.o: %c
	gcc -c $< -o $@

clean:
	rm -f *.o
```

A $@ makró a fordítási cél nevét jelenti, a @^ makró pedig az összes hivatkozott célt. Az egyes .o állományok előállítása ciklus segítségével történik. A %o kifejezés az összes előállítandó .o fájlt szimbolizálja, míg a %c az összes mappában található c fájlt. A $< makró az összes c fájl közül az aktuálisan feldolgozás alatt lévőt jelenti.

A makrók használatának előnye, hogy nem kell nagyon bonyolultan módosítani a make fájl tartalmát, elég egy új fájl esetén egy helyen függőségként megjelölni a fájlból generálandó .o fájt.

Több szálon futó alkalmazások készítése

A mai modern operációs rendszerek szolgáltatása a több szálon futó alkalmazások létrehozásának képessége. Több szál létrehozásával és futtatásával kihasználhatjuk az összes rendelkezésre álló processzort, illetve egzotikus alkalmazásokat tudunk létrehozni.

A C++ 11 szabvány beépítetten tartalmaz egy szálak kezelésére és létrehozására alkalmas könyvtárat, aminek hatalmas előnye, hogy operációs rendszer független. Azonban A C++ 11 támogatás jelenleg még nem igen készült el a GCC fordító programban, amit a legtöbb beágyazott eszköz használ.

Ez nem jelenti azt, hogy nincs lehetőségünk szálak kezelésére, csupán azt jelenti, hogy a kódunk nem lesz hordozható Windows és Linux rendszerek között.

A szálak használatbavételéhez két fejléc állományt kell importálnunk kezdésképpen. A unistd.h és pthread.h állományokat. A pthread.h állomány tartalmazza a szálak kezeléséhez szükséges eljárásokat, míg az unistd.h állomány vegyes unix funkciókat biztosít.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,→
void *(*start_routine) (void *), void *arg);
```

Létrehoz egy szálat. Első paramétere egy pthread_t típusú változó, ami a szál azonosítására szolgál, második paramétere a szál viselkedésének meghatározására szolgál, általában NULL értékű, ami azt jelenti, hogy az operációs rendszer az alap beállításokat használja.

*A harmadik paramétere azon függvény neve, amely a szálon futtatandó kódot tartalmazza. A függvénynek void * típusú paramétert kell átvennie és static void * visszatérési értékkel kell rendelkeznie.*

*A függvény utolsó paramétere pedig a szálnak átadandó argumentumok void * típusban.*

A szál sikeres létrehozása esetén a függvény 0 értéket ad vissza.

```
void pthread_exit(void *retval);
```

A hívó szálat kilépteti, megszünteti. A paramétere érték visszaadásra szolgál, amivel adatok adhatóak át egy másik szálnak. A függvény meghívása nem kötelező, mivel a szál függvény return utasítása is hasonló módon jár el.

```
int pthread_join(pthread_t thread, void **retval);
```

*Az első paramétere által meghatározott szál kilépését, végzését várja meg. A fő szálban szokás bejegyezni, lehetővé teszi azt, hogy a létrehozott szálak végezzenek, mielőtt a főprogram kilépne. Második paraméter egy void * pointer, ami a szál által visszaadott érték lekérdezésére szolgál. A függvény visszatérési értéke 0, ha a művelet közben nem történt hiba.*

```
int pthread_cancel(pthread_t thread);
```

A fő szálból lehetővé teszi a paramétere által meghatározott szál megszüntetését.

```
void sleep(int seconds);
```

A meghívó szál altatása a paraméter által meghatározott másodpercig.

Fordítási beállítások

A szálakat tartalmazó programjainkat a -lpthread opcióval kell fordítanunk. Ez az extra argumentum biztosítja a szálak könyvtár linkelését a programhoz.

Az alábbi mintaprogram a szálak egyszerű kezelését mutatja be:

```
#include <iostream>
#include <unistd.h>
#include <pthread.h>

using namespace std;

static void * szal1(void * param)
```

```
{
     for (int i=0; i<5; i++)
     {
          cout << "Elso szal" << endl;
          sleep(1);
     }
     return (void*)"szal1 vegzett";
}

static void * szal2(void * param)
{
     for (int i=0; i<5; i++)
     {
          cout << "Masidik szal" << endl;
          sleep(2);
     }
     return (void*)"szal2 vegzett";
}

int main()
{
     void *uzenet1, *uzenet2;
     pthread_t sz1, sz2;

     pthread_create(&sz1, NULL, szal1, (void*)NULL);
     pthread_create(&sz2, NULL, szal2, (void*)NULL);

     pthread_join(sz1, &uzenet1);
     cout << (char*)uzenet1 << endl;

     pthread_join(sz2, &uzenet2);
     cout << (char*)uzenet2 << endl;

     return 0;
}
```

Architektúra független típus rendszer

A C és C++ nyelvek típusrendszere a processzortól függ. Ez megnehezíti a programok hordozását különböző bitszámú processzorok között. Ezért a C és a C++ is rendelkezik egy architektúra független típus rendszerrel, hogy a programok hordozhatóak legyenek a különböző architektúrák között.

Ehhez c esetén a stdint.h fájlt kell használatba vennünk, míg C++ esetén a csdint header fájlt.

```
/*C esetén*/
#include <stdint.h>

//C++
#include <cstdint>
```

A header fájl az alábbi típusokat biztosítja:

Típus	***Leírás***
int8_t	*8 bites előjeles egész szám*
uint8_t	*8 bites előjel nélküli egész*
int16_t	*16 bites előjeles egész szám*
uint16_t	*16 bites előjel nélküli egész szám*
int32_t	*32 bites előjeles egész szám*
uint32_t	*32 bites előjel nélküli egész szám*
int64_t	*64 bites előjeles egész szám*
uint64_t	*64 bites előjel nélküli egész szám*

10. Táblázat: A header típusai

Ahhoz, hogy a kód hordozható legyen egész típusok esetén a headerben definiált típusokat kell alkalmazni a kódolás során.

Adatszerkezetek

Programjaink során az adatokat valamilyen struktúrában a memóriában kell tárolnunk. A legegyszerűbb adatstruktúra a tömb. Ennek kezeléséről a C és C++ ismeretek szekcióban is volt szó, mind statikus, mind dinamikus módon, azonban nem csak a tömb létezik adattárolási struktúraként.

Láncolt lista

A láncolt lista egy olyan adatszerkezet, amelyben lánc elem típusú struktúrákat tárolunk. Minden láncelem struktúra tartalmazza egy adat mezőt és egy mutatót a következő elemre. Ebben az esetben a lánc egy irányú, vagyis csak az elejétől bejárható. Ha két mutatót tartalmaz a láncelem (előző és következő elem), akkor kétirányú láncról beszélünk.

Ezen adatstruktúra előnye a tömbökkel szemben, hogy tetszőlegesen bővíthető bármelyik pontján, anélkül hogy, új tömböt kellene létrehozni és újrafoglalni az elemeket. Hátránya, hogy nem tesz elérhetővé véletlen elérést az adatszerkezet. Az elemeket csak a lánc szerint rögzített sorrendben lehet elérni.

Verem

A verem egy olyan adatstruktúra, amelyből mindig csak az utolsó behelyezett elemet tudjuk kivenni.

5. Python alapismeretek

A Python egy könnyen tanulható és olvasható programozási nyelv, amelyet készítője Guido van Rossum eredetileg azért alkotott meg, hogy könnyen elsajátíthatóak legyenek a programozás alapjai. Ezért a nyelv a programozási munka megkönnyítését helyezi előtérbe a sebességgel szemben.

Ebből kifolyólag mikrovezérlős környezetben nem is alkalmazott, azonban olyan SOC rendszerek, mint a RaspberryPi igen sok érdekességet lehet vele művelni.

A nyelv erősen objektum-orientált, dinamikus típuskezeléses, interpretált. A dinamikus típuskezelés azt jelenti, hogy programozás közben nem kell megadnunk a változók típusait és nem a programozónak kell közöttük konvertálgatni. Mindezt elvégzi helyettünk a feldolgozó. Az interpretált nyelvek olyan programozási nyelvek, amelyeket egy parancsértelmező dolgoz fel. Az ilyen nyelvek sajátossága, hogy nem kell a programokat külön lefordítani. Ez a filozófia általában a sebesség rovására megy, viszont cserébe az ilyen nyelvek általában számos kiegészítő modullal érkeznek, ami megkönnyíti a fejlesztést. Nincs ez másként a Python esetén sem.

A feldolgozó program elérhető a legfőbb operációs rendszerekre, így nem feltétlen kell Linux a fejlesztéshez. Jelenleg a nyelvnek két fő verziója létezik. A 2.x és a 3.x vonal. A 3.x vonal nem kompatibilis változásokat hozott a nyelvben, ezért kell őket megkülönböztetni. Annak ellenére, hogy a 3.x vonal egy jó ideje elérhető a mai napig is a legnépszerűbb változat a 2.x. Ennek oka az, hogy számos kiegészítő modul, ami nem képezi részét az alaptelepítésnek csak a 2.x vonal számára elérhető.

Éppen ezért a könyvben a Python 2.x vonaláról lesz szó, szintén csak olyan szinten, amennyire szükséges.

Számos könyv és elektronikus könyv érhető el a Python programozási nyelvről, illetve a hivatalos dokumentáció is elég jól összerakott. A dokumentáció a http://www.python.org/doc/ címen lelhető fel.

A könyv ezen fejezete feltételezi, hogy az olvasó elolvasta és megpróbálta megérteni a könyv C és C++ nyelvről szóló szakaszait.

Python típusok

Kategória	Típus neve	Leírás
Szám	bool	Logikai típus, értéke True vagy False lehet.
	int	Egész szám
	long	Hosszú egész szám
	float	Lebegőpontos szám
	complex	Komplex szám
Halmaz	set	Megváltoztatható halmaz
	frozenset	Rögzített halmaz
Sorozat	str	Karakterlánc, szöveg
	unicode	UTF kódolású karakterlánc, szöveg
	basestring	Szöveg típusok őstípusa
	list	Lista
	tuple	Sor
	xrange	Rögzített sorozat
Leképezés	dict	Szótár
Egyebek	type	Változó elem típusáról információt szolgáltató típus
	object	Minden objektum ős osztálya
	types.ModuleType	Modul
	types.NoneType	Üres, null típus

11. Táblázat: A Python programozási nyelv fő típusai

Szám típusok

Az alapvető szám típusok megvalósítása hasonlít a C/C++ esetén tárgyaltakhoz. A típusok hosszai itt 32 bites C nyelvnek felelnek meg. A float típus pedig a C nyelvben ismertetett float típusnak. A complex típus két lebegőpontos számot tartalmaz, amely alkalmas komplex számrendszerbeli számok leírására. Példák:

```
Egesz = 10
lebegopontos = 10.13
complex = complex(40, 10)
```

Üres típus

Olyan típus, amely egy üres objektumra utal. Objektumok kinullázására, megsemmisítésére használható. A None kulcsszóval érhető el.

Halmazok

A halmaz típus egyedi elemek rendezetlen gyűjteményét jelenti. Két fő típusa van. A megváltoztatható, amelyhez menet közben elemeket lehet adni és elvenni, valamint a nem változtatható, amely létrehozása után nem módosítható. Elemeit valamely rögzített típusból kell képezni, így nem tartalmazhatnak listákat, vagy leképezéseket elemként, viszont karakterláncokat és sorokat tartalmazhatnak. Létrehozási példák:

```
paros = set([2, 4, 6, 8, 10])
paratlan = set([1, 3, 5, 7, 9])
```

Sorozatok

A sorozat rendezetten tárolt elemek gyűjteménye, amelyeket nem negatív egész számokkal sorszámozhatunk,

elemei mindenféle típusból képezhetőek. Leggyakrabban használt altípusa a karakterlánc és a lista. Karéktérláncok esetén a " és ' jel egyaránt használható szövegek jelölésére. Annyi megkötés van, hogy ugyan olyan idézőjellel kell lezárni a karaktersorozatot, mint amilyennel elkezdtük.

```
Szoveg1 = "Ez egy szöveg"
Szoveg2 = 'Ez is helyes'
```

Ha az idézőjelet háromszor alkalmazzuk a szöveg elején és végén, akkor a szöveg több soros is lehet.

```
tobbsor = """Ez egy
több sorból álló
szöveg"""
```

Lista példák:

```
lista = [22, 33, 55, 44]
x = lista[2] #értéke 55 lesz, mivel a lista 2. indexű eleme 55
szoveg = "Hello"
y = szoveg[0] #értéke H, mivel a szöveg 0. indxexű eleme H
```

Sorok

A sor típus sokban hasonlít a sorozat típushoz. Fő különbség azonban, hogy a sorozatok menet közben megváltoztathatóak, míg a sorok nem. A listák általában azonos típusú elemeket tartalmaznak, míg a sorok eltérőeket. Továbbá a sorok egymásba tetszőlegesen láncolhatóak. Leginkább funkcionalitás programozási kiegészítések miatt részei a nyelvnek.

A sorok elemeit normál zárójelek között kell megadni.

```
Sor = (44.12, 55, "hello")
```

Leképezések

A leképezések kétféle objektumhalmazt tárolnak. Az egyik halmaz a kulcsobjektumok halmaza, a másik halmaz pedig az érték objektumok halmaza. A kulcsobjektumok halmaz elemei képezik az érték objektumok indexeit. A kulcsobjektumok halmaza rögzített típusú, míg az érték halmaz elemei szinte bármilyen típusból kikerülhetnek. Létrehozási példa:

```
szotar = { "cat" : "macska", "dog" : "kutya" }
#új elem felvételével létrehozás, ha meg létezik, akkor hozzáfűzés:
szotar["fox"] = "róka"
```

A nyelv kulcsszavai

and as assert break class continue def del elif else except False finally for from global if import in is lambda None nonlocal not or pass print raise return True try while with yield

A nyelv szintaxisa, vezérlési szerkezetek

A Python elég sok elemben hasonlít a C és C++ nyelvekre, azonban van egy lényeges különbség. Python esetén nincsenek blokk zárójelek. Funkciójukat a program megfelelő formázása váltja ki. A nem megfelelően formázott python program nem fog elindulni, tehát az ember rákényszerül arra, hogy szép, rendezett kódot adjon ki.

A megfelelő formázás alatt a sorbehúzások megfelelő alkalmazása értendő. Egy utasítás blokk sorbehúzásához használt szóközök száma változhat, azonban egy blokkon belül az utasításokat azonos sorbehúzással kell jelölni. A nyelv hivatalos ajánlása 4 szóköz behúzásonként.

Megjegyzések létrehozhatóak a # karakterrel. A # karakter utáni részt a feldolgozó megjegyzésként értelmezi. Több soros megjegyzések létrehozása a triplázott idézőjelekkel lehetséges.

```
#ez egysoros megjegyzés

" " "
Ez meg egy több sororos
Annyiban különbözik a szöveg létrehozástól, hogy
az idézőjelek között szóköz van, illetve a szöveg nincs értékül
adva semminek.
" " "
```

Mivel nincs sorvég jel, ami az utasítások végét jelenti ezért több soros utasításokat a sor végén el kell látni egy \ jellel, ami azt jelenti, hogy az utasítás folytatódni fog.

```
ossze = 1.1111153184461 + \
        22.55314 + \
        3.14
```

Feltételes utasítások

Feltételes utasítások szintén if utasítással hozhatóak létre, amely szintaxisa a következő:

```
if kifejezés:
      utasítások
```

A kifejezés hamis voltához szintén kapcsolható elágazás az else utasítással:

```
if kifejezes:
      utasítások
else:
      utasítások
```

Több if-else utasítás szintén összefűzhező, azonban az else if szerkezet helyett egy speciális kulcszót kell alkalmazni. Ez a kulcsszó az elif.

```
if kifejezes:
      utasítások
elif kifejezes:
      utasítások
else:
      utasítások
```

A Python nyelv nem rendelkezik switch-case szerkezettel, helyette if-elif-else szerkezetek alkalmazhatóak.

Elől tesztelő ciklus

Az elől tesztelő while ciklus működése megegyezik a C nyelv esetén tárgyalt while utasítás működésével, vagyis

ameddig a feltételben megadott kifejezés értéke igaz, addig a ciklusmag végre fog hajtódni. Szintaxisa:

```
while feltétel:
      utasítások
```

Példa:

```
x = 0;
while x < 10:
      x += 1
```

for ciklus

A ptyhon for ciklusa a c nyelv esetén ismertetettől jócskán eltér, mert a ciklust sorozatokon végighaladáshoz módosították. Szintaxisa:

```
for elem in sorozat:
      utasítások
```

A ciklusmag annyiszor fut le, amennyi elemet tartalmaz a sorozat. Az elem változó minden egyes ciklus futás alkalmával felveszi a sorozat egy elemét, egész addig, amíg a végére nem ér. Példa:

```
lista = ["ez", "egy", "lista"]
for elem in lista:
      print elem
```

Abban az esetben, ha egy olyan for ciklust szeretnénk létrehozni, amelyben egy változó értékeit szeretnénk felhasználni egy adott értéktől egy másik értékig, akkor a range függvénnyel létrehozhatunk egy ideiglenes listát.

```
for i in range(0, 10):
      print i
```

Ha a range függvényt csak egy paraméterrel hívjuk meg, akkor a számsorozat kezdő értéke 0 lesz, míg a végértéke a paraméter által megadott szám. Ha pedig három argumentummal hívjuk meg, akkor a harmadik argumentum az elemek közötti különbséget határozza meg. Az alábbi kód egy olyan for ciklust csinál, amely 0-tól 100-ig kiírja a páros számokat:

```
for i in range(0, 100, 2):
      print i
```

Amennyiben egy lista konkrét értékei mellett a ciklusban szükséges tudni azok indexét is, akkor az enumerate függvénnyel létrehozható egy ideiglenes szótár, amelyben az indexek a lista elemek indexi lesznek és a hozzá tartozó értékek pedig a lista értékei.

```
lista = ["ez", "egy", "lista"]
for index in enumerate(lista):
      print index
      print lista[index]
```

Függvények, objektumok definiálása

Függvények a def kulcsozóval hozhatóak létre. A def kulcsszó után meg kell adni a függvény nevét, zárójelben pedig azon változóneveket, amelyeket a függvény átvesz. A függvény, ha akar értéket a return utasítással tud visszaadni.

```
def függvény(paraméter1, paraméterN):
      return érték
```

A paramétereknek adhatunk a függvény definíciójakor kezdő értéket. Amennyiben a függvény egyik paramétere rendelkezik kezdő értékkel, akkor az a paraméter opcionálissá válik. Ha nem adjuk meg az értékét, akkor a kezdő értéke fog érvényesülni. Az ilyen paramétereknek minden esetben a paraméter lista végégén kell elhelyezkedünk, ha a függvény rendelkezik nem opcionális paraméterekkel is. Ennek szemléltetésére egy példa:

```
def fv(x, bool kiir = false):
      if kiir == true:
            print x*x*x
      else:
            return x*x*x
```

A függvény opcionális paramétere a kiir. Ha ez igaz, akkor a függvény kiírja képernyőre a művelet eredményét, azonban ha hamis, akkor értékként adja vissza. Egy függvénydefinícióval így kétféleképpen i használható függvényt alkottunk.

Osztályok, objektumok

A Python nyelv erősen objektum orientált, mivel minden típus a nyelvben egy objektum. Minden objektumnak van egy azonosítója, típusa és értéke. Az azonosító az a memóriacím, amelyen az objektum helyezkedik el, a típus írja le a memória reprezentációját, míg az érték az objektumban tárolt adat.

Az objektumok azonosítója az id függvénnyel kérdezhető le, míg a típus a type függvénnyel.

```
Szoveg = "Hello"
print id(Szoveg)
print type(Szoveg)
```

Saját osztályokat definiálni és példányosítani is igen egyszerű. Egy osztály az alábbi szintaxis segítségével definiálható:

```
class osztaly(ososztaly):
```

Az ősosztály neve elhagyható, amennyiben nem öröklünk. Bár ebben az esetben is történik egy implicit öröklés, méghozzá az object osztályból, amely python esetén minden osztály ősosztálya. Az öröklődés a C++ publikus öröklődési modelljéhez hasonló, vagyis minden ami nyilvános az osztályban öröklődik. Igazából privát metódusok és adattagok létrehozására nincs lehetőség, azonban az ilyennek szánt adattagok elérése megnehezíthető azáltal, hogy a privát adattagokat és függvényeket kettő aláhúzás karakterrel látjuk el a nevük elején. De ez nem azonos a C++ esetén tárgyalt privát adattaggal, mivel ekkor is van lehetőség az osztályon kívül hozzáférésre az adattaghoz/függvényhez, csupán nehezebben megvalósítható.

Az örökölt függvények bármelyike felülírható az öröklötetett osztályból, csupán ugyan olyan névvel kell ellátni a függvényt, mint az ősosztályban. Ekkor nem az ősosztály függvénye fog végrehajtódni, hanem az új osztály függvénye.

Az osztályok esetén nincs szükség destruktor megírására, mivel a nyelv memória menedzselt. Ez azt jelenti, hogy egy szemét gyűjtő a None típusú változókat előbb-utóbb eltávolítja a memóriából. Az eltávolítás időpontja előre nem jósolható meg, mivel leginkább a számítógépben elérhető memória mennyiségétől függ a dolog. Ha ez kevés, akkor viszonylag sűrűn fog megtörténni a szemét takarítás, viszont ha van bőven, akkor nem biztos, hogy sűrűn fog megtörténni.

Az osztály természetesen rendelkezhet konstruktorral, azonban itt nem az osztály nevével azonos típus nélküli függvényként kell definiálni. Minden osztály konstruktora ugyan azt a nevet használja. Az erre fenntartott függvény

definíciója:

```
def __init__(self, paraméterek):
```

A konstruktor függvény első paramétere minden esetben egy self változó, amely az objektumhoz kötésért felelős. Ezen keresztül érhetőek el az objektum adattagjai és tagfüggvényei. Ha a konstruktor külső paramétereket nem vesz át, akkor is a self változónak szerepelnie kell az argumentumok listájában. Amennyiben nem szerepel ott, akkor a függvény nem lesz része az osztály definíciónak. Természetesen ha nem hozzuk létre a konstruktor függvényt, akkor az értelmező paraméter nélküli konstruktorral egészíti ki az osztály definícióját.

A tagfüggvények első paraméterének is egy self változónak kell lennie, máskülönben szintén nem tudnak hozzáférni az objektum adattagjaihoz és nem is kötődnek majd az objektumhoz.

Az alábbi egyszerű példa az osztályok létrehozását és az objektumok használatát mutatja be.

```
class pelda:
      def __init(self)__:
            self.uzenet = "Objektum orientalt"

      def print(self):
            print self.uzenet

objektum = pelda()
objektum.print()
```

A függvényeknek mind objektumokon belül, mind kívül paramétereket átadhatunk sorrendi felsorolással is és név szerinti megadással. A sorrendi átadással C esetén már találkoztunk. Ebben a módszerben a függvény definíciójának megfelelő sorrendben adjuk át az adatokat a függvények. A különböző módszerek szemléltetéséhez definiáljunk egy függvényt, ami két paramétert vár el. Egy nevet és egy életkort. Ennek a definíciója:

```
def fuggveny(nev, eletkor):
      print nev + "ma ennyi éves: "+eletkor
```

A sorrendi átadással a függvényünk így használható:

```
fuggveny("Teszt Elek", 18)
```

Lehetőség van a függvény argumentumok név szerinti átadására. Ezen módszer előnye, hogy a változók keverhetőek tetszőleges sorrendben az átadáskor, a név alapján be fogja tudni azonosítani az argumentumokat. A példaként definiált függvényünk így hívható meg ilyen módon:

```
fuggveny(eletkor=33, nev="Gipsz Elek")
```

*Argumentumok akár szótárként is átadhatóak. Ekkor a szótár kulcsainak meg kell egyeznie a függvény által várt paraméterek nevével. Továbbá a függvény hívásakor a szótár elé ** karakterek írandóak.*

```
teszt = { 'nev' : 'Példa Kálmán', 'eletkor' : 17 }
fuggveny(**teszt)
```

*A függvény definícióban szereplő paraméterek felfoghatóak sor típusként is (mivel belsőleg azok is), ezért lehetőség van a függvények számára az argumentumokat átadni sorként is. Ehhez a használandó sor elé * karakter írandó.*

```
sor = ("Példa Lajos", 24)
fuggveny(*sor)
```

Lehetőségünk van Python esetén névtelen függvények létrehozására is. A névtelen függvényekkel megoldható, hogy egy függvényt valózóként definiáljunk, adjunk át. Ez erősen funkcionális programozási képessége a nyelvnek. A névtelen függvényeket szokás lambda kifejezéseknek nevezni, innen jön a használandó lambda kulcsszó is, ennek a szintaxisa:

```
lambda paraméterek : kifejezés
```

Az alábbi példa a Lambda kifejezés használatát mutatja be:

```
kisebb = lambda a, b : a < b
kisebb(3, 2)
```

Modulok és használatuk

A modul jó közelítéssel azonos a C nyelvben megismert header fájlal. Osztályokat és függvényeket egyaránt tartalmazhat. A modul fájlokat import utasítással tudjuk becsatolni a programunkba. A becsatolt komplett modulok objektumoknak látszanak, tehát ugyan olyan analógiával használhatóak a benne található függvények, mint ha egy osztály tagfüggvényét hívnánk meg.

Amennyiben csak egy-egy elemet importálunk egy modulból, akkor a modul neve nem jön elébe objektum azonosítónak, modul név nélkül hivatkozhatunk rá.

Modulok importálása a from és import kulcsszavakkal történik. Ennek a szintaxisa:

```
from modul neve import importálandó
```

Lehetőség van az importálandó objektum helyi kontextusbeli átnevezésére is. Ez lehetővé teszi, hogy más néven hivatkozzunk a modulra. Ehhez az import utasítás után as kulcszót követően meg kell adnunk az új nevet.

```
from modul neve import importálandó as újnév
```

Amennyiben egy komplett modult szeretnénk importálni, akkor csak a modul nevét kell megadnunk egy import utasítással.

```
import modul neve
```

Természetesen ekkor is alkalmazható az as kulcsszó a komplett modul nevének módosítására

```
import modul neve as újnév
```

A modulokat az értelmező az aktuális szkript mappában kezdi el keresni először, ha ott nem találta meg a keresett modult, akkor az értelmező beépített modul könyvtárában kezdi el keresni azokat.

A modulok létrehozásában semmi ördöngösség nincs, létre kell hoznunk egy .py kiterjesztésű fájlt, ami azokat a dolgokat tartalmazza, amiket a modulba szánunk. A modul neve a fájl neve lesz a kiterjesztés nélkül. Az alábbi példa egy egyszerű modul létrehozását és használatát mutatja be.

A modul fájl: modulom.py

```
def osszead(x, y):
      return x+y

def hatvany(alap, kitevo):
      ertek = 1

      for i in range(0, kitvo):
            ertek *= alap

      return ertek
```

A modult használó program:

```
import modulom as matek

osszeg = matek.osszead(2, 5)
kb = matek.hatvany(2, 10)
```

A Python igen kiterjedt modul könyvtárral érkezik alap telepítésben is, interneten azonban számos kiegészítő modul beszerezhető még pluszban amelyek még tovább bővítik a lehetőségeket. RaspberryPi esetén a fontos modulokat és beszerzési helyeiket az alábbi táblázat foglalja össze. A táblázatban szereplő csomagnév azon csomag nevét tartalmazza, amellyel a modul könnyen telepíthető a rendszerre. Ha a modul neve mellett ilyen nem szerepel, akkor manuálisan telepíthető az adott modul.

Modul	***Leírás***	***Beszerzési hely***	***Csomagnév***
Rpi.GPIO	*GPIO portok közvetlen elérése és kezelése*	*http://code.google.com/p/raspberry-gpio-python/*	*python-rpi.gpio*
Pygame	*Játékok és grafikus alkalmazások készítésére alkalmas keretrendszer*	*http://pygame.org*	*python-pygame*
SimpleCV	*Egyszerűen használható keretrendszer alakzatfelismeréshez.*	*http://simplecv.org/*	
Scipy	*Tudományos számítások elvégzését megkönnyítő modul*	*http://www.scipy.org/*	*python-scipy*
Numpy	*Numerikus algoritmusokat tartalmazó modul, a SciPy erre épül*	*http://numpy.scipy.org/*	*python-numpy*
Flask	*Webes alkalmazások készítéséhez alkalmazható modul*	*http://flask.pocoo.org/*	*python-flask*
Requests	*HTTP lekéréseket és kommunikációt megvalósító modul*	*http://ocs.pythonrequests.org*	*python-requests*
PIL	*Képfeldolgozási modul*	*http://www.pythonware.com/products/pil/*	*python-imaging*
wxPython	*Wx grafikus keretrendszeres alkalmazások készítését megkönnyítő modul*	*http://wxpython.org*	*python-wxgtk2.8*
PySerial	*Soros kommunikáció és hozzáférés a soros portokhoz.*	*http://pyserial.sourceforge.net/*	*python-serial*
pyUSB	*FTDI Soros USB átalakítók kezeléséhez használható modul*	*http://bleyer.org/pyusb*	

12. Táblázat: Fontosabb külső Python modulok

Azon modulok telepítése sem bonyolult, amelyek nem rendelkeznek csomaggal. Ebben az esetben a modulok készítői szoktak biztosítani egy telepítő szkriptet, amely segítségével a csomag letöltése után könnyen telepíthető lesz. A szabványos Python modul telepítő utasítás, amit ki kell adnunk a kicsomagolt modul mappában a telepítéshez:

```
python setup.py install
```

Modulok nem csak Python nyelven fejleszthetőek. Lehetőség van a Python tudásának bővítésére C és C++ nyelvek felhasználásával is. Ez annak köszönhető, hogy a Python értelmező C és C++ nyelven fejlesztett. A gyári modulok többsége ebből adódóan C kódot hajt végre. A modulok fejlesztéséről a http://docs.python.org/2/extending/extending.html#a-simple-example oldalon lehet tájékozódni. Egy lépésről lépésre útmutató a www.tutorialspoint.com/python/python_further_extensions.htm címen található.

A külső modulok letöltését és telepítését nagymértékben megkönnyíti, ha telepítjük a PIP csomag kezelőt. Ez hasonló szolgáltatás, mint a rendszer csomagkezelője, viszont ez kifejezetten python modulokhoz lett kifejlesztve.

A program telepítése egyszerű RaspberryPi alatt, mivel csak a python-pip csomagot kell feltenni.

```
sudo apt-get install python-pip
```

További információk a PIP használatáról a http://www.pip-installer.org/ címen található. A PIP által is telepíthető csomagok listája a https://pypi.python.org/pypi címen található.

A modulok és a programjaink sebessége tovább növelhető a modulok előfordításával. Az elődordítás során a Python értelmező olyan fájlokat hoz létre a modulokból, amelyek már feldolgozottak és könnyen betölthetőek számára a memóriába.

Az alap telepítésben a modulok nincsenek lefordítva. A beépített modulok lefordíthatóak egy beépített szkript segítségével. Ez a compileall nevet viseli. Az alap modulok segítségével az alábbi parancs kiadásával fordíthatóak le:

```
python -m compileall
```

A szkript használható saját moduljaink lefordítására is. Ebben az esetben meg kell neki adni plusz paraméternek a fordítandó fájl nevét. Ha a paraméter egy mappa neve, akkor a mappában található összes modult lefordítja.

Amennyiben a modul forráskódját módosítjuk, akkor az értelmező a forráskódból tölti be a modult és nem a lefordított fájlból. Az értelmező azt, hogy a fájl módosult e a legutóbbi fordítás óta a fájl dátuma alapján dönti el.

A modul fordító használatáról a http://docs.python.org/2/library/compileall.html#module-compileall címen lehet több információt találni.

Hibakezelés

Programozás során kivételnek nevezünk minden olyan eseményt, amely a normál programvégrehajtás megszakadását eredményezheti. Más szóval ezek futás közben fellépő hibák. Ilyen hiba bekövetkezhet az operációs rendszer vagy a hardver miatt is, esetlegesen a program felépítéséből is adódhat kivétel.

A futási időben fellépő hibákat kezelni kell, hogy a programunk stabil legyen. Erre a modern programozási nyelvek tartalmaznak egy vezérlési szerkezetet, amit kivételkezelésnek nevezünk. A kivételkezelés előnye, hogy a fellépő hibák észlelését és kezelését szétválaszthatjuk.

Az olyan nyelvekben amelyek nem rendelkeznek kivételkezeléssel, azokban az alábbi módszereket tudjuk alkalmazni egy függvényben fellépő hiba kezelésére:

- ***Befejezzük a program futását.***
  Ez a módszer abban az esetben használhatatlan, ha a programnak nem szabad hirtelen kilépnie. Amennyiben megengedett is, akkor is kerülendő az alkalmazása, főleg grafikus programok esetén, mert a felhasználói élményt drámaian lerontja.
- ***Egy hiba jelzésű értéket adunk vissza***
  Ez a módszer nem alkalmazható minden esetben, mivel nem minden esetben található hibát jelentő érték a függvény visszatérési értéktartományában. Példa erre egy olyan függvény, ami egy szövegben található számokat konvertálja számmá. Ilyen esetben a visszatérési érték szám, amiből nem köthetünk le egy számot fixen hibajelzésre, mivel szerepelhet a bemenet között, így hibás hibajelzést okozna a programban. Továbbá ha találunk is értéket, azt később kezelnünk kell egyedileg, ami a program méretét feleslegesen növeli, valamint minden egyes esetre sok idő megírni. Ezért gyakran el is hagyják.
- ***Normális értéket adunk vissza, és a programot szabálytalan állapotban hagyjuk.***
  Hibák halmozódásához vezető módszer, mivel nem derül ki, hogy a program hibás állapotban van. Előbb-utóbb a program futás közbeni összeomlásához vezet a módszer.
- ***Meghívunk egy hibakezelésre szolgáló függvényt.***
  Igazából ez nem külön módszer, mivel a függvényben a fentebb ismertetett három módszer közül tudunk választani.

A fenti módszerek hibáinak kiküszöbölésére találták ki a kivételkezelést. A C++ is támogatja ezt a szerkezetet, azonban azért nem lett említve a C++ szekcióban, mert a beépített függvénykönyvtár függvényei nem alkalmazzák a módszert.

Python esetén kivételkezelést az alábbi szintaxis segítségével tudunk megvalósítani:

```
try:
      utasítások
except Error:
      hibakezelés
```

A try blokkban szereplő utasítások az úgynevezett védett utasítások, amelyeket megpróbál végrehajtani az értelmező. Amennyiben az utasítások bármelyikében hiba lépne fel, akkor a try blokk futása megszakad és a vezérlés az except blokkra ugrik.

Az except kulcsszó után a kezelendő hiba típusát kell megadni. Python esetén az összes hiba őse az Exception osztály, amely Error néven van példányosítva a futtatóban. Amennyiben Error típust adunk meg, akkor minden futás közben lévő hibát elkaphatunk és kezelhetünk.

Amennyiben csak specifikus hibatípusokat kívánunk kezelni, akkor érdemes az Exception típus valamely származtatott típusára korlátozni az except ágat. A specifikus hibakezelő típusok mind Error végződéssel vannak ellátva.

Abban az esetben, ha két specifikus hibát szeretnénk kezelni, akkor nem feltétlen kell több except ágat írnunk, elég egyet, mivel több típusú kivétel is elkapható. Ebben az esetben a hibák típusait vesszővel kell elválasztani.

```
try:
      utasítások
except (RuntimeError, TypeError, NameError):
      hibakezelés
```

A hibáról bővebb információ is kinyerhető, ha a hibát változóvá konvertáljuk. Ehhez az as kulcsszó használható.

```
try:
      f = open("pelda.txt")
except IOError as hiba:
      print "Hiba tortent: ", hiba
```

A kivételkezelő blokkunkban definiálhatunk egy olyan ágat is amely minden esetben le fog futni, ha történt hiba, ha nem. Ez különösen akkor hasznos, ha a programunk fájlokkal dolgozik. Ekkor ha fellépet hiba, ha nem nem lenne célszerű, hogy a műveletek végezete után is a megnyitás miatt zárolva maradjon az operációs rendszernek a fájl. Ilyen utasítások létrehozásához a kivételkezelő blokkot ki kell egészíteni egy finally blokkal:

```
try:
      utasítások
except Error:
      hibakezelés
finally:
      mindig lefutó kód
```

A függvényeinkkel hibát dobni a raise kulcsszóval tudunk. Amint a raise kulcsszóhoz érkezik a feldolgozó megszakítja a függvény futtatását. A raise utasítás után szintén az Exception osztály leszármazottjait használhatjuk.

Alap modulkönyvtár

A Python kiterjedt modulkönyvtár rendszerrel rendelkezik, amely függvényei és osztályai segítségével igen sok feladat megvalósítható küldő modulok használata nélkül is. A beépített modulok függvényi a könyv ezen szekciójában nem lesznek ismertetve, mivel a függvények kiadásonként kisebb-apróbb változásokon esnek keresztül.

A teljes modulkönyvtár dokumentációja a http://docs.python.org/2/library/index.html címen lelhető fel.

Számos ingyenes E-book is foglalkozik a Python programozási nyelvvel. Ezen könyvek közül talán a legjobb a Python Tutorial, amely megalkozásában Guido van Rossum, a Python nyelv kitalálója is részt vett. A ~130 oldalas könyv magyar fordítása ingyenesen letölthető a http://pythontutorial.pergamen.hu/ címről.

Számos rövidebb, velős összefoglaló is létezik a nyelvről és a modulokról. Ezek közül egy igen jó a Python Quick Reference Card, amely a http://perso.limsi.fr/pointal/python:pqrc címről szerezhető be ingyenesen, illetve a könyvhöz készült mellékletben is megtalálható.

Python bővítése C/C++ kóddal

Python esetén lehetőségünk van modulokat fejleszteni C/C++ nyelven is, amennyiben a Cpython (hivatalos Python) implementációt használjuk. A C/C++ modul fejlesztés előnye, hogy további beépített típusokat adhatunk hozzá a nyelvhez, illetve C függvénykönyvtárakat használhatunk, valamint nem utolsósorban a C/C++ nyelven megírt kódunk gyorsabban fog futni, mint a Python kód.

A modulok fejlesztéséről egy jó leírás a http://docs.python.org/2/extending/extending.html címen található. Ez részletesen belemegy az értelmező lelki világába és a működésébe.

C függvénykönyvtárak függvényeinek meghívására létezik egy egyszerűbb módszer is, ami a ctypes modult használja. Ezen modul lehetővé teszi a Python értelmező számára, hogy meghívjon C-ben íródott függvény könyvtár funkciókat. Így típusokat nem tudunk definiálni, azonban ezen megoldás egyszerűbb, mivel nem kell belemászni az értelmező lelki világába.

A megoldás használatát a legegyszerűbb egy példán keresztül bemutatni, ezért létrehozunk egy minta függvény könyvtárat C++ segítségével, aminek a függvényeit meghívjuk Pythonból.

A függvénykönyvtár létrehozás első lépése, hogy tisztázzuk mit is nevezünk függvénykönyvtárnak. A függvénykönyvtár egy bináris fájl, ami függvényeket tárol, amelyek más programok által meghívhatóak. Linux/Unix rendszerek ezen fájlok .so kiterjesztéssel rendelkeznek, míg Windows rendszerek esetén a .dll kiterjesztés alkalmazott.

Linux és GCC esetén az, hogy függvény könyvtár lesz e egy programból, vagy futtatható program lényegében a fordításon múlik. A kódot annyiban kell módosítani, hogy a könyvtár által exportálandó függvényeket sima C típusú linkeléssel kell ellátnunk. Erre való az extern "C" utasítás. A következő kód négy függvényt definiál, amelyeket később meghívunk.

```
extern "C" int Szoroz(int mit, int mivel)
{
     return mit * mivel;
}

extern "C" double Pi2()
{
     return 3.1415 * 2;
}

extern "C" double Osszead(double param1, double param2)
{
     return param1 + param2;
}

extern "C" const char * Szoveg()
{
     return "Ez egy fuggveny konyvtar";
}
```

A kódunk G++ segítségével az alábbi két utasítással fordítható le meghívható függvénykönyvtárrá:

```
g++ -shared -c -fPIC minta.cpp -o minta.o
g++ -shared -Wl,-soname,minta.so -o minta.so minta.o
```

A fenti két parancs kiadása után létrejön a minta.so fájl, ami binárisan tartalmazza a függvényeinket. Ezt az so fájlt tudjuk betölteni a ctypes segítségével. Az alábbi python mintakód sorban meghívja a so fájlban definiált függvényeinket.

```
import ctypes
from ctypes import *
```

```
lib = cdll.LoadLibrary('./minta.so')

lib.Pi2.restype = ctypes.c_double
lib.Osszead.restype = ctypes.c_double
lib.Szoveg.restype = ctypes.c_char_p

print 'Szoroz(3, 2): ', lib.Szoroz(3, 2)
print 'Pi2(): ', lib.Pi2()
print 'Osszead(11.1, 12.2): ', lib.Osszead(c_double(11.1), c_double(12.2)
print 'Szoveg(): ', Szoveg()
```

A cdll osztály LoadLibrary függvénye betölti a függvénykönyvtárt, amely függvényeit ezután úgy használhatunk, mint egy modul. A függvények alapértelmezett visszatérési értékeként int értéket feltételez, így ha nem int a függvényünk visszatérési értéke, akkor konvertálni kell a restype tulajdonság beállításával. A típusokat a ctypes osztály tartalmazza. Amennyiben a függvényünk nem csak int paraméterekkel dolgozik, akkor szintén konvertálni kell az argumentumok típusát is. Az alábbi táblázat a ctypes által definiált c típusokat és konverziós függvényeket foglalja össze.

ctypes típus	***C típus***
c_bool	*_Bool*
c_char	*char*
c_byte	*char*
c_ubyte	*unsigned char*
c_short	*short*
c_ushort	*unsigned short*
c_int	*int*
c_uint	*unsigned int*
c_long	*long*
c_ulong	*unsigned long*
c_longlong	*long long*
c_ulonglong	*unsigned long long*
c_float	*float*
c_double	*double*
c_longdouble	*long double*
c_char_p	*char **
c_void_p	*void **

13. Táblázat: A ctypes típusai és a hozzájuk tartozó C típusok

Olyan függvények használata is lehetséges, amelyek struktúrában adnak vissza adatot. Itt azonban bejön a dinamikus memória kezelés problémája. A legegyszerűbb, ha a struktúrákon dolgozó függvényeinket úgy írjuk meg, hogy referencia értékként adjanak vissza értéket.

Ebben az esetben a Python értelmező feladata a memória kezelése, ami nem zavar be a függvénykönyvtár oldalon túlzottan.

A megoldás használatát szintén példán keresztül mutatom be. Az alábbi kód definiál egy struktúrát és egy függvényt, ami a struktúrát referenciaként veszi át és módosítja annak a tartalmát.

```
extern "C"
{
	typedef struct
	{
		int egesz;
		double lebeg;
		char ch;
	} struktura;
```

```
	void strukttest(struktura * adat)
	{
		adat->egesz = 42;
		adat->lebeg = 3.14;
		adat->ch = 'A';
	}
}
```

Fontos, hogy a kódban a struktúráink typedef utasítással is ki legyenek egészítve. Amennyiben ez elmarad, akkor problémákba ütközünk a kód használata közben. A kódban újdonság lehet, hogy az extern "C" utasítás blokként is használható.

A kód ugyan azon G++ utasítások segítségével lefordítható, mint amit az előző példában alkalmaztunk. Fordítás után az alábbi python kód mutatja be az egyedi típusunk használatát.

```
from ctypes import *
import ctypes

class struktura(Structure):
	_fields_=[('egesz', c_int), ('lebeg', c_double), ('ch', c_char)]

data = struktura()
lib = CDLL("./lib.so")
lib.strukttest(ctypes.byref(data))

print data.egesz
print data.lebeg
print data.ch
```

A struktúránk felépítését definiálnunk kell az értelmező számára, méghozzá úgy, hogy egy osztályt származtatunk a Structure típusból és a származtatott osztály _fields_ listájában megadjuk a struktúra elemeit és C típusait. Az osztályunkat példányosítjuk, majd referenciaként átadjuk a függvénynek a ctypes.byref függvényével. Ha nem referenciaként adjuk át, akkor a program összeomlik.

6. Elektronikai alapismeretek

Az elektronika témakör nagysága miatt ezen fejezet elsősorban csak az egyenfeszültségű elektronikai ismereteket tárgyalja, mivel a legtöbb esetben erre lesz szükségünk mikrovezérlők kapcsán.

Alapfogalmak

- ***Feszültség:***
  Villamos erőtérben A és B pont közötti töltés mozgatásához szükséges munka egységnyi töltésre vonatkoztatva.
  *Jelölés: **U**, mértékegység: [**V**] Volt*
- ***Áramerősség:***
  Egységnyi idő alatt a vezetőn keresztülhaladó töltésmennyiség.
  *Jelölés: **I**, mértékegység: [**A**] Amper*
- ***Teljesítmény:***
  Munkavégzés sebessége. Az egyenáramú villamosteljesítmény a feszültség és az áramerősség szorzata.
  *Jelölés: **P**, mértékegység: [**W**] Watt*
- ***Ellenállás:***
  Az anyagok azon tulajdonsága, amely az áram folyását akadályozza. Azért keletkezik, mert a töltéshordozó részecskék ütköznek az anyag atomjaival.
  *Jelölés: **R**, mértékegység: [**Ω**] Ohm*

Alap összefüggések

$$R=\frac{U}{I}\ I=\frac{U}{R}\ U=R\cdot I\ P=U\cdot I\ I=\frac{P}{U}\ U=\frac{P}{I}\ P=I^2\cdot R\ P=\frac{U^2}{R}$$

Prefixumok

Villamos mennyiségek leírásához gyakran használunk prefixumokat, mivel könnyebb kimondani azt, hogy 320mA a 0,320A helyett. Továbbá nem utolsósorban esztétikusabban is néz ki leírva. Az elektronikában a leggyakrabban használt prefixumok értékeit és neveit az alábbi táblázat foglalja össze jelöléseikkel:

Név	***Érték***	***Jelölés***
Giga	10^{9}	*G*
Mega	10^{6}	*M*
Kilo	10^{3}	*k*
Mili	10^{-3}	*m*
Mikro	10^{-6}	*μ*
Pico	10^{-9}	*p*
Nano	10^{-12}	*n*

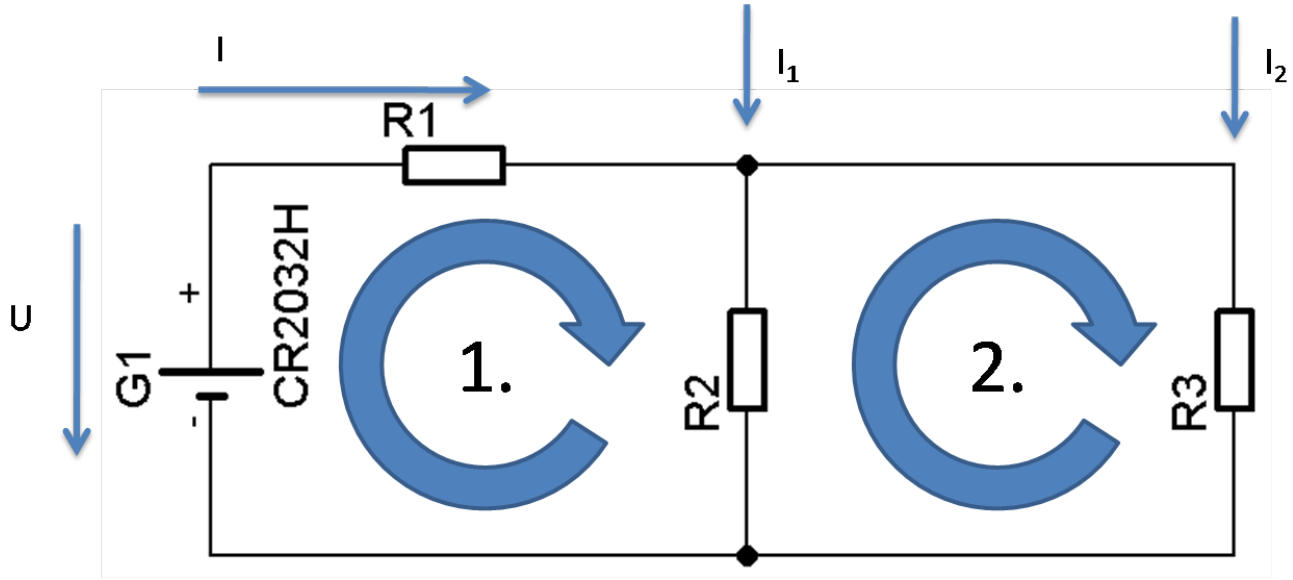
14. táblázat: Prefixumok értékei és jelölésük

Kirchhoff törvények

A villamosenergia-megmaradás törvényei. Gustav Robert Kirchhoff német fizikus mondta ki ezen két törvényt először.

1. ***Bármely zárt áramhurokban a részfeszültségek előjelhelyes összege zérus.***
2. ***Bármely csomópontba befolyó áramok összege megegyezik az onnan elfolyó áramok összegével.***
   Ezen összefüggésekkel tetszőlegesen bonyolult hálózatok leírhatóak, amennyiben az egyenletek megoldásához

kellő adattal rendelkezünk. Minden esetben a független hurkok száma +1 egyenletből álló egyenletrendszert kapunk. Példaként íme egy egyszerű kapcsolás és a hozzá tartozó egyenletek:



$$1.-U+I\cdot R_1+I_1\cdot R_2=0 \quad 2.\ I_2\cdot R_3-I_1\cdot R_2=0 \quad 3.\ I=I_1+I_2$$

Ellenállás

Az ellenállás az anyagok szerkezeti felépítéséből következik, vagyis az áram folyása során a töltések nekiütköznek az anyag atomjainak, ami akadályozza az áram folyását. Az anyag ezen tulajdonságát nevezzük ellenállásnak.

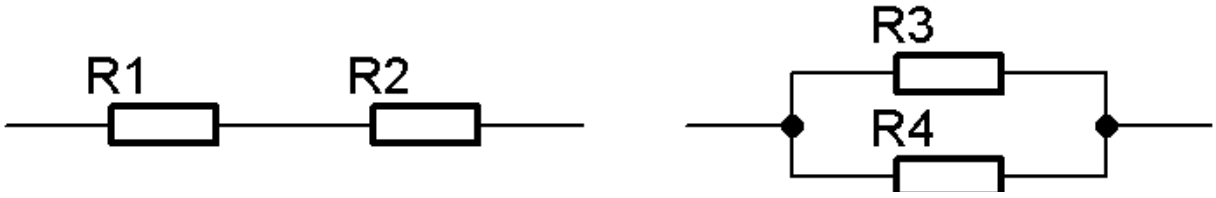
Az anyagokat az ellenállásuk szerint három fő csoportba lehet sorolni.

- ***Szigetelők***
  Olyan anyagok, amelyek az elektromos áramot elhanyagolható mértékben vezetik. Tipikusan az ellenállásuk $10^{12}\,\Omega$ *felett van. Ezen anyagokban kevés a szabad elektron, ezért a vezetőképességük kicsi. Egy ideális szigetelőben nincs szabad elektron.*
- ***Vezetők***
  Olyan anyagok, melyek az elektromos áramot jól vezetik. Tipikusan ilyenek a kristályos szerkezetű fémek, mivel a kristályos szerkezetű fémekben sok a szabad elektron.
- ***Félvezetők***
  Olyan anyagok, amelyek nagyon gyengén vezetik az áramot, ezért nem jók vezetőknek és szigetelőknek sem használhatóak. Ezen anyagok ellenállása nagyon hőmérséklet függő. Önmagukban nem igen alkalmazzák őket, különböző szennyező anyagokkal módosítják szerkezetüket félvezető eszközök kialakításához.

Az anyagok ellenállása hőmérséklet függő. Léteznek PTC típusú anyagok és NTC típusúak. A PTC anyagok ellenállása a hőmérséklet emelkedésével nő, míg az NTC anyagoknál csökken. PTC típusú anyagok a fémek, NTC típusúak meg a félvezetők.

A Kirchhoff egyenletek segítségével könnyen bebizonyítható, hogy ***a sorosan kapcsolt ellenállások eredője a tag ellenállások összegével egyenlő.*** $R_e = R_1 + R_2$

9.



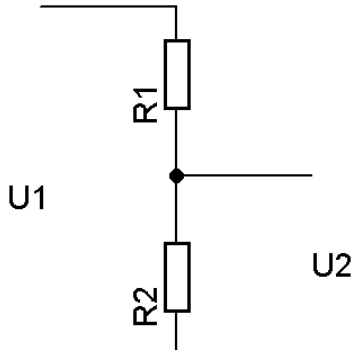
Párhuzamos kapcsolás esetén az eredő ellenállás reciproka egyenlő a tag ellenállások reciprokával. $\frac{1}{R_e} = \frac{1}{R_3} + \frac{1}{R_4}$ *Ez másként kifejezve:* $R_e = \frac{R_1 \cdot R_2}{R_1 + R_2}$ *Ezen képlet a matematikában replusz műveletként ismert. Segítségével explicit módon fejezhető ki az ellenállás.* $R_e = R_1 \times R_2$ *A replusz művelet kommutatív és asszociatív tulajdonságú. A párhuzamosan kapcsolt ellenállásokra mindig igaz, hogy kisebb lesz a párhuzamos kapcsolásban szereplő legkisebb ellenállásnál.*

Feszültségosztó

A Kirchhoff huroktörvény gyakorlati alkalmazása a feszültségosztó. Vagyis két darab sorosan kapcsolt ellenállás esetén, ha a körre U_1 feszültség jut, akkor a második ellenálláson U_2 feszültség mérhető, amely arányos az ellenállások értékével. Kiszámítani a következőképpen lehetséges:

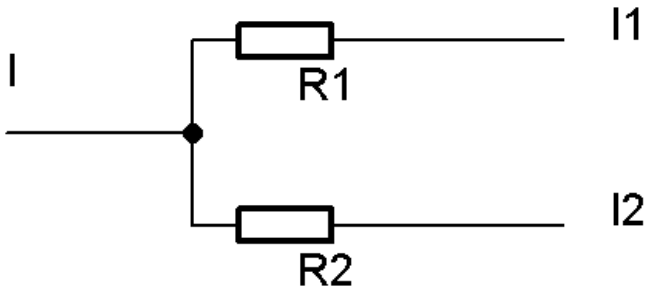
$$U_2 = \frac{U_1 \cdot R_2}{R_1 + R_2}$$

10. ábra:



Áramosztó

Kirchhoff csomóponti törvényét felhasználva belátható, hogy egy csomópont áramosztást végez a csomópont ágai között. Az áramok nagyságát az egyes ágak ellenállása határozza meg.



A fenti kapcsolási rajzon R_2 ellenálláson folyó I_2 áram a következőképpen számítható ki:

$$I_2 = \frac{I \cdot R_1}{R_1 + R_2}$$

Amennyiben több párhuzamos ág van, nem csak kettő, akkor R_1 ellenállás helyére a többi ág eredő ellenállása írandó.

Kapacitás

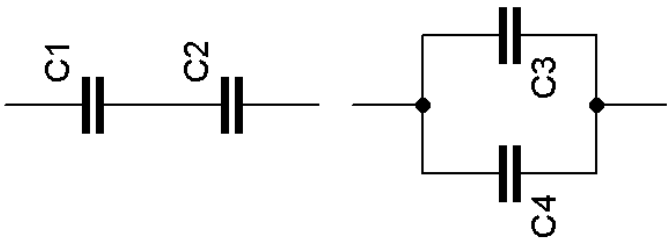
Az ellenállásokon kívül passzív áramköri elem még a kondenzátor és a tekercs. A kondenzátoroknak kapacitásuk van. A kapacitás azt adja meg, hogy egy kondenzátor egységnyi feszültség hatására mennyi töltést képes tárolni. A mennyiség jele C, mértékegysége a Farad [F]. A kapacitás számítható a felhalmozott töltések mennyiség és feszültség ismeretében az alábbi képlet segítségével:

$$C=\frac{Q}{U}$$

Az 1 Farad kapacitás igen nagy mennyiségnek számít, mivel becslések szerint a föld bolygó teljes kapacitása nem éri el az 1 Farad-ot. Éppen ezért a gyártott kondenzátorok többsége esetén a kapacitás egy mikro, piko vagy nano prefixummal van ellátva.

A gyártástechnológia fejlődésével megjelentek, az úgynevezett szuper kondenzátorok. Ezek kapacitása elérheti az 50 Farad-ot is, azonban a maximális feszültség, amivel elbírnak tipikusan pár volt nagyságrendű. Leginkább kis akkumulátorok kiváltására szánták őket kezdetben. Elektromos autók esetén is alkalmazzák őket, ott arra vannak használva, hogy az akkumulátor energiáját puffereljék. Erre azért van szükség, mert az akkumulátorok belső felépítéssülből adódóan rendelkeznek egy belső ellenállással, ami a hőmérséklet és terhelő áram hatására változik. Vagyis hiába tárol rengeteg energiát, nem lehet belőle kivenni gyorsan nagy mennyiségben. A kondenzátorokból viszont igen, mivel a belső ellenállásuk elhanyagolható méretű az akkumulátorok ellenállásához képest.

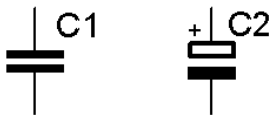
Ha két kondenzátort párhuzamosan kötünk, akkor az eredő kapacitásuk összeadódik, soros kapcsolás esetén pedig az eredő kapacitás az ellenállások esetén ismertetett replusz művelet segítségével számítható ki, vagyis az eredő kapacitás az eredő kapacitás reciproka megegyezik a részkapacitások reciprokainak összegével.



Soros kapcsolás esetén a rendszerre kapcsolt feszültség megoszlik a tagok esetén, így a rendszerre összességében nagyobb feszültség kapcsolható, mint külön-külön az egyes kondenzátorokra.

Kondenzátorokból létezik polarizált és nem polarizált. A polarizált kondenzátorok az elektrolit kondenzátorok. Itt a töltéstárolásért egy elektrolit oldat/zselé felel. Itt kimondottan ügyelni kell a helyes polaritásra, mivel ha véletlen fordított polaritással építünk be egy ilyen kondenzátort, akkor könnyen felrobbanhat.

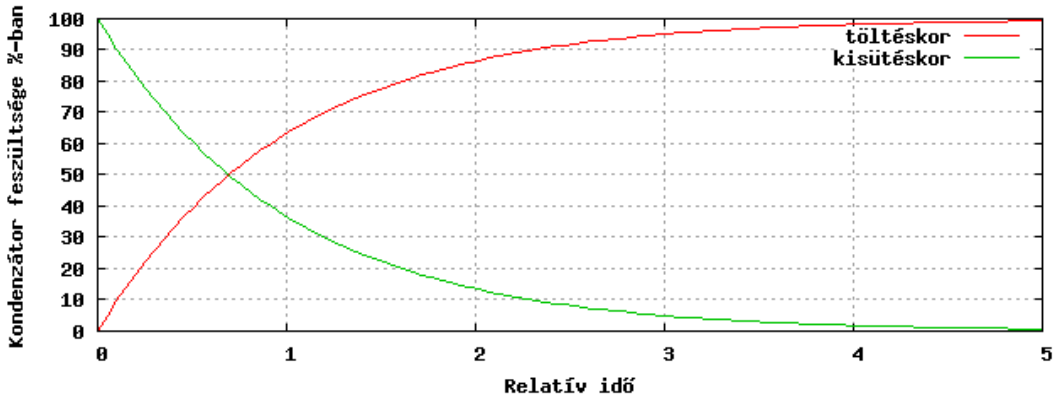
13. Ábra:



Polarizálatlan és polarizált kondenzátor rajzjele

Ezen kondenzátorok további jellemzője, hogy a névleges kapacitásuk az idő előrehaladtával fokozatosan csökken, mivel a benne lévő elektrolit oldat fokozatosan szárad ki. Az ilyen kondenzátorok élettartalma nagyon függ a környezeti hőmérséklettől és magától a gyártás minőségétől is. Az ilyen kondenzátorokat tipikusan tápegységekben energia tárolásra szokás alkalmazni, míg a nem polarizált kondenzátorokat leginkább zavar szűrésre.

Ha a kondenzátort feszültség alá helyezzük, akkor nem azonnal töltődik fel energiával, hanem fokozatosan.



Az ábrán a relatív idő függ a töltő áramerősségtől, képletek esetén τ (görög tau) betűvel jelölt. 1τ idő alatt a kondenzátor a kapacitásának 63%-ra töltődik, 5τ idő alatt pedig 99,3%-ra. Mivel a kondenzátorok egy ellenálláson keresztül töltődnek és sülnek ki általában a relatív idő az alábbi képlet segítségével számítható: $\tau = R \cdot C$ *Amennyiben áramgenerátor segítéségével töltünk egy kondenzátort a relatív idő a* $\tau = \frac{U \cdot C}{I}$ *képlet segítségével számítható ki, ahol U a kondenzátor névleges feszültségét jelöli.*

A kondenzátorban tárolt energia a $E = \frac{1}{2} \cdot C \cdot U^2$ *képlettel határozható meg és Joule-ban értendő. Ezen tárolt energia miatt nem érdemes tápegységekben közvetlen kikapcsolás után szerelni, mivel a nagy feszültségű kondenzátorokban annyi energia könnyen lehet, hogy halálos áramütést szenvedjen a szerelő.*

Induktivitás

Az induktivitás két fogalmat is jelöl. Egy eszközt, amely rendelkezik induktivitással (tekercsek tipikusan) és magát a jelenséget. A tekercsen átfolyó áram létrehoz a tekercs körül egy mágneses teret, amely mágneses tér változása ellentétesen hat az áram növekedésére. Ha tekercsre egy feszültségforrást kapcsolunk, a rajta átfolyó áram nem ugrásszerűen jön létre, hanem folyamatosan növekszik. Az áram növekedésének korlátozódása a tekercs induktivitása.

$$I = \frac{U}{L} \cdot t$$

A képletben L betűvel van jelölve az induktivitás, aminek a mértékegysége Henry [H]. A t betű egy időt jelent, amely a feszültség rákapcsolásától számított.

Egy tekercs induktivitása a geometriai adatoktól és a tekercsben felhasznált mágneses anyag adatainak segítségével számítható ki.

Ha az árammal járt tekercsről hirtelen leválasszuk a feszültségforrást, akkor a tekercsben kialakult mágneses tér megpróbálja fenntartani a rajta átfolyó áramot. Így a feszültségforrás pozitív oldalán nagyméretű negatív feszültség mérhető ebben az esetben, amely a fenti képlet alapján t idő alatt omlasztja össze a tekercs mágneses terét.

Az induktivitások soros és párhuzamos kapcsolására az ellenállások esetén ismertetett képletek alkalmazhatóak, vagyis két tekercs soros kapcsolása esetén az eredő induktivitás az egyes induktivitások összege. Párhuzamos kapcsolás esetén pedig az eredő induktivitás reciproka egyenlő a tag induktivitások reciprokával.

Váltakozó áramú hálózatok

Elektronikában alapvetően kétféle hálózat létezik. Egyenáramú, mikor az időtől függetlenül a feszültség hatására létrejövő áram minden időpillanatban azonos nagyságú. Váltakozó áramú hálózatok esetén a feszültség az időtől függ, így a kialakuló áram is időfüggő lesz.

A leggyakoribb váltakozó áramú hálózatokban a feszültség szinuszosan változik, ezt nevezik köznapi néven váltóáramnak. Azonban gyakorlatilag bármilyen függvény szerint változhat a feszültség egy hálózatban. Digitális rendszerek esetén tipikusan négyszögjel formában változik a feszültség.

Váltakozó áramú rendszerekben egy feszültségforrás vagy áramforrás korrekt leírásához szükséges a forrás csúcsértéke, frekvenciája, a függvény, ami leírja a kimenő jelet, valamint szükséges a kezdő vagy más néven a fázis szög, ami megadja, hogy a bekapcsolás pillanatában a kimeneti függvény melyik pontjáról indult a rendszer.

Gyakorlatban ezt igen ritkán látjuk feltüntetve bárhol is. Ennek az oka az, hogy nem minden esetben szükséges a dolgokat túlbonyolítani. Általában a váltakozó áramú forrásokat az effektív értékükkel szoktuk jellemezni és a frekvenciával, feltételezve, hogy 0 a fázisszög.

Az effektív érték a jel négyzetes átlaga, ami szinuszos hálózatok esetén a Maximális érték ismeretében a következő képletek segítésével számolható ki:

$$U_{eff} = \frac{U_{max}}{\sqrt{2}} \; I_{eff} = \frac{I_{max}}{\sqrt{2}}$$

Az effektív érték annak az egyenáramnak az értékével egyenlő, amely azonos idő alatt ugyanakkora munkát végez, mint a vizsgált váltakozóáram.

Négyszögjelet használó rendszerek esetén az effektív érték attól függ, hogy mennyi ideig van magas szinten és mennyi ideig van alacsony szinten a jel. Ha a jel a periódus idő felében magas szinten és felében alacsony szinten van, akkor az effektív érték a maximális érték fele.

A váltakozó áramú rendszerekben a kondenzátorok és tekercsek rendelkeznek egy frekvencia függő ellenállással, amit látszólagos ellenállásnak nevezünk. Kondenzátorok esetén a frekvencia növekedésével ez az ellenállás csökken, míg az induktivitásoknál ez az ellenállás növekszik. A látszólagos ellenállás jelölése X_L tekercsek és X_C kondenzátorok esetén. Kiszámításukra az alábbi képletek használhatóak:

$$X_L = 2 \cdot \pi \cdot f \cdot L = \omega \cdot L \; X_C = \frac{1}{2 \cdot \pi \cdot f \cdot C} = \frac{1}{\omega \cdot C}$$

A képletekben szereplő ω betű a körfrekvenciát jelenti. Ez azt adja meg, hogy a hullám fázisa mennyit változik egy periódusidő alatt. Mértékegysége rad/s.

Kondenzátorok esetén megfigyelhető az a jelenség, hogy a feszültség késni fog az áramhoz képest 90 fokkal, míg tekercsek esetén az tapasztalható, hogy a feszültség siet 90 fokkal az áramhoz képest. A feszültség és áram egymáshoz viszonyított fázis helyzete a teljesítmény meghatározása miatt fontos.

Egy hálózat frekvencia függő ellenállását Impedanciának nevezzük és Z betűvel jelöljük. Kiszámítása bonyolultabb hálózatok esetén csak komplex számok alkalmazásával lehetséges, mivel a kondenzátorok és tekercsek eltolják a fázisszöget, amit figyelembe kell venni számításkor. Az alábbi példa egy soros RLC kör impedancia számítását mutatja be.

A példa kapcsolás adatai:

$$R = 300\,\Omega \; L = 740\,mH \; C = 60\,\mu F \; U = 200\,V \; f = 50\,Hz$$

Az adatokból és a fentebb említett képletek segítségével gyorsan kiszámítható a kondenzátor és tekercs

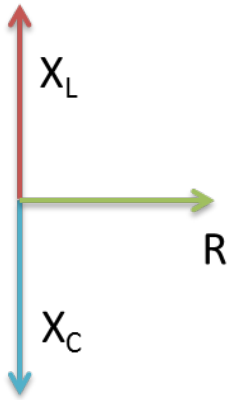
látszólagos ellenállása a megadott frekvencián:

$$X_L = 2 \cdot \pi \cdot 50\,Hz \cdot 740 \cdot 10^{-3} H = 232.478\,\Omega$$

$$X_C = \frac{1}{2 \cdot \pi \cdot 50\,Hz \cdot 60 \cdot 10^{-6} F} = 53{,}051\,\Omega$$

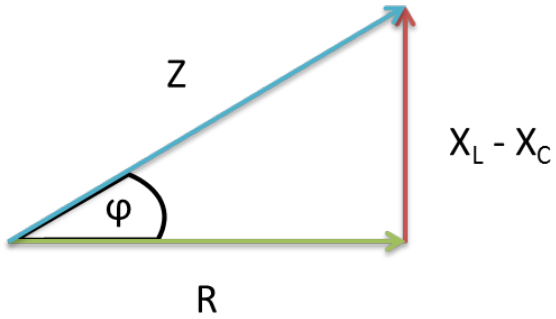
Az impedancia kiszámításához egy vektor ábrát kell felvennünk, amiben az Y tengelyen helyezkednek el a látszólagos ellenállások, az X tengelyen meg a valós ellenállás. A tekercs látszólagos ellenállása az Y tengely pozitív irányába mutat, míg a kondenzátor látszólagos ellenállása az Y tengely negatív irányába. A kettőjük különbsége lesz a kör látszólagos ellenássa.

17. Ábra: Az ellenállás



és a látszólagos ellenállás vektorábrája

Az imedancia a látszólagos ellenállás és a valós ellenállás vektoriális összege.



Az impedancia könnyen kiszámítható a Pitagorasz tétel alkalmazásával

$$|Z| = \sqrt{R^2 + (X_L - X_C)^2}$$

A fázisszög pedig a szögfüggvények segítségével számítható: $tg\phi = \frac{X_L - X_C}{R}$

Váltakozó áram esetén a teljesítmény számítása esetén figyelembe kell venni a fázisszöget, mivel szerepet játszik a hatásos teljesítmény kialakulásában. A hatásos teljesítmény váltakozó áramú hálótatok esetén az alábbi képlet segítségével határozható meg: $P = U \cdot I \cdot \cos\phi$

A feszültség és áramerősség szorzatát váltóáram esetén látszólagos teljesítménynek nevezzük. Ez az a teljesítmény mennyiség, amit multiméterrel való mérések után számolni tudnánk. Ezt a teljesítményt váltóáram esetén S betűvel jelölük és mértékegysége a VA, voltamper.

A harmadik teljesítmény összetevő az úgynevezett meddő teljesítmény, amit a hálózatban lévő induktív és kapacitív elemek látszólagos ellenállása emészt fel. Minél nagyobb, annál kisebb az elérhető hatásos teljesítmény, nehezíti a villamos energia szállítását. Ezért nagyfogyasztók esetén ezt is méri az áramszolgáltató. A meddő teljesítményt Q betűvel jelöljük, mértékegysége a voltamper reaktív, a VAr. A következő képlet segítségével számítható ki:

$$Q = U \cdot I \cdot \sin\phi$$

A három teljesítmény között összefüggés áll fenn, ami szintén vektorosan ábrázolható az impedanciához hasonlóan. $S=\sqrt{P^2+Q^2}$

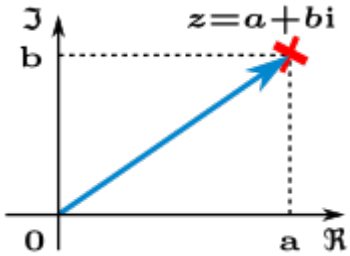
Komplex számok

A probléma a másodfokú egyenletekkel az, hogy bizonyos egyenletek megoldhatatlannak bizonyultak a megoldóképlet által. Például a $(x+1)^2=-9$ nincs megoldása, mivel a negatív valós számok gyöke nem értelmezhető.

A probléma áthidalható úgy, hogy a számokat kiegészítjük egy képzetes egységgel, amit i-vel jelölünk i a következőképpen definiált: $i^2=-1 i=\sqrt{-1}$

A komplex számokat $a+i\cdot b$ formában definiáljuk (ezt nevezzük algebrai alaknak), a és b valós számok. A szám valós részét a reprezentálja, míg a képzetes részt b. Ezáltal a komplex számok felfoghatóak egy helyzetvektorként. A számok valós részét mindig az X tengelyen, a képzetes részt pedig az Y tengelyen ábrázoljuk.

Villamos számítások esetén a képzetes kitevőt j betűvel jelölik, mivel az i betű alkalmazása keverhető lenne az áramerősség jelével. A leírás további részeiben j betűvel lesz jelölve a képzetes rész.



Ha a komplex számot vektorként értelmezzük, akkor megadható a szám a vektor hosszával és a valós tengellyel bezárt szöggel. Ezt a megadási módszert nevezzük trigonometrikus alaknak. A trigonometrikus alak a következőképpen néz ki: $z=r(cos\phi+j\cdot sin\phi)$

A képletben r jelképezi a vektor hosszát. a Pitagorasz tétel és a derékszögű koordináta-rendszerről tanult összefüggések segítségével felírhatóak a következő összefüggések:

$$|z|=r=\sqrt{a^2+b^2}\ arctg\phi=\frac{b}{a}$$

Továbbá érdemes az algebrai alak és a trigonometrikus alak között további összefüggéseket is felírni:

$a=r\cdot cos\phi b=r\cdot sin\phi$

A számítások során j ugyan úgy kezelendő, mint egy valós szám. A j hatványai négyes periódust követnek:

$$j^1=j$$
$$j^2=j\cdot j=1$$
$$j^3=j\cdot j^2=-j$$
$$j^4=j\cdot j^3=1$$
$$j^5=j\cdot j^4=j$$

A komplex számok halmaza nem rendezett, ezért nem mondható, hogy az egyik komplex szám nagyobb a másiknál.

Azonosságok

$$1.|z|=0 \rightarrow z=0$$
$$2.|z^2|=z\cdot -z$$
$$3.|z|=|-z|$$
$$4.-z_1\cdot z_2=-z_1\cdot -z_2$$
$$5.|z_1\cdot z_2\cdots z_n|=|z_1|\cdot|z_2|\cdots|z_n|$$
$$6.|z^n|=|z|^n$$
$$7.|\frac{z_1}{z_2}|=\frac{|z_1|}{|z_2|}\,z_2\neq 0$$
$$8.\frac{1}{z}=\frac{-z}{|z|}$$

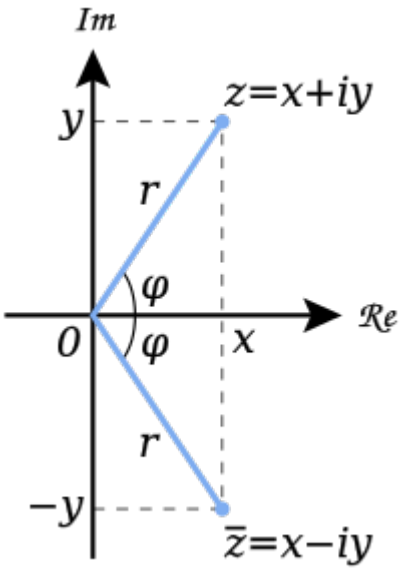
Műveletek

Egyenlőség

Két komplex szám akkor és csak akkor egyenlő, ha a valós és képzetes részük is egyenlő.

Konjugáció

Amennyiben a komplex számunkat $z=a+b\cdot j$ *formában definiáljuk, akkor létezik hozzá egy konjugált szám, amely* $-z=a-b\cdot j$ *formában definiált. Geometriai értelmezésben a komplex szám konjugálása a valós tengelyen való tükrözést jelenti.*



Ebből adódóan ha kétszer konjugáljuk z-t, akkor z-t kapjuk vissza. $\bar{\bar{z}}=z$

Összeadás

Algebrai alakban: $z_1+z_2=(a+b\cdot j)+(c+d\cdot j)=(a+c)+(b+d)\cdot j$

Példa:

$$z_1=3+2\cdot j$$
$$z_2=5+6\cdot j$$
$$z_1+z_2=(3+5)+(2+6)\cdot j=8+8\cdot j$$

Kivonás

Algebrai alakban: $z_1 - z_2 = (a + b \cdot j) - (c + d \cdot j) = (a - c) + (b - d) \cdot j$

Példa:

$$z_1 = 3 + 2 \cdot j$$
$$z_2 = 5 + 6 \cdot j$$
$$z_1 - z_2 = -2 - 4 \cdot j$$

Szorzás

Algebrai alakban: $z_1 \cdot z_2 = (a + b \cdot j) \cdot (c + d \cdot j) = (a \cdot c - b \cdot d) + (b \cdot c + a \cdot d) \cdot j$

Trigonometrikus alakban: $z_1 \cdot z_2 = r_1 \cdot r_2 \cdot (cos(\phi_1 + \phi_2)) + j \cdot sin(\phi_1 + \phi_2)$

Példa:

$$z_1 = 3 + 2 \cdot j = 3{,}605 \cdot (cos\,33{,}69 + j \cdot sin\,33{,}69)$$
$$z_2 = 5 + 6 \cdot j = 7{,}810 \cdot (cos\,50{,}19 + j \cdot sin\,50{,}19)$$
$$z_1 \cdot z_2 = 3 + 28 \cdot j = 28{,}15 \cdot (cos\,83{,}88 + j \cdot sin\,83{,}88)$$

Osztás

Algebrai alakban: $\frac{z_1}{z_2} = \frac{z_1 \cdot - z_2}{z_2 \cdot - z_2} = \frac{(a + b \cdot j) \cdot (c - d \cdot j)}{(c + d \cdot j) \cdot (c - d \cdot j)} = (\frac{a \cdot c + b \cdot d}{c^2 + d^2}) + (\frac{b \cdot c - a \cdot d}{c^2 + d^2}) \cdot j$

Trigonometrikus alakban: A komplex szám reciprokának ismeretében az osztás visszavezethető szorzásra:

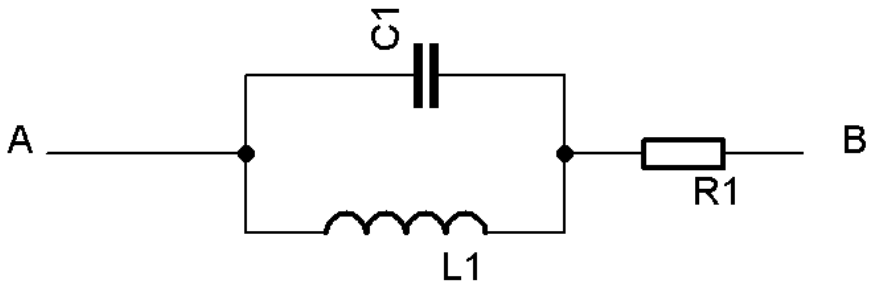
$$\frac{1}{z} = \frac{-z \cdot 1}{r^2} \rightarrow \frac{z_1}{z_2} = \frac{z_1 \cdot - z_2 \cdot 1}{r_2^2} \rightarrow \frac{z_1}{z_2} = \frac{r_1}{r_2} \cdot (cos(\phi_1 - \phi_2) + j \cdot sin(\phi_1 - \phi_2))$$

Példa:

$$z_1 = 3 + 2 \cdot j = 3{,}605 \cdot (cos\,33{,}69 + j \cdot sin\,33{,}69)$$
$$z_2 = 5 + 6 \cdot j = 7{,}810 \cdot (cos\,50{,}19 + j \cdot sin\,50{,}19)$$
$$\frac{z_1}{z_2} = 0{,}443 - 0{,}131 \cdot j = 0{,}462 \cdot (cos - 16{,}5 + j \cdot sin - 16{,}5)$$

Komplex számok alkalmazása impedancia számításban

A vegyesen kapcsolt tekercsek, kondenzátorok és ellenállások eredő impedanciája hatékonyan csak komplex számok segítségével számolható ki. A megoldás menetét egy példán keresztül szemléltetni egyszerű, így az alábbi ábrán látható kapcsolás eredő impedanciáját fogjuk kiszámítani.



A számítások könnyebben elvégezhetőek, ha a számológépünk tud komplex számokkal számolni. A legtöbb mai tudományos számológép erre képes, azonban gyártónként és típusonként is igen nagy eltérések vannak a használatban, így a számológép kézikönyvét érdemes átlapozni a funkció használata előtt.

A példa kapcsolás impedanciájának levezetéséhez szükséges adatok:

$$f = 50\,Hz$$
$$R = 100\,\Omega$$
$$L = 0{,}5\,H$$
$$C = 10\,\mu F$$

A tekercs és kondenzátor látszólagos ellenállását komplex formában az alábbi képletek segítségével tudjuk meghatározni:

$$X_L = j \cdot \omega \cdot L$$
$$X_C = \frac{1}{\omega \cdot C \cdot j} = \frac{-j \cdot 1}{\omega \cdot C}$$

A megoldás első lépéseként ki kell számolni a látszólagos ellenállásokat. Mivel ezekkel később osztani és szorozni is kell majd érdemes átírni trigonometrikus alakba is őket:

$$\omega = 2 \cdot \pi \cdot f = 314{,}15$$
$$x_L = 157{,}075 \cdot j = 157{,}075 \cdot (cos\,90 + sin\,90 \cdot j)$$
$$X_C = -138{,}319 \cdot j = 138{,}319 \cdot (cos\,-90 + sin\,-90 \cdot j)$$

A kapcsolásban a tekercs és a kondezátor párhuzamosan van kapcsolva, ezért az eredő impedanciájuk meghatározásához replusz műveletet kell végezni a kettő látszólagos ellenállásán.

$$X_L \cdot X_C = 157{,}075 \cdot 138{,}319 \cdot (cos(90 + (-90)) + sin(90 + (-90))) = 21726{,}457 \cdot (cos\,0 + sin\,0)$$
$$X_L + X_C = 157{,}075 \cdot j + (-138{,}319 \cdot j) = 18{,}756 \cdot j = 18{,}756 \cdot (cos\,90 + sin\,90 \cdot j)$$
$$X_L \times X_C = \frac{X_L \cdot X_C}{X_L + X_C} = \frac{21726{,}457}{18{,}756} \cdot (cos(0 - 90) + sin(0 - 90) \cdot j) = 1158{,}373 \cdot (cos\,-90 + sin\,-90 \cdot j)$$
$$X_L \times X_C = -1158{,}373 \cdot j$$

A kör teljes impedanciája:

$$Z = (X_L + X_C) + R = 100 - 1158{,}373 \cdot j \quad |Z| = 1162{,}6814\,\Omega \quad cos\,\phi = 0{,}086$$

Feszültség, áramerősség, ellenállás mérése

Minden elektronikai szerelés kapcsán jól jön, ha az embernek van egy multimétere. A multiméter egy univerzális mérőeszköz, ami több villamosmennyiség mérésére képes. A legegyszerűbbek feszültséget, áramerősséget és ellenállást tudnak mérni, a fejlettebbek tekercs induktivitást, kondenzátor kapacitást és sok egyéb mást is képesek mérni.

A műszerek felépítés szempontjából lehetnek analóg és digitális műszerek. Az analóg műszerek mára szinte kimentek a divatból, jóval drágábban szerezhetőek be a digitális műszerekhez képest és jobban is kell rájuk vigyázni a digitális műszerekhez viszonyítva. Az analóg műszerek esetén nagyon ügyelni kell a helyes bekötésre (pozitív és negatív ág nem felcserélhető!) és a mozgó alkatrészek miatt nagyon óvni kell őket a fizikai behatásoktól.

A digitális műszerekkel ilyen probléma nincs, mozgó alkatrész nincs bennük és a pozitív-negatív ág is felcserélhető, nem okozza az eszköz meghibásodását.

*Arra azonban ügyelni kell mindkét típusú műszer esetén, hogy ha **árammérő** üzemmódban használjuk, akkor **sorosan** kell bekötni a műszert, ha pedig **feszültségmérő/ellenállásmérő** üzemmódban akkor **párhuzamosan** kell bekötni őket. A nem megfelelő bekötés az eszköz meghibásodását okozza.*

A középső tárcsa üzemmódváltásra szolgál. Feszültség és árammérésből van egyen-és váltakozó típusú. A műszer típusától függően beszélhetünk automatikus méréshatár beállítós típusról és manuális méréshatár beállítós típusról. A fenti képen látható műszer automatikusan állítja a méréshatárt. Manuális típusok esetén ezt az üzemmódváltó tárcsával állítjuk be.

Amennyiben manuális méréshatár állítást igénylő műszerrel dolgozunk, akkor minden esetben a legnagyobb méréshatárról indulva kezdjük el a mérést és addig haladjunk a legkisebb felé, amíg lehetséges a mért mennyiség függvényében a méréshatár csökkentése. Kisebb méréshatáron általában pontosabb értékeket kapunk.

A fenti képen is látható műszernek 4db kivezetése van. Feszültség és ellenállás méréshez a COM jelű csatlakozóba kell csatlakoztatni a fekete mérővezetéket és a V/Ω jelűbe a piros mérővezetéket és korábban említett módon párhuzamosan bekötni a műszert a mérendő dologhoz képest.

Áramerősség mérés esetén két lehetőségünk is van. Szintén a fekete mérővezetéket a COM jelű csatlakozóba kell csatlakoztatni, a piros mérővezetéket csatlakoztathatjuk két helyre is. Az, hogy melyik helyre kell csatlakoztatni, a méréshatártól függ. A képen szereplő műszeren (és kb mindegyik műszeren) a bal oldali A jelű csatlakozó a viszonylag nagy áramerősségek mérőpontja. Legtöbb esetben ezt a csatlakozót kell használni méréskor. A COM csatlakozó bal oldalán megtalálható csatlakozó a mA mérésre szolgáló bemenet. Mindkét esetben a maximálisan mérhető áramerősség fel van tüntetve a csatlakozó alatt, illetve az is, hogy biztosított-e a bemenet, vagy nem. Jelen műszer mindkét bemenete biztosított, ami azt jelenti, hogy a műszerben belül található egy olvadó biztosíték, ami a megengedettnél nagyobb terhelés esetén megszakítja az áramkört és nem engedi a műszert károsodni.

Szinte minden műszer rendelkezik dióda vizsgáló üzemmóddal is, amelyet egy dióda jellel jelölnek. Ezen üzemmódban a műszer képes meghatározni egy dióda nyitófeszültségét. Ez különösen hasznos tud lenni, ha LED-ekkel dolgozunk.

Továbbá szinte minden műszer rendelkezik szakadásvizsgáló üzemmóddal is. Ez arra jó, hogy két pont között ellenőrizzük, van-e összeköttetés. Ez az üzemmód gyakran a dióda vizsgálattal közösen egy üzemmódban van, vagy a legkisebb ellenállás méréshatáron van. Ha a két pont között kapcsolat van, akkor a műszer sípolni kezd és a két pont közötti ellenállást írja ki.

Amennyiben nagy a két pont közötti ellenállás, akkor célszerű ellenállásmérő üzemmódban megnézni a két pont közötti ellenállást szakadásvizsgálat szempontjából, mivel ha a két pont között az ellenállás nagyobb, mint 200 ohm, akkor a műszer nem fog vezetést jelezni, de ez nem feltétlen jelenti azt, hogy a két pont között szakadás van.

A mérőműszerek egy speciális típusát lakatfogónak szokás nevezni. Ezen műszerek hagyományos multiméterek, amelyek egy áramváltó tekerccsel vannak felszerelve. Az áramváltó tekercs nyitható, könnyen közé helyezhetőek vezetékek, amelyeken az átfolyó áramot meg tudja mérni a műszer anélkül, hogy az áramkört megszakítanánk a mérőműszer beiktatása végett.

Tipikusan villanyszerelők által használt eszköz, mivel viszonylag nagy áramerősségek (több tíz amperes áramerősség) mérésére is alkalmasak ezen műszerek.

Félvezetők és gyártásuk

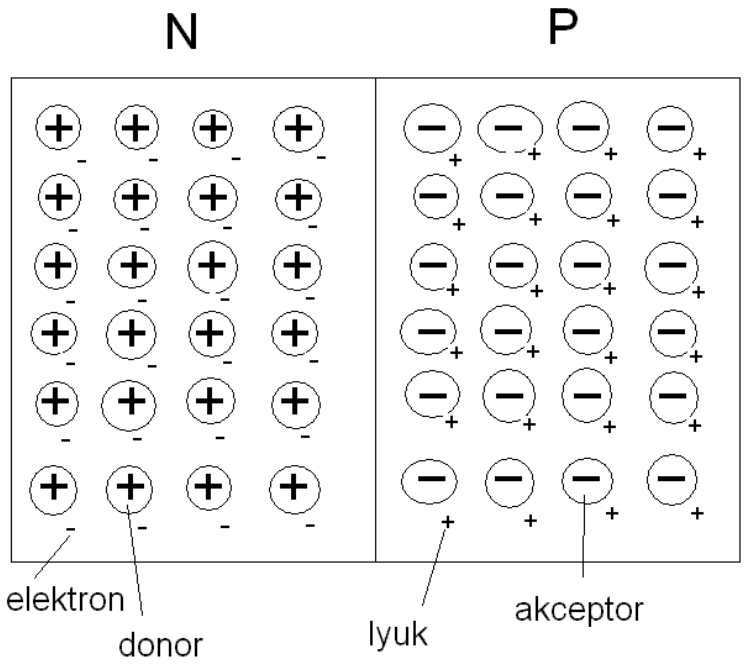
A félvezetők olyan anyagok, amelyek fajlagos ellenállása a vezető és szigetelő anyagok közé esik. Szobahőmérsékleten ezen anyagok nagyon gyengén vezetik az áramot, nagyon alacsony hőmérséklet hatására szigetelőként viselkednek, tehát az ellenállásuk hőfoktól függő.

Tiszta formájukban a félvezető anyagok kevés dologra jók, ezért félvezető áramkörök gyártásánál szennyezett formájukban szokták őket alkalmazni. Kétféle szennyezett félvezetőt különböztetünk meg: N típusút és P típusút.

N típus esetén a tiszta félvezetőt elektrontöbblettel rendelkező anyaggal szennyezik a gyártás során, így ez a típusú félvezető negatív töltésű lesz.

P típus esetén a tiszta félvezetőt elektronhiánnyal rendelkező anyaggal szennyezik, így „lyukak" keletkeznek az anyag szerkezetében, vagyis pozitív töltésűnek tekinthető az anyag.

Az N és P típusú félvezetőket nem külön gyártják, hanem általában egy tiszta félvezető felületén alakítják ki őket különböző eljárásokkal, így alakul ki a két eltérő típusú félvezető találkozásának határán az úgynevezett P-N átmenet. Az átmenet töltések szempontjából semleges, mivel az elektrontöbblettel rendelkező anyag a szabad elektronjait átadja az elektronhiányos anyagrésznek. Ez az egyensúly azonban nem állandó, elektromos áram hatására megváltoztatható.



Dióda

A dióda az egyik legegyszerűbb félvezető alkatrész, amely egyetlen P-N átmenetből áll. A tulajdonsága az, hogy rajta keresztül az áram csak egy irányba tud folyni. Ezen tulajdonsága miatt egyenirányításra és polaritásvédelemre alkalmazzák.

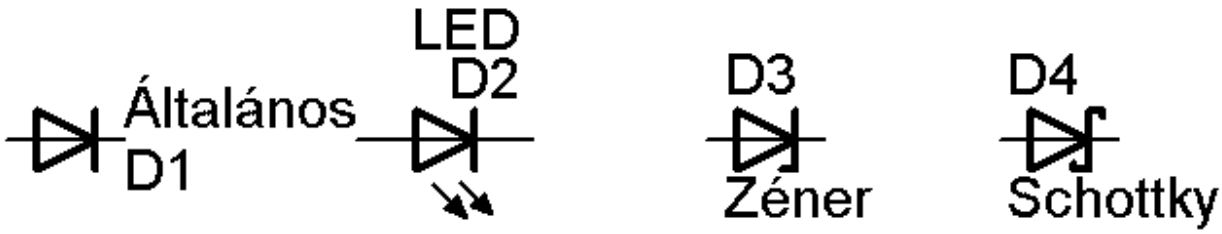
Jellemzője a nyitófeszültség. Ez az a feszültségszint, ami hatására a dióda vezetni kezd, a diódában használt félvezető anyagtól függ, szilícium diódák esetén általában 0,7V körüli. Több különleges típusa is létezik, mint például a LED, Zener dióda és a Schottky dióda.

A LED, magyarosabban fénykibocsájtó dióda olyan dióda, amelynél a P-N átmenet egy speciális szennyező réteggel van beborítva, ennek hatására az fényt bocsájt ki. A kibocsájtott fény hullámhosszát (színét) a szennyező anyag összetétele és a diódalencse is befolyásolja. Ezen diódák jóval nagyobb nyitófeszültséggel rendelkeznek, ami hullámhossztól függő, tipikusan 1,6-3,5V körül mozog.

A Zener dióda feszültség stabilizálásra alkalmas. Speciálisan záró tartományú működésre van kialakítva. A diódák azon tulajdonságát alkalmazzák itt, hogy ha fordítva kötünk be egy diódát, akkor a zárófeszültség (ami a két kivezetés között mérhető) állandó lesz.

A Schottky diódák jellemzője, hogy a nyitófeszültségük alacsonyabb a hagyományos diódáknál, tipikusan 0,3-0,4V körüli. Ebből kifolyólag jóval nagyobb működési sebességre képesek, tipikusan olyan helyeken alkalmazzák, ahol fontos a gyors működési sebesség.

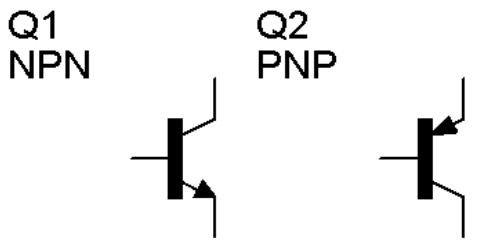
A diódák két kivezetését anódnak és katódnak nevezzük. Az anód a pozitív kivezetés, a katód pedig a negatív kivezetés. A különböző diódák rajzjelei az alábbi ábrán láthatóak.



Tranzisztor, FET

A tranzisztor az angol transfer-resistor (átviteli ellenállás vagy átengedés ellenállás) szavakból alkotott mozaikszó. Alapjaiban alakította át az elektronikai ipart.

Három darab félvezető rétegből áll. Megkülönböztetünk NPN és PNP tranzisztorokat. NPN tranzisztorok esetén 2db N típusú réteg között egy P típusú réteg helyezkedik el. PNP típus esetén pedig 2db P típusú réteg között helyezkedik el egy N típusú. A középső réteg kivezetését bázisnak nevezik, a két szélsőt pedig kollektornak és emitternek. Ezek nem cserélhetőek fel, mivel a félvezető belső kialakítása nem szimmetrikus. Minden esetben az áram a kollektor felől az emitter felé áramlik, a bázisáram függvényében. Ha a bázisáram kicsi, akkor a tranzisztor nem nyit ki, vagyis a bázispont igen nagy ellenállást fog képviselni a kollektor és emitter között. Amennyiben a bázisáram kellően nagy, akkor tranzisztor telítésben van, vagyis a bázis csak egy csekély ellenállást képvisel a kollektor és emitter között. A bázisáram nagysága tranzisztorfüggő. A tranzisztorokat kapcsolóknak és erősítőknek szokták használni.



A FET angol mozaikszó, az angol Field Effect Transistor szavak rövidítése. Ez magyarra fordítva feszültség vezérléses tranzisztort jelent. Ezen tranzisztorok nem áram, hanem feszültség vezéreltek. A három kivezetés neve nem azonos a tranzisztor kivezetéseinek nevével. A vezérlő bemenet neve Gate (kapu), a két szélső láb neve pedig Source (forrás) és Drain (nyelő). Áram a Source felől a Drain felé tud folyni a Gate feszültség függvényében.

Integrált áramkörökben szeretik ezen típusú tranzisztorokat alkalmazni, mivel kisebb fogyasztású integrált áramkörök alakíthatóak ki velük, mint hagyományos tranzisztorokkal.

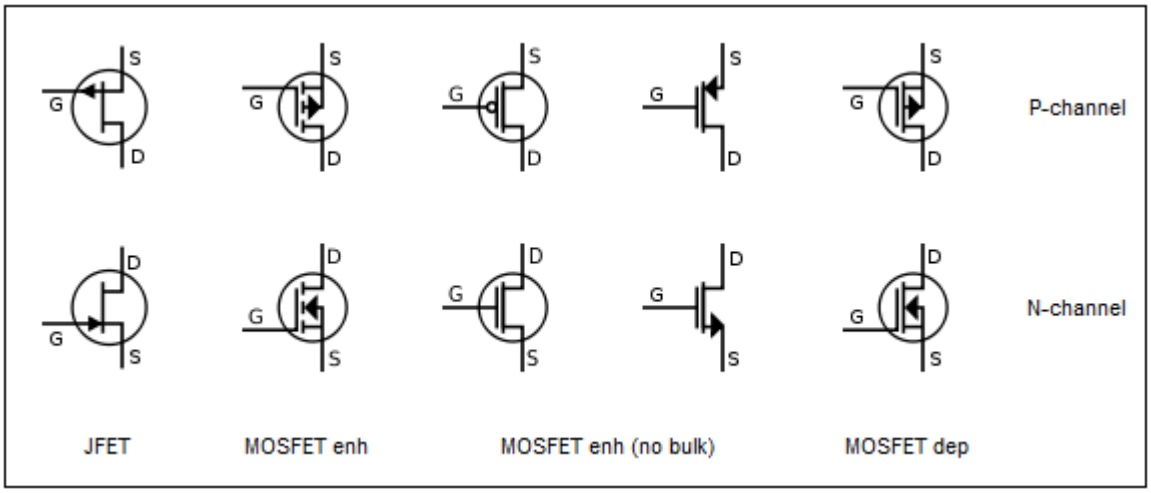
Kapcsolónak is ideálisabb a hagyományos tranzisztornál, mivel telített (teljesen nyitott) állapotában jóval kisebb a belső ellenállása, így nem annyira melegszik nagy teljesítmény mellett sem.

A FET-ek egyetlen hátránya, hogy mivel elektromos töltésre kapcsolnak és nem töltésáramlásra, érzékenyebbek a statikus elektromosságra. Konkrétan, ha egy FET tranzisztor Gate lábát egy hosszú vezetékre kötjük, akkor a légköri statikus töltések felhalmozódása már ki tudja nyitni. Ez érdekes működési módokat tud eredményzeni. Ezért ha FET tranzisztorokkal dolgozunk, akkor gondoskodni kell róla, hogy a Gate bemeneten kikapcsolásnak szánt állapotban ne halmozódhasson fel töltés, ami ki tudná nyitni. Növekményes MOSFET tranzisztorok esetén ez a Gate földelését jelenti.

Gyártástechnológiai és belső kialakítási szempontból igen sok fajta FET létezik. Az egyik legnépszerűbb típus a MOSFET, amiben a MOS előtag angolul a fémoxidos félvezető mivoltára utal. Működési szempontból ez áll a legközelebb a hagyományos tranzisztorhoz.

Szinte minden FET-ből létezik növekményes és kiürítéses típus. A növekményes típusú FET-ek esetén csak akkor tud áram folyni a Source és a Drain között, ha a Gate-re pozitív feszültséget kapcsolunk.

A kiürítéses típusú FET-ek esetén a Source és a Drain között tud áram folyni kikapcsolt állapotban, azonban ha a Gate bemenetre negatív feszültséget kapcsolunk, akkor nem fog folyni áram a két pont között.



IGBT

Az IGBT angol mozaikszó az Insulated Gate Bipolar Transistor szavak rövidítése, ami magyarra fordítva szigetelt kapuval rendelkező bipoláris tranzisztort jelent. A hagyományos bipoláris (mivel kétnemű töltéshordozó kell a működéséhez) és az unipoláris tranzisztor (más néven FET) keresztezéséből alkotott félvezető.

Főként teljesítményelektronikában használják. Jellemzője, hogy hatalmas feszültségeket (több száz volt) és több tíz amper áramerősséget tud gond nélkül kapcsolni. Nagy teljesítményű erősítőkben és kapcsoló üzemű tápegységek esetén használják őket.

Tranzisztorok és FET-ek alkalmazása

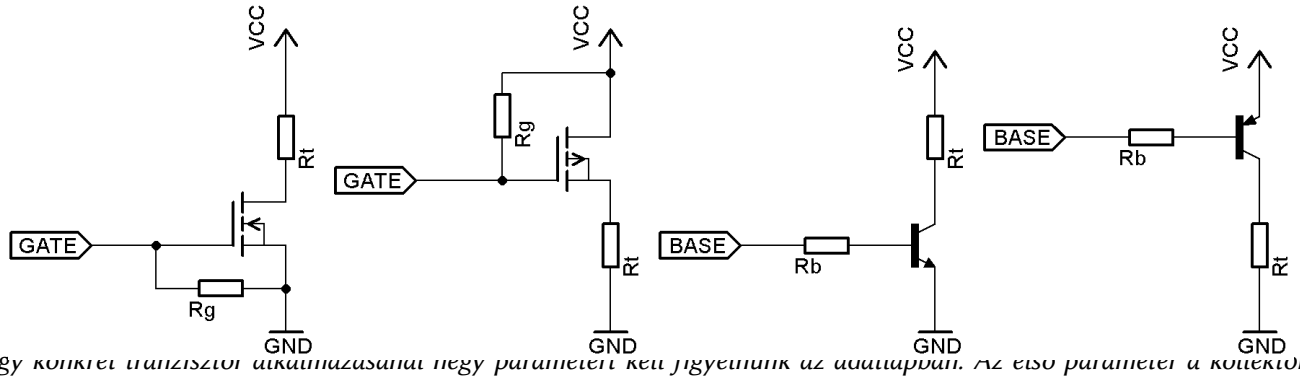
A Tranzisztorok és FET-ek gyakorlati alkalmazásánál kezdőként a legnagyobb problémát az okozza, hogy mi alapján is válasszunk tranzisztort vagy FET-et. A könyv ezen rövid szekciójában ebben kívánok segítséget nyújtani. Az ezen részben tárgyalt ismeretek kapcsoló üzemű használatra vonatkoznak.

Mielőtt belemennénk a dolgokba túlzottan tisztázzuk, hogy vannak NPN és PNP tranzisztorok, FET-ek. Ezek alkalmazása némiképpen eltér. Az NPN tranzisztorokat a kollektor ágban kell terhelni és pozitív bázis feszültség hatására kialakuló bázis árammal kell vezérelni. A FET-ekre szintén ez vonatkozik, megfelelő analógiával: Pozitív gate feszültség és drain ágban terhelés.

A PNP rétegsorrendes eszközöknél negatív Gate feszültség, illetve ennek hatására kialakuló bázis áram szükséges, valamint a terhelést a Source, illetve emitter ágban kell elhelyezni.

FET-ek esetén a tranzisztorhoz analógiában legközelebb álló FET típus a MOSFET. Ezen FET típus alkalmazásánál ügyelni kell arra, hogy van egy nagyon kicsi gate kapacitása a felépítéséből adódóan. Ez a kapacitás problémákat tud okozni, mivel a légköri töltések képesek felhalmozódni és működésbe hozni a kapcsolót. Ezen tulajdonság kiküszöbölése érdekében a gate lábát minden MOSFET-nek egy stabil pontra kell kötni egy viszonylag nagyméretű ellenálláson keresztül.

NPN típus esetén a stabil pont a föld pont lesz, míg PNP típus esetén a tápfeszültség. Az ellenállás nagysága és a kapcsolási feszültség fordítottan arányos. Minél kisebb ellenállást alkalmazunk, annál gyorsabban fog kapcsolni. Viszont kis ellenállás esetén indokolatlanul sok áram folyhat keresztül a lehúzó vagy felhúzó ellenálláson keresztül. Ezért minden esetben érdemes utánaszámolni, hogy a gate meghajtására szolgáló feszültségforrás mennyi áramot képes szolgáltatni.

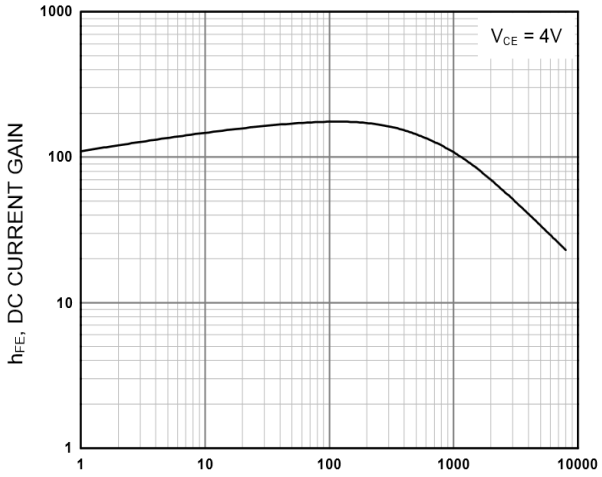


Egy konkrét tranzisztor alkalmazásánál négy paraméterét kell figyelnünk az adatlapban. Az első paraméter a kollektor-emitter feszültség, ami tipikusan V_{CEO} jelöléssel szerepel az adatlapban. Ez határozza meg, hogy mekkora feszültséget képes kapcsolni a tranzisztor. NPN tranzisztor esetén pozitív szám, PNP esetén negatív.

A második paraméter amit figyelembe kell vennünk az a emitter-bázis feszültség. Ez adja meg azt, hogy a tranzisztor bázisára mekkora feszültséget kell adni, hogy a feszültség hatására létrejövő áram kinyissa az emitterhez viszonyítva. NPN tranzisztorok esetén ez egy pozitív szám, PNP tranzisztorok esetén negatív szám. Az adatlapban V_{EBO} jelöléssel szokták szerepeltetni.

A harmadik paraméter amit figyelembe kell vennünk az, hogy mennyi árammal is lehet megterhelni a tranzisztorunkat. Ez a kollektor áram és I_c *jelöléssel kell keresni az adatlapban. NPN rétegsorrend esetén pozitív, PNP rétegsorrend esetén negatív szám. Ezen paraméter esetén érdemes megjegyezni, hogy ez egy olyan számérték, amely kapcsolását még biztonsággal képes elvégezni a tranzisztor, tehát ha ekkora áram kerül rá, akkor nem megy tönkre. Éppen ezért szokás megjelölni egy abszolút maximumot is, amit még éppen elbír a tranzisztor, de nem folyamatos üzemben. Ezt gyártó függően másképpen szokták jelölni vagy* I_{CP} *vagy* I_{CM} *jelöléssel.*

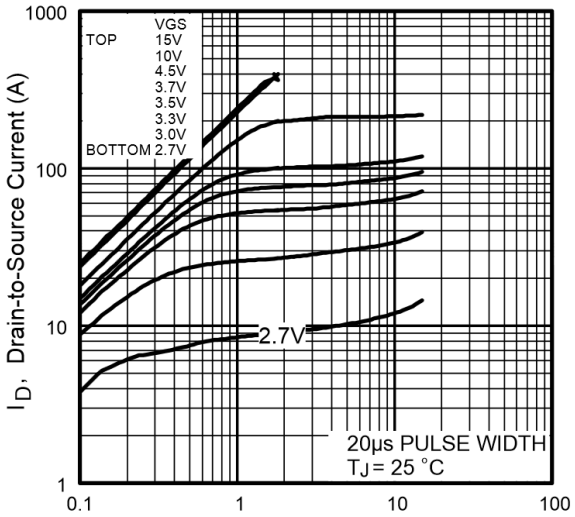
A negyedik legfontosabb paraméter az erősítési tényező, ami h_{FE} *jelöléssel szerepel az adatlapban. Ennek segítségével tudjuk beállítani a tranzisztorunk munkapontját. Ezen paraméter értéke függ a kollektor áramtól, így minden adatlapban található egy vagy több grafikon, amiről leolvasható ezen paraméter változása.*



Az egyszerűség kedvéért esetben meghatározva. Ezek a Legjobb eset, egy tipikus érték és egy legrosszabb eset. Kapcsoló üzem esetén célunk az lenne, hogy maximalizáljuk az erősítést, vagyis adott terhelés mellett fixen kinyisson a tranzisztorunk. Ezért a h_{FE} *paraméter minimális értékével érdemes számolni.*

A bázis áram nagysága a $I_C = h_{FE} * I_B$ *képlet átrendezéséből adódóan a* $\frac{I_C}{h_{FE}} = I_B$ *képlet segítségével számolható ki. Mikrovezérlők esetén olyan tranzisztorokat kell alkalmazni, amelyek kellően nagy erősítési tényezővel rendelkeznek, hogy a vezérlő kellő áramot tudjon szolgáltatni.*

MOSFET-ek esetén hasonló paraméterek adottak. A maximális kapcsolható feszültség V_{DSS} jelöléssel szerepel, a maximálisan kapcsolható áram I_{DS} jelöléssel. A legfontosabb paraméter a gate feszültség, amit a source kivezetéshez képest szoktak mérni. Ennek a maximuma szokott megadva lenni az adatlapban és a könnyebb érthetőség kedvéért egy vagy több grafikon amin szerepeltetve van a gate feszültség függvényében a kapcsolt feszültség és átfolyó áram. Ezen diagram segítségével határozható meg, hogy mekkora feszültség szükséges ahhoz, hogy a FET teljesen kinyisson.



A FET-ek alkalmazása mik kell áram a vezérléshez, vagyis nem terheli a vezérlőt, valamint sokkal kisebb ellenállással rendelkezik egy FET kinyitott állapotban, mint egy tranzisztor. Ez pedig azt jelenti, hogy egy FET kevésbé melegedik nagy feszültség hatására átfolyó áram esetén, mint egy tranzisztor.

Ezen jó tulajdonságok mellé felróható hátrány, hogy a tranzisztorok lassabban kapcsolnak köszönhetően a gate kapacitásnak. Ez főleg olyan helyeken okoz problémát, mikor nagyon gyorsan kell kapcsolgatni egy jelet. Tipikusan pár kHz tartományban még használhatóak, de elképzelhető, hogy a kimeneti jel torzulni fog. Ez főként PWM moduláció esetén okozhat problémákat.

Műveleti erősítők és alapkapcsolásaik

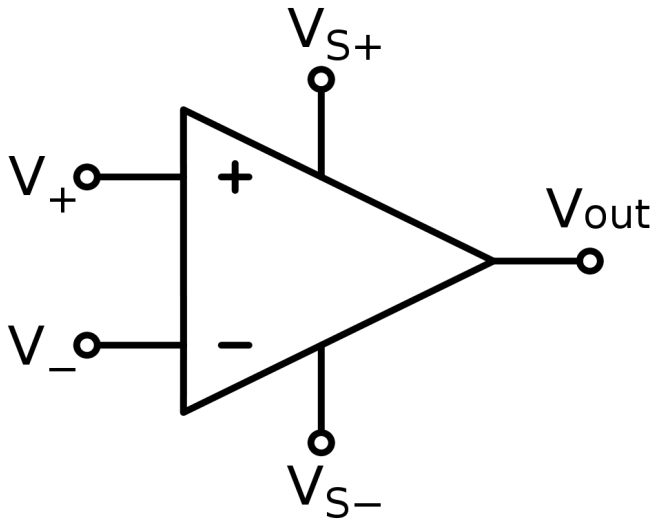
A műveleti erősítők analóg áramkörök, azonban széles körben elterjedtek és mikrovezérlős rendszerfejles során bizonyos szenzortípusok illesztésénél alkalmazzák őket. De mi az a műveleti erősítő? A műveleti erősítő egy pár tucat tranzisztorból álló integrált áramkör. Széles körben elterjedt a rengeteg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

30.



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

15. 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.

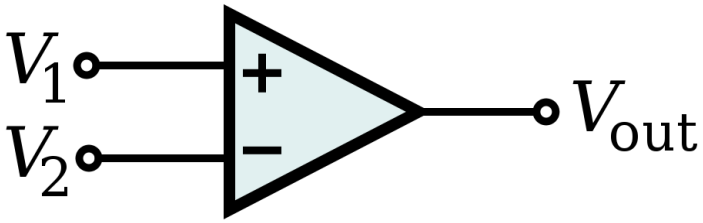
Műveleti erősítős kapcsolások levezetéséhez, megértéséhez az alábbi három szabályt kell tudni:

- *Differenciált bemenetből adódóan visszacsatolás esetén azért fog küzdeni az erősítő, hogy a két bemenet azonos potenciál szinten (feszültségen) legyen.*
- *Visszacsatolás nélkül a kimeneten az üres járási erősítés érvényesül, ami 100 000-es nagyságrendű*
- *A bemenetek végtelen nagy ellenállásából adódóan a műveleti erősítőbe áram nem folyik be.*

Alapkapcsolások

A műveleti erősítőknek számos kapcsolása létezik, az alábbiakban csak a legfontosabb, gyakran alkalmazott alapkapcsolásokkal ismerkedünk meg.

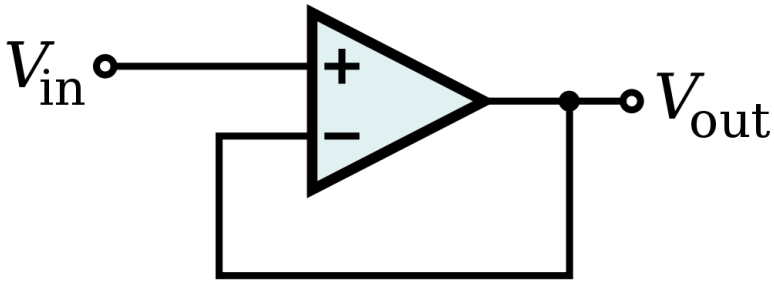
Komparátor



$$U_{ki} = \begin{pmatrix} V_S + ha\ V_1 > V_2, \\ V_S - ha\ V_1 < V_2, \\ 0\ ha\ V_1 = V_2 \end{pmatrix}$$

A kapcsolás a műveleti erősítő differenciált bemeneteiből adódik. Ebben a kapcsolásban a két bemenet feszültség különbsége erősítve jelenik meg a kimeneten. Mivel az áramkör nincs visszacsatolva az üres járási erősítés nagysága miatt a kimeneten vagy a pozitív vagy a negatív tápfeszültség jelenik meg.

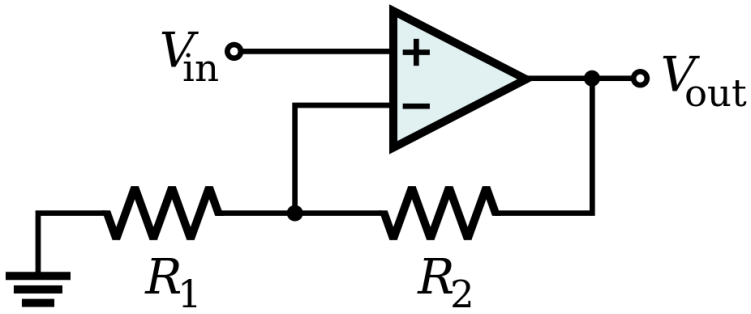
Követő erősítő



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

Feszültség követő üzemmódban a bemenetre adott feszültség jelenik meg a kimeneten. Mivel ez módosítás nélkül van visszacsatolva ezért az első szabályból adódóan ez a feszültség marad a kimeneten.

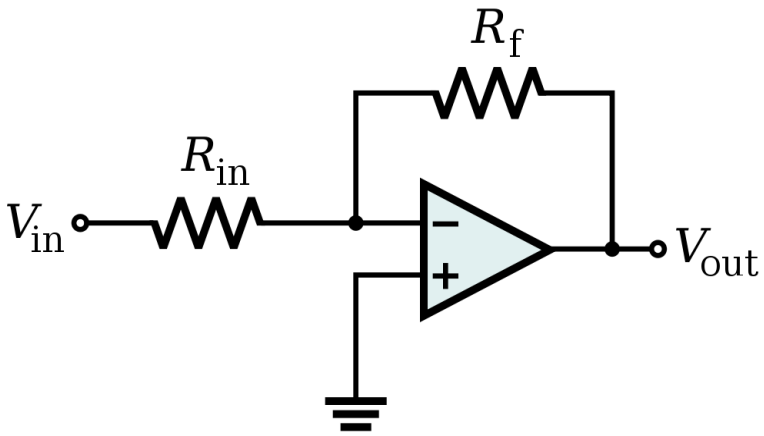
Nem invertáló alapkapcsolás



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

A nem invertáló kapcsolásban a kimeneti jel egy feszültségosztón keresztül van visszacsatolva, így a kimenet a feszültség osztó által meghatározott arány reciprokával fog növekedni az eredeti feszültséghez képest.

Invertáló alapkapcsolás

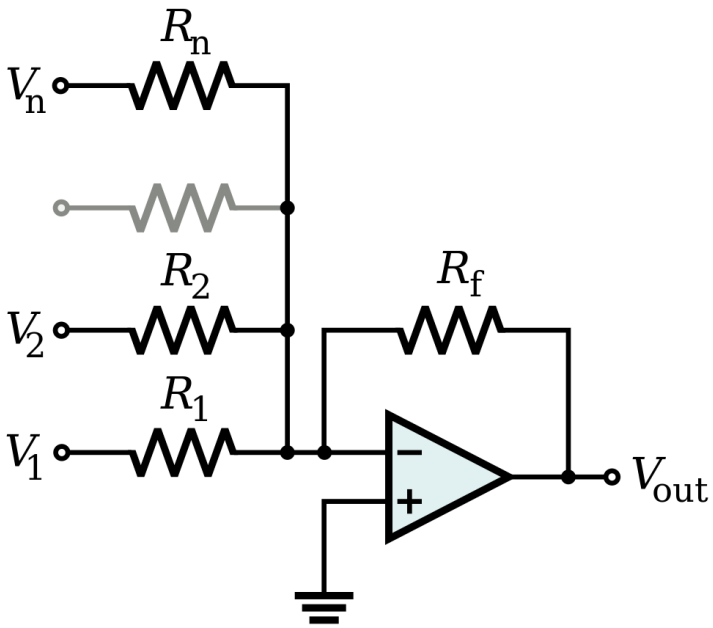


$$U_{ki} = \frac{-R_f}{R_{in}} \cdot U_{be}$$

Invertáló kapcsolás esetén a nem invertáló bemenet földeléséből következően az invertáló bemenet virtuális földpont lesz. (A bemenetek azonos szinten tartása miatt)

Mivel áram az erősítőbe nem folyik be, ezért az áram Rin és Rf ellenállásokon keresztül fog folyni. Rf ellenálláson az átfolyó áram feszültségesést hoz létre. Abból adódóan, hogy Rf egyik oldala földpontként viselkedik, a feszültségesés csak a bemeneti feszültség polaritásával ellentétes lehet.

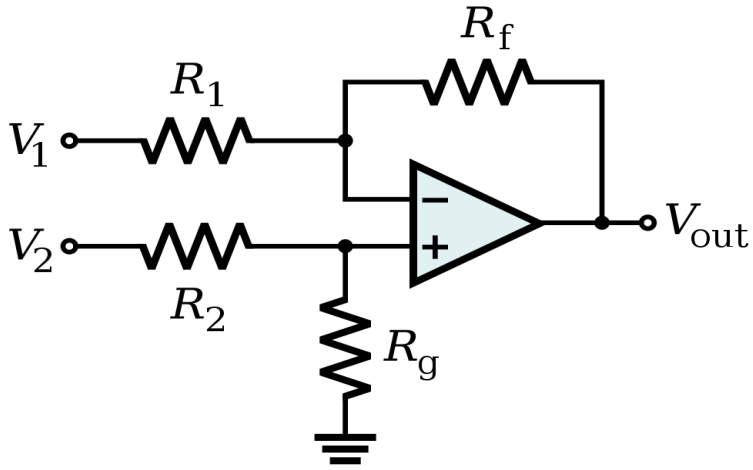
Összegző erősítő



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

Az invertáló erősítőből könnyen levezethető. A sorszámozott ellenállások a feszültségosztó súlyozását változtatják meg, így a különböző jelek különböző súlyozással kerülnek erősítésre és összegzésre.

Differenciál erősítő, kivonó áramkör

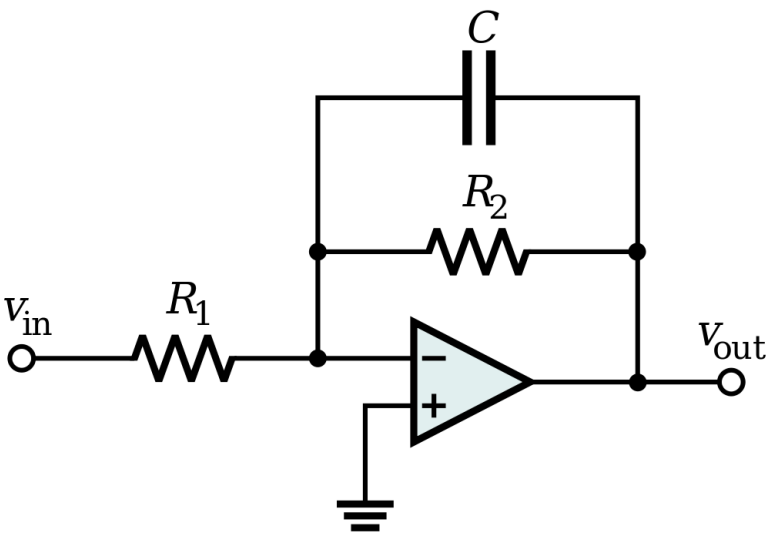


$$U_{ki} = \frac{(R_f + R_1)\cdot R_g}{(R_g + R_2)\cdot R_1}\cdot U_2 - \frac{R_f}{R_1}\cdot U_1$$

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

A differenciál erősítő esetén a műveleti erősítő azon tulajdonsága van kihasználva, hogy a kimeneteket megpróbálja azonos szinten tartani. Mindkét bemenetre a jel egy feszültségosztón kerül és a két jel különbsége kerül erősítésre.

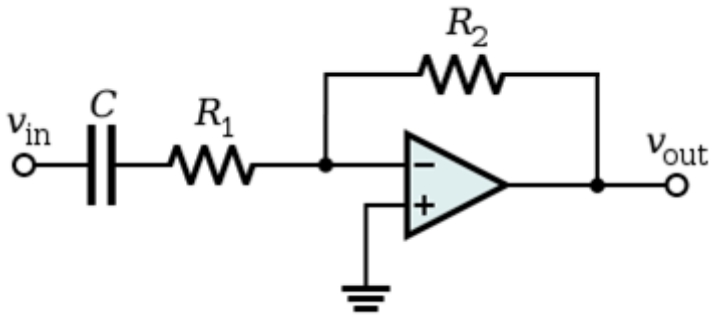
Aktív alul áteresztő szűrő



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

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

Aktív felül áteresztő szűrő



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

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

Alul és felül áteresztő szűrő kapcsolás esetén a kimeneti feszültség erősítés az invertáló alapkapcsolásnál bemutatott képletek segítségével határozhatók meg. A szűrt jel ellentétes fázishelyzetű lesz a bementi jelhez képest.

7. Digitális technikai alapismeretek

Digitális technikai alapismeretekből ebben a fejezetben csak a minimálisan szükséges ismeretek lesznek közölve. Ennek az oka az, hogy körülbelül egy olyan hosszú könyvet lehetne írni a digitális technikáról, mint ez a könyv, úgy, hogy a mikrovezérlőkről és a mikroszámítógépekről egy szót sem ejtenénk.

Számrendszerek, átváltások és számábrázolási módok

Az összes számítógép kettes számrendszerben dolgozik, tehát az összes adatunk 0-k és 1-esek sorozataként van tárolva. Ezt más néven nevezhetnénk bitek sorozatának is. Egy bit 0 vagy 1 értéket vehet fel. 8 bit alkot egy byte-ot.

Átváltás számrendszerek között

Egy tízes számrendszerbeli számból 2-es számrendszerbeli számot az alábbi algoritmus alapján tudunk csinálni, kettővel osztással papíron:

1. *Hosszú függőleges vonal húzása.*
2. *Szám 10-es alakban felírása a vonal bal oldalára.*
3. *2-vel osztás, majd a maradék (0 vagy 1) vonal jobb oldalára írása a szám mellé.*
4. *A kapott eredmény leírása a szám alá.*
5. *Ismétlés, míg az utolsó osztandó szám nem 1.*
6. *Utolsó egyes átírása a vonal bal oldalára.*
7. *Vonal bal oldalán található szám az átváltás eredménye. A legalsó számjegy (1) a legnagyobb helyi értéken lévő számjegy.*

Az átváltandó és az átváltott szám alatt alsó indexben szokás jelölni a számrendszert. A fenti algoritmust követve az 1337 tízes számrendszerben ábrázolt szám bináris alakja a következő lesz:

1337	1
668	0
334	0
167	1
83	1
41	1
20	0
10	0
5	1
2	0
1	1

$$1337_{10} \equiv 10100111001_2$$

Visszaváltani a számot úgy tudjuk, hogy a bináris számjegyek fölé jobbról, a legkisebb helyi értéken lévőtől felírjuk kettő hatványait nullától kezdve. Ahol a bináris számban egyes értékek találhatóak, összegezzük a nekik megfelelő hatványértékeket.

$$1337_{10} = 1 \cdot 2^{10} + 0 \cdot 2^9 + 1 \cdot 2^8 + 0 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$1337_{10} = 1024 + 0 + 256 + 0 + 0 + 32 + 16 + 8 + 0 + 0 + 1$$

Nagy mértékben megkönnyíti az átváltást, ha az átváltó tisztában van kettő hatványaival legalább tízes kitevőig.

Hatvány kitevő	0	1	2	3	4	5	6	7	8	9

Érték	1	2	4	8	16	32	64	128	256	512
Hatvány kitevő	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Érték	1024	2048	4096	8192	16 384	32 768	65 536	131 072	262 144	524 288

16. Táblázat: Kettő hatványai

A kettes számrendszerrel az a probléma, hogy könnyű belekavarodni a sok nullásba és egyesbe. Emiatt informatikában és digitális technikában is szokás bináris adatot kifejezni 8-as vagy 16-os számrendszerben. A 8-as számrendszerbe történő átváltás úgy történik, hogy a számot a legkisebb helyi értéken lévő bittől kezdve három bites csoportokra bontjuk. Amennyiben a szám végén nem jönne ki a 3-as csoport, akkor nullákkal kiegészítjük. A kapott hármas csoportokat pedig egyszerűen átváltjuk a tízes számrendszernél ismertetett visszaváltási módszerrel. A kapott számjegyek nyolcas számrendszerben reprezentálják a számot.

Tizenhatos számrendszernél hasonló az eljárás, csak ott négyes csoportokra bontunk és azokat váltjuk át. Amennyiben a szám értéke 10 vagy nagyobb, akkor A és F közötti betűkkel jelöljük őket. A következő példán az 1337 tízes számrendszerben ábrázolt szám bináris, oktális és hexadecimális reprezentációja látható:

0	1	0	1	0	0	1	1	1	0	0	1
2			4			7			1		
5			3			9					

17. táblázat: Az 1337-es szám bináris, oktális és hexadecimális számrendszerben

Egész számok ábrázolása előjellel

Pozitív egész számokat átváltani gond nélkül tudunk. Azonban, ha a számunk negatív szám, akkor már trükköznünk kell., továbbá ábrázolhatunk egy számot úgy, hogy a legnagyobb helyi értéken lévő bitet a bal oldalinak tekintjük, vagy akár lehet a legnagyobb helyi értéken lévő szám a jobb oldalon is. Ezt a két ábrázolási módot Big Endian és Low Endian kódolásnak nevezik. Számítógépek esetén mindkettő használatos, CPU-tól függ. Az Intel x86 és x64 kompatibilis processzorok mindegyike a Low Endian sémát követi, míg a telefonokban használatos ARM és egyéb RISC processzorok meg a Big Endian elvet követik. A legnagyobb helyi értéken lévő bitet angolul Most Significant bitnek szokás nevezni (rövidítve MSB), míg a legkisebb helyi értéken lévőt meg Least Significant bitnek. (rövidítve LSB)

Negatív számok bináris számrendszerben csak úgy ábrázolhatóak, hogy egy bitet előjelnek használunk fel. Ez a bit a legnagyobb helyi értéken lévő bit szokott lenni. Ha ennek az értéke nulla, akkor a szám pozitív szám, ha az értéke egy, akkor a szám negatív.

További módosítás a negatív számoknál a komplemens kódolás alkalmazása. Ebből megkülönböztetünk egyest és kettest.

A komplemens szó kiegészítőt jelent. Jelen esetben azt a számot, amely kiegészíti az eredeti számunkat az n biten ábrázolható legnagyobb számra. Gyakorlatban ezt úgy szokás elvégezni, hogy a szám összes bitjét negáljuk. (lásd Boole algebra)

Egy adott n biten felírható szám kettes komplemens kódja alatt azt a számot értjük, amely a kérdéses számot kiegészíti az n biten felírható legnagyobb számnál eggyel nagyobb számra. Ennek az előállítása úgy történik, hogy a szám összes bitjét negáljuk, majd hozzáadunk egyet. Ennél a módszernél azonban létezik egy sokkal egyszerűbb gyakorlati megoldás is. Az eljárás a következő: balról jobbra haladva az első egyesig a szám összes bitjét változatlanul leírjuk az első egyessel együtt, majd a többi bitet negáljuk.

A komplemens kódokat azért találták ki, hogy egyszerűbb legyen az aritmetikai egységek felépítése. Ha egyes komplemens kódban adunk össze két számot, akkor előjelhelyes eredményt fogunk kapni. A kettes komplemens kódot azért alkalmazzák, mert így egyszerűbb a túlcsordulások (bit átvitelek) kezelése.

Példaképpen a –56 a következőképpen ábrázolható 8 biten, ha a legnagyobb helyi értéken lévő bit előjelként használt:

A 10-es számrendszerbeli 56 bináris alakban 7 biten: $56_{10}=0111000_2$ A szám egyes komplemens alakja: 1000111 , kettes komplemens alakja: 1001000 A szám végleges alakja 8 biten: $-56_{10}=11001000_2$

Fixpontos tört számok ábrázolása

A fixpontos számábrázolás lényege az, hogy előre meghatározzuk, hány bitet szeretnénk használni egész számok és a törtrész leírására. Ennek a legnagyobb hátránya az, hogy a pontosság rögzített. A szám egész részét a fentebb említett módon ábrázoljuk. A kettedes pont utáni részt pedig úgy kapjuk meg, hogy a szám tizedespont utáni részét elkezdjük szorozgatni kettővel, majd a szorzás eredményeképpen kapott szám egész részét (0 vagy 1) használjuk fel a tizedesjegyek ábrázolására. Ezt a folyamatot addig ismételjük, míg 1-et nem kapunk. Ez ritkán fog előfordulni, mivel nem minden szám ábrázolható ilyen módon. Sok esetben hamarabb fogunk kifutni a tizedes részre fenntartott bitek számából.

A kettedes pont utáni szám kiolvasása nem alulról felfelé történik, hanem fentről lefelé. A folyamat szemléetetésképpen a 235,6875 tízes számrendszerbeli számot ábrázoljuk binárisan. A szám egész része a korábban bemutatott eljárás alapján a következő lesz binárisan:

$$235_{10}=11101011$$

A tört rész átváltása a következő:

$$2\cdot 0{,}6875=1{,}375\Rightarrow 1$$
$$2\cdot 0{,}375=0{,}75\Rightarrow 0$$
$$2\cdot 0{,}75=1{,}5\Rightarrow 1$$
$$2\cdot 0{,}5=1\Rightarrow 1$$

Tehát a szám kettedes pont utáni része: $0{,}6875_{10}=0.1011_2$ Az átváltott szám pedig:

$$235{,}6875_{10}=11101011.1011_2$$

Lebegőpontos számok ábrázolása

A lebegőpontos számok ábrázolásának módja az IEEE754-es szabvány alatt lett definiálva 1985-ben. A szabványt 2008-ban felülvizsgálták és némileg kibővítették. A mostani használatos szabvány hivatalos neve az IEEE754-2008.

A fix bithosszúságú lebegőpontos számábrázolás matematikai alapja az, hogy minden bináris szám felírható a következő alakban:

$$N=\pm M\cdot 2^{\pm k}$$

A képletben M az előjeles mantisszát jelöli, k pedig az előjeles karakterisztikát. Így lényegében a számunk két szám ábrázolásából áll elő. Ezen módszer előnye, hogy a tizedespont vándorolhat és általa a pontosság is változó tud lenni. Ez néhány esetben pozitívum, míg néhányban hátrány. Ezen ábrázolási móddal n bit esetén 2^n-4 lebegőpontos számot tudunk ábrázolni pontosan. Mivel végtelen sok lebegőpontos szám létezik, ezért előbb-utóbb kerekítési hibába fogunk botlani.

A különböző IEEE754-2008 szabványú számformátumok között a különbség nem csak a legnagyobb és legkisebb ábrázolható számban van, hanem a pontosságban is. Minél nagyobb méretű számokkal dolgozunk, annál kisebb a kerekítési hiba valószínűsége. A definiált formátumok a következőek:

Típus	Karakterisztika	Mantissza	Σ bitszám	Karakterisztika eltolása	Pontosság bitekben	Pontosság tizedesjegyekben
Half	5	10	16	15	11	~3,3
Single	8	23	32	127	24	~7,2
Double	11	52	64	1023	53	~15,9
Double Ext.	15	64	80	16 383	64	~19,2

Quad	*15*	*112*	*127*	*16 383*	*113*	*~34,0*

18. táblázat. Az IEEE754–2008 szabvány által definiált számformátumok fontosabb adatai

A definiált formátumok mindegyikének bitsorrendje a következő: előjelbit (1), karakterisztika bitek (5,8,11 vagy 15), mantissza bitek (10, 23, 52, 64, vagy 112).

A szabvány definiál négy darab számértéket, amelyek kiesnek az ábrázolható számtartományból. Ezek a speciális számok:

- *Nulla*
  Nulla ábrázolására fenntartott kód
- *Pozitív végtelen*
  Akkor keletkezik, ha a szám nem negatív és túlmutat a formátum ábrázolási tartományán
- *Negatív végtelen*
  Akkor keletkezik, ha a szám negatív és kisebb a legkisebb ábrázolható számnál a kiválasztott formátumban
- *NaN-nem szám*
  Akkor keletkezik, ha matematikailag értelmetlen műveletet próbálunk végrehajtani a számon. Pl: végtelennel osztás, nullával osztás, stb...

Az átváltási folyamat a következő lépésekből áll:

1. *Átváltandó szám abszolút értékének átváltása fixpontos tört átváltás szerint.*
2. *A bináris pont eltolásával a mantissza törtre normalizálása, a bináris pont eltolásának értéke lesz a k értéke.*
3. *Ábrázolandó mantissza az implicit bit elhagyásával áll elő, az ábrázolandó karakterisztikát el kell tolni a formátumhoz tartozó karakterisztika eltolással (ábrázolandó = k + formátum eltolás).*
4. *Előjelbit meghatározása. Egyes negatív számok esetén, nulla pozitív számok esetén. Amennyiben a karakterisztika vagy a mantissza nem éri el a formátumban definiált hosszúságot, akkor ki kell őket egészíteni nullák írásával.*

A folyamat szemléltetésének érdekében a –37,25 átváltása Single típusra átváltáskor a kiinduló bináris szám 100 101.01 lesz. Ebből a normalizált mantissza 0.100 10 101 lesz, k értéke pedig 6. Az ábrázolandó mantissza 0.100 10 101 lesz, az ábrázolandó karakterisztika szám pedig 134 (128 + 6) lesz. Ennek bináris értéke 10 000 110. Mivel a szám negatív, az előjelbit 1 lesz. Így végül az átváltott szám nullákkal kipótolva 1 10 000 110 0 000 000 000 0 000 010 0 10 101

Kódolások

Számokon kívül sokféle adattal dolgozhatnak a digitális rendszerek. Ahhoz, hogy az adat értelmezhető legyen binárisan, valamilyen logika szerint az információt bináris jelek sorozatává kell alakítani. Erre valók a különböző kódolási eljárások.

Ezen eljárások az informatika fejlődésével együtt fejlődnek. Manapság több száz formátum és kódolási technika létezik. Ezért ezen fejezetben csak a legalapvetőbb kódolási eljárásokat mutatom be.

ASCII kód

Az ASCII kód egy karakterkódolási szabvány, amit 1963-ban dolgoztak ki annak érdekében, hogy a különböző számítógépek azonos kódokkal jelöljenek betűket. Eredetileg egy 7 bites kód, ami az angol ABC betűit és néhány írásjelét tartalmazza. Az első 32 karakter nem nyomtatható vezérlőkarakter, amelyek olyan gombokat reprezentálnak, mint a TAB, és DEL. Eredetileg a vezérlőkarakterek a kor modemei számára kerültek bele a szabványba.

A szabványt később felbővítették 8 bitesre, ami plusz 127 karakterrel egészítette ki a szabványt. A plusz 127 kód jelentése kódtáblákkal módosítható, így megoldható a különböző kultúrák írásjeleinek támogatása. Később ezeket is szabványosították, azonban ez nem oldotta meg az ASCII legnagyobb problémáját.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

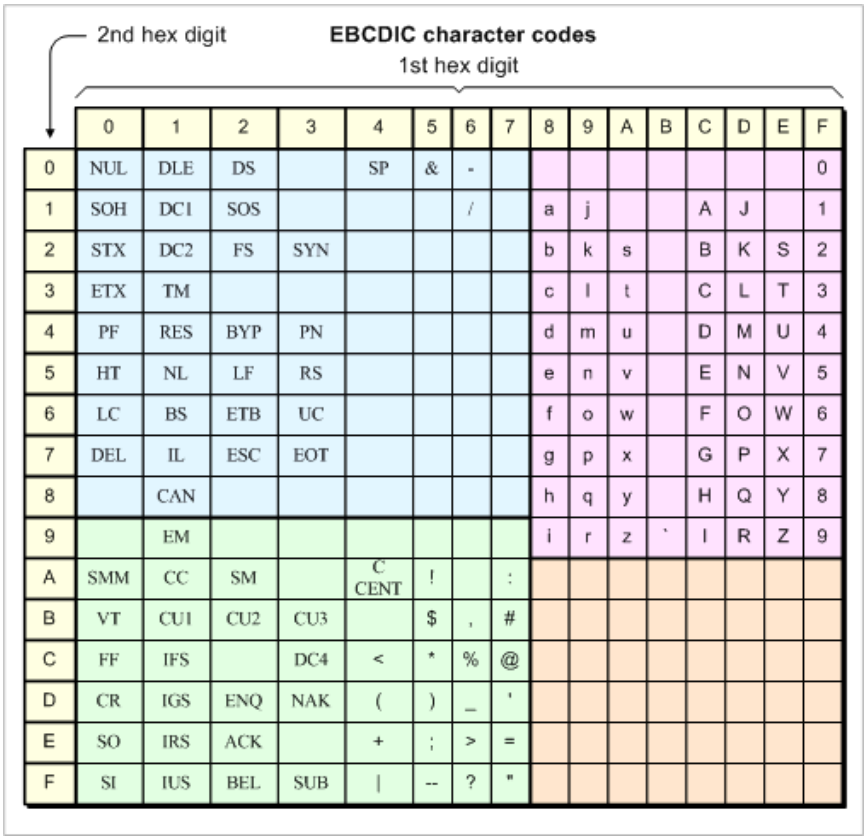
A probléma abból fakadt, hogy az eltérő kódlapok más és más karaktereket rendeltek egy adott kódhoz. Ezért ha a szöveg eredeti kódlapja nem volt ismert, akkor igencsak meg tudta nehezíteni a szöveg elolvasását. Az internet térhódításával a problémát ideiglenesen úgy próbálták megoldani, hogy nem használtak ékezetes betűket a kommunikáció során. Végérvényesen a problémát az UTF kódolások elterjedése oldotta meg a 2000-es évek elején.

EBDIC kódolás

Az EBDIC kódolást az IBM fejlesztette ki 1963-ban a gépei számára. Ugyan ebben az évben jelent meg az ASCII szabvány is. Az EBDIC létrehozásának, megjelenésének oka az volt, hogy a korábbi fejlesztéseiket időben nem tudták módosítani ASCII kódolás használatára.

Nem lett egy nagyon elterjedt szabvány, egy-két IBM és kompatibilis klón mainframe gépen terjedt el. Nem rendelkezik semmi előnnyel az ASCII kódhoz képest, szintén 8 bites kód, amelynek alapja a BCD kódolás.

Hátránya a kódolásnak, hogy az ASCII esetén bemutatott kódlapok problémája itt is fennáll, sőt súlyosabb, mivel ezek nem lettek semmilyen formában szabványosítva. Így sajnos az EBDIC kódolásból legalább 6 -féle egymással teljesen nem kompatibilis változat létezik. Emiatt nem terjedt el elsősorban széleskörűen.



EBCDIC character codes

2nd hex digit \ 1st hex digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	-									0
1	SOH	DC1	SOS				/		a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	TM							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	ETB	UC					f	o	w		F	O	W	6
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z	`	I	R	Z	9
A	SMM	CC	SM		¢ CENT	!		:								
B	VT	CU1	CU2	CU3		$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	'								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	|	--	?	"								

UTF kódolások

A Unicode kódolásokat az eltérő kódlapos ASCII kód hordozhatósági problémájának megszüntetése érdekében hozták létre. A szabvány első változata 1991-ben került publikálásra, azóta folyamatos fejlesztés alatt áll. A könyv írásának pillanatában a legfrissebb verzió a 6.2-es, amit 2012 szeptemberében publikáltak.

A kódlapok problémáját az UTF úgy oldotta meg, hogy eredetileg 16 bitet alkalmazott egy karakter kódolására. Ez a lehetséges karakterek számát 256-ról 65 536-ra növelte. Ma ez a kódolás UTF-16 néven ismert. Azonban mivel ennél jóval több írásjel használt a földön a kódolást tovább kellett fejleszteni. Ezért mára az UTF kódolás 32 bites, ami elvben több, mint 4 milliárd írásjel kódolására alkalmas. A legfrissebb szabvány szerint ebből nagyjából csak 100 ezer írásjelet határoz meg. Így nem valószínű, hogy a kódolásban használt biteket egyhamar megnövelik.

Kompatibilitási okok miatt az UTF első 127 karaktere megegyezik az ASCII kódtáblázattal. Azonban UTF esetén a vezérlő jelek nem használtak.

A kódolásnak három típusa terjedt el. Ezek: UTF-16, UTF-32 és UTF-8. A kötőjel utáni szám a bitek számát jelenti. Unicode és UTF alatt külön bitszám jelzés nélkül a 32 bites változat értendő.

A 8 bites UTF változat a legelterjedtebb. Ez a kódolás változó bithosszúsággal jelöli a karaktereket. 8 bitet alkalmaz egy karakter kódolására, ha a karakter az ASCII kódtáblázatban is megtalálható, 16 bitet, ha a 16 bit elegendő az ábrázoláshoz és 32 bitet, ha csak 32 biten ábrázolható a karakter.

A számítógépes programok többsége képes felismerni a karakter kódolást automatikusan, azonban a szabványba bevezettek egy bájt sorrend jelzést, amit angol mozaik szóval BOM-nak neveznek.

Ez 3 byte a fájl elején, ami a használt program számára azt jelzi, hogy a fájl UTF kódolású. Használata UTF-8 esetén nem ajánlott, mivel megtöri a kompatibilitást az ASCII kóddal.

BCD Kódolás

A BCD mozaikszó magyarra fordítva a Binárisan kódolt decimális angol kifejezésből származik. Ez egy olyan kódrendszer ami bináris, de az emberek is könnyen tudják értelmezni, mivel decimálisan tárolja az adatokat.

Ezt úgy érték el, hogy az egyes számjegyek (0-9) vannak reprezentálva bináris formában. Egy számjegy ábrázolásához 4 bitre van szükség. Két formája terjedt el. Pakolt és pakolatlan.

A pakolatlan formátumban egy számjegy 8 biten ábrázolt. Az alsó 4 bit tárolja a szám értékét, a felső 4 pedig jelentéssel nem rendelkezik. Létezésének oka az, hogy a legtöbb számítógép egy byte-nál (8 bit) kevesebb adattal nem tud dolgozni.

Pakolt formátumban egy byte 2db számjegy tárolására alkalmas. Adattárolásban ezen változata alkalmazott. A felső négy bit tárolja a tízesek számát, az alsó négy bit pedig az egészeket.

A kódolás megszületésének több oka is volt. Elsősorban az, hogy a kor számítógépei igencsak limitált képességekkel rendelkeztek sebesség terén. A Binárisan kódolt decimális számokkal könyebb volt műveleteket végezni, valamint könnyebb egy decimális számot ilyen formátumra konvertálni, valamint a konverziós algoritmus sebessége lineáris. Ez nem mondható el a decimális számrendszerből binárisra alakításról.

További előny, ami ezen kódolás mellett szólt az az volt, hogy ezzel ugyan azt a pontosságot lehetett elérni, mint a tízes számrendszerrel. A kódolás kifejlesztésekor nem létezett még az IEEE754 szabvány. Így a bináris tizedes törtek ábrázolása fix hosszúságon volt csak lehetséges, ami az algoritmus hibájából adódóan egy csomó kerekítési és ábrázolási hibát vont magával. Ezért olyan rendszerekben, ahol fontos volt a tizedes rész BCD kódolást alkalmaztak.

Példaképpen, ha a 0,2 decimális számot binárisan ábrázolni szeretnénk 8 biten úgy, hogy a 8 bitet csak a decimális rész ábrázolására fordítjuk, akkor azt kapjuk visszaváltás után, hogy a számunkban történt egy kis ábrázolási hiba.

$$0{,}2_{10} = 00110011_{2} \approx 0{,}19921875_{10}$$

BCD kódolás esetén 8 biten a szám egész részét is tudjuk ábrázolni, valamint ábrázolási hiba sincs.

$$0{,}2_{10} = 0000.0010_{BCD}$$

Előnyei mellett hátrányai is vannak ennek az ábrázolási módnak. Elsősorban hátrány, hogy a műveletvégzés sokkal komplexebb algoritmusokat igényel a bináris számokhoz viszonyítva. Hátrány továbbá az, hogy a rendelkezésre álló tárterület felhasználása nagyon nem optimális.

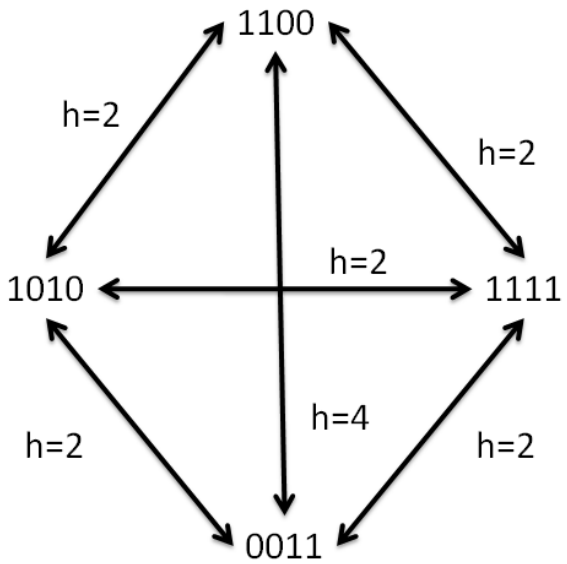
Később készült a BCD-ből 6 bites változat is, amely már írásjelek kódolására is alkalmas volt. Korai mágnesszalagos tárolók alkalmazták.

Digitális jelek hibavédelme

A digitális jelátvitel előnye, hogy az információ bármilyen rossz minőségű, zajos csatornán keresztül átvihető, ha megfelelő kódolást használunk hozzá.

A hibavédelem alapja, hogy az információ hordozó kódszavakat az információ ábrázolás által megszabott minimális bitszámnál hosszabbra kell kialakítani. A kódszavak bővítésének mértéke attól függ, hogy milyen mértékű hibára számítunk az átvitel során.

Matematikailag egy kódszó rendszer hatékonyságát, jóságát, a Hamming távolsággal tudjuk jellemezni. A Hamming távolság legalább két kódszó között definiálható, azt mutatja meg, hogy a két kódszó hány biten tér el egymástól. Ha a kódszórendszer összes kódszavát összehasonlítsuk a rendszerben található összes kódszóval, akkor a Hamming távolságok közül meghatározható a minimális Hamming távolság.



Ez az érték fogja jellemezni a rendszer jóságát, mivel a rendszer csak annyira jó, mint amennyire a leggyengébb láncszeme jó. Ha a minimális távolságot d-vel jelöljük, akkor d-1 bithiba érzékelhető az átvitel során. Így a rendszer érzékelheti a hibás átvitelt és kérheti az adat újraküldését.

Amennyiben az átvitel során d/2-nél kevesebb bithiba következett be, akkor újraküldés nem szükséges, mert a fogadó fél képes javítani az átvitel során keletkezett hibákat.

Az előző ábrán látható rendszer minimális távolsága 2, így 2db bit meghibásodása esetén is észlelni tudja a rendszer a hibát, illetve 1 bithibát javítani is tud.

A rendszerünk jóságát növelhetjük paritásbitek bevezetésével. A paritásbitet a kódszó végére helyezzük, értéke pedig attól fog függeni, hogy a kódszóban található egyesek száma páros, vagy páratlan. Gyakorlatban teljesen mindegy, hogy 0-val vagy eggyel jelöljük azt, hogy a kódszóban található egyesek száma páros. Azonban elfogadott szabály, hogy ha a paritásbit értéke egy, akkor a kódszóban található egyesek száma páros.A csupa nullát tartalmazó kódszó párosnak tekintett. Egy paritásbit alkalmazása a rendszer Hamming távolságát egyel növeli meg.

Az olyan kódrendszereket, amelyek két bithiba érzékelésére és egy javítására képesek általánosan Hamming kódoknak nevezzük. A kritériumoknak megfelelő n bites kódrendszer kepézési szabálya a következő:

1. *A bitek számozása balról jobbra egytől kezdve n-ig*
2. *bitsorszámok felírása bináris formában*
3. *Azon bitek, amelyek kettő egész hatványai (1, 2, 4, 8, stb...) paritásbitek lesznek*
4. *A többi bitpozíció adat tárolásra szolgál.*
5. *A paritásbitek értékének meghatározása:*
    1. *Az első paritásbit értékének számításakor azon biteket kell figyelembe venni, ahol a sorszám legkisebb helyi-értékének bitje 1-es.*
    2. *A második paritásbit értékének számításakor azon biteket kell figyelembe venni, ahol a sorszám második*

legkisebb helyi-értékének bitje 1-es.

3. *A harmadik paritásbit értékének számításakor azon biteket kell figyelembe venni, ahol a sorszám harmadik legkisebb helyi-értékének bitje 1-es.*
4. *A fenti séma folytatása amíg ki nincs számítva minden paritásbit.*

Bit pozíció		**1**	**2**	**3**	**4**	**5**	**6**	**7**
Bit szerepe		**p1**	**p2**	**d1**	**p4**	**d2**	**d3**	**d4**
Paritásbitek által lefedett bitek	**p1**			X		X		X
	p2			X			X	X
	p4					X	X	X

19. Táblázat: Hamming kód képzés vizuális reprezentációja 4 bitre

Az ismertetett algoritmus a Hamming(7, 3) kódolás általánosított formája, ami hét bitet alkalmaz, 3 paritásbittel, így az adatbitek száma 4. A Hamming(8,4) kódolás annyiban különbözik csupán, hogy plusz egy extra paritásbit van bevezetve az egész kódszóra.

Az alábbi táblázat a paritásbitek által a kódrendszerhez adott plusz információt tartalmazza százalékosan.

Paritásbitek száma	**Kódrendszer bitszáma**	**Adatbitek száma**	**Plusz információ a kódrendszerben**
2	3	1	66,666%
3	7	4	42,857%
4	15	11	26,666%
5	31	26	16,129%
n	2^n-1	2^n-n-1	$\left(1-\frac{2^n-n-1}{2^n-1}\right)\cdot 100$

20. Táblázat: Hamming kódrendszerek adatai

Boole algebra

A Boole algebra a nevét George Boole angol matematikus után kapta. Felfedezése sokáig homályban maradt, azonban halála után 70 évvel rátalált Claude Shannon, aki rámutatott arra, hogy a Boole algebra segítségével optimalizálni lehet az elektromechanikus relék rendszerének a tervezését. A számítástechnika fejlődésével Boole felfedezése még nagyobb jelentőséget kapott.

Logikai műveletek

A Boole algebra logikai műveleteket definiál, amelyekkel műveletet végezhetünk bináris számjegyeken.

Ezen műveletek közül a legegyszerűbb és az egyetlenegy változós művelet a tagadás, negáció. Angolul a szakirodalomban NOT műveletnek nevezik. A művelet lényege, hogy a bemeneti érték inverzét fogja eredményül adni.

A	**NOT A**
0	1
1	0

21. táblázat: A NOT művelet igazságtáblázata

A műveletek leírása igazságtáblázatokkal történik, amelyben fel van tüntetve az összes lehetséges bemeneti kombinációhoz a kimenet értéke.

Kétváltozós műveletek az ÉS a VAGY, a KIZÁRÓ VAGY, az EKVIVALENCIA (egyenlőség), a NAND és a NOR. Az ÉS művelet igaz eredményt akkor fog adni, ha mind a két bemeneti változó értéke igaz. A VAGY művelet akkor fog igaz értéket adni, ha a bemeneti változók közül bármelyik igaz. A KIZÁRÓ VAGY, művelet igaz értéket csak akkor ad, ha a bemeneti változók értéke nem azonos. Ellentettje, az EKVIVALENCIA akkor fog igaz értéket adni, ha a két bemeneti

változó értéke azonos.

A NAND és a NOR műveletek az integrált áramkörök elterjedésével jelentek meg. Kombinált műveletek. A NAND művelet eredményül az ÉS művelet eredményét adja negálva, a NOR művelet pedig eredményül a VAGY művelet eredményét adja negálva.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

22. táblázat: Az ÉS művelet igazságtáblázata

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

23. táblázat: A VAGY művelet igazságtáblázata

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

24. táblázat: A KIZÁRÓ VAGY művelet igazságtáblázata

A	B	A EQ B
0	0	1
0	1	0
1	0	0
1	1	1

25. táblázat: Az EKVIVALENCIA művelet igazságtáblázata

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

26. táblázat: A NAND művelet igazságtáblázata

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

27. táblázat: A NOR művelet igazságtáblázata

A Boole algebrai műveletek jelölésére kétféle rendszer is létezik. Az egyik a matematikai jelölésrendszer, a másik pedig a mérnöki jelölésrendszer. Nevéből adódóan a matematikai jelölésrendszert a matematikusok használják előszeretettel, míg a mérnöki jelölésrendszert a villamos és egyéb mérnökök. A két jelölésrendszer közötti különbségeket az alábbi táblázat foglalja össze

	Matematikai jelölés	Mérnöki jelölés
Változó tagadása (NOT)	$\neg A$	$\bar{A}$
ÉS művelet	$A \wedge B$	$A \cdot B$
VAGY művelet	$A \vee B$	$A + B$
KIZÁRÓ VAGY művelet	$\neg(A \leftrightarrow B)$	$A \oplus B$
EKVIVALENCIA	$A \leftrightarrow B$	$A \oplus B$

28. táblázat: Matematikai és Mérnöki jelölésrendszer

Az ÉS, VAGY és a tagadás művelettel bármelyik másik összetett művelet kifejezhető. Az igazságtáblázatban az olyan sorokat, amelyek igaz értéket adnak kimenetnek, össze kell kapcsolni egy VAGY művelettel., az egyes változók között pedig ÉS kapcsolatot kell létesíteni. Azon változók, amelyek értéke 0, azok negálva fognak megjelenni. Erre példának nézzük meg a KIZÁRÓ VAGY műveletet. Ezt ÉS, VAGY, és nem műveletekkel a következőképpen lehet leírni:

$$(\neg A \wedge B) \vee (A \neg B) \equiv \bar{A} \cdot B + A \cdot \bar{B}$$

A műveleteink felfoghatóak függvényekként. Az ilyen függvényeket logikai függvénynek nevezzük. A NAND és NOR úgynevezett univerzális logikai függvények, mert pusztán csak NAND vagy NOR függvények használatával leírható bármilyen logikai függvény.

Mivel algebráról beszélünk, az alapműveletek között lehet definiálni egy azonossági rendszert némi gondolkodással. Ezen azonosságokat 6 fő csoportba szokták sorolni, valamint az azonosságokat szokás egy sorszámmal is jelölni.

Első azonossági csoport a 0 és 1 konstans értékek és a változók között fennálló azonosságokat tartalmazza:

$$1, A+0=A \quad 3, A\cdot 0=0$$
$$2, A+1=1 \quad 4, A\cdot 1=A$$

A második azonossági csoport a ponált (nem tagadott) és a negált változó azonosságait tartalmazza:

$$5, A+\overline{A}=1 \quad 7, A\cdot\overline{A}=0$$
$$6, A+A=A \quad 8, A\cdot A=A$$
$$9, \overline{\overline{A}}=A$$

A harmadik csoport a műveletek közötti kommutativitásra (felcserélhetőség) vonatkozik:

$$10, A+B=B+A \quad 11, A\cdot B=B\cdot A$$

A negyedik csoportba az asszociativitás (csoportosíthatóság) azonosságai tartoznak:

$$12, A+B+C=(A+B)+C=A+(B+C) \quad 13, A\cdot B\cdot C=(A\cdot B)\cdot C=A\cdot(B\cdot C)$$

Az ötödik csoportba a disztributivitás (kiemelhetőség) azonossági tartoznak:

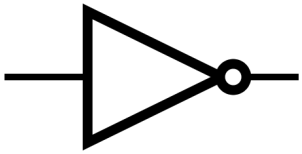
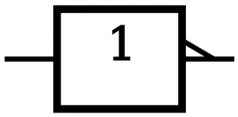
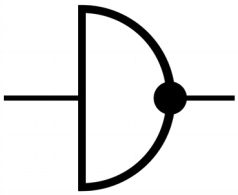
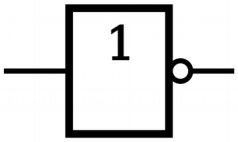
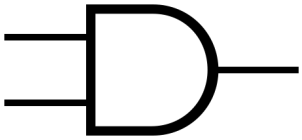
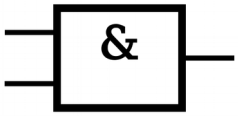
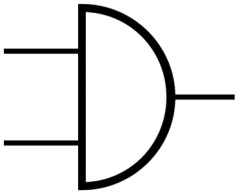
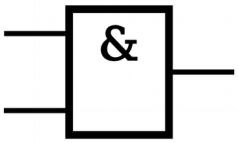
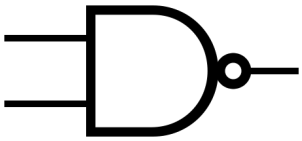
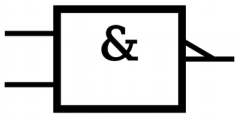
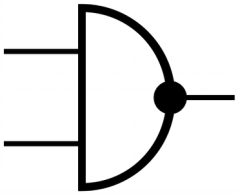
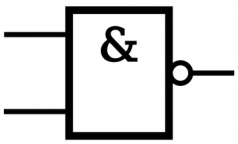
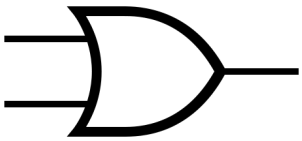
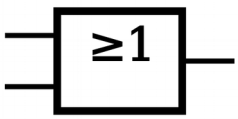
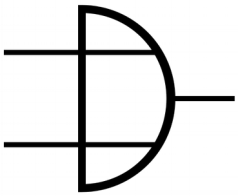
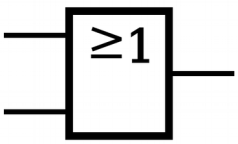
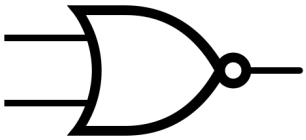
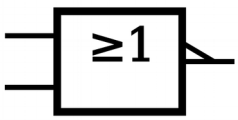
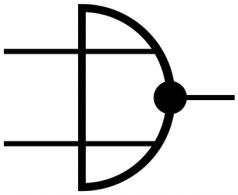
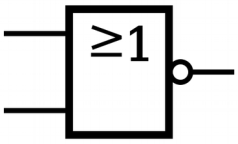
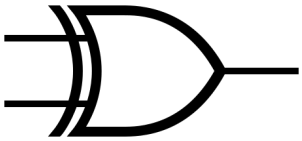
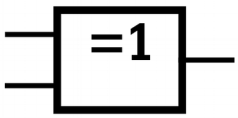
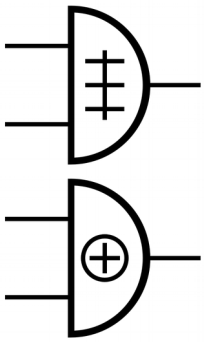
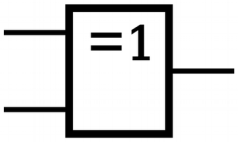
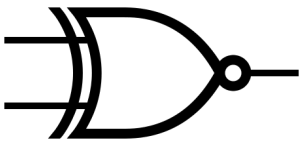
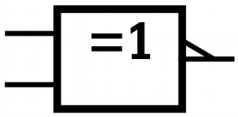
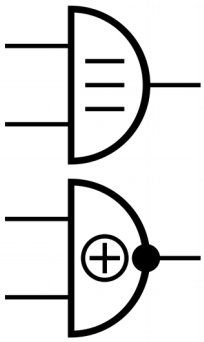
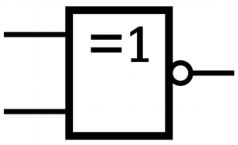
$$14, A\cdot(B+C)=A\cdot B+A\cdot C \quad 15, A+B\cdot C=(A+B)\cdot(A+C)$$

A hatodik csoportba a de Morgan azonosságok tartoznak. Ezen azonosságok teljes kifejezések negálására vonatkoznak:

$$16, \overline{A+B+C}=\overline{A}\cdot\overline{B}\cdot\overline{C} \quad 17, \overline{A\cdot B\cdot C}=\overline{A}+\overline{B}+\overline{C}$$

Kapuáramkörök rajzjelei

Az említett logikai alapműveletek léteznek elektronikai alkatrészekként is. Ezeket logikai kapuáramköröknek nevezzük. Mivel ezek áramköri elemek, ezért kapcsolási rajzjel is tartozik hozzájuk. A használatos kapuáramköri rajzjeleket az alábbi táblázatban foglaltam össze, szabványonként csoportosítva. Az egyes szabványok rajzjeleit egy kapcsolási rajzon belül keverni nem esztétikus.

Kapu neve	*ANSI/MIL*	*IEC*	*DIN*	*Britt*
NOT				
AND				
NAND				
OR				
NOR				
XOR				
EQ				

29. táblázat: A használatos kapuáramkörök rajzjelei szabványonként

Logikai függvények megadása és egyszerűsítése

A logikai függvényeket különbözőképpen adhatjuk meg. A legegyszerűbb megadási forma az, hogy a teljes függvény igazság táblázatát felírjuk. Ezen módszer hátránya az, hogy az információt továbbítani ilyen formában nem éppen egyszerű. Ezért a függvényeket továbbítani olyan formában szokták, hogy megadják a változók számát, valamint a függvény kimeneti értékét decimálisan.

Példaképpen az ÉS függvény így adható meg: $F^4{:}8$ Ebből a formából könnyen visszanyerhető az igazság táblázat. A decimális értéket át kell váltani binárisba, majd a kapott számot úgy kell beírni az igazság táblázatba, hogy a legnagyobb helyi értéken lévő bit kerüljön ahhoz a sorhoz, ahol az összes bemeneti változó egyes értéket vesz fel.

Egy másik függvény megadási módszer az, ha a függvényből csak az olyan minterm sorszámokat írjuk ki, ahol a függvény kimenete igaz. Példaképpen az ÉS függvény ebben az alakban megadva így néz ki: $\sum m_i^2(3)$

A logikai függvényeinket célszerű áramköri megvalósítás előtt valamilyen módszerrel egyszerűsíteni, ha lehetséges. Ez anyagi okok és átláthatósági szempontok miatt fontos. Minél egyszerűbb a függvényünk, annál kisebb a hibázás valószínűsége az áramkör megépítésekor, valamint nem elhanyagolandó szempont anyagi oldalról, hogy ha valami megvalósítható 3 kapu felhasználásával, akkor minek használjunk dupla annyit. Egyszerűsítési módszerekből létezik az úgynevezett algebrai (azonosságok felhasználásával), grafikus és a Quine–McCluskey módszer.

Az algebrai módszer az ismertetett azonosságok felhasználásával való egyszerűsítési eljárás. Ez némi gyakorlatot igényel. Leginkább olyan függvények egyszerűsítésénél alkalmazzák, ahol a változók száma több, mint 4. Négy, vagy kevesebb változót tartalmazó függvények esetén grafikus módszereket alkalmazunk az egyszerűségük miatt. Az algebrai módszer hatékony használatához az ismertetett 17 azonosságon kívül érdemes még megjegyezni a következő azonosságokat:

$$18,\ A+-A\cdot B=A+B \qquad 20,\ A+A\cdot B=A$$
$$19,\ A\cdot(-A+B)=A\cdot B \qquad 21,\ A\cdot(A+B)=A$$

A grafikus módszerek közül a Minterm és a Maxterm táblázatok használata a legelterjedtebb. A függvényünket attól függően, hogy Minterm vagy Maxterm alakban van, ezzel fogjuk tudni egyszerűsíteni.

A függvény akkor van Minterm alakban, ha a független változók között ÉS kapcsolat van, a változó csoportok között pedig VAGY kapcsolat. Maxterm alakban a független változók között VAGY kapcsolat van, a változó csoportok között pedig ÉS kapcsolat. Minterm alak esetén minden Minterm egyetlen egy bemeneti kombináció esetén produkál igaz értéket. Maxterm alak esetén minden Maxterm egyetlen egy bemeneti kombináció esetén ad vissza hamis értéket.

Minterm alakban megadott függvényből Maxterm alakot a teljes függvény negálásával kapunk. Ez a módszer alkalmazható a Maxterm alakból Minterm alakba átalakításkor is a teljes függvény negálásakor. Alternatív módszerkén alkalmazható az, hogy a függvényt átírjuk a maxterm táblázatba, majd pedig onnan olvassuk ki a függvény alakot. Példaképpen a $F=A\cdot-B+-A\cdot B$ Minterm alakú függvény Maxterm alakja a $F=-A+B\cdot A+-B$ lesz.

A Minterm/Karough táblázat használata népszerűbb egy kicsivel. Ennek oka az, hogy a függvények, ha Minterm alakban vannak, akkor könnyen átalakítható a hálózat csak NAND kapuk használatára. A Maxterm táblázatok főleg NOR kapus tervezéskor alkalmazottak.

A táblázatok lényegében a lehetséges változók Venn diagramjának négyszögesített, majd táblázatba foglalt változatai. Változók számától függően létezik 2, 3, 4 és 5 változós Minterm és Maxterm táblázat. Négy változó felett azonban a használat kicsit problémás (5 változó esetén már térben kell gondolkodni), A Minterm és Maxterm táblázatok peremezésénél arra kell ügyelni, hogy minden változó a tábla egyik felén 1 értékkel, a másik felén pedig 0 értékkel szerepeljen.

Az egyszerűsítési módszer logikáját Minterm táblázatok használatával írtam le.

34. ábra: Minterm táblázatok *35. ábra: Maxterm táblázatok*

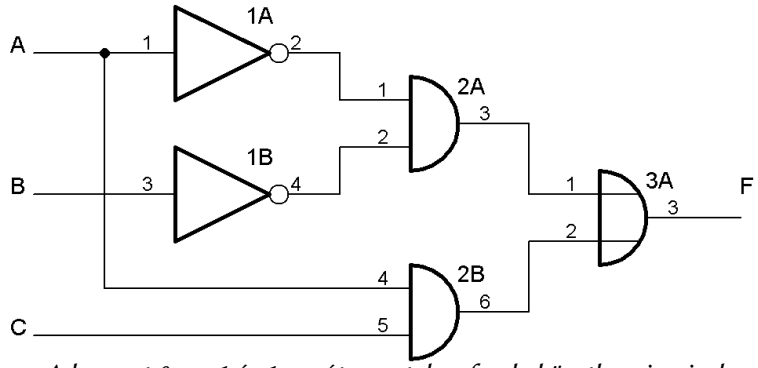
A Minterm táblázatban szereplő számok az igazságtáblázat sorainak számát reprezentálják. Ezekbe a mezőkbe kell beírni a függvény kimenetétől függően egy egyest, vagy nullát. A táblázatban az egyes értékek vonhatóak össze, ha egymás mellett vagy alatt vannak kettő hatványai szerint. Tehát 2, 4, 8 vagy 16 mezőt tudunk összevonni. Ha minden mezőt össze tudtunk vonni a táblázatban, az azt jelenti, hogy a kimenet nem függ egyetlen bemenet értékétől sem.

A táblázat szélein lévő számok összevonhatóak. Tehát például a négyes táblában lévő 4-es és 6-os jelzésű mező összevonható, a 0-s mező összevonható a 2-es és a 8-as mezővel is.

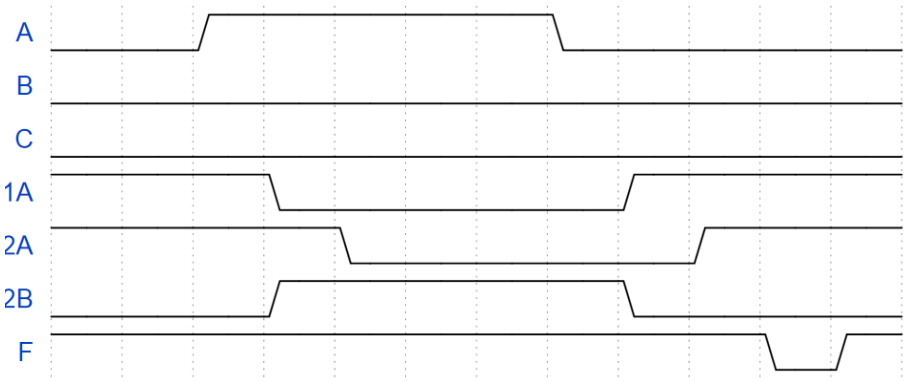
Az összevont blokkok esetén csak azon változókat kell leírni az egyszerűsített alakban, amelyek egyértelműen benne vannak egy változóban, vagy nem. Ha a blokk egyik fele belelóg az egyik változóba, de nincs teljesen benne, akkor az a blokk elhagyható.

Amennyiben olyan helyzet adóik, hogy két blokk határos egymással, de nincsenek egymással összekapcsolva, akkor a függvényünk hazárdos. Ez azt jelenti, hogy az inverterek és a kapuk késleltetése miatt adódhat olyan időpillanat, amikor a kimenet nem a bemenetnek megfelelő lesz. Ezt a jelenséget hazárdnak nevezik.

A hazárd jelenséget a legjobban a $F=-A-B+AC$ *függvény szemlélteti. A függvény kapcsolási rajza:*



A hazárd jelenség az A bement 0 ⟶ 1 és 1 ⟶ átmenetekor fog bekövetkezni, mivel a rajzon 2A és 2B jellel lejölt kapuk kimenetváltásai átfedik egymást. Ez jól látható az alábbi idő diagramon is.



A hazárd nem kívánt jelensége elkerülhető olyan módon, hogy az említett szomszédos blokkokat összekötjük és a keletkezett összevonási lehetőséget is kiolvassuk a függvénnyel együtt. Ez minden esetben extra kapukat iktat a kész áramkörbe.

A kapuk késleltetéséből adódó hazárd jelenséget statikus hazárnak nevezzük. Dinamikus hazárd jelenség a kapcsolók, elektromechanikus érzékelők pergéséből adódhat. Ezen jelenség elkerülése hardveres úton a Az Arduino megszakítások kezelése résznél részletesen lesz tárgyalva.

Előfordulhat, hogy a függvényünk működése szempontjából mindegy, hogy egy adott helyen a függvény értéke igaz vagy hamis. Az ilyen kombinációkat nevezzük közömbös mintermeknek vagy maxtermeknek a függvény alakjától függően.

A közömbös mintermek és maxtermek esetén a táblázatba X-et kell írni. Az egyszerűsítés során a függvénytől függően ezek tekinthetőek igaz értékűnek vagy hamisnak, hogy egyszerű alakot kapjunk.

Egyszerűsítés Quine–McCluskey módszerrel

A Minterm táblázatos egyszerűsítés könnyen alkalmazható 4 változóig. Négy változó felett azonban exponenciálisan bonyolódik a táblázat, így 5 változó felett mondhatni használhatatlan a módszer. Pont ilyen kombinációs hálózatok egyszerűsítésére találták ki a Quine–McCluskey algoritmust, amely a nevét a kitalálóiról kapta. Mivel ez egy algoritmus, tetszőleges programozási nyelven implementálható tetszőleges számítógépre. Az algoritmus egyszerű lépésekből áll, így nem kell hozzá gépi erőforrás.

Az algoritmus működését egy példán keresztül fogom szemléltetni. A példához azonban kelleni fog egy tetszőleges logikai függvény, amely esetén ismerjük azokat a minterm sorszámokat, ahol a függvény igaz értéket ad eredményül. Ha a függvényünk nem ilyen alakban van, akkor át kell alakítani úgy, hogy megkapjuk a minterm sorszámokat. A függvény amit egyszerűsíteni fogunk legyen a következő:

$$\sum m_i^2(0,1,4,5,6,7,14,15)$$

Az első lépés az, hogy a függvény minterm számaihoz meghatározzuk, hogy a bináris alakjukban hány darab egyes van, majd ez alapján sorba rendezett csoportokra bontjuk őket. A csoport bontást egy függőleges oszlopban felírva kell megtenni. Mivel az algoritmus során n darab ilyen csoportokból álló oszlopot alakítunk ki, az oszlop felett index számmal érdemes jelezni, hogy melyik lépésnél is tartunk. Ezt jelölhetjük arab, vagy Római számokkal is.

A kiindulási oszlop a függvény alapján:

	I.
	0
0 - 0	
1 - 1	1
4 - 1	4
5 - 2	
6 - 2	5
7 - 3	6
14 - 3	
15 - 4	7
	14
	15

30. Táblázat: Az egyesek számának meghatározása és a kiindulási oszlop

A kiindulási oszlopból a következő oszlop elemeit és csoportjait úgy kapjuk meg, hogy az egymással szomszédos csoportokban szereplő elemeket összehasonlítjuk egymással. Ha az összehasonlításkor azt tapasztaljuk, hogy a két szám különbsége kettő egész hatványa (0, 1, 2, 4, 8, stb...), akkor a két szám összevonható.

Az összevont számokat vesszővel elválasztva egymás mellé írjuk, mögéjük pedig zárójelben azt, hogy mennyi volt az összevonási különbség. Az egyes csoport összehasonlítások után kapott értékeket szintén el kell választani. Tehát a 0–1, 0-4 csoport-összehasonlítás eredményei fognak egy csoportot alkotni.

A kiindulási oszlopban, azon számjegyek mellé, amelyek valamilyen formában átkerültek a második oszlopba egy pipát kell tenni, ezzel jelezve, hogy átkerült a második oszlopba és további teendők vannak vele. Ha egy szám vagy számsor bármelyik oszlopban nem kap pipát, akkor az ABC kis betűinek egyikével jelölni kell. Ez azt jelenti, hogy az a számsor további összevonásban nem tud részt venni. A példában mindegyik számjegy átkerült a második oszlopba:

I.	II.
0 ✓	0, 1 (1)
	0, 4 (4)
1 ✓	
4 ✓	1, 5 (4)
	4, 5 (1)
	4, 6 (2)

5 ✓ 6 ✓ --- 7 ✓ 14 ✓ --- 15 ✓	--- 5, 7 (2) 6, 7 (1) 6, 14 (8) --- 7, 15 (8) 14, 15 (1)

31. Táblázat: Az első oszlopból képzett második oszlop

A harmadik oszlop elkészítése esetén hasonló az eljárás, de azt is figyelembe kell venni, hogy a zárójelben szereplő számok azonosak legyenek. Például a 0,1 számpár összevethető a 4,5 számpárral és össze is vonhatóak. Az új összevonás mellett fel kell tüntetni a zárójelben az új összevonási számot is, így a harmadik oszlopba a 0, 1, 4, 5 (1, 4) sor kerül.

A 0, 4 és 1, 5 számpárok összevonásánál azt tapasztaljuk, hogy ugyan azok a számjegyek szerepelnek benne, mint amit a korábban leírtunk, csak más sorrendben. Az ilyen párokat ismételten nem kell leírni, viszont pipával jelezni kell, hogy továbbjutottak a következő oszlopba.

I.	II.	III.
--- 0 ✓ --- 1 ✓ 4 ✓ --- 5 ✓ 6 ✓ --- 7 ✓ 14 ✓ --- 15 ✓	--- 0, 1 (1) ✓ 0, 4 (4) ✓ --- 1, 5 (4) ✓ 4, 5 (1) ✓ 4, 6 (2) ✓ --- 5, 7 (2) ✓ 6, 7 (1) ✓ 6, 14 (8) ✓ --- 7, 15 (8) ✓ 14, 15 (1) ✓	--- 0, 1, 4, 5 (1, 4) a --- 4, 5, 6, 7 (2, 1) b --- 6, 7, 14, 15 (1, 8) c

32. Táblázat: A második oszlopból képzett harmadik oszlop

A negyedik oszlop előállításánál is hasonló módon kellene eljárni, de mivel nincs lehetséges összevonási lehetőség (mivel nem egyeznek meg egyik párnál sem a zárójelben feltüntetett számok), így a végére értünk az algoritmusnak.

A lépéseket addig kell ismételni, amíg el nem fogy a lehetséges összevonás. Ha a zárójelben több szám is szerepel, akkor a számok sorrendje nem számít, vagyis az 1, 8 összevonható a 8, 1 jelöléssel, ha a sor számjegyeinek különbsége kettő egész hatványa.

A kapott összevonási lehetőségekből ki kell választani azokat, amelyek minimálisan szükségesek a függvény megvalósításához. Ezáltal a függvény legegyszerűbb alakját kapjuk meg, amely nem biztos, hogy hazárd mentes lesz.

A legegyszerűbb alakot úgy kapjuk meg, hogy készítünk egy táblázatot, amelynek a soraiban az összevonási lehetőségek betűjele szerepel, az oszlopokban pedig azok a mintermek, amelyeket tartalmaz az összevonási lehetőség.

	0	**1**	**4**	**5**	**6**	**7**	**14**	**15**
a	x	x	x	x				
b			x	x	x	x		
c					x	x	x	x

33. Táblázat: A minimálisan szükséges összevonások kiválasztása

A fenti táblázatból látható, hogy a függvény legegyszerűbb alakjának megvalósításához az a és c jelű összevonási lehetőség kell, mivel a b csak olyanokat tartalmaz, amelyek az a és c jelűben is megtalálhatóak. Ha a függvény hazárd mentes alakját szeretnénk megkapni, akkor az összes kapott összevonási lehetőséget fel kell használnunk.

Konkrét logikai bemenetekkel jelölt függvény alakot az összevonási lehetőségekből úgy kapunk, hogy kiválasztunk egy tetszőleges minterm sorszámot, ami az összevonásban szerepel. Ezt a számot átírjuk bináris alakba. A bináris alakjában a számjegyek alá írjuk a logikai bemeneteket olyan sorrendben, ahogy az igazság táblában is felvettük őket. Ahol a számjegy 0 értékű, ott az adott bemenetet reprezentáló változót negálni kell.

Ez még nem az egyszerűsített alak, viszont már közel járunk hozzá. Azon számjegyeket és betűket el lehet hagyni az alakból, amelyek szerepelnek az összevonási lehetőség mellett zárójelben. Példaképpen az a jelű összevonási lehetőség leképezése:

- *Kiválasztott számjegy: 0, ennek a bináris alakja: 0000*
- *Az alkalmazott bemenet sorrend: ABCD, ami a számjegyek alapján negálás után* $\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}$ *lesz.*
- *A kihúzandó bit pozíciók értékei az 1 és 4, amit betűre fordítva a* $\overline{B}\cdot\overline{D}$ *párt jelenti, mivel az A betű értéke 8, a B betűé, 4, a C betűé 2, a D betűé pedig 1*
- *Így az a jelű összevonási lehetőség betűkkel:* $\overline{A}\cdot\overline{C}$

A függvény legegyszerűbb alakja: $\neg A \cdot \neg C + B \cdot C$ *, hazárd mentes alakja:* $\neg A \cdot \neg C + B \cdot C + \neg A \cdot B$ *Minterm táblás egyszerűsítéssel vagy algebrai módszerrel ellenőrizhető a kapott eredmény. Demonstrációs célból az algebrai módszer segítségével mutatom be, mivel erről külön nem volt példa.*

Első lépésben a minterm számokat a bemenetek betűjeleivel kell reprezentálni. Ehhez ugyan az az algoritmus alkalmazható, mint a Quine–McCluskey módszer utolsó lépésében, csak itt nincs mit elhagyni.

Ez alapján a kezdő függvény:

$$F \equiv \neg A \cdot \neg B \cdot \neg C \cdot \neg D + \neg A \cdot \neg B \cdot \neg C \cdot D + \neg A \cdot B \cdot \neg C \cdot \neg D + \neg A \cdot B \cdot \neg C \cdot D$$
$$+ \neg A \cdot \neg B \cdot \neg C \cdot \neg D + \neg A \cdot B \cdot C \cdot \neg D + \neg A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot \neg D + A \cdot B \cdot C \cdot D$$

Az egyszerűsítéshez a 6, 5, 4 és 10-es azonosságokat felhasználva azt tapasztaljuk, hogy a D betű kiejthető minden tagból:

$$F \equiv \neg A \cdot \neg B \cdot \neg C + \neg A \cdot B \cdot \neg C + \neg A \cdot B \cdot C + A \cdot B \cdot C$$

Ha az azonosságokat még egyszer alkalmazzuk, akkor a következő eredményt kapjuk:

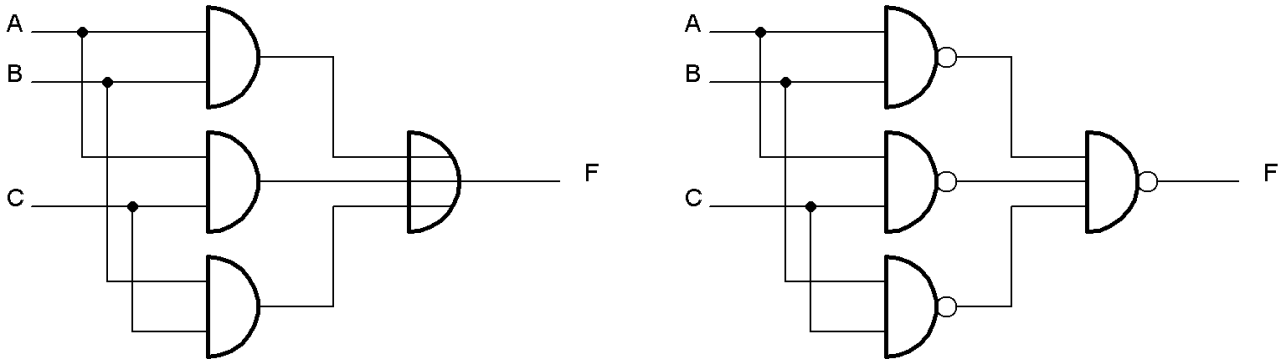
$$F \equiv \neg A \cdot \neg C + B \cdot C$$

Hálózatok megvalósítása NAND és NOR kapuk segítségével

A Minterm alakban lévő logikai függvényünk megvalósítható kizárólag csak NAND kapuk felhasználásával is. Ehhez azt kell tenni, hogy a hálózatot kétszer negáljuk. A folyamat jól szemléltethető egy példán keresztül. Példaképpen valósítsuk meg a $F = A \cdot B + A \cdot C + B \cdot C$ függvényt. A kétszeres negálás után a következő alakot kapjuk:

$$F = \overline{\overline{A \cdot B + A \cdot C + B \cdot C}} = \overline{\overline{A \cdot B} \cdot \overline{A \cdot C} \cdot \overline{B \cdot C}}$$

Ebben az alakban a változó tagok felett szereplő negálás jel egy NAND kaput reprezentál. Ez azt jelenti, hogy a $A \cdot B, A \cdot C, B \cdot C$ tagok egy-egy NAND kapura csatlakoznak, majd a NAND kapuk kimenetei egy újabb NAND kapura, amely kimenete előállítja az F függvényt.



Az F függvény megvalósításakor a kapcsolási rajz megegyezik a kétszintes rajzzal azzal a különbséggel, hogy csak egy fajta kapu szerepel a kapcsolásban.

Többszintes NAND kapcsolások létrehozása is megoldható, amennyiben ezt a függvény indokolja. Ebben az esetben a Negálást és a De morgan szabályokat addig kell alkalmazni, míg a függvényben csak és kapcsolatok vannak negálva. Példaképpen a $F = A \cdot B + C$ függvény NAND alakja a következő lesz: $F = \overline{\overline{\overline{A \cdot B} \cdot C}}$

Az ismertetett tervezési szabályok megfelelően alkalmazva NOR hálózatok tervezésekor is alkalmazhatóak, ha a függvény maxterm alakban van. Többszintes NOR hálózatok esetén a Negálást és a De morgan szabályokat addig kell alkalmazni, míg a függvényben csak vagy kapcsolatok vannak negálva.

Digitális áramkörök csoportosítása

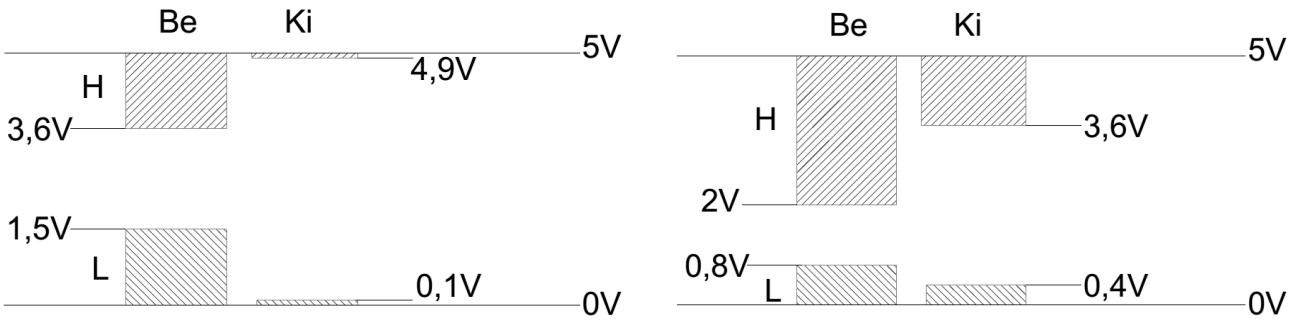
A legegyszerűbb digitális áramkörök, amik építhetőek, az úgynevezett kombinációs hálózatok. Ezen hálózatok a nevüket onnan kapták, hogy egy adott bemeneti jelkombinációra minden esetben ugyanazt a kimeneti jelsorozatot fogják adni. Ezen hálózatok működése könnyen leírható igazságtáblázatokkal és különböző Boole algebrai azonosságokkal.

A digitális áramkörök másik nagy csoportja a sorrendi hálózatok (más elnevezéssel időrendi, vagy szekvenciális hálózatok). Ezen hálózatok tulajdonsága, hogy adott bemeneti jelkombinációkra időfüggően eltérő kimeneti értéket adhatnak. Tehát a kimenet értéke függ a bementi jelkombinációktól és a kimenetek előző állapotától. Ezért az ilyen áramkörök minden esetben tartalmaznak valamilyen memóriát (lásd tárolók) az állapot megőrzésre., valamint egy visszacsatolást is, hogy az új állapot kialakításában szerepet játsszon a hálózat korábbi állapota.

Digitális integrált áramkörök

A digitális technikában használatos integrált áramkörök gyártástechnológia alapján két fő csoportra oszthatóak: tranzisztoros megvalósítású áramkör és fém oxid félvezetős technológia. Ezek angol rövidítése a TTL és a CMOS.

A legnagyobb különbség a két típus között az, hogy különböznek a logikai jelszintjeik, mivel nem azonos a fogyasztásuk. A CMOS áramkörök belső felépítésük miatt sokkal kevesebbet fogyasztanak TTL társaikhoz képest, ezáltal szimmetrikusabb, egyenletesebb ki-és bemenő jelszinteket fogadnak. Az eltérő jelszintek miatt a TTL áramkörök feszültségszint illesztés nélkül nem keverhetőek CMOS áramkörökkel, amennyiben a CMOS áramkör nem 5V feszültségről van működtetve. A CMOS és a TTL jelszintek közötti különbség jól látszik az alábbi ábrákon.



A CMOS áramköröknél konkrét feszültségértékek helyett érdemesebb a tápfeszültséghez viszonyítva százalékosan megadni a jelszint karakterisztikát. Ennek oka az, hogy a CMOS áramkörök jóval szélesebb tartományban működőképesek. Általában ez a tartomány 3 és 15 Volt közötti.

A 3,3 Voltos jelszint karakterisztika kiemelten fontos számunkra, mivel ezen feszültség-és jelszint nagyon elterjedt, valamint a nagyobb tudású mikrovezérlők mindegyike 3,3 Voltos jelszintekkel dolgozik.

	Be %	***Ki %***	***Be, ha Ut = 3,3V***	***Ki, ha Ut = 3,3V***
Magas szint	$H \geqslant 72\,\%$	$H \geqslant 98\,\%$	$H \geqslant 2{,}376\,V$	$H \geqslant 3{,}234\,V$
Alacsony szint	$L \leq 30\,\%$	$L \leq 2\,\%$	$L \leq 0{,}99\,V$	$L < 0{,}066\,V$

34. táblázat: CMOS áramkörök bemeneti és kimeneti jelszint karakterisztikája a tápfeszültség %-ban megadva

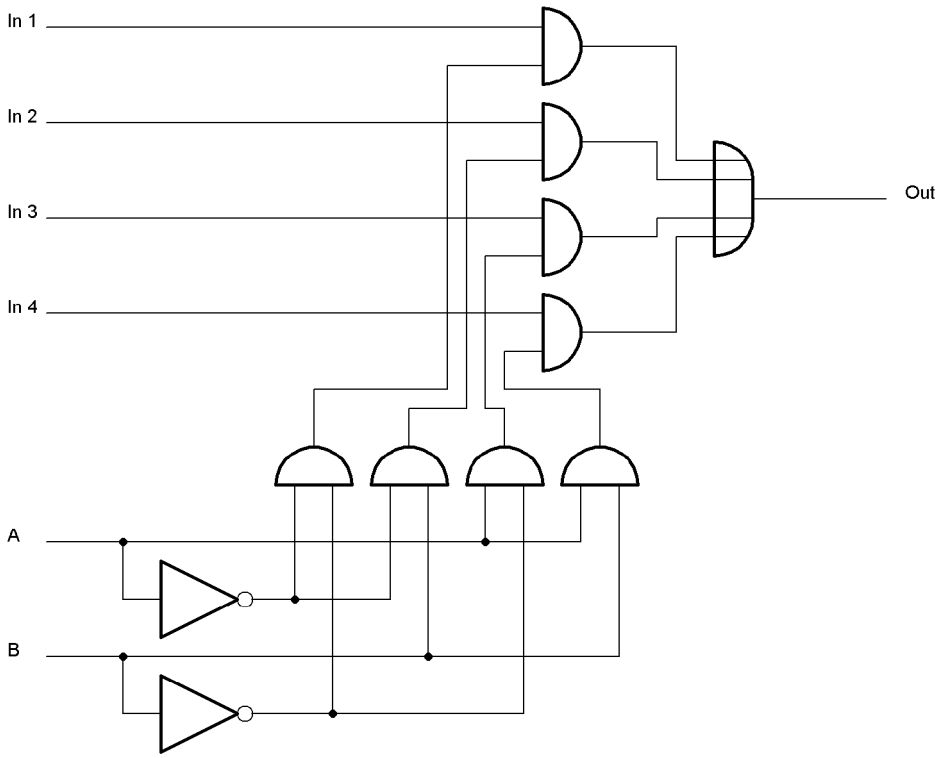
Funkcionális áramkörök

Multiplexer

A multiplexer egy olyan áramkör, amely rendelkezik N darab vezérlő bemenettel és 2^N adat bemenettel. A

vezérlő bemenetek kombinálásától függően az áramkör kimenetén a bemenetek közül az egyik fog megjelenni.

Bemenet választáson kívül a multiplexer áramkör alkalmas még kombinációs hálózatok megvalósítására is. Egy 2 vezérlő bemenetes multiplexerrel minden 2 változós logikai függvény megvalósítható. Az adat bemeneteket a megvalósítandó függvényünk igazságtáblázata alapján vagy magas szintre, vagy alacsony szintre kell kapcsolni fixen. Ekkor a vezérlő bemenetek változtatásával előáll a függvény kimenete. Mikrovezérlős témakörben alkalmas bemeneti port vagy analóg csatornák bővítésre.



1. Ábra: Egy 4 bemenetes multiplexer.

Analóg bemenet bővítésre a multiplexer egy speciális fajtája, az analóg multiplexer használható. Ez úgy működik, mint egy digitális multiplexer, azonban nem csak digitális jel továbbítására képes.

A multiplexerek alaklámázhatók univerzális minterm generátorként is, vagyis tetszőleges, minterm alakban felírt függvény megvalósítására képesek.

A megvalósítás alapkoncepciója egyszerű és tetszőlegesen adaptálható. Egy négy változós logikai függvény megvalósítható egy 16 bemenetű multiplexer segítségével. Egy ilyen multiplexer 4 választó bemenettel rendelkezik. Ezek lesznek a választó bemenetek.

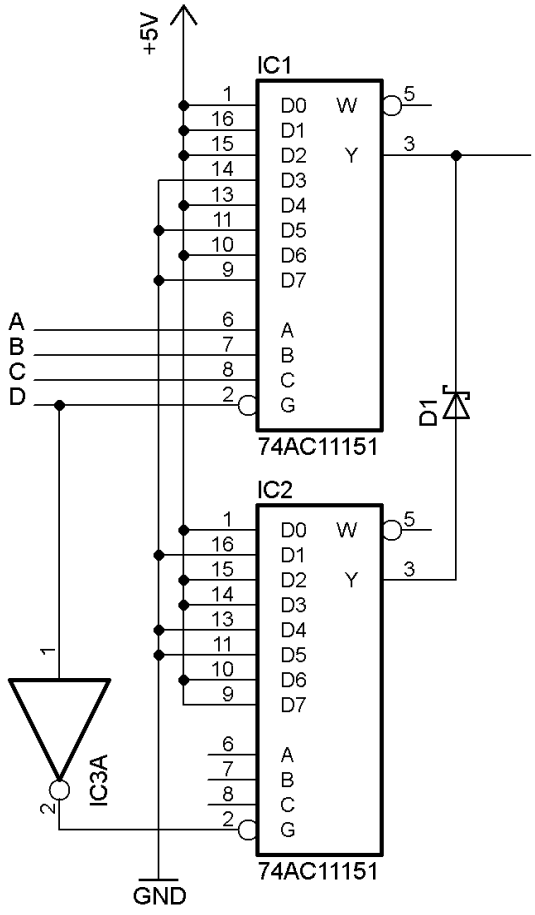
A multiplexer bemenetei pedig fixen +5v vagy föld pontra kell, hogy kötve legyenek. A bemenet sorszáma határozza meg a minterm számot. Amennyiben az adott szám szerepel a függvény felírásban (vagyis az igazságtáblázat adott sorában a függvény kimenete 1-es értékű), akkor a bemenetet tápfeszültségre, ha pedig nem szerepel földelésre kell kötni.

Egy négyváltozós logikai függvény megvalósítható 2db 8 bemenetű multiplexer segítségével vagy 4db 4 bemenetű segítségével. Ezen megvalósítások sem sokkal bonyolultabbak. A megvalósítást egy konkrét példán keresztül szemléltetni egyszerűbb. A példában a következő logikai függvény lesz megvalósítva 2db 8 bemenetű multiplexer segítségével. $\sum m_i^2(0,1,2,4,6,8,10,11,14,15)$

A megvalósítás első lépéseként a multiplexerek bemeneteit be kell kötni, mégpedig úgy, hogy az egyik multiplexer a függvény igazságtáblázatának első 8 sorát valósítsa meg, míg a másik a függvény második 8 sorát.

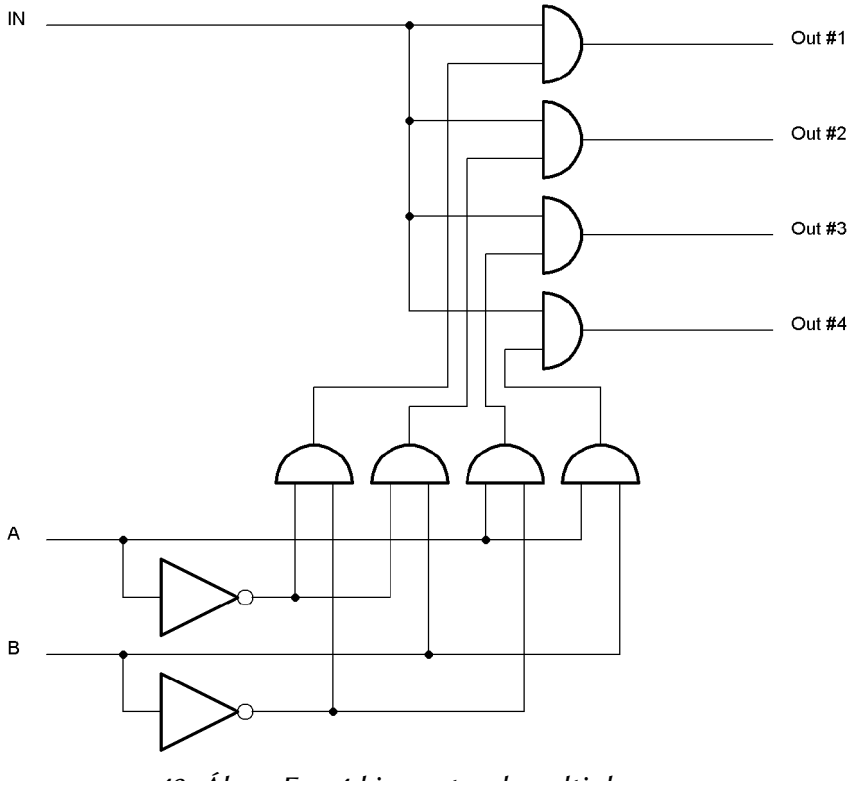
Ezután a megfelelő címzést kell megvalósítani. A gyártott multiplexerek többsége rendelkezik egy Enable jelllel, ami a multiplexer működését engedélyezi. Ezt felhasználva megvalósítható a címzés. A címzés megvalósításához érdemes még tudni, hogy a gyártott multiplexerek címzése esetén az A jelű változó a legkisebb helyiértékű. Ebből adódóan a címzésnél a két multiplexer A, B és C bemenetei összeköthetőek. Ezek továbbra is A, B és C bemenetek lesznek. A D változó (a legmagasabb helyiértékű bemeneti bit) fogja kijelölni, hogy éppen melyik multiplexer működhet.

Az alábbi ábrán látható kapcsolásban a multiplexerek engedélyező jele negált. A felső multiplexer valósítja meg a függvény igazságtáblázatának első 8 sorát, míg az alsó multiplexer az utolsó 8 sort. A két multiplexer kimenetét közvetlenül nem szabad összekötni (lásd áramkörök kimeneti típusai résznél) direktben. Vagy egy OR kaput kell alkalmazni a kimeneten, vagy az egyik multiplexer kimenetét védeni kell dióda segítségével.



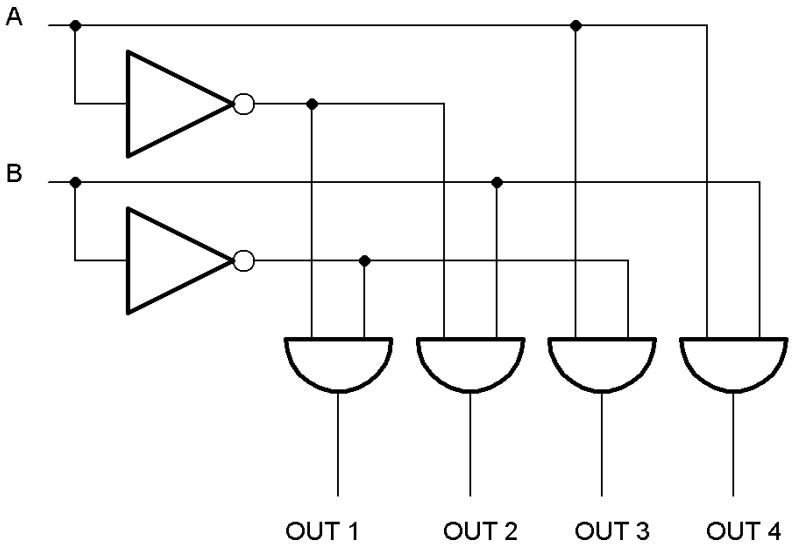
Demultiplexer

A demultiplexer, más néven dekóder a multiplexer ellentettje. Egy bemenetet képes szétosztani N vezérlő bement által meghatározott 2^N kimenet között. Együtt használva egy multiplexerrel útválasztásra használható. Ezen felül a demultiplexer alkalmazható még mikrovezérlők esetén kimeneti portbővítő áramkörként.



Cím dekóder

A cím dekóder egy olyan integrált áramkör, amely N bementi kombinációt felhasználva 2N kimenet közül egy időben csak egyet választ ki. Elsősorban memóriák illesztésénél használt áramkör, valamint kimenet bővítésre is alkalmas.

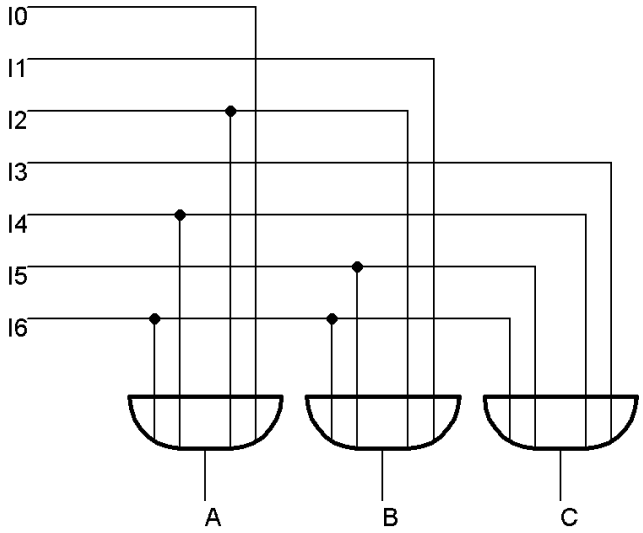


Kódoló

Az Kódóló a címdekóder ellentettje. N darab bemenettel rendelkezik, melyből egyszerre csak egy lehet aktív. Ebből állít elő egy bináris számrendszerben ábrázolható számot. Így N darab bemenet esetén $log_2 N$ kimenettel rendelkezik, amelyek egyértelműen beazonosítják a bemenetet.

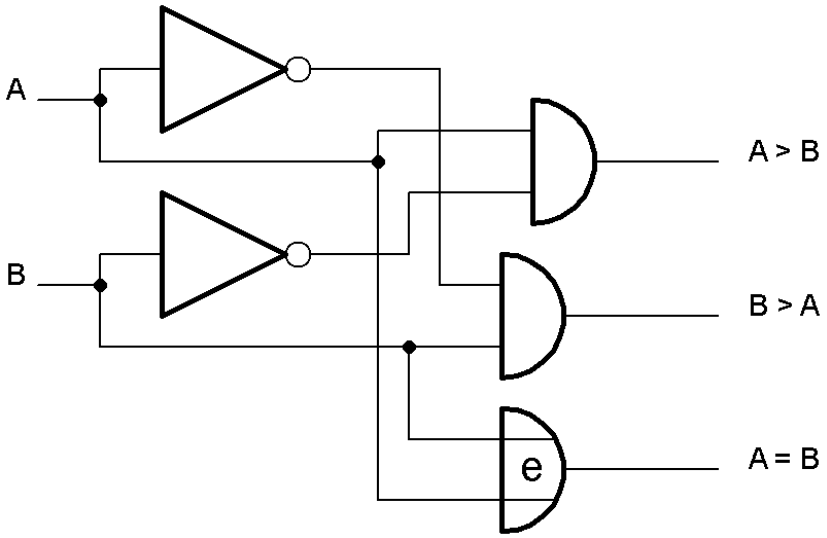
Az alábbi ábrán egy egyszerű 3 bites kódoló látható. A kódoló hibája, hogy nincs felkészítve arra az esetre, ha

egyszerre több bemenet aktív. Az ilyen hibákat az integrált áramkörként kapható kódolók esetén kiküszöbölik.



Komparátor

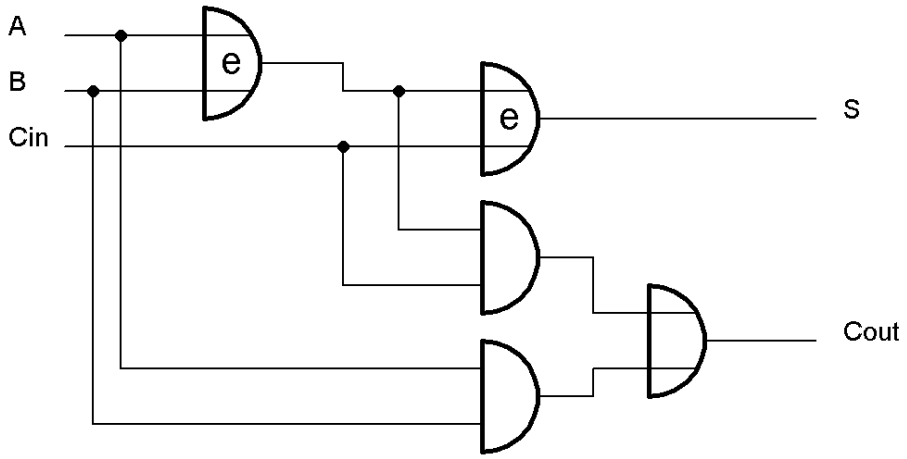
A digitális komparátor olyan kombinációs hálózat, amely a bemenetére adott két szám között relációs viszonyt képes felállítani, és ezt megmutatni a kimeneten. Egy komparátornak 3 tipikus kimenete van. Ezek A<B, B>A és A = B.



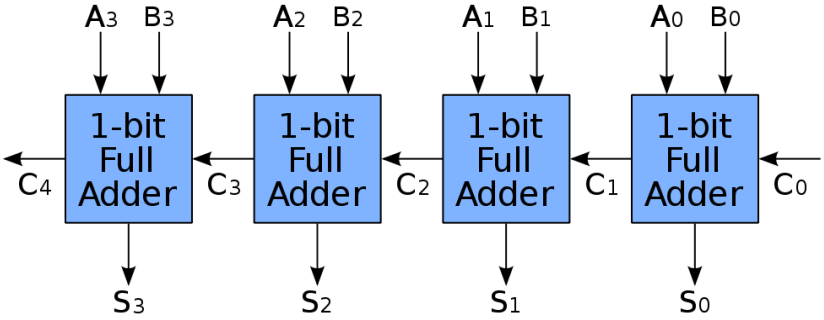
46. Ábra: 1 bites digitális komparátor

Összeadó, teljes összeadó

Az összeadó olyan áramkör, amely 2 darab bináris jegy összegét képes kiszámítani. Szokás fél összeadónak is nevezni ezen áramkört, mivel nem kapcsolható össze más összeadókkal, mivel nem vesz figyelembe bemeneti átvitelt. Az összeadó, amely ezt is figyelembe veszi, a teljes összeadó. Teljes összeadók segítségével megoldható két n bites szám összeadása is. Ehhez n darab teljes összeadóra van szükség.

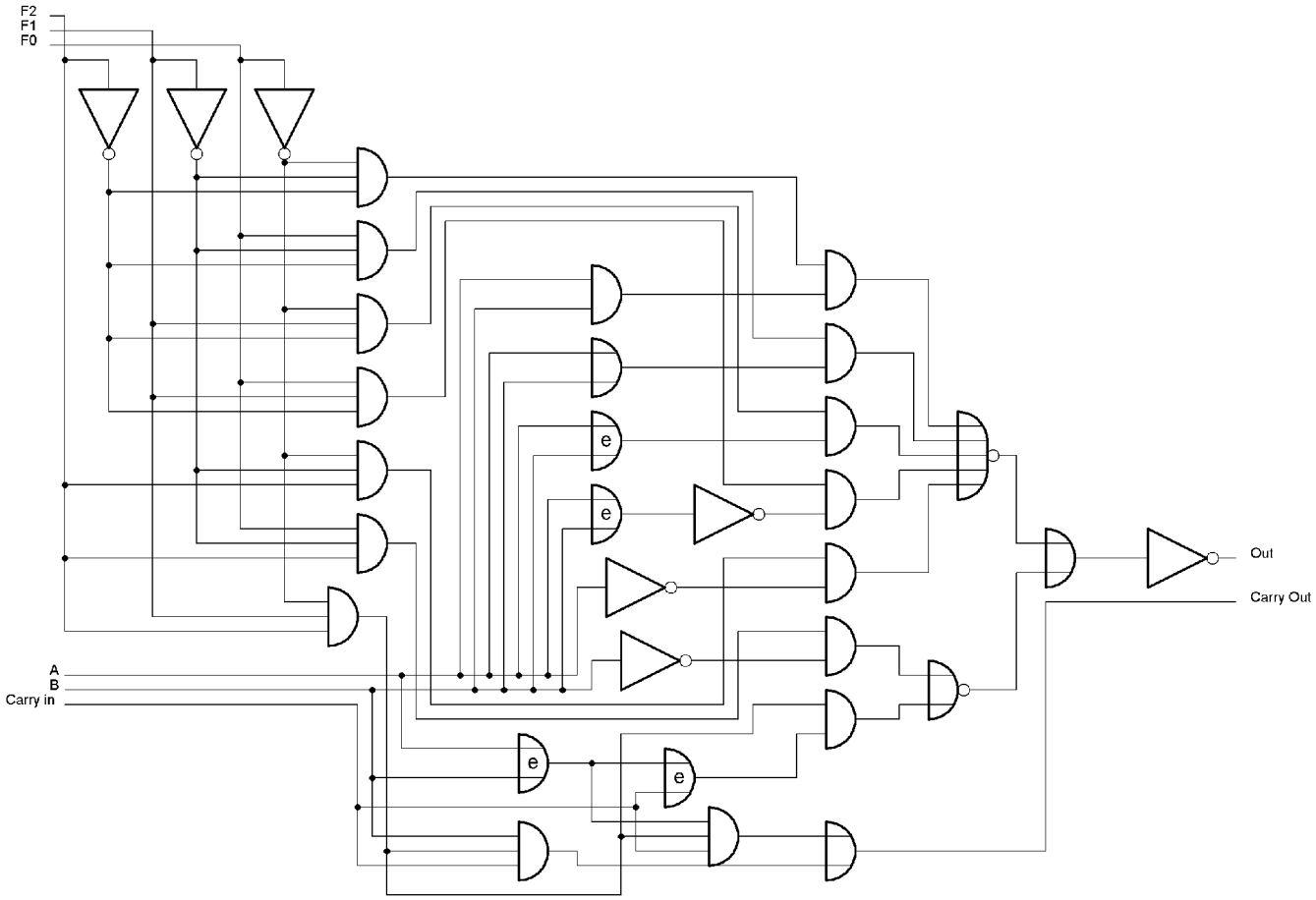


A teljes összeadókból kialakított összeadót a szakirodalom Ripple-Carry összeadónak nevezi, mivel az összeadó fokozatok egymás után hullámszerűen működnek. A kapcsolás problémája, hogy viszonylag lassú tud lenni nagy számok esetén, mivel a fokozatok egymás után működnek és az egyes fokozatok késleltetési ideje összeadódik.



Aritmetikai és logikai egység

Az aritmetikai és logikai egység (rövidítve ALU) minden processzorban megtalálható. Ez nem más, mint egy összetett műveletvégző egység. Minden ALU rendelkezik műveletkiválasztó bemenetekkel és adat bemenetekkel. A következő ábrán látható ALU 7db 2 bites művelet elvégzésére képes. A kezdetleges processzorokban lévő aritmetikai egység az ábrán láthatónál is jóval bonyolultabb. Az ábrán látható egység a következő műveletek elvégzésére képes: A AND B, A OR B, A XOR B, A EQ B, A + B, NOT A, NOT B. Aritmetikai és logikai egységet önálló integrált áramkörként nem gyártanak, de mivel minden mikrovezérlőben és processzorban megtalálható, ezért célszerű ismerni a felépítését. Az áramkör Carry In és Carry Out bemenetei több ALU összekapcsolására használhatóak.



Digitális – Analóg / Analóg – Digitális átalakítók

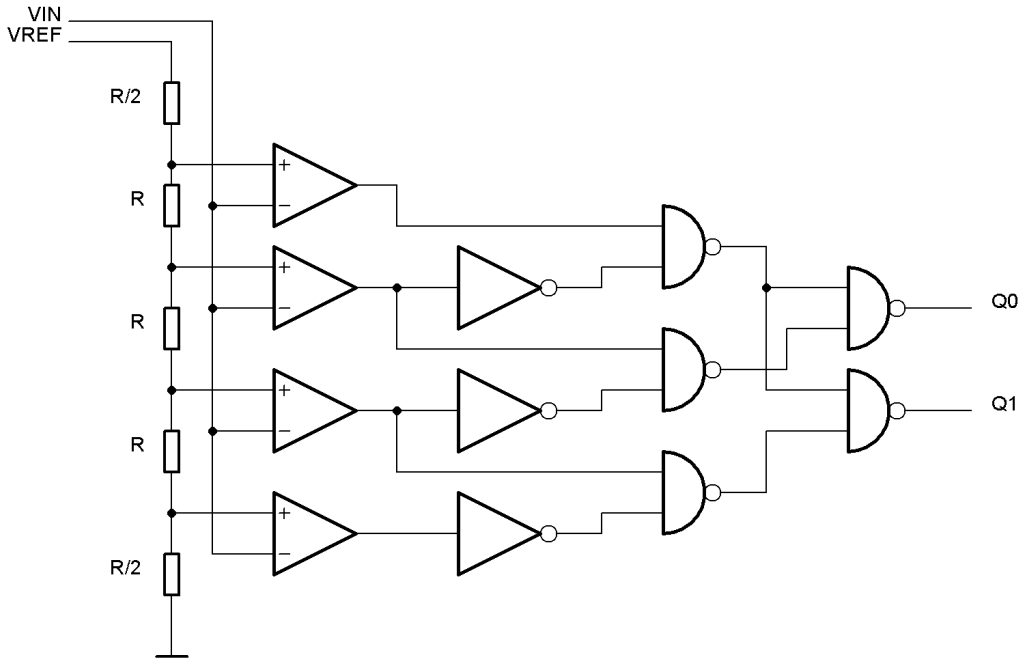
A digitális-analóg átalakítók (DAC) átjárást képeznek a digitális világból az analóg világ felé. Az analóg – digitális (ADC) átalakítók pedig analóg jelek digitalizálására szolgálnak. Mindkét áramkörtípusból számos változat létezik. Az egyszerűség kedvéért mindkét típusból a legegyszerűbb felépítést tárgyaljuk.

Mindkét áramkör esetén két fontos jellemező a felbontás és a mintavételezési sebesség. A felbontás ADC áramkörök esetén megadja, hogy hány különböző feszültség értéket tud megkülönböztetni az áramkör. DAC áramkörök esetén pedig azt adja meg, hogy a kimeneti feszültség mennyi lépésben tud változni. Értékét Bitben szokás megadni.

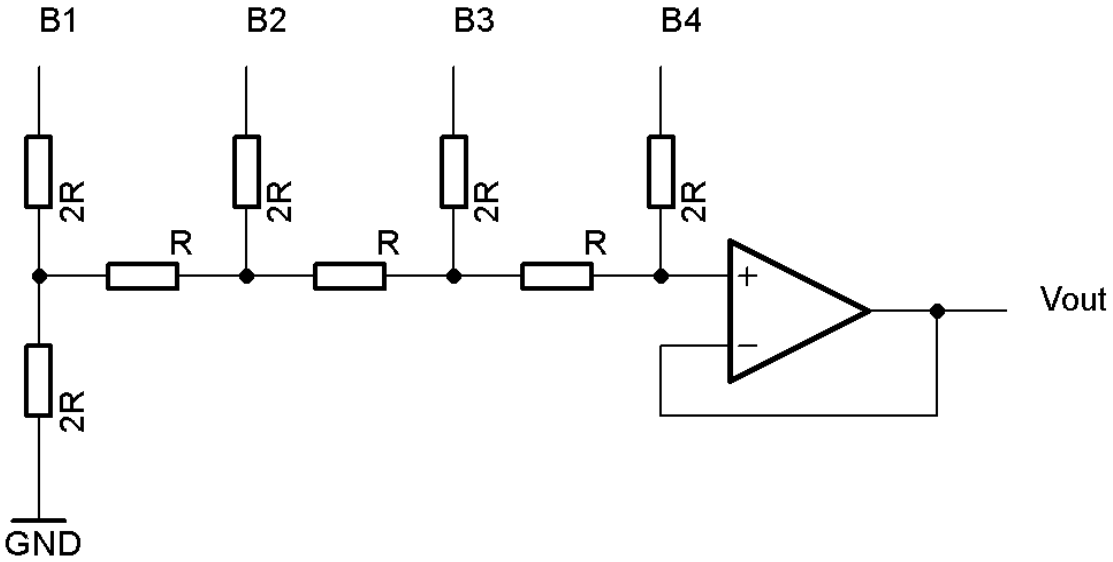
A mintavételezési sebesség azt mutatja meg, hogy másodpercenként mennyiszer képes az analóg jelből mintát venni az áramkör, DAC áramkörök esetén pedig azt mutatja meg, hogy egy másodperc alatt hányszor képes a kimeneti feszültség változni.

Az úgynevezett Flash Analóg-digitális átalakító a feszültségosztó és a komparátorok tulajdonságait használja ki. A referencia feszültség egy ellenállás hálózattal van leosztva. Erre a leosztott feszültségekre kapcsolódnak komparátorok, amelyek a bemeneti feszültséget hasonlítják a referenciához.

A komparátorok kimenetén megjelenő jelek bináris kódolása után meg is van a digitalizált érték. Ezen áramkörtípus hátránya, hogy a felbontás növelésével exponenciálisan nő az alkatrészek száma, viszont nagy mintavételezési sebesség érhető el.



A legegyszerűbb DAC áramkör szintén feszültségosztókra épül. A kapcsolás neve R-2R létra. Ebben a kapcsolásban két értékű ellenállás szerepel, amelyek a kimeneti feszültséget határozzák meg. Az alábbi kapcsolásban egy 4 bites R-2R létra található. Az áramkör végén szereplő követő erősítő szerepe az, hogy az átalakítás pontossága ne másszon el egyik irányba sem, ha a kapcsolásra terhelést kapcsolunk.



A kapcsolás előnye, hogy a bemenetek száma könnyen növelhető. A gyakorlati megvalósítás hátránya, hogy az ellenállások értékei között lesznek eltérések, amelyek apróbb hibákat okoznak az átalakításban. A kimeneti feszültség az alábbi képlet segítségével határozható meg:

$$V_{out} = \frac{B1}{16} + \frac{B2}{8} + \frac{B3}{4} + \frac{B4}{2}$$

PWM jelgenerátor is alkalmazható digitális-analóg átalakításra, méghozzá úgy, hogy a kimeneti PWM jelre tenni kell egy alul áteresztő szűrőt, ami a nagy frekvenciás négyszögjelből feszültség értéket képez. A PWM jelekről később lesz szó, részletesebben az Arduino programozása szekcióban.

Ezen megoldás hátránya az, hogy valamennyi váltakozó összetevője marad a kimeneti feszültségnek, ami zajként jelentkezik. Továbbá az alul áteresztő szűrő felépítéséből adódóan a kondenzátor késleltetést jelent, így az ilyen típusú DAC áramkörökkel nem nagy sebesség érhető el, ha a kondenzátor értéke magas. Viszont olcsón megvalósítható, főleg mikrovezérlős környezetben, hiszen a legtöbb vezérlő rendelkezik PWM kimenettel.

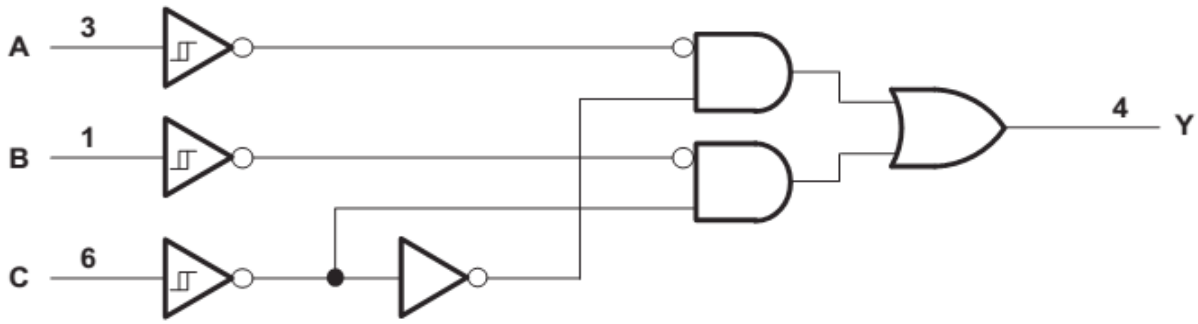
Univerzális kapu áramkörök

Az univerzális kapuáramkörök olyan három vagy négy bemenetes kombinációs hálózatok, amelyek a bemenetek konfigurálásával többféle kapu működését képesek megvalósítani. A három bemenetes hálózatok esetén attól függően, hogy az egyik bemenetet földre vagy tápfeszültségre kötjük más működést fog produkálni a hálózat.

A négy bemenetes hálózatok esetén a negyedik bemenet a kimeneti buffer engedélyező jele, az ilyen hálózatok három állapotú kimenettel rendelkeznek.

Számos gyártó gyárt ilyen áramköröket eltérő belső felépítéssel és igazság táblázatokkal. Ebből adódóan az egyes univerzális kapuáramkörökből eltérő kapuk alakíthatóak ki.

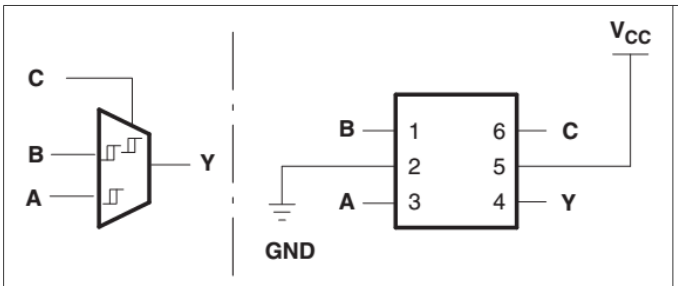
Példaképpen az SN74AUP1T97 áramkört elemezzük ki. Az IC belső felépítése és igazság táblázata az alábbi ábrákon látható.



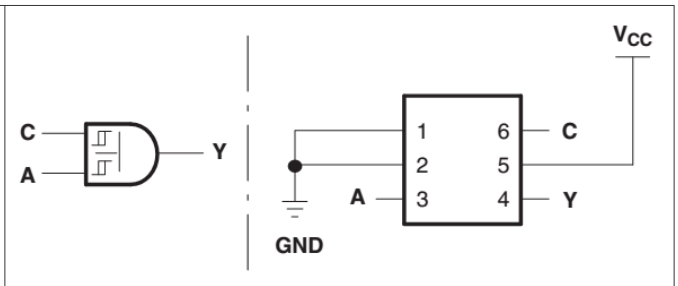
Bemenetek			Kimenet
C	B	A	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

35. Táblázat: Az áramkör igazság táblázata

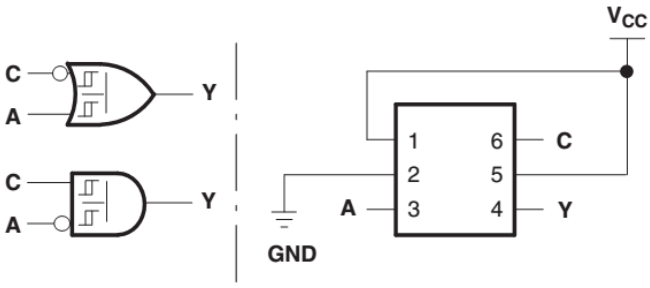
Az IC sokféleképpen alkalmazható, az adatlap által tárgyalt alkalmazási lehetőségek az alábbi táblázat foglalja össze.



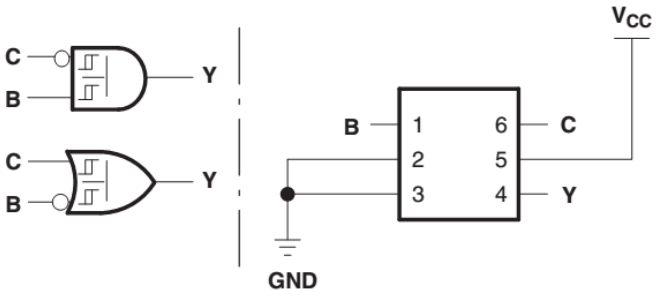
Két bemenetes multiplexer



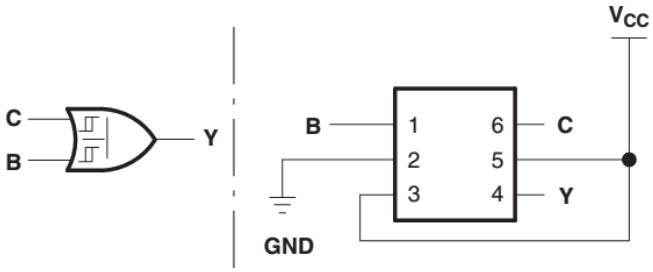
ÉS kapu



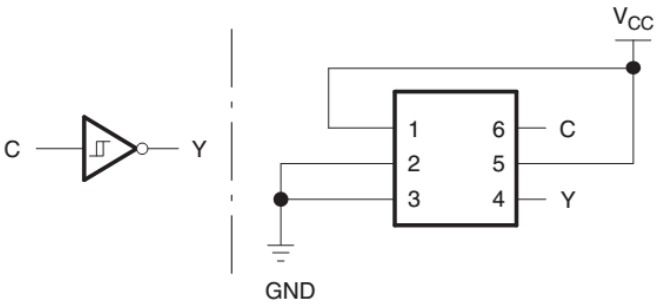
VAGY/NAND kapu egy invertált bemenettel



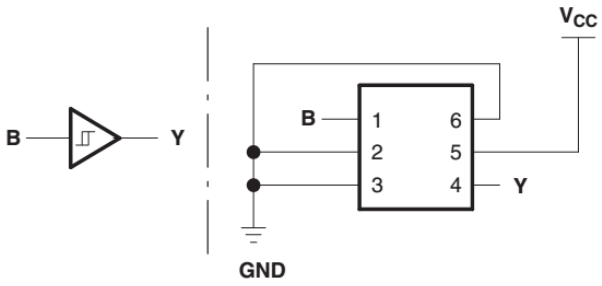
ÉS/NOR kapu egy invertált bemenettel



VAGY kapu



Inverter



Buffer

36. Táblázat: SN74AUP1T97 alkalmazási lehetőségei

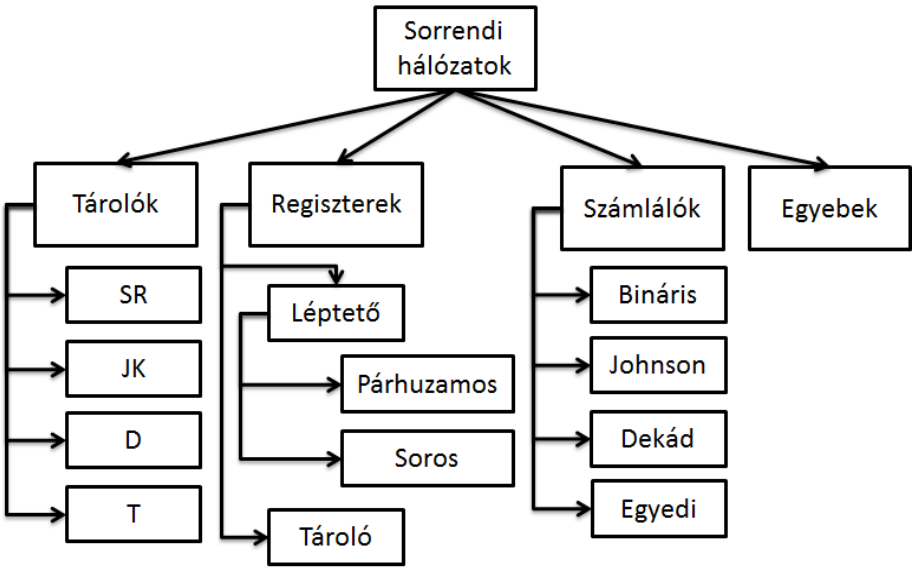
Sorrendi hálózatok

Az eddigiekben tárgyalt áramkörök és kapcsolások mindegyike időfüggetlen, vagyis az ilyen típusú áramkörök esetén a kimenetek értéke csak a bemeneti kombinációktól függ. Ezért az ilyen típusú áramkör hálózatokat kombinációs hálózatoknak nevezzük.

Amennyiben a kimenet pillanatnyi értéke függ a hálózat előéletétől, akkor a hálózatunk sorrendi hálózat, mivel azonos bemeneti kombinációk mellett eltérő időpillanatban különböző kimeneteket produkál a hálózat.

Az ilyen hálózatok megvalósításukban tartalmaznak minden esetben valamilyen fajta memóriát, ami az előző állapotok tárolására szolgál.

A sorrendi hálózati áramkörök működési funkcióik szerint csoportosíthatóak. Ez alapján a hálózatok lehetnek tárolók, regiszterek és számlálók, illetve egyéb típusúak. Az egyéb típusba az olyan komplex áramkörök tartoznak, amelyek több alap típus keverékének felelnek meg.

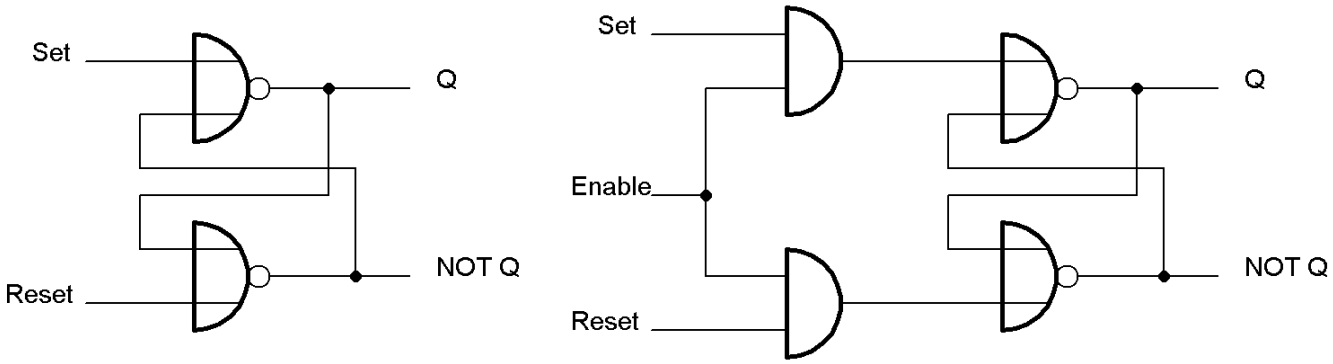


Tárolók

A tároló áramkörök olyan elemi sorrendi hálózatok, amelyek két stabil állapottal rendelkeznek, és egy bit információ tárolására képesek. A tároló áramkörök típusai megkülönböztethetőek logikai működésük és vezérlésük szerint.

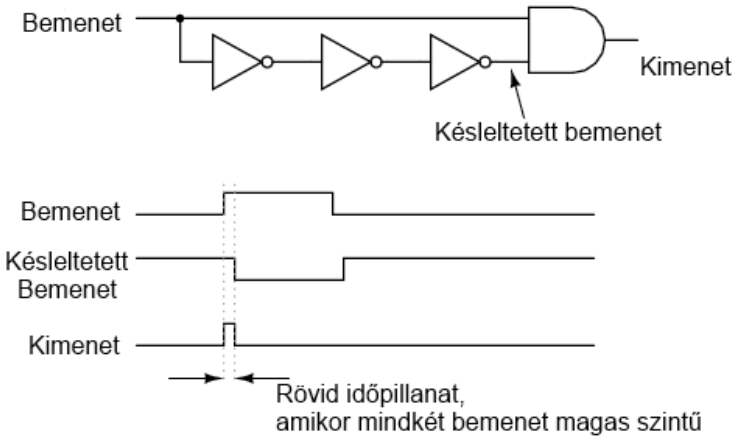
Logikai csoportosítás alapján létezik SR, JK, T és D típusú tároló, míg vezérlési elv szerint léteznek aszinkron (együtemű) és szinkron (kétütemű) tárolók.

Az aszinkron tárolók jellemzője, hogy a bemenetre adott megfelelő jelek hatására szinte azonnal (áramköri késleltetések persze számítanak) létrejön a kimeneti változás. A szinkron működésű tárolóknál a kimeneti változás a megfelelő bemeneti jelek és az órajel hatására jön létre. Az órajel az engedélyező jel szerepét tölti be.

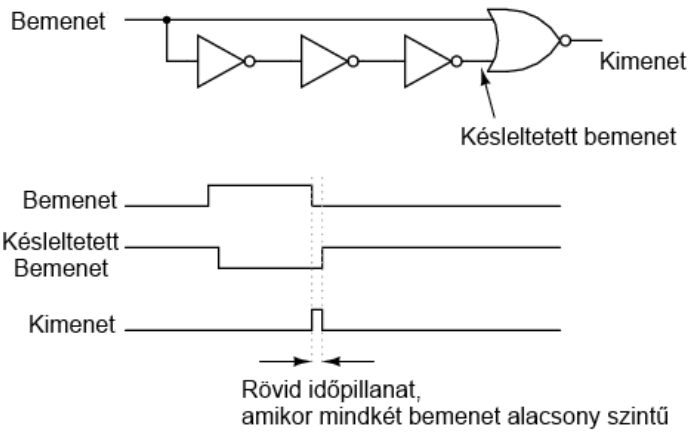


Az órajel lehet elvezérelt. Az élvezérléssel a szinkron tárolók azon kedvezőtlen tulajdonságát szokták kiküszöbölni, hogy a bemenetek az órajel magas jelszintjének teljes időtartalma alatt hatásosak. Vannak lefutó és felfutó élre reagáló tárolók. A felfutó élre reagáló tárolók az órajel alacsony → magas átmenetére reagálnak, míg a negatív élvezéreltek a magas → alacsony átmenet esetén reagálnak. Vannak olyan szinkron tárolók is, amelyek az órajel magas jelszintjének teljes időtartalma alatt reagálnak a bemenetekre. Ezen tárolók esetén CP (Clock Pulse, órajel) jelzés helyett E (Enable, engedélyezés) jelzés szerepel a tárolókon.

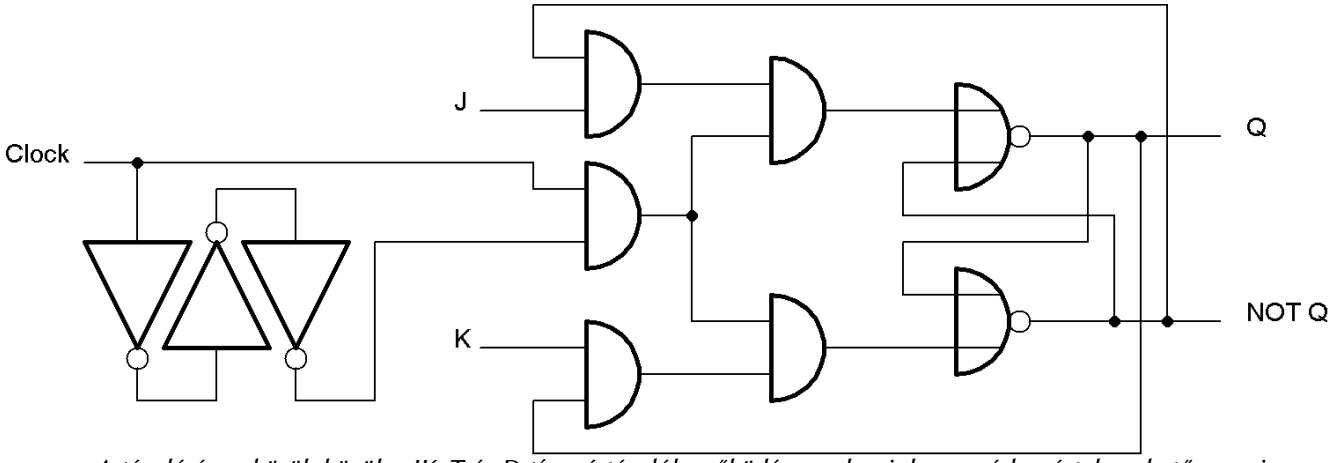
Az élvezérlés könnyen megoldható inverterek és kapuk segítségével kihasználva a hazárd jelenséget.



A fenti ábrán látható egy felfutó él érzékelésre alkalmas áramkör és a működése. Hasonlóan oldható meg a lefutó él érzékelése is, annyit kell módosítani a kapcsoláson, hogy a kimenetet előállító AND kaput kicseréljük NOR kapura. Ekkor a kimeneten impulzus a bemeneti jel végén jelenik meg.



Tároló áramkörök esetén az élvezérlést ezen ismertetett megoldások helyett inkább Master-Slave kialakításban valósítják meg, ennek előnye, hogy az integrált áramkör gyártásakor csak egyfajta áramköri elemet kell beleintegrálni.



A tároló áramkörök közül a JK, T és D típusú tárolók működése csak szinkron módon értelmezhető, vagyis ezen tárolókból aszinkron változat nem létezik. Az SR tároló működése értelmezhető aszinkron és szinkron módon is. Gyakorlatban általában csak aszinkron változatát használják.

A tároló áramkörök és a sorrendi hálózatok leírását állapotgráffal szokás leírni. A tárolók működésének leírása során használt állapotgráf felépítése a következő:

57. ábra: A tárolók leírása során használt állapotgráf felépítése

58. ábra: SR tároló állapotgráfja

59. ábra: JK tároló állapotgráfja

60. ábra: D tároló állapotgráfja

61. ábra: T tároló állapotgráfja

Állapot gráfok mellett a tárolók és a sorrendi hálózattok működése megadható vezérlési táblázat segítségével is. A vezérlési táblázat tervezés során alkalmazható kiválóan. Felépítése igen egyszerű. Egy általános vezérlési táblázat felépítése az alábbi táblázaton látható:

Jelenlegi kimeneti állapot	***Következő kimeneti állapot***	***A következő kimeneti jel előállításához szükséges vezérlőjelek***

37. Táblázat: Vezérlési táblázat felépítése

Az elemi tárolók vetélési táblázatai:

Q_t	Q_{t+1}	***R***	***S***
0	*0*	*X*	*0*
0	*1*	*0*	*1*
1	*0*	*1*	*0*
1	*1*	*0*	*X*

38. Táblázat: SR tároló vezérlési táblázata

Q_t	Q_{t+1}	***J***	***K***
0	*0*	*0*	*X*
0	*1*	*1*	*X*
1	*0*	*X*	*1*
1	*1*	*X*	*0*

39. Táblázat: JK tároló vezérlési táblázata

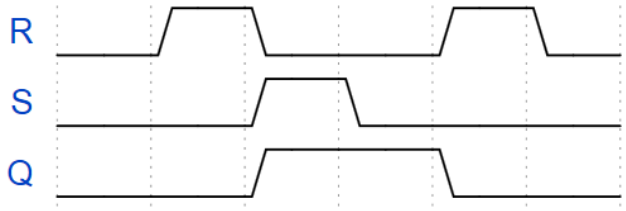
Q_t	Q_{t+1}	***T***
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

40. Táblázat: T tároló vezérlési táblázata

Q_{t+1}	***D***
0	*0*
1	*1*

41. Táblázat: D tároló vezérlési táblázata

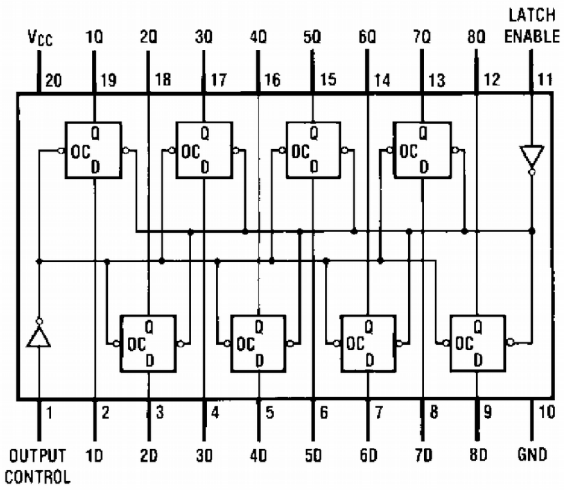
A hálózatok leírásának másik módszere az idődiagram felvétele. Ez egy egy grafikus módszer, amiből könnyen megállapítható az áramkör viselkedése. Az alábbi ábrán a SR tároló működése látható idődiagramon.



Az idődiagram nem csak a hálózat megértését segíti, segítségével a hazárd jelenségek forrása is beazonosítható. Számos szoftver létezik ilyen diagramok készítésére. Az általam legegyszerűbben használhatónak ítélt szoftver a wavedrom. Ez egy böngészőből futó program, így telepíteni sem kell, csak meg kell látogatni a http://wavedrom.googlecode.com/svn/trunk/editor.html oldalt.

Regiszterek

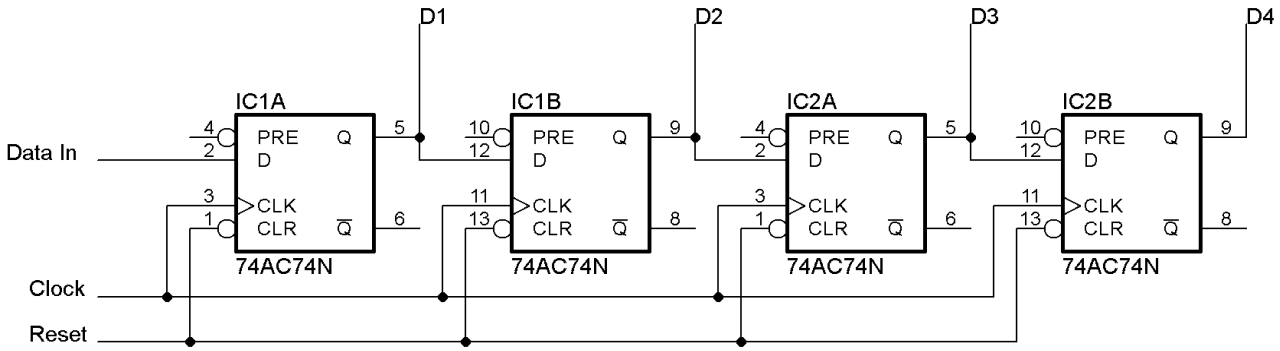
A regiszterek információ tárolására szolgáló áramkörök. Pár darab azonos típusú tárolóból épülnek fel. A tárolókba egyszerre írható adat, illetve egyszerre olvasható ki az összes tároló tartalma. A közös összehangolt működést a közös órajel biztosítja.

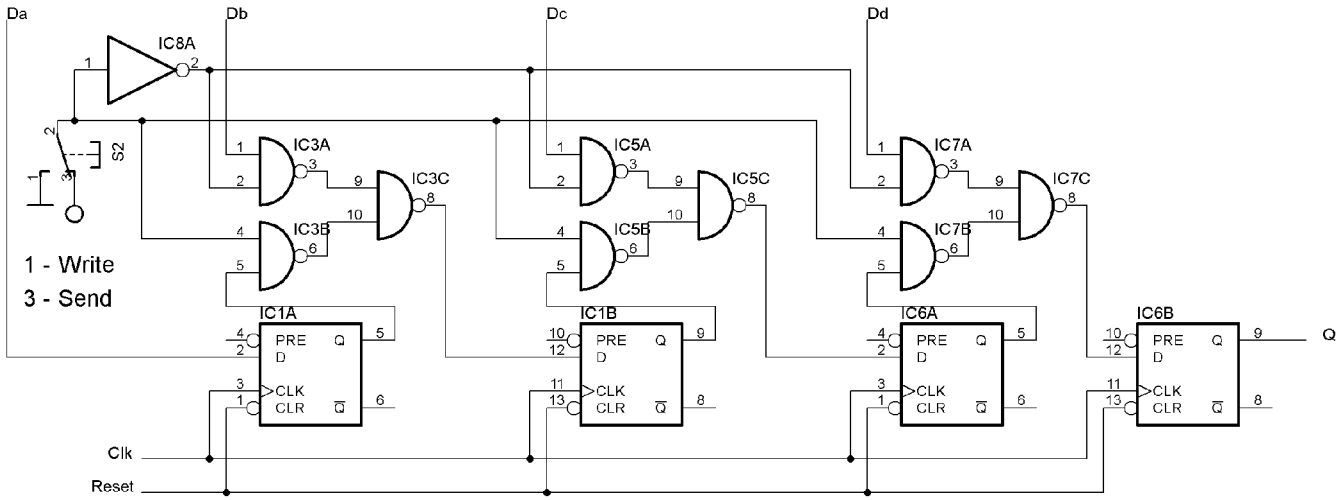


A léptető regiszterek sorosan beérkező információt alakítanak át párhuzamosan értelmezhető adattá, vagy párhuzamosan beérkező adatot alakítanak vissza soros adattá. Alapvető konverziós elemek a soros és párhuzamos adattovábbítás megvalósításában.

A párhuzamos adatátvitel során a buszrendszer szélességének megfelelő mennyiségű adatot viszünk át egy órajel ciklus alatt. Ennek a módszernek az előnye az, hogy az információ átvitel gyors tud lenni, viszont a buszrendszer szélességének növelésével egyre problémásabb a kivitelezés, mert egy 32 bites rendszer esetén legalább 33db vezeték (32 jel és egy órajel) kell az átvitelkor.

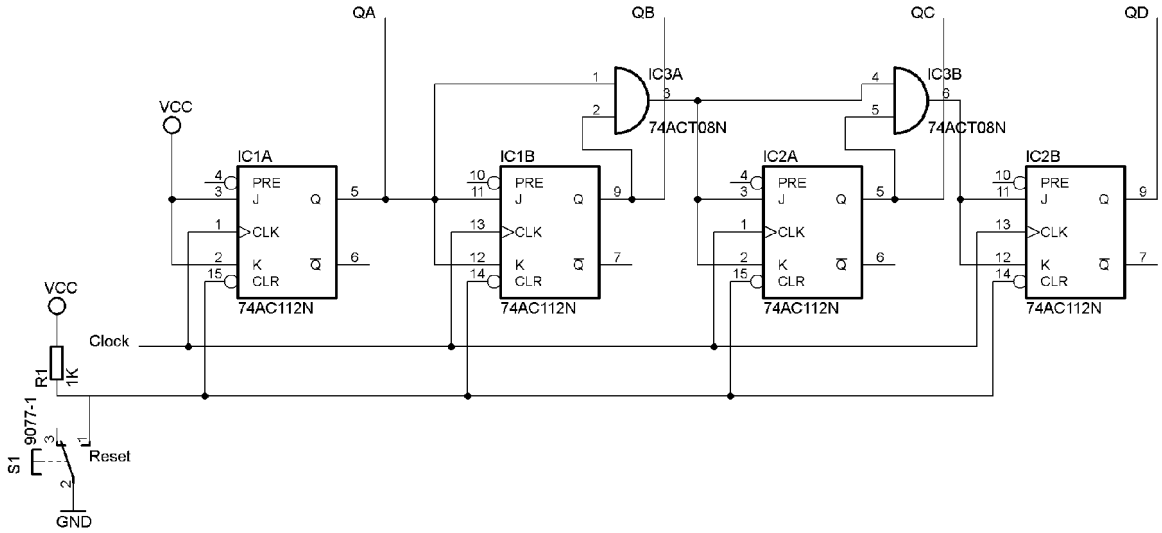
Soros adattovábbítás esetén a buszrendszert bitjeit egyesével sorban továbbítjuk. Így a buszrendszer sávszélességétől függetlenül elég két vezeték. Az adat és az órajel. Hátránya a módszernek, hogy lassabb, mint a párhuzamos átvitel. Példaképpen 32 bit adatot egy párhuzamos rendszeren 1 órajel ciklus alatt tudunk átvinni, míg soros átvitel esetén ez 32 órajelet vesz igénybe. Ebből adódóan azonos sebesség eléréséhez a soros rendszernek 32x nagyobb órajelen kell üzemelnie.




Da
Db
Dc
Dd
IC8A
S2
1 - Write
3 - Send
IC3A
IC3B
IC3C
IC5A
IC5B
IC5C
IC7A
IC7B
IC7C
IC1A
IC1B
IC6A
IC6B
PRE
D
CLK
CLR
Q
Clk
Reset

Számlálók

A számlálók olyan áramkörök, amelyek a kimeneteiken jelzik valamilyen kódolt formátumban a bemenetként kapott órajelpulzusok számát. A számlálás minden esetben egy tárolásból és egy aritmetikai műveletből áll. A tárolt értékek lényegében a kimenetek, amelyekhez különböző kódrendszereket társíthatunk. Integrált áramkörként a legfontosabb létező számlálók a bináris és a dekád számlálók. A dekád elnevezés arra utal, hogy a számláló egység egy decimális helyi értéket valósít meg.



A számlálók fontos jellemzője a számlálási irány. Ez lényegében nem más, mint az aritmetikai művelet típusa. Az előre számláló minden egyes órajelpulzus hatására növeli a tárolt értéket, a hátrafelé számláló csökkenti. Léteznek kétirányú számlálók is, amelyeknél egy vezérlőjel segítségével dönthető el, hogy előre, vagy hátrafelé számoljon az áramkör.

A tárolt értékek növelése csak a számláló maximális kapacitásáig tud menni. Ezt átlépve a számlálás újrakezdődik. Egy n bites bináris számláló maximális értékének eléréséhez szükséges N lépésszám a következő összefüggés alapján számolható: $N=2^n$ *. Dekád számlálók esetén a maximális érték eléréséhez szükséges lépésszám, ha a számláló d darab dekáddal rendelkezik:* $N=10^d$

Számláló áramkörök megvalósíthatók szinkron és aszinkron módon is. Az aszinkron számláló felépítés jelentősen lecsökkenti a maximális számlálási frekvenciát. Egy n darab tárolóból kialakított aszinkron számláló maximális számlálási frekvenciája a következő képlettel határozható meg: $f_{Max}=\frac{1}{n\cdot t_{pd}}$ *, ahol* n *a tárolók darabszáma,* t_{pd} *pedig egy tároló késleltetése. Szinkron számláló esetén a képlet a következőképpen alakul:* $f_{Max}=\frac{1}{t_{pd}+n\cdot t_k}$ *, ahol* t_{pd} *szintén egy tároló késleltetése,* t_k *pedig egy kapu késleltetése. A képletben szereplő* n *itt a használt kapuk számát jelöli.*

Egyedi számlálók kialakítása is lehetséges. Ennek első lépése, hogy elkészítjük a számláló vezérlési táblázatát. A vezérlési táblázat minden egyes állapotváltozásához rögzítjük a használt tárolók vezérlő jeleit, majd a vezérlési kódokat egyszerűsítjük. Az egyszerűsítés után kapott függvényekkel meg vezérejük a tároló áramköröket.

Az alábbi példában egy 4 bites számlálót készítünk, el, amely kimenetén a prím számok jelennek meg. Vagyis négy biten az alábbi számjegyek bináris kódjai fognak megjelenni: 0, 1, 2, 3, 5, 7, 11, 13. A 13-as számjegy elérése után a számláló 0 állapotra áll be. A számláló JK tárolók segítségével lesz megvalósítva.

Jelenlegi állapot				**Következő állapot**				**A**		**B**		**C**		**D**	
A	**B**	**C**	**D**	**A**	**B**	**C**	**D**	**J**	**K**	**J**	**K**	**J**	**K**	**J**	**K**
0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1

0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
0	0	1	1	0	1	0	1	0	x	1	x	x	1	x	0
0	1	0	1	0	1	1	1	0	x	x	0	1	x	x	0
0	1	1	1	1	0	1	1	1	x	x	1	x	0	x	0
1	0	1	1	1	1	0	1	x	0	1	x	x	1	x	0
1	1	0	1	0	0	0	0	x	1	x	1	0	x	x	1

42. Táblázat: A számláló vezérlési táblázata

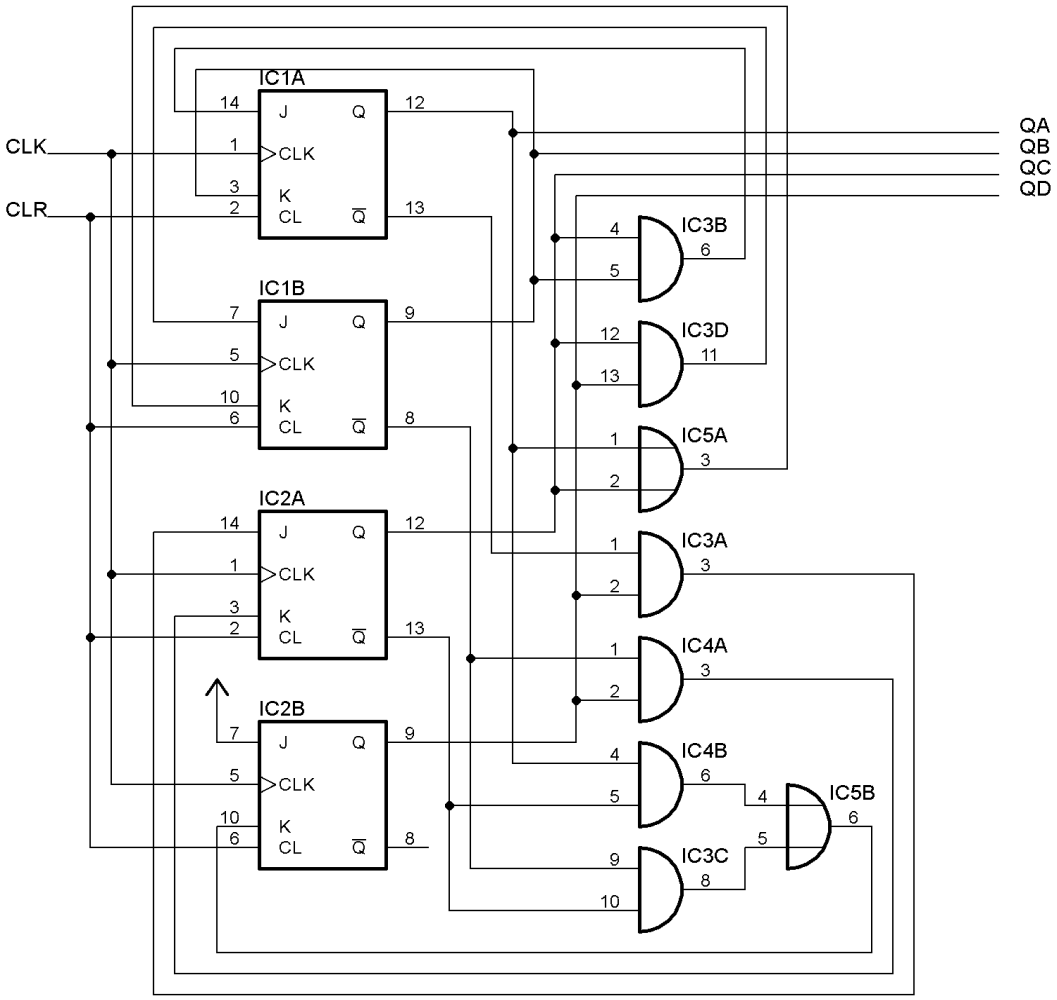
A vezérlési táblázat minden egyes bit változás mellé rögzítetten tartalmazza a JK tárolók vezérlő jeleit. Ez alapján kitölthető minden vezérlőjelhez tartozó minterm táblázat. A kitöltött minterm táblázatok:

63. Ábra: A tárolók vezérlő jelei minterm táblázatokban

Az egyszerűsítés után a következő vezérlő függvényeket kapjuk:

$$J_A = C \cdot B \; K_A = B$$
$$J_B = C \cdot D \; K_B = A + C$$
$$J_C = -A \cdot D \; K_C = -B \cdot D$$
$$J_D = 1 \; K_D = A \cdot -C + -B \cdot -C$$

A vezérlési függvények alapján elkészíthető a kapcsolási rajz:


IC1A
IC1B
IC2A
IC2B
IC3A
IC3B
IC3C
IC3D
IC4A
IC4B
IC5A
IC5B
J
K
CLK
CL
Q
CLK
CLR
QA
QB
QC
QD

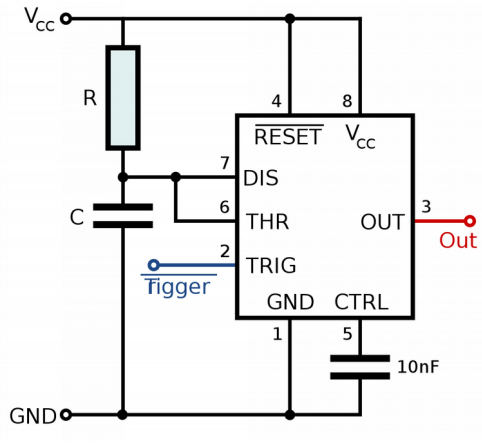
Órajel előállítása

Szinkron hálózatok esetén igen fontos a megfelelő órajel előállítása, mivel ez vezérli részben a hálózatot. Különböző módszerekkel tudunk órajelet előállítani. A legegyszerűbb megoldás, ha az impulzusokat manuálisan, egy gombnyomás segítségével adjuk.

Itt némi cselt kell alkalmaznunk, mivel egy kapcsoló a kontaktusok csúszása miatt több órajel impulzust is adhat egy gombnyomás hatására. Ezt nevezzük pergésnek. Ezen kedvezőtlen jelenség kiküszöbölhető egy monostabil multivibrátor beiktatásával. A monostabil multivibrátorok jellemzője, hogy egy stabil állapotuk van, ami általában a kikapcsolt állapot. Külső jel hatására átváltanak egy másik állapotba, amiből egy idő eltelte után visszaváltanak a stabil állapotba.

Ilyen áramkör könnyen építhető a klasszikus 555-ös időzítő áramkör segítségével, az alábbi kapcsolási rajz szerint:

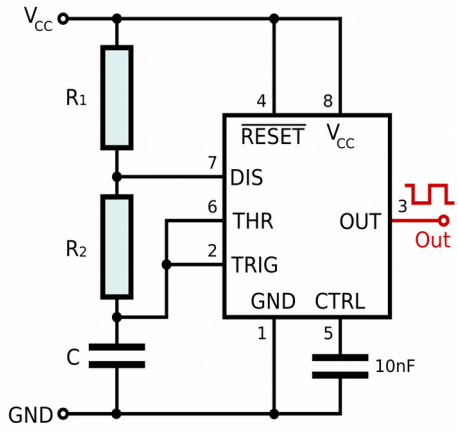
65.



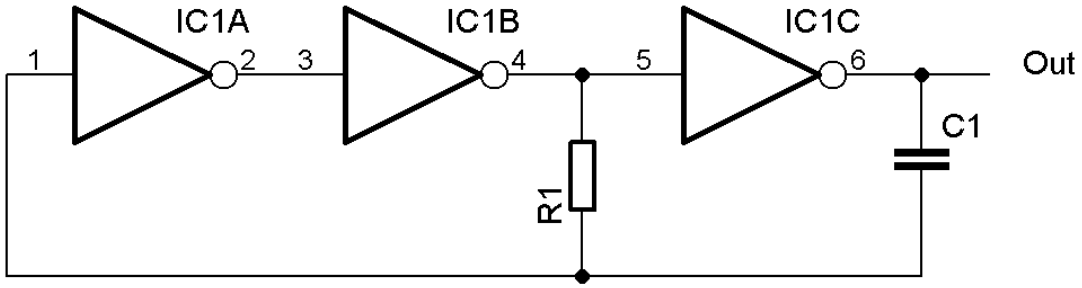
Ábra: Monostabil 555 kapcsolás

A Trigger láb földelése esetén az Out láb átvált t időre Vcc tápfeszültségre. A t idő hossza a következőképpen határozható meg: $t = 1{,}1 \cdot R \cdot C$

Amennyiben nem gombnyomások segítségével szeretnénk üzemeltetni az áramkörünket, akkor szintén az 555-ös felhasználásával készíthetünk astabil multivibrátort is. Az astabil multivibrátor kimenete két két állapot között váltakozik periodikusan.



Az astabil multivibrátorok kimenete négyszögjel, amely megfelelő órajelnek. A kapcsolás által generált órajel frekvenciája a következő módon határozható meg: $f = \frac{1}{ln(2) \cdot C \cdot (R_1 + 2 \cdot R_2)}$ *A generált négyszögjel nem szimmetrikus, a kikapcsolt és bekapcsolt állapotok ideje eltér. Ez kifejezetten hasznos tud lenni, ha PWM jelet generálunk, a legtöbb integrált áramkör esetén ez nem okoz problémát az élvezérlés miatt. A kikapcsolt és bekapcsolt*



állapotok ideje a következő képletek segítségével határozható meg: $t_{kikapcsolt}=ln(2)\cdot R2\cdot C$, $t_{bekapcsolt}=ln(2)\cdot(R_1+R_2)\cdot C$

Amennyiben nem szeretnénk alkalmazni egy külön integrált áramkört órajelt tudunk előállítani egy kondenzátor, ellenállás és két inverter segítségével is.

Ebben a kapcsolásban az órajel frekvencia a következőképpen határozható meg: $f=\frac{1}{1{,}4\cdot R\cdot C}$

Ezen órajel előállítási módok problémája, hogy hőmérséklet függőek, mivel az ellenállások és a kondenzátorok értékei is azok. Továbbá figyelembe kell venni azt is, hogy az ellenállásoknak és kondenzátoroknak van némi gyártási hibájuk, így az ilyen módon előállított órajel nem lesz sosem pontos. Továbbá ezen áramkörök segítségével maximum pár kHz frekvenciájú órajelet tudunk előállítani.

Ennél nagyobb vagy pontosabb órajel esetén rezgőkvarcot kell alkalmazni. A rezgőkvarc álltál generált órajel frekvenciáját csak a kvarc anyagi összetétele határozza meg. Az órajel frekvencia szorozható és osztható is, így a kvarc órajelétől eltérő frekvenciák is előállíthatóak.

Az osztás műveletet egy bináris számlálóval szokták elvégezni, mivel egy bináris számláló felfogható frekvencia osztóként is. A számláló legkisebb helyi értékű bitje minden órajel változás hatására állapotot vált, az egyel nagyobb helyi értéken lévő már csak minden második órajel változás esetén vált, a harmadik legkisebb helyi értékű kimenet már csak minden negyedik órajel hatására váltja állapotát. Egy n bites számláló n-edik kimenete a $\frac{1}{2^{n-1}}$ *-edik órajel hatására fog értéket váltani.*

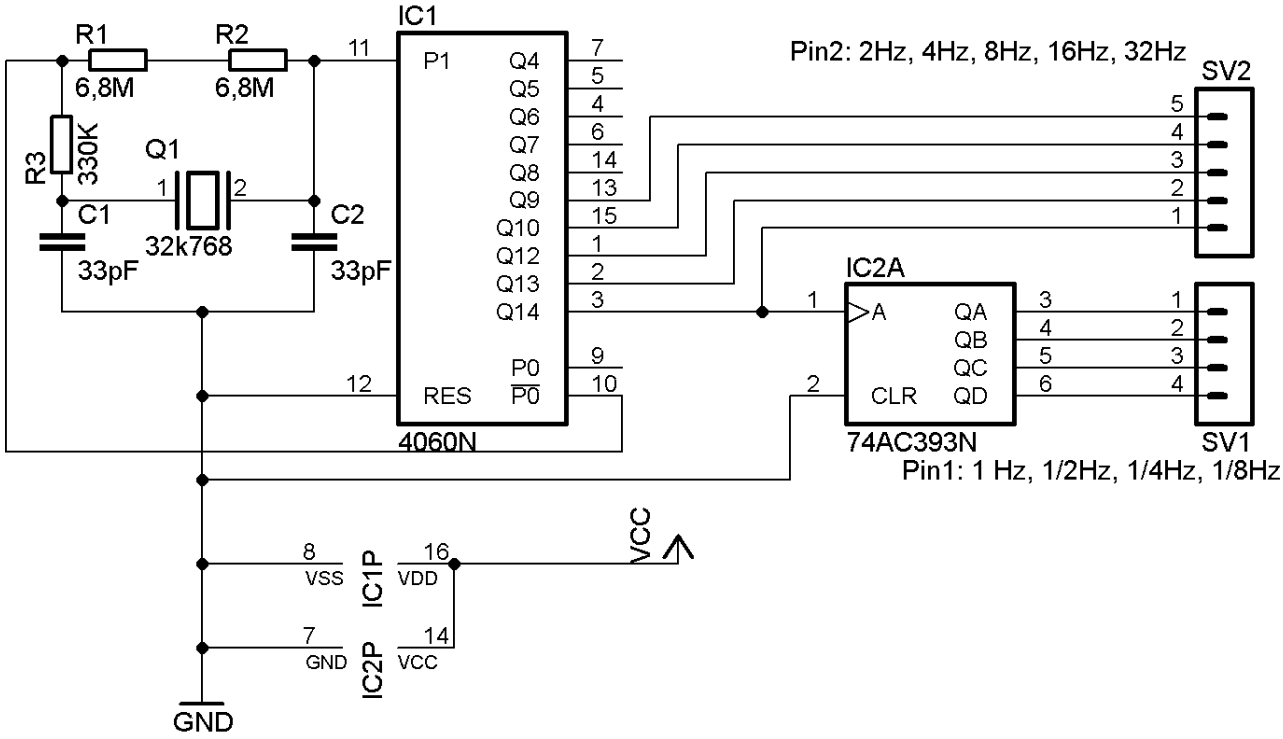
A kvarc órákban a pontos 1Hz impulzusok előállítását egy 32,768kHz-es rezgőkvarc és egy 16 bites számláló segítségével szokták megoldani.

Az órajel duplázás, szorzás művelet egyszerűen nem oldható meg, speciális integrált áramköröket igényel a pontos jel előállítása. Ezeket leginkább számítógépekben és olyan helyeken alkalmazzák, ahol több száz vagy ezer MHz órajel kell.

A pontosság miatt a rezgőkvarc és a szorzó áramkör egy áramköri elembe kerül integrálva. Az ilyen precíziós rezonátorok általában programozhatóak, ami lehetővé teszi azt, hogy a számítógépek teljesítménye igény szerint változtatható legyen.

Rezgőkvarc segítségével előállított órajelhez szükségünk lesz egy 4060-as integrált áramkörre, amely bemenete rezgőkvarc fogadására alkalmas. Ez egy 14 bites bináris számláló, amely képes az órajel osztására is. Amennyiben további osztásra van szükségünk további számlálók csatolásával megoldható.

Az alábbi ábrán egy 0,125 Hz és 32Hz között működő órajel generátor kapcsolása látható.

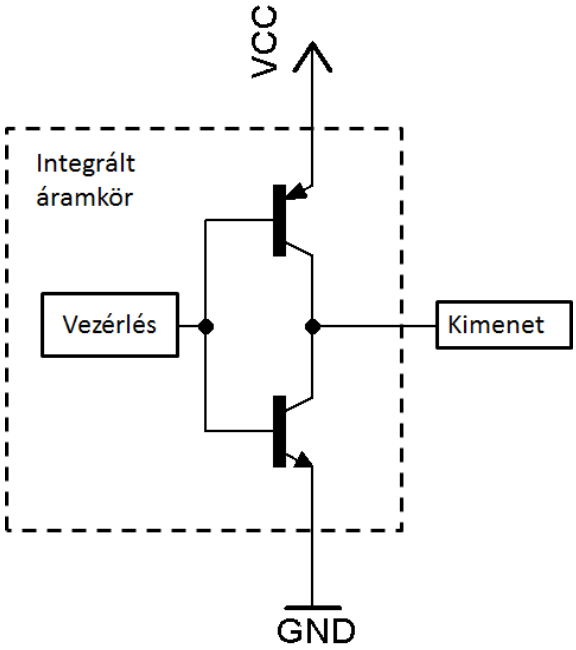

IC1
R1
R2
6,8M
6,8M
11
P1
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q12
Q13
Q14
7
5
4
6
14
13
15
1
2
3
9
10
12
RES
P0
P0
4060N
R3
330K
Q1
1
2
C1
C2
32k768
33pF
33pF
Pin2: 2Hz, 4Hz, 8Hz, 16Hz, 32Hz
SV2
5
4
3
2
1
IC2A
1
A
QA
QB
QC
QD
CLR
2
3
4
5
6
74AC393N
1
2
3
4
SV1
Pin1: 1 Hz, 1/2Hz, 1/4Hz, 1/8Hz
VCC
8
VSS
IC1P
16
VDD
7
GND
IC2P
14
VCC
GND

Áramkörök kimeneti típusai

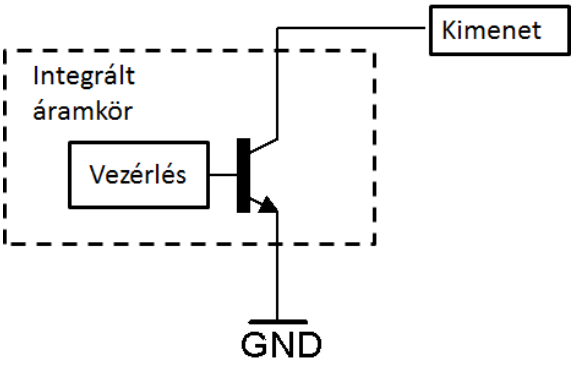
Logikai integrált áramkörök esetén a kimenetek megvalósításánál tipikusan három megoldást szoktak alkalmazni. A totem pole kimenetet, a nyitott kollektoros kimenetet, illetve a három állapotú kimenetet.

A leggyakoribb megoldás a totem pole. Ezen kimenet értéke vagy magas, vagy alacsony lehet. Belső felépítéséből adódóan ezen kimeneti típus nem viseli el azt, ha több kimenetet közvetlenül összekötünk, mivel ebben az esetben, ha az egyik kimenet alacsony szintű, míg a másik magas, akkor az alacsony szinten lévő kimenet felé folyik az áram, ami könnyen károsíthatja az áramköri elemet.

Amennyiben több kimenet összekötéséhez nem szeretnénk aktív áramköri kapukat felhasználni, akkor a kimeneteket mindenképpen védeni kell diódákkal. Lehetőség szerint Schottky típusúakkal az alacsony nyitófeszültségük és nagy sebességük miatt.



A nyitott kollektoros megoldásban az áramkör végén egy tranzisztor helyezkedik el, amelynek a bázisát az áramkör kimenete hajtja meg, az emittere pedig földre van kötve. A külvilág felé pedig a kollektora van kivezetve a tranzisztornak. Ha az áramkör kimenete alacsony szintű, akkor a tranzisztor nem nyit ki, így mondhatni a kimenet lebeg, mivel akkora ellenállást képvisel, hogy nem tud rajta áram folyni. Amennyiben pedig az áramkör kimenete magas szintű, akkor a kollektor felől a föld felé tud áram folyni, vagyis a kimenet alacsony szintű lesz.

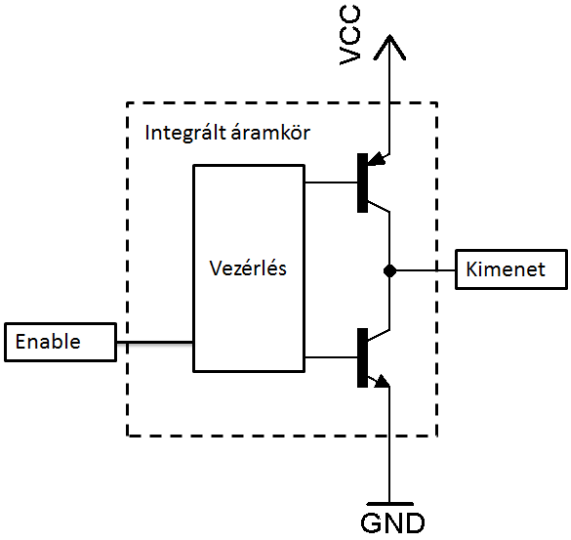


A háromállapotú kimenetek a fenti két megoldást kombinálják. A kimenet lehet alacsony szintű, lehet magas szintű, illetve lehet lebegő állapotban. Ezen megvalósításhoz plusz egy vezérlőjelet szoktak beiktatni, amely egy engedélyező jel. Ha az engedélyező jel aktív, akkor az áramkör úgy működik, mint egy totem pole kimenettel ellátott logikai IC esetén, azonban ha az engedélyező jel nem aktív, akkor az áramkör kimenete nagy impedanciás módba vált, vagyis a kimenet lebegni kezd.

Ilyen kimenetű áramköröket tipikusan buszrendszerek esetén alkalmazzák, ahol adott számú vezetéken több eszköznek kell kommunikálnia eltérő adat irányban.

Amennyiben az áramkör kimenete nem három állapotú, akkor is könnyen képezhető belőle három állapotú kimenet. Erre a célra találták ki a kimeneti buffereket és buszrendszer meghajtókat. A buffer egy engedélyező jel segítségével lebegő állapotba tudja kapcsolni a kimenetet, míg a buszrendszer meghajtó rendelkezik irányváltó vezérlőjellel is.

Az engedélyező jel lehet alacsony szinten és magas szinten aktív is. Integrált áramkör függő a megvalósítás, de a legtöbb esetben az alacsony szinten aktív megvalósítást preferálják.



Schmitt trigger

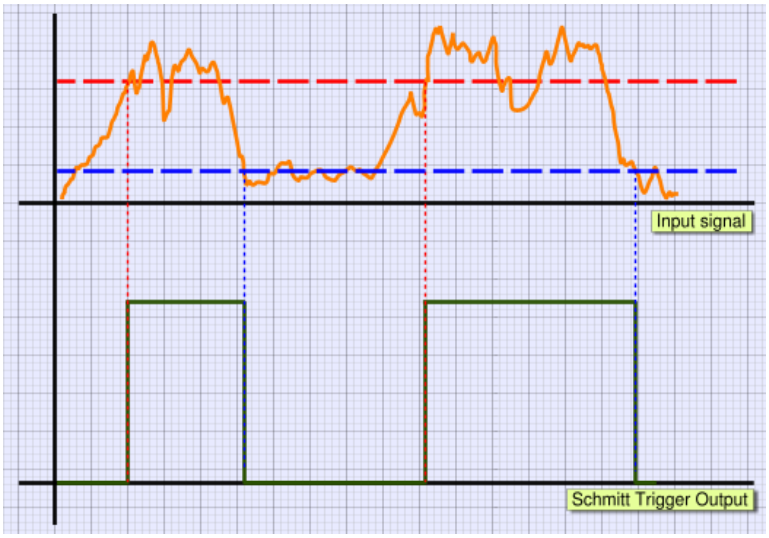
A Schmitt trigger lényegében egy hiszterézissel rendelkező komparátor áramkör. A komparátor tulajdonsága, hogy a kimenete azonnal átbillen, ha a bemeneti feszültség meghalad egy bizonyos feszültségszintet. Schmitt trigger esetén ez a változás késleltetett.

Ebből adódóan kiválóan alkalmas digitális elektronikai áramkörökben bemeneti zajszűrésre. Erre azért van leginkább szükség, mert a TTL és CMOS áramkörök esetén is a bemeneten a 0 és 1 állapothoz egy feszültségtartomány van rendelve. Ezen két feszültségtartomány között meg van egy instabil tartomány, amiben az áramkör viselkedése bizonytalan.

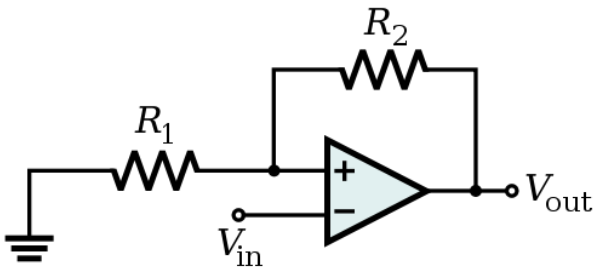
Ilyen bizonytalan működés bekövetkezhet akkor, ha az átvitel zajos, de akár egy egyszerű gombnyomás is kiválthatja, mivel a kapcsolók/nyomógombok kontaktusai csúsznak.

A Schmitt trigger a problémát úgy oldja meg, hogy ha a a bemeneti feszültségszint eléri a magas feszültségszinthez rendelt hiszterézis pontot, akkor a trigger kimenete magas szintre vált. Magas szintről alacsony szintre a kimenet csak akkor fog átváltani, ha a bemeneti feszültségszint az alacsony hiszterézis pont alá esik.

72. Ábra:



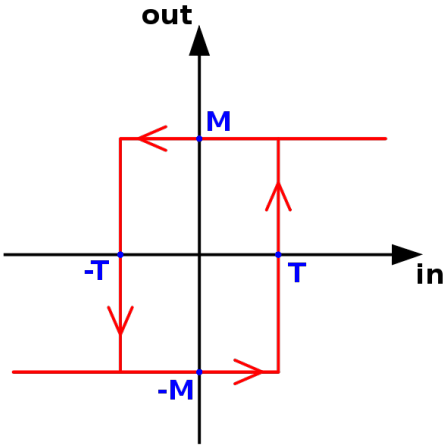
Számos integrált áramkör (NAND kapuk és bufferek) rendelkezik Scmitt triggeres bemenettel. CPLD/FPGA áramkörök esetén bevett gyakorlat, hogy az összes bemenet Scmitt triggeres. Az áramkör könnyen megvalósítható műveleti erősítők segítségével.



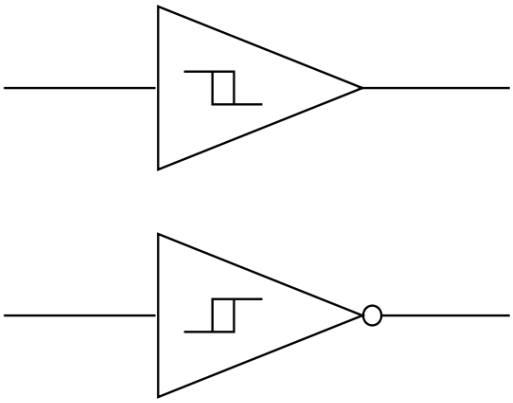
A kapcsolás érdekessége, hogy pozitív visszacsatolást alkalmaz. Az átbillenési feszültségeket R1 és R2 ellenállások, valamint a műveleti erősítő tápfeszültsége határozzák meg:

$$U_{Bill} = \frac{R_1}{R_1 + R_2} \cdot U_T$$

A pozitív billenő feszültség meghatározásához a képletbe U_T helyére a pozitív tápfeszültséget kell behelyettesíteni, a negatív billenési feszültség meghatározásához pedig a negatív tápfeszültséget.



Az olyan digitális integrált áramkörök rajzjeleit, amelyek rendelkeznek Schmitt triggeres bemenetűek ki szokták egészíteni a Schmitt trigger hiszterézis görbéjét ábrázoló szimbólummal, ezzel utalva a hiszterézises viselkedésre.



8. Verilog alapismeretek

A Verilog hardver leíró nyelvet eredetileg áramköri elemek szimulálására alkalmas nyelvnek tervezték 1984-ben. Azóta a nyelv rengeteget fejlődött. Eredeti célja az alkotóknak az volt, hogy egy könnyen megtanulható hardver leíró nyelvet alkossanak meg, amivel könnyen lehet szimulálni hardver elemeket. Ezért a Verilog nagymértékben merít elemeket a Pascal és C programozási nyelvekből.

Áramköri elemek tényleges tervezésére 1987 óta alkalmazzák, de csak 1995-ben szabványosította az IEEE. A Verilog IEEE szabvány száma: 1364

A Verilog és egyéb hardverleíró nyelvek megjelenésének az oka az volt, hogy a digitális rendszerek annyira bonyolulttá váltak, hogy kapuk szintjén kapcsolási rajzok segítségével átláthatatlanná váltak, ezért szükség volt egy magasabb szintű leírási módra. Így születtek meg a HDL nyelvek (Hardware Descriptor Language). A Verilog a hagyományos HDL nyelvektől eltér, mivel magasabb szintű szolgáltatásokat is tartalmaz, így könnyen leírhatóak vele bonyolult vezérlési szerkezetek is.

Nyelvi elemek

Megjegyzések

A megjegyzések szintaxisa megegyezik a C++ említett szintaxissal.

```
/*
Többsoros
megjegyzés
*/

//egysoros megjegyzés
```

Szövegek

Szövegek esetén is a C++ szintaxis alkalmazandó, azaz a szövegeket idézőjelek között kell elhelyezni. A szövegek több sorban nem adhatóak meg.

Számok

Egész számok megadhatóak bináris (b vagy B), decimális (d vagy D), oktális(o vagy O) és hexadecimális(h vagy H) formátumban. Az általános szám megadási formátumok a következőek:

```
[bit hossz]'[számrendszer betűjele][érték]
[számrendszer betűjele][érték]
[érték]
```

Amennyiben a bit hossz nincs megadva, akkor a szám bithossza gép függő, de legalább 32 bit hosszú lesz. Amennyiben a számrendszer betűjelét nem adjuk meg, akkor a szám decimális számrendszerbeli számot reprezentál.

Negatív számok mind a három formátumban megadhatóak, de minden esetben meg kell előznie az előjelnek a bit hosszat és a számrendszer betűjelet.

A számokban megadásánál lehetőségünk van közömbös számjegyek és magas impedanciájú számjegyek megadására, amelyek segítségével közömbös sorokat és három állapotú kimeneteket tudunk leírni. A közömbös számjegy jelölésére az x karakter használható, de csak bináris, oktális, vagy hexadecimális számrendszerben. Magas impedanciájú számjegy jelölésére a z betű vagy a ? jel alkalmazható. A magas impedanciájú számjegy jelölésénél nincs számrendszer kikötés.

A közömbös számjegy által jelölt bit hossz számrendszer függően változik. Az x karakter hexadecimális számrendszer esetén négy bitet reprezentál, oktális számrendszer esetén három bitet, bináris számrendszer esetén pedig egyet.

Lebegőpontos számok megadhatóak decimális alakban és tudományos formátumban is. Ha decimális

formátumban adunk meg számot, akkor a decimális pont mindkét oldalán szerepelnie kell számjegynek, tehát egész lebegőpontos számok esetén is ki kell tenni a 0 végződést. Lebegőpontos számok esetén nem kell megadni a bit hosszt.

A számok jobb olvashatósága érdekében a szám értékének megadásban bárhol elhelyezhető a _ karakter, amellyel tetszőlegesen csoportosíthatóak a számjegyek.

Példák számok megadására

```
8'd132          //decimális számrendszerű szám 8 biten
8'b1000_0100    //bináris szám 8 biten, 4 bites csoportosítással
8'h3F      //hexadecimális szám 8 biten
-8'd32          //-32 8 biten, kettes komplemens kóddal ábrázolva
4'b100x         //4 bites szám, amelyben a legkisebb helyiértékű bit
ismeretlen
12'bz      //12 bites szám, amelyből mind a 12 bit magas impedanciájú
3.14            //pi
1.2e12          //1,2 x 10^12
```

Azonosítók neveire vonatkozó megkötések

Az azonosítók neveinek definiálásakor a kis és nagybetűk különbözőnek számítanak. Egy azonosító betűkből (angol ABC betűi) és számokból állhat, valamint szerepelhet benne _ jel és $ jel. Azonosító nem kezdődhet számmal, csak betűvel vagy _ jellel.

Blokkok

A blokkok bontása pascal stílusú, ami azt jelenti, hogy kapcsos zárójelek helyett a ***begin*** *és* ***end*** *kulcsszavak alkalmazhatóak blokkok jelölésére.*

Adat típusok

Vezetékek

*A vezetékeket a **wire** kulcsszó és típus írja le. Segítségével az egyes egységek között kapcsolat definiálható. A vezeték adatot nem tárol, ezért folyamatosan értékkel kell ellátni. Ez úgy érhető el, hogy egy modul vagy kapu kimenetéhez csatlakoztatjuk őket, vagy egy folyamatos értékadásban szerepeltetjük őket.*

*A **suply1** és **suply0** speciális vezeték típusok. A tápfeszültségforrás modellezésére szolgálnak. A suply1 mindig igaz értéket fog szolgáltatni, a suply0 pedig mindig hamis értéket.*

Regiszterek

*A regiszterek programozási nyelvi szemléletmódot követve változóknak feleltethetőek meg. Hardver szemszögből nézve pedig elemi tárolóegységeknek, amelyek az értéküket addig őrzik, amíg felül nem írjuk őket egy másik értékkel. Általában procedurális értékadás közben szoktak értéket kapni. A regiszterek definiálására a **regi**kulcsszó használható.*

Paraméterek

*A paraméterek a konstansok megfelelői a Verilog nyelvben. Segítségükkel átláthatóbbá, könnyebben módosíthatóvá válnak a programok. Nem meglepő módon paraméter definiálásra a **parameter** kulcsszó használható. Paraméter csak modulon belül definiálható, de a modul használatbavételekor módosítható az értéke.*

Vektorok

A vektorok a Verilog nyelv tömbjei. Segítségével a vezetékekből és regiszterekből több bites típusokat, buszrendszereket definiálhatunk. Az alábbi példa egy 4 bites regiszter definíciót mutat be:

```
reg [3:0] vektor;
```

A vektorok definiálásakor nem elég megadni a vektor elemeinek számát. Az szögletes zárójelen belül az első szám a legnagyobb helyiértékű bit azonosítóját definiálja, a kettőspont utáni rész pedig a legkissebb helyiértékű bit azonosítója. Az egyes bitekre később ugyan úgy hivatkozhatunk, mint C esetén a tömb elemeire: A vektor azonosítójának megadása után kapcsos zárójelben az azonosító számmal.

```
vektor[3] = 1;
```

Operátorok

A nyelv operátorainak viselkedése majdnem megegyezik a C nyelv operátoraival. Lényeges eltérés az adattípusok kezelésében és a a bit szintű operátorokban, valamint a redukciós és bit összefűző operátorokban van.

A nagy különbség az adattípusok kezelése esetén az, ha aritmetikai operátorokat alkalmazunk regisztereken, akkor a regiszterek értékét a nyelv előjel nélküli egészként kezeli, míg C esetén egész számok esetén az alap feltételezés az, hogy előjeles a szám.

A bit szintű operátorok esetén újdonság az összetett operátorok megjelenése. C esetén csak a három alap logikai művelet (NOT, AND, OR) és a kizáró vagy (XOR) lett implementálva. Verilog esetén bekerült a NAND, NOR és XNOR (más néven ekvivalencia) művelet is. Ezek összetett operátorok, amelyek két karakterből állnak:

- NAND: ~&
- NOR: ~|
- XNOR: ~^

A redukciós és bit összefűző operátorok a Verilog sajátosságai. A Redukciós operátorok segítségével egy vektort egy bitté redukálhatunk úgy, hogy a bitjeit operandusként szerepeltetjük az operátor alkalmazása során. A redukció során használható operátorok megegyeznek a bit szintű operátorokkal. Az alábbi rövid példa azt mutatja be, hogy egy buszrendszer vezetékei között hogyan létesíthető vagy kapcsolat:

```
wire [4:0] adat;
wire kimenet;

assign kimenet = |adat;
```

A bit összefűző operátor segítségével egyedi vezetékekből képezhető egy vektor. Az alábbi példa ennek a használatát mutatja be:

```
wire a, b, c, d;
wire [4, 0] busz;

assign busz = { a, b, c, d };
```

Ha ugyan azt a vezetéket szeretnénk n alkalommal felhasználni az értékadás során, akkor két szintaxis is használható:

```
assign busz = { a, a, a, a }
assign busz = { a{4} }
```

A bit összetűzést vektor típusok részeire is használhatjuk:

```
wire busz_be [3:0];
wire busz_ki [3:0];

assign busz_ki = { busz_be[1:0], busz_be[3,2] };
```

Értékadások

Verilolog nyelv esetén két fő értékadási típus van. A folyamatos és a procedurális.

Folyamatos értékadás

Folyamatos értékadás segítségével vezetékekre kényszeríthetünk egy értéket. Az értékadás attól lesz folyamatos, mert mindig aktív. Tehát ha az értékadás jobb oldalán álló kifejezés értékében bármi változás történik, az rögtön frissíteni fogja a kifejezés bal oldalát. Ez a fajta értékadás kombinációs hálózatok szimulálására, leírására alkalmas. Az értékadás szintaxisa:

```
assign [vezeték típusú változó] = [kifejezés]
```

Procedurális értékadás

*Procedurális érzékcsalódással csak regiszter típusú változóknak adhatunk értéket. Az értékadás csak olyan utasítás szerkezetben lehetséges, amely esemény vezérelt, vagyis az értékadás nem folyamatos. Ilyen vezérlési szerkezetek az i**nitial** és az **always**. Ezen blokkokon belül további feltételes utasításokkal befolyásolhatjuk az értékadást.*

Az értékadás bal oldalán szerepelhet egy egy bites regiszter vagy több bites vektor regiszter, vagy egy több bites vektor regiszter része. Vektor regiszterek esetén az értékadás csak konstans számokkal lehetséges, változóval nem indexelhetünk. Ennek oka az, hogy ha ez lehetséges lenne, akkor azt jelentené, hogy a hardver menet közben változásokra képes. Ha ilyen jellegű funkciót szeretnénk megvalósítani a programban, akkor multiplexereket kell alkalmaznunk.

Az értékadás történhet blokkoló és nem blokkoló módon. Blokkoló értékadás során a programban felsorolt értékadások sorrendben fognak végrehajtódni. A következő értékadás csak azután fog végbemenni, ha az előtte lévő értékadás befejeződött. Ez egy kis késleltetést jelent, vagyis a hálózat működése aszinkron lesz. A blokkoló értékadás szintaxisa:

```
[regiszter változó] = [kifejezés]
```

Nem blokkoló értékadás során az összes nem blokkoló értékadás egy időben fog végrehajtódni, vagyis a hálózat működése szinkron. A nem blokkoló értékadás szintaxisa:

```
[regiszter változó] <= [kifejezés]
```

Vezérlési szerkezetek

Always és Initial blokkok

A sorrendi hálózatok viselkedéseinek modellezésére szolgálnak. A program kezdetekor aktivizálódik mindkét blokk. Az inital blokk csak egyszer fut le, a program indításakor, de csak szimulációs környezetben használható. Az always blokk minden olyan esetben lefut, ha a vezérlési feltételként megadott esemény bekövetkezik. Példa a két szerkezet megadására:

```
initial
      begin
            //utasítások
      end

always @(negedge x)
      begin
            //utasítások
      end
```

Az always blokk esetén a zárójelben szereplő kifejezés az esemény, ami alapján vezérelni szeretnénk. A negedge kulcszó azt teszi lehetővé, hogy x minden lefutó éle esetén történjen valami esemény. Az események leírása esetén az alábbi kulcsszavak használhatóak:

Kulcsszó	***Példa***	***Leírás***
posedge	`posedge x`	*Reagálás x felfutó élére*
negedge	`negedge x`	*Reagálás x lefutó élére*
and	`a and b`	*Reagálás, ha az ÉS kapcsolat a és b között igaz eredményt ad.*
or	`a or b`	*Reagálás, ha a VAGY kapcsolat a és b között igaz eredményt ad.*
not	`not b`	*Reagálás, ha b nem igaz*
nand	`a nand b`	*Reagálás, ha a NAND kapcsolat a és b között igaz eredményt ad.*
nor	`a nor b`	*Reagálás, ha az NOR kapcsolat a és b között igaz eredményt ad.*
xor	`a xor b`	*Reagálás, ha az XOR kapcsolat a és b között igaz eredményt ad.*
xnor	`a xnor b`	*Reagálás, ha az XNOR kapcsolat a és b között igaz eredményt ad.*

43. Táblázat: always blokk lehetséges feltételei

A blokk esetén nem csak egy feltétel adható meg, hanem tetszőlegesen kombinálhatóak:

```
always @(posedge clk or (a and b))
      begin
            //utasítások
      end
```

A Példában szereplő always blokk akkor fog végrehajtódni, ha clk jelen felfutó él van, vagy ha a és b igaz állapotú.

Ha az always blokkhoz nem adunk meg eseményt, akkor végtelen ciklusként viselkedik, vagyis a benne szereplő utolsó utasítás végrehajtása után következik a benne elhelyezett első utasítás. A feltétel nélküli always blokkok csak szimulációk során használhatóak, tényleges hardver tervezése esetén nem.

Feltételes utasítások

A feltételes utasítások if – else szerkezettel és case szerkezettel írhatóak le, mint C nyelv esetén. Az if – else szerkezet a C-ben ismertetett módon működik. Feltéles utasításokat csak szekvenciális utasításblokkokban adhatunk meg. (initial, always)

A Case szerkezet működése azonos, azonban a szintaxis eltérő. A használanadó szintaxis:

```
case ([kifejezés]):
      [érték 1]: [utasítások];
      [érték 2]: [utasítások];
      default: [default ág, mint c esetén]
endcase
```

Ha az értékekben szeretnénk don't care biteket és magas impedanciát is kezelni, akkor vagy casez vagy casex utasítást kell használnunk. A casez utasítás a z értékeket képes feldogozni, a casex utasítás pedig a z és x értékeket is.

Késleltetett utasítás végrehajtás

Késleltetett utasítás végrehajtásra csak szimuláció során van lehetőségünk, a szerkezet segítségével megadható, hogy mennyi órajel ciklussal akarjuk késleltetni egy adott kifejezés végrehajtását. A szerkezet megadási formája:

```
# [konstans] [utasítás]
```

Ezen szerkezet segítségével szimulálható áramkörök esetén a minden esetben jelentkező késleltetés.

Modulok

A modulok segítségével egy nagyobb, bonyolult rendszer több kisebb részegységre bontható. C++ szemléletet követve a modul felfogható egy osztálynak. Digitális technikai szemlélet módot követve pedig felfogható egy integrált áramkörként.

Modulok összeépítésével hozhatunk létre összetettebb rendszereket verilog-ban. A modulok definiálásakor meg kell adni a ki és bemeneteiket és a működését a korábban ismertetett szerkezetek egyikével. A modul definíció szintaxisa:

```
module [modul neve]([külvilág felé látszódó kapcsolati pontok]);
      input [bemenet neve];
      input [másik bemenet neve];
      output [kimenet neve];
      output [másik kimenet neve];

      //változók

      //modul működése
endmodule
```

A modulban a bemeneteket külön vezetékként nem kell definiálni, mivel az input kulcsszó ezt magával vonza. Az kimenetek és bemenetek deklarálásakor csak olyan nevet használhatunk, amely a külvilág felé is látszik.

A kész modulunkat más modulokban is használatba vehetjük. Ehhez példányosítani kell a modult, amelynek a szintaxisa:

```
[modul neve] [változónév]([kapcsolódási pontok]);
```

A modul példányosításakor megváltoztatható a benne használt paraméterek értéke. Az ilyen példányosítás szintaxisa:

```
[modul neve] [paraméter specifikáció] [változónév]([kapcsolódási pontok]);
```

Hogyan tovább

A modulokkal való megismerkedés után tisztában vagyunk a Verilog nyelv alapjaival. Az ismertető alapján nekiállhatunk logikai hálózatok megvalósításának. A Verilog projektek fejezetben számos alkalmazási példa található, amelyekből saját hardverek tervezéséhez ötlet meríthető.

9. Mikrovezérlő alapismeretek

A mikrovezérlő egy olyan speciális mikroszámítógép, amely egyetlenegy chipből áll. Egy chip tartalmazza a processzort, a memóriát (RAM és ROM), valamint az I/O vezérlőket is, amelyek kellenek ahhoz, hogy a feladatukat teljesítsék ezen eszközök Szokták ezen vezérlőket beágyazott vezérlőknek is nevezni, mivel gyakran beágyazottak abba az eszközbe, amit vezérelnek.

A *belső*

76. ábra. Egy mikrovezérlő tipikus belső felépítése

processzor két feladatot lát el és két részből is áll. Az aritmetikai és logikai egység (ALU) a matematikai számításokat végzi el, míg a vezérlő egység (CU) vezérlő jelekre reagál és azokat ad ki. Vezérli a csatlakoztatott perifériákat.

Tradicionálisan ezek az eszközök Assembler nyelven programozottak, azonban az utóbbi időben ezen a fronton is elterjedt a C, C++, BASIC és PASCAL programozási nyelv, amely leegyszerűsítette a programok készítését, ezáltal olcsóbbá is vált ezen eszközök használata, mivel a fejlesztőnek kevesebb időt kell ráfordítania a programra, ami kevesebb munkaköltséget jelent.

Az különböző gyártók eltérő vezérlői másféle képességekkel rendelkeznek értelemszerűen. Az eszközök processzora 8, 16 és 32 bites szokott lenni.. A 8 bites vezérlők széles körben elterjedtek. Ezek általában DIP tokozásban és SMD tokozásban is kaphatóak. 16 bitesekből is léteznek DIP változatok. 32 bites vezérlőkből pedig inkább az SMD kivitel a jellemző a sok láb és sok szolgáltatás miatt.

RISC és CISC CPU típusok

A CISC (Complex Instruction Set CPU – Komplex Utasításkészletű CPU) és RISC (Reduced Instruction Set CPU – Csökkentett Utasításkészletű CPU) kifejezések a mikrovezérlő által értelmezhető utasítások számát jelölik.

Általában a RISC CPU-k esetén az utasítások hossza jóval hosszabb, mint a CPU regisztereinek hossza. (PIC16 esetén a regiszterek 8 bitesek, de az utasítások 14 bitesek) Ezáltal feldolgozás közben az utasítások és az azokhoz szükséges adatok egy ciklusban töltődnek be a processzorba, így eredményezve egy gyors működést. Ez a gyorsaság csak kevés utasítás esetén tartható fent, így általában a RISC CPU-k kevesebb, mint 150 utasítással rendelkeznek. PIC16 esetén ez az utasításszám csupán 35. További jellemzője a RISC processzoroknak, hogy viszonylag sok regiszterrel rendelkeznek (jóval többel, mint a CISC-es társaik).

A CISC CPU-k általában több, mint 200 különböző utasítással rendelkeznek. Itt az adatok és az utasítások betöltése különböző ciklusokban történik. Az utasítások jellemzője a változó paraméterszám. Előfordulhat, hogy egy utasítás a hozzá tartozó adatokkal egy 8 bites CPU esetén 8, 16, 24 vagy 32 bit hosszú legyen. Ezen CPU típus főként asztali számítógépekben és szerverekben használatos.

Harvard és Neumann architektúrák

A mikrovezérlők, eltérően a számítógépektől nem Neumann architektúrára épülnek, ami a memóriakezelésüket illeti. Ehelyett a Harvard architektúrát használják.

A Neumann architektúrában egy közös memóriában (még ha a memória több chipből is áll, akkor is azonos címtérben dolgoznak) tárolódnak a programok és az adatok. Ezzel a probléma az, hogy előfordulhat, hogy a végrehajtás adatterületre keveredik. Ezt szokták aktívan kihasználni kártékony kódok futtatására. Ugyan az elmúlt években már szoftveresen és hardveresen is védekeznek ez ellen, mégis megtörténhet.

77. ábra. A Neumann memória architektúra

Harvard memória architektúra esetén két memória létezik, amelyek fizikailag is el vannak különítve egymásól. Van egy memória külön a programkód számára, és egy külön memória az adatoknak, amivel az eszköz dolgozik. Tipikusan a programmemória mérete 32Kb szokott lenni, míg az adatmemória mérete 256 byte. Természetesen a mikrovezérlő típusától függ, hogy mennyi memória is található az eszközben. Programkód alatt a tényleges program utasításai értendőek, míg adat alatt jelen esetben azon változók és adatok, amelyekkel a program futása közben dolgozik.

A

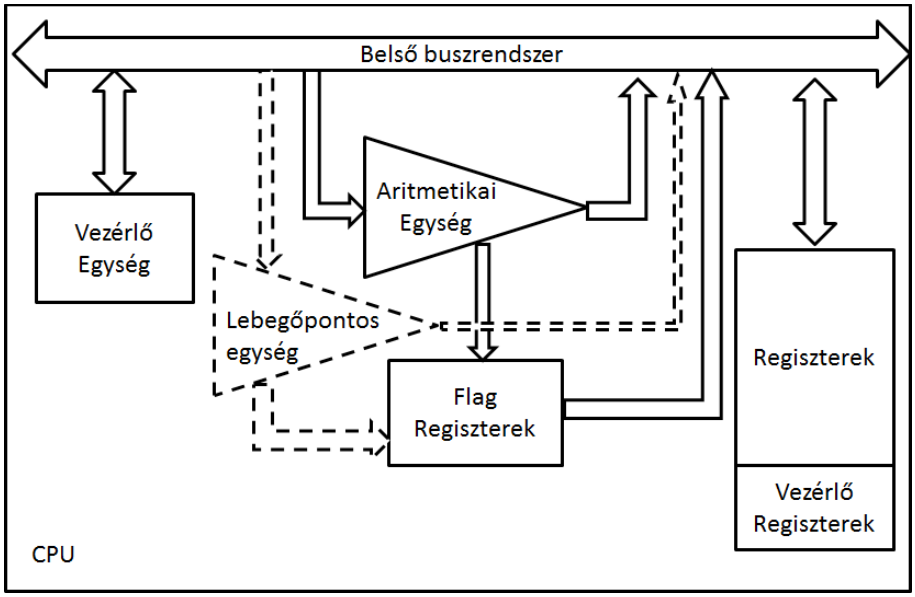
78. ábra. A Harvard memória architektúra

programmemória tipikusan egy elektronikusan programozható ROM memória (EEPROM) vagy FLASH memória, míg az adat memória egy RAM memória.

A Processzor belső felépítése

A mikrovezérlő lelke a processzor. A processzorok belső felépítésükben igen változatosak tudnak lenni, azonban konkrét kialakítástól függetlenül minden processzor rendelkezik az alábbi részegységek közül legalább egyel:

- *Vezérlőegység, CU*
- *Aritmetikai és logikai egység, ALU*
- *Lebegőpontos egység, FPU (csak bizonyos típusokban van benne)*
- *Általános regiszterek*
- *Flag regiszterek*
- *Vezérlő regiszterek*
- *Belső buszrendszer*



CU, ALU

A vezérlőegység feladata a CPU-n kívül eső egységek vezérlése, az utasítások beolvasása és dekódolása. A dekódolt utasításokat az Aritmeitkai és logikai egység dolgozza fel. Az utasítások végrehajtása CPU kialakítástól függően lehet fix időt igénybe vevő (tipikus RISC CPU megoldás) és változó időt igénybe vevő (tipikusan CISC processzorokra jellemző). Az aritmetikai egység által elvégezhető műveleteket az utasításkészlet határozza meg. Minél bonyolultabb az utasításkészlet, annál komplexebb az aritmetikai egység felépítése. Az Aritmetikai egység bitszáma meghatározza az egy utasítás által feldolgozható adatmennyiséget. Ez az adat csak egész szám típus lehet.

FPU

Lebegőpontos műveletek elvégzésére egy processzor aritmetikai egysége sem képes hardveresen. Így az ilyen számokkal való műveleteket szoftveresen kell megoldani. Ez főként mikrovezérlőkre igaz.

A nagy teljesítményű processzorok a lebegőpontos számítások elvégzésére tartalmaznak egy külön lebegőpontos egységet, amit angolul floating point unit-nak (FPU) neveznek. Ez az egység nagyságrendekkel gyorsabb műveletvégzésre képes. Mikrovezérlők esetén nem igen fogjuk hiányát érezni, de egy asztali gép esetén már jelentős lassulás lenne tapasztalható, ha nem lenne benne a processzorban. Ennek az az oka, hogy az olyan multimédiás alkalmazások, mint a filmek, zene, 3D grafika erősen épít a lebegőpontos egységre.

Érdekesség, hogy a manapság oly népszerű ARM architektúra esetén a lebegőpontos egység megléte opcionális, nem kötelező. Ebből adódóan igen sokféle ARM eszközzel találkozhatunk.

Általános regiszterek

Az Általános regiszterek feladata az adatok átmeneti tárolása. Minél több van a processzorban ebből, annál gyorsabb tud lenni a működése, mivel ebben az esetben nem kell külső memóriába írni az átmeneti eredményeket. A külső RAM memóriák nagyságrendekkel lassabbak, mint a processzorba integrált regiszterek.

Programozás esetén hátrányt tud jelenteni, az is ha nagyon kevés a regiszter és az is, ha nagyon sok van, bár ha sok van az a kisebbik baj, mivel ebben esetben könnyebb olyan fordítót írni, ami hatékonyan kezeli ezt a helyzetet. Kevés regiszter esetén a fordító készítése igen komplikált tud lenni.

Regiszter felépítésénél két megoldás jellemző. A fájl felépítés és a blokkokból álló regiszterek. A fájl felépítés onnan kapta a nevét, hogy a processzor regiszterei kezelhetőek egyben, egy nagy memóriatömbként. A memóriatömbök kezelése pedig nagymértékben hasonlít a fájlkezeléshez. Van egy regiszter mutató regiszter, amely egy egyszerű számláló. Ennek a léptetésével folyamatosan változik a kijelölt regiszter, amit írni és olvasni tudunk. A regisztere közvetlenül hivatkozhatunk annak címével is.

A blokkos felépítésnél nincs regiszter mutató regiszter, tehát nem kezelhetőek a regiszterek tömbként. Az egyes belső tárolókra csak azok nevével hivatkozhatunk. Ezt a megoldást akkor szokták alkalmazni, ha csak pár darab regiszter van a processzorban.

A fájl felépítést tipikusan RISC processzorok esetén alkalmazzák, míg a blokkos felépítés inkább a CISC utasításkészlet jellemzője.

Flag regiszterek

A Flag regiszterek az aritmetikai egység műveleteinek visszajelzésére szolgáló memória cellák. Ezek a processzor többi része számára csak olvashatóak, állapotukat csak az ALU képes megváltoztatni. Itt olyan jelzések állítódnak be műveletvégzések során, hogy történt-e túlcsordulás, vagy hogy az eredmény páros-e, esetleg az eredmény nulla-e.

Processzoronként és utasításkészletként eltérő, hogy milyen flag regiszterek vannak és hogy melyik utasítás melyikre van hatással.

A Flag regiszterek értékei használhatóak fel feltételes utasítások és ciklusok létrehozására speciális utasítások segítségével.

Vezérlő regiszterek

A vezérlő regiszterek száma és feladata is processzorfüggő, azonban minden processzor rendelkezik egy utasítás mutatóval (IP) és verem mutató regiszterrel (SP)

Az utasítás mutató egy memória címre mutat. A processzor az ezen mutató által mutatott címről olvassa be az utasítást. Minden egyes utasítás beolvasás megváltoztatja ezen mutató értékét. RISC processzorok esetén ez a változás mindig fix értékű, míg CISC processzorok esetén változó. A CPU újraindítása esetén az IP regiszter értéke 0-ról vagy egy meghatározott számról indul. Ezt szokták reset vektornak nevezni.

A verem mutató egy hardveresen, a RAM-ban megvalósított verem szerkezetben az aktuális elemet mutatja. A verem olyan esetekben használt, mint a függvény hívás vagy megszakítás hívás.

Függvény és megszakítás hívás esetén a processzor éppen amit csinál felfüggeszti, majd egy másik memóriaterületen folytatja a működését. Ez az IP regiszter átírását jelenti. Azért, hogy a függvény vagy megszakítás végrehajtása után folytatni tudja a működést a processzor az eredeti helyről, a hívás előtt az IP regiszter aktuális tartalmát átmenti a verembe. Ezután változik csak meg az IP regiszter értéke. A megszakítás/függvény végén a veremből visszaíródik az IP értéke és a programfuttatás folytatódik az eredeti helyről.

A hardveres verembe a felhasználó is helyezhet a programjában értékeket. Természetesen ezek is módosítják a veremmutatót. A veremmutató értéke közvetlenül a legtöbb processzor esetén nem írható, csak közvetett módon lehet módosítani az értékét.

Belső buszrendszer

Feladata a processzor egyes részegységeinek összekötése. A processzor felépítésétől függ a szerkezete. Elképzelhető, hogy a processzor 8 bites külső buszrendszerrel rendelkezik, de könnyen adódhat, hogy a belső összekötő rendszer 16 vagy akár 32 bites.

Memória típusok

- **ROM**

  Read Only Memory, azaz csak olvasható memória. Tartalmát tápfeszültség nélkül is megőrzi.

- ***EPROM***

  Elektronikusan programozható ROM. Jellemzője, hogy nagyobb feszültséggel van írva, mint olvasva. Többször is újraprogramozható. Ezen tulajdonsága úgy van kivitelezve, hogy UV fény segítségével lehet teljesen törölni a tartalmát. Ez gyakorlatban úgy van megoldva, hogy a chip tetején van egy kis ablak rész, amely alapesetben le van takarva. A védő eltávolításával a fény képes bejutni a chipbe, ezzel törölve a tartalmát. Napfénytől is óvni kell annak sokszor magas UV tartalma miatt, ugyanis, ha nem is törli teljes mértékben a tartalmat, károsítja azt.

- ***EEPROM***

  Elektronikusan törölhető és programozható ROM. A törléshez és a programozáshoz szintén jóval nagyobb feszültség szükséges, mint a kiolvasáshoz. Manapság már csak ez a fajta ROM van elterjedve. Felépítése szerint lehet soros és párhuzamos. A párhuzamos felépítésűek egyre ritkábbak, mivel a soros kezeléshez elég 2 szál vezeték memóriaszervezéstől függetlenül, így több láb marad szabadon a mikrovezérlőn. Ezzel szemben egy 8 bites párhuzamos EEPROM kezeléséhez 8 szál vezetéket kell bekötni a mikrovezérlőbe.

- ***FLASH***

  Az EEPROM egyik változata. Jellemzője az, hogy igencsak gyorsan írható és olvasható. Két fő típusa létezik: NAND Flash és NOR Flash. A két változat között az a fő eltérés, hogy a NOR típus párhuzamos adatelérést tesz lehetővé, míg a NAND típus soros adathozzáférésű. Mikrovezérlőkbe NOR típusú memóriát szoktak integrálni, mert a párhuzamos adatelérés lehetővé teszi a közvetlen programfuttatást. Pendrive-ok és egyéb FLASH memóriát tartalmazó tárolóeszközök esetén a NAND típust részesítik előnyben, mivel a soros elérés végett kevesebb vezetékre van hozzá szükség. Összehasonlításképpen egy 8GB kapacitású memóriához párhuzamos csatolófelülettel 33db címvezetékre lenne szükség és 8 adatvezetékre, ami összesen 41 csatlakozó pontot jelentene. Ezzel szemben a soros típusokhoz kell 2 vezeték.

- ***RAM***

  Random Access Memory, vagyis véletlen elérésű memória. Tartalmát csak tápfeszültség jelenlétében őrzi meg. Lehet statikus és dinamikus. A statikus RAM memóriák úgy vannak gyártva, hogy a beléjük töltött értéket kikapcsolásig megőrizzék. A dinamikus RAM-ok olyan RAM áramkörök, amelyek tartalmát időközönként frissíteni kell tápellátás mellett is. Azért népszerűek ezen áramkörök, mert olcsóbb a gyártásuk és gyorsabbak statikus társaiknál.

Mikrovezérlők szolgáltatásai és jellemzői

Tápfeszültség

Tápfeszültség tekintetében a mikrovezérlők vagy 5V feszültségről, vagy 3,3V-ról működnek tipikusan. A legtöbb vezérlő univerzális, ami azt jelenti, hogy képes működni 3,3V és 5V feszültségről is. A komplex belső felépítésű vezérlők esetén jellemző, hogy maximum 3,3V feszültségen hajlandóak működni.

Léteznek olyan vezérlők is, amelyek működtetéséhez elég egyetlen egy 1,5V-os ceruzaelem is. A tápfeszültség értékről érdemes a katalógus adatok között tájékozódni.

Azon vezérlők amelyek rendelkeznek analóg bemenettel rendelkeznek egy külön analóg referencia feszültség forrás bemenettel. Ez a feszültség forrás csak az analóg-digitális átalakító referencia feszültségét határozza meg. Ezt érdemes egy alacsony zajszinttel rendelkező referencia forrásra kötni, ha igen pontos értékeket szeretnénk mérni.

Órajel

Minden mikrovezérlőnek szüksége van órajelre a működéshez. Az órajel előállítása általában egy rezgőkvarc és kondenzátorok párosából álló áramkörrel történik meg. Egyes mikrovezérlők rendelkeznek beépített órajel generátorral is. Külső órajel forrást akkor érdemes használni, ha számítás igényes dolgokat vagy mikroszekundum pontosságú időzítéseket szeretnénk csinálni. A belső órajel források jellemzője, hogy nem annyira pontosak, mint egy külső kvarc, illetve maximum 8MHz frekvenciára képesek általában.

Minden mikrovezérlő és processzor működése során utasításokat olvas és dolgoz fel a memóriából. Ez a folyamat általában nem egy órajel ciklust igényel. Az utasítások végrehajtása processzortól és utasítástól függően több, vagy kevesebb órajel ciklust vehet igénybe. PIC mikrovezérlők esetén minden utasítás végrehajtása 4 órajel ciklust vesz igénybe. Ez azt jelenti, hogy a mikrovezérlő valójában negyed akkora sebességen dolgozik, mint az órajel. 20Mhz-es órajel frekvencia mellett ez azt jelenti, hogy a mikrovezérlő másodpercenként 5 millió utasítás végrehajtására képes.

Időzítők

Az időzítők lényegében egyszerű számlálók, amelyeket vagy egy külső órajel, vagy a mikrovezérlő belső oszcillátora vezérel. 8 vagy 16 bites pontosságúak lehetnek. Működésük programból indítható és állítható le. Számolási esemény bekövetkeztekor megszakítást generálnak, amely segítségével pontosan időzíthető az események végrehajtása.

Watchdog timer

A Watchdog timer egy olyan speciális számláló, amelyet a program végrehajtása vezérel. Minden egyes utasítás sikeres végrehajtásakor értéke nő eggyel. Amennyiben a mikrovezérlő a program futtatása közben lefagy, vagy valami hiba történik, akkor a számláló értéke nem nő tovább. Ebben az esetben némi várakozás után ez a számláló újraindítja a mikrovezérlőt.

Ez a biztonsági szolgáltatás megakadályozza, hogy a mikrovezérlő huzamosabb ideig lefagyott állapotban maradjon.

Reset bemenet

Minden mikrovezérlő rendelkezik egy külső reset bemenettel, amelyre ha egy nyomó gombot kötünk, akkor a program végrehajtását előröl kezdi.

Megszakítások

A megszakítások olyan speciális események, amelyek hatására a mikrovezérlő felfüggeszti a normális program végrehajtását, és végrehajtja a megszakításhoz rendelt utasítássorozatot. A megszakítási utasítássorozat végeztével a normális programvégrehajtás folytatódik. Megszakítást generálhat egy külső esemény vagy egy belső esemény is. (pl. időzítő)

Mikrovezérlőtől függően a vezérlő rendelkezhet egy vagy több megszakítással is. A több megszakítással rendelkező vezérlők esetén általában lehetőség van a megszakításokhoz prioritást rendelni.

Brown-out érzékelő

A Brown-out angol kifejezés azt a jelenséget jelzi, mikor a mikrovezérlő működtetéséhez szükséges tápfeszültség a névleges minimum érték alá csökken. Ekkor az érzékelő felfüggeszti a mikrovezérlő működését. Ez a biztonsági szolgáltatás az alacsony feszültség miatt kiszámíthatatlan működés elkerülése végett van beiktatva, valamint az EEPROM/Flash memóriák tartalmának védelmében, amelyek nem szeretik, ha a tápfeszültség a névleges minimum érték alatt van (tartalomvesztés következhet be feszültségingadozás hatására).

Analóg-digitális átalakítók

A legtöbb mikrovezérlőben megtalálható. Felbontásuk mikrovezérlőtől függően változik (8,10 vagy 12 bit). PIC mikrovezérlők esetén az analóg bemenetként is használható bemenetek az A porton kapnak helyet. Az átalakítási folyamatot a programnak kell indítania. A konverziós folyamat miután végzett megszakítást vált ki.

Az analóg bemenetek igencsak hasznosak különböző szenzorok kezeléséhez, amelyek feszültség értéket állítanak elő. Ilyen szenzorok: hőmérséklet érzékelők, nyomás érzékelők, stb...

A vezérlőkbe épített analóg-digitális átalakítók jellemzője, hogy nem éppen pontosak érzékenyek a külső zajra, hőmérsékletre és egyéb dolgokra. Viszont relatívan elég gyorsak. Elméletben a legtöbb ilyen átalakító képes másodpercenként 12-14 ezer alkalommal is mintát venni, ami szenzorok esetén bőven sok.

Nagy pontossághoz vagy nagy sebességhez érdemes külső átalakítót alkalmazni. Az előző mondatban a két fogalom közötti vagy kapcsolatot az indokolja, hogy általában a gyorsaság és a pontosság az átalakítók felépítéséből következően ellentétes. Ezek azonban igen speciális alkalmazási területek. Az esetek többségében bőven meg fog felelni a vezérlő beépített átalakítója.

Soros I/O

A soros ki-és bemeneti port lehetőséget biztosít összekötésre egy másik mikrovezérlővel, vagy egy számítógéppel. Amennyiben dedikáltan nem is támogatná ezen funkciót a kiválasztott mikrovezérlő, akkor sincs gond, hiszen könnyen implementálható szabványról van szó.

Amennyiben PC-re kíván csatlakozni a mikrovezérlő, akkor nem árt egy feszültségszint konvertálót beiktatni az áramkörbe, mint a MAX232-es.

Szabványos soros porton kívül egyes mikrovezérlők rendelkeznek dedikáltan SPI (Serial Peripheral Interface) és/vagy I²C (Integrated Inter Connect) portokkal is, amelyek még szélesebb lehetőséget biztosítanak a kommunikációra a külvilággal.

EEPROM adatmemória

Beintegrált EEPROM memóriák igen gyakoriak a mikrovezérlőkben. Segítségükkel egyszerű adatmentés valósítható meg olyan alkalmazásokban, ahol erre szükség van. Mivel ROM memóriáról beszélünk, megőrzi a tartalmát a tápfeszültség megszűntekor is.

Az EEPROM memória mérete vezérlőnként igen eltérő lehet. Egyes vezérlők nem rendelkeznek beépített memóriával, mások csak 16 byte-ot kínálnak, míg megint mások akár több kilobyte-ot is.

Vezérlőtől független jellemző, hogy az EEPROM memóriáknak van egy maximális újraírási életciklusuk. Ez 1000 és 10 000 írási ciklus között mozog. Ha a memória a ciklusa végén jár, akkor az adott cellája nem fog többet működni, az oda írt adatok elvesznek. Ezért ha a belső memóriát alkalmazzuk beállítások mentésére, akkor ezt mindenképpen figyelembe kell venni. Ha sűrűn kell menteni az adatokat, akkor érdemes külső EEPROM memóriát alkalmazni, vagy egy külső SRAM memóriát akkumulátor táplálással.

Analóg komparátorok

Analóg komparátorok segítségével 2db feszültségszint hasonlítható egymáshoz. A komparátorok csak a nem belépő szintű, fejlett vezérlőkben találhatóak meg általában.

Valós idejű óra

A valós idejű óra lehetővé teszi, hogy a vezérlő rendelkezzen abszolút dátum és idő információval. Mikrovezérlőbe integrált megjelenésük nem nagyon jellemző. Olyan eszközök esetén, amelyek beépítetten nem

rendelkeznek ezen funkcióval, ott fennáll a lehetőség egy dedikált chip használatára, vagy a szoftveres implementációra.

Alvó mód

Ezen funkció a mikrovezérlőkben arra szolgál, hogy a mikrovezérlőt egy olyan módba kapcsolja, ahol csak annyi energiát fogyaszt el, amely mindenképpen szükséges a belső állapotainak megőrzéséhez.

Áram generátor / nyelő

Fontos aspektusa minden mikrovezérlőnek, hogy egyes lábain mekkora áramerősséget tolerál bemenetnek és mekkora áramot tud kiadni külső eszközök meghajtásához. PIC mikrovezérlők esetén ez az érték nagyjából 25mA. Ez az érték bőven elég ahhoz, hogy LED-eket és egyéb alkatrészeket meghajtson. Amennyiben nagyobb áramerősség, vagy nagyobb feszültségek kapcsolására van szükség, akkor egy relét, vagy egy kapcsoló tranzisztort kell a kimenetre tenni.

A nagy bejövő áramerősség ellen optocsatolókkal lehet és célszerű is védekezni. Az optocsatoló egy tokban elhelyezett fényforrásból és egy fototranzisztorból áll. A bemenő jel egy fényforrást vezérel, aminek a fényét a fototranzisztor érzékeli, majd előállítja a biztonságos feszültség-és áramerősség értéket, ami a mikrovezérlőnek kell. Az optocsatolók jóval olcsóbb alkatrészek, mint egy új mikrovezérlő.

ICD

Az ICD a PIC mikrovezérlők szolgáltatása, amely lehetővé teszi a hardveres hibakeresést. Ekkor az eszköz minden egyes utasítás végrehajtása után megáll, és a belső állapot regisztereinek értékét visszajelzi a hibakereső szoftvernek. A program végrehajtása csak a hibakereső szoftver engedélyező jelének hatására folytatódik.

PWM generátorok

A PWM generátorok az időzítőket használják impulzus szélesség modulált jel előállítására. Ezek sok mindenre alkalmazhatóak. Részletesen az alkalmazási példáknál és projekteknél esik szó a PWM jelekről. Tipikusan ezen jelek frekvenciája 1-2KHz nagyságrendben mozog 8 vagy 10 bit pontossággal. Nagyobb frekvencia vagy nagyobb pontosság érdekében érdemes külső áramkört alkalmazni.

Fontos, mikrovezérlőkre jellemző buszrendszerek

SPI buszrendszer

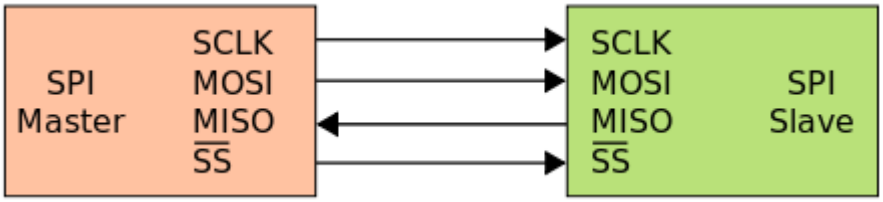
Az SPI buszrendszer egy soros átvitelű buszrendszer, amely mester-szolga (master-slave) módban működik. Ez azt jelenti, hogy a kommunikációban az egyik fél (tipikusan a mikrovezérlő) kitüntetett szerepkörrel bír. A mester eszköz feladata a kommunikáció vezérlése. Mindig csak egy mester eszköz lehet a buszrendszeren, azonban a szolga eszközök száma (elméletileg) végtelen sok lehet.

Az évek során a buszrendszer de facto szabvánnyá vált, azonban hivatalos szabvány alatt nincs bejegyezve. Emiatt az egyes eszközök közötti teljes kompatibilitás nem garantálható. Ezért, ha SPI buszon szeretnénk kommunikálni, alaposan át kell tanulmányozni a leírásában a kommunikációra vonatkozó fejezeteket.

A buszrendszer négy logikai jelből áll, melyek az alábbi táblázatban lehettek összefoglalva:

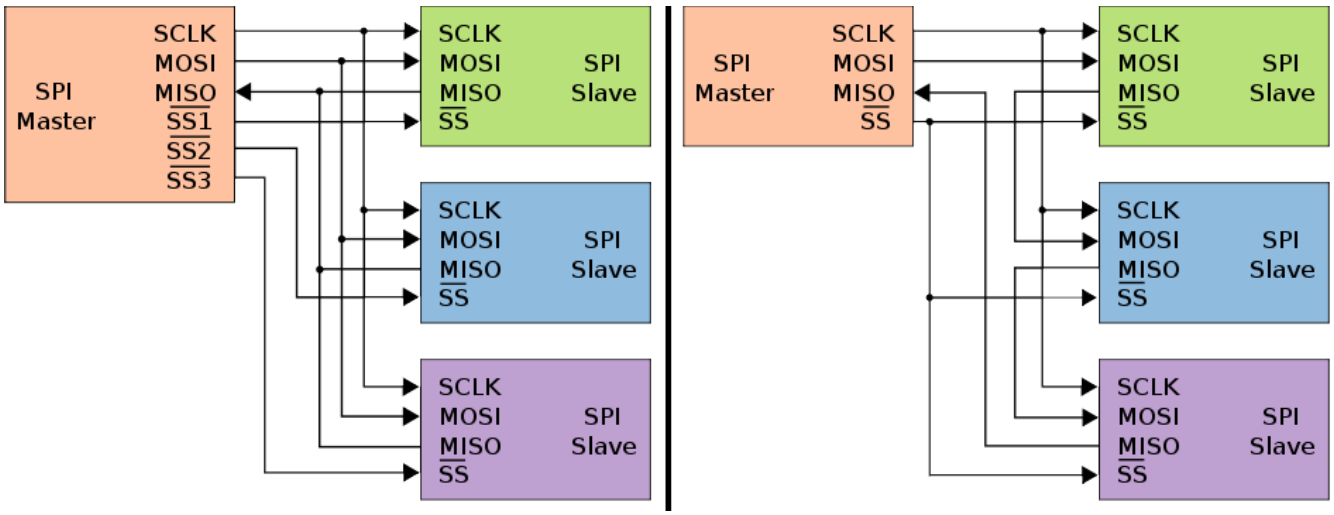
Jel jelölése	Jelentése
SCLK	Soros órajel, 1-70Mhz közötti érték, a mester eszköz határozza meg az alapján, hogy a szolga eszközök maximálisan mit támogatnak.
MOSI; SIMO	Mester kimenet, szolga bemenet. (Mester eszköz kimenő jele)
MISO; SOMI	Mester bemenet, szolga kimenet. (Szolga eszköz kimenő jele)
SS	Szolga kiválasztó jel, alacsony logikai jelszintet vesz fel, ha engedélyezve van.

44. táblázat: Az SPI buszrendszer logikai jelei



Több szolga eszköz bekötésére a buszrendszerre két opció is lehetséges. Ha a mester eszköz több szolga kiválasztó jellel is rendelkezik, akkor a szolga kiválasztó jelek kombinálásával címezhetőek az eszközök.

Amennyiben csak egy szolga kiválasztó jellel rendelkezik az eszköz, akkor több szolga eszköz elérése csak úgy oldható meg, hogy a szolga eszközöket sorba kötjük. Ezt a konfigurációt az angol szakirodalom Daisy Chain néven tartja számon. Ilyen bekötés mellett az eszközök hálózata egy Shift regiszter működéséhez hasonlít. A két kialakítási topológia közötti különbségeket az alábbi ábra jól szemlélteti:



A buszrendszer előnyei és hátrányai

Előnyök:

- *Egy időben kétirányú kommunikáció (full duplex adatátvitel)*
- *Szabad üzenethossz, tartalom és jelentés*
- *Csak négy vezeték kell a működéséhez*
- *Egyszerű hardverkialakítás*
- *A szolga eszközöket nem kell címezni az üzenetekben*
- *A jelek egyirányúak, ezáltal a galvanikus leválasztás könnyen megoldható.*

Hátrányok:

- *Csak egy mester eszköz lehet a buszrendszerben*
- *Az adatátviteli protokollban nincs hibavédelem*
- *Viszonylag csak rövid távolságon belül működik*
- *Nincs hardveres szolga visszajelzés, így a mester eszköz a semmibe is küldhet jeleket anélkül, hogy tudna róla.*
- *Nincs hardveres adatforgalom irányítás*

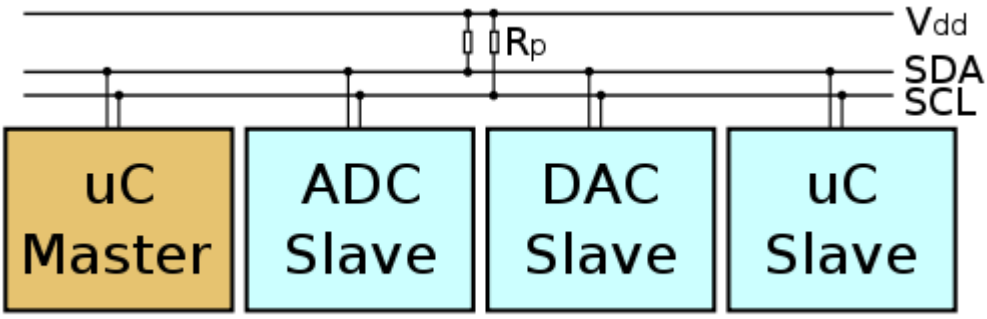
I²C buszrendszer

Az I²C buszrendszert a Philips cég találta ki olyan eszközök összekapcsolására, amelyek között nem szükséges nagy sebesség. Ez egy kétvezetékes, soros buszrendszer. Szintén alá-és fölérendelt viszonyban helyezkednek el rajta az eszközök, azonban itt többfelé rendelt eszköz is lehet a buszon. 2006 óta licencdíj mentes szabvány, azonban a szolga eszköz gyártásához szükséges HW címért még mindig jogdíjat kell fizetni az NXP vállalatnak (A Philips leányvállalata).

Az Intel által 1995-ban kifejlesztett SMBus rendszer lényegében az I ²C részhalmaza, azonban lényeges eltérések vannak a két rendszer elektromos karakterisztikájában (feszültségszintek és az áramértékek eltérőek). Továbbá az átviteli protokollt az SMBus sokkal szigorúbban definiálja a robusztusság és a kompatibilitás végett. Ezen elvek nagy részét az I²C eszközök gyártásánál is átvették. Sok olyan eszköz is kapható, amely egyszerre támogatja az SMBus és az I²C rendszert.

82. ábra: Az I²C buszrendszer logója

Az évek során számos változata jelent meg a szabványnak. Az eszközök lefele kompatibilisek, tehát egy 4.0 szabványt követő mikrovezérlő tud kommunikálni egy 1.0-s szabványt követő eszközzel is probléma nélkül. A szabványok közötti fontosabb eltérések:



Verzió	***Megjelenési év***	***Címtér***	***Szabvány által hozzáadott buszfrekvencia***	***Fontosabb jellemzők a teljesség igénye nélkül***
-	*1982*	*7 bit*	*100 Khz*	*Eredeti Philips által tervezett busz különböző chipek összekötésére.*
1.0	*1992*	*10 bit*	*400 Khz*	*Első szabványosított változat, bevezette a 400Khz-es „Fast Mode” buszfrekvenciát.*
2.0	*1998*	*10 bit*	*3,4 Mhz*	*3,4 MHz-es „High Speed Mode” bevezetése, valamint energiatakarékossági funkciók bevezetése*

2.1	*2000*	*10 bit*	*3,4 Mhz*	*A 2.0-s szabvány letisztultabb változata. Lényeges változás nem történt.*
3.0	*2007*	*16 bit*	*1 Mhz*	*1 MHz-es „Fast Mode Plus" bevezetése, valamint az eszköz címtér bővítése.*
4.0	*2012*	*16 bit*	*5 Mhz*	*5 MHz-es „Ultra Fast-Mode" bevezetése, legfrissebb szabvány.*

45. táblázat: A szabványok rövid összefoglalója

A buszra köthető eszközök számát egyrészről a címtér mérete korlátozza, másrészről a busz eredendő kapacitása is, amely maximum 400pF lehet. Ez gyakorlatban annyit jelent, hogy a busz maximum 1-2 méter távolságot képes áthidalni.

Mivel 2 típusú eszköz lehet a hálózaton (mester és szolga), ezért egyszerű kombináció számítással levezethető, hogy négyféle kommunikációs állapot lehet a rendszerben. Ezen állapotok:

- ***Mester küld***
  A mester eszköz adatot küld egy szolga eszköz felé.
- ***Mester fogad***
  A mester eszköz adatot fogad egy szolga eszköztől.
- ***Szolga küld***
  A szolga eszköz adatot küld egy mester eszköz felé.
- ***Szolga fogad***
  A szolga eszköz adatot fogad egy mester eszköztől.

A kommunikáció mindig mester küld üzemmódban indul. A mester eszköz megcímezi a használni kívánt eszközt egy üzenettel, amely start bitből, szolga cím bitekből és egy írás, vagy olvasásjelző bitből áll. (0 – olvasás, 1 – írás). Amennyiben létezik az eszköz a buszon, akkor egy ACK (nyugta, 0 értékű) bittel válaszol a mester eszköznek.

Ezután a kommunikációt a mester eszköz fogadás vagy küldés üzemmódban folytatja a címzési beállítástól függően. A szolga eszköz is hasonlóan fog cselekedni.

Az adat és címbitek küldése a legnagyobb helyi értéken lévő bit (MSB) küldésével kezdődik. A start bit jelet az SCL vezeték magas szintje mellett az SDA vezeték magas → alacsony átmenete jelzi, míg a stop bitet az SDA vezeték alacsony → magas átmenete jelzi az SCL vezeték magas szintje mellett.

A busz kommunikációs protokollja 3 fajta üzenetet definiál, melyek mindegyike egy START és STOP bittel végződik. A lehetséges üzenetek:

- *egyszeri üzenet, amiben a mester eszköz adatot közöl a szolga eszközzel,*
- *egyszeri üzenet, amelyben a szolga eszköz adatot közöl a mester eszközzel,*
- *kombinált üzenet, ahol a mester eszköz legalább 2 olvasás és/vagy 2 írás kérést küld egy, vagy több szolga számára.*

Az üzenetek felépítése szabadon hagyott, a szolga eszköz gyártójára van bízva, hogy milyen üzenetekkel kommunikál. Ezen információk általában a szolga eszköz dokumentációjában találhatóak meg. Az I²C buszrendszer definiál egy általános hívó címet is, amely a csupa 0-ból álló cím. Ha ez a cím kerül ki a buszra, akkor minden szolga eszköz címezve van egyszerre.

Ezen buszrendszert főként soros felépítésű EEPROM memóriák alkalmazzák, valamint valós idejű óra IC-k is, alacsony sebességű AD/DA átalakítók, de akad az I²C buszra illeszthető hardverek között komplett FM rádiót implementáló integrált áramkör is.

JTAG

A JTAG az elterjedtebb neve az IEEE 1149.1 szabványnak. Ezt azért dolgozták ki, hogy a beágyazott rendszerek paneljeinek a tesztelése egyszerűbb legyen. Azonban nem csak ilyen tesztelési feladatok elvégzésére alkalmas.

Lehetővé teszi a mikrovezérlő programjának hibakeresését közvetlenül az eszközön, illetve program feltöltésre is alkalmas. Legnagyobb előnye, hogy gyártó független szabványos megoldás, így az olyan mikrovezérlők, amelyek támogatják a felületet relatíve könnyen programozhatóak JTAG programozó segítségével.

A tesztelési mechanizmusa azért tud működni, mert az eszköz speciálisan van kialakítva, ami a kimeneteket illeti. Ez azt jelenti, hogy például egy mikrovezérlő esetén bármelyik kimeneti láb értéke írható és olvasható anélkül, hogy a tényleges programot módosítanánk.

Ez leginkább fejlesztés során hasznos, de SMD eszközök esetén is kifejezetten hasznos tud lenni gyártás utáni gyors tesztelés elvégzésére. Könnyen kideríthető a segítségével, hogy melyik kimeneti láb forrasztása nem megfelelő úgy, hogy kiküldünk a tesztelendő lábon adatot és ha azt az adatot bármelyik szomszédos lábon visszakapjuk, akkor több mint valószínű a két láb összeköttetésben van, ami problémát jelent, ha eredetileg nem ez volt a terv.

A teszt funkció működéséhez minden tesztelendő chip-hez szükség van egy működését leíró eszközfájlra. Ez a chip gyártójának weblapjáról szerezhető be.

A JTAG működéséhez viszonylag sok (minimum 4) vezeték kell, ezért nem minden chip-ben megoldott a támogatása. Amely eszközök támogatják az alábbi táblázatban felsorolt kivezetésekkel biztos rendelkeznek:

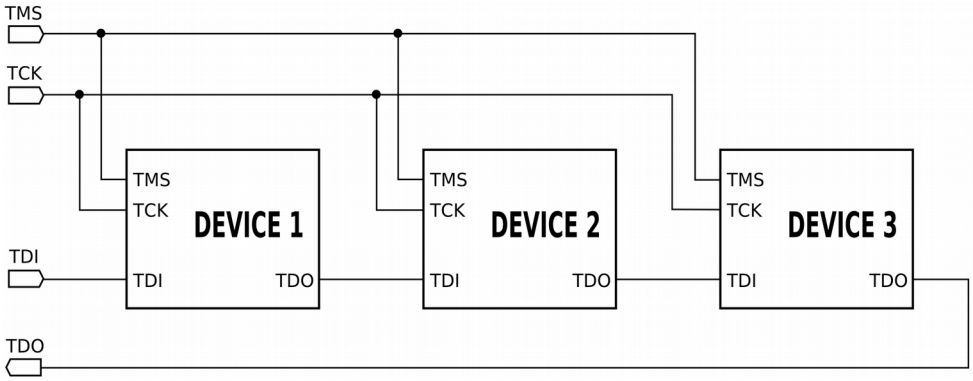
Kivezetés neve	**Funkciója**
TDI	Teszt adat bemenet
TDO	Teszt adat kimenet
TCK	Tesz órajel
TMS	Teszt üzemmód választás
TRST	Teszt reset, használata opcionális

46. Táblázat: JTAG buszrendszer lábkiosztása

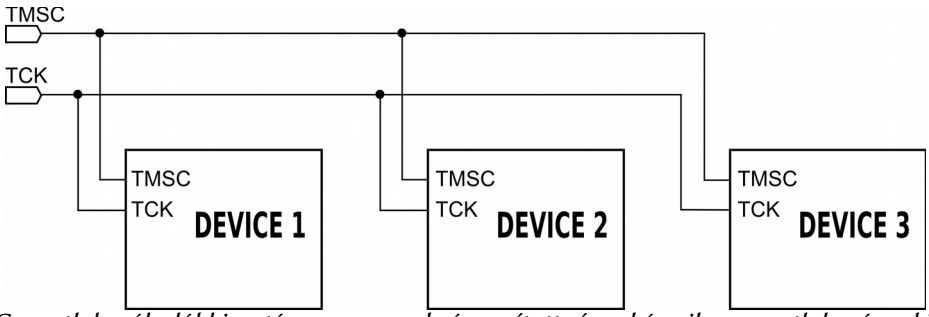
A rendszer felépítése hasonlít a Shift regiszterekhez és az SPI buszrendszerhez. Adatot küldünk be az eszköznek aTDI bemeneten, majd adatokat fogadunk a TDO kimeneten. A beküldött adat mennyiség minden esetben megegyezik a visszaérkező adat mennyiséggel.

A reset bemenet nem minden eszközön van meg, mivel opcionális. Alapértelmezetten csak a tesztelés újraindítására szolgál, nem feltétlenül indítja újra magát az egész eszközt, bár ez implementáció függő.

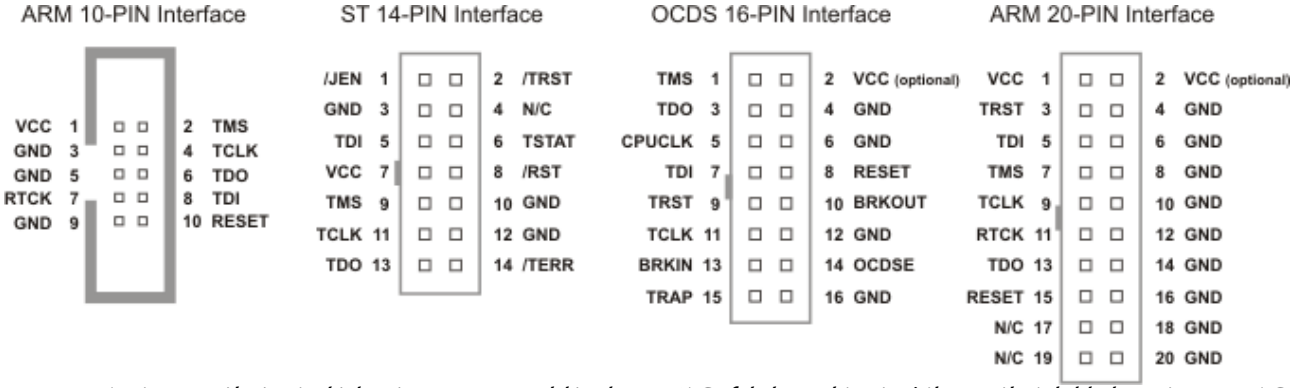
A tesztelési sebességet az órajel határozza meg. Ez különösen akkor fontos, hogy ha több eszközt kötünk ugyan arra a JTAG buszra. Több eszköz összekötése esetén az eszközök TDI és TDO lábait kell megfelelően összekötni. Az alábbi ábrán ennek az alkalmazása látható.



IEEE 1149.7 alatt lett szabványosítva a JTAG kevesebb lábat használó változata. Ez az eredeti 4+1 vezeték helyett csupán két vezetéket használ. Adat és órajel vezetéket, amelyek TMSC és TCK jelöléssel vannak ellátva. Ebben a konfigurációban az eszközöket nem lehet sorba kötni, ami lehetővé teszi azt, hogy a rendszer egyes részei le legyenek kapcsolva a tesztelés során.



A JTAG csatlakozók lábkiosztása nem szabványosított, így bármilyen csatlakozóra kivezethetőek. A gyakorlatban azonban érdemes egy gyártók által is használt lábkiosztáshoz ragaszkodni. Ezek a megoldások többségében 2 soros tűs csatlakozókat használnak. Az alábbi ábrán a népszerűbb csatlakozó lábkiosztások szerepelnek:



Amint az ábrán is látható egyes megoldások a JTAG felületet kiegészítik egyéb jelekkel. Ezért a JTAG adapter vásárlásakor érdemes figyelembe venni azt, hogy mihez szeretnénk használni, mert nem biztos, hogy bármelyik JTAG adapter jó bármelyik chip-hez.

DVS, LVDS

A DVS mozaikszó az angol Differential Voltage Signaling szavak rövidítése, amely azt jelenti, hogy differenciált feszültség alapú jelzés. Az LVDS ennek az alacsony feszültséggel működő változata.

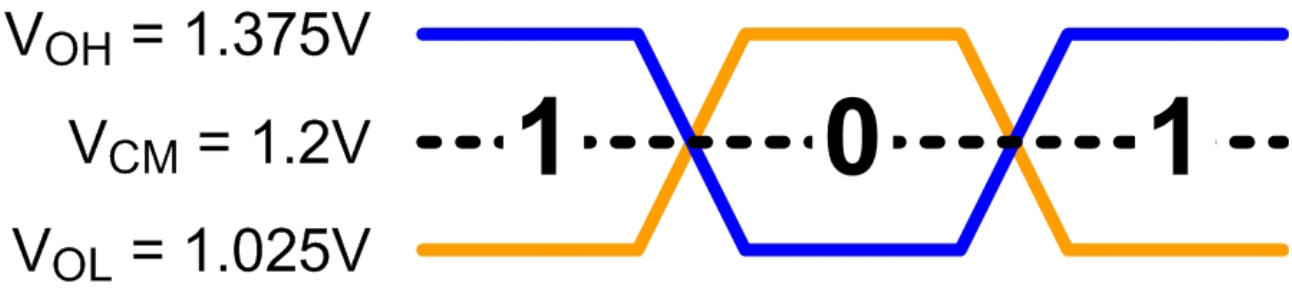
LVDS jelzési rendszert tipikusan az LCD panelek alkalmaznak, DVS jelzési sémát pedig az USB és a CAN busz alkalmaz.

A differenciált jelzés soros átviteli mód. Alapja az, hogy nem az jelenti a 0 és 1 szintet, hogy a jelhordozó vezeték 0 vagy 5V feszültségre vált. Két vezetéket használ az átvitel. Ezeket tipikusan + és – jellel szokták jelölni.

A + és – jelű vezeték alapesetben azonos feszültségszinten van, ami nem felel meg a 0 és 1 állapotnak sem. LVDS esetén ez a feszültségszint 1,2V. Az 1 állapotot az jelzi, hogy a + jelű vezeték magas logikai szintre áll, a 0 jelű vezeték pedig alacsony szintre. LVDS esetén a magas szintű állapot 1,375V. Az alacsony szint pedig 1,025V.

A 0 állapotot az jelzi a buszrendszeren, hogy a – jelű vezeték magas szintre vált, a + jelű pedig alacsony szintre.

Ezen átviteli mód előnye, hogy a zavaró hatásokat igen jól kivédi, valamint nem kell külön órajel a z átvitelhez, mivel a váltások egyértelműen jelzik a bitek hosszúságát, tehát ez egy önmagát órajelző átviteli mód. Ez lehetővé teszi azt, hogy a buszrendszeren változó sávszélességet alkalmazzunk, illetve nem kell szinkronizálni a küldő és fogadó egységet, mint I2C és SPI esetén.



CAN

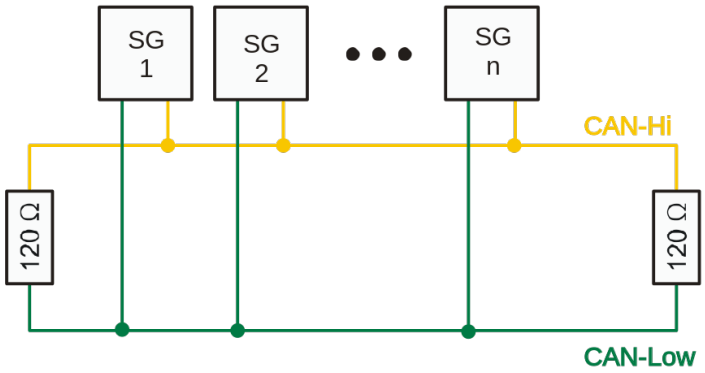
A CAN szó az angol Controller Area Network szavakból képzett mozaikszó. Egy üzenet alapú protokoll, amit kifejezetten autók számára terveztek. Egy modern autóban legalább egy tucat mikrovezérlő található. Ezen rendszerek kommunikációjának számára fejlesztették ki. Első változatát 1986-ban publikálta a Bosch. 1993 óta ISO szabvány is.

A CAN azon öt szabvány közé tartozik, amely részét képezi az ODB-II diagnosztikai csatlakozásnak. Az európai unióban 2001 (Diesel autók esetén 2004) óta elvárás, hogy minden autót fel kell szerelniük a gyártóknak ezzel a diagnosztikai eszközzel.

A CAN jogdíj mentességéből és robusztusságából adódóan ipari szabvány is lett, így ma már nem csak az autókban használt.

A Buszrendszer felépítése kicsit hasonlít az Ethernet hálózatra, azonban nem rendelkezik olyan bonyolult réteg felépítéssel, csupán négy rétegből áll a kommunikáció.

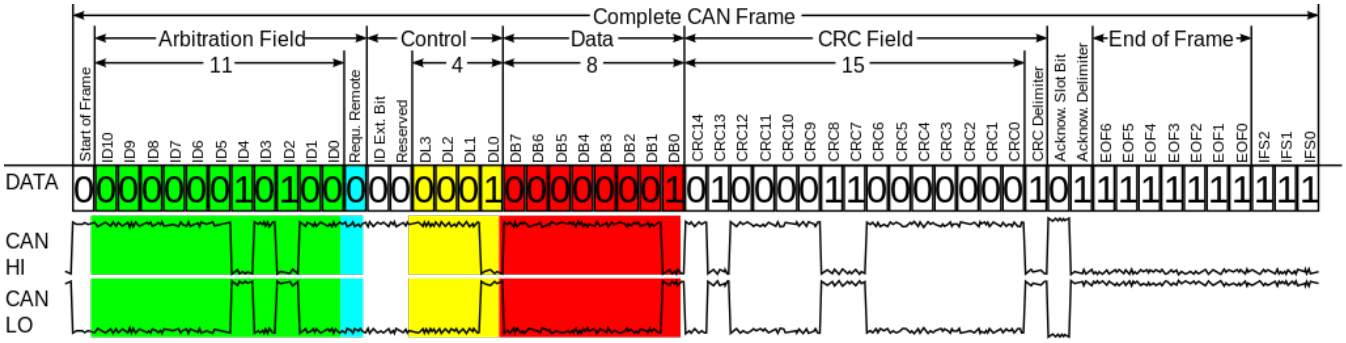
A fizikai rétege négy vezetékből áll. Ezek: tápfeszültség, földelés, adat + és adat -. Az adat + és adat – vonalak differenciált adatátvitelt alkalmaznak. Ezen jelek párhuzamosan kötendőek a többi CAN eszköz között. A buszrendszeren egyszerre több mester eszköz is lehet. Az adatvezetékeket az eszközök után mindkét oldalon egy 120 ohm-os ellenállással le kell zárni.



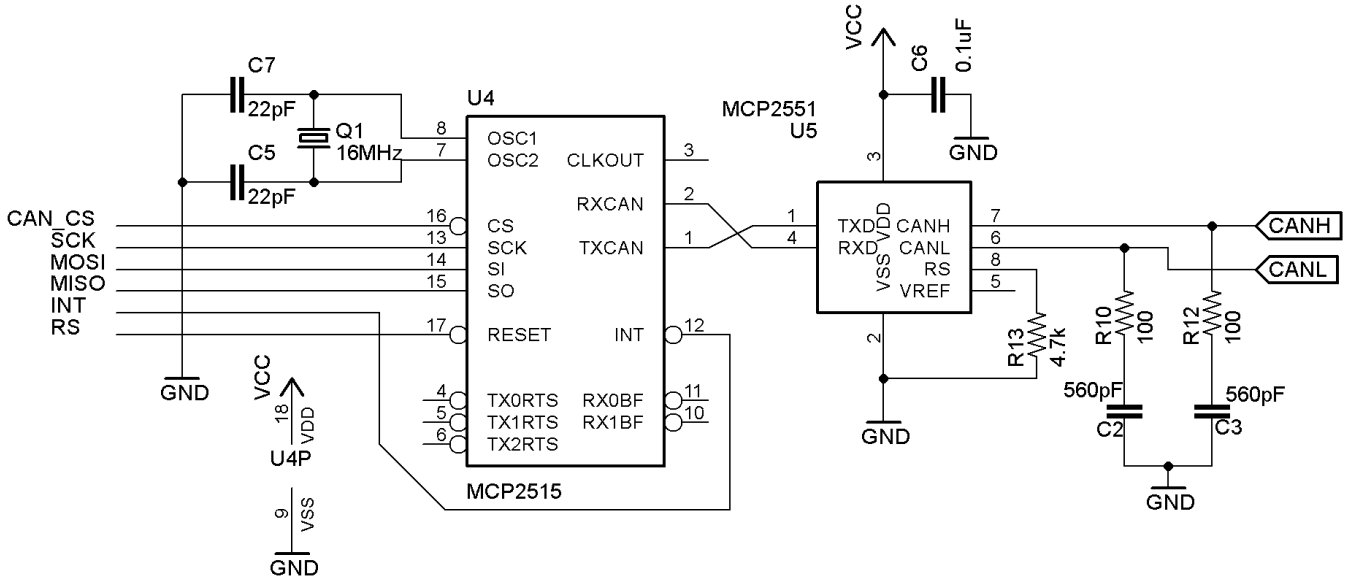
Az eszközök azonosítására 11 bites címzést alkalmaz a buszrendszer. Ez összesen 2048 különböző eszköz címzését teszi lehetővé. A CAN 2.0 B szabványban ezt kibővítették 29 bitre. A 29 bit az eredeti 11 bit azonosító mellett tartalmaz egy 18 bites kiegészítő címet. Ezt a formátumot szokás kiegészített címformátumnak nevezni. Ennek a segítségével 536 870 912 eszköz címezhető.

A buszrendszer támogatása csak komolyabb mikrovezérlőkben van hardveresen integrálva, ennek az az oka, hogy az adatátvitel során az üzenetek keretekre lesznek bontva. A keretek egy 16 bites CRC ellenőrző résszel is rendelkeznek, ami alapján el tudja dönteni a vevő, hogy a keret sérült-e vagy használható. Egy keret maximum 64 byte adatot képes tárolni.

Ennek ellenére alacsonyabb teljesítményű mikrovezérlők esetén is használható a buszrendszer, csak kiegészítő CAN vevővel.

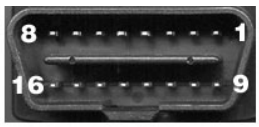


A Can illesztés megoldható a Microchip MCP2515 és MCP2551 áramkörével. A 2515 Egy SPI buszrendszerre csatlakozó CAN vezérlő. A 2551 pedig egy CAN adó/vevő.



A fenti kapcsolásban ábrázolt CAN illesztő megoldás a megszokott SPI jelek mellett egy megszakítást is publikál. Ezt a mikrovezérlő külső megszakítást fogadni tudó lábára kell csatlakoztatni, mivel ez jelzi a feldolgozott kereteket a mikrovezérlőnek.

A buszrendszer nem rendelkezik szabványosított csatlakozófelülettel, de de-facto szabványként 9 tűs csatlakozót szoktak használni az ODB-II esetén, ami a következő lábkiosztással rendelkezik.



1	5	Signal Ground
2	4	Chassis Ground
3	6	Can High J-2284
4	7	ISO9141-2 K Line
5	14	Can Low J-2284
6	10	J1850 Bus-
7	2	J1850 Bus+
8	15	ISO9141-2 L Line
9	16	Battery Power

RS-232

A PC-k szabványos RS-232 néven ismert soros portja a legalapvetőbb és a legegyszerűbb lehetőség arra, hogy a mikrovezérlőnket összekössük a számítógépünkkel. Emellett rengeteg eszköz rendelkezik a mai napig is sima, soros kapcsolódási felülettel.

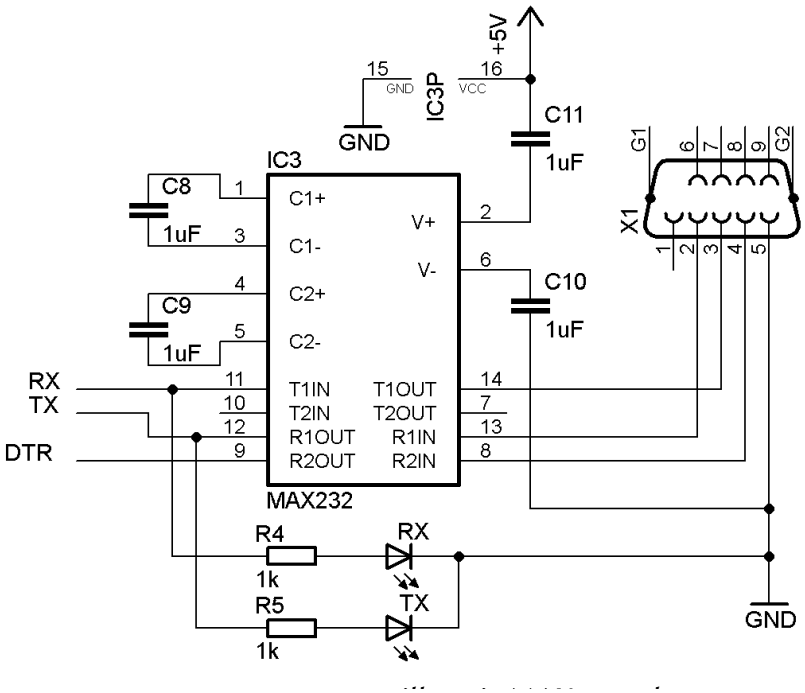
Mikrovezérlőtől függően a soros port megvalósítása lehet szoftveres vagy hardveres. Az is előfordulhat, hogy a mikrovezérlőnk nem egy, hanem több soros portot tartalmaz.

Szoftveres megoldás minden olyan mikrovezérlő esetén alkalmazható, amely rendelkezik legalább egy megszakítás fogadására képes bemenettel. Ebben az esetben a megszakítást kezelni tudó bemenet felel az adat fogadásért. Az RS-232 két adatjelből és egy pár vezérlőjelből áll. Az Adatjelek küldés, Rx és fogadás, Tx. Ahhoz, hogy a kommunikáció működjön két eszköz között az Rx és Tx vezetéketeket keresztbe kell kötni, így az egyik eszköz amit küld, az a másik eszköz fogadó pontjára érkezik.

A vezérlőjelek az adatforgalmazás működését szabályozzák. Létezik szoftveres és hardveres vezérlés. Szoftveres vezérlés esetén a kommunikáció aszinkron, vagyis adat küldés és fogadás esetén nincs várakozás a másik eszközre feldolgozás miatt. Ebben az esetben elég az Rx és Tx jeleket továbbítani. Szinkron, hardveres vezérlés esetén van szinkronizáció az eszközök között. Ez esetben az Rx és Tx jelek mellett további vezérlőjelek is kellenek. Ezen megoldás mikrovezérlők között nem igen elterjedt.

A mikrovezérlőkön alkalmazott soros portok eltérnek a PC szabványától. Mivel tipikusan a mikrovezérlők alacsony szintű jelnek 0V-ot, magas szintű jelnek meg a tápfeszültséget feleltetik meg. Ezért, hogy szabványos soros portot kapjunk ezen jelekből, a jelszinteket illeszteni kell. Erre alkalmas a MAX232-es IC. Ennek a bekötése az alábbi ábrán látható:

92. ábra. Soros port



Az RS-232 kommunikáció szimmetrikus tápfeszültséget használ. A magas szintet pozitív tápfeszültség jelenti, míg az alacsony szint egy negatív tápfeszültséghez van rendelve.

A kapcsolásban a DTR egy vezérlő jel, amely aszinkron vezérlés esetén jelzi a fogadó egységnek, hogy az adattovábbítás kész. Ezen vezérlőjel használata nem szükséges, azonban Bootloader-el programozott Arduino lapok esetén felhasználható arra, hogy a soros program átküldés után a mikrovezérlő automatikusan újrainduljon.

A mai számítógépek többségén nincs kivezetve a soros port, így ezen kapcsolódási mód PC és vezérlő között bővítőkártya beiktatásával lehetséges vagy egy USB átalakító segítségével. USB átalakító bármilyen hardveresen USB-t beszélő mikrovezérlőből fabrikálható, de kézenfekvőbb ezen célra dedikált vezérlőt alkalmazni. Ilyen vezérlőket az FTDI Chip gyárt.

A Soros port sebessége szabványosított. Sebesség helyett azonban jelzés sebességet definiál a szabvány, amit Baud-nak neveznek. A Baud szám az egy másodperc alatt továbbítható bitek számát adja meg. Aszinkron esetben 1 byte adat átvitele 10 bit átvitelt jelent, mivel minden byte elején van egy start bit és minden byte végén van egy stop bit. A szabványosított Baud értékeket az alábbi táblázat foglalja össze.

Sebesség (Baud)	***Max. Kábel hossz (m)***	***Sebesség (KB/s)***
2400	*60*	*~0,234*
4800	*30*	*~0,469*
9600	*15*	*~0,937*
19200	*7,5*	*~1,875*
38400	*3,7*	*~3,75*
56000	*2,6*	*~5,468*

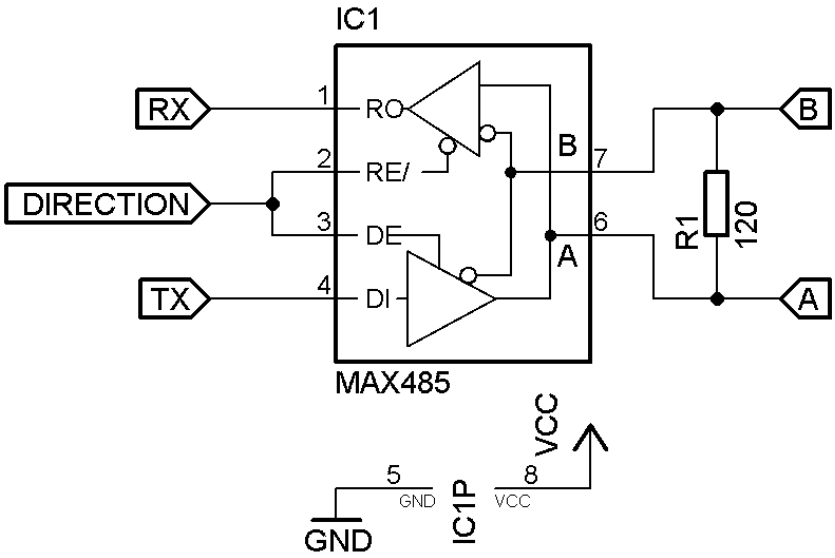
47. Táblázat: Szabványos Baud értékek

RS-485

Az RS-485 egy viszonylag nagy hatótávolságú átviteli szabvány, amely maximum 32db eszköz összekötését teszi lehetővé egy buszrendszeren. A buszrendszer maximális hossza 1,2Km lehet, de két eszköz között a távolság maximum csak 500m lehet.

A buszrendszer fél duplex és differenciált átvitelt alkalmaz, mint az USB vagy a CAN busz. Lehetőség van teljes duplex üzemmódra is, de ekkor a végpontot két darab RS-485 adóvevővel kell felszerelni.

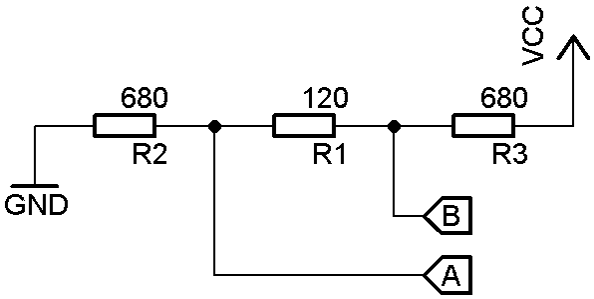
Az RS-485 adóvevő lényegében egy Soros (RS-232) átalakító, amely az RX és TX jelekből előállít egy differenciált jelet. Ilyen átalakítókból számos változat létezik, ilyen például a MAX485.



Az átalakító RX és TX lábát a mikrovezérlőn az RX és TX lábhoz kell kötni. A Direction vezeték állapota határozza meg, hogy az adóvevő küldés vagy fogadás állapotban van. Jelen kapcsolásban ha alacsony szinten van, akkor adat fogadás üzemmódban van.

A és B vezeték a differenciált kimenet. A köztük elhelyezett 120 ohm értékű ellenállás egy lezáró. Ezzel minden RS-485 eszköznek rendelkeznie kell.

A buszrendszerben A és B vezetékeket alapértelmezetten egy meghatározott feszültségszintre kell beállítani. Ez elvégezhető egy ellenállásokból alkotott feszültség osztó segítségével.



A szabvány nem definiálja az ellenállások pontos értékét. A kapcsolásban szereplő ellenállás értékek de-facto szabványként alkalmazottak.

Hasonlóan az RS-232-höz az RS-485 sem definiál szoftveres kommunikációs protokollokat, vagyis itt is a felhasználóra van bízva, hogy milyen módon kíván kommunikálni. RS-485 esetén két kommunikációs protokoll terjedt el.

A Schneider Electric által kifejlesztett Modbus és a DMX512. A Modbus egy jogdíj menetes protokoll, amelyet kifejezetten ipari elektronikai alkalmazások számára találtak ki. Három keret formátumot definiál a rendszer, amelyből egy időben csak egy használható eszközök között. Az alábbi táblázatok a Modbus keretformátumait foglalják össze.

	Bit szám	**Leírás**
Start	28	Csomag kezdete. Két csomag között legalább 3,5 bit időnyi szünet kell.
Cím	8	Eszköz címzése
Funkció	8	Funkció azonosító (Pl: bemenetek olvasása, kimenetek írása)
Adat	N * 8	Funkció végrehajtásához szükséges adatok
CRC	16	Hiba detektáló kód
Vége	28	Csomag vég jelző. Két csomag között legalább 3,5 bit időnyi szünet kell.

48. Táblázat: Modbus RTU keret felépítés

	Bit szám	**Leírás**
Start	1	Start byte, ASCII : karakter (hex: 0x3A)
Cím	2	Eszköz címzése
Funkció	2	Funkció azonosító (Pl: bemenetek olvasása, kimenetek írása)
Adat	n	Funkció végrehajtásához szükséges adatok
LRC	2	Hiba detektáló kód
Vége	2	Kocsi vissza és Soremelés jelek ASCII CR (hex: 0x0D) és LF (hex: 0x0A)

49. Táblázat: Modbus ASCII keret

	Méret Byte-ban	**Leírás**
Tranzakció azonosító	2	Szerver és kliens közti szinkronizáció
Protokoll azonosító	2	Protokoll azonosító, 0 értékű ModBus TCP esetén
Hossz	2	Az üzenet teljes hossza leszámítva az első 6 byte adatot.
Cím	1	Szolga címzése, értéke 255 ha nincs használva
Funkció kód	1	Funkció azonosító (Pl: bemenetek olvasása, kimenetek írása)
Adat	n	Funkció végrehajtásához szükséges adatok

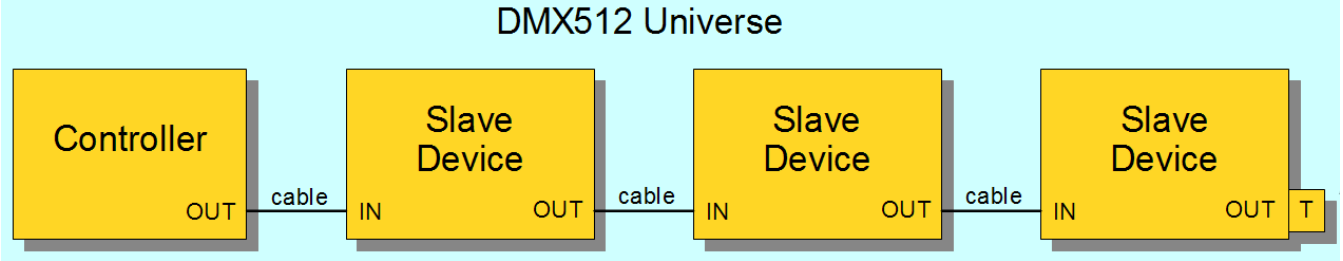
50. Táblázat: Modbus TCP keret

A DMX512 szintpadtechnikában alkalmazott. Szabványos módja lámpák és effektek fényerejének és működésének szabályzásának.

A nevében az 512 arra utal, hogy ezen kommunikációs protokoll egy buszrendszeren 512db eszköz kommunikációját teszi lehetővé. A buszrendszer építésénél különösen ügyelni kell a kábelek minőségére, a megfelelő

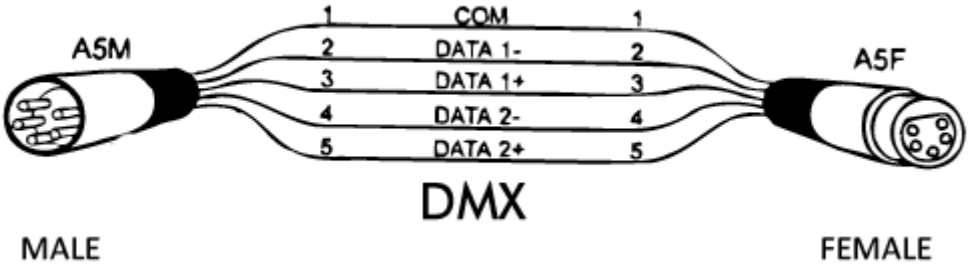
csatlakozásokra, mivel protokoll szinten nem tartalmaz semmi hibavédelmet a rendszer.

A DMX buszrendszeren egy időben csak egy központi vezérlő lehet és maximum 511 darab szolga. A szolga eszközök láncba fűzhetőek. Mindegyik rendelkezik egy be és kimeneti porttal és az alábbi ábrán látható módon kell elkészíteni a kapcsolatokat. Az utolsó szolga eszköz kimeneti portját le kell zárni egy 120 Ω-os lezáró ellenállással.



Az 512db eszköz címzése DMX szplitterek segítségével oldott meg. Szplittert abban az esetben kell alkalmazni, ha az egy buszon lévő eszközök száma 32db. A Szplitter egy olyan speciális szolga eszköz, amely több párhuzamos kimenettel rendelkezik és időosztásos alapon kapcsolja a párhuzamos kimeneteket a központi vezérlőre.

Eredetileg a DMX512-t két busszal tervezték, azonban a 2. busz sosem került bele a szabványba, ennek ellenére csatlakozókon és a kábelekben meg van a helye. A 2. busz nem rendelkezik szabványos funkcióval, gyártó és eszköz függő, hogy mire használt.

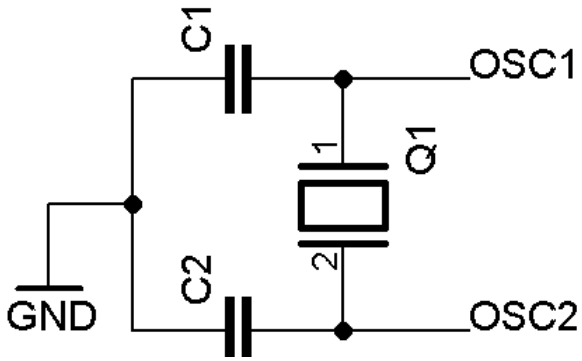


10. Áramkörök Mikrovezérlőkhöz és SOC eszközökhöz

Külső oszcillátor

A legtöbb mikrovezérlő rendelkezik beépített órajel előállító áramkörrel. Azonban ezen áramkörök pontossága nagyságrendekkel elmarad a külső oszcillátor használatától. Mivel nem egy drága alkatrészről beszélünk, ezért majdhogynem minden esetben javasolnám külső órajel előállítását. Ehhez csupán 3db alkatrészre lesz szükségünk: 2db kondenzátorra és egy rezgőkvarcra. A két kondenzátorunk kapacitása a rezgőkvarc órajelének függvényében változik.

97.



Az ábrán látható kapcsolást PIC mikrovezérlők esetén az OSC1 és OSC2 jelű lábaira kell csatlakoztatni. Más mikrovezérlők esetén hasonló jelöléseket alkalmaznak a gyártók.. A kapcsolásban szereplő C1 és C2 értékei oszcillátor sebességtől függőek. A Mikrochip által ajánlott értékeket a következő táblázat foglalja össze:

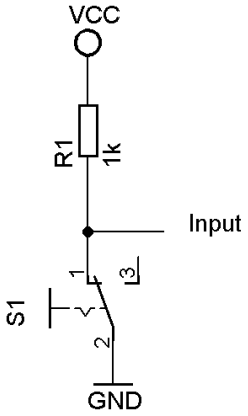
Mód megnevezése	***Órajel frekvencia***	***C1 és C2 értéke***
LP	*23 kHz*	*68-100pF*
LP	*200 kHz*	*15-33pF*
XT	*100 kHz*	*100-150pF*
XT	*2 MHz*	*15-33pF*
XT	*4 MHz*	*15-33pF*
HS	*4 MHz felett*	*15-33pF*

51. táblázat. C1 és C2 ajánlott értékei

Más gyártók mikrovezérlői esetén is alkalmazható a fenti táblázat.

Logikai bemenet

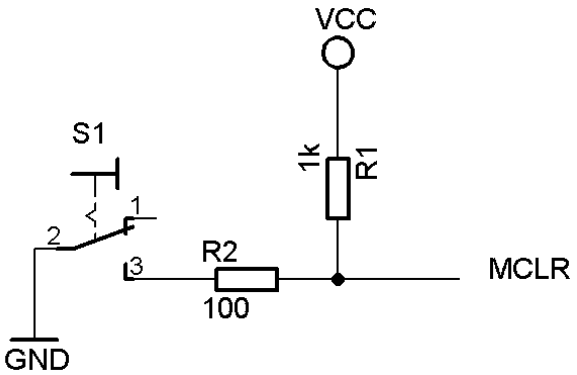
Logikai 1-es előállításához célszerű minden esetben egy olyan kapcsolást alkalmazni a bemeneten, amely alacsony szinten tartja a bemenetet, egészen a jelváltozásig. Erre az eszköz belső FET-es felépítése miatt van szükség. Ha nem lenne alapértelmezetten földön a jel, akkor előfordulhatna az, hogy a program hibásan működjön, mivel a FET érzékenységéből adódóan akár a légköri töltésváltozás is befolyásolhatja a láb állapotát. A következő ábrán látható kapcsolás alkalmazása javasolt:



Alapesetben a zárt kapcsolón keresztül a mikrovezérlő bemeneti lába mindig földön van. Azonban, ha a kapcsoló bontja a kapcsolatot, akkor a bemenet tápfeszültségre kerül, ami a bemeneten logikai egyest jelent. A kapcsolásban szereplő R1 értéke 1 KOhm és 10 KOhm között legyen. Minél nagyobb ellenállást használunk, annál jobban befolyásolja a kapcsolási sebességet. Szóval ha fontos a kapcsolási sebesség érdemes 1 KOhm -ot használni.

Hardveres reset gomb

Elvileg a Watchdog Timer-nek köszönhetően nem szabadna tartósan fagyott állapotban maradni a mikrovezérlőnek. Azonban Murphy szerint ami megtörténhet, az megfog. Ezért érdemes tenni egy reset gombot az alkalmazásunkra.



Az MCLR láb, amennyiben földpontra kerül, az eszköz újraindul. Tartós reset impulzusnak van kitéve az eszköz, akkor a mikrovezérlő azt feltételezi, hogy programozva lesz.

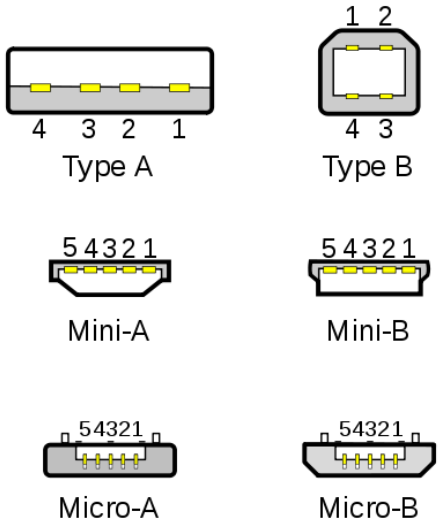
Tápfeszültség ellátás

Tápegységek tervezéséről külön könyvet lehetne írni, hiszen rengeteg paramétert lehet változtatni egy tápegységen, amitől az ideális, jó lesz az adott célfeladatra. Ezen része a könyvnek csak a mikrovezérlő tápellátás kérdésének megoldásával foglalkozik.

Mivel a legtöbb mikrovezérlő 5V feszültségről üzemel, ezért kézenfekvő megoldás lehet USB-ről táplálni a

vezérlőnket, ha megoldható. Az összes USB csatlakozó rendelkezik egy stabilizált 5V-os feszültséggel. USB 1.1 szabvány szerint 300mA áramerősséget engedélyez a vezérlő egy eszköz számára. Ezen áramerősség meghaladása után a vezérlő letiltja a portot. Tehát a tanulság az lenne, hogy ha „nagy" teljesítményre van szükségünk, akkor nem célszerű az USB tápellátását használni, hiszen nem minden esetben fog működni. 2010-es szabvány szerint egy USB porton akár 5A áram is folyhat. Azonban a kábelek vékonysága miatt nem célszerű maximumra járatni a konfigurációt.

100.



Láb	Név	Kábel szín	Leírás
1	VBUS	Piros	+5V
2	D -	Fehér	Adat -
3	D +	Zöld	Adat +
4	GND	Fekete	Földpont

52. táblázat. A és B típusú csatlakozók lábkiosztása

Láb	Név	Kábel szín	Leírás
1	+5V	Piros	+5V
2	D -	Fehér	Adat -
3	D +	Zöld	Adat +
4	ID	Nincs	Azonosítás*
5	GND	Fekete	Földpont

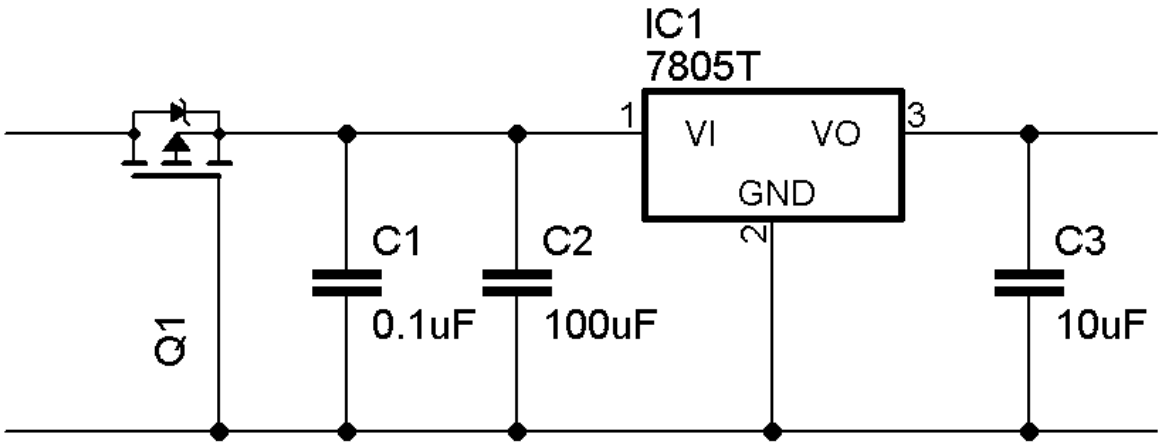
53. táblázat. Mini és Micro típusú csatlakozók lábkiosztása

**Azonosítás: kiszolgáló és szolga megkülönböztetésére szolgál. Kiszolgáló eszközökön földpontra kötött, míg szolga eszközök esetén nincs bekötve.*

Egy másik megoldás saját tápegység építése. Egy egyszerűbb tápegység a következő szerkezeti elemekből áll: transzformátor, egyenirányítás, feszültség stabilizálás, kimenet pufferelés.

101. ábra. Egy egyszerű tápegység blokkvázlata

Egyenirányításra két módszer jöhet szóba: félvezető diódával megoldott, úgynevezett Graetz – hidas megoldás, vagy transzformátor közép leválasztásos megoldás. Mindkét esetben az egyenirányító fokozat után egy 0 és Umax között pulzáló feszültséget kapunk eredményképpen.



Ezt stabilizálni kell, hogy alkalmas legyen mikrovezérlő meghajtásra. Sok opció merülhet fel ennek megoldására is. A legegyszerűbb, és talán a legelterjedtebb megoldás stabilizátorok használata.

A feszültségstabilizátor egy olyan integrált áramkör, amely bemenetén egy szabályozatlan feszültséget vár, kimenetén pedig egy tökéletes, simított feszültségérték jelenik meg. A feszültségértéket befolyásolja a használt stabilizátor fajtája. Egy közös jellemzője az összes ilyen áramkörnek, hogy lineáris működésűek. Szóval a be-és kimenet között lévő felesleges feszültséget hőenergiává alakítják át az átfolyó áramerősségtől függően, így ezen alkatrészek az esetek nagy többségében hűtést igényelnek. A stabilizátor hőteljesítménye az alábbi képlettel számolható ki:

$$P_{term} = (U_{be} - U_{ki}) \cdot I_{ki}$$

További fontos jellemzője a stabilizátoroknak az úgynevezett kiesési feszültség. Típustól függően változhat. Ez nem más, mint a minimális feszültségérték, amivel nagyobbnak kell lennie a bemeneti feszültségnek a kimeneti stabilizált feszültséghez képest. Ez általában 2V, de léteznek alacsony kiesési feszültségű változatok is. Ezek általában már 0,5V többlettel tudnak stabil kimenetet biztosítani.

A kimenet pufferelés további feszültségstabilizálás végett szükséges, ez általában kondenzátorokkal van megoldva. Minél nagyobb kondenzátorok vannak a kimeneten, annál nagyobb a rendszer energiatartaléka, így a kisebb, esetleges megingások hatása is teljes mértékben kiküszöbölhető. Az alábbi ábrán egy tipikus LM7805-tel megoldott 5V-os stabil feszültség előállító kapcsolás látható. A kapcsolás bemenetére minimum 7V-ot, de maximum 16V egyenirányított feszültséget kell kapcsolni. A kapcsolásban szereplő 0.1 µF kondenzátor zavarszűrő szerepet lát el. A legtöbb stabilizátor igényel ilyen megoldást. Konkrét értékekről a stabilizátor adatlapjában lehet tájékozódni.

A kapcsolásban, a bemeneten látható egy P csatornás MOSFET. Ez azért került bele a kapcsolásba, hogy megvédje az alkatrészeket a fordított polaritású bekötéstől. Amennyiben a polaritás véletlen felcserélődne, akkor a FET lezár, és nem engedi az áram folyását tovább. Ez a megoldás azért előnyösebb, mint egy védődióda, mert nem keletkezik feszültségesés a dióda miatt.

Nagyobb hatásfok érhető el, ha a tápegység kapcsoló üzemű. Az ilyen tápegységek hatásfoka elérheti a 98%-ot is, míg a hagyományos transzformátoros lineáris stabilizálású tápegységek hatásfoka csak 40-60% között mozog megvalósítástól függően.

További előnye ezen fajta tápegységeknek, hogy egy adott teljesítményű kapcsoló üzemű tápegység sokkal kisebb transzformátort fog tartalmazni, mint az azonos teljesítményű lineáris változat. Ebből kifolyólag a berendezés súlya csökkenthető.

Az ilyen tápegységek ára egy bizonyos teljesítmény alatt drágább, mint a lineáris megoldások, viszont nagy teljesítmény mellett olcsóbb tud lenni.

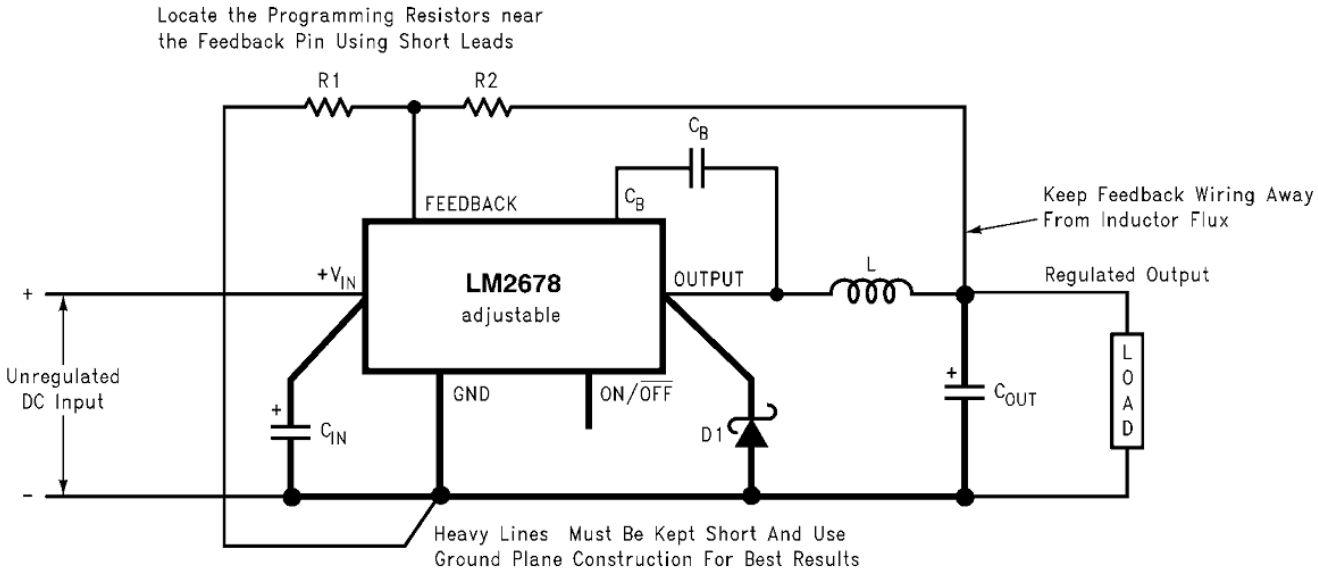
Az ilyen tápegységek úgy működnek, hogy a bemeneti oldalt közvetlenül a kimeneti terhelés függvényében szabályozzák. A szabályozás a transzformátor primer oldalán egy tranzisztor gyors kapcsolgatásával történik. A

kapcsoló frekvencia elérheti az 1-2 MHz-t is. Ehhez, hogy ez lehetséges legyen a bejövő 230V-os hálózati feszültséget egyenirányítják a szabályzás előtt. Ebből adódóan az ilyen tápegységek működhetnek egyen feszültségről is. Ezen elvet alkalmazzák a nagy hatásfokú DC/DC konverter integrált áramkörökben is.

A kapcsolóüzemű tápegységek tervezéséről és megépítéséről szintén egy külön könyvet lehetne írni. A legtöbb esetben az SMD kivitelű alkatrészeknek köszönhetően házilag nem igen lehetséges komplett tápegység építése. Ezért a legegyszerűbb, ha veszünk egy, az igényeknek megfelelő komplett tápegységet. Számos gyártó gyárt ilyen modulokat, és ezek nem is túl drágák.

Amennyiben akkumulátorról szeretnénk üzemeltetni a kész mikrovezérlős kapcsolásunkat, akkor érdemes kapcsolóüzemű DC/DC konverterekkel megoldani a feszültség konverziókat lineáris stabilizátorok helyett, főleg ha fontos az, hogy mennyit működik az eszköz.

A DC-DC konverterek kapcsoló üzemű mivoltuk miatt, legalább 75% hatásfokra képesek, azonban a kimeneten minden esetben lesz valamennyi zaj. A zaj nagysága terheléstől és a kapcsolás összeállításától függ. Ezen okból kifolyólag DC-DC konverterrel nem érdemes műveleti erősítőknek tápfeszültséget szolgáltatni abban az esetben, ha hangfrekvenciás jelet szeretnénk velük erősíteni, mivel a tápegység zaja átkerülhet a kimenetre erősítetten, így a kimenet zajos lehet. Jelenleg számos DC-DC átalakító áramkör érhető el a piacon. Rengeteg gyártó gyárt ilyen áramköröket. Ezek közül talán a legegyszerűbben használható az LM2678.



Az LM358-ADJ típusa esetén R1 és R2 értéke határozza meg a kimeneti feszültséget. Az L tekercs és a Cout kondenzátor egy szűrőt alkot, amely a kimenet stabilizálásáért felelős. D1 dióda a tekercs védődiódája, nem hagyható el a kapcsolásból.

A kapcsolásban a vastag fekete vonallal jelölt részek a stabilizátor IC közelében kell elhelyezni, amennyire csak lehet.

A kimeneti feszültség az alábbi képlet segítségével határozatozható meg:

$$U_{ki} = 1.21\,V \cdot \left(1 + \frac{R_2}{R_1}\right)$$

R1 és R2 értékét nem érdemes túl nagyra választani, mivel ha visszacsatoló ág ellenállása $1\,M\Omega$ *fölé kerül, akkor a visszacsatolás antennaként működik és a kimeneten ez zaj formájában jelentkezik ami a hatásfok csökkenését eredményezi.*

Az adatlap ajánlása szerint R1 értékét érdemes fixálni egy $1\,k\Omega$ *ellenállással. Ebben az esetben a kívánt kimeneti feszültséghez az R2 ellenállás nagysága a következő képlet segítségével határozható meg:*

$$R_2 = 1\,K\,\Omega \cdot \left(\frac{U_{ki}}{1.21\,V} - 1 \right)$$

A kapcsolásban szereplő további alkatrészek ajánlott értékei:

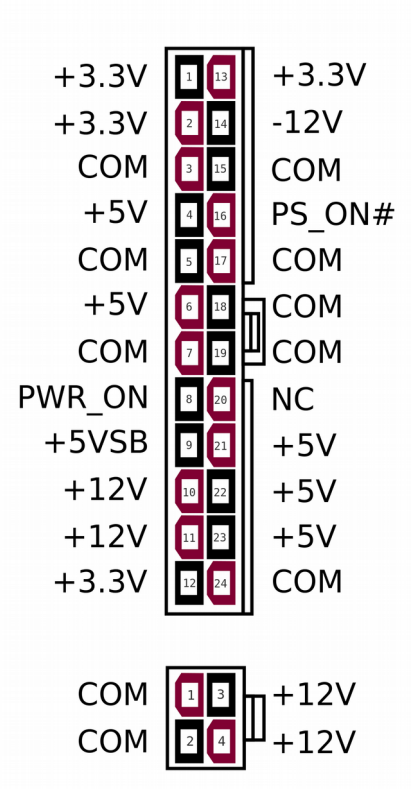
CI_n	$220\,\mu F$	L	$22\,\mu H, 6\,A$
C_B	$10\,nF$	C_{out}	$1000\,\mu F$

ATX tápegységek használata

Nagy áram igényű projektek esetén kézenfekvő megoldás egy kapcsolóüzemű ATX tápegység használata. Ezen tápegységek könnyen beszerezhetőek és igen nagy áram leadására képesek változatos feszültségek mellett. Az egyes ágak terhelhetősége nem azonos. Egy adott tápegység maximális paramétereiről minden esetben az adatlapján tájékozódhatunk. Ezen részletesen fel van sorolva minden ág terhelhetősége.

105. Ábra: Egy tápegység adatlapja

Egy szabványos ATX tápegység számos csatlakozóval rendelkezik. A legfontosabb csatlakozó az alaplap 20 vagy 24 tűs csatlakozója. Ezen csatlakozón található meg az indító jel. Az alábbi táblázatban a 24 tűs csatlakozó lábkiosztása és a szabványos vezeték színek lettek összefoglalva.



Funkció	Vezeték szín
+3.3V	Narancs
COM/GND	Fekete
+5V	Piros
+5VSB/Standby	Lila
+12V	Sárga
–12V	Kék
PWR_ON	Szürke
PS_ON#	Zöld
NC/-5V	Fehér

54. Táblázat: ATX funkciók és vezetékek színei

A tápegység elindításához annyit kell tenni, hogy a Zöld, PS_ON funkciójú vezetéket földre kötjük. Ameddig földre van kötve ez a jel, addig üzemel a tápegység. Gyártó függő, hogy mennyire jól reagál a tápegység arra, ha terhelés nélkül üzemel. Ezért előfordulhat, hogy terhelés nélkül be sem kapcsol, vagy ha be is kapcsol, akkor nagyon instabil lesz a működése.

Ezért a +12V ág és a föld közé célszerű beiktatni egy fix terhelő ellenállást a szabályzás stabilitása érdekében. A minimális terhelés szintén gyártó és típus függő. Nagyjából 5Watt terheléssel már stabil szabályzást

kapunk. Ehhez egy 33 Ohm-os 9 Wattos teljesítmény ellenállást kell beépíteni. Az is előfordulhat, hogy a tápegység több terhelést igényel, azonban ha 10-15 Watt terhelés esetén sem stabilizálódik a kimenet, akkor az egységgel problémák vannak.

A PWR_ON vezeték és a föld között +5V feszültség mérhető, ha a tápegység sikeresen elindult. Ez egy visszajelző vezeték, ami kivezethető egy LED-re.

A Lila Standby vezeték és a föld pont között a tápegység kikapcsolt állapotában is 5V feszültség mérhető, ha a tápegység csatlakoztatva van az elektromos hálózatra. Ez a számítógépek esetén USB kiválasztott USB portok meghajtására és egyéb funkciókra szolgál. Ezen ág maximális terhelhetősége 1 amper. Ez bőven elég a szabályozó elektronikánk működtetésére, így ha nincs szükség a nagy teljesítményre a tápegység nagy áramú része kikapcsolható.

A Fehér vezeték régebbi tápegységek esetén –5V feszültséget szállított, azonban ezt mára már nem használják a számítógépek, viszont kompatibilitási okokból nem változott a csatlakozó kiosztása. Újabb tápegységek esetén egyszerűen nem kötik be ezt a vezetéket.

A 3.3V vezetékkel párhuzamosan szokott futni egy barna vezeték is. Ez egy kompenzációs vezeték, amit arra használnak, hogy a 3.3V ágon a vezetéken jelentkező feszültségesést kiküszöböljék.

A tápegység azonos színű vezetékei közösíthetőek gond nélkül, sőt igen nagy teljesítmény eléréséhez ajánlott is közösíteni őket, mivel a kijövő vezetékpárok egyenként túláram és rövidzárlat védettek. Így ha aszimmetrikusan terheljük le a tápegységet, akkor ki fog kapcsolni.

Az ágak összekötését a tápegységen kívül megfelelő csatlakozók segítségével lenne célszerű megoldani, mivel az egység szétszerelése után a garancia nem lesz érvényesíthető, ha valami történne az egységgel. Amennyiben ez nem fontos szempont, akkor az egység kikapcsolása után a szétszedéssel várjuk egy jó 20-30 percet, mivel az egységben található nagyfeszültségű kondenzátorok érintése feltöltött állapotban potenciálisan halálos tud lenni.

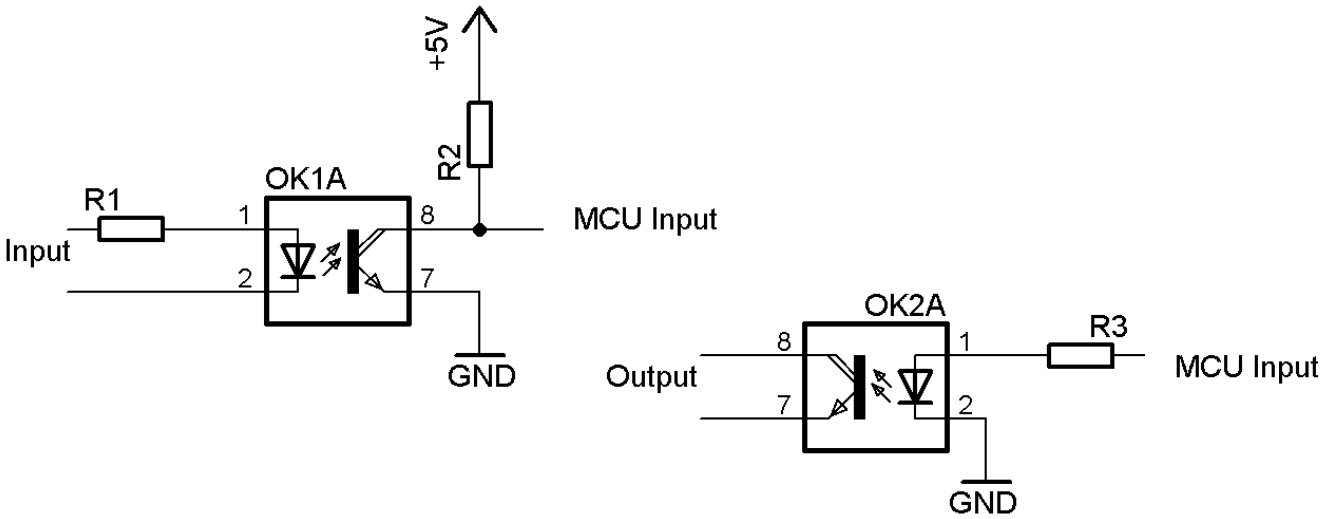
Továbbá érdemes tudni, hogy egy ATX tápegység sem tud megfelelően működni, ha nem földelt konnektorra csatlakozik. Ilyen esetben az egység burkolatának érintésekor megrázó élményben lehet részünk. Ennek az az oka, hogy a közép földelt transzformátor kialakítás miatt az tápegység föld pontja a földelésre csatlakozik. Amennyiben ez nincs meg, akkor a feszültség értékek egymáshoz viszonyítva el fognak térni a tervezett értékektől.

Nagyobb terhelhetőség érdekében több tápegység párhuzamosan köthető, azonban ezen tápegységek sorba nem köthetőek nagyobb feszültségek eléréséhez. Amennyiben nagyobb feszültségre van szükségünk effektív módon, akkor érdemes utánanézni az elektronikai üzletek kapcsoló üzemű tápegység modul kínálatának.

Ki és bemenetek védelme optocsatolókkal

A mikrovezérlő ki és bemenetei nem nagy feszültségek és teljesítmények elviselésére lettek tervezve, így mondhatni igen érzékenyek. Főleg abban az esetben, ha 3,3v-os változatú mikrovezérlőt alkalmazunk. A ki és bemenetek megvédésének egyik kézenfekvő módja az, hogy optocsatolóval védjük meg.

Az optocsatoló nem más, mint egy tokba zárt LED dióda és egy fototranzisztor. Amennyiben a LED világít, akkor a tranzisztor kapcsol, amennyiben nem, akkor nem kapcsol. Mivel a két oldal között tényleges fizikai összeköttetés nem áll ezért lehetséges az, hogy az egyik oldalon nagyobb feszültséget használjunk, mint a másikon.



A bemeneti oldal a legtöbb esetben egyen feszültséget vár, de léteznek olyan csatolók is, amelyek két egymással szembefordított LED-et tartalmaznak párhuzamosan kapcsolva, így az ilyen típusok váltakozó feszültség érzékelésére is alkalmasak.

A rajzon szereplő bemenet védelmi áramkör fordított logikájú, mivel ha a LED nem világít, akkor a mikrovezérlő bemenetére 5 V feszültség kerül. A LED előtét ellenállása minden esetben a csatoló paramétereitől és a használt bemeneti feszültségtől függ. A tranzisztor kollektor ágában szereplő ellenállás értéke 1K és 10K Ohm között szabadon választható. Az érték növelésével csökken a kapcsolási sebesség.

Ha nagy teljesítményeket szeretnénk kapcsolni, akkor a csatoló kimenetét egy tranzisztor bázisának meghajtására érdemes felhasználni, mivel a csatoló tranzisztora nem nagy áramok kapcsolására lett kifejlesztve.

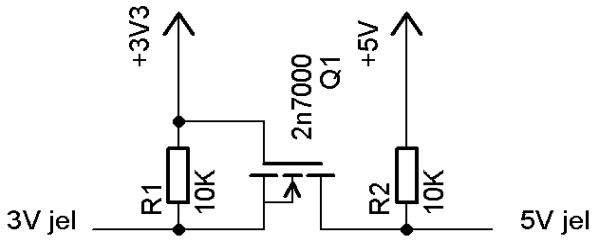
Jelszint illesztés

A digitális elektronikában leginkább két jelszint terjedt el. Az 5 Voltot használó TTL logika és a 3,3V-os jelszinteket alkalmazó CMOS logika. Bár igen sok áramkör 5V toleráns, előfordulhat azonban, hogy egy specifikus áramkör csak 3,3V-on hajlandó működni. Nagy teljesítményű mikrovezérlők esetén találkozhatunk a problémával leginkább.

Ebben az esetben, hogy az áramkör ne sérüljön, de mégis kommunikálni tudjon eltérő jelszinteket alkalmazó eszközökkel illeszteni kell a jelszinteket. Erre számos módszer létezik, azonban többségük hátránya, hogy egyirányú illesztést végeznek. (Bufferek, mint a 74 125 vagy a 4050) Vagy 5V szintről 3,3V-ra vagy pedig 3,3V szintről 5V-ra. Ez különösen akkor okoz problémát, ha a buszrendszerünk szimmetrikus, azaz ugyan azon a vezetéken két irányba áramlanak az adatok (Pl. I²C buszrendszer)

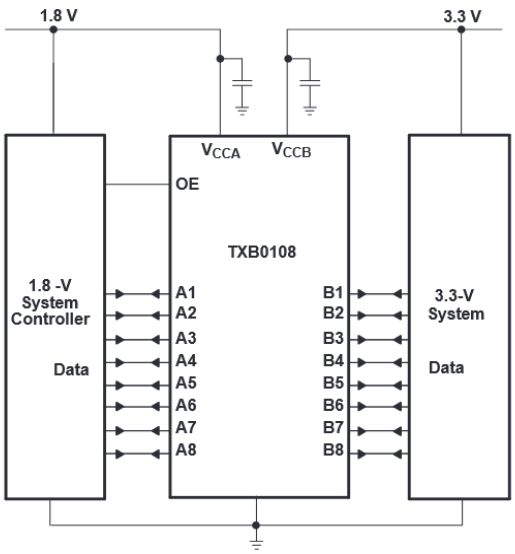
Ebben az esetben speciális illesztő áramkör használata lenne javasolt. Ennek a hátránya, hogy csak smd kivitelben kapható ilyen áramkör, ami otthoni tervezésben nem igen használható. A probléma könnyen megoldható egy MOSFTET tranzisztor alkalmazásával és két ellenállással az alábbi kapcsolás szerint:

107. Ábra:



Amennyiben egyszerre 4-5 jel között kell illeszteni, akkor érdemes elgondolkodni a TXB0108 áramkör alkalmazásán, amely 8 ki és bemenettel rendelkezik, viszont csak SMD tokozásban létezik, viszont 1,8V (Processzorok jelszintje) és 3,3V jelszintek között is tud illeszteni.

108



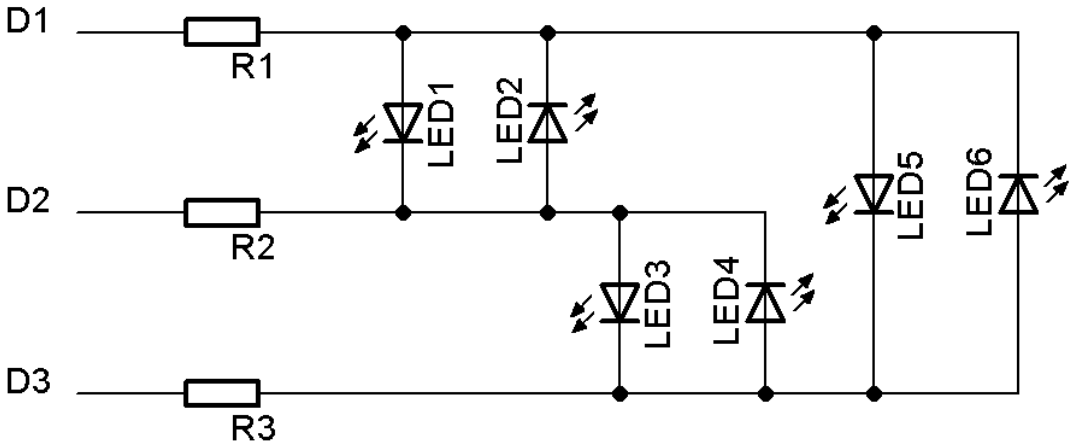
. Ábra: TXB0108 alkalmazása

LED Multiplexelés

A kimenetek spórolása érdekében a LED meghajtásra használt kimenetek multiplexelhetőek több kapcsolás

segítségével is. Az itt tárgyalt kapcsolás előnye, hogy nem kell hozzá aktív elem, közvetlenül a mikrovezérlő kimenetére illeszthető a kapcsolás. Az alábbi kapcsolás három vezeték felhasználásával 6 LED meghajtására képes, az elvet követve azonban bővíthető a kapcsolás további LED-ek meghatására.

109.



A kapcsolás használatához az alábbi vezérlőjeleket kell kiadni:

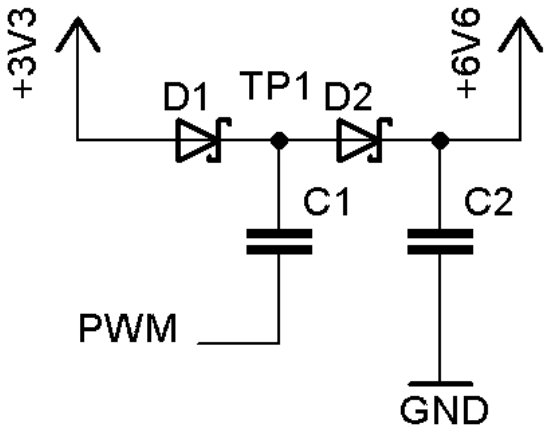
D1	D2	D3	L1	L2	L3	L4	L5	L6
L	L	L	0	0	0	0	0	0
L	H	I	1	0	0	0	0	0
H	L	I	0	1	0	0	0	0
I	L	H	0	0	1	0	0	0
I	H	L	0	0	0	1	0	0
L	I	H	0	0	0	0	1	0
H	I	L	0	0	0	0	0	1

55. Táblázat: Multiplexelt LED meghajtó vezérlési táblázata

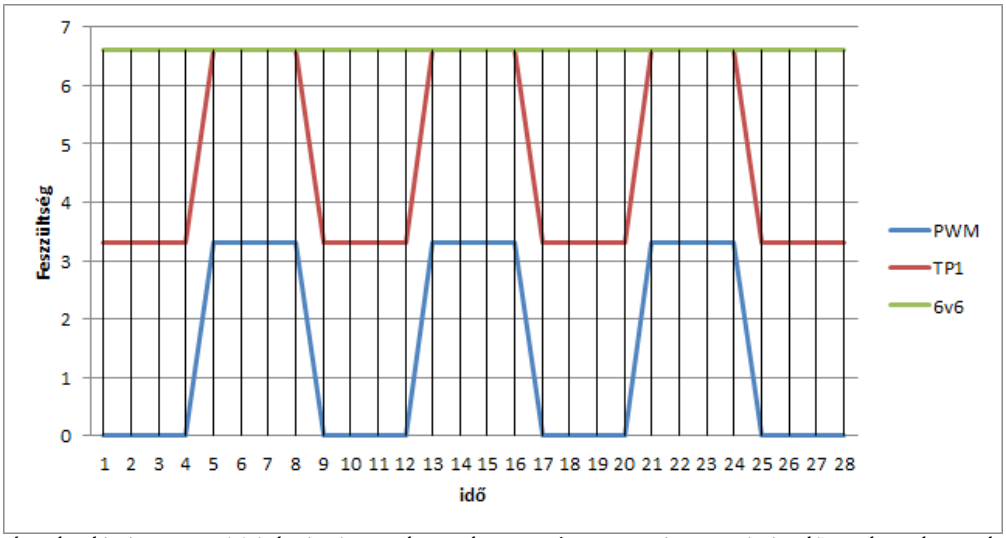
A táblázatban L betűvel vannak jelölve az alacsony szintre húzandó kimenetek, H betűvel a magas szintre kapcsolandóak és I betűvel a bemeneti állapotra kapcsolandóak.

Feszültség duplázó és triplázó

Az itt bemutatott feszültségduplázó áramkör akkor jön jól, ha 3V feszültségről üzemeltetjük a mikrovezérlőnket és nincs lehetőségünk nagyobb feszültség használatára, de egy 5V feszültséget igénylő eszközt is meg kellene hajtanunk.



A kapcsolás működése igen egyszerű. A D1 diódán keresztül a C1 kondenzátor feltöltődik 3,3V-ra, ha a PWM jel alacsony szinten van. Ezután, ha a PWM jel magas szintre kerül, akkor már eleve 3,3V-ra van töltődve a kondenzátorunk, ami nem tud kisülni a 3,3V tápfeszültség felé D1 miatt, így a feszültség szint 6,6V-ra emelkedik. D2 dióda megakadályozza, hogy áram folyjon vissza a tápfeszültség felé, C2 kondenzátor pedig a PWM jelből csinál egy stabil egyen feszültséget.

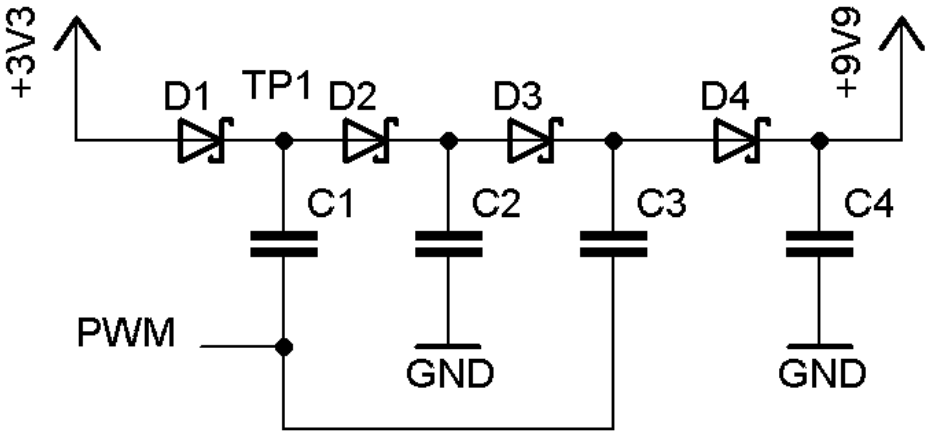


Mivel a duplázást a PWM jel végzi, amely szakaszos, így nagy áramot igénylő eszközök meghajtására nem képes az áramkör. Továbbá a kimeneti feszültség nem lesz 6,6v köszönhetően a diódák nyitófeszültségének. 3,3v meghajtásról Schottky diódák használatával ~6V körüli feszültség állítható elő.

C1 és C2 értékét variálva növelhető a kimenet terhelhetősége. További fontos tényező a PWM jel kitöltési tényezője és frekvenciája.

Gyakorlatban legalább 1KHz frekvenciát érdemes alkalmazni 50% kitöltési tényezővel. C1 és C2 értékét pedig a terhelés nagysága határozza meg. Érdemes pár száz mikro Farád környékén megválasztani a kondenzátorokat.

A kapcsolás némi átalakítással használható triplázóként is. A triplázó kapcsolás előnye, hogy a kimenetére nyugodtan tehető egy stabilizátor kocka, amely segítségével stabil kimeneti feszültség állítható elő.

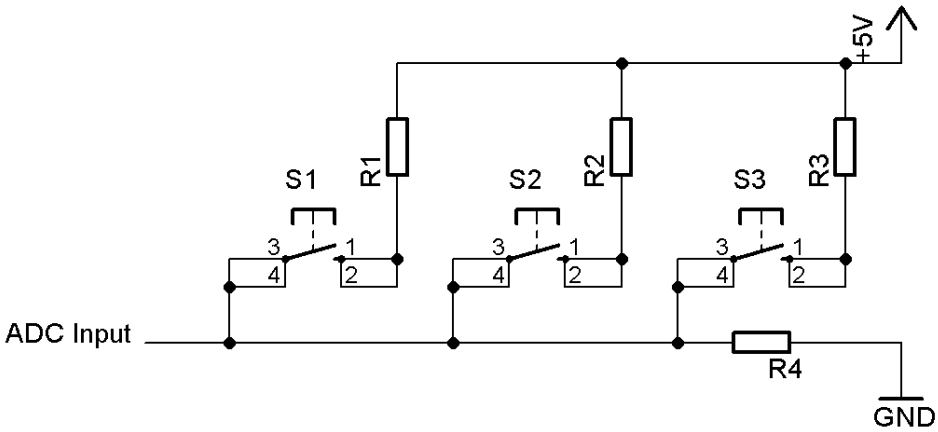


Természetesen a duplázó és a triplázó sem képes végtelen nagy áram továbbítására. A továbbítható áram nagyságát C1 és C2 komponensek határozzák meg, valamint a táplálás paraméterei. Ha a PWM jelforrás 25mA áram szolgáltatására képes 3,3V mellett, akkor ha 100%-os hatásfokkal számolunk és a feszültséget duplázzuk, akkor a legjobb esetben 12,5mA áram szolgáltatására képes az áramkör kimeneti oldala.

A valóságban azonban a 100%-os hatásfok elérhetetlen a diódák és a kondenzátorok nem ideális viselkedése miatt. Viszont 12,5mA elég lehet kijelzők és műveleti erősítők meghajtásához.

Egy bemenettel több gomb olvasása

Szinte minden mikrovezérlő rendelkezik legalább egy analóg-digitális átalakítóval. Ezt felhasználva egy analóg bemenet segítségével több nyomógomb kezelését is megvalósíthatjuk. A gombokat feszültségosztóként kell elrendezni.



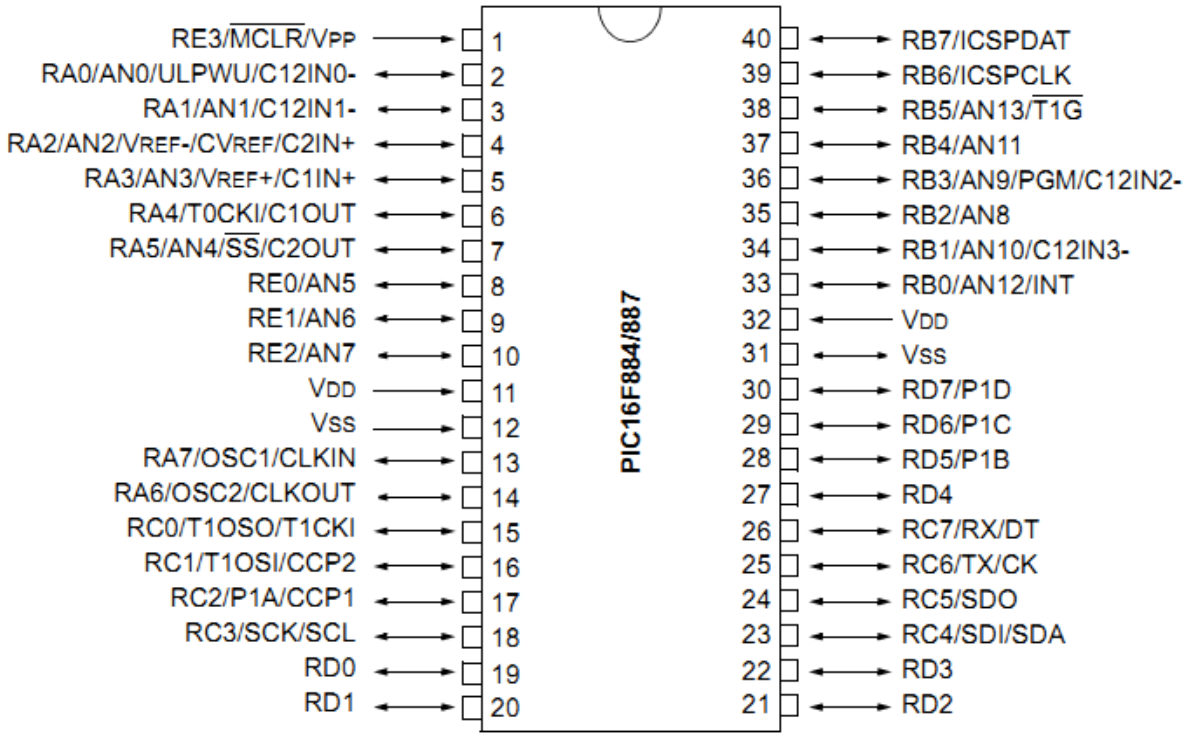
A fenti ábrán látható kapcsolás elméletileg az Analóg Digitális átalakító pontosságáig bővíthető. Ez 8 bit esetén 255 különböző gomb kezelését tenné lehetővé. A valóságban azonban jóval kevesebb kezelése lehetséges, mivel az ellenállásoknak van egy tűrése, valamint a vezérlőkbe integrált analóg-digitál átalakítók sem éppen zajmentesek.

Megfelelő ellenállás arányok választásával a kapcsolás 20-30 gomb kezelését teszi lehetővé egy 10 bites átalakítóval

11. Ismerkedés a PIC16F887-es mikrovezérlővel

A PIC16F887-es mikrovezérlő a PIC16 széria egy olyan modellje, ami igen elterjedten használt egyszerű felépítése és programmemória mérete miatt. Ezen mikrovezérlő a PIC16 882/883/884/886/887 mikrovezérlőkkel egyetemben különböző tokozásokban érhető el. Ezen eszközök lábkiosztása egymással kompatibilis. Egyetlen eltérés köztük a belső memória és EEPROM memória mérete. Az alábbi ábrán a 40 lábas DIP tokozás lábkiosztása látható. A többi tokozásról azok otthoni nehéz szerelhetősége miatt nem esik szó.

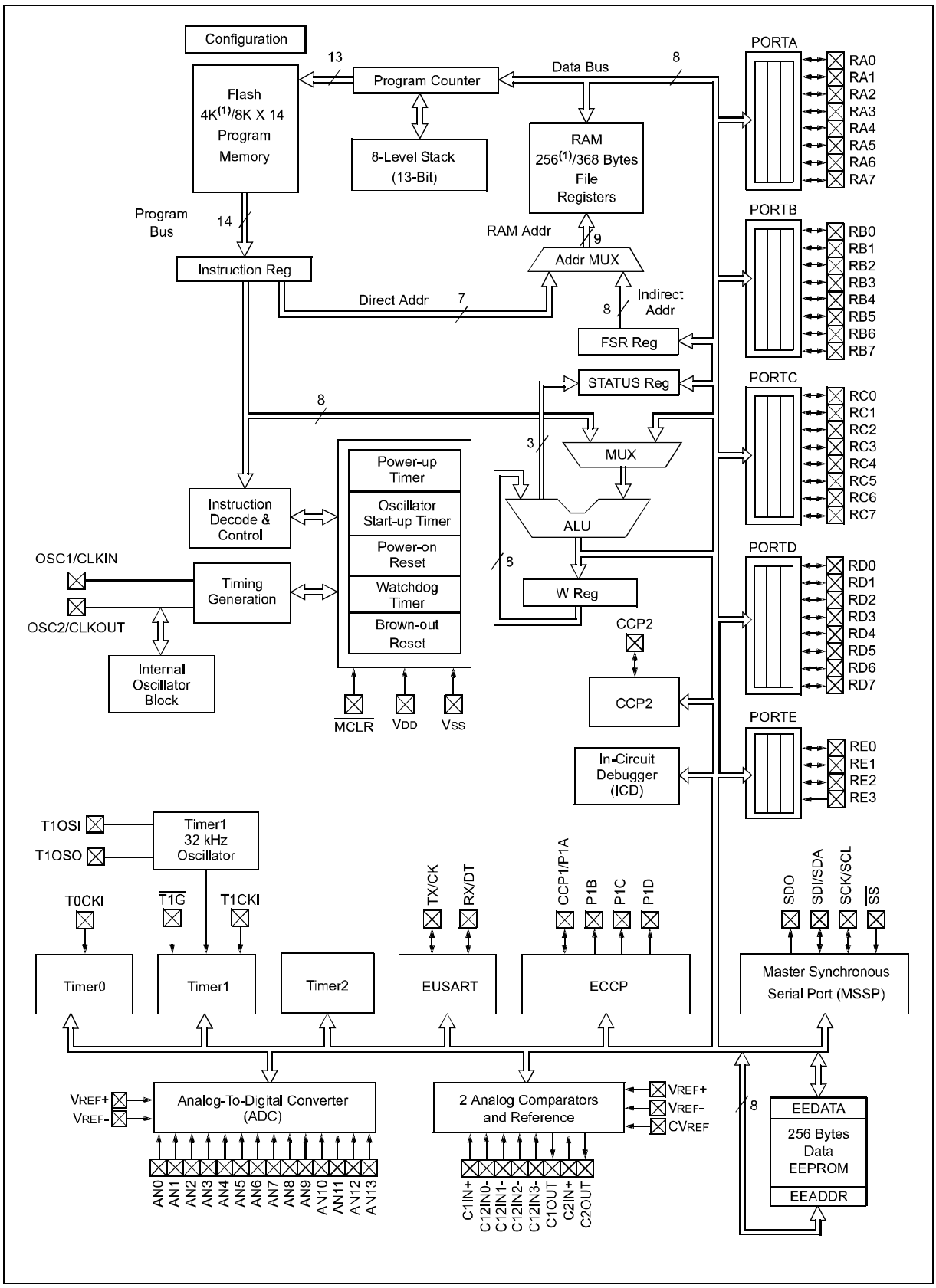
40-pin PDIP



A fenti ábrán jól látható, hogy egyes lábaknak több funkciója is van. Ezek működése vagy a program során speciális regiszterek beállításával történik, vagy speciális áramköri megoldásokkal. Azért alkalmazzák az egy lábhoz több funkció tervezési sémát, mivel elvétett esetekben van szükség a mikrovezérlő minden funkciójára. Ezáltal csökkenthetőek a gyártási költségek.

A ki és bemeneti portok lábai: RAx, RBx, RCx, RDx, és REx, ahol az x a kimenet bitjeinek száma (0-7-ig, vagy speciális esetben több vagy kevesebb).

Az eszköz teljes belső felépítésének blokkdiagramja a következő ábrán látható. Az ábrán látható buszokon a számok a buszvezeték szélességét jelölik.



A mikrovezérlő számos regiszterrel rendelkezik konfigurációbeállítási és egyéb okokból. Ezen belső regiszterek szervezése fájl alapú. A fájl alapú regiszterek abban térnek el a normál regiszterektől, hogy memóriacímhez hasonlóan is kezelhetőek. Két hivatkozási módjuk létezik: memóriacím alapú és rögzített nevű. A 16F887-es mikrovezérlő belső regisztereinek felépítése a következő képen látható:

	File Address		File Address		File Address		File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General Purpose Registers	116h	General Purpose Registers	196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers 96 Bytes	20h – 3Fh, 40h – 6Fh	General Purpose Registers 80 Bytes	A0h – EFh	General Purpose Registers 80 Bytes	120h – 16Fh	General Purpose Registers 80 Bytes	1A0h – 1EFh
accesses 70h-7Fh	70h – 7Fh	accesses 70h-7Fh	F0h – FFh	accesses 70h-7Fh	170h – 17Fh	accesses 70h-7Fh	1F0h – 1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Unimplemented data memory locations, read as '0'.

Note 1: Not a physical register.

2: PIC16F887 only.

A PIC16F887-es mikrovezérlő fontosabb adatai:

- *14Kb programmemória*
- *368 byte RAM*

- *256 byte adat EEPROM*
- *3db időzítő (2×8 bit + 1×16 bit)*
- *14 csatornás, 10 bites A/D konverter*
- *2db komparátor*

Az olvas, módosít, ír probléma

Az olvas, módosít, ír probléma (angol kifejezéssel: Read, Modify, Write, vagy röviden RMW) probléma egy a PIC architektúrából következő probléma, amely olyan esetekben jelentkezik, mikor egy kimenet értékét 0-ról egyre állítjuk.

Mikor a programunkban egy láb kimeneti értékét, például az RB0 láb értékét szeretnénk módosítani 0-ról egyre, akkor először a mikrovezérlő processzora ***beolvassa*** *a PORTB regiszter értékét, amely belsőleg tárolja a B port összes lábának értékét. Ezután a processzor ezt a 8 bites értéket* ***módosítja****, majd visszaírja a PORTB regiszterbe.*

Olvasás során nem a PORTB regiszter tartalma olvasódik be, hanem a kimenetek állapota közvetlenül.

A probléma akkor jelentkezik, mikor a kimenetre kötött terhelés befolyásolja a kimenet logikai állapotát. Ez akkor fordulhat elő, ha a terhelés nagy kapacitív, vagy induktív ellenállással rendelkezik.

Gyakorlatban ez már jelentkezhet akkor is, ha áramkorlátozó ellenállás nélkül kötünk diódát/LED-et a mikrovezérlő lábaira. Mivel a diódáknak karakterisztikájukból adódóan van egy kapacitív jellegük.

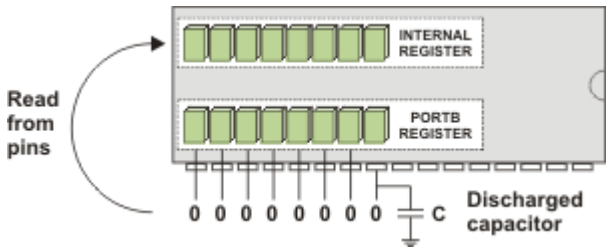
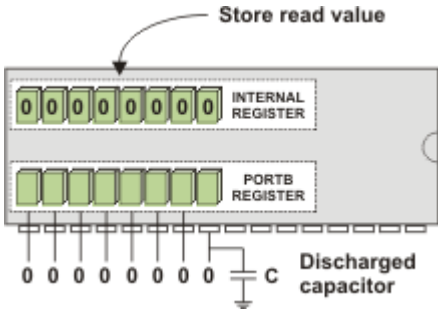
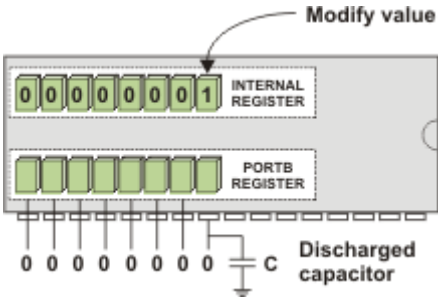
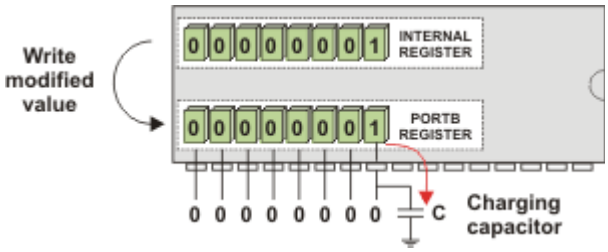
A probléma elkerülése érdekében érdemes várakozásokat beiktatni a programba, vagy ha PIC18-as családba tartozó mikrovezérlőket használunk, akkor szóba jöhet a LATx jelű regiszterek használata alternatív megoldásnak. Ezen regiszterek csak a PIC18-nál fejlettebb PIC mikrovezérlőkben vannak.

Példaképpen nézzünk egy olyan esetet, ahol a kimeneti láb és a föld pont közé beiktatunk egy kondenzátort. Ebben az esetben némi időt vesz igénybe a láb értékének módosítása után, hogy ténylegesen a kívánt kimenet jelenjen meg rajta, mivel, ha a lábat 1-es állapotba állítjuk, akkor a kondenzátornak töltődnie kell. 0-ra állítás után pedig a kondenzátornak ki kell sülnie.

Analizáljuk a következő problémát:

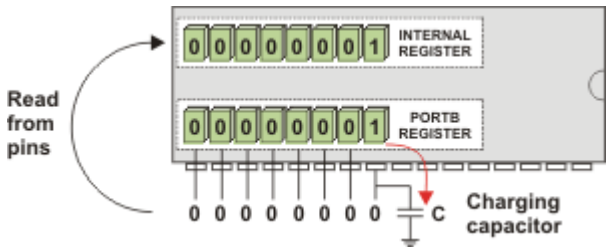
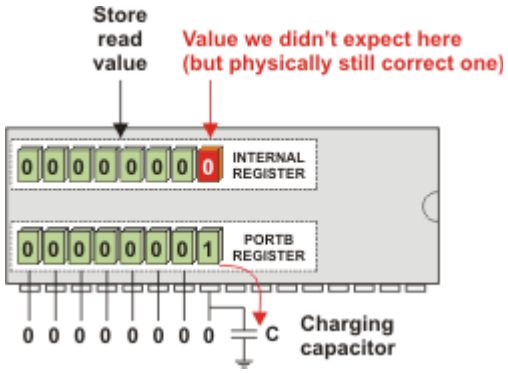
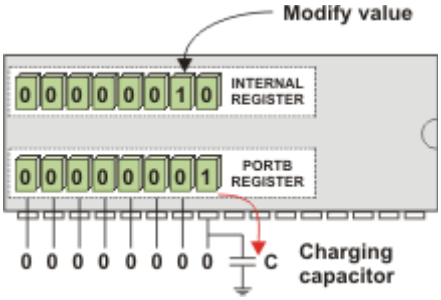
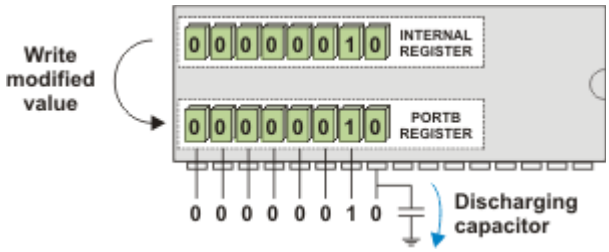
```
PORTB.B0 = 1;
PORTB.B1 = 1;
```

Első utasítás végrehajtásának menete:

1.  *Kimenetek tényleges értékének olvasása. PORTB kimarad a folyamatból.*	*2.*  *Olvasott értékek tárolása egy belső, átmeneti regiszterben*
3.  *Belső, átmeneti regiszter tartalmának módosítása az utasításnak megfelelően.*	*4.*  *Átmeneti regiszter tartalom visszaírása PORTB regiszterbe, ami hatására a kimeneten is megjelenik az érték. A C kondenzátor töltődni kezd.*

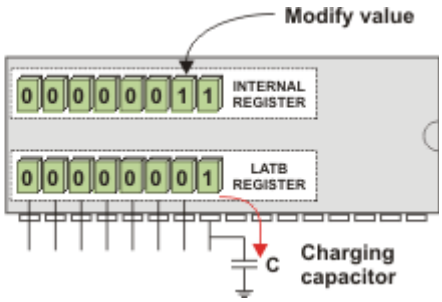
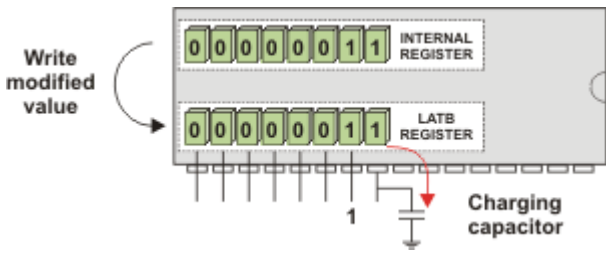
56. táblázat. Az első sor végrehajtásának menete

Második utasítás végrehajtásának menete

1.  *Kimenetek tényleges értékének olvasása. PORTB kimarad a folyamatból.*	*2.*  *Mivel C kondenzátor még mindig töltődik, ezért a kimeneten a feszültség alacsony szintű, ezért azon lábon a kimenet értéke 0 lesz.*
3.  *Belső, átmeneti regiszter tartalmának módodosítása az utasításnak megfelelően.*	*4.*  *Átmeneti regiszter tartalom visszaírása PORTB regiszterbe, ami hatására a kimeneten is megjelenik az érték. A C kondenzátor elkezd kisülni.*

57. táblázat. A második sor végrehajtásának menete

A probléma könnyen elkerülhető a LATx regiszterek használatával, mivel itt a tényleges LATx regiszter tartalma kerül beolvasásra és módosításra. Ezt akkor érdemes alkalmazni, mikor várakozások beiktatása nem lehetséges.

1.  *LATB átkerül egy átmeneti regiszterbe, ahol módosul értéke az utasításnak megfelelően.*	*2.*  *Az átmeneti regiszter értéke átkerül LATB-be, minek hatására megjelenik az érték a kimeneten.*

58. táblázat. LATx regiszterek működése

Assembly utasítások

Elsősorban ebben a könyvben az eszköz C nyelvű programozásáról van szó. Azonban néha előfordulhat, hogy valamit assembly nyelven kell megoldanunk, vagy mondjuk egy kész assembly programot kell átírnunk C-be

és kibővítenünk. Ezen esetekben nem kerülhetjük el az assembly utasítások megismerését. Szerencsére „csak” 35 darab van ezekből. A PIC16F887 utasításai három kategóriába sorolhatóak:

1. *Byte orientált regiszter műveletek*
2. *Bit orientált regiszter műveletek*
3. *Vezérlő és érték utasítások*

A teljes utasításkészletet az alábbi táblázatok foglalják össze:

Utasítás neve		***Leírás***	***Ciklusok száma***	***Érintett státusz regiszterek***
ADDWF	*f, d*	*W és f összeadása*	*1*	*C, DC, Z*
ANDWF	*f, d*	*Bitenkénti ÉS művelet W és f között*	*1*	*Z*
CLRF	*f*	*f törlése*	*1*	*Z*
CLRW		*W törlése*	*1*	*Z*
COMF	*f, d*	*f regiszter tartalmának negálása*	*1*	*Z*
DECF	*f, d*	*f értékének csökkentése 1-gyel*	*1*	
DECFSZ	*f, d*	*f értékének csökkentése 1-gyel, kihagyás ha 0*	*1 (2)*	
INCF	*f*	*f értékének növelése 1-gyel*	*1*	*Z*
INCFSZ	*f*	*f értékének növelése 1-gyel, kihagyás a 0*	*1 (2)*	
IORWF	*f, d*	*Bitenkénti VAGY W és f között*	*1*	*Z*
MOVF	*f*	*f mozgatása*	*1*	*Z*
MOVWF	*f*	*W mozgatása f-be*	*1*	
NOP		*Üres művelet*	*1*	
RLF	*f, d*	*f bitforgatása balra*	*1*	*C*
RRF	*f, d*	*f bitforgatása jobbra*	*1*	*C*
SUBWF	*f, d*	*W kivonása f-ből*	*1*	*C, DC, Z*
SWAPF	*f, d*	*f regiszter alsó és felső 4 bitjének cseréje*	*1*	
XORWF	*f, d*	*Bitenkénti KIZÁRÓ VAGY W és f között.*	*1*	*Z*

59. táblázat. Bájt orientált regiszter műveletek

Utasítás neve		Leírás	Ciklusok száma	Érintett státusz regiszterek
BCF	f, b	f bit törlése	1	
BSF	f, b	f bit beállítása	1	
BTFSC	f, b	f bit teszt, kihagyás ha 0	1 (2)	
BTFSS	f, b	f bit teszt, kihagyás ha 1	1 (2)	

60. táblázat. Bit orientált regiszter műveletek

Utasítás neve		Leírás	Ciklusok száma	Érintett státusz regiszterek
ADDLW	k	Érték és W összeadása	1	C, DC, Z
ANDLW	k	Bitenkénti ÉS művelet érték és W között	1	Z
CALL	k	Eljárás meghívása	2	
CLRWDT		Watchdog Timer törlése	1	TO, PD
GOTO	k	Ugrás memóriacímre	2	
IORLW	k	Bitenkénti VAGY művelet érték és W között	1	Z
MOVLW	k	Érték W-be mozgatása	1	
RETFIE		Megszakításból visszatérés	2	
RETLW	k	Visszatérés W értékével	2	
RETURN		Eljárásból visszatérés	2	
SLEEP		Standby üzemmódba lépés	1	TO, PD
SUBLW	k	W kivonása értékből	1	C, DC. Z
XORLW	k	Bitenkénti KIZÁRÓ VAGY érték és W között.	1	Z

61. táblázat. Érték és vezérlő utasítások

Utasításokban szereplő jelölések

Jelölés	Leírás	Jelölés	Leírás
f	Fájl regiszter cím 0×00 és 0×7F között	PC	Programszámláló
W	Munka regiszter	TO	Idő túllépés bit
b	Bit helyzet egy 8 bites regiszteren belül	C	Átvitel bit
k	Adat, konstans, címke	DC	Szám átvitel bit
x	Nem számító bit érték.	Z	Nulla bit
d	Cél kiválasztó bit. 0: Cél a W regiszter, 1: cél egy f regiszter	PD	Kikapcsolás bit

Utasítások leírása

Utasítás	*Szintaxis*	*Operandusok*	*Működés*	*Érintett státusz*	
ADDLW	`ADDLW k`	$0 \leqslant k \leqslant 255$	*(W) + k → (W)*	*C, DC, Z*	*W regiszter tartalma és k*
ADDWF	`ADDWF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(W) + (f) → (cél)*	*C, DC, Z*	*W regiszter és f regiszte eredmény a W regiszterbe kerül.*
ANDLW	`ANDLW k`	$0 \leqslant k \leqslant 255$	*(W) ÉS (k) → (W)*	*Z*	*W regiszter tartalma é eredmény tárolása a W re*
ANDWF	`ANDWF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(W) ÉS (f) → (cél)*	*Z*	*W regiszter tartalma és végzése. Ha a d 0, akkor eredmény az f regiszterbe*
BCF	`BCF f, b`	$0 \leqslant f \leqslant 127$ $0 \leqslant b \leqslant 7$	*0 → (f<b>)*	*Nincs változás*	*b bit az f regiszterben 0-r*
BSF	`BSF f, b`	$0 \leqslant f \leqslant 127$ $0 \leqslant b \leqslant 7$	*1 → (f<b>)*	*Nincs változás*	*b bit az f regiszterben 1-re*
BTFSC	`BTFSC f, b`	$0 \leqslant f \leqslant 127$ $0 \leqslant b \leqslant 7$	*Kihagyás ha (f<b>) = 0*	*Nincs változás*	*Ha b bit az f regiszter hajtódik végre. Ha 0, akk esetben 2 ciklust vesz igé*
BTFSS	`BTFSS f, b`	$0 \leqslant f \leqslant 127$ $0 \leqslant b \leqslant 7$	*Kihagyás ha (f<b>) = 1*	*Nincs változás*	*Ha b bit az f regiszter hajtódik végre. Ha 1, akk esetben 2 ciklust vesz igé*

62. táblázat. A PIC16F887 Utasításai #1

Utasítás	*Szintaxis*	*Operandusok*	*Működés*	*Érintett státusz*	
CALL	`CALL k`	$0 \leqslant k \leqslant 2047$	*(PC)+1 → TOS, k->PC<10:0>, (PCLATH<4:3>) → PC<12:11>*	*Nincs változás*	*Szubrutin hívás. Először a v majd a cél cím a PC regiszte elő.*
CLRF	`CLRF f`	$0 \leqslant f \leqslant 127$	*00h → (f)* *1 → Z*	*Z*	*f regiszter tartalma törlődik,*
CLRW	`CLRW`	*Nincs*	*00h → (W)* *1 → Z*	*Z*	*W regiszter tartalma törlődi*
CLRWDT	`CLRWDT`	*Nincs*	*00h → WDT* *0 → WDT előszorzó* *1 → TO, 1 → PD*	*TO, PD*	*Watchdog időzítő nullázása előszorzóját is. A TO és a PD*
COMF	`COMF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) → (cél)*	*Z*	*f regiszter tartalma negálód regiszterbe, ha d 1, akkor ere*
DECF	`DECF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) –1 → cél*	*Z*	*f regiszter értékének értének eredmény a W regiszterbe, h*

63. táblázat. A PIC16F887 Utasításai #2

Utasítás	***Szintaxis***	***Operandusok***	***Működés***	***Érintett státusz***	
DECFSZ	`DECFSZ f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) –1 ⟶ cél, kihagyás, ha az eredmény 0*	*Nincs változás*	*f regiszter értékének érté* *eredmény a W regiszterbe* *kerül. Ha az eredmény 1,* *Ha 0, akkor egy NOP u* *ciklust vesz igénybe az uta*
GOTO	`GOTO k`	$0 \leqslant k \leqslant 2047$	*k ⟶ PC<10:0>* *PCLATH<4:3> ⟶ PC<12:11>*	*Nincs változás*	*A Goto egy feltétel nélküli ug* *és a PCLATH regiszterből*
INCF	`INCF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) + 1 ⟶ cél*	*Z*	*f regiszter értékének növelés* *regiszterbe, ha a d 1, akko*
INCFSZ	`INCFSZ f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) +1 ⟶ cél, kihagyás, ha az eredmény 0*	*Nincs változás*	*f regiszter értékének növelés* *regiszterbe, ha a d 1,* *Ha az eredmény 1, akkor* *akkor egy NOP utasítás h* *igénybe az utasítás*
IORLW	`IORLW k`	$0 \leqslant k \leqslant 255$	*(W) VAGY k ⟶ (W)*	*Z*	*W regiszter és k érték közö* *regiszterbe kerül.*
IORWF	`IORWF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(W) VAGY k ⟶ (W)*	*Z*	*W regiszter és f regiszter tart* *akkor az eredmény a W* *regiszterbe kerül.*

64. táblázat. A PIC16F887 Utasításai #3

Utasítás	*Szintaxis*	*Operandusok*	*Működés*	*Érintett státusz*	
MOVF	`MOVF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) → cél*	*Z*	*f regiszter tartalma a cél regi W regiszterbe, ha a d 1, al*
MOVLW	`MOVLW k`	$0 \leqslant k \leqslant 255$	*(k) → (W)*	*Nincs változás*	*k érték betöltése a W regiszte*
MOVWF	`MOVWF f`	$0 \leqslant f \leqslant 127$	*(W) → (f)*	*Nincs változás*	*W regiszter tartalmának mozg*
NOP	`NOP`	*Nincs*	*Nem végez műveletet*	*Nincs változás*	*Nem végez műveletet.*
RETFIE	`RETFIE`	*nincs*	*TOS → PC, 1 → GIE*	*nincs*	*Visszatért megszakításból. Ve a globális megszakításkezelő*
RETLW	`RETELW k`	$0 \leqslant k \leqslant 255$	*(k) → (W) TOS → PC*	*nincs*	*Visszatérés eljárásból értékke teteje (TOS) betöltődik a PC*
RETURN	`RETURN`	*nincs*	*TOS → PC*	*nincs*	*Visszatértés eljárásból érték*
RLF	`RLF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*Lásd leírás*	*C*	*Bitforgatás balra egy bittel ca állapota változik. Ha a d érté 1, akkor az f regiszterbe kerül*
SLEEP	`SLEEP`	*nincs*	*0 → WDT, 1 → TO, 0 → PD*	*TO, PD*	*Alvó állapotba belépés, Watc Power Down (PD) bit törlése.*

65. táblázat. A PIC16F887 Utasításai #4

Utasítás	*Szintaxis*	*Operandusok*	*Működés*	*Érintett státusz*	
RRF	`RRF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*Lásd leírás*	*C*	*Bitforgatás jobbra egy bittel bit állapota változik. Ha a d pedig 1, akkor az f regiszterbe*
SUBLW	`SUBLW k`	$0 \leqslant k \leqslant 255$	*(k) – (W) → W*	*C, DC, Z*	*W regiszter kivonása a k é regiszterbe.*
SUBWF	`SUBWF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(f) – (W) → cél*	*C, DC, Z*	*W regiszter kivonása az f reg regiszterbe, ha pedig 1, akkor*
SWAPF	`SWAPF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*Lásd leírás*	*nincs*	*f felső 4 bitje helyet cserél az W regiszterbe, ha pedig 1, ak*
XORWF	`XORWF f, d`	$0 \leqslant f \leqslant 127$ $d \in [0,1]$	*(W) XOR (f) → cél*	*Z*	*Bitenkénti kizáró vagy művel értéke 0, akkor az eredmény kerül az eredmény.*
XORLW	`XORLW k`	$0 \leqslant k \leqslant 255$	*(W) XOR (k) → (W)*	*Z*	*Bitenkénti kizáró vagy művel után az eredmény a W regiszt*

66. táblázat: A PIC16F887 Utasításai #5

Speciális regiszterek felépítése

Status regiszter

A status regiszterbe kerülnek az ALU különböző állapotai, a Reset állapot és az adatmemória szegmens kiválasztója. A status regiszter minden utasítás célja lehet, azonban egyes bitek csak olvasható mivolta miatt az utasítások eredménye eltérő lehet.

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC(1)	C(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h-1FFh)
0 = Bank 0, 1 (00h-FFh)

bit 6-5 **RP<1:0>:** Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h-7Fh)
01 = Bank 1 (80h-FFh)
10 = Bank 2 (100h-17Fh)
11 = Bank 3 (180h-1FFh)

bit 4 **$\overline{TO}$:** Time-out bit
1 = After power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$:** Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z:** Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit Carry/$\overline{Borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)(1)
1 = A carry-out from the 4th low-order bit of the result occurred
0 = No carry-out from the 4th low-order bit of the result

bit 0 **C:** Carry/$\overline{Borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)(1)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note 1: For $\overline{Borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high-order or low-order bit of the source register.

Option Regiszter

Az Option regiszter minden utasítás által írható és olvasható. Különböző beállításokat tárol.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual PORT latch values

bit 6 **INTEDG:** Interrupt Edge Select bit
1 = Interrupt on rising edge of INT pin
0 = Interrupt on falling edge of INT pin

bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (Fosc/4)

bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA:** Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS<2:0>:** Prescaler Rate Select bits

Bit Value	Timer0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Intcon regiszter

Az Intcon regiszter a megszakítások beállításait vezérli. Minden művelet által írható és olvasható.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE(1,3)	T0IF(2)	INTF	RBIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** Timer0 Overflow Interrupt Enable bit
1 = Enables the Timer0 interrupt
0 = Disables the Timer0 interrupt

bit 4 **INTE:** INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

bit 3 **RBIE:** PORTB Change Interrupt Enable bit(1,3)
1 = Enables the PORTB change interrupt
0 = Disables the PORTB change interrupt

bit 2 **T0IF:** Timer0 Overflow Interrupt Flag bit(2)
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** INT External Interrupt Flag bit
1 = The INT external interrupt occurred (must be cleared in software)
0 = The INT external interrupt did not occur

bit 0 **RBIF:** PORTB Change Interrupt Flag bit
1 = When at least one of the PORTB general purpose I/O pins changed state (must be cleared in software)
0 = None of the PORTB general purpose I/O pins have changed state

Note 1: IOCB register must also be enabled.
2: T0IF bit is set when Timer0 rolls over. Timer0 is unchanged on Reset and should be initialized before clearing T0IF bit.
3: Includes ULPWU interrupt.

PIE1 regiszter

A PIE1 regiszter a perifériális megszakítások engedélyezését végzi párban a PIE2 regiszterrel.

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6 **ADIE:** A/D Converter (ADC) Interrupt Enable bit
1 = Enables the ADC interrupt
0 = Disables the ADC interrupt

bit 5 **RCIE:** EUSART Receive Interrupt Enable bit
1 = Enables the EUSART receive interrupt
0 = Disables the EUSART receive interrupt

bit 4 **TXIE:** EUSART Transmit Interrupt Enable bit
1 = Enables the EUSART transmit interrupt
0 = Disables the EUSART transmit interrupt

bit 3 **SSPIE:** Master Synchronous Serial Port (MSSP) Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE:** Timer2 to PR2 Match Interrupt Enable bit
1 = Enables the Timer2 to PR2 match interrupt
0 = Disables the Timer2 to PR2 match interrupt

bit 0 **TMR1IE:** Timer1 Overflow Interrupt Enable bit
1 = Enables the Timer1 overflow interrupt
0 = Disables the Timer1 overflow interrupt

120. ábra. A PIE1 regiszter felépítése

PIE2 regiszter

A PIE2 regiszter a perifériális megszakítások engedélyezését végzi párban a PIE1 regiszterrel.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	—	CCP2IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **OSFIE:** Oscillator Fail Interrupt Enable bit
1 = Enables oscillator fail interrupt
0 = Disables oscillator fail interrupt

bit 6 **C2IE:** Comparator C2 Interrupt Enable bit
1 = Enables Comparator C2 interrupt
0 = Disables Comparator C2 interrupt

bit 5 **C1IE:** Comparator C1 Interrupt Enable bit
1 = Enables Comparator C1 interrupt
0 = Disables Comparator C1 interrupt

bit 4 **EEIE:** EEPROM Write Operation Interrupt Enable bit
1 = Enables EEPROM write operation interrupt
0 = Disables EEPROM write operation interrupt

bit 3 **BCLIE:** Bus Collision Interrupt Enable bit
1 = Enables Bus Collision interrupt
0 = Disables Bus Collision interrupt

bit 2 **ULPWUIE:** Ultra Low-Power Wake-up Interrupt Enable bit
1 = Enables Ultra Low-Power Wake-up interrupt
0 = Disables Ultra Low-Power Wake-up interrupt

bit 1 **Unimplemented:** Read as '0'

bit 0 **CCP2IE:** CCP2 Interrupt Enable bit
1 = Enables CCP2 interrupt
0 = Disables CCP2 interrupt

PIR1 regiszter

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = A/D conversion complete (must be cleared in software)
0 = A/D conversion has not completed or has not been started

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer is full (cleared by reading RCREG)
0 = The EUSART receive buffer is not full

bit 4 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer is empty (cleared by writing to TXREG)
0 = The EUSART transmit buffer is full

bit 3 **SSPIF**: Master Synchronous Serial Port (MSSP) Interrupt Flag bit
1 = The MSSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
SPI
A transmission/reception has taken place
I^2C Slave/Master
A transmission/reception has taken place
I^2C Master
The initiated Start condition was completed by the MSSP module
The initiated Stop condition was completed by the MSSP module
The initiated restart condition was completed by the MSSP module
The initiated Acknowledge condition was completed by the MSSP module
A Start condition occurred while the MSSP module was idle (Multi-master system)
A Stop condition occurred while the MSSP module was idle (Multi-master system)
0 = No MSSP interrupt condition has occurred

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 **TMR2IF**: Timer2 to PR2 Interrupt Flag bit
1 = A Timer2 to PR2 match occurred (must be cleared in software)
0 = No Timer2 to PR2 match occurred

bit 0 **TMR1IF**: Timer1 Overflow Interrupt Flag bit
1 = The TMR1 register overflowed (must be cleared in software)
0 = The TMR1 register did not overflow

PIR2 regiszter

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	—	CCP2IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **OSFIF:** Oscillator Fail Interrupt Flag bit
1 = System oscillator failed, clock input has changed to INTOSC (must be cleared in software)
0 = System clock operating

bit 6 **C2IF:** Comparator C2 Interrupt Flag bit
1 = Comparator output (C2OUT bit) has changed (must be cleared in software)
0 = Comparator output (C2OUT bit) has not changed

bit 5 **C1IF:** Comparator C1 Interrupt Flag bit
1 = Comparator output (C1OUT bit) has changed (must be cleared in software)
0 = Comparator output (C1OUT bit) has not changed

bit 4 **EEIF:** EE Write Operation Interrupt Flag bit
1 = Write operation completed (must be cleared in software)
0 = Write operation has not completed or has not started

bit 3 **BCLIF:** Bus Collision Interrupt Flag bit
1 = A bus collision has occurred in the MSSP when configured for I^2C Master mode
0 = No bus collision has occurred

bit 2 **ULPWUIF:** Ultra Low-Power Wake-up Interrupt Flag bit
1 = Wake-up condition has occurred (must be cleared in software)
0 = No Wake-up condition has occurred

bit 1 **Unimplemented:** Read as '0'

bit 0 **CCP2IF:** CCP2 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

PCON regiszter

A PCON regiszter a mikrovezérlő áramellátással kapcsolatos szolgáltatásait vezérli.

U-0	U-0	R/W-0	R/W-1	U-0	U-0	R/W-0	R/W-x
—	—	ULPWUE	SBOREN(1)	—	—	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **ULPWUE:** Ultra Low-Power Wake-up Enable bit
1 = Ultra Low-Power Wake-up enabled
0 = Ultra Low-Power Wake-up disabled

bit 4 **SBOREN:** Software BOR Enable bit(1)
1 = BOR enabled
0 = BOR disabled

bit 3-2 **Unimplemented:** Read as '0'

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit
1 = No Power-on Reset occurred
0 = A Power-on Reset occurred (must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
1 = No Brown-out Reset occurred
0 = A Brown-out Reset occurred (must be set in software after a Brown-out Reset occurs)

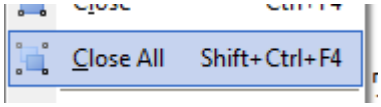
Note 1: BOREN<1:0> = 01 in the Configuration Word Register 1 for this bit to control the $\overline{\text{BOR}}$.

12. Programozás MikroC környezettel

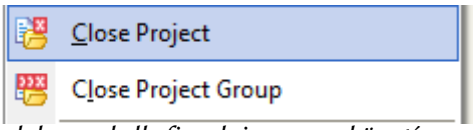
A további ismereteket projekt formátumban teszem közzé. Minden projekt mellett található egy kapcsolási rajz (vagy minimum egy bekötési útmutató) és természetesen a működéshez szükséges kód C nyelven megírva. Ezen projektek teljes forrással együtt megtalálhatóak a lemezmellékleten is.

Új projekt létrehozása

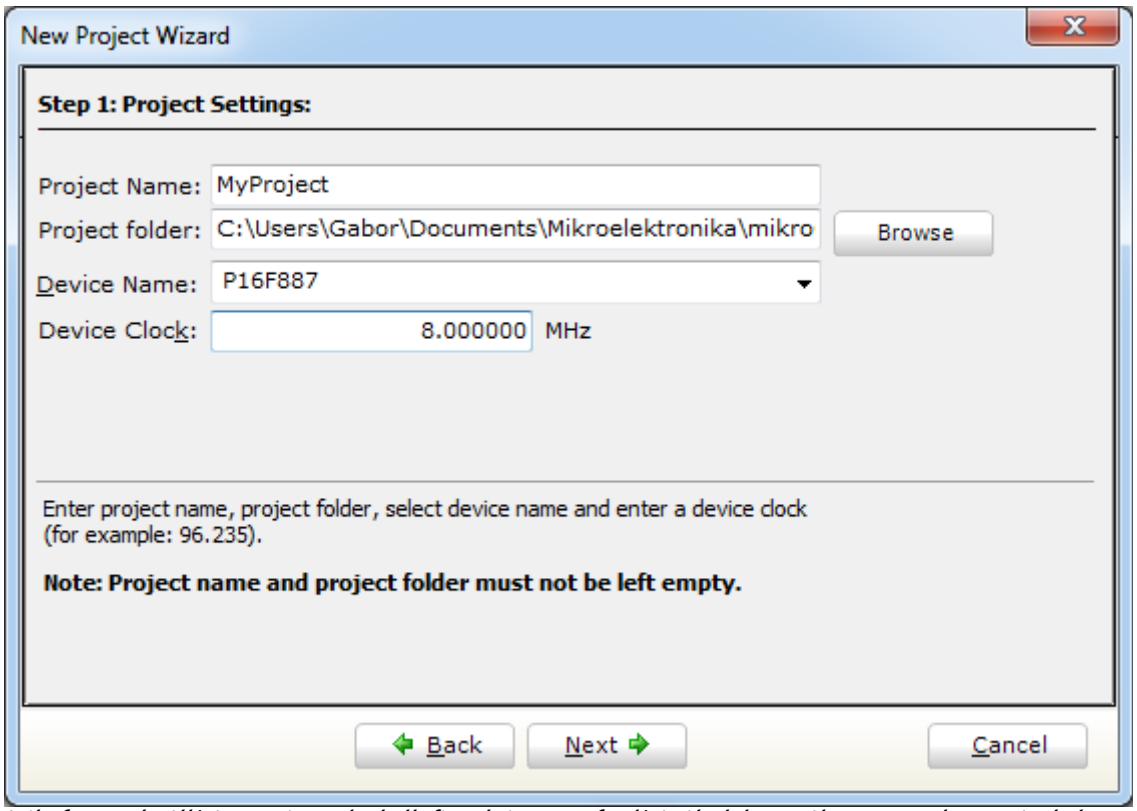
A MikroC, mint minden fejlesztő eszköz, projektekben dolgozik. Projekt alatt egy konkrét alkalmazás megvalósításához szükséges kódok és adatok értendőek. A fejlesztőkörnyezet minden egyes megnyitás alkalmával betölti az utolsó alkalommal használt projektet. Ezért új projekt kezdésekor be kell zárni ezen fájlokat. Itt megjegyzem, nem elég az éppen szerkesztett fájlt bezárni, be kell zárni az egész projektet és a nyitott fájlokat. A nyitott fájlok a File menü Close All parancsával zárhatóak be, míg a nyitott projektek a Project menü Close Project és Close Project Group menüpontjaival zárhatóak be.



Ezután létre kell hoznunk egy új projektet. Ez szintén a File menüből intézhető. Itt a New menüpont alatt megnyíló New Project menüpontot kell választani. Ennek hatására elindul az új projekt varázsló.



A projekt beállításánál két dologra kell figyelni: az eszköz típusa, valamint a használt oszcillátor frekvenciája. Az utóbbi azért fontos, mivel a MikroC a szükséges késleltetéseket, amiket használni szeretnénk, ebből számolja ki. Így, ha rossz a megadott érték, a programunk sem fog az elvárásainknak megfelelően működni.



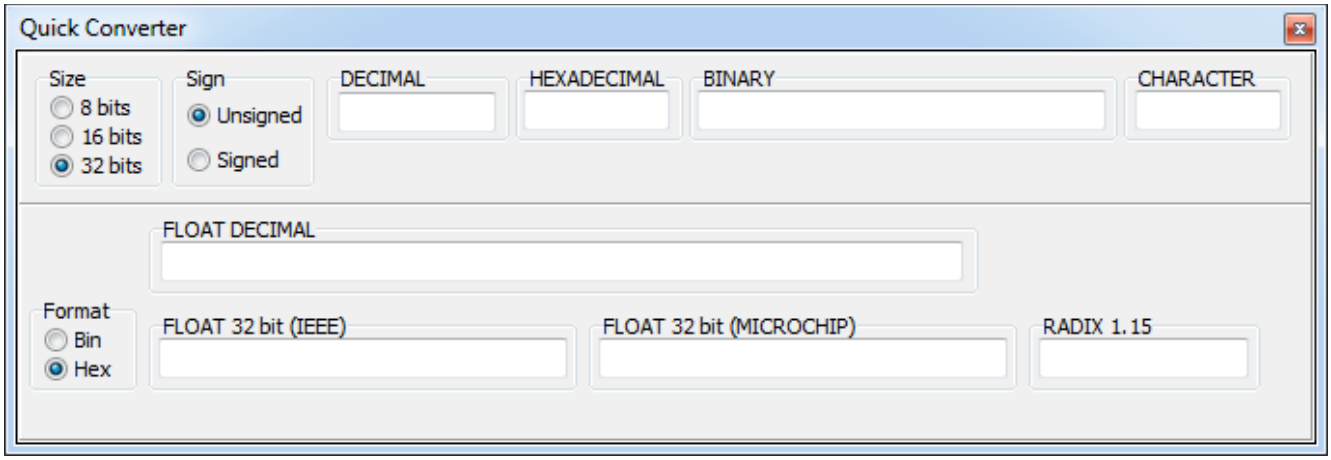
Egy másik fontos beállítás, amire oda kell figyelni, az a fordító által használt programkönyvtárak használatba

vétele. Alapértelmezetten minden könyvtárat használatba vesz a fordító, így egy csomó beépített függvényt tudunk használni. A lefordított kód mérete csak akkor fog ténylegesen nőni, ha ezen könyvtárakból használtunk is valamit. Ellenkező esetben nem fog. Másik opció az, hogy nem kérünk ebből és majd manuálisan beállítjuk, hogy mire lesz szükségünk.

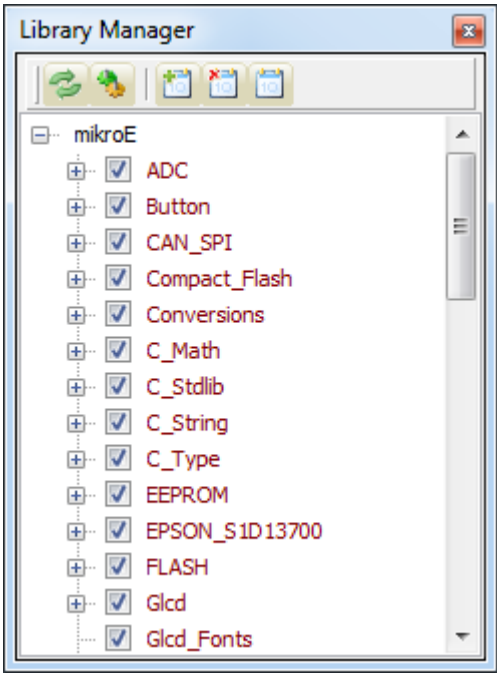
A projekt elkészülte után a kód megírása következhet. A varázsló ehhez létrehoz egy C fájlt. Fordítani a munkánkat az eszköztáron található Build gomb segítségével tudjuk, vagy menüből a Build menü Rebuild All parancs segítségével.

A MikroC fejlesztőkörnyezet

A MikroC fejlesztőkörnyezet egy csomó beépített funkcióval rendelkezik, amely megkönnyíti a mikrovezérlőkre történő programok fejlesztését. Első ilyen eszköze a beépített számátváltó, amely az alapértelmezett nézetben az ablak alján helyezkedik el. Segítségével gyorsan és könnyen tudunk különböző számrendszerek között váltogatni. Megjelenése a View menü → Quick Converter menüpontjával szabályozható.

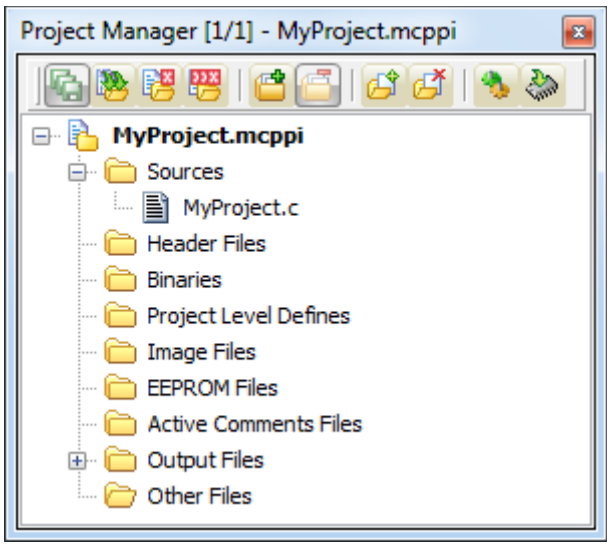


Következő fontos ablak a Library Manager, ahol a projekt által használható osztálykönyvtárak szabályozhatóak. Alapértelmezetten ez jobb oldalról nyitható ki.



Szintén jobb oldalról nyitható ki a Projekt Manager, ami segítségével a projektjeinket kezelhetjük. Ez lényegében egy fájlkezelő szerűség, mivel minden projekt több különböző fájlból állhat. Egyszerre több projekt is nyitva lehet. Ez akkor hasznos, ha egy olyan nagy projekten dolgozunk, amelyben több mikrovezérlőnek kell együttműködnie.

Az éppen aktuálisan kijelölt projekt vastag betűvel szerepel. Ez azt jelenti, hogy ő az aktív projekt, amin dolgozunk. Az eszköztáron elhelyezett és a menüből is elérhető fordítás parancs hatására ez a projekt fog lefordulni.



A fejlesztőkörnyezet számos külső eszközzel rendelkezik. Ezek a fejlesztőeszköz Tools menüjén keresztül érhetőek el. Ezen eszközök részletes használatáról a projektek kapcsán lesz szó.

A környezet nagyon fontos és jó szolgáltatása, hogy képes statisztikát generálni az egyes projektjeinkről, amiből könnyen leolvasható, hogy a programmemória hány százalékát használtuk fel, és mennyi szabad adatmemóriánk van. Ezt a statisztikát függvényenként is megkapjuk. Ez az eszköz az eszköztáron elhelyezett View Statistics gomb megnyomása hatására jelenik meg.

Tanácsos a projektünket rendszeresen menteni, mivel a MikroC hajlamos néhanapján összeomlani, különösebb ok nélkül is.

Az eszköz programozása

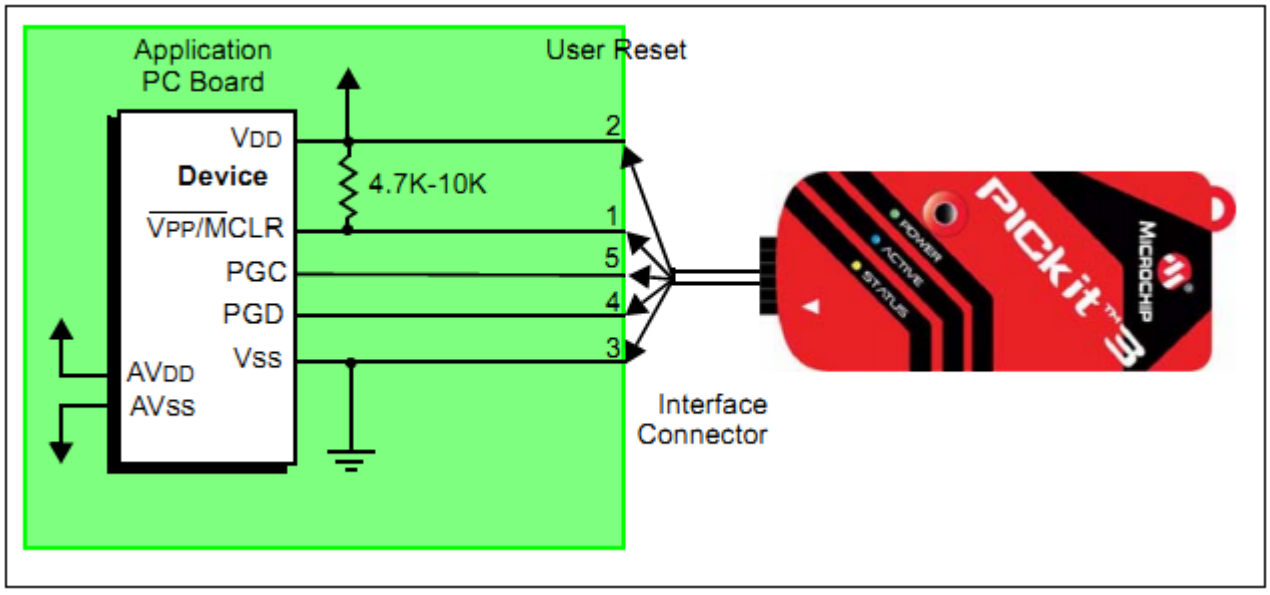
A lefordított fájlunk, amit a mikrovezérlőre rá tudunk tölteni, az egy .hex kiterjesztésű fájl lesz. Ezt opcionálisan választott programozó eszközünkkel fogjuk tudni rátölteni a mikrovezérlőre, jelen esetben ez a PICKIT3 lesz. Ehhez használható az MPLAB és az önálló programozó eszköz is.

Maga a .hex fájl, egy olyan szöveges fájl, ami lényegében leírja, hogy melyik címnél mi a memória értéke. Szokás az ilyen fájlokat dump fájlnak nevezni. Maga a dump kifejezés arra utal, hogy a memória tartalma egy az egyben a fájlban található.

A PICKit 2 és 3-as programozó egy 6 tűből álló csatlakozón kapcsolódik a programozandó áramkörhöz. Programozás folyamán nincs szükség az áramkör tápellátására, mivel ezt a programozó is szolgáltatja. A lábak bekötése a következő:

Connector Pin	Microcontroller Pin
1	$\overline{\text{MCLR}}$/VPP
2	VDD
3	Ground
4	PDG (ICSPDAT)
5	PGC (ICSPCLK)
6	LVP

Az MCLR/Vpp láb csatlakozását úgy kell kialakítani, hogy alapértelmezetten tápfeszültség kerüljön erre a lábra. Amennyiben ez a láb földelve van, az azt jelzi az eszköznek, hogy programozás fog következni, ezért nem is fog működni. Szabadon hagyni (lebegve) sem szabad, mivel ekkor előfordulhat, hogy némi statikus feszültség érdekes működést vált ki az eszközből belső CMOS felépítéséből következően. A helyes bekötést az alábbi ábra mutatja:



A programozón 3db LED található, amelyek az eszköz működése közben különböző módon villognak, ezáltal jelezve az jelenlegi működési állapotát. Ezen LED-ek jelentései:

- ***Power***

  Tápellátást jelző lámpa. Zölden világit, ha az eszköz a mini USB kábelen keresztül tápfeszültség alatt van.

- ***Active***

  Kéken világit, ha USB kapcsolat van az eszköz és a számítógép között.

- ***Status***

  Sárga, ha az eszköz dolgozik, feladatot hajt végre. Amennyiben pirosan világít, akkor a feladat végrehajtása közben hiba történt.

Az önálló programozó eszköz esetén a program nem fogja engedélyezni az eszközválasztó lenyíló menüt, amíg nincs a programozó eszköz csatlakoztatva.

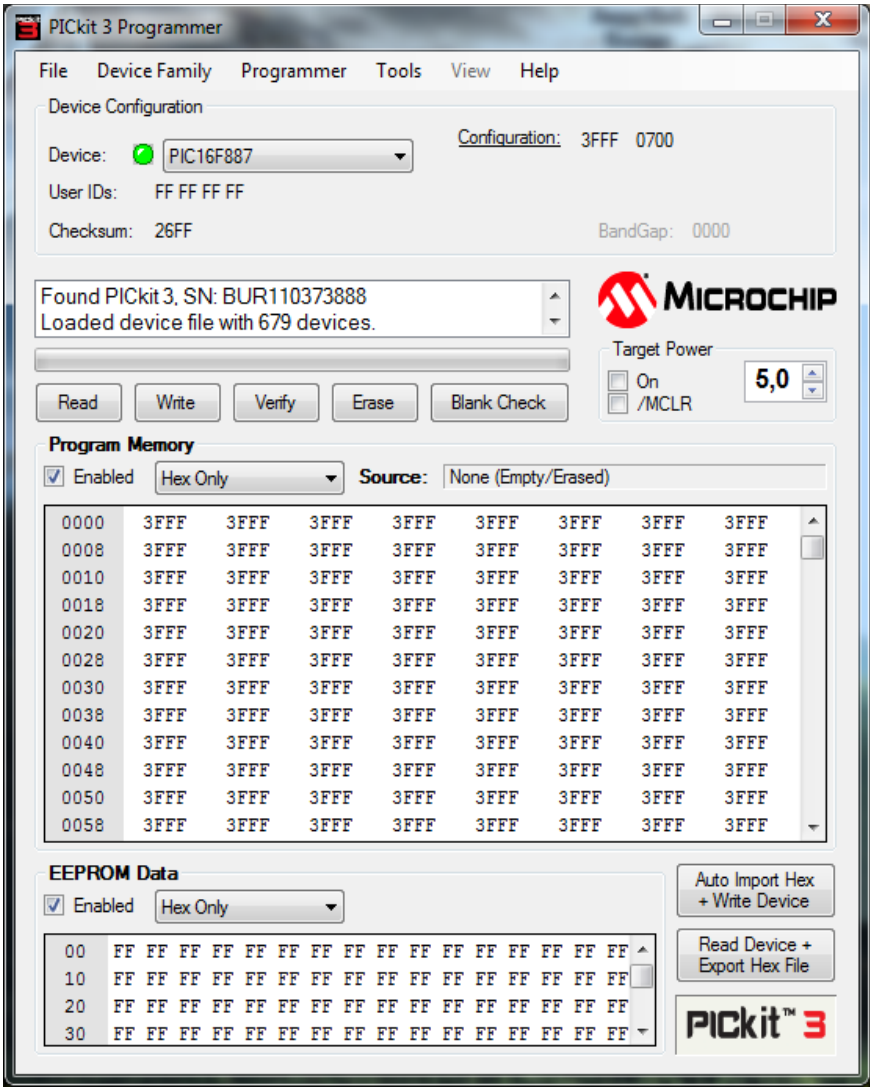
Amennyiben csatlakoztatva van, és akkor sem enged választani, akkor a programot újra kell indítani. Ha ez sem segít, akkor az eszközt húzzuk ki, majd csatlakoztassuk újra. A program újraindítása után használható lesz az eszköz. Előfordulhat, hogy nem, mert a programozón nem jó Firmware fut. Ez MPLAB segítségével javítható probléma.

Programozás mellett a program használható PIC mikrovezérlők memóriájának olvasására is. Segítségével kinyerhető az eszközön található program. Ez különösen hasznos funkció, amennyiben az embernek van egy olyan alkalmazása, amely forráskódja már nincs meg, de reprodukálni kell azt.

Programozáshoz az eszköz kiválasztása után meg kell nyitni egy .hex fájlt. Ezt a File menü Import HEX parancsával tehetjük meg. Az eszköz kiválasztására az eltérő belső programmemória és EEPROM méret miatt van szükség.

A fájlunk megnyitása utána Program Memory ablak tartalma meg fog változni. Ott, ahol nem üres a programmemória, ott 3FFF helyett valami más fog megjelenni. A 3FFF érték azt jelenti, hogy a FLASH memória 14 bites szervezésben magas szinten van. Ez a FLASH memória törölt, üres állapotát jelzi.

Ezután az eszközre a fájl írását a Write gomb megnyomásával tudjuk megtenni. A folyamat végén a memória tartalmát és épségét a Read és Verify gombok segítségével tudjuk ellenőrizni. Az Erase gomb megnyomása üríti a memóriát, míg a Blank Check gomb a memória tényleges ürességét ellenőrzi.




```
0000  282C  3FFF  3FFF  0020  0821  0023  0091  0192
0008  0020  0822  0023  0093  1315  1395  1515  080B
0010  00F0  138B  3055  0096  30AA  0096  1495  1BF0
0018  281B  138B  281C  178B  1115  1C95  2820  281D
0020  0008  0020  0821  0023  0091  0192  1395  1315
0028  1415  0813  00F0  0008  0023  018C  018D  018F
0030  0190  0022  1391  1393  0020  018D  018E  018F
0038  0021  018D  018E  018F  0020  01A0  3020  0220
0040  1803  284B  0820  3E80  00A1  0820  00A2  2003
0048  0020  0AA0  283E  3002  00A1  30AA  00A2  2003
0050  3050  0020  00A1  3055  00A2  2003  300B  00FB
0058  3026  00FC  305D  00FD  0BFD  285C  0BFC  285C
```

A MikroC beépített C kiegészítései

A MikroC tartalmaz néhány olyan függvényt, amely nem része az alap C osztálykönyvtárnak, azonban ezen függvények nagyon hasznosak tudnak lenni jó néhány esetben. Ezen függvények más, kiegészítő függvénykönnytárak használata nélkül is működnek.

Lo makró

A paraméterként kapott egész szám legalsó byte-ját adja vissza, vagy állítja be. Példa:

```
int num = 0xDABC11AF;
char lowbyte = Lo(num); //értéke 0xAF-lesz
Lo(num) = 0xBA; //ezután num értéke 0xDABC11BA lesz.
```

Hi makró

Hasonlóképpen működik a Lo makróhoz. Ez azonban a paraméterként kapott egész szám legalsó helyi értéken lévő byte melletti, magasabb helyi értéken lévő byte-ját adja vissza, vagy állítja be. Példa:

```
int num = 0xDABCABF3
char hibyte = Hi(num); //értéke 0xAB
Hi(num) = 0xFF; //ezután num értéke 0xDABCFFF3 lesz.
```

Higher makró

A Hi makró által visszaadott byte melletti magasabb helyi értéken lévő byte-ot adja vissza, vagy állítja be. Példa:

```
int num = 0xDABC11AF;
char higherwbyte = Higher(num); //értéke 0xBC-lesz
Higher(num) = 0xBA; //ezután num értéke 0xDABA11AF lesz.
```

Highest makró

A kapott egész szám legfelső byte-ját adja vissza, vagy állítja be. Példa:

```
int num = 0xDABC11AF;
char highestwbyte = Highest(num); //értéke 0xDA-lesz
Highest(num) = 0xBA; //ezután num értéke 0xBABC11AF lesz.
```

LoWord függvény

A paraméterként kapott egész szám alsó 16 bitjét adja vissza. Példa:

```
int num = 0xDABC11AF;
short lo = LoWord(num); //0x11AF értéket vesz fel
```

HiWord függvény

A paraméterként kapott egész szám felső 16 bitjét adja vissza. Példa:

```
int num = 0xDABC11AF;
short hi = HiWord(num); //0xDABC értéket vesz fel
```

Swap függvény

A paraméterként kapott byte alsó és felső 4 bites részét cseréli meg. Példa:

```
char s = 0xAD;
char swaped = Swap(s); //értéke 0xDA lesz
```

Clock_kHz függvény

Az aktuális órajelet adja vissza KHz-ben kifejezve. Amennyiben nem egész érték jönne ki, akkor kerekít a legközelebbi egész számra.

Clock_MHz függvény

Az aktuális órajelet adja vissza MHz-ben kifejezve. Amennyiben nem egész érték jönne ki, akkor kerekít a legközelebbi egész számra.

Késleltetések

A MikroC számos, beépített késleltetést kiváltó függvényt tartalmaz. Ezek közös jellemzője, hogy Delay_ előtaggal kezdődnek. Ezen függvények nem megszakítás alapon működnek, és nem is a belső időzítőket használják. A fordító a késleltetést a projektben meghatározott órajelből és abból a tényből határozza meg, hogy minden utasítás végrehajtása fix számú órajelciklus alatt történik meg. A különböző késleltetési funkciókat az alábbi táblázat foglalja össze:

`Delay_us(const unsigned long → time_in_us)` *Mikroszekundum nagyságrendű késleltetést hoz létre. Paramétere csak konstans érték lehet, változó nem.*	`Delay_ms(const unsigned long → time_in_ms)` *Milliszekundum nagyságrendű késleltetést hoz létre. Paramétere csak konstans érték lehet, változó nem.*
`Vdelay_ms(unsigned time_in_ms)` *Milliszekundum nagyságrendű késleltetést hoz létre. Paramétere változó is lehet.*	`Delay_Cyc(char → Cycles_div_by_10)` *Órajel ciklus alapú várakozást hoz létre. A várakozás 10x annyi ideig fog tartani, mint az érték amit megadtunk. Tehát 10 esetén 100 órajel ciklust fog várni. A megadott számnak oszthatónak kell lennie 10-zel.*

13. 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.

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.

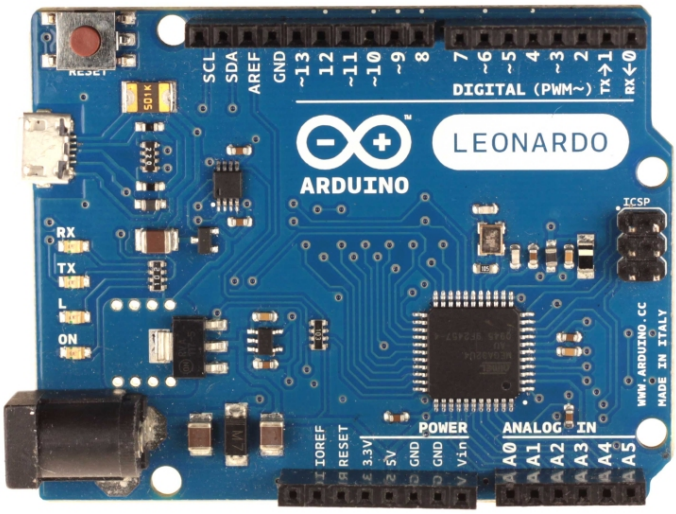
Arduino Uno / Arduino Nano

135. 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.

67. 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

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.*

68. 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

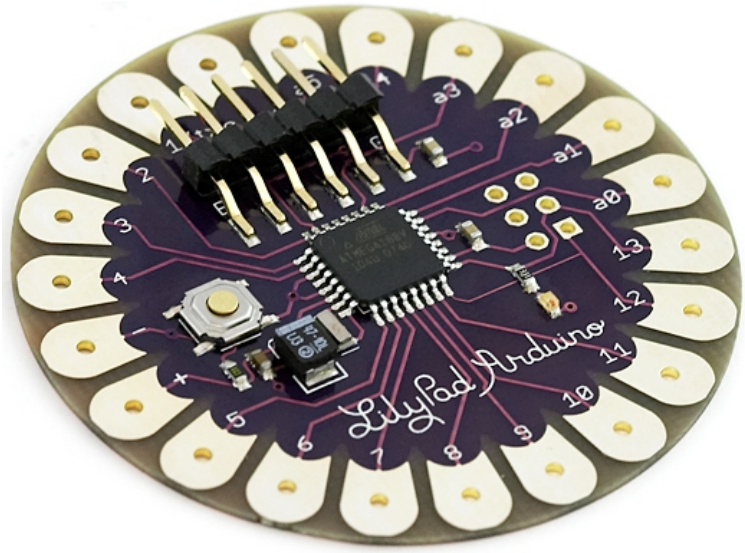
137. ábra: Arduino Mega 2560 és Mega ADK

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.

69. 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

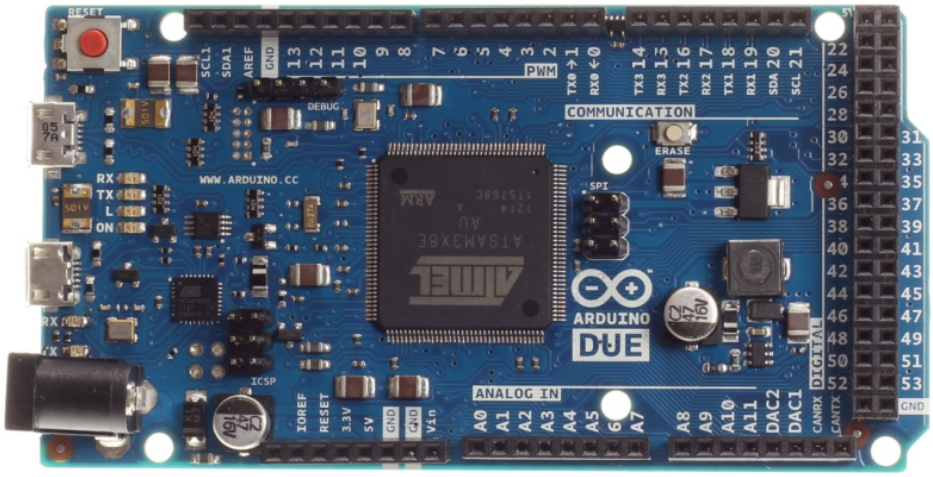


Digitális ki/bemenetek száma	*14, ebből 6db PWM képes*
Analóg bemenetek száma	*6*
Maximális áramerősség I/O lábanként	*40mA*
Programmemória / Adatmemória	*16KB, ebből 2KB foglalt / 1KB*
EEPROM memória mérete	*512b*
Használt mikrovezérlő	*ATmega128V/ATmega328V 8 MHz órajellel.*

70. 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

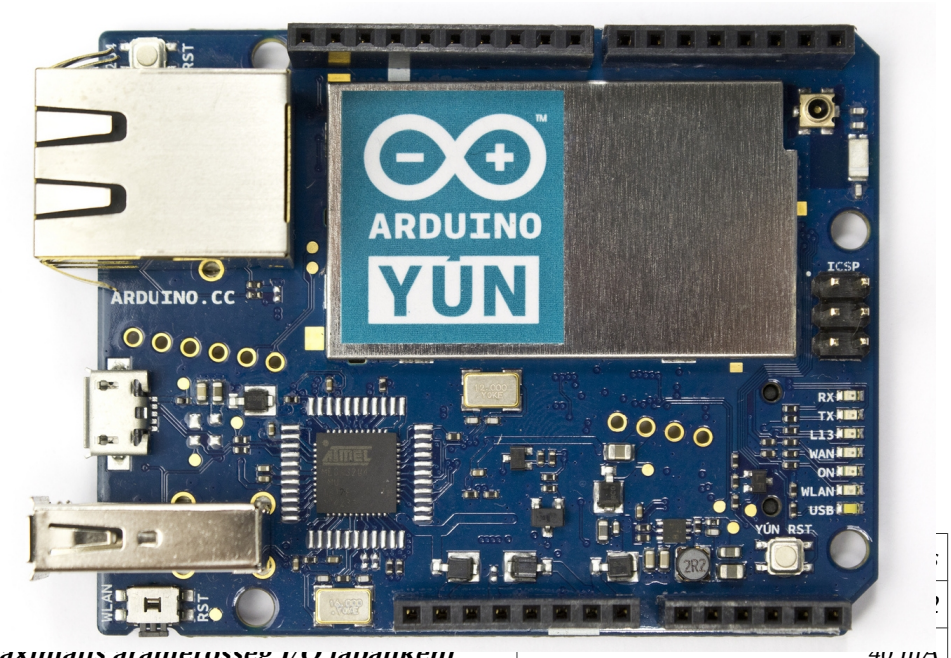


Digitális ki/bemenetek száma	*54, ebből 14db PWM képes*
Analóg bemenetek száma	*12*
Analóg kimenetek száma (DAC)	*2*
Maximális áramerősség I/O lábanként	*130mA, az összes lábbon együtt*
Programmemória / Adatmemória	*512KB / 96KB*
EEPROM memória mérete	*nincs*
Használt mikrovezérlő	*Atmel SAM3X8E 84 MHz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

71. táblázat: Arduino Due technikai paraméterei

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.

Arduino Yún

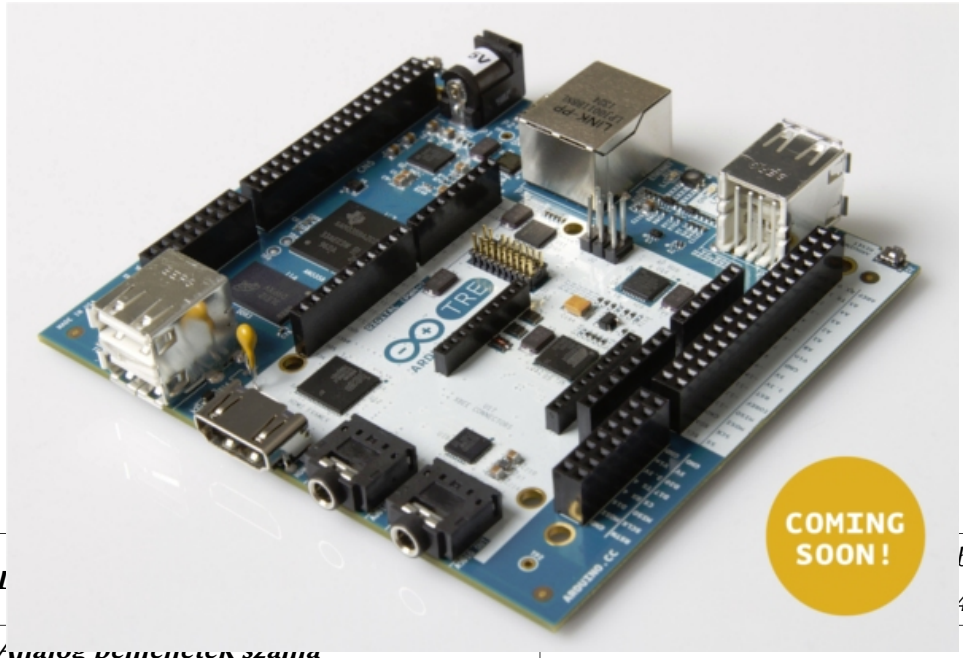


Maximális áramerősség I/O lábanként	*40 mA*
Programmemória / Adatmemória	*32 Kb, ebből 4KB foglalt/ 2,5KB, 16MbBFlash és 64MB RAM*
EEPROM memória mérete	*1Kb*
Használt mikrovezérlő	*ATmega32u4 16Mhz-en és Atheros AR9331 400Mhz-en*

72. táblázat: Arduino Yún technikai paraméterei

Az első hibrid Arduino. A rajta alkalmazott mikrovezérlő megegyezik az Arduino Leonardo mikrovezérlőjével. Emellett tartalmaz egy MIPS architektúrára épülő Atheros processzort, amely képes Linux futtatására. Emellett beépített Ethernet és WLAN támogatással érkezik az eszköz. A két processzor között a kommunikáció a soros porton keresztül van megoldva. A lap az alján rendelkezik egy MicrosSD kártyafoglalattal is.

Arduino Tre

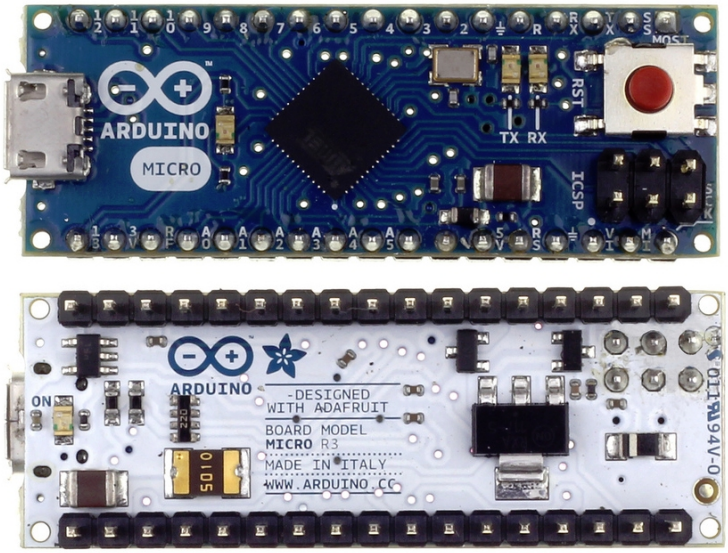


L	*b PWM képes* *4 PWM képes*
Analóg bemenetek száma	*12*
Maximális áramerősség I/O lábanként	*40 mA*
Programmemória / Adatmemória	*32 Kb, ebből 4KB foglalt/ 2,5KB,* *és 512 MB RAM*
EEPROM memória mérete	*1Kb*
Használt mikrovezérlő	*ATmega32u4 16 MHz-en és* *TI Sitara AM3359AZCZ100 1 GHz-en*

73. táblázat: Arduino Tre technikai paraméterei

Ezen modell hivatalosan csak április környékén jelenik meg. Szintén hibrid Arduino, amely a Beaglebone fejlesztői és az Arduino fejlesztők közötti együttműködés eredményeképpen született meg. A lap szintén a Leonardo modellen alkalmazott mikrovezérlőt használja, illetve a Texas Instruments által gyártott ARM Cortex A8 alpú SOC megoldást. Ezen chip 100Mbit ethernet illesztéssel rendelkezik, valamint 4db USB porttal, HDMI videokimenettel és sztereó hang be és kimenettel.

Arduino Micro



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.*

74. táblázat: Az Arduino Leonardo technikai paraméterei

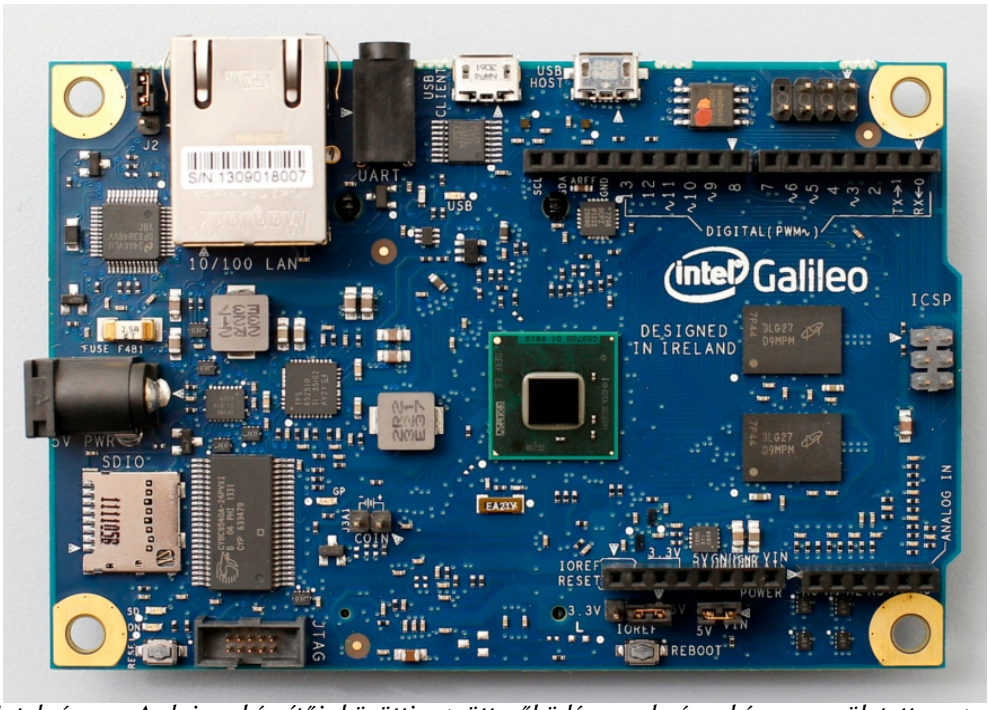
Az Arduino Micro nem más, mint egy kisebb méretű Leonardo. Tudásban, paraméterekben megegyeznek. A két eszköz között csak fizikai méretbeli különbség van. Ideális választás USB perifériák készítéséhez kis mérete miatt.

Nem hivatalos, Arduino környezet kompatibilis lapok

Az Arduino platform sikerének köszönhetően számos gyártó forgalmaz Arduino környezet kompatibilis lapokat. A kompatibilitás abban az esetben, ha nem Atmel vezérlő alpú a lap, akkor külön fejlesztőkörnyezetben valósul meg. Ezen környezetek jellemzője, hogy bár az Arduino fejlesztőkörnyezeten alapulnak mégsem teljesen kompatibilisek az eredeti környezettel. Ez leginkább a külső könyvtárakra vonatkozik, mivel ezek többsége Atmel vezérlő specifikus. Azonban a beépített gyári függvények használhatóak és működnek.

Az évek során számos lap készült amely Arduino kompatibilis. Ezen fejezetben csak egy pár népszerűbb lapról lesz szó.

Intel Galileo



Az Intel és az Arduino készítői közötti együttműködés eredményeképpen született meg ez a lap. A processzora egy Pentium kompatibilis alkalmazás processzor, ami 32 bites. A Lap 256Mb RAM memóriával rendelkezik, valamint integrált 8Mb Flash memóriával, amibe gyárilag telepítve van egy Linux rendszer, ami a programot futtatja. A tárterület tovább bővíthető SD kártya segítségével. A lap Ethernet és USB host és slave portokkal is rendelkezik, valamint a hátoldalán egy mini-PCIe csatlakozóval. További érdekessége a lapnak, hogy a digitális kimenetei 3,3V és 5v feszültség szint között jumper segítségével állíthatóak.

Adafruit Trinket / Gemma

144. Ábra: Adafruit Trinket / Gemma

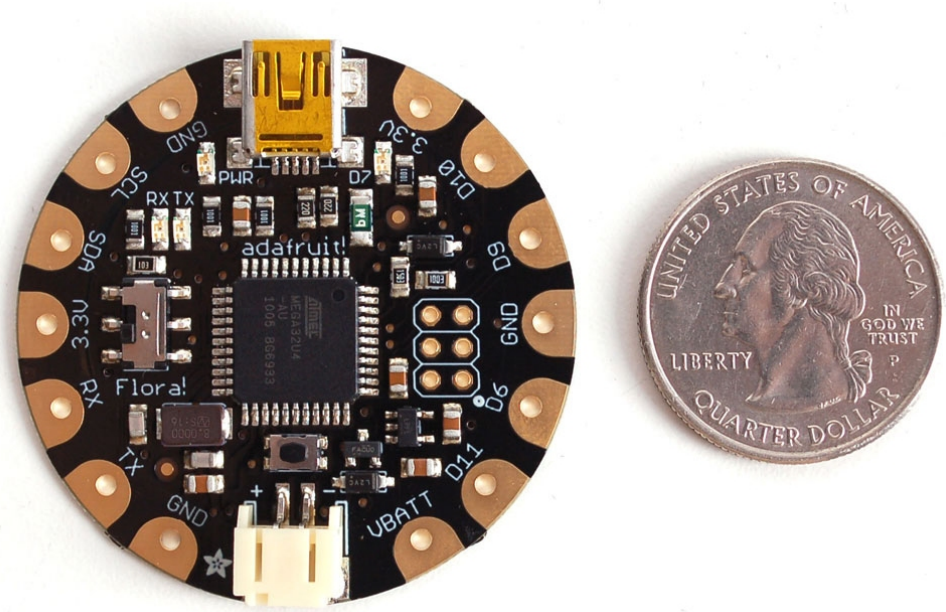
A Trinket egy igen apró mikrovezérlő lap, amely az ATTiny85 mikrovezérlőre épül. Ez a vezérlő hardveresen nem rendelkezik soros porttal és USB támogatással sem, így szoftveresen van megoldva benne az USB támogatás. Ennek koszönhető, hogy a 8Kb FLASH tárterületből nagyjából 5Kb használható csak program tárolására. Azonban így is ideális választás lehet kisméretű projektekbe, mivel elég olcsó.

5V-os és 3,3V tápfeszültségű változatban is gyártják. 5Db I/O lábbal rendelkezik, amiből kettő az USB-vel osztozik. Használatához nem kell külön fejlesztőkörnyezet, csak egy beépülő az Arduino környezetbe.

A Gemma hasonló paraméterekkel rendelkezik, mivel ugyan az a mikrovezérlő található meg rajta és szintén kis méretű. Azonban ez egy kör alakú nyomtatott áramkörre van gyártva és elsősorban viselhető elektronikai felhasználásra tervezték, mint a Lilipad Arduino-t. Azonban ennek sokkal kisebb a mérete. A Gemma csak 3,3V-os változatban kapható.

A szoftveres USB porton keresztül tud egérként és billentyűzetként is funkcionálni mindkét lap. Soros port emulációra is van lehetőség, azonban ezen funkció használata kicsivel macerásabb, mint a gyári Arduino lapok esetén.

Adafruit Flora

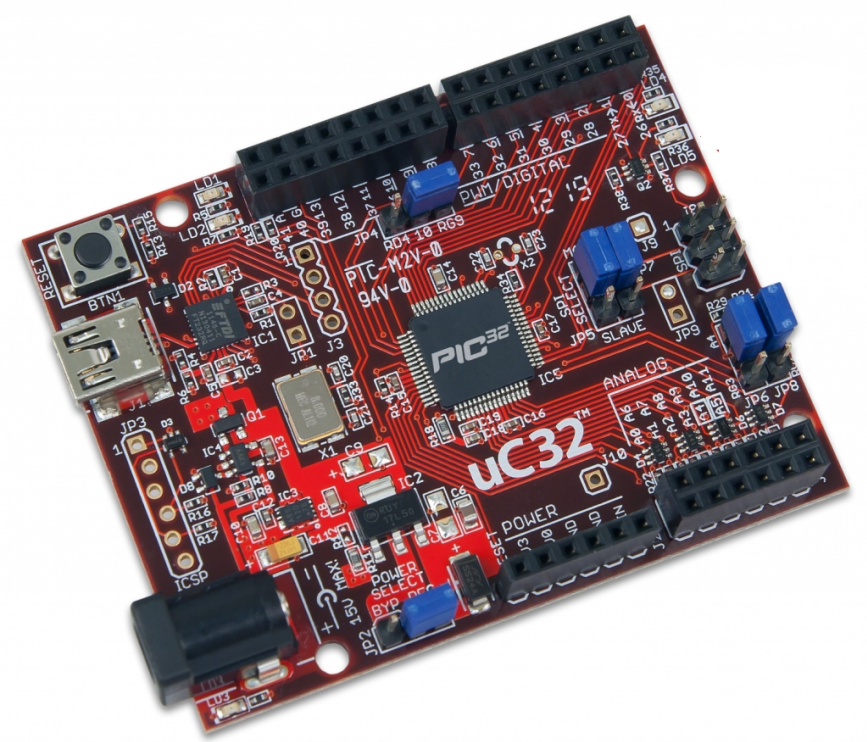


Az Adafruit Flora a Gemma nagytestvére, vagy más szemszögből nézve a Lilipad Arduino Adafruit által továbbfejlesztett változata. Technikai paraméterei megegyeznek az Arduino Leonardoval, mivel ugyan az a vezérlő van rajta.

A lap különlegessége, hogy speciálisan viselhető elektronikai projektekhez lett kialakítva, így 3,3V-ról fut, rendelkezik bekapcsoló gombbal és egy akkumulátor csatlakóval is. A tápellátás történhet bármilyen akkumulátorról, a lap stabilizátora 16V-ig képes kezelni a feszültségeket.

Az Adafruit a youtube csatornáján egy külön heti rendszerességű show műsort szentel a viselhető elektronikai projekteknek, amiben általában Adafruit Flora vagy Trinket projekteket, felhasználási ötleteket mutatnak be. A showműsor eddigi részei a https://www.youtube.com/playlist?list=PLjF7R1fz_OOU4hKjkk2Rs0_dGk_gvsazt címen található youtube lejátszási listán tekinthetőek meg.

chipKIT Uno32



A chipKIT Uno32 egy PIC32 mikrovezérlőre épülő Arduino kompatibilis mikrovezérlő platform. A használatához külön PIC kompatibilis fejlesztőkörnyezet szükséges, ami végül is az Arduino környezet PIC kompatibilis változata. A szoftver a gyártó weblapjáról szerezhető be, a http://chipkit.net/ címről.

Az Uno32 mellett további PIC alapú, de Arduino kompatibilis lapokat is forgalmaznak még. A lap processzora 32 bites és 84MHz-es órajelen fut és 128KB kódmemóriát kínál beépítetten.

Továbbá 42DB I/O lábbal rendelkezik, így alkalmas olyan esetekben az UNO leváltására, ahol fontos a sebesség és a kis méret.

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:

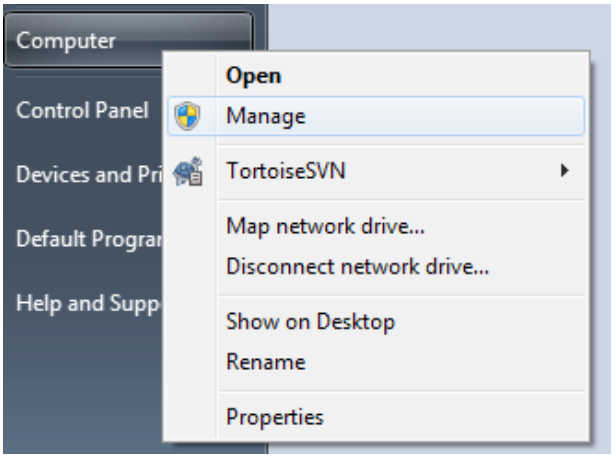
```
sudo apt-get install arduino
```

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

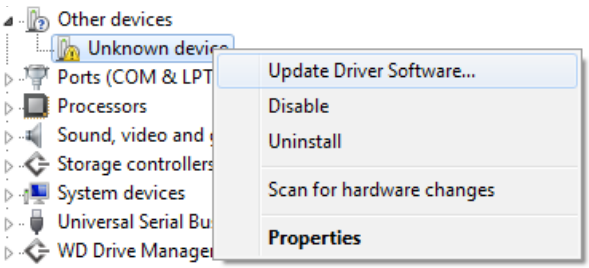
Amennyiben a letöltési vagy telepítési folyamat nem teljesen tiszta, a weblap kínál egy kezdő lépések leírást is a http://arduino.cc/en/Guide/HomePage címen.

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.



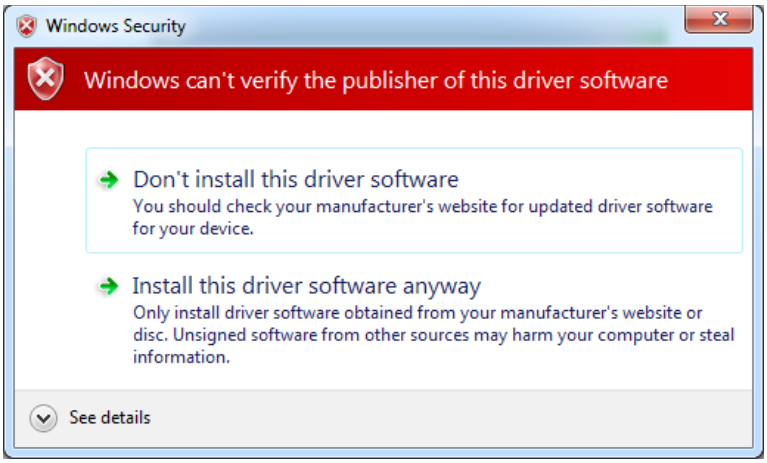
Az eszközkezelőben lesz egy kategória, amiben a fel nem ismert eszközök szerepelnek. Itt jó esetben egy eszköz található csak. Az ismeretlen eszközön kattintsunk jobb gombbal, majd válasszuk az Illesztőprogram frissítése opciót.



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.

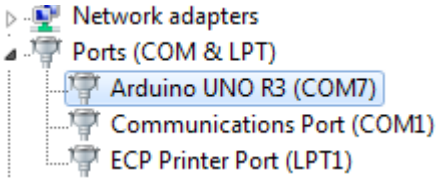
Az illesztőprogram telepítése előtt meg kell erősíteni a felbukkanó figyelmeztetésben, hogy valóban

telepíteni szeretnénk. A figyelmeztetést azért dobja fel a rendszer, mivel az illesztő nem rendelkezik digitális aláírással. Ezt most nyugodtan figyelmen kívül hagyhatjuk.



149. ábra: Illesztőprogram aláírás hiány miatti figyelmeztetés

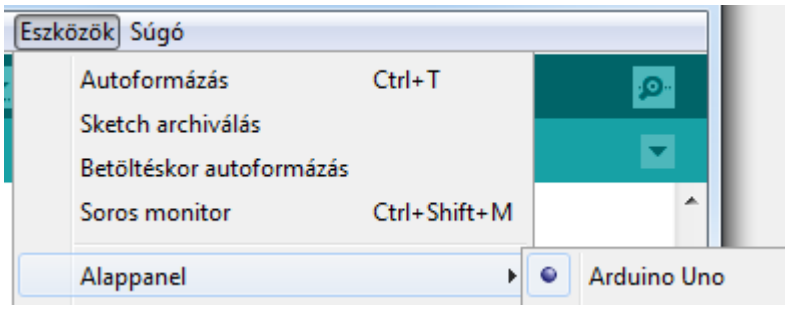
Az illesztő telepítése után az eszközkezelőben a Portok kategóriában meg kell jelennie az Arduino lapunknak. Mellette egy COM port szám lesz olvasható. Erre a portszámra szükségünk lesz a programozó beüzemelésekor.



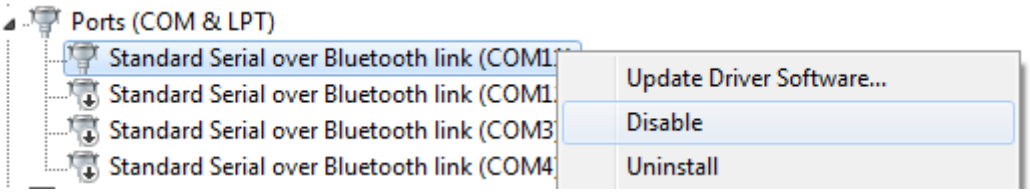
Lap tesztelése, kész programok feltöltése

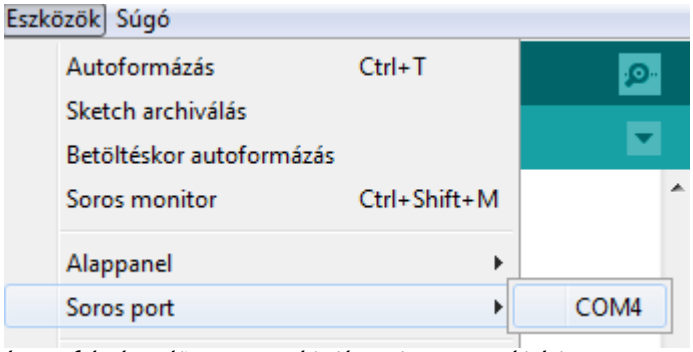
A folyamat végeztével használatba vehető a lapunk. Új, sosem használt lapok esetén érdemes leteszteltelni a működést egy egyszerű programmal. Erre a legmegfelelőbb az egyszerű LED villogtató alkalmazás. Azonban ehhez kell egy LED és egy ellenállás is. Azonban ha nincs kéznél ilyen, akkor tesztelhetjük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

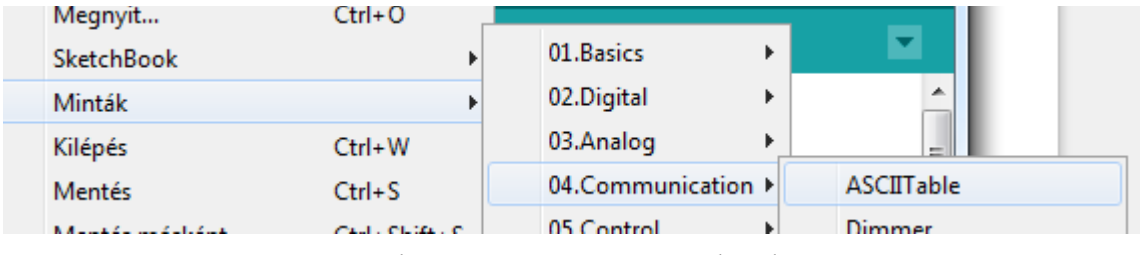


Az alappanel kiválasztása után ellenőrizzük a használt COM port számát. Ennek meg kell egyeznie az eszközkezelőben feltüntetett számmal. Amennyiben a gépen nem található integráltan soros port, akkor csak az Arduino lapot fogja látni a fejlesztőeszköz, mint ahogy az az alábbi képen is látható. Az eszközök 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.

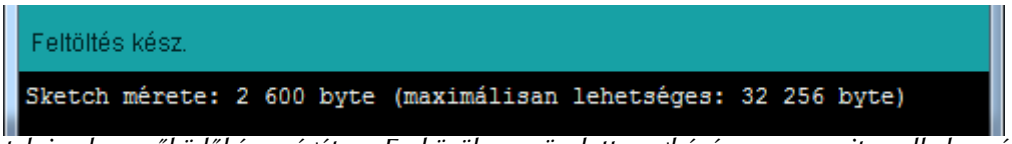




A port kiválasztása után jöhet a feltöltendő program kiválasztása. Tesztelésképpen most egy ASCII tábla programot töltünk fel az eszközre. Ez a program a beépített mintaprogramok között található meg. Megnyitni a Fájl/Minták/0.4 Communication/ASCIITable menüpont segítségével tudjuk.



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.



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.

14. Arduino programozása

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

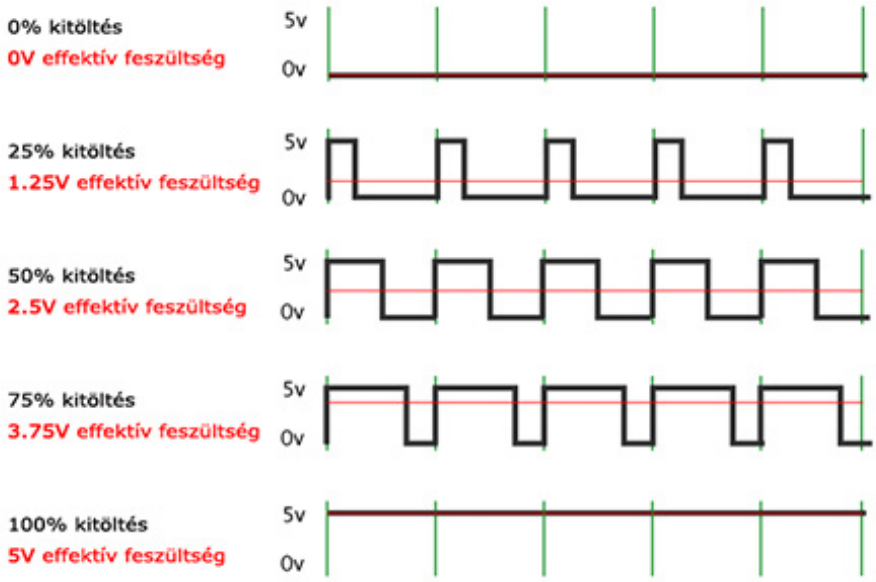
Az analóg referencia feszültség nem lehet negatív és nem haladhatja meg a lap által használt logikai egyes feszültség szintjét. Az analóg referencia feszültség változtatása a maximális érték lefelé tolására szolgál, így növeli a felbontóképességet.

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.



156. 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*

75. 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	

76. 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.

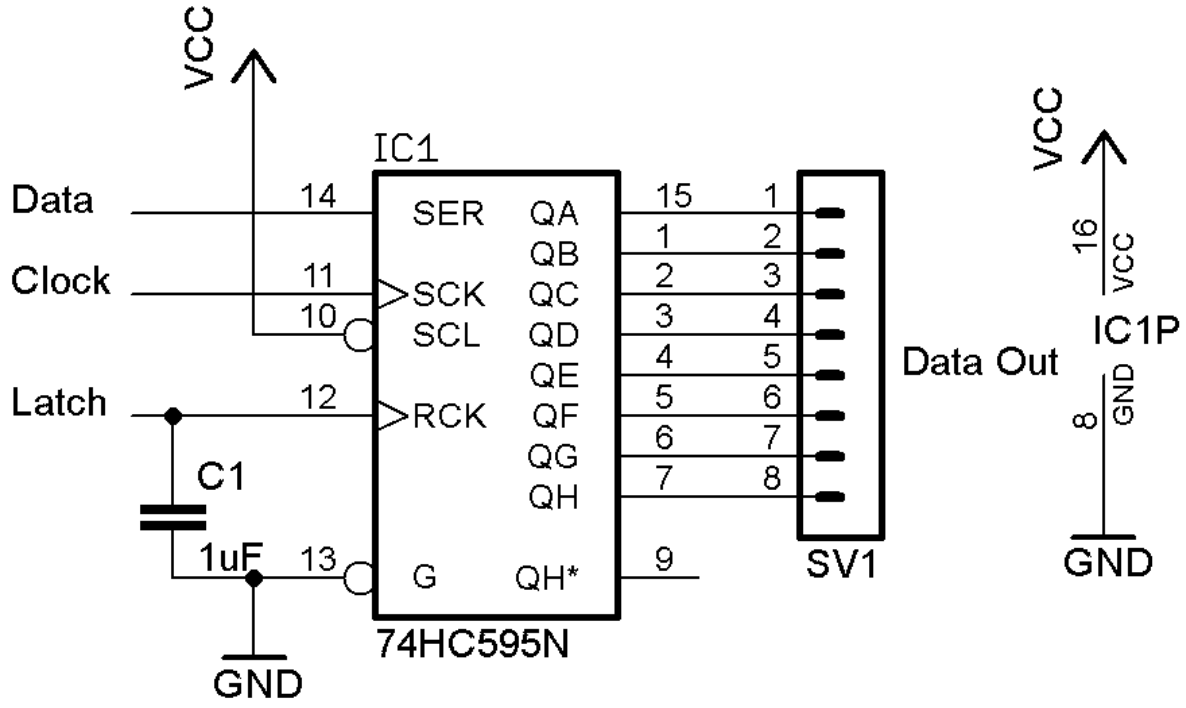
```
noTone(pin);
```

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).*

A shiftOut függvény kifejezetten 74595 számú Shift regiszter meghajtásához lett tervezve. Az alábbi ábrán a 74595 bekötése látható. A kapcsolásban a Latch láb egy tároló fokozatot valósít meg. Adat küldés előtt ezt alacsony értékre kell állítani, az adat küldés végén pedig magas szintre. Ekkor kerül át a regiszterbe írt adat ténylegesen a kimenetekre.

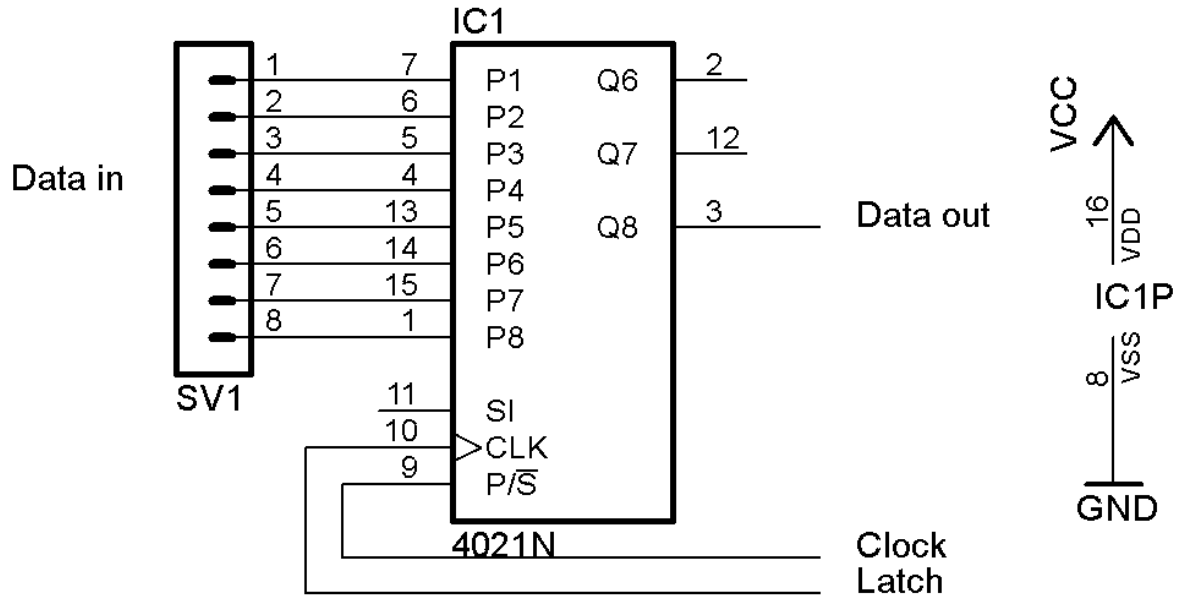



```
shiftIn(dataPin, clockPin, bitOrder);
```

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).

A függvény kifejezetten a 4021-es párhuzamos bemenetű, soros kimenetű shift regiszterhez lett fejlesztve. Az alábbi ábrán a 4021-es bekötése látható. Adat olvasás előtt a Latch lábra kell adni legalább egy 20 μs ideig

magas szinten lévő négyszögjelet. Ennek hatására a bemenetek állapotai átkerülnek a belső tárolóba, ahonnan sorosan kiolvasható az adat a shiftIn függvénnyel




```
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

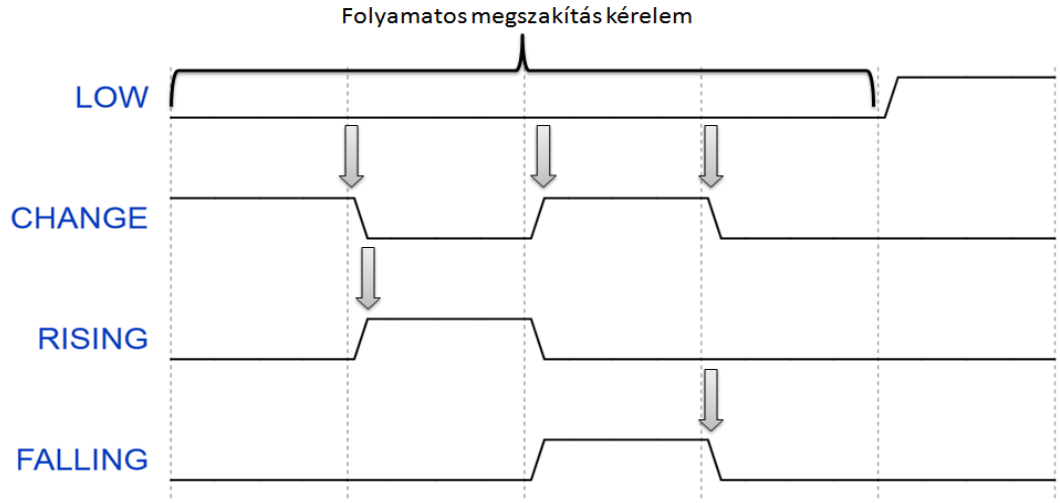
A bemenet alapú megszakítás vezérlés nem minden digitális lábon lehetséges, csak olyanok esetén, amelyek támogatják ezen funkciót. Az egyes Arduino modellek megszakítás fogadására képes lábait az alábbi táblázat foglalja össze.

	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	-	-

77. 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.

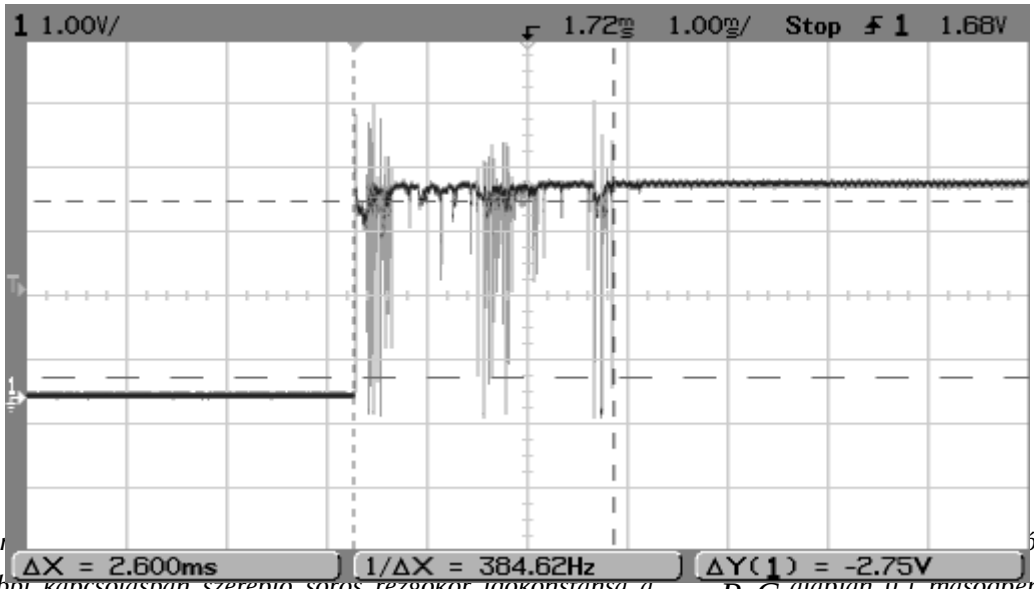


159. ábra: Megszakítási módok és kód futtatások gyakorisága

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

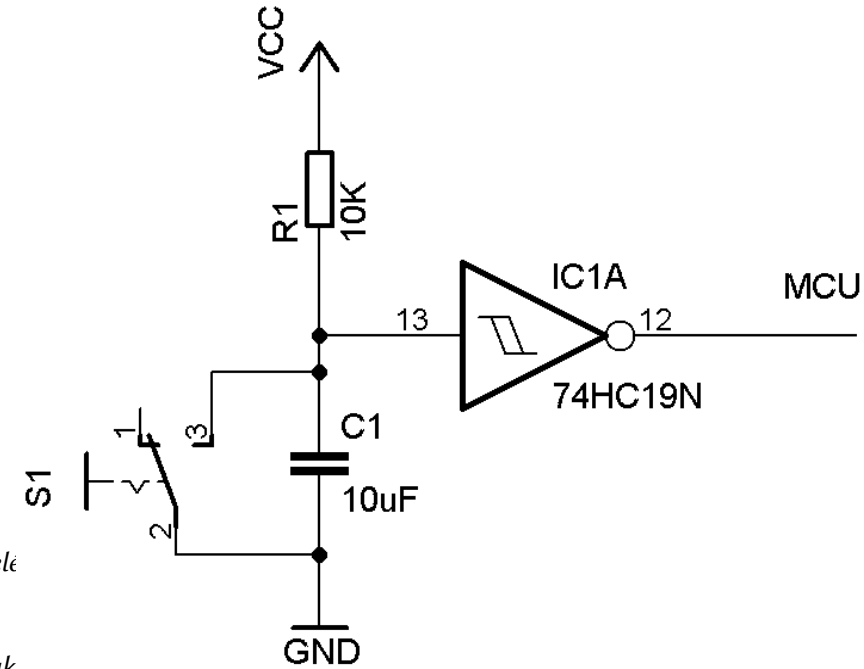
megszakítás működésképtelenné válna, mivel a késleltetések belsőleg időzítő megszakításokra támaszkodnak.

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.



Hardver

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.*



A megszakítások kezel

```
interrupts();
```

Engedélyezi a megszak *ghívni 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.

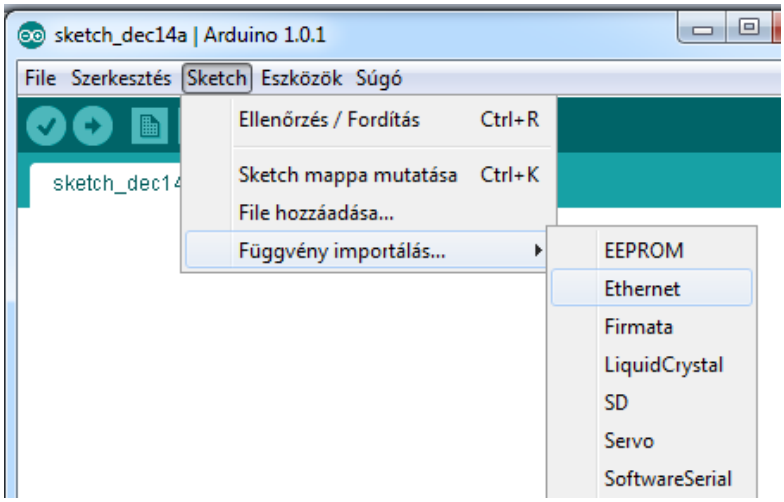
```
float(x);
```

Lebegőpontos számmá konvertálja a bemeneti x változót.

15. Arduino osztálykönyvtárak

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.



162. 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);`

78. 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 bufferből 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.

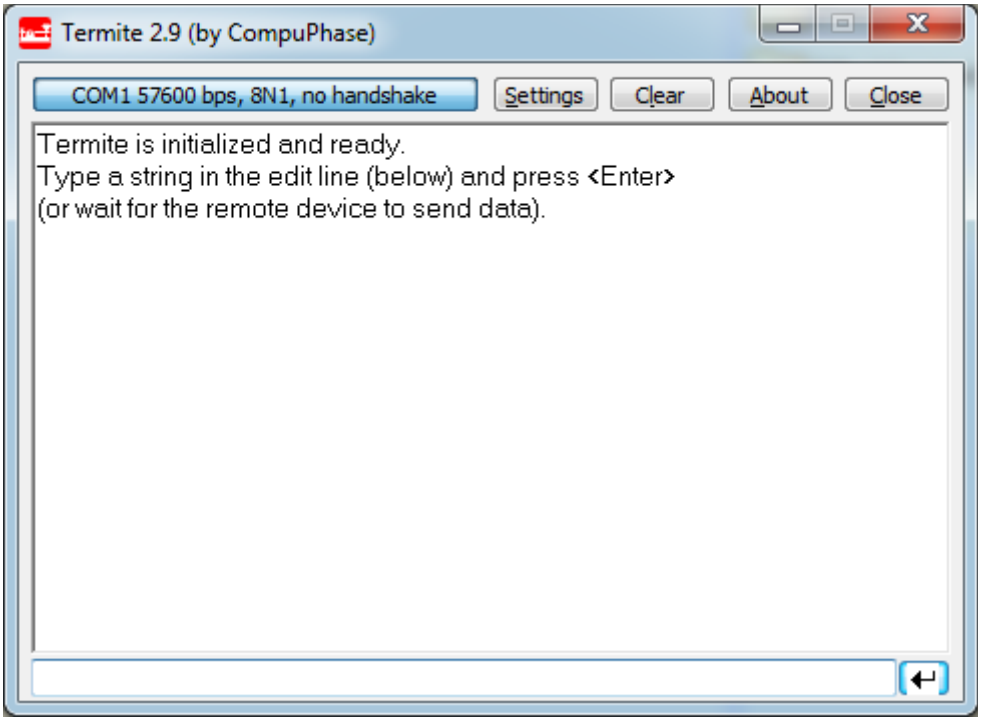
Az osztály példányosításakor a konstruktornak meg kell határozni az Rx (adatfogadás) és a Tx (adattovábbítás) lábakat.

```
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.


Termite 2.9 (by CompuPhase)
COM1 57600 bps, 8N1, no handshake
Settings
Clear
About
Close
Termite is initialized and ready.
Type a string in the edit line (below) and press <Enter>
(or wait for the remote device to send data).

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

79. 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

80. 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)

81. 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*

82. 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.

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:

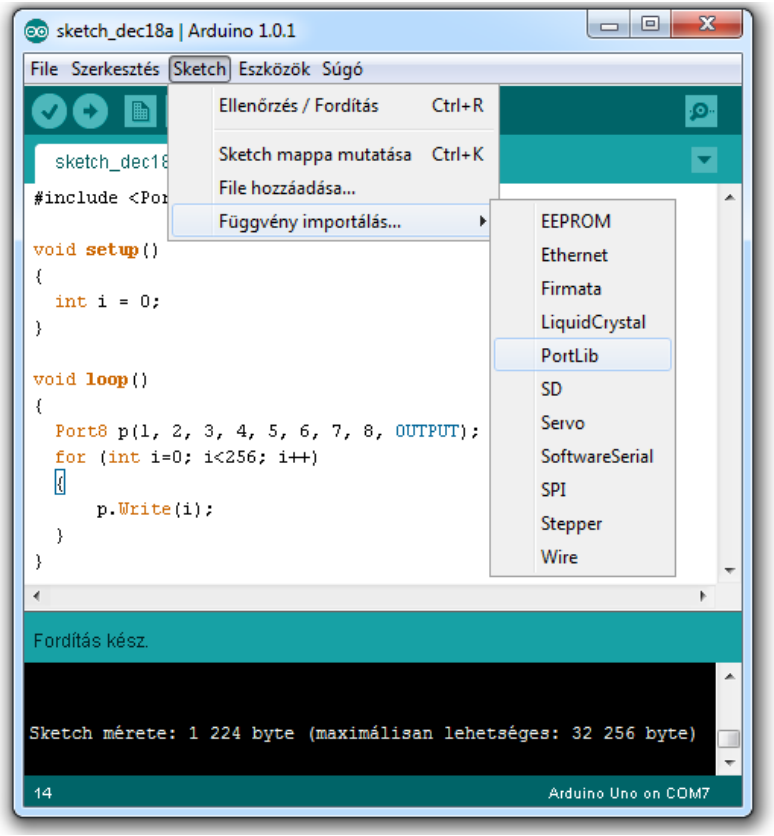
```
# Datatypes (KEYWORD1)
#######################################
Port8     KEYWORD1

# 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.



16. Kiegészítő szoftver mikrovezérlős fejlesztéshez, az 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.

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:

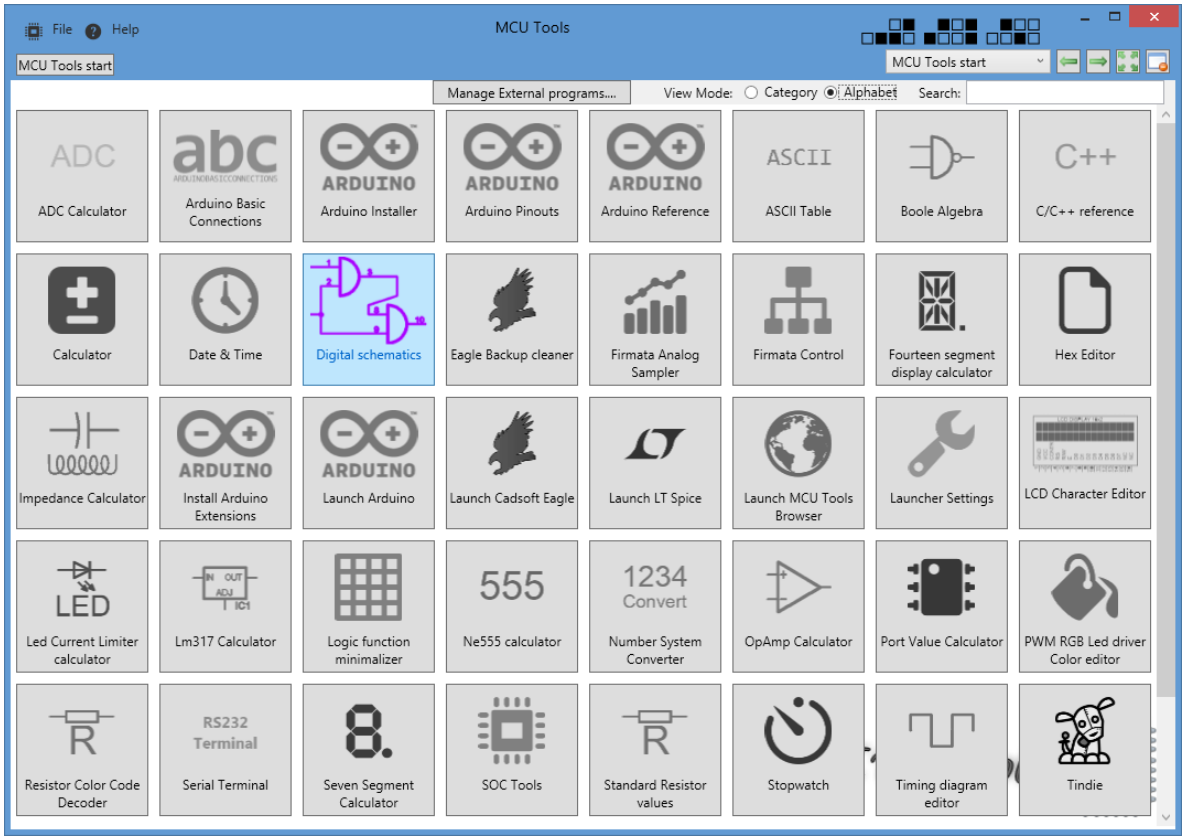
- ***Intel i szériába tartozó 2 magos processzor, AMD esetén Fx szériába tartozó processzor***
- ***DirectX 10.0-t hardveresen is támogató videovezérlő legalább 64Mb grafikus memóriával***

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/



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

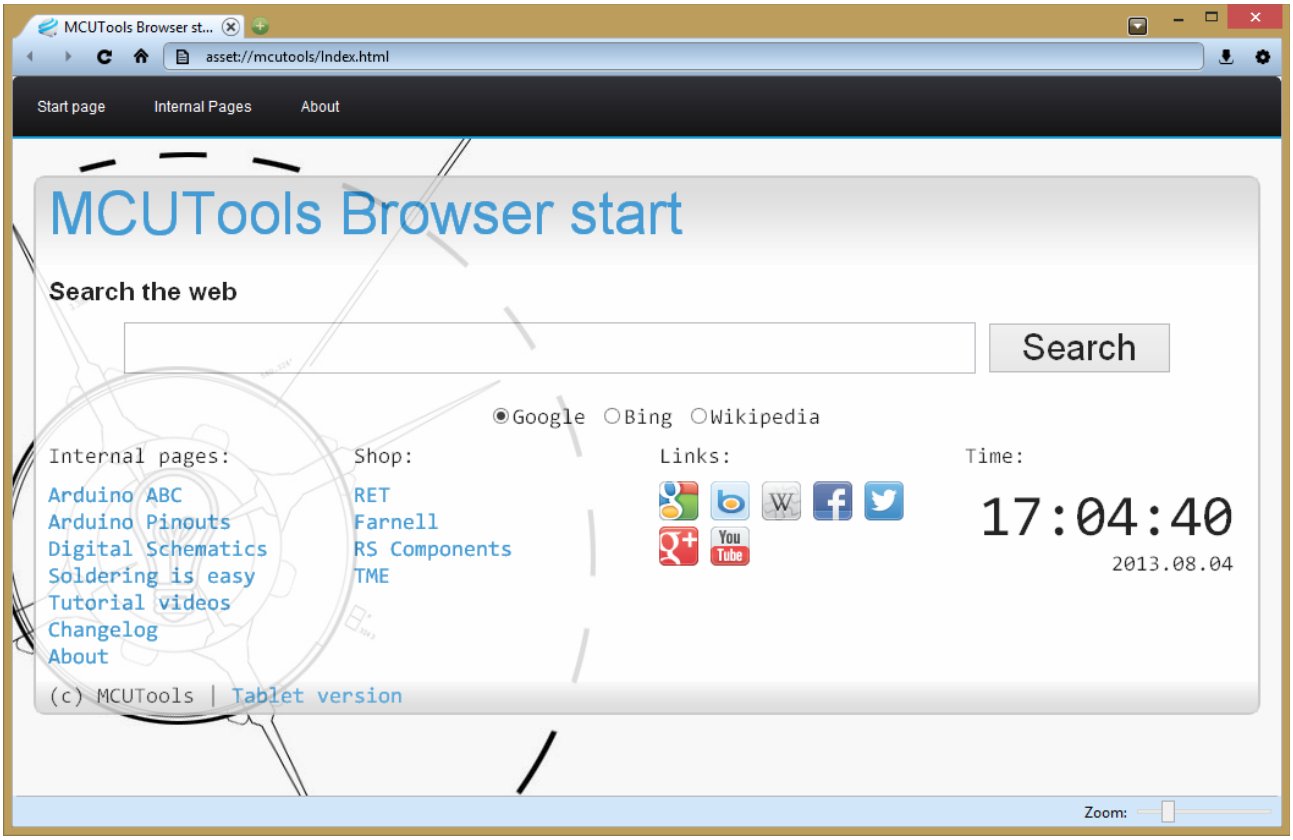
környezetben nem megtalálható mikrovezérlők és egyéb hardverek támogatásával. Professzionális fejlesztéshez mindenképpen ajánlott.

- ***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.



17. SoC eszközök és programozásuk

Az SoC angol mozaikszó a System on a Chip szavakból származik. Egy olyan integrált áramkör ez, amely egy teljes számítógépet valósít meg egy chip-en belül integrálva. Technikailag a mikrovezérlők is a SoC eszközök családjába tartoznak, azonban a Harvard architektúrás felépítés miatt külön szokták őket sorolni, mivel a legtöbb SoC megoldás a Neumann architektúrát követi memóriakezelés tekintetében.

Az SoC chip a memóriát tartalmazhatja integrálva vagy külső áramkörökből megvalósítva, illetve egy köztes megoldásként a chip-re építve. Általában az integrált megoldást vagy a chipre építést választják a gyártók, mivel a legtöbb SoC áramkör szórakoztató elektronikai eszközökbe és telefonokba kerül, ahol nem lényeges a későbbi memória bővíthetőség.

Az SoC megoldások működésükhöz komplett operációs rendszert igényelnek. A belső ROM memória, ha van, akkor funkciójában a számítógépek BIOS rendszeréhez hasonlóan csak egy nagyon minimális programot tartalmaz, ami ahhoz kell, hogy az áramkör el tudjon indítani egy fejlett operációs rendszert.

Ezen operációs rendszer fogja azt meghatározni, hogy milyen programozási nyelven és hogy tudunk programot írni az eszközre.

A legelterjedtebb SoC operációs rendszerek Linux vagy BSD alapúak. Ez a Linux kernel és BSD rendszerek ingyenességének és szabadon módosíthatóságának tudható be.

Kifejezetten barkácsolási célokra szánt SoC fejlesztő lapok nem igazán léteztek 2010 előtt, azonban az okos telefonok fejlődésével együtt fejlődtek a processzorok is, és szépen lassan megjelent az igény az olyan lapok után, amelyek a mikrovezérlőknél jóval nagyobb teljesítményt tudnak, de nem feltétlen kerülnek annyiba, mint egy komplett asztali számítógép.

Mindegyik SoC chip tartalmaz általános célú programozható ki és bemeneteket, amelyeket felhasználhatunk vezérlési célokra. Az általános célú ki és bemenetek száma függ a lap gyártójától és a lapon alkalmazott SoC chip-től is.

A ki és bemenetek általában nem 5V feszültségről működnek, hanem 3,3V-ról, valamint nem igazán terhelhetőek, mivel nem közvetlen vezérlésre tervezik őket, hanem vezérlő jelek kiadására és fogadására. Így sajnos egy LED dióda villogtatásához is hardver illesztés kell, mert könnyen tönkretehető a kimenet, rosszabb esetben a processzor is károsodhat.

A fejezet leginkább a RaspberryPi módosításáról és alkalmazásáról fog szólni, de a legtöbb alapismeret ami tárgyalva lesz az eszközöktől függetlenül alkalmazható.

Népszerű SOC architektúrák

ARM

A legtöbb SOC lap ARM architektúrára épülő processzort alkalmaz. Az ARM architektúra az 1980-as években alakult ki. Az Acorn cég tervezte az első változatokat, amelyek a kezdetektől 32 bites RISC utasításkészlettel rendelkeztek.

A megalkotott terv annyira sikeres lett, hogy egy külön céget hoztak létre, ez lett az Advanced Risc Machines, aminek a rövidítéséből jön az ARM név. Maga az ARM cég nem gyárt processzorokat, csak processzor magokat tervez, ezeket a terveket értékesíti a gyártóknak, mint a Samsung, Sony, Broadcom és még lehetne sorolni.

A processzorok két utasításkészlettel rendelkeznek. Egy 16 bites és egy 32 bites utasításkészlettel. A 16 bites utasításkészlet is 32 bites adatokkal dolgozik, de az utasítás maga csak 16 bitet tesz ki. Értelem szerűen, mivel az utasítás hossz kisebb kevesebb processzor funkció érhető el, de növekszik a kódsűrűség, még akkor is, ha egy adott probléma megoldásához több utasítás kell. A nagyobb kódsűrűség gyorsabb végrehajtást eredményez, így ez egy optimalizálási lehetőség a fordítóprogramok számára.

Az architektúra 16 általános célú regiszterrel rendelkezik. Az utasítás készlet eleve úgy lett megtervezve, hogy minden utasításnak van feltételes megfelelője, így nagy mértékben optimalizálható a feltételes kódrészletek végrehajtása.

Az utasításkészletben nincsenek utasítások lebegőpontos számok kezelésére, hasonlóan az X86 architektúrához. A lebegőpontos adatokkal való munkára kifejlesztett processzor a VFP nevet viseli, beépítése opcionális a processzorba, így léteznek olyan ARM processzorok, amelyek nem rendelkeznek vele. Ezen processzorokon is lehetséges lebegőpontos adatokkal műveleteket végezni, viszont ezt szoftverből kell megoldania a fordítónak, így a generált kód lassabb lesz.

Egy ARM mag mellé 16db társprocesszor társítható a belső társprocesszor rendszeren keresztül, ennek segítségével a gyártók igen specializált áramköröket gyárthatnak. A legtöbb SOC megoldásban társprocesszor a processzor mellé a videó vezérlő és a hang vezérlő. A VFP társprocesszor számára külön buszrendszer van a processzorban.

A multimédiás igények növekedésével az architektúra kapott egy SMID (Single Instruction Multiple Data, Egy utasítás több feldolgozott adat) utasításkészletet is, amely a NEON nevet kapta. Ennek a beépítése is szintén opcionális. Az SMID utasítások lényege, hogy hardverből egy utasítás segítségével több adatot dolgoz fel a processzor. Ennek segítségével megvalósítható hardver szinten az, hogy egy utasítással összeadjon 3D vektorokat vagy hogy egy utasítás segítségével dekódoljon MP3 tömörítést.

Az architektúrához nem csak a VFP és NEON lebegőpontos/SMID utasításkészletet fejlesztették ki, létezik számos másik hasonló szolgáltatásokkal rendelkező utasításkészlet/társprocesszor is, azonban ezek nem kód kompatibilisek a VFP és NEON utasításokkal.

A processzor magoknak lehetőségük van Neuman és Harvard memória modellt is alkalmazni, így az ARM magok nem csak SOC eszközökben találhatóak meg, hanem mikrovezérlőkben is.

x86 / x64

Az x86 architektúra az IBM PC kompatibilis leszármazott gépek architektúrája. Nevét még az Intel korai processzor számozási sémája alapján kapta (8086, 80286, 386, 486). A számozási sémát a Pentium név bevezetésével váltották le. Ennek az oka elsősorban az volt, hogy egy számsorozat nem védhető le márkanévként. Az x64 jelölés az x86 architektúra 64 bites bővítése. A 64 bites bővítés minden modern x86 processzorban megtalálható. Amellett, hogy a belső regisztereket és a megcímezhető memória méretét növeli 64 bitre további általános célú regisztereket is ad az architektúrához. X86 üzemmódban csak 8 db általános célú regiszter van a processzorban, x64 üzemmódban pedig 16db. Összehasonlításképpen az ARM architektúra 128db általános regisztert tartalmaz.

Minden x64 processzor 3db utasításkészlettel rendelkezik, mivel a processzorok visszafelé kompatibilisek a 80286 processzorral. Tehát van egy 16 bites utasításkészlet, egy 32 bites utasításkészlet és egy 64 bites utasításkészlet. A működési módok között szintén utasításokkal lehet váltani. 64 bites üzemmódból lehetőség van 32 bites működésre váltani, 32 bites üzemmódból meg lehetőség van 16 bites üzemmódra váltani.

SOC megoldásokban az Intel Atom igen elterjedt ebből az architektúrából. Ez egy alacsony fogyasztású processzor. Legnagyobb előnye, hogy utasításkészlete megegyezik a PC-n használt processzorokéval, így akár Windows-t vagy DOS rendszert is képes futtatni.

További előnye ezen processzoroknak annak ellenére, hogy CISC utasításkészlettel rendelkeznek, hogy képesek az utasításokat nem sorrendben végrehajtani. A nem sorrendi végrehajtásnak CISC utasításkészlet esetén van értelme, mivel itt változóak az utasítások hosszai. A processzor belsőleg nem egy aritmetikai egységgel rendelkezik, hanem többel, így egyszerre több utasítás végrehajtását kezdi meg nem feltétlenül sorrendben. A működéshez kell egy nagyméretű cache memória a processzoron belül, mivel egyszerre nagyjából 10-12 utasítást és a hozzá tartozó adatokat kell betöltenie. Ezek közül kiválassza a leghosszabb időt igénylő utasítást a processzor, majd az egyik ALU-n elindítja annak a végrehajtását, közben pedig a többi ALU-n a működést úgy időzíti, hogy mire végez a hosszú utasítás végrehajtásával az ALU, addigra a többi rövidebb utasítás is végre legyen hajtva.

Másik nagy előnye ennek az architektúrának a virtuális magok, a Hyper Thread-ek megjelenése. A virtuális mag valójában egy teljes értékű második processzorként fogható fel, de mégsem teljes értékű, mivel saját gyorsítótárral nem rendelkezik. Ez költség csökkentést tesz lehetővé. Előnye, hogy ha olyan szoftvert futtatunk, ami eleve rendelkezik több processzoros optimalizációval, akkor a program úgy fogja kezelni a processzort, mint ha valóban két maggal rendelkezne. Viszont ha két különböző programot futtatunk, akkor a rendszer úgy viselkedik, mint egy egy magos processzor. A Hyper Thread konfiguráció általában 2 valós mag mellé 2 virtuális párosításban szokott megjelenni.

Az architektúra velejárója, hogy egy FPU-val gyárilag rendelkezik a processzor, ami nem elhagyható komponens. Az FPU működésére több SMID utasításkészlet van építve, amik durván megnövelik a processzor teljesítményét. Korábban ezek az utasítások csak assembler nyelven voltak használhatóak, de a legtöbb mai fordító már képes SMID optimalizációkat iktatni a kódba.

Népszerű SOC lapok

RaspberryPi

A legnépszerűbb ezen lapok egyike a RaspberryPi. A hitelkártya méretű számítógép megalkotóinak célja egy olcsó, programozás oktatására és tanulására alkalmas eszköz létrehozása volt.

Az eszköz a Broadcom BCM2835 chip-re épít, ami 700Mhz-en fut, de szoftveresen akár 1Ghz sebességre is állítható. Az eszközre 512Mb RAM van integrálva, a korábbi változatok 256Mb memóriával készültek, azonban ez nem bizonyult elégnek. Az integrált videokártya teljesítménye elegendő 1080p filmek lejátszására és dekódolására, így sokan médialejátszónak alkalmazzák TV mellé.

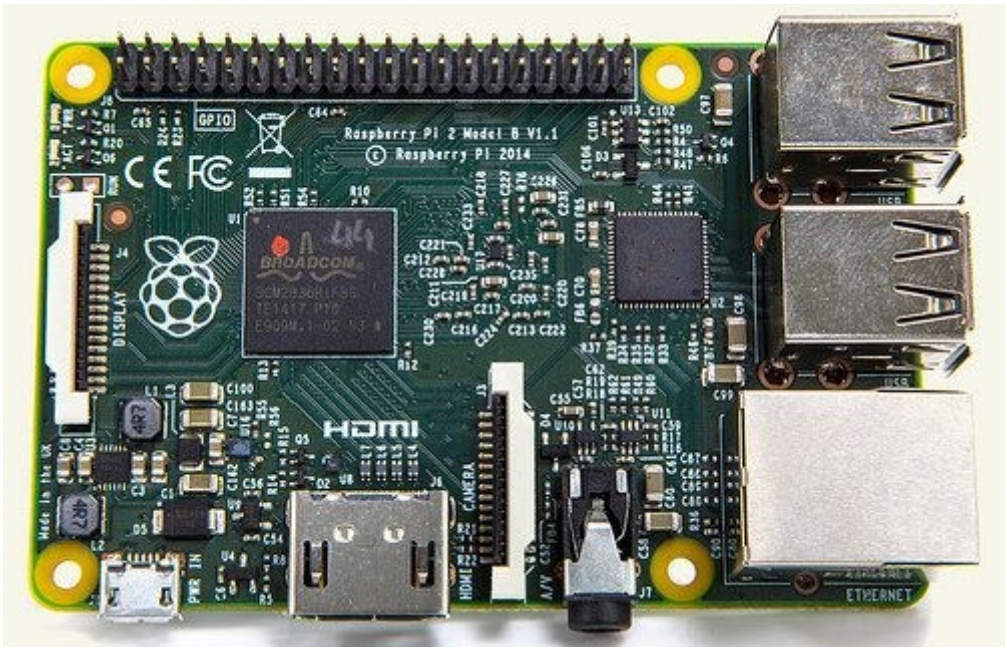
Számos disztribúció érhető el az eszközre, a készítők a Debian Linuxot preferálják, pontosabban a Raspbian-t, ami a Debian Linux RaspberryPi-re optimalizált változata.

Jelenleg három változatban kapható. Az A modellen nincs hálózati csatoló és csak egy USB porttal rendelkezik, valamint 256Mb memóriával. A B változat a legelterjedtebb. Hálózati csatolóval és 2 USB porttal rendelkezik.

Nemrég megjelent a B+ változat is, amely a B változat továbbfejlesztett változata. A B+ ugyan akkora fizikailag, mint a B modell, de kettő helyett 4 USB porttal rendelkezik és 40 tűs GPIO csatlakozója van, valamint MicroSD kártyákat fogad hagyományos SD kártyák helyett. Valamint jobb a fogyasztása is, mivel kapcsoló üzemű tápegység elektronikával rendelkezik, így nagyobb hatásfokkal tud működni akkumulátoros táplálásról is.

A B+ változat mellett létezik A+ változat is, amely az A modell továbbfejlesztett változata. Az A+ alapját a B+ változat adja, azonban ezen lap jóval kisebb méretű, továbbra sem rendelkezik hálózati csatolóval és még mindig egy USB porttal rendelkezik.

A Pi2 modell a B+ modellel teljesen kompatibilis (méretben, csatlakozók kiosztásában és számába), viszont ez a modell egy négy magos ARM Cortex-A7 alapú processzorral rendelkezik, amely alap órajele 900MHz. A Pi és a Pi2 esetén a videokártya ugyan az. További újdonság, hogy a Pi 2 változat 1GB memóriával rendelkezik.


Raspberry Pi 2 Model B V1.1
© Raspberry Pi 2014
GPIO
DISPLAY
HDMI
CAMERA
ETHERNET

BananaPi

A tervező csapat nem titkolt célja az volt, hogy létrehozzanak egy izmosabb Raspberry kompatibilis lapot. A kompatibilitás nem 100%-os, mivel a Banana változat picivel nagyobb méretű, mint a Raspberry. Valamint a csatlakozók se pontosan ott vannak, mint a Pi-n. Ebből adódóan néhány Raspberry-hez készített kiegészítő nem fog működni a Banana-val, mivel egyszerűen nem fér rá a lapra.

Viszont, ha gond nélkül csatlakozik a kiegészítő lap, akkor működni is fog, mert a lap GPIO lábkiosztása és szoftvere azonos a Raspberry változattal. A Banana is Raspbian operációs rendszert futtat. De ha ez nem tetszik, akkor választhatunk Fedora vagy akár Android lemezképet is.

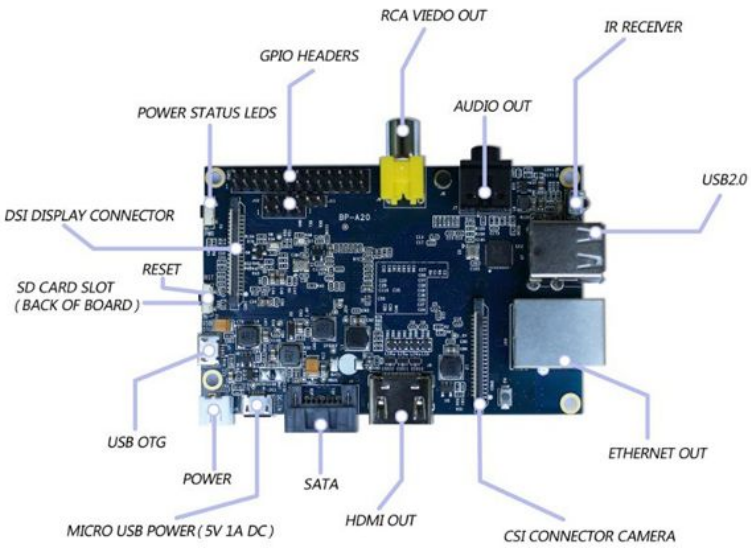
A hardver elég "izmos" ezen kategóriához képest. A fejlesztők igyekeztek a Pi hiányosságait pótolni. A processzor két magos és 1GB memória áll rendelkezésre, ami manapság megszokott konfigurációnak számít egy telefon esetén is.

Ezen felül van Gigabites Ethernet vezérlő és SATA csatlakoztatási lehetőség is, így közvetlenül csatlakoztathatunk a gépre egy 2,5" merevlemezt/SSD meghajtót egy speciális kábel segítségével, amely nem csak az adatot, hanem a feszültséget is továbbítja a meghajtó felé.
3,5" merevlemezek / DVD meghajtók is csatlakoztathatóak, de ezek 12V tápfeszültségről működnek, így itt a tápellátás megoldása a felhasználóra van bízva.

A lap tápellátása sokkal stabilabb, mint a Raspberry esetén. A Raspberry sok esetben körülbelül használhatatlan egy extra táplálással rendelkező USB hub nélkül. A Banana esetén nem találkoztam olyan problémával, hogy USB csatlakoztatás után újrainduljon a rendszer a feszültségesés miatt. Viszont szintén csak 2 USB port van a lapon, így a több eszközhöz kelleni fog a Hub itt is.

További extra a bekapcsoló gomb és az infra vevő. Az infra vevő HTPC építési célokra jöhet jól, de egy nagy problémája a lapnak, hogy elég hülye helyre került a vevő.

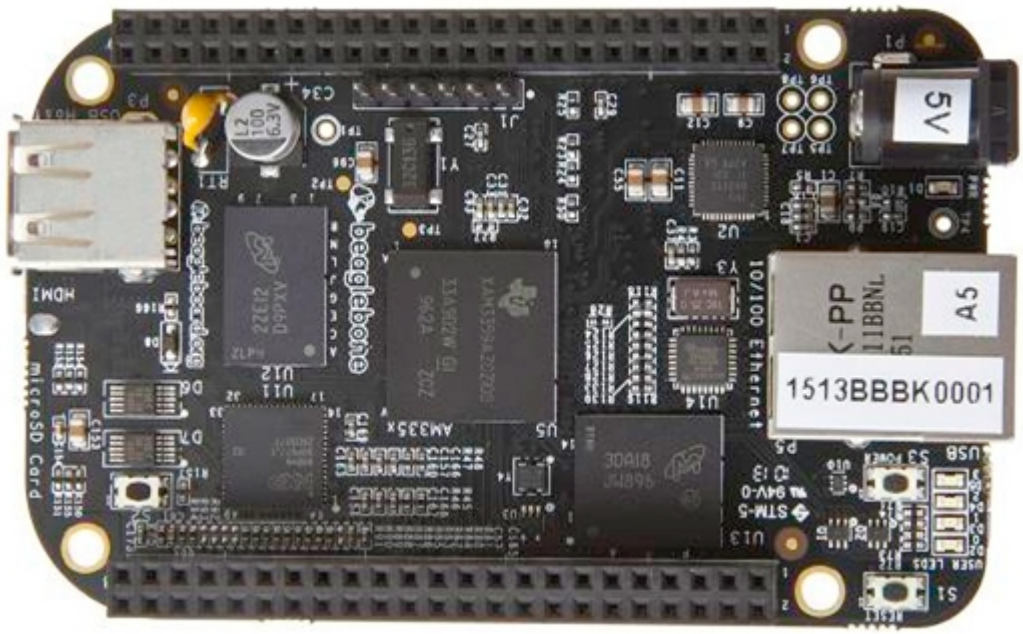
A lapon továbbá található egy nagyon pici mikrofon is, amely méretéből adódóan nem egy stúdió mikrofon, de hasznos lehet bizonyos alkalmazási területeken.



BeagleBone

Talán a második legnépszerűbb SOC fejlesztőplatform. Elsősorban beágyazott rendszerek vezérlésére tervezték, ebből kifolyólag jóval több általánosan programozható kimenete van, mint mondjuk egy RaspberryPi-nek. Jelenleg két változatban érhető el. Az eredeti BeagleBone kicsivel gyengébb teljesítményű, 256Mb memóriával van felszerelve, míg a Black változat gyorsabb, 512Mb memóriával rendelkezik és HDMI kimenete is van. Mintkét lap 2x46 láb kivezetéssel rendelkezik, amelyek nagy része általánosan programozható.

Szintén számos disztribúció érhető el az eszközre. A fejlesztő cég három rendszert preferál. Android-ot, Ubuntu-t és Angström Linuxot.

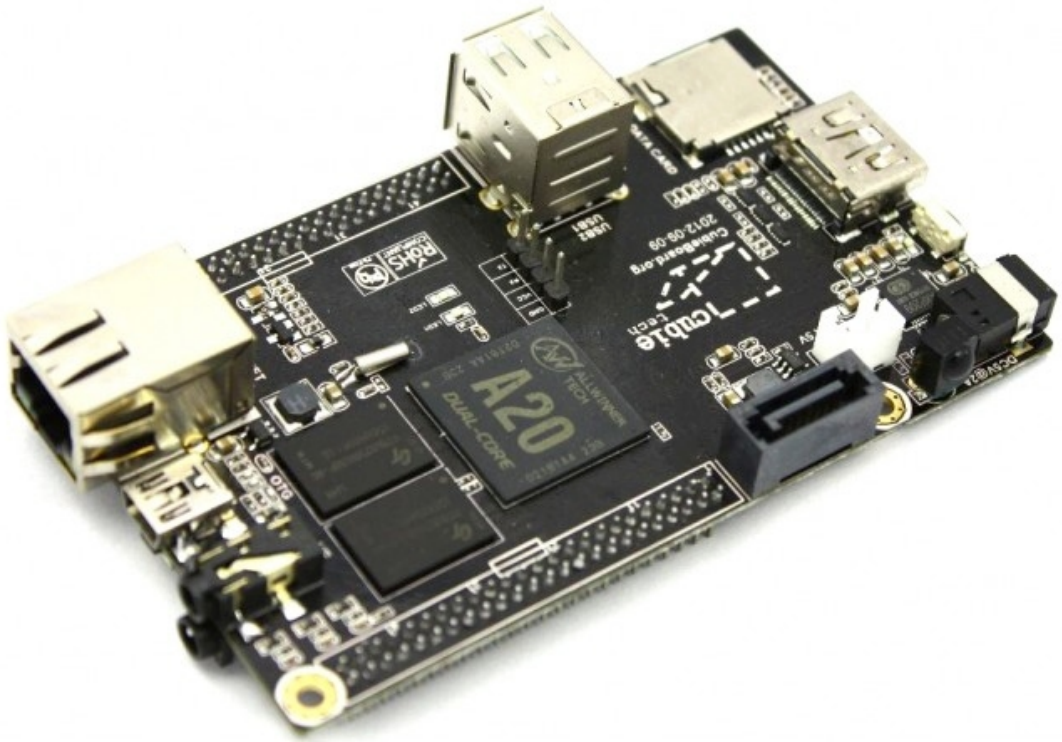


Cubieboard

A Cubieboard egy RaspberryPi-hez hasonló kínai fejlesztésű SOC lap. Három változatban kapható jelengeg. A jelenlegi legnépszerűbb modell a Cubieboard2. Amely processzora 1 GHz-en fut és két maggal rendelkezik. Az eredeti Cubieboard-al teljesen kompatibilis, ami egymagos processzorral és egy gyengébb videokártyával rendelkezik. A Cubieboard 3 pedig egy komplett kis számítógép több memóriával és több csatlakozóval, kifejezetten irodai és otthoni média PC építésre.

96 GPIO lábbal rendelkezik, amelyen van hardveresen VGA jel is, valamint 4GB integrált Flash memóriával rendelkezik, amire telepíthető az operációs rendszer. A tároló kapacitás MicroSD kártyával és SATA merevlemezekkel is bővíthető. A rendszer memóriája 1GB.

A lap számos Linux disztribúciót és Android operációs rendszert támogat.



Linux alapismeretek

A Linux rendszerek rövid történelme

Mielőtt belevágnánk az alapismeretek elsajátításába kicsit a Linux rendszer történelmével kell foglalkoznunk.

A történet azon része, ami a Linux szempontjából értdekes 1969-ig nyúlik vissza. Ekkor született meg az első Unix rendszer. Ez egy kifejezetten nagy gépekre tervezett többfelhasználós operációs rendszer volt, ami az akkori kor elvárásainak maximálisan megfelelt, egy hátránya az volt, hogy ez még teljesen Assembly nyelven volt írva, így ha egy másik gép típus támogatását el akarták készíteni, akkor az egész operációs rendszert újra kellett írni az alapoktól kezdve.

Éppen ezen feladat bonyolultsága miatt született meg 1972-ben a C programozási nyelv, amelyet kifejezetten arra alkottak meg, hogy a Unix hordozható legyen. Az alap ötlet az volt, hogy egy magas szintű nyelven megírják a teljes operációs rendszert, amit aztán csak le kell fordítani a cél gépen. Így elegendő lesz csak a fordítót újraírni a célgépre.

Ezen koncepció annyira bevált, hogy az 1980-as évek elejére a Unix rendszer egyeduralkodó lett majdnem minden területen, ahol nagy számítógépek kelltek. Azonban a Unix kereskedelmi termék volt, ebből kifolyólag nem lehetett belelátni a belső működésébe.

Ez azért volt probléma, mivel a rendszer igen komplex volt és ebből adódóan elkerülhetetlen volt az, hogy ne legyen benne programhiba. Így ha az ember találkozott egy rendszerhibával, akkor maximum jelenteni tudta azt a program készítőjének, aki vagy javította vagy nem. Ugyan ez vonatkozott a plusz funkciókat mevalósító fejlesztési kérelmekre is.

Ezen állapotokat megelégelte 1983-ban Richard M. Stallman, aki létrehozta a GNU alapítványt, illetve kidolgozta a GNU Licencet.

Ez egy olyan szabad szoftveres licenc szerződés, amiben a program eredeti alkotója lemond a jogai egy részéről és azokat átruházza a GNU alapítványra. Kiköti a Licenc szerződés továbbá azt, hogy a program szabadon terjeszthető, szabadon módosítható, egészen addig, amíg a származékos programok és fejlesztések is GNU Licenc alatt elérhetőek.

Ez azt biztosítja, hogy egy GNU alatt kiadott program mindig szabadon módosítható maradjon. Ez nem jelenti azonban azt, hogy egy GNU alatt kiadott programért nem lehet pénzt kérni

A GNU projekt rohamos fejlődésnek indult, szépen-lassan újraalkották az egész Unix rendszer minden segédprogramját nyílt forráskód alatt, azonban ez még kevés volt egy saját operációs rendszerhez, mivel nem volt meg a szükséges mag, a kernel.

A 80-as évek közepén és a 19920-es évek elején a PC piacot a Microsoft DOS rendszere uralta, ami igen távol állt a kereskedelmi Unix rendszerektől, mivel nem ugyan azokra a célokra lettek szánva. A Unix-nak ebben az időben nem volt olyan változata, amely otthoni személyi számítógépeken futtatható lett volna.

Valamikor a 80-as években jelent meg Andrew S. Tanenbaum Operációs rendszerek könyvének első változata. Ezen könyv megalkotásával az volt a célja, hogy az egyetemi hallgatói egy működő operációs rendszeren tudják elsajátítani az operációs rendszerek tervezésének alapjait. Ehhez megalkotta a Minix nevű operációs rendszert, ami nem volt kimagaslóan jó, viszont ingyenes volt és nyílt forráskódú, amibe bárki belenézethetett. Éppen ezért elég sokan módosítgatták és játszottak vele.

1991 környékén Linus Torvalds a Helsinki Egyetem másodéves hallgatója volt. Ebben az időszakban határozta el, hogy ír egy saját operációs rendszert, mivel a GNU operációs rendszerére akkori számítások szerint 3-4 évet kellet volna várni. (A GNU/HURD rendszer projekt legfrissebb verziója 0.5, ami még igen messze áll egy mindennapos munkára alkalmazható rendszertől)

Úgy gondolta, ha már operációs rendszert ír, akkor kihasználja az Intel 80386-os processzorának minden képességét, így megkezdődött a Linux rendszer fejlesztése. A projektet először 1991. Augusztus 25-én írt e-mail üzenetében jelentette be a Minix levelezési listán. A levél szövege magyar fordításban a következő volt:

> „Üdv minden Minix-felhasználónak odaát! Egy (ingyenes) operációs rendszert csinálok (csak hobbiból, nem lesz olyan nagy és profi, mint a gnu) a 386-os (486-os) AT-klónokhoz. Április óta érlelem, és lassan elkészül. Szeretnék visszajelzéseket arról, hogy mi tetszik és mi nem tetszik a Minixben az embereknek, mivel az én operációs rendszerem némileg hasonlít rá (többek között (gyakorlati okokból) azonos a fájlrendszer fizikai kiosztása).
>
> A bash (1.08) és a gcc (1.40) programokat már átültettem, és úgy tűnik, működik a dolog. Ez azt jelenti, hogy pár hónapon belül valami hasznosat fogok kapni, és kíváncsi lennék, milyen funkciókat szeretnének legtöbben. Minden javaslatot szívesen veszek, azt viszont nem ígérem, hogy meg is valósítom őket :-)
>
> -Linus (torvalds@kruuna.helsinki.fi)
>
> Ui.: Igen! Nincs benne Minix-kód és többszálú fs-sel rendelkezik. Nem hordozható (a 386 feladatváltást használja stb.), és lehet, hogy soha nem is fog az AT-merevlemezeken kívül bármi mást támogatni, minthogy nekem csak ez van :-(."

Levele számos programozót inspirált, akik bekapcsolódtak a fejlesztésébe. A rendszer az elmúlt évek alatt rengeteget fejlődött.

Disztribúciók

Maga a Linux "csak" a rendszermag, vagyis az operációs rendszer fő komponense. Ahhoz, hogy ebből futtatható rendszert kapjunk egy csomó programot kell még mellé tenni. Az ilyen csomagokat, amelyek Linuxon alapulnak és komplett rendszerek disztribúciónak nevezzük.

Disztribúciókból némi túlzással annyi fajta létezik, mint amennyi csillag az égen van. Azonban ha csak a komolyabb felhasználói bázissal rendelkezőket nézzük, amelyek alkalmasak beágyazott rendszereken való futtatásra, akkor csak pár darab rendszer marad ami szóba jöhet.

Természetesen mindig adott a lehetőség egy újabb rendszer megalkotására, azonban ez nem egy egyszerű feladat, ezen felül időigényes is. Ezért célszerűbb egy meglévő rendszert az igényeinknek megfelelően átszabni. További rendszerekről, amelyek nem szerepelnek a leírásban érdemes a http://distrowatch.com/ oldalon tájékozódni.

Érdeklődők olyan leírásokat, amelyek segítenek saját rendszer készítésében, fejlesztésében a http://www.linuxfromscratch.org/ címen találhatnak.

Debian GNU/Linux

A Debian disztribúció az egyik legrégebben fejlesztett és karbantartott projekt. A projekt 1993-ban indult útjára, azóta számos disztribúciónak képezi alapját. Fejlesztési irányelveiben megfelel a GNU követelményeknek.

Három fő ága létezik. A Stabil rendszer (stable), ez mindig az aktuális ajánlott verzió lesz. Legtöbbször csak biztonsági frissítések érkeznek hozzá, új funkciók nem. Mivel a fejlesztési elv része, hogy megbízható és stabil legyen a rendszer ezért az új funkciók és programok ellenőrzése általában eltart egy darabig, így nem mindig a legfrissebb programokat tartalmazza.

A tesztelési ágból (testing) lesz a következő stabil változat, fejlesztői ág, éppen ezért nem ajánlják a fejlesztők a használatát éles körülmények közben.

A harmadik ág, az úgynevezett instabil ág. Ide kerülnek be legelőször az új programok és fejlesztések. Azonban mivel ezen programokat még nem tesztelték alaposan semmi garancia nincs arra, hogy a programok működni fognak. Csak tapasztalt felhasználóknak ajánlott ezen ág használata, mivel előfordulhat, hogy egy hibás program után javításokat kell végezni a rendszeren, hogy az működőképes maradjon.

A stabil kiadások érhetőek el telepítő médián, a többi ág a frissítési csatornákon aktiválható. A stabil kiadások minden esetben kódnévvel vannak ellátva. Az eddigi kiadások a kódneveiket a Pixar Toy Story animációs sorozatának szereplőiről kapták.

A rendszer telepítőképei a http://www.debian.org/ címről szerezhetőek be. RaspberryPi esetén a Pi-ra szabott disztribúció a http://www.raspbian.org/ címről szerezhető be.

Fedora

A Fedora projekt 2003-ban indult, miután a Red Hat felhagyott az Otthoni felhasználóknak szánt operációs rendszer fejlesztéssel. A projektet a Red Hat szponzorálja elsősorban. A projekt keretén belül létrehozott fejlesztések később átkerülnek a Red Hat kereskedelmi célokra alkotott operációs rendszerébe, a Red Hat Enterprise Linux-ba. Ez egy kifejezetten vállalati környezet számára kialakított stabil, ám naprakész rendszer, aminek meg is kérik az árát.

A Fedora afelé kísérletezési terep, de nem jelenti azt, hogy instabil, vagy használhatatlan lenne. Nagyjából fél évente jelenik meg új verziója, de ez nem szent írás. A fejlesztők gyakran csúsztatják a határidőt annak érdekében, hogy a kiadott rendszer stabil és használható legyen. Az Ubuntu után talán a második vagy harmadik legnépszerűbb Linux disztribúció.

A rendszer a http://fedoraproject.org/hu/get-fedora címen szerezhető be, RaspberryPi-ra telepítési útmutató a http://fedoraproject.org/wiki/Raspberry_Pi#Raspberry-PI_Fedora_Remix címen érhető el.

Ubuntu

Az Ubuntu magát a világ legnépszerűbb ingyenes operációs rendszereként reklámozza. Debian Linux variánsként indult a projekt pályafutása 2004-ben. A fejlesztők eredeti ötlete az volt, hogy létrehoznak egy olyan Debian variánst, amely programok tekintetében naprakészebb a stabil Debian ágnál. A projekt hat hónapos fejlesztési ciklussal dolgozik, minden év áprilisában és októberében van újabb kiadás.

A fél évente megjelenő változatok közül két évente az egyik rendszer hosszútávon támogatott minősítést kap. Ezen változatok használata erősen javasolt, mivel a fél évente kiadott változatokban gyakran akadnak hibák, főleg az utóbbi időben.

Számos disztribúciót inspirált az Ubuntu, rengeteg variánsa létezik. A rendszer telepítője a http://www.ubuntu.com/download/desktop címen érhető el asztali számítógépek számára. RaspberryPi támogatása a rendszernek jelenleg nincs.

Ångström Linux

Speciálisan beágyazott rendszerek számára fejlesztet Linux variáns. Akár 4Mb tárhelyre is telepíthető. Fejlesztői szerint nem kizárt, hogy egy napon kenyérpirítókon is fut majd.

Nem éppen kezdőknek szánt disztribúció, mivel forráskódból kell telepíteni, viszont azokon a rendszereken, amelyeket támogat kimagaslóan jó teljesítményt fog nyújtani.

A BeagleBoard Black alapértelmezetten telepítve tartalmazza egy változatát, ami tesztelési célokat szolgál elsősorban, de ennél többre is használható a rendszer.

A rendszer telepítéséhez és létrehozásához információk a http://www.angstrom-distribution.org/building-angstrom oldalon találhatóak. A fejlesztők biztosítanak egy online rendszerkép generátor programot is, amely a

http://narcissus.angstrom-distribution.org/ oldalon érhető el.

fájlok, könyvtárak, fájlrendszerek

A Unix rendszerek és a Windows rendszerek közötti hatalmas különbség a fájlkezelés szemlélete. Windows esetén meghajtókon belül mappákat kezelünk klasszikus értelemben véve. A meghajtó egy speciális mappa, ami egy eszközhöz/partícióhoz rendelődik. Unix rendszerek alatt elsősorban mappákat kezelünk. Egy mappa lehet normál mappa egy eszközön belül, vagy a mappa reprezentálhat egy teljes meghajtót/partíciót.

Az összes mappa a / jelű gyökér fájlrendszerbe csatlakozik, Windows terminológiában ezt felfoghatjuk C meghajtóként. A / jelű gyökér mappa többsége egy meghajtón vagy partíción belül fog elhelyezkedni, de mint említettem lehet a mappa egy teljes más meghajtón is.

Ezen szemléletmód előnye az, hogy korlátlan számú lemez csatlakoztatható így a rendszerbe, míg Windows esetén klasszikus kezelési módszerekkel „csak" 26.

További érdekesség, hogy Linux alatt a számítógépben található összes hardver elem megjelenik fájlként egy speciális mappában. Ezen fájlok ténylegesen nem foglalnak helyet a merevlemezen, mivel a memóriában vannak.

Mivel a hardver elemek egyszerre fájlok is, egyszerű fájl írás és olvasás művelettel többségüket kezelhetjük (GPIO portok programozása kapcsán lesz róla szó)

Mivel a legtöbb mappa neve rövidítés ezért elsőre nem biztos, hogy egyértelmű lesz, hogy melyik mappa mire való, illetve mit tartalmaz. Az alábbiakban a szabványos Linux mappák információi vannak összefoglalva

- ***/bin***
  Futtatható programok és parancsok bináris fájljainak helye.
- ***/boot***
  Rendszer betöltőfájljai. Külön partíció is szokott lenni, itt található a kernel is.
- ***/dev***
  Virtuális mappa. Itt minden a számítógépben található hardver mappaként és fájlként látszik.
- ***/etc***
  Rendszer-konfigurációs fájlok.
- ***/home***
  Felhasználók saját könyvtárai. Külön partíció szokott lenni komolyabb telepítésekben.
- ***/lib***
  Programkönyvtárak lelőhelye. A Programkönyvtár Windows analógiát használva hasonló a .dll fájlhoz.
- ***/lost+found***
  Fájlrendszer meghibásodása esetén sikeres javítás után ide kerülnek a töredék fájlok.
- ***/media***
  Ezen mappa tartalma alá szokás csatolni a hordozható eszközöket. Pl: Mobil HDD, Pendrive, CD/DVD. A /media mappa nagyon régi kernelek esetén /mnt néven szerepel.
- ***/opt***
  Opcionálisan telepített szoftverek telepítési helye. Ide csak azon szoftverek kerülnek, amiket nem a disztribúció szállítója terjeszt.
- ***/proc***
  Futó folyamatokról információk mappák és fájlok formájában. A lemezen valóságban 0 byte a mérete a mappának.
- ***/root***
  Rendszergazda felhasználó saját könyvtára.
- ***/sbin***

Hasonló a /bin mappához, azzal a különbséggel, hogy itt a rendszer-adminisztrációs parancsok binárisai vannak.

- ***/tmp***
  Ideiglenes, átmeneti fájlok helye.
- ***/usr***
  Felhasználók által telepített programok, dokumentációk, forráskódok és sokminden egyéb lelőhelye. Amit telepítünk a disztribúció szállítója által, az nagy valószínűséggel itt fog landolni.
- ***/sys***
  Hasonló a tartalma a /dev mappához, de itt több információ érhető el az eszközökről, valamint itt kategóriába szervezve vannak az eszközinformációk. Ténylegesen helyet nem foglal a lemezen.
- ***/var***
  Programok naplófájljai, levelezési fiókok, valamint egyes programok ideiglenes fájljai, amit hosszabb ideig tárolnak.

Minden fájlhoz és mappához jogosultságok rendelhetőek három szerepkörre osztva. A három alapvető jogosultság az írás, olvasás és a futtatás. A három csoport pedig a tulajdonos, tulajdonos csoportja, világ többi része.

Linux esetén jó néhány fájlrendszer szóba jöhet használatra. A fájlrendszerek sokaságának oka a nyílt forráskódban keresendő. Egyes fájlrendszerek bizonyos célokra jobban megfelelnek, mint mások, mivel egy mindenre jó fájlrendszert elkészíteni szinte lehetetlen. A fő linuxos fájlrendszerek, a teljesség igénye nélkül:

- *ext2*
- *ext3*
- *ext4*
- *reiserfs*
- *xfs*

ext2

Az első komolyabb Linux fájlrendszerek egyike. Az ext fájlrendszert váltotta. Az ext név onnan jön, hogy kezdetben a Linux nem rendelkezett saját fájlrendszerrel, helyette a Minix lemezformátumát alkalmazta, ami hibátlan volt, de számos limitációt hordozott magában. Például a fájlnevek nem lehettek hosszabbak 14 karakternél. Ezen limitációk megszüntetésének érdekében kezdték el fejleszteni az ext fájlrendszert. Az ext szó az extended angol szóból jön, ami kiterjesztettet jelent. A fájlrendszer első verziója a 0.96c Linux kernel verzióban bukkant fel, de voltak vele problémák. Az ext2 a 0.99 változatban mutatkozott be, javítva az eredeti fájlrendszer problémáit, széles körben ez terjedt el. Sokáig számos Linux rendszer alapértelmezett lemezformátuma volt, egészen addig, amíg nem jelent meg a frissített verziója.

ext3

Az ext2 rendszer legnagyobb baja, hogy nem naplózó fájlrendszer. Ez bizonyos esetekben (Pl. pendrive, floppy, stb...) jól jön, mert gyorsabb tud lenni a rendszer. Azonban adatvesztést tud okozni egy hirtelen áramszünet esetén, mivel a fájlrendszernek fogalma se lesz róla, hogy az áram szünet előtt mit csinált. A problémát úgy oldották meg a fejlesztők, hogy hirtelen leállás után a fájlrendszert átvizsgálta a rendszer és a fájltöredékeket, amelyek írása, másolása nem fejeződött be a lost+fond mappába tette a rendszer, ahonnan a felhasználó válogathatott, hogy mit kezd velük.

Naplózó fájlrendszerek esetén nincs ilyen gond, mivel a rendszer minden egyes írási/olvasási műveletet naplóz, így egy áramszünet után iis tudja, hogy mit csinált. Az adatvesztés esélye ilyenkor is fennáll, de kisebb a

kockázat. Cserébe a naplózás lassabbá teszi a fájlrendszert. Az ext3 lemez formátuma megegyezik az ext2-vel, így az ext3 partíciók használhatók napló nélkül régebbi rendszereken is.

ext4

A negyedik kiterjesztett fájlrendszer szintén visszafelé kompatibilis, képes működni ext3 fájlrendszerként. Azonban ha ténylegesen ext4 rendszerként van kezelve, akkor az ext3 számos problémáját (lassúság) javítja. Valamint az elvi maximális korlátokat feljebb tornázta. A naplózása opcionálisan kikapcsolható, ami alkalmassá teszi pendrive és egyéb hordozható meghajtókon való alkalmazásra. Használni már csak a gyorsasága miatt is érdemes.

resiserfs

Nevét fő fejlesztőjéről Hans Reiser-ről kapta. Az első naplózó fájlrendszer volt a Linux-ban. Ezért megjelenésekor hatalmas előnye volt az ext2 rendszerrel szemben. További nagy előnye, hogy a hagyományos szektor osztásos rendszer helyett az adatokat egy dinamikus bináris fa szerkezetben tárolja, így a fájlok pontosan annyi helyet foglalnak a lemezen, mint amennyi a méretük. Hátránya, hogy a sok használattól a sebessége drámaian le tud csökkenni, ezért nem igen alkalmazott manapság.

XFS

Az első 64 bites naplózó fájlrendszer, 2000-ben vált nyílt forráskódúvá, azóta része a Linux kernelnek. Jellemzője a nagy teljesítmény és a skálázhatóság. Hátránya, hogy nem használható rendszermeghajtó típusként, mivel nem támogatja az operációs rendszer betöltő programok telepítését.

	maximális fájlméret	Maximális Fájl darabszám	Partíció maximális mérete
ext2	2 TiB*	10^18*	32TiB*
ext3	2 TiB*	10^18*	32TiB*
ext4	16 TiB	4 billió	1EiB*
ReiserFs	8 TiB	4 billió	16TiB
XFS	8 EiB	nincs adat	16EiB

83. Táblázat: Linux fájlrendszerek fontosabb technikai adatai

*A fenti táblázatban *-al megjelölt adatok szektor osztás függőek, a maximális méretek lettek feltüntetve. Ext4 típusú partíció esetén a maximális partíció méret 1EiB lehet, azonban egyelőre a fejlesztők nem ajánlják 16TiB méret feletti partíciók alkalmazása esetén.*

A fő fájlrendszerek mellett számos más fájlrendszer is támogatott. Beleértve a Windows által használt FAT16, FAT32 és NTFS rendszereket is. NTFS esetén érdemes megjegyezni, hogy a kernel csak olvasási támogatással rendelkezik jogi és implementálási okok miatt. Azonban ez nem azt jelenti, hogy NTFS meghajtóra írni nem lehet.

Az NTFS írás megoldható a magyar fejlesztésű NTFS-3G program telepítésével. A program a kernel FUSE lehetőségét használja ki, ami lehetővé teszi fájlrendszerek létrehozását és kezelését felhasználói szinten. Ennek azonban ára van. A felhasználói szinten kezelt fájlrendszerek jóval lassabbak lesznek a natív kernel által kezelteknél. Így sajnos elmondható, hogy az NTFS partíciók írása nem éppen a leggyorsabb.

Bővebb információ az NTFS-3G kezeléséről a http://www.tuxera.com/community/ntfs-3g-manual/ címen található. A FUSE projekt által támogatott fájlrendszerek sokaságáról a http://sourceforge.net/apps/mediawiki/fuse/index.php?title=FileSystems címen lehet tájékozódni.

A Parancsértelmező

Parancssoros rendszerek esetén az általános felépítése egy parancsnak a következő: parancs neve kapcsolók fájlok. A kapcsolók olyan paraméterek, amelyek kötőjellel (-), vagy dupla kötőjellel (–) kezdődnek. A – jeles kapcsolókat rövid nevű kapcsolóknak nevezik, mivel ezek általában 1-2 karakterből állnak.

A — jeles kapcsolók hosszú nevűek. Ezen kódolási konvenció nem minden programra jellemző, nincs univerzális szabvány. Ezért egy új parancs használata előtt érdemes áttekinteni a használati utasítását.

Az összes Linuxos parancsértelmező kis – és nagybetű érzékeny. Tehát a hello.txt és Hello.txt két különböző fájlt jelent az operációs rendszer számára.

Elektronikus használati utasítás minden parancssoros programhoz elérhető. Erre a man parancs szolgál, amely megjeleníti az argumentumnak megadott program dokumentációját. A dokumentum megjelenítőből a q gomb megnyomásával lehet kilépni.

Elérési útvonalakban használhatóak speciális rövidítések is. Ezek:

Rövidítés	**Jelentése**
~	A bejelentkezett felhasználó saját mappáját jelenti.
.	Aktuális könyvtár, amiben vagyunk.
..	Aktuális könyvtártól eggyel feljebb lévő könyvtárat jelenti.
-	A konzol bemenetet jelentő fájl. Olyan programok esetén használható, amelyek képesek szabványos bemenetről információ olvasásra. Különösen hasznos tud lenni parancsok összekapcsolásakor.

84. Táblázat: Elérési útvonalakban használt speciális karakterek

Linux rendszerek esetén sok parancsértelmező közül választhatunk, azonban az általánosan elterjedt értelmező, amit rengeteg rendszer alapértelmezetten kínál, az a BASH. Mint minden parancsértelmező, ez is kínál programozási lehetőségeket. A programozási nyelve nem éppen egyszerű, de azt szokták mondani, hogy aki jártas a programozásában, az csak nagyon ritkán nyúl C nyelvhez egy feladat megoldásához. A BASH-hez kapcsolódó programozási ismeretekből a legfontosabbak az operátorok, mivel ezek használata elkerülhetetlen, ha az ember valami komolyabb dolgot akar tenni.

Ki – és bemenet fájlba/fájlból átirányító operátorok

Mint ahogy a név is mutatja, ezen operátorok arra jók, hogy egy a lemezen lévő fájlt bemenetképpen szolgáltassunk parancsnak, vagy a parancs kimenetét fájlba írjuk. Ezen célra a [és] jelek használhatóak. Használatuk:

```
parancs > kimenet
```

A parancs által a képernyőre írandó szöveget nem jeleníti meg, helyette beleírja a kimenet fájlba.

```
parancs >> kimenet
```

Hasonlóképpen működik a sima] operátorhoz, azonban ha a kimenet fájl már létezik, akkor nem írja felül azt, hanem a fájl végére fűzi a kimenetet.

```
parancs < bemenet
```

Olyan programok esetén használható, amelyek billentyűzetről várnának bemenetet. Ezen operátor segítségével a bemenet szolgáltatható egy fájlból.

Parancsok kimenetének összekapcsolása

Unix rendszerek alatt is lehetőség van egy parancs kimenetének közvetlen beadagolására egy másik programba. Ezen technikát Pipe-oknak nevezik, amely magyarul csővezetéket jelent. A Pipe lényegében úgy működik, mintha a kimenetbe átirányító fájl a memóriában helyezkedne el. A másik program, amibe meg átirányítunk ezen „fájlt" olvassa. Ezen fájl csak a Pipe-on keresztül összekapcsolt programok számára látható,illetve csak az összekapcsolás ideje alatt létezik. Valamint nem az egész fájl helyezkedik el a memóriában, hanem mindig csak egy fix méretű részlete, hiszen csak előre lehet haladni a fájlban.

Pipe-on keresztüli összekapcsolásra a Pipe jel (|) operátor használható. Ezen operátor magyar billentyűzetkiosztás mellett az ALTGR+W gombok megnyomásával csalogatható elő. Használata:

```
program1 | program2
```

Parancs kimenetének argumentumként való felhasználása

Egy parancs kimenete használható parancssori kapcsolóként, vagy argumentumként. Erre a célra a „beszúró idézőjel" karakter (`) használható. Magyar billentyűzet kiosztás mellett a karakter az ALTGR+7 gomb megnyomásával hozható elő. Használata:

```
program1 `program2`
```

Folyamatok kezelése

Folyamatnak a futó programokat, parancsokat szokás nevezni. A shell-en keresztül futó folyamatok kezeléséhez a bash rendelkezik egy pár billentyűkombinációval, amit érdemes megjegyezni:

- *CTRL+C:*
  Aktuálisan futó folyamat megszakítása, majd a vezérlés visszaadása a parancsértelmezőnek.
- *CTRL+K:*
  Az éppen futtatott folyamatot megállítva a háttérbe küldi. A folyamat ekkor nem ér véget, csupán felfüggesztésre kerül. A felfüggesztett parancs az fg parancs segítségével hozható ismételten előtérbe.

Továbbá, amennyiben egy parancsot & karakterrel zárva indítunk el, akkor az a program a háttérben fog futni. Így eközben a terminálon tudunk mást csinálni. Példa:

```
program &
```

Parancs álnevek

A bash lehetőséget biztosít parancsokhoz álnevek rendeléséhez. Ezen álnevek hosszú parancsok esetén előnyösek, mivel így egy álnévvel tudunk hivatkozni a parancsunkra, ahelyett, hogy több kilométer paramétert írnánk be. Az álnevek kezelésére két parancsot kínál a bash: alias és unalias.

Az alias paranccsal definiálunk egy álnevet a következő formában:

```
alias [álnév] [hivatkozott parancs és paraméterek]
```

Az unalias parancs meg törli a definiált álnevet:

```
unalias [álnév]
```

Gyorsbillentyűk

Alapvetően egy parancssoros rendszert kezelni nem annyira kényelmes, mint mondjuk kattintgatni egy grafikus felületen. Ezért a bash alkotói igyekeztek mindent megtenni, hogy kézen fekvő, kényelmes legyen a shell használata. Ez azt jeleni, hogy számos gyorsbillentyű van beépítve a rendszerbe:

- *TAB:*
  Fájl – és mappanév automatikus kiegészítése.
- *CTRL+A:*
  Az aktuálisan gépelt parancssor elejére teszi a kurzort.
- *CTRL+E:*
  Az aktuálisan gépelt parancssor végére teszi a kurzort.
- *CTRL+L:*
  Képernyő törlés, hasonlóan működik, mint a clear parancs.
- *CTRL+H:*
  Ugyanazt az eredményt produkálja, mint a Backspace gomb.
- *CTRL+R:*
  Az előzőleg begépelt parancsok közötti keresést teszi lehetővé.
- *CTRL+W:*
  A kurzor után álló szó törlése.
- *CTRL+T:*
  Kurzor előtt álló utolsó két karakter cseréje.

- *ESC+T:*
  Kurzor előtt álló utolsó két szó cseréje.
- *ALT+F:*
  Aktuális sorban a következő szóra mozgatja a kurzort.
- *ALT+B: Aktuális sorban az előző szóra mozgatja a kurzort.*

Fájl és könyvtárkezelő parancsok

cd

```
cd [útvonal]
```

Könyvtárváltás. Paraméterek nélkül nem csinál semmit. A könyvtárelérési útvonal lehet relatív az aktuális mappához képest, vagy abszolút útvonal is a / könyvárhoz képest. Példák:

```
cd ~
cd /
cd ..
```

mkdir

```
mkdir [útvonal]
```

Könyvtárlétrehozás. Paraméterezett használata megegyezik a cd parancséval.

rmdir

```
rmdir [mappanév]
```

Üres könyvtár törlése. Amennyiben a könyvtár nem üres, és a teljes tartalmát törölni akarjuk, akkor az rm parancs használható.

ls

```
ls [kapcsolók]
```

Könyvtár tartalmának listázása. Alapértelmezetten csak a fájl – és könyvtárneveket jeleníti meg. Kapcsolói:

- *-l*
  Hosszú listát készít, amiben a fájl/mappa jogosultságai, tulajdonos neve, a fájl mérete és utolsó módosítási dátuma is látszik a fájlnév előtt.
- *-a*
  Rejtett fájlokat is megjeleníti. Unix rendszerek alatt rejtett fájl az, amely neve . karakterrel kezdődik.
- *-A*
  Hasonlóan működik a -a kapcsolóhoz, azonban ekkor nem jeleníti meg a . és .. bejegyzéseket, amelyek az adott mappára és a felette lévő mappára vonatkoznak.
- *-hA*
  -l kapcsolóval együtt használható. Ekkor a fájl méretét emberileg is olvasható formátumban írja ki.
- *-R*
  Az alkönyvtárak tartalmait is kilistázza.

- *-S*

  A listát fájlméret szerint rendezi.
- *-t*

  A listát a fájlok utolsó módosítási dátuma szerint rendezi.

cp

```
cp [kapcsolók] [fájlok]
```

Fájlok másolása. Első argumentuma a forrásfájl, második argumentuma a célfájl. Kapcsolói:

- *-R vagy -r*

  Ha a forrásfájl egy mappa, akkor ezen kapcsolóval a mappa összes almappáját is másolja a célra.
- *-l*

  A forrásfájlt nem másolja, helyette hardlinket készít a célon, amely a forrásra mutat.
- *-s*

  A forrásfájlt nem másolja, helyette softlinket készít a célon, amely a forrásra mutat.
- *-i*

  Amennyiben a célfájl létezik, akkor megerősítést kér a felülírás előtt.
- *-n*

  Amennyiben a célfájl létezik, akkor nem írja felül.

mv

```
mv [kapcsolók] [fájl1] [fájl2]
```

Fájlok áthelyezése, átnevezése. Szintén első argumentum a forrásfájl, második meg a célfájl. Kapcsolói:

- *-i*

  Amennyiben a célfájl létezik, akkor megerősítést kér a felülírás előtt.
- *-n*

  Amennyiben a célfájl létezik, akkor nem írja felül.

chmod

```
chmod [jogok] [fájl]
```

Fájl / mappa jogosultságainak módosítása. A jogokat vagy számok megadásával, vagy betűjelek megadásával változtathatjuk meg. A betűjeles megadást én túlzottan sosem szerettem, így csak a numerikus megadásról lesz szó. A következő jogok adhatóak ki:

- *Olvasás, betűjele r, száma: 4*
- *Írás, betűjele w, száma: 2*
- *Futtatás, betűjele x, száma 1*

A jogokat tulajdonosnak, tulajdonos csoportjának és az egyéb felhasználóknak is meg kell adni. Numerikus jogosultság beállítás esetén a megfelelő jogok számát össze kell adni. Amennyiben semmilyen jogot nem adunk, a számunk nulla lesz. Példaképpen nézzük azt az esetet, hogy a tulajdonosnak minden jogot meg szeretnénk adni, csoportjának csak az olvasás és futtatás jogot akarjuk megadni, és mindenki másnak csak olvasási jogot akarunk adni. Ekkor, amennyiben a fájlunk neve zene.mp3, akkor a chmod parancsunk így fog kinézni:

```
chmod 754 zene.mp3
```

A parancs esetén egyetlen fontos kapcsolóról beszélhetünk, ez a -R. Ennek segítségével, ha mappán futtatjuk a parancsot, akkor az összes almappa és fájl ugyanazokat a jogokat kapja meg.

chown

```
chown [felhasználó] [fájl]
```

Fájl / mappa tulajdonosának beállítása. Ezen parancs segítségével egy másik felhasználó tulajdonába helyezhetjük át a fájljainkat. A felhasználónév paraméter az új tulajdonos bejelentkezési neve. A -R paraméterrel mappán futtatva a parancsot a hatása az az összes almappát és fájlt érinti

chgrp

```
chgrp [csoport] [fájl]
```

Fájl / mappa csoportjának beállítása. Használata megegyezik a chown paranccsal. Kapcsolói:

- *-R*
  Rekurzív végrehajtás. Mappa esetén az összes almappára és fájlra alkalmazza a változásokat.
- *-h*
  Szimbolikus linkek esetén csak magát a linket érinti a változás, a fájlt, amire a link mutat, nem.

df

Szabad helyet listázza ki a meghajtókon. Alapértelmezetten byte-ban írja ki az információkat, ezért a -h paraméterrel érdemes használni, amely emberileg is olvasható formában írja ki az információkat.

ln

```
ln [kapcsolók] [forrás] [link]
```

Egy fájlra, mappára mutató linket hoz létre. A link egy virtuális fájl, felfogható úgy, mint egy parancsikon, azzal a különbséggel, hogy a linkelt fájl mérete megegyezik az eredetivel. Linux és Unix rendszerek alatt két típusú link létezik, hard és soft link. Különbség a kettő között az, hogy a hardlink a fájl tényleges fizikai elhelyezkedése alapján mutat a fájlra, míg a softlink a fájl neve alapján. Használható kapcsolók:

- *-n*

  Amennyiben a célfájl létezik, nem írja felül.
- *-s*

  Softlinket hoz létre hardlink helyett.

find

`find [kapcsolók] [fájlnév]`

Fájlok, mappák keresése töredék fájlnév alapján. Kapcsolói:

- *-ls*

  ls -l parancs szerűen írja ki a találatokat.

file

`file [kapcsolók] [fájlnév]`

A paraméterként megadott fájl típusát azonosítja be a tartalma alapján. A Unix rendszerek a fájlok típusait nem kiterjesztés alapján azonosítják, hanem tartalom alapján. Kapcsolói:

- *-h Nem követi a linkeket.*

Komolyabb fájlkezelő parancsok

wget

```
wget [kapcsolók] [url]
```

Http vagy ftp szerverről tölt le fájlt helyi használatra. Kapcsolói:

- *-i [fájl]*
  URL címek olvasása fájlból.
- *-O [fájl]*
  Kimeneti fájl meghatározása. Alapértelmezetten a fájl neve az URL címből kerül meghatározásra.
- *-t [szám]*
  Sikertelen kapcsolódás esetén az újrapróbálkozások száma.
  -C
  Korábban megszakított letöltés folytatása, ha a szerver támogatja. Ebben az esetben ugyan azon mappában kell futtatni, mint ahol a részlegesen letöltött fájl is van.
  -T [másodperc]
  Sikertelen kapcsolódás esetén a próbálkozási időt adja meg.

mc

Kétpaneles fájlkezelő program, hasonló a norton commander-hez.

tar

```
tar [funkciók és kapcsolók] [fájlok]
```

Tar formátumú tömörítő és kitömörítő program. Unix rendszerek esetén a leggyakoribb archívumformátumok egyike. Önmagában nem tömörít, csak fájlokat tárol. Ezért sok esetben valamilyen extra tömörítéssel együtt használatos. Maga a tar program fel van készítve a tömörítésre és fogadásra egyaránt. Funkciói:

- *c*
  Archívum létrehozása.
- *r*
  Fájlok hozzáfűzése az archívumhoz.
- *t*
  Archívumban található fájlok listázása.
- *z*
  Gzip tömörítés használata (gyors, viszont nem egy nagy tömörítés).
- *j*
  Bzip2 tömörítés használata (lassabb, kisebb méret).

- *J*

  Xz tömörítés használata (legjobb és leglassabb, ugyan azon az elven alapul, mint a 7z formátum).
- *f [név]*

  Meghatározza az archívum nevét.
- *v*

  Képernyőre írja, hogy mit bontott ki vagy mit csomagolt éppen be.

tar kibontása tömörítéstől függően:

```
tar xvf [tar] [hova]
tar xvzf [tar.gz] [hova]
tar xvjf [tar.bz2] [hova]
tar xvJf [tar.xz] [hova]
```

tar készítése tömörítéstől függően:

```
tar cvf [tar] honnan
tar cvzf [tar.gz] [honnan]
tar cvjf [tar.bz2] [honnan]
tar cvJf [tar.xz] [honnan]
```

gzip

```
gzip [kapcsolók] [fájl]
```

Egyetlen fájlt tömörít vagy kicsomagol gzip algoritmussal. Ha a fájl gz kiterjesztéssel végződik, akkor kitömörít, ha nem, akkor becsomagol. Kapcsolói:

- *-C*

  Képernyőre írja a kimenetet fájl helyett.
- *-t*

  Teszteli a tömörített fájl épségét.
- *–1*

  Kisebb tömörítés, gyorsabb működés.
- *–9*

  Nagyobb tömörítés, lassabb működés.

bizp2

```
bzip2 [kapcsoló] [fájl]
```

Egyetlen fájlt tömörít bzip algoritmussal. Kapcsolói:

- *–1*

  Kisebb tömörítés, gyorsabb működés
- *–9*

  Nagyobb tömörítés, lassabb működés

bunzip2

```
bunzip2 [fájl.bzip2]
```

Kitömörít egy bzip algoritmussal tömörített fájlt.

bzcat

```
bzcat [fájl.bzip2]
```

Képernyőre tömörít ki egy bzip algoritmussal tömörített fájlt. Hasonlóan működik a cat programhoz.

7z

```
7z [művelet] [kapcsolók] [fájl]
```

7zip formátumú tömörítő és kitömörítő program. Műveletei:

- *a*
  Fájlok hozzáadása az archívumhoz.
- *e*
  Kicsomagolás az archívumból mappa nevek kihagyásával.
- *x*
  Kicsomagolás az archívumból mappa nevek figyelembe vételével.
- *l*
  Archívumban tárolt fájlok listázása.
- *t*
  Achívum épségének tesztelése.

Kapcsolók:

- *-o[könyvtár]*
  Kimeneti könyvtár beállítása. A -o és a könyvtárnév között nem kell szóköz.
- *-p[jelszó]*
  Jelszó beállítása. A -p és a jelszó között szintén nem kell szóköz.
- *-so*
  A kimenetet nem fájlba írja, hanem a képernyőre.
- *-y*
  Minden kérdésre alapértelmezett igen a válasz.

md5sum

```
md5sum [kapcsolók] [fájlok]
```

Md5 ellenőrző összeg számítása fájlokhoz. Segítségével ellenőrizhető egy fájl épsége. Kapcsolói:

- *-b*

  Bináris fájl olvasási mód (ajánlott).
- *-t*

  Szöveges fájl olvasási mód.
- *-c*

  Visszaellenőrzés fájlból.

fájlokhoz tartozó md5 generálása:

```
md5sum -b *.* >> fajlok.md5
```

fájlok visszaellenőrzése generált md5-ből:

```
md5sum -b -c fajlok.md5
```

sha1sum

```
sha1sum [kapcsolók] [fájlok]
```

Sha1 ellenőrző összeg számítása fájlokhoz. Segítségével ellenőrizhető egy fájl épsége. Kapcsolói és használata megegyezik az md5sum parancséval.

Fájlok tartalmi kezelése

cmp

`cmp [kapcsolók] [fájl1] [fájl2]`

Fájlok összehasonlítása tartalom szerint. Alapértelmezetten kiírja, hogy hány bájt eltérés van a két fájl között. Kapcsolói:

- *-b*
  Kiírja az eltérő bájtokat is.
- *-i*
  Kihagyja az első x bájtot az összehasonlításból. Ha mindkét fájl esetén ki akarunk hagyni x és y byte-ot az összehasonlításból, akkor a két számot kettősponttal kell elválasztani.

cat

`cat [kapcsolók] [fájlok]`

Fájlok összefűzése és kiírása a képernyőre. Használható kapcsolói:

- *-n*
  Sorok számozása (forráskódhoz ideális).
- *-A*
  Speciális vezérlőkaraktereket is mutatja.
- *-b*
  Csak a nem üres sorok sorszámozása (felülírja a -n kapcsolót).
- *-T*
  Tabulátor karakterek mutatása.

cut

`cut [kapcsolók] [fájlok]`

Megadott paraméterek alapján fájlokból információ kivágása, kinyerése. Kapcsolói:

- *-c*
  Oszlop kinyerése egy fájlból. Az oszlop szélességét kezdő és végkarakterrel kell megadni, tehát a -c 1-10 egy olyan oszlopot jelöl, amely az első karakternél kezdődik és a 10. karakternél végződik.
- *-b*
  Byte-ok meghatározása. Hasonló a -c paraméterhez, csak itt az oszlop szélességet byte-ban határozza meg.
- *-d*
  Elválasztó karakter meghatározása az alapértelmezett tab helyett.

diff

`diff [kapcsolók] [fájl1] [fájl2]`

Két fájl közötti eltérést mutat meg soronként. Kapcsolói:

- *-q*

Jelentést ad, ha a két fájl különbözik

- *-s*
  Jelentést ad, ha a két fájl azonos.
- *-y*
  Oszlopos megjelenítés, a két fájlt két különböző oszlopban jeleníti meg.
- *-W*
  Megadja oszlopos megjelenítés esetén, hogy a az oszlopok milyen szélesek (mennyi karakter) legyenek.
- *-i*
  Kis és nagybetű azonosnak számít.

grep

```
grep [kapcsolók] [minta] [fájlok]
grep [kapcsolók] -e [minta] [fájlok]
```

A megadott fájlban olyan sorokat keres, amelyek illeszkednek a kapcsoló által megadott mintára, majd ezeket kiírja a képernyőre. A minták megadása reguláris kifejezésekkel történik. A reguláris kifejezések olyan szöveg részletek, amelyek illeszkedési mintát definiálnak. A reguláris kifejezések jelölőkészlete az alábbi táblázatban olvasható.

Karakter	***Leírás***	***Példa***
^	*Sor kezdete*	*^a egy a betűvel kezdődő sort jelöl.*
$	*Sor vége*	*z$ egy z betűvel végződő sort jelöl.*
?	*0 vagy 1 előfordulás az őt megelőző karakterre, mintarészletre*	*ab? értéke lehet a és ab.*
***	*0 vagy több előfordulás az őt megelőző karakterre, mintarészletre*	*ab* értéke lehet a, ab, abb, abbb, stb...*
+	*1 vagy több előfordulás*	*ab+ értéke lehet ab, abb, abbb, stb...*
[...]	*Alternatív karakterek*	*zorro[12] értéke lehet zorro1 vagy zorro2.*
- (szögletes zárójelen belül)	*Értékek sorozata*	*[a-z] értéke az angol ABC kisbetűit jelöli.*
^ (szögletes zárójelen belül)	*Utána következő karakteren kívül bármit jelölhet*	*[^a-z] értéke lehet b, c, d, e, stb...*
|	*Alternatív minták*	*sör|bor értéke lehet sör vagy bor.*

85. Táblázat: A Grep mintaillesztő karakterei

Karakter	***Leírás***	***Példa***
{min,max}	*Az előfordulások legkisebb és legnagyobb száma. Ha min és max nincs megadva, akkor 0 és végtelennel dolgozik.*	*a{1,3} értéke a, aa vagy aaa lehet. a{,3} értelmezhetetlen, ezért nincs értéke a{0,} értéke lehet a, aa, aaa, stb..*
(...)	*Részminták*	*(a)(b) értéke ab, de két részmintából áll.*
.	*bármely karaktert jelölheti*	
**	*Védőkarakter. A speciális mintadefiniáló karakterek mintába illesztésére szolgál.*	*A . karakter a mintába \. írásával illeszthető.*

86. Táblázat: A Grep mintaillesztő karakterei (folytatás)

Kapcsolói:

- *-i*
  Kis és nagybetű nem különbözik.
- *-f [fájl]*
  Minták olvasása megadott fájlból. Egy sorban csak egy minta szerepelhet.
- *-v*
  Azon sorokat listázza, amelyekre nem illeszkedik a minta.

head

```
head [kapcsolók] [fájlnév]
```

A megadott fájl első sorait/karaktereit listázza ki. Ha paraméterek nélkül futtatjuk, akkor a fájl első 10 sorát írja ki. Kapcsolói:

- *-c [karakterszám]*
  Megadott számú karaktert ír ki a fájl elejéről.
- *-n [sorok]*
  Megadott számú sort ír ki a fájl elejéről.

tail

```
tail [kapcsolók] [fájlnév]
```

A megadott fájl első/utolsó karaktereit listázza ki. Ha paraméterek nélkül futtatjuk, akkor a fájl utolsó 10 sorát írja ki. Kapcsolói megegyeznek a head parancséval.

less

```
less [fájlnév]
```

Szövegmegjelenítő. A szövegben a fel-le, page up és page down gombokkal mozoghatunk. A megjelenítésből a q gomb segítségével tudunk kilépni.

wc

```
wc [kapcsolók] [fájl]
```

A bemeneti fájl szó, karakter és üres sor számainak megadására használható, kapcsolóktól függően. Kapcsolói:

- *-c*
  Bájtok számolása.
- *-m*
  Karakterek számolása.
- *-l*
  Sorok számolása.
- *-w*
  Szavak számolása.

sort

```
sort [kapcsolók] [fájl]
```

Bemeneti fájl sorainak rendezése. Kapcsolói:

- *-i*

  Kis és nagybetű nem különbözik.
- *-r*

  Fordított sorrend.
- *-R*

  Véletlenszerű sorrend.

Rendszer-adminisztrációs parancsok

who

Kilistázza a bejelentkezett felhasználókat, az általuk használt terminálokat és bejelentkezési idejüket. Kapcsolói:

- *-b*
  Csak az utolsó rendszerindítás időpontját jeleníti meg.
- *-q*
  Belépett felhasználók neve és az összes bejelentkezett felhasználó darabszámának megjelenítése.

whoami

Megjeleníti az aktuálisan belépett felhasználó nevét.

users

Megjeleníti az aktuálisan bejelentkezett felhasználók listáját. Kevesebb információt ad, mint a who parancs.

useradd

```
useradd [kapcsolók] [felhasználónév]
```

Hozzáad egy felhasználót a rendszerhez. Kapcsolói:

- *-d vagy –home*
  Felhasználó home könyvtárának útvonala. Nem kötelező megadni, mivel létrejön az alapértelmezett útvonalon a felhasználónév alapján. Ha meg van adva, akkor az alapértelmezett helyett fogja használni a rendszer a könyvtárat. Ha ez a könyvtár nem létezik, létre kell hozni manuálisan.
- *-e vagy –expiredate*
  Fiók lejárati ideje. A megadott dátum után nem lesz használható a fiók. A dátumot ÉÉÉÉ-HH-NN formátumban (Év, hó, nap) kell megadni.
- *-g vagy –gid*
  Felhasználó csoportjának megadása név vagy ID alapján. A csoportnak létező csoportnak kell lennie.
- *-u vagy -uid*
  Felhasználói azonosító ID megadása.
- *-s vagy –shell*
  Alapértelmezett parancsértelmező megadása. Itt a parancsértelmező elérését kell megadni. pl. /bin/bash

userdel

```
userdel [kapcsolók] [felhasználónév]
```

Felhasználó eltávolítása, törlése a rendszerből. Kapcsolói:

- *-f vagy –force*
  Erőszakos eltávolítás. Akkor is eltávolítja a felhasználót, ha éppen be van jelentkezve.
- *-r vagy –remove*
  Felhasználó könyvtárában tárolt fájlok törlése.

usermod

```
usermod [kapcsolók] [felhasználónév]
```

Felhasználói fiók módosítása. Kapcsolói megegyeznek a useradd parancséval. Továbbá usermod esetén az alábbi kapcsolók használhatók még:

- *-l vagy –login*
  Bejelentkezési név módosítása.
- *-L vagy –lock*
  Fiók zárolása. A fiókkal nem lehet bejelentkezni a rendszerbe.
- *-U vagy –unlock*
  Zárolás feloldása.

groupadd

```
groupadd [kapcsolók] [csoportnév]
```

Új csoport létrehozása. Kapcsolói:

- *-g vagy –gid*
  Csoport ID megadása. A 0 és 999 közötti értékek fenntartottak a rendszer számára. Ezért érdemes 1000 feletti értéket megadni.
- *-f vagy –force*
  Erőltetett létrehozás. Abban az esetben is lefut a parancs, ha az adott csoport már rendelkezik. Ha csoport ID ütközés van egy másik csoporttal (-g vagy –gid paraméter meg van adva), akkor a csoport ID véletlenszerűen lesz kiválasztva.

groupmod

Létező csoport módosítása. Opciói megegyeznek a groupadd parancséval. Továbbá használható a következő kapcsoló is:

- *-n vagy –new-name*
  Csoport átnevezése.

groupdel

```
groupdel [csoport]
```

Létező csoport törlése a rendszerből. Csak olyan csoport törölhető, amely egy felhasználónál sem szerepel elsődleges csoportként. Ha mégis lenne olyan felhasználó, akinek az eltávolítandó csoport az elsődleges csoportja, akkor a felhasználót vagy törölni kell, vagy az elsődleges csoportját meg kell változtatni.

passwd

Felhasználó jelszavának megváltoztatása.

sudo

```
sudo [panarncs]
```

Parancs futtatása rendszergazdaként. Csak olyan felhasználó adhatja ki ezt a parancsot, aki a /etc/sudoers konfigurációs fájlban szerepel csoportja vagy felhasználóneve alapján. Debian alapú rendszereken az adminisztrátorok (administrators) csoport tagság szükséges a használatához.

su

```
su [felhasználó]
```

Shell indítása más felhasználónévvel. Amennyiben nem adunk meg felhasználónevet, akkor a root felhasználót fogja használni. Kapcsolói:

- *-c vagy –command*
  Csak a megadott parancsot fogja futtatni a felhasználó jogosultságaival. A parancs után a parancsértelmező kilép.
- *-s vagy –shell*
  Megadott shell használata felhasználó alapértelmezettje helyett.
- *-m*
  Az aktuális elérési útvonalon marad, nem módosítja az elérési útvonalat a felhasználó alapértelmezettjére.

ps

Futtatott programok megjelenítése. Alapértelmezetten csak az általunk futtatott parancsokat írja ki. Azonban ax paraméterrel minden folyamatot megjelenít.

pstree

A futó folyamatok megjelenítése fa szerkezetben. Ilyen nézetben látszik, hogy melyik folyamatnak melyik a szülő folyamata.

nice

```
nice [kapcsoló] [folyamatnév]
```

Folyamat prioritásának módosítása. Kapcsolói:

- *-n*
  Prioritási szint megadása.

kill

`kill [pid]`

Futó folyamat leállítása a PID értéke alapján. a PID értéket a futó folyamatok listájából lehet megtudni.

shutdown

`shutdown [kapcsolók] [idő]`

A rendszer leállítása adott időben. Csak rendszergazda teheti meg. Idő megadásának formátumai:

- *+m*

  Parancs kiadásától számítja az idő elteltét. A +m kapcsoló után megadott számot percként értelmezi. Ezen idő eltelte után fogja végrehajtani a rendszerleállítást.
- *now*

  Parancs kiadásának pillanatában elkezdi végrehajtani a leállítást.
- *hh:mm*

  Óra:perc alapú idő megadás. Az idő 24 órás intervallumban értelmeződik.

Kapcsolói:

- *-r*

  Leállítás helyett újraindítás.
- *-h*

  A rendszer leállítása, majd az áram kikapcsolása, ha ez támogatott (ATX tápegységek esetén).
- *-H*

  Csak a rendszer leállítása, az áram marad. Régi, AT tápegységgel rendelkező gépek esetén használatos.

Exit

Kijelentkezés a rendszerből. Terminál emulátorok esetén a terminál bezárása.

sync

Lemezre írja a pufferelt fájlokat. Hordozható meghajtó esetén leválasztás előtt tanácsos kiadni. Máskülönben adatvesztés történhet.

mount

`mount [kapcsolók] [eszköz] [mappa]`

Rendszerbe csatlakoztat egy fájlrendszert. Linux alatt a hordozható meghajtók (CD-ROM, Pendrive, srb...) automatikusan nem csatolódnak. Ezeket kézzel a mount parancs segítségével kell a rendszerhez csatolni. Az eszköz az egy /dev mappában található eszköz leíró. DVD-rom meghatjtó esetén pl: /dev/dvdrom.
A mappa pedig egy útvonal, ahova csatolni szeretnénk az eszközt. Pl: /media/dvd. Grafikus felülettel rendelkező rendszerek rendelkeznek automatikus mount képességgel, így a mount csak parancssoros és egyéb egzotikus esetekben kellhet. Csak root felhasználó adhatja ki. Kapcsolói:

- *-a*
  Csatlakoztat minden olyan fájlrendszert, ami szerepel a /etc/fstab fájlban (rendszer indításkor is ez zajlik le.). Ebben az esetben nem kell eszközt és mapát megadni.
- *-r*
  Csatolás csak olvasható módban.
- *-t [fájlrendszer]*
  Fájlrendszer meghatározása. Értelme akkor van, ha az eszköz, amit csatlakoztatni akarunk egy lemezkép, vagy a mount automatán nem ismerte fel a partíciót. Kerneltől függően eltérő a támogatott fájlrendszerek típusa. A man mount megfelelő szekciója kiírja a támogatott fájlrendszereket.
- *-l*
  Kilistázza a csatlakoztatott fájlrendszereket. Ebben az esetben nem kell eszközt és mappát megadni.

umount

`umount [kapcsolók] [mappa]`

Leválaszt egy csatlakoztatott fájlrendszert. Paraméternek a csatlakoztatási pontot kell megadni. Kapcsolói:

- *-a*
  Minden csatolt fájlrendszert leválaszt. Ez esetben leválasztandó csatolási pontot nem kell megadni.
  -f
  Erőszakos leválasztás. Adatvesztéssel járhat. Egy fájlrendszer általában akkor nem választható le, ha éppen lemezművelet van folyamatban.
- *-l*
  Késleltetett leválasztás. Csak akkor válassza le a fájlrendszert, ha tényleges lemezművelet nincs felé.

Szoftverek telepítése apt-get segítségével

Az apt-get debian alapú rendszerek internetes csomagkezelő és telepítő rendszere. Az összes debian alapú disztribúcióban megtalálható. Futtatása csak rendszergazda jogosultságokkal lehetséges. Használata:

```
apt-get [művelet] [csomag neve]
```

Használható műveletek:

- *install*
  Megadott nevű csomag telepítése.
- *remove*
  Megadott nevű csomag eltávolítása.
- *update*
  Elérhető csomagok listájának frissítése.
- *upgrade*
  Elérhető frissítések telepítése.
- *dist-upgrade*
  Disztribúció frissítése.
- *clean*
  Letöltött csomagok törlése. Ez nem jár a csomag eltávolításával. A programnak telepítés előtt le kell töltenie a csomagot.

Szoftverek telepítése forráskódból

A programok forráskódból való telepítésére univerzális leírás csak igen nagy vonalakban adható. Ezért az ilyen telepítés előtt érdemes elolvasni a mellékelt olvass el (readme) fájlt, mivel abban részletes információk találhatóak a tényleges telepítés menetéről.

Amire biztos szükségünk lesz, az a fordítóprogram, amivel lefordítjuk a programot. Ez a fordító legtöbb esetben egy C és/vagy C++ fordítót jelent, mivel a legtöbb szoftvert ezeken a nyelveken írják. A C/C++ fordító alapértelmezetten nem része a legtöbb debian alapú telepítésnek, úgyhogy a művelet előtt apt-get segítségével fel kell tenni őket a rendszerre. Erre a következő parancsot használhatjuk:

```
apt-get install gcc g++ build-essential
```

Ez a parancs telepíteni fogja nekünk a gcc C fordítót és a G++ C++ fordítót, valamint a fordításhoz szükséges extra eszközöket is. Ezután következhet a forráskód beszerzése, majd kicsomagolása. Ha ezzel megvolnánk, akkor egy terminál segítségével abba a mappába kell navigálnunk, ahova kicsomagoltuk a forráskódot. Ezután a következő parancsokat kell kiadni:

```
./configure
make
make install
```

A ./configure parancs a forráskód konfigurálására szolgál. Ez állítja be, hogy milyen rendszerrel dolgozunk. Az ezután következő make parancs végzi a tényleges fordítást. Ez a gép sebességétől és a forrás komplexitásától függően eltarthat egy ideig, úgyhogy ez az idő alkalmas egy kávé vagy cigi szünet beiktatására.

A make install parancs telepíti ténylegesen a lefordított bináris fájlokat. Ehhez rendszergazda jogosultság kell a legtöbb esetben.

Előfordulhat, hogy a lefordítani kívánt programunknak szüksége van programkönyvtárakra is. Ebben az esetben vagy az apt-get segítségével beszerezzük a programkönyvtárak fejlesztői verzióját (forráskódját), vagy ezeket is manuálisan lefordítjuk mi magunk. Így egy komolyabb program lefordítása fél napot is igénybe vehet első alkalommal, és akkor sem garantált, hogy működni fog elsőre a program.

Ennek oka az, hogy a konfigurálás során általában megadhatunk paramétereket. És ha nem a megfelelő paraméterekkel fordítunk egy programot, vagy valamelyik programkönyvtár hiányzik, akkor belefuthatunk egy fordítási hibába. Rosszabb esetben működni fog a szoftver, de nem úgy, ahogy kellene neki.

18. RaspberryPi beüzemelése

A RaspberryPi beüzemeléséhez először is szükségünk lesz pár eszközre magán a RaspberryPi lapon kívül. Ezen eszközök:

- *MicroUSB töltő, legalább 1A terhelhetőséggel, vagy 5W teljesítménnyel*
- *SD Kártya, legalább 4Gb kapacitással*
- *USB SD kártyaolvasó, ha a számítógépben nem lenne beépített kártyaolvasó*
- *UTP kábel*

Ha nem hálózaton keresztül szemeljük be a lapot, akkor szükségünk lesz még:

- *HDMI csatolóval ellátott monitor HDMI kábellel*
  Ha ez nincs, akkor DVI bemenetes monitor is megteszi HDMI-DVI átalakítóval, esetlegesen egy TV is szóba jöhet kompozit bemenettel (RCA)
- *USB billentyűzet és egér*

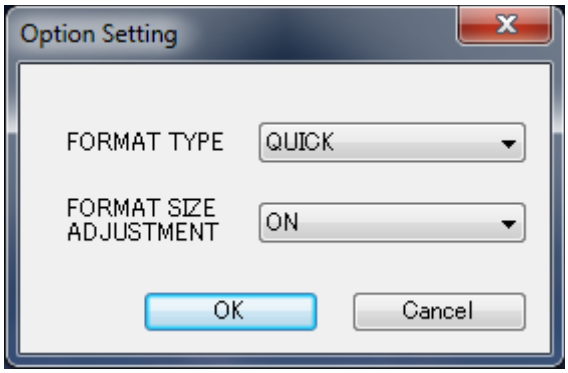
Hálózati beüzemeléshez szükségünk lesz még egy SSH kliens programra is, mint a KiTTY. A program beszerezhető a http://kitty.9bis.net/ címről.

Mindkét úton történő beüzemelés első lépéseként meg kell formázni a memória kártyát. A memóriakártya formázásához a Windows beépített formázó eszköze nem lesz jó, mivel az SD kártyák fájlrendszerének felépítése egyedi. Ha nem a megfelelő eszközzel formázzuk meg a kártyát, akkor az élettartalma rövidülhet, valamint az elérhető sebesség is csökkenhet.

Kifejezetten SD kártyák számára készült formázó eszközt a https://www.sdcard.org/downloads/formatter_4/ címről tudunk letölteni. Ezt a programot a hivatalos SD kártya gyártó egyesület fejleszti Windows és Mac OS rendszerekre.

Linux esetén a GParted particionáló program segítségével lehetséges a kártya formázása. A GParted a http://gparted.sourceforge.net/download.php címről szerezhető be.

A kártya formázásakor be kell állítani a FORMAT SIZE ADJUSTMENT opciót, máskülönben nem lesz indítható a rendszer a kártyáról.



A kártya előkészítése után szükségünk lesz egy operációs rendszerre. A Pi esetén jópár operációs rendszer szóba jöhet telepítésre. Ezeket a http://www.raspberrypi.org/downloads címről lehet letölteni.

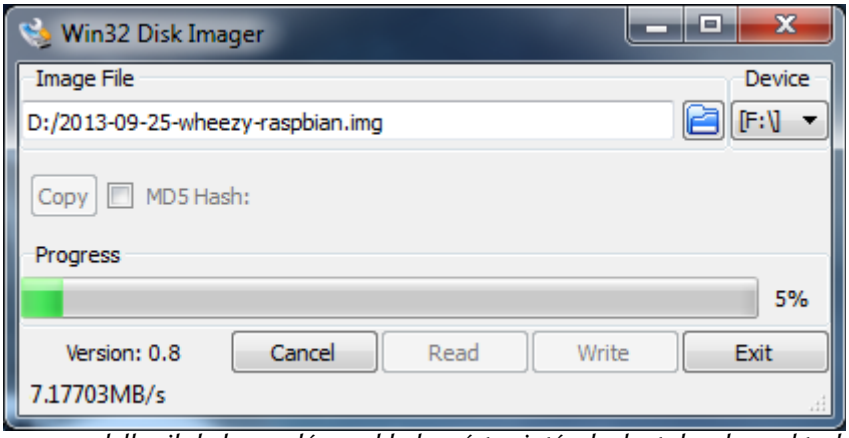
A készítők ajánlása a NOOBS program, amely lényegében egy operációs rendszer telepítő. A programot letöltve, majd az SD kártyára másolva a PI első indításakor kiválasztható, hogy milyen operációs rendszert szeretnénk telepíteni és használni.

Az operációs rendszerek közül a könyvben használt a Raspbian rendszer, mivel a rendszerek közül ez a leginkább támogatott.

Az egyes operációs rendszerek lemezképei manuálisan is letölthetőek a fentebb említett címről. A telepítés ebben az esetben sem bonyolult. A letöltött lemezképet Windows alatt a Win32 Disk Imager programmal tudjuk a kártyára varázsolni, Linux alatt pedig a DD parancs segítségével. A Win32 Disk Imager a http://sourceforge.net/projects/win32diskimager/ címről szerezhető be.

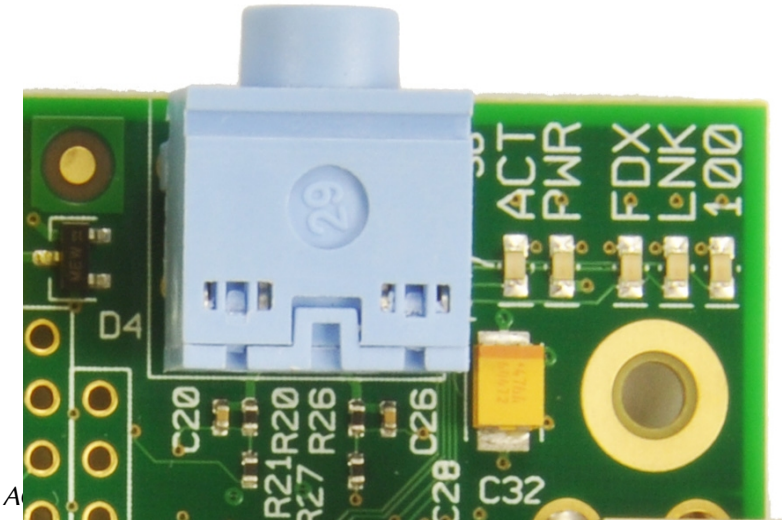
A letöltött képek ZIP vagy más tömörített formátumban érhetőek el, lemezre írás előtt ki kell őket csomagolni. Erre érdemes valami tömörítő programot használni, mint a 7zip, amely teljesen ingyenes és nyílt forrású. Beszerezhető a http://www.7-zip.org/ címről.

A Win32 Disk Imager használata pofon egyszerű. A letöltött lemezképet kell neki megadni, illetve azt, hogy melyik meghajtóra szeretnénk írni azt. Ezután meg kell nyomni a Write gombot és a folyamat végeztével már indítható is a RaspberryPi.



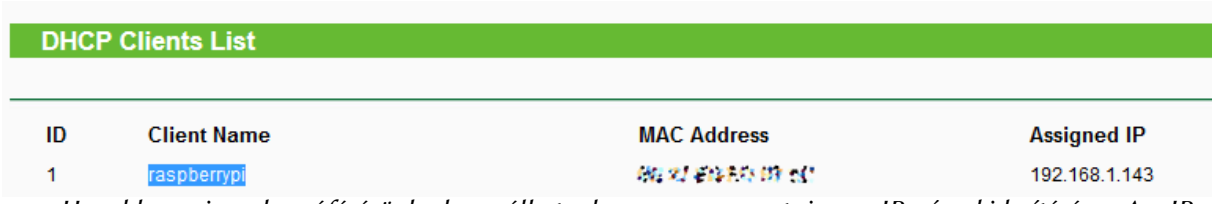
Mivel a Pi nem rendelkezik bekapcsoló gombbal, ezért miután bedugtuk a konnektorba automatikusan el fog indulni a rendszer betöltése. A kábeleket és perifériákat ezért a tápkábel csatlakoztatása előtt végezzük el.

Ha minden rendben van a betöltés során, akkor két LED-től kellene visszajelzést kapnunk. A PWR jelzésű LED-nek pirosan kell világítania, míg az ACT jelzésűnek zölden villognia kártyaolvasás közben.



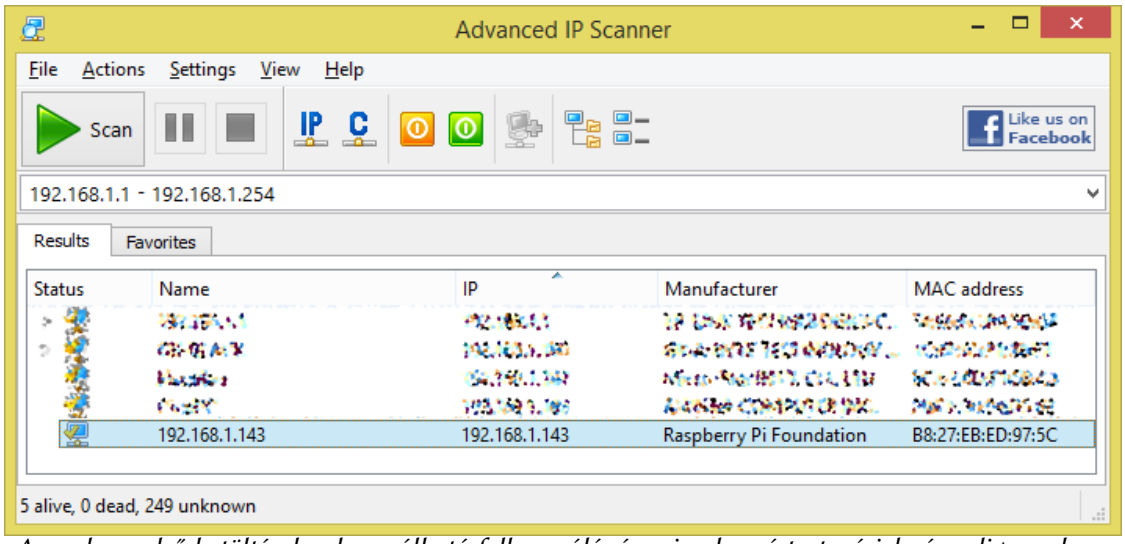
Amennyiben az A az SD kártyával. Ez lehet érintkezési gond is, de elképzelhető, hogy a kártya amit használunk nem kompatibilis a lappal. A támogatott és nem támogatott kártyák típusáról a http://elinux.org/RPi_SD_cards címen tájékozódhatunk.

Ha monitort kötöttünk a lapra, akkor egy rövid idő után a rendszernek haladnia kellene a betöltéssel, majd eljut arra a pontra, hogy bejelentkezhetünk. Monitor nélküli használat esetén az SSH szervernek kevesebb, mint 1 perc alatt fel kellene élednie, azonban ehhez kell tudnunk a Pi IP címét, amit a router konfigurációs menüjéből megtudhatunk.



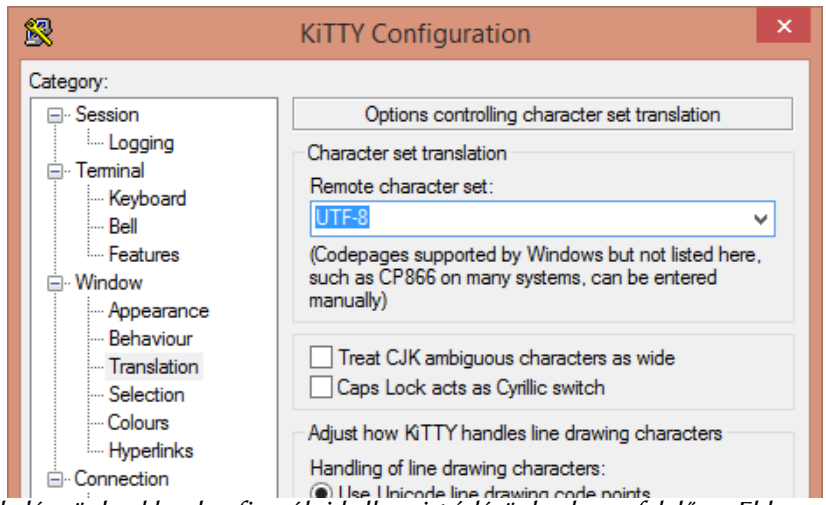
Ha ehhez nincs hozzáférésünk, használhatunk egy programot is az IP cím kiderítésére. Az IP cím kiderítésére alkalmas szoftver például a Famatech Advanced IP Scanner, amely a http://www.radmin.hu/download/utilities.php címről beszerezhető.

A program telepítést nem igényel, elindítása után csak a SCAN feliratú gombra kell kattintanunk. Ez megkeresi az összes elérhető számítógépet a helyi hálózaton. Számukra a Manufacturer oszlop lesz a fontos. A Pi esetén a gyártó a Raspberry Pi Foundation lesz. Az IP oszlopban található cím pedig a lap IP címe.



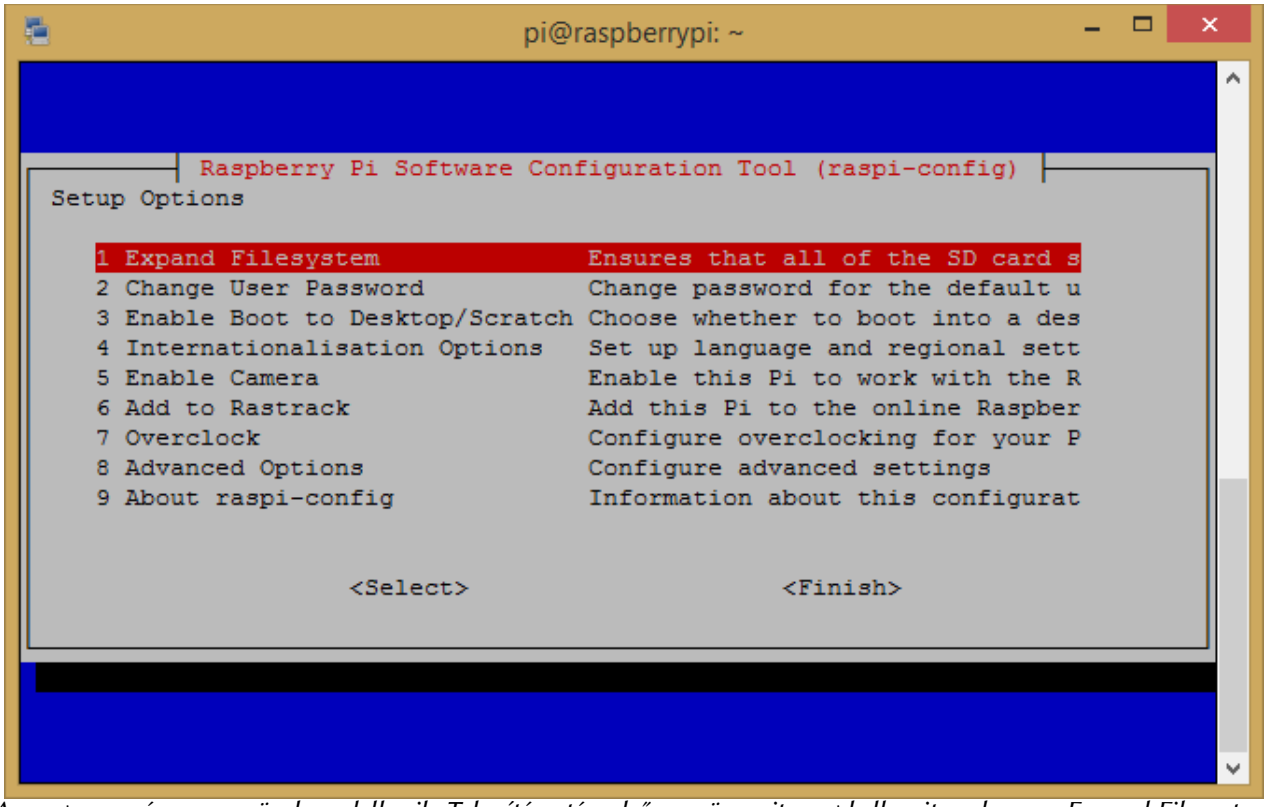
A rendszer első betöltésekor használható felhasználónév: pi, a hozzá tartozó jelszó pedig raspberry. A root felhasználóként bejelentkezés tiltott, rendszergazdaként parancsok a sudo parancs segítségével futtathatóak.

SSH csatlakozás előtt a PuTTY vagy KiTTY programban a beállítások között át kell állítanunk a karakterkészletet UTF-8 típusúra. Erre azért van szükség, hogy az átvitel során minden karakter megfelelően jelenjen meg. A beállítás a Window kategória Translation lapján jelenik meg.



Ha sikerült belépnünk, akkor konfigurálni kell a pi-t ízlésünknek megfelelően. Ehhez a rendszer beépítetten tartalmaz egy konfiguráló eszközt, amelyet az alábbi parancs kiadásával tudunk elindítani:

```
sudo raspi-config
```



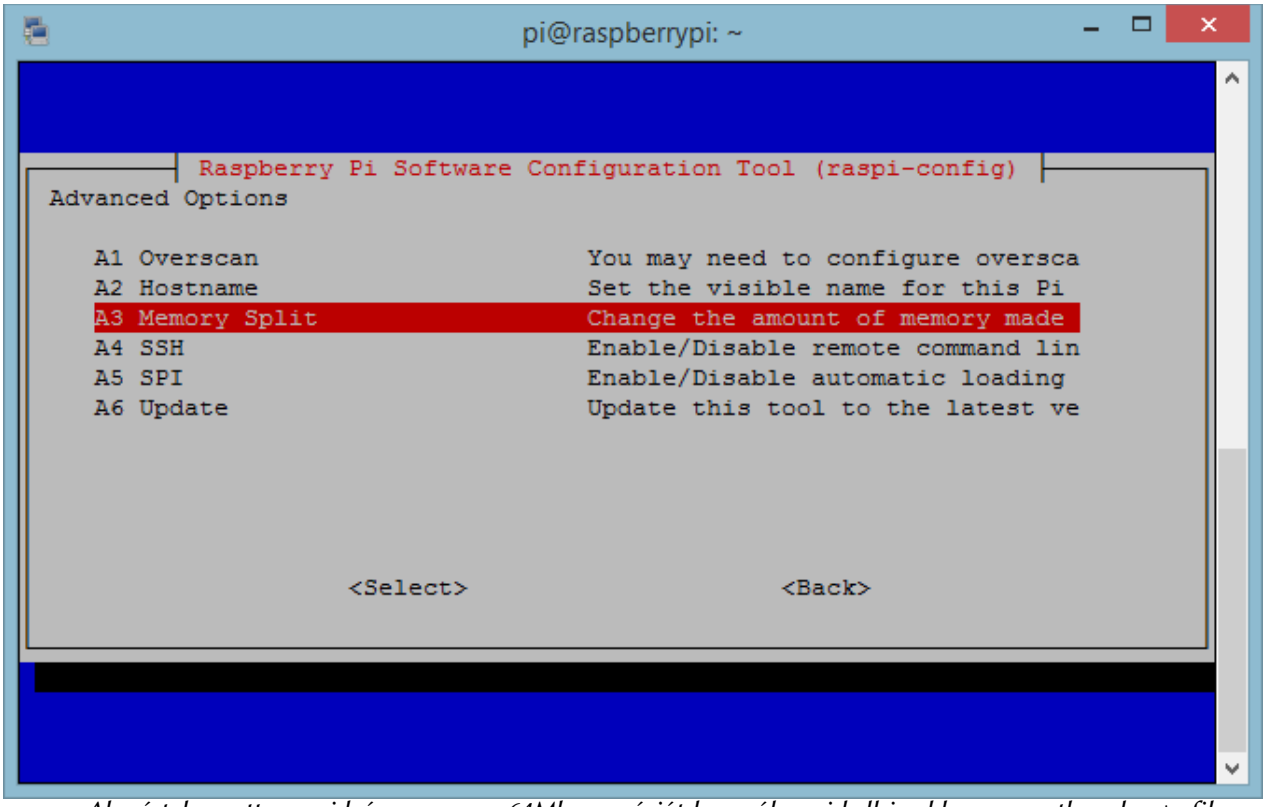
A program számos menüvel rendelkezik. Telepítés után első menü, amit meg kell nyitnunk az az Expand Filesystem menüpont, amely átméretezi a kártyára másolt partíciót a kártya teljes méretére.

Ezután, ha fizikai billentyűzetet alkalmazunk, akkor a nyelvi és területi beállításokat kell elvégezni az Internationalisation Options menüponton belül. Itt többek között az Időzónát is be tudjuk állítani. A pontos idő megtudásához hálózatra kell csatlakoznia a rendszernek, mivel a Pi nem rendelkezik valós idejű órával.

Amennyiben hálózaton keresztül elérhetővé tesszük a Pi-t, akkor érdemes (és amúgy is) megváltoztatni az alap jelszót. Ezt a Change User Password menüben tehetjük meg.

Az Advanced Options menüponton belül a következők beállítására van lehetőségünk: A lap host neve, az SSH beállítása, HDMI képhez hibaelhárító túlpásztázás, valamint a memória megosztása.

Ezen menüpontokból a memória megosztása a lényeges számunkra.



Alapértelmezetten a videó processzor 64Mb memóriát használ, ami kell is abban az esetben, ha grafikusan használjuk a rendszert. Azonban ha csak hálózaton keresztül lesz használva a lap, akkor felesleges ennyi memóriát adni neki, hiszen nem lesz kihasználva. Egészen 16Mb-ig probléma mentesen levihetjük a memória használatát ebben az esetben.

A beállítások végeztével a Finish gombra navigálva kiléphetünk a programból, ami menteni fogja a beállításainkat. Elvileg meg kellene kérdeznie, hogy újraindítunk e a változások életbelépéséhez. Amennyiben nem teszi ezt meg, akkor a következő parancs segítségével újraindíthatjuk a lapot:

```
sudo shutdown -r now
```

19. Programozható logikai eszközök

A programozható logikai eszközök története nagyjából egyidős a digitális integrált áramkörökkel. Körülbelül a TTL áramkörök megjelenésének idején már felmerültek igények egyedi logikai függvényeket megvalósító integrált áramkörök iránt. Ilyen áramkörök tervezése abban az időben igencsak drága volt, hiszen a gyártónak sok esetben nulláról kellett megterveznie az áramkört. Ezért az egyedi megoldások abban az időben nem igen voltak jellemzőek, inkább diszkrét elemekből valósították meg az áramköröket.

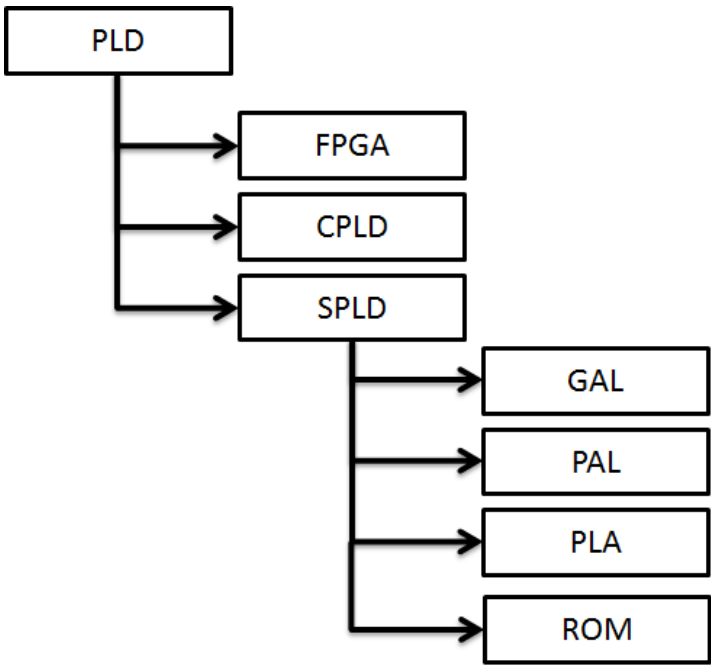
Az egyedi gyártású, speciálisan egy feladat elvégzésére kialakított integrált áramköröket nevezzük ASIC megoldásoknak. Az ASIC mozaikszó az angol Application Specific Integrated Circuit szavak rövidítése.

A programozható logikai áramkörök iránti igény a mikroprocesszorok megjelenésével ismét erősödött, így a gyártók elkezdtek kísérletezni ilyen áramkörök gyártásával, köszönhetően a fejlettebb anyag technológiának.

Az évek során számos programozható logikai eszközt fejlesztettek ki és gyártottak. Ezek mára teljesen felváltották a diszkrét áramköri kapukat nagyobb projektek során. Ennek ok az, ha egy kapcsolás megvalósítása 30db diszkrét logikai elemet igényelne, akkor jóval olcsóbb egy drágább programozható áramkört alkalmazni, mint 30db áramkört és hozzá egy nagyobb méretű panelt.

Programozható logikai eszköz kategória alatt nem a mikrovezérlők és mikroprocesszorok értendőek, hanem olyan eszközök, amelyek architektúrája szabadon alakítható, vagyis magát a hardvert programozzuk, nem a szoftvert, ami a hardveren fut.

A programozás megvalósulása eszköz típus függő. Mielőtt ténylegesen rátérnék egy konkrét típus programozására érdemes áttekinteni a programozható logikai eszközök családfáját. A programozható logikai áramköröket az angol PLD kifejezéssel szokták jelölni.



Egyszerű programozható eszközök (SPLD)

Ezen áramkörcsalád tagjai a klasszikus kétszintes hálózatok megvalósítására lettek kifejlesztve. Nagyjából 1000db logikai kapuig képesek hálózatok megvalósítására.

ROM

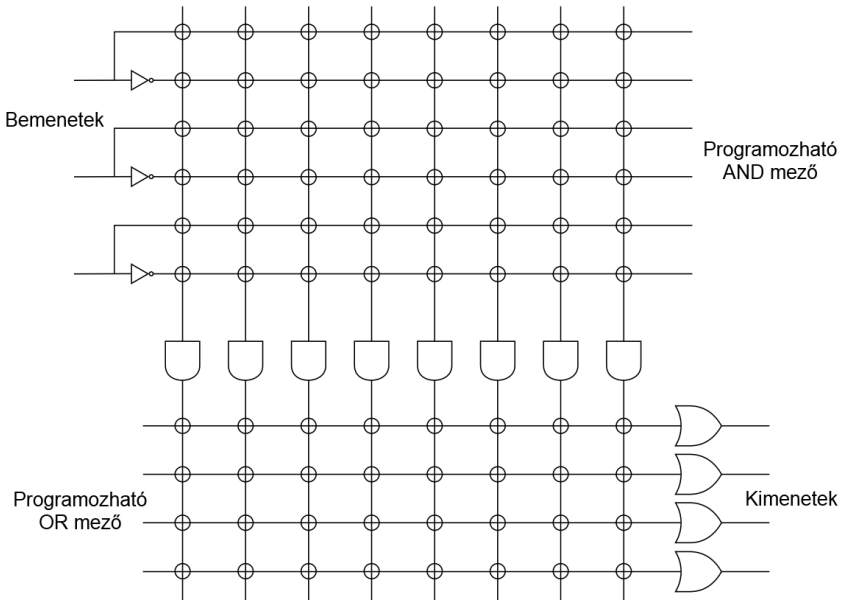
Egy ROM áramkör felfogható programozható logikai eszközként is. Egy párhuzamos kialakítású 8Kb-os 8 bites szervezésű ROM memória felfogható egy 13 bemenetes, 8 kimenettel rendelkező kombinációs hálózatként. Ez annak köszönhető, hogy egy ROM kimenetén minden esetben a címbitek által meghatározott rekesz kimenete jelenik meg. Ha programozható logikai eszközként akarunk egy ilyen áramkört használni, akkor a függvény igazság tábláját kell a memóriában tárolni. Hátránya ennek a megoldásnak, hogy a függvényt nem tudjuk egyszerűsíteni és hazárd mentesíteni, valamint az EPROM és EEPROM memóriák sebessége korlátolt.

A hazárd mentesíthetőség hiánya miatt és a viszonylag lassú működés miatt nem igen terjedtek el ilyen célra ezen áramkörök.

PLA

A PLA kifejezés a Programable array logic kifejezésből jön, ami programozható logikai tömböt jelent. Ezen áramkörök esetén az AND és OR kapuk hálózata is programozható, ami lehetővé teszi bármilyen nem időfüggő függvény megvalósítását. Hátránya volt ezen áramköröknek, hogy viszonylag drágán tudták őket előállítani, valamint működési sebességük sem volt túl gyors, mivel a jelnek két programozható mátrixon is át kellett haladnia. Előnyük viszont az, hogy a függvény hazárd mentesíthető, egyszerűsíthető, valamint könnyebben implementálhatóak a közömbös kombinációk.

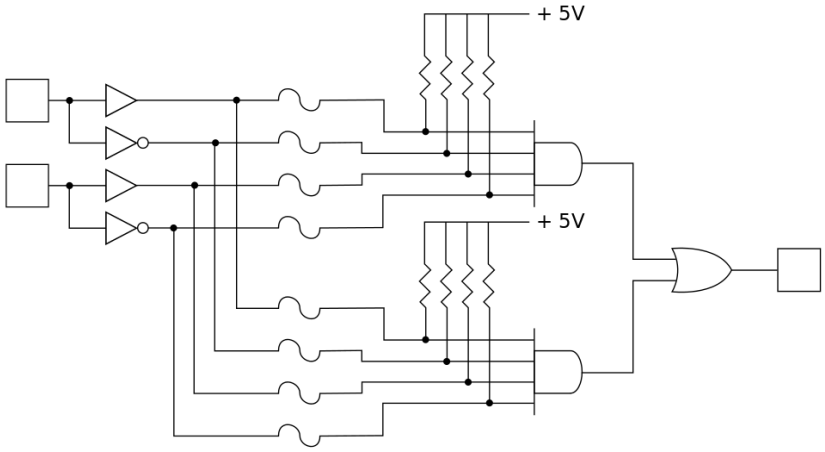
További hátrány volt a PLA áramkörök esetén, hogy sok esetben nem lehetett őket újraprogramozni, a gyártás során programozták őket, mint a kezdetleges ROM áramköröket. Így viszonylag csak nagy tételben volt érdemes alkalmazni őket.



PAL

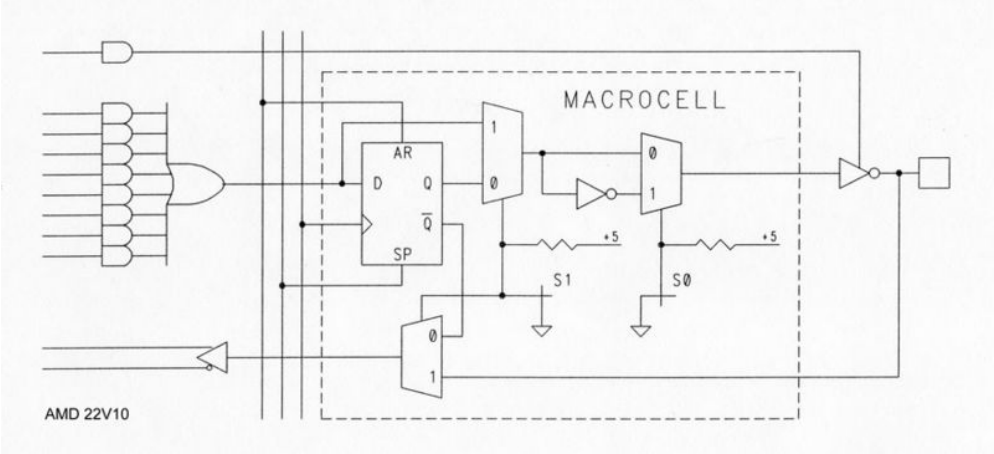
A PAL szó a Programable Array Logic kifjezés rövidítése, ami programozható tömb logikát jelent. Ezen áramkörök 1978-ban jelentek meg, a ma már nem létező Monolithic Memories fejlesztette ki ezen áramköröket. A PLA áramköröktől kezdetben annyival tértek el, hogy az AND mező fixen programozott volt és az OR mező volt csak programozható. Ez lehetővé tette nagyobb sebesség használatát, mivel a jelnek csak egy programozott mezőn kellett áthaladnia, hátrány volt azonban, hogy nehezebb volt őket programozni ezért. Ezt fejlett fordítóprogramokkal oldották meg.

További előnye ezen eszközöknek az volt, hogy nem a gyártás során programozták őket, hanem a vásárló egy programozó eszközzel fel tudta programozni az áramkört. Az egyszer programozhatósági hátrány sajnos itt itt is megmaradt.



GAL

A GAL kifejezés a Generic Array Logic kifjezés rövidítése, ami általános tömb logikát jelent. Ezen áramkörök a PAL áramkörök továbbfejlesztéseként születtek meg. Itt a kimeneten már makrocellák voltak alkalmazva, mint a PAL áramkörök utolsó szériáiban. A fejlesztés komoly hozadéka azonban az volt, hogy ezen áramkörök belsőleg a programot egy EEPROM memóriában tárolták, így többször újraprogramozhatóak voltak, ami lehetővé tette a gyors prototípus fejlesztést.



Komplexen programozható logikai eszközök (CPLD)

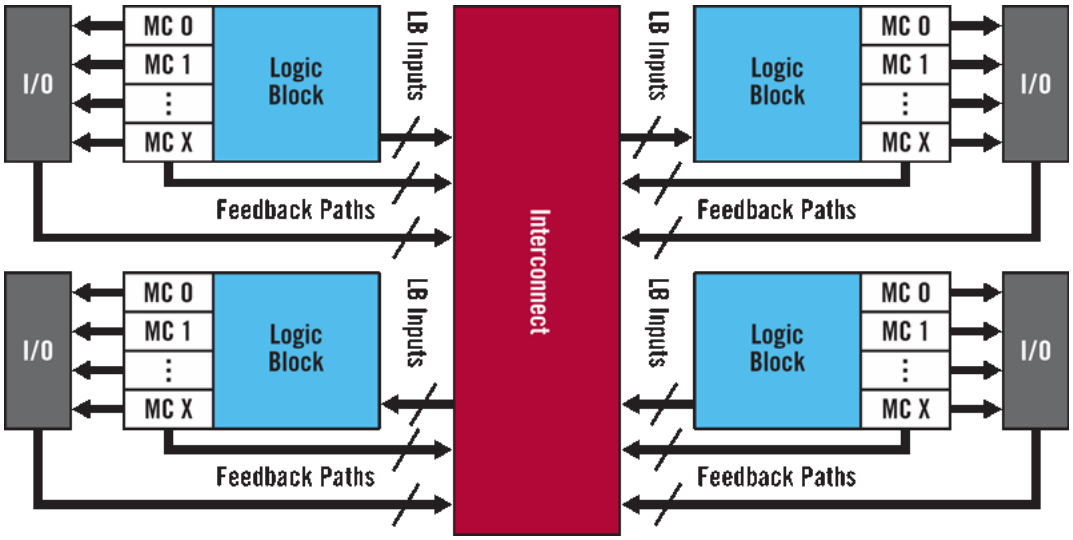
A CPLD áramkörök az 1980-as évek mádodik felében jelentek meg. Lényegében több egyszerűen programozható eszköz van egy áramkörre integrálva. A program EEPROM vagy FLASH memóriában tárolódik.

A CPLD beljesében funkció blokkok vannak, amelyekben makrocellák helyezkednek el. A kívánt kapcsolás a makrocellák programozásával érhető el. Ezek száma adja meg a CPLD kapacitását. Egyszerűbb áramkörök esetén pár százról beszélhetük, míg egy komolyabb áramkör esetén akár pár ezres nagyságrend is elérhető.

A funkció blokkok között összeköttetést egy programozható kapcsolómátrix segítségével oldják meg. Egy kimeneti egységhez tartozó logikai blokk bármelyik lába lehet kimenet és bemenet programtól függően. A kimeneti egységektől visszacsatolás van a programozható kapcsoló mátrixra, így komoly időfüggő hálótzatok megvalósítása is lehetséges.

A CPLD áramkörök nagy előnye, hogy az áramkörbe beültetés után is programozhatóak, mivel a

szabványos JTAG felületet alkalmazzák program feltöltsre és tesztelésre.



Helyileg programozható kapu tömbök (FPGA)

Az FPGA áramkörök a CPLD áramkörök továbbfejlesztéseként jöttek létre. Fő különbség egy FPGA és egy CPLD között az, hogy az FPGA áramkörök nem belsőleg tárolják a programjukat, hanem egy külső FLASH memóriában, ezért indítás után egyből nem működőképesek, viszont a program betöltése után jóval nagyobb működési sebesség érhető el, mert belsőleg a program egy RAM áramkörben tárolódik. A másik nagy különbség az, hogy az FPGA áramkörök bármelyik lába működhet ki és bemenetként is köszönhetően egy belső kapcsoló mezőnek.

Logikai cellák tekintetében egy komolyabb FPGA akár milliónál több cellát tartalmaz, amely alkalmassá teszi őket arra, hogy akár processzorokat valósítsanak meg.

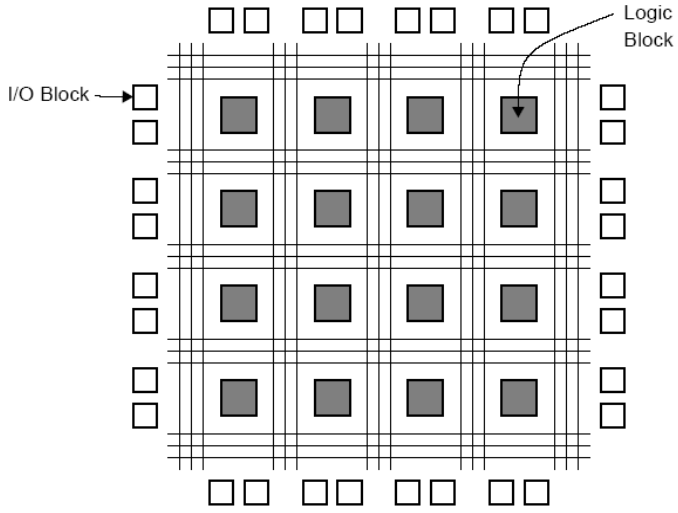
Egy FPGA továbbá rendelkezik fejlett kommunikációs protokoll támogatással is. Beleértve az I^2C, SPI és USB buszokat, így ideális kiegészítői mikrovezérlőknek és mikroprocesszoroknak.

A nagy számú funkció blokk miatt megoldhatóak párhuzamos adatfeldolgozást igénylő műveletek is, mint például képfeldolgozás, videó tömörítés, illetve egyéb egzotikus feladatok.

Hátránya ezen áramköröknek, hogy a komolyabb modellek csak BGA tokozásban érhetőek el, valamint egy FPGA fogyasztása fixen nem határozható meg, mivel a fogyasztás a használt funkcióblokkoktól függ.

A legújabb felső kategóriás FPGA áramkörök már processzorokat is tartalmaznak beépítve, amely tovább növeli az alkalmazási területüket.

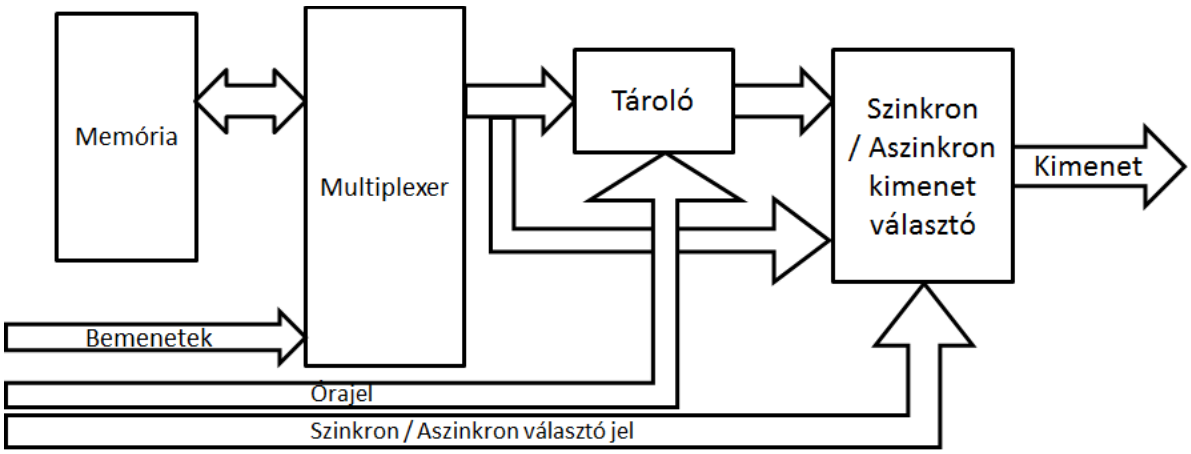
A program külső FLASH memóriában tárolása komoly előnyt jelent, mivel akár működés közben megváltoztatható a chip működése, de komoly problémát is jelent biztonságtechnikailag, mert a külső memória könnyebben módosítható, mint egy belső memória. Ezért a legtöbb mai FPGA áramkör lehetővé teszi a kód titkosítását. Vagyis a program ami a memóriába kerül már eleve titkosítva van egy olyan algoritmussal, amit csak az adott FPGA chip tud olvasni. Így kiküszöbölhetőek a rosszindulatú kód módosítások és megnehezíthető a program másolása.



Egy CPLD / FPGA cella működési felépítése

Mind az FPGA és CPLD áramkörök alapja a logikai cella, ami bármilyen logikai függvény megvalósítására képes. Felmerülhet a kérdés, hogy valójában hogyan is működik, illetve hogyan képes erre. Az egyes gyártók logikai cellái igen eltérőek lehetnek. Az itt bemutatott cella az elvi működést hivatott szemléltetni.

Minden logikai cella több részre bontható. Egy logikai cella elvi felépítése az alábbi ábrán látható.



A tetszőleges kimeneti függvény előállításáért egy multiplexer felel, amely konstrukciótól függően n választó jellel rendelkezik. Ezen választójelek vannak bemenetként használva. A multiplexer 2^n *bemenetére pedig egy memória cella kapcsolódik, ami a függvény igazságtáblázatát tárolja. Ezen két elem valósítja meg az úgynevezett Look Up Table funkciót. CPLD áramkörök esetén a memória Flash vagy EEPROM, míg FPGA chip-ek esetén SRAM vagy DRAM memória.*

A Multiplexer kimenetén előálló jel egy tárolóra van kivezetve, ami órajel vezérelt, valamint a kimenet közvetlenül becsatlakozik a szinkron/aszinkron működés között váltó elembe, ami lényegében egy kapcsoló hálózat. Ha a hozzá tartozó jel engedélyezve van, akkor a cella tényleges kimenete a tároló kimenete lesz, ha pedig nincs engedélyezve, akkor a multiplexer kimenete jelenik meg a cella kimeneten.

A CPLD és FPGA áramkörök másik fontos összetevője a kapcsolómátrix, ami lehetővé teszi azt, hogy egy cella jele tetszőleges kimeneten jelenjen meg, illetve a bemeneteket tetszőleges bemenetről vegye le. Ezen funkció megvalósításáért lényegében egy nagy kapcsolómátrix felelős.

A kapcsolómátrix állása szintén programozható, általában ez az információ ugyan úgy a Look Up Table információinak tárolására szolgáló memóriában van tárolva. A kapcsoló mátrix felépítése FPGA / CPLD típus függő, mivel egy kevesebb cellát tartalmazó áramkör kevesebb kapcsolási lehetőséggel rendelkezik, mint egy több millió cellát tartalmazó eszköz.

Néhány szó CPLD-k és FPGA áramkörök alkalmazásáról.

A CPLD áramköröket leginkább olyan esetben érdemes alkalmazni, amikor nincs feltétlenül szükség nagyfokú párhuzamosításra, csupán egy jó adag logikai IC-t szeretnénk leváltani. Modelltől és gyártótól függően igen eltérő lehet a chip-ek ára és képessége, viszont ha több, mint 5 logikai integrált áramkört kellene alkalmazni a kapcsolás megvalósításához, akkor már kényelmesebb egy CPLD-t használni. Árban nem feltétlen lesz olcsóbb, de ha később módosítani kell az áramkört, akkor könnyen megtehetjük.

Az FPGA áramkörök előnye a sok cellában és a menet közbeni konfigurálható működésben keresendő. Olyan helyeken érdemes elgondolkodni egy FPGA alkalmazásában, ahol párhuzamosítani kell feladatokat. A komolyabb FPGA chip-ek felfoghatóak SOC áramköröknek is, mivel a legtöbb esetben tartalmaznak egy integrált processzort is. Ebből adódóan egy FPGA működésre bírása jóval nagyobb feladat, mert kell hozzá külső FLASH memória, RAM, több órajel és feszültség forrás. Több órajel és feszültség forrásra CPLD chip esetén is szükségünk lehet.

Ezen eszközök programozására több lehetőségünk is van. A legnagyobb gyártók az Altera és Xilinx fejlesztő környezetében kapcsolási rajz segítségével is programozhatjuk az eszközöket. A kapcsolási rajzok megtervezéséhez a kombinációs hálózatok és szinkron hálózatok tervezésénél használt alapelvek, eljárások használandóak, így manapság az egyetemek/főiskolák kezdik bevezetni digitális technika oktatásba ezen áramköröket.

Bonyolultabb feladatok megvalósításához használható VHDL vagy Verilog hardver leíró nyelv. Ezen nyelvek megalkotására azért volt szükség, mert a kapcsolási rajz alapú tervezés egy bizonyos méret felett átláthatatlan.

A VHDL nyelv kifejezetten nagy sebességű integrált áramkörök leírására lett kifejlesztve az 1980-as években. A nyelv a Pascal programozási nyelvre épít erősen, mára teljesen szabványosított. A Verilog leíró nyelv az 1980-as évek végén született meg és az 1990-es években fejlesztették ki mai szintre. A VHDL és a hagyományos programozási nyelvek előnyeit egyesíti. Eredetileg szimulációs célokra fejlesztették ki, de népszerűsége miatt mára már chip-ek tervezésénél is alkalmazzák.

Felmerülhet a kérdés, hogy melyiket érdemes használni? Ez igazából egyén függő. Van aki a VHDL-t preferálja, van aki a Verilog nyelvet. A VHDL talán egy kicsivel jobban támogatott, de ugyan akkor bonyolultabb, messzebb áll a hagyományos programozási nyelvek világától.

A CPLD és FPGA áramkörök közös jellemzője, hogy nem igen támogatnak 5V-os logikát, működésükhöz 3,3V vagy bonyolultabb áramkörök esetén 1,8V feszültség kell. A legtöbb áramkör nem 5V kompatibilis, azaz nem képes 5V-os logikai jel fogadására. Ezért az adott chip használata előtt érdemes tájékozódni, hogy 5V kompatibilis vagy nem.

Altera Quatrus beüzemelése és használata

Az Altera gyártmányú CPLD és FPGA áramkörök fejlesztőeszköze a Quatrus II nevű szoftver, amely létezik ingyenes és fizetős változatban. Az ingyenes változat a Web Edition. A Web Edition és a teljes verzió között az a különbség, hogy a Web Edition nem tud programot fordítani a csúcskategóriás FPGA áramkörökre.

A szoftver a http://dl.altera.com/?edition=web címről szerezhető be. Letöltés előtt regisztrálni kell egy felhasználói fiókot. A teljes letöltés igen nagy méretű. Ha minden eszköz támogatást letöltünk, akkor 6,5GB körüli szabad helyre lesz szükségünk. A Quatrus és a ModelSim szoftver mindenképpen kelleni fog. Az eszköz támogatás opcionális. Itt elég letölteni a programozni kívánt eszköz szériájának megfelelőt.

Letöltés egy picit problémás, mivel egy letöltés menedzsert akar az oldal ráerőltetni a felhasználóra. Van közvetlen letöltési opció is, azonban az esetek többségében nem akar működni. A probléma úgy kerülhető meg, hogy a letöltés menedzsert választjuk, majd nem telepítjük a programot a felugró ablak szerint, hanem az ablak alján található „cannot complete the installation, click here". Feliratra kattintunk, majd a megjelenő ablakban az OK gombra, és megjelennek a közvetlen letöltési linkek, amik egyszerűen letölthetőek.

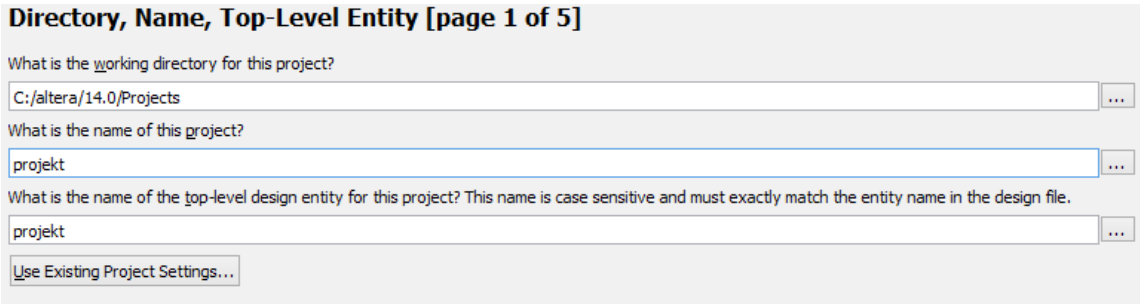
A fájlok letöltése eltarthat egy darabig letöltési sebességtől függően. A letöltött fájlokat egy mappába kell pakolni letöltés után. Ha minden ugyan abban a mappában van, akkor a Quatrus telepítője automatikusan felteszi az összes komponenst. A telepítéshez eszköz támogatástól függően 3,5-8GB szabad hely kell.

Projekt és terv kapcsolási rajz alapján.

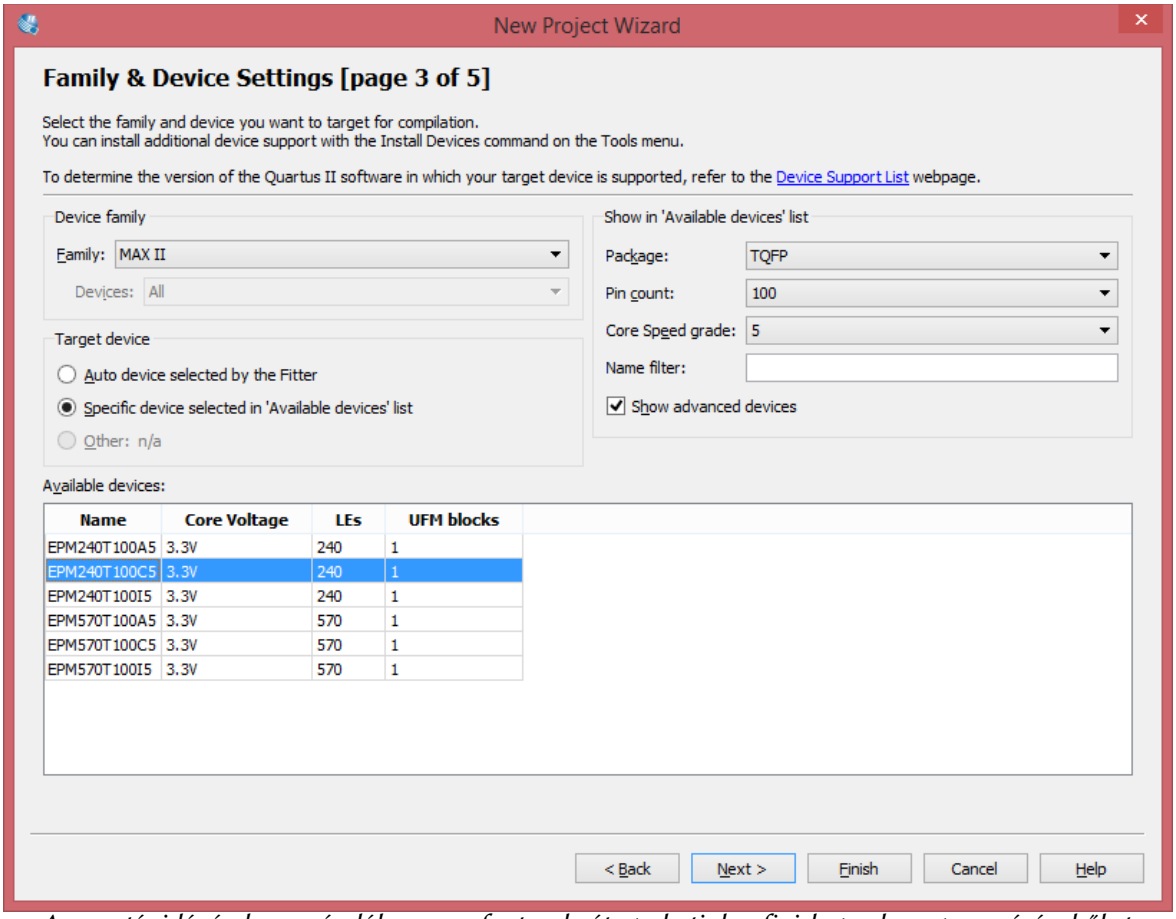
A Quatrus projektekben dolgozik, ezért a szoftver elindítása után a File menü new project wizzard... parancsával tudunk létrehozni egy új projektet, vagy a kezdőlapon a new project wizzard gombra kattintva.

A varázsló első fontos beállítása a projekt neve és helye. Egy üres mappát adjunk meg projekt helynek. A projekt nevében ügyeljünk arra, hogy ne legyenek benne ékezetes betűk és szóközök, mivel a Quatrus nem szereti a speciális karaktereket.

A top-level design név a kapcsolási rajz, vagy fő program neve. A rajzot/program fájlt ezen a néven kell elmenteni. Ellenkező esetben a fordítás sikertelen lesz. Új rajz/program létrehozásakor ezt a nevet kínálja fel mentéskor a program. Később a projekt beállításaiban módosítható.



Az ez után következő beállítás az eszköz típusára vonatkozik. A listát szűkíthetjük eszköz család, név, tokozás típus és lábszám alapján is. Fontos, hogy a megfelelő eszközt válaszuk ki, mert ha nem megfelelőt választottunk, akkor a feltöltés és a lábak hozzárendelésekor hibába ütközhetünk.



Az ez utáni lépések a varázslóban nem fontosak, átugorhatjuk a finish gomb megnyomásával őket, aminek hatására elkészül a projektünk, ami jelenleg még nagyon üres.

Ezért a File menü new... parancsával hozzá kell adni egy new block diagram/schematic fájlt. Ennek hatására megnyílik a kapcsolási rajz szerkesztő.

A kapcsolási rajz szerkesztőben alkatrészeket úgy tudunk felvenni, hogy duplán kattintunk egy üres részre. A megnyíló alkatrész választóban ki tudjuk választani, hogy milyen alkatrészt tegyünk le. A Primitives mappában alap kapuáramkörök, tárolók és egyéb egyszerű eszközök találhatóak. Az Others mappában található maxplus2 könyvtárban a 74xx szériás logikai áramkörök közül jó pár megtalálható.

Az egyes eszközökre nevük alapján is rákereshetünk. A kapuáramkörök esetén a kapu neve után álló szám a bemenetek számát jelöli. Így ha beírjuk a nand2 szót a keresőbe, akkor egy két bemenetes NAND kaput kapunk, amit tegyünk is le.

A kapcsolási rajzba nagyítani az egér görgő segítségével tudunk gyorsan, ha közben lenyomva tartjuk a CTRL gombot.

Több kaput és eszközt is hasonlóan tudunk letenni. Az eges eszközöket vezetékkel úgy tudjuk összekötni, hogy a bemeneteikre vagy kimeneteikre kattintunk. Ekkor átvált vezeték rajzolásra a szerkesztő. Opcionálisan az eszköztárról is tudunk vezetékezni.

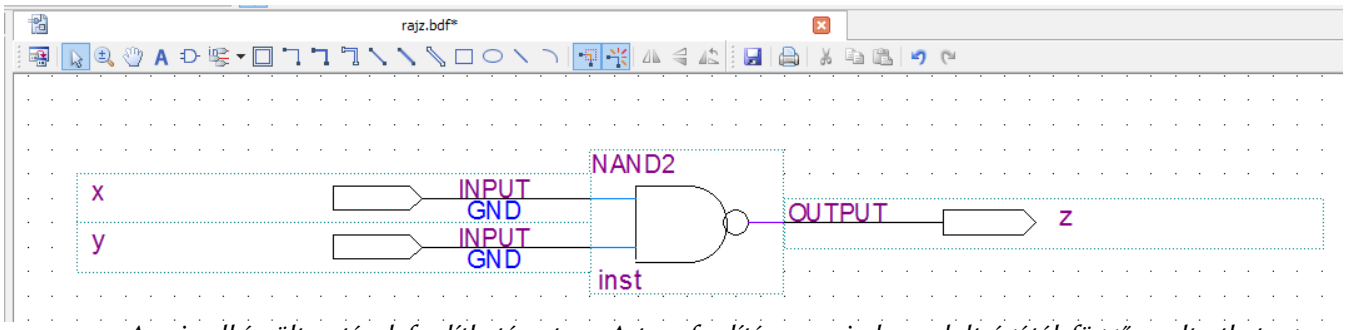
A vezetékeknek név alapján is linkelhetőek egymáshoz, mint más kapcsolási rajz szerkesztőkben. A vezeték megrajzolása után a bal gommbal kattintás után megjelenő menüben a Properties parancs segítségével elnevezhető a vezeték. Az elnevezésnél ügyeljünk arra, hogy ékezetes betű és szóköz ne legyen benne.

Ha két vezetéket azonos névvel látunk el, akkor azokat logikailag összekötöttnek tekinti a program.

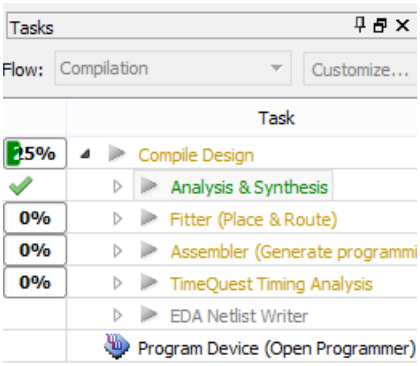
A kész rajz működtetéséhez szükségünk lesz bemenetekre és kimenetekre, amelyek a külvilágot reprezentálják. Ezek a primitives menüben találhatóak meg. A bemenetek neve input, míg a kimenetek neve output.

A ki és bemenetek elhelyezése után el kell őket nevezni. Erre azért van szükségünk, hogy könnyebb dolgunk

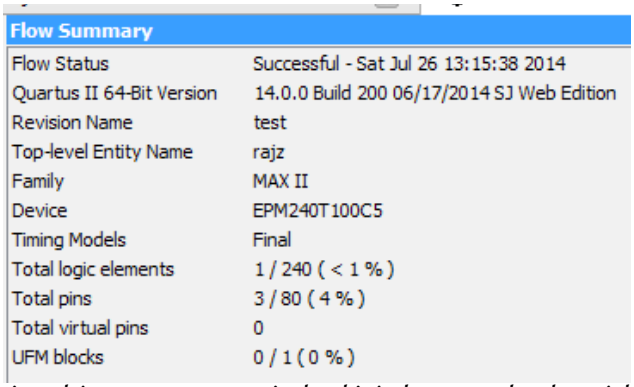
legyen akkor, mikor a kimeneteket és bemeneteket fizikailag a chip lábaihoz rendeljük. Elnevezni a be és kimeneteket úgy tudjuk, hogy duplán kattintunk rájuk, majd a pin name mezőbe beírjuk a kívánt nevet. A default value mezőben megadhatjuk az alapértelmezett állapotot is.



A rajz elkészülte után lefordítható a terv. A terv fordítása a rajz bonyolultságától függően eltarthat egy ideig. A fordítás a CRTL+L gomb segítségével indítható el, vagy a Processing menü Start Compilation parancs segítségével. A fordítás folyamatáról a Tasks ablakban tájékozódhatunk.

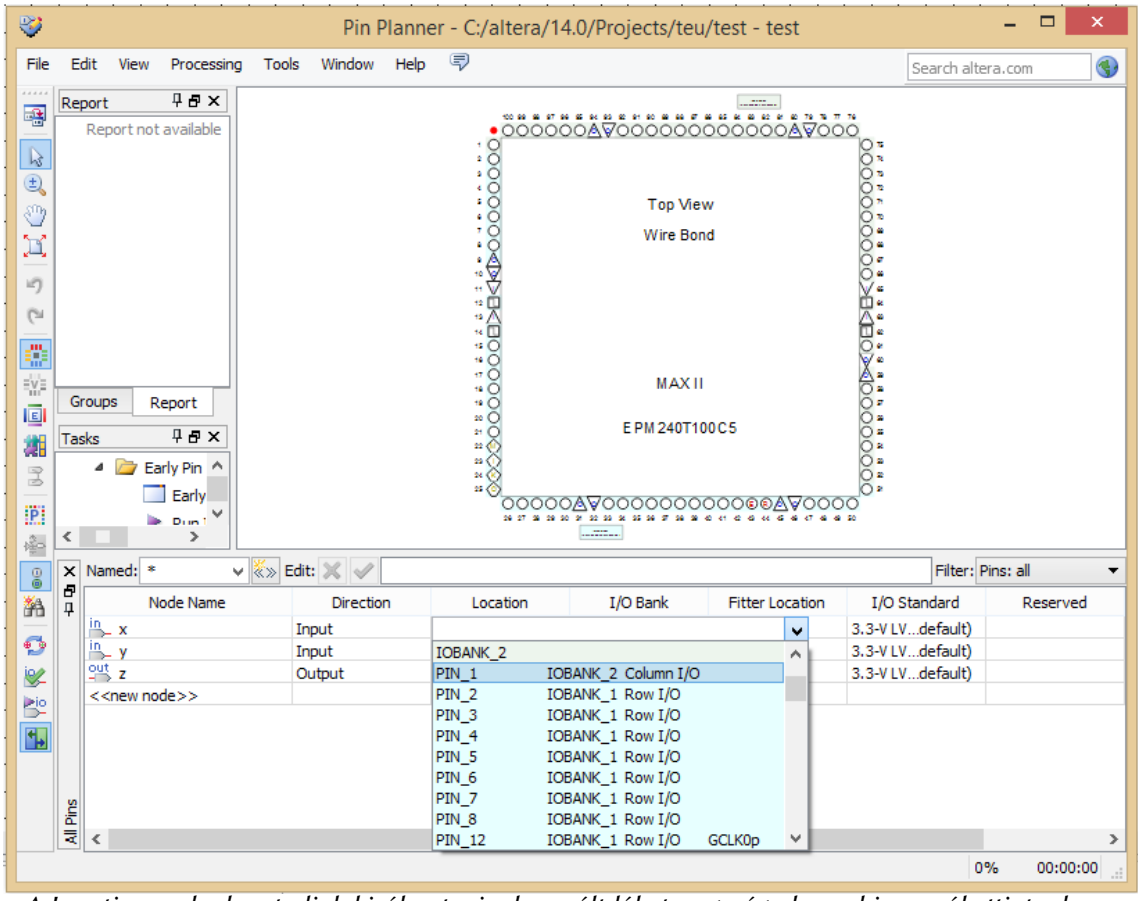


A fordítás végeztével kapunk egy statisztikát a felhasznált logikai bolokkok számáról és a felhasznált I/O lábak számáról, illetve más hasznos adatokat olvashatunk ki a fordítási jelentésből.



Flow Summary	
Flow Status	Successful - Sat Jul 26 13:15:38 2014
Quartus II 64-Bit Version	14.0.0 Build 200 06/17/2014 SJ Web Edition
Revision Name	test
Top-level Entity Name	rajz
Family	MAX II
Device	EPM240T100C5
Timing Models	Final
Total logic elements	1 / 240 (< 1 %)
Total pins	3 / 80 (4 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

Ezzel még azonban nincs kész a program, mivel a ki és bemeneteket hozzá kell rendelni a chip megfelelő lábaihoz. Ezt úgy tudjuk megtenni, hogy az Assignments menüből megnyitjuk a Pin planner ablakot. Ezt azért érdemes fordítás után megnyitni, mivel ekkor automatikusan feltölti a ki és bemeneteket, nekünk csak a fizikai lábakat kell hozzárendelni. Fordítás előtt megnyitva azonban manuálisan nekünk kell kitölteni a táblázatot.



A Location oszlopban tudjuk kiválasztani a használt lábat, vagy úgy ha a chip-en rákattintunk a megfelelő lábra és hozzárendeljük a kívánt be vagy kimenetet.

Az összerendelés után ismét le kell fordítani a tervet. A második fordítás után a terv készen áll arra, hogy feltöltsük az eszközre.

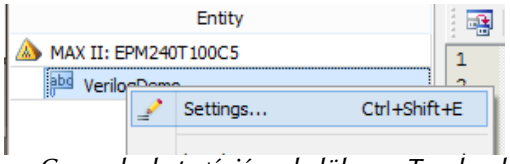
Projekt és terv verilog kód alapján

A Verilog kóddal való tervezés nagymértékben hasonlít a kapcsolási rajzos módszerhez. Azonban a fájl hozzáadása ablakban a Verilog HDL fájlt válasszuk ki, a kapcsolási rajz helyett.

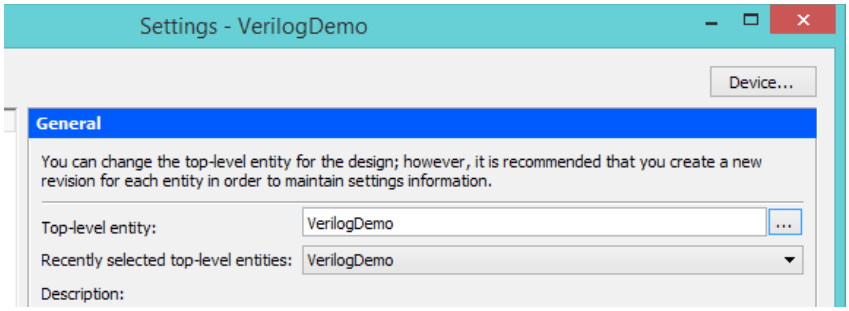
A kód begépelése után ismét két körös fordítást érdemes alkalmazni. Az első fordítási kör után szintén be kell állítani az összerendeléseket és a második kör után feltölthető az eszközre a program.

A Verilog alapú tervezésnél arra ügyelni kell, hogy egy forrásfájl több modult is tartalmazhat, de a fő modul (amelyik a chip-be kerül) az lesz, amelyiknek a neve megegyezik a projekt névvel.

A fő modul neve megváltoztatható, a projekt beállításaiban. Az entity ablakban bal kattintás után a Settings menüpontot választva megnyílnak a projekt beállítási lehetőségei.



A megnyíló ablakban a General kategórián belül a Top-level entity beállítás módosításával megváltoztathatjuk a fő modult.


Settings - VerilogDemo
Device...
General
You can change the top-level entity for the design; however, it is recommended that you create a new revision for each entity in order to maintain settings information.
Top-level entity:
VerilogDemo
Recently selected top-level entities:
VerilogDemo
Description:

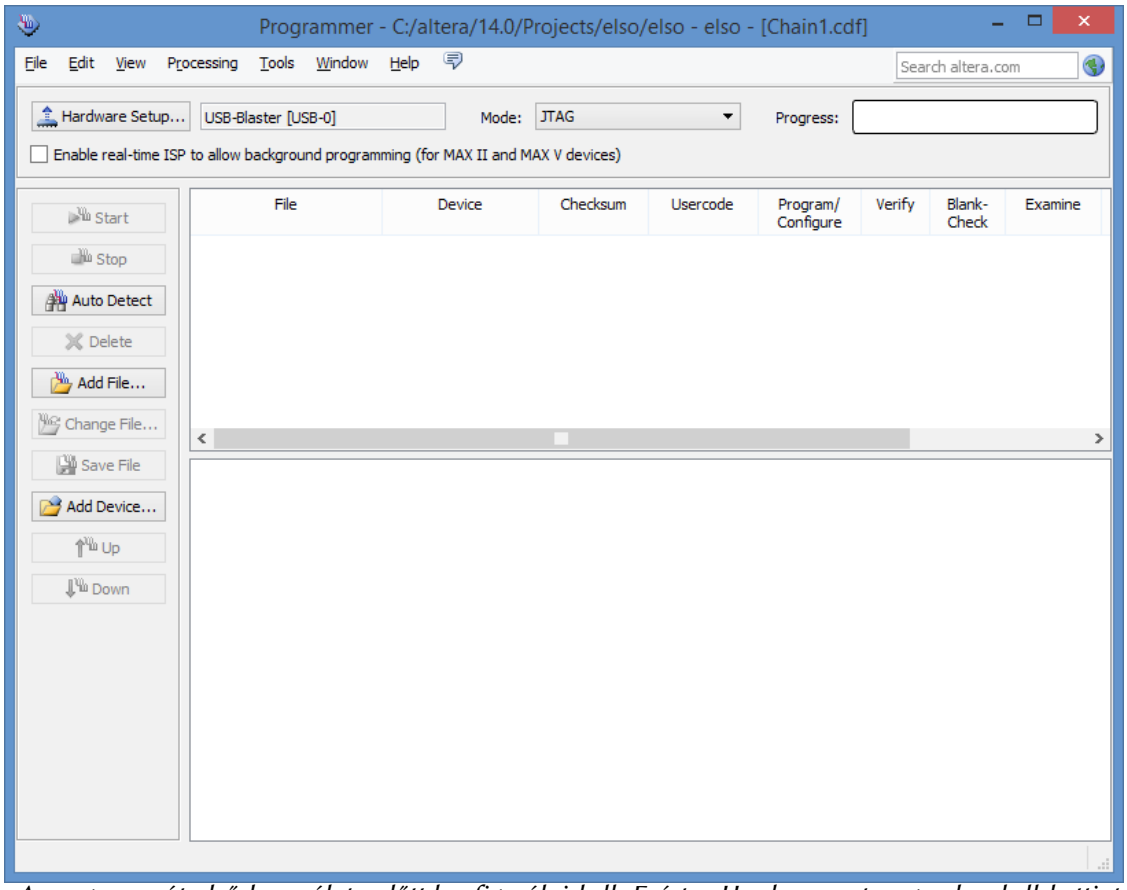
Program feltöltése

A program feltöltéshez szükségünk lesz egy JTAG eszközre. Ez az Altera esetén az USB Blaster. A Quatrus alapértelmezetten telepíti a használatához szükséges illesztő programot, de nem aktiválja. Ezt nekünk manuálisan kell megtenni.

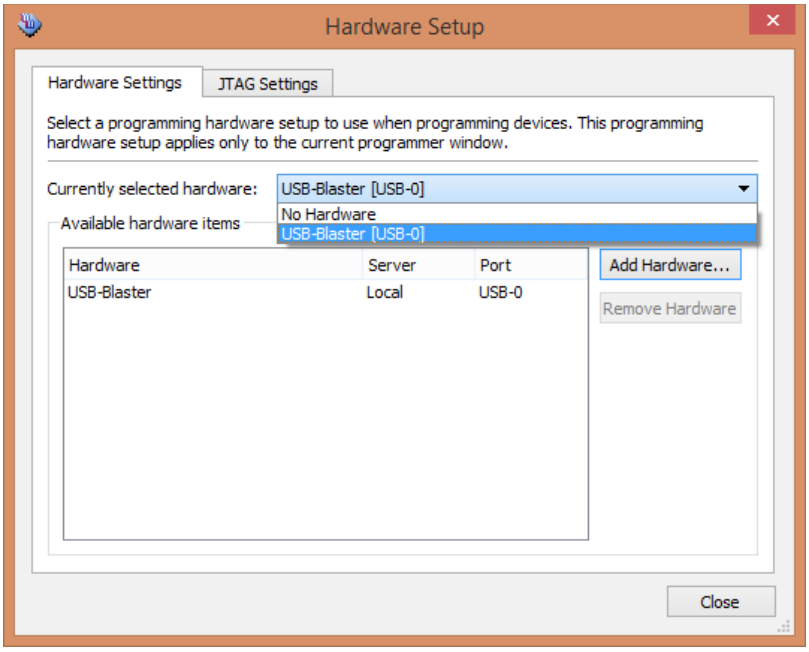
Az USB Blaster csatlakozása után az eszköz kezelőben ismeretlen eszközként fog megjelenni. Ezért bal kattintás után az illesztőprogram frissítése menüpontra kattintva elindul az illesztőprogram telepítő varázsló. Első lépésben azt kell választani, hogy a saját számítógépen keressen friss illesztőt, majd meg kell keresni az Illesztőt. Ez alapértelmezett telepítés esetén a C:\altera\14.0\quartus\drivers\usb-blaster\ mappában található meg.

Az illesztő telepítése után a programozó készen áll a használatra. Programozás előtt gondoskodni kell róla, hogy a lap kapjon megfelelő táp ellátást, mivel önmagában az USB Blaster nem biztos, hogy képes lesz ellátni a programozáshoz szükséges feszültséggel a lapot.

A JTAG felület és a tápellátás csatlakoztatása után a projekt feltöltése a programozó a Tools menü programmer menüpontjára kattintva indítható el.

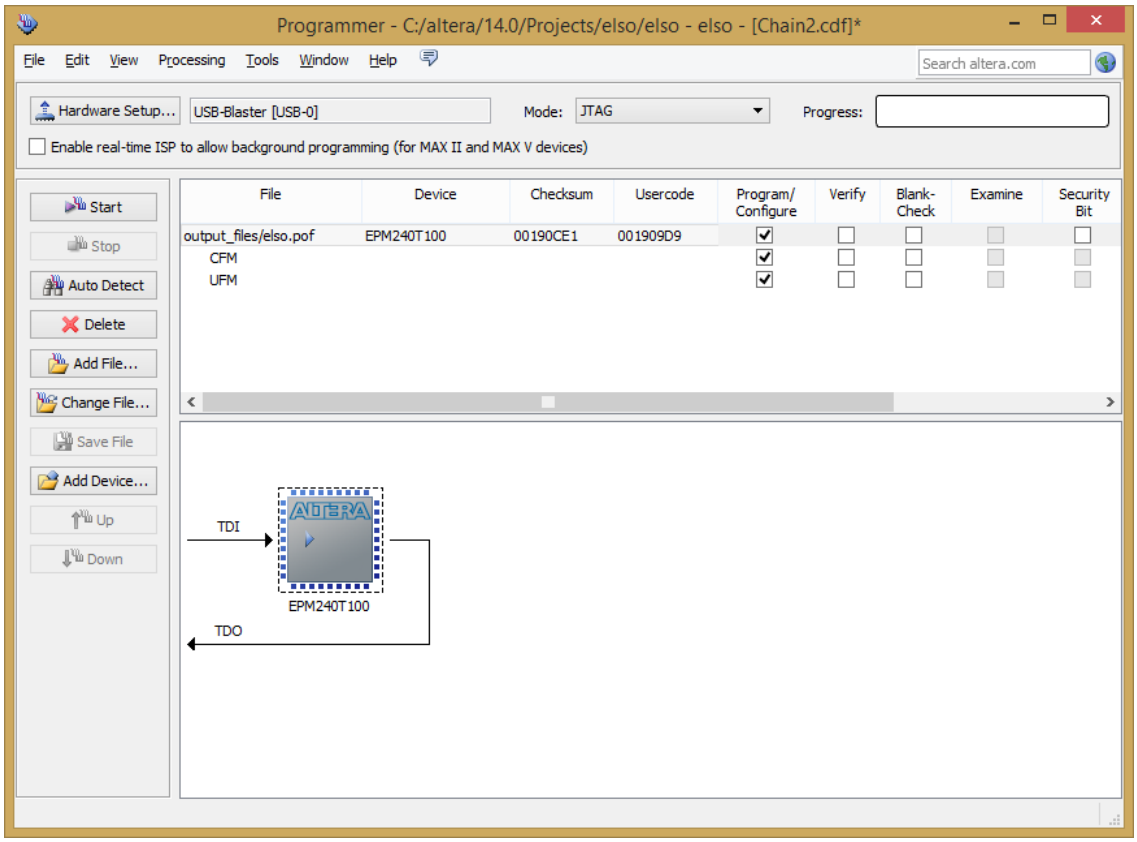


A programozót első használata előtt konfigurálni kell. Ezért a Hardware setup gombra kell kattintani. Az ennek hatására megjelenő ablakban ki kell választani, hogy a használt eszköz az USB Blaster az alapértelmezett No Hardware helyett.



A beállítás után az Add file... gomb segítségével tudjuk kiválasztani a feltölteni kívánt fájlt. A megnyíló mappa választó a projekt mappáját fogja alapértelmezetten mutatni. A feltölteni kívánt fájl az output_files könyvtárban lesz és neve megegyezik a projekt nevével.

A program beolvasása után a programozó alsó részében végig követhető a teljes JTAG lánc, mivel egy programozás során nem csak egy eszközre tölthető program. A program / configure oszlopban a betöltött fájlunk mellé tegyünk egy pipát, ezzel jelezve, hogy ténylegesen programozni szeretnénk az eszközt.



A programozási folyamat a Start gomb megnyomásával indítható el. A program bonyolultságától és a JTAG lánc bonyolultságától függően eltarthat akár 1-2 percig is, mivel a JTAG buszrendszer nem a leggyorsabb.

A program feltöltése után tesztelhető a kapcsolás a fizikai eszközön. Tesztelés közben ügyeljünk arra, hogy CPLD/FPGA függően az egyes lábak kimeneti jelszintje és terhelhetősége eltérő lehet. Ezért használat előtt tájékozódjunk minden esetben az adott chip képességeiről.

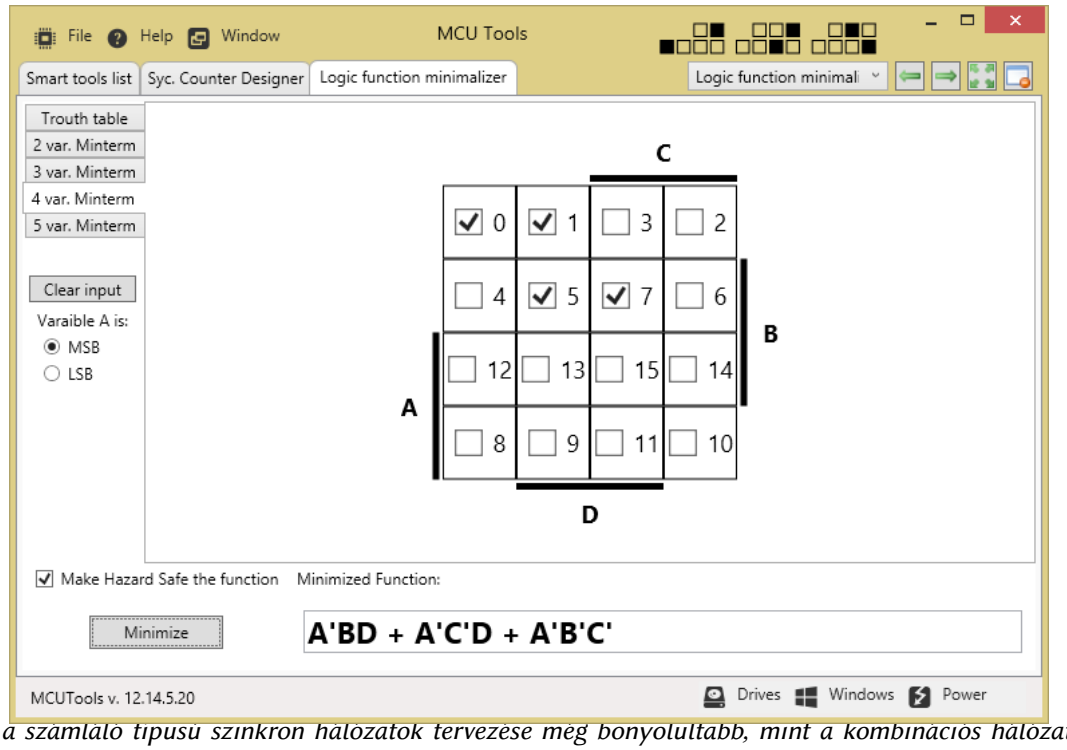
Előfordulhat, hogy a feltöltés sikertelen lesz. Ennek legtöbbször az oka az, hogy valami érintkezési probléma van a feltöltő és a chip között. Ezért ellenőrizzük a kapcsolatot.

Sikertelen lesz a feltöltés akkor is, ha a lapra később adunk feszültséget, mint ahogy a programozót csatlakoztatjuk. Ezen probléma úgy orvosolható, hogy az USB Blastert kihúzzuk, majd újra csatlakoztatjuk, a programozó szoftverben pedig az Auto Detect gombra kattintunk.

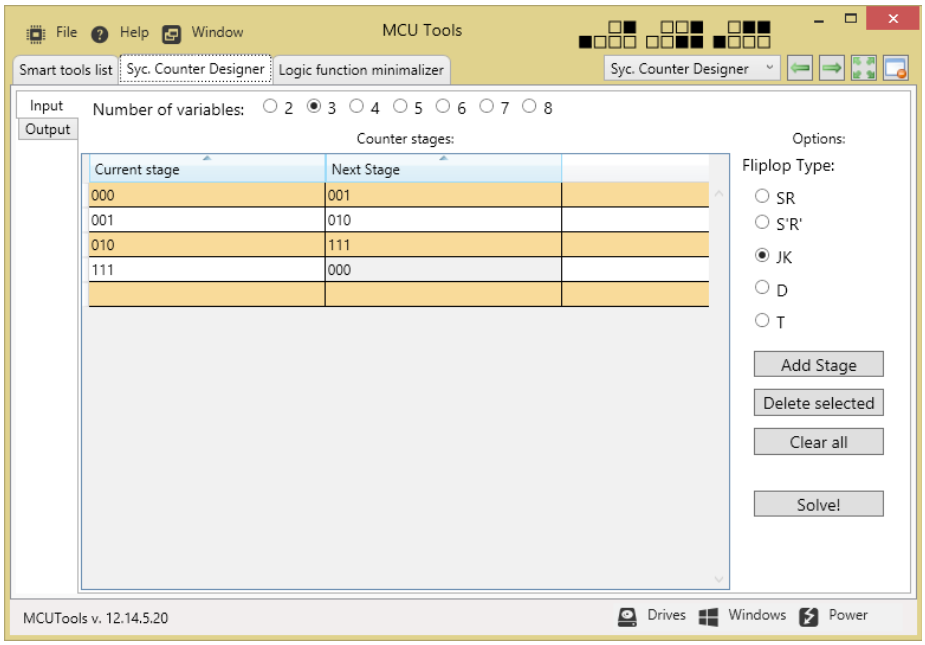
Kombinációs és szinkron hálózatok tervezése

A CPLD és FPGA eszközök programozása esetén a legnagyobb probléma az, hogy először manuálisan ki kell számolni a vezérlőfüggvényeket, amelyek segítségével megvalósítható a kombinációs hálózat vagy a szinkron hálózat.

Ez igen sok időt és erőforrást vehet igénybe. Az MCU Tools program rendelkezik egy logikai függvény egyszerűsítő alkalmazással, amely 8 változós függvényekkel is elboldogul.



Mivel a számláló típusú szinkron hálózatok tervezése még bonyolultabb, mint a kombinációs hálózatoké, ezért a programba beépítettem egy szinkron számláló tervezőt is, amely ugyan azt a módszert implementálja, mint amit a digitális technikai alapismeretek fejezetben ismertettem.



20. Kiegészítő IT ismeretek

Ebben a fejezetben olyan témák kerülnek elő, amelyek ismerete nem feltétlenül szükséges mikrovezérlős projektek készítéséhez, azonban előfordulhat, hogy egy-egy komolyabb projekt tervezésekor és megvalósításakor, hogy szükségessé válik bizonyos témák részletesebb ismerete.

Az itt érintett témák nagy része kivonatos, sok esetben csak a minimálisan szükséges ismereteket tartalmazza egy-egy alfejezet. Ez annak köszönhető, hogy az itt érintett témák nagy részének külön-külön is több száz oldalas szakirodalma van. Erre egy jó példa a hálózatok, hiszen erről a témáról Andrew S. Tanenbaum egy több, mint 500 oldalas könyvet írt.

Ezért ezen témák részletesebb, mélyrehatóbb ismerete érdekében elengedhetetlen kiegészítő szakirodalom olvasása.

Hálózatkezelési alapok

A hálózatok alapjaival érdemes foglalkoznunk mikrovezérlők esetén is, mivel manapság egyre több téren elvárás, hogy internetre köthető legyen egy berendezés. Egyetlen egy mikrovezérlős könyvben sem olvastam a hálózatok alapjairól, mikor a hálózatkezelés szóba került. Pedig nem feltétlen kell az olvasónak értenie a témához.

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*

87. 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*

88. 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*

89. 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. Erre úgy képes, hogy az OSI modell második rétegében dolgozik és figyelembe veszi a küldő és célállomás MAC címét a továbbítás előtt. Bekapcsolása pillanatában nem ismeri a hálózati topológiát, ezért bekapcsolása után feltérképezi a hálózat számára releváns részleteit.

Bridge

Alhálózatok összekapcsolására alkalmas eszköz, a Switch-hez hasonlóan az OSI modell második rétegében dolgozik, ezért nem képes útvonal-választási feladatok megoldására, mint a Router. Általában a fejlettebb Switch-ek rendelkeznek Bridge funkcióval is.

Router

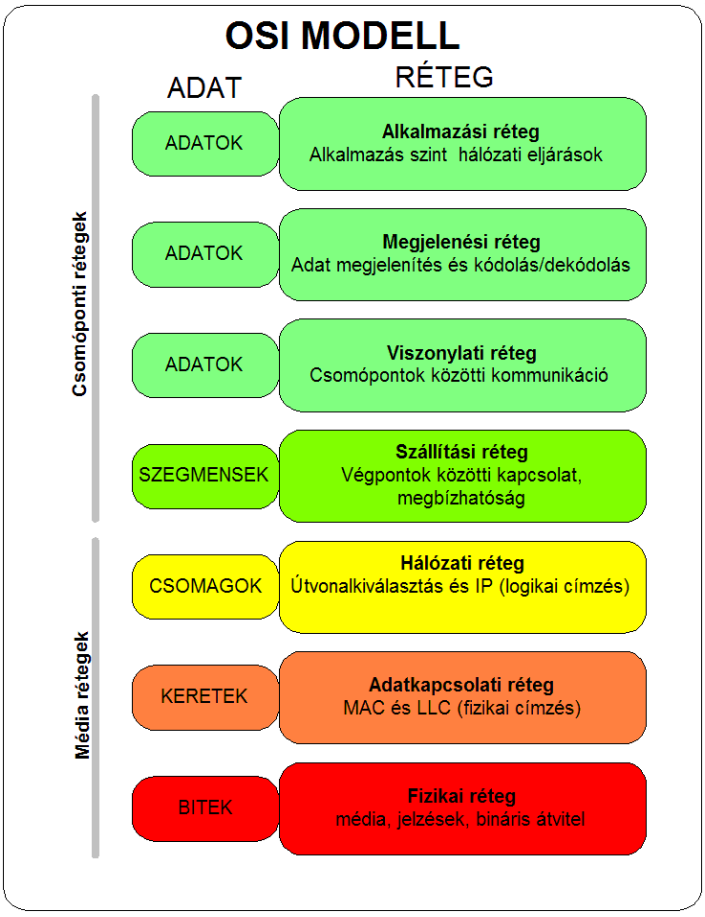
A router hálózatok összekapcsolására alkalmas. Definíció szerint két portja van. Egy WAN port, ami egy másik hálózathoz kapcsolódik, és a LAN port, ami ahhoz a hálózathoz, amiben a router van. Valóságban azonban a legtöbb router egyben egy switch is, tehát több LAN portja van. Annyival fejlettebb a bridge-nél,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

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.

204. 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 kioszthatóó 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

90. 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 65 536db 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

91. táblázat: Fontosabb szolgáltatások által használt portok

Web technológiai alapismeretek

Az internet a 90-es évek eleje óta egyre inkább begyűrűzik a hétköznapokba. Manapság olyan szinten, hogy a hűtő és mosó gépeinket is akár internetre köthetjük és távolról vezérelhetjük őket.

Természetesen ez még manapság nem mindennapos dolog, de már létezik rá a technológia és kereskedelmi forgalomban is kaphatóak az eszközök, így mondhatni csak idő kérdése a dolog.

Elnézve ezen technológiákat jogosan felmerül előbb-utóbb egy komolyabb mikrovezérlős projekt esetén is, hogy kössük internetre és vezéreljük távolról a kütyünket.

Ezen fejezetben a Web technológiai alapismeretivel foglalkozom, szintén sok helyen felületesen, korábban már említett ok miatt: nagy téma. Másik ok annak tudható be, hogy az internet és a webes technológiák mondhatni 1-2 évente hatalmas fejlődésen mennek át, így konkrétan egy részletesebb technológia ismertetése felesleges lenne, mivel hamar elavul. Így ezen fejezetben csak az úgymond időtálló ismeretekről lesz szó, ami alapján az olvasó beleáshatja magát a témába nagyon részletesen, ha szükségesnek látja.

Internet és a Web kapcsolata

A köznapi nyelvben internetnek nevezzük a böngészőnk által elérhető, megkereshető tartalmakat. Azonban valójában az Internet egy globális TCP/IP protokoll párost használó hálózatot jelent, amin keresztül számos programot használhatunk.

A web ennek egy részhalmaza, ha úgy tetszik. Az 1990-es évek elején jelent meg, a World Wide Web szavak rövidítéséből mára már csak a web szó maradt meg. A HTTP/HTTPS valósítja meg a működését, ami manapság rengeteg helyen használt, köszönhetően annak, hogy a szerver oldali felépítése viszonylag egyszerű. Ebből adódóan mikrovezérlős projektjeink kapcsán, ha internet vezérlést szeretnénk megoldani, akkor célszerű ezt a protokollt alkalmazni. További előnye a technológiának, hogy használatához csak egy böngésző kell, ami mondhatni minden operációs rendszeren alapértelmezetten adott.

A HTTP működése

A HTTP szó a Hyper Text Transfer Protocol szavak rövidítése. Eredetileg egy szöveges információk átvitelére megalkotott protokoll, ami lehetővé teszi bináris adatok átvitelét is. Az átvitt szövegek speciális felépítésűek web lapoknak nevezzük őket. Egy weblap HTML kódot tartalmaz, ami egy leírónyelv, meghatározza azt, hogy hogyan néz ki egy oldal.

Adatot két -féle kérés segítségével lehet lekérni vagy továbbítani egy HTTP szerver felé. GET és POST kéréssel. Legtöbbször GET kérés alkalmazott. Ez maximum 1024 karakterből állhat, legtöbb esetben arra használt, hogy lapokat kérjünk le, de arra is alkalmazható, hogy kis méretű adatokat kódoljunk a kérésbe, amit később a szerver oldalon feldolgozunk. Egyszerre maximum három kérésre válaszol kliensenként a szerver.

A POST kérés nagyobb adatok továbbítására alkalmazható. Segítségével a kliens fájlokat és űrlap adatokat tud rejtetten továbbítani a szerver felé. A rejtett alatt nem titkosított értendő, csak annyi, hogy a böngésző címsorában nem jelennek meg a kéréssel kapcsolatos adatok.

A Kérésre a szerver egy válaszkódot küld és a válasz tartalmát. A válaszkódokból tudja megállapítani a kliens, hogy mit is kell kezdenie a dokumentummal, illetve, azt is megkapja, hogy a kapott adat milyen formátumú. Erre a mime típus jelzők szolgálnak, amelyek minden fájltípus esetén adottak. Ez az üzenet manuálisan felülbírálható, de általában a szerver társítja ezt a választ a fájlhoz.

A HTTPS ugyan úgy működik, mint a HTTP, csak titkosítottan. A szerver és a kliens között zajló kommunikáció titkosítva van egy kétkulcsos titkosító algoritmussal. Ez azt jelenti, hogy kliens és a szerver is két kulccsal rendelkezik. Az egyik segítségével csak titkosítani lehet, a másik kulcs segítségével csak feloldani lehet azt. A kommunikáció kezdete elején eljuttatják egymáshoz a titkosító kulcsukat, amit publikus kulcsnak szokás nevezni. Ennek segítségével tudnak egymásnak adatokat küldeni ami csak az adott kulcshoz tartozó feloldó kulcs (titkos kulcsnak szokták nevezni) segítségével fejthető vissza. Így a kommunikációba nem tud belehallgatni egy harmadik fél, mivel ehhez két titkos kulcsra is szüksége lenne.

A harmadik fél kizárása mellett meg kell még oldani az identitás ellenőrzést is, hogy valójában azzal kommunikálok e, akivel szeretnék. Erre a tanúsítványok szolgálnak. Egy HTTPS protokollt használó web oldalnak rendelkeznie kell egy tanúsítvánnyal, amit egy harmadik személy bocsájt ki. A tanúsítvány azt jelzi a kliens felé, hogy a szerver megbízható, valóban az a szerver amivel kommunikálni szeretnék.

Egy ilyen tanúsítvány beszerzése az igényelt időtől és a tanúsítvány kibocsájtótól függően nagyon sok pénzbe is kerülhet. Igényelt idő alatt az érvényessége értendő. Az érvényesség lejárta után egy tanúsítvány nem alkalmazható tovább. A súlyos költségek miatt szokták kisebb rendszerek esetén az üzemeltetők saját maguk „aláírni" a tanúsítványt. Ilyen formában ez identitás azonosításra nem alkalmas, de megkövetelt a titkosítás működéséhez ezért szokták ezt megtenni. Ilyen esetekben a böngészők többsége figyelmeztet, hogy az oldal használata kockázatos lehet, mert nem megbízható.

A vásárolt tanúsítványt a kibocsájtó bármikor érvényteleníthetí, amennyiben a tanúsítvány segítségével bejelentett visszaélések történnek.

Mikrovezérlős projektek esetén nem igen fogunk HTTPS protokollt alkalmazni, mivel az ehhez szükséges számítási teljesítmény nem áll rendelkezésre egy mikrovezérlőben sem.

A szerver oldalon adat feldolgozásra és módosításra alapértelmezetten nem kínál lehetőséget a protokoll, mivel eredetileg statikus, előre megírt lapok továbbítására találták ki. Később jelent meg az igény a dinamikus weboldalakra, amelyek felhasználói interakciók alapján módosított tartalmakat szolgáltassanak a látogatóknak. Erre az évek folyamán rengeteg technológia jelent meg.

Szerver oldali programozási nyelvek

CGI

A CGI nem programozási nyelv, hanem egy technológia, ami lehetővé teszi dinamikus weboldalak létrehozását. A technológia lényege, hogy ha a web szerverről egy futtathatónak megjelölt fájlt kérnek le, akkor a szerver lefuttatja a programot a klienstől kapott paraméterekkel, majd a kliensnek a program kimenetét küldi vissza. Ezzel a technológiával lényegében bármilyen programozási nyelvet alkalmazhatunk dinamikus weboldalak létrehozására, amennyiben a webszerver támogatja a CGI kódok kiszolgálását.

Perl

Az első dinamikus weboldalak létrehozására használt Script nyelvek egyike. A nyelv alapjait a C és a Unix Shell nyelvéből meríti. Érdekes tulajdonsága a nyelvnek, hogy nagyon erős reguláris kifejezés illesztő rendszerrel van felszerelve, amely szintaxisát mára minden modern nyelv átvette. Mai napig aktívan használt programozási nyelv, Unix rendszerek alatt számos eszköz épít rá.

PHP

Talán a legrégebbi szerver oldali programozási nyelvek egyike. A neve onnan jön, hogy megalkotója személyes weblapok karbantartására tervezte. A projektje túlnőtt az eredeti célján és talán ma a legnépszerűbb webes programozási nyelvek egyike. C szerű szintaxist alkalmaz, a kódja közvetlenül HTML lapokba is ágyazható, objektum orientált szkript nyelv.

A népszerűség nem feltétlen jelenti azt, hogy a legjobb is. Számos hibája és problémája van, ennek ellenére rengeteg helyen használt. Könnyű a nyelv segítségével viszonylag gyorsan látványos dolgokat alkotni, azonban komoly szaktudást igényel, hogy a létrehozott weboldal mögött található kód minőségi legyen. Mondhatni a webes programozás C nyelve.

ASP.NET

A .NET webes keretrendszere. Web alkalmazások és lapok létrehozására kifejlesztett eszköz, előnye, hogy komponens rendszerű, ami lehetővé teszi alkalmazások gyors fejlesztését. Igen jól bánik az erőforrásokkal, így remek választás nagyméretű weboldalak létrehozására. Hátránya, hogy igazán jól, probléma mentesen csak Windows Server rendszereken működik, így leginkább csak belső hálózati céges alkalmazások létrehozásához használják előszeretettel.

Python

A python nyelvvel külön fejezet foglalkozik. Webes alkalmazási célokra is használható, számos keretrendszer létezik hozzá webes alkalmazások létrehozásához, ennek köszönhetően egyre népszerűbb ilyen célokra is.

HTML5, a weblapok leíró nyelve

A weblapok kinézetét a HTML, Hyper Text Markup Language határozza meg. Ez kezdetben ténylegesen egy szöveg formátum volt, amit jegyzet tömbben lehetett szerkeszteni és ha böngészőben megnyitottuk, akkor kaptunk egy formázott szöveget. Jelenleg az ötös változata van kidolgozás alatt, illetve részben szabványosítva. Mára azonban a HTML5 alatt három technológia értendő. Ténylegesen a HTML, a JavasScript és CSS.

A JavaScript egy kliens oldali szkript alapú programozási nyelv elsősorban. A 90-es évek végén jelent meg

először a Netscape böngészőben, aztán más böngészők is többé-kevésbé kompatibilisen átvették és kiegészítették. Első felvirágzása a 2000-es évek elején volt. Akkoriban előszeretettel vitték túl az alkalmazását, így rövid időn belül kikopott, pontosabban háttérbe szorult az alkalmazása. Második felvirágzása 2006 környékére tehető, mikor felfedezték a web fejlesztők a benne rejlő komoly lehetőségeket, pontosabban akkor csak egyet. Az AJAX-nak elnevezett technológiát, amely segítségével a web szerverről az oldal újratöltése nélkül lehet adatokat lekérni, illetve átvinni. Ezen korszakot nevezték web 2.0-nak ami azért ironikus, mivel a technika régen adott volt már, szóval tényleges előrelépés nem volt mögötte. Azonban azóta a web minden területén alkalmazott.

A CSS mozaikszó a Cascading Style Sheets szavak rövidítése. Ez egy Stílus meghatározó nyelv. Segítségével szétbontható az oldal felépítése és a stílusa. Ez különösen akkor hasznos, ha sok HTML lapot kell készítenünk. Ebben az esetben elég megalkotni a kinézetet leíró stílust, aztán ezt alkalmazni tudjuk az összes lapra. Így elérhető az egységes megjelenés. Ezen technológia is elsőként a Netscape böngészőben jelent meg. Jelenleg a 3.0-s változata a legelterjedtebb.

A böngészők eltérő mértékben támogatják a webes technológiákat, de a Firefox és Chrome böngészők tudása nagyjából azonos szinten van. Web fejlesztés során ezen két böngészővel érdemes tesztelni az elkészített lapjainkat. Ha ezen két böngészően működik az oldal, akkor a böngészők több, mint 90 százalékában működni fog az oldal. A Maradék 10 százalék okozhat gondot. Ebbe elsősorban az Internet Explorer tartozik. Ezen böngésző korábbi változatai nem igen követték az egységesített szabványokat. Ebből adódóan sok kellemetlenséget okozott a webfejlesztőknek. A mostani modern változatai igen jól követik a szabványokat, azonban még a mai napig bele lehet futni kompatibilitási problémákba.

Az Internet Explorer problémái abból adódtak régen, hogy a Microsoft böngészője volt a piacvezető termék, így mondhatni de-facto szabvány volt. Azonban jogosan felmerült az igény, a konkurencia és mások részéről, hogy legyenek szabványosítva a dolgok rendesen. A szabványosítás során azonban nem minden esetben a Microsoft megoldásait alkalmazták, mivel azok csak részben vagy egyáltalán nem voltak hordozhatóak operációs rendszerek között. A HTML nyelv és webes technológiák szabványainak kidolgozásáért a The World Wide Web Consortium, röviden W3C felel.

A szabványok kidolgozása mellett még egy jelentős tevékenységet végeznek webfejlesztői szempontból. Üzemeltetnek egy szabványosság ellenőrző oldalt (http://validator.w3.org/), amely segítségével ellenőrizhető, hogy egy oldal mennyire felel meg egy adott szabványnak.

HTML fejlesztéshez szükséges eszközök

Alapvetően két eszközre van szüksége az embernek HTML5 lapok fejlesztésekor. Egy böngészőre és egy szövegszerkesztőre. Szövegszerkesztőből érdemes egy nagy tudásút választani platform függetlenül. Windows esetén ilyen például a Notepad++, ami beszerezhető a http://notepad-plus-plus.org/ címről. Unix rendszerek alá a Geany nevezetű szerkesztőt ajánlom, ami a legtöbb Linux disztribúció csomagkezelőjében megtalálható, manuálisan a http://www.geany.org/ címről lehet beszerezni.

Természetesen minden platformra léteznek komolyabb szerkesztők, amelyek megkönnyítik a szerver oldali programozási nyelvek használatát és a HTML lapok létrehozását. Minden szerver oldali nyelv esetén megvan a fejlesztők közkedvelt fejlesztőeszköze. Egyszerű lapok létrehozásához azonban bőven elég egy jó jegyzettömb szerű alkalmazás.

HTML alapismeretek

A HTML lapok lényegében szöveg fájlok, amelyekben a < és > karakterek között elhelyezett szövegrészek felelnek a formázásért. Ezeket nevezzük egy angolos szakkifejezéssel tag-eknek. (a tag szó teg-nek ejtendő), amit magyarul szokás elemnek is nevezni. Minden tag rendelkezik egy lezáró elemmel, ami a <> jelek között / jellel

*kezdődik. Például a tag lezárója a *

Speciális tag-ek önlezáróak. Ez azt jelenti, hogy a tag végén helyezkedik el a / jel. Ilyen például a <hr/> tag.

A tag-ek közül kiemelt fontosságúak a fejléc tag-ek. Ezek határozzák meg, hogy a lapot hogyan értelmezze a böngésző. Kihagyhatóak, bár ebben az esetben böngésző függő, hogy működni fog e a lap, vagy sem. Mindenesetre az biztos, ha kihagyjuk a fejléc tageket, akkor a lapunk már nem lehet szabványos.

A HTML szabvány több változtatáson esett át az elmúlt években. Egyes tag-ek el lettek távolítva, míg mások belekerültek. Ezen leírás a HTML 5 szabvánnyal foglalkozik. Egy HTML5 lap szabványos fejléce a következőképpen néz ki:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
</body>
</html>
```

Az első sor a dokumentum típusát határozza meg. Jelen esetben ez HTML5. A dokumentum rész <html> tag-en belül helyezkedik el. A <head> tag rész a dokumentum fejléce. Ami itt van definiálva, az nem lesz látható. Itt szokás megadni a stíluslapokat, a javascript fájlokat, a dokumentum címsorban megjelenő címét (<title> tag), valamint az aktuális lap karakter kódolását és egyéb meta információkat.

Ékezetes betűk esetén célszerű az UTF-8 kódolást használni, mivel ez univerzálisan működni fog. Ehhez azonban olyan szerkesztő kell, ami ezt támogatja. Szerencsére a szerkesztők többsége mára már ilyen.

A <body> tag-en belül kell meghatározni a dokumentum kinézetét. A böngészők csak azt fogják megjeleníteni, ami ezen az elemen belül van definiálva.

A tag-ek rendelkezhetnek attribútumokkal, tulajdonságokkal is. A tulajdonságokat a következő formátumban kell megadni: tulajdonsag="ertek"

Minden elemnek megvan a saját tulajdonság készlete, ami alkalmazható rá. A tulajdonságokkal lényegében extra információt adunk a tag jelentéséhez. Ez az extra információ lehet kiegészítő jellegű formázás, vagy egy olyan fontos információ ami nélkül nincs értelme az alap tag-nek. Négy attribútum minden olyan elem esetén alkalmazható, amely megjelenítésre alkalmas a böngészőben. Ez a négy elem a class, az id, a style és a title attribútum.

A class attribútum egy css osztály hozzárendelésére szolgál, míg az id attribútummal egy egyedi azonosítót adhatunk az adott tag-nek, ami alapján később javascript-ből módosíthatjuk a tartalmát, kinézetét, de css segítségével is egyedi megjelenést adhatunk neki.

A style attribútum segítségével beágyazott stílus információk adhatóak meg egy elemhez, míg a title attribútummal egy szöveges tippként megjelenő információ társítható az elemhez, ha az egérkurzor felette van. Ennek a megadása különösen hasznos a vakok és gyengén látók számára képek esetén, mivel a legtöbb felolvasó program ezen leírásokat is felolvassa.

A beágyazott css alkalmazására ugyan van lehetőség, azonban nagyon nem ajánlott, mivel nagymértékben rontja a weblap kódjának olvashatóságát.

Az alap html elemek használatát az egyszerűség kedvéért táblázatos formában foglaltam össze. A táblázat az alapok elsajátítására alkalmasak, azonban koránt sem tekinthető teljes, vagy részletes dokumentációnak.

Tag	***Önlezáró***	***Leírás***
h1, h2, h3, h4, h5, h6	*nem*	*Címsorok létrehozása. H1-es a legnagyobb betűméretű, h6 pedig a legkisebb méretű címsor.*
p	*nem*	*Bekezdés létrehozása. A bekezdés igazítása az align attribútummal módosítható, amely értéke lehet left, right, center*
br	*igen*	*Sortörés beiktatása*
i	*nem*	*A tag-ben elhelyezett szöveg dőlt betűs lesz*
b	*nem*	*A tag-ben elhelyezett szöveg félkövér stílusú lesz*
u	*nem*	*A tag-ben elhelyezett szöveg aláhúzással fog megjelenni*
sub	*nem*	*Alsó index formázású szöveg*
sup	*nem*	*Felső index formázású szöveg*
pre	*nem*	*Előformázott szöveg, a szövegből nem lesznek eltávolítva a felesleges szóközök*

92. Táblázat: Alapvető HTML tag-ek

Tag	***Önlezáró***	***Leírás***
code	*nem*	*Program kódok megjelenítésére szolgál. Fix szélességű betűtípussal rajzolódik. Forráskódok közzétételéhez lett kitalálva*
hr	*igen*	*Vízszintes elválasztó vonal beiktatása*
ul	*nem*	*Számozatlan felsorolási lista létrehozása*
ol	*nem*	*Számozott felsorolási lista létrehozása*

li	*nem*	*Felsorolási listában lista elem megadására szolgál.*
a	*lehetséges*	*hivatkozás létrehozása vagy oldalon belüli könyvjelző létrehozása, amire hivatkozhatunk. Hivatkozás esetén szükséges a href attribútum, ami megadja a link címét. Könyvjelző esetén a name attribútum megadása kell.*
img	*igen*	*Kép létrehozása. A kép elérési útvonala a src attribútumon keresztül adható meg. A width és height attribútumok megadásával szabályozható a kép mérete*

93. Táblázat: Alapvető HTML tag-ek (folytatás)

A tag-ek megadásánál érdemes ügyelni a zárási sorrendre. Például ha, az utolsó tag egy bekezdésen belül egy félkövér szöveg, akkor először a félkövér szöveg tag-je zárandó le, utána pedig a bekezdés. Ezen szabály be nem tartása megjelenítési problémákat okozhat böngésző függően.

HTML esetén bizonyos írásjelek (<, >, ") úgymond védett karakterek, amelyek a szövegekben nem alkalmazhatóak. Ezen karakterek és más speciális karakterek leírására a szabvány tartalmaz egy speciális karakter leíró rendszert, amely segítségével még az UTF-8 előtti időkben is lehetséges volt ékezetes betűk és egyéb egzotikus karakterek megjelenítésre kódlap függetlenül. Mára már csak a védett karakterek leírásánál van értelme ezek használatának. A legfontosabb speciális karakterek leírására használható kódokat az alábbi táblázat foglalja össze:

Karakter	***Leírója***
"	*"*
&	*&*
<	*<*
>	*>*
space	* *

94. Táblázat: Fontos HTML speciális karakterek

HTML esetén nem csak szövegeket és képeket jeleníthetünk meg, hanem adatokat is kérhetünk be űrlapok segítségével. Egy űrlap létrehozásánál meg kell adni a küldés módját, ami lehet GET vagy POST.

Az űrlap elemet a <form> tag definiálja. Ennek a legfontosabb attribútuma a method, ami meghatározza a küldés módját. Opcionálisan adat kódolási forma is meghatározható.

Minden beviteli mezőt amit a form továbbít a szerver felé el kell látni egy name attribútummal, ami meghatározza annak a változónak a nevét, amilyen néven a szervernek kezelnie kell az adatot.

A legtöbb vezérlő elem az input tag-en keresztül érhető el, amelyen belül a type attribútum adja meg a vezérlő elem típusát. Az input tag önlezáró. Ezen az elemen keresztül elérhető vezérlőket az alábbi táblázat foglalja össze. Az input tag vezérlőinek alapértelmezett érték is adható a value attribútum megadásával.

típus	***leírás***
text	*Szöveges beviteli mező*
password	*Jelszó beviteli mező*
button	*Általános gomb, amihez feladat javascript segítségével rendelhető.*
submit	*Küldés gomb. Erre kattintva az adatok el lesznek küldve a szervernek.*
reset	*Űrlap visszaállítása az alapértelmezett értékekre. Törli a felhasználói beviteleket*
file	*Fájl felöltésre használható elem. Egy szövegmezőt és egy tallózás gombot jelenít meg. A fájl feltöltés csak POST továbbítással működik.*
checkbox	*Jelölőnégyzet*
radio	*Választó négyzet*
number	*Szám bevitelre szolgáló mező. Kinézetében megegyezik a Text mezővel, de ebbe csak szám írható.*

95. Táblázat: Input tag fontosabb típusai

Két beviteli mező nem az input elem leszármazottja. A nagyobb szövegek bevitelére szolgáló textarea és a legördülő listák létrehozására használható select tag. A külön tag oka, hogy ezen elemek nem önlezáróak. A textarea két fontos attribútummal rendelkezik. Ezek a rows és cols tulajdonságok, amelyekkel meghatározható, hogy a szövegmező hány sorból álljon és soronként hány karaktert tartalmazzon.

A select tag egy lenyíló listát hoz létre, melyben az elemeket option tag-ek használatával tehetjük be. Az option tag-ek rendelkezhetnek value tulajdonsággal, ami a választott elem értékének meghatározására szolgál. A lenyíló listában az option tag-ben elhelyezett szöveg fog megjelenni, ebből adódóan ez sem önlezáró.

Hogyan tovább?

Az interneten számos weboldal foglalkozik a weblap készítés tudományával, ami igencsak összetett dolog. Az alapvető web programozási ismereteket a http://www.w3schools.com oldalról lehet véleményem szerint a legjobban megtanulni. Számos leírással és példával segítik a HTML, CSS és JavaScript technológiák használatának megtanulását.

Szép és ízléses weblapok létrehozásához nem csak programozási ismeretekre van szükség. Kell hozzá szépérzék és valamilyen grafikai program ismerete. A legtöbb esetben egy profi weboldal a pályafutását valamilyen grafikai programban kezdi látványtervként, amit aztán a web programozók átültetnek ténylegesen működő weblapra.

Ezen tervezési forma előnye, hogy átlátható képet ad a weblap későbbi megjelenéséről, valamint könnyebben módosíthatóak a tervezés során a dolgok, nem kell hozzá átírni a teljes mögöttes kódot.

Webre nem csak HTML segítségével lehet fejelszteni, hanem különböző böngésző beépülőkön is keresztül, mint az Adobe Flas, vagy a Microsoft Silverlight. Ezen termékekkel weblapot tervezni olyan szempontból nem előnyös, hogy csak PC-n működnek, azon belül is csak Windows-on rendesen.

A böngésző beépülők létrejöttének az oka A HTML korábbi limitáltsága volt. A HTML5-ről sok minden elmondható, csak az nem, hogy limitált lenne képességek terén. Ennek ellenére a Flash és a Silverlight és társaik nem fognak még egy jó darabig eltünni, mivel általában ezen beépülő megoldáokra sokkal jobb, komplexebb szerkesztők érhetőek el, amivel könnyebb igen látványos dolgokat alkotni.

Ennek a látványosságnak megvan az ára. Egyrészt platformhoz kötött lesz az oldal, valamint csak a FLASH-ből kiindulva nem biztos, hogy a legjobban képes egy beépülő kihasználni az adott hardver képességeit. Továbbá ha a beépülő biztonságilag sebezhető, akkor böngészőtől függetlenül fertőzhetővé válik a felhasználó gépe.

Verziókezelő rendszerek alapjai

A verziókezelő rendszerek történelme nagyjából egyidős a szoftverfejlesztéssel. Kezdetben ezen technológiákat csak nagy cégeknél nagy programok gyártására használták. A nyílt forráskód mozgalom megjelenésével megjelentek a nyílt forráskódú, szabadon elérhető verziókezelők. Ezen szoftverek megjelenésével megjelent az a fejlesztői nézet is, hogy programot nem érdemes fejleszteni verziókezelés nélkül.

Mielőtt konkrét rendszerekről lenne szó érdemes pár szót ejteni arról, hogy mire és miért jó az, ha szoftver fejlesztés közben verziókezelést is használunk?

- ***Biztonsági mentés***

A fájlokról a program írása közben tetszőleges számú biztonsági másolatot készíthetünk. A fájlok másolatai egyesével vagy teljes egészében visszaállíthatóak.

- ***Szinkronizáció***

Ennek akkor van értelme, ha több ember dolgozik egy időben ugyan azon a kódon. A verziókezelő rendszerek szinkronizálnak, így több ember különálló munkája összefűzhető egy nagy szoftver kóddá.

- ***Változások követése***

Minden egyes mentéshez üzenetek társíthatóak, így egyfajta naplót vezethetünk arról, hogy mit változtattunk és miért.

- ***Felelősség követése***

Minden egyes felhasználó mentései külön tárolódnak a rendszerben, így egyértelműen megállapítható, hogy a szoftver melyik részét ki írta.

- ***Elágazások és összefűzések***

Ennek segítségével létrehozható ugyan azon szoftver több verziója.

Verziókezelő rendszerek alapfogalmai

A verziókezelő rendszerek megértése az alapfogalmak megértésével lehetséges. A probléma „csak” ezzel annyi, hogy a magyar szakirodalom nem adaptálta még magyarul a szükséges kifejezéseket. Ezért az alapfogalmak ismertetésénél az angol megfelelőket használom, mivel nem szándékozok nyelvújító lenni.

Repository

Az adatbázis neve amely a verzió kezelt fájlokat tárolja.

Server

A szerver, amely a repository-t tárolja és kiszolgálja a fejlesztőknek. A fejlesztők a saját gépükön rendelkezhetnek privát repository-val is, azonban a többi fejlesztőnek csak az lesz elérhető, ami a szerveren van.

Client

A kliens gép, amely a szerverhez csatlakozik.

Working Set / Working Copy

A fejlesztő helyi gépén tárolt változata a repository-nak.

Revision number

A repository-ban tárolt változatokra ennek segítségével tudunk hivatkozni. Egy egyszerű számláló, amely minden check in művelet hatására egyel nő.

Head

A legutolsó, legnagyobb revision number hivatkozó kifejezése.

Trunk / Main

A repository valójában egy fa, amelybe a kód különböző ágait, változatait tudjuk tenni. Ezen ágak közül a Trunk ág, kitüntetett szereppel rendelkezik, mivel ez a fő változat tárolására szolgál.

Verziókezelő rendszerek műveletei

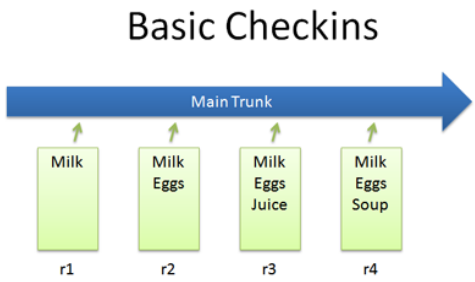
Clone

Egy meglévő repository összes változásának klónozása teljes klónozása egy másik szerverre.

Add

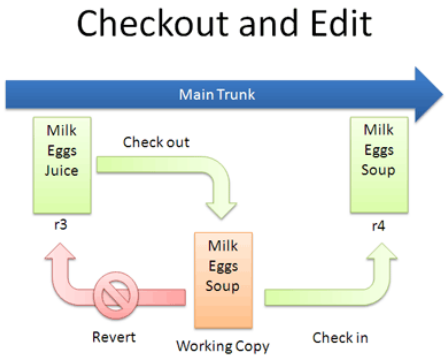
Hozzáadás. Felveszünk a repository-ba egy olyan fájlt, amelyet az korábban még nem tartalmazott. A felvétel után a fájl verziókezelhető.

Checkin



Az alap eltárolt verzióhoz képest az időbeli változások tárolása. Minden chekin műveletet megelőz egy checkout. Minden checkin utánn egyel nőni fog a repository revision number mezője.

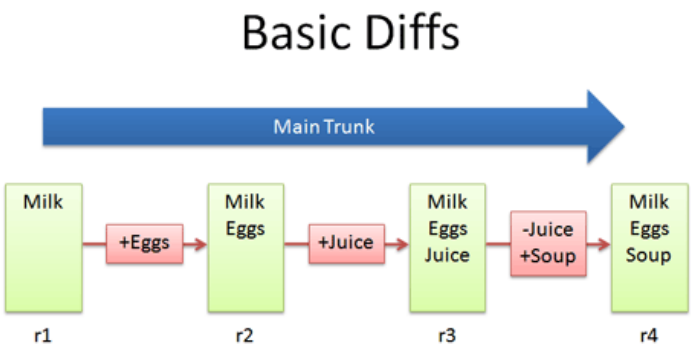
Checkout és Revert



A repository egyik ágából fájlok letöltése szerkesztésre, szinkronizáció a szerveren tárolt változattal. A checkout eredménye a Working Copy. A Working copy szerkesztése után két opciónk van. Vagy dobjuk a munkát egy revert művelettel, vagy check in segítségével mentjük a szerverre.

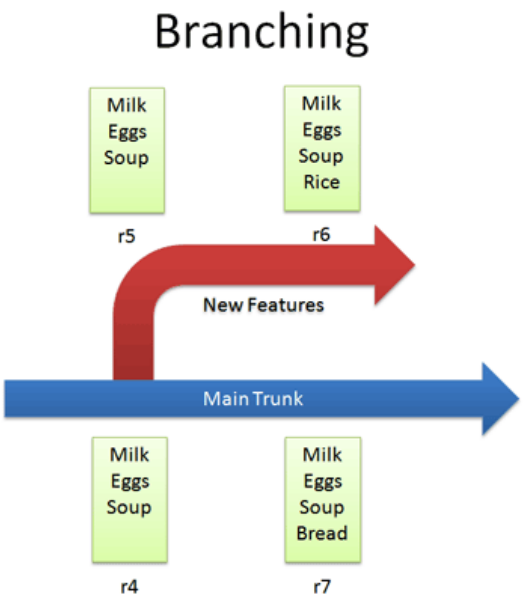
Diff és Patch

A diff művelet segítségével egy alap verzióhoz képest lehámozhatóak a módosítások a fájlról. Ha ezeket a módosításokat mentjük egy fájlba, akkor patch-et kapunk. A kapott patch fájl beintegrálható a szerver repostitory-ba. Patch fájl létrehozásának akkor van jelentősége, ha nem rendelkezünk az adott repository írási jogával. Ekkor a munkánk eredményét Patch formában átküldhetjük olyan személynek, aki be tudja olvasztani a repository megfelelő ágába.



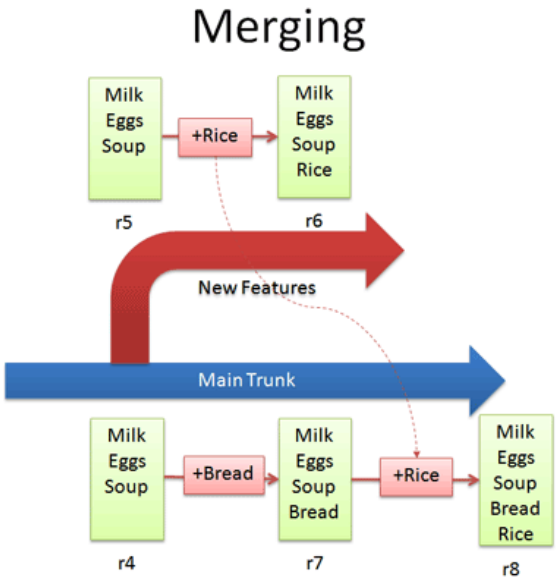
Branching

Egy meglévő kód ágból egy másik létrehozása. Előnye akkor van, ha a szoftver következő verzióját fejlesztjük, de még a régi verziót karban kell tartani.



Merging

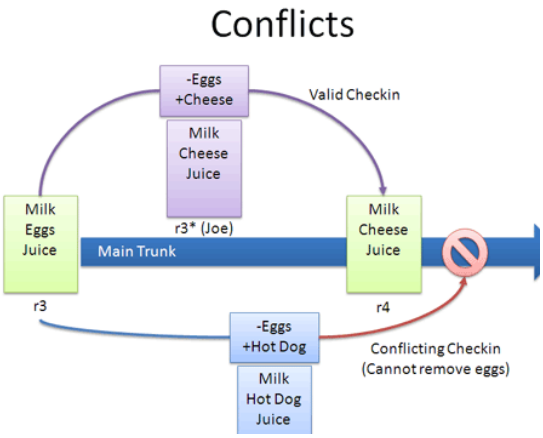
Változások beolvasztása. Nem egyszerű másolás. A verziókezelő rendszer figyelembe veszi a két fa közötti eltéréseket és csak az eltérések másolódnak át a beolvasztás cél ágába.

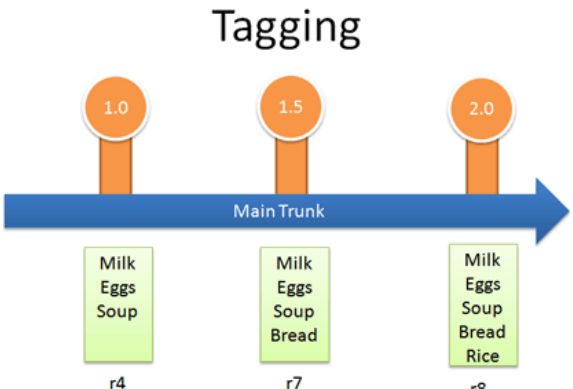


Conflict

Konfliktus akkor keletkezik, amikor a verziókezelő rendszer nem tud dönteni a változtatások sorsáról. Ez tipikusan akkor tud bekövetkezni, ha többen dolgoznak ugyan azon a kódon és egynél több ember azonos változtatást eszközöl. Például kitöröl egy sort. A leggyorsabb ember checkin művelete sikeres lesz, de a többieké nem, mivel nem lehet 2x törölni ugyan azt. Ebben az esetben konfliktus keletkezik, amit manuálisan kell nekünk megoldanunk.

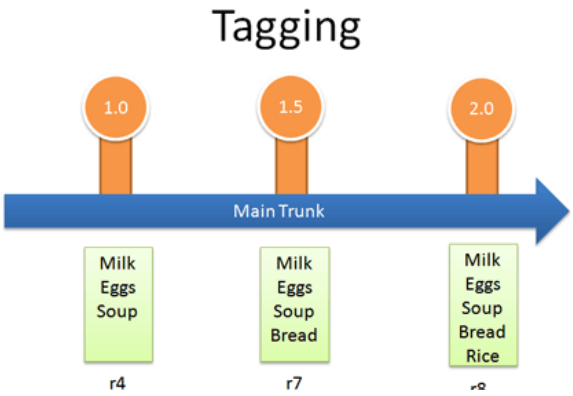
A megoldás lehet az, hogy újra letöltjük a szerver repository-t és ismét elvégezzük a módosításainkat és utána zárunk check in művelettel, vagy felülírjuk a repositoryban tárolt fájlt a saját verziónkkal.





Tagging

Adott revízió szám megjelölése egy szöveges leírással. Például hasznos a program fő verzióinak azonosítására.



Verziókezelő rendszerek fajtái

Alapvetően két típusú verziókezelő rendszert különböztetünk meg. Vannak a szerver – kliens felépítésűek és vannak az elosztott rendszerek.

A szerver-kliens felépítésű rendszereknél a kliens oldalon csak egy working copy tárolásra van lehetőségünk. A repository-k a szerveren tárolódnak. A szerver és a kliens lehet egy és ugyan azon a gépen, viszont nem éppen praktikus. Ennek oka az, hogy a szerver-kliens felépítésű rendszerek esetén a szerver futtatásához jóval több erőforrás kell.

Az elosztott rendszerekben a kliens oldalon nem csak a workig copy tárolódik, hanem egy helyi repository is, amely csak azon a gépen érhető el. Előnye, hogy a kliens a saját repository-ba is tud menteni kisebb-nagyobb változásokat, amelyeket bármikor visszagörgethet. Ezen változások közül a felhasználó jelöli ki, hogy melyik kerüljön át a szerver repository-ba is. Ezen művelet az ilyen rendszerek esetén nem check in kifejezéssel jelölt, hanem push műveletnek nevezik. A központi repositroy letöltését pedig pull műveletnek.

A verziókezelő rendszerek közös tulajdonsága modelltől függetlenül, hogy leginkább szöveges adatok kezelésére alkalmasak, mivel két szöveges fájl között az eltérések megtalása igen egyszerű feladat. Így elég csak mentéskor a különbségeket (diff) tárolni. Bináris fájlok esetén még nincs effektív, jól működő diff algitmus kidolgozva, így a verziókezelő rendszerek bináris fájlok esetén nem is keresnek változásokat, inkább letárolják a teljes fájlt mégegyszer.

Verziókezelő szoftverek

A könyv elején említettem, hogy verziókezelő rendszerből annyi van, mint égen a csillag és nagyjából mindegyik rendszer ugyan azon képességekkel rendelkezik. Így kezdőként igen nehéz eldönteni, hogy melyik rendszert is használjuk. Személy szerint én a GIT és SVN rendszerek használatát preferálom, mivel ez a két rendszer rendelkezik a legnagyobb szoftver támogatással.

Az SVN és GIT rendszerek használata Windows rendszerek esetén mondhatni pofon egyszerű a TortoiseSVN és TortoiseGIT Shell bővítményeknek köszönhetően. Ezen programokkal megszokott shell könyezetből tudjuk vezérelni a verziókezelésünk működését.

Linux renszerekre is számos grafikus program érhető el verziókezelési célokra. Az alábbi szeVkcióban a GIT és SVN rendszerek parancssoros klienseinek használatát mutatom be.

Művelet	***Git parancs***	***SVN parancs***
Chekout	`Git clone http://cím/tarolo.git`	`Svn co http://cím`
Update	`Git pull`	`Svn up`
Add	`Git add fájlnév`	`Svn add fájlnév`
Delete	`Git rm fájlnév`	`Svn delete fájlnév`
Revert	`git checkout -- fájénév`	`Svn revert fájlnév`
Commit	`Git commit -m ”leírás”`	`Svn ci -m ”leírás”`

96. Táblázat: GIT és SVN rendszerek alapvető parancsai

GIT és SVN rendszerek esetén is a kód központi tárolásához kell egy szerver. A GIT beüzemelése ilyen szempontból könnyebb, mint az SVN esetén, de nem érdemes sokat vesződni saját verziókezelő szerver készítésével, üzemeltetésével, mivel az interneten számos weboldal kínál verziókezelési szolgáltatást.

Ezek közül kiemelkedő a Github. A Github leginkább a GIT verziókezelő köré épít, de használhatók a weboldal szolgáltatásai SVN kliens segítségével is. További előnye a Github oldalnak, hogy a tárolók lehetnek privát tárolók is, nem csak publikusak. A privát tárolók szolgáltatásért egy minimális havi díjat fizetni kell, cserébe megbízható infrastruktúra mögött tárolhatjuk a kódunkat.

A legtöbb verziókezelést biztosító oldal előnye és egyben hátránya az ingyenesség. Az ingyenesség előny, ha nyílt forráskódú szoftvert fejlesztünk, hátrány viszont akkor, ha a kódunkat privát használatra szánjuk.

A Github használata továbbá azért előnyös, mert a legtöbb nyílt forráskódú szoftver itt található meg, így ha ide töltjük fel a projektünket, akkor itt a legnagyobb a valószínűsége annak, hogy mások is csatlakoznak a projekt fejlesztéséhez.

21. PIC Projektek

1. LED villogtatás

Az első projektünk igencsak egyszerű. Fix időközzel egy LED-et felvillantatunk, majd kialtatunk. A projekt kapcsolási rajza nem túl bonyolult Az eszközünk RB0 lábára egy LED-et és egy áramerősség korlátozó ellenállást kötünk. A korlátozó ellenállás mértéke függ a LED-en eső feszültségtől és a rajta átfolyó áramerősségtől. A korlátozó ellenállást a következő képletekkel szokás számolni:

Soros kapcsolás esetén $R=\frac{U_T-n\cdot U_L}{I_L}$ *, párhuzamos kapcsolás esetén* $R=\frac{U_T-U_L}{n\cdot I_L}$

A képletben szereplő betűjelek jelentései:

n *: LED-ek darabszáma*

U_T *: Tápfeszültség, amelyre a LED csatlakozik*

U_L *: A LED működéséhez szükséges feszültség*

I_L *: A LED által igényelt áramerősség*

LED-es projekteknél hasznos segédlet tudni, hogy színtől és a dióda anyagától függően mekkora nyitófeszültség kell a dióda működéséhez. Az alábbi táblázat a LED-ek tipikus nyitófeszültségeit tartalmazza szín függvényében.

Szín	***Hullámhossz (nm)***	***Nyitófeszültség (V)***
Infravörös	$\lambda > 760$	$U < 1{,}9$
Piros	$610 < \lambda < 760$	$1{,}63 < U < 2{,}03$
Narancs	$590 < \lambda < 610$	$2{,}03 < U < 2{,}10$
Sárga	$570 < \lambda < 590$	$2{,}10 < U < 2{,}18$
Zöld	$500 < \lambda < 570$	$1{,}9 < U < 4$
Kék	$450 < \lambda < 500$	$2{,}48 < U < 3{,}7$
Lila	$400 < \lambda < 450$	$2{,}76 < U < 4$
Ultra ibolya	$400 < \lambda$	$3{,}1 < U < 4{,}4$
Rózsaszín	*Több típus*	*~3,3*
Fehér	*Széles spektrum*	*~3,5*

97. táblázat. LED-ek hullámhosszai és tipikus nyitófeszültségük

A projekt forráskódja a következő lesz:

```
void main()
{
     ANSEL  = 0;
     ANSELH = 0;
     C1ON_bit = 0;
     C2ON_bit = 0;
     TRISA = 0x00;
     TRISB = 0x00;
     TRISC = 0x00;
     TRISD = 0x00;
     TRISE = 0x00;

     while (1)
     {
          PORTB = 1;
          Delay_ms(1000);
          PORTB = 0
          Delay_ms(1000);
     }
}
```

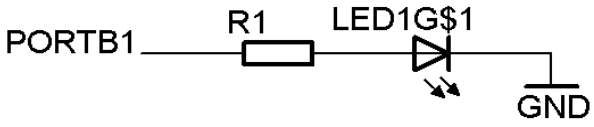
A fő függvény egy pár konfigurációs regiszter beállításával kezdődik. Ezek minden programra jellemzőek. Az ANSEL és ANSELH regiszterek nullázása azt állítja be, hogy minden láb digitális üzemmódban működik.

A C1ON_bit és C2ON_bit regiszterek a komparátorokat kapcsolják ki vagy be. Jelen esetben ki.

A TRISA, TRISB, TRISC, TRISD, TRISE regiszterek az A, B, C, D, E portok irányait állítják be. Ahol a TRIS regiszterek értéke 0-t vesz fel, ott a hozzá tartozó port lába kimenet lesz. Ahol pedig 1 lesz az értéke, ott a hozzá tartozó port lába bemenetként fog üzemelni. Példaképpen, ha azt szeretném, hogy a B port fele (0-3 lábig) bemenet, másik fele (4-7 lábig) kimenet legyen, akkor TRISB értéke a következő lesz:

```
TRISB = 0x0F;
```

Ezután egy végtelen ciklusban egy nagyon egyszerű kód van elhelyezve. Először B portra kiküld egy egyest (bekapcsolja RB0 jelölésű lábat), majd 1000ms (1 mp) várakozás után egy 0 értéket küld Port B-re (kikapcsolja az előzőleg bekapcsolt RB0 jelzésű lábat).



2. LED-es futófény

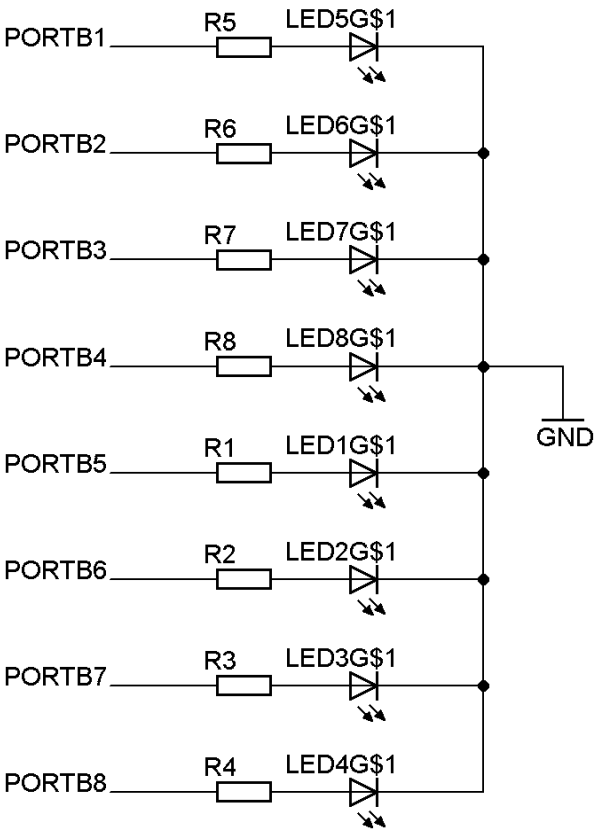
Egy LED kezelése nem túl izgalmas. Mivel a mikrovezérlőnk legtöbb portja 8 bites, ezért építünk egy 8 bites futófényt. Ezt úgy valósítjuk meg, hogy a B port-ra 8db LED-et és 8 db áramkorlátozó ellenállást kötünk. Ezt a 8 LED-et rengeteg módon tudjuk működtetni. Ezen példában a „klasszikus" oda-vissza futófényt fogjuk megvalósítani.

A program kivitelezése szempontjából használhatnánk tömböket is, ahol minden egyes lépés fix kimeneti értékét rögzítjük. Sokkal elegánsabb megoldás azonban, ha a C beépített bit eltolás jobbra és bit eltolás balra operátoraival implementáljuk a programot, ami a következőképpen fog kinézni:

```
int i;
char out;

void main()
{
	ANSEL  = 0;
	ANSELH = 0;
	C1ON_bit = 0;
	C2ON_bit = 0;
	TRISA = 0x00;
	TRISB = 0x00;
	TRISC = 0x00;
	TRISD = 0x00;
	TRISE = 0x00;

	while (1)
	{
		out = 1;
		for (i=0; i<8; i++)
		{
			PORTB = out;
			out = 1 << i;
			Delay_ms(100);
		}
		out = 0x80;
		for (i=0; i<8; i++)
		{
			PORTB = out;
			out = 0x80 >> i;
			Delay_ms(100);
		}
	}
}
```

Az out segédváltozó tartalmazza a kimenetre küldendő állapotot. Először ennek az értéke 1 lesz. Egy 8 lépéses ciklusban ezen érték jelenik meg a kimeneten, majd az out tartalma el van tolva lépésenként egy bittel balra. Így az első lépés végén a kimeneti érték már 2 lesz, majd 4, 8, 16, 32, 64, és végül 128.

Ezután az out értékét manuálisan 128-ra állítva (hexa 80) ismét egy 8 lépéses ciklus következik, ahol az out változó tartalma lépésenként egy bittel jobbra van eltolva. Így jön létre az oda-vissza futó fény effektus. További minták hasonlóan egyszerű program segítségével állíthatóak elő.


PORTB1
R5
LED5G$1
PORTB2
R6
LED6G$1
PORTB3
R7
LED7G$1
PORTB4
R8
LED8G$1
GND
PORTB5
R1
LED1G$1
PORTB6
R2
LED2G$1
PORTB7
R3
LED3G$1
PORTB8
R4
LED4G$1

3. Több funkciós LED-es futófény

Az előző projektben tárgyalt villogó alapötletből kiindulva született meg ez a projekt. A funkciók közötti váltáshoz kell egy bemeneti gombot kezelni, amely segítségével kiválasztható lesz a villogási mód. A LED-ek kimeneti kapcsolása megegyezik az előző projektben ismertetettekkel. A program öt villogási módot kínál, melyek a következőek:

1. *Az összes LED villog.*
2. *A LED sor két széléről indul a villogás, majd halad a közepe felé. A közép elérése után a fény kifelé halad a soron.*
3. *Futófény 2 LED használatával, majd mikor kiért a sor végére, akkor minden második LED megvilágításával halad a sor elejére.*
4. *Futófény 4 LED használatával.*
5. *4 LED villogtatása ellentétesen.*

Bemenet kezeléshez először is a kiválasztott portunkhoz tartozó TRIS regiszter értékét kell helyesen meghatároznunk. Ezen projekt a D port 0. lábát használja erre a célra, ezért a TRISD regiszter értéke a következő lesz:

```
TRISD = 0x01;
```

A portok egyes lábainak állapotának lekérdezésére és módosítására a MikroC kínál tag értékeket minden egyes PORT értéken belül. Tehát a PORT regiszterek nem egyszerűen számértékekként vannak definiálva MikroC esetén, hanem egy összetett struktúraként. Ha mondjuk a C port 4. lábát szeretném egyesre állítani anélkül, hogy a jelenlegi konfigurációt átállítanám rajta, akkor először a jelenlegi PORTC értéket el kellene mentenem egy átmeneti változóba, majd növelnem a megfelelő értékkel és újra kiküldeni. Ez konkrétan így nézne ki:

```
char tmp;

tmp = PORTC;
tmp += 0x08; //4. láb értéke 8, mivel 2^4 = 8
PORTC = tmp;
```

Ezen kódrészlet helyett azonban használhatom a következőt:

```
PORTC.B3 = 1;
```

Minden egyes port minden lábához hozzá tudok férni a B0-tól B7-ig tartó bit tagértékekkel. Az e módon történő kezelés nagyon hasznos bemenetek kezelésénél, főleg ha több bemenetet kell egymástól függetlenül kezelnem.

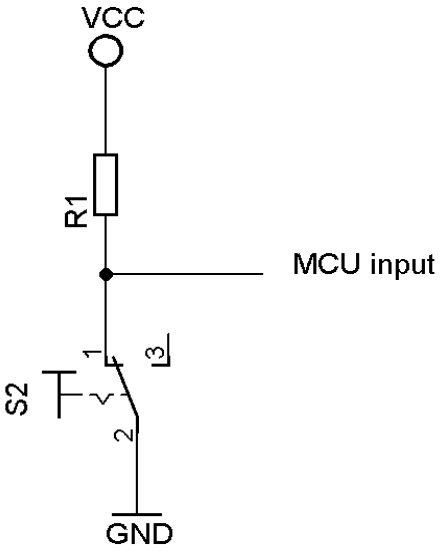
Egy digitális bemenetet a következőképpen tudunk kezelni:

```
if (PORTB.B0 == 1)
{
     while (PORTB.B0 == 1) {}
     //gombnyomáskor végrehajtandó kód
}
```

A fenti kódrészlet a PORTB első lábát használja bemenetnek. A kezelés érthető módon egy if feltétellel van megvalósítva. Ami újdonság, a beiktatott üres blokkal rendelkező while ciklus. Ezen ciklus arra szolgál, hogy blokkolja a vezérlést, amíg a gomb nyomva van. Erre azért van szükség, mert a „nagy" órajel miatt a gomb megnyomásához tartozó utasítások többször is (16 Mhz mellett papírforma szerint 4 milliószor) végrehajtódhatnának egy másodperc alatt.

A gomb bemenet kapcsolási rajza az alábbi ábrán látható. A gomb kialakításánál cél a belső FET-es

felépítés miatt, hogy alapértelmezetten földponton legyen a bemenet, és csak akkor kerüljön tápfeszültség alá, ha ez szükséges. Ha a gomb kikapcsolt állapotában nincs földelve a bemenet, az okozhat és valószínűleg fog is okozni kiszámíthatatlan működést. A kapcsolásban szerepló R1 értéke 1 KOhm és 10 KOhm közötti legyen. Az ellenállás nagysága befolyásolja a kapcsolási sebességet. Ez nyomógomb esetén általában nem szokott számítani az esetek többségében.



A projekthez tartozó forráskód:

```
#define IDO 100

int i, mode;
static char mode2[] = { 0x81, 0x42, 0x24, 0x18 };
static char mode3[] = { 0x03, 0x0C, 0x30, 0xC0, 0x40, 0x10, 0x04,
→ 0x01 };
static char mode4[] = { 0x0F, 0x1E, 0x3C, 0x78, 0xF0 };

void main()
{
     TRISB = 0x00;
     TRISC = 0x00;
     TRISD = 0x01;
     TRISA = 0x00;
     ANSEL = 0;
     ANSELH = 0;
     C1ON_bit = 0;
     C2ON_bit = 0;
     mode = 0;
     while (1)
     {
          if (PORTD.B0 == 1)
          {
               while (PORTD.B0 == 1) {}
               ++mode;
               if (mode > 4) mode = 0;
          }
          switch (mode)
          {
               case 0:
                    PORTB = 0xFF;
                    Delay_ms(IDO);
                    PORTB = 0x00;
                    Delay_ms(IDO);
                    break;
               case 1:
                    for (i=0; i<4; i++)
                    {
                         PORTB = mode2[i];
                         Delay_ms(IDO);
                    }
                    for (i=3; i>=0; i--)
                    {
                         PORTB = mode2[i];
                         Delay_ms(IDO);
                    }
                    break;
               case 2:
                    for (i=0; i<8; i++)
                    {
                         PORTB = mode3[i];
                         Delay_ms(IDO);
```

```
                    }
                    for (i=7; i>=0; i--)
                    {
                        PORTB = mode3[i];
                        Delay_ms(IDO);
                    }
                    break;
                case 3:
                    for (i=0; i<5; i++)
                    {
                        PORTB = mode4[i];
                        Delay_ms(IDO);
                    }
                    for (i=4; i>=0; i--)
                    {
                        PORTB = mode4[i];
                        Delay_ms(IDO);
                    }
                    break;
                case 4:
                    PORTB = 0x0F;
                    Delay_ms(IDO);
                    PORTB = 0xF0;
                    Delay_ms(IDO);
                    break;
            }
        }
}
```

A program a 2., 3. és 4. villogási módot tömbök segítségével valósítja meg. A tömbök statikusként vannak megjelölve. Ekkora méretek esetén nem lenne kötelező, azonban ha a későbbiekben bővülne a program, nem árt, ha van elég RAM az eszközben. Az i változó a tömb elemeinek indexelésére szolgál, míg a mode változó a villogási módot határozza meg.

A fő végtelen ciklus elején van lekezelve a gombnyomás. Ez azt jelenti, hogy villogási módot a felhasználónak akkor van lehetősége választani, ha az éppen aktuálisan ismételt villogási minta végére ért. Az utasítások eltérő hosszúsága miatt ez némi várakozási időt eredményezhet a felhasználói oldalon. Elegánsabb megoldás lenne megszakítás használata.

A villogási módokat egy összetett switch-case utasítás ág kezeli le. A program a LED-ek állapotai között fix 100 milliszekundum várakozást használ. Ezen idő módosítható a program elején elhelyezett IDO makró utasítás átírásával.

4. Számlálók

A számláló IC-k rengeteg aszinkron logikai hálózat alapját képezik. Ezen hálózatokban általában egy speciális számláló funkciót megvalósító IC és egy órajel generátor szokott helyet kapni. Órajel generálásra egy 555-ös időzítő IC-t szokás használni. Azonban adódhat a helyzet, hogy szükségünk van egy számlálóra a projektünkben, de éppenséggel nincs otthon, és a kedvenc boltunk is zárva van, ahol beszerezhetnénk a szükséges alkatrészt. Ilyenkor jön jól, hogy ha van kéznél egy PIC mikrovezérlő.

PIC-et használni egyetlen számláló helyett anyagi okok miatt nem éppen kifizetődő ötlet, azonban ha mondjuk 4 különböző számlálót kell leváltanunk, akkor már megérheti PIC-et használni.

Dekád számláló

A dekád számláló egy olyan számláló, amely kimenetein az értékek kettő hatványai szerint következnek. Ezen számláló programozását külön a könyvben nem tárgyalom, mivel az előző LED-es futófény projektben le lett programozva a működése.

Bináris számláló

Egy bináris számláló megépítése igen egyszerű. Csupán egy ciklust kell készítenünk, a ciklusmagban meg egy értéket növelünk, majd az értéket kiküldjük egy kimeneti portra. Ezután tetszőleges időt várakozunk. A várakozási idő fogja meghatározni a számláló frekvenciáját az ismert, fizikából mindenki által tanult képlet alapján:

$$f = \frac{1}{T}$$

Bináris számlálók két fő típusa a felfelé és lefelé számláló számlálók. A számláló program mindkét típust megvalósítja. A program:

```
int i = 0;
#define LEFELEIS 0
#define DELAY 1000

void main()
{
     ANSEL = 0;
     ANSELH = 0;
     C1ON_bit = 0;
     C2ON_bit = 0;
     TRISA = 0;
     TRISB = 0;
     TRISC = 0;
     TRISD = 0;

     while (1)
     {
          for (i=0; i<256; i++)
          {
               PORTB = i;
               Delay_ms(DELAY);
          }
          if (LEFELEIS)
          {
```

```
                for (i=255; i>=0; i--)
                {
                      PORTB = i;
                      Delay_ms(DELAY);
                }
           }
      }
}
```

A program alapértelmezett konfiguráció szerint csak a felfelé számlálást valósítja meg 8 biten. Azonban a LEFELEIS konstans 1-es értékre állításával már fel-le bináris számlálót kapunk. A DELAY konstans a várakozási időt befolyásolja.

Johnson számláló

*A Johnson kód egy speciális bináris kód, amely N biten 2*N kódszóból áll. A kódszavak képzési szabálya az, hogy a következő kód az előzőtől csupán egy biten tér el. A képzési forma a következő példából jól kivehető. A példában a 8 bites Johnson kód szerepel.*

```
00000000
00000001
00000011
00000111
00001111
00011111
00111111
01111111
11111111
11111110
11111100
11111000
11110000
11100000
11000000
10000000
```

A számláló megvalósítása:

```
int i = 0;
const char johnsonk[16] =
                          { 0x00, 0x01, 0x03, 0x07,
                            0x0F, 0x1F, 0x3F, 0x7F,
                            0xFF, 0x7F, 0x3F, 0x1F,
                            0x0F, 0x07, 0x03, 0x01
                          };

#define DELAY 1000

void main()
{
      ANSEL = 0;
      ANSELH = 0;
      C1ON_bit = 0;
      C2ON_bit = 0;
      TRISA = 0;
```

```
	TRISB = 0;
	TRISC = 0;
	TRISD = 0;

	while (1)
	{
		for (i=0; i<=16; i++)
		{
			PORTB = johnsonk[i];
			Delay_ms(DELAY);
		}
	}
}
```

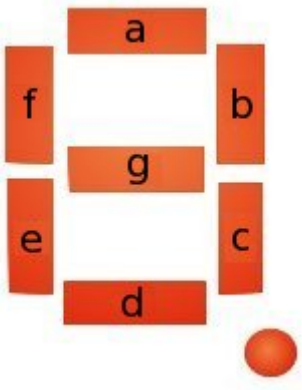
A megvalósításban definiálva van egy 16 elemű tömb, amely konstansként van megadva. Ezáltal a tömb tartalma a programmemóriában fog elhelyezkedni (mivel nem módosulhat futás közben a tömb tartalma). A tömb a 8 bites Johnson kód kódszavait tárolja. Ezek megadása a kevesebb gépelés miatt hexadecimális alakban van megadva.

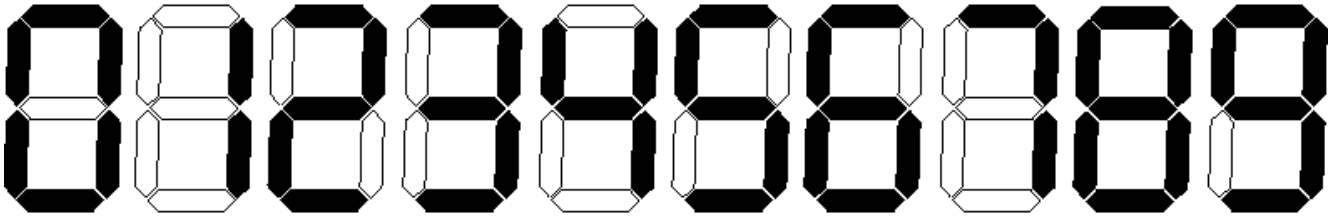
A főprogramban a szükséges konfiguráció után végtelenítetten ezen kódszavak kerülnek kiküldésre sorban a PORTB lábaira. A várakozási időt a DELAY konstans szabályozza.

5. Hétszegmenses kijelzők vezérlése

Alapvető kezelési információk

A 7 szegmenses kijelző 7 darab LED-ből áll, amelyek 8-as formában vannak elrendezve. Ennek segítségével számokat tudunk megjeleníteni 0 és 9 között, illetve egyéb olyan ábrákat, amelyek megjeleníthetőek egy ilyen kijelzőn. A szegmensek számozása az angol ABC kisbetűivel van megoldva a-tól g-ig. Egyes kijelzők valójában 8 szegmensesek, mivel tartalmaznak egy decimális pontot is. A decimális pont kijelzőtől függően D-vel, vagy dp-vel van jelölve.

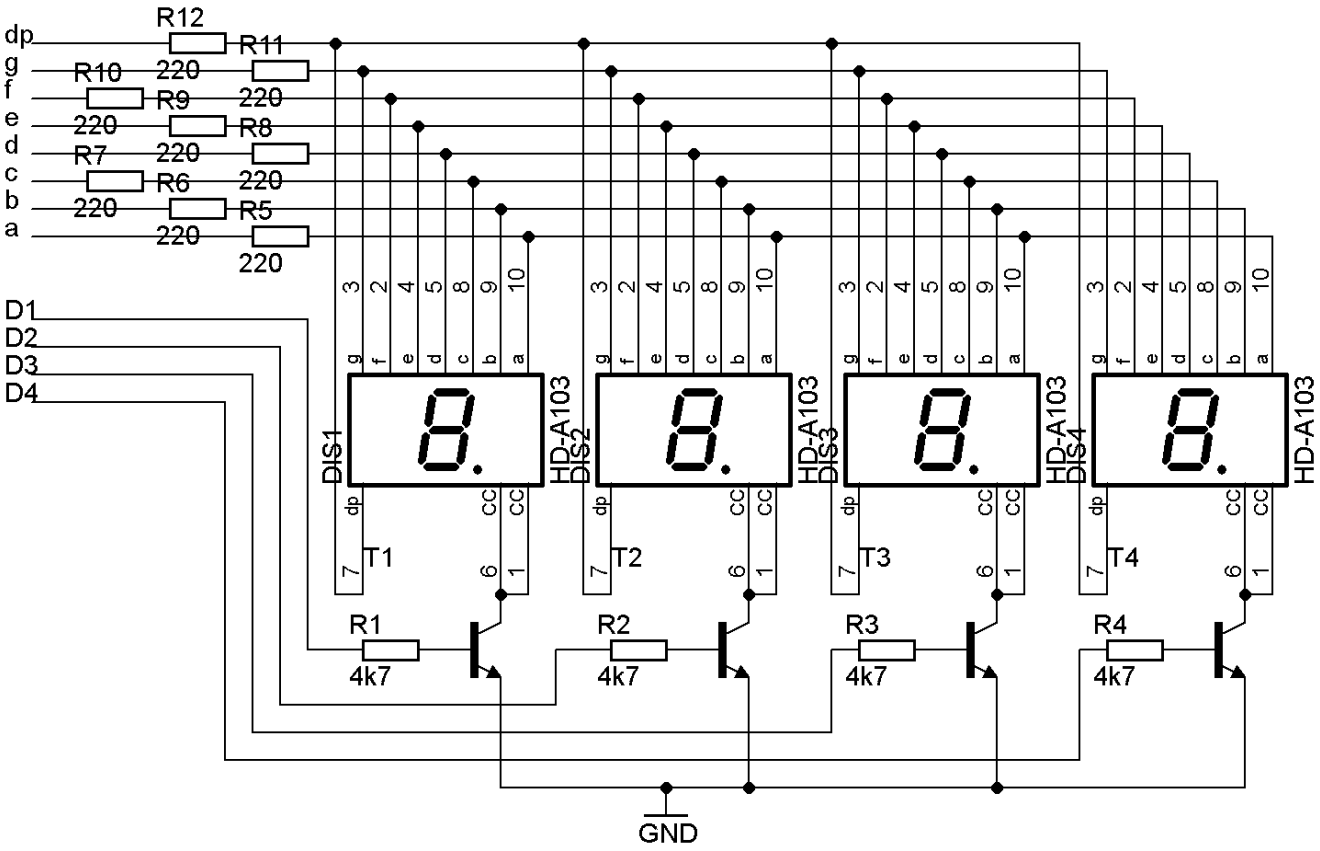




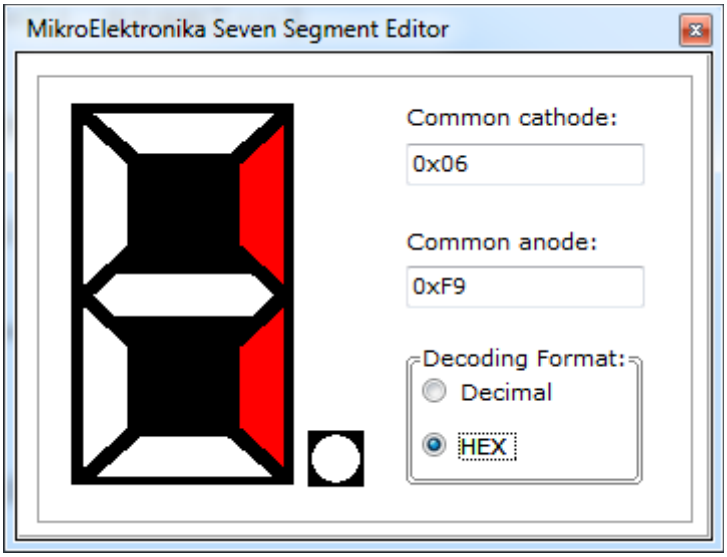
Megvalósítástól függően kétfajta 7 szegmenses kijelző létezik: közös anódos és közös katódos felépítésű, attól függően, hogy a LED-ek melyik lába van közösítve. A különbség a kettő megvalósítás között annyi, hogy a vezérlőjelek egymás ellentettjei.

*Egy kijelzővel azonban nem sok mindent lehet csinálni, ezért ezen kijelzőket multiplexelni szokták, így többet összekapcsolva. Ez azt jelenti, hogy az egyes kijelzők egymás után vannak kapcsolgatva, méghozzá olyan gyorsan, hogy az emberi szem ne tudja követni a folyamatot. Előnye ezen megoldásnak, hogy 4 kijelző esetén nem 7*4 vezeték kell a vezérléshez. A feladat ellátásához bőven elég 7+4 vezeték, ami 11-et jelent. Így egy kisebb mikrovezérlővel is megoldható a feladat.*

Mivel a mikrovezérlők által kibocsátott/elnyelni képes áram igen kicsi, ezért egy tranzisztoros kapcsolással szokás a tápfeszültséget szegmensenként kapcsolni. A kijelző felépítésétől függően itt PNP vagy NPN tranzisztort kell használni.



Első kihívást, amit mikrovezérlős vezérlés esetén át kell hidalnunk, a megfelelő vezérlőkódok kitalálása jelenti. Használhatnánk speciális cél IC-t a célra, mint a 7447-es (közös anód) vagy 7448-as (közös katód) BCD/7 szegmenses dekódoló. Azonban, ha a projekt engedi, akkor célszerű egy olyan mikrovezérlőt választani a célra, amely rendelkezik elég lábbal.



A szoftveres vezérlésnél némi matekot és papírmunkát okozhatna a vezérlőjelek kitalálása. Szerencsére a MikroC fejlesztői gondoltak erre a problémára, ezért a fejlesztőkörnyezetbe építettek egy dekódoló programot. Ezen program a MikroC Tools menüjén belül 7 Segment Editor menüpont alatt érhető el. A program a vezérlőjeleket a bekattintott (megvilágítani kívánt) szegmensek alapján állítja elő közös anódos és közös katódos kijelzőkre is. A kimenet kódolása választható decimális és hexadecimális formában is. Ezen vezérlőjelek alapján meg lehet alkotni egy függvényt, amely a kapott decimális számot kimenetre küldendő vezérlőjellé alakítja át. A függvény megalkotásakor érdemes alkalmazni a C switch-case vezérlési szerkezetét if-else elágazások helyett.

```
unsigned char BCD7Szegmens(unsigned short num)
{
      switch (num)
      {
            case 0: return 0x3F;
            case 1: return 0x06;
            case 2: return 0x5B;
            case 3: return 0x4F;
            case 4: return 0x66;
            case 5: return 0x6D;
            case 6: return 0x7D;
            case 7: return 0x07;
            case 8: return 0x7F;
            case 9: return 0x6F;
            default: return 0xFF;
      }
}
```

A fenti kódrészletben szereplő függvény lényegében egy BCD/7 szegmenses dekódolást végez közös katódos kapcsolás számára. Amennyiben a függvény olyan számot kap, amelyet nem tud dekódolni, akkor a kijelzőn nem fog érték megjelenni.

Egy kijelzőn számlálás

Első mintaprogramunk igen egyszerű lesz. Egyetlenegy szegmensen valósít meg egy számlálót, amely 0-tól 9-ig számol másodpercenként egy váltással.

```
int i;

unsigned char BCD7Szegmens(unsigned short num)
{
      switch (num)
      {
            case 0: return 0x3F;
            case 1: return 0x06;
            case 2: return 0x5B;
            case 3: return 0x4F;
            case 4: return 0x66;
            case 5: return 0x6D;
            case 6: return 0x7D;
            case 7: return 0x07;
            case 8: return 0x7F;
            case 9: return 0x6F;
            default: return 0xFF;
      }
```

```
}

void main()
{
	ANSEL = 0;
	ANSELH = 0;
	C1ON_bit = 0;
	C2ON_bit = 0;
	TRISA = 0;
	TRISB = 0;
	TRISC = 0;
	TRISD = 0;

	PORTC = 0x01;

	while (1)
	{
		for (i=0; i<10; i++)
		{
			PORTD = BCD7Szegmens(i);
			Delay_ms(1000);
		}
	}
}
```

A C portra kiküldött egyes érték a szegmenset kapcsolja be, míg a D portra kiküldött szám a szegmenseket kapcsolja. Több kijelző kezeléséhez a C port értékét kell módosítani kettő hatványai szerint. Erre példát a következő program nyújt.

Több kijelzőn számlálás

Több kijelző multiplexelt vezérlésére két opció van. Használhatunk megszakítást, de enélkül is megoldható a program. Megszakításos kezelés esetén szétválik a számláló kezelése és kijelzők vezérlése funkció, míg nem megszakítást alkalmazó megoldások esetén ez együtt működik. Ez olyan nehézséget okoz, hogy a kijelzőket gyorsan kell frissíteni. Emiatt elmászhat a számláló funkció időzítése.

Ezen példában az egyszerűség kedvéért egyelőre csak a megszakítás nélküli vezérlés lesz tárgyalva. A megszakításos vezérlésről későbbiekben a számlálók kapcsán lesz szó.

```
int i;

unsigned char BCD7Szegmens(unsigned short num)
{
	switch (num)
	{
		case 0: return 0x3F;
		case 1: return 0x06;
		case 2: return 0x5B;
		case 3: return 0x4F;
		case 4: return 0x66;
		case 5: return 0x6D;
		case 6: return 0x7D;
		case 7: return 0x07;
		case 8: return 0x7F;
```

```
            case 9: return 0x6F;
            default: return 0xFF;
        }
}

void main()
{
      ANSEL = 0;
      ANSELH = 0;
      C1ON_bit = 0;
      C2ON_bit = 0;
      TRISA = 0;
      TRISB = 0;
      TRISC = 0;
      TRISD = 0;
      i = -1;

      while (1)
      {
            ++i;
            if (i > 9999) i = 0;

            //ezresek számának meghatározása
            PORTC = 0x08;
            PORTD = BCD7Szegmens(i/1000);
            Delay_ms(5);

            //százasok számának meghatározás
            //pl i=1337 i/100 után i= 13 majd i%10 = 3
            PORTC = 0x04;
            PORTD = BCD7Szegmens((i/100)%10);
            Delay_ms(5);

            //tizesek számának meghatározása
            //pl i=1337 i/10 után i= 133 majd i%10 = 3
            PORTC = 0x02;
            PORTD = BCD7Szegmens((i/10)%10);
            Delay_ms(5);

            //egyesek számának meghatározása
            PORTC = 0x01;
            PORTD = BCD7Szegmens(i%10);
            Delay_ms(5);
      }
}
```

A fenti program egyszerű osztogatással kijelzőnként meghatározza a kiküldendő értéket. A programban alkalmazott késleltetés kapcsolásonként vagy kijelzőnként változhat.

Óra készítése megszakítások nélkül

Ezen projekt alapötlete igen egyszerű kezelési szempontból. A trükk a pontosság eléréséhez az, hogy a kijelzők frissítése közötti várakozási időket úgy kell megadni milliszekundumban, hogy ezek összege (x) 1000

milliszekundumnak osztója legyen. Ekkor a kijelzőm 1 másodperc alatt 1000/x alkalommal fog villogni.

Tehát ha a kijelzőim összegzett frissítési ideje 10ms, akkor 1 másodperc alatt 100 alkalommal fog felvillanni ugyanaz a kijelző ugyanazzal az értékkel. Ezt már az emberi szem nem tudja megkülönböztetni egymástól, így csak annyit látunk, hogy folyamatos lesz a kiírás.

Ebben az esetben minden 100. kiírás után kell növelnem a másodperc számlálóm értékét 1-gyel, hiszen így jön ki az egy másodperces késleltetésem a számlálásban. Amennyiben a számlálóim frissítését nem tudom kihozni 1000 osztójaként, akkor az órám előbb-utóbb többet-kevesebbet fog késni, sietni.

A pontatlanság jelensége hosszú távon pontos időzítések mellett is fennállhat. Ennek elkerülése érdekben az órát érdemes lenne ellátni egy RTC modullal is.

A következő program perc és másodperc pontossággal számol 99 perc 60 másodpercig 4db 7 szegmenses kijelzőn. A 4 kijelző frissítése közötti összegzett késleltetés 5ms.

```
int perc, masodperc, i, clk;

unsigned short Decode(unsigned short input)
{
     switch(input)
     {
          case 0:
               return 0x3F;
          case 1:
               return 0x06;
          case 2:
               return 0x5B;
          case 3:
               return 0x4F;
          case 4:
               return 0x66;
          case 5:
               return 0x6D;
          case 6:
               return 0x7D;
          case 7:
               return 0x07;
          case 8:
               return 0x7F;
          case 9:
               return 0x6F;
          default:
               return 0x00;
     }
}

void KijelzoUpdate(int mp)    //5ms
{
     perc = mp / 60;
     masodperc = mp - (perc * 60);

     PORTB = Decode(perc/10); //   perc
     PORTC = 0x01;
     Delay_ms(1);
```

```
     PORTB = Decode(perc%10); //   perc
     PORTC = 0x02;
     Delay_ms(1);

     PORTB = Decode(masodperc/10);
     PORTC = 0x04;
     Delay_ms(1);

     PORTB = Decode(masodperc%10);
     PORTC = 0x08;
     Delay_ms(2);
}

void main()
{
     TRISA = 0x00;
     TRISB = 0x00;
     TRISC = 0x00;
     TRISD = 0x00;
     TRISE = 0x00;
     ANSEL = 0x00;
     ANSELH = 0x00;
     C1ON_bit = 0;
     C2ON_bit = 0;
     i = 0;
     clk = 0;
     while(1)
     {
          if (clk > 200)
          {
               ++i;
               clk = 0;
          }
          KijelzoUpdate(i);
          ++clk;
     }
}
```

A KijelzoUpdate függvény egy másodperc érték alapján osztogatással meghatározza a 4 db 7 szegmenses kijelző kimenetét. Az egyes kijelzők között 1ms késleltetés van, kivéve az utolsót, ahol 2ms van. Erre azért volt szükség, hogy az összegzett késleltetés 5ms legyen.

A fő programban végtelen ciklusban fut egy clk nevezetű változó inkrementálása és a kijelzők frissítése. Abban az esetben, ha a clk változó értéke nagyobb lesz, mint 200 (1000 / 5 = 200 miatt), akkor az i változó értéke növelve lesz 1-gyel. Az i változó tárolja a ténylegesen eltelt másodperceket, majd a clk változó értéke 0-ra lesz állítva.

6. Kommunikáció a PC és a mikrovezérlő között soros porton

A MikroC kínál szoftveres és hardveres soros port megoldást is. Természetesen a szoftveres soros megoldás némi kód-és adatmemóriát emészt fel.

A PIC16F887-es mikrovezérlő dedikált soros láb kimenetei az RC6 és RC7 lábak. Az RC6 láb a kimenő adat (Tx), míg az RC6 a bejövő adat(Rx). Értelemszerűen soros kommunikációt használva ezen lábak nem használhatóak más célra. TRISC regiszter konfigurációjánál vigyázni kell abban az esetben, ha a PORTC maradék 6 lábát más célra is használni szeretnénk. Ekkor a legfelső helyi értéken lévő bitnek 1, míg a mellette lévő bitnek 0-nak kell lennie. Más konfiguráció esetén a kommunikáció vagy nem megfelelően, vagy egyáltalán nem fog működni.

Hardveres soros kommunikációs függvények

MikroC fejlesztő környezetben a hardveres soros port kezelő függvények UARTx_ jelzéssel kezdődnek. Az x a soros port számát jelenti. Mivel PIC16F887 esetén egyetlenegy ilyen van, ezért az x 1-es értéket vesz fel minden esetben.

Hardveres soros portot tartalmazó mikrovezérlők esetén csak a BAUD rátát választhatjuk meg. A többi paraméter adott, nem módosítható. A paraméterek a következőek:

- *1 stop bit*
- *Nincs paritás*
- *8 adat bit.*

Inicializáció

```
void UARTx_Init(const unsigned long baud_rate);
```

A könyvtár legalapvetőbb függvénye. A baud_rate paraméter alapján beállítja a mikrovezérlő belső regisztereit a kommunikáció létrejöttéhez. Ezen parancs kiadása után érdemes legalább egy 100ms várakozást beiktatni, hogy a kommunikáció biztosan beálljon. A Baud ráta érték az RS232 esetén ismertetett táblázat értékeit veheti fel.

Adatfogadás

Adatfogadás esetén 3db függvényt biztosít a programkönyvtár. Kettőt adatolvasásra, valamint egy függvényt arra, hogy olvasni tudunk-e egyáltalán.

```
char UARTx_Read();
```

A legalapvetőbb az olvasási függvény. Egyetlenegy bájt adatot olvas be a soros portról. Ezen alapfüggvény mellett kínál még egy függvényt a MikroC adat beolvasásra, amely segítségével szövegeket olvashatunk be. Ez a függvény a következő:

```
void UARTx_Read_Text(char *Output, char *Delimiter, char
Attempts);
```

A függvény az output változóba olvassa a kapott karaktereket, míg nem talál a szövegben egy olyan mintát, amit a Delimiter nevű változó ír le. A függvény kudarc esetén az adatolvasást az Attempts változó által meghatározott alkalommal próbálja meg.

A beolvasott adatok nem közvetlenül a meghatározott változókba kerülnek, hanem egy átmeneti regiszterbe. Az átmeneti regiszter tartalmának olvasásával meggyőződhetünk arról, hogy van-e tényleges kommunikációs adat, amit olvashatunk. Erre az alábbi függvényt kínálja a MikroC

```
char UARTx_Data_Ready();
```

Ez a függvény 1-et ad vissza, ha van adat készen olvasásra. 0-t abban az esetben ad, ha nincs olvasható adat. Ezen függvény használata célszerű minden adatolvasás megkezdése előtt.

Adatküldés

Adatküldés esetén szintén 3db függvényt biztosít az osztálykönyvtár.

```
void UARTx_Write(char data_);
```

Legalapvetőbb írási függvény. Soros porton keresztül küld egy bájt adatot.

```
void UARTx_Write_Text(char * UART_text);
```

Ezen függvény a paramétereként kapott szöveget továbbítja soros porton keresztül.

```
char UARTx_Tx_Idle();
```

Ezen függvénnyel ellenőrizhetjük, hogy tudunk-e adatot küldeni. A függvény 1-es értéket fog visszaadni, amennyiben a küldés lehetséges, 0 értéket, ha nem.

Mintaprogram

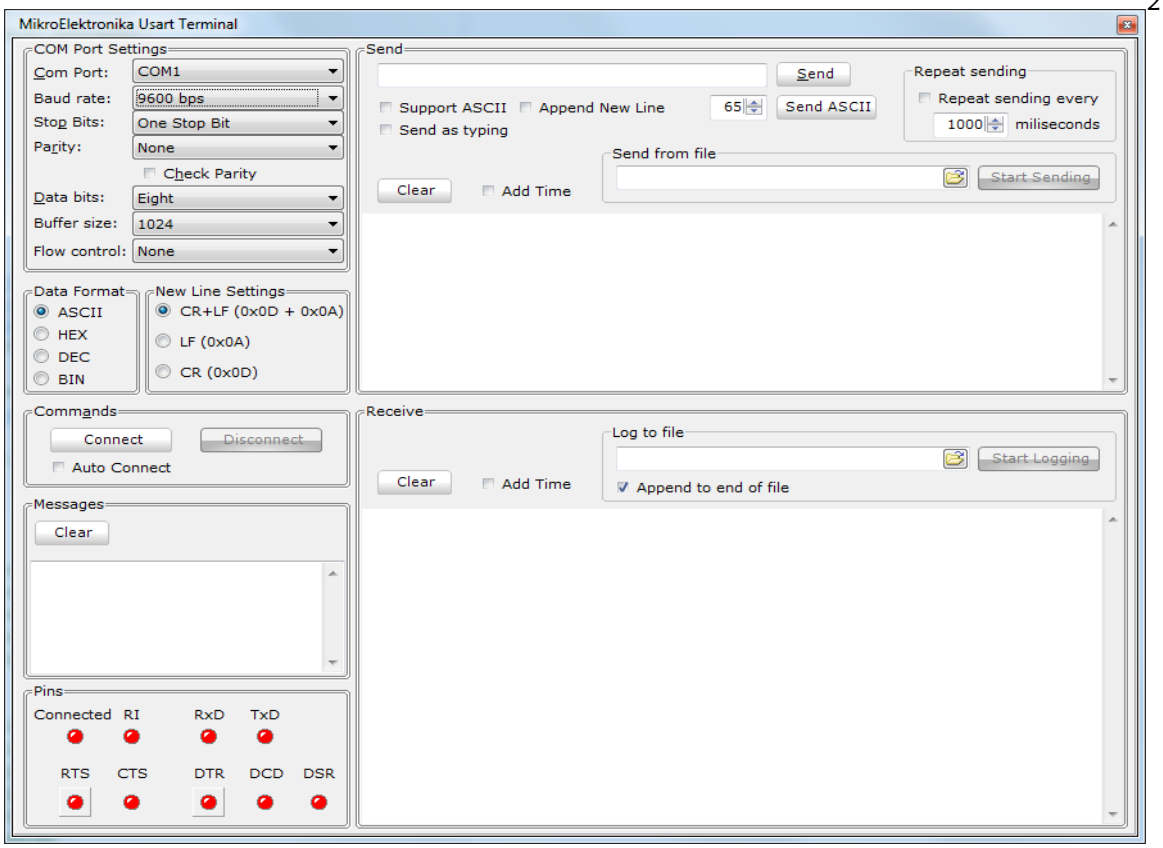
```
unsigned short i;

void main()
{
     UART1_Init(19200);
     Delay_ms(100);

     while (1)
     {
          if (UART1_Data_Ready() > 0)
          {
               i = UART1_Read();
               if (UART1_Tx_Idle()) UART1_Write(i);
          }
     }
}
```

A fenti kódrészletben szereplő mintaprogram annyit tesz inicializáció után, hogy a soros porton keresztül kapott adatot visszaküldi a küldő PC számára. Az inicializáció utáni 100ms várakozás a kommunikáció biztos létrejöttét biztosítja.

Soros kommunikációs alkalmazások tesztelése

Egy soros kommunikációt tartalmazó alkalmazás elkészítése közben sok hiba adódhat, hiszen két helyen is lehet hibázni, a vevő, valamint az adó oldalon is. Azért, hogy a hibakeresés ne legyen olyan nehéz, a MikroC készítői a Tools menü alatt elhelyeztek egy USART Terminal névre hallgató programot. Ezen programmal könnyen letesztelhetjük, hogy a kommunikáció az eszközünk és a PC között zökkenőmentesen zajlik e.

219.



ábra. A MikroC USART Terminál programja

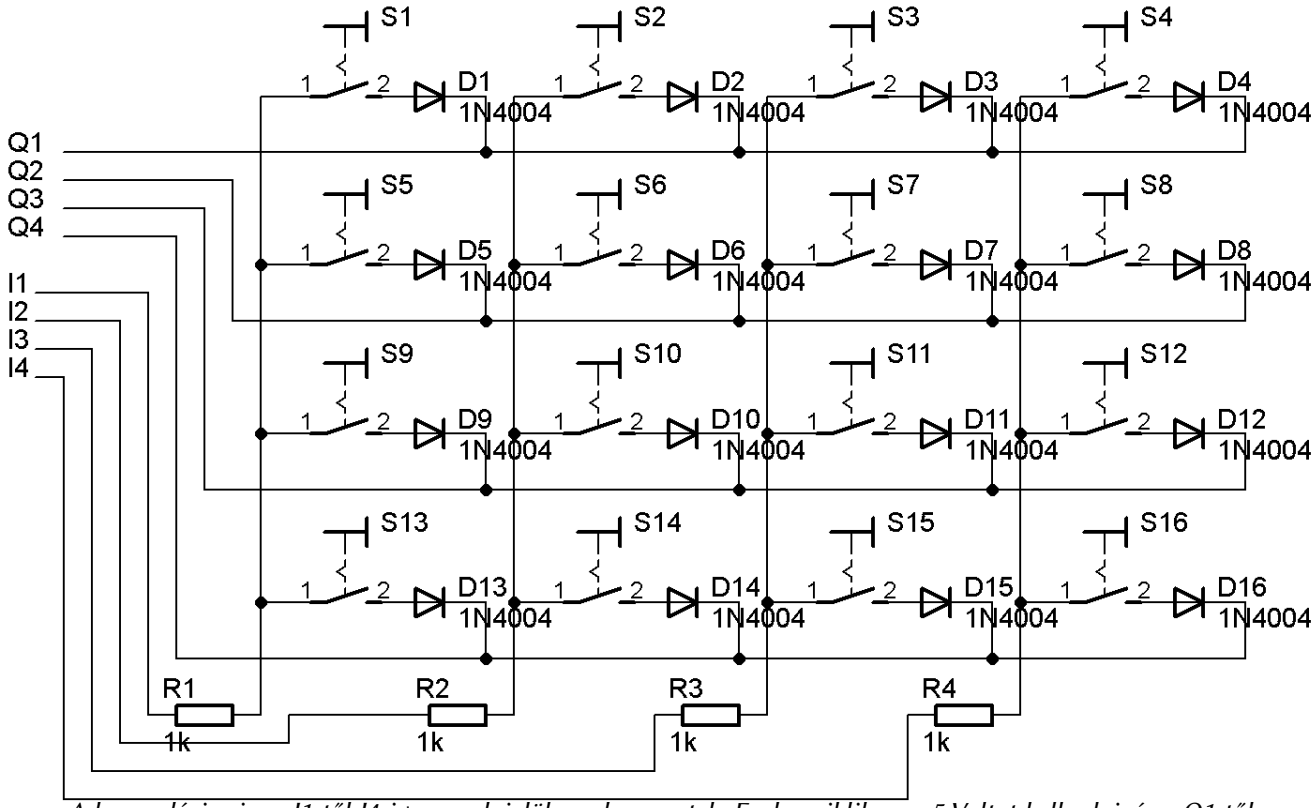
A program az összes soros kommunikációs beállítást támogatja, valamint az adatokat képes bináris, hexadecimális, decimális és ASCII formában kezelni.

Szoftveres soros kommunikációs függvények

A MikroC által biztosított soros kommunikációs megoldások másik családja. Használatuk olyan mikrovezérlők esetén lehet előnyös, amelyek nem rendelkeznek dedikált soros porttal, vagy akkor is előnyös lehet, ha az alkalmazásunknak több soros portot kell kezelnie. A hardveres és szoftveres megoldás közötti főbb különbség a kommunikáció szempontjából az, hogy a szoftveres megoldás használatakor nem csak a Baud ráta választható meg szabadon.

7. Mátrixbillentyűzet kezelése

A mátrixbillentyűzet lényegében egy 2 dimenziós tömb kapcsolókból. Ilyen mátrix elven működik a PC billentyűzete is. A billentyűzet minden esetben bemeneti és kimeneti pontokból áll. 4 bemeneti ponttal és 4 kimeneti ponttal összesen 16db billentyű kezelhető.



A kapcsolási rajzon I1-től I4-ig vannak jelölve a bemenetek. Ezekre ciklikusan 5 Voltot kell adni, és a Q1-től Q4-ig kimeneteken megjelennek azok a gombok, amelyek megnyomva vannak. A nyomógomb utáni diódák azért kellenek, hogy egyértelműen meg lehessen különböztetni a bemeneteket. A diódák nélkül a mátrix nem lenne képes elkülöníteni több gomb egyszeri megnyomását.

Mivel 8 láb kell a vezérléshez, kézenfekvő megoldás egy portról vezérelni az egész mátrix működését. A MikroC kínál beépítetten függvényeket, azonban ezen könyvtár „csak" maximum 4×4-es billentyűzetek kezelésére alkalmas. Előfordulhat egyedi esetben, hogy ennél nagyobb mátrixok kezelésére van szükség. Ilyenkor az alábbi függvényt felhasználva könnyen áthidalható a gond.

```
static char kimenetek[] = { 0x01, 0x02, 0x04, 0x08 };
int i;

char ReadSingleKey()
{
     for (i=0; i<4; i++)
     {
          PORTC = kimenet[i];
          if (PORTC.B4 == 1) return (i * 1) - 1;
          else if (PORTC.B5 == 1) return (i * 2) - 1;
          else if (PORTC.B6 == 1) return (i * 3) - 1;
          else if (PORTC.B7 == 1) return (i * 4) - 1;
     }
}
```

A példában szereplő ReadSingleKey() függvény egy 0 és 15 közötti számot ad vissza, attól függően, hogy melyik gombot nyomtuk meg. A függvény működéséhez feltétlen szükséges, hogy a C port be-és kimenetnek legyen konfigurálva egyszerre.

Ehhez a fő függvényben, a TRISC regiszter értékét a következőre kell módosítani:

```
TRISC = 0xF0;
```

16 gombnál több kezeléséhez értelemszerűen nem elég egyetlenegy port. Ekkor többet kell használni. Továbbá a fenti kódrészlet csak arra alkalmas, hogy egy időben egy gomb lenyomását érzékelje. Egy bizonyos gombszám felett kifizetődő PS/2 csatlakozású billentyűzetet használni, a tömeggyártásból fakadó olcsósága miatt.

Kezelés a MikroC beépített függvényeivel

A MikroC mátrixbillentyűzet kezelőkönyvtára igencsak egyszerű, csupán 3 függvényből áll. Ebből egy az inicializálásra, míg 2 olvasásra.

Inicializáció

```
void Keypad_Init(void);
```

Az inicializálási folyamat 2 részből áll. A függvény hívása előtt be kell definiálni, hogy a billentyűzet mátrixunk melyik portra csatlakozik. Ehhez az alábbi utasítást kell használni:

```
char keypadPort at PORTD;
```

Billentyűzet olvasása

```
char Keypad_Key_Press(void);
```

Ezen függvény egy 1 és 16 közötti számot ad vissza, attól függően, hogy melyik gombot nyomtuk meg. 0-t ad vissza abban az esetben, ha egy gomb sem lett megnyomva.

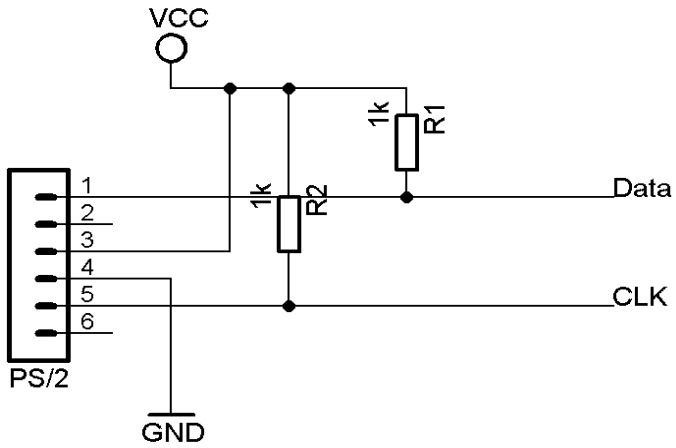
```
char Keypad_Key_Click(void);
```

Ezen függvény annyiban tér el az előző működésétől, hogy ez blokkolja a program végrehajtását egészen addig, amíg egy gomb is nyomva van. Szintén 1 és 16 közötti értéket ad vissza. Abban az esetben, ha nem lett gomb megnyomva, 0-t ad vissza.

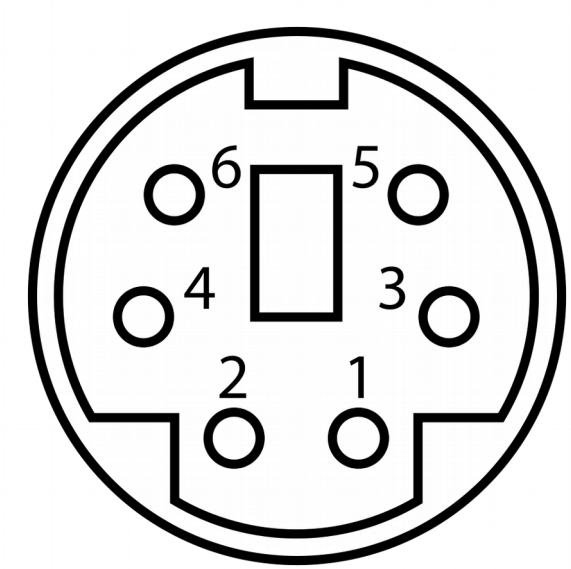
8. PS/2 billentyűzet kezelése

A PS/2 igen régóta a számítógépek csatlakozási felülete billentyűzetek és egerek számára. Eredetileg a soros egerek által használt nagy csatlakozó kiváltására lett tervezve. Később a PC AT billentyűzet csatlakozóját váltotta le.

Egy igen egyszerű soros kommunikációra épülő protokollt használnak ezen eszközök. Használatukhoz nincs szükségünk külön illesztő áramkörökre. Kezeléshez csupán két láb kell.



A PS/2 csatlakozó felület lényegében egy mini DIN6-os csatlakozó, amely fizikai láb kiosztása az alábbi ábrán látható. A 2-es és a 6-os láb nincs semmire sem használva.



PS/2 billentyűzetek kezelésére a MikroC az alábbi függvényeket biztosítja:

```
void Ps2_Config();
```

Ezen függvény segítségével inicializálható a kommunikáció a PS/2 billentyűzet és a mikrovezérlő között. A függvény meghívása előtt az alábbi konstansokat definiálni kell:

Konstans	*Példa*	*Leírás*
`extern sfr sbit PS2_Data;`	`sbit PS2_Data at RC0_bit;`	*Data láb helye*
`extern sfr sbit PS2_Clock;`	`sbit PS2_Clock at RC1_bit;`	*Clock láb helye*
`extern sfr sbit PS2_Data_Direction;`	`sbit PS2_Data_Direction at TRISC0_bit;`	*Data láb iránya*

extern sfr sbit PS2_Clock_Direction;	**sbit** PS2_Clock_Direction **at** TRISC1_bit;	*Clock láb iránya*

unsigned short Ps2_Key_Read(**unsigned short** *value, **unsigned short** *special, **unsigned short** *pressed);

A függvény karakterolvasást tesz lehetővé. Visszatérési értéknek 1-et ad, ha az olvasás sikeres volt. 0 értéket akkor fog adni, amennyiben nem történt gomblenyomás.

A Value paraméterbe kerül a lenyomott gomb ASCII karakter kódja. A függvény felismeri a Shift és a Caps Lock módosító hatását.

A Special paraméter jelzi, hogy a leütés speciális gomb (ESC, TAB, ENTER, stb...) volt-e. Értéke 1 lesz, amennyiben igen, 0 abban az esetben, ha nem.

A Pressed paraméter arról ad információt, hogy a gomb még mindig lenyomva van-e, vagy már felengedték. Felengedett állapot esetén értéke 0, míg lenyomott állapotban 1.

A következő mintaprogram a billentyűzetről olvasott értékeket soros porton át továbbítja:

```
unsigned short keydata = 0, special = 0, down = 0;

sbit PS2_Data at RC0_bit;
sbit PS2_Clock at RC1_bit;
sbit PS2_Data_Direction  at TRISC0_bit;
sbit PS2_Clock_Direction at TRISC1_bit;

void main()
{
     ANSEL  = 0;
     ANSELH = 0;
     C1ON_bit = 0;
     C2ON_bit = 0;

     UART1_Init(19200);
     Ps2_Config();
     Delay_ms(100);

     UART1_Write_Text("Ready");
     UART1_Write(10);
     UART1_Write(13);
     do
     {
          if (Ps2_Key_Read(&keydata, &special, &down))
          {
               if (down && (keydata == 16)) //backspace
                    UART1_Write(0x08);
               else if (down && (keydata == 13)) //enter
                    UART1_Write('r');
               else if (down && !special && keydata)
                    UART1_Write(keydata);
          }
          Delay_ms(1);
     }
     while (1);
```

```
}
```

9. Karakteres LCD kezelése

Karakteres LCD esetén ipari szabvánnyá váltak a Hitachi által gyártott HD44 780-as típusú kijelzővezérlőt használó kijelzők. Ezen vezérlő áramkör a sikerét az egyszerűség és a szolgáltatások helyes egyensúlyban tartásának köszönheti.

Ilyen típusú kijelzők vezérlése esetén 2 járható utunk van. Vagy mi készítünk hozzá egy vezérlőprogramot, vagy támaszkodunk a MikroC beépített vezérlő függvényeire. Saját vezérlőprogram készítésére akkor lehet szükségünk, ha olyan LCD-t szeretnénk kezelni, amelynek a felépítése nem szabványos. A helyzet az, hogy a HD44 780-as vezérlő nem köti ki, hogy az LCD hány sor és hány karakteres legyen. Egészen addig, amíg az integrált RAM-ba fér az adat, nem lesz gondunk.

Vannak bevett típusok, amelyek mondhatni „szabványosak". Ezekre lett felkészítve a MikroC környezet. A „szabványos" felépítéstől eltérő kijelzőkkel azonban nem tud mit kezdeni a beépített vezérlő függvény könyvtár.

Saját vezérlő szoftver készítéséhez és a beépített vezérlő szoftver hatékony használatához meg kell ismerkednünk a HD44 780 utasításkészletével és tulajdonságaival.

A HD44 780-AS kijelzők tulajdonságai

- *5×8 vagy 5×10 pontból álló karakterek.*
- *Maximum 80 karakter kezelése a kijelzőn. Ezen belül a sor és oszlop felbontás nincs rögzítve.*
- *4 vagy 8 bites vezérlő busz.*
- *Beépített karakterek mellett egyéni karakterek létrehozásának lehetősége is.*

Utasításkészlet

A HD44 780 utasításkészlete igencsak egyszerű, 11db utasítása van. Az utasítások hossza 8 bites, mint a vezérlő busz. A vezérlő busz 4 bites üzemmódban is képes működni, ám ekkor is 8 bites adatokkal dolgozik. Tehát a parancsküldés némi késleltetéssel 2×4bit formában történik meg.

Utasítás	Kód										L
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	
Clear display	0	0	0	0	0	0	0	0	0	1	Kijelző törlése, majd kurzor
Cursor home	0	0	0	0	0	0	0	0	1	*	Kurzor kezdőpozícióba állítá alapértelmezett pozícióba tesz
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Kurzor mozgási irány beá eltolásána
Display On/Off control	0	0	0	0	0	0	1	D	C	B	Kijelző Ki/be kapcsolása (D), villogtatás Ki
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Kurzor mozgatása vagy k kurzor/eltolási
Function set	0	0	0	0	1	DL	N	F	*	*	Adatbusz hosszának (DL), kijel betűtípusán
Set CGRAM address	0	0	0	1	CGRAM Cím						CGRAM cím beállítása. A CG ki
Set DDRAM address	0	0	1	DDRAM Cím							DDRAM cím beállítsa. A DDR ki
Read busy-flag and address counter	0	1	BF	CGRAM/DDRAM Cím							Busy Flag (BF) olvasása, a valamint kiolvassa a CG
Write to CGRAM or DDRAM	1	0	Beírandó adat								Adat írása a CGRA
Read from CGRAM or DDRAM	1	1	Kiolvasott adat								Adat olvasása a CGRA

98. táblázat. A HD44780-as kijelző vezérlő utasítás készlete

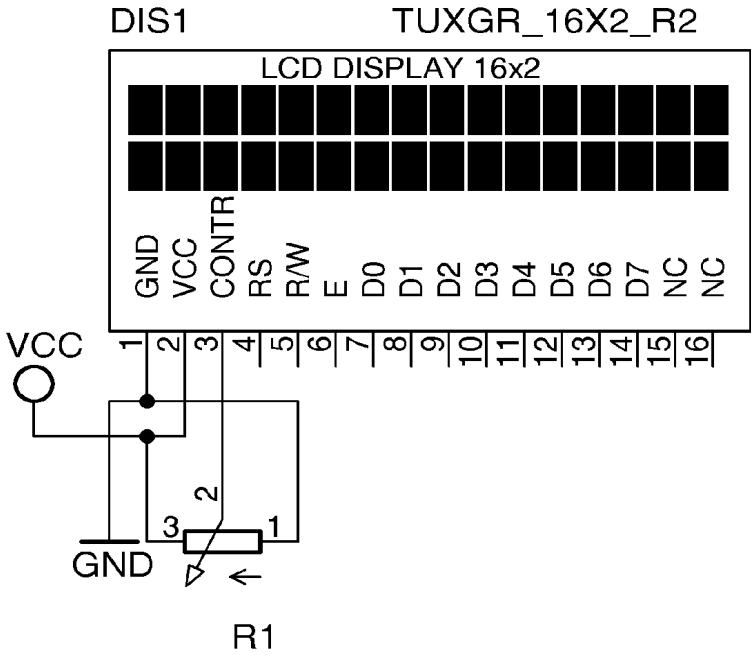
*A táblázatban * jelöléssel lettek megjelölve azon bitek, amelyek állapota mindegy a vezérlés szempontjából. Ezen bitek lehetnek 1-es és 0-s értékűek is. A vezérlő programban célszerű ugyanazon jelöléseket használni. A CGRAM a karaktergenerátor RAM címet jelöli, amelybe egyedi karakterek vehetőek fel. A DDRAM a kijelzőn megjelenő karakterek RAM-ja, egyfajta háttérbuffer. A táblázatban szereplő további bitjelölések az alábbi táblázatban találhatóak meg.*

Bit Név	***Funkció***	
I/D	*0 = Kurzor pozíció csökkentése*	*1 = Kurzor pozíció növelése*
S	*0 = Nincs kijelző eltolás*	*1 = Kijelző eltolása*
D	*0 = Kijelző kikapcsolása*	*1 = Kijelző bekapcsolása*
C	*0 = Kurzor kikapcsolása*	*1 = Kurzor bekapcsolása*
B	*0 = Kurzor villogás kikapcsolása*	*1 = Kurzor villogás bekapcsolása*
S/C	*0 = Kurzor mozgatása*	*1 = Kijelző eltolása*
R/L	*0 = Eltolás balra*	*1 = Eltolás jobbra*
DL	*0 = 4 bites adatbusz szélesség*	*1 = 8 bites adatbusz szélesség*
N	*0 = 1/8-os vagy 1/11-es ciklus (1 vonal)*	*1 = 1/16-os ciklus. (2 vonal)*
F	*0 = 5×7 pontos karakterek*	*1 = 5×10 pontos karakterek*
BF	*0 = Utasítás fogadására képes*	*1 = Utasítás végrehajtás folyamatban*

99. táblázat. A HD44 780-as kijelző vezérlő utasításkészletében használt bit jelölések

Bekötés

Az alábbi ábrán egy tipikus LCD modul lábkiosztása látható egy kontraszt beállító áramkörrel.



A kapcsolási rajzon szereplő LCD kijelző lábainak jelentése a következő:

Név	***Funkció***
GND	*Földpont*
VCC	*Tápfeszültség, 5V*
CONTR	*LCD kontrasztbeállító. Ide egy Trimer potenciométert érdemes kötni a kapcsolási rajzon szereplő módon. A potenciométer értéke tipikusan 10K Ohm szokott lenni.*
RS	*Ha értéke 0, akkor a buszon érkező jel utasítás, amennyiben értéke 1, akkor a buszon érkező jel adat.*
R/W	*Olvasás/Írás váltó. 0 érték esetén az LCD írva van, 1 érték esetén pedig olvasva.*
E	*Engedélyező jel.*
D0...D7	*Adatbusz lábai. 4 bites üzemmódban D4...D7 közötti lábakon (felső byte) várja az utasítást az LCD. 4 bites üzemmódban az alsó 4 bitet (D0...D3) földpontra kell kötni.*

100. táblázat: HD44780 kijelző vezérlő lábkiosztása

A fenti lábkiosztás mellett előfordulhat, hogy a kijelző további lábakkal is rendelkezik. Ez akkor történhet meg, ha a kijelző rendelkezik háttérvilágítással, vagy ha több, mint 80 karakter megjelenítésére képes.

A háttérvilágítás beépített LED dióda segítségével van megoldva, ezért, mielőtt a tápfeszültségre csatlakozna, kell elébe kötni egy korlátozó ellenállást. Abban az esetben, ha a kijelző több, mint 80 karaktert képes megjeleníteni, akkor 2db engedélyező jellel rendelkezik, mivel ezt úgy lehet megoldani, hogy 2 kijelzővezérlőt kötnek párhuzamosan. A 2 engedélyező jel jelölése általában E1 és E2.

Az ilyen kijelzők esetén a teljes kijelzőtartomány felét az egyik kijelző vezérlő működteti, míg a másik felét a második kijelző. Az ilyen kijelzőket érdemes 8 bites üzemmódban vezérelni, mivel így is 2 ciklusban kell elküldeni az adatot a kijelző felé.

Inicializációs folyamat

Sajnos a HD44780-as kijelzők bekapcsolása nem csak abból áll, hogy tápfeszültséget adunk rá, és már működik is a dolog. Ezen kijelzőknek van egy inicializációs folyamata, amit ha nem végzünk el, vagy nem megfelelő, akkor a kijelző nem fog, vagy hibásan fog adatokat megjeleníteni. Az inicializációs folyamat a következő lépésekből áll:

1. *Function set utasítás 8 bites adatbusz beállítással.*
2. *Engedélyező jel küldése.*
3. *5ms várakozás.*
4. *Engedélyező jel küldése.*
5. *160µs várakozás.*
6. *Engedélyező jel küldése.*
7. *160µs várakozás.*
8. *Function set utasítás küldése, itt kijelző beállítása megfelelően.*
9. *40µs várakozás.*
10. *Display on/off paranccsal kikapcsolás.*
11. *40µs várakozás.*
12. *Clear Display parancs.*
13. *40µs várakozás.*
14. *Entry mode set parancs.*
15. *40µs várakozás.*
16. *Display on/off paranccsal bekapcsolás.*

Karakterkészlet

A vezérlő az alábbi ábrán található karaktereket képes megjeleníteni, melyek némi kiegészítéssel az ASCII

224. ábra. A Vezérlő beépített karakterei

karakter táblázatnak felelnek meg.

Egyedi karakterek is létrehozhatóak, és megjelenítéshez társíthatóak. Ezen karakterek kényelmes kezelésére a MikroC tartalmaz egy LCD karakter szerkesztőt, amivel könnyen létrehozható a karakter. A karakter betöltéséhez használható kódot is legenerálja ez az eszköz, amely a MikroC Tools menüjében LCD Custom Character címszó alatt érhető el.

Kezelés a MikroC beépített függvényeivel

A MikroC karakteres LCD könyvtára a karakteres LCD modulokat 4 bites üzemmódban tudja kezelni. Alapértelmezetten a könyvtár feltételezi, hogy a kijelző 4 soros, soronként 20 karakterrel. Ettől függetlenül bármilyen kijelző esetén használhatóak, azonban a pozicionálás nem biztos, hogy jól fog működni. A könyvtár használatbavétele előtt pár konstanst be kell állítanunk, amely megmondja a kezelő könyvtárnak, hogy az LCD kijelző melyik lábát hova kötöttük. Ezek a konstansok az alábbi táblázatban lettek összefoglalva:

Konstans	*Példa*	*Funkció*
`extern sfr sbit LCD_RS`	`sbit LCD_RS at RB4_bit;`	*RS, EN, D7, D6, D5, D4 lábak hozzárendelése a mikrovezérlő megfelelő lábaihoz.*
`extern sfr sbit LCD_EN`	`sbit LCD_EN at RB5_bit;`	
`extern sfr sbit LCD_D7`	`sbit LCD_D7 at RB3_bit;`	
`extern sfr sbit LCD_D6`	`sbit LCD_D6 at RB2_bit;`	
`extern sfr sbit LCD_D5`	`sbit LCD_D5 at RB1_bit;`	
`extern sfr sbit LCD_D4`	`sbit LCD_D4 at RB0_bit;`	
`extern sfr sbit LCD_RS_Direction`	`sbit LCD_RS_Direction at TRISB4_bit;`	*RS, EN, D7, D6, D5, D4 hozzárendelt portlábak irányát beállító bitek meghatározása. A TRIS regisztereknek illeszkedniük kell a megadott porthoz. Tehát ha az RS láb PORTB 0-lábán van, akkor az irányt TRISB azonos bitje határozza meg.*
`extern sfr sbit LCD_EN_Direction`	`Sbit LCD_EN_Direction at TRISB5_bit;`	
`extern sfr sbit LCD_D7_Direction`	`sbit LCD_D7_Direction at TRISB3_bit;`	
`extern sfr sbit LCD_D6_Direction`	`sbit LCD_D6_Direction at TRISB2_bit;`	
`extern sfr sbit LCD_D5_Direction`	`sbit LCD_D5_Direction at TRISB1_bit;`	
`extern sfr sbit LCD_D4_Direction`	`sbit LCD_D4_Direction at TRISB0_bit;`	

101. táblázat. Az LCD kezelő függvények működéséhez szükséges konstansok

Ezen konstansok beállítása után a MikroC a következő függvényeket kínálja a kezeléshez:

```
void Lcd_Init();
```

Ez a függvény inicializálja a mikrovezérlő és az LCD vezérlő közötti kapcsolatot, az inicializációs folyamat fejezetben megadott algoritmus szerint. A többi kezelőfüggvény helyes működéséhez e parancs futtatása elengedhetetlen.

```
void Lcd_Out(char row, char column, char *text);
```

A függvény a kijelzőre ír egy szöveget, amit a text paraméter határoz meg. A szöveg kezdetét a kijelzőn a row és column paraméterek határozzák meg.

```
void Lcd_Out_Cp(char *text);
```

Az aktuális kurzor pozíciónál kezdődően írja a kijelzőre a text paraméter által meghatározott szöveget.

```
void Lcd_Chr(char row, char column, char out_char);
```

Karakter írása a kijelzőre megadott pozícióban. A pozíciót a row és column paraméterek határozzák meg, míg a kiírandó karaktert az out_char paraméter.

```
void Lcd_Chr_Cp(char out_char);
```

Kurzor pozíció után ír egy karaktert a kijelzőre.

```
void Lcd_Cmd(char out_char);
```

Parancsküldés az LCD modulnak. A parancsot az out_char paraméter határozza meg. Itt az utasításkészletben szereplő utasítások használhatóak. Az egyszerű kezeléshez azonban a könyvtár tartalmaz egy pár konstans értéket, amelyek megkönnyítik a kezelést.

Parancs konstans	*Parancs hatása*

_LCD_FIRST_ROW	*Kurzor az első sor elejére*
_LCD_SECOND_ROW	*Kurzor a második sor elejére*
_LCD_THIRD_ROW	*Kurzor a harmadik sor elejére*
_LCD_FOURTH_ROW	*Kurzor a negyedik sor elejére*
_LCD_CLEAR	*Kijelző törlése*
_LCD_RETURN_HOME	*Kurzor mozgatása a kezdő pozícióra, valamint a kijelző eltolást is alapértelmezett helyre állítja.*
_LCD_CURSOR_OFF	*Kurzor kikapcsolása*
_LCD_UNDERLINE_ON	*Aláhúzás jel kurzor bekapcsolása*
_LCD_BLINK_CURSOR_ON	*Kurzor villogás bekapcsolása*
_LCD_MOVE_CURSOR_LEFT	*Kurzor mozgatása balra*
_LCD_MOVE_CURSOR_RIGHT	*Kurzor mozgatása jobbra*
_LCD_TURN_ON	*Kijelző bekapcsolása*
_LCD_TURN_OFF	*Kijelző kikapcsolása*
_LCD_SHIFT_LEFT	*Kijelző eltolása balra*
_LCD_SHIFT_RIGHT	*Kijelző eltolása jobbra*

102. táblázat. LCD parancs konstansok és hatásuk

Kezelés saját készítésű illesztőprogrammal

A saját készítésű kezelőprogramnak megvan az az előnye, hogy 8 bites üzemmódban is képes kezelni a kijelzőt, valamint PIC-en kívül más mikrovezérlőre is átültethető a kód.

Az alábbi programrészletben szereplő kód 4 bites üzemmódban inicializálja a kijelzőt. Biztosít egy inicializációs függvényt, egy parancsküldési függvényt és egy karakter kiírási függvényt. A könyvtár a PORTD portra kötve vezérli a kijelzőt. Funkcionalitásban a kódrészlet szegényesebb, mint a MikroC program könyvtára.

A konkrét bekötési információk a kódrészlet elején megtalálhatóak megjegyzés formájában.

```
//RW = GND RS = PORTD.B2 E = PORTD.B3 Data PORTD.B4 -> PORTD.F7

//E-n csinál egy fel és lefutó élt.
void E_HiLo()
{
     PORTD.B3 = 1;
     Delay_us(1);
     PORTD.B3 = 0;
}
//LCD Parancs küldés 4 bites üzemmódban
void MLCD_Cmd(char data)
{
     char felso, also;
     felso = data & 0xF0;
     also = data << 4;
     PORTD = (PORTD & 0x0F) | felso;
     E_HiLo();
     PORTD = (PORTD & 0x0F) | also;
     E_HiLo();
```

```
     if (data > 2) Delay_us(40);
     else Delay_ms(2);
}

//Inincializálja az LCD képernyőt.
void MLCD_Init()
{
     PORTD.B2 = 0;
     PORTD.B3 = 0;
     Delay_ms(50);
     PORTD = 0x30;
     E_HiLo();
     Delay_ms(5);
     E_HiLo();
     Delay_us(160);
     E_HiLo();
     Delay_us(160);
     PORTD = 0x20; //4Bites Interfész beállítása
     E_HiLo();
     Delay_us(160);
     MLCD_Cmd(0x2a); //Function set 4 bit interfész
     MLCD_Cmd(0x08);  //Display on/off beállítás
     MLCD_Cmd(0x01); //clear display
     MLCD_Cmd(0x06); //entry mode set
     MLCD_Cmd(0x0C); //Display on/off
}

void MLCD_Kiir(char data)
{
     PORTD.B2 = 1;
     MLCD_Cmd(data);
     PORTD.B2 = 0;
}
```

10. I2C buszrendszer kezelő függvények

A MikroC I2C kezelőkönyvtára olyan mikrovezérlőket is támogat, amelyek több I2C porttal is rendelkeznek, ezért a kezelőfüggvényekben található egy szám is. Az 1-es szám a példákban azt jelzi, hogy az első portra vonatkozik az utasítás. Értelemszerűen ezen funkciók használatához a vezérlőnek rendelkeznie kell hardveres I2C támogatással.

```
void I2C1_Init(const unsigned long clock);
```

Inicializálja az I2C buszrendszert paraméterben megadott frekvenciával. A használt frekvenciaérték maximumát a mikrovezérlő korlátozza. Erről részletesebben a használt vezérlő adatlapjában lehet tájékozódni.

```
unsigned short I2C1_Start(void);
```

Megállapítja, hogy a busz szabad e, és ha igen, akkor küld egy start jelet. Ha nem történt hiba, akkor 0 értéket ad vissza a függvény.

```
void I2C1_Repeated_Start(void);
```

Küld egy ismételt start jelet a buszrendszeren.

```
unsigned short I2C1_Is_Idle(void);
```

Megállapítja, hogy szabad e a busz. 1-es értéket ad vissza, amennyiben szabad.

```
unsigned short I2C1_Rd(unsigned short ack);
```

Egy byte-ot olvas a buszról. Olvasás után, ha az ack paraméter értéke 1, akkor küld egy nyugta jelet. A függvény csak egy startjel küldése után használható.

```
unsigned short I2C1_Wr(unsigned short data_);
```

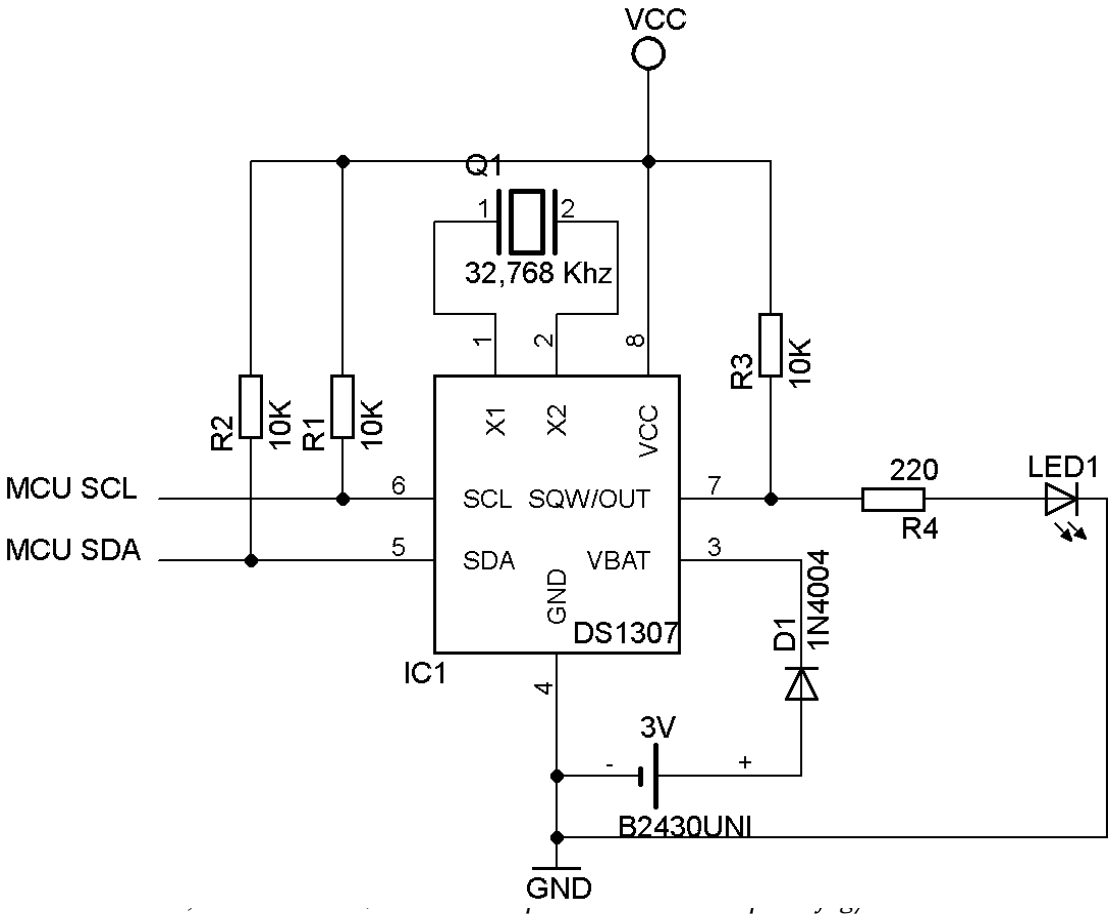
Egy byte-ot ír a buszra. A függvény csak egy startjel küldése után használható.

```
void I2C1_Stop(void);
```

Küld egy stop jelet a buszrendszerre.

11. DS1307 RTC kezelése

A DS1307 egy igencsak népszerű valós idejű óra, amely I2C buszra illeszthető. RTC funkciók mellett 56 byte szabadon felhasználható RAM területet is tartalmaz az áramkör, amely értékét nem veszti el, köszönhetően az RTC funkcionalitás fenntartásáért felelős elemnek. Naptár szolgáltatásokkal is rendelkezik a chip, tehát nyilván tud tartani év, hónap és nap információkat is. Az évek nyilvántartásánál figyelembe veszi a szökőéveket. Az áramkör



bekötése igencsak

Ahogy a k

maximum 100 KH.

szükséges egy elem

az elem feszültség

elemmel körülbelül 10 évig képes pontos dátum/idő tartásra.

Az SCL és SDA I2C adatvezetékeket a busz szabványa miatt fel kell húzni egy-egy 10K Ohm ellenállással +5V szintre. Az SQW/OUT láb egy négyszögjel kimenet, ami impulzusadásra használható és konfigurálható. Ezt a kimenetet szintén fel kell húzni. Az impulzusgyakoriság konfigurálható, arra használható például, hogy legyen visszajelzés az óra funkció működéséről.

Az áramkör kezelése igencsak egyszerű, mivel EPROM memóriaként viselkedik, így lényegében a megfelelő memóriarekeszek olvasásával jutunk hozzá az idő egyes részeihez. Ezen rekeszek írásával pedig beállíthatjuk az időt egy kívánt időpontra. Az alábbi táblázat a DS1307 memóriakiosztását foglalja össze.

Cím	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Funkció	Tartomány
00h	CH	10 másodpercek			másodpercek				Másodpercek tárolása	0-59
01h	0	10 percek			percek				Percek tárolása	0-59
02h	0	12	DU / DE	10 órák	órák				Órák tárolása	1-12 +DE/DU
		24	10 órák							0-23

03h	0	0	0	0	0	7 napja			Hét napjának tárolása	1-7
04h	0	0	10 napok		Nap				Hónap napjának tárolása	1-31
05h	0	0	0	10 hó	Hónap				Hónapok tárolása	1-12
06h	10 évek				Év				Évek tárolása	0-99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Vezérlő regiszter	-
08h - 3Fh									RAM 56×8	00h-FFh

103. táblázat: A DS1307 memória felosztása

Az összes regiszterben található érték BCD formátumban van tárolva. A hét napjainak tárolására használt regiszter értéke mindig éjfélkor növekszik eggyel. A chip vasárnaphoz rendeli a hét első napját. A CH jelzésű bit az óra megállítására szolgál. Amennyiben ezen bit értéke egyes, akkor az óra áll, 0 értéknél pedig fut.

A chip első indításánál az alapértelmezett dátum és idő, év, hónap, nap, óra, perc, másodperc formátumban: 00-01-01 00:00:00. Továbbá ekkor a CH bit 1-es értéket fog alapértelmezetten felvenni.

A DS1307 futtatható 12 órás vagy 24 órás üzemmódban is. Az órák tárolására használt regiszter 6. bitje a választóbit. Ha ez a bit egyes értéket vesz fel, akkor 12 órás üzemmódban van a chip. Ekkor az 5. bit tárolja a délelőtt/délután jelzést, az 1-es érték itt a délutánt jelzi 24 órás üzemmódban.

24 órás üzemmódban az 5. bit második 10 órás regiszter funkciókat lát el. (20-23 óra) 12 órás/24 órás működés váltáskor az óra értékét újra kell állítani.

Esetlegesen attól, hogy rossz dátum/idő érték kerül kiolvasásra vagy beírásra, nem kell aggódni, mivel a DS1307 az írási és olvasási kérelmeket egy belső bufferből szolgálja ki, amit szinkronizál a memóriával, amikor belső működése engedélyezi. Azonban az összes dátummódosító utasításnak egy másodpercen belül kell lezajlania.

A vezérlő regiszter az SQW/OUT láb működését befolyásolja. A 7. OUT bit az SQW/OUT láb logikai szintjét határozza meg, ha a négyszögjel kiadás le van tiltva. Ha ez a bit egyes értékű, akkor az SQW/OUT láb logikai magas értéket vesz fel, 0 bitérték esetén logikai alacsony szintet. Alapértelmezetten ezen bit értéke 0.

Az SQWE bit a négyszögjel kiadást kapcsolja be, vagy ki. Ez alapértelmezetten 0 értékű. 1-es érték esetén a négyszögjel kiadás engedélyezve van. A négyszögjel frekvenciáját az RS1 és RS2 bit határozza meg. Ezek lehetséges kombinációját és a kiadott négyszögjel frekvenciáját az alábbi táblázat tartalmazza:

RS1	RS0	Frekvencia
0	0	1 Hz
0	1	4,096 kHz
1	0	8,192 kHz
1	0	32,768 kHz

104. táblázat: A kimeneti frekvencia lehetséges értékei

Kezelőszoftver

Mivel a DS1307-hez a MikroC nem rendelkezik saját függvénykönyvtárral, ezért írnom kellett egyet. Szerencsére a Google-ben találtam egy kódrészletet, amelyet Deniz Elmasili írt DS1307 kezeléséhez MikroC-ben. A kód a következő:

```
void start1307s();
void ds1307_init();
void ds1307_set_date_time(char day,char mth,char year,char dow, →
char hr,char min,int sec);
void ds1307_get_time_date(char *day,char *mth,char *year,char →
*dow,char *hr,char *min,char *sec);
void ds1307_set_controlreg(char out, char sqwe, char rs);

void start1307s()
{
     I2C_Start();   //issue start signal
     I2C_Wr(0xD0);  //address
     I2C_Wr(0);     //start from word at address 0
     I2C_Wr(0);     //write 0 to config word (enable counting)
     I2C_Stop();    //issue stop signal
}

void ds1307_init()
{
     int seconds=0;
     I2C_Start();
     I2C_Wr(0xD0);  //WR -> RTC
     I2C_Wr(0x00);  //REG 0
     I2C_Start();
     I2C_Wr(0xD1);  //RD from RTC
     seconds = Bcd2Dec(I2C_Rd(0));
     I2C_Stop();
     seconds = (seconds & 0x7F); //óra stop
     Delay_ms(50);

     I2C_Start();
     I2C_Wr(0xD0);  //WR to RTC
     I2C_Wr(0x00);  //REG 0
     I2C_Wr(Dec2Bcd(seconds));
     I2C_Start();
```

```
     I2C_Wr(0xD0);  //WR to RTC
     I2C_Wr(0x07);  //Control Register
     I2C_Wr(0x80);  //Disable squarewave output pin
     I2C_Stop();
}

void ds1307_set_date_time(char day,char mth,char year,char dow, →
char hr,char min,int sec)
{
     sec =(sec & 0x7F); //halt kikapcs
     hr =(hr & 0x3F);
     I2C_Start();
     I2C_Wr(0xD0);  //I2C write address
     I2C_Wr(0x00);  //Start at REG 0 - Seconds
     I2C_Wr(Dec2Bcd(sec));    //REG 0
     I2C_Wr(Dec2Bcd(min));    //REG 1
     I2C_Wr(Dec2Bcd(hr));     //REG 2
     I2C_Wr(Dec2Bcd(dow));    //REG 3
     I2C_Wr(Dec2Bcd(day));    //REG 4
     I2C_Wr(Dec2Bcd(mth));    //REG 5
     I2C_Wr(Dec2Bcd(year));   //REG 6
     I2C_Stop();
}

void ds1307_get_time_date(char *day,char *mth,char *year,char →
*dow, char *hr,char *min,char *sec)
{
     I2C_Start();
     I2C_Wr(0xD0);
     I2C_Wr(0x00);
     I2C_Start();
     I2C_Wr(0xD1);
     *sec=Bcd2Dec(I2C_Rd(1) & 0x7F);
     *min=Bcd2Dec(I2C_Rd(1) & 0x7F);
     *hr=Bcd2Dec(I2C_Rd(1) & 0x3F);
     *dow=Bcd2Dec(I2C_Rd(1) & 0x7F); // REG 3
     *day=Bcd2Dec(I2C_Rd(1) & 0x3F); //REG 4
     *mth=Bcd2Dec(I2C_Rd(1) & 0x1F); //REG 5
     *year=Bcd2Dec(I2C_Rd(0)); //REG 6
     I2C_Stop();
}

void ds1307_set_controlreg(char out, char sqwe, char rs)
{
     char rsw = rs;
     if (rs > 3 || rs < 0) rsw = 0;
     char sqwew = (sqwe & 0x01) * 0x10;
     char outw = (out & 0x01) * 0x80;
     char writereg = outw + sqwew + rsw;
     I2C_Start();
     I2C_Wr(0xD0);  //I2C write address
     I2C_Wr(0x07);  //Start at REG 7
     I2C_Wr(writereg);
```

```
    I2C_Stop();
}
```

A könyvtár az alábbi funkciókat biztosítja:

```
void ds1307_init()
```

Elindítja a kommunikációt a DS1307 chippel, valamint alapértelmezetten kikapcsolja az SQW/Out lábon a négyszögjel kimenetet.

```
void start1307s()
```

Gondoskodik róla, hogy a DS1307 ténylegesen futó üzemmódban legyen.

```
void ds1307_set_date_time(char day,char mth,char year,char dow, →
char hr,char min,int sec)
```

Dátum és idő beállítására szolgáló függvény.

```
void ds1307_get_time_date(char *day,char *mth,char *year,char →
*dow,char *hr,char *min,char *sec)
```

Kiolvassa a tárolt dátumot és időt a DS1307-ből.

12. Belső EEPROM kezelése

A legtöbb PIC mikrovezérlő rendelkezik néhány byte beépített EEPROM memóriával, amit adatok tárolására használhatunk fel. Ezen memória kezelése MikroC esetén igencsak egyszerű, mivel mindösszesen 2 függvényt biztosít a MikroC a memória byte-onkénti olvasására.

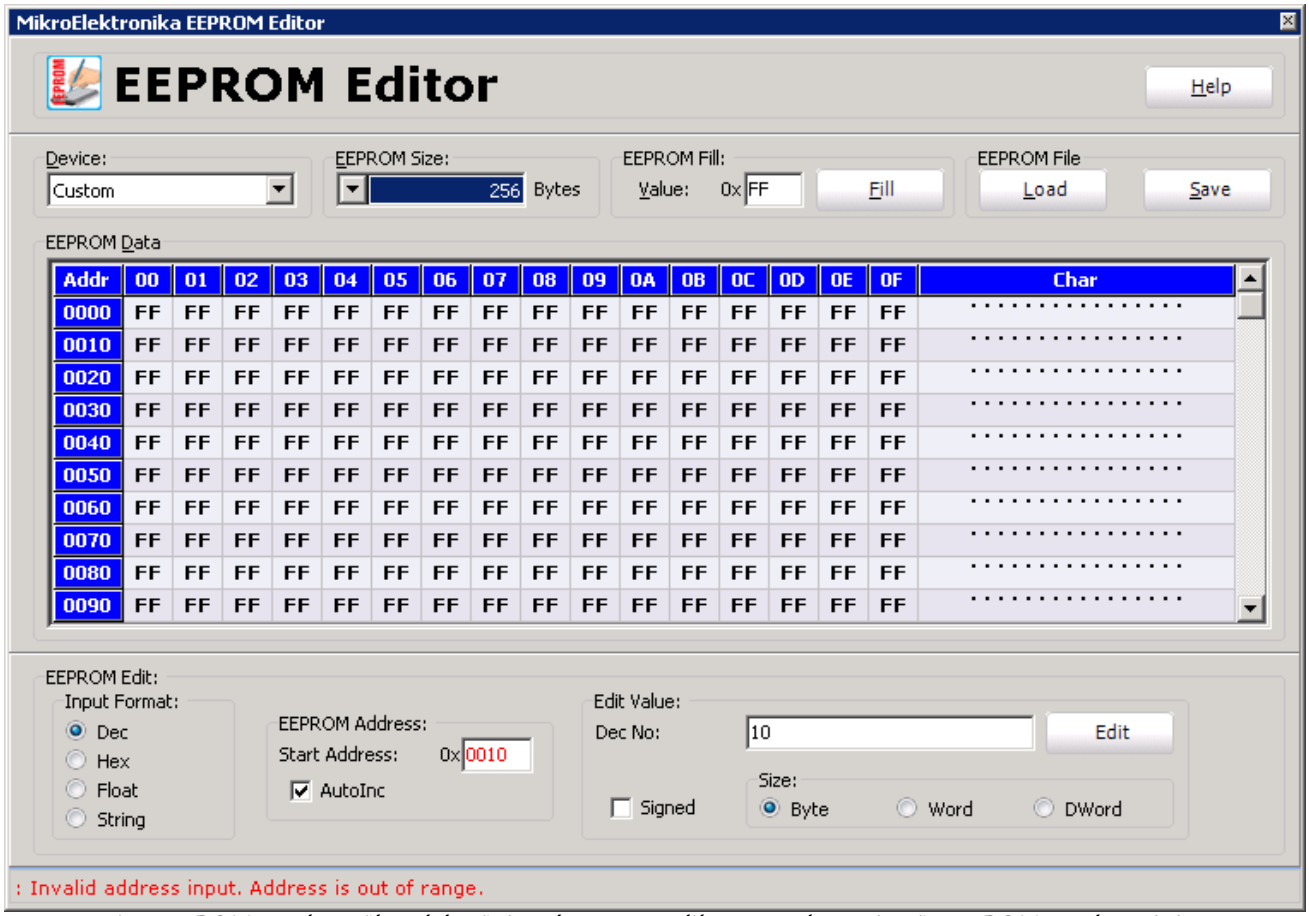
```
unsigned short Eeprom_Read(unsigned short adress);
```

A megadott memóriacímről kiolvas egy byte-ot

```
void Eeprom_Write(unsigned int adress, unsigned short data);
```

A megadott memóriacímre ír egy byte-ot.

A fejlesztőkörnyezet tartalmaz egy EEPROM szerkesztő felületet is. Ezt a Tools menü EEPROM Editor menüpontja alatt találjuk meg.



Az EEPROM szerkesztőben lehetőségünk van egyedileg megadott méretű EEPROM szerkesztésére, vagy a projekt alapján a mikrovezérlőben elhelyezett méretű EEPROM fájl szerkesztésére. Az adat megadható szövegként, vagy hexadecimális formátumban is. Ha a szerkesztő által létrehozott fájlt, a projekthez csatoljuk, akkor a .hex fájl betöltésekor a PICKIT3 programozó alkalmazása be fogja tölteni az EEPROM fájlt is, és az eszközre is fogja írni, ha az Auto import HEX + Write to Device gombbal programozzuk fel az eszközt. A Belső EEPROM használatára itt egy rövid mintaprogram:

```
char i;

void main()
{
     ANSEL = 0;
     ANSELH = 0;
     C1ON_bit = 0;
     C2ON_bit = 0;
     TRISB = 0;
     TRISC = 0;
     TRISD = 0;

     for(i = 0; i < 32; i++)
     {
          EEPROM_Write(0x80+i, i);
     }

     Delay_ms(1000);

     for(i = 0; i < 32; i++)
     {
          PORTD = EEPROM_Read(0x80+i);
          Delay_ms(250);
     }
}
```

A mintaprogram azt csinálja, hogy 32 byte információt beleír a mikrovezérlőbe, majd 1 mp várakozás után elkezdi kiolvasni a beírt adatot, az adatot pedig a D porton megjeleníti az eszköz.

13. Analóg értékek olvasása

A legtöbb PIC mikrovezérlőben található egy beépített analóg-digitális átalakító. Ezen átalakító pontossága a kiválasztott mikrovezérlőtől függően eltérő lehet. Tipikusan 10 vagy 12 bit pontosságra lehet számítani. A csatornák száma szintén mikrovezérlőnként eltérő.

Az analóg bemenetek használatba vételének első lépése, az, hogy a megfelelő portot (legtöbb esetben PORTA és esetlegesen PORTE) bemenetre állítjuk, majd ezzel nem ütköző módon az ANSEL és ANSELH regiszter értékét beállítjuk. Az ANSEL regiszter teljes egészében az A porthoz van rendelve, míg az ANSELH regiszter az E porthoz. Mikrovezérlő függően az ANSELH regiszter nem minden bitje használatos. Erről érdemes tájékozódni a kiválasztott mikrovezérlő dokumentációjában.

Maga a MikroC 3db igen egyszerűen használatot biztosít analóg értékek olvasására. Ezek a következőek:

```
void ADC_Init();
```

Ez a függvény inicializálja az analóg-digitális átalakító könyvtárt.

```
unsigned ADC_Get_Sample(unsigned short channel);
```

Egy pillanatnyi mintát leolvas a kiválasztott analóg bemeneti csatornáról. Visszatérési értéke egy előjel nélküli egész szám, ami 10 vagy 12 bites, mikrovezérlőtől függően.

```
unsigned ADC_Read(unsigned short channel);
```

Beolvas egy analóg értéket a kiválasztott csatornáról. A fő különbség ezen függvény és a minta olvasó függvény között az, hogy ez a függvény fix időkeretig olvassa az analóg bemenetet. Az időkeretet az oszcillátor határozza meg, ezáltal pontosabb értéket ad vissza, viszont némileg több időt is vesz igénybe a használata.

Az alábbi példaprogram a könyvtár használatát mutatja be:

```
unsigned analog;

void main()
{
     ANSEL = 0x02;
     ANSELH = 0x00;
     TRISA = 0x02;
     TRISB = 0x00;
     TRISC = 0x00;
     TRISD = 0x00;
     TRISE = 0x00;

     ADC_Init();

     while(1)
     {
          analog = ADC_Read(1);
          PORTB = analog;
          PORTC = analog >> 4;
     }
}
```

A példa kód az RA1 lábra kötött potenciométer feszültségét digitalizálja, majd a beolvasott értéket megjeleníti a B és C porton úgy, hogy a beolvasott digitális érték alsó 8 bitje a B porton jelenik meg, míg a felső 4 bit a C porton.

A PIC16F887-es mikrovezérlő 10 bites pontosságú analóg bemenetekkel rendelkezik, ami azt jelenti, hogy 5

Voltos analóg referencia feszültség mellett 0,0048 828 125V eltérés jelent egy bitet.

Szinte minden mikrovezérlőn lehetőség van az analóg referencia feszültség megváltoztatására. PIC mikrovezérlők esetén a dokumentációban ezen lábakat VREF+ és VREF- feliratokkal szokták jelölni.

Az analóg referencia feszültség beállításánál ügyelni kell arra, hogy a feszültség ne haladja meg a tápfeszültséget (mikrovezérlőfüggő, de nem igen szereti egyik sem), és az sem mindegy, hogy mennyire pontos a feszültségünk. Ha a referencia feszültség változik, akkor az analóg érték olvasása is pontatlan lesz. Amennyiben a referencia feszültség kisebb a tápfeszültségnél, akkor a mérés pontosságát növelem, de a méréshatárom csökken. Példaképpen, ha a referencia feszültségemet 3,3V-ra állítom be, akkor 0,00322 265 625V eltérés jelent egy bitet, viszont maximum 3,3V-ig fogok tudni csak mérni.

Sok esetben a 10 vagy 12 bites pontosság bőven elegendő, ahol viszont nem, ott speciális célhardver alkalmazása szükséges, jelen esetben egy analóg-digitális átalakító. Ezekből többfajta kivitel létezik pontosság és kapcsolódási felület szempontjából, de általában vagy párhuzamos kapcsolódási felülettel rendelkeznek (ahány bit, annyi vezeték), vagy I2C buszrendszerre képesek csatlakozni, esetlegesen SPI buszra. A legtöbb analóg-digitális átalakítónak külső órajelre is szüksége van.

14. Véletlen szám generálása

Ténylegesen véletlen számot igencsak nehéz előállítani pusztán matematikai módszerekkel. Neumann János szerint: „Bárki, aki aritmetikai módszerekkel akar előállítani egy véletlen számot, a bűn állapotában leledzik.” A dolog nehézsége a tényleges véletlenszerűségben van.

A legtöbb aritmetikai megoldású véletlenszám generátor úgy működik, hogy inicializálni kell a generátort egy értékkel, majd ezután tudunk ténylegesen véletlen számot generáltatni.

Amennyiben ez elmarad, akkor a generátor mindig ugyanazokat a számokat fogja produkálni. Ezért a legegyszerűbb megoldás szerint a generátort számítógépek esetén az aktuális dátuminformációval szokták inicializálni. A legtöbb mikrovezérlő esetén ezt csak akkor tudjuk megtenni, ha egy RTC chip-et alkalmazunk az áramkörünkben.

Extra áramköri elemek nélkül is tudjuk inicializálni a véletlenszám generátort. Ehhez csupán egy szabadon hagyott analóg csatornára van szükségünk. Ha egy szabadon hagyott (sehova sem kötött) analóg csatornát próbálunk digitális értékként olvasni, azt tapasztaljuk, hogy minden esetben más értéket kapunk.

Ez annak tudható be, hogy a CMOS áramkörök belső felépítésükből adódóan igen érzékenyek a légköri töltésekre. Ezt az (általában) rossz tulajdonságot kihasználva előre kiszámíthatatlan inicializációs értéket kapunk minden esetben, így ténylegesen véletlen számokat fogunk kapni. Ilyen konfigurációk esetén az analóg referencia feszültséget érdemes kisebb értékre venni, ezáltal nagyobb véletlenszerűség érhető el.

A MikroC csupán két függvényt kínál véletlen számok kezeléséhez. Ezek:

```
int rand();
```

Generál egy véletlen számot 0 és az integer típus maximuma között, ami 32767. A maximálisan generált érték egyszerűen imitálható a maradékos osztás operátorral:

```
int random = rand() % 100;
```

A példában random értéke 0 és 100 közé eső szám lesz.

```
void srand(unsigned x);
```

Inicializálja a véletlenszám generátort x értékkel, számsorozat generálása előtt érdemes lefuttatni.

Az alábbi példában szereplő mintaprogram analóg csatornát felhasználva inicializálja a véletlenszám generátort, majd generál egy 8 bites számot, és azt a D porton megjeleníti binárisan:

```
void main()
{
	ANSEL = 0x02;
	ANSELH = 0x00;
	TRISA = 0x02;
	TRISB = 0x00;
	TRISC = 0x00;
	TRISD = 0x00;
	TRISE = 0x00;

	ADC_Init();
	while(1)
	{
		srand(ADC_Read(1));
		PORTD = rand()%255;
	}
}
```

15. Digitális dobókocka

A dobókocka program egy hétszegmenses kijelzőre generál egy vagy kettő 1 és hat közötti számot. Attól függően, hogy melyik bemenentre kötött gombot nyomta meg a felhasználó. A projekt a véletlen szám generálásra és a hétszegmenses kijelzők kezelésére épít.

```
unsigned char BCD7Szegmens(unsigned short num)
{
	switch (num)
	{
		case 0: return 0x3F;
		case 1: return 0x06;
		case 2: return 0x5B;
		case 3: return 0x4F;
		case 4: return 0x66;
		case 5: return 0x6D;
		case 6: return 0x7D;
		case 7: return 0x07;
		case 8: return 0x7F;
		case 9: return 0x6F;
		default: return 0xFF;
	}
}

int szam1, szam2, i;

void GenRandom()
{
	do { szam1 = rand()%7; }
	while (szam1 == 0);
	do { szam2 = rand()%7; }
	while (szam2 == 0);
}

void main()
{
	ANSEL = 1;
	ANSELH = 0;
	C1ON_bit = 0;
	C2ON_bit = 0;
	TRISA = 0x00;
	TRISB = 0x00;
	TRISC = 0x00;
	TRISD = 0x03;
	TRISE = 0x00;

	srand(ADC_Read(0));
	szam1 = 0;
	szam2 = 0;

	while (1)
	{
		if (PORTD.B0 == 1)
```

```
        {
            while (PORTD.B0 == 1) {}
            GenRandom();
            PORTB = BCD7Szegmens(szam1);
            delay_ms(3000);
        }
        if(PORTD.B1 == 1)
        {
            while (PORTD.B1 == 1) {}
            GenRandom();
            for (i=0; i<5; i++)
            {
                PORTB = BCD7Szegmens(szam1);
                delay_ms(500);
                PORTB = BCD7Szegmens(szam2);
                delay_ms(500);
            }
        }
        for (i=0; i<7; i++)
        {
            PORTB = 1 << i;
            delay_ms(100);
        }
    }
}
```

16. Egyszerű szintetizátor

Egy PS/2 csatlakozóval ellátott billentyűzetnek van legalább 100 gombja, ami bőven alkalmas arra, hogy egyszerű szintetizátort építsünk belőle. A MikroC könnyen tud generálni hangfrekvenciás tartományba eső négyszögjelet. A következő akadály a szintetizátor építés előtt az, hogy tudnunk kell a zenei hangok frekvencia értékét, amit majd a programunkba kell kódolni. Az alábbi ábrán a hangokhoz tartozó frekvencia értékek olvashatóak

Note	Hz	Note	Hz	Note	Hz	Note	Hz	Note	Hz	Note	Hz	Note	Hz
C1	32.7	C2	65.4	C3	130.8	C4	261.6	C5	523.3	C6	1046.5	C7	2093.0
C#1	34.6	C#2	69.3	C#3	138.6	C#4	277.2	C#5	554.4	C#6	1108.7	C#7	2217.5
D1	36.7	D2	73.4	D3	146.8	D4	293.7	D5	587.3	D6	1174.7	D7	2349.3
D#1	38.9	D#2	77.8	D#3	155.6	D#4	311.1	D#5	622.3	D#6	1244.5	D#7	2489.0
E1	41.2	E2	82.4	E3	164.8	E4	329.6	E5	659.3	E6	1318.5	E7	2637.0
F1	43.7	F2	87.3	F3	174.6	F4	349.2	F5	698.5	F6	1396.9	F7	2793.8
F#1	46.2	F#2	92.5	F#3	185.0	F#4	370.0	F#5	740.0	F#6	1480.0	F#7	2960.0
G1	49.0	G2	98.0	G3	196.0	G4	392.0	G5	784.0	G6	1568.0	G7	3136.0
G#1	51.9	G#2	103.8	G#3	207.7	G#4	415.3	G#5	830.6	G#6	1661.2	G#7	3322.4
A1	55.0	A2	110.0	A3	220.0	A4	440.0	A5	880.0	A6	1760.0	A7	3520.0
A#1	58.3	A#2	116.5	A#3	233.1	A#4	466.2	A#5	932.3	A#6	1864.7	A#7	3729.3
B1	61.7	B2	123.5	B3	246.9	B4	493.9	B5	987.8	B6	1975.5	B7	3951.1

Hang generálásra a MikroC két függvényt biztosít. Az egyik függvény segítségével meghatározhatjuk a hang kimenetet, a másik segítségével pedig hangot tudunk megszólaltatni.

```
void Sound_Init(char *snd_port, char snd_pin);
```

Inicializálja a hang generátort a megadott port megadott lábán. A port változó referenciaként adandó át. A porthoz tartozó TRIS regiszter értékét módosítani fogja a függvény.

```
void Sound_Play(unsigned freq_in_hz, unsigned duration_ms);
```

Adott frekvenciájú hangot szólaltat meg a második paraméter által meghatározott időig. Az időt milliszekundumban értelmezi.

A fenti táblázatban elrendezett hangok közül a # végződésűek úgynevezett félhangok. Ezek a zongorán a fekete billentyűknek felelnek meg, amelyek a fehér billentyűk felett helyezkednek el. Éppen ezért a félhangok a billentyűzeten a Q és Ú gomb között fognak elhelyezkedni, míg a normál hangok az A és Ű billentyűk között. Az egyszerűség kedvéért a program csak a C4 és B4 hangok közötti részt valósítja meg.

A kész program:

```
static char gombok[] = { 'a', 'w', 'd', 'e', 'f', 'g', 'z', 'h', →
'u', 'j', 'i', 'l' };
static unsigned int frekvencia = { 262, 277, 293, 311, 330, 349, →
370, 392, 415, 440, 466, 494 };
unsigned short keydata = 0, special = 0, down = 0;
int i=0, index = 0, hanghossz = 100;

sbit PS2_Data at RC0_bit;
sbit PS2_Clock at RC1_bit;
sbit PS2_Data_Direction at TRISC0_bit;
sbit PS2_Clock_Direction at TRISC1_bit;

short int keres(char gomb)
{
     for (i = 0; i<12; i++)
     {
          if (gombok[i] == gomb) return i;
     }
     return -1;
}

void hanghoszbeallit(char muvelet)
{
     if (muvelet == '+')
     {
          hanghossz += 25;
          Sound_Play(2000, 50);
     }
     else
     {
          hanghossz -= 25;
          Sound_Play(1000, 50);
     }
     if (hanghossz > 2000 || hanghossz < 25)
     {
          hanghossz = 100;
          Sound_Play(440, 50);
          delay_ms(50);
          Sound_Play(440, 50);
     }
}

void main()
{
     ANSEL = 0;
     ANSELH = 0;
     TRISA = 0;
     TRISB = 0;
     TRISC = 0;
     TRISD = 0;
     Ps2_Config();
     Sound_Init(&PORTD, 0);
```

```
	while (1)
	{
		if (Ps2_Key_Read(&keydata, &special, &down))
		{
			if (down)
			{
				switch (keydata)
				{
					case '0':
						hanghoszbeallit('-');
						break;
					case '1':
						hanghoszbeallit('+');
						break;
					default:
						i = keres(keydata);
						if (i <= 0) break;
						Sound_Play(frekvencia[i],
hanghossz);
						break;
				}
			}
		}
	}
}
```

A program a gombokat és a hozzájuk tartozó frekvenciákat két tömbben tárolja. A keres függvény egy adott gomb indexét adja vissza, majd ezen index alapján szólal meg a megfelelő hang, amennyiben sikerült olyan gombot megnyomni amely engedélyezett.

A 0 és 1 gombok segítségével módosítható az egy lenyomás által megszólaltatott hang hossza. Az alapértelmezett hang hossz 100 ms. Ehhez egy gombnyomás 25ms időt ad hozzá vagy vesz el, attól függően, hogy melyik gombot nyomtuk meg. Egy hang maximális ideje 2 másodpercre van korlátozva, minimális ideje pedig 25 ms.

A hang megszólaltatásához egy Piezzo hangszórót kell kötni a kimenetre, amely jelen program esetén a PORTD 0. lába. A Billentyűzet óra és adat jelét a program a PORTC 1-es és 0-s lábán várja.

17. SPI buszrendszer kezelése

A MikroC beépítetten biztosít függvényeket az SPI buszrendszer kezeléséhez is. A PIC16F887-es vezérlő esetén a C port lábaira vannak kivezetve a SPI funkciók. Az RC3 lábon található az SCK jel, az RC4 láb a MISO, az RC5 pedig a MOSI. A könyvtár által biztosított funkciók:

```
void SPIx_Init();
```

Inicializálja az x számmal rendelkező SPI buszt. Egyes PIC vezérlők több SPI busszal is rendelkezhetnek. A gyári alapértelmezett beállítások a következőek: Az eszköz Master módban fog működni, az órajel a rezgőkvarc órajelének negyede lesz, az órajel alapértelmezett állapota alacsony szint, adatátvitel felfutó él esetén történik és az adat az órajel közepén mintavételezendő.

```
void SPIx_Init_Advanced(unsigned short master_slav, unsigned
short→ data_sample, unsigned short clock_idle, unsigned short →
transmit_edge);
```

Bővebben konfigurált SPI inicializáció. A lehetséges beállítási értékeket és leírásukat az alábbi táblázat foglalja össze.

Paraméter neve	Lehetséges értékei	Leírás
master_slav	_SPI_MASTER_OSC_DIV4	Master üzemmód, órajel a kvarc 1/4-ed része
	_SPI_MASTER_OSC_DIV16	Master üzemmód, órajel a kvarc órajelének 1/16 -od része
	_SPI_MASTER_OSC_DIV64	Master üzemmód, órajel a kvarc órajelének 1/64-ed része
	_SPI_MASTER_TMR2	Master üzemmód, az órajelet a Timer2 megszakítása szolgáltatja
	_SPI_SLAVE_SS_ENABLE	Slave üzemmód engedélyezése
	_SPI_SLAVE_SS_DIS	Slave üzemmód kikapcsolása
data_sample	_SPI_DATA_SAMPLE_MIDDLE	Adat mintavételezés az órajel ciklus közepén.
	_SPI_DATA_SAMPLE_END	Adat mintavételezés az órajel ciklus végén
clock_idle	_SPI_CLK_IDLE_HIGH	Órajel alapértelmezett állapota magas szintű
	_SPI_CLK_IDLE_LOW	Órajel alapértelmezett állapota alacsony szintű
transmit_edge	_SPI_LOW_2_HIGH	Felfutó élvezérlés
	_SPI_HIGH_2_LOW	Lefutó élvezérlés

105. Táblázat: Bővebben konfigurált inicializáció paraméterei és konstansai

```
unsigned short SPIx_Read(unsigned short buffer);
```

Egy bájt olvasása az x jelű SPI buszról. Paramétere a küldendő adat, az SPI shift regiszter felépítése miatt van rá szükség. Nem kell tényleges adatnak lennie, de létező változónak kell lennie.

```
void SPIx_Write(unsigned short data_);
```

Adat írása a buszrendszerre. Paramétere az írandó byte.

18. Hálózatkezelés

A MikroC tartalmaz kiegészítő könyvtárt az SPI buszrendszerre illeszthető ENC28J60 Ethernet modul

kezeléséhez. Egy ilyen vezérlőre épülő Ethernet modul kapcsolása megtalálható a könyv végén szereplő mellékletek részben.

Ezen könyvtár kihasználásához egy PIC18 vezérlő ajánlott, mivel a PIC16-os sorozat számítási teljesítménye kevés a megfelelő használathoz. Ezen könyvtár bemutatásakor törekedtem az egyszerűségre és az érthetőségre, bár nem volt könnyű dolgom, mert a könyvtár dokumentációja igencsak hiányos.

A könyvtár használatához szükségünk lesz egy teljes verziós MikroC-re, mert a kód mérete át fogja lépni a méretkorlátot.

A könyvtár függvényei:

```
void SPI_Ethernet_Init(unsigned char *mac, unsigned char *ip,
unsigned char fullDuplex);
```

Inicializálja az Ethernet vezérlőt megadott MAC és IP címmel, full duplex vagy half duplex kommunikációs móddal. A mac paraméter egy 6 elemű karakter tömbként adandó meg, míg az IP cím egy négy elemű karakter tömbként. Az utolsó paraméter, ha 0, akkor a kommunikációs mód half duplex, egyes esetén pedig full duplex.

A függvény és a könyvtár rendes működéséhez definiálni kell az illesztő reset és cs vezetékeinek elhelyezkedését és irányát a következő módon:

```
sfr sbit SPI_Ethernet_Rst at RC0_bit;
sfr sbit SPI_Ethernet_CS at RC1_bit;
sfr sbit SPI_Ethernet_Rst_Direction at TRISC0_bit;
sfr sbit SPI_Ethernet_CS_Direction at TRISC1_bit;
```

Valamint a modul inicializálása előtt inicializálni kell az SPI buszrendszert is.

```
void SPI_Ethernet_Enable(unsigned char enFlt);
```

MAC rétegbeli bizonyos csomagok fogadásának engedélyezése. Meghívása kötelező, máskülönben nem fog működni a hálózat. Paramétere helyére vagy művelettel több konstans kapcsolható össze. A mindenképpen szükséges beállítás a CRC összeg számítása és a Unicast elérési mód. Ezek beállításához a következő konstansokkal kell meghívni a függvényt: _SPI_Ethernet_CRC | _SPI_Ethernet_UNICAST.

```
void SPI_Ethernet_Disable(unsigned char disFlt);
```

MAC rétegbeli bizonyos csomagok fogadásának tiltása. Alapértelmezetten nincsen szükségünk bármilyen csomag tiltására. A tiltható csomagok listája a MikroC dokumentációjában megtalálható.

```
unsigned char SPI_Ethernet_doPacket();
```

A programunk végtelenített ciklusában meghívandó függvény, ami a csomagok feldolgozását végzi. A tényleges csomag feldolgozás eseménykezelő függvényekben van feldolgozva.

Visszatérési értéke 0, ha minden rendben volt. 1-es, ha buffer túlcsordulás történt feldolgozás közben, 2-es, ha a csomag nem a mikrovezérlőnek lett címezve, 3-as ha nem IPv4 csomag, 4-es ha a csomag feldolgozhatatlan a mikrovezérlő által.

```
void SPI_Ethernet_putByte(unsigned char v);
```

Egy byte továbbítása Ethernet hálózaton keresztül.

```
void SPI_Ethernet_putBytes(unsigned char *ptr, unsigned int n);
```

Byte sorozat továbbítása Ethernet hálózaton keresztül. Első paramétere egy tömb, második paramétere pedig a tömbből küldendő elemek száma.

```
void SPI_Ethernet_putConstBytes(const unsigned char *ptr, unsigned
int n);
```

Működése megegyezik az SPI_Ethernet_putBytes függvénnyel, azzal a különbséggel, hogy ez egy program memóriában tárolt byte sorozatot továbbít.

```
unsigned int SPI_Ethernet_putString(unsigned char *ptr);
```

Szöveg továbbítása Ethernet hálózaton keresztül. Visszatérési értéke a továbbított karakterek száma.

```
unsigned int SPI_Ethernet_putConstString(const unsigned char
*ptr);
```

Működése megegyezik a SPI_Ethernet_putString fügvénnyel, azzal a különbséggel, hogy ez egy program memóriában tárolt szöveget továbbít.

```
unsigned char SPI_Ethernet_getByte();
```

Egy byte adatot fogad Ethernet hálózaton keresztül.

```
void SPI_Ethernet_getBytes(unsigned char *ptr, unsigned int addr,
unsigned int n);
```

Több byte adat fogadása Ethernet hálózaton keresztül. Lényegében a hálózati illesztő RAM memóriájából másol adatot a mikrovezérlőbe. Első paramétere a cél tömb, amibe másol, második paramétere a hálózati illesztő egy memória címe, ahonnan a másolás kezdődni fog, harmadik paramétere pedig a másolandó byte-ok száma.

```
unsigned int SPI_Ethernet_UserTCP(unsigned char *remoteHost,
unsigned int remotePort, unsigned int localPort, unsigned int
reqLength, TEthPktFlags *flags);
```

TCP csomag fogadása esetén megvalósítandó eseménykezelő függvény prototípusa. A programunkban létre kell hozni egy ilyen függvényt. Ez fog felelni a TCP csomagok feldolgozásáért, a SPI_Ethernet_doPacket() függvény fogja automatikusan meghívni TCP adat esetén.

Első paramétere a kliens IP címe egy 4 elemű karakter tömbben, második paramétere a kliens TCP portja, harmadik paramétere a mikrovezérlő portja, amire a kérés érkezett, negyedik paramétere a kérés hossza. Az ötödik paraméter egy a kérésre vonatkozó jellemzők struktúrált formában. A struktúra két paramétert tartalmaz. Ezzel az esetek többségében nem kell foglalkozni.

A függvény visszatérési értéke a küldendő byte-ok hosszát határozza meg. Amennyiben a kérésre nem kívánunk válaszolni a függvénynek 0 értéket kell visszaadnia.

```
unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost,
unsigned int remotePort, unsigned int destPort, unsigned int
reqLength, TEthPktFlags *flags);
```

Működése és paraméterezése hasonló a SPI_Ethernet_UserTCP függvényhez, azonban ez a függvény UDP csomagok kezelésére szolgál. A programban kötelező implementálni, ha nem végez adatfeldolgozást, akkor 0-t kell visszaadnia.

PIC18 Esetén használható további függvények

```
unsigned char * SPI_Ethernet_getIpAddress();
```

Lekérdezi a vezérlő által jelenleg használt IP címet. A címet egy négy karakterből álló tömbként adja vissza.

```
unsigned char * SPI_Ethernet_getGwIpAddress();
```

Lekérdezi a vezérlő által jelenleg használt alapértelmezett átjáró IP címét. A címet egy négy karakterből álló tömbként adja vissza.

```
unsigned char * SPI_Ethernet_getDnsIpAddress();
```

Lekérdezi a vezérlő által jelenleg használt DNS kiszolgáló IP címét. A címet egy négy karakterből álló tömbként adja vissza.

```
unsigned char * SPI_Ethernet_getIpMask();
```

Lekérdezi a vezérlő által jelenleg használt alhálózati maszkot. A maszkot egy négy karakterből álló tömbként adja vissza.

```
void SPI_Ethernet_confNetwork(char *ipMask, char *gwIpAddr, char
*dnsIpAddr);
```

Hálózat konfigurálása DHCP nélkül. A függvény összes paraméterének típusa egy négy karakterből álló tömb. Az első paraméter az IP címet, a második az alapértelmezett átjáró, a harmadik paraméter pedig a DNS kiszolgáló címét határozza meg.

```
unsigned char *SPI_Ethernet_arpResolve(unsigned char *ip, unsigned
char tmax);
```

ARP protokoll segítségével meghatározza az első paraméterként kapott IP címhez tartozó MAC címet. A második paramétere egy maximális várakozási idő.

```
unsigned char SPI_Ethernet_sendUDP(unsigned char *destIP, unsigned
int sourcePort, unsigned int destPort, unsigned char *pkt,
unsigned int pktLen);
```

UDP csomag továbbítása a hálózaton. Első paramétere a cél IP cím, második paramétere a forrás port, harmadik paramétere a cél port, negyedik paramétere a csomag karaktertömbként, ötödik paramétere pedig a küldendő byte-ok száma. Visszatérési értéke 1, ha a csomag küldése sikeres volt.

```
unsigned char * SPI_Ethernet_dnsResolve(unsigned char *host,
unsigned char tmax);
```

Egy DNS címhez tartozó IP cím meghatározása. Első paramétere a DNS cím, második paramétere egy maximális várakozási idő.

```
unsigned char SPI_Ethernet_initDHCP(unsigned char tmax);
```

DHCP hálózati konfiguráció engedélyezése. Paramétere egy maximális várakozási időt határoz meg. Visszatérési értéke 1-es, ha a konfigurálás sikeres volt.

```
unsigned char SPI_Ethernet_doDHCPLeaseTime();
```

Ellenőrizi a DHCP konfigurációs információk lejártát. Visszatérési értéke 0, ha a kapott DHCP információk még érvényesek. Ha a visszatérési értéke 1, akkor az információkat meg kell újítani.

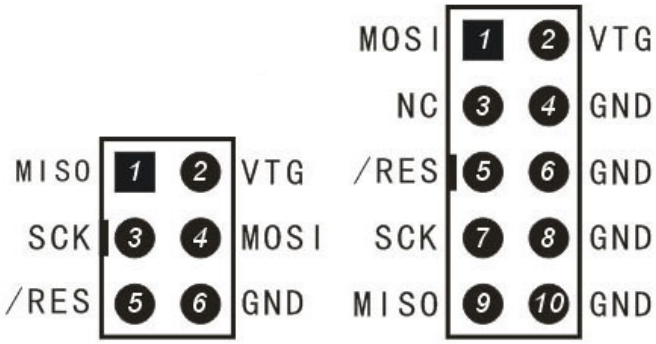
```
unsigned char SPI_Ethernet_renewDHCP(unsigned char tmax);
```

DHCP konfiguráció megújítása. Paramétere egy maximális várakozási időt határoz meg. Visszatérési értéke 1-es, ha a megújítás sikerült.

22. Arduino 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.



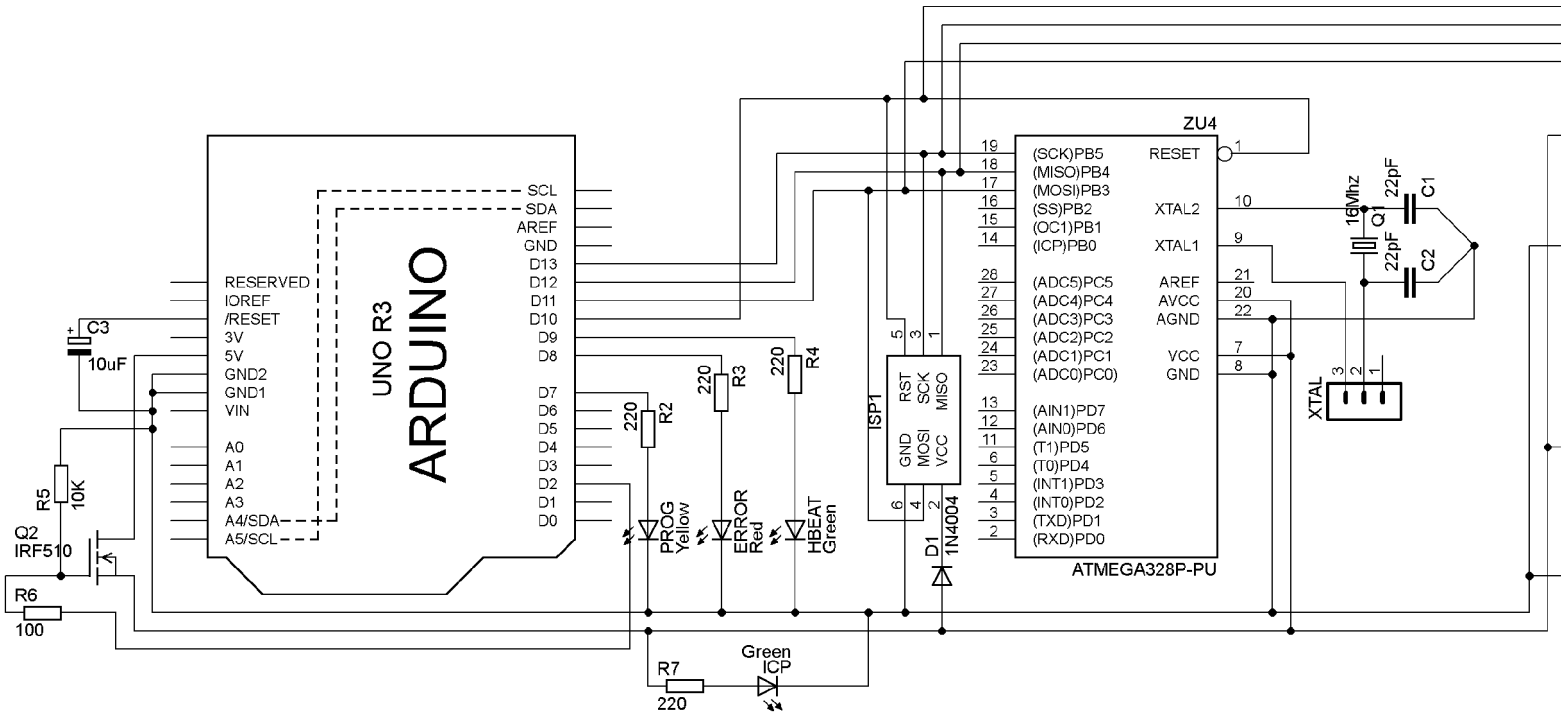
Az Arduino-ban használt mikrovezérlő a bootloader nélkül nem lenne használható az Arduino környezetben, így ha egy projektünket meg akarjuk valósítani nyomtatott áramkörben, akkor vennünk kell vezérlő chip-et (mondjuk egy Atmega328-at, amit az Uno-ban használnak). Erre először fel kell programoznunk 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.


ARDUINO
UNO R3
SCL
SDA
AREF
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
RESERVED
IOREF
/RESET
3V
5V
GND2
GND1
VIN
A0
A1
A2
A3
A4/SDA
A5/SCL
C3
10uF
R5
10K
Q2
IRF510
R6
100
220
R2
220
R3
220
R4
PROG
Yellow
ERROR
Red
HBEAT
Green
ISP1
RST
SCK
MISO
GND
MOSI
VCC
D1
1N4004
ZU4
(SCK)PB5
(MISO)PB4
(MOSI)PB3
(SS)PB2
(OC1)PB1
(ICP)PB0
(ADC5)PC5
(ADC4)PC4
(ADC3)PC3
(ADC2)PC2
(ADC1)PC1
(ADC0)PC0
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
ATMEGA328P-PU
16Mhz
Q1
22pF
C1
22pF
C2
XTAL
R7
220
Green
ICP

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

106. 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:

230. 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.

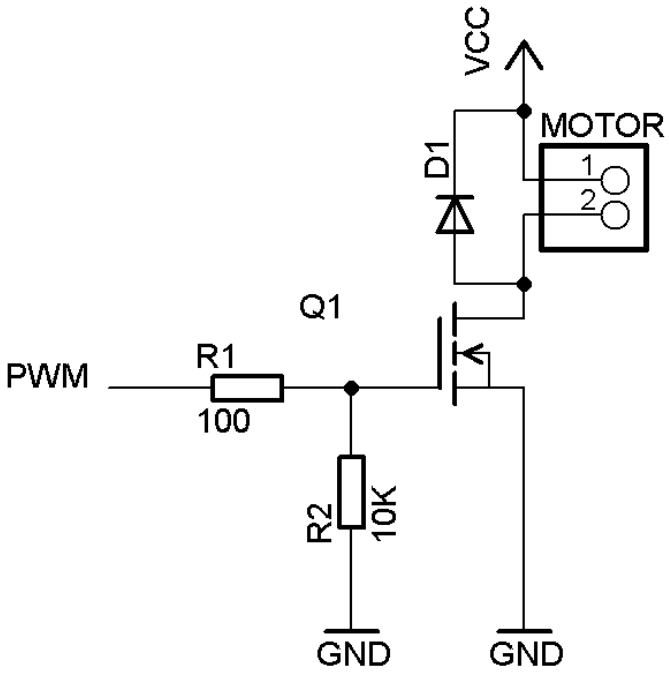
A két projekt közös jellemzője, hogy a limitált memóriaméret (4Kb vagy 8Kb) miatt nem minden Arduino könyvtár működik ezen mikrovezérlőkkel. Az alap analóg és digitális ki/bemenet kezelő függvények és a késleltetések biztosan működnek, más azonban kétséges. Ezen függvényekkel is igen sok minden megvalósítható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

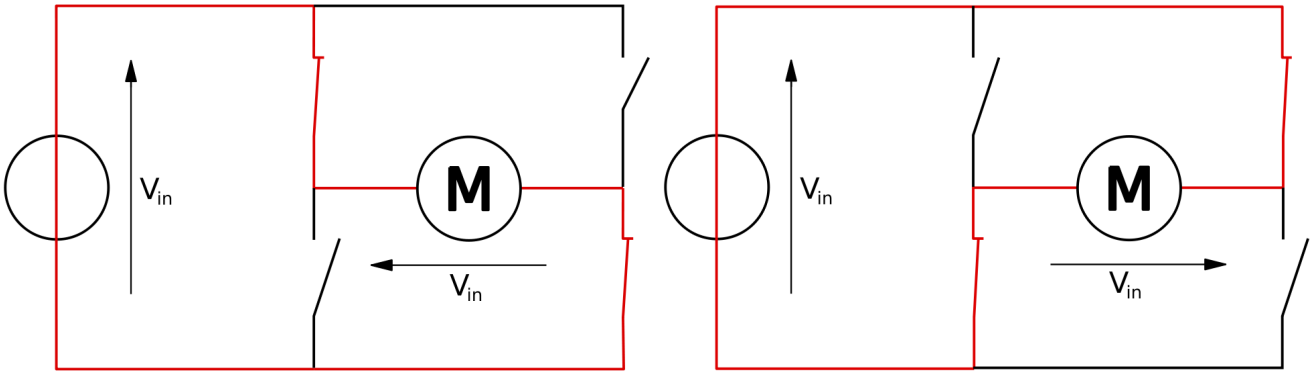
A vezérlése ezen motoroknak igen egyszerű, mivel a két kivezetés polaritása határozza meg a motor forgásirányát. A sebesség szabályozása PWM moduláció segítségével lehetséges. Az alábbi ábrán egy PWM modulációt használó sebesség szabályozó kapcsolás látható.



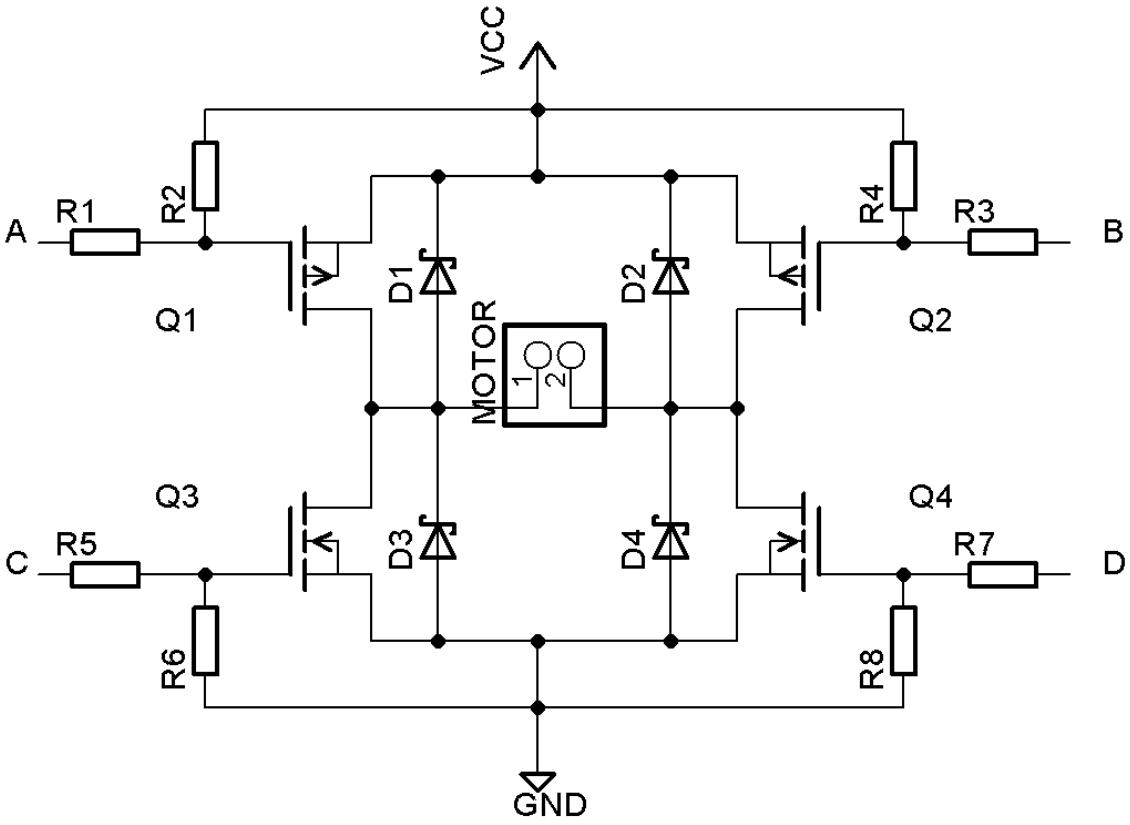
A mikrovezérlő által szolgáltatott PWM jel áramerőssége szinte sosem elég egy motor meghajtásához. Ezért a PWM jelet egy tranzisztorra/FET-re kell vezetni, ami a motort fogja közvetlenül kapcsolgatni. Tranzisztor helyett érdemes FET-et alkalmazni, mivel FET-ek segítségével jobb hatásfok érhető el. A 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.



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.



233. 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*

107. 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 működés főbb állapotait 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*

108. Táblázat: Négy vezérlőjeles H híd vezérlése négy vezérlőjellel

A táblázat nem tartalmazza, de az egy oldalon lévő FET-ek egyszerre sosem nyithatóak ki, mivel ekkor a kapcsolás zárlati állapotba kerül.

Léptetőmotorok

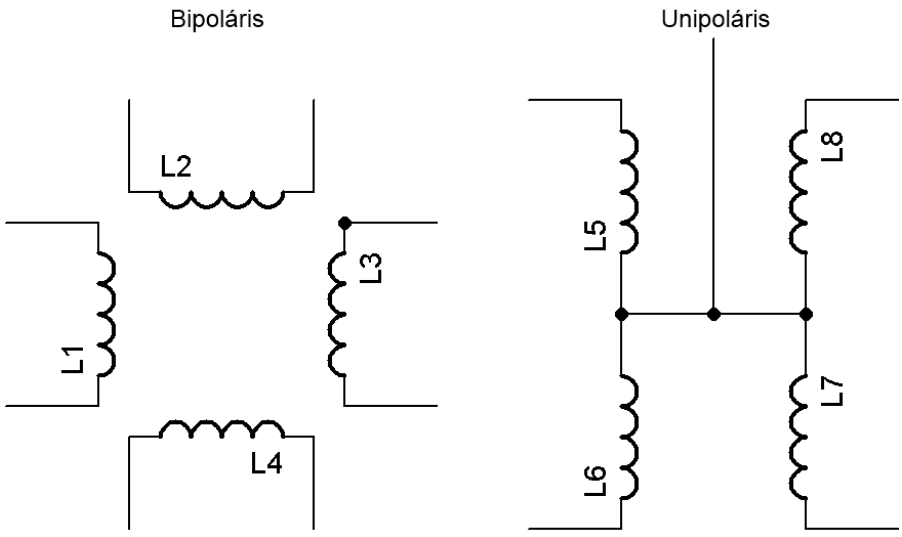
A léptetőmotorok felépítésükben igencsak különböznek a hagyományos egyenáramú motoroktól. Ezen motorok digitális vezérlésűek, jel hatására egy előre meghatározott pozícióba fordul a motor forgórésze. Olyan területen alkalmazzák ezen motorokat, ahol lényeges az, hogy pontos pozícióban álljon meg a motor. 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.



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.

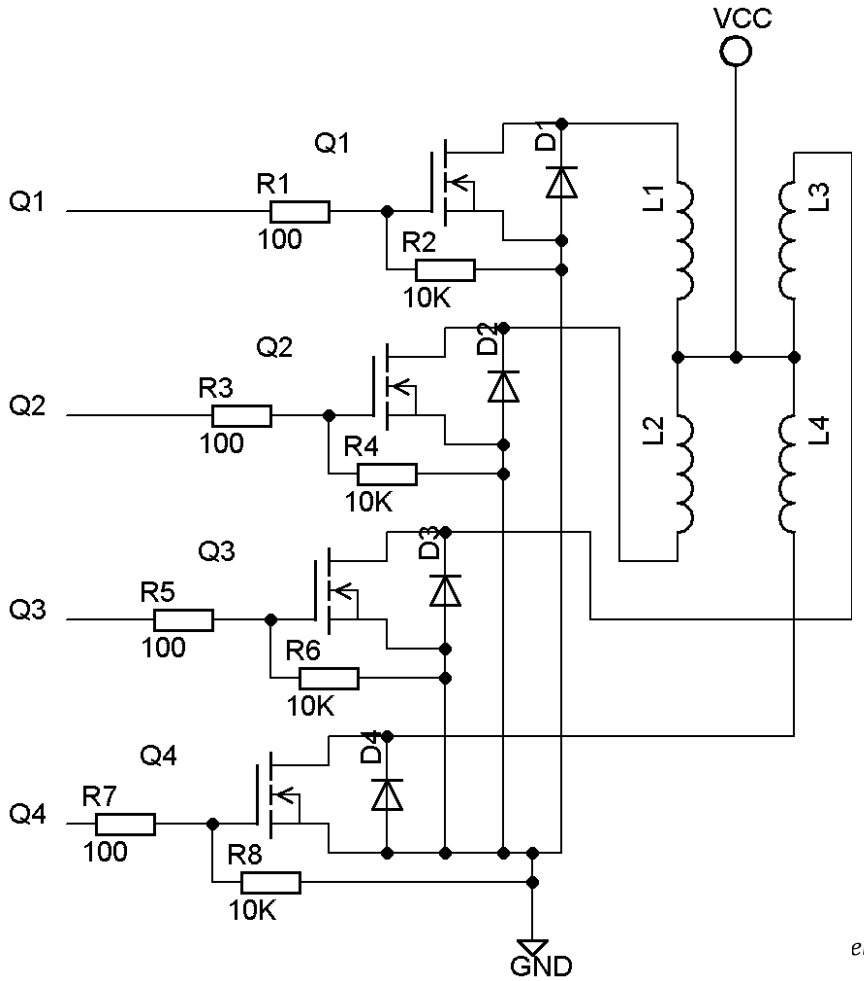
- ***Kétfázisú teljes léptetéses üzemmód***

  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***

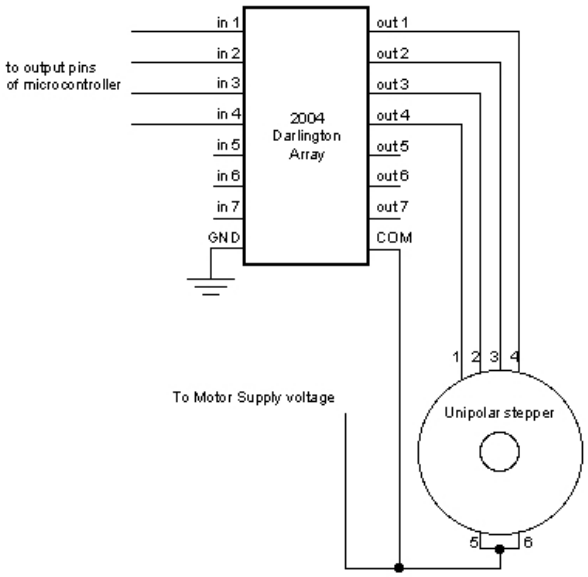
  Ezen üzemmód az egyfázisú és a kétfázisú léptetés keveréke. Egy tekercs kapcsolása után az adott tekercsnek és a mellette lévő tekercsnek adunk feszültséget egyszerre. Így a lépésszám megduplázható, pontosabb lesz a motor, cserébe viszont egy teljes fordulat több időt fog igénybe venni.

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.

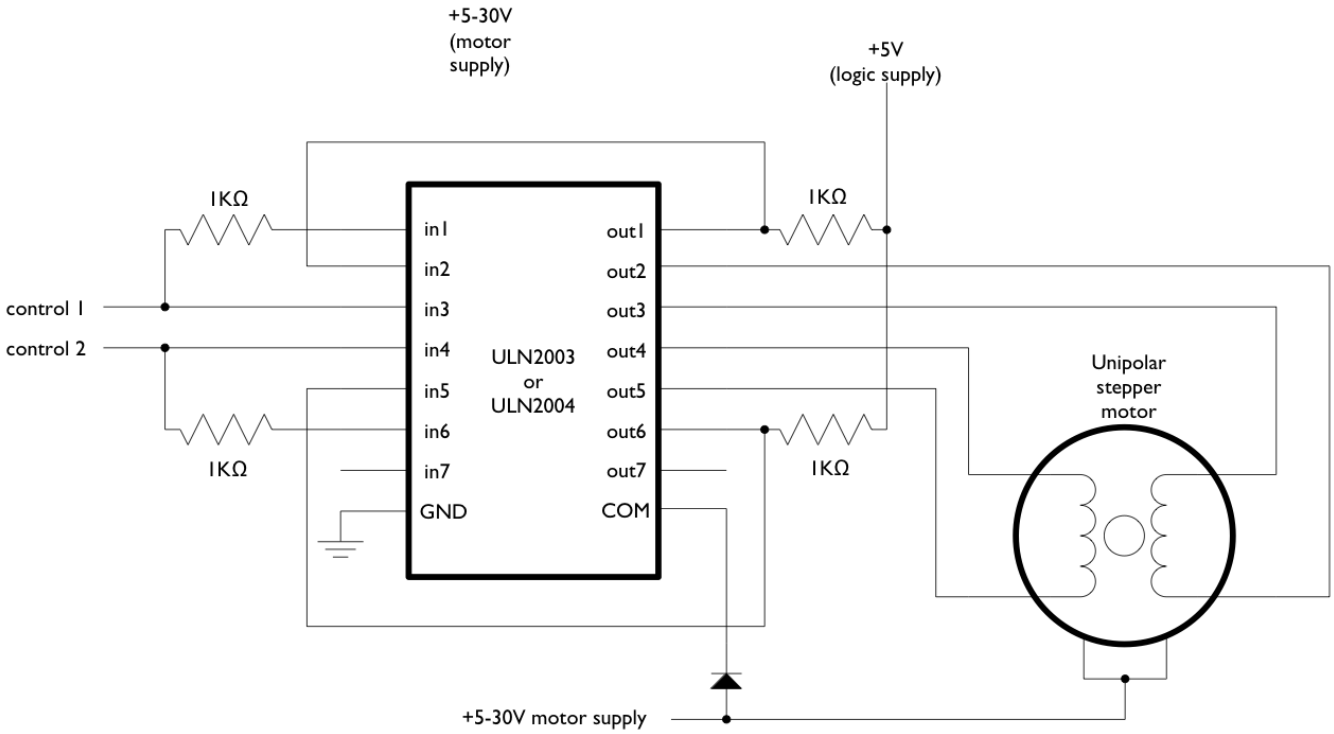


Kisebb motorok esetén elegendő az első felépítésű, ezáltal rosszabb hatásfokot fog biztosítani. Az Arduino oldalon ezt az áramkört ajánlják unipoláris motorokhoz. Az alábbi ábrán látható egy unipoláris motor bekötési rajza. A rajz a hivatalos Arduino oldalról származik.

236. ábra:



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.



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.

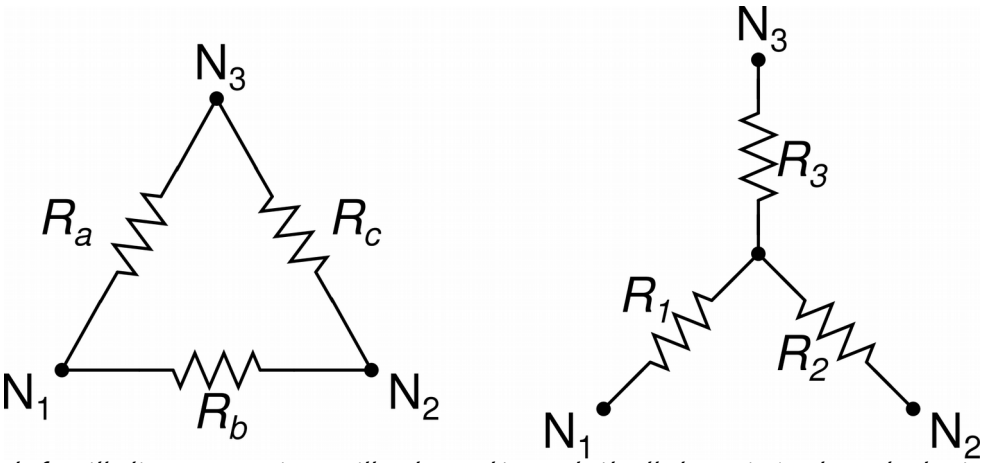
Kefe nélküli egyenáramú motorok (BDLC)

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.



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. 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:

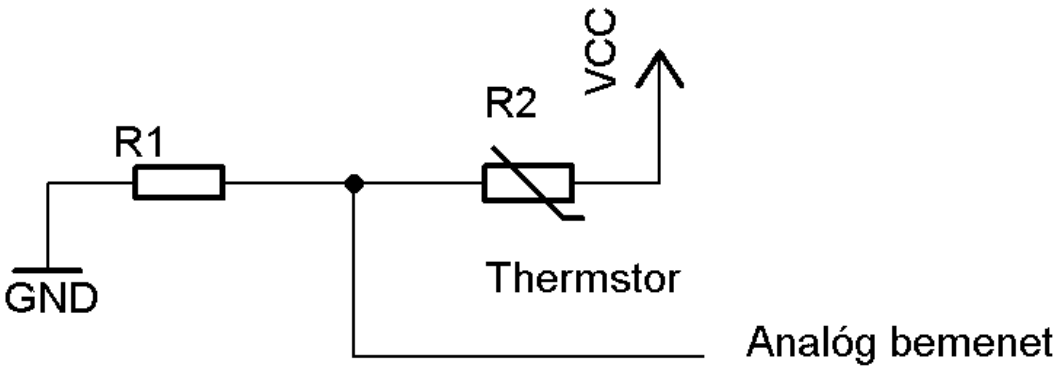
$$\frac{1}{T} = A + B \cdot ln(R) + C \cdot (ln(R))^3$$

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:

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

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:



A kezelő kód feszültségosztó ellenáll együtthatókat, az adott termisztor esetén jó közelítéssel alkalmazhatjuk a következő konstansokat:

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

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;
}
```

4. 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.

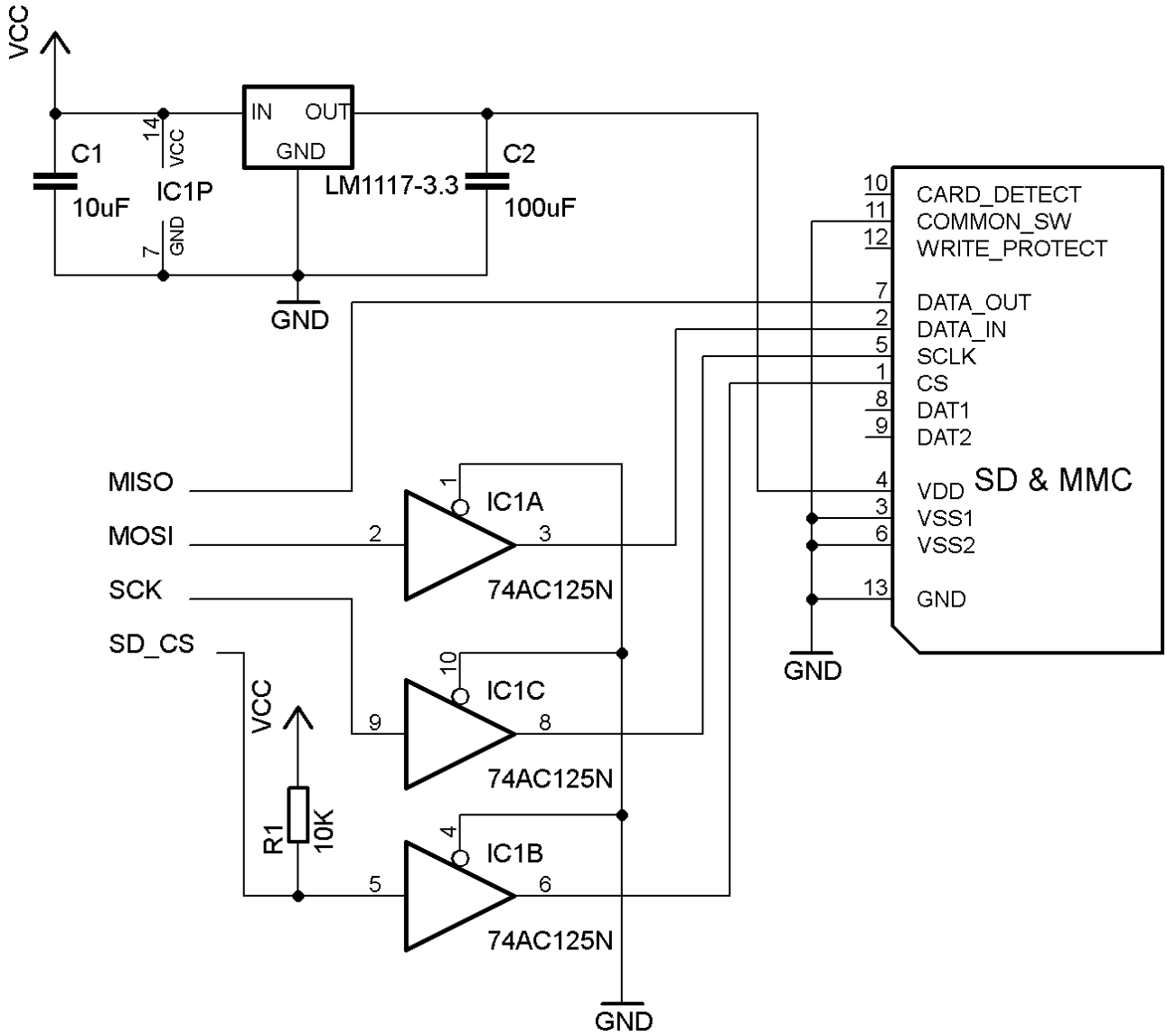
Ezen információk közé tartozik a kártya típusa és mérete. Mivel ezek nem olyan információk, amelyekre minden SD kártya kezelő alkalmazás esetén szükségünk van, ezért nincsenek külön dokumentálva.

Kapcsolási rajz

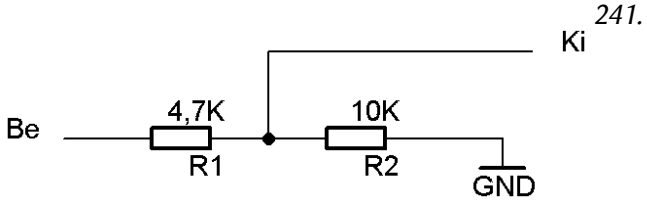
Az SD kártyák 3,3V feszültségről működnek és emiatt 3,3V-os logikai jelekkel kommunikálnak. Közvetlenül, jelszint konverzió nélkül nem köthetőek rá egy Arduino-ra sem, mert ez a kártya károsodásához vezetne. Ezért a jelszinteket illeszteni kell. Ezt megtehetjük passzív módon, ellenállásokból kialakított 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.



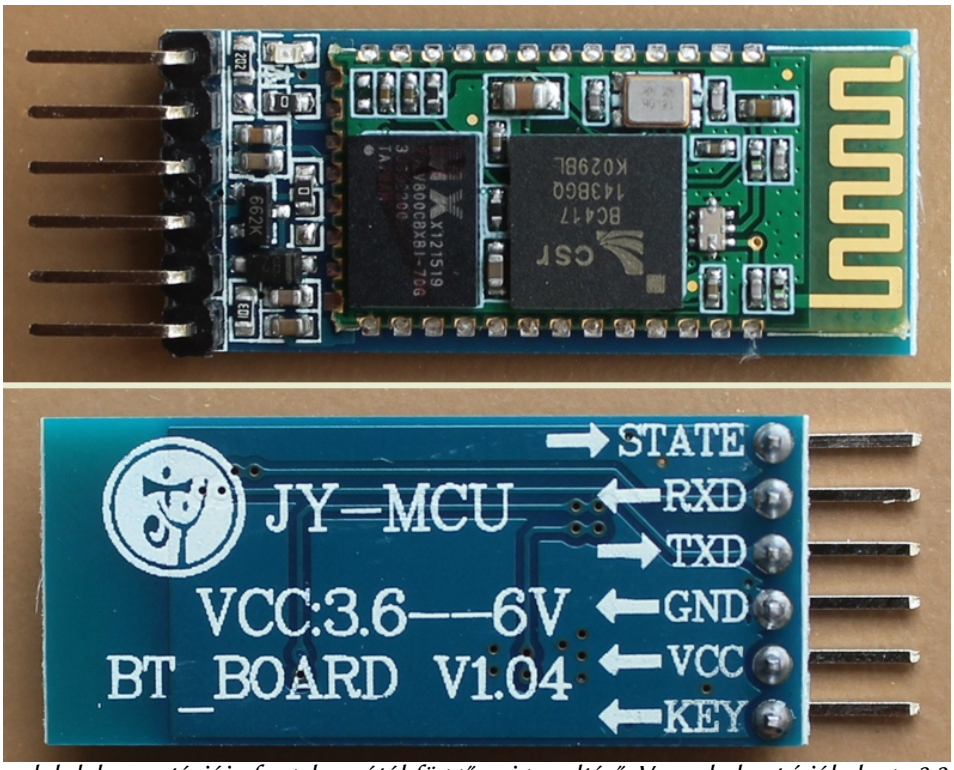
Az alábbi ábrán látható jelszint illesztés csak tesztelési célokra ajánlott:



ábra: Feszültségosztós jelszint illesztés

5. 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.



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.

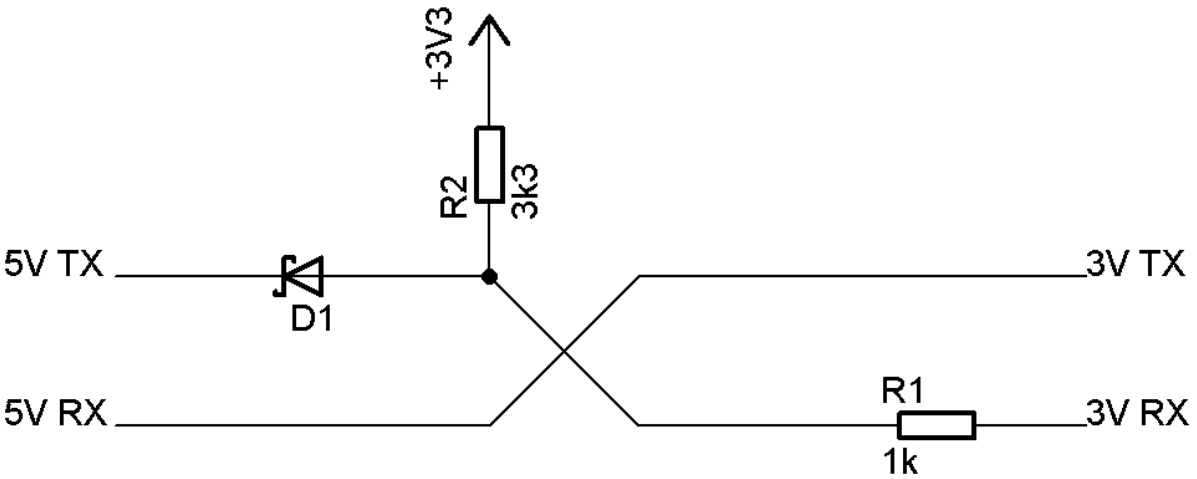
Az eszköz bekötése után a soros átalakítóra, majd megnyitva a megfelelő portot parancsszavak segítségével konfigurálhatjuk az eszközt. Válasz az eszköztől csak felismert parancs esetén fog érkezni. Az általam használt modul parancskészletét az alábbi táblázatban foglaltam össze.

<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>

109. 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.

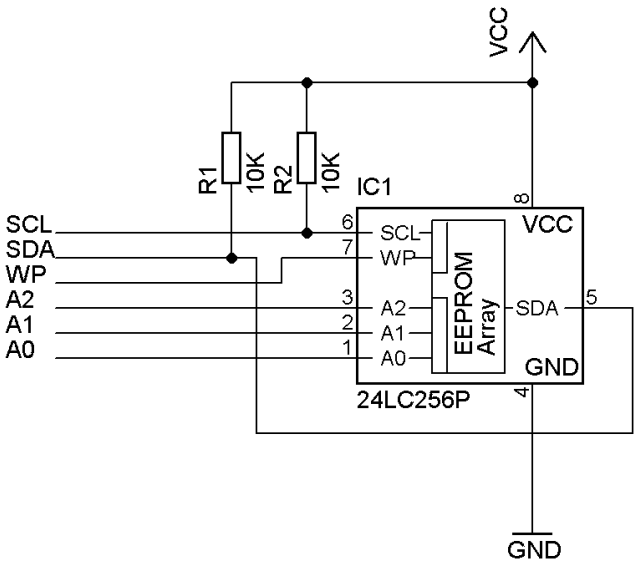
Az EGBT modulok dokumentációja alapján a 3.3V-os jelszintű modulok helyes illesztése 5V-os mikrovezérlőre a következő ábrán látható.



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

6. 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.



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*

110. táblázat: A memória lehetséges címei

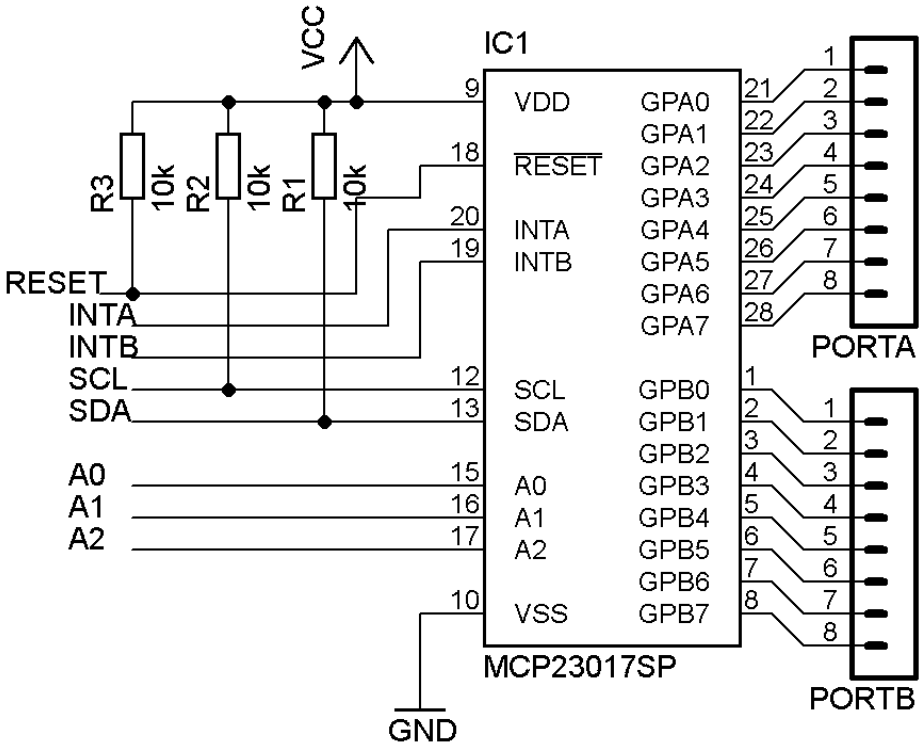
A memória kezelése úgy történik, hogy elküldjük a buszon keresztül a használni kívánt eszköz címét. Ezután két byte-on elküldjük a kért memóriacímet, majd a címzési módtól függően (írási vagy olvasási kérés) vagy küldünk még egy byte-ot, ami az adat lesz, vagy az eszköz küld egy byte-ot, ami a memóriacímen 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;
}
```

7. 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.



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*

111. 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.*

112. 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);
```

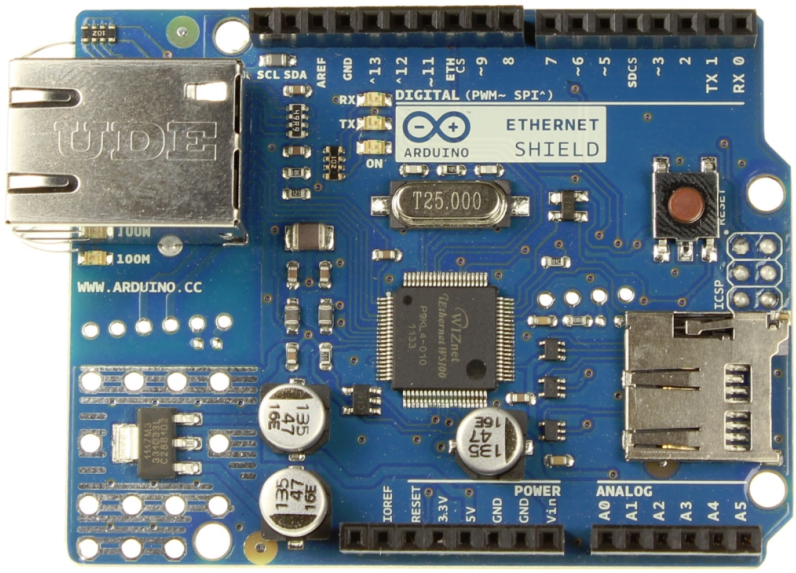
Adott láb értékének beállítása. A második paraméter két konstans lehet: HIGH vagy LOW.

A könyvtár használatához szükséges minden esetben az I²C 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);
}
```

8. Az Arduino Ethernet könyvtára

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.



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.

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 matricán. 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();
```

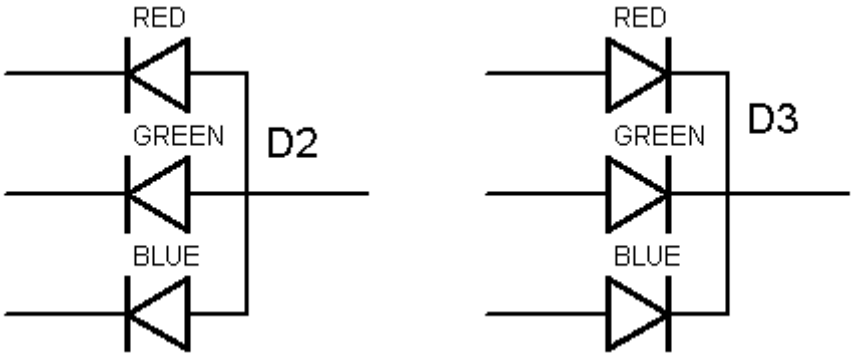
Egy csomagból olvassa ki a feladó IP címét, ezért használata előtt meg kell hívni a parsePacket(); függvényt. Az IP címet egy négy byte-ból álló tömb formájában adja vissza, vagy IP cím osztályként.

```
UDP.remotePort();
```

Egy csomagból olvassa ki a feladó által használt portot, ezért használata előtt meg kell hívni a parsePacket(); függvényt.

9. 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.



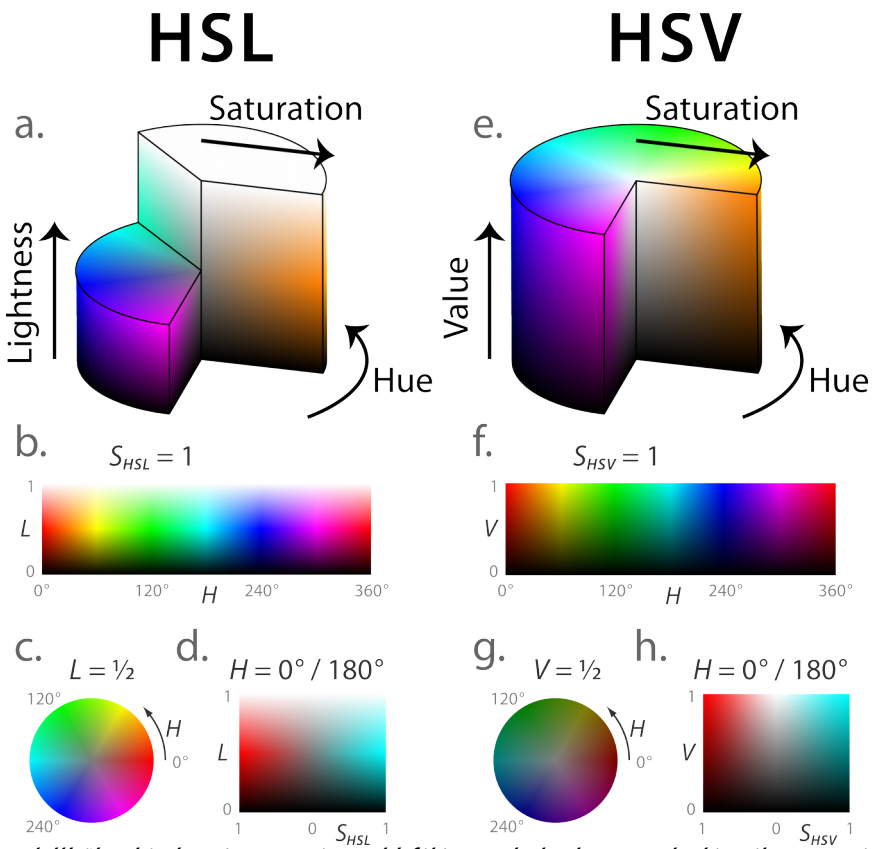
Színkeverési modellek

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.

248. 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.



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. A könyvtár a https://github.com/webmaster442/ArduinoExtensions/tree/master/libraries/RGBLedLib címen szerezhető be. A függvénykönyvtár két osztályt definiál. A Color osztályt és az RGBLed osztályt.

A Color osztály szín információk tárolására szolgál, míg az RGBLed osztály a színkeverést valósítja meg.

Color osztály dokumentációja

```
Color();
```

A paraméter nélküli konstruktor fekete színnel (csupa nulla érték) inicializálja az osztály adattagjait.

```
Color(byte r, byte g, byte b);
```

Inicializálja a színt a három komponens megadásával.

```
Color(unsigned long int value);
```

Inicializálja a színt egy egész szám alapján. A számot hexadecimális formátumban érdemes megadni, mint a webes színeket. A 0xFF0000 érték a piros színt reprezentálja.

```
byte R();
byte G();
byte B();
```

Lekérdezik a piros, zöld, kék színkomponensek értékeit.

```
void R(byte value);
void G(byte value);
void B(byte value);
```

Beállítják a piros, zöld és kék színkomponensek értékeit.

RGBLed osztály dokumentációja

```
RGBLed(int pwmR, int pwmG, int pwmB, int mode);
```

Létrehozza az RGBLed objektumot. Az első három paraméter a színkeveréshez használt piros, zöld és kék PWM csatornát állítja be, míg az utolsó paraméter az üzemmódot. Az üzemmód a LED típusát határozza meg. CC konstans reprezentálja a közös katódos LED-et, míg CA konstans a közös anódos LED-et reprezentálja.

```
void CurrentColor(Color c);
```

Beállítja az aktuális kimeneti színt egy Color objektum alapján.

```
void CurrentColor(byte r, byte b, byte g);
```

Beállítja az aktuális kimeneti színt a három színkomponens megadásával.

```
void CurrentColor(unsigned long int value);
```

Beállítja az aktuális kimenetet gy egész szám alapján. A számot hexadecimális formátumban érdemes megadni, mint a webes színeket.

```
Color CurrentColor();
```

Lekérdezi az akutális kimeneti színt.

```
void Alpha(byte value);
```

Beállítja a LED globális fényerejét.

```
byte Alpha();
```

Lekérdezi a LED globális fényerejét.

```
void Fade(Color &in, Color &out);
void Fade(unsigned long int in, unsigned long int out);
```

Lassú, egyenletes átkeverés az egyik színből a másikba. A színek megadhatók Color objektumokal vagy egész számokként.

Beépített konstans színek

A kezelőkönyvtár rendelkezik pár beépített konstans színnel. Az alábbi táblázat a konstansok neveit tartalmazza

Konstans	Konstans
COLOR_RED	COLOR_GREEN
COLOR_BLUE	COLOR_TURQOISE
COLOR_BLACK	COLOR_DARKBLUE
COLOR_WHITE	COLOR_VIOLET
COLOR_DARKRED	COLOR_GRAY25
COLOR_PINK	COLOR_GRAY50
COLOR_TEAL	COLOR_DARKYELLOW
COLOR_LIGHTGREEN	COLOR_YELLOW

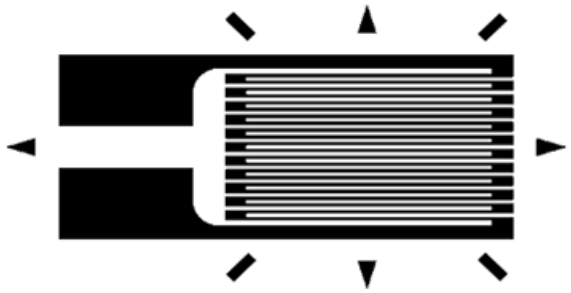
113. Táblázat: Az RGBLib szín konstansai

10. Nyúlásmérő bélyegek alkalmazása

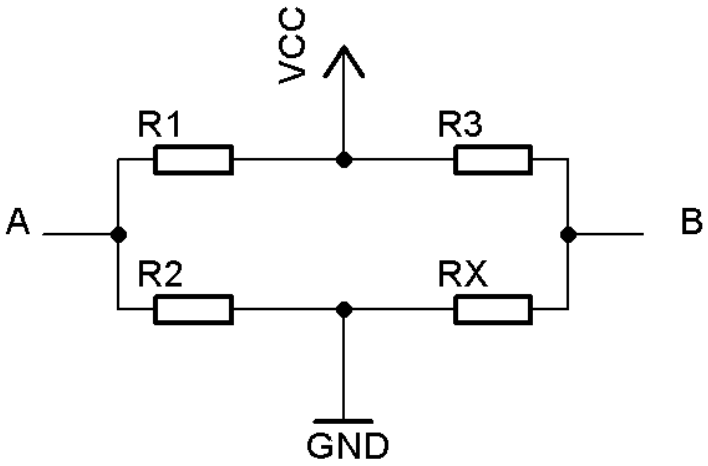
A nyúlásmérő bélyegek belső felépítésének az elve az, hogy a benne alkalmazott villamos vezető anyag

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.



A Wheatstone-híd egy mérőhíd, amelyet Sir Charles Wheatstone fejlesztett ki ellenállások mérésére. A kapcsolásban 4db ellenállás szerepel, amelyek közül egyik értéke nem ismert.



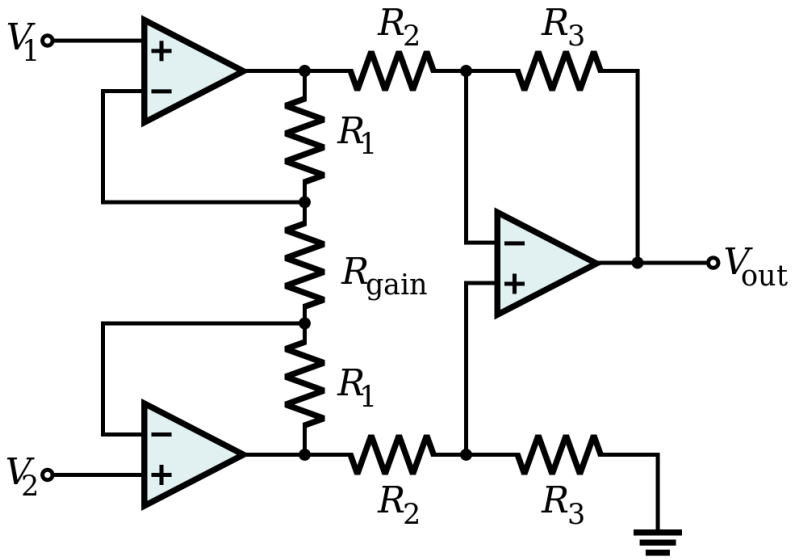
Amennyiben az ábrán jelölt A és B pont között egyik irányba sem folyik áram, akkor a híd kiegyenlített állapotban van. Ebben az esetben az RX ellenállás értéke a következő képlet alapján határozható meg:

$$RX = \frac{R2}{R1} \cdot R3 = \frac{R3}{R1} \cdot R2$$

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.



A kapcsolás kimeneti feszültsége a bemeneti feszültségek függvényében az alábbi képlet segítségével számolható ki:

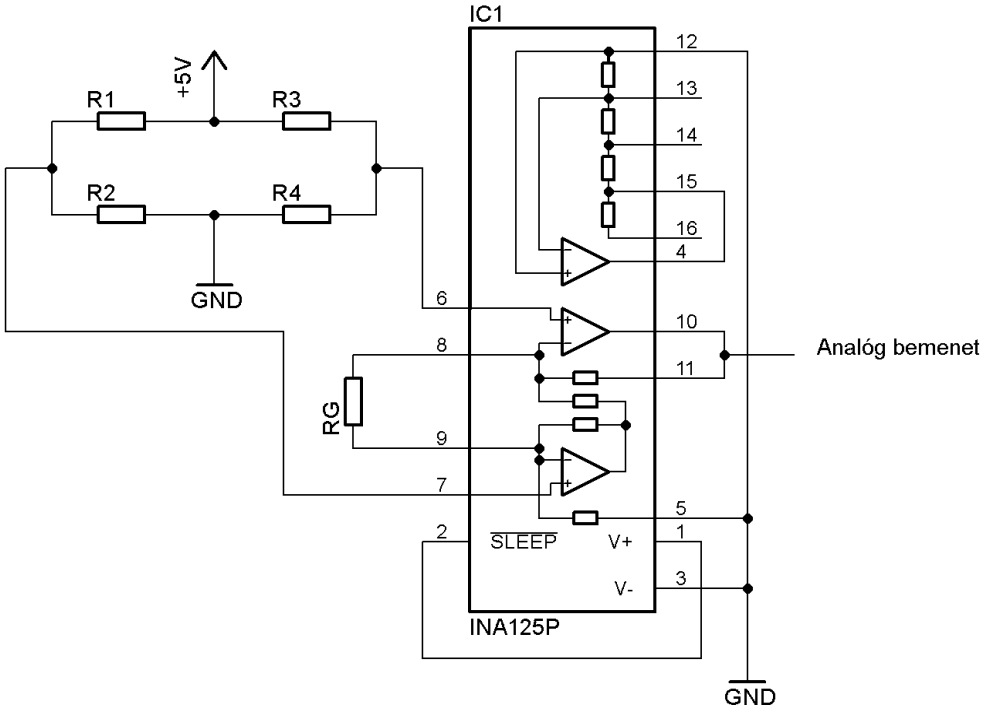
$$U_{Ki} = \frac{\left(1 + \frac{2 \cdot R_1}{R_{gain}}\right) \cdot R3}{R2} \cdot (U_2 - U_1)$$

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.



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.

11. 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();
}
```

12. 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.

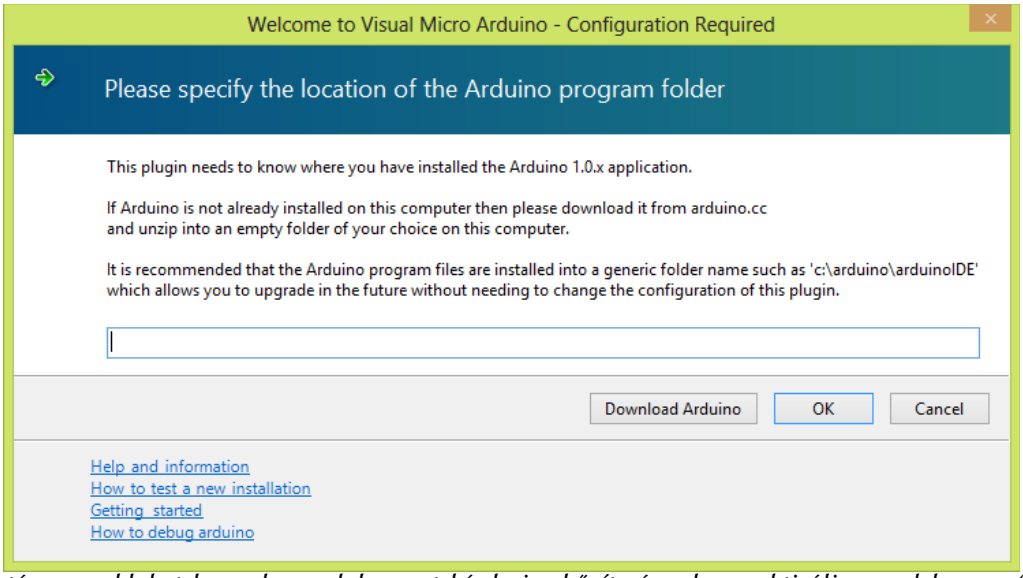
Használatához két előfeltétel is van. A Visual Studio-ba telepítve kell lennie legalább egy alapszintű C++ támogatásnak. Erre azért van szükség, mert másképpen nem működőképes az InteliSense támogatás.

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.

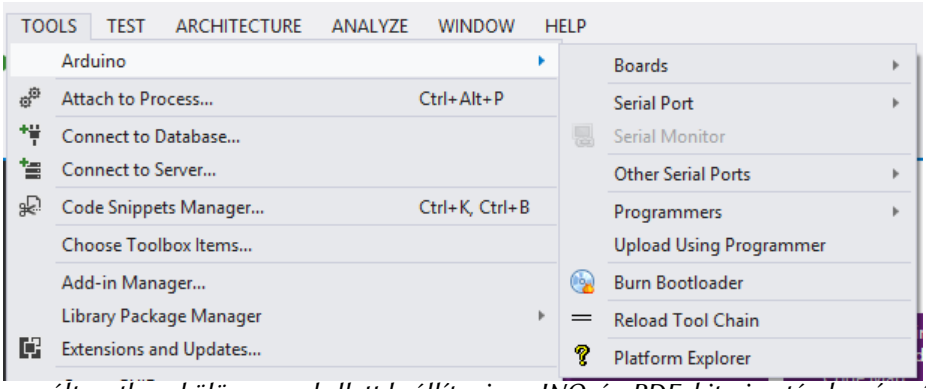
Telepítés és beüzemelés

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.

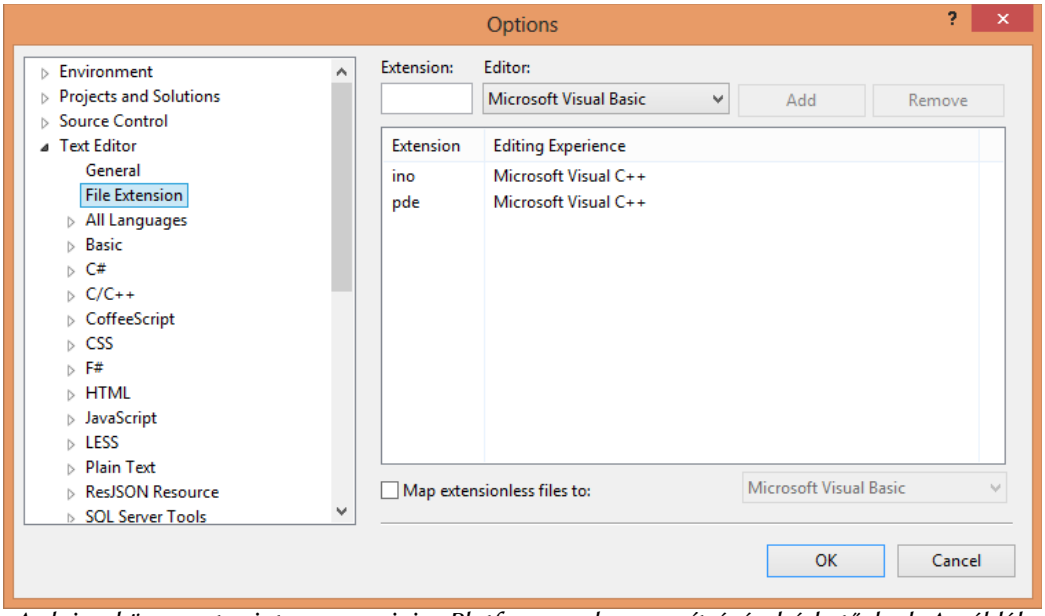


Ezután egy ablakot kapunk, amelyben azt kérdezi a bővítmény, hogy aktiválja-e a debugger (hibakereső) funkciót. Ez a része a programnak fizetős, azonban 30 napig ingyenesen kipróbálható, később ez a funkció csak vásárlás után lesz használható. Itt választhatjuk a 30 napos próbát vagy egyszerűen kattinthatunk a mégsem gombra. Ekkor a funkció egyáltalán nem elérhető.

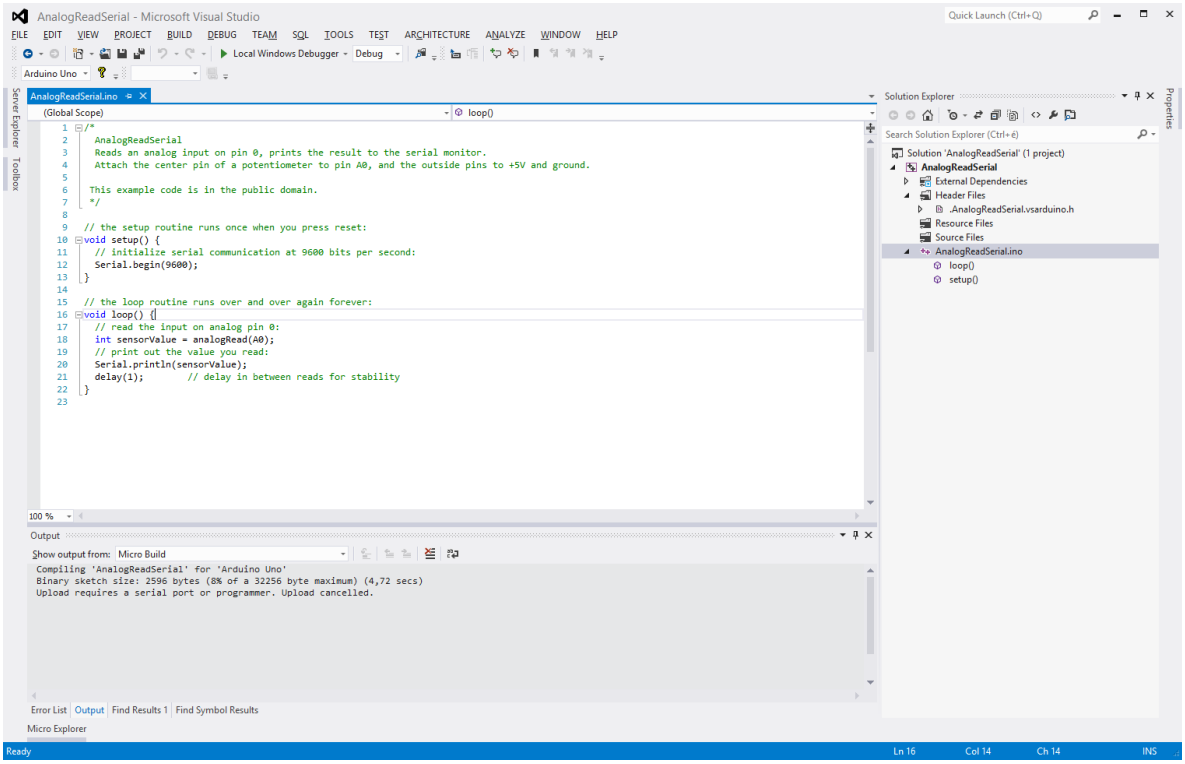
A program indulása után a TOOLS menüben láthatóvá válik egy Arduino menüpont. Innen érhetjük el az Arduino funkciókat.



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.



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.



13. 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.*

114. 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.

14. 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.

15. Arduino Uno készítése házilag

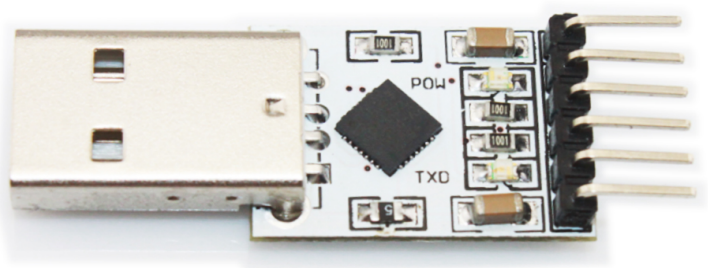
Az Arduino nyílt forrású hardver és szoftver. Ennek ellenére igencsak nehéz házilag a tervrajzok alapján építeni egy klónt bármelyik modellről. Ennek hardveres oka van.

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.



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.

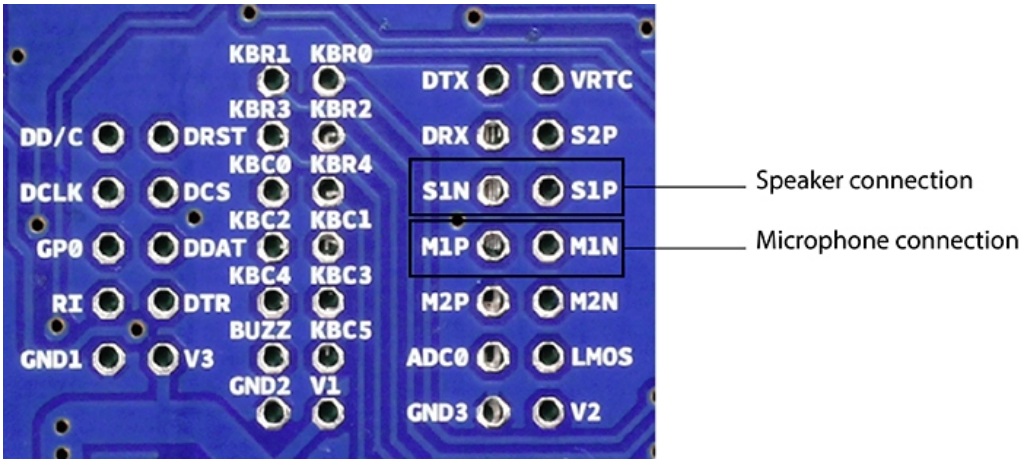
Ezen okokból, és mert a terv nyílt forráskódú, nem tudok rá vállalni semmilyen felelősséget. A kapcsolási rajz igen egyszerű felépítésű, megtalálható a mellékletek fejezetben és a CD mellékleten is a nyomtatott áramköri tervvel együtt.

16. 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.

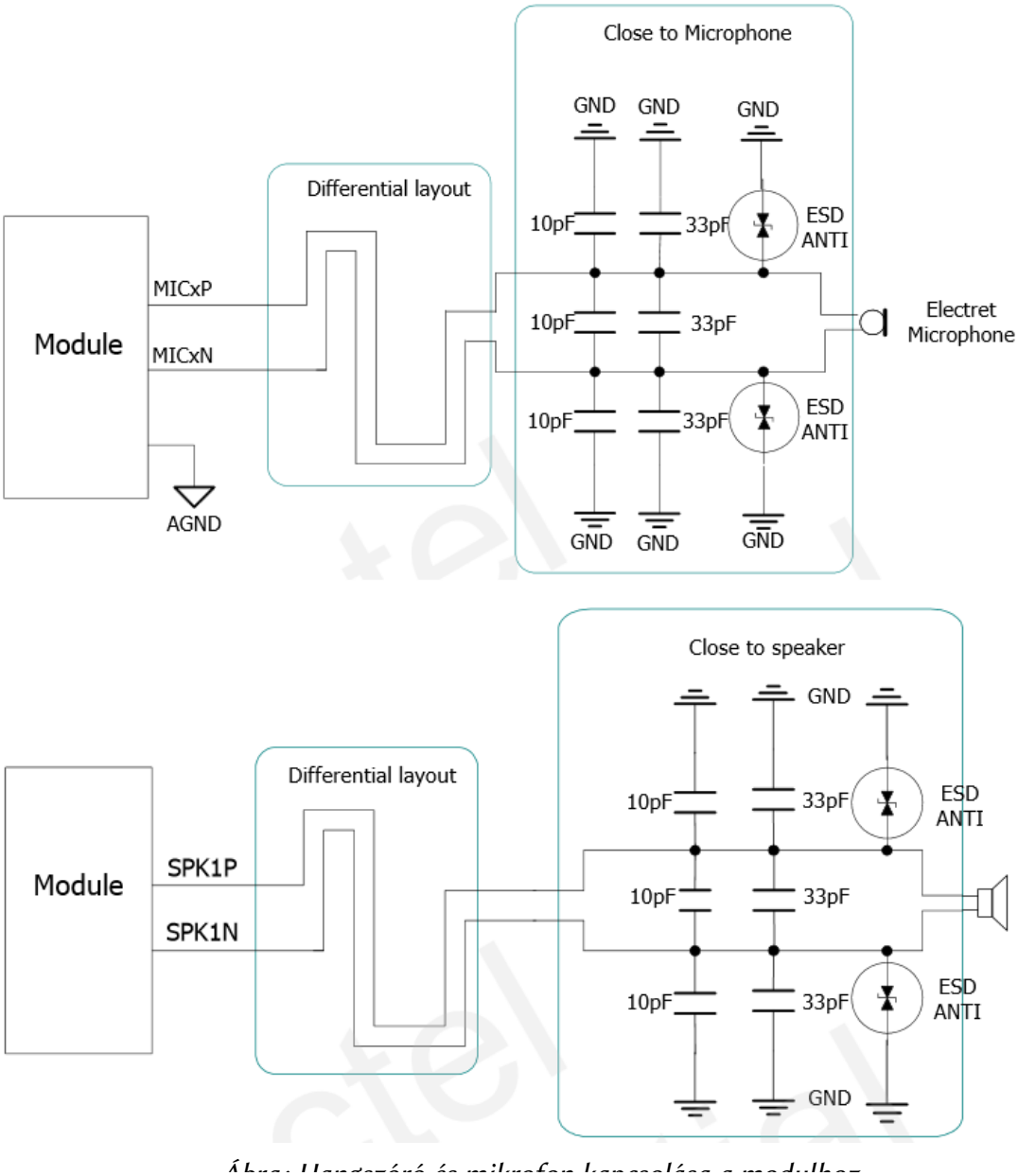
Amennyi USB csatlakozó szabvá odem fogyasztása elérheti a 800mA-t is, amely azt eredményezi, hogy a számítógép bontani fogja a kapcsolatot az Arduino-val, így igen instabillá válhat a működés.

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.



A modulon található modem dokumentációja alapján a mikrofon és a hangszóró helyes bekötése az alábbi ábrán látható.

261.



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.*

115. 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*		

116. 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.

17. 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

117. 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.

18. Megszakításkezelő bővítése

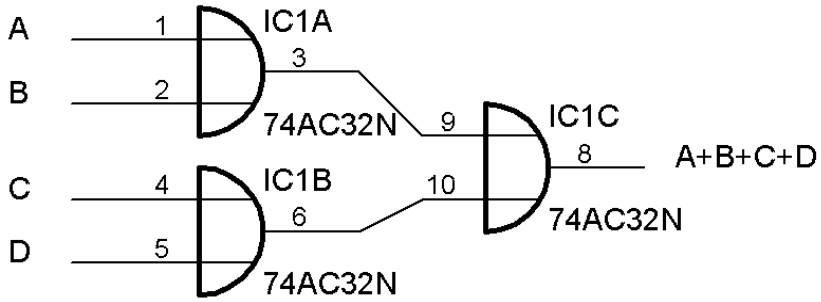
Uno esetén csupán két láb alkalmas megszakítások fogadására. Némi kiegészítő elektronikával azonban egy megszakítás fogadásra képes láb könnyen felbővíthető 4 vagy több megszakítás fogadására. A projektben használt elektronikák egy megszakítást bővítenek négyre.

Az elektronika megtervezése kétféle is lehet. Az egyszerűbb áramkör öt bemeneti jelet állít elő, így a programunk több egyidejűleg több megszakítás érzékelésére is képes.

Amennyiben nincs elég bemeneti lábunk, akkor kiegészítő elektronikával a bemeneti jelek száma csökkenthető négyre, de ebben az esetben, nem lesz képes érzékelni a program több megszakítás egyidejű érkezését.

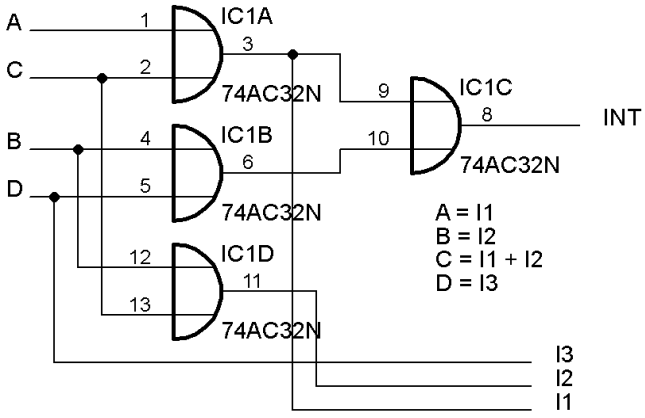
Mindkét elektronika esetén a megszakításra képes lábra egy vagy kapun keresztül érkeznek a megszakítást kérő jelek. Ebben az esetben, ha bármelyik megszakítást kér, akkor megszakítás állapotba kerül a programunk. A négy bemenetes vagy kapu könnyen kialakítható egy 7432-es IC segítségével, mivel ebben az áramkörben 4db kétbemenetes vagy kapu van.

262.



Az egyszerűbb megoldás a megszakítást kérő jeleket négy bemenetre vezeti. Amennyiben a programunk megszakítás állapotba kerül, akkor csak ki kell olvasnunk, hogy melyik bemenet váltott ki megszakítást, majd azokhoz a megszakításokhoz tartozó kódot kell lefuttatni.

Amennyiben nem lenne elég bemeneti lábunk, akkor egy kiegészítő áramkör segítségével a szükséges lábak száma csökkenthető egyel, azonban ezen megoldás alkalmazása esetén nem lehetséges több egyidejűleg érkező megszakítás érzékelése. A kiegészítő áramkör egy egyszerű kombinációs hálózat, melynek a rajza az alábbi ábrán látható.



A bonyolultabb áramkör is megvalósítható egy darab 7432-es áramkör segítségével. Ebben az esetben az integrált áramkör összes kapuja ki van használva. A megvalósítás gyenge pontja, hogy a bemeneti jeleket nem lehet felcserélni. Az INT kimenet egyes értéket vesz fel, ha a négy bemenet közül valamelyiken is megszakítás érkezik. A

megszakítás címének dekódolására az I1, I2 és I3 lábak használhatóak. A megszakítások dekódolási táblázata szintén az ábrán látható.

Az alábbi kód a bonyolultabb áramkör dekódoló szoftverét mutatja be. A használt megszakítási láb a 2-es, az I1, I2, I3 jelek a 3, 4, 5 lábakra vannak bekötve.

```
#define I1 3
#define I2 4
#define I3 5

void setup()
{

     attachInterrupt(0, megszakitas, RISING); //2-es lab
     pinMode(I1, INPUT);
     pinMode(I2, INPUT);
     pinMode(I3, INPUT);
}

void megszakitas()
{
     int state = 0;
     if (digitalRead(I1)) bitSet(state, 0);
     if (digitalRead(I2)) bitSet(state, 1);
     if (digitalRead(I1)) bitSet(state, 2);

     switch (state)
     {
          case 0: //nincs megszakítás
               return;
          case 1:
               //A bemeneten érkezett megszakítás
               break;
          case 2:
               //B bemeneten érkezett megszakítás
               break;
          case 3:
               //C bemeneten érkezett megszakítás
               break;
          case 4:
               //D bemeneten érkezett megszakítás
               break;
     }
}

void loop()
{
     //ismétlődő kód
}
```

19. ATmega644 és ATmega1284 alkalmazása

Az Arduino lapok hátránya, hogy az Uno után megjelent nagyobb kódmemóriával és tudással rendelkező modellek mind csak SMD tokozásban létező mikrovezérlőkre épülnek. Így az ezen modelleket alkalmazó áttervezett

lapok otthoni gyártása és összeszerelése szinte lehetetlen. Azonban van megoldás.

A megoldást az Atmega664 és ATmega1284 mikrovezérők jelentik. Ezek tudásban és kódmemória méretében félúton helyezkednek el a Mega és az Uno modell vezérlői között. Előnyük, hogy DIP tokozásban is elérhetőek.

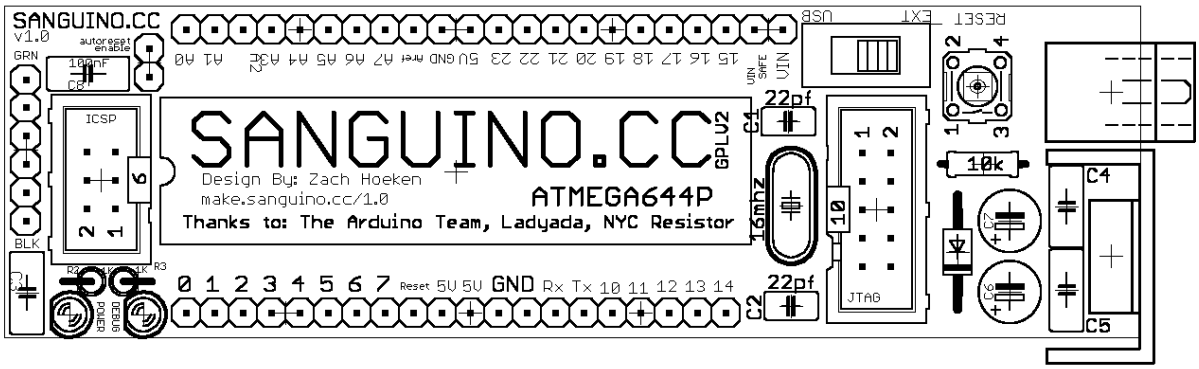
Hivatalosan nem támogatottak az Arduino környezet által, azonban könnyen hozzájuk adható a támogatás, mint az ATtiny vezérlők esetén.

Mindkét vezérlő DIP40 tokozásban érhető el, egymással lábkiosztás szempontjából kompatibilisek. A fő különbség a két modell között a kódmemória méretében van. A 644-es 64KB kódmemóriával rendelkezik, míg a 1284 128KB memóriával.

Mindkét modell programozható az Arduinio ISP programozó segítségével, vagy JTAG felületen keresztül is, mivel ezen vezérlők ezt is támogatják.

A mellékletek szekcióban megtalálható mindkét processzor részletes adat összehasonlítása, valamint a lábkiosztásuk is.

A 644-es modellre épül egy klón lap is, amely a Sangunio nevet kapta. Ezen klón megalkotóinak az volt a céljuk, hogy az Uno és a Mega modellek közötti hatalmas rést betöltsék egy olyan lap segítségével, amely könnyen otthon is megépíthető. A panel csak furat szerelt alkatrészekből áll, így otthon bárki által legyártható, utánaépíthető.



Egyetlen hátránya a modellnek, hogy lábkiosztásában és fizikai méreteiben nem követi a többi Arduino modell elrendezését, így a kiegészítő panelek használata nem lehetséges.

A modell kapcsolási rajza referenciaként elérhető, viszont a nyomtatott áramköri terv nem, így az otthoni gyártás előtt meg kell tervezni a panelt hozzá. A legfrissebb támogató szoftver a http://sanguino.cc/useit címről szerezhető be.

A szükséges szoftver a Mega1284 modell támogatásához a következő címről szerezhető be: http://maniacbug.wordpress.com/2011/11/27/arduino-on-atmega1284p-4/

20. Ultrahangos távolságmérés

Az ultrahangos távolságmérők a visszhang effektust használják ki. Olyan magas frekvenciás hanghullámokkal dolgoznak, amit az emberi fül nem hall. A kiadott hang és visszaverődésként érkező hanghullám közötti időeltérés és a hang sebességének ismeretében kiszámolható, mérhető a szenzor előtt elhelyezett objektum távolsága a szenzortól.

Ilyen szenzorok igen olcsón beszerezhetőek, két fő változatban kaphatóak. Három és négy vezetékes változatokban. A két modell között annyi különbség van, hogy a három kivezetéses modellek esetén a hang bemenet és visszhang kimenet ugyan arra a lábra van kötve, így mérés közben váltogatni kell a láb funkcióját.

Ebből adódóan a négy vezetékes változatú szenzorok némiképpen pontosabbak és gyorsabb működésre képesek. Ezek kezeléséhez külön osztálykönyvtár is elérhető. Ez az osztálykönyvtár a NewPing nevet viseli, a https://code.google.com/p/arduino-new-ping/ címről szerezhető be. A könyvtár használatának hátulütője, hogy a hang generálásra képes függvények használatát lehetetlenné teszi. A probléma orvosolható egy kompatibilis hang generáló könyvtár használatával, mint a toneAC, ami a https://code.google.com/p/arduino-tone-ac/ címről szerezhető be.

Az alábbi kód könyvtár nélkül mutatja be a szenzor kezelését egy három vezetékes szenzor esetén. Amennyiben négy vezetékkel rendelkezik a szenzorunk, akkor a két jel vezetéket kell összekötni.

```
#define PINGPIN 7;

long int ping(int pingPin)
{
     pinMode(pingPin, OUTPUT);
     digitalWrite(pingPin, LOW);
     delayMicroseconds(2);
     digitalWrite(pingPin, HIGH);
     delayMicroseconds(5);
     digitalWrite(pingPin, LOW);
     pinMode(pingPin, INPUT);
     return pulseIn(pingPin, HIGH);
}


float centimeter(long int microseconds)
{
     return microseconds / 29.0 / 2.0;
}

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

void loop()
{
     long int ido = ping(PINGPIN);
     Serial.print("Tavolsag: ");
     Serial.print((int)centimeter(ido));
```

```
      Serial.println(" cm");
      delay(100);
}
```

A ping függvény egy négyszögjelet ad ki a meghatározott lábon, majd a jel kiadása után a lábat átváltja bemenetre. Ezután vár a visszaverődés megérkezésére. A várakozási idő megadja a tárgy távolságát.

A távolság konverzió a centimeter nevű függvényben van megoldva. A hang terjedési sebességéből kiszámolható, hogy 1 cm utat 29 mikroszekundum alatt tesz meg, tehát a mikroszekundumban mért időt el kell osztani 29-cel, hogy megkapjuk a tárgytól mért távolság kétszeresét. Azért a kétszeresét kapjuk a távolságnak, mert a hanghullámnak a tárgyhoz meg kell érkeznie, majd vissza is kell érkeznie, így a távolságot kétszer teszi meg. Ezért szerepel még a függvény végén egy kettővel osztás.

Ezen kezelőfüggvény hátránya, hogy lassú, mivel a pulseIn függvényt használja, ami akár 1 másodpercig is futhat. Továbbá két távolságmérés között legalább 50ms várakozási időt be kell iktatni, mivel kisebb várakozási idők mellett áthallás következhet be a két mérés között.

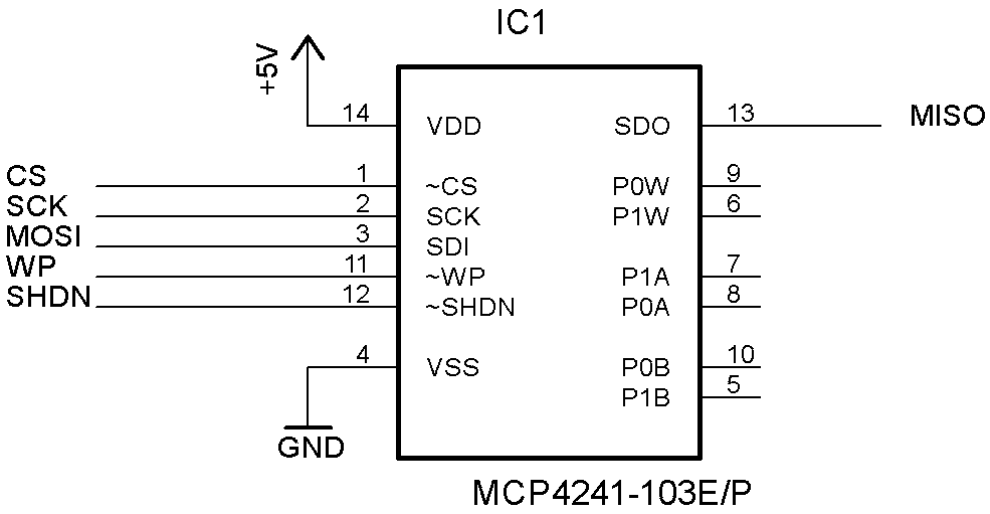
Továbbá szenzor függően érdemes lehet több mérést is végezni, majd a méréseket átlagolni a pontosabb eredményhez.

21. Digitális potméterek kezelése

A digitális potméterek olyan integrált áramkörök, amelyek egy digitális érték hatására változtatják ellenállásukat. Rengeteg -féle kivitelben kaphatóak. SPI vagy I²C buszrendszerre csatlakoztathatóak. Általában 8 bit pontossággal szabályozhatóak, ami 255 fokozatot tesz lehetővé. Ennek ellenére nem tekinthetőek ezen áramkörök precíziós megoldásnak, mivel a legtöbb áramkör 10-20%-os tűréssel rendelkezik. Továbbá teljesítmény ellenállásnak sem nevezhetőek, mivel tipikusan pár miliamper áram átfolyását tolerálják, így leginkább feszültség osztóként használhatóak.

Ezen eszközök kezelése típustól függetlenül igen könnyű, mivel úgy viselkednek, mint egy EEPROM memória. Megadott memória címre beírva az IC megadott lábai között mérhető egy ellenállás. Egyes modellek rendelkeznek olyan memória területtel is, amely kikapcsolás esetén is megőrzi a potméter állást.

A jelenlegi projektben egy MCP4251-103E/P típusú digitális potméter kezelését mutatom be. Ez egy SPI buszrendszerre kapcsolódó potméter 8 bites felbontással, DIP14 tokozásban két darab potmétert is tartalmaz. A potenciométerek névleges ellenállása 10K Ohm.



Az SPI vezérlőjeleken kívül a chip rendelkezik speciális vezérlőjelekkel. A WP láb földelése az eszközben található EEPROM memóriát teszi írásvédetté, míg az SHDN láb földelése a potmétert kikapcsolja.

A két potméter P0 és P1 jelöléssel szerepel. A névleges ellenállások az A és B jelölések között mérhetőek, a középső kivezetés W jelöléssel szerepel.

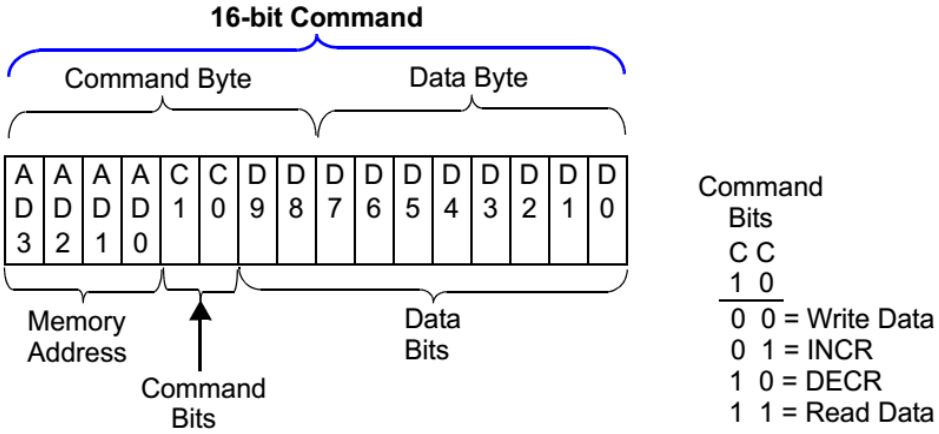
A kezelés abból áll, hogy megfelelő címekre értékeket írunk be. A címek a memóriacím térképről olvashatóak ki. A chip 16x9 bit memóriaként kezelhető, általános memóriacímeket is tartalmaz, amelyek alkalmasak adat tárolására.

Address	Function	Memory Type
00h	Volatile Wiper 0	RAM
01h	Volatile Wiper 1	RAM
02h	Non-Volatile Wiper 0	EEPROM
03h	Non-Volatile Wiper 1	EEPROM
04h	Volatile TCON Register	RAM
05h	Status Register	RAM
06h	Data EEPROM	EEPROM
07h	Data EEPROM	EEPROM
08h	Data EEPROM	EEPROM
09h	Data EEPROM	EEPROM
0Ah	Data EEPROM	EEPROM
0Bh	Data EEPROM	EEPROM
0Ch	Data EEPROM	EEPROM
0Dh	Data EEPROM	EEPROM
0Eh	Data EEPROM	EEPROM
0Fh	Data EEPROM	EEPROM

A 02h és 03h memóriacímek a potméter bekapcsolás után felveendő ellenállás értékét tárolják, vagyis a potméter jelenlegi pozíciójának mentésére szolgálnak. A 04h címen található TCON regiszter a kivezetések állapotát kontrollálja. A 05h címen található Státusz regiszter a chip jelenlegi állapotáról szolgáltat információt. Ezen két regiszter tartalmának olvasása/írása nem szükséges az alapvető kezeléshez.

A megfelelő memóriacímre írt érték alapján a B és W kivezetés között mérhető ellenállás az alábbi képlet segítségével határozható meg: $R_{WB}=\frac{R_{AB}\cdot n}{256}+R_W$ *A képletben n egy 0 és 255 közötti digitális értéket jelöl.* R_W *pedig a potméter középső kivezetésének ellenállása, amely jelen chip esetén 70 Ohm.*

A Chip 16 bites parancsokkal vezérelhető. A parancsok felépítését az alábbi ábra szemlélteti.



Az alábbi kód a potméterek egyszerű kezelését mutatja be, jelen példában a potméter regisztereiből olvasás nem történik.

```
#include <SPI.h>
#define CS 2

void setup()
{
     pinMode(CS, OUTPUT);
     digitalWrite(CS, HIGH);
     SPI.begin();
```

```
      SPI.setDataMode(SPI_MODE0);
      SPI.setBitOrder(LSBFIRST);
}

void WritePot(byte potadress, byte value)
{
      digitalWrite(CS, LOW);
      if (potadress == 0)
      {
            SPI.transfer(value);
            SPI.transfer(0x00);
      }
      else
      {
            SPI.transfer(value);
            SPI.transfer(0x10);
      }
      digitalWrite(CS, HIGH);
}

void loop()
{
      int i=0;
      for (i=0; i<256; i++)
      {
            WritePot(0, i);
            delay(500);
      }
      for (i=0; i<256; i++)
      {
            WritePot(1, i);
            delay(500);
      }
}
```

A setup függvény beállítja a CS lábat, ami jelen példában az Arduino kettes lába, majd magas állapotba kapcsolja, mivel ezen potméter chip esetén ez alacsony szinten aktív. Ezután az SPI buszrendszert inicializálja a függvény, MODE0 üzemmódban, ami felfutó él reagálással állítja be az órajelet, valamint a legkisebb helyi értéken lévő bittel kezdi az átvitelt.

A WritePot függvény a potméter írásáért felelős. Első paramétere a potméter címét határozza meg, ami 0 vagy egy lehet. Második paramétere pedig a potméter értékét határozza meg. A loop függvény pedig a két potméter értékét pörgeti végig fél másodperces késleltetéssel.

22. Energiatakarékosság

Az Arduino lapokon használt Atmega processzorok többsége támogat energiatakarékossági üzemmódokat, amelyek segítségével csökkenthető a vezérlő energia fogyasztása.

Ez hálózati feszültségi betáplálásnál nem feltétlenül hasznos, vagy szükséges, viszont akkumulátorról működő projekteknél kifejezetten hasznos.

Több energiatakarékossági mód is definiált, amellyel bizonyos részei kikapcsolhatóak a mikrovezérlőnek. Az energiatakarékossági üzemmódok vezérlők vezérlő típusonként egyediek. Használatukhoz hivatalos kezelő könyvtár nincs, a http://playground.arduino.cc//Code/Enerlib címen található könyvtár Arduino Uno-t támogat és feltehetően Arduno Megat-is, mivel a két vezérlő processzora azonos.

A könyvtár az alábbi függvényeket biztosítja az Energy osztály példányosításán keresztül. Az osztály konstruktora paraméter nélküli.

```
energy.Idle();
```

Idle üzemmódba kapcsolja a mikrovezérlőt, ami a belső CPU-t és FLASH memóriát kapcsolja ki, minden más eszköz bekapcsolva marad, így ebből az állapotból a vezérlő külső megszakítás, vagy soros kommunikációs esemény hatására is fel tud ébredni.

```
energy.SleepADC();
```

Hasonló az Idle üzemmódhoz, csak az IO egységeket is lekapcsolja. A vezérlő ebből az állapotból külső vagy időzítő alapú megszakítás segítségével tud kilépni. Ezen üzemmód alkalmazható nagyon pontos ADC érték olvasásra, mivel a belső I/O órajel nem zavarja az olvasást.

```
energy.PowerDown();
```

A külső órajelet állítja meg, így a teljes vezérlő kikapcsol, csak a Watch dog timer és a külső megszakítások kezelése marad aktív. Ebből az állapotból a vezérlő külső megszakítás, külső reset vagy Watch dog timer reset jel segítségével hozható ki.

```
energy.PowerSave();
```

Hatása megegyezik a PowerDown üzemmóddal, azzal a különbséggel, hogy ebben az üzemmódban a Chip kettes azonosítójú számlálója tovább fut, amennyiben a megszakítással rendelkezik. Ebből az üzemmódból külső megszakítás vagy időzítő alapú megszakítás segítségével tud kikerülni a vezérlő. Arduino környezet esetén a chip-ben található összes időzítő használt valamilyen célra.

```
energy.Standby();
```

Hatása megegyezik a PowerDown üzemmóddal, csak a külső órajel forrás használatban marad. A mikrovezérlő ebből az állapotból visszatéréshez 6 órajel ciklust igényel.

```
energy.WasSleeping();
```

A függvény igaz állapotot ad vissza, ha a mikrovezérlő valamilyen alvó állapotban volt. Megszakításon belül használható ébresztési esemény tesztelésre.

Az alábbi minta kód a könyvtár használatát mutatja be:

```
#include <Enerlib.h>

Energy energy;

void Megszakitas(void)
{
     if (energy.WasSleeping())
```

```
	{
		//alvasbol visszatereskor elvegzendo dolgok
	}
	else
	{
		//nem alvo megszakitas volt
	}
}

void setup()
{
	//megszakitas csatolasa, LOW tipusunak kell lennie
	attachInterrupt(0, Megszakitas, LOW);
	pinMode(13, OUTPUT);
}

void loop()
{
	for (int i=0; i<10; i++)
	{
		digitalWrite(13, HIGH);
		delay(500);
		digitalWrite(13, LOW);
		delay(500);
	}
	energy.Standby();
}
```

A kód működése igencsak egyszerű. A 0. jelű megszakítás bemenetre definiájuk a Megszakitas függvényt, ami jelen esetben tényleges munkát nem végez, csupán a vezérlő felélesztésére szolgál alvó állapotból. Az ismétlődő kódrészletben a 13-as lábra kötött LED-et villogtatja 10 alkalommal a program, majd Standby állapotba kapcsol. Ebben az állapotban a gomb megnyomása hatására újraindul az ismétlődő kódrészlet, majd 10 villogás után ismét alvó állapotba vált a vezérlő. A megszakításnál használt gomb ebben az esetben is pergés mentesítendő.

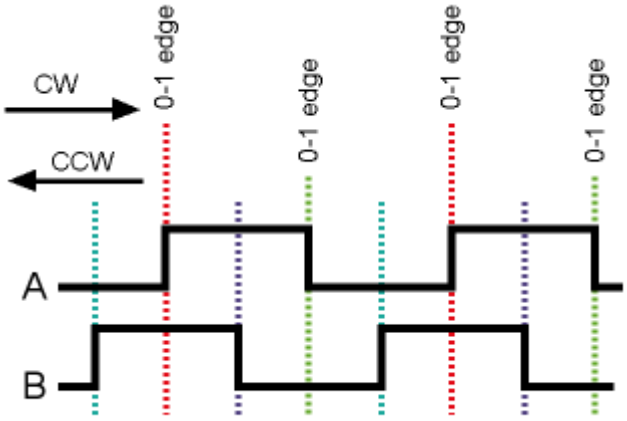
23. Forgó jeladó kezelése

A forgó jeladót köznapi nyelvben szokás végtelenített potenciométernek nevezni, holott ténylegesen köze nincs a potenciométerekhez. Az elnevezés talán abból adódik, hogy ilyen megoldásokat általában autó rádiókban szoktak alkalmazni a hangerő szabályzására.

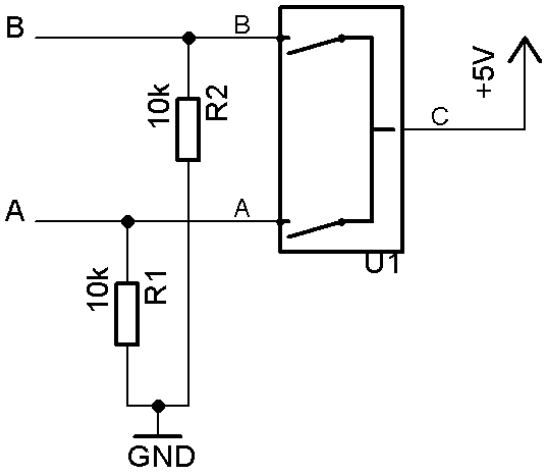
Ezen jeladók igen hasznosak tudnak lenni, kezelésük sem olyan bonyolult. A kezelés kapcsán két megközelítés is alkalmazható. Megszakításos kezelés, vagy megszakítás nélküli. A megszakítás nélküli kezelés előnye, hogy egyszerű a program, hátránya pedig az, ha gyorsan változik a jeladó állapota akkor történhet kihagyás. Megszakításos kezelés esetén nem történhet kihagyás, de cserébe pergésmentesíteni kell a bemenetet, valamint egy megszakítást felhasznál a jeladó kezelése, ami problémás, mivel az Uno modell csak 2 külső megszakítással rendelkezik.

A forgó jeladók általában három kimenettel rendelkeznek. Egy közös ponttal és egy A, valamint B jelű kimenettel. Az A és B kimenetek jelváltozásának irányából lehet megállapítani, hogy a jeladó merre forog. Az alábbi ábra a jeladó kimenetein észlelhető jelformákat mutatja be forgás irány függően.

269. Ábra:



Amennyiben megszakítás nélkül szeretnénk használni a jeladót, az alábbi kapcsolás alkalmazandó a jeladó bekötésénél:



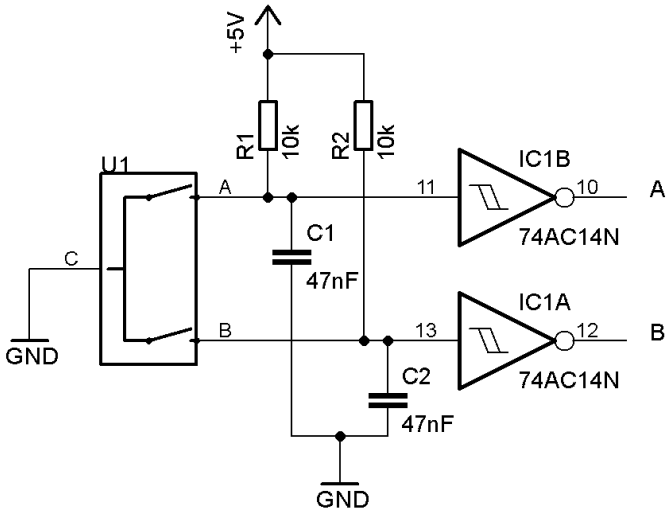
Megszakítások nélküli kezelése az alábbi kód alkalmazható:

```
#define encoder0PinA 3
#define encoder0PinB 4
int val;
int n = LOW;
```

```
int k = LOW;

void setup()
{
     pinMode (encoder0PinA,INPUT);
     pinMode (encoder0PinB,INPUT);
     Serial.begin (9600);
}

void loop()
{
     n = digitalRead(encoder0PinA);
     k = digitalRead(encoder0PinB);

     if (n == LOW && k == HIGH)
     {
          val++;
          while (digitalRead(encoder0PinA) != →
digitalRead(encoder0PinB)) {}
          Serial.println(val);
     }

     if (n == HIGH && k == LOW)
     {
          val--;
          while (digitalRead(encoder0PinA) != →
digitalRead(encoder0PinB)) {}
          Serial.println(val);
     }
}
```

A kód működése igen egyszerű, a bemenetként definiált lábak értékeit hasonlítja össze. Amennyiben az A bemenet alacsony szinten van, a B bemenet pedig magas szinten, akkor a jeladó egyik irányba forog, ha ezen állápot ellentettje áll fent, akkor pedig másik irányba forog a jeladó. A változásnak megfelelően egy számláló értéke módosul, amely aztán soros porton továbbítva van a számítógép felé. Amennyiben kellően sokáig és gyorsan tekerjük a kódolót, akkor előfordulhat, hogy lesz olyan állapot, hogy a számláló értéke nem változik, kihagy.

Ezen kellemetlen hiba elkerülhető, ha megszakításokat alkalmazunk a kezelésre. Ehhez először azonban pergésmentesíteni kell a kódoló kimenetét. Egy pergésmentesítő áramkör kapcsolása az alábbi ábrán látható. A működési elve azonos a gombok esetén alkalmazott pergésmentesítéssel, csak itt kisebb kondenzátorok alkalmazása javasolt, mivel ha túl nagy értéket választunk, akkor a jeladó sebessége nem lesz gyors.



A megszakításokkal megvalósított kezelést az alábbi kódrészlet mutatja be:

```
#define pinA 2
#define pinB 3
int iValue = 0;

void encoderClick()
{
     int valA = digitalRead(pinA);
     int valB = digitalRead(pinB);

     if (valA == LOW && valB == HIGH) iValue++;
     else if (valA == HIGH && valB == LOW) iValue--;
}

void setup()
{
     Serial.begin(9600);
     pinMode(pinA, INPUT);
     pinMode(pinB, INPUT);
     attachInterrupt(0, encoderClick, CHANGE);
     attachInterrupt(1, encoderClick, CHANGE);
}

void loop()
{
     Serial.println(iValue);
     delay(100);
}
```

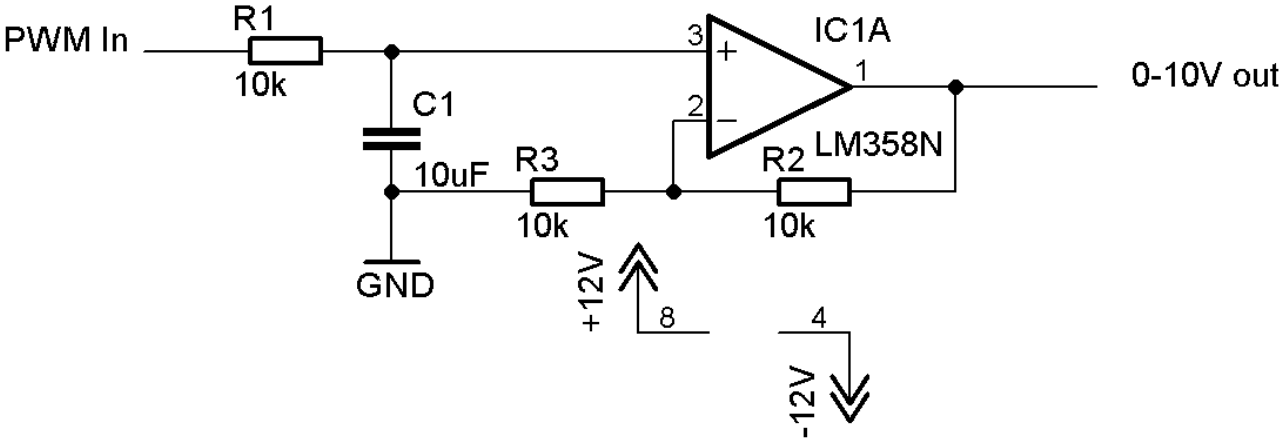
A kód alapelve ugyan az, mint a megszakítás nélküli változatban. Megszakítás esetén összehasonlításra kerülnek a jelek, ami alapján a számláló értéke módosulni fog. A számláló értékét 100ms időközzel közli a vezérlő a számítógéppel.

24. Feszültség előállítása PWM modulációval

A mikrovezérlők PWM kimenete alkalmazható Digitális-Analóg átalakítási célokra is. Ehhez a PWM kimenetre tenni kell egy alul áteresztő szűrőt, amit ha igen kicsi frekvenciára méretezünk, akkor a gyorsan változó impulzusok effektív értéke egyen feszültségként jelenik meg a szűrő kimenetén. A kicsi frekvenciára méretezés hátránya, hogy a kimenet nem lesz olyan gyorsan változtatható köszönhetően a szűrőben alkalmazott viszonylag nagy méretű kapacitás miatt, viszont ebben az esetben kevesebb zaj lesz a kimeneten, mint ha egy nagyobb frekvenciára méretezett szűrőt alkalmaznánk. Ezért az ilyen konverziók esetén egyensúlyt kell találni a sebesség és a kimeneti zaj között.

Jelen projektben egy 0-10V szelep vezérlését valósítjuk meg. A 0-10V vezérlőjelek ipari alkalmazásban szabványosak. Ez egy analóg jel, százalék információ átvitelére. Egy szelep esetén, ha a vezérlő jel 0 Volt, akkor a szelep egyáltalán nem nyit ki, viszont ha 3 Voltot a vezérlő jel, akkor 30%-ban fog kinyitni. 100%-os kinyitást 10V jel mellett fog produkálni. Ezen jel csak a vezérlést szolgáltatja, a használt eszközünk saját tápellátást igényel, így a vezérlő jel előállítására szolgáló elektronikának nem kell nagy teljesítményűnek lennie.

A 10 Voltos maximális jelet az 5 Voltos bemenő maximális jelből egy kétszeres erősítési tényezőjű erősítővel könnyen előállíthatjuk, amit megépíthetünk tranzisztorokból is akár, de egyszerűbb, ha egy műveleti erősítőt alkalmazunk erre a célra nem invertáló alapkapcsolásban.



A fenti kapcsolásban R1 és C1 ellenállás egy ~15 Hz-re méretezett alul áteresztő szűrőt valósít meg, amely kétszeresen erősítve van az LM358-as műveleti erősítő által. Ezen típusra a választás sokoldalúsága miatt esett, valamint azért, mert a kimenete rövidzárlat védett, amely ipari alkalmazások során nem elhanyagolható.

Amennyiben 3,3V-os mikrovezérlővel (Pl. Due) kívánjuk alkalmazni a fenti kapcsolást, akkor R3 és R2 értékei 33K és 68K Ohmra módosítandóak.

A legtöbb 0-10V vezérlőjelet fogadó eszköz esetén a 15Hz-es szűrés nem jelent problémát, mivel viszonylag lassú reagálásúak ezen eszközök. A legtöbb szelep 100%-os vezérlő jel mellett 30-35Mp idő alatt nyit ki. Az alábbi mintakód az átalakító használatát mutatja be

```
void setup()
{
     pinMode(3, OUTPUT);
}

void PercentToPWM(int percent)
{
     byte val = (percent * 255) / 100;
```

```
      analogWrite(3, val);
}

void loop()
{
      int percent = 0;
      int i=0;
      for (i=0; i<20; i++)
      {
            percent += 5;
            PercentToPWM(3, percent);
            delay(1000);
      }

      delay(3000);

      for (i=0; i<20; i++)
      {
            percent -= 5;
            PercentToPWM(3, percent);
            delay(1000);
      }

      delay(3000);

}
```

A példaprogramban az Uno 3-as lába használt, a PercentToPWM függvény egy százalék értéket számít át PWM jelértékké, majd ki is küldi azt. Az ismétlődő loop függvényben a kód 20 másodperc alatt teljesen kinyitja a szelepet, majd három másodperc várakozás után ismét 20 másodperc alatt lezárja a szelepet.

25. Egy mindenes függvénykönyvtár: UtilLib

Az Arduino környezet beépített függvényei igen sok mindent tudnak, azonban vannak olyan algoritmusok, függvények, amelyek nincsenek megírva, és néha igen hasznos lenne, ha készen adottak lennének és nem kellene a kereket minden egyes programunk alkalmával újra feltalálni. Ezért alkottam meg az UtilLib névre keresztelt osztálykönyvtáramat. A könyvtár forrása a https://github.com/webmaster442/ArduinoExtensions/tree/master/libraries/UtilLib címről tölthető le.

A függvénykönyvtár függvényei:

```
byte BcdDecode(byte value);
```

Egy pakolt formátumú BCD számot konvertál bináris számmá

```
byte BcdEncode(byte value)
```

Egy 8 bites, 0 és 99 közötti bináris számot konvertál pakolt formátumú BCD számmá.

```
Byte HighByte(int value);
byte HighByte(unsigned int value);
```

Egy 16 bites egész szám felső byte-ját adja vissza. A szám lehet előjeles vagy előjel nélküli is.

```
byte LowByte(int value);
byte LowByte(unsigned int value);
```

Egy 16 bites egész szám alsó byte-ját adja vissza. A szám lehet előjeles vagy előjel nélküli is.

```
void AanlogWriteMilivolts(int ch, int milivolts);
```

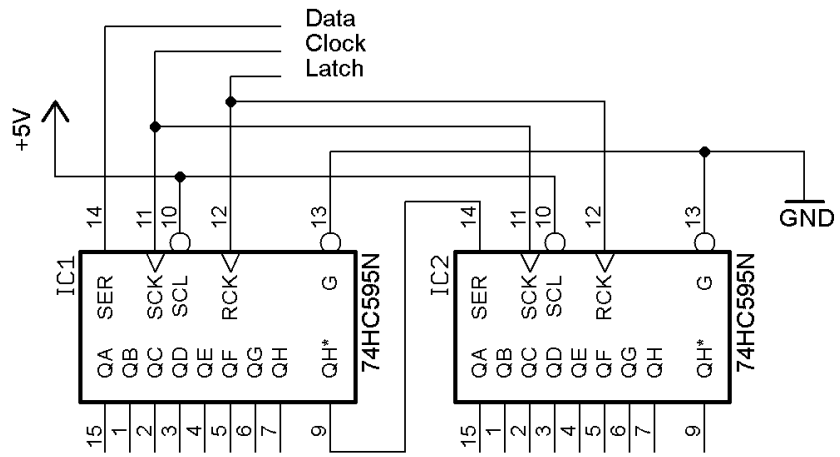
Millivoltban meghatározott effektív értékű PWM jel írása egy PWM képes kimenetre. Első pataméter a kimenetet határozza meg, a második paraméter pedig a jel effektív értékét határozza meg, ami maximum 5000 mV (5V) lehet.

```
byte Map(byte x, byte in_min, byte in_max, byte out_min, byte
out_max)

int Map(int x, int in_min, int in_max, int out_min, int out_max)

long int Map(long int x, long int in_min, long int in_max, long
int out_min, long int out_max);
```

Működése megegyezik a beépített map függvénnyel, viszont ezen implementációk gyorsabbak és a típusokat jobban kezelik.

```
void ShiftOut(byte dataPin, byte clockPin, byte bitOrder, byte
val);

void ShiftOut(byte dataPin, byte clockPin, byte bitOrder, int
val);

void ShiftOut(byte dataPin, byte clockPin, byte bitOrder, long int
val);
```

Shift regiszter számára adat írása. A beépített gyári függvénytől annyiban tér el, hogy lehetővé teszi 16 vagy 32 bites regiszterek kezelését is. 16 vagy 32 bites regiszter összerakható több 8 bites regiszterből, mint a 74595. Ilyen kapcsolásra az alábbi ábrán látható egy példa.



A függvény első paramétere az adat láb, második paraméter az órajel láb, harmadik paramétere pedig a bitsorrend. Ez két konstans közül kerülhet ki. MSBFIRST vagy LSBFIRST. MSBFIRST hatására a 74595 regiszter Q0 kimenetére kerül a legnagyobb helyiértékű bit, míg LSBFIRST esetén a Q7 kimenet lesz a legnagyobb helyértékű. A negyedik paraméter a küldendő adatot határozza meg.

```
byte SevenSegmentNumber(byte number, byte common);
```

Hétszegmenses kijelző számára dekódol egy bináris számjegyet. Az első paramétere a számjegy, második paramétere pedig azt határozza meg, hogy a kijelző közös anódos vagy közös katódos. Értéke két konstans közül kerülhet ki. CA (közös anód) vagy CC (közös katód) lehet.

```
unsigned int UnusedRAM()
```

Visszatérési értéke a szabad RAM memória byte-ban.

```
int GetVccMiliVolts()
```

Visszaadja millivoltban a jelenlegi tápfeszültséget.

26. Sebesség növelése, optimalizálás

Az Arduino környezet függvényeinek alkalmazása a kimenetek kezelésére több mint kényelmes. De ennek a kényelemnek komoly ára van, amit a sebesség fizet meg. Sajnos ezen függvények alkalmazása fájdalmasan lassú a regiszter szintű módosításhoz képest. A regiszter szintű módosításhoz képest nagyjából 35x lassabb a digitalRead és digitalWrite használata, ami a legtöbb esetben nem okoz problémát, viszont vannak olyan alkalmazási területek, ahol igen is sokat jelent ez a plusz sebesség.

Arduino környezet alatt is lehetőségünk van hozzáférni a kimeneti portok regisztereihez. Az AVR architektúra működése némiképpen eltérő.

Minden porthoz három regiszter tartozik. A portok jelölése itt is szintén az ABC betűivel történik, így a B jelű porthoz tartozó három regiszter: PORTB, DDRB, PINB

A PORT regiszterek tárolóként funkcionálnak. Amit beleírunk, az a kimeneten is megjelenik. Viszont ha kiolvassuk a regisztert, akkor nem feltétlen a bemenetek aktuális állapotát kapjuk vissza, hanem az utolsó szoftveres módosítását, ami vagy egybe esik a bemenetek állapotával vagy nem.

A bemenetek aktuális állapotának lekérdezésére szolgálnak a PIN regiszterek. Ezek csak olvashatóak és minden esetben a bemenetek aktuális állapotát tükrözik, viszonylag azonban csak lassan olvashatóak.

A DDR jelű regiszterek állítják be portok irányát. A 0 bit jelentése itt bemenetet jelöl, az egyes bit érték jelöli a kimenetet.

Az 35x sebesség különbség abból adódik, hogy az Arduino függvényeknek először is be kell azonosítaniuk a kezelt lábhoz tartozó regisztert. Ezután olvasási kérelem esetén ki kell olvasnia a megfelelő PORT és PIN regisztereket, majd ebből el kell döntenie, hogy melyik regiszter tartalmát adja vissza. Kimenet kezelés esetén pedig meg kell nézni a PORT és PINC regisztereket, hogy kell e egyáltalán módosítani valamit.

A regiszterek kezelése ennél természetesen egy jobb megoldás, de ehhez tudni kell, hogy melyik láb melyik port melyik bitje a regiszterekben. Ez mikroszervezetként más és más. A legegyszerűbben az Arduino kapcsolási rajzok tanulmányozásából hámozható ki, de az egyszerűség kedvéért Uno, Leonardo és Mega modellek esetén kigyűjtöttem táblázatba is.

Uno				Leonardo					
Bit	**B**	**C**	**D**	**Bit**	**B**	**C**	**D**	**E**	**F**
0	8	A0/14	0	0	RX LED	x	3	x	A5
1	9	A1/15	1	1	SCK	x	2	x	A4
2	10	A2/16	2	2	MOSI	x	0	x	x
3	11/MOSI	A3/17	3	3	MISO	x	1	x	x
4	12/MISO	A4/18/SDA	4	4	8	x	4/A6	x	A3
5	13/SCK	A5/19/SCL	5	5	9	x	TX LED	x	A2
6	x	x	6	6	10	5	12	7	A1
7	x	x	7	7	11	13	6/A7	x	A0

118. Táblázat: Uno és Leonardo modellek regiszter bitkiosztása

Mega											
bit	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**J**	**K**	**L**
0	22	53/SS	37	21/SCL	0	A0	41	17	15	A8	49
1	23	52/SCK	36	20/SDA	1	A1	40	16	14	A9	48
2	24	51/MOSI	35	19	x	A2	39	x	x	A10	47
3	25	50/MISO	34	18	5	A3	x	6	x	A11	46
4	26	10	33	x	2	A4	x	7	x	A12	45

5	27	11	32	x	x	A5	4	8	x	A13	44
6	28	12	31	x	x	A6	x	9	x	A14	43
7	29	13	30	38	x	A7	x	x	x	A15	42

119. Táblázat: Mega modell regiszter bitkiosztása

A kényelem és gyorsaság egyensúlyban tartása végett született meg a digitalWriteFast osztálykönyvtár, ami a ki és bemenetek gyors kezeléséhez biztosít függvényeket. Ezt úgy oldja meg, hogy a ki és bemenet kezelő függvények makróként vannak definiálva minden egyes mikrovezérlő esetén, így lényegében a függvény hívások a fordításkor regiszter módosító utasításokra lesznek helyben cserélve. A kód mérete ennek következtében nőni fog némiképpen.

A könyvtár a https://code.google.com/p/digitalwritefast/ címről szerezhető be. A kódot módosítani kell némileg, hogy Arduino 1.0 feletti környetekkel használható legyen. A fejléc fájl elején az #include "Wprogram.h" sort kell cserélni #include "Arduino.h"-ra.

A könyvtár az alábbi függvényeket biztosítja, melyek használata megegyezik a Fast prefixum nélküli Arduino függvényekkel.

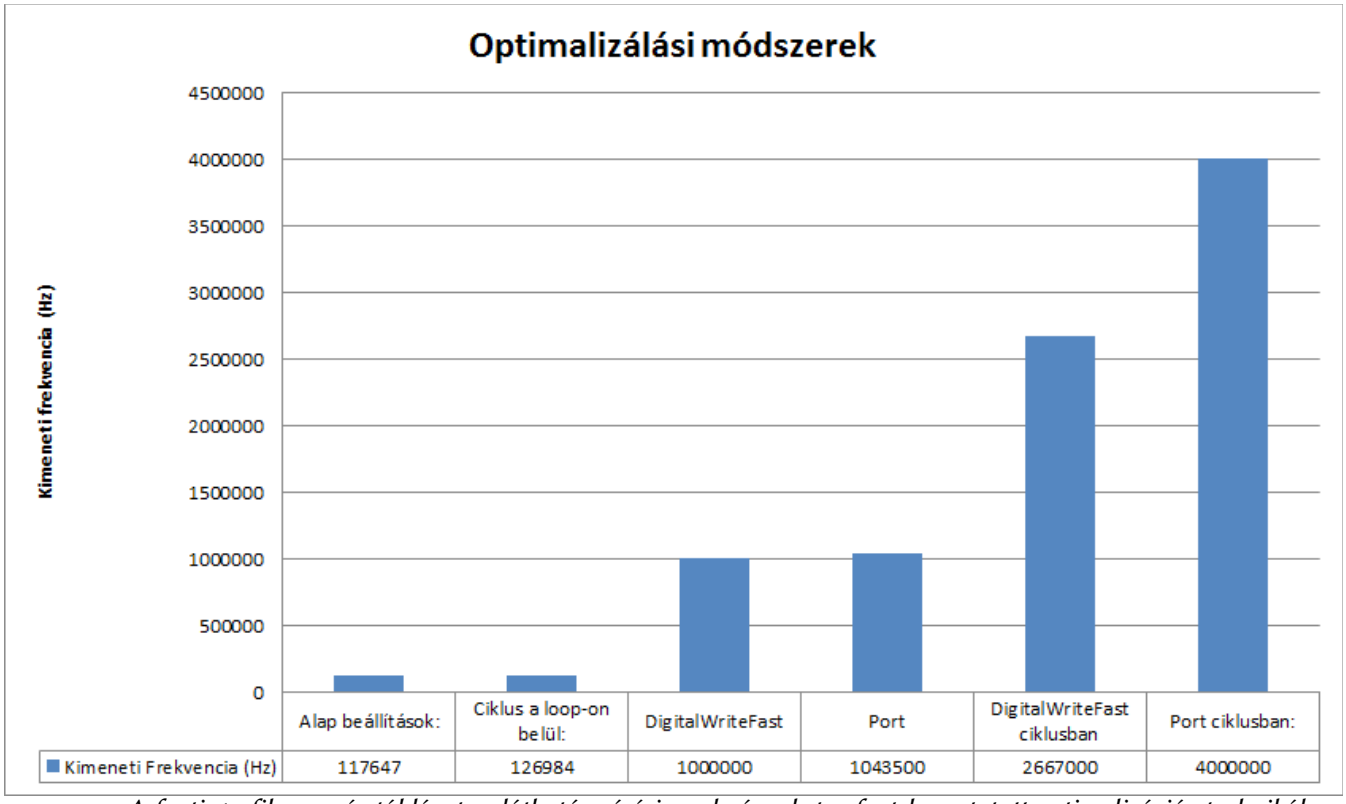
```
pinModeFast(pin, mode);
digitalReadFast(pin);
ditialWriteFast(pin, value);
```

Egy kis sebesség javulás érhető el a loop függvény átdolgozásával is. A probléma az, hogy a loop egy függvény, ami egy végtelen cikluson belül van folyamatosan megívva, így minden egyes alkalommal, mikor újraindul egy függvényhívás keletkezik feleslegesen, mivel a loop függvény másra nem használt.

A probléma kikerülhető úgy, hogy a loop függvényen belül létrehozunk egy végtelen ciklust, ebből a program nem fog tudni kilépni, így nem lesznek felesleges hívások. A soros kommunikáció működéséhez azonban plusz egy sort be kell iktatni a végtelen ciklusba. A plusz egy sort tartalmazó optimalizált kód:

```
void loop()
{
      while (1)
      {
            //kód ide
            if (serialEventRun) serialEventRun();
      }
}
```

Mérési eredmények



A fenti grafikonon és táblázaton látható mérési eredményeket a fent bemutatott optimalizációs technikák segítségével értem el. A méréshez egy Arduino Uno-t használtam. A 7-es kimeneten egy logikai analizátor segítségével mértem a kimenet változásának a sebességét.

A Port módosítás végtelen ciklusban elérte az eszköz maximális sebességét. Ennél gyorsabb kimenet változás nem érhető el, mivel a 16Mhz-es órajelet negyedeli a vezérlő processzora abból adódóan , hogy egy utasítás feldolgozása 4 órajel ciklust igényel.

27. Grafikus kijelzők kezelése

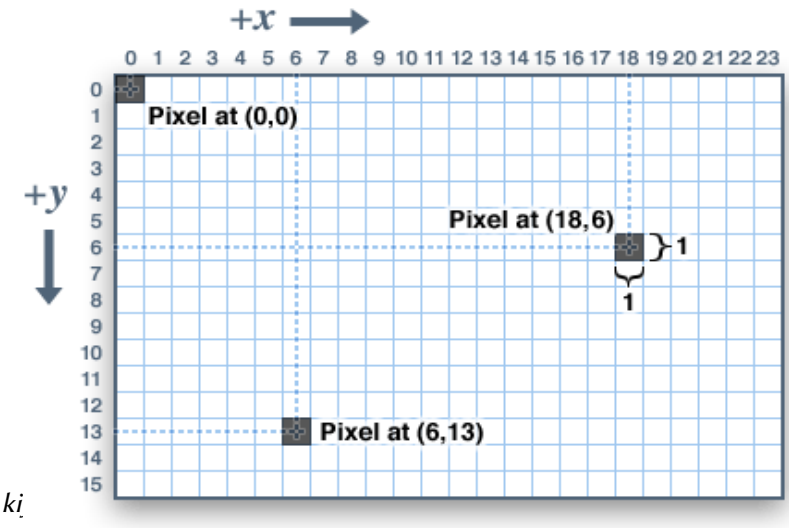
Grafikus kijelzők kezelése esetén két probléma adódik. Az első probléma a jelszint illesztés, mivel a legtöbb eszköz 3,3V feszültségről működik. Ez még viszonylag könnyen orvosolható jelszint illesztők segítségével. A komolyabb probléma az, hogy kijelző típusból és meghajtóból annyi van, mint égen a csillag és jellemzően mindegyik egyedi működésű, ami azt jelenti, hogy nincs igazán kiforrott univerzális kezelőkönyvtár megoldás minden kijelzőre.

Ez alól kivételt képeznek az Adafruit által forgalmazott kijelzők, mivel ezek ugyan azt a grafikus könyvtárrendszert alkalmazzák függetlenül az eszköz típusától. Ez úgy van megoldva, hogy a kód két részből áll. Egy grafikus könyvtárból, ami kijelző független és az ehhez kapcsolódó illesztőkből. Ezen projekt keretein belül a grafikus könyvtárt nézzük meg általánosságban, majd egy Nokia 5110/3310 kijelző kezelését.

Kijelzőtől függetlenül a grafikus könyvtár a https://github.com/adafruit/Adafruit-GFX-Library címről tölthető le. A könyvtár hivatalos dokumentációja és a támogatott kijelzők listája és az azokhoz tartozó illesztők a http://learn.adafruit.com/adafruit-gfx-graphics-library/overview címről tölthetőek le.

Alapismeretek

A kijelző koordináta rendszerének nulla pontja a kijelző bal felső sarkában található, ebből adódóan az Y koordináták jobbra haladva növekednek, míg lefelé haladva az X koordináták fognak növekedni. A legkisebb címezhető egység egy kijelzőpont.



Egy pixel színes ki i, hogy a piros és zöld csatorna 5 bit információt hordoz, míg a zöld csatornára 6 bit adat jut. Monokróm kijelzők esetén a pixel adat egy bit, ahol 0 a nem aktív pixelt jelenti, az 1-es érték pedig az aktív pixelt.

Könnyebb kezelhetőség végett a következő színek definiáltak a könyvtárban konstans formában:

Konstans	***Értéke hexadecimálisan***
BLACK	0x0000
BLUE	0x001F

RED	0xF800
GREEN	0x07E0
CYAN	0x07FF
MAGENTA	0xF81F
YELLOW	0xFFE0
WHITE	0xFFFF

120. Táblázat: Definiált alapszínek és értékük

Grafikus függvények

```
void drawPixel(x, y, color);
```

Egy pixel adott színűre állítása. Első paraméter az X koordináta, második az Y, a harmadik pedig a pixel színét határozza meg.

```
void fillScreen(color);
```

Kijelző kitöltése adott színnel.

```
void setRotation(rotation);
```

Kijelző elforgatásának beállítása. A kijelző csak 90 fok egész számú többszörösével forgatható el. Az elforgatás értéke 0 és 4 közötti szám kell, hogy legyen. 0 esetén az elforgatás 0 fok, 1-es esetén 90 fok, kettes esetén 180 fok, 3-as esetén pedig 270 fok.

```
void drawLine(x0, y0, x1, y1, color);
```

Vonal rajzolása x0, y0 és x1,y1 pontok között megadott színnel. X0, y0 paraméter a kezdőpontokat határozzák meg, míg x1 és y1 paraméterek a végpont koordinátái. Az utolsó paraméter a vonal színét határozza meg.

```
void drawFastVLine(x0, y0, length, color);
```

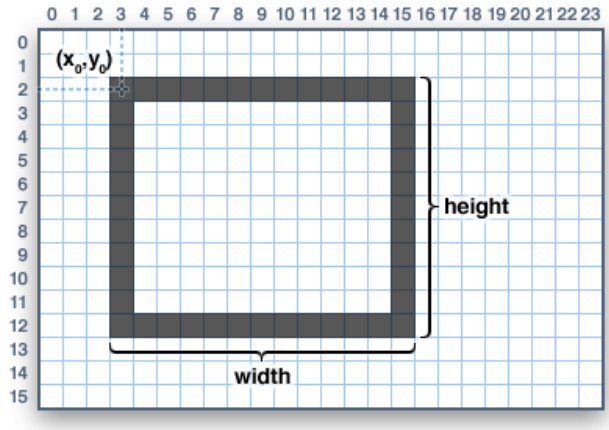
Optimalizált vízszintes vonal rajzoló függvény. X0 és Y0 a kezdőpontot határozzák meg, a length paraméter a vonal hosszát, míg az utolsó paraméter szintén a vonal színét.

```
void drawFastHLine(x0, y0, length, color);
```

Optimalizált függőleges vonal rajzoló függvény. Paraméterezése és használata megegyezik a drawFastVLine függvénnyel.

```
void drawRect(x0, y0, w, h, color);
```

Négyzet rajzolása. X0 és Y0 pontok a négyzet bal felső sarkát határozzák meg. A w és h paraméterek a négyzet szélességét és magasságát, míg a color paraméter a négyzet színét.




```
void fillRect(x0, y0, w, h, color);
```

Kitöltött négyzet rajzolása. Használata megyegyezik a drawRect függvénnyel.

```
void drawCircle(x0, y0, r, color);
```

Kör rajzolása. X0 és Y0 paraméterek a kör középpontját határozzák meg. Az r paraméter a sugarat határozza meg, míg az utolsó paraméter szintén a szín meghatározására szolgál.

```
void fillCircle(x0, y0, r, color);
```

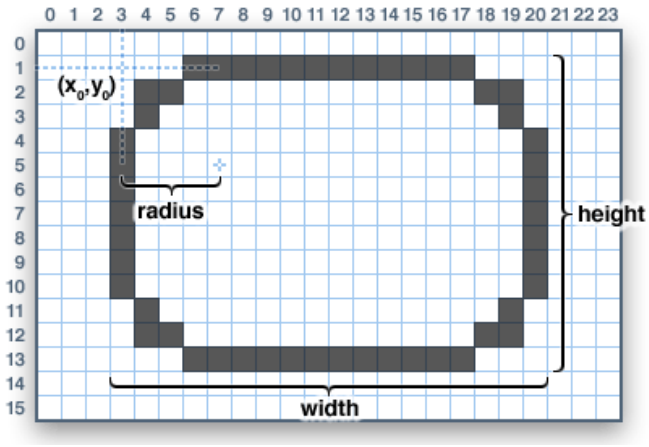
Kitöltött kör rajzolása. Használata megegyezik a drawCircle függvénnyel.

```
void drawRoundRect(x0, y0, w, h, radius, color);
```

Lekerekített sarkú téglalap rajzolása. X0 és Y0 paraméterek a bal felső sarok koordinátái, a w és h paraméter pedig a szélességet és magasságot határozza meg. A radius paraméter a lekerekítés értéke, a color pedig a pixel színe.

```
void fillRoundRect(x0, y0, w, h, radius, color);
```

Kitöltött lekerekített sarkú téglalap rajzolása. A használata megegyezik a drawRoundRect függvénnyel.



278. Ábra: Lekerekített sarkú téglalap

```
void drawTriangle(x0, y0, x1, y1, x2, y2, color);
```

Háromszög rajzolása x0, y0, x1, y1 és x2, y2 által meghatározott pontok között. A color paraméter a színt határozza meg.

```
void fillTriangle(x0, y0, x1, y1, x2, y2, color);
```

Kitöltött háromszög rajzolása. A függvény használata megegyezik a drawTriangle függvénnyel.

```
unsigned int width();
```

Kijelző szélességének lekérdezése

```
unsigned int height();
```

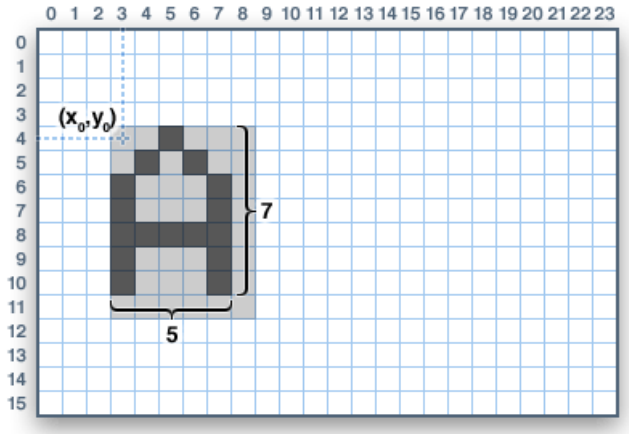
Kijelző magasságának lekérdezése

Szövegkezelés

Egy grafikus kijelző is tud szöveges üzemmódban működni, a szövegek kezelésére több függvény is rendelkezésre áll.

```
void drawChar(x, y, char, color, bg, size);
```

Egy karaktert rajzol a megadott X és Y pozícióba. A char paraméter a karaktert határozza meg, a color paraméter a karakter színét, a bg paraméter a háttér színét, a size paraméter pedig a karakter skálázását határozza meg. Alapértelmezetten 1-es skála esetén egy karakter 5 pixel széles és 8 pixel magas. 2-es skála esetén egy karakter 10 pixel széles és 16 pixel magas lesz.



A Fenti egy függvénnyel egy szöveg kirajzolása fájdalmasan sokáig tartana, ezért a könyvtár az alábbi extra függvényeket tartalmazza még szövegek kiírásához.

```
void setCursor(x0, y0);
```

Kurzor pozíciójának megadása x és y koordináták segítségével.

```
void setTextColor(color);
void setTextColor(color, backgroundcolor);
```

Szöveg színének meghatározása. Kétparaméteres változatában a második paraméter a háttér színét határozza meg.

```
void setTextSize(size);
```

Szöveg skála méretének meghatározása

```
void setTextWrap(w);
```

Szövegtördelés engedélyezése vagy kikapcsolása. Paramétere bool típusú és amennyiben igaz értékű, akkor a kijelző szélére érve a szöveg egy új sorban fog folyatódni.

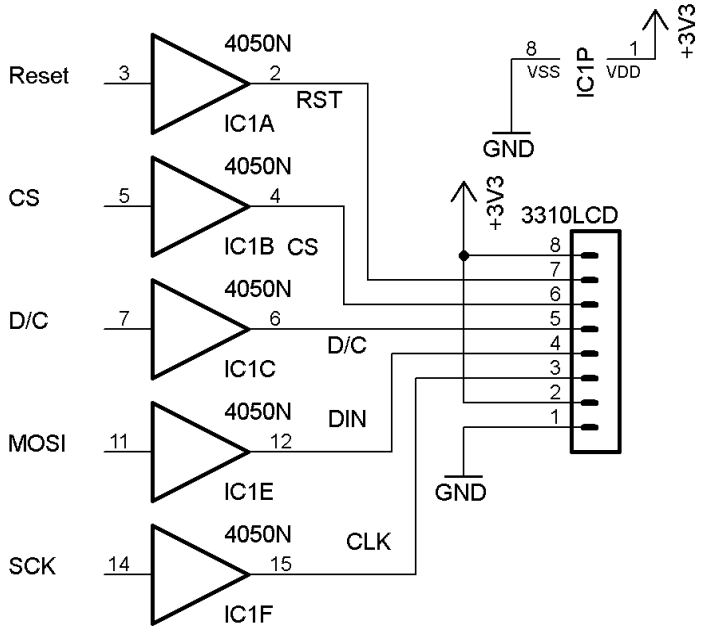
```
void print(string);
void println(string);
```

A Serial objektum hasonló nevű függvényeihez hasonlóan működik. Szöveg kiírására szolgál a kijelzőre. A

println változat a szöveg kiírása után sort tör.

Nokia 5110 kijelző kezelése

A Nokia5110-es és a 3310-es mobiltelefonokban használt kijelző egy igen egyszerűen kezelhető grafikus kijelző, amit igen könnyen és olcsón lehet manapság beszerezni, valamint nem csak az Adafruit forgalmazza. 84 pixel széles és 48 pixel magas rajzterülettel rendelkezik. 3,3 volt feszültségről működik, CD4050-es jelszint illesztő segítségével könnyen illeszthető Arduino-hoz. A kapcsolási rajz az alábbi ábrán látható:



A grafikus könyvtár által biztosított függvények a kijelző objektumán keresztül érhetőek el minden esetben. A kijelző objektumának konstruktora kijelző függően más és más. A paraméterek általában a használt lábak. Színes és SPI-t használó kijelzők esetén a MISO, MOSI és SCK lábakat nem kell a konstruktorban megadni, mivel ezek hardveresen adottak. Ezen kijelző is SPI buszrendszerre épül, azonban a kis mérete miatt nem feltétlen kell a hardveres SPI sebesség, a szoftveres is bőven jó. Ezért a kezelő könyvtár konstruktorának létrehozásakor ezen lábakat is meg kell adni.

A konstruktor paraméterezése és az objekum létrehozása:

```
Adafruit_PCD8544 ki = Adafruit_PCD8544(SCK, MOSI, D/C, CS, Reset);
```

Kijelző kezelő függvények

```
ki.begin();
```

Kijelző inicializálása

```
ki.display();
```

A legtöbb kijelző pixel szinten bufferel, így az adat beírás után tényleges kijelzőn megjelenítés csak parancs kiadása után fog történni. Ez a parancs a buffert jeleníti meg, vagyis csak a kiadása után lesz bármilyen változás látható a kijelzőn

```
ki.clearDisplay();
```

Kijelző buffer törlése, alaphelyzetre állítása

```
ki.setContrast(contrast);
```

Kijelző kontraszt beállítása. A paramétere 0 és 100 közötti érték lehet.

Példaprogram

```
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

Adafruit_PCD8544 ki = Adafruit_PCD8544(7, 6, 5, 4, 3);

void setup()
{
     ki.begin();
     ki.setContrast(50);
     ki.clearDisplay();
     ki.display();

     ki.drawRect(1, 1, ki.width()-1, ki.height()-1, BLACK);
     ki.setTextSize(1);
     ki.setTextColor(BLACK);
     ki.setCursor(5,5);
     ki.println("Nokia 3310 :)");
     ki.ki();
}

void loop()
{
}
```

Színes TFT kijelzők kezelése

A grafikus könyvtár használható a megfelelő illesztő használatával grafikus kijelzők kezeléshez is. Grafikus kijelzők esetén a függvények azonosak a korábban ismertetettekkel, viszont a szín meghatározására 16 bit információ használható ott.

Az MCU tools Színkeverő alkalmazása képes 24 bites színkódokat 16 bitessé alkalmazni, ami hasznos lehet rajzolási projekteknél.

Az alábbi táblázat azon illesztő könyvtárak beszerzési helyét sorolja fel grafikus vezérlőkhöz, amelyek alkalmazhatóak az Adafruit grafikus könyvtárával:

Vezérlő típusa	Letöltési hivatkozás
ILI9325, ILI9328	https://github.com/adafruit/TFTLCD-Library
HX8340B	https://github.com/adafruit/Adafruit-HX8340B
ST7735	https://github.com/adafruit/Adafruit-ST7735-Library
ILI9340	https://github.com/adafruit/Adafruit_ILI9340

121. Táblázat: Kompatibilis vezérlők és illesztő szoftverük beszerzési helye

Ezen kijelző modulok többsége rendelkezik egy SD kártya illesztéssel is, ami lehetővé teszi képek megjelenítését. Rengeteg képformátum létezik, de talán a legegyszerűbb felépítésű a népszerűbb képformátumok közül a BMP.

A táblázatban szereplő kijelző illesztők többsége tartalmaz egy minta programot, amely SD kártyáról rajzol ki a kijelzőre egy 24 bites színmélységgel tárolt BMP képet.

A BMP formátum előnye mikrovezérlős környezetben, hogy képes tömörítetlenül tárolni a kép adatokat, valamint a fejléc szerkezete pár byte-ból áll, így könnyű feldolgozni. Továbbá minden népszerű képszerkesztő program támogatja ezt a formátumot.

Persze a BMP képek alkalmazása nem annyira optimális, mint egy a könyvtár képességeihez igazított formátum használata. Azonban ehhez egy új formátumot kellene megalkotni, amihez egy konvertáló programot is írni kellene.

A betöltő kód az illesztők között példaképpen megtalálható és mivel eléggé hosszú, valamint a formátum részletes ismertetése nélkül nem érthető ezért nem került bele a könyvbe. A BMP formátum részletes felépítéséről és működéséről rengeteg információt a következő Wikipedia cikk tartalmaz: http://en.wikipedia.org/wiki/BMP_file_format

28. Szoftveres referencia feszültség kompenzáció

Az Arduino lapok táp ellátása megfelelő a lap meghajtásához, azonban ha nagyon pontos analóg érték olvasásra van szükségünk, akkor nem lesz megfelelő. Ez annak köszönhető, hogy az USB betáplálása nem minden esetben 5V. Valamint ha külső tápegységről tápláljuk, akkor is a stabilizátornak van némi hibája.

Ebből adódóan pontos analóg méréshez szükséges egy referencia feszültség forrás beiktatása. Ezekkel a probléma az, hogy típustól függően drágák is lehetnek, valamint ha 5V referencia feszültséget akarunk, akkor az USB táplálást kizárhatjuk.

Egy köztes megoldás, ha szoftverből kompenzáljuk a nem pontos referenciából adódó hibákat. Ezt megoldhatjuk manuálisan is, a feszültség lemérésével, majd programba írásával, azonban a következő bekapcsoláskor szinte biztos, hogy más feszültség értéket fogunk kapni, így mondhatni ez nem sokat ér, ha csak nem automatizáljuk a feladat elvégzését.

Automatizálni úgy tudjuk, hogy az Atmel mikrovezérlők belső referencia feszültsége 1,1V. Ehhez hasonlítva a külső tápfeszültséget megállapítható a pontos külső feszültség, amiből kompenzálható az olvasás pontatlansága. Az alábbi kód a https://code.google.com/p/tinkerit/wiki/SecretVoltmeter oldalról származik és a kompenzált olvasást mutatja be:

```
const long scaleConst = 1156.300 * 1000;
int readVccMv()
{
      // Read 1.1V reference against Avcc
      #if defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) ||
→ defined(__AVR_ATmega2560__)
      ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) |
_BV(MUX1);
      #elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || →
defined(__AVR_ATtiny84__)
      ADMUX = _BV(MUX5) | _BV(MUX0);
      #elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || →
defined(__AVR_ATtiny85__)
      ADMUX = _BV(MUX3) | _BV(MUX2);
#else
      ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
#endif

      delay(2); // Wait for Vref to settle
      ADCSRA |= _BV(ADSC); // Start conversion
      while (bit_is_set(ADCSRA,ADSC)); // measuring
      uint8_t low  = ADCL; // must read ADCL first - it then locks ADCH
      uint8_t high = ADCH; // unlocks both
      long result = (high<<8) | low;

      result = scaleConst / result;
      // Calculate Vcc (in mV); 1125300 = 1.1*1023*1000
      return (int)result; // Vcc in millivolts
}



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

```
void loop()
{
      double Vcc = readVccMv() / 1000.0;
      int ADCValue = analogRead(0);
      double Voltage = (ADCValue / 1023.0) * Vcc;
      Serial.println(Voltage);
      delay(1000);
}
```

A kód lényegi funkciója a readVccMv függvény, ami a tápfeszültséget hasonlítja a belső referenciához, majd a kettő eltéréséből kikövetkezteti a tényleges tápfeszültség értéket. Ezt a loop függvényben a soros port kimenetére írja a program minden másodpercben. A kód makrókkal van teletűzdelve a könnyű hordozhatóság miatt, így a readVccMv függvény használható Uno, Mega, Leonardo és Attiny 45/85/44/84 vezérlőkön is.

A pontosságot a kód nagymértékben növelte. Tesztelés során az Uno lapom tápfeszültsége 4,72 volt volt, amire analóg-digitális konverzió után azt mondta a lap, hogy 5V. Ez 0,28 V hibát jelentett.

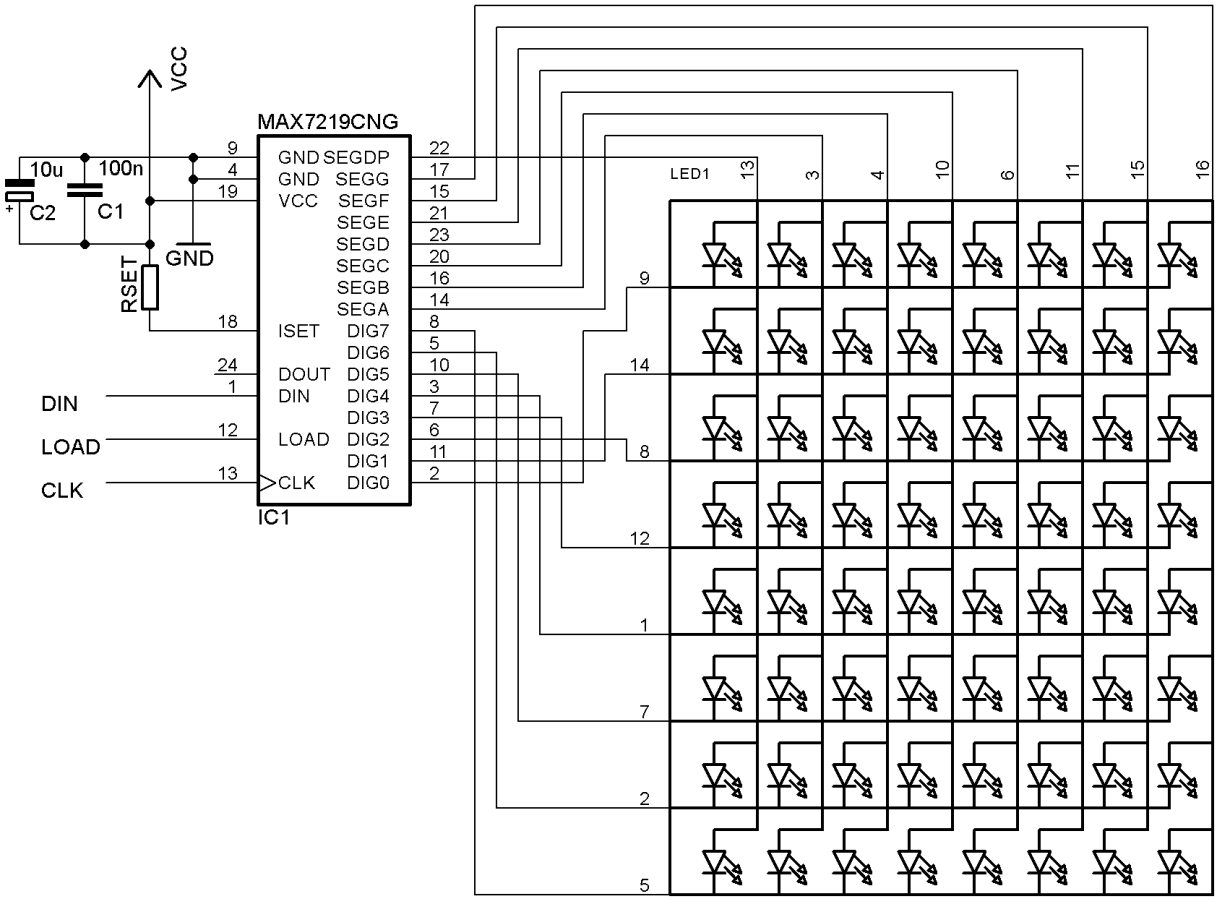
A módosított kód által mért feszültség érték 4,75 Volt volt, ami már csak 0,03 Volt hibát jelent, ami bőven a külső referencia feszültség források hibahatárán belül van.

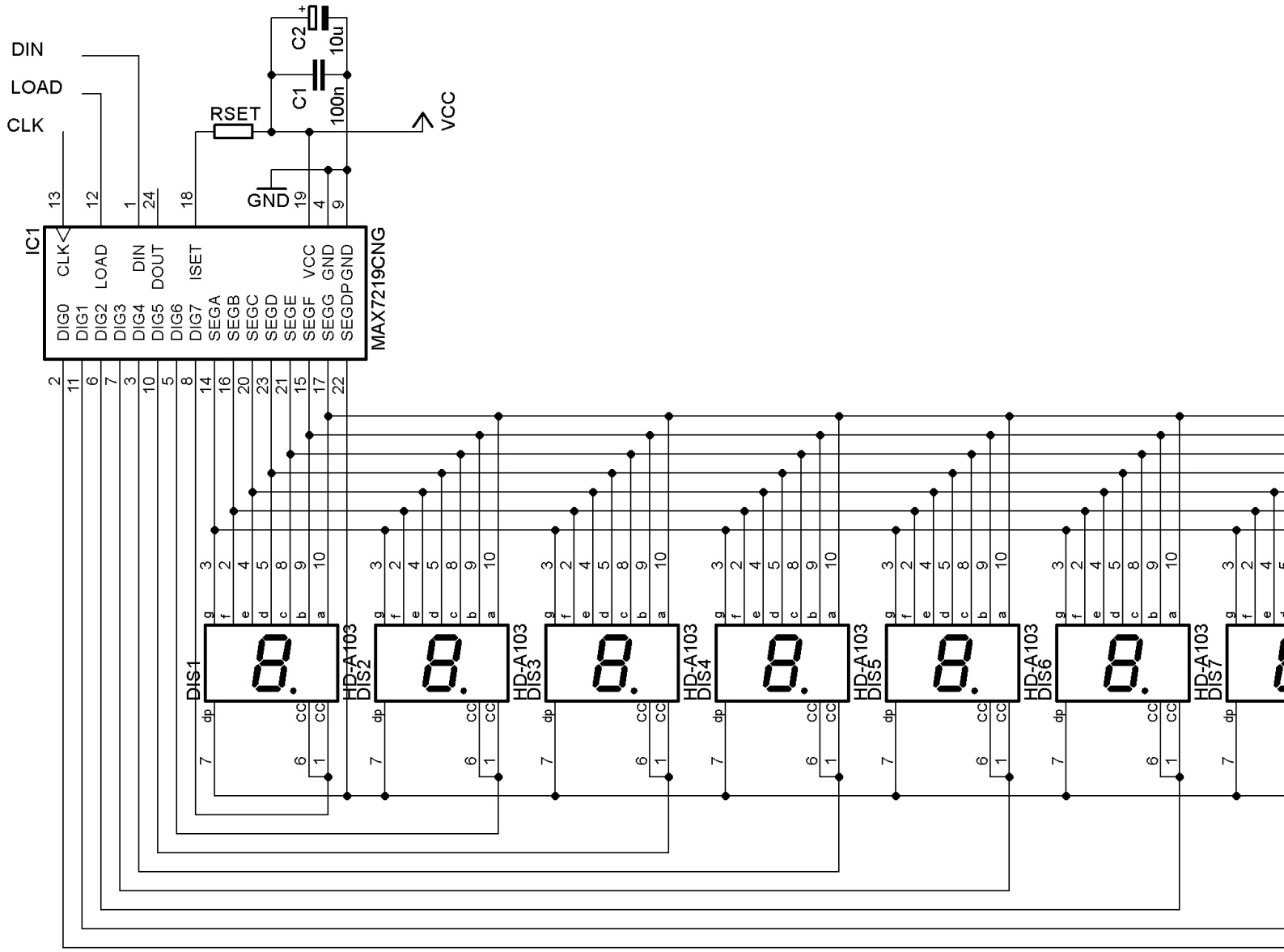
30. LED Mátrixok és hét szegmenses kijelzők kezelése

A hétszegmenses kijelzők és a LED mátrixok kezelése megvalósítható pár tranzisztorból álló kiegészítő áramkörrel, ha a mikrovezérlő rendelkezik elég kimenettel. Azonban ha nem rendelkezik, vagy nem akarunk időzítő alapú megszakításokkal vagy egyéb szoftveres trükkökkel bajlódni, akkor érdemes egy kiegészítő áramkört használni, mint a MAX7221 vagy MAX7219.

A két áramkör azonos lábkiosztással és funkcionalitással rendelkezik, amiben eltérnek az, hogy a MAX7221 teljes mértékben követi az SPI szabványt (már amennyire az SPI szabvány), míg a MAX7219 kicsit eltér ettől. Ettől függetlenül a 7219 ugyan úgy támogatott a legtöbb kezelő szoftver által, mint a 7221. A 7219 általában egy picivel olcsóbb, viszont ha a projektben analóg értékeket kell olvasni, akkor a 7219 használata nem javasolt, mivel a chip nem rendelkezik annyira jó elektromágneses árnyékolással, mint a 7221, ezért az analóg értékek olvasása torz adatokat fog visszaadni, ha az analóg bemenet közelében van egy 7219-es chip.

Ezen vezérlők áramgenerátoros kimenettel rendelkeznek, így a fényerő beállításhoz elég egyetlen egy ellenállás, nem kell minden egyes szegmens kimenetre ellenállást kötni, valamint hétszegmenses kijelző meghajtás mellett használhatóak LED mátrix meghajtásra is.




DIN
LOAD
CLK
RSET
C1
100n
C2
10u
VCC
GND
IC1
MAX7219CNG
CLK
LOAD
DIN
DOUT
ISET
VCC
GND
GND
DIG0
DIG1
DIG2
DIG3
DIG4
DIG5
DIG6
DIG7
SEGA
SEGB
SEGC
SEGD
SEGE
SEGF
SEGG
SEGDP
DIS1
DIS2
DIS3
DIS4
DIS5
DIS6
DIS7
HD-A103

Az Rset ellenállás értéke határozza meg az egyes szegmensek fényerejét. Az alábbi táblázat a hivatalos adatlap által ajánlott értékeket foglalja össze.

Szegmens Áram (mA)	**Nyitófeszültség (V)**				
	1,5	**2,0**	**2,5**	**3,0**	**3,5**
40	12,2kΩ	11,8kΩ	11kΩ	10,6kΩ	9,69kΩ
30	17,8kΩ	17,1kΩ	15,8kΩ	15,0kΩ	14,0kΩ
20	29,8kΩ	28,0kΩ	25,9kΩ	24,5kΩ	22,6kΩ
10	66,7kΩ	63,7kΩ	59,3kΩ	55,4kΩ	51,2kΩ

122. Táblázat: RSet ajánlott értékei nyitófeszültség és szegmens áram függvényében

Mindkét Chip esetén a maximális chip hőmérséklet 150 Celsius fok. Tokozástól függően változó a chip termikus ellenállása (θ_{JA}), amely 75 fok / Watt DIP tokozás esetén. Ebből az adatból és az elfűtött teljesítményből meghatározható, hogy mennyi lesz a legrosszabb esetben az áramkör hőmérséklete adott konfiguráció mellett. Az elfűtött teljesítmény az alábbi képlet segítségével határozható meg.

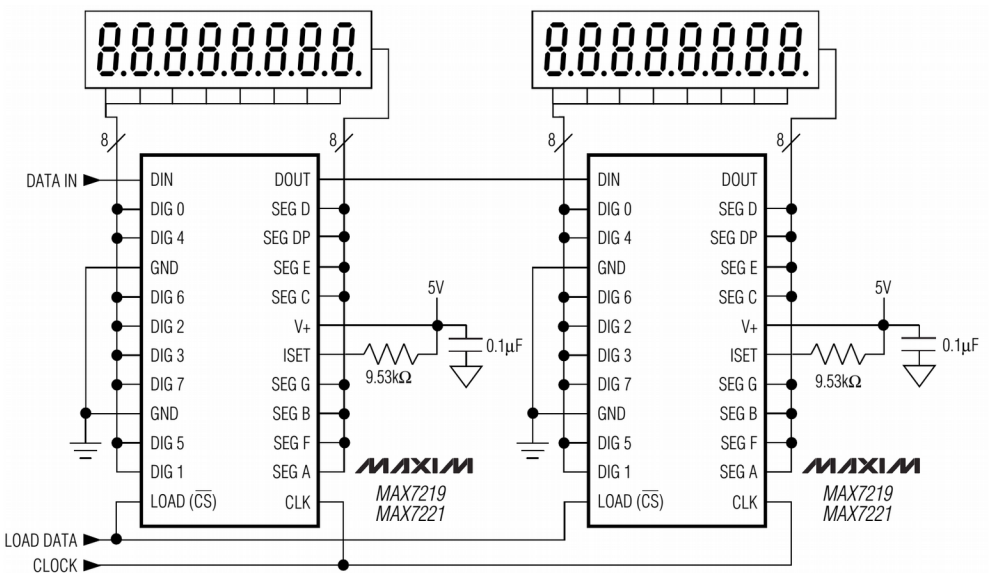
$$P_D=(U_{be}\cdot 8\,mA)+(U_{be}-U_{LED})\cdot(0{,}96875\cdot I_{SEG}\cdot N)$$

A képletben N a kijelzők számát jelenti, míg I_{SEG} az egyes szegmensek áramát, U_{be} pedig a tápfeszültséget.

A maximálisan várható chip hőmérséklet az alábbi képlet segítségével határozható meg az elfűtött teljesítmény és a termikus ellenállás ismeretében:

$$T_J=61{,}2\,°C+P_D\cdot\theta_{JA}$$

A soros adat bemeneti felépítésnek köszönhetően több MAX7221/MAX7219 kaszkádba köthető, így több kijelző/mátrix vezérlése lehetséges. Kaszkád kötés esetén az összes vezérlő osztozik a load és clk bemeneteken. Az első vezérlő esetén a DIN bemenetre kell az adatot küldeni, a második vezérlő pedig a DIN bemenetét az első kijelző DOUT kimenetéről kapja. Ilyen módon 8 vezérlő összekötése lehetséges.



Kezelő szoftver

Számos implementáció található az interneten a vezérlők kezeléséhez, azonban a legteljesebb a LedControl névre hallgató könyvtár, amely a https://github.com/wayoda/LedControl/ címről szerezhető be. Ez a könyvtár LED mátrixokat és hétszegmenses kijelzőket is támogat.

A könyvtár a LedControl osztályt tartalmazza, amelyet használat előtt példányosítani kell a következő

módon:

```
LedControl lc = LedControl(DataIn, CLK, Load, N)
```

A DataIn, CLK, Load paraméterek azokat a lábakat határozzák meg, amelyeken keresztül elérhető a vezérlő. Az N jelű paraméter a vezérlők számát határozza meg. Ez egy 1 és 8 közötti szám.

Általános függvények

```
lc.getDeviceCount();
```

Visszatérési értéke az lc objektumon keresztül kezelhető vezérlők száma.

```
lc.shutdown(N, SLEEP);
```

Az első paraméter által meghatározott számú vezérlőt energiatakarékos állapotba kapcsolja, vagy energiatakarékos állapotból ébreszti. A működési módot a második paraméter határozza meg. Amennyiben ez igaz, akkor a vezérlőt alvó módba teszi. Hamis érték esetén pedig felébreszti a kijelzőt. A vezérlők tápfeszültség kapcsolás után alapértelmezetten alvó állapotban vannak.

```
lc.setScanLimit(addr, limit);
```

Az első paraméter által meghatározott vezérlőn beállítja a hétszegmenses kijelzők számát vagy a mátrix sorainak számát. A legtöbb mátrix kijelző esetén nem szükséges, mivel alapértelmezetten mind a 8 kijelzőt vagy sort meghajtja a vezérlő.

```
lc.setIntensity(addr, intensity);
```

Az első paraméter által meghatározott vezérlőn beállítja a fényerőt. A fényerőt a második paraméter határozza meg. A fényerő 0 és 15 közötti szám lehet, ahol 0 esetén a kijelző kikapcsolt állapotban van, 15 pedig maximális fényerőn. A kijelző inicializálásakor be kell állítani a fényerőt is, mivel alapértelmezetten 0 értékre van állítva.

```
lc.clearDisplay(addr);
```

A paraméter által meghatározott vezérlőbe írt megjelenítési adatokat törli.

LED mátrixok függvényei

```
lc.setLed(addr, row, col, state);
```

Az első paraméter által meghatározott vezérlőn egy LED-et bekapcsol vagy kikapcsol. Az állapotot az utolsó paraméter határozza meg, amely logikai típusú. A row és column paraméter a LED koordinátáit határozzák meg oszlop, sor formában.

```
lc.setRow(addr, row, value);
```

Első paraméter által meghatározott vezérlőn egy sor értékének beállítása. A sort a második paraméter határozza meg, a harmadik paraméter pedig egy bájt formában a 8 LED értékét határozza meg. A legnagyobb helyiértékű bit van megfeleltetve a sor bal szélén található LED-nek.

```
lc.setColumn(addr, col, value);
```

Első paraméter által meghatározott vezérlőn egy oszlop értékének beállítása. Az oszlopot a második paraméter határozza meg, a harmadik paraméter pedig egy bájt formában a 8 LED értékét határozza meg. A legnagyobb helyiértékű bit van megfeleltetve az oszlop tetején található LED-nek. Ezen függvény használata a vezérlő belső felépítéséből adódóan lassabb, mint ha sorokat állítanánk be.

7 szegmenses kijelzők függvényei

```
lc.setDigit(addr, digit, value, dp);
```

A digit paraméter által meghatározott kijelző értékét állítja be az addr paraméter által meghatározott vezérlőn. A value paraméter az érték, ami 0 és 15 közötti lehet. Ha a dp paraméter igaz értékű, akkor a kijelzőhöz tartozó decimális pont jelző be lesz kapcsolva.

```
lc.setChar(addr, digit, value, dp);
```

Egy karaktert jelenít meg a digit paraméter által meghatározott kijelzőn az az addr paraméter által meghatározott vezérlőn. A karaktert a value paraméter határozza meg. Ha a dp paraméter igaz értékű, akkor a kijelzőhöz tartozó decimális pont jelző be lesz kapcsolva.

A value paraméter helyére, ha olyan karakter kerül, amit a vezérlő nem tud értelmezni, akkor az adott kijelző összes LED-je ki lesz kapcsolva. A megjeleníthető karakterek a következőek:

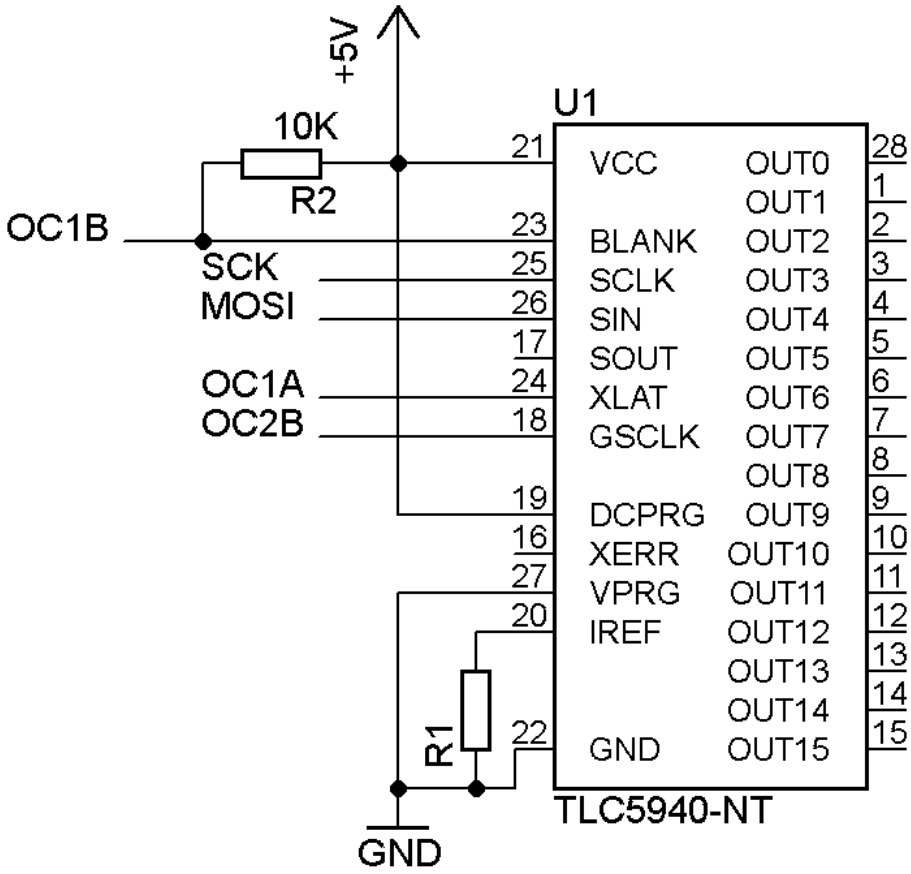
```
0 1 2 3 4 5 6 7 8 9 A B C D E F H L P - . _
```

31. PWM kimenetek bővítése

Modell függően az Arduino rendelkezik pár darab PWM kimenettel, de ezek csak 8 bites pontosságúak. Ha nagyobb pontosságot szeretnénk elérni, vagy 16 PWM kimenetre van szükségünk, akkor használhatunk egy dedikált PWM vezérlőt. Ezekből igen sokfajta létezik a piacon. A Texas Instruments által gyártott TLC5940-es vezérlő azért ideális, mivel viszonylag olcsón beszerezhető és létezik DIP tokozású változata, amely otthoni barkácsolási célokra ideálissá teszi.

Az IC 16 kimeneti csatornával rendelkezik. Mindegyik csatorna 12 bit pontosságú, amely 16x pontosabb vezérlést tesz lehetővé, mint a legtöbb Arduino beépített PWM generátora. Az IC kimenetei nyitott kollektoros felépítésűek, amelyek konstans áram vezérlésűek. Elsősorban LED-ek vezérlésére találták ki.

Az IC bekötése az alábbi ábrán látható:



A konstans áram kimenet nagyságát az Iref és GND lábak közé kötött R1 ellenállás határozza meg. A csatornákra jutható maximális áram az alábbi képlet segítségével határozható meg:

$$I_{MAX} = \frac{1{,}24}{R_{IREF}} \cdot 31{,}5$$

A csatornánkénti áramerősség 5mA és 120Ma között szabályozható. Ha az áramerősség 5mA-nál kisebb, akkor a kimenet nem stabil. A chip teljes terhelhetősége 130mA. 25mA kimeneti áramerősséghez a fenti képlet alapján $1562\,\Omega$ *ellenállás kell.*

A szükséges kezelő könyvtár a https://code.google.com/p/tlc5940arduino/ címről tölthető le. A könyvtár statikus bekötésű, ami a lábakat illeti. Ezért a szükséges csatlakozások kialakítása Arduino modellenként eltér. Az alábbi táblázat a kapcsolási rajzban szereplő jelöléseket foglalja össze, hogy adott Arduino modell esetén melyik lábhoz kell kötni. A könyvtár csak Uno és Mega modelleket támogat.

	Uno	Mega
MOSI	11	26
SCK	13	25
OC1A	9	24
OC1B	10	23
OC2B	3	18

123. Táblázat: A vezérlő IC bekötése Arduino modellek esetén

A könyvtár az alábbi kezelőfüggvényeket biztosítja:

```
Tlc.init(value);
```

Inicializálja az összes kimenetet adott PWM értékkel 0% kitöltési tényezőnek 0 felel meg, 100%-os kitöltési tényezőnek meg 4095

```
Tlc.clear();
```

Törli a beírt PWM értékeket az összes csatornáról. Hatása csak egy frissítés után látszik a kimeneteken.

```
Tlc.set(channel, value);
```

Adott csatorna kitöltési tényezőjét állítja be. Az első paraméter a csatorna, amely 0 és 15 között kell, hogy legyen. A második paraméter pedig a kitöltési tényező amely, 0 és 4095 között lehet. Hatása csak egy frissítés után látszik a kimeneteken.

```
Tlc.setAll(value);
```

Összes csatorna adott értékre állítása. Hatása csak egy frissítés után látszik a kimeneteken.

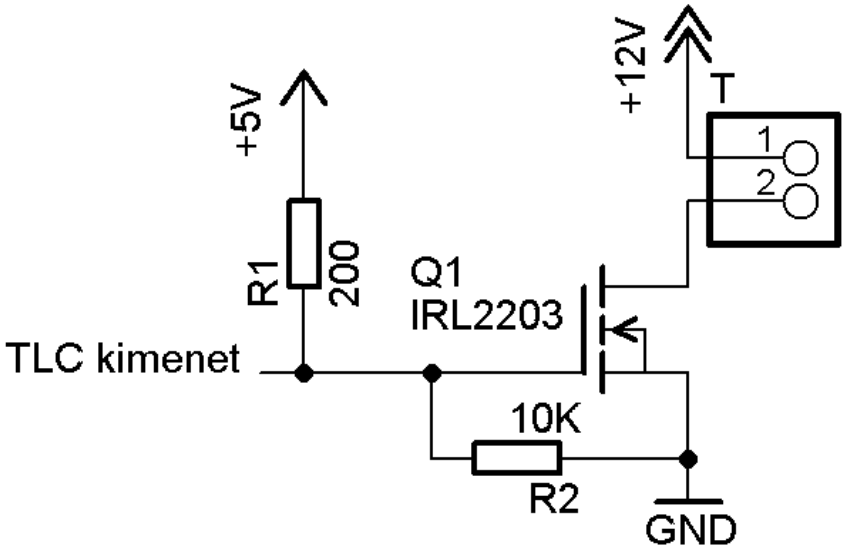
```
short int Tlc.get(channel);
```

A paraméterként megadott csatorna kitöltési tényezőjének lekérdezése.

```
Tlc.update();
```

Kimenetek frissítése a beírt PWM értékek alapján. A függvényt meg kell hívni minden set, clear és setAll hívás után, hogy a beírt értékek a kimenetre is átkerüljenek.

Az egyes LED-ek feszültsége elérheti a 15V-ot is a nyitott kollektoros kialakítás miatt, így nagy fényerejű LED-ek meghajtása is lehetséges az IC-vel, egészen addig, amíg az IC terhelése nem éri el a 130mA áramot. A kimenetekre FET-ek is kapcsolhatóak, amely segítségével tovább növelhető a terhelhetőség. Azonban ehhez egy speciális kapcsolást kell alkalmazni a kimeneti fokozatok nyitott kollektoros mivoltából adódóan.

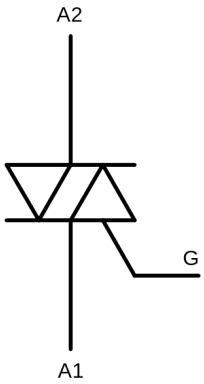


A kapcsolás az IRL2203 jelű logikai FET-hez lett igazítva. A TLC áramkörön az áramkorlátozó ellenállás $1{,}5K\,\Omega$ *nagyságú, amely ~26mA maximális áramot enged át maximum. A* $200\,\Omega$ *felhúzó ellenállás ez alapján került megállapításra, amely ténylegesen 25mA áramot fog átengedni. Az ellenállások értékei növelhetőek, mivel a FET-nek nincs szüksége áramra. Gyorsabb kapcsolás érdekében tranzisztor is alkalmazható.*

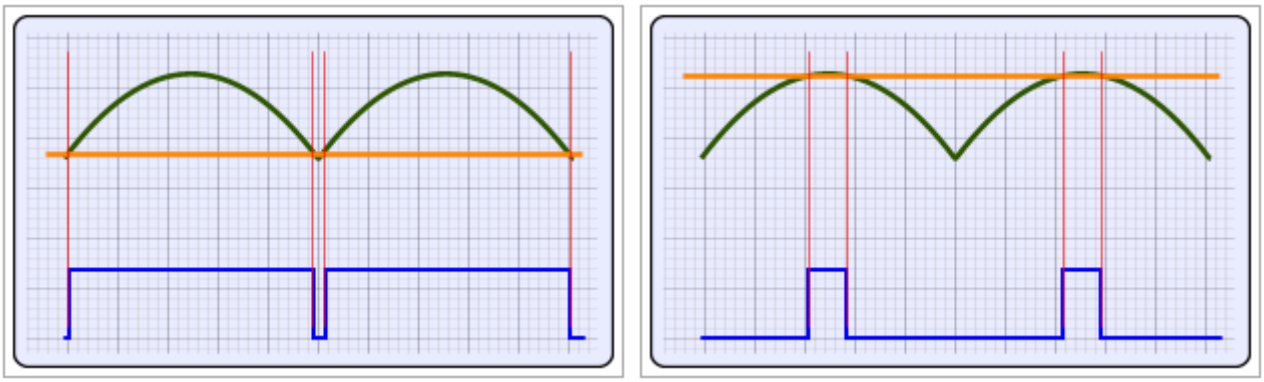
32. Váltakozó áram szabályzása

Váltakozó áram könnyen kapcsolható Arduino segítségével, csupán egy relé kell és egy tranzisztoros meghajtás, amely kapcsolja a relét ki és be. Azonban ha ki és bekapcsolás helyett tényleges szabályzás szeretnénk végezni, akkor bonyolultabb áramkörre lesz szükségünk.

Váltakozó áram szabályzására a legjobb áramköri elem a Triac. Ez egy olyan elektronikai eszköz amely képes áram vezetésre mindkét irányba és a Gate elektróda feszültségétől függően a váltakozó áram negatív vagy pozitív periódusában is kapcsolásra képes.

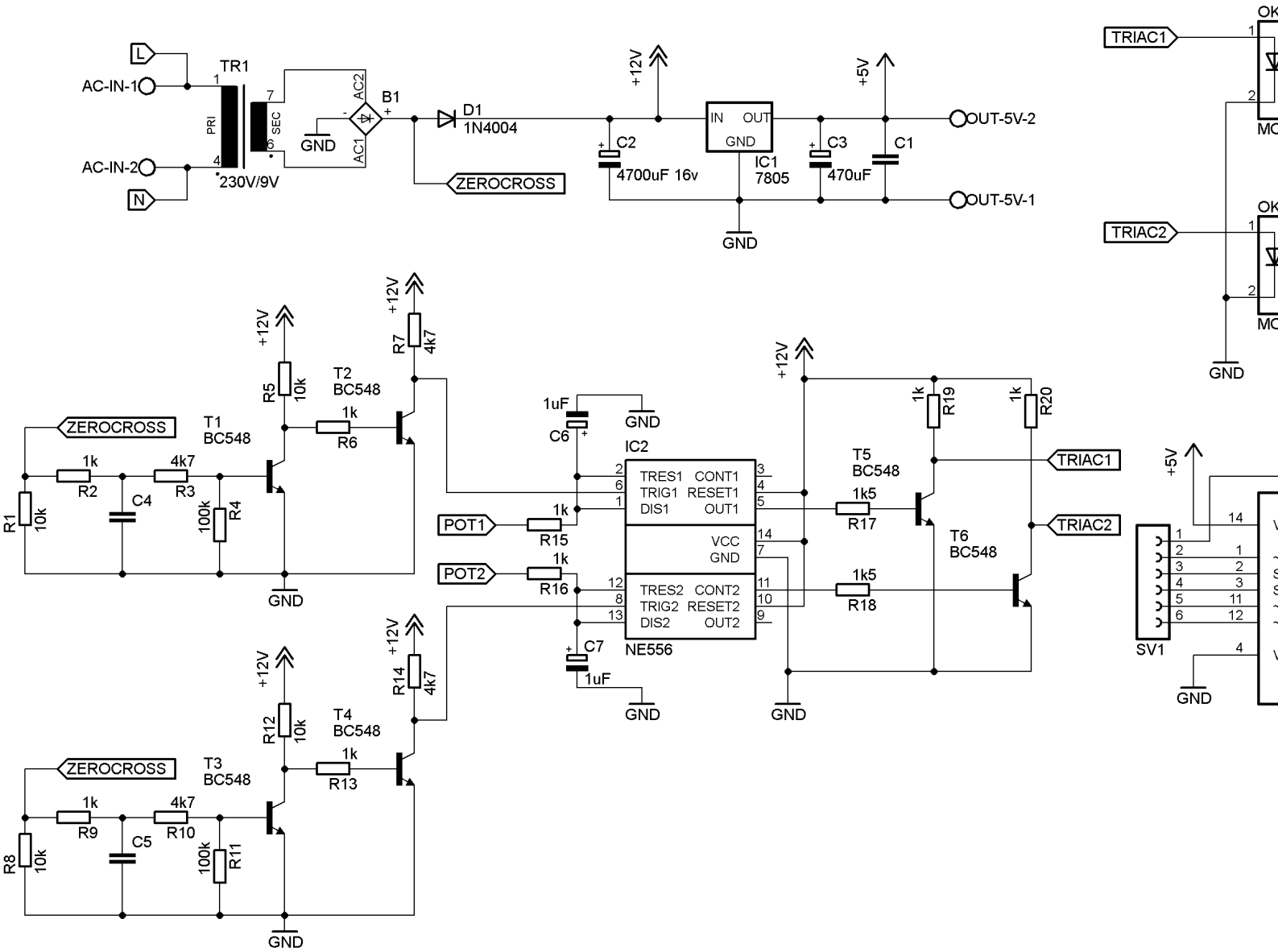


Egy triac esetén a bekapcsolt állapot hosszától függ a kimeneti feszültség effektív értéke. Ha a szinusz hullám teljes periódus ideje alatt bekapcsolva van, akkor az effektív érték változatlan marad. De ha csak a periódus idő 10%-ban van bekapcsolva, akkor az effektív érték az eredeti 10%-a lesz.



A fenti ábrán kék színnel van jelölve a triac gate feszültsége, zöld színnel egy egyenirányított váltakozó áramú jel látható, amely a gate időzítésének meghatározására szolgál. Ha teljes időben be van kapcsolva a triac, akkor változás nélkül továbbjut a szinusz jel. Viszont ha csak az idő egy részében, akkor a szinusz jel periódusából csak egy rész jut tovább, amely befolyásolja az effektív értéket.

A triac működése időzíthető mikrovezérlővel. Azonban jobb megoldás, ha külön időzítőt használunk erre a célra, így az Arduino könyvtárak és függvények működését nem befolyásolja a vezérlés.


L
AC-IN-1
AC-IN-2
N
TR1
PRI
SEC
230V/9V
AC2
AC1
B1
GND
D1
1N4004
ZEROCROSS
+12V
C2
4700uF 16v
IC1
7805
IN
OUT
GND
C3
470uF
C1
+5V
OUT-5V-2
OUT-5V-1
TRIAC1
TRIAC2
T1
BC548
T2
BC548
T3
BC548
T4
BC548
R1
10k
R2
1k
R3
4k7
R4
100k
R5
10k
R6
1k
R7
4k7
R8
10k
R9
1k
R10
4k7
R11
100k
R12
10k
R13
1k
R14
4k7
C4
C5
1uF
C6
C7
IC2
NE556
TRES1
CONT1
TRIG1
RESET1
DIS1
OUT1
VCC
GND
TRES2
CONT2
TRIG2
RESET2
DIS2
OUT2
POT1
POT2
R15
1k
R16
1k
T5
BC548
T6
BC548
R17
1k5
R18
1k5
R19
1k
R20
1k
SV1

A kapcsolási rajzon egy 555-ös időzítővel megvezérelt Triac szabályzó áramkör látható. A Triacok vezérlő jelét egy 556-os időzítő szolgáltatja. Ez az integrált áramkör 2db 555-ös időzítőt tartalmaz egy tokban. Minkét időzítő monostabil üzemmódban működik, a kapcsolási időt egy digitális potméter szabályozza. A szabályzó jelet a 230V-os letranszformált feszültség szolgáltatja.

A kapcsolás mikrovezérlő oldalról egy digitális potméterként látszik. Ezen változat tervezésekor a korábbi projektben bemutatott potméter $100\,K\,\Omega$ *ellenállású változatát használtam. Az ellenállás változtatásával változtatható az effektív feszültség.*

A kapcsolásban a fázis és nulla nem cserélhető fel biztonsági okokból. A fázist az L bemenetre kell kötni, a nulla vezetőt pedig az N jelű bemenetre. Opcionálisan a kimenet ki és bekapcsolására relék is beiktathatóak.

Csak rezisztív jellegű terhelések vezérlése alkalmas az áramkör a fenti konfigurációt alkalmazva, vagyis villamos fűtések és nem energiatakarékos lámpák gond nélkül szabályozhatóak vele 6A áramerősségig a kapcsolásban szereplő Triac-ot alkalmazva.

Elméletben lehetséges induktív terhelések, egyfázisú motorok szabályzása is, de ehhez módosítani kell a triac kimeneti fokozatot az induktív visszahatások miatt.

23. RaspberryPi projektek

1. Vezeték nélküli hálózatra csatlakozás

A RaspberryPi nem rendelkezik beépített WLAN adapterrel, azonban az USB portjain keresztül könnyen felruházható WLAN képességekkel. Pár kivételtől eltekintve a boltokban kapható USB WLAN adapterek többsége gond nélkül működik Linux alatt. Vásárlás előtt azonban érdemes rákeresni az adott adapter támogatottságára.

Az USB adaptert kikapcsolt állapotban érdemes bedugni, mivel a legtöbb USB tápegységgel előfordulhat, hogy az adapter csatlakoztatásakor újraindul a lap. Ez akkor következik be, ha a használt tápegység nem rendelkezik kellő kimeneti pufferelléssel, ezért a hirtelen szükséges többlet áramot nem tudja biztosítani, minek következtében leesik egy rövid időre a feszültség, ami a lap újraindulását idézi elő.

Az USB adapter csatlakoztatása után arról, hogy a rendszer felismerte e az adaptert a következő parancs kiadásával tájékozódhatunk:

```
sudo lsusb
```

Ez a parancs kilistázza az összes csatlakoztatott USB eszközt. Alapértelmezetten ha nincs semmi csatlakoztatva, akkor 3 sort kellene kapnunk B modell esetén. Felismert esetben pedig eggyel többet. Az általam használt TP-LINK eszköz esetén a következő sort kaptam:

```
Bus 001 Device 004: ID 0cf3:9271 Atheros Communications, Inc. AR9271
802.11n
```

A WPA és WPA2 titkosítást alkalmazó hálózatok használatához telepíteni kell a wpasupplicant csomagot a következő parancs segítségével:

```
sudo apt-get install wpasupplicant
```

Ezután konfigurálható a hálózati eszköz. A /etc/network/ mappában található interfaces fájl tartalmát kell módosítani. Ezen fájl megnyitásához használhatunk bármilyen szövegszerkesztőt, azonban ha parancssorról állítjuk be a hálózatot, akkor a nano szerkesztő jöhet szóba, mivel alapértelmezetten ez telepítve van. A fájl módosításához rendszergazdai jogok kellenek.

sudo nano /etc/network/ifaces

A fájlba a következő bejegyzéseket kell hozzáadnunk/módosítanunk:

```
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
     wpa-ssid "[SSID]"
     wpa-psk "[jelszó]"
```

Az auto wlan0 sor az automata konfigurációt engedélyezi, ha nincs csatlakoztatva adapter, az alatta lévő sor pedig a menet közbeni csatlakoztatást/leválasztást engedélyezi. Az iface kezdetű sor utáni rész pedig az adapter konfigurációja, ha csatlakoztatva van. Ebben az esetben DHCP protokoll segítségével konfigurálja az eszközt.

A konfigurációban az [SSID] bejegyzést cseréljük ki a hálózatunk SSID nevére, valamint a [jelszó] helyére pedig a csatlakozáshoz szükséges jelszót írjuk be.

A módosítások befejezésével a nano-ban ctrl+x billentyűkombinációval menthetjük el a fájlt. A konfiguráció életbeléptetéséhez az alábbi parancsokat kell kiadni:

```
sudo ifdown wlan0
sudo ifup wlan0
sudo /etc/init.d/networking restart
```

Ha minden rendben van, akkor a parancsok kiadása után az ifconfig parancs kimenetében látszódnia kellene, hogy a wlan0 adapter címet kapott. Esetemben így nézett ki a kimenet:

```
wlan0  Link encap:Ethernet  HWaddr 74:ea:3a:8f:69:82
       inet addr:192.168.1.102  Bcast:255.255.255.255
Mask:255.255.255.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:3407 errors:0 dropped:0 overruns:0 frame:0
       TX packets:1187 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:1037607 (1013.2 KiB)  TX bytes:154482 (150.8 KiB)
```

Ha esetlegesen olyan problémába ütközünk, hogy a helyi hálózat elérhető, de az internet nem, akkor újra kell indítani a lapot, utána a problémák nagy valószínűséggel megoldódnak.

Amennyiben a lapot több WLAN környezetben használni szeretnénk, akkor definiálhatunk virtuális hálózati kártyákat, amelyekkel könnyebben konfigurálhatóak a különböző hálózatok. Egy virtuális konfiguráció így néz ki:

```
iface wlan_munkahely inet dhcp
     wpa-ssid "munkahely"
     wpa-psk "jelszo"
```

A virtuális konfiguráció az ifup parancs segítségével alkalmazható, méghozzá így:

```
sudo ifup wlan0=wlan_munkahely
```

Biztonsági okokból érdemes az ifaces fájl tartalmát elrejteni a felhasználók elől, mivel titkosítatlanul tartalmazza a jelszavakat. Az ehhez kiadandó parancs:

```
sudo chmod 600 /etc/network/ifaces
```

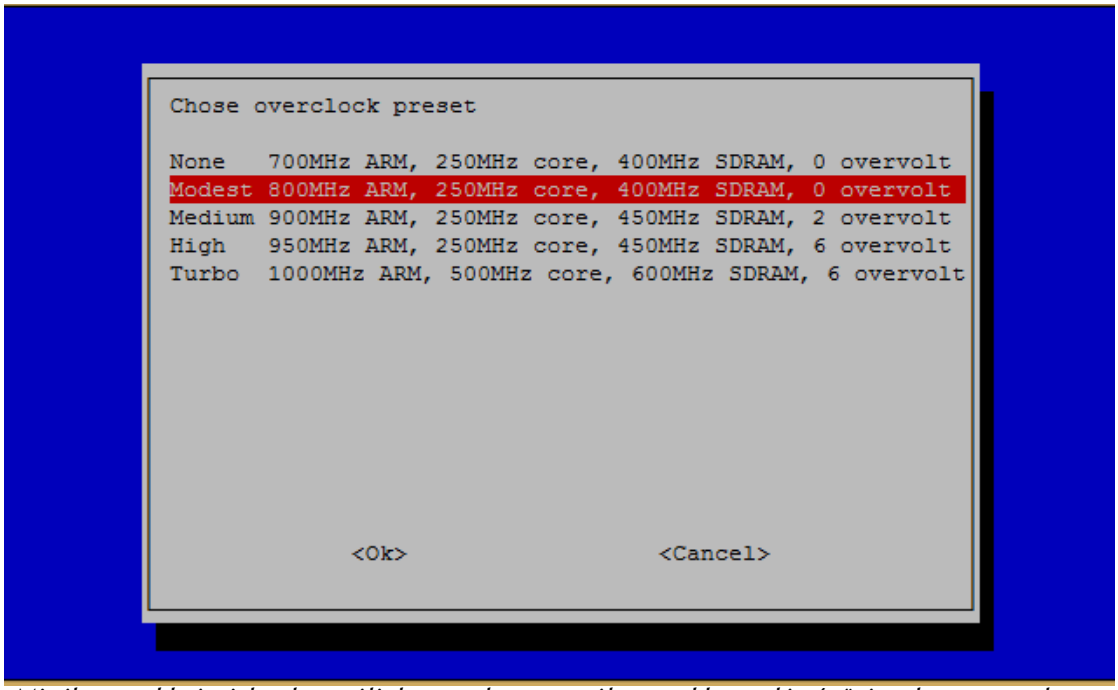
2. Magasabb órajel használata

A Pi processzora 700Mhz-en megy, ami nem túl gyors. A teljesítmény növelésének egyik módszere az órajel növelése. Ennek azonban megvannak a veszélyei. A legnagyobb veszélyt a Pi számára a túlmelegedés jelenti. Ezért ajánlatos a memóriát és a processzort hűteni. A túlmelegedés nem feltétlen azonnal öli meg az eszközt, elég ha az élettartalmát megrövidíti.

Az órajel növelése előtt első lépésben ajánlatos biztonsági másolatot készíteni a rendszerről. Erre azért van szükség, mivel elképzelhető, hogy a nagyobb órajel mellett nem fog elindulni a rendszer. Biztonsági mentésre használható a Win32 Disk Imager program, de elég ha csak a kártya első boot partícióján található config.txt fájlról készítünk biztonsági másolatot.

Amennyiben a módosított beállítások után a rendszer nem lenne stabil, vagy nem indulna el, akkor a config.txt fájl visszaállítása után ismét működőképes lesz a rendszer.

Az órajel a beüzemeléskor használt raspi-config alkalmazásból állítható. A program indítása után az overclock menüből állítható az órajel a gyárilag megadott értékek között. Elméletileg egészen 1GHz-ig lehet elvinni a sebességet.



Minél nagyobb órajelen használjuk a rendszert annál nagyobb a valószínűsége, hogy a rendszer nem lesz stabil, vagy a lap nem lesz képes elindulni. Ez több mindentől függ. Elsősorban a használt tápegységtől és a memóriakártyától. Másodsorban pedig magától a laptól is függ, mivel két processzor sem egyforma a gyártási technológiából adódóan.

Amennyiben asztali gépnek használjuk a Pi-t, akkor a rendszer indulása közben a SHIFT gomb nyomva tartása megakadályozza a nagyobb órajelre átállást.

A stabilitás tesztelése érdekében írtam egy CPU tesztelő programot, amely az Ackermann függvény értékeinek számításával teszteli a processzort. Az ackermann függvényről azt kell tudni hogy az értéke nagyon gyorsan növekszik, így már kis helyeken is hatalmas értékeket vesz fel. A (4,3) argumentum esetén a függvény értéke akkora, hogy tízes számrendszerben több jegy kell a felírásához, mint ahány részecske az univerzumban van. A tesztelő program (4, 1) argumentumig számítja az értékeket. Ez a 800MHz-es órajelen futó teszt gépemen kicsivel kevesebb, mint 400 másodpercet vett igénybe.

Ha a teszt közben megfagyna a lap, akkor újraindítás után állítsuk kisebb órajelre.

A memória kártya tesztelése már nem ennyire egyértelmű, mivel olvasási és írási műveleteket is kell végezni rajta. Nem a legjobb teszt, de kezdetnek jó az APT csomagtároló szoftverlistájának frissítése:

```
sudo apt-get update
```

Ez írási és olvasási műveleteket is generál. Ha ezen teszt közben elvérezne a Pi, akkor az azt jelenti, hogy az általunk használt kártya nem képes lépést tartani a megnövekedett sebességgel. Ha a tesztet sikeresen teljesítette a Pi, akkor sem lehetünk száz százalékig biztosak, mivel lehet, hogy csak huzamosabb terhelés, használat esetén jelentkezik lassulás.

Mondhatni azt, hogy lutri, hogy mekkora órajelen képes maximálisan működni a lapunk, viszont megéri próbálkozni kellő odafigyeléssel, főleg olyan esetekben amikor jól jönne az extra teljesítmény.

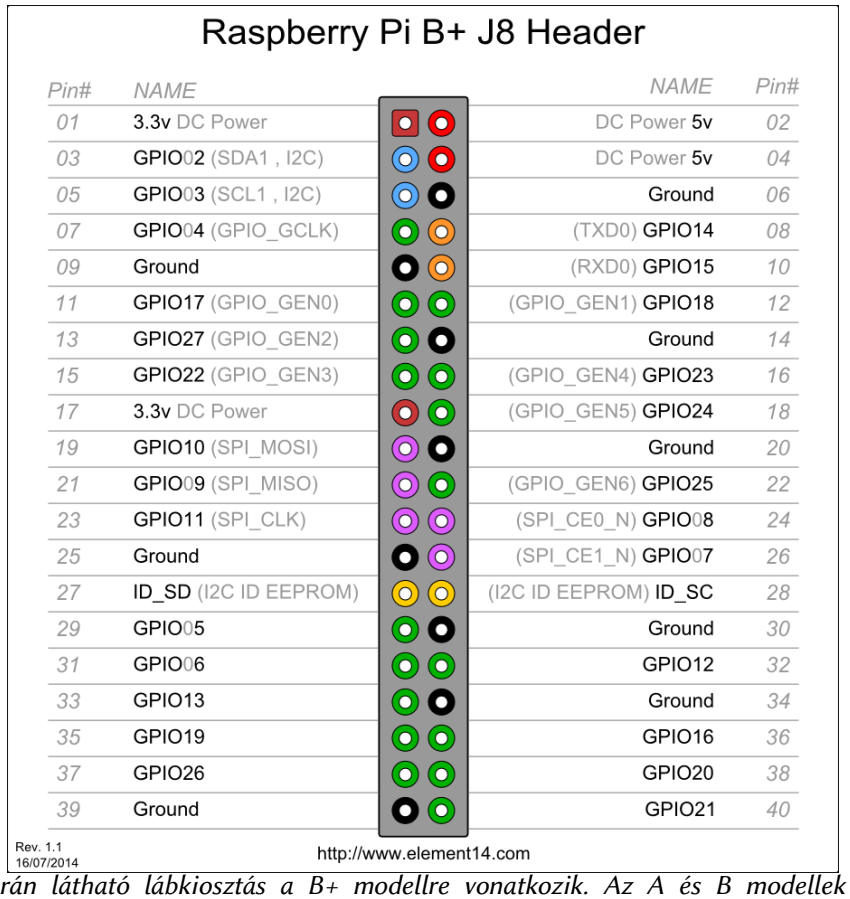
A processzor tesztelésére használt program:

```
//compile with: g++ -lpthread AckermannTest.cpp -o AckermanTest
#include <iostream>
#include <ctime>
#include <pthread.h>
#include <unistd.h>

using namespace std;

unsigned long Ack(int m, int n)
{
     while (m != 0)
     {
          if (n == 0) n = 1;
          else n = Ack(m, n-1);
          m = m -1;
     }
     return n+1;
}

static void * Thread1(void* params)
{
     time_t start, finish;
     start = time(NULL);
     for (int m=0; m<5; m++)
     {
          for (int n=0; n<4; n++)
          {
               if (m == 4 && n > 1) break;
               Ack(m, n);
          }
     }
     finish = time(NULL);
     long t = finish - start;
     cout << "Test runtime: " << t << " second(s)" << endl;
     return NULL;
}

static void * Thread2(void* params)
{
     unsigned long elapsed = 0;
```

```
     while (1)
     {
          cout << "Elapsed time: " << elapsed << " sec." << endl;
          elapsed += 5;
          sleep(5);
     }
     return NULL;
}

int main()
{
     cout << "-----------------------------------------" <<
endl;
     cout << "Ackermann CPU Speed test" << endl;
     cout << "Aprox. Runtime: 400 seconds" << endl;
     cout << "-----------------------------------------" <<
endl;
     cout << endl;
     cout << endl;

     pthread_t thread1, thread2;
     pthread_create( &thread1, NULL, Thread1, (void*)0);
     pthread_create( &thread2, NULL, Thread2, (void*)0);

     pthread_join(thread1, NULL);
     pthread_cancel(thread2);

     return 0;
}
```

3. GPIO portok elérése shell környezetből

A RaspberryPi rendelkezik általánosan programozható GPIO kimenetekkel. A kimenetek egy 2x16-os tüskesoron helyezkednek el. A tüskesoron az első láb mellett található a panelen egy P1 jelzés, illetve ha úgy tartjuk a panelt, hogy a kompozit kimenet felfelé néz és jobb oldalon helyezkednek el az USB csatlakozók, akkor a tüskesor bal alsó lába lesz az egyes láb, a felette található pedig a kettes láb. A csatlakozósor lábkiosztása a következő:

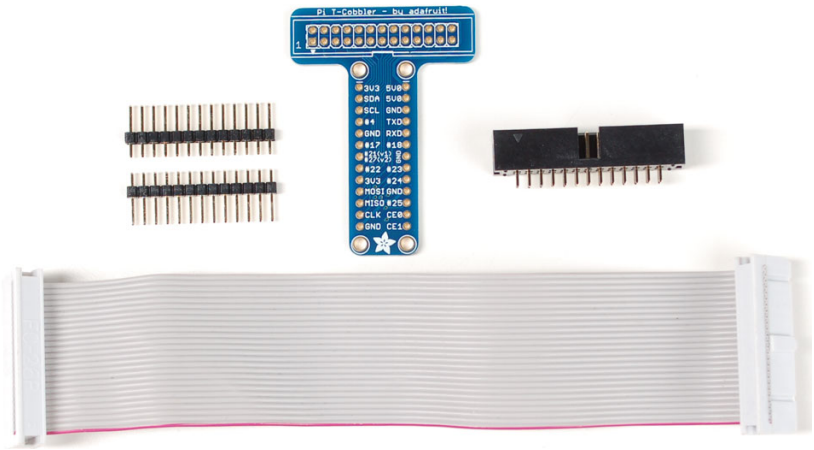


A fenti ábrán látható lábkiosztás a B+ modellre vonatkozik. Az A és B modellek 40 helyett 26 tűs csatlakozóval rendelkeznek. Kompatibilitási célokból az összes modell esetén az első 26 láb funkciója azonos.

A GPIO portok, mint fájl eszközök a /sys/class/gpio/ mappán belül helyezkednek el. A mappa alapértelmezetten két fájt tartalmaz. Egy export és unexport nevezetűt. Ezen két fájl arra szolgál, hogy a beléjük írt számokat exportálja kezelésre a kernel vagy ha már exportálva van a láb, akkor megszünteti a hozzáférési lehetőséget.

Példaképpen készítsünk egy LED villogtató programot. Ehhez a LED anód lábát egy 330 ohm ellenálláson keresztül csatlakoztassuk a 23-as tüskéhez, a katód lábát pedig a 6-os tüskéhez kössük.

Az összekötések létesítését nagymértékben egyszerűsíteni tudjuk egy GPIO összekötő kábel segítségével, mint az Adafruit Pi T-Cobbler



Az összekötés után a szoftveres megvalósítás van már csak hátra. Első lépésként a terminálban root felhasználóra kell váltanunk, mivel a /sys mappa fájljainak módosításához rendszergazdai jog kell. A terminált rendszergazda módba a következő paranccsal tudjuk kapcsolni:

```
sudo su
```

A 23-as tüskén a 11-es GPIO port van kivezetve ezért, hogy használható legyen egy 11-es értéket kell az export fájlba beleírnunk.

```
echo 11 > /sys/class/gpio/export
```

Ez létrehoz egy gpio11 nevű mappát, ami tartalmaz pár fontos konfigurációs fájlt. Természetesen, ha másik számot írunk az export fájlba, akkor is létrejön egy gpio mappa megfelelő lábszámmal, ami ugyan ezeket a konfigurációs fájlokat tartalmazza:

Fájl	***Szerepe***
`direction`	*A port irányát állítja be. Két érték írható bele: in vagy out. Kimenet esetén a láb állapota alacsony jelszintet vesz fel alapértelmezetten.*
`value`	*A port aktuális értékét tartalmazza. 0 érték alacsony szintet jelent, az 1-es érték pedig magas szintet.*
`edge`	*Amennyiben a port megszakítás fogadására képes, akkor a láb él reagálását állítja be. Három értéket vehet fel, melyek: none (nincs), low (lefutó) és high (felfutó).*
`acitve_low`	*A port jelszintjének jelentéseit fordítja meg. Értéke 0 vagy 1 lehet. Alapértelmezetten 0 érékű. 1-es érték esetén az alacsony jelszint jelenti az igaz értéket és a magas jelszint a hamis értéket.*

124. Táblázat: GPIO konfigurációs fájlok

A LED villogtatós példánkban a port értékének kimenetnek kell lennie, így a direction fájlba out értéket kell írni.

```
echo out > /sys/class/gpio/gpio11/direction
```

Ezután kapcsolható a LED, csupán egy egyes értéket kell írnunk a value fájlba.

```
echo 1 > /sys/class/gpio/gpio11/value
```

A kikapcsolása is hasonlóan egyszerű:

```
echo 0 > /sys/class/gpio/gpio11/value
```

Amennyiben végeztünk, a GPIO port exportálása megszüntethető a gpio port számának beírásával az unexport fájlba.

```
echo 11 > /sys/class/gpio/unexport
```

4. GPIO elérése c/c++ programokból

Az előző projektben tárgyalt példán keresztül lehetséges egy olyan C++ kezelő könyvtár létrehozása, amely a megfelelő parancsokat adja ki a portok programozásához. Azonban ezt nem kell megírnunk, mivel volt aki már megtette helyettünk, így a WiringPi könyvtárat fogjuk használni, amely az Arduino függvénykönyvtárának működését utánozza, illetve tovább bővíti a RaspberryPi képességeihez.

A könyvtár alapértelmezetten nincs telepítve, így manuálisan kell telepíteni forráskódból, de szerencsére ez nem egy bonyolult feladat.

A telepítés első lépéseként telepíteni kell a git-core csomagot, amely a GIT verziókezelőt tartalmazza. Ez azért kell, mert a kód fejlesztése GIT tárolók segítségével történik és a fejlesztők ajánlása szerint érdemes minden esetben a legfrissebb változatot használni.

```
sudo apt-get install git-core
```

Ezután a kód a következő parancs segítségével tölthető le:

```
git clone git://git.drogon.net/wiringPi
```

A letöltés végezte után a következő két parancs segítségével fordítható le és telepíthető a könyvtár:

```
cd wiringPi
sudo ./build
```

A fordítás el fog tartani egy ideig. Ha ezzel végzett a gép, akkor tesztelhető is a GPIO portok épsége és a könyvtár működése. Abban a mappában, amelyikben a wiringPi forrása van van egy gpio mappa, amely tartalmaz egy port tesztelő programot, amely a következő parancs segítségével indítható:

```
./pintest
```

A program futtatása előtt el kell távolítani minden perifériát a GPIO tüskékről, mivel a program működését megzavarhatják. Erre a teszt futtatása előtt a program is figyelmeztet.

Ha minden rendben volt, akkor valami hasonló kimenetet kellene produkálnia a programnak:

```
PinTest
=======

This is a simple utility to test the GPIO pins on your revision 2
Raspberry Pi.


NOTE: All GPIO peripherals must be removed to perform this test. This
  includes serial, I2C and SPI connections. You may get incorrect
results
  if something is connected and it interferes with the test.

This test can only test the input side of things. It uses the internal
pull-up and pull-down resistors to simulate inputs. It does not test
the output drivers.

You will need to reboot your Pi after this test if you wish to use the
serial port as it will be left in GPIO mode rather than serial mode.

Please make sure everything is removed and press the ENTER key to
continue,
or Control-C to abort...
```

```
The main 8 GPIO pins  0: 7:  OK
The 4 pins on the P5 connector 17:20:  OK
The 5 SPI pins 10:14:  OK
The serial pins 15:16:  OK
The I2C pins  8: 9:  OK
```

A teszt program futtatása után a GPIO tüskesoron elhelyezkedő soros port használhatatlan lesz, mivel GPIO üzemmódban marad. A rendszer újraindítása után azonban ismét működőképes lesz soros portként.

A könyvtár dokumentációja a http://wiringpi.com/reference/ oldalon található meg, illetve a http://wiringpi.com/examples/ oldalon számos példaprogram található meg.

A WiringPi könyvtár sajátos számozást alkalmaz a GPIO tüskék azonosítására. Ennek oka az, hogy a készítője kompatibilissé akarta tenni a későbbi hardver verziókkal úgy, hogy a programokat ne kelljen módosítani. (Személyes véleményem szerint inkább csak feleslegesen bonyolulttá tette a használatát.) A WiringPi lábkiosztása az alábbi ábrán látható:

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

A rev. 2 Pi modellek rendelkeznek egy extra kimeneti GPIO tüskesorral is, amelyet szintén támogat a WiringPi.

P5: Secondary GPIO connector (Rev. 2 Pi only)							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		5v	1	2	3.3v		
17	28	GPIO8	3	4	GPIO9	29	18
19	30	GPIO10	5	6	GPIO11	31	20
		0v	7	8	0v		
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

Az előző projektben tárgyalt LED villogatós program C++ nyelven, WiringPi használatával:

```
#include <iostream>
#include <wiringPi.h>

using namespace std;

int main ()
{
     cout << "WiringPi pelda" << endl;
     cout << "LED felkapcsolasa es lekapcsolasa 100x" << endl;
     wiringPiSetup () ;

     pinMode (14, OUTPUT);

     for (int i=0; i<100; i++)
     {
          digitalWrite (14, HIGH);
          delay (250);
          digitalWrite (14,  LOW);
          delay (250);
     }
     return 0;
}
```

A program sikeres lefordításosához linkelni kell a WiringPi könyvtárat is a -lwiringPi paraméterrel.

```
g++ WiringPiBlink.cpp -lwiringPi -o WiringPiBlink
```

5. GPIO elérése Python programokból

A GPIO portok python programozási nyelvből is elérhetőek. Az ehhez szükséges modul az RPi.GPIO. Ez alapértelmezetten telepítve van raspbian esetén, azonban ha véletlenül nem lenne, akkor könnyen feltehetjük az alábbi parancs segítségével:

```
sudo apt-get install python-rpi.gpio
```

A modult ezután az alábbi kód segítségével tudjuk használatba venni Python környezetben:

```
import RPi.GPIO as GPIO
```

Ez a modult GPIO osztály néven importálja. Ez után használhatjuk a modul függvényeit, amiből mindösszesen csak négy van.

```
GPIO.setmode(mode)
```

A GPIO lábkiosztási sémát állítja be későbbi függvények számára. A mode helyére két konstans választható. A GPIO.BCM, ami a chip által használt elnevezésnek felel meg. Ezen elnevezéseket használtuk a Shell környezet elérés során is. A másik konstans a GPIO.BOARD, ami a lábkiosztást a tényleges fizikai láb alapján veszi.

```
GPIO.setup(pin, direction)
```

Egy láb irányának beállításra szolgál. Első paramétere a lábat határozza meg, második paramétere pedig a port irányát. A második paraméter helyére két konstans közül választhatunk. A GPIO.IN konstans használatával a láb bemenet lesz, míg a GPIO.OUT használatával kimenet lesz.

```
GPIO.input(pin)
```

Egy láb aktuális állapotát olvassa ki, majd visszatérési értékben visszaadja azt. A függvény visszatérési értéke True, ha a bemenet magas szinten van, False pedig, ha alacsony szinten.

```
GPIO.output(pin, value)
```

Egy láb aktuális állapotát határozza meg. Első paraméter a láb, a második paraméter pedig az állapot, ami két konstans közül kerülhet ki. GPIO.LOW vagy GPIO.HIGH közül választhatunk, a LOW alacsony értéket ír a kimenetre, míg a HIGH magas értéket.

A második paraméter lehet logikai érték is, ebben az esetben a false érték alacsony szintnek felel meg, míg a true érték magas állapotnak.

LED villogtatás

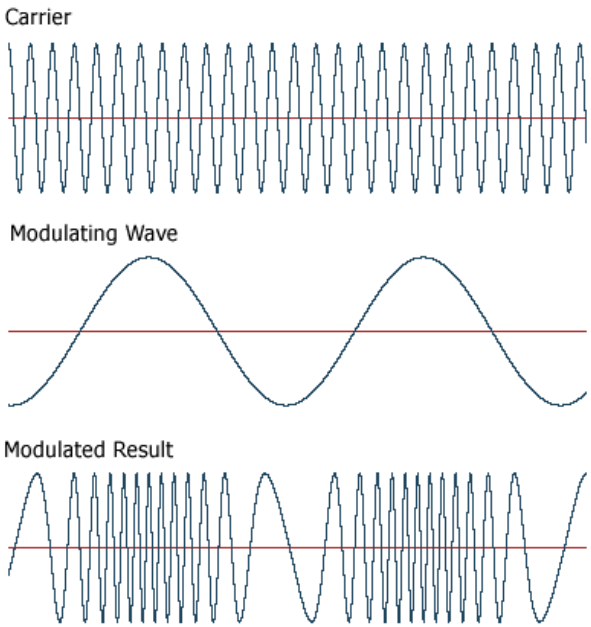
A fentebb ismertetett függvények segítségével elkészíthető az előző projektekben is tárgyalt LED villogtató program Python változata. Jelen kód C++ forráskód python-ra átírt változata.

```
import RPi.GPIO as GPIO
import time

print "Python Pelda"
print "LED felkapcsolasa es lekapcsolasa 100x"

GPIO.setmode(GPIO.BCM)
GPIO.setup(11, GPIO.OUT)

for i in range(0, 100):
      GPIO.output(11, GPIO.HIGH)
      time.sleep(0.250)
      GPIO.output(11, GPIO.LOW)
      time.sleep(0.250)
```

6. FM transzmitter

A RaspberryPI processzora a GPIO4-es lábon képes órajelet generálni külső eszközök meghajtásához. Ezen órajel elérheti a 250MHz-et is, amely bőven elég az FM rádiósáv 87,5 és 108Mhz-es tartományának kielégítéséhez.

Az FM rádió neve a frekvencia moduláció szóból ered. Ezen moduláció úgy működik, hogy a vivő hullám pillanatnyi frekvenciáját módosítja, modulálja a szállítandó jel. Jelen esetben a vivő hullám a generált órajel, a modulációs forrás pedig egy wave fájl.



A program a CD mellékleten megtalálható bináris és forráskódként is, vagy a legfrissebb verzió a http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter címről szerezhető be.

A program futtatásához rendszergazdai jogosultság kell. Erre azért van szüksége a programnak, hogy memória és CPU hatékony tudjon lenni. Ehhez pedig közvetlen hozzáférés kell a rendszer memóriához.

A program használata igen egyszerű.

```
Sudo ./PiFm [wav] [frekvencia] [wav minták]
```

Az első paraméter a lejátszandó, küldendő wave fájl elérési útvonala, a második paramétere a tranzmitter által használt frekvencia, harmadik paramétere pedig a wave fájl mintavételezési frekvenciája. A mintavételezési frekvencia megadása nem kötelező abban az esetben, ha az 22 500Hz.

A wave fájllal kapcsolatban további elvárás, hogy tömörítetlen PCM kódolású hangot tartalmazzon 16 bites mintavételezéssel és egy csatornával.

Ilyen hangforrás előállítására használható az univerzális konvertáló program a FFMPEG. Ez alapértelmezetten nincs telepítve, de könnyen telepíthető a következő parancs segítségével:

```
sudo apt-get install ffmpeg
```

A program telepítése után ezzel konvertálni tudunk fájlokat, de az előre konvertálásra nincs szükség, mivel pipe-on keresztül is átadhatjuk a hanganyagot. Ehhez a következő parancsot kell kiadni:

```
ffmpeg -i be.mp3 -ac 1 -ar 44100 -f wav - | sudo ./PiFM - frek.
44100
```

A fenti parancsban a -i kapcsoló utáni be.mp3 a lejátszandó fájlt határozza meg. A -ac 1 paraméter azt

jelenti, hogy egy csatornára dekódolja a hangot, a -ar 44100 azt jelenti, hogy 44100Hz-es mintavételezésre dekódoljon. A -f wav paraméter a kimeneti formátumot határozza meg, míg a pipe jel előtti kötőjel azt, hogy a képernyőre dekódoljon. Erre azért van szükség, hogy pipe formájában a dekódolt hang átadható legyen a FM továbbbító programnak, amely indító parancsában a kötőjel paraméter azt harázza meg, hogy pipe-ból olvasson a program.

A frek. Argumentum helyére azt a frekvenciát írjuk, amelyen működtetni szeretnénk a tranzmittert.

Hardver követelménye nincs a programnak, azonban ha a hatósugarat növelni szeretnénk pár centiről pár méterre, akkor a GPIO4-es lábra egy 5-20Cm méretű vezetéket kössünk antenna gyanánt.

Nagyobb kimeneti teljesítmény érhető el egy FM erősítő alkalmazásával, amely kapcsolása relatíve egyszerűen beszerezhető az interneten. Azonban mivel bizonyos teljesítmény felett a rádiófrekvenciák használata engedélyköteles ezért itt a könyvben nem szerepeltetek konkrét kapcsolást.

A tranzmitter üzemeltetésekor ügyeljünk arra, hogy lehetőleg olyan frekvenciát válasszunk, amely nem használt egyetlen egy rádióállomás által sem. Ezzel egyrészt csökkentjük az interferenciát, másrészről meg a közvetlen környezetünkben lévőktől nem vesszük el a lehetőséget, hogy meghallgathassák kedvenc rádióadójuk műsorait.

7. Grafikus asztal távoli elérése

SSH protokollon keresztül lehetőségünk van grafikus alkalmazások futtatására is, amennyiben a gépünkön van egy X kiszolgáló. Azonban az X szerver protokollja nem éppen sávszélesség barát, illetve nem is a gyorsaságáról híres. További hátránya a dolognak, hogy az ssh kapcsolat megszűnésekor a programok futtatása is megszakad, ami bizonyos körülmények között nem baj, de ha távoli kiszolgálóként akarjuk üzemeltetni a Pi-t, akkor bizony gond lehet.

Az X szerver és az SSH protokoll ezen nyűgje könnyen kiküszöbölhető egy VNC szerver telepítésével. A VNC távoli asztali elérést tesz lehetővé platformfüggetlenül. Létezik kliens és szerver program is az összes fontos operációs rendszerre.

Raspbian alatt a legegyszerűbben a TightVNC szerver telepíthető apt segítségével:

```
sudo apt-get install tightvncserver
```

A program telepítése után indítható is a szerver a tightvncserver parancs kiadásával. A szerver indításakor megadható egy pár paraméter. Ezek:

- *-geometry*
  képernyőméret megadása pixelekben. Pl.: Az 1280x720 paraméter 1280 pixel széles 720 pixel magas felbontást állít be.
- *-depth*
  Képernyő színmélység meghatározása. Értéke lehet 16 vagy 24
- *-kill*
  Futó vnc szerver leállítása. A leállításhoz meg kell adni a vnc szerver azonosító számát, mivel több is futhat egy gépen. Általában az első vnc szerver 1-es azonosítóval fut. Ennek a leállítására használható formája: -kill :1
- *:*
  Szerver port meghatározása. A portot a kettőspont után kell közvetlenül írni. Érdemes olyan portot megadni, amely használatához nem kell bajlódni a tűzfalon kiengedéssel és engedélyezéssel. Ilyen port például az egyes.

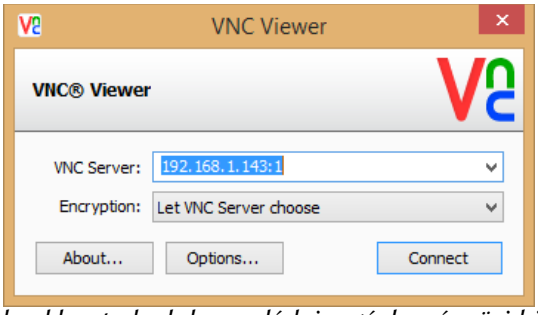
Kliens programnak bármilyen VNC kompatibilis program megteszi. Ilyen program például a RealVNC Viewer, amely elérhető a főbb operációs rendszerekre és igen kis méretű. Letölteni a http://www.realvnc.com/ címről lehet.

A VNC szerver első indításakor kérni fog egy jelszót, amely a szerverhez csatlakozáshoz kell. Ezen jelszó bekérése után szintén első indításkor meg fogja kérdezni, hogy létre szeretnénk e hozni egy csak megtekintésre szóló jelszót. Ez a mód képernyő megosztás engedélyezésére alkalmazható, itt én nem hoztam létre jelszót, mivel nem kívántam élni a lehetőséggel.

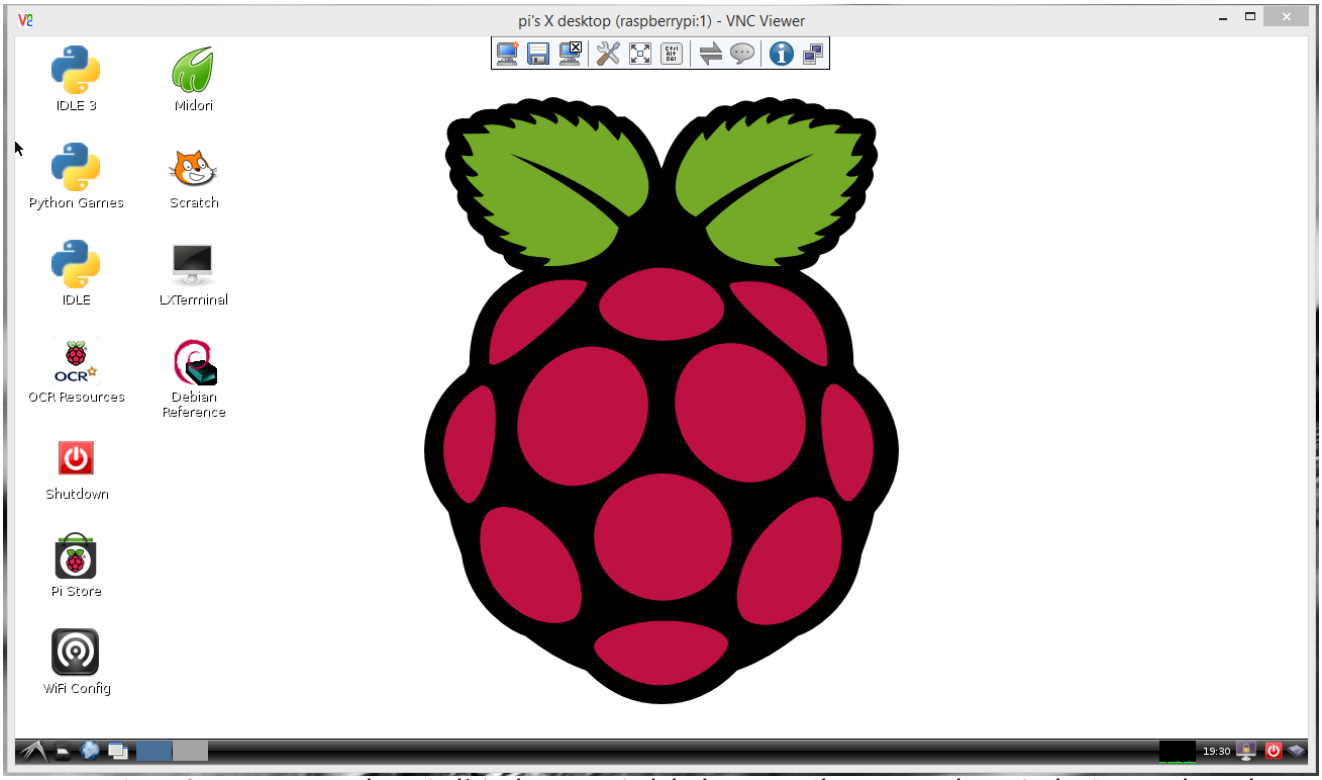
A jelszó és a csak megtekintési jelszó bármikor módosítható. A program amit futtatni kell ehhez a következő:

```
tightvncpasswd
```

A VNC megtekintő alkalmazásunk induláskor egy ablakot ad be, ahol a szerver IP címe mellett a portot is meg kell adni. Ha ezt nem helyesen, vagy nem adjuk meg egyáltalán, akkor kapcsolódási hibát kapunk.



Ha mindent jól csináltunk, akkor tudunk kapcsolódni a géphez és rövid időn belül az asztal képét kellene látnunk, miután megadtuk a megfelelő jelszót.



A VNC szerver a rendszer indításakor nem indul el automatikusan, azonban rávehető a rendszer, hogy indítsa el. Ehhez létre kell hozni egy indító shell szkript fájlt. A szkript fájlt a /etc/init.d mappában kell létrehozni. Ennek én a vncserver nevet adtam. A fájl tartalma:

```
#! /bin/sh
# /etc/init.d/vncserver

USER=root
HOME=/root

export USER HOME

case "$1" in
 start)
     echo "Starting VNC Server"
     /usr/bin/vncserver :1 -geometry 1280x720 -depth 24 -pixelformat
     ;;
 stop)
     echo "Stopping VNC Server"
     /usr/bin/vncserver -kill :1
```

```
     ;;
 *)
     echo "Usage: /etc/init.d/vncserver {start|stop}"
     exit 1
     ;;
esac


exit 0
```

A fájlt futtathatóvá kell tenni, ezért a chmod paranccsal módosítani kell a jogosultságokat:

```
sudo chmod 755 /etc/init.d/vncserver
```

Ezután már csak engedélyezni kell azt, hogy automatikusan induljon a betöltő program. Ehhez az alábbi parancsot kell lefuttatnunk.

```
sudo update-rc.d /etc/init.d/vncboot defaults
```

Ha a fenti parancs hibával ér véget, akkor az azt jelenti, hogy nem tudja olvasni az állományt, amit megadtunk. A hiba bekövetkezése attól függ, hogy melyik mappában vagyunk. Ha hibába ütköznénk, akkor az alábbi parancsot futtassuk:

```
sudo update-rc.d vncboot defaults
```

A grafikus felület futtatásához a Pi esetén 32Mb videó memória javasolt, azonban komolyabb használat mellett a 64Mb memória is elkel. A grafikus processzor által használható videó memória méretét a raspi-config programban lehet módosítani.

8. Webmin telepítése

Egy Linux szerver esetén a kezdők számára a legnagyobb kihívást a rengeteg konfigurációs fájl manuális szerkesztgetése, módosítása jelenti. Ezen problémán kíván segíteni a webmin nevezetű program, amely segítségével könnyen adminisztrálhatóak szerver szolgáltatások, akár távolról is.

A Webmin egy böngészőben elérhető konfigurációs felületet ad a szerverünknek, ahonnan távolról is tudjuk adminisztrálni. Ez önmagában egy komoly biztonsági problémát jelent, főleg akkor, ha a távoli eléréshez használt jelszó és felhasználónév nem elég erős.

Éppen ezért a webmin állandó futtatása nem javasolt, csak igény szerint legyen elindítva, ezzel minimalizálva a biztonsági kockázatokat.

A webmin telepítése előtt telepíteni kell pár csomagot, amelyek szükségesek a program megfelelő működéséhez. A szükséges csomagokat a következő parancs segítségével tudjuk letölteni és telepíteni:

```
sudo apt-get install perl libnet-ssleay-perl openssl libauthen-
pam-perl sudo apt-get install libpam-runtime libio-pty-perl
sudo apt-get install apt-show-versions libapt-pkg-perl
```

A függőségi csomagok telepítése után magát a webmin programot kell telepítenünk. Ez nem része az alap APT telepítőrendszernek, manuálisan a http://webmin.com/download.html címről kell letölteni a legfrissebb verziót. Több formátumban is letölthető a telepítő csomag, a .deb változatot kell letölteni.

A telepítő letöltése után telepíteni a csomagot a következő parancs segítségével tudjuk:

```
dpkg -i webmin.deb
```

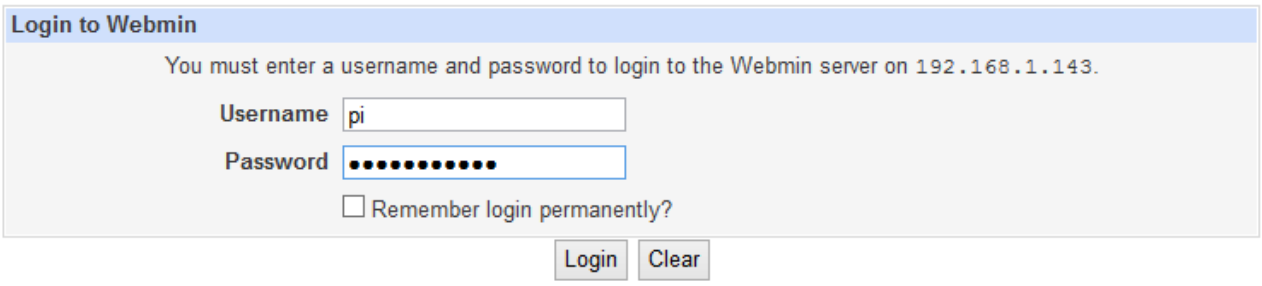
Ahol a webmin.deb a letöltött deb telepítő teljes neve. A telepítés az SD kártya sebességétől függen el fog tartani egy ideig, de ha minden rendben ment akkor egy ilyen üzenetet kellene kapnunk a telepítés végén:

```
Webmin install complete. You can now login to https://raspberrypi:10000/
as root with your root password, or as any user who can use sudo
to run commands as root.
```

Ezen üzenet megadja, hogy a konfigurációs felületet milyen címen tudjuk elérni, illetve azt, hogy bármely olyan felhasználóval bejelentkezhetünk, amely sudo parancs használatára jogosult. A jelszó természetesen meg fog egyezni az SSH és rendszer bejelentkezéskor használandó jelszóval.

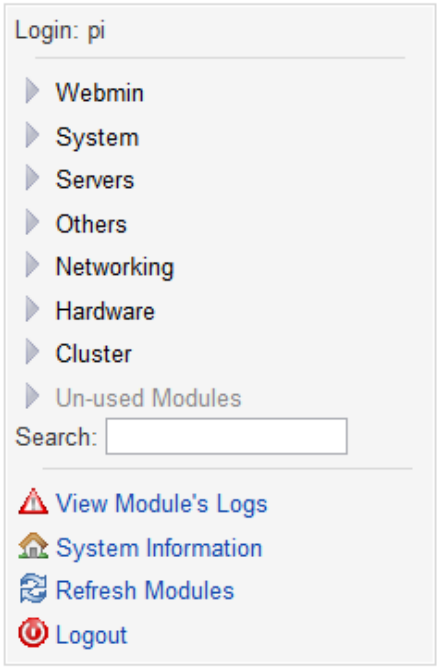
Amennyiben a címet a böngészőbe beírva „A Kiszolgáló nem található" hibába botlunk, akkor a címben szereplő raspberrypi helyére írjuk be a szerver IP címét. (A hiba Windows kliens gépek esetén jelentkezik, kikerülhető a gond Samba csomagok telepítésével, amelyek megoldják a két rendszer közötti DNS problémát)

Első csatlakozáskor a böngésző figyelmeztetni fog minket, hogy a kapcsolat nem megbízható, vagy hogy a webhely tanúsítványa nem hiteles. Ennek oka az, hogy a konfigurációs felület saját maga számára írta alá a titkosító kulcsot. Böngésző függően erősítsük meg a biztonsági kivételt, ezután a bejelentkező felületen lyukadunk ki, ahol az ssh belépés során használt felhasználónévvel és jelszóval bejelentkezhetünk.

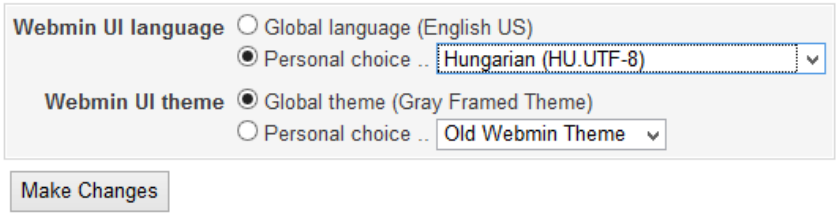


A bejelentkezés után a főképernyőn a szerver állapotáról kapunk egy visszajelzést, ami igen hasznos. A bal

oldali menüből érhetjük el a rendszer funkcióit.



- ***Webmin***:
  Ebben a menüpontban tudjuk beállítani a webmint. Legfontosabb menüpontja elsősorban a Nyelv és téma beállítására szolgáló Change Language and theme menüpont. Itt a Personal Choise mellé kell tenni egy pipát, majd kiválasztani a Hungarian, magyar nyelvet. Majd a Make changes gombra kattintva a folyamat végesztével és az oldal újratöltése után a menürendszer átvált magyarra többé-kevésbé. Ennek oka az, hogy a magyar fordítás nem teljes még, így elég sok helyen találkozhatunk angol szövegekkel.



- ***Rendszer***
  Ebben a menüpontban a rendszer adminisztrációja végezhető el, mint a fájlrendszerek csatolása, futó programok leállítása, szoftverek telepítése és frissítése és még sok egyéb.
- ***Szerverek***
  Ebben a menüben jelennek meg a telepített szerverek, innen lehet őket adminisztrálni. Alapesetben csak az SSH van telepítve.
- ***Egyebek***
  Itt vegyes eszközök találhatóak, mint fájlkezelő, vagy SSH terminál. Vagyis olyan eszközök, amelyek más kategóriákba nem fértek bele.
- ***Hálózat***
  Hálózati beállítások elvégzésére alkalmas menüpont, piszkálására nem igen lesz szükségünk.
- ***Hardver***
  Hardver elemek konfigurálására, beállítására szolgáló menüpont. Itt tudunk nyomtatókat hozzáadni, valamint a partíciókat szerkeszteni. Különösen akkor lesz hasznos, ha a Pi gépünket nyomtató szerverré

akarjuk varázsolni.

- ***Cluster***

  Itt további webmint futtató szervereket tudnánk adminisztrálni, amelyek helyi hálózaton vannak.

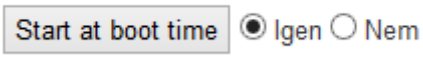
- ***Nem használt modulok***

  Ebben a menüpontban fognak megjelenni azon szerver programok amelyek nincsenek még telepítve, ezért nem adminisztrálhatóak. A kiválasztott szerver innen telepíthető egy kattintással. Ezután a modulja átkerül a szerverek fül alá, ahonnan adminisztrálhatóvá válik.

A konfigurációk végeztével érdemes leállítani a webmin felületet. Ehhez az alábbi parancsot kell kiadni:

```
sudo /etc/init.d/webmin stop
```

Az alap telepítés beállításában újraindítás után is automatikusan el fog indulni a webmin felület. Ez egyrészt biztonsági kockázat, valamint értékes pár megabyte memória spórolható, ha csak igény szerint használjuk. Ezért érdemes letiltani az alapértelmezett indulását. Ezt a webmin panelen a webmin kategória webmin beállítások menüpontban tudjuk megtenni. A lap alján található egy Start at boot time jelzés, amely segítségével kikapcsolható ezen funkció.



Ha kikapcsoltuk a felületet, akkor bármikor elindíthatjuk a webmin felületet az alábbi parancs kiadásával:

```
sudo /etc/init.d/webmin start
```

9. GIT szerver telepítése

A könyv elején volt némi szó a verziókezelő rendszerekről. Otthoni fejlesztésű projektjeink kapcsán célszerű alkalmazni egy verziókezelő rendszert, főleg, ha többen is dolgoznak ugyan azon a forráskódon.

A GIT verziókezelőre azért esett a választás, mivel egyre népszerűbb, illetve jóval gyorsabb más megoldásoknál, valamint igen egyszerű a telepítése, ami nem utolsó szempont.

Szerver oldalon két csomagot kell csupán telepíteni:

```
sudo apt-get install git git-core
```

A csomagok telepítése után létre kell hozni egy felhasználót, aki be tud lépni a szerverre és hozzáférhet a GIT tároló tartalmához. A külön felhasználó létrehozása biztonsági okok miatt kell. A felhasználó neve tetszőleges, jelen esetben a git nevet választottam.

```
sudo adduser --system --shell /bin/bash --gecos 'git verzió
kezelés' --group --home /home/git git
```

A felhasználó létrehozása után be kell állítani a jelszavát, amihez az alábbi parancs használható:

```
sudo passwd git
```

Ezután át kell váltanunk a git felhasználóra, majd létrehozni a GIT tárolót, amibe tudunk írni. A váltáskor szükség lesz az előzőleg beállított jelszóra.

```
su git
```

A szerveren a tároló mappaneveknek .git kiterjesztéssel kell végződniük. Az alábbi példa a pelda.git nevezetű tárolót hozza létre, amibe tudunk majd kódot publikálni.

```
mkdir pelda.git
cd pelda.git
git --bare init
```

Ezután a szerver beállítása kész is. A helyileg tárolt GIT tárolónk szerverre exportálásához először is társítani kell a tárolóhoz egy távoli szervert. Ehhez a tárolóban az alábbi parancsot kell kiadni:

```
git remote add [szerver] git@[ip]:pelda.git
```

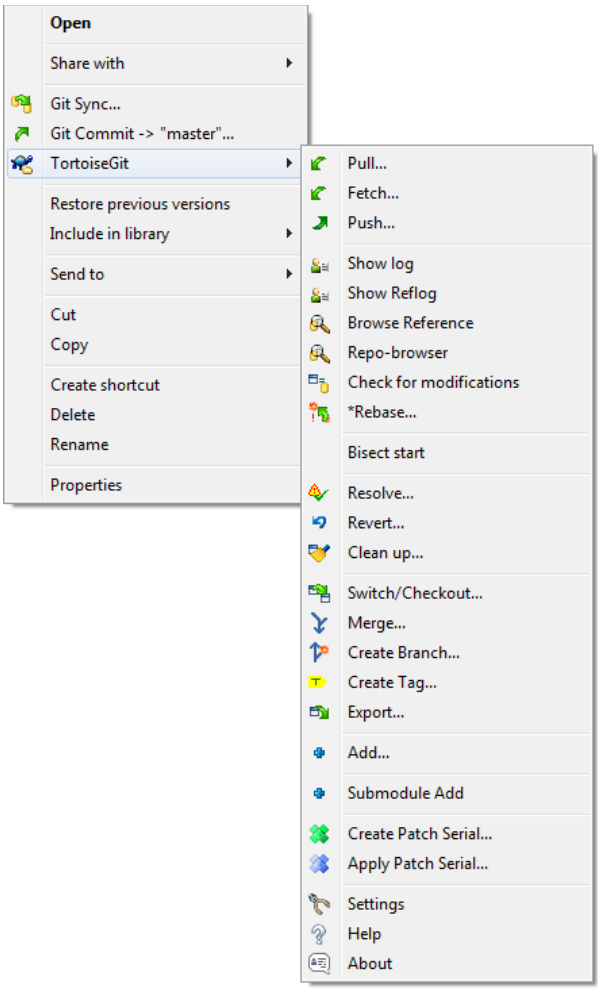
A parancssorban a [szerver] mező a távoli szerver nevének azonosítására szolgáló mező, ez tetszőleges. Az [ip] mező a szerver IP címe, míg a pelda.git a használandó tároló neve. Ezen lépés után exportálható a kód szerverre az alábbi parancs segítségével:

```
git push [szerver] [ág neve]
```

Kód letöltéséhez pedig a következő parancs használható:

```
git pull [szerver] [ág neve]
```

Windows-ra létezik egy Shell beépülő modul, ami megkönnyíti a GIT tárolók kezelését. Az eszköz neve TortoiseGit. Letölteni a http://code.google.com/p/tortoisegit/ címről lehet. Működéséhez szükség lesz a GIT Windows változatára is. Ez a http://code.google.com/p/msysgit/ címről szerezhető be.



Amennyiben a GIT szerverünket komolyan használni szeretnénk, akkor mindenképpen ajánlott egy merevlemezzel üzemeltetni a szervert, mivel az SD kártyák nem igen tolerálják a sűrű újraírási kérelmeket, valamint nem biztos, hogy kis fájlok esetén is tűrhető átviteli sebességet kapunk.

10. USB tárolók, merevlemezek kezelése

Amennyiben nem grafikus felületen keresztül használjuk a rendszert, akkor a csatlakoztatott USB tárolóeszközök alapértelmezetten nem csatolódnak a rendszerbe. Ezeket manuálisan kell csatlakoztatnunk.

Tároló, különösen merevlemez csatlakoztatása előtt érdemes beszerezni egy USB HUB-ot, amely elegendő árammal tudja ellátni az eszközt. HUB alkalmazása nélkül előfordulhat, hogy a rendszer nem fogja felismerni az eszközt, illetve az is előfordulhat, hogy nem megfelelően fog működni.

A csatlakoztatás után meg kell tudni a lemez hardver azonosítóját ezt az alábbi parancs kiadásával tudjuk lekérdezni:

```
ls /dev/sd*
```

Amennyiben csak egy lemezt csatlakoztattunk, akkor valószínűleg csak egy sda1 bejegyzésünk lesz. A kis a jelzés a lemezt azonosítja a-tól z-ig, a szám pedig a partíciót. Értelem szerűen, ha több partíció is található az eszközön, akkor több bejegyzést is kapunk.

Csatlakoztatni a lemezt a mount parancs kiadásával lehet. Azonban ennek a használata előtt létre kell hozni egy mappát, amibe csatoljuk majd a fájlrendszert. Erre a célra én a /media mappán belül csináltam egy usb mappát:

```
sudo mkdir /media/usb
```

Ezután a fájlrendszert a mount paranccsal csatlakoztatni a következőképpen lehet:

```
sudo mount -t vfat /dev/sda1 /media/usb
```

A parancsban a -t kapcsoló a fájlrendszer meghatározására szolgál. FAT partíciók esetén a vfat típus alkalmazandó, NTFS kötetek esetén pedig az ntfs kulcsszó. A FAT fájlrendszerek írásra olvasásra támogatottak, míg az NTFS csak olvasásra alapértelmezetten. Továbbá fájlrendszertől függetlenül csak a rendszergazdának lesz írási jogosultsága az eszközre, amennyiben ezt a fájlrendszer támogatja.

A műveletek végeztével a fájlrendszer az umount paranccsal választható le. Itt paraméternek a csatolt fájlrendszer elérését kell megadni, szintén rendszergazdaként futtatandó. Adatvesztés elkerülése érdekében a leválasztás előtt érdemes kiadni legalább kétszer öt másodperces különbséggel a sync parancsot, ami lemezre írja a még függő műveleteket.

```
sync
sync
sudo umount /media/usb
```

Partícionálás

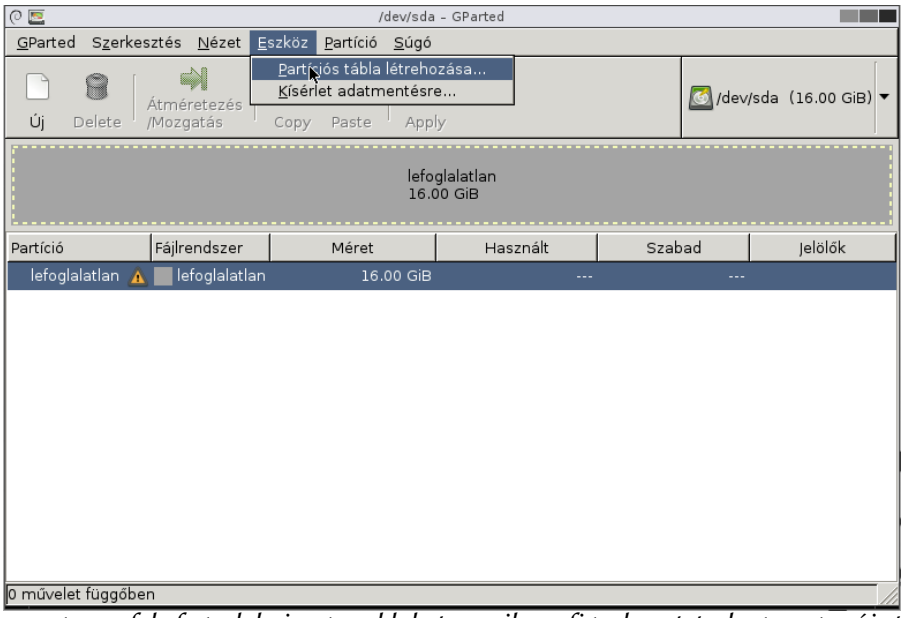
Amennyiben szerverként használjuk a Pi lapunkat és tartósan merevlemezzel bővíteni szeretnénk, akkor particionálnunk és formáznunk kell a lemezt. A particionálást grafikus felület segítségével érdemes elvégezni, ekkor használhatjuk a remek gparted particionálót. Ez ha nem lenne telepítve, akkor könnyen telepíthető az alábbi parancs kiadásával:

```
sudo apt-get install gparted
```

A particionálás elvégezhető bármilyen gépen, mivel a gparted a legtöbb cd-ről is futtatható Linux disztribúció része.

A gparted használata igen egyszerű. Első lépésképpen a jobb oldalon található lenyíló menüből ki kell választani azt a lemezt, amit szabni szeretnénk. A program ugyanis automatikusan az első merevlemezt jeleníti meg indításkor. Amennyiben a lemezünk még sosem volt használva, akkor nincs rajta partíciós tábla. Ebben az esetben

ahhoz, hogy partíciókat tudjunk tenni a lemezre, létre kell hoznunk először a partíciós táblát. Ezt az eszköz menüpont alatt található partíciós tábla létrehozása menüpontban tehetjük meg.



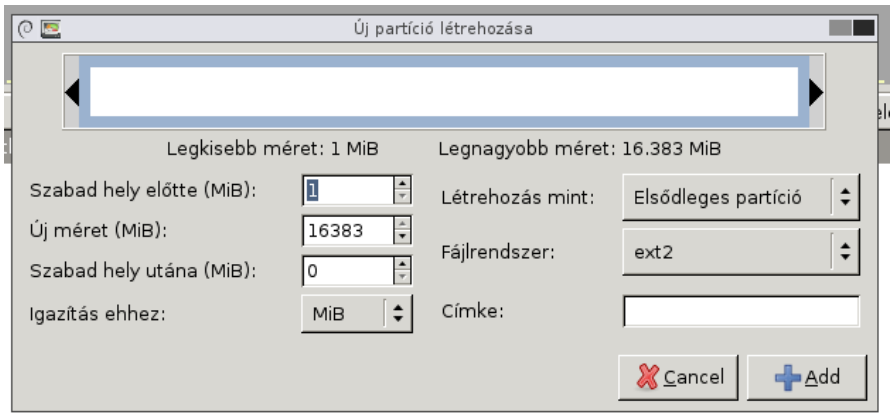
Ekkor a program fel fog dobni egy ablakot, amiben figyelmeztet, hogy egy új tábla létrehozása adatvesztéssel jár. A speciális részt lenyitva lehetőségünk van partíciós tábla rendszert választani. A viszonylag hosszú listából nekünk az msdos és GPT jelzésűek fontosak.

Az MS-DOS típusú partíciós tábla jelenleg a leggyakoribb. Ezen táblában maximum 4db elsődleges partíciót definiálhatunk és korlátlan számú logikai meghajtót egy kiterjesztett partícióban. Rendszert indítani közvetlenül csak elsődleges partícióról lehet. Ezen táblával maximum 2Tb méretig kezelhetünk lemezeket.

2Tb felett GPT táblát érdemes használni. GPT tábla esetén a partíciók számában nincs megkötés, és nincs is a típusuk megkülönböztetve, mivel ekkor minden partíció elsődleges típusú. Ezen táblaformátum hátulütője annyi, hogy a korábbi Windows rendszerek (Windows XP) nem fogják ismerni és kezelni a lemezt.

Ha kész a partíciós tábla, akkor a lemezen elkezdhetünk partíciókat létrehozni, az új gomb megnyomásával. A partíció létrehozása ablak felépítése igen egyszerű. Meg kell adni a partíció méretét, típusát, fájlrendszerét, és kezdő helyét a lemezen. A kezdő hely meghatározásáért a szabad hely előtte mező felelős. Az itt megadott méretek ténylegesen Mb méretben értendőek. Ismétlésképpen: 1Gb = 1024 Mb, 1 Mb = 1024Kb, 1kb = 1024 byte, 1 byte = 8 bit.

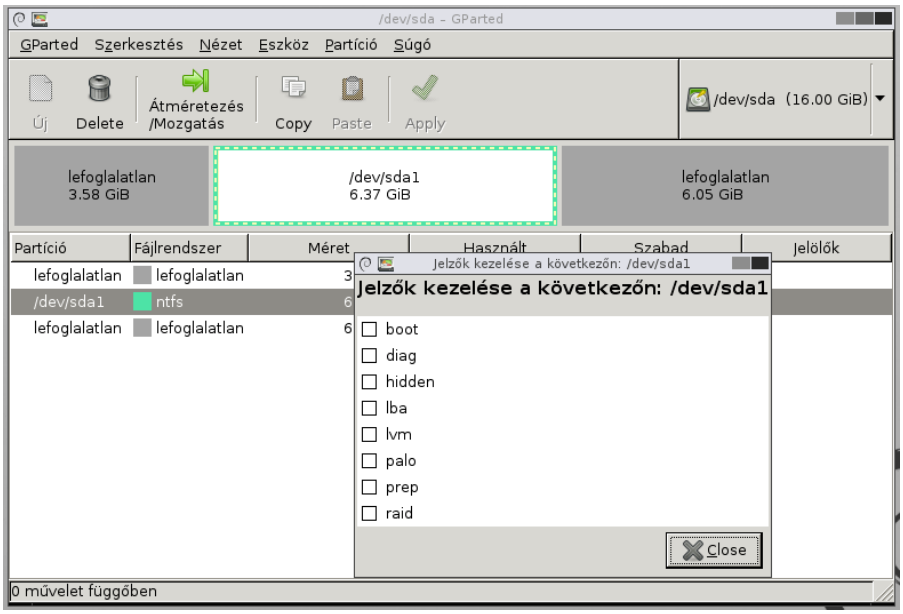
A partíció fájlrendszerének megadásával a gparted automatán formázza is a partíciót (gyors formázás üzemmódban). Itt megjegyezném, hogy bár tud NTFS köteteket kezelni Windows alatt, biztos, ami biztos alapon átszoktam azért formázni a partíciókat. Linux esetén fájlrendszernek az Ext4 típust érdemes választani.



A partíciók megadása után a fő ablak eszköztárán kattinthatóvá vált apply (alkalmaz) gombra kattintva

kapunk ismét egy figyelmeztetést, hogy adatvesztés is történhet és biztos szeretnénk e ezt. A művelet befejeztével kész az új partíciónk.

A partíciók kezelésénél fontos megjegyezni a jelzők (flag-ek) szerepét. A jelzők szerkesztése ablakhoz a partícióra jobb gombbal kattintás után megjelenő helyi menüből jutunk. A jelzők közül számunkra a legfontosabb kettő jelen esetben a boot és a hidden. A boot jelzővel megjelölt partícióról fog megkezdődni a számítógép indításakor a betöltés. Ezt csak elsődleges partícióra tudjuk alkalmazni. A hidden jelző azt jelzi az operációs rendszer felé, hogy a partíció rejtett. Akkor lehet hasznos ezen jelző, ha a futtatni kívánt második operációs rendszer elől szeretnénk rejteni tartalmat, vagy akkor, ha a rendszer betöltője nem támogat más típusú partíciókról indulást.



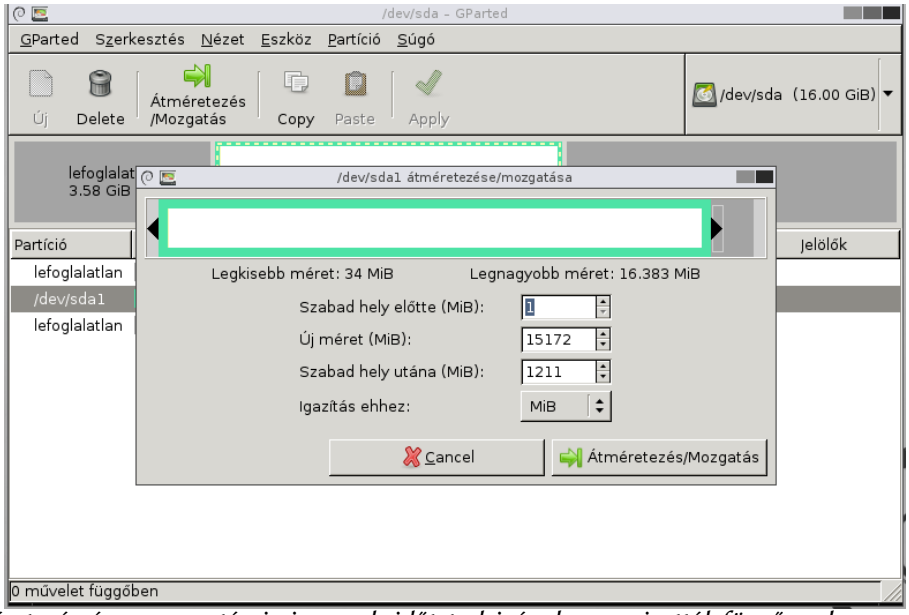
A RaspberryPi esetén a boot flag-nek nincs szerepe, mivel a lap operációs rendszert csak memóriakártyáról tud futtatni.

Amennyiben a meghajtón már vannak partíciók és adatok, valamint nem szeretnénk az egészet felhasználni adatok tárolására, akkor érdemes valamelyik partíciót átméretezni kisebbre és a felszabaduló helyen létrehozni egy új partíciót.

Partíció átméretezése, mozgatása

Az átméretezés és mozgatás műveletek hasonlóan egyszerűen működnek. Itt kiemelném, hogy partíció átméretezésénél és mozgatásánál kiemelten fontos a biztonsági mentés készítése a mozgatott, átméretezett partíció tartalmáról. Ha nem szeretnénk sokat várni az átméretezés vagy mozgatás művelet befejeződésére, akkor erősen ajánlott töredezettség mentesíteni a partíciót, mielőtt belevágunk a műveletbe. SSD esetén a meghajtó kímélése érdekében jobban járunk egy teljes mentéssel, majd egy teljes újraosztással, mivel ez kevesebb írási művelettel jár.

Az átméretezés/mozgatás ablak hasonló az új partíció létrehozása ablakhoz. Itt azonban csak a méreteket kell beállítanunk, semmi mást. A beállítások elvégzése után figyelmeztet a rendszer arra, ha a partíción van operációs rendszer, akkor előfordulhat, hogy a mozgatás/átméretezés után indíthatatlan lesz a rendszer. Ez főleg akkor fordul elő, ha a lemezen eltoljuk a partíció helyzetét. Az indíthatatlanná vált rendszer nem jelenti azt, hogy a rendszert teljes egészében újra kell telepíteni. Ekkor elég csak a rendszerbetöltőt újratelepíteni.



Az átméretezés és a mozgatás is igen sok időt tud igénybe venni attól függően, hogy a lemez mennyire töredezett és mekkora sebességre képes a gép.

Partíció csatolása rendszerindításkor

A Kész partíciónkat ha indításkor is csatlakoztatni szeretnénk a rendszerbe, akkor létre kell hoznunk egy mappát, ahova csatolódik.

Ezután a /etc/fstab fájl tartalmát kell módosítanunk. Ez a fájl tárolja az indításkor csatolandó partíciókat. Módosítások nélkül a tartalma a következő:

```
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p1  /boot           vfat    defaults          0       2
/dev/mmcblk0p2  /               ext4    defaults,noatime  0       1
# a swapfile is not a swap partition, so no using swapon|off from here
on, use  dphys-swapfile swap[on|off]  for that
```

A partíciók adatai egy táblázatban tárolódnak. Az első oszlop a fájlrendszert tartalmazó eszköz fájljának elérési útvonala. A második oszlop a csatolási pontot határozza meg. A harmadik oszlop a fájlrendszer típusát határozza meg. Ext4 esetén ez nem meglepő módon ext4 lesz.

A negyedik oszlop a csatolási beállításokat tartalmazza. Ezekkel bizonyos célokra optimalizálhatóak különböző módon a fájlrendszerek. Merevlemez esetén a defaults jelző bőven elég, SSD vagy Flash meghajtók esetén érdemes kiegészíteni egy noatime jelzővel is, amely csökkenti az írási műveletek számát azáltal, hogy nem tárol a fájlokhoz utolsó módosítás dátumot.

Az ötödik oszlopnak jelen alkalmazás mellett nincs különösebb jelentősége. Ezért ide 0 írandó. Egyébként azt határozza meg, hogy az adott fájlrendszer melyik másik fájlrendszerre biztonsági menthető a dump parancs segítségével.

A hatodik oszlop azt határozza meg, hogy az fsck program milyen sorrendben ellenőrizze a lemezeket egy hirtelen leállás után. Jelen esetben a csatolandó merevlemezünk esetén ennek a számnak 3-nak kell lennie. Amennyiben valamelyik másik számmal ütközik a szám, az azt jelenti, hogy mind a két lemez ugyan olyan prioritással bír, ezért véletlenszerűen fog a kettő közül választani elsőt.

Egy USB merevlemez esetén, amely egy ext4 partíciót tartalmaz a hozzáadandó sor ehhez lesz hasonló:

```
/dev/sda1       /hdd            ext4    defaults          0       3
```

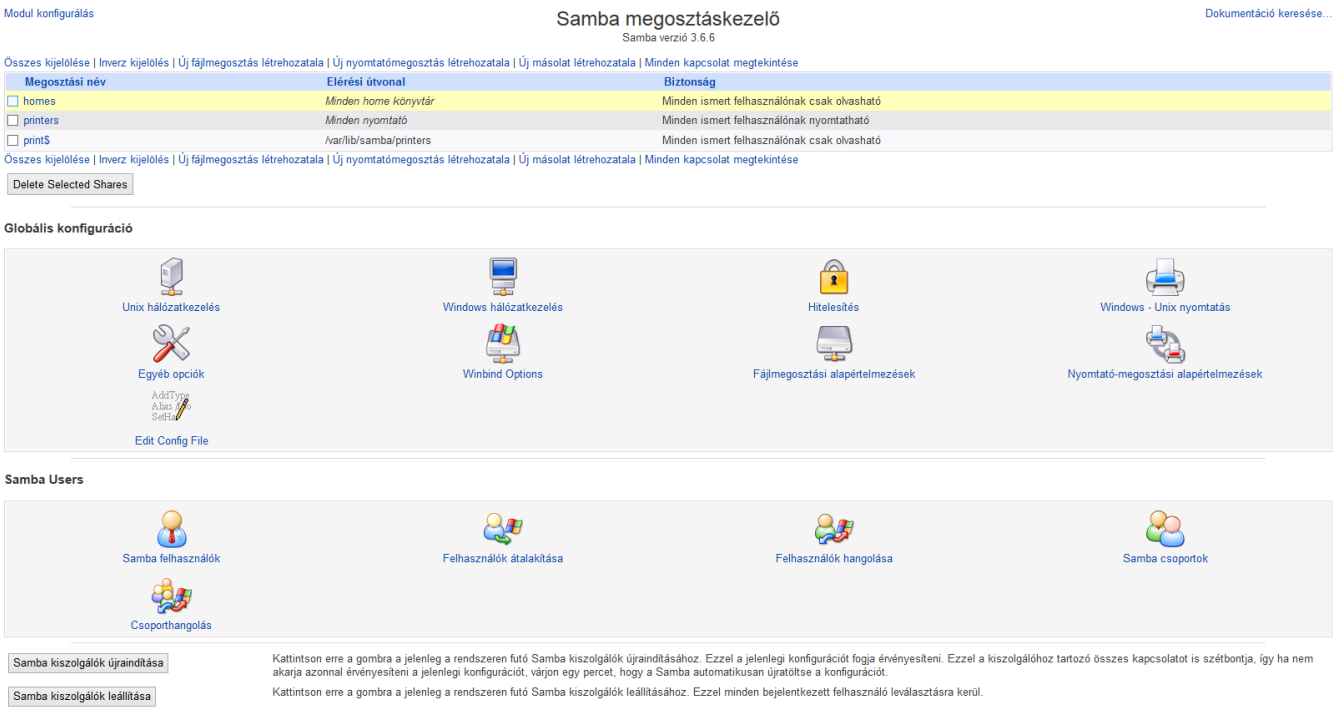
11. Windows fájl és nyomtató megosztás

Az előző projektben a Pi-hez hozzáadott merevlemezt kényelmes lenne Windows fájl és nyomtató megosztás szolgáltatással elérni, mivel az SCP kicsit lassúcska, valamint kliens program kell hozzá Windows alatt, így egy ilyen környezetben nem a legalkalmasabb fájlok fel és letöltésére a szerverünkre.

A projekt megvalósítás feltételezi, hogy a Webmin adminisztrációs felület fel lett telepítve, mivel ezen felület segítségével pillanatok alatt konfigurálható a szükséges szerver.

A szükséges szerver a Samba nevet viseli. Webmin felületen a nem használt modulok között Samba Windows File Sharing néven lehet megtalálni. A modul által igényelt csomagok az adminisztrációs lapjáról könnyen feltelepíthetőek. A letöltés és a telepítés végeztével a modul átkerül a szerverek közé.

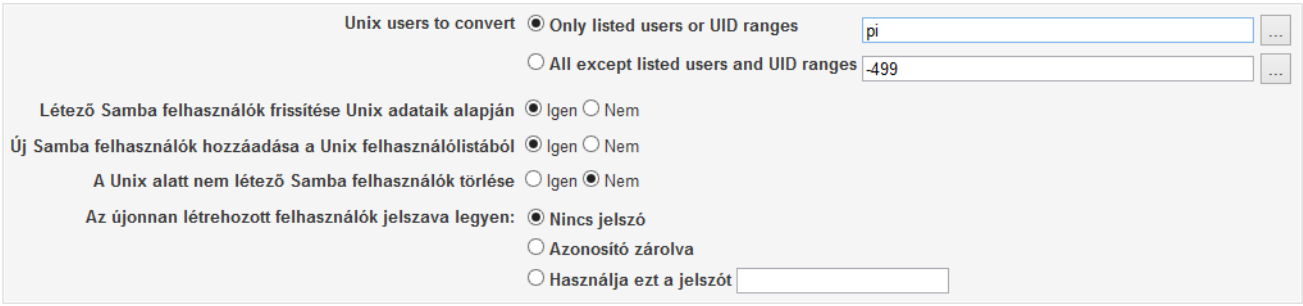
A szerver adminisztrációs felületén könnyen megtekinthetőek és kezelhetőek az aktuális megosztások.



A megosztás első lépése, hogy konfiguráljunk egy vagy több felhasználót, akik tudják használni a Samba szerverünket. Erre azért van szükség, mert egyrészt biztonságosabb tőle a szerver, másrészről pedig a Windows Vista-nál újabb rendszerek nem igen szeretnek felhasználónév és jelszó nélkül csatlakozni megosztásokhoz.

Alapértelmezetten unix felhasználókat kell létrehoznunk, majd ezeket a felhasználókat tudjuk átkonvertálni olyan felhasználókra, akik ténylegesen is hozzáférnek a megosztáshoz.

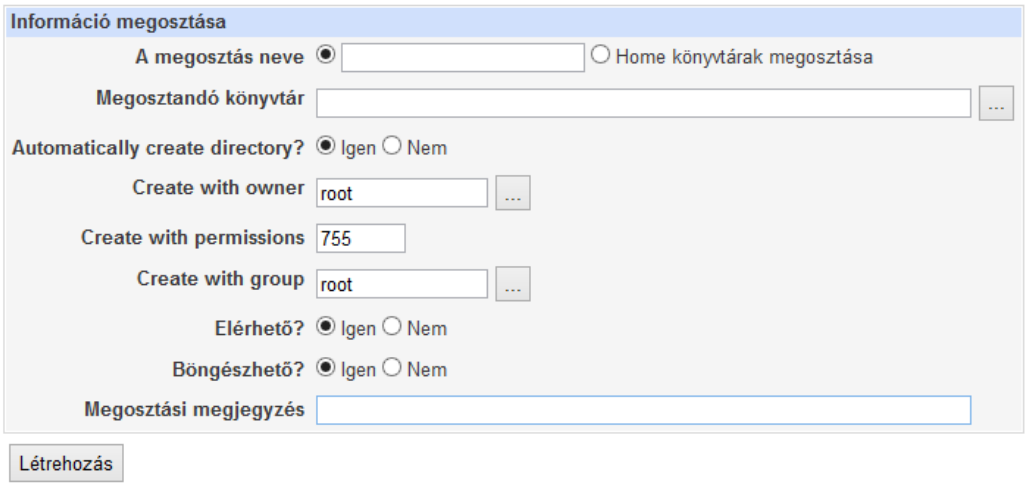
A felhasználók adminisztrációja a Rendszer menü felhasználók és csoportok menüpontja alatt érhető el. Ezután a létrehozott felhasználó a Samba konfigurációs felület felhasználók átalakítása menüpontja segítségével léptethető elő samba felhasználóvá.



Egyszerre több felhasználó is átalakítható. Ezeket megadhatjuk UID vagy név alapján. Jelen esetben az alapértelmezett felhasználót léptetem elő. A jelszó mező maradhat üresen, bár célszerű megadni egy jelszót. Ez később a kliens gépeken elmenthető.

A felhasználó létrehozása után jöhet a megosztás létrehozása. Megosztani csak létező mappát tudunk. Amennyiben még a mappánk nem létezik és most hozzuk létre, akkor ügyeljünk arra, hogy a mappa tartalma írható és olvasható legyen azáltal a felhasználó által, akit kitüntettünk a Samba használatára.

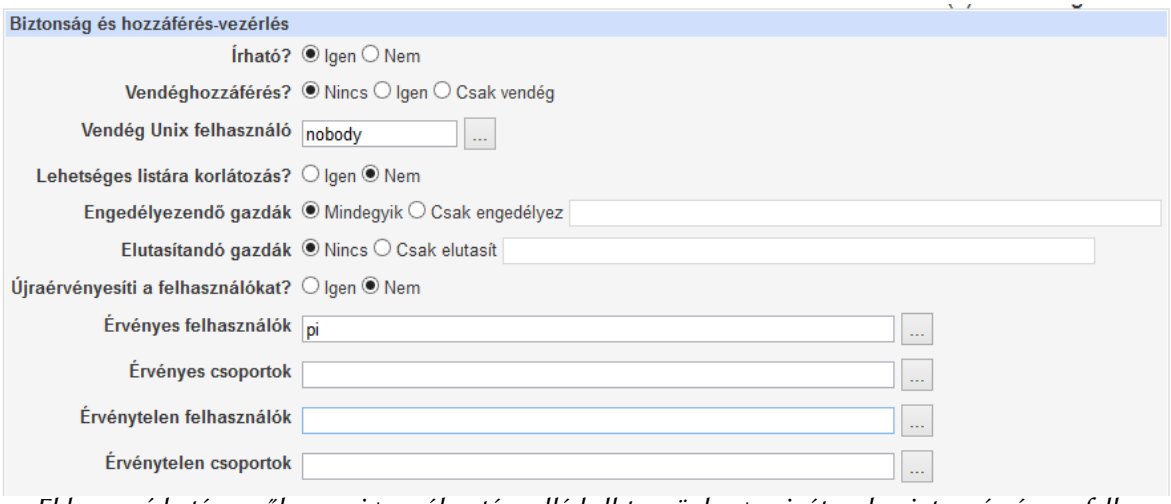
Megosztás a megosztások alatt található új megosztás létrehozása menüponttal hozható létre.



Itt meg kell adni a megosztás nevét, illetve ki kell választani a megosztandó mappát. Amennyiben ez a mappa még nem létezne, akkor létre is hozható, illetve a csoport és tulajdonos, valamint a jogok is beállíthatóak.

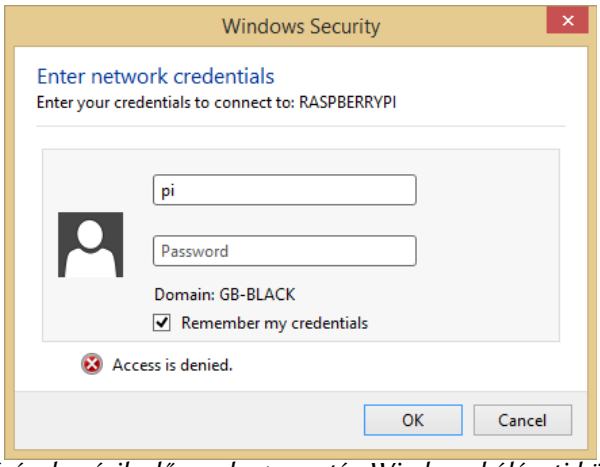
A megosztási megjegyzés a mappa rövid szöveges leírásának megadására szolgál, kitöltése nem kötelező.

Ezután már csak a felhasználót kell hozzárendelni a megosztásunkhoz, amely létrejötte után a megosztás módosítása lapra kerülünk. Itt az első opció a biztonság és hozzáférés-vezérlés. Ezt megnyitva különböző dolgokat tudunk beállítani. Számunkra jelen esetben a legfontosabb, hogy adatokat tudjuk írni a szerverre.



Ehhez az írható mezőben az igen választó mellé kell tennünk egy pipát, valamint az érvényes felhasználók közül ki kell választani azt a felhasználót, aki írhatja a megosztást.

Amennyiben a vendég hozzáférést nem engedélyezzük, akkor látható lesz a megosztásunk (a tartalma nem) ugyan, de felhasználónév és jelszó hiányában nem fog hozzácsatlakozni senki.



A samba csomag telepítésének másik előnye, hogy ezután Windows hálózati környezetben is elérhető lesz a Pi a DNS neve alapján, ami RASPBERRYPI. Így a továbbiakban nem kell az IP címével bajlódnunk.

Nyomtató megosztás

A nyomtató megosztás is hasonlóan egyszerű feladat. Ez két lépésből áll. Először létre kell hozni egy Unix nyomtatót, majd a létrehozott nyomtatót kell megosztani.

Nyomtató a rendszerhez a Hardver menüpont nyomtatók almenüje alatt adható hozzá a rendszerhez.

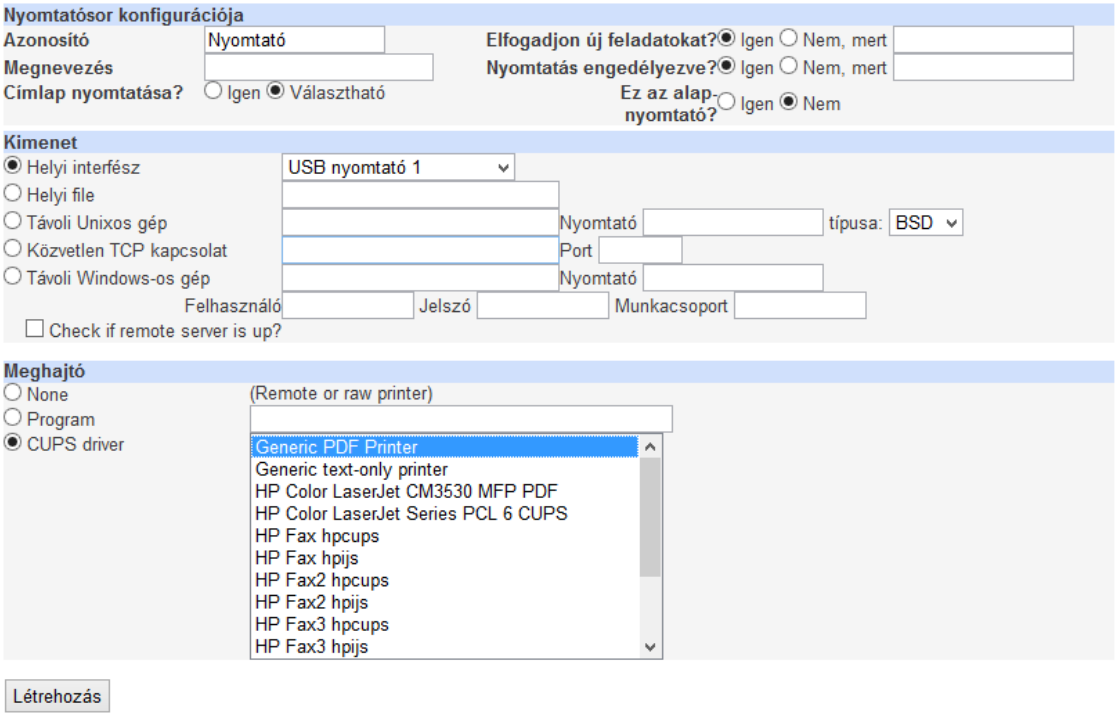
A nyomtatónak adni kell egy azonosítót, ami a nyomtató neve lesz. Rövid leírása is megadható, illetve beállítható, hogy alapértelmezett nyomtató legyen e.

Ezek után be kell állítani, hogy milyen porton csatlakozik a rendszerhez. Mivel a Pi csak USB porttal rendelkezik, ami alkalmas nyomtató fogadásra, így a legördülő listából az USB nyomtató menüpontot kell kiválasztani. Eszköz függően itt több menüpont is megjelenhet számozottan. Ekkor az 1-es végződésűvel érdemes kezdeni. Az USB nyomtató menüpont nem választható egészen addig, amíg a nyomtató nincs a rendszerhez csatlakoztatva.

Az illesztőprogram esetén a CUPS illesztőt kell bejelölni, majd a listából ki kell választani a nyomtató típusát. Amennyiben a lista üres, akkor telepíteni kell egy driver csomagot. Ehhez az alábbi parancsot kell lefuttatni:

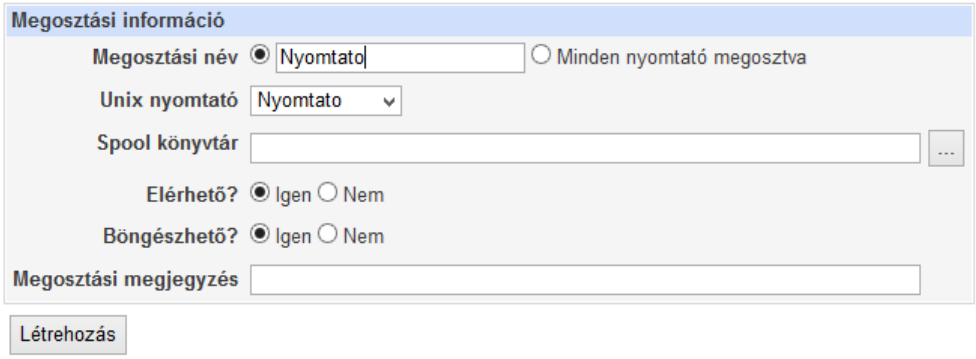
```
sudo apt-get install cups-driver-gutenprint
```

A csomag telepítése után a modul lapot frissítve elérhetővé válik egy csomó nyomtató választásra. Ekkor is előfordulhat, hogy a használni kívánt nyomtató nincs a listában. Ebben az esetben manuálisan kell feltelepíteni az eszközt. Ehhez azonban szerezni kell egy cups illesztő programot. Ezt a legkönnyebben a Google kereső megfelelő használatával tudjuk beszerezni, feltéve, hogy létezik illesztőprogram. A keresést a nyomtató neve és cups driver kulcsszavakkal érdemes megkezdeni.



HP nyomtatók estén a http://hplipopensource.com/ oldalt kell meglátogatni, itt több, mint valószínű fogunk illesztőt találni a HP nyomtatónkhoz.

A létrehozott nyomtató ezután megosztható. A megosztás a Samba konfigurációs felületről intézhető el, az új nyomtató megosztás létrehozása menüpont alatt. Itt nem sok mindent kell megadni.



Ezután a nyomtató használható hálózaton keresztül is. Itt megjegyezném, hogy a megosztás után minden olyan gépre telepíteni kell a nyomtató illesztőprogramját, amiről használni szeretnénk a nyomtatót. Ezért célszerű létrehozni egy külön megosztást, ami a nyomtató Windows illesztőprogramját tartalmazza.

12. dlna végpont

A dlna szabvány nem új keletű dolog. A technológia alapjait 2003-ban fektették le, UnPNP vagy UPNP néven is ismert. Számos lejátszó és program támogatja. Lehetővé teszi azt, hogy hálózaton keresztül média anyagokat továbbítsunk egy lejátszó felé. Így megoldható az, hogy az emeleten lévő személyi számítógépről a nappaliban található tv-re továbbítsunk egy filmet anélkül, hogy hang és videó kábelekkel bajlódnánk.

A szabvány három típusú eszközt definiál. Ezek:

- *Szerver*
  Média tárral rendelkező eszköz, feladata a fájlok továbbítása, kiszolgálása
- *Kliens, végpont*
  A média anyag megjelenítője. A továbbított média anyag fizikailag itt fog átalakulni látható és hallható tartalommá
- *Vezérlő*
  A média tartalom irányítását végző eszköz. Tipikusan okos telefon, amivel vezérelhetjük a kommunikációt.

A Média anyag lehet kép, hang és videó is. A támogatott formátumok eszköz és lejátszó függőek.
Ebben a projektben egy távolról vezérelhető hanglejátszót konfigurálunk be működésre.
Amennyiben kép és videó is kell, akkor érdemes egy olyan disztribúciót választani, amely tartalmaz mindent ilyen célra. Ilyen például a Raspbmc rendszer.

Ez az XBMC nevezetű média központ futtatására kihegyezett rendszer, amelyel könnyen felokosítható bármelyik „buta" TV. A rendszer a http://www.raspbmc.com/download/ címről tölthető le, telepítése hasonlóan egyszerű a Raspbian rendszerhez.

A projektben kitűzött célunk eléréséhez első lépésként telepítenünk kell jó néhány csomagot.

```
sudo apt-get install alsa-base alsa-tools alsa-utils
gstreamer0.10-alsa \
gstreamer0.10-ffmpeg gstreamer0.10-fluendo-mp3 \
gstreamer0.10-plugins-base libgstreamer-plugins-base0.10-0 \
libgstreamer0.10-0 libgstreamer0.10-dev gstreamer0.10-plugins-good
\ gstreamer0.10-tools libupnp-dev automake cvs
```

A szoftverek telepítése után ellenőrizzük, hogy a hangkártya modul betöltődik e indításkor. Ehhez adjuk ki a következő parancsot:

```
aplay -l
```

Ha a parancs kimenete a következő sorral kezdődik, akkor minden rendben:

```
card 0: ALSA [bcm2835 ALSA], device 0: bcm2835 ALSA [bcm2835 ALSA]
```

Ha ez nem jelenne meg, akkor a /etc/modules fájl tartalmához hozzá kell adni a snd_bcm2835 modult, majd a rendszer újraindítása után működnie kellene.

Mivel nem a HDMI kimeneten szeretnénk hangot lejátszani ezért át kell állítani a hang kimenetet az analóg hangkimenetre:

```
sudo amixer cset numid=3 1
```

A kimenet működőképessége tesztelhető a speaker-test parancs kiadásával. Ennek hatására statikus fehér zajt kellene hallanunk a kimeneten. A programból a CTRL+C billentyűkombinációval léphetünk ki.

Ezzel a hardver elő lett készítve, már csak a GmediaRenderer programot kell telepítenünk. Ezt érdemes forráskódból feltenni, hogy biztos jól működjön. Először le kell töltenünk a forráskódot az alábbi parancs segítségével:

```
cvs -d:pserver:anonymous@cvs.sv.gnu.org:/sources/gmrender co .
```

A letöltött forráskód lefordítása és telepítése igen egyszerű:

```
cd gmrender
./autogen.sh
./configure
make
sudo make install
```

A program telepítése után el kell indítani a végpont szolgáltatást.

```
gmediarender -f "Raspberry Pi"
```

Ezután bármely DNLA képes lejátszóból küldhetünk tartalmat az eszközre. A -f paraméter a végpont nevét határozza meg. Windows alatt tesztelésképpen a Windows Media Player segítségével küldtem zenét át az eszközre, ami problémamentesen működött.

Android esetén a Bubble UpnP/dnla alkalmazást célszerű alkalmazni, amely beszerezhető a https://play.google.com/store/apps/details?id=com.bubblesoft.android.bubbleupnp&hl=en címről.

A végpont szolgáltatás automatikusan nem fog elindulni, ezért érdemes létrehozni egy indítószkriptet, amit a /etc/init.d/gmediarenderer fájlba mentünk. A fájl tartalma:

```
#! /bin/sh
# /etc/init.d/gmediarender
USER=root
HOME=/root
export USER HOME
case "$1" in
start)
 echo "Starting GmediaRender"
 /usr/local/bin/gmediarender -f "Raspberry Pi"
 ;;
```

```
stop)
 echo "Stopping GmediaRender"
 killall gmediarender
 ;;
*)
 echo "Usage: /etc/init.d/gmediarender {start|stop}"
 exit 1
 ;;
esac
exit 0
```

Ezután a fájlt futtathatóvá kell tenni, majd hozzá kell adni az automatikusan induló szolgáltatásokhoz:

```
sudo chmod 755 /etc/init.d/gmediarender
sudo update-rc.d gmediarender defaults
```

13. dlna szerver

A Pi segítségével kiváló dlna szervert hozhatunk létre az otthoni „okos” eszközeink számára. A konfigurációja egyáltalán nem nehéz. Ha profi dnla megoldást szeretnénk létrehozni, akkor érdemes a dlna kialakítása előtt merevlemezt csatlakoztatni az eszközhöz, valamint a merevlemezt megosztani Windows megosztásként, hogy könnyen tudjunk rá fájlokat másolni.

Maga a szerver telepítése igen egyszerű, mert csak a minidnla szerver programot kell telepítenünk. Ez könnyen megtehető az apt-get parancs segítségével:

```
sudo apt-get install minidlna
```

A telepítés végeztével a /etc/minidlna.conf fájl módosításával konfigurálható a szerver. Közvetlenül fájlok nem oszthatóak meg, csak könyvtárak. Egy könyvtár csak egy típus megosztására alkalmas.

Könyvtárakat az alábbi szintaxis segítségével vehetünk fel:

```
media_dir=TÍPUS,/eleres
```

A típus a mappában található fájlok típusát jelzi. A típus egy karakter, értéke lehet A, P, vagy V. Az M karakter azt jelzi, hogy a mappa zene fájlokat tartalmaz, a P karakter képek esetén alkalmazandó, a V karakter pedig videó fájlokat jelöl. Az elérés helyére a mappa elérési útvonala írandó, amit meg szeretnénk osztani.

Ezt követően be kell állítani, hogy a szerver automatikusan érzékelje az új fájlok behelyezését. Így, ha fájlokat töltünk fel, akkor nem kell újraindítani a szervert. Ehhez az alábbi sort kell hozzáadni a konfigurációhoz:

```
inotify=yes
```

A program a fájlok adatait gyorsítótárazás miatt adatbázisban tárolja. Ennek az útvonala az alábbi parancs segítségével határozható meg:

```
db_dir=/home/pi/minidlna
```

Az adatbázis könyvtárnak a root felhasználó által írhatónak és módosíthatónak kell lennie, így vagy az ő nevében hozzuk létre, vagy írási jogokat adunk neki hozzá. Ehhez a chmod parancs használható.

```
chmod 777 /home/pi/minidlna
```

Az utolsó beállítás amit el kell végeznünk a szerver címének beállítása. Ehhez az alábbi sort kell beírni a konfigurációs fájlba:

```
presentation_url=http://raspberrypi:8200
```

A DNS cím működéséhez szükséges, hogy telepítve legyen a samba. Amennyiben ez nincs telepítve, akkor a DNS cím helyére a szerver IP címe írandó.

Opcionálisan megadható egy szerver név is. Ehhez még egy sort kell beírni a konfigurációs fájlba:

```
friendly_name=Szerver
```

A Szerver szó helyére a szerverünk neve írandó. A névben szerepelhetnek szóközök is.

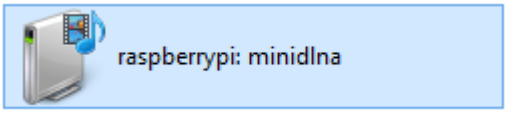
A teljes konfigurációs fájl tartalma jelen példa esetében:

```
media_dir=A,/share/Zene
media_dir=V,/share/Video
media_dir=P,/share/Kep

db_dir=/home/pi/minidlna
inotify=yes
presentation_url=http://raspberrypi:8200
```

A konfigurációs fájl módosítása után a szerver a következő két parancs kiadásával indítható újra:

```
sudo service minidlna start
sudo service minidlna force-reload
```

Ha mindent jól csináltunk, akkor a szervernek meg kell jelennie a Windows hálózatok alatt, valamint a dlna képes programokban is.



A minidlna szerver automatikusan a pi indításakor nem fog elindulni, viszont könnyen megoldható. Csupán az alábbi parancsot kell kiadni, és ezután automatikusan el fog indulni a dlna szolgáltatás:

```
sudo update-rc.d minidlna defaults
```

14. I²C és SPI buszrendszer kezelése

A Pi rendelkezik I2C buszrendszerrel. Az SCL láb a GPIO3 kivezetésen található, míg az SDA láb a GPIO2 kivezetéssel osztozik. A buszrendszer 3,3 volton működik. Ez nem probléma a legtöbb I2C eszköz számára a belső CMOS felépítés miatt. A buszrendszert nem szabad 5 volt feszültségre felhúzni ellenállásokon keresztül, mert ezt a Pi nem tolerálja. 3,3V feszültségre sem kell húzni a buszrendszert, mivel a Pi lapon ezt megoldották helyettünk.

Az I²C eszközöket 5 voltról érdemes üzemeltetni a 3,3 voltos feszültség stabilizátor gyenge terhelhetősége miatt. Ez a kommunikáció során nem okoz problémát, mivel a jelszinteket a buszrendszer felhúzó ellenállásai határozzák meg.

A buszrendszerek használatához először is engedélyezni kell azokat, mivel Raspbian esetén alapértelmezetten a szükséges kernel modulok nem töltődnek be. A szükséges modulok betöltéséhez módosítani kell a /etc/modules fájl tartalmát. A fájl végére be kell írni az alábbi sorokat:

```
i2c-bcm2708
i2c-dev
spi-bcm2708
```

Ezzel a modulok betöltését majdnem engedélyeztük, még a /etc/modprobe.d/raspi-blacklist.conf fájl tartalmát is módosítani kell. Ez a fájl felelős bizonyos modulok letiltásáért. Ebben a fájlban vannak letiltva az I²C és SPI modulok.

A modulok engedélyezéséhez a megfelelő sorok elől a # eltávolítandó. A fájl tartalma:

```
# blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
blacklist i2c-bcm2708
```

A fájl mentése és a rendszer újraindítása után a szükséges modulok betöltődnek, ezután használhatóak a buszrendszerek a megfelelő szoftverekkel. Ezek alapértelmezetten szintén nincsenek feltelepítve. Az I2C buszrendszer kezelő szoftvereket az alábbi parancs segítségével tudjuk telepíteni:

```
sudo apt-get install python-smbus i2c-tools
```

A python-smbus csomag egy Python modult telepít, amely segítségével Python programjainkból tudjuk kezelni a buszrendszert és a rákötött eszközöket, míg az i2c-tools parancssori eszközöket biztosít a buszrendszer kezeléséhez. Ezen eszközök közül a leghasznosabb az i2cdetect. Ez kilistázza a buszrendszerre kötött eszközök címeit.

A programot rendszergazdaként kell futtatni. Paraméterként meg kell adni egy számot utána, hogy melyik buszrendszert tesztelje. Ennek oka az, hogy a Pi processzora több I²c buszrendszerrel is rendelkezik. Az 512Mb-os B modell esetén 1-es paraméterrel kell futtatni, korábbi 256Mb-os változatok esetén pedig 0-s paraméterrel. A listázás előtt figyelmeztetni fog a program, hogy ez összezavarhatja a buszra kapcsolt eszközök működését. A listázást csak megerősítés után fogja folytatni. Ez a megerősítés a -y kapcsoló használatával elkerülhető.

Az SPI buszrendszer telepítéséhez forráskódból kell szoftvert telepíteni, ha Pythont szeretnénk alkalmazni. Az ehhez szükséges parancsok:

```
mkdir python-spi
cd python-spi
wget https://raw.github.com/doceme/py-spidev/master/setup.py
wget https://raw.github.com/doceme/py-
spidev/master/spidev_module.c
sudo python setup.py install
```

A python-spi modul használatáról a http://www.100randomtasks.com/simple-spi-on-raspberry-pi címen található több információ, míg a python-smbus modul dokumentációja a http://wiki.erazor-zone.de/wiki:linux:python:smbus:doc címen érhető el.

SPI buszrendszerből kettővel is rendelkezik a Pi, de csak a 0. jelzésű van kivezetve a lapra. Továbbá ezen buszrendszer esetén a jelszinteket az eszköz tápfeszültsége határozza meg. Ha az eszköz 3,3V tápfeszültségre van kapcsolva, akkor 3,3V jeleket fog kiadni, viszont ha 5V-os tápfeszültséggel használt, akkor jelszint illesztés kell a Pi és az eszköz közé.

Kezelés WiringPi segítségével

Az SPI és az I²C buszrendszer is kezelhető C és C++ programokból a WiringPi könyvtár segítségével. Ennek telepítéséről és általános Használatáról a 4. projekt ad kezdésképpen információt.

SPI

Az SPI buszrendszer kezeléséhez két függvényt biztosít a környezet, melyek a wiringPiSPI.h fájlban találhatóak.

```
int wiringPiSPISetup (int channel, int speed);
```

Inicialálja az SPI buszrendszert. Első paramétere a buszrendszer azonosítója, ami jelen Pi modellek esetén mindig 0, második paramétere a buszrendszer órajel jebessége Hz-ben megadva. Ez minimum 500 000 kell, hogy legyen (500 kHz), maximum pedig 32 000 000 lehet. (32 MHz)

```
int wiringPiSPIDataRW (int channel, unsigned char *data, int len);
```

Szimultán olvasás és írás. Az első paramétere az SPI csatornát határozza meg (0), második paramétere egy előjel nélküli karakter töm, ami az írandó adatot tartalmazza, a harmadik paramétere pedig a tömbből írandó karakterek száma. A visszaérkező adat a második paraméterben meghatározott tömbbe érkezik vissza, felülírva az ott található értékeket.

Mindkét kezelő függvény –1 értéket ad vissza hiba esetén.

I²C

Az I2C buszrendszer kezeléséhez szükséges függvények a wiringPiI2C.h fájlban találhatóak. A függvények visszatérési értéke –1 hiba esetén.

```
int wiringPiI2CSetup (int devId);
```

Inicializál egy eszközt a paraméterként megadott címen.

```
int wiringPiI2CRead (int fd);
```

Egy byte adat olvasása a paraméter által meghatározott című eszközről.

```
int wiringPiI2CWrite(int fd, int data);
```

Egy bájt adat írása az első paraméter által meghatározott eszközre, második paramétere a küldendő byte.

```
int wiringPiI2CWriteReg8(int fd, int reg, int data);
int wiringPiI2CWriteReg16(int fd, int reg, int data);
```

Nyolc vagy 16 bites regiszter írása az első paraméter által meghatározott eszközön. A második paraméter a regiszter címe, a harmadik paraméter pedig az írandó adat.

```
int wiringPiI2CReadReg8(int fd, int reg);
int wiringPiI2CReadReg16(int fd, int reg);
```

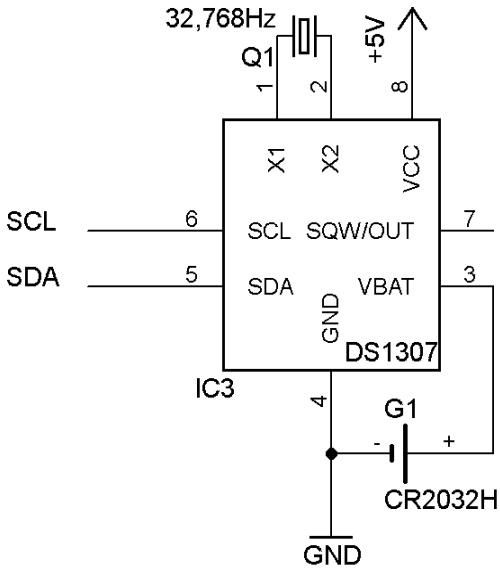
Egy nyolc vagy 16 bites regiszter tartalmának olvasása az első paraméter által meghatározott címről. A

második paraméter a reigiszter címét határozza meg.

15. DS1307 RTC kezelése

A RaspberryPi költség spórolás miatt nem rendelkezik hardveres órával, így kikapcsolás esetén nem tartja meg a pontos időt. Az Időt NTP protokoll segítségével hálózatról szerzi be. Ez a megoldás igen jó, viszont abban az esetben, ha nem net közelben szeretnénk alkalmazni a Pi-t és szükségünk van a pontos időre, akkor ki kell egészíteni a Pi-t egy valós idejű órával. Erre a célra a legkézenfekvőbb a DS1307-es áramkör.

Az IC bekötése igencsak gyerekjáték, hasonló módon kell bekötni, mint a PIC mikrovezérlők esetén, azzal a különbséggel, hogy az SDA és SCL lábakat nem kell 5 volt feszültségre húzni.



A bekötés után a buszrendszer címtérképét kérdezzük le az alábbi parancs segítségével:

```
sudo i2cdetect -y 1
```

Ha a bekötés helyes 68-as azonosítóval fel kell bukkannia a DS1307-nek:

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -
```

Ezután be kell tölteni a ds1307 kezeléséért felelős kernel modult, majd meg kell neki határozni az eszköz címét. Ehhez az alábbi két parancsot kell kiadni:

```
sudo modprobe rtc-ds1307
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
```

Ezután a chip-ben tárolt idő az alábbi parancs segítségével kérdezhető le:

```
sudo hwclock -r
```

A pontos idő beállításához a rendszer időt kell először módosítani. Ha a Pi hálózaton van, akkor az idő biztos pontos lesz. A dátum és idő a date parancs segítségével módosítható.

A hardveresen tárolt idő a következő parancs segítségével menthető:

```
sudo hwclock -w
```

A hardveres óra indítás utáni használatához a /etc/modules fájlhoz hozzá kell adni a következő sort:

```
rtc-ds1307
```

Ezzel betöltődik a szükséges modul, azonban, hogy az óra el is induljon a /etc/rc.local fájlba, az exit 0 sor elé az alábbi két sort még be kell írni:

```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
sudo hwclock -s
```

Ezzel újraindítás után a rendszeridő a hardveres órához lesz igazítva, így akkor is lesz pontos rendszer idő, ha a pi nincs hálózaton. Amennyiben hálózaton van, akkor pedig NTP protokollon keresztül fogja az időt frissíteni.

16. Keresztfordító környezet létrehozása

A Pi processzorának a sebessége túlhúzva sem mondható gyorsnak, ha C vagy C++ forráskód fordításra kerül a sor. Ez különösen nagy programok esetén okoz problémát, mivel ebben az esetben a programok fordítása eltarthat egy jó ideig.

A Keresztfordító környezet lényege, hogy a Pi processzora számára fordítunk programokat egy másik architektúrát használó gyorsabb, asztali gép segítségével. Egy ilyen környezet kialakításához szükségünk lesz egy Linux rendszerre. Virtuális gépre is telepíthetjük, amennyiben a processzorunk támogat virtualizáció kisegítést és van memóriánk bőven, akkor nem fog lassulást okozni az, hogy virtuális gépen belül fut a Linux.

Elméletileg bármilyen disztribúció jó, mivel a crosstool-ng program forráskódból települ. Az egyszerűség kedvéért Debian Linux rendszert telepítettem fel és ezen mutatom be a fordító telepítését.

Először is le kell tölteni a crosstool-ng programot a http://crosstool-ng.org/download/crosstool-ng/ címről. Minden esetben a legfrissebb verziót érdemes letölteni. Ezen projekt esetén az 1.19.0 verziót használtam. A program letöltéséhez és kibontásához használt parancsok:

```
cd ~
mkdir -p src
cd src
wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-
1.19.0.tar.bz2
tar xjf crosstool-ng-1.19.0.tar.bz2
```

Ahhoz, hogy a fordítás sikeres legyen telepíteni kell pár csomagot a rendszere.

```
sudo apt-get install bison cvs flex gperf texinfo automake libtool
gawk cvs libncurses-dev gcc g++ build-essential
```

Ezután a crosstool lefordítható és telepíthető. A telepítést én a felhasználói mappámban létrehozott programs mappába tettem. A felhasználóm neve kreatív módon user.

```
mkdir -p programs
cd crosstool-ng-1.19.0
./configure –prefix=/home/user/programs/crosstool-ng
make
make install
```

A fordítás után a program használható is, ha hozzáadtuk az elérési könyvtárát a keresési útvonalhoz.

```
export PATH=$PATH:/home/user/programs/crosstool-ng/bin/
```

A telepített crosstool arra jó, hogy létrehozzuk a fordítási környezetet. A környezet létrehozása el fog tartani egy ideig, mivel forráskódból tölti le a program az eszközöket. Gép függően a telepítéshez 1-1,5 óra szükséges. A telepítés előtt azonban konfigurálni kell a programot, hogy milyen környezetünk legyen. A környezetet szintén a programs könyvtárba telepítem, de ehhez először létre kell hozni egy ideiglenes mappát, ami a fordítási beállításokat tárolja. Ezt az src mappába hoztam létre.

```
cd ~/src
mkdir -p cross
cd cross
mkdir -p ~/programs/crosstool
ct-ng menuconfig
```

Az utolsó parancs hatására megnyílik a terminál ablakban egy karakteres beállító program, amivel konfigurálhatjuk a létrehozandó környezetet. A legfontosabb menüpontok, amiket be kell állítani:

Menüpont neve	***Értéke***	***Leírása***
Paths and misc options -->		
Try features marked as experimental	*	*Kísérleti beállítások engedélyezése, mutatása*
Prefix Directory	*${HOME}/programs/crosstool/${CT_TARGET}*	*Telepítési mappa*
Target options -->		
Target architecture	*arm*	*Cél architektúra*
Endiannes	*litle endian*	*CPU Litle endian és 32 bites*
Bitness	*32-bit*	
Architecture level	*armv6*	*Armv6 utasításkészlet alkalmazása*
Floating point	*hardware fpu*	*Hardveres fpu használata lebegőpontos számokhoz*
Operating system -->		
Target OS	*Linux*	*Cél operációs rendszer*
Binary utilities -->		
Binutils version	*Legfrissebb, ami nem experimental. Jelen esetben 2.22 volt.*	*Bináris létrehozó parancsok verziója*
C Compiler -->		
Gcc version	*Legfrissebb, jelen esetben 4.8.1 volt.*	*GCC verzió beállítása*
C++	*	*C++ fordító telepítése*
C-library -->		
C library	*glibc*	*Glibc használata*
glibc version	*Legfrissebb, jelen esetben 2.17 volt.*	*Glibc verziószáma*

125. Táblázat: A crosstool-ng módosítandó beállításai

A beállítások végeztével a főmenüben az exit megnyomásakor megkérdezi a program, hogy mentse e a konfigurációt. A metés után a mappában ki kell adni az alábbi parancsot ami megkezdni a környezet létrehozását. Ez virtuális gépen ~80 percet vett igénybe:

```
ct-ng build
```

A Sikeres fordítás után ~/programs/crosstool mappában létrejön egy arm-unknown-linux-gnueabi nevű mappa, ami tartalmazza a Pi számára használható fordítót. Az itt található fordítóprogramokat szintén hozzá kell adni a keresési útvonalhoz, hogy használhatóak legyenek. Ennek tartós módja, ha a felhasználói mappánkban található .bashrc fájl végére beírjuk az alábbi sort:

```
export PATH=$PATH:/home/user/programs/crosstool/arm-unknown-linux-
gnueabi/bin
```

A pi számára használható c fordító parancs a arm-unknown-linux-gnueabi-gcc, míg a C++ fordításhoz használható parancs a arm-unknown-linux-gnueabi-g++

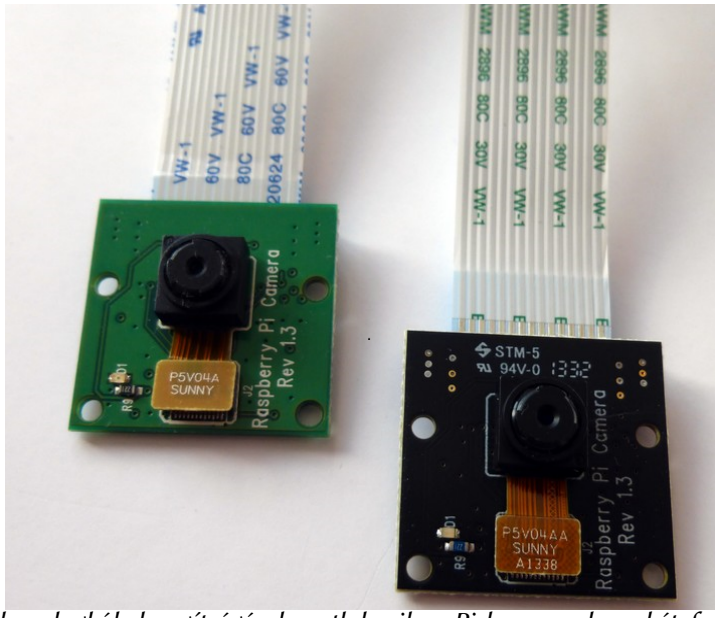
Ezen két paranccsal fordított programjaink asztali gépen nem lesznek futtathatóak, de ssh kapcsolaton keresztül a Pi-re másolva tökéletesen fognak működni, ha mindent jól csináltunk. Komplexebb programok fordítása esetén elképzelhető, hogy módosítani kell majd a make fájlokat a fordító használatához. Szerencsésebb esetben a configure parancsnak megadható kapcsolóként a használandó C fordító parancs.

A könyv DVD mellékletén megtalálható egy virtuális gépre telepített Debian rendszer, amire a szükséges fordító környezet telepítve lett, az egyszerűség kedvéért. A virtuális gép használatához Oracle Virtualbox szükséges, amely szintén megtalálható a DVD mellékleten.

17. Kamera modul kezelése

A RaspberryPi rendelkezik egy hivatalos kamera modullal, ami majdnem annyiba kerül, mint maga a Pi. Ezen kamera modul használatának előnye, hogy közvetlenül a processzorban található grafikus egységgel kommunikál, így nem terheli feleslegesen a processzort, mint egy USB web kamera. A kamera 5MP felbontású, amely segítségével a Pi képes Full HD felbontásban is rögzíteni másodpercenként 30 képkockát.

A kamera modul két változatban érhető el. Hagyományos és Infra szűrő nélküli változatban. Az infra szűrő nélküli változatot a felhasználók kérésére készítették el. Ezen kamera modul jobban alkalmas éjszakai felvételek készítésére mert a kamerába jutó fényből nincs ami kiszűrje az infravörös tartományba eső fényt, amit amúgy érzékelne a szenzor. A két kamera modult úgy lehet megkülönböztetni, hogy a normál kamera zöld színű nyomtatott áramkörre van helyezve, míg az infra szűrő nélküli változat fekete nyomtatott áramkörön van.

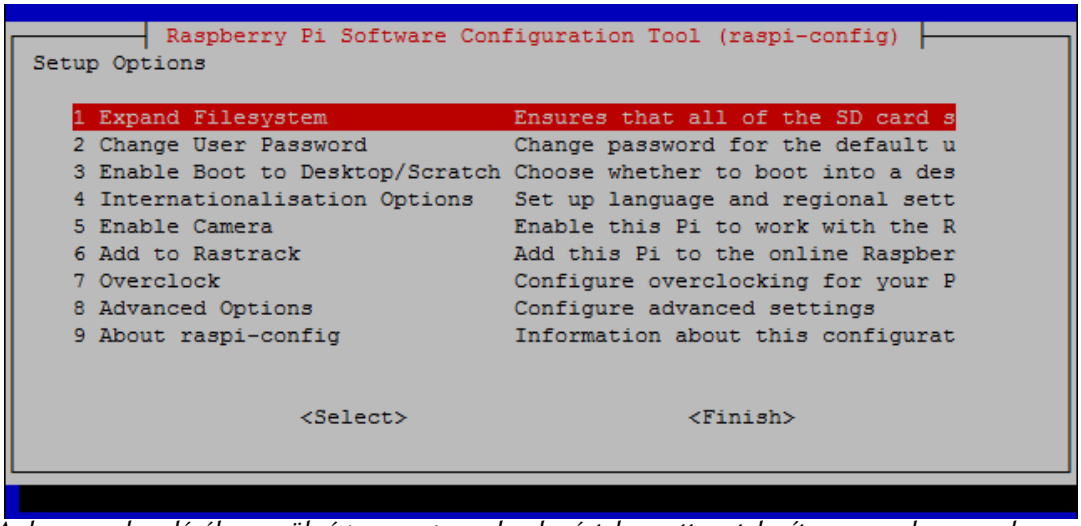


A kamera modul szalagkábel segítségével csatlakozik a Pi-hez, amelyen két foglalat is van ilyen típusú kábelek fogadásához. A kamerát az Ethernet csatoló mögött található foglalatba kell csatlakoztatni. A csatlakozóba egyszerűen becsúszik a kábel a csatlakozó oldalán található rögzítő fülek felhúzása után. A kábel behelyezése után a rögzítő fülek lenyomásával lehet a csatlakozást véglegesíteni. A kábel érintkezősorának a HDMI port felé kell állnia.

A csatlakoztatás után a használatbavételhez engedélyezni kell a kamerát a raspi-config alkalmazásban, azonban ez előtt célszerű a rendszert frissíteni, hátha van frisebb illesztő a kamerához. A frissítés az apt-get segítségével végezhető el, korábban már ismertetett módon:

```
sudo apt-get update
sudo apt-get upgrade
```

A kamera modul bekapcsolása előtt érdemes megjegyezni, hogy nem árt ellenőrizni a GPU memóriáját sem. Minimum 128Mb memóriát kell adni a grafikus egységnek ahhoz, hogy rendesen működjön a kamera.



A kamera kezeléséhez szükséges programok alapértelmezetten telepítve vannak a rendszerre. Ezek a raspstill és raspvid parancssori programok, amelyek segítségével képeket és videó felvételeket készíthetünk.

Mindkét program esetén a -o paraméter segítségével határozhatjuk meg a kimeneti fájlt, aminek a formátuma képek esetén jpeg, videók esetén h264 lesz. A h264 tároló nélküli HD film formátum, amit könnyen le tudunk játszani mplayer segítségével, illetve nagyon könnyen mp4 formátumra is tudjuk alakítani a következő ffmpeg parancs segítségével:

```
ffmpeg -i be.h264 -vcodec copy ki.mp4
```

Szintén mindkét program esetén alkalmazható a -t paraméter ami a felvétel készítésének kezdetét tolja el a -t után megadott értékkel képek esetén. Videó esetén a felvétel hosszát határozza meg. Az értéket a program milliszekundumként értelmezi.

Az utolsó megosztott közös opció a két program között az élő előnézet bekacsolására vonatkozik. Ezt a -p paraméter segítségével lehet bekapcsolni. A -p kapcsoló után a program négy szám értéket vár. Az első két szám az előnézeti kép x és y koordinátáját határozza meg a képernyőn, míg az utolsó két számpár a kép szélességét és magasságát. Az előnézeti kép közvetlenül a GPU-ból jön, így ha a kép útjában van egy ablak, akkor egyszerűen felette fog megjelenni.

raspstill specifikus beállítások

A készítendő kép méretét a -w és -h kapcsolók segítségével szabályozhatjuk. A -w kapcsoló a kép szélességének megadására szolgál, míg a -h kapcsoló a kép magasságát állítja be. Mindkét kapcsoló után írt szám pixelekben értendő.

A JPEG kép minőségét a -q paraméterrel szabályozhatjuk. A -q után ért szám a minőséget jelzi százalékos értékben, így értéke 0 és 100 közötti szám lehet.

Raspvid specifikus beállítások

A videofelvétel méretét szintén a -w és -h kapcsolókkal lehet meghatározni. A magasság érték minimum 64 lehet, maximum pedig 1080. Szélesség érték esetén a minimum 64, a maximum pedig 1920.

A film bitrátája (minősége) a -b paraméterrel határozható meg. Ezt a program bit/szekundum formátumban értelmezi és nem lehet több, mint 25Mbit, ami 25 000 000 bit/s-nek felel meg. A minőséget HD filmek esetén 10 és 17 Mbit között érdemes megválasztani. 17Mbit felett észrevehető minőség javulás nem vehető észre.

A film képkocka sebességét a -f paraméterrel lehet meghatározni. Ez minimum 2 kell, hogy legyen, maximum pedig 30 lehet.

További beállítások

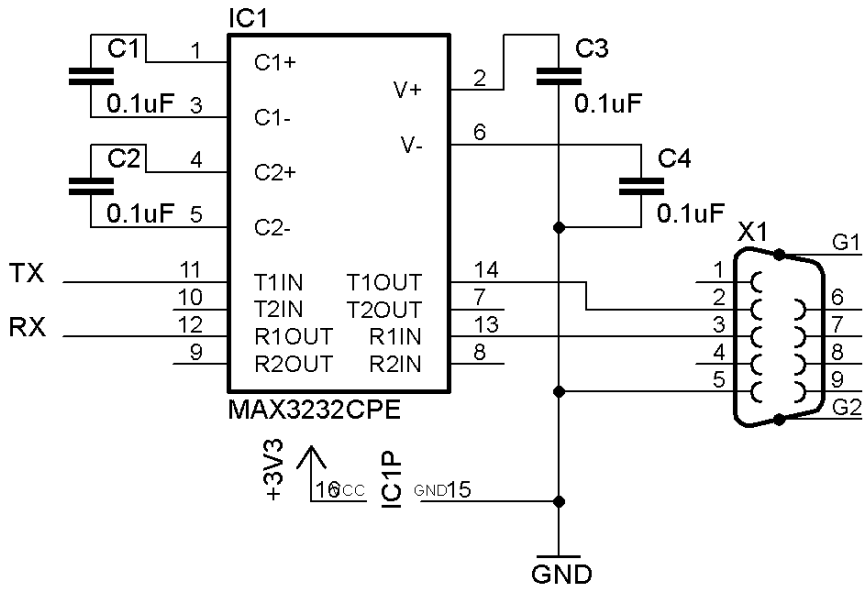
Mindkét program esetén további beállítások is megadhatóak. Ezen beállítások olyan képkészítési beállítások, amelyek bármelyik profibb fényképezőgépen megtalálhatóak, így akár a Pi kamerájából egy profi fényképezőgépet is készíthetünk, már ami a szoftvert illeti. Sajnos az 5 megapixel felbontás sem tudja pótolni a megfelelő optika hiányát, így a képeink minőségben nem igen lesznek jobbak egy telefonnal készített képnél.

A készítők mellékeltek egy tömörítetlen képeket készítő programot is, amely a raspstillyuv nevet kapta. Használatában megegyezik a raspstill programmal, azonban ez veszteség mentesen tárolja le a kép adatait yuv420 formátumban, amit aztán egy profi képszerkesztő segítségével módosítani tudunk.

A kamera és a videó készítő összes lehetséges beállításáról egy listát kapunk, ha paraméterek nélkül próbáljuk meg az alkalmazásokat futtatni. Illetve egy nyomtatható dokumentum is elérhető a https://github.com/raspberrypi/userland/blob/master/host_applications/linux/apps/raspicam/RaspiCamDocs.odt címen.

18. Soros port kezelése

A RaspberryPi rendelkezik egy RS232 soros porttal. Ez a soros port a mikrovezérlőknél megszokott RX és TX láb hardveres vezérlések nélkül. A soros port jelszintje 3,3 volt, amit természetesen illeszteni kell a PC számára. Erre a korábbi PIC projektben használt MAX23 nem alkalmazható, mivel ez TTL jelszintekre lett kitalálva. Helyette a MAX3232 alkalmazható. Ezen áramkör kompatibilis a 3,3v-os jelszintekkel. Az alábbi ábrán egy MAX3232-es illesztő kapcsolása látható.



A Pi soros portja a GPIO14 (TX) és GPIO15 (TX) lábakra van kivezetve. Alternatív módon alkalmazhatunk a PC-vel összekötésre egy USB-soros átalakítót is. Ezen átalakítók többsége 5V-os TTL jelet állít elő. Ezen átalakítók jelszintjét konvertálni kell. Erre számos módszer létezik, korábbi projektek során pár ismertetve is lett.

A Pi soros portja alapértelmezetten távoli bejelentkezésre van bekonfigurálva hibakeresési célokból. Számítógépre kötve a megfelelő COM port és sebesség kiválasztása után PuTTY segítségével bejelentkezhetünk a gépre, mintha SSH-val kapcsolódnánk. A soros port baud rátája alapértelmezetten 115 200

Amennyiben a soros portot más célra szeretnénk használni, akkor le kell tiltanunk a soros porton a bejelentkezési lehetőséget. Ehhez a /etc/inittab fájlban az alábbi sort kell megkeresni és módosítani:

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

A sor elé be kell szúrni egy kettős kereszt (#) karaktert, ami hatására a konfigurációs beállítás megjegyzéssé válik.

A /boot/cmdline.txt fájl tartalmát is módosítani kell, mivel a sosos portra a rendszer kiírja a kernel indítási üzeneteit is. A fájlban meg kell keresni a következő sort:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline
rootwait
```

A sorból el kell távolítani a soros portra vonatkozó bejegyzéseket. Így kell kinéznie a módosítás után a sornak:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4
elevator=deadline rootwait
```

A rendszer újraindítása után a Pi soros portja használható normál soros portként. A soros port eszköz fájlja a /dev/ttyAMA0.

Portkezelésre számos szoftver megoldás áll rendelkezésünkre. Használhatjuk a Linux rendszerhívásokat a port kezelésére C és C++ programjainkból, de ettől sokkal kényelmesebb megoldás, ha WiringPi könyvtár függvényeit használjuk.

Python esetén a pyserial modul segítségével lehetséges a port kezelése.

Kezelés WiringPi segítségével

A WiringPi soros port kezelő függvényei a wiringSerial.h header fájlban vannak definiálva. A könyvtár az alábbi függvényeket biztosítja kezeléshez:

```
int serialOpen (char *device, int baud);
```

Megnyitja a soros portot adott baud rátával. Első paramétere a soros port eszköz fájlja, második paramétere a baud ráta. A függvény –1 értéket ad vissza, ha a megnyitás során hiba történt. Ellenkező esetben egy számot, amely a könyvtár többi függvénye számára meghatározza, hogy melyik portot kell használni. Ez egy szabványos Unix fájlleíró, amin alkalmazhatóak a rendszer read és write, valamint egyéb fájlkezelő függvényei is.

```
void serialClose(int fd);
```

Soros port lezárása. Paramétere a portot meghatározó szám.

```
void serialPutchar(int fd, unsigned char c);
```

Egy karakter továbbítása soros porton keresztül. Első paraméter a portot meghatározó szám, második paramétere a küldendő karakter.

```
void serialPuts(int fd, char *s);
```

A soros portra írja egy karaktertömb tartalmát. Első paraméter a portot meghatározó szám, második paramétere a küldendő karaktertömb.

```
int serialDataAvail(int fd);
```

Visszaadja a soros bufferben lévő még kiolvasatlan karakterek számát. Paramétere a soros portot határozza meg számként.

```
int serialGetchar(int fd);
```

Egy karakter olvasása a soros portról. Olvasási hiba esetén a visszatérési értéke –1. A paraméter a portot határozza meg.

```
void serialFlush(int fd);
```

Törli a soros porthoz tartozó buffer tartalmát. A paraméter a portot határozza meg.

Kezelés pyserial segítségével

A modul nem része az alap python telepítésnek, külön kell telepíteni. Az alábbi parancs segítségével telepíthető könnyen:

```
sudo apt-get install pyserial
```

Ezután a programunkba a szükséges osztályokat a serial modulban találjuk meg. A modul legfontosabb osztálya a serial, amin keresztül kezelhető a port. Az osztály konstruktora az alábbi formában hívható meg:

```
ser = serial.Serial(dev, baud, timeout)
```

A konstruktor első paramétere a soros port eszközfájlja. Második paramétere a baud rátát határozza meg, míg a harmadik egy várakozási időt, amely megadása nem kötelező.

```
ser.read(bytes)
```

Adat olvasása a portról. A paramétere az olvasandó byte-ok számát határozza meg. A paraméter megadása nem kötelező. Paraméter nélkül egy byte adatot olvas.

```
ser.readline()
```

Egy sor beolvasása. A sor vége jel a \n (enter) kódig történik.

```
ser.write(data)
```

Adat írása a soros portra. Az adat lehet szöveg vagy byte.

```
ser.inWaiting()
```

Visszaadja a bemeneti bufferben lévő még kiolvasatlan karakterek számát.

```
ser.flushInput()
```

A bemeneti buffer tartalmát törli.

```
ser.close()
```

A portot zárja le. A függvény meghívása után adat írása és fogadása nem lehetséges a portról.

Arduino programozása

A Pi soros portja használható Arduino program feltöltésre is, azonban ehhez egy picit módosítani kell a rendszert. Ennek az oka az, hogy a Pi nem rendelkezik hardveres forgalomirányítási rendszerrel, ami kell ahhoz, hogy az Arduino a program feltöltés automatikusan megtörténjen a reset gomb nyomogatása nélkül.

A módosítás első lépéseként fel kell telepítve lennie az Arduino környezetnek. A telepítési parancs a következő:

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

Ezután le kell tölteni és telepíteni kell a szoftveres vezérlőjel előállító programot, ami a GPIO11-es lábat kinevezi DTR lábnak, ami az újraindításhoz kell. A szükséges szoftver a következő parancs segítségével telepíthető:

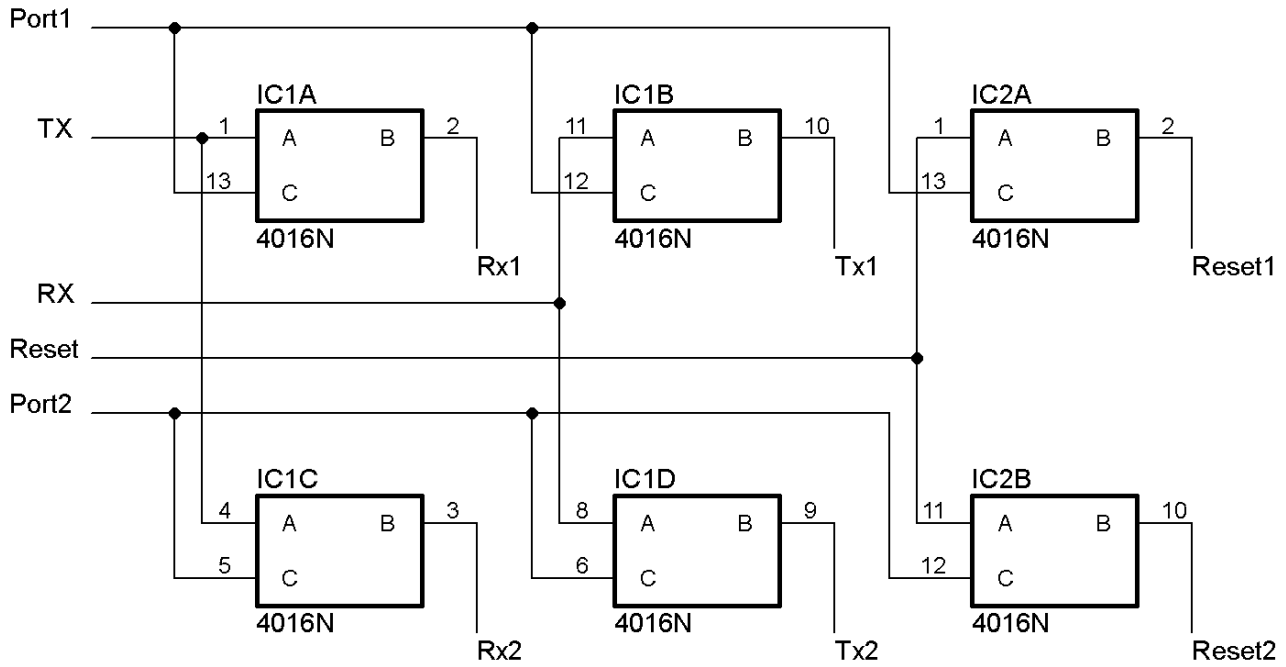
```
sudo apt-get install git
git clone https://github.com/deanmao/avrdude-rpi.git
cp autoreset /usr/bin
cp avrdude-autoreset /usr/bin
mv /usr/bin/avrdude /usr/bin/avrdude-original
ln -s /usr/bin/avrdude-autoreset /usr/bin/avrdude
```

Ezután a Pi soros portjára csatlakoztatva az Arduino lapunk RX és TX lábát képes lesz a Pi program feltöltésre. Ehhez azonban az Arduino reset lába és a Pi GPIO11-es lába közé be kell iktatni egy 0,1 uF kapacitású kondenzátort. Természetesen a GPIO11-es lábat először 5V-ra illeszteni kell.

A közvetlen soros portra illeszkedő programozás előnye az USB megoldással szemben, hogy így készíthetőek I/O kiegészítő lapok a Pi számára, amelyek Arduino környezettel újraprogramozhatóak.

A soros programfeltöltés működéséhez először azonban a chip-be hagyományos programozás útján be kell égetni a bootloadert.

A Pi soros portjára több Arduino is köthető, amennyiben multiplexerjük a soros portot. Az alábbi ábrán egy ilyen kapcsolás látható.



A kapcsolás a 4016-os analóg kapcsoló IC-re épül, aminek az egyetlen előnye, hogy létezik DIP kivitelben. Az Analog Devices sokkal jobb analóg kapcsoló áramköröket gyárt, viszont ezek többsége csak SMT technológiával készülnek. Több port multiplexelése esetén célszerű egy analóg multiplexer áramkört alkalmazni.

Az ábrán látható kapcsolás jelszint illesztési megoldásokat nem tartalmaz, illetve a reset lábakon sincs rajta a 0,1uF kondenzátor.

19. Arduino és a RaspberryPi összekapcsolása

Az Arduino környezet platform függetlenségének előnye, hogy a Pi is használható fejlesztő gépnek. USB segítségével igen könnyű összekötni a Pi-t bármilyen Arduino segítségével, mivel ebben az esetben ugyan úgy működik a dolog, mint PC esetén. Az Arduino fejlesztő környezet az alábbi parancs segítségével telepíthető:

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

Ez feltelepíti a programozáshoz szükséges szoftvereket. A környezet előfordulhat, hogy nem támogatja a legújabb lapokat. Ebben az esetben forráskódból kell feltenni a szükséges új szoftvert. Ennek a lefordítása Pi esetén nagyon elhúzódhat köszönhetően annak, hogy a Pi nem éppen a leggyorsabb Java fordításban.

A két hardver között több kommunikációs lehetőségünk is van: soros, I²C és SPI. A leg kézenfekvőbb USB esetén a soros kommunikáció, mivel ehhez nem kell jelszint illesztés. A soros kommunikáció esetén is számos lehetőségünk van. Vagy saját protokollt írunk, vagy Firmata segítségével kommunikálunk.

Python esetén a Firmata protokoll használatához a pyfirmata telepítése szükséges. Ezen modul hátránya, hogy csak Uno, Mini, Leonardo, és Mega modelleket támogat. A modul az alábbi parancsok segítségével telepíthető:

```
sudo apt-get install git python-serial
git clone https://github.com/tino/pyFirmata
cd pyFirmata
pip install pyserial
python setup.py install
```

C és C++ nyelvek esetén a FirmataC könyvtár alkalmazását ajánlom. Ezen könyvtár előnye, hogy viszonyag kevés fájlból áll, így könnyen integrálható a projektünkbe. A kód a https://github.com/jdourlens/FirmataC oldalról szerezhető be.

Sajnos a WiringPi könyvtár csak soros kommunikációs támogatással rendelkezik, a Firmata protokoll nincs implementálva.

Az Arduino és a Pi összekapcsolása több szempontból is jó. Az Arduino lap rendelkezik Analóg bemenetekkel és PWM kimenetekkel, amikkel a Pi nem. Továbbá mivel az Arduino nem futtat operációs rendszert könnyen képes mikroszekundum pontos időzítési feladatok elvégzésére, amire a Pi nem képes.

Valamint használható I/O processzornak is szabályzási rendszerek jelének előfeldolgozására. Mondhatni a lehetőségek végtelenek.

Firmata protokoll és a PyFirmata segítségével könnyen kezelhető bármelyik Arduinio lap. Az Arduino oldalon annyi követelmény van, hogy a mintaprogramok közül a StandardFirmata programot futtassa a lap (az ehhez szükséges információk megtalálhatóak az Arduino projektek fejezetben).

A modul kezelése igen egyszerű, mivel a különböző típusú lábakat tömbökön keresztül valósítja meg. A tömb minden egyes eleme egy objektum, ami leírja a lábat. Ezen objektumoknak van egy read és write függvénye, amivel kiolvasható vagy írható a láb. Az alábbi példaprogram a digitális kimenetek egyszerű kezelését mutatja be.

```
import pyfirmata
import time

brd = Arduino('/dev/tty.usbserial-A6008rIF')

it = util.Iterator(board)
it.start()

allapot = brd.digital[12].read()
println "A 12. lab allapota:", allapot

println "13. lab valtoztatasa"
for i in range(0, 10):
      brd.digital[13].write(1)
      time.sleep(1)
      brd.digital[13].write(0)
      time.sleep(1)

board.analog[0].enable_reporting()
analog = board.analog[0].read()
println "A0 erteke:", analog

println "2,5V PWM jel a 6-os labon"
board.pwm[6].write(0.5)
```

A kódban a brd változó reprezentálja az Arduino lapot. Ezen objektum létrehozásakor meg kell adni a soros port eszköz leíró fájlját. Az it változó egy olvasó szálat hoz létre, ami a soros port bemenetet olvassa folyamatosan. Ez azért szükséges, hogy a visszaérkező, de feldolgozatlan adatok miatt ne akadjon be a soros port.

20. Kijelző illesztési opciók

Több lehetőség is adott, ha kijelzőt szeretnénk illeszteni a Pi-re. A legkézenfekvőbb megoldás a HDMI port használata, amely régebbi monitorokhoz is illeszthető egy HDMI-DVI átalakító segítségével.

Kisesebb méretű LCD kijelzők esetén a legegyszerűbb megoldás az, ha olyan kijelzőt választunk, amely rendelkezik kompozit RCA videobemenettel. Ilyen kijelzők a mini LCD TV-k és az autókba szánt megoldások. Ezen megoldás problémája, hogy az átvitel nem digitális, valamint HDMI esetén is igen limitált a választék kijelzőméret tekintetében.

A Pi elméletileg képes lenne LVDS porttal rendelkező kijelzők fogadására is, mivel van rajta LVDS csatlakozó, ami az SD kártya oldalához közelebb lévő szalag kábel csatlakozó. A feltételes mód oka az, hogy a Broadcom jó ideig nem tette nyílttá a GPU driver forráskódját, ami kell az LVDS csatlakozó használatához. A Driver jelenleg már nyílt forráskódú, azonban még nem igen kapni LVDS-re illeszthető paneleket a Pi-hez. Vagy ha lehet is kapni, kérdéses a megbízhatóságuk.

Ez azonban nem jelenti azt, hogy nem lehetséges LVDS felülettel rendelkező kijelzők használata. Használatukhoz kell egy HDMI-LVDS átalakítópanel. Ilyen átalakító panelt igen sok cég forgalmaz, LVDS kijelző panelből pedig annyi van, mint égen a csillag.

Ezen megoldás alkalmazásánál érdemes tudni, hogy a panel és a hozzá tartozó konverter kijelző mérettől és a pixelek számától függően igen drága is lehet.

Van egy negyedik lehetőség is, méghozzá SPI felületre csatlakozó kijelzők használata. Ezen eszközök saját kijelző vezérlővel rendelkeznek és legtöbbjük 3,3V feszültséget használ, így még illeszteni sem kell a jelszinteket.

Szoftveres oldalról viszont a probléma az, hogy a Pi-n futó kernelt újra kell fordítani, hogy képes legyen ilyen kijelzők kezelésére. Ez a Pi-n fél napba is beletelhet, így érdemes olyan kijelzőt választani, ha ilyent szeretnénk alkalmazni, amihez a gyártó biztosít újrafordított kernelt, vagy egy nagyon jó beüzemelési útmutatót. Máskülönben esélytelen, hogy ilyen kijelzőn képet jelenítsünk meg.

LVDS paneleket, illesztőket és SPI kijelzők közül széles választékát az Adafruit forgalmazza a Pi számára. A teljes választékuk a http://www.adafruit.com/category/105_160 címen lelhető fel.

Soros portra is illeszthetünk kijelzőt. A 4D systems direkt soros portra illeszthető intelligens kijelző modulokat gyárt, amelyek programozhatóak, grafikus felület alkotható rájuk. Ezt a grafikus felületet soros porton keresztül parancsokkal vezérelhetjük és bizonyos modellek esetén információkat is fogadhatunk az érintőképernyőről. Ezen modulok nem csak a Pi számára használhatóak, hanem bármilyen soros porttal rendelkező mikrovezérlő és SOC képes a kijelző vezérlésére.

Ezen eszközök programozásához a gyártó biztosít egy grafikus felület tervező és programozó szoftvert, amely ingyenesen beszerezhető a honlapjukról, ami a http://www.4dsystems.com.au/ címen lelhető fel. A kijelzők programozásához szükséges egy USB-Soros átalakító kábel is, amit szintén a gyártó oldaláról be lehet szerezni. Sajnos az általánosan elérhető USB-soros átalakítók nem jók ilyen célokra, csak a gyári kábellel lehet őket programozni.

21. Internetes kimenet vezérlés

A Pi esetén számos technológia és szoftver áll rendelkezésünkre, ha olyan projektet szeretnénk készíteni, amiben hálózaton keresztül vezéreljük a Pi működését és kimeneteit.

Általában webes projektek esetén a PHP nyelv szokott befutó lenni, mivel ez nagymértékben hasonlít a C nyelvhez. Azonban a PHP működéséhez szükséges egy Apache webszerver is, ami nem éppen arról híres, hogy erőforrás takarékos lenne.

Sokkal kézenfekvőbb megoldás a Python alkalmazása webes alkalmazások fejlesztésére, mivel a Python környezet tartalmaz egy web szervert is, amit könnyen programozhatunk. Illetve számos webes keretrendszer épül a pythonra. Ilyen keretrendszer például a Flask, amit ezen projekt kapcsán fogunk alkalmazni.

A Flask még egyszerűbben használhatóvá teszi a Python webes képességeit, illetve remek dokumnetációval és számos példával rendelkezik, amiből könnyen lehet működő weblapokat kreálni. További előnye a Flask keretrendszernek, hogy rendelkezik sablon támogatással gyárilag.

A sablon támogatás lényege, hogy a HTML kódot elkészítjük valamilyen szerkesztőben, majd ebbe a HTML lapba töltjük bele a dinamikusan generált tartalmakat, így a megjelenítés nem keveredik az alkalmazás kódjával. PHP esetén ez gyárilag nem adott, ott nekünk kellene megválasztani a sablon rendszert, azonban sok esetben ezt nem teszik meg, aminek az a következménye, hogy a kód keveredik a megjelenéssel, ami későbbi fejleszthetőség és hibakeresés szempontjából nagyon nem ideális.

A Flask alapértelmezetten nincs telepítve a Raspbian rendszeren, de gyorsan telepíthető az alábbi parancsok kiadásával:

```
sudo apt-get install python-pip
sudo pip install flask
```

A Flask telepítése után kezdhetünk is ismerkedni a keretrendszerrel. Az alábbi egyszerű példa egy hello world alkalmazást hoz létre.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
      return "Hello World!"

if __name__ == "__main__":
      app.run(host='0.0.0.0', port=80, debug=True)
```

A példa lényegi része a Flask importálása, majd a Flask objektum példányosítása. A weblapot a hello függvény hozza létre, ami egyszerűen kiírja a hello world szöveget. A kívánt URL társítást a függvény előtt megadott route parancs határozza meg.

Az utolsó két sor a script main függvénye, ami elindítja az alkalmazás futtatását az alapértelmezett IP címen 80-as porton és engedélyezi a hibakeresést.

A Script futtatása után a böngészőnket megnyitva a Pi IP címét beírva a Hello World példának kell minket fogadnia. A Script rendszergazdai jogosultságokkal futtatandó.

A kimenet vezérlő program

```
from flask import *
import RPi.GPIO as GPIO
```

```
pins = [2, 3, 4, 17, 22, 10, 9, 11, 7, 8, 25, 24, 23, 18, 14 ]
app = Flask(__name__)
GPIO.setmode(GPIO.BCM)

def allapotok():
      buffer = []
      for i in pins:
            GPIO.setup(int(i), GPIO.IN)
            buffer.append('GPIO '+ str(i) +' allapota: ' + →
str(GPIO.input(int(i))) + '\r\n')
            return ''.join(buffer)

@app.route("/")
def index():
      return render_template('index.html')

@app.route("/bemenetek")
def bemenetek():
      sablon = { 'allapot' : allapotok() }
      return render_template('be.html', **sablon)



@app.route("/kimenetek")
def kimenetek():
      return render_template('ki.html')

@app.route("/kimenetvezerel", methods=['GET', 'POST'])
def kimenetvezerel():
      if request.method == 'POST':
            GPIO.setup(24, GPIO.OUT)
            GPIO.setup(25, GPIO.OUT)

            if request.form['p24'] == 'on':
                  GPIO.output(24, 1)
            else:
                  GPIO.output(24, 0)

            if request.form['p25'] == 'on':
                  GPIO.output(25, 1)
            else:
                  GPIO.output(25, 0)

            return "A kimenetek valtasa megtortent"

      else:
            return ""

if __name__ == "__main__":
      app.run(host='0.0.0.0', port=80, debug=True)
```

A példaprogram két funkcióval rendelkezik. Képes lekérdezni a GPIO lábak állapotát bemenetre átváltva azokat, illetve a GPIO 24 és GPIO 25 lábak állapotát képes megváltoztatni. A példa sablon fájlokat használ. A sablon fájlokat az alkalmazás könyvtárán belül létrehozott templates mappában fogja keresni a script.

Sablonok esetén a HTML kódba a dinamikusan generálódó tartalom egy szótár objektumból lesz minden esetben beillesztve. A HTML kódba a szótár kulcs elemeit kell beírni arra a helyre, ahova a dinamikus tartalomnak

kell mennie. A kulcs elemek dupla kapcsos zárójelek közzött adandóak meg. Példaképpen az allapot kulcs a fájlban: {{allapot}}

A kimenetek állapotának változtatása web űrlap segítségével történik. Az űrlap feldolgózókódja a kimenetvezérel függvény, ami POST kérés fogadása esetén az űrlap adatoknak megfelelően beállítja a kimeneteket.

A sablonok

index.html

```
<!DOCTYPE html>
<html>
    <head>

        <title>RaspberryPi Web példa</title>
    </head>
    <body>
        <h1>RaspberryPi webes példa</h1>
        <hr/>
        <ul>
            <li>
                <a href="/bemenetek">Bemenetek állapota</a>
            </li>
            <li>
                <a href="/kimenetek">Kimenetek vezérlése</a>
            </li>
        </ul>
    </body>
</html>
```

be.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>RaspberryPi Web példa</title>
    </head>
    <body>
        <h1>RaspberryPi bemenetek állapota</h1>
        <hr/>
        <pre>{{allapot}}</pre>
    </body>
</html>
```

ki.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>RaspberryPi Web példa</title>
    </head>
    <body>
        <h1>RaspberryPi kimenet vezérlés</h1>
        <hr/>
        <p>Kimenetek állapotának módosítása</p>
        <form action="kimenetvezerel" method="post">
```

```
            <input type="checkbox" name="p24"/> GPIO 24
            <br/>
            <input type="checkbox" name="p25"/> GPIO 25
            <br/>
            <input type="submit" value="Küldés"/>
        </form>
    </body>
</html>
```

A példa természetesen tovább fejleszthető lenne beléptető rendszerrel és szebb, jobb sablonokkal. A Flask projekt dokumentációja a http://flask.pocoo.org/docs/ címen található. A http://flask.pocoo.org/snippets/ címen meg hasznos kis mintaprogramokat találunk, amivel a web alkalmazásunk tudását tovább bővíthetjük.

22. Minecraft Szerver telepítése

A Pi kiválóan alkalmas arra, hogy kedvenc játékaink házi szervere legyen. Ez alól a Minecraft sem kivétel. Azonban a Minecraft nem arról híres, hogy kevés memóriával és gyenge processzorral beérné szervernek. Ez abból adódik, hogy a játék logikai része Java-ban íródott.

A hivatalos szerver is futtatható a lapon, azonban siralmas teljesítményt tud produkálni, mivel a Java futtatókörnyezetet nem igen optimalizálták ARM processzorokra. Ha 20 perc után be is indul a szerver, akkor kb 10 perc játék alatt felfalja az összes memóriát, aminek következtében a szerver megáll.

Ha ennél többet szeretnénk játszani, akkor a hivatalos szervert el kell felejteni. Helyette az MCServer program használata javasolt. Ez a Minecraft szerver kódjának C++ átirata. Igen jó teljesítménnyel és memória felhasználással bír. Nagyjából 10 játékos kiszolgálására képes a Pi-n.

A program forráskódból telepíthető. Ehhez szükségünk lesz a cmake és git programokra. Ezeket a következő parancs segítségével tudjuk telepíteni:

```
sudo apt-get install git cmake
```

A programok telepítése után le kell tölteni a szerver forráskódját, ezután pedig le kell fordítani a programot. A lefordítás a 800MHz-es sebességen futó Pi-m esetén 1-1,5 óra körül lehetett, úgyhogy közben elfoglalhatjuk magunkat mással. A letöltéshez és fordításhoz szükséges parancsok:

```
git clone https://github.com/mc-server/MCServer
cd MCServer_
cmake . -DCMAKE_BUILD_TYPE=RELEASE && make
```

A fordítás után a mappában meg fog jelenni egy MCServer mappa. Ez a lefordított szervert tartalmazza. A szerver a következő parancs segítségével indítható el:

```
./MCServer
```

A szerverből a CTRL+C gombok segítségével tudunk kilépni. Első betöltéskor elkezd generálni a program egy világot. A világ generálása után kiléphetünk a szerverből, mivel ezzel együtt létrejönnek a konfigurációs fájlok is, amiket módosíthatunk.

Az fő beállításokat a settings.ini fájl tartalmazza. Ezen fájl legfontosabb beállításai a szerver neve, a játékosok száma, valamint az azonosítás típusa. A külső netes azonosítást érdemes kikapcsolni, ha csak helyi hálózati szerverünk lesz, mivel így internet nélkül is lehet játszani a szerveren. A játékosok számát érdemes 10-re vagy kevesebbre állítani a túlterhelés elkerülése érdekében. Az általam használt settings.ini fájl tartalma a következő:

```
[Server]
Description=MCServer - in C++!
MaxPlayers=10
HardcoreEnabled=0
Port=25565
PortsIPv6=25565
DefaultViewDistance=10

[RCON]
Enabled=0

[Authentication]
Authenticate=0
Server=session.minecraft.net
```

```
Address=/game/checkserver.jsp?user=%USERNAME%&serverId=%SERVERID%

[Worlds]
DefaultWorld=world

[Plugins]
Plugin=Core
Plugin=TransAPI
Plugin=ChatLog

[DeadlockDetect]
Enabled=1
IntervalSec=20

[Physics]
Water=1

[Monsters]
AnimalsOn=1
AnimalSpawnInterval=20
Types=Spider,Chicken,Cow,Pig,Sheep,Squid,Enderman,Cavespider,Creeper,Ske
leton,Slime,Spider,Zombie
```

A játékmód világonként szabható meg, úgy mint a játékmód. A világ beállításai a világ mappájában található world.ini szerkesztésével módosíthatóak.

A felhasználóknak jogosultságok is adhatóak. Kitüntethetünk moderátor felhasználókat, akik más játékosokat el tudnak távolítani a játékból, illetve több mindent tudnak megtenni, mint egy hagyományos játékos. A játékosok és a csoportok adminisztrációja a users.ini és a groups.ini fájl módosításával tehető meg.

A konfigurációs fájlok beállításának lehetőségeiről a program wiki oldalán érdemes tájékozódni, ami a http://www.mc-server.org/wiki/doku.php?id=start címen lelhető fel.

Amennyiben mindent jól csináltunk, akkor a szerverünk játékra kész. A szerver böngészőben fel kellene bukkannia, vagy manuálisan hozzá kell adni a szerver címének és portjának megadásával.

23. PWM moduláció

A RaspberryPi-n a 18-as GPIO kimenet képes hardveresen PWM jelek előállítására, ami nem sok és alapértelmezetten nincs szoftver telepítve a raspbian disztribúcióban, amely képes ezt elérni.

Szoftveres megoldásokkal bármelyik láb PWM képessé változtatható, azonban ezen megoldások hátránya az, hogy ha a saját programunkból állítjuk elő a PWM jeleket, akkor:

- *Az alkalmazás kilépésekor megszűnnek a PWM jelek*
- *Több mint valószínű, hogy a jel generálást külön szálon kell futtatni, ami bonyolítja az alkalmazás felépítését programozási nyelvtől függően*
- *A feladat ütemező bezavarhat a jel generálásba*
- *A processzort nagy frekvenciás jel előállítása megterheli.*

Ezen problémák kivédhetők kétféleképpen is. Vagy dedikált hardvert használunk PWM jel előállítására, vagy szoftveresen módosítjuk egy kicsit a Pi rendszerét.

A szoftveres módosítás abból áll, hogy egy speciális programot kell telepíteni a Pi-re, amely a 18-as GPIO-n működő PWM jelgenerátor működését terjeszti ki további 8 GPIO kimenetre, így a 4, 17, 18, 21, 22, 23, 24, 25 GPIO kimenetek is rendelkezni fognak PWM képességekkel.

A szoftver manuálisan telepíthető forráskódból, a következő parancsok segítségével telepíthető:

```
sudo apt-get install autoconf
git clone https://github.com/sarfata/pi-blaster.git
cd pi-blaster
./autogen.sh
./configure
make
sudo make install
```

A frekvencia és a lépésköz konfigurálásához a forrásfájlt kell módosítani. A pi-blaster.c fájlban a következő két sor határozza meg a frekvenciát és a periódus időt:

```
#define CYCLE_TIME_US           100000
#define SAMPLE_US               10
```

Ezen beállítás mellett a frekvencia 100Hz 1000 lépéssel, ami jó közelítéssel tekinthető 10 bites felbontásnak. A lépésköz ideje 2-ig levihető gond nélkül, és a frekvencia is emelhető. 1Khz mellett 500 lépés adódik, ami jó közelítéssel 9 bites pontosságnak felel meg. Ezen beállításhoz a módosított értékek:

```
#define CYCLE_TIME_US           1000
#define SAMPLE_US               2
```

Ezen konfigurációval LED-es fényforrások és szervomotorok is vezérelhetőek gond nélkül. A jelenlegi PWM frekvenciáról és lépésközről a következő parancs kiadásával tájékozódhatunk:

```
sudo ./pi-blaster
```

A program telepítése után automatikusan el fog indulni rendszer indításkor és egy fájl felületen keresztül bármilyen programozási nyelvből vezérelhető, amely képes fájl írásra és olvasásra.

A vezérlő fájl a dev mappában pi-blaster néven található. Ebbe kell beleírni a vezérlő információkat. A vezérlő információkat GPIO szám = érték formában kell a fájlba írni.

Például, ha 8%-os kitöltési tényezőjű PWM jelet szeretnénk a 17-es GPIO kimeneten, akkor a következő parancsot kell kiadni:

```
echo ”17=0.08” > /dev/pi-blaster
```

A PWM beállítása kimenetként konfigurálja az adott GPIO portot. Ha bemenetként vagy más célra szeretnénk használni egy korábban konfigurált lábat, akkor előbb el kell azt engedni. Ez a release paranccsal tehető meg, ami után szóközzel elválasztva a GPIO számot kell megadni. Az előző példában bekapcsolt PWM jel generálás az alábbi parancs segítségével állítható le:

```
echo ”release 17” > /dev/pi-blaster
```

A PWM jel generálás a SOC PWM időzítőjét alkalmazza. Mivel a Pi hangkimenete is PWM alapú, így ha PWM jeleket generálunk, akkor a hang lejátszás akadozni fog, vagy egyáltalán nem fog működni a választott kimenettől függően.

A probléma kiküszöbölhető, ha –pcm paranccsal indítjuk a pi-blaster programot. Ebben az esetben a hangkártya kimenetének időzítőjét használja jel generálásra a program. Ez egy kicsivel több CPU terhelést fog okozni, de párhuzamosan megy a zene lejátszás és a PWM kimenet kezelés.

24. Grafikus asztal távoli elérése RDP protokollal

Az RDP protokoll a Microsoft alapértelmezett távoli asztali kapcsolat protokollja. Ebből adódóan alapértelmezetten telepítve van az RDP kliens XP óta minden Windows rendszerre. Így a Pi esetén, ha távoli gépünkön Windows rendszert használunk, akkor a legkézenfekvőbb megoldás az, ha egy RDP szervert telepítünk a Pi-re.

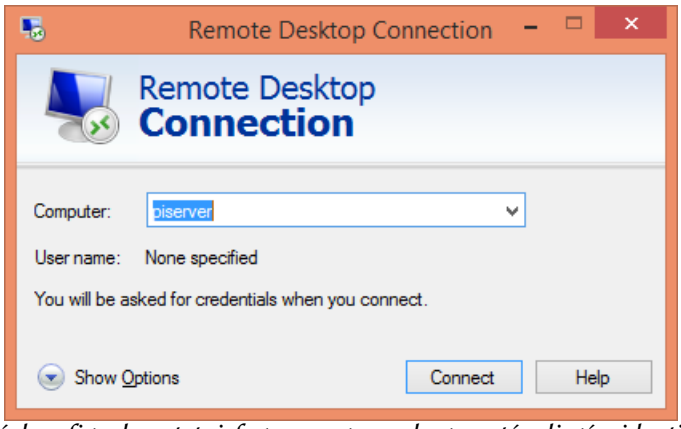
Ez egy parancs segítségével megtehető:

```
sudo apt-get install xrdp
```

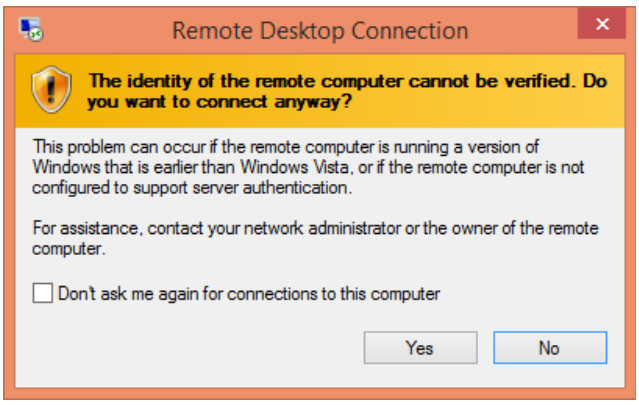
A program telepítése után automatikusan el fog indulni minden rendszerindításkor. Érdemes a grafikus memória méretét 64MB-ra vagy többre állítani, mivel kevesebb memóriával hajlamos használhatatlanná válni a grafikus felület az akadások miatt.

Az RDP protokoll kliens neve Távoli asztalkapcsolat magyar nyelvű Windows rendszereken, angol rendszereken Remote Desktop connection néven érhető el.

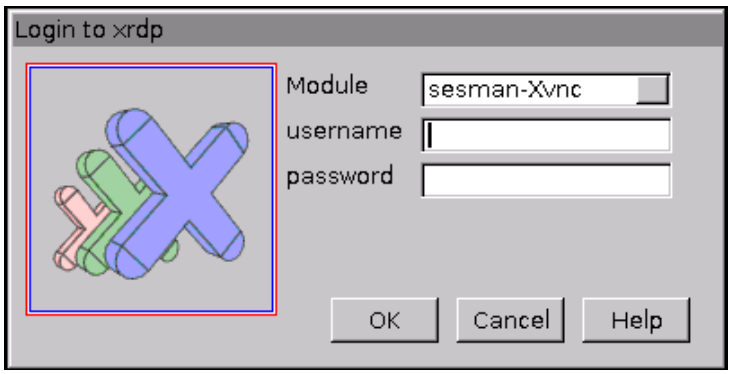
A kliens elindítása után a Pi IP címét kell beírni, vagy ha a samba szerver is telepítve van, akkor használható a gép dns címe is.



Az első kapcsolódáskor figyelmeztetni fog a program, hogy a távoli gép identitása nem ellenőrizhető. Ez egy Vista óta létező Windows szolgáltatás, jelen esetben nem fontos a megléte, így folytatjuk a kapcsolódást. Egy jelölő négyzet segítségével beállítható, hogy többet ne tegye fel ezt a kérdést.



A kapcsolódás után egy XRDP bejelentkező ablak fog bennünket fogadni, ahol be tudunk jelentkezni a gépre. A modul lenyíló menüben a sesman-Xvnc legyen kiválasztva, más beállítás esetén a bejelentkezés nem fog működni.


Login to xrdp
Module
sesman-Xvnc
username
password
OK
Cancel
Help

25. USB webkamerák kezelése

A RaspberryPi esetén nem csak a hivatalos kamera lapot használhatjuk fényképek készítésére, hanem egy általános USB webkamerát is. A hivatalos kameramodul előnye, hogy a GPU kezeli közvetlenül, ezáltal nem terheli a processzort a videók és képek készítése. USB kamerák esetén ez a gyorsítás nem érhető el.

A kamera típusa és márkája majdnem mindegy, mivel a legtöbb kamera vezérlő támogatott, de előfordulhat az olcsóbb, névtelen kamerák esetén, hogy a rendszer nem ismeri fel őket. Ez esetben meg lehet próbálni a rendszert frissíteni, de ebben az esetben érdemesebb egy másik kamerával próbálkozni. Arról, hogy a rendszer felismerte-e a kamerát az lsusb parancs segítségével tájékozódhatunk.

Képek készítéséhez az fswebcam programra lesz szükségünk, amely a következő parancs segítségével gyorsan telepíthető:

```
sudo apt-get install fswebcam
```

A program telepítése után képet könnyen készíthetünk az alábbi parancs segítségével:

```
fswebcam teszt.jpg
```

A program kép mentésekor egy időbélyeget nyom a készített képekre, mint ahogy az az alábbi ábrán látható. Az időbélyeg készítése kikapcsolható az alábbi kapcsoló megadásával:

```
--no-banner
```

323. Ábra: Az fswebcam által készített kép

A program számos kapcsolót és beállítási lehetőséget biztosít. A legfontosabb paraméterek és beállítási lehetőségek:

```
--top-banner
```

Az időbélyeget a kép tetejére helyezi az alja helyett

```
--title ”leírás”
```

Az időbélyegen megjeleníti az idézőjelek között meghatározott szöveget

```
--no-timestamp
```

Az időbélyeg sávról leveszi az időbélyeget.

```
-d /dev/video0
```

Eszköz meghatározása. Akkor hasznos, ha egyszerre több webkamerát akarunk kezelni.

```
--rotate 90
```

A készített kép elforgatása 90 fokkal. A kép elforgatható 90, 180 vagy 270 fokkal.

```
--png 1 ki.png
```

A kimenet PNG formátumra állítása 1-es tömörítési faktorral. A tömörítési faktor 1 és 9 között változhat, a kép minőségére nincs hatással, csak a kimeneti fájlméretre, mivel a PNG veszteségmentes formátum.

```
--jpeg 80 ki.jpg
```

JPG kimeneti formátum alkalmazása 80-as minőséggel. A minőség 0 és 95 között változtatható. A 95 jelképezi a legjobb minőséget, míg a 0 a legrosszabbat.

```
-r 640x480
```

A kimeneti kép mérete 640x480 legyen. A kimeneti kép mérete eltérhet a beállítottól, amennyiben a kamera nem támogatja a kívánt felbontást. Ebben az esetben a kért mérethez legközelebb álló méretet fogja alkalmazni a program.

Videók rögzítése

Videófelvételek készítéséhez az avconv program szükséges. Ez könnyen telepíthető az alábbi parancs segítségével:

```
sudo apt-get install avconv
```

Videó készítésére az alábbi kapcsolók segítségével kell meghívni a programot:

```
avconv -f video4linux2 -r 5 -s 640x480 -i /dev/video0 kimenet.avi
```

A -f kapcsoló a bemeneti videó formátumát határozza meg. A video4linux2 a webkamerák általános formátuma Linux rendszerek alatt. A -r 5 kapcsoló azt állítja be, hogy 5 képkocka/mp sebességgel rögítsen. Ennél többre nem érdemes állítani, mivel a PI CPU-ja igen kis teljesítményű. A -s 640x480 kapcsoló a kimeneti videó méretet állítja be. A -i paraméter pedig a használt eszközt. A kimenet.avi pedig a kimeneti videó fájl neve.

A rögzítési sebesség fordítottan arányos a képmérettel. 320X240-es felbontás mellett 15 képkocka / mp sebességgel tud rögzíteni a pi 800Mhz-es órajelen.

A kimeneti videó xvid formátumú lesz, VLC programmal visszajátszható, illetve a legtöbb videoszerkesztő és konvertáló program is meg fogja tudni nyitni. Avi fájlok esetén érdemes megjegyezni, hogy a kimeneti fájl maximum 4GB méretű lehet. A rögzítést a CTRL+C gombokkal lenyomásával lehet megállítani.

26. Torrent kliens készítése

A RaspberryPi kiválóan alkalmas alacsony fogyasztású BitTorrent kliensnek és szervernek. A BitTorrent egy elosztott rendszerű protokoll, amely segítségével gyorsan tudunk tartalmakat letölteni, mivel az adatot letöltő kliensek egyben felfelé is töltik a már letöltött adatokat, így tehermentesítve a központi szervert.

A közhiedelemmel ellentétben nem csak jogsértő tartalmak letöltésére alkalmas és használt, hanem kiválóan alkalmas szinkronizációs célokra sok gép között.

Számos BitTorrent kliens létezik Linuxra, ugyan úgy, mint Windowsra. A leírás készítésekor a választás a deluge kliensre esett, mivel ez rendelkezik asztali és webes kliens felülettel is. Szerver készítésekor kézenfekvő megoldás a webes kezelőfelület telepítése, mivel ehhez nem kell grafikus felületet futtatni.

A szükséges csomagok könnyen telepíthetők az alábbi parancs segítségével:

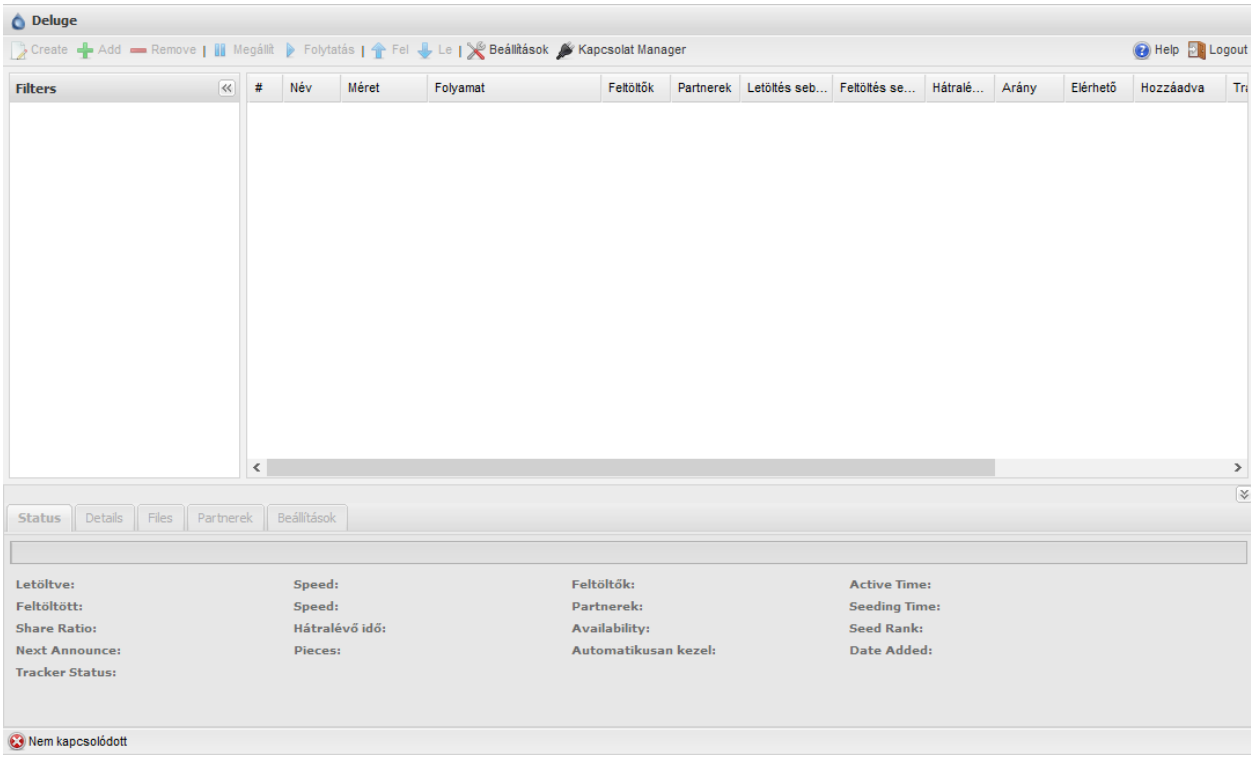
```
sudo apt-get install deluged deluge-webui
```

A szoftverek telepítése után a deluge és a webes felülete nem fog a rendszerrel együtt elindulni. Manuálisan kell elindítani a klienst és a kezelőfelületet az alábbi parancsok segítségével:

```
deluged
deluge-web --fork
```

A fork opció a kezelőfelület futtatása esetén azt teszi lehetővé, hogy az SSH kapcsolat bontása / a terminál bezárása után is futni fog a webes felület.

A webes felület a Pi ip címének megadásával érhető el, azonban nem a 80-as porton, hanem a 8112-es porton. A felület hozzáféréséhez kelleni fog egy jelszó, amely alapértelmezetten deluge.

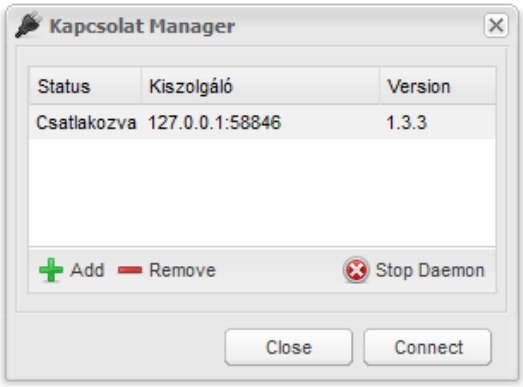


324. Ábra: A deluge webes kezelőfelülete

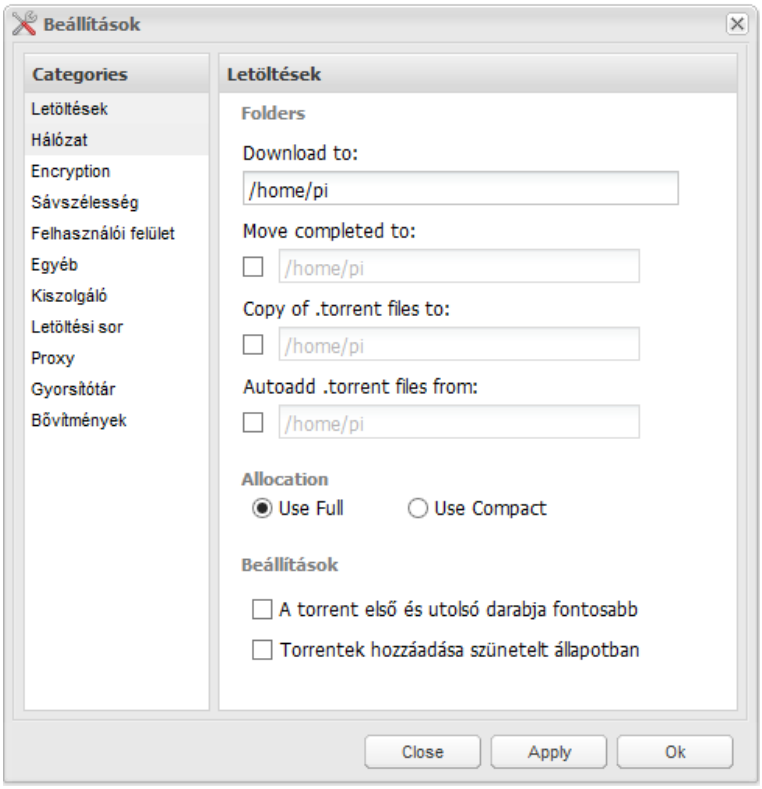
Az első bejelentkezés után a rendszer felkínálja, hogy változtassuk meg az alapértelmezett jelszót. Ezt a beállítások ikonra kattintva a felhasználói felület alatt tudjuk megtenni.

A webes felület egyszerre több szerver adminisztrálására is képes. A kapcsolat manager menüpontban

tudunk szerverekhez hozzákapcsolódni. Alapértelmezetten a listában csak az a szerver van benne, amelyiken elindítottuk a deluge szolgáltatást és a webes felületet. A kliens vezérléséhez először hozzá kell kapcsolódni a helyileg futtatott szerverhez. Csak ezután lesz elérhető minden funkció a webes kezelőfelületen.



A kliensen keresztül .torrent fájlokat url megadásával és fájl feltöltésével is megadhatunk, azonban fájlok letöltése és kiszolgálása előtt be kell állítani az alapértelmezett mappákat, hogy hova történjen a letöltés, illetve honnan történjen a megosztás. Ezt a beállítások menüben tudjuk konfigurálni. A beállítások elvégzése után a házi Torrent szerver működésre kész.



326. Ábra: Mappák beállítása

27. HAT készítése

A B+ modell bejelentésével a Pi fejlesztői egy Arduino szerű kiegészítő lap „szabványt" is definiáltak a Pi-hez. Ezeket HAT-nek nevezték el, ami A Hardware Attached on Top szavak rövidítése.

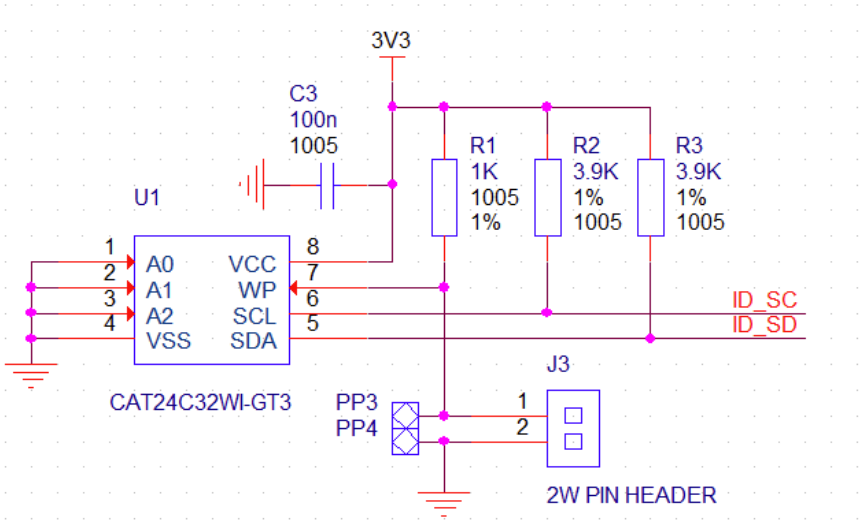
A kiegészítő lapok készítéséről az összes információ megtalálható a https://github.com/raspberrypi/hats címen. Ezen leírás a legfontosabb tudnivalókat fogalalja össze.

Minden kiegészítő lapnak rendelkeznie kell egy konfigurációs EEPROM memóriával, amit a rendszer

indításkor beolvas.

Az EEPROM tartalma alapján a rendszer képes automatikusan megfelelően konfigurálni a GPIO portokat, illetve képes további driver fájlok betöltésére is, ha szükséges. Így a felhasználónak nem kell manuálisan vesződni a konfigurálással.

Ezért a GPIO 0 és GPIO 1 lábak (lásd GPIO használata projekt) csak és kizárólag csak az EEPROM számára fenntartottak. Az EEPROM memóriát a következő módon kell bekötni:



Az EEPROM programozásához és a megfelelő adatok beletöltéséhez és letöltéséhez készítettek programokat is, amelyek a forrása a https://github.com/raspberrypi/hats/tree/master/eepromutils címről szerezhető be. A program telepítése:

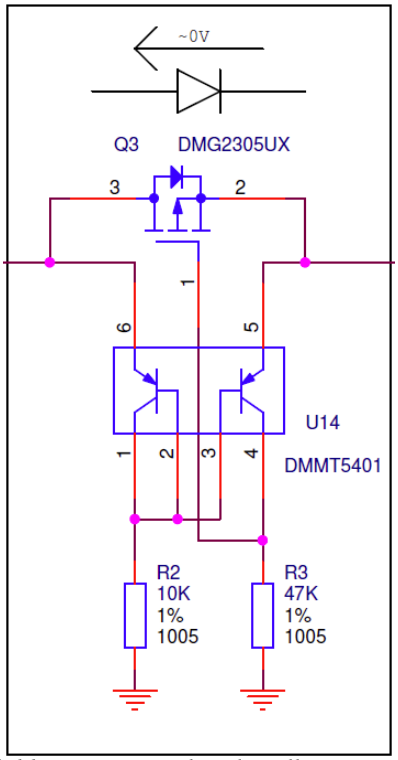
```
git clone https://github.com/raspberrypi/hats.git
cd hats/eepromutils
./make
```

Elektromos követelmények

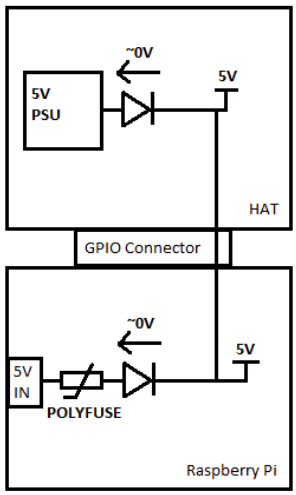
HAT panelek esetén lehetséges HAT-en keresztül a Pi táplálása. Ebben az esetben a betáplálást a 2-es és 4-es tüskén keresztül lehet megtenni. Ezen lábak közvetlenül az 5V és föld pontokra csatlakoznak, így 5V feszültséget kell a lábakra juttatni. A maximális engedélyezett eltérés 5%.

Ha a Pi-t ezen pontokról tápláljuk, akkor a külső tápegység áramkörnek legalább 1.3A áramot kell biztosítania a Pi számára. Valamint a külső tápegység bejövő oldala elé tenni kell egy védődiódát. Ez azt a célt szolgálja, hogy ha két tápegység van csatlakoztatva a lapra (Micro USB és HAT visszatáplálás), akkor az egyik tápegység ne táplálja vissza a másikat.

A Pi microUSB bemenete egy 0v feszültségeséses „diódával" van védve. A tervezői dokumentáció alapján a külső tápegység bemenetét is érdemes egy ilyen megoldással védeni. A megoldás kapcsolási rajza az alábbi ábrán látható:



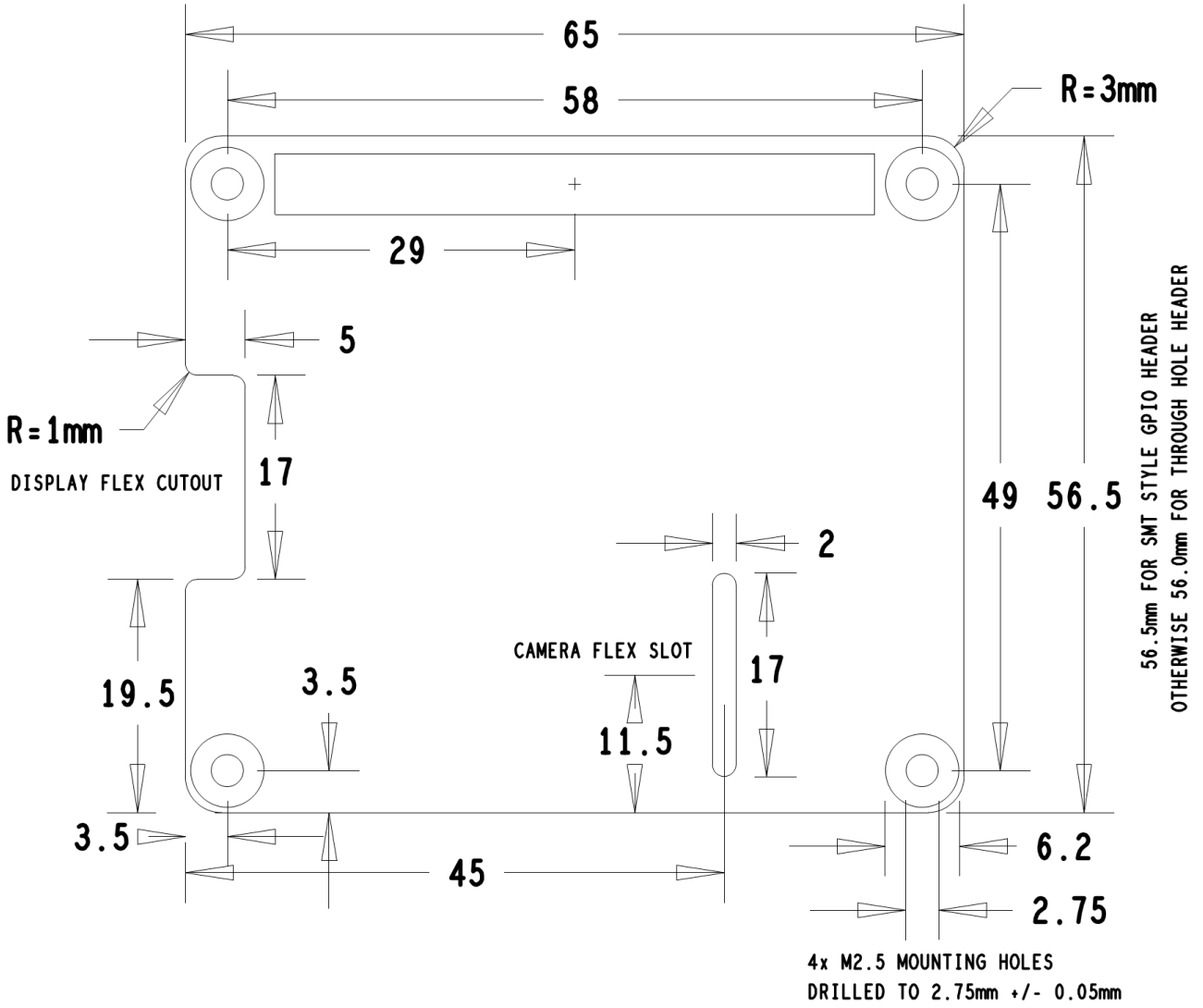
Amennyiben a tápegység legalább 2A árammal tudja ellátni a Pi-t, akkor az EEPROM konfigurációban engedélyezhető az USB nagy áramú mód, amely lehetővé teszi nagyobb fogyasztású USB eszközök használatát kiegészítő HUB nélkül.



Mechanikai követelmények

A HAT lapok méretezése az alábbi rajzon látható. A rajzon nem látszik, de a GPIO csatlakozónak a kiegészítő lapot legalább 12mm magasba kell emelnie az alap pi laptól. Ez a minimális magasság azért szükséges, hogy a kiegészítő panel ne érintkezzen az USB és Ethernet portokkal.

A kamera és a DSI csatlakozó számára ajánlatos kimarásokat készíteni, hogy a kiegészítő lap segítségével is használhatóak legyenek.



24. Verilog projektek

1. kombinációs hálózatok megvalósítása

Bonyolult, sok bemenetes kombinációs hálózatok esetén jobban járunk, ha a hálózat működését Verilog nyelven implementáljuk. Tételezzük fel, hogy adott egy három változós logikai függvény, amely valamilyen vezérlést valósít meg és két kimenettel rendelkezik. A függvény esetén ismerjük az igazság táblázatot, amely a következőképpen néz ki:

A	B	C	Q1	Q2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

126. Táblázat: A vezérlési táblázat

Az igazság táblázat alapján felírható a Q1 és Q2 kimenet függvényeinek teljes alakja, amely alapján már le is írható a hálózat működése, mivel a legtöbb verilog fordító a függvényeket egyszerűsíteni fogja, de nem biztos, hogy hazárd mentesíti is. A teljes függvény alak bekódolása igen hosszú lehet, így jobb ha előre egyszerűsítjük a függvényeinket.

A példában szereplő Q1 és Q2 hazárd mentes vezérlőfüggvényei:

$$Q1 \equiv A \cdot B + B \cdot \overline{C} + \overline{A} \cdot \overline{C} + \overline{A} \cdot \overline{B}$$
$$Q2 \equiv A + B \cdot C$$

Ez alapján a hálózat működését leíró Verilog kód:

```
module vezerles(a, b, c, q1, q2);
      input a, b, c;
      output q1, q2;

      assign q1 =  (a & b) | (b & ~c) | (~a & ~c) | (~a & ~b);
      assign q2 = a | (b & c);
endmodule
```

*Kombinációs hálózat működésének leírására lehetőségünk van a **table** kulcsszó használatával is. A table kulcsszó segítségével lehetőség van bevinni a kombinációs hálózat teljes igazság táblázatát, amit majd a fordító dolgoz fel és hazárd mentesít.*

A table kulcszó csak primitív modulok definiálásakor használható. A primitív modul egy egyszerű logikai hálózatot jelöl. Ha a table kulcsszóval adjuk meg a kimeneteket, akkor egy primitív modul csak egy kimenetet írhat le.

```
primitive kimenet1(q1, a, b, c)
      output q1;
      input a, b, c;

      table
```

```
      //a b c q1
      0 0 0 : 1;
      0 0 1 : 1;
      0 1 0 : 1;
      0 1 1 : 0;
      1 0 0 : 0;
      1 0 1 : 0;
      1 1 0 : 1;
      1 1 1 : 1;
      endtable
endprimitive

primitive kimenet2(q2, a, b, c)
      output q2;
      input a, b, c;

      table
      //a b c q2
      0 0 0 : 0;
      0 0 1 : 0;
      0 1 0 : 0;
      0 1 1 : 1;
      1 0 0 : 1;
      1 0 1 : 1;
      1 1 0 : 1;
      1 1 1 : 1;
      endtable
endprimitive

module vezerles(a, b, c, q1, q2)
      input a, b, c;
      output q1, q2;

      kimenet1(q1, a, b, c);
      kimenet2(q2, a, b, c);

endmodule;
```

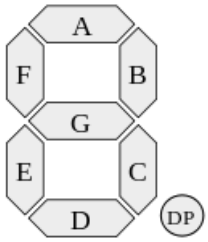
Primitív modul definiálásakor a kapcsolódási pontok felsorolásában mindig a kimenetnek kell az elsőnek lennie. A táblázatban a bemenetek sorrendjét a primitív input utasításának sorrendje határozza meg.

A primitív a modulban úgy használható, mint egy másik modul, vagyis meg kell adni a kapcsolódási pontokat és már használatba is van véve.

2. 4 bites - 7 Hétszegmenses dekódoló

A legtöbb kereskedelmi forgalomban kapható hétszegmenses dekódolóval az a baj, hogy csak BCD bemenetek fogadására képesek, az-az, ha a bemenet 10 vagy fölé kerül, akkor az áramkör nem jelenít meg semmit a kimeneten, pedig a hétszegmenses kijelzőkkel meg lehet jeleníteni hexadecimális számjegyeket is.

A tervezéshez felhasznált igazságtáblázat és kijelző kiosztás:



Bemenet	Kimenetek							Dekódolt kimenet (g= LSB)
	a	b	c	d	e	f	g	
1	1	1	1	1	1	1	0	0x7E
1	0	1	1	0	0	0	0	0x30
2	1	1	0	1	1	0	1	0x6D
3	1	1	1	1	0	0	1	0x79
4	0	1	1	0	0	1	1	0x33
5	1	0	1	1	0	1	1	0x5B
6	1	0	1	1	1	1	1	0x5F
7	1	1	1	0	0	0	0	0x70
8	1	1	1	1	1	1	1	0x7F
9	1	1	1	1	0	1	1	0x7B
A	1	1	1	0	1	1	1	0x77
b	0	0	1	1	1	1	1	0x1F
C	1	0	0	1	1	1	0	0x4E
d	0	1	1	1	1	0	1	0x3D
E	1	0	0	1	1	1	1	0x4F
F	1	0	0	0	1	1	1	0x47

127. Táblázat: Dekódolási táblázat

A hálózat megvalósítható lenne úgy is, hogy felírjuk a vezérlő függvényeket, majd ez alapján kódoljuk le, de jelen esetben egyszerűbb a függvényt leírni a ? : operátor segítségével.

```
module bin2hetszegmens(input_bin, output_segments);
      input [3..0] input_bin;
      output [6..0] output_segments;

      assign output_segments =
            (input_bin == 4'h0) ? 7'h7E :
```

```
        (input_bin == 4'h1) ? 7'h30 :
        (input_bin == 4'h2) ? 7'h6D :
        (input_bin == 4'h3) ? 7'h79 :
        (input_bin == 4'h4) ? 7'h33 :
        (input_bin == 4'h5) ? 7'h5B :
        (input_bin == 4'h6) ? 7'h5F :
        (input_bin == 4'h7) ? 7'h70 :
        (input_bin == 4'h8) ? 7'h7F :
        (input_bin == 4'h9) ? 7'h7B :
        (input_bin == 4'hA) ? 7'h77 :
        (input_bin == 4'hB) ? 7'h1F :
        (input_bin == 4'hC) ? 7'h4E :
        (input_bin == 4'hD) ? 7'h3D :
        (input_bin == 4'hE) ? 7'h4F :
        (input_bin == 4'hF) ? 7'h47 : 7'bxxxxxxx;

endmodule;
```

A fenti kódrészlet a bemeneti adat alapján feltételesen társít értéket a kimenethez. Mivel teljes dekódolás van, ezért a kimenet sosem kerül át az utolsó állapotba. Az utolsó állapot azért került a hálózatba, hogy a kód hiba nélkül le tudjon fordulni.

3. Multiplexer és demultiplexer

Mikrovezérlős projektjeink során előfordulhat, hogy szükségünk lenne egy 16 vagy még több bemenettel rendelkező multiplexerre. Ebben az esetben a multiplexert összerakhatjuk diszkrét logikai elemekből, vagy elkészítjük egy CPLD áramkör segítségével.

Az alábbi verilog kód egy 32 bemenettel rendelkező multiplexer megvalósítást mutat be. A megvalósítás szintén a ? : operátoron alapul folyamatos értékadást felhasználva.

A kód alapján a multiplexer szabadon bővíthető tovább.

```
module multiplexer(q, data, select);
	output q;
	input [31:0] data;
	input [4:0] select;

	assign q =
		(select == 5'h00) ? data[0] :
		(select == 5'h01) ? data[1] :
		(select == 5'h02) ? data[2] :
		(select == 5'h03) ? data[3] :
		(select == 5'h04) ? data[4] :
		(select == 5'h05) ? data[5] :
		(select == 5'h06) ? data[6] :
		(select == 5'h07) ? data[7] :
		(select == 5'h08) ? data[8] :
		(select == 5'h09) ? data[9] :
		(select == 5'h0A) ? data[10] :
		(select == 5'h0B) ? data[11] :
		(select == 5'h0C) ? data[12] :
		(select == 5'h0D) ? data[13] :
		(select == 5'h0E) ? data[14] :
		(select == 5'h0F) ? data[15] :
		(select == 5'h10) ? data[16] :
		(select == 5'h11) ? data[17] :
		(select == 5'h12) ? data[18] :
		(select == 5'h13) ? data[19] :
		(select == 5'h14) ? data[20] :
		(select == 5'h15) ? data[21] :
		(select == 5'h16) ? data[22] :
		(select == 5'h17) ? data[23] :
		(select == 5'h18) ? data[24] :
		(select == 5'h19) ? data[1] :
		(select == 5'h1A) ? data[0] :
		(select == 5'h1B) ? data[1] :
		(select == 5'h1C) ? data[0] :
		(select == 5'h1D) ? data[1] :
		(select == 5'h1E) ? data[0] :
		(select == 5'h1F) ? data[1] : 1'bx;
endmodule
```

A demultiplexer a multiplexer fordítottja. A hálózat működése felfogható úgy, hogy a bemenetre adott jelet annyi értékkel kell eltolni balra, amennyi a kimenetválasztóra adott jel értéke. Ez alapján egyszerű eltolás operátorral megvalósítható a demultiplexer kód verilog nyelven.

Az alábbi példa kód egy 32 kimenettel rendelkező demultiplexert valósít meg:

```
module demultiplexer(q, in, select);
      input in;
      input [4:0] select;
      output [31:0] q;

      assign q = in << select;
endmodule
```

4. Dekóderek és enkóderek

A dekóderek hasonlóan a multiplexerekhez a legegyszerűbben ? : operátor segítségével valósíthatóak meg, azonban ha nagy a bitszélesség, akkor a sok egymásba ágyazott ? : operátoros megvalósítás nem a leghatékonyabb.

Sokkal hatékonyabb a megvalósítás, ha case utasítással van megvalósítva a dekóder. A case csak always blokkban használható, ami elsősorban sorrendi hálózatok megvalósítására alkalmas. Azonban nem csak sorrendi hálózatok leírására lehet használni.

Az alábbi példakód egy 3 bites címdekóder megvalósítást mutat be:

```
module cimdekoder(q, in);
	input [2:0] in;
	output [7:0] q;

	reg [7:0] q;

	always @(in)
	begin
		case (in)
			3'b000: q = 8'b0000_0001;
			3'b001: q = 8'b0000_0010;
			3'b010: q = 8'b0000_0100;
			3'b011: q = 8'b0000_1000;
			3'b100: q = 8'b0001_0000;
			3'b101: q = 8'b0010_0000;
			3'b110: q = 8'b0100_0000;
			3'b111: q = 8'b1000_0000;
			default: q = 8'bxxxx_xxxx;
		endcase
	end
endmodule
```

Hasonló logikával bármilyen kód átalakító lekódolható. A fenti kódrészletből azért nem lesz tárolót tartalmazó sorrendi hálózat, mert az always blokk vezérlő feltételében a vezérlő jel nem él vezérelt, hanem szint vezérelt, valamint a vezérlő szerkezetben minden bemenet szerepel. Ha ezen előírásokat betartjuk, akkor always szerkezettel is kódolhatunk kombinációs hálózatokat.

Az enkóderek hasonló logikával működnek, mint a dekóderek. Digitális technikában azonban az enkóder alatt általában egy olyan hálózat értendő, amely n darab bemenettel rendelkezik, amelyek közül egy időben csak egy lehet aktív. A kimeneten az aktív bemenetnek a sorszáma jelenik meg. Probléma akkor van, ha több bemenet is egyszerre aktív. Ebben az esetben két megközelítés is szóba jöhet a hiba kezelése szempontjából.

Létrehozhatunk egy speciális kimenetet, amely azt adja meg, hogy a kimeneti kód helyes-e. Ebben az esetben egy olyan függvényt kell megalkotni, amely képes detektálni, ha egynél több egyes van a bemeneteten. Ez a függvény nem lesz egyszerűsíthető, ebből adódóan sok logikai elemet használna fel.

Ezért a gyakorlatban az enkóder áramkörök prioritásos enkóderek. Ez azt jelenti, hogy ha több bemenet is egyszerre aktív, akkor a legnagyobb helyiértéken lévő bemenet sorszáma kerül kódolásra. Ez a megközelítés azért előnyös, mert don't care értékek bevezetését jelenti a hálózatba.

Az alábbi példa kódrészlet egy 8 bemenetű 3 bites bináris kimenetű enkódert valósít meg:

```
module prioritas_enkoder(q, in);
	input [7:0] in;
	output [3:0] q;

	reg [3:0] q;
```

```
      always @(in)
      begin
            casex (in)
                  8'b1xxx_xxxx: q = 3'b111;
                  8'bx1xx_xxxx: q = 3'b110;
                  8'bxx1x_xxxx: q = 3'b101;
                  8'bxxx1_xxxx: q = 3'b100;
                  8'bxxxx_1xxx: q = 3'b011;
                  8'bxxxx_x1xx: q = 3'b010;
                  8'bxxxx_xx1x: q = 3'b001;
                  8'bxxxx_xxx1: q = 3'b000;
                  default: q = 3'bxxx;
            endcase
      end
endmodule
```

A fenti kódrészletben a casex utasítás segítségével van megvalósítva az enkóder. A default sor elhagyható lenne, mivel minden esetet kódoltunk. A default sor csak olyan esetekben szükséges mindenképpen, ahol nem teljes a dekódolás vagy kódolás.

5. Aritmetikai és logikai egység

A Processzrokban az aritmetikai és logikai egység egy olyan kombinációs hálózat, amely 2 darab n bites bemenettel rendelkezik és egy m bites művelet választóval. A beadott műveletkód alapján az n bites kimeneten az elvégzett művelet jelenik meg.

Egy Ilyen hálózat megvalósítása nagy kihívást jelent, ha kombinációs logikával kívánjuk megvalósítani a hálózatot, de Verilog nyelven könnyen megvalósítható. Az alábbi kódrészlet egy 8 bites 16 művelet elvégzésére alkalmas aritmetikai egységet valósít meg.

```
module ALU(out, op1, op2, code)
      output [7:0] out;
      input [7:0] op1;
      input [7:0] op2;
      input [4:0] code;

      reg [7:0] out;

      always @(op1 or op2 or code)
      begin
            case (code)
                  4'h0: out = op1 + op2;
                  4'h1: out = op1 - op2;
                  4'h2: out = op1 * op2;
                  4'h3: out = op1 / op2;
                  4'h4: out = op1 & op2;
                  4'h5: out = op1 | op2;
                  4'h6: out = ~op1;
                  4'h7: out = ~op2;
                  4'h8: out = op1 ^ op2;
                  4'h9: out = op1 ~^ op2;
                  4'hA: out = op1 ~& op2;
                  4'hB: out = op1 ~| op2;
                  4'hC: out = op1 << op2;
                  4'hD: out = op1 >> out2;
                  4'hE: out = op1+1;
                  4'hF: out = op1-1;
            endcase
      end
endmodule
```

25. Vegyes projektek

Ezen fejezetben olyan projektek kaptak helyet, amelyek a könyv más szekcióiba nem igen fértek volna be. Így kompromisszumok kötése helyett úgy döntöttem, külön fejezetbe teszem őket.

1. Adafruit Trinket használata

A Trinket egy igen apró univerzális mikrovezérlő, amely sok célra használható, feltéve ha működik. A problémája a lapnak, hogy nem rendelkezik hardveres USB támogatással, csak szoftveressel. Ebből adódóan egyes gépeken nem hajlandó működni USB eszközként. Ennek a tápfeszültség ellátáshoz is köze van. A 3,3V-os változat gond nélkül működik mindkét gépemen, de az 5V-os csak akkor működik az asztali gépemen, ha elég rövid kábelt használok hozzá. A Laptopomon nem sikerült működésre bírnom.

Az USB problémája a kód feltöltést is érinti. Az eszköz USBTinyISP programozóként látszik a buszrendszeren, ha bootloader üzemmódban van. A bootloader üzemmód úgy érhető el, hogy a reset gombot megnyomjuk. Ekkor 10Mp időre feltöltő üzemmódba vált. Ekkor az Arduino környezet segítségével feltölthető a program. Program feltöltés közben ha valami hiba lép fel, akkor jó eséllyel sérül a bootloader is, mivel a lapon alkalmazott ATTiny vezérlő nem rendelkezik védett bootloader kód szegmenssel.

Ebben az esetben újra kell írni bele a bootloader-t. Ehhez egy Arduino-ra lesz szükségünk, ami egy program rátöltéssel képes lesz újraéleszteni a trinket vezérlőnket, vagy ha nem élesszük raja újra a bootloadert, akkor egy ISP programozó (amit az Arduino-ból is tudunk barkácsolni) segítségével tölthetünk rá kódot.

Ezen okokból kifolyólag a Trinket nem egy tipikusan kezdőknek szánt vezérlő. Használatához hatalmas türelem is kell.

Szükséges szoftverek telepítése és beszerzése

A Trinket alapértelmezetten nem támogatott az Arduino környezetben, mivel nem is hivatalos Arduino lap. Ezért támogató szoftvereket kell telepíteni hozzá. A gyors és egyszerű mód az, ha letöltjük az Adafruit módosított Arduino környezetet, amely rendelkezik Trinket támogatással.

Ez a http://adafruit-download.s3.amazonaws.com/Adafruit%20Arduino%201.05%20-%20Win%2011-11-13%20.zip címről szerezhető be.

Amennyiben a meglévő Arduino kötelezettünket szeretnénk kiegészíteni támogatással, akkor pár dolgot le kell töltenünk és be kell másolnunk a környezetbe.

Első körben szükségünk lesz illesztőprogramra, hogy a gép helyesen felismerje az eszköz. Ez a http://learn.adafruit.com/system/assets/assets/000/010/319/original/usbtinyisp_w32_driver_v1.12.zip címről szerezhető be. Windows 8 és újabb rendszereken kicsit macerás a telepítése, mivel digitális aláírással nem rendelkezik.

Ezután a támogató hardver fájlokat kell letöltenünk és bemásolnunk a környezet hardware mappájába: http://learn.adafruit.com/system/assets/assets/000/010/777/original/trinkethardwaresupport.zip?1378321062

Mivel az alkalmazott vezérlő nem a legyorsabb USB emulációban a környezet program feltöltő konfigurációs fájlját ki kell cserélni egy módosítottra, amely nagyobb várakozási időt határoz meg a vezérlőhöz. Így a program feltöltés nem fog hibákkal elszállni. A http://learn.adafruit.com/system/assets/assets/000/010/769/original/avrdude.conf?1378221965 címről letöltött avrdude.conf fájlt az arduino mappán belül található hardwares/tools/avr/etc mappába kell másolni.

Utolsó lépésként a program linkelőt kell frissíteni. Erre azért van szükség, mert az Arduino környezetekben alkalmazott linkelő hibás, ATTiny vezérlők esetén 4KB feletti programok esetén hibás kódot generál. A javított linkelő a következő címről tölthető le:

http://learn.adafruit.com/system/assets/assets/000/010/983/original/WINDOWS_ld.zip?1379343097

A fájlban található ld.exe a \hardware\tools\avr\avr\bin mappába másolandó.

A lépések végigjátszása után a környezet rendelkezik Trinket támogatással. Tesztelni a lapot a klasszikus LED villogtató programmal tudjuk. A Trinket 1-es lábán található a bootloader visszajelző lába, amit használhatunk. A program:

```
void setup()
{
     pinMode(1, OUTPUT);
}

void loop()
{
     digitalWrite(1, HIGH);
     delay(1000);
     digitalWrite(1, LOW);
     delay(1000);
}
```

A lap kiválasztása után az eszközök menüből a programozó menüponton belül az USBTinyISP-t kell kiválasztani. A lapon található reset gomb megnyomása után 10 másodpercünk van a feltöltés gomb megnyomására.

Ha az alábbi hibaüzenetbe futunk bele, akkor két hibalehetőség adódik. Vagy nem voltunk elég gyorsak és az eszköz visszaváltott programozó üzemmódból, vagy a számítógép nem ismerte fel rendesen a vezérlőt

```
avrdude: Error: Could not find USBtiny device (0x1781/0xc9f)
```

Amennyiben a második hibalehetőség áll fent, akkor próbálkozni kell más USB portokkal. Előfordulhat azonban, hogy USB-n keresztül az adott gépen sosem lesz programozható a lap.

Amennyiben sok hibaüzenetet kapunk, akkor is két hibalehetőség van. Vagy nem frissítettük az avrdude.conf fájlt, vagy egyszerűen makacs usb-n keresztül a lap. Ha ebbe a hibába futottunk bele, akkor sajnos a bootloader tuti sérül az eszközön, így azt tölthetjük újra.

A bootloader újratöltéséhez szükségünk lesz egy Arduino Uno lapra és a feltöltő szoftverre, amit a http://learn.adafruit.com/introducing-trinket/repairing-bootloader címről tudunk beszerezni.

A program használata egyszerű, mivel csak fel kell tölteni Uno-ra és a megfelelő lábait a trinket-re kell kötni az alábbi séma alapján:

Arduino láb	***Trinket Láb***
5V	*BAT+*
GND	*GND*
10	*Reset*
11	*0 (MOSI)*
12	*1 (MISO)*
13	*2 (SCK)*

128. Táblázat: Bootloader javító program lábkiosztása

A bekötés után a környezet soros monitorát elindítva egy G üzenet küldése után a bootloader visszamásolódik a vezérlőbe, ezután ismét használható lesz a trinket. Programozóval való feltöltéskor ugyan ezt a bekötést kell alkalmazni.

USB billentyűzeteztet és egeret is képes emulálni az eszköz. Ehhez illesztőkönyvtárakra lesz szükségünk, amelyek a https://github.com/adafruit/Adafruit-Trinket-USB címről szerezhetőek be. A könyvtár függvénynevei megegyeznek a Leonardo esetén bemutatott függvényekkel, csak a kezelő objektum más. A könyvtárak számos példával rendelkeznek az egyszerűbb használat elsajátításához.

2. Bluetooth eszközök vezérlése RaspberryPi-vel

Az Arduino Bluetooth projekt kapcsán említettem, hogy a beszerezhető Bluetooh-Soros átalakítók többségét macerás használni, ha azt akarjuk, hogy ők kezdeményezzék a kommunikációt. Ezért ha több ilyen modultól kell adatot beszerezni és feldolgozni, akkor kézenfekvő megoldás lehet egy RaspberryPi vagy más SOC lap beszerzése, mivel ezek teljes értékű Bluetooth támogatással rendelkeznek és többségük olcsó is.

A konkrét példa, amit ez a projekt megvalósít, az nem más, mint egy Bluetooth-on működő LED fényerő szabályzó rendszer. A rendszer középpontja egy RaspberryPi, ami a kommunikációt irányítja több soros Bluetooth adapterrel ellátott Arduino között.

Az egyik Arduino az érzékelő szerepét tölti be, és semmi mást nem csinál, csak annyit, hogy egy potenciométer által meghatározott analóg értéket szolgáltat a vezérlő számára kérés esetén. A vezérlő a kapott adatot továbbítja a másik Arduino lap felé, ami a kapott értékből egy PWM jelet állít elő, ami alkalmas LED fényerő szabályzásra.

Az érzékelőként programozott Arduno vezérlő kódja:

```
void setup()
{
	Serial.begin(115200);
}

void loop()
{
	while (Serial.available() > 0)
	{
		char c = (char)Serial.read();
		if (c == 'L')
		{
			int val = analogRead(0);
			Serial.println(val);
			Serial.flush();
		}
	}
}
```

A LED-et vezérlő Arduino kódja:

```
#define LAMPA 8

String inString = "";
char rec;
int value = 0;
int pwm = 0;

void setup()
{
	Serial.begin(115200);
	pinMode(LAMPA, OUTPUT);
}
void loop()
{
	while (Serial.available() > 0)
	{
```

```
            rec = Serial.read();
            switch (rec)
            {
                  case '\n':
                  case '\r':
                  case '\0':
                        inString += rec;
                        value = inString.toInt();
                        pwm = map(value, 0, 1023, 0, 255);
                        analogWrite(LAMPA, pwm);
                        inString = "";
                        Serial.println("OK");
                        break;
                  case '0':
                  case '1':
                  case '2':
                  case '3':
                  case '4':
                  case '5':
                  case '6':
                  case '7':
                  case '8':
                  case '9':
                        inString += rec;
                        if (inString.length() > 5) inString = "";
                        break;
            }
      }
}
```

A Lámpa vezérlő kódja azért bonyolultabb, mert a soros kommunikáció során az érzékelő a jelet szövegként továbbítja, amit konvertálni kell, mielőtt használni tudná a rendszer. A konvertálgatás kikerülhető lenne, ha a szenzor az adatokat byte folyamként továbbítaná, viszont ebben az esetben az Arduino környezet soros terminálja hibakérésere nem lenne használható. Másik megoldás Firmata protokoll alkalmazása lehetne, ám jelen esetben a példa egyszerűsége ezt nem indokolta.

A Pi esetén a Bluetooth kezeléséhez szükséges szoftverek alapértelmezetten nincsenek feltelepítve, de az alapértelmezetten elérhető szoftverforrásokból könnyen feltehetőek a szükséges szoftverek az alábbi parancs segítségével:

```
sudo apt-get install bluetooth bluez-utils blueman python-serial
```

A csomagok telepítése el fog tartani egy darabig, utána viszont gond nélkül alkalmazhatunk szinte bármilyen USB-Bluetooth adaptert, amit parancssor segítségével is tudunk kezelni. Mielőtt még konkrét kezelő szoftvert alkotnánk először a Bluetooth kezelésre vonatkozó parancsokat ismertetem.

```
hciconfig
```

Kilistázza az elérhető Bluetooth csatolókat. Ha egy adapter van csatlakoztatva és a rendszer felismerte, akkor a kimenetén egy hci0 szöveget tartalmazó sornak kellene megjelennie.

```
hcitool scan
```

Ezen parancs segítségével tudjuk lekérdezni a Bluetooth kapcsolaton látott eszközök nevét és MAC címét. A kapcsolódás szempontjából a MAC cím lesz a lényeges.

```
sudo bluez-simple-agent hci0 xx:xx:xx:xx:xx:xx
```

Eszköz párosítása. A sok xx az eszköz MAC címét reprezentálja. A parancs kiadása után a PIN kódot kell beírni. Egy parancs segítségével is automatizálható a folyamat:

```
echo <pin> | sudo bluez-simple-agent hci0 xx:xx:xx:xx:xx:xx
```

Ebben az esetben a <pin> helyére a tényleges párosító pin kód beírása kell.

```
sudo rfcomm bind /dev/rfcomm1 xx:xx:xx:xx:xx:xx
```

Létrehozza a soros portot az adott MAC című eszközhöz. A parancs lefutása után a /dev mappában létrejövő rfcomm eszköz egy soros port leíró, amit megnyitva tudunk adatot küldeni Bluetooth-on keresztül. Az eszköznév megválasztása szabad, azonban érdemes az rfcomm előtagot meghagyni, mivel ez utal arra, hogy Bluetooth-on keresztül elérhető soros port az eszköz.

```
sudo rfcomm unbind /dev/rfcomm1 xx:xx:xx:xx:xx:xx
```

Kapcsolat bontása és soros port törlése.

Kezelő szoftver

*Mivel kész kezelő szoftvert nem találtam ezért alkottam egy python modult, amely a fent említett parancsok segítségével megkönnyíti a kommunikáció kezelését. A párosítás automatán történik az eszköz neve alapján, PIN kód megadása nem kell, mivel az eszköz nevéből következtet a program a kódra. Ennek a feltétele az, hogy az eszköz ténylegesen a várt kóddal legyen ellátva. A szenzor tag neve SENSOR1, a LED vezérlő tag neve pedig LAMP1. A lámpák esetén az alkalmazott kódképzési szabály: 1234 + (1111 * lámpa indexe), míg a szenzorok esetén: 1234 + (1010 * szenzor indexe)*

A kezelést megkönnyítő functions modul tartalma:

```
import subprocess
import serial

#Run a System command and return its output lines as tuple
def RunCommand(command):
      p = subprocess.Popen(command,
stdout=subprocess.PIPE,stderr=subprocess.PIPE)
      return p.communicate()

#Run a Command in a Pipe
def RunCommandPipe(command1, command2):
      p1 = subprocess.Popen(command1, stdout=subprocess.PIPE)
      p2 = subprocess.Popen(command2, stdin=p1.stdout, stdout=subprocess.PIPE)
      p1.stdout.close()
      return p2.communicate()


#List Bluetooth Devices
def BTList():
      res = RunCommand(['sudo', 'hcitool', 'scan'])
      lines = res[0].split("\n")
      ret = {}
      for line in lines:
            data = line.strip().split('\t')
            if len(data) > 1:
                  ret[data[1]] = data[0]
      return ret
```

```
#Pair Device
def PairDevice(device, adress):
	basen = 1234
	code = 1234
	if device.startswith("LAMP"):
		index = int(device.replace("LAMP", ""))
		code = basen + index *  1111
	elif device.startswith("SENSOR"):
		index = int(device.replace("SENSOR", ""))
		code = basen + index *  1010
	return RunCommandPipe(['echo', str(code)],  ['sudo', 'bluez-simple-
agent', 'hci0', adress])

def GetPortFromDevice(device):
	if device.startswith("LAMP"):
		index = int(device.replace("LAMP", ""))
		code = index + 10
	elif device.startswith("SENSOR"):
		index = int(device.replace("SENSOR", ""))
		code = index + 20
	return "/dev/rfcomm"+str(code)

#Connect to serial device
def BTConnect(device, adress):
	devfile = GetPortFromDevice(device)
	RunCommand(['sudo', 'rfcomm', 'bind', devfile, adress])

#Disconnect from serial device
def BTDisconnect(device, adress):
	devfile = GetPortFromDevice(device)
	RunCommand(['sudo', 'rfcomm', 'unbind', devfile, adress])

def SendLampData(device,pwmvalue):
	try:
		port = serial.Serial(GetPortFromDevice(device), baudrate=115200,
timeout=4)
		port.write(str(pwmvalue)+"\n")
		val = port.readline()
		port.close()
	except:
		pass




def ReadSensor(device):
	try:
		port = serial.Serial(GetPortFromDevice(device), baudrate=115200,
timeout=4)
		port.write("L\n")
		val = port.readline()
		port.close()
	except:
		return -1
	return int(val)
```

A tényleges szabályzás egy külön modulban van megvalósítva szálkezelés segítségével. A szabályzó szálnak egy eszköz listát kell átadni, illetve egy szótár objektumot, ami a szenzor nevekhez egy listát tartalmaz a szabályozott eszközökről. A szabályzó modul forráskódja:

```
import functions
import threading
import thread
import time

class Controller(threading.Thread):
	def __init__(self, devices, tree, debug):
		self.devices = devices
		self.devtree = tree
		self.debug = debug
		threading.Thread.__init__(self)

	def run(self):
		while 1:
			for i in self.devtree.keys():
				if (self.debug):
					print "Parositas: ", i, self.devices[i]
				functions.PairDevice(i, self.devices[i])
				if (self.debug):
					print "Csatlakozas: ", i, self.devices[i]
				functions.BTConnect(i, self.devices[i])
				time.sleep(0.2)
				sensorval = functions.ReadSensor(i)
				if (self.debug):
					print "Szenzor ertek: ", sensorval
				time.sleep(0.1)
				functions.BTDisconnect(i, self.devices[i])
				time.sleep(0.2)
				for j in self.devtree[i]:
					if (self.debug):
						print "Parositas: ", j, self.devices[j]
					functions.PairDevice(j, self.devices[j])
					if (self.debug):
						print "Csatlakozas: ", j, self.devices[j]
					functions.BTConnect(j, self.devices[j])
					time.sleep(0.2)
					if (self.debug):
						print "adat kuldese: ", j, sensorval
					functions.SendLampData(j, sensorval)
					time.sleep(0.1)
					functions.BTDisconnect(j, self.devices[j])
					time.sleep(0.2)
			time.sleep(0.5)
```

A szabályzó és funkciók modult alkalmazó főprogram forráskódja:

```
import functions
import szabalyzo

if __name__ == "__main__":
	print "Szabalyzo teszt"
	print "Eszkozok listazasa..."
	dev = functions.BTList();
```

```
print "Eszkozok:"
print dev
print "szabalyzas inditasa"
szab = szabalyzo.Controller(dev, {"SENSOR1" : ["LAMP1"]}, True)
szab.run()
```

A szabályzózó objektum példányosításánál alkalmazott utolsó True paraméter a hibakeresést megkönnyítő konzol üzenetek kiírását kapcsolja be. Az üzenetek jobb láthatósága miatt tartalmaz a kód viszonylag sok késleltetést, valamint azért, hogy az egyes kritikus műveletek után a rendszernek legyen ideje végrehajtani a parancsot, mielőtt a következő ráépülő parancs végrehajtódna.

3. Egér és billentyűzet emulálás információ biztonsági szemszögből

Azon mikrovezérlők amelyek képesek Egérként vagy billentyűzetként viselkedni, azok sok jó mellett sok rosszra is alkalmazhatóak. Ezen vezérlők azért olyan veszélyesek, mivel programozható billentyűzetek készíthetőek belőlük, amelyek akár komoly károkat is okozhatnak. Ebben a fejezetben pár károkozásra használható módszer kerül ismertetésre pusztán tájékoztató jelleggel. Az egyes módszerek megvalósítása és alkalmazása jogi következményekkel járhat. Ebből adódóan semmilyen fejelősséget nem vállalok.

Egy minta rosszindulatú program, ami viszonylag ártalmatlan, de felhasználói beavatkozás nélkül nyitja meg a google.com oldalt:

```
void setup()
{
      Keyboard.begin();
}

void loop()
{
      delay(1000);
      Keyboard.press(KEY_LEFT_GUI);
      Keyboard.press('r');
      delay(100);
      Keyboard.releaseAll();
      Keyboard.println("iexplore.exe google.com");
      delay(400);
      while(true);
}
```

A kód azt használja ki, hogy minden operációs rendszer kezelőfelülete rendelkezik speciális billentyűkombinációkkal, amelyek megkönnyítik az operációs rendszer használatát. Ez alól a Windows sem kivétel. A Windows+R gombok megnyomása hatására megjelenik a futtatás ablak, amibe, ha egy program nevét írjuk be, majd utána enter gombot nyomunk, akkor a program elindul.

Ezt kihasználva és azt a tényt, hogy az Internet Explorer a rendszerek többségén telepítve van meg tudunk nyitni weboldalakat.

A kódot feltöltve egy Arduino Leonardo-ra az eszköz csatlakoztatásakor automatikusan meg fog nyílni a google az Internet Explorerben.

A dolog pikantériája, hogy nem csak erre használható a dolog. A példa elég ártalmatlan jelen esetben, de ha parancssort nyit meg, akkor már több rosszindulatú dologra lenne használható. A teljesség igénye nélkül felhasználói beavatkozás nélkül lehetne törölni mappákat és fájlokat is.

Egy másik rosszindulatú alkalmazási lehetőség lehet például az, ha a Leonardo-val készített PS/2 USB billentyűzet átalakítónkat kiegészítjük SD kártya tárhellyel és minden egyes gomblenyomást rögzítünk a kártyára. Ha ezt kelően apró méretűre megtervezzük és megépítjük, akkor van lényegében egy hardveres keyloggerünk, ami ellen szoftveresen nem lehet semmit tenni.

Ennek oka az, hogy a vírusirtó és egyéb védelmi szoftverek nem tudnak belelátni a hardverelemek programjaiba. Az egyetlen egy védelmi megoldás az, hogy kétes helyről származó átalakítókat és billentyűzeteket ne alkalmazzunk. Továbbá irodai környezetben nem árt néha ellenőrizni a gépet felesleges USB eszközök után kutatva.

A legdurvább viszont ezekben a támadási módszerekben az, hogy operációs rendszer függetlenné tehetőek némi módosítással.

26. Nyomtatott áramkör tervezése és gyártása

Az EAGLE igen hasznos és sokoldalú kapcsolási rajz szerkesztő és nyomtatott áramkör tervező program, amely ismerete nagymértékben megkönnyíti a saját mikrovezérlős és egyéb elektronikai alkotásaink megvalósítását.

A program tudása igen nagy, előszeretettel alkalmazzák ipari tervezések során is. Hátránya, hogy nem ingyenes program. Pontosabban ingyenes, de korlátolt. A teljes verzió ára változat függő, a legolcsóbb hobbi célra szánt változat jelenleg 162 Euró áron szerezhető be, Oktatási intézmények számára ingyenes licenc igényelhető.

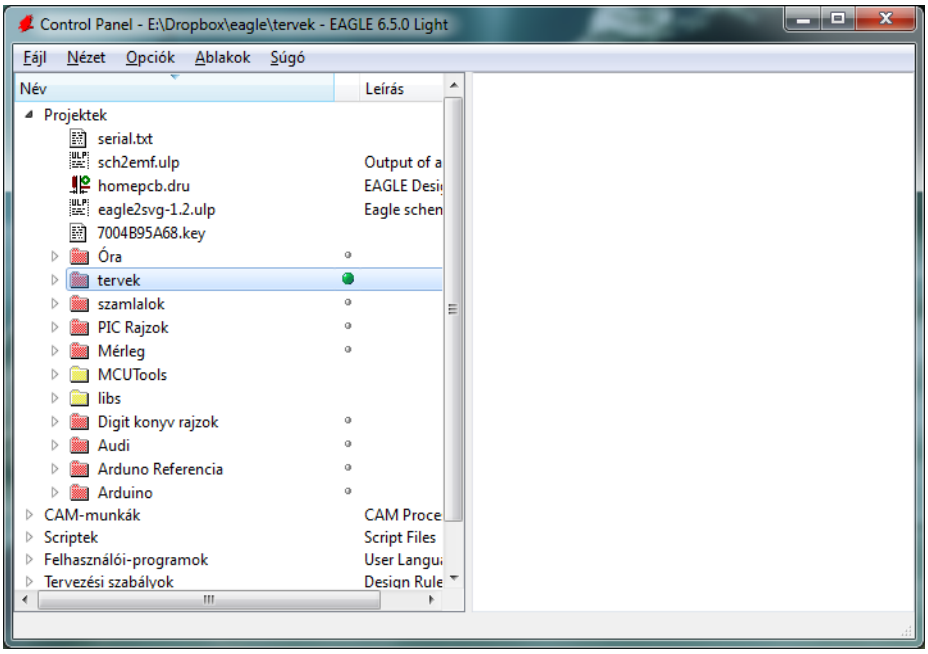
A Light verzió ingyenesen letölthető és használható, azonban a kapcsolási rajz esetén csak egy lap használható, valamint a nyomtatott áramkör mérete maximum 100x80mm lehet és maximum két oldalas lehet. Kezdésképpen ez is tökéletesen megfelel, sőt a legtöbb kis méretű hobbi projekt igényeit ez is bőven kielégíti.

A program multiplatform, elérhető Windows, OS-X és Linux rendszerekre is, sőt a Light verzió a legtöbb Linux disztribúció szoftver áruházában is szerepel, így a telepítéssel nem kell sokat bajlódni. Letölteni a http://www.cadsoftusa.com/download-eagle/ címről lehet.

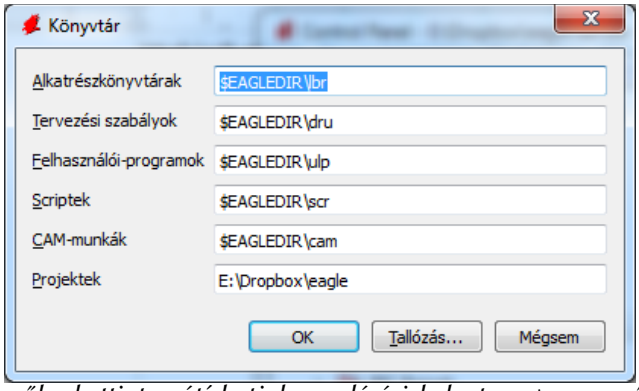
EAGLE Alapismeretek

A program projektekben dolgozik. Egy projekt, ha megvalósul tartalmazni fog legalább egy kapcsolási rajzot és egy nyomtatott áramköri tervet. A nyomtatott áramköri tervező és a kapcsolási rajz szerkesztő használható egymástól függetlenül, azonban nem biztos, hogy érdemes őket így használni, ha csak nem célunk, hogy direkt megnehezítsük a dolgunkat.

Indítás után minden esetben meg fog nyílni egy vezérlő ablak, amiben a projektek áttekinthetőek. Ez a program főablaka. Ha ezt bezárjuk, akkor az egész program bezáródik.

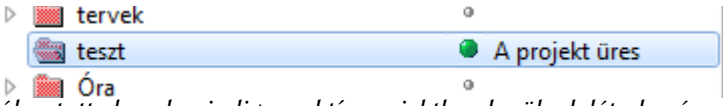


Mielőtt új projektet készítenénk érdemes beállítani a programot. A legfontosabb beállítás, hogy hova kerüljenek a projektek. A mappa meghatározható az opciók → mappák menüpontra kattintás után megnyíló ablakban. Itt több mappa is beállítható. Jelen esetben számunkra a legfontosabb a projektek nevezetű.



A szöveg mezőbe kattintva átírhatjuk az elérési helyet, vagy a mezőbe kattintás után a tallózás gombra kattintva kiválaszthatunk egy mappát. Alapértelmezetten ezt a mappát a projektek menühöz fogja fűzni, mivel a már ott lévő elérési útvonal mögé fogja beírni a program a mappa nevét pontosvesszővel elválasztva. Ezért ha a teljes mappát át szeretnénk helyezni, akkor az ott meglévő útvonalat ki kell onnan törölni.

Új projektet a fájl → új → projekt menük segítségével tudunk Létrehozni. Ez a projektek mappában létrehoz egy új mappát. Ezt nevezzük el tetszőlegesen. A létrehozott projektünk ezután aktiválódik is. Ez onnan mondható meg, hogy a mappa neve mellett megjelenik egy zöld kör.



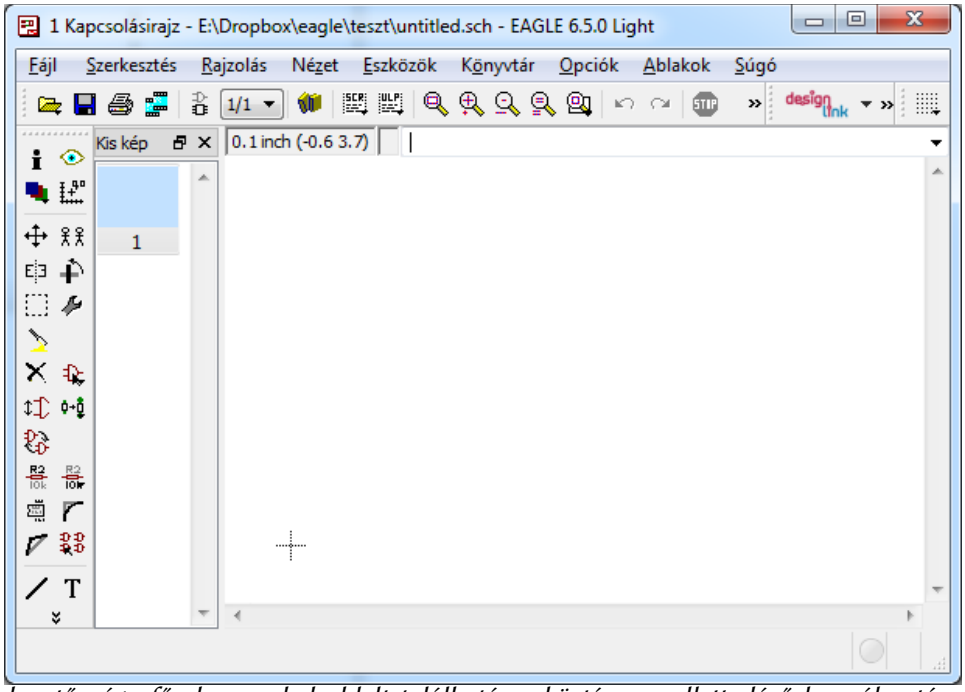
Az új menüből kiválasztott elemek mindig az aktív projektben kerülnek létrehozásra.

Kapcsolási rajz készítése

Egy terv fontos eleme a kapcsolási rajz. Ez lényegében az áramkörünk alapterve. A projekt tervezését ezzel érdemes elkezdeni. Amennyiben ezt jól készítjük el, akkor a tervünk realizálása igencsak egyszerű lesz.

A főablak fájl menüjéből az új kapcsolási rajz határa megnyílik a kapcsolási rajz szerkesztőprogram.

A program további használatát egy konkrét feladat megvalósításán keresztül mutatom be. Egy olyan egyszerű kapcsolást tervezünk meg, amely 8 darab LED-et és előtét ellenállást tartalmaz kivezetve egy csatlakozóra. A kapcsolás gyakorlati haszna az, hogy egy 8 bites port állapotát mutatja meg vizuálisan.



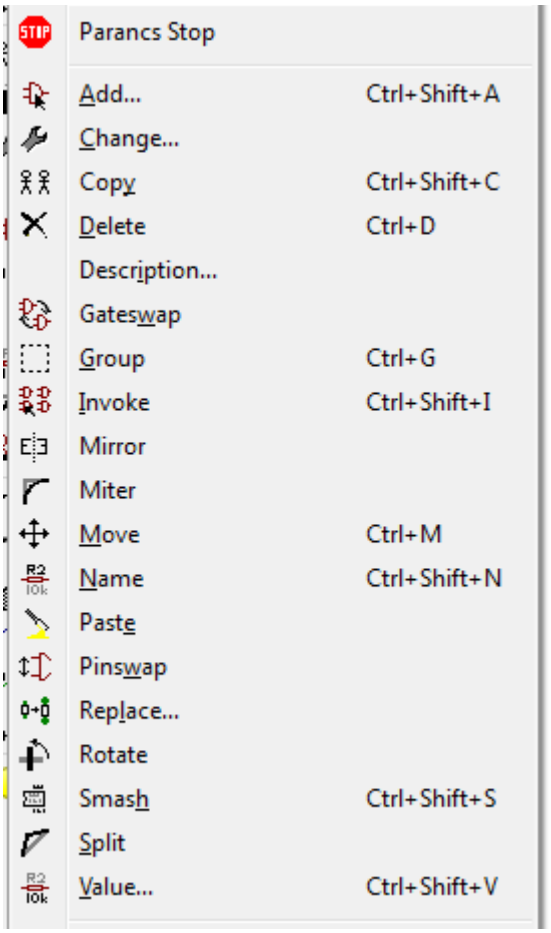
A szerkesztő négy fő eleme a bal oldalt található eszköztár, a mellette lévő lap választó, a jobb oldalon nagy területen elhelyezkedő kapcsolási rajz nézet és az eszköztáron lévő nagyítók.

A bal oldali legszélső nagyító ikon az egész kapcsolási rajzot úgy rendezi, hogy lássuk minden elemét, míg a + és – jelölésű nagyító ikonok nagyításra és kicsinyítésre szolgálnak.

Az egér görgője is használható nagyításra és kicsinyítésre, viszont a nagyítás középpontja nem a dokumentum közepe lesz, hanem az a pont, amelyen aktuálisan az egérmutató áll.

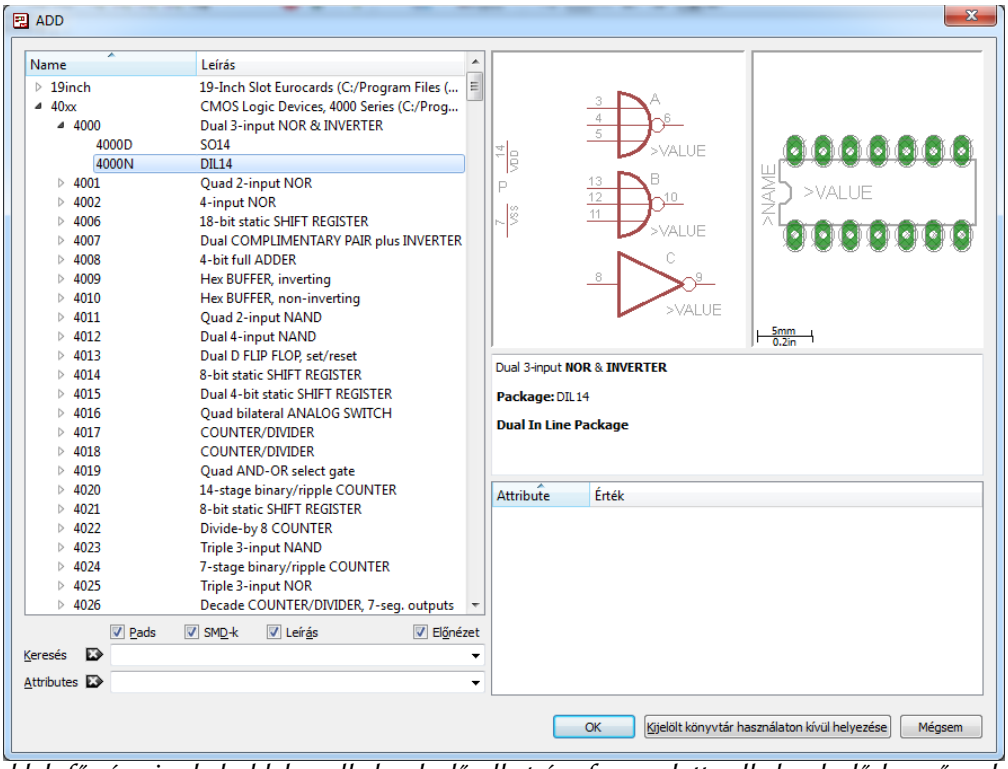
Amennyiben a programot komoly célokra is szeretnénk használni, akkor ajánlott egy nagy méretű FULL HD monitor beszerzése. Pontosabban ha tényleg nagyon komolyan szeretnénk használni, akkor inkább két monitor beszerzése ajánlott. Ekkor egy monitoron tudjuk tervezni/nézni a kapcsolási rajzot, míg a másikon készíthetjük a nyomtatott áramköri tervet.

Az eszköztár elemei a szerkesztés menüben is elérhetőek, mindkét helyen azonos ikonnal vannak jelölve a parancsok. Az alábbi táblázat a legfontosabb parancsokat foglalja össze, amik a szerkesztés menüben is megtalálhatóak.

	***Stop:*** *A jelenleg használt eszközből lép ki. Az ESC gomb is használható helyette.* ***Add:*** *Alkatrész beillesztése a kapcsolási rajzba.* ***Change:*** *A letett alkatrészek különböző tulajdonságainak megváltozatására szolgál.* ***Copy:*** *Másolat készítése egy alkatrészről, vagy a kapcsolási rajz egy részletéről.* ***Delete:*** *Elemek eltávolítása* ***Description:*** *Kapcsolási rajz lapjának leírásának megadására szolgál.* ***Gateswap:*** *Olyan eszközökben, amelyek több kapuból állnak egy letett kaput cserél egy másik hasonlóra.* ***Group:*** *Csoport kijelölése* ***Invoke:*** *Több kapuból álló elemek további fel nem használt kapuinak a rajzba illesztése* ***Mirror:*** *Letett alkatrész tükrözése* ***Move:*** *A kapcsolási rajz elemeinek mozgatása* ***Name:*** *Lerakott áramköri elemek neveinek módosítására szolgáló eszköz* ***Paste:*** *Vágólapról beillesztés* ***Rotate:*** *Elemek forgatása* ***Value:*** *Elemek értékeinek megadására szolgál*

129. Táblázat: Fontosabb EAGLE parancsok

A megvalósítást a rajzban az Add parancs segítségével kell kezdeni. Ez megnyitja az alkatrész kiválasztó ablakot.



Az ablak fő részei a bal oldalon elhelyezkedő alkatrész fa, az alatta elhelyezkedő kereső, valamint a jobb oldalon elhelyezett alkatrész kapcsolási rajz és nyomtatott áramköri nézete.

EAGLE esetén egy alkatrész két elemből áll. Egy kapcsolási rajz szimbólumból és egy nyomtatott áramköri alkatrészből, amely konkrétan az alkatrész fizikai méreteit írja le. Ezért nagyon fontos a megfelelő alkatrész kiválasztása. Az alkatrészek két fő típusba sorolhatóak. SMT (Felület szerelt) és THT (Furaton át szerelt) alkatrészek.

Az SMT alkatrészek otthoni gyártás szempontjából nem megfelelőek, mivel ezen alkatrészek méretei igencsak kicsik, leginkább ipari gyártásban használatosak. Otthoni gyártás során THT alkatrészeket érdemes alkalmazni.

A kereső segítségével egy konkrét alkatrészre tudunk rákeresni a neve alapján. Amennyiben nem tudjuk az eszköz pontos nevét érdemes a keresendő kifejezést csillag karakterek közé tenni. Ez a kereső számára azt jelenti, hogy olyan alkatrészeket keressen, amelyek tartalmazzák az adott kifejezést.

Persze ekkor is előfordulhat, hogy a keresett alkatrészt nem találjuk meg. Ekkor két lehetőségünk adott. Amennyiben egyszerűbb komponensről van szó, mint tranzisztor, dióda, stabilizátor szabványos lábkiosztást használó tokozásban, akkor betehetünk egy másik tranzisztort, amely ugyan olyan lábkiosztással és tokozással rendelkezik, majd módosítjuk az értékét. Így létre is jön az „új" alkatrész.

Integrált áramkörök esetén a módszer nem alkalmazható az eltérő funkciók miatt. Ekkor le kell töltenünk és telepítenünk kell egy olyan alkatrészkönyvtárat ami tartalmazza a keresett alkatrészt. Gyártó specifikus alkatrészkönyvtárakat az Element14 oldalán találunk. Ezen könyvtárak tartalmazzák azokat az alkatrészeket, amelyeket konkrétan ők is forgalmaznak. Sőt a könyvtárakban az alkatrészek mellett fel van tüntetve a katalógus szám is az egyszerű rendelés miatt.

A könyvtárak a http://element14.com/community/community/knode/cadsoft_eagle/eagle_cad_libraries címről tölthetőek le.

Ezeket telepíteni igen egyszerű, mivel csak be kell másolni az EAGLE alkatrész könyvtárakat tartalmazó mappájába. Ez alapértelmezetten az EAGLE telepítési mappájában található lib néven. Természetesen több alkatrész könyvtár mappa is megadható, mint projekt mappa.

A program újraindítása után az alkatrészkönyvtárak megjelennek a főablak alkatrészkönyvtárak szekciójában. Innen már csak használatba kell őket venni. Egy alkatrészkönyvtár akkor van használatban, ha a neve mellett egy zöld kör jelzés látható. Az összes könyvtár egyszerre használatba vehető, ha az alkatrészkönyvtárak menüponton jobb kattintva rákattintunk a megjelenő menüben az összes használatbavétele menüpontra.

A sok alkatrész betöltése időt vesz igénybe. Ezért ajánlatos lenne projektenként kiszelektálni azon könyvtárakat, amelyeket használunk vagy sem. A használt könyvtárak listája projektenként mentődik.

Az alábbi táblázatban az általam leggyakrabban használt alkatrészkönyvtárak nevét és tartalmukat foglaltam össze röviden.

Könyvtár neve	***Leírása***
40xx	*Logikai áramkörök a 40xx családból*
45xx	*Logikai áramkörök a 45xx családból*
74xx-eu	*Logikai áramkörök a 74xx családból Európai kapcsolási rajz jelekkel*
battery	*Elemek és elem csatlakozók*
con-lsta	*Tüskesorok anya végződéssel*
con-lstb	*Tüskesorok apa végződéssel*
con-wago-500	*Wago nyomtatott áramkörbe ültethető sorkapcsok*
diode	*Diódák*
led	*LED diódák*
pinhead	*Tüske kivezetések*
pot	*Változtatható ellenállások, potenciométerek*
rcl	*Ellenállások, kondenzátorok, induktivitások*
suply1	*Feszültségek jelölésére használható jelölők*
transistor	*Tranzisztorok*
v-reg	*Lineáris feszültség stabilizátorok*

130. Táblázat: Gyakran használt könyvtárak és tartalmuk

A példa projektben az alábbi alkatrészeket kell a kapcsolásba illesztenünk:

- *1db LED. Ezt a led könyvtáron belüli LED könyvtárban találjuk meg. A neve LED5MM. Ez egy 5mm átmérőjű szabványos LED diódát illeszt a kapcsolásunkba.*
- *1db ellenállás, amit az rcl könyvtáron belül lévő R-EU_ könyvtárban találunk meg. Ebben a könyvtárban ellenállások vannak Európai rajzjellel. Az alkatrész pontos neve R-EU_0207/7. Ez az ellenállásnak 7mm lábtávolságot határoz meg, ami kényelmesen kezelhető.*
- *1db 1 soros 9 tűs anya tüskesor. Ezt a con-lsta könyvtárban találjuk meg FE-09-1 néven.*

Az ellenállásokból és a LED-ekből azért tettünk be csak 1-1 darabot, hogy tudjuk gyakorolni a copy eszköz működését. A Copy eszköz egyszerre csak egy alkatrészt másol le. Azonban ha az alkatrészeket csoportba foglaljuk a group eszközzel, majd utána kiválasztjuk a copy eszközt és a csoporton jobb kattintunk, akkor a menüben megjelenik egy olyan menüpont, aminek a neve Copy: Csoport. Erre kattintva az egész csoport másolódik. Ezen módon a másolt rész a vágólapra is bekerül. A vágólap tartalmát a Paste eszközzel tudjuk beilleszteni.

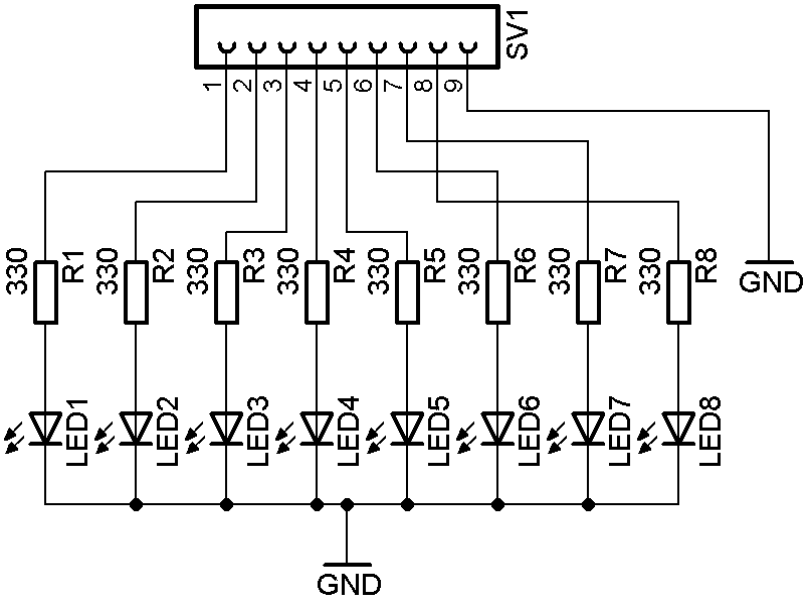
Az alkatrészek elhelyezése után következhet azok elhelyezése. Az elhelyezésnél figyelembe kell venni, hogy az alkatrészek nevei és értékei jól láthatóak legyenek. Az alkatrészek nevei és értékei a Name és Value eszközökkel adhatóak meg.

Összekötésre az eszköztáron található Net parancs szolgál. Ezzel összekötve az eszközeinket az elemek között nyomtatott áramkör tervező módban is logikai kapcsolatokat alakítunk ki.

337. Ábra: Net parancs

Mivel a vezetékek egymáson keresztül menve igencsak kuszák tudnak lenni lehetőségünk van a vezetékeket feliratozni. Ebben az esetben ha két vezetékdarabot ugyan azzal a névvel látunk el, akkor a program tudni fogja, hogy azokat össze kell kötni egymással majd nyomtatott áramkör tervezéskor.

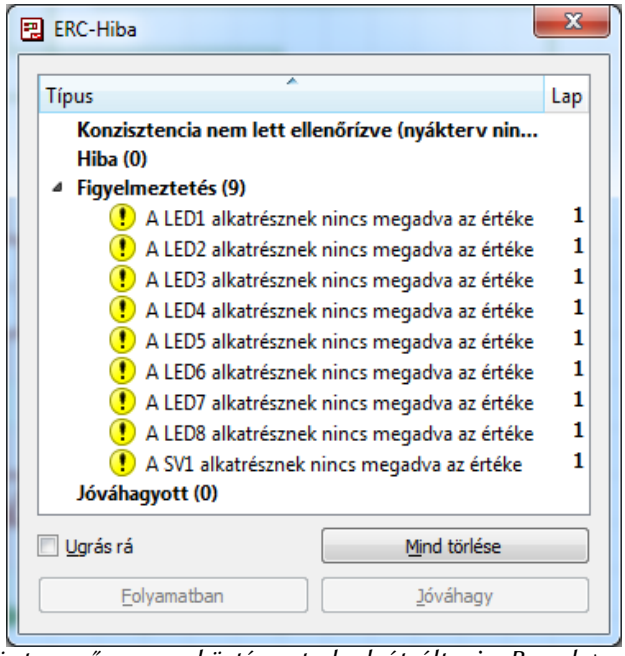
A vezetékek feliratozásához szükségünk lesz legalább két darab vezetékre. Ezután a vezetékdarab mindkét végére egy címkét kell helyeznünk a Label parancs segítségével. Ezután a két vezetékdarabot azonos néven kell elnevezni. Tápfeszültségek hasonló módon jelölésére és összekötésére használhatóak a suply1 könyvtár elemei.



A kapcsolás elkészítése után egyből ne rohanunk a nyomtatott áramkör tervezéssel. Először elektromosan a kapcsolásunkat. Erre az Errors ablak használható, amely az eszköztárról érhető el. Sárga alpon fekete felkiáltójel ikonnal.

345. Ábra: Errors

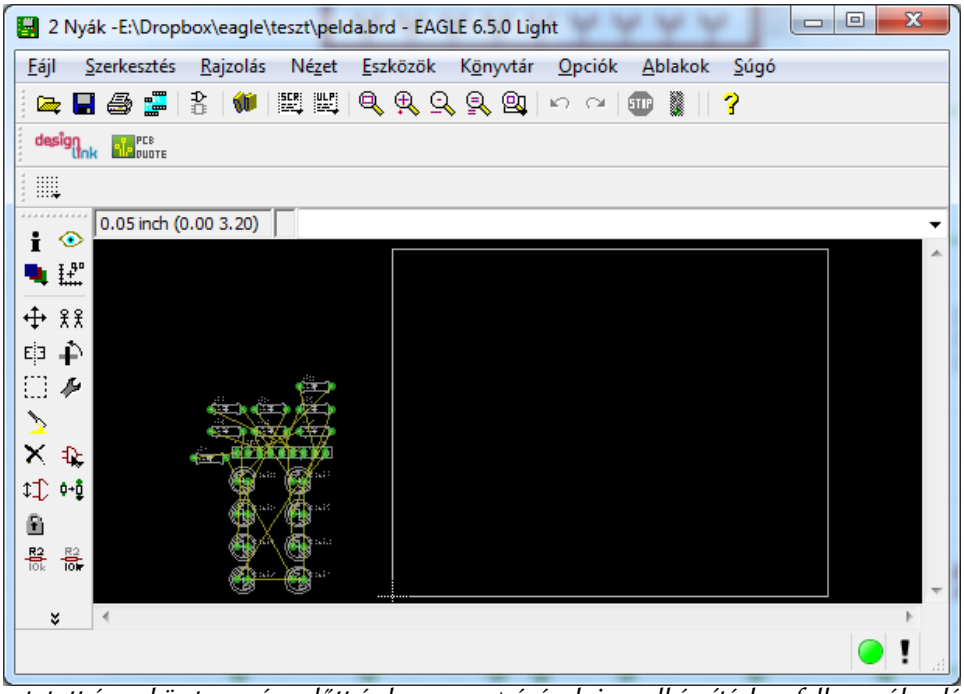
Erre kattintva megkapjuk a kapcsolásban szereplő hibákat és figyelmeztetéseket. A figyelmeztetések figyelmen kívül hagyhatóak, bár érdemes azokat is elhárítani. A hibák elhárítása mindenképpen szükséges a nyomtatott áramkör tervezés megkezdése előtt.



A nyomtatott áramkör tervezőre az eszköztáron tudunk átváltani a Board gombra kattintva.

Ennek hatására a program megkérdezi, hogy a kapcsolási rajz alapján létrehozza e a nyomtatott áramköri tervet. Itt válaszoljunk igennel, mivel ez a célunk.

Nyomtatott áramkör tervezése



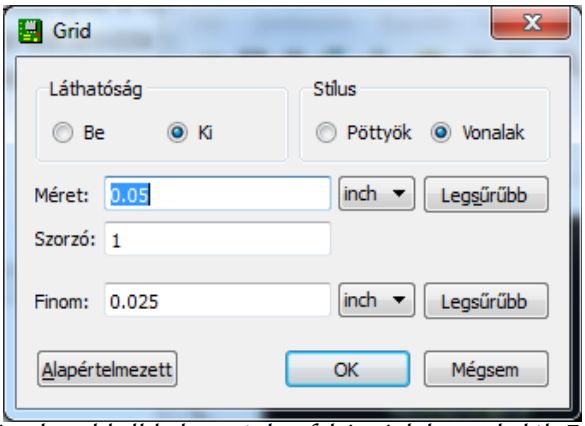
A nyomtatott áramkör tervezése előtt érdemes megvásárolni az elkészítéshez felhasználandó alkatrészeket. Erre azért van szükség, hogy a méreteiket pontosítani tudjuk. Ez főként kondenzátorok esetén fontos, mivel millió egy méretben kaphatóak.

Ha a kapcsolási rajzban azon alkatrészek szerepelnek, amikkel ténylegesen rendelkezünk, akkor kezdhetjük a tervezést. EAGLE esetén egy rácson tudjuk mozgatni az alkatrészeinket. A rács finomsága határozza meg a minimális mozgatási távolságot. Alapértelmezetten ez igen nagyra van állítva, viszont kezdő tervekhez tökéletesen megfelel.

A rács finomsága az estköztáron elhelyezett rács ikonra kattintva módosítható.

348. Ábra: Rács beállítása

A rács finomsága igen sok méretben megadható. Nyomtatott áramkör tervezéskor egy speciális mértékegység a mil. 1 mil az az inch ezred része. Ez 0,0254 mm-t jelent. Személy szerint én a rács finomságát 10 mil méretre szoktam állítani, a finom mozgást pedig 5 mil-re. Első terveink esetén nem érdemes módosítani a rács finomságát.



Ezt követően az alkatrészeket el kell helyeznünk a fehér téglalapon belül. Ez a Light verzióban a tervezhető maximális nyomtatott áramkör méretet jelöli. Az alkatrészek elhelyezése tervezés és vezetékezés közben

folyamatosan változhat.

Az alkatrészek megfelelő elhelyezése és összekötése manuálisan igen sok gyakorlást igényel, így ne aggódjunk, hogy ha elsőre nem lesz olyan a tervünk, mint amilyent szeretnénk.

Tervezés esetén két réteget használhatunk. Egy alsó réteget, amely kék színnel van jelölve és egy felső réteget, ami pedig piros színnel. Otthoni tervezéseink során egy oldalas terveket igyekezzünk készíteni, mivel ezen panelokat sokkal könnyebb lesz utána legyártani. A használandó réteg ebben az esetben az alsó lesz. Méghozzá azért, mert így az alkatrészek a panel azon oldalán fognak elhelyezkedni, ahol nincs vezetősáv, a sávok pedig a panel alján kapnak majd helyet.

A program rendelkezik egy autoruter eszközzel is, amely képes összekötögetni automatikusan a nyomtatott áramkör részeit. Az összekötéshez a kapcsolási rajzból kapott információkat használja fel. Ezeket a nyomtatott áramkörtervező sárga légvezetékekkel jelöli.

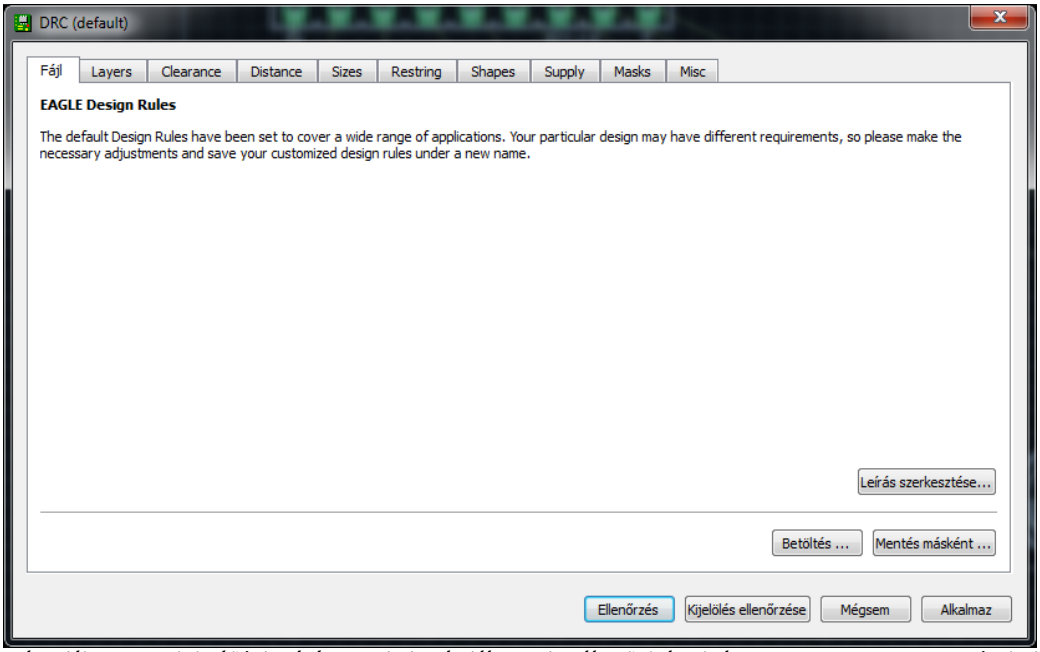
Ha az autorouter funkcióra szeretnénk hagyatkozni, akkor érdemes észben tartani, hogy ez megfelelően csak két rétegű nyomtatott áramkörök tervezése esetén működik rendesen, valamint nem éppen a legoptimálisabb terveket generálja. Az alkatrészek elhelyezésekor itt is érdemes figyelembe venni a légvezetékek elhelyezkedését az alkatrészeknél.

Az autorouter tervezési szabályokkal konfigurálható. A tervezési szabályok geometriai szabályok gyűjteménye. Segítségükkel rengeteg minden konfigurálható. Amennyiben nem is használjuk az autorouter funkciót, akkor is az ebben megadott forrasztási pont méretek fogna alkalmazódni. A beépített tervezési szabályok nem éppen otthoni tervezéshez lettek kialakítva. Ezért a CD mellékleten elhelyeztem az általam használt szabály gyűjteményt is, ami kifejezetten otthoni gyártáshoz lett optimalizálva.

Ha a lapunkat gyártóval szeretnénk gyártatni, akkor a gyártók a legtöbb esetben biztosítják a szükséges tervezési szabályok fájlt. Amennyiben nem, akkor ezt manuálisan specifikáció alapján nekünk kell beállítanunk.

A tervezési szabályok a DRC feliratú ikonra kattintva érhetőek el.

350. Ábra: Tervezési szabályok



Az aktuális tervezési előírások betartásának állapotát ellenőrizhetjük az Errors parancs segítségével.

Egy alkatrész áthelyezése után a légvezetékek útvonala újra generálható a legrövidebb útvonalra. Ehhez az eszköztáron lévő Rastnet parancsot kell kiválasztanunk.

352. Ábra: Rastnet parancs

Amennyiben úgy gondoljuk a tervünk kész, illetve az elhelyezésük rendben van, akkor megkezdhetjük a vezető sávok kihúzását. Ezt a Route parancs segítségével tehetjük meg. Ekkor a vezetősáv méretét nekünk kell megadni. A vezetősáv méretét elsősorban az áramerősség határozza meg, másodsorban pedig a gyártási technológia. A vezetősávok méreténél található lenyíló menüben a vezeték méret mértékegysége a rács meghatározásánál alkalmazott mértékegységben értendő.

Mértékegység a rács méretének módosítása nélkül is változtatható. Ha átállítjuk a mértékegységet, akkor a rácsméretet a program átszámolja az új méretre.

A példában bemutatott áramkör tervezésekor 1mm széles vezetősávokat alkalmaztam. Ennek a fele is elég lenne, azonban első pár tervünk gyártásakor a munkánkat könnyítjük meg azzal, hogy ha vastagabbra hagyjuk a szélességet.

A tervezés bonyolultsága, hogy a két különböző vezeték nem keresztezheti egymást. Ennek az elkerülése igen sok gyakorlást igényel. Előfordulhat, hogy nem lehet ezt megoldani. Ekkor átkötés beiktatására van szükség.

Átkötést úgy tudunk létrehozni, hogy a két összekötendő vezetősáv végére a Via eszközzel teszünk egy átkötési pontot. Ezután ezeket összekötjük a felső rétegen vezetősávval. Ez nem jelenti azt, hogy a panelnak két oldalasnak kell lennie. Egy-két átkötés, sőt akár 20 átkötés is megoldható vezetékdarabok segítségével, ha ha a panelunk elég nagy. Viszont az biztos, hogy minél több átkötésünk van, annál lassabban szerelhető lesz a kész panel.

Amennyiben az összes alkatrészt bekötöttük, akkor nincs más dolgunk, mint a tervből egy nyomtatható képet generálni, amit aztán valamilyen úton, módon átviszünk a panelra.

340. Ábra: A kész nyomtatott áramköri terv

Először ehhez érdemes a Wire eszköz segítségével egy keretet rajzolni a tervünk köré a befoglaló méretek meghatározása szempontjából. Ezt követően ki kell kapcsolnunk bizonyos rétegeket, amik nem kellenek nyomtatás során.

A rétegek az eszköztár Layers parancsa segítségével módosíthatóak

354. Ábra: Rétegek ikon

A terv nyomtatásakor csak a Bottom, vias, és pads rétegre van szükségünk. Amennyiben nem csak egy tervet szeretnénk egyszerre nyomtatni, akkor érdemes a tervet exportálni képként, majd egy grafikai szoftver vagy szövegszerkesztő segítségével nyomtatni, úgy hogy többet egymás mellé teszünk.

Exportálni a tervet a Fájl → Export → Image menüpont alatt tudjuk. Formátumnak érdemes PNG-t választani, mivel ez veszteségmentes, továbbá elég ha fekete-fehér képet exportálunk. A felbontás legalább 300dpi legyen.

Grafikai vagy szövegszerkesztőből nyomtatás esetén érdemes megjegyezni, hogy ezen programok nem minden esetben pixel pontos képet nyomtatnak. Ez összeszerelés esetén jelentkezhet úgy, hogy nem minden alkatrész illeszkedik.

Egyes esetekben a nyomtatók sem teljesen pontos képet nyomtatnak, így a használt szoftver és hardver kombinációt ki kell kísérletezni.

Nyomtatott áramkör készítése

Nyomtatott áramkört két féleképpen készíthetünk. Vagy legyártatjuk, vagy legyártjuk magunknak. A gyártatás előnye, hogy nem kell a gyártási folyamattal pepecselnünk és a kész panel ipari előírásoknak is megfelelő professzionális kivitelben készül el. A folyamat hátránya, hogy gyártótól függően egyetlen egy panel gyártása igen drága tud lenni. Így ez a módszer prototípus gyártás esetén, ha csak 1-2 panel kell, akkor nem éri meg. Viszont ha több panelt kell gyártani, akkor érdemes elgondolkodni a gyártatáson.

A házi gyártáshoz szükségünk lesz jó néhány eszközre és szerszámra, valamint vegyszerek is kelleni fognak. Házi gyártás esetén két technika adódik. A vasalásos és a fotózásos technológia. A fotózásos technológia lényege, hogy a panelt UV érzékeny lakkal vonjuk be, majd a maratandó sávok helyét kimaszkoljuk és UV fénnyel megvilágítjuk a panelt. A maszkolás helyén a lakk sav álló marad, ahol pedig az UV fény érte a panelt azokat a részeket gond nélkül fogja feloldani a maratő folyadék.

Ezen módszer hátránya, hogy pepecselősebb, illetve a lakk minőségétől igen sok minden függ. Ezért itt a Vasalásos technológia lesz ismertetve. Ennek lényege az, hogy a lézer nyomtatók által használt lézer festék megfelelő papírra nyomtatás után átvasalható a panelra, ami aztán maratható. Ezen gyártáshoz szükséges eszközök:

- *Háztartási sósav*
- *Hidrogén-peroxid*
- *Lézer nyomtató*
- *Fényes műnyomó papír*
- *Lapos talpú vasaló*
- *Nyomtatott áramkör panel*
- *Kémiai ónozó*
- *Alkoholos festéktoll*
- *Finom szemcsés csiszolópapír*
- *Hígító*
- *Fűrész és/vagy lemezolló*
- *0,8mm és 1mm átmérőjű fúrók*
- *Oszlopos fúró*
- *Forrasztó és forrasztóón.*

A folyamat a nyomtatott áramkör megtervezésével kezdődik. A kész tervet ki kell nyomtatni lézer nyomtatóval a fényes műnyomó papírra. A kész nyomtatott részhez nem szabad hozzányúlni, mivel a kezünkön lévő zsírok megakadályozhatják a tökéletes átvitelt.

A marás előtt a panelt elő kell készíteni. Méretre kell vágni vagy egy lemez olló vagy fűrész segítségével. Minél kisebb panelt gyártunk, annál nagyobb a valószínűsége, hogy jó lesz a vasalásunk. Ez abból adódik, hogy ebben az esetben egyenletesebben át tud melegedni a papír és az alatta lévő panel.

Az előkészítés második lépése a panel zsírtalanítása és felső oxid rétegének eltávolítása. Erre a célra a finom szemcsés csiszolópapír használható. Ezzel alaposan és egyenletesen át kell csiszolni a panelt. A csiszolt panel réz fóliás részéhez szintén nem szabad hozzáérni.

A tisztítás után jöhet a vasalás. A papír nyomtatott felét össze kell borítani a panellal, úgy, hogy a festékes oldal és a réz fólia érintkezzen egymással. A vasalót 200 fok környékére kell melegíteni. A legtöbb vasaló rendelkezik gyapjú vasalási lehetőséggel, ami megfelel a célra. Bár vasalónként változhat az ideális beállítás, amit ki kell tapasztalni.

A vasalást nem túl erősen kell kezdeni, mivel ekkor még a papír elcsúszhat. Finom mozdulatokkal el kell érni, hogy a tinta felső rétege hozzátapadjon a rézhez egyenletesen. Ezután erősebb mozdulatokkal át kell melegíteni az egész panelt és papírt. Túl erős mozdulatok és meleg vasaló esetén a nyomat ugyan átragad, viszont elmaszatolódik. Túl hideg és túl finom mozdulatok esetén viszont nem tapad át a festék. A vasalási folyamat akkor jó, ha a papír hátoldalán igencsak átlátszik a vasalt nyomat.

341. Ábra: Vasalás után

A levasalás után a papírt el kell távolítani réz fólia felületéről. Ehhez jó 10 percre víz alá kell tenni a papírt, ami ezután gond nélkül eltávolítható lesz. Amennyiben a lehúzás során sérülnek a vezetősávok, akkor a sérülések mértékétől függően vagy újravasaljuk a panelt, vagy kijavítjuk az apróbb hibákat alkoholos festéktollal.

342. Ábra: Rosszul és jól sikerült vasalás

Ezután következhet a maratás. Itt két opciónk van vagy Vas(III)-klorid oldatot használunk, vagy Hidrogén-peroxid és sósav keverékét.

A Vas(III)-klorid oldat egy barna színű folyadék, ami oldja a rezet. A marási folyamat igen lassú, melegítéssel gyorsítható. Maró hatású, ezért szabad kézzel belenyúlni nem szabad. Ruhára kerülve befesti azt, a folt mosással sem lesz eltávolítható, így nagy körültekintéssel kell alkalmazni.

A Hidrogén-peroxid és sósav maratő folyadék előnye, hogy a maratási reakció sebessége jól szabályozható. A reakció sebessége annál gyorsabb, minél több hidrogén-peroxid kerül a vegyületbe. A gyors reakció a pontosságot rontja, mivel ha sok hidrogén-peroxid kerül az oldatba, akkor az maratás közben igencsak melegedni fog, ez pedig deformálja a vezető sávokat. A maratási folyamat víz adagolásával lassítható.

A keverési arányok a használt oldatok erősségétől függenek, így arányokat írni nem igen lehet. Egy bevált keverési eljárás, hogy annyi sósavat öntünk egy tálba, hogy a sav ellepje a panelt. Ehhez pedig szépen, lassan, mondhatni cseppenként addig adagoljuk a hidrogén-peroxidot, amíg a marási folyamat meg nem indul.

343. Ábra: Panel marás kezdetén és majdnem készen.

A Hidrogén-peroxidos maratást csak és kizárólag csak jól szellőztetett helyen végezzük el, mivel a folyamat során klór szabadul fel, amely belélegzése nem egészséges, túlzott mértékben halált is okozhat.

Bármely maratási folyamatot is válasszuk kellő körültekintéssel járjunk el. Ügyeljünk a megfelelő szellőztetésre és a megfelelő egyéni védőfelszerelések használatára, mint védőszemüveg és gumikesztyű. Továbbá mivel a maratás során használt oldatok a környezetre károsak ezért még véletlenül se engedjük őket a lefolyóba. Ártalmatlanításukról megfelelően gondoskodni kell, ezek veszélyes hulladéknak minősülnek.

A veszélyes hulladékkal kapcsolatos tevékenységek végzésének feltételeit a 98/2001. (VI. 15.) Korm. rendelet szabályzza. A rendelet szövege megtekinthető a http://net.jogtar.hu/jr/gen/hjegy_doc.cgi?docid=A0 100 098.KOR címen.

Veszélyes hulladékok elszállítására és megfelelő ártalmatlanításukra cégek szoktak vállalkozni. Ezért mielőtt maratási folyamatba kezdünk érdemes tájékozódni a környékbeli cégeknél, hogy mennyiért vállalják a hulladékok szakszerű szállítását és kezelését. Ennek tükrében állapítható meg, hogy megéri e egy panelt gyártatni.

A maratás után a panelt le kell mosni vízzel. A nyomtató festék hígító segítségével távolítható el. Ezután következhet a panel fúrása. Ehhez 0,8mm átmérőjű fúrót érdemes alkalmazni, egyes alkatrészek lábai azonban vastagabbak, ezekhez vagy 1mm vagy 1,2mm átmérő szükséges. A fúrást oszlopos fúró segítségével érdemes elvégezni.

A fúrás után jöhet a forrasztás. Előtte azonban érdemes elgondolkodni a panel védelméről. A réz a levegő nedvességének hatására oxidálódik, ami rontja a vezetőképességét. Ezért a forrasztás előtt érdemes kémiai ónozó segítségével egy vékony ón réteget a panelra juttatni, ami a rezet megvédi az oxidációtól.

Amennyiben forrasztás előtt nem ónoztuk le a panelt, akkor sincs baj, mivel lakk segítségével is megvédhető a kész panel, azonban ha később szerelni kell a panelt, akkor ez igazán megnehezíti az alkatrészek cseréjét.

A jó forrasztáshoz szükséges egy jó forrasztó, jó minőségű forrasztóón és némi gyakorlat. A forrasztás tudománya igen jól elsajátítható a Forrasztani Egyszerű képregényből, amely a könyvhöz tartozó mellékletben is megtalálható, önállóan pedig a https://mightyohm.com/blog/2012/06/forrasztani-egyszeru-soldering-is-easy-hungarian-translation/ címről szerezhető be.

27. Mellékletek

Arduino környezet által támogatott processzorok listája

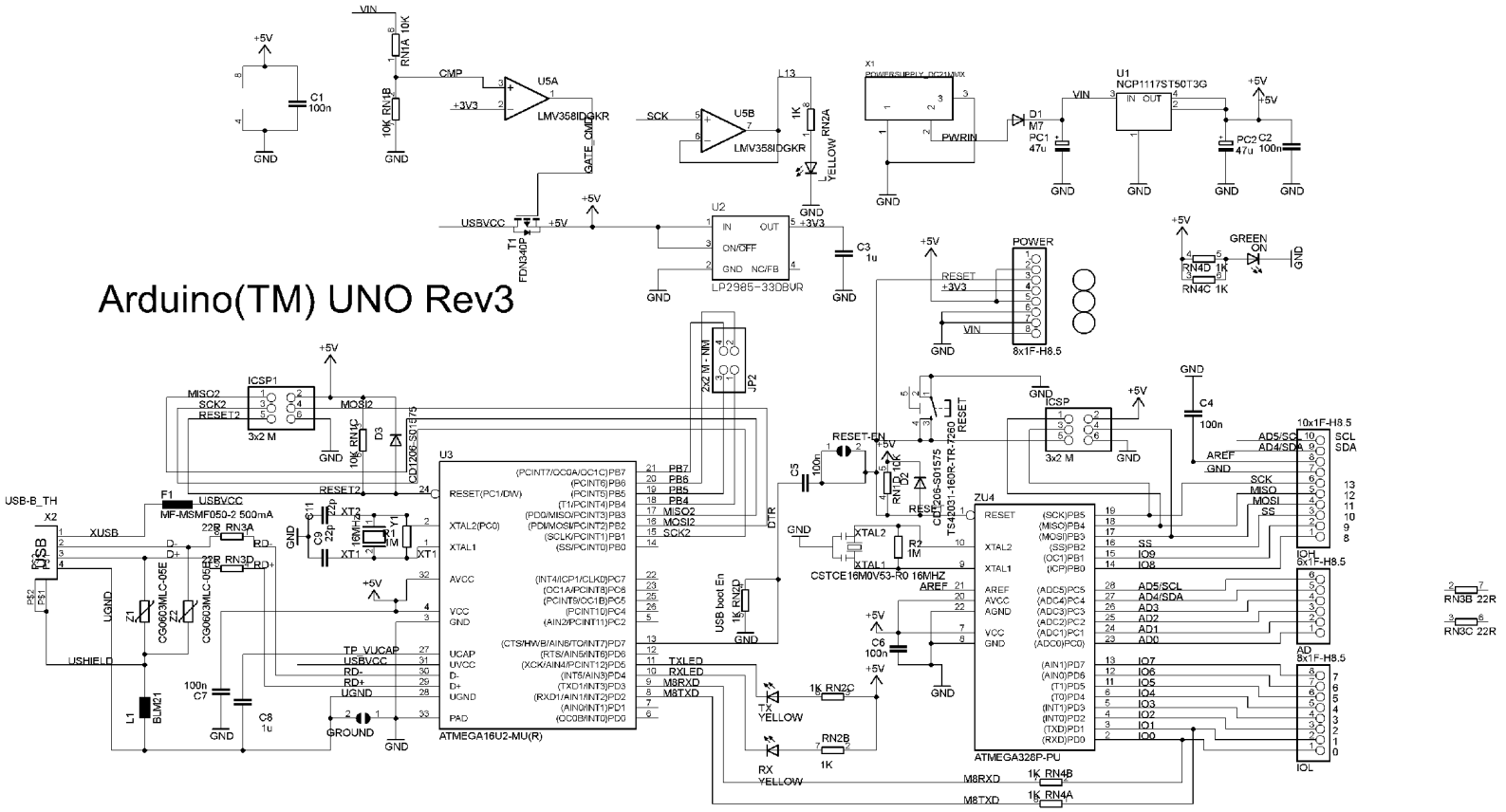
Az alábbi táblázat az Arduino környezet által hivatalosan és nem hivatalosan támogatott mikrovezérlők főbb tulajdonságait foglalja össze. Az X a megfelelő mezőkben azt jelenti, hogy az eszköz hardversen rendelkezik ilyen funkcióval, az S azt jelzi, hogy a funkció szoftveresen támogatott.

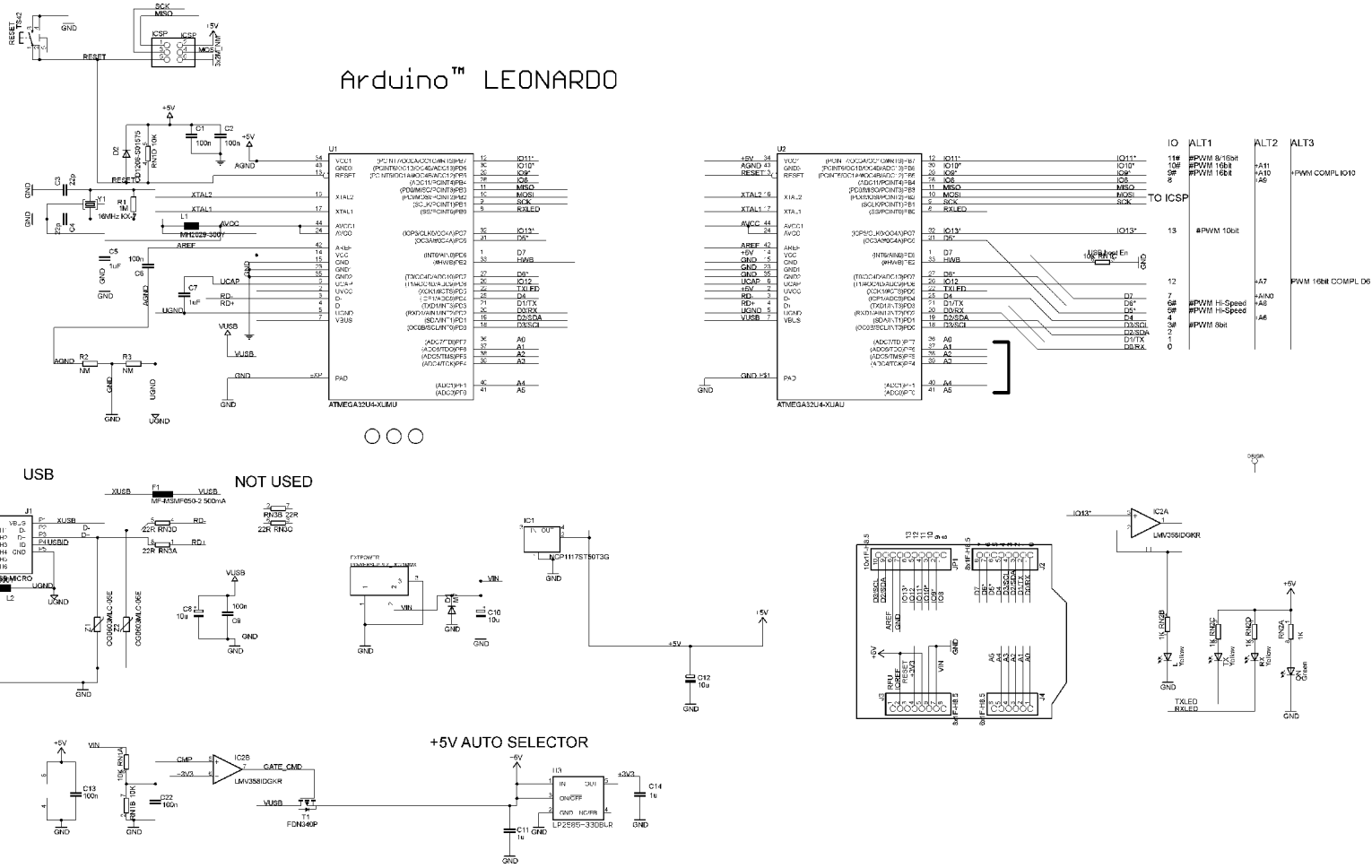
Neve	ROM (KB)	EEPROM (KB)	RAM (KB)	Digitális ki/be	Analóg be	PWM	I2c	SPI	USART	USB	JTAG	DIP tokozás
ATTiny44	4	0,25	0,25	11	8	4	S	x				van
ATTiny45	4	0,25	0,25	5	3	2	S	x				van
ATTiny84	8	0,5	0,5	11	8	4	S	x				van
ATTiny85	8	0,5	0,5	5	3	2	S	x				van
ATMega168	16	0,5	2	20	6	6	x	x	x			van
ATMega328	32	1	2	20	6	6	x	x	x			van
ATMega32u4	32	1	2,5	20	12	7	x	x	x	x		nincs
ATMega644	64	2	4	32	8	6	x	x	2		x	van
ATMega1284	128	4	16	32	8	6	x	x	2		x	van
ATMega2560	256	4	8	54	16	15	x	x	4			nincs

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/

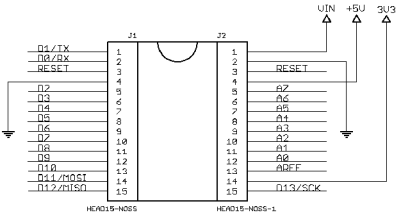
Az egyes modellek tervei időközönként frissülnek. Az egyes változatok naprakész tervei a http://arduino.cc/en/Main/Products oldalon a modell kiválasztása után tekinthetőek meg.

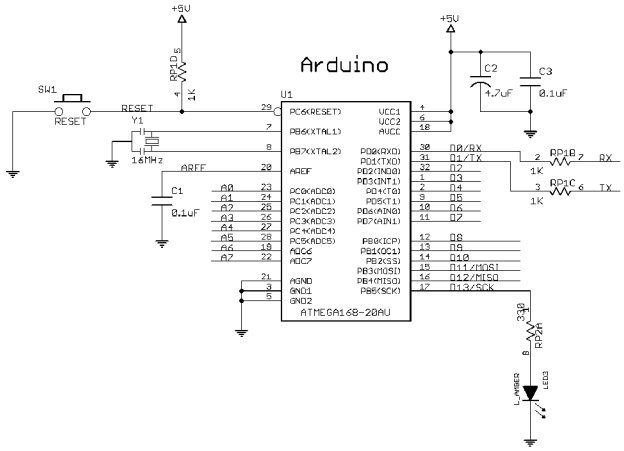

Arduino(TM) UNO Rev3
VIN
RNTA 10K
CMP
U5A
LMV358IDGKR
+3V3
GATE_CMP
SCK
U5B
LMV358IDGKR
L13
1K
RN2A
YELLOW
X1
PWRIN
D1
M7
U1
NCP1117ST50T3G
IN OUT
PC1
47u
PC2
47u
C2
100n
C1
100n
USBVCC
T1
FDN340P
U2
IN OUT
ON/OFF
GND NC/FB
LP2985-33DBVR
C3
1u
+3V3
POWER
RESET
8x1F-H8.5
GREEN
ON
RN4D 1K
RN4C 1K
ICSP1
3x2 M
MISO2
SCK2
RESET2
MOSI2
RN1C
10K
D3
CD1206-S01575
2x2 M - NM
JP2
U3
ATMEGA16U2-MU(R)
RESET(PC1/DW)
XTAL2(PC0)
XTAL1
AVCC
VCC
GND
UCAP
UVCC
D-
D+
UGND
PAD
TP_VUCAP
USBVCC
RD-
RD+
UGND
GROUND
USB-B_TH
X2
USB
XUSB
F1
MF-MSMF050-2 500mA
22R RN3A
22R RN3D
RD-
RD+
Z1
Z2
CG0603MLC-05E
USHIELD
UGND
L1
BLM21
C7
100n
C8
1u
XT2
XT1
16MHz
C11
22p
C12
22p
SY1
1M
USB boot En
18K RN2D
TXLED
RXLED
M8RXD
M8TXD
TX
YELLOW
RX
YELLOW
1K RN2C
RN2B
1K
DTR
RESET-EN
C5
100n
RN1D 10K
D2
CD1206-S01575
TS42031-16DR-TR-7260
RESET
XTAL2
R2
1M
XTAL1
CSTCE16M0V53-R0 16MHZ
ZU4
ATMEGA328P-PU
AREF
AVCC
AGND
C6
100n
M8RXD
M8TXD
1K RN4B
1K RN4A
C4
100n
10x1F-H8.5
AD5/SCL
AD4/SDA
SCK
MISO
MOSI
SS
IOH
8x1F-H8.5
AD
8x1F-H8.5
IOL
RN3B 22R
RN3C 22R

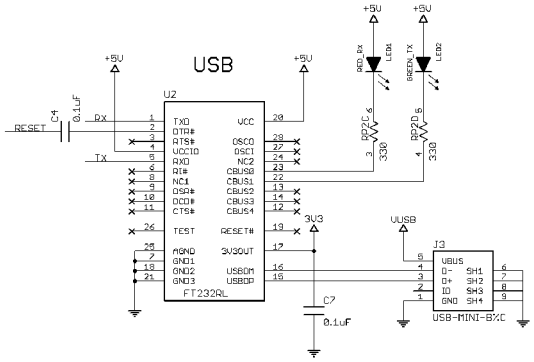

Arduino™ LEONARDO
TO ICSP
ATMEGA32U4-XUMU
ATMEGA32U4-XUAU
IO
ALT1
ALT2
ALT3

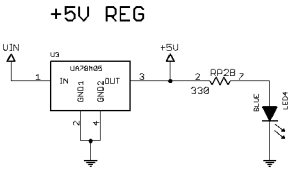
USB
NOT USED
+5V AUTO SELECTOR

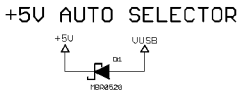
Arduino Nano

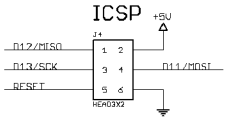


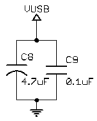


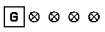








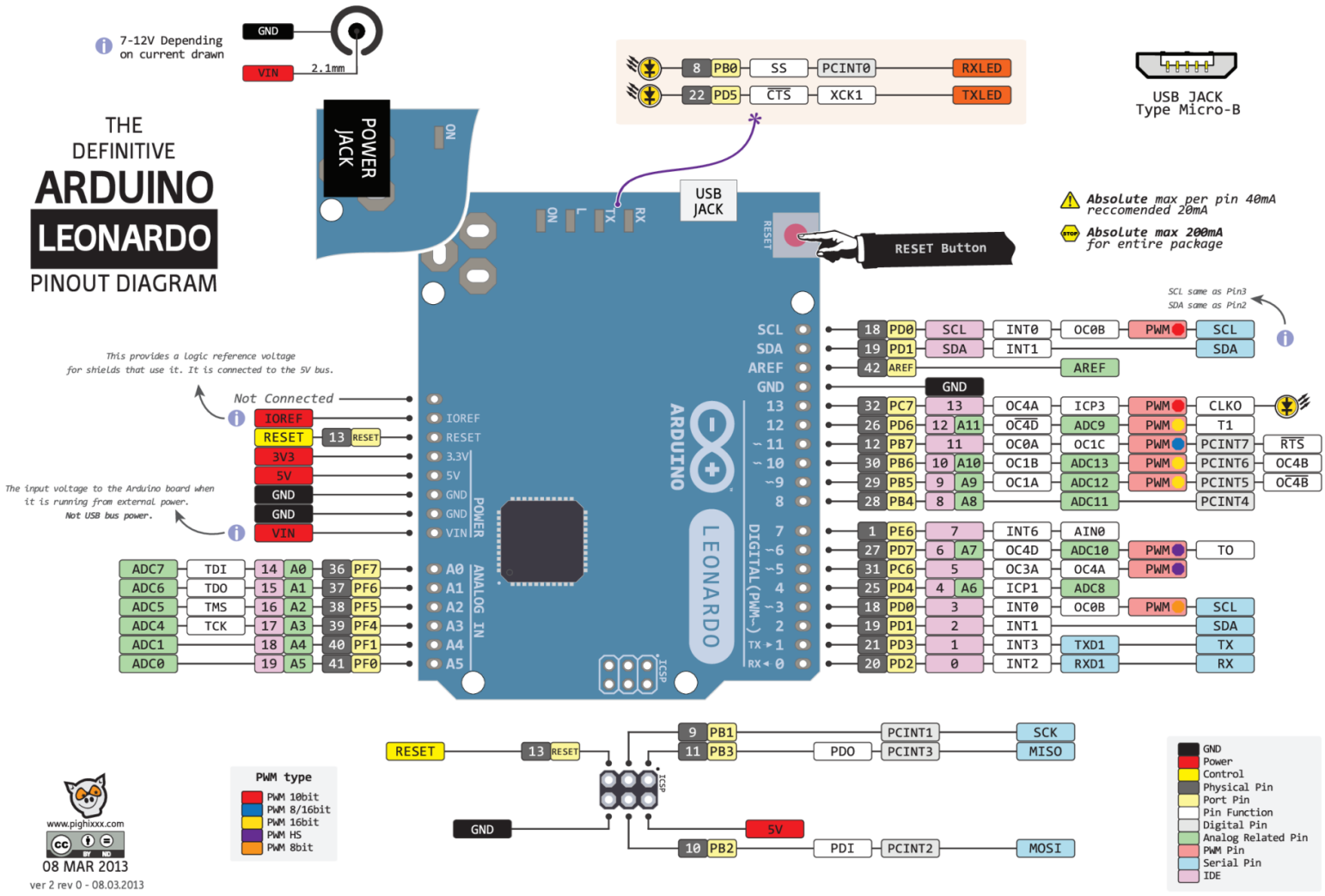




Lábkiosztások

Az alábbi lábkiosztás diagramok a http://www.pighixxx.com/ címről származnak.

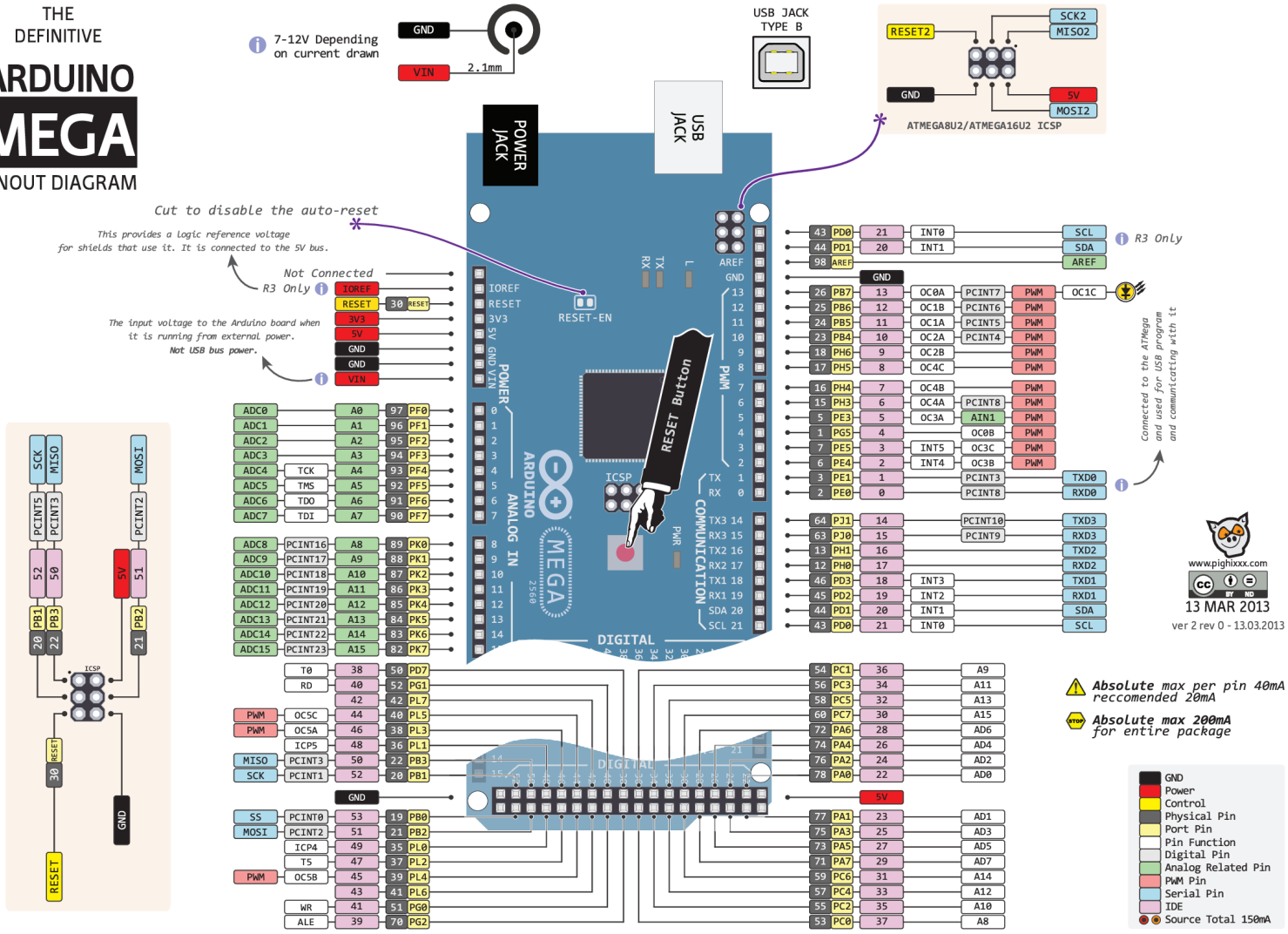
THE DEFINITIVE ARDUINO LEONARDO PINOUT DIAGRAM



08 MAR 2013

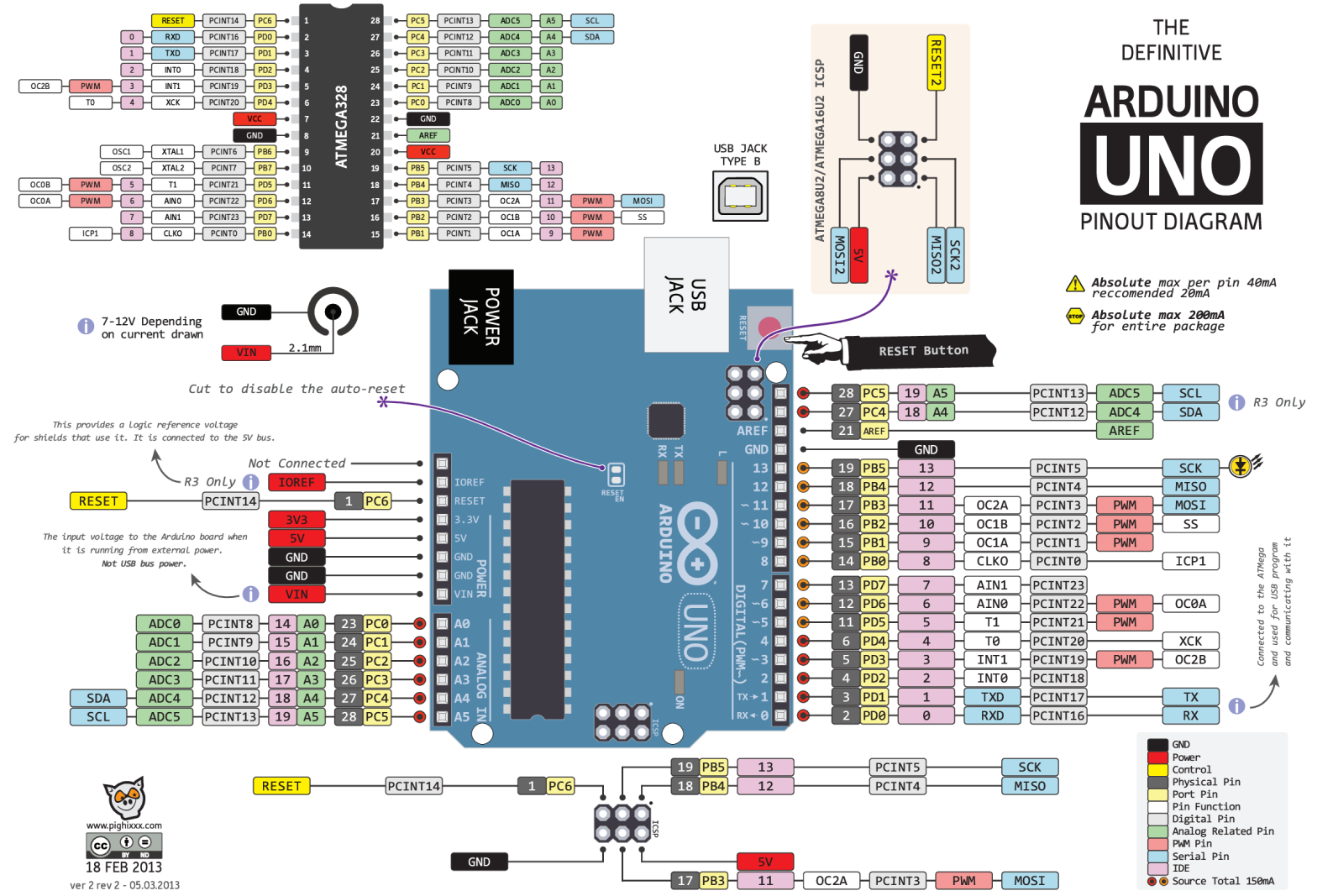
ver 2 rev 0 - 08.03.2013

THE DEFINITIVE ARDUINO MEGA PINOUT DIAGRAM

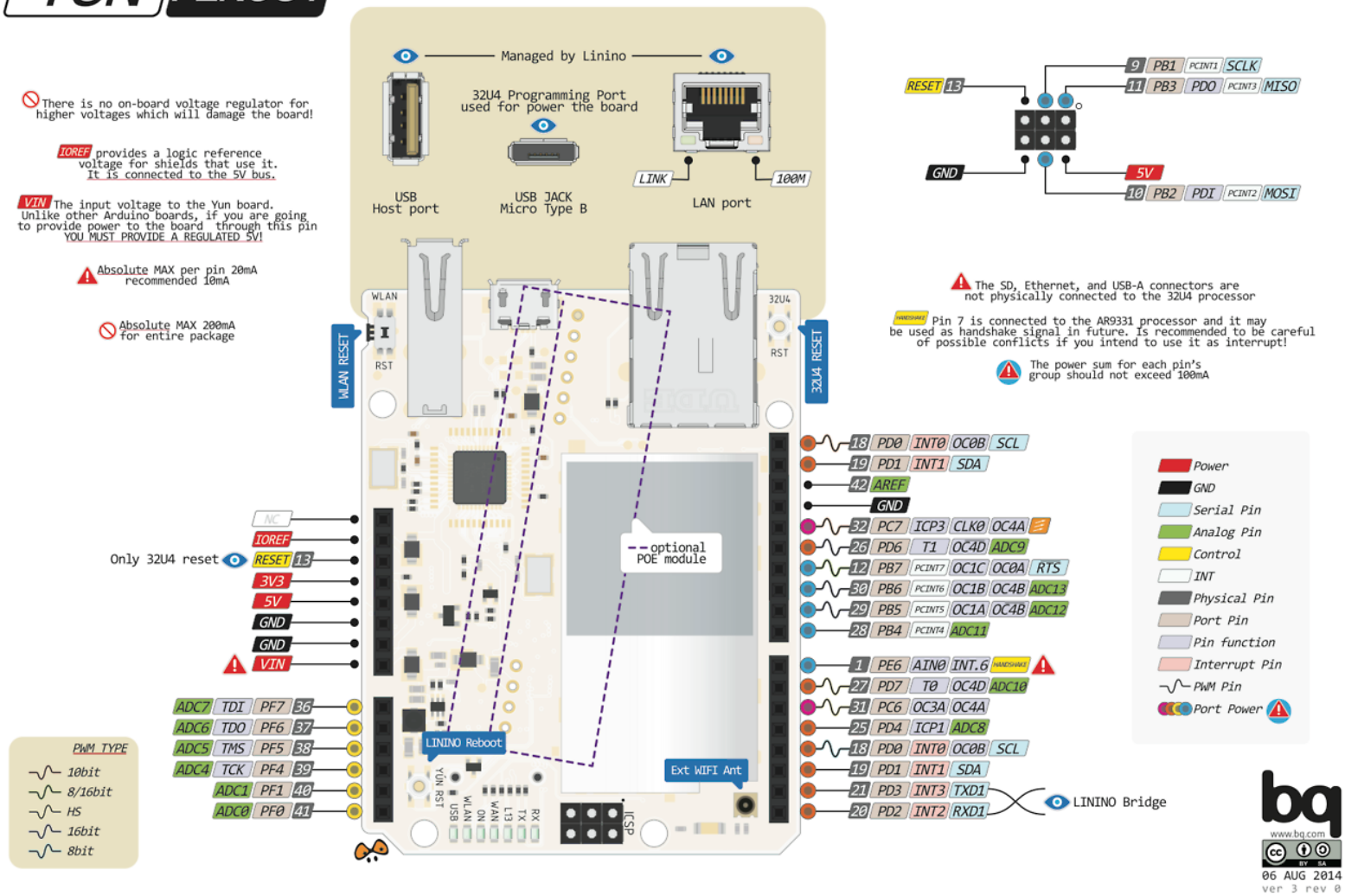


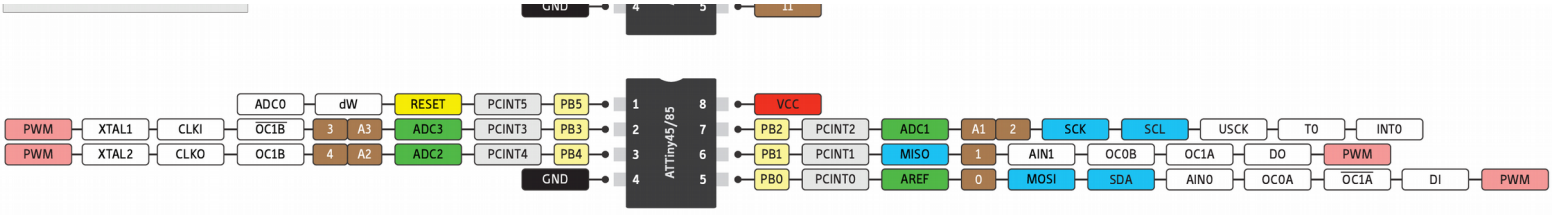
13 MAR 2013

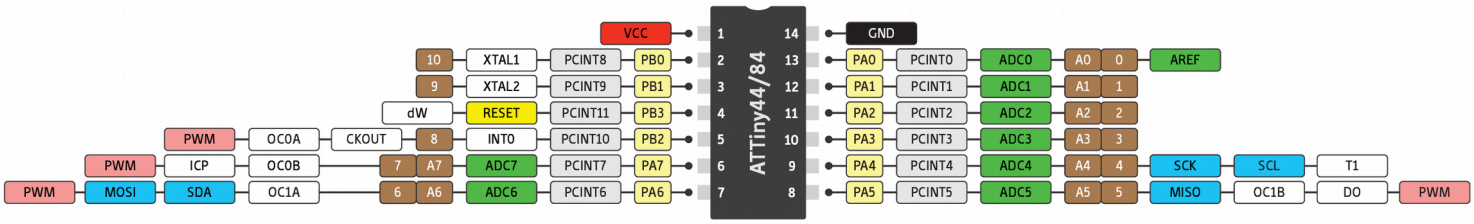
ver 2 rev 0 - 13.03.2013


THE DEFINITIVE
ARDUINO UNO
PINOUT DIAGRAM
ATMEGA328
ATMEGA8U2/ATMEGA16U2 ICSP
USB JACK TYPE B
POWER JACK
USB JACK
RESET Button
7-12V Depending on current drawn
2.1mm
Cut to disable the auto-reset
This provides a Logic reference voltage for shields that use it. It is connected to the 5V bus.
The input voltage to the Arduino board when it is running from external power.
Not USB bus power.
R3 Only
Not Connected
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

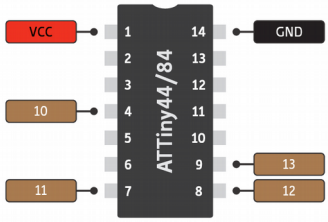
YÚN PINOUT







Using Arduino as ICSP Programmer for ATTiny44/84



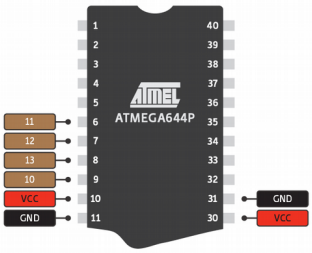
www.pighixxx.com
BY PIGHIXXX
29 JAN 2013

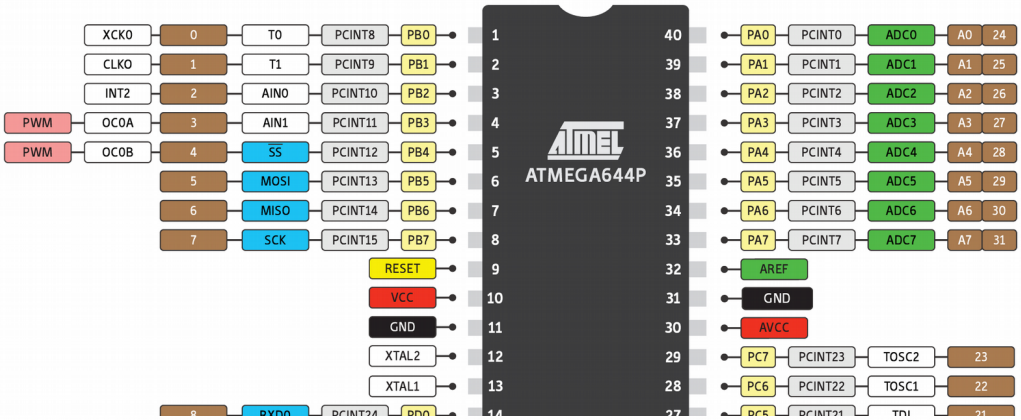
LEGEND

- GND
- POWER
- CONTROL
- PORT PIN
- ATMEGA PIN FUNC
- DIGITAL PIN
- ANALOG-RELATED PIN
- PWM PIN
- SERIAL PIN
- ARDUINO PIN

Using Arduino as ICSP Programmer for ATMEGA644P

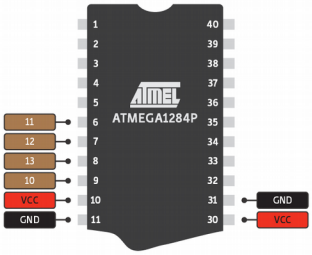
THE UNOFFICIAL **ARDUINO & ATMega644P** PINOUT DIAGRAM

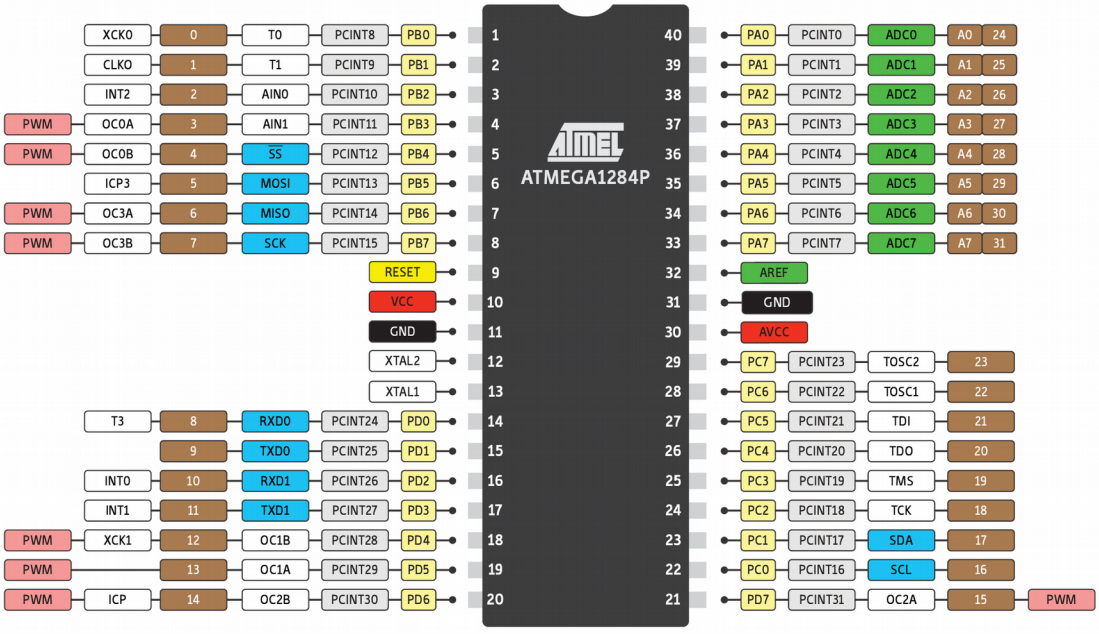




Using Arduino as ICSP Programmer for ATMEGA1284P

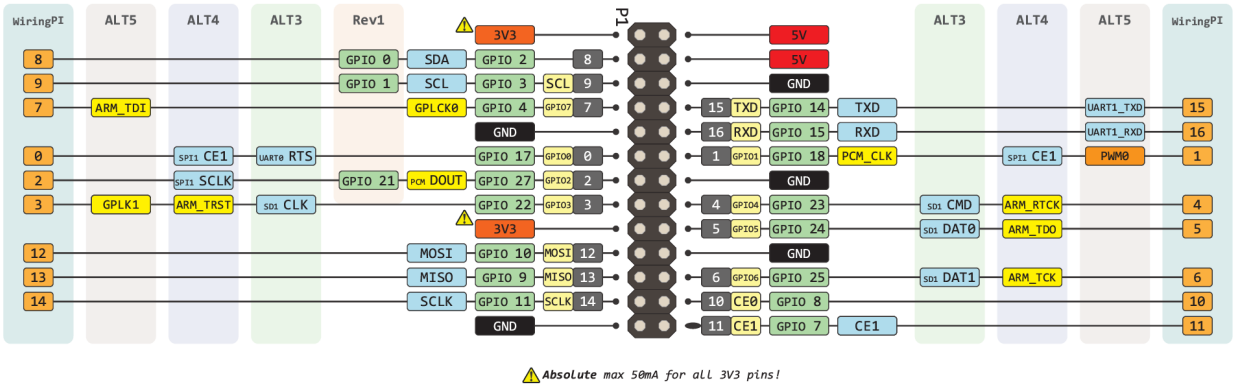
THE UNOFFICIAL **ARDUINO & ATMega1284P** PINOUT DIAGRAM

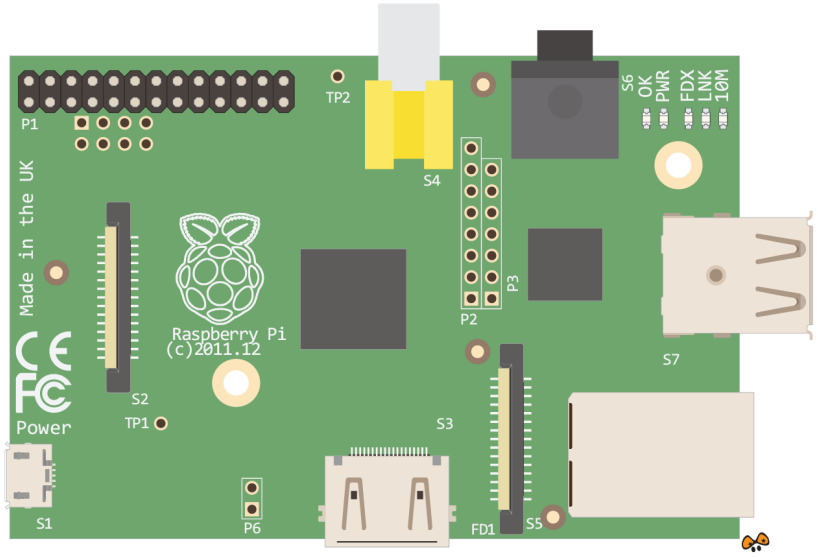


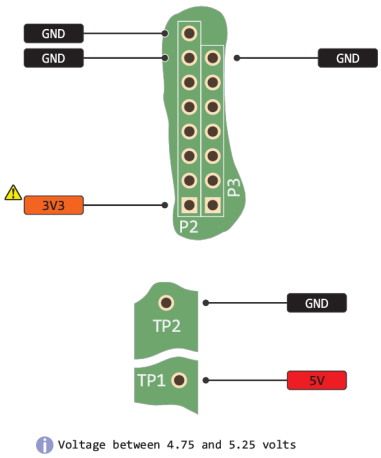


03 FEB 2013

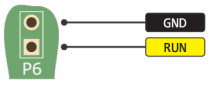
THE DEFINITIVE RASPBERRY PI PINOUT







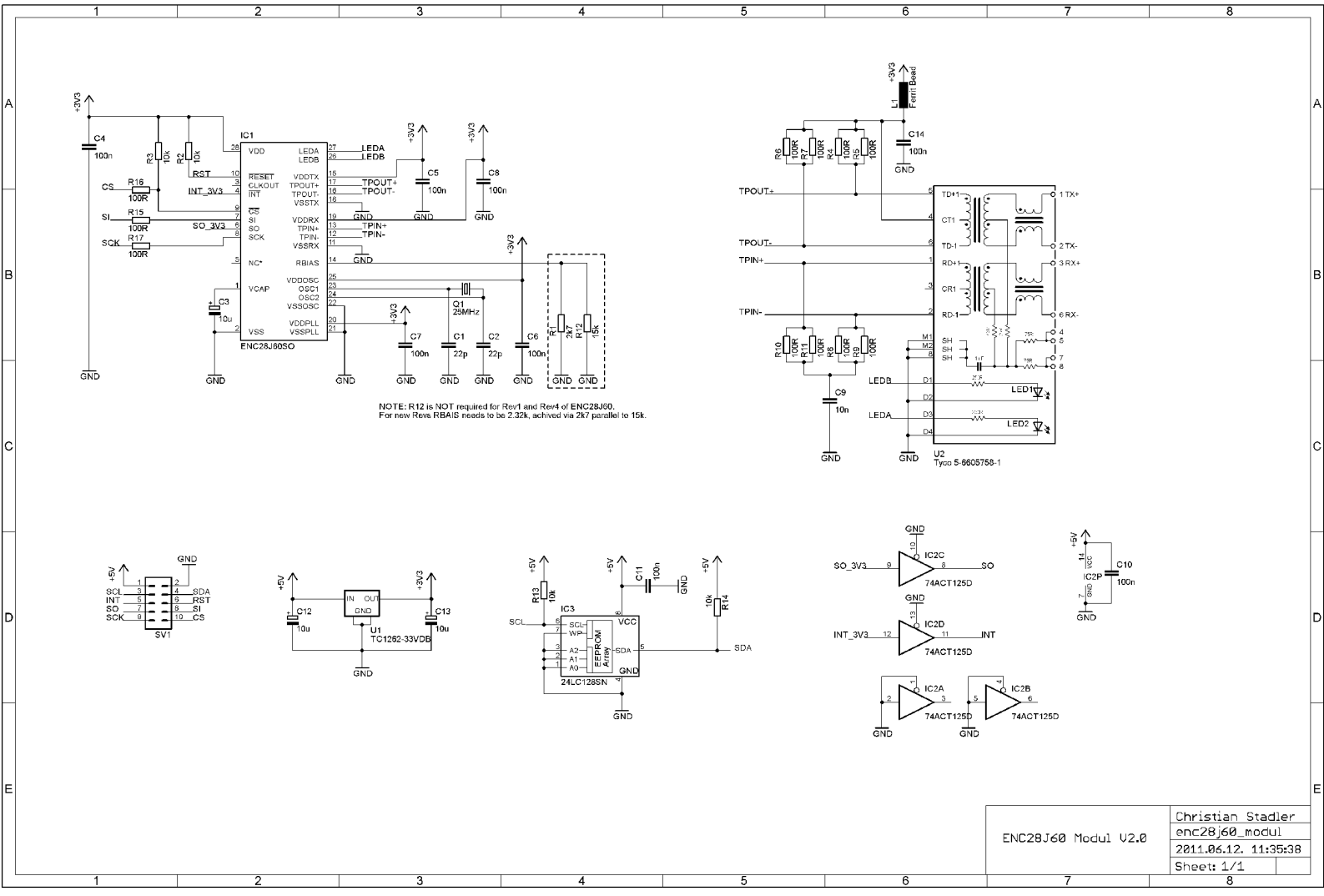
Voltage between 4.75 and 5.25 volts



Short to ground to reset the BCM2835

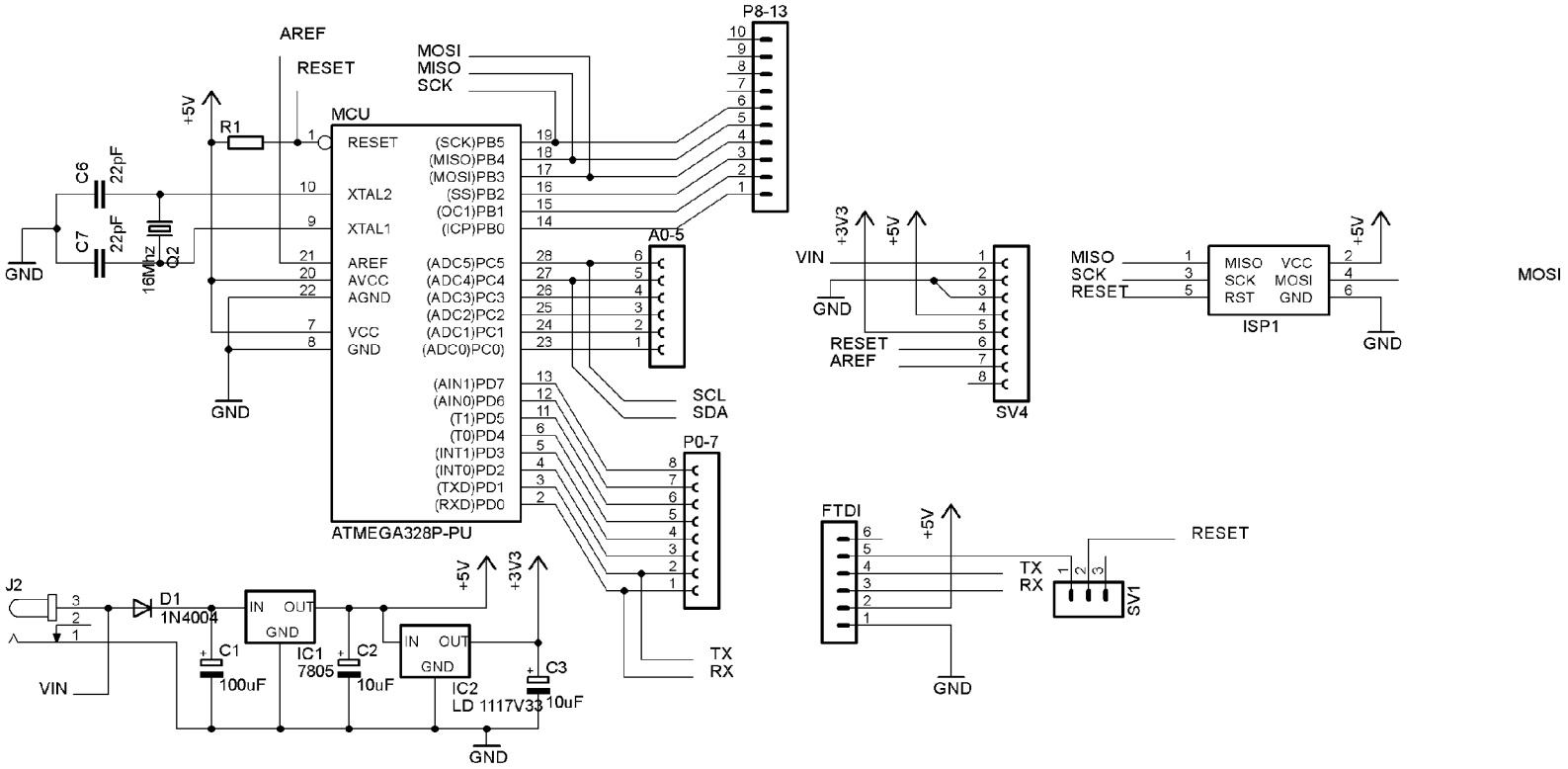
ENC26J60 vezérlőn alapuló Ethernet illesztő

Az Ethernet hálózatkezelés alapjai projektben említett modul kapcsolási rajza, amely a http://www.picprojects.net/enc28j60_modul/ címről származik.



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.


AREF
RESET
MOSI
MISO
SCK
+5V
R1
MCU
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
(SCK)PB5
(MISO)PB4
(MOSI)PB3
(SS)PB2
(OC1)PB1
(ICP)PB0
(ADC5)PC5
(ADC4)PC4
(ADC3)PC3
(ADC2)PC2
(ADC1)PC1
(ADC0)PC0
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
ATMEGA328P-PU
C6
22pF
C7
22pF
16Mhz
Q2
GND
P8-13
A0-5
SCL
SDA
P0-7
TX
RX
+3V3
+5V
VIN
GND
RESET
AREF
SV4
MISO
SCK
RESET
MISO
SCK
RST
VCC
MOSI
GND
ISP1
MOSI
FTDI
TX
RX
RESET
SV1
J2
D1
1N4004
IN
OUT
GND
C1
100uF
IC1
7805
C2
10uF
IC2
LD 1117V33
C3
10uF
VIN

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

Fejlesztéshez ajánlott eszközök

A könyvben szereplő hardver elemek többsége kedvező áron megvásárolható a Málna PC Magyarország webáruházában. Az általam ajánlott termékek:

Termék neve	Beszerzési hely
4GB SD kártya	http://malnapc.hu/yis/kingston-4gb-sdsd-micro-sdhc-class10
8GB SD kártya	http://malnapc.hu/yis/adata-8gb-sdsd-micro-sdhc-class-10-uhs-i-memoria
Adafruit Trinket 3,3V	http://malnapc.hu/yis/a1500-trinket-33v-mini-microcontroller-attiny85
Adafruit Trinket 5V	http://malnapc.hu/yis/a1501-trinket-5v-mini-microcontroller-attiny85-mcu
Arduino ADK	http://malnapc.hu/yis/arduino-adk-rev3-a000069
Arduino DUE	http://malnapc.hu/yis/arduino-due-a000062
Arduino Ethernet Shield	http://malnapc.hu/yis/arduino-ethernet-shield-rev3-without-poe-a000072
Arduino GSM Shield	http://malnapc.hu/yis/arduino-gsm-shield-integrated-antenna-a000043
Arduino Leonardo	http://malnapc.hu/yis/arduino-leonardo-headers-a000057
Arduino Mega 2560	http://malnapc.hu/yis/arduino-mega2560-rev3-a000067
Arduino Uno	http://malnapc.hu/yis/arduino-uno-rev3-a000066
Arduino Workshop kit	http://malnapc.hu/yis/kit-workshop-basic-with-arduino-board-a000010
Csipesz szett	http://malnapc.hu/yis/csipesz-szett-4dbcsomag
DS 1307 timer modul	http://malnapc.hu/yis/a264-ds1307-real-time-clock-rtc-modul-kit
Forrasztó ón	http://malnapc.hu/yis/forraszto-on-1-0mm-17g-hobby-celra
HDMI -> DVI átalakító	http://malnapc.hu/yis/adapter-hdmi-dvi-hdmi-aljzat-dvi-dugo
HDMI -> VGA átalakító	http://malnapc.hu/yis/hdmi-vga-atalakito-audio-adapter
I2C 12 bit DAC	http://malnapc.hu/yis/a935-dac-panel-mcp4725-12bit-i2c-interfesszel
Jumper kábelek	http://malnapc.hu/yis/premium-jumper-kabel-szett-64db-papa-papa
Logikai jelszint illesztő	http://malnapc.hu/yis/a395-logic-level-converter-txb0108
Próbapanel	http://malnapc.hu/yis/probapanel-125x160-forrasztas-nelkuli-800-pont
Raspberry PI GPIO próbapanel kábel	http://malnapc.hu/yis/a914-raspberry-pi-gpio-probapanel-osszekoto-kit
Raspberry PI kamera	http://malnapc.hu/yis/raspberry-pi-camera-board-5mp
Raspberry Pi kamera noIR	http://malnapc.hu/yis/raspberry-pi-noir-camera-board
Raspberry Pi model A	http://malnapc.hu/yis/raspberry-pi-a-verzio-256mb-ram
Raspberry Pi model B	http://malnapc.hu/yis/raspberry-pi-b-verzio-512mb-ram
Raspberry Pi model B+	http://malnapc.hu/yis/raspberry-pi-b-512mb-ram-2014-es-verzio
USB HUB	http://malnapc.hu/yis/trust-barra-mini-4-portos-usb-hub
USB kártyaolvasó	http://malnapc.hu/yis/kartyaolvaso-mp39-10in1-usb
USB WLAN modul	http://malnapc.hu/yis/raspberry-wi-pi-usb-s-wlan-modul-802-11n

131. Táblázat: Málna PC ajánlott termékek listája

Felhasznált irodalom

- ***Dr. Madarász László – Digitális jelfeldolgozás alapjai***
  Kecskeméti Főiskola Gépipari és Automatizálási Műszaki Főiskolai Kar, 1996
- ***Matijevics István – A digitális technika alapjai***
  Szegedi tudományegyetem, 2008
- ***Szabolcsi Judit – Bevezetés az objektum-orientált programozásba***
  Kecskeméti Főiskola Gépipari és Automatizálási Műszaki Főiskolai Kar, 2004
- ***Dr. Gruber Gábor – Digitális Technika, Tároló áramkörök és számlálók***
  Budapesti Műszaki Egyetem Híradástechnikai tanszék, 1977
- ***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
- ***Brad Dayley – Python zsebkönyv***
  ISBN: 978 963 9637 41 2
- ***Microchip PIC16F887 dokumentáció***
- ***Microchip PICKit 3 felhasználói kézikönyv***
- ***Mikroelektronika MikroC Súgó***
- ***Hitachi HD44780 kijelző vezérlő adatlap***
- ***Maxim DS1307 Real Time Clock adatlap***
- ***MAX 7221/7219 adatlap***
- ***MCP42xx adatlap***
- ***MCP23017 adatlap***
- ***EGBT-045MS adatlap***
- ***Microchip 24LC256 adatlap***
- ***How to control a HD44780-based Character-LCD***
  http://robo.fe.uni-lj.si/~kamnikr/sola/urac/vaja3_display/How%20to%20control%20HD44780%20display.pdf
- ***Serial Peripheral Interface Bus***
  http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
- ***I²C***
  http://en.wikipedia.org/wiki/I%C2%B2C
- ***DS1307 Real Time Clock RTC Library***
  http://www.circuitechs.com/embedded/embedded_codes/Ds1307_RTC_library/page_DS1307_RTC_library.html
- ***p-n átmenet***
  http://hu.wikipedia.org/wiki/PN_%C3%A1tmenet
- ***Dióda***
  http://hu.wikipedia.org/wiki/Di%C3%B3da
- ***Tranzisztor***
  http://hu.wikipedia.org/wiki/Tranzisztor
- ***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
- ***RaspberryPi Quick start guide***
  http://www.raspberrypi.org/wp-content/uploads/2012/04/quick-start-guide-v2_1.pdf
- ***FM Radio***
  http://en.wikipedia.org/wiki/FM_Radio
- ***Turning the Raspberry Pi Into an FM Transmitter***
  http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter
- ***C++ Reference***
  http://en.cppreference.com/w/
- ***Python Documentation***
  http://docs.python.org/2/

- ***Joint Test Action Group***
  http://en.wikipedia.org/wiki/Joint_Test_Action_Group
- ***Adafruit Graphics library***
  http://learn.adafruit.com/adafruit-gfx-graphics-library/
- ***Programmable logic array***
  http://en.wikipedia.org/wiki/Programmable_logic_array
- ***Programmable Array Logic***
  http://en.wikipedia.org/wiki/Programmable_Array_Logic
- ***Generic array logic***
  http://en.wikipedia.org/wiki/Generic_array_logic
- ***Complex programmable logic device***
  http://en.wikipedia.org/wiki/Complex_programmable_logic_device
- ***Field-programmable gate array***
  http://en.wikipedia.org/wiki/Field-programmable_gate_array
- ***Arduino Secret voltemeter***
  https://code.google.com/p/tinkerit/wiki/SecretVoltmeter
- ***Nagy Gergely – Bevezetés a Verilog alapú digitális tervezésbe***
  http://www.eet.bme.hu/~nagyg/mikroelektronika/verilog.pdf
- ***Verilog Manual***
  http://www.see.ed.ac.uk/~gerard/Teach/Verilog/manual/index.html
- *http://betterexplained.com/articles/a-visual-guide-to-version-control/*

Ábrajegyzék

Táblázatjegyzék