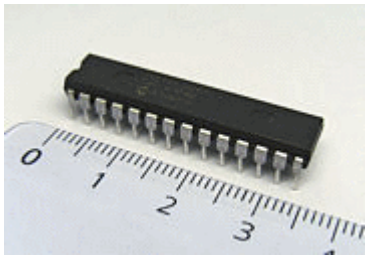
Mikrokontrollerek

– mérési leírás –

Ez a mérés a hallgató számára rövid betekintést kíván nyújtani a mikrokontrollerek világába. Semmiképp sem törekszik a teljességre, nem kíván átfogó mű lenni a témakörben. A téma iránt behatóbban érdeklődők bőséges információt találnak az interneten.
A mérés célja a "kedvcsinálás", a "műszaki étvágy" felkeltése ezen technika (hardver-szoftver határterülete) iránt.

Mikrokontrollerek

A mikrokontrollerek (másnéven mikrovezérlők) megjelenése a korszerű elektronikai ipar eredményei.
Az integráltsági fok látványos növekedése egyrészt jelentősen növelte a processzorok bonyolultsági fokát, javította teljesítőképességüket, másrészt lehetővé tette – az eddig külön lapkákon helyet foglaló – memória és perifériákat tartalmazó áramkörök egy tokban való integrálását a központi egységgel.
Mikrokontrollernek nevezzük az egy tokban helyet foglaló központi egységet, memóriát, ki- bemeneti egységeket és járulékos áramköröket tartalmazó "miniszámítógépet"!



"miniszámítógép"

A mikrokontrollerek a PC-kel ellentétben külön programmemóriával és külön adatmemóriával rendelkeznek (Harvard architektúra). A PC-k tervezése a Neumann elvet követte, vagyis ugyanazon operatív memóriában foglal helyet a program kódja is, mint a program által használt adatok.
Egy gépi utasítás lényegében az utasítás kódját (mit kell csinálni) és az operandus(ok) címét (mivel) tartalmazza. Az, hogy egy ilyen gépi utasítás hány biten írható le, attól függ, hogy hány darab utasítást kell kódolni, és mekkora a lehetséges operandusok halmaza. A PC gépi utasításainak hossza egy-két vagy több byte-os. (A Neumann elv miatt a gépi utasítások is byte-os szervezésűek kell legyenek, ugyanúgy mint az adatok.)
Ezzel szemben a mikrovezérlők programmemóriáját olyan szervezésűre ("szélesre") választották, hogy egy gépi utasítás egy programmemória címen férjen el, ezáltal sokkal gyorsabb programvégrehajtás érhető el.

Felhasználási kör

A mikrovezérlők felhasználási köre igen széles. Összességében elmondható, hogy minden olyan feladat, ahol valamit elektronikusan kell/lehet mérni ill. vezérelni, az mikrokontrollerek használatával eredményesebben megoldható, mint hagyományos áramkörökkel.

A feladat megoldásának lépései: egyrészt megtervezni a megvalósítandó áramkör hardverét, másrészt meg kell írni az adott hardvert működtető programot. A hardver megtervezése a mikrokontroller család kiválasztásával kezdődik, majd a családon belüli igen szerteágazó és széles skálájú típusválaszték konkrét kijelölésével folytatódik. Az előbbinél jelentős szerepet játszik a családhoz kapható ill. meglévő fejlesztőkörnyezet elérhetősége, míg a típusválasztásnál az adott feladat megoldásához leginkább "simuló" mikrovezérlő kiválasztása a cél.

Fejlesztőkörnyezet

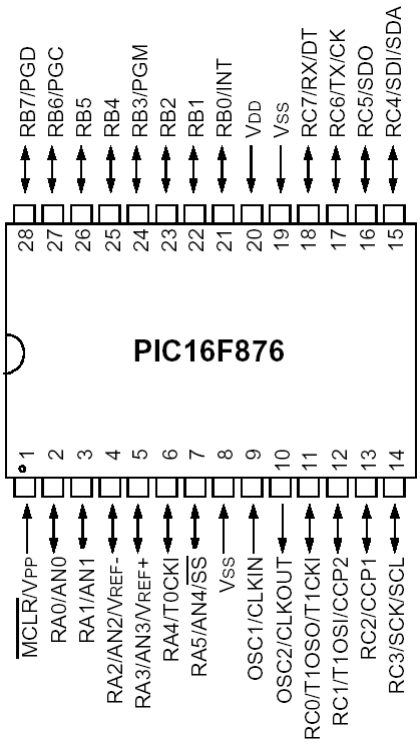
Az elektronikus alkatrészboltban megvásárolt mikrovezérlővel önmagában nem sokat tudunk kezdeni, szükségünk lesz legalább egy – a kedvenc programozási nyelvünkön működő – *fordító*ra, és egy *programozó egység*re. Találunk fordítókat asm, pascal, c és basic nyelvekre, ezek egy része fizetős, más részük bizonyos felhasználói korlátozásokkal ingyenes.

A laboratóriumi mérés készítésekor a választás a MikroElektronika cég (`http://www.mikroe.com/en/compilers/`) *fordító*ira esett, mindhárom "alapnyelven" (pascal, c, basic) kínálnak fordítókat mind a "PIC"-es, mind pedig az "Atmel" mikrokontroller család részére. A fordítók szabadon letölthetők és használhatók mindaddig, amíg a fordított programkód el nem éri a 2k (2048) szót. Ez a méretkorlát elegendően nagy ahhoz, hogy egyszerű (de nagyszerű) alkalmazásokat fejleszthessünk.

A mikrokontroller család kiválasztása a Microchip cég által gyártott (PIC=Programmable IC) termékekre esett. A családnak több ága és tagja van (500-2000Ft között), amelyek használatával az alkalmazások ár/teljesítmény aránya a legkedvezőbbé tehető.

A fejlesztőkörnyezet másik fontos eleme a *programozó egység*, ennek segítségével visszük át a mikrokontrollerbe a PC-n előállított programkódot. Ha vásároljuk a programozót (n x 10000Ft) általában integrált hardveres nyomkövető (programjaink "debuggolására") rendszerrel kapjuk, de ha ez utóbbira egyelőre nincs szükségünk, és egy kis affinitást is érzünk az elektronikus áramkörök építése iránt, magunk is készíthetünk programozó egységet (n x 100Ft). Minimális alkatrészigényű programozók kapcsolási rajza az internetről könnyen beszerezhető.

16F876 mikrovezérlő rövid ismertetése



PIC16F876 bekötése

Fontosabb műszaki jellemzők:

-hatékony teljesítményű – csökkentett utasításszámú (35 utasítás) – RISC processzor,
-órajelfrekvencia: 0 – 20 MHz,
-programmemória: 8192 szavas (14 bit széles) FLASH memória,
-adatmemória: 368 byte RAM és 256 byte EEPROM,
-veremmemória (8 szint mély)
-megszakításkezelés (14 forrás válthat ki megszakítást)
-watchdog timer
-energiatakarékos SLEEP üzemmód
-3 időzítő áramkör Timer0, Timer1 és Timer2 (8, 16, 8 bitesek)
-10 bites A/D konverter
-2 db PWM (impulzus szélesség modulált) kimenet,
-soros RS232 kommunikáció létrehozására alkalmas USART port
-digitális be/kimenetek PortA, PortB, PortC (5, 8, 8 bitesek)

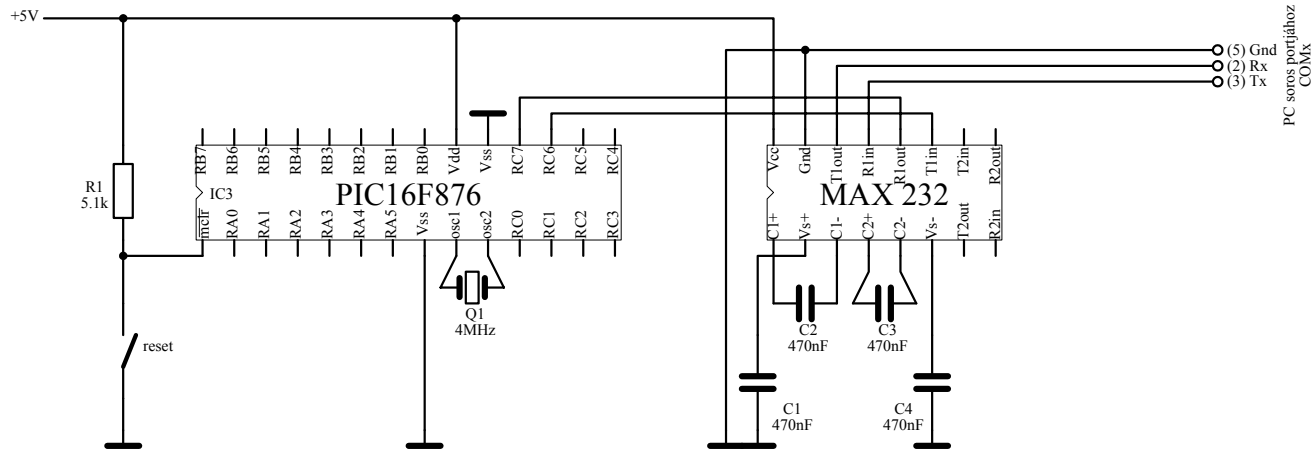
Mérési összeállítás

A laboratóriumi mérési összeállítást sikerült úgy leegyszerűsíteni, hogy még programozó egység használatára se legyen szükség! A *flash* programmemóriával rendelkező PIC-ekben futó program képes módosítani a programmemória tartalmát. Ezért ezekbe a PIC-ekbe elegendő, ha egyszer írunk programozó egységgel egy ún. "*boot*" programot.

A *boot* program pici és a programmemória felső részén foglal helyet, szabadon hagyva a memória "alsó" felének nagy részét a felhasználói alkalmazás számára. Feladata, hogy bekapcsolást ill. resetet követő néhány másodpercben figyelje a PIC soros perifériáját (*Usart*), hogy óhajt-e vele valaki adatot cserélni. Ha igen, és sikeresen azonosította a PC-n futó *bootloader* programot, akkor megkezdődik a PC felől jövő adatfolyam programmeóriába írása (a PIC programozása), ellenkező esetben lezárja, vagy meg sem kezdi a nem létező soros kommunikációt. Ilyenkor (az esetek többségében, amikor nem programozzuk, "csak" egyszerűen használjuk a

PIC-et) néhány másodperces türelmi idő letelte után, a program végrehajtás a PIC-ben található felhasználói programmal folytatódik.

A laboratóriumban használt "minimális mérési összeállítás" az alábbi ábrán látható. Ennyi legalább mindig legyen összerakva, **ennél jobban ne "csupaszítsuk le" a próbalapot!**



"minimális mérési összeállítás"

A próbalapon összeállított fenti kapcsolás soros kábelen keresztül kapcsolódik egy számítógép soros portjához. Ezen a kábelen keresztül töltjük le a számítógépen előállított (megírt és lefordított) programot a mikrovezérlőbe, és ugyanezen kábelen keresztül kommunikálhat a mikrovezérlőben futó felhasználói program pl. egy PC-s programmal.

A programírás, fordítás lépései:

Indítsuk el a PC-n a *mikroPascal* programot. Ha egy új programot szeretnénk elkezdeni, akkor a *mikroPascal* által megnyitott legutóbb használtat csukjuk be (*File/Close All*). Kezdjünk egy új projectet (*Project/New Project...*) Adjuk meg az új programunk nevét (*Project Name:*), elérési útvonalát *(Project Path:*), a mikrovezérlő típusát (*Device:P16F876*), és a kristályrezonátor frekvenciáját (*Clock: 4MHz*), majd nyomjuk meg az *Ok* gombot.

Megj.: Új programot mindig új mappában kezdjünk!

A fenti módon létrehozott új project két file-t hoz létre a választott mappában. Egy *.ppp* kiterjesztésű project file-t és egy *.ppas* kiterjesztésű pascal forrásfile-t. A pascal forrásfile-t látjuk megnyitva a szerkesztőben. Írjuk meg a programunkat. Szerkesztés közben sokat segít az integrált fejlesztői környezet *kódkiegészítő* funkciója, mely Ctrl-Szóköz-el hívható elő az egyes utasítások valamint függvény/eljárások gépelése közben (beleértve a beépített könyvtárak és a magunk által deklarált függvény/eljárásainkat is). A beírt függvény/eljárás nevet követő nyitó zárójel máris előhívja a használandó paraméterlistáról tájékoztató súgót. (Egyébként ez utóbbi funkció Ctrl-Shft-Szóköz-el csalogatható elő).

F1-el pedig bármikor előhívható a *mikroPascal* súgója, amiben részletes segítséget találunk egyrészt a pascal szintaktika és használható utasítások, másrészt a beépített könyvtárak függvény/eljárásait illetően.

A program fordítása *Project/Build* menüpontokkal lehetséges, az esetleges szintaktikai hibákról a fordító a szerkesztő alatti *Messages* ablakban tájékoztat. Ugyanitt jelenik meg a sikeres fordítás után előálló – PIC-be beírandó – programkód hossza, valamint a felhasznált RAM méret.

Megj: A PIC RAM memóriája – változóink helye – igencsak véges: **368 byte!** Ezt a tényt végig tartsuk szem előtt a program tervezése és írása közben! Nem egy PC-re írunk "mamutprogramot" mindenféle összetett adatstruktúrák és objektumok deklarálásával, hanem egy mikrovezérlőre!

A kész, hiba nélkül lefordított programjainkat "debuggolhatjuk", a változók, regiszterek értékeit nyomon követhetjük a futás során. Ehhez előbb indítsuk el a nyomkövetőt Run/Start Debugger, majd a Watch ablakban válasszuk ki a megfigyelni kívánt változók és regiszterek neveit. A szerkesztő ablakban levő programsorokat a szövegkurzorig F4-el egyhuzamban végrehajtja, F7 és F8 a lépésenkénti végrehajtást szolgálja, azzal a különbséggel, hogy függvény/eljárás esetén az előbbi belép, míg az utóbbi egylépésben hajtja végre. Mindezek mellett a szerkesztőablak bal margóján töréspontokat (Breakpoints) is bekapcsolhatunk.

Az előbbi sorokban említett nyomkövető rendszer egy nagyon hatékony eszköz programunk logikai (amikor nem úgy működik, ahogy elképzeltük) hibáinak kiszűrésére.

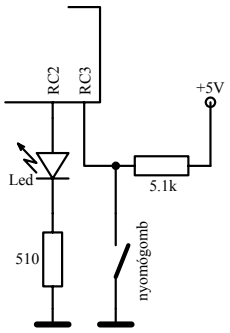
A kész lefordított program letöltése PIC-be

Tools/mikroBootloader menüpontokkal indítsuk el a letöltő programot. Az egérkurzorral álljunk rá, de még ne nyomjuk meg a *Connect* gombot! Előbb a PIC resetjét nyomjuk meg a próbalapon, majd ezt követően néhány másodpercen belül a *Connect* gombot. Ha jól csináltuk (PIC táp alatt, soros kábel a helyén, gombnyomkodás sorrendje helyes) a *History Window*-ban azt látjuk: *"Connected"*, az *Open HEX file*-al válasszuk ki a letöltendő *hex* formátumba fordított PIC programot, majd *Start bootloader*. Ekkor látható módon elindul a program letöltése. A letöltés végét a PIC újraindítására figyelmeztető üzenet jelzi. Hajtsuk végre a PIC újraindítását a reset gombbal!

Feladatok:

1. Led villogtatás, azaz a mikrovezérlő digitális ki- bemeneteinek megismerése, használata

A műszaki jellemzők között láttuk, hogy mikrovezérlőnk A, B, C portokkal (`PortA, PortB, PortC`) rendelkezik, amelyek rendre 5, 8, 8 bitesek. Ezen bitek mindegyike használható digitális ki- és bemenetként. A portokhoz kapcsolódó irányregiszterek tartalmai szabják meg azt, hogy egy adott bit éppen bemenet-e, avagy kimenet. Kimenet akkor, ha az irányregiszter megfelelő bitje nulla, egyébként bemenet. Alapértelmezésben a portok összes bitjei bemenetek. Az irányregiszterek elnevezései `TrisA, TrisB` és `TrisC`, ezek mindegyike programból szabadon írható olvasható.



LED és nyomógom csatlakoztatása a PIC-hez

Minimális mérési összeállításunkat egészítsük ki az előbbi kis ábrán látható kapcsolással, azaz a C port 3.-ik bitjét használjuk bemenetként és egy nyomógombbal (vagy kapcsolóval) határozhassuk meg logikai állapotát (elengedett nyomógomb → magas szint → logikai 1, megnyomott gomb → alacsony szint → logikai nulla)!
C portunk 2.-ik bitje viszont legyen digitális kimenet, ha éppen magas logikai szintű Ledünk világítani fog, ha alacsony a Led nem világít!
A hardver összeállításhoz írjunk programot úgy, hogy elengedett nyomógomb mellett egy bizonyos kitöltési tényezővel villogjon a Led, megnyomott nyomógomb esetén egy másik kitöltési tényezővel!

```
program led_villog;
var kapcsolo:boolean;
{-------------------------------}
procedure Villan(tbe,tki:word);
begin
  PortC.2:=1; VDelay_ms(tbe);
  PortC.2:=0; VDelay_ms(tki);
end;
{-------------------------------}
begin
  TRISC:=TRISC AND %11111011;       //C2 legyen kimenet
  TRISC:=TRISC OR  %00001000;       //C3 legyen bemenet
  while true do begin
    kapcsolo:=PortC.3;
    if kapcsolo then Villan(300,200) else Villan(50,450);
  end;
end.
```

A főprogram elején beállítjuk a C irányregiszter 2. és 3. bitjét. A másodikat töröljük, a harmadikat 1-be állítjuk, a többit nem bántjuk! Szokjuk meg ezt a "programozási fogást" még akkor is, ha jelen helyzetben mondhattuk volna, hogy bekapcsolás után a TrisC összes bitje 1, azaz nekünk csak a 2.-at kell nullázni, tehát: `TRISC:=$FB;` írok a két TrisC-s sor helyett.
Ezután a főprogram egy végtelen ciklussal folytatódik, vagyis rendre villantjuk a Ledet a nyomógomb állásától függő tbe, tki paraméterekkel meghatározott módon.

2. Led fényerő változtatása, azaz a PWM modul megismerése és használata

PWM = Pulse Width Modulation, azaz impulzusszélesség moduláció egy olyan négyszögjelet jelent, aminek amplitúdója és periódusideje állandó, csak kitöltési tényezője változik. Könnyen belátható, hogy egy ilyen négyszögjel átlagértéke (egyen komponense) a kitöltési tényezővel arányos.

A vizsgált mikrovezérlőnkbe két ilyen egységet is építettek, az egyik a 13-as, a másik a 12-es lábon érhető el (lásd: 16F876 mikrovezérlő rövid ismertetésénél közölt bekötési rajz *CCP1* és *CCP2* kimenetei).

A hardver összeállítás marad az előző feladatnál bemutatott: ezúttal a nyomógombot/kapcsolót nem használjuk, csak a 13-as láb (*CCP1*) és föld közé kapcsolt Led + ellenállás kétpólust.

Írjunk olyan programot a PIC-re, amely a *PWM1*-es modult kelti életre egy 1000Hz-es ismétlődési frekvenciájú és 0-100%-ig növekvő, majd csökkenő kitöltési tényezőjű négyszögjellel. A Led valójában villogni fog, de a szemünk folyamatosan erősödő, majd gyengülő Led-fényerőt érzékel (miért is?), aszerint, ahogy a generált négyszögjel kitöltési tényezője nő vagy csökken.

```
program led_fenyero;
var i:byte;
    irany:boolean;
{-------------------------------}
begin
  TRISC:=TRISC AND %11111011;       //C2 legyen kimenet
  PWM1_Init(1000);
  PWM1_Start;
  i:=0; irany:=False;
  while true do begin
    PWM1_Change_Duty(i);
    Delay_ms(10);
    if (i=0) OR (i=255) then irany:=Not(irany);
    if irany then inc(i) else dec(i);
  end;
end.
```

A C irányregisztert ismét beállítjuk úgy, hogy a C port 2.-ik bitje kimenet legyen, `PWM1_Init` inicializálja az első *PWM* modult, paraméterként a majdan generálandó négyszögjel frekvencia értékét kell megadnunk (1000 Hz). A *PWM* modul bekapcsolását a `PWM1_Start` végzi, kikapcsolását a `PWM1_Stop`. Ez utóbbira most nincs szükség, hisz azt szeretnénk, ha *PWM* modul végig bekapcsolva maradna. Természetesen, ha a feladat úgy kívánná bármikor a program futása közben is (pl. valamilyen feltétel teljesülésekor) leállíthatjuk a `PWM1_Stop` eljárással. A `PWM1_Change_Duty` segítségével lehet állítani a kitöltési tényezőt, amely paraméterként egy *byte*-ot vár, vagyis 0 érték → 0%-os, 128-as érték → 50%-os, míg 255-ös érték → 100%-os kitöltési tényezőt jelent.

3. Soros kommunikáció a PIC-ben futó felhasználói program és a PC között, az USART egység használata

A PIC-be épített USART (USART=Universal Synchronous Asynchronous Receiver Transmitter) egységet regisztereken keresztül az alábbi módokban lehet felprogramozni és használni:

-aszinkron kommunikáció,

-szinkron kommunikáció - Master üzemmód,

-szinkron kommunikáció - Slave üzemmód.

Az elsőt többnyire PIC és PC közötti soros adattovábbítás céljaira használják, míg a következő kettő PIC és perifériák (*AD konverter, EEPROM*) közötti kommunikációt segíti.

A *mikroPascal* szerencsére tartalmaz egy *USART Library* beépített könyvtárat az *USART* egység kezelésére, így annak beállításait a PIC regisztereinek írása helyett egyszerű parancsokkal végezhetjük.

Egy nagyon egyszerű feladatot tűzzünk ki magunk elé célul, a PIC-ben levő program küldjön adatokat (számokat) a PC-n futó terminálablak számára.

```
program Usart_teszt;
var i:byte; t1, t2:string[20];
{-------------------------------------------------------}
begin
  i:=0; t1:='Számokat küldözget'; t2:='ek a következőkben:';
  Usart_Init(9600);
  Usart_Write_Text(t1);
  Usart_Write_Text(t2);
  while true do begin
    Usart_Write(i);
    Delay_ms(500);
    inc(i);
  end;
end.
```

A program elején feltöltünk kezdőértékkel két karaktertömböt (`t1` és `t2`), majd az *USART* egység 9600 *baud* átviteli sebességre való felprogramozása után (`USART_Init`), ezt a két tömböt kiküldjük a PC *soros portja* felé. A továbbiakban belépünk a végtelen ciklusba, ahol félmásodpercenként egy *byte*-os változó lépésenként növelt értékét továbbítjuk sorosan.

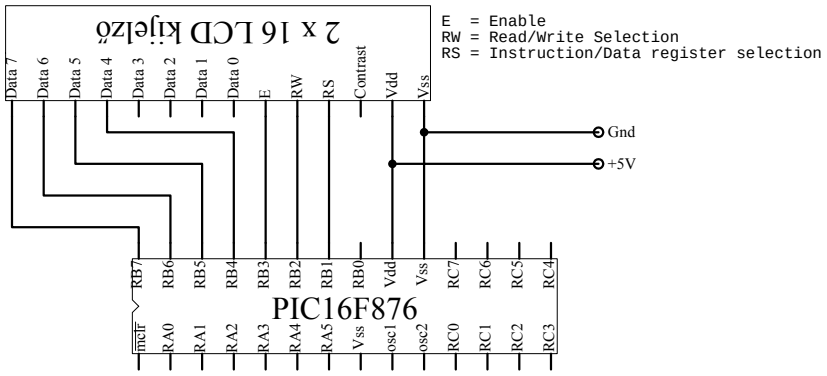
Hogy mindezt kellőképpen élvezhessük a PC oldalon szükségünk van egy terminálablakra, amit a *mikroPascal Tools/USART Terminal* menüjével nyithatunk. A terminálablakban egy *Connect*-el kapcsolódjunk PIC-ünkhöz és figyeljük a beérkező adatokat különböző formátumokban *ASCII, HEX, DEC*!

Figyeljük meg azt is, mi történik miután reseteljük a PIC-et (vajon mi az a közvetlen újraindítást követő adatfolyam?)!

Megj.: A *bootloader* használata előtt, mindenképp zárjuk le a terminálablak soros kommunikációját, különben nem tudjuk új programmal frissíteni a PIC tartalmát!

4. Karakteres LCD kijelző használata

Kétsoros, soronként 16 karakter hosszúságú karakteres LCD kijelzőt használunk. Az LCD kijelző hardveres csatlakoztatása az alábbi ábra szerint történik.



2x16 LCD kijelző csatlakoztatása a PIC-hez

Az LCD kijelző drága és sérülékeny eszköz, óvatosan bánjunk vele! Csak azután helyezzük tápfeszültség alá, ha 100%-osan biztosak vagyunk abban, hogy jól csatlakoztattuk! Kérjük a gyakorlatvezető segítségét!

```
program Lcd_teszt;
var i:byte;
    t1:string[16];
    t2:string[16];
{--------------------------------------------------------}
begin
  i:=0;
  t1:='*** LCD teszt **';
  t2:='                ';
  Lcd_Config(PORTB,7,6,5,4,PORTB,1,2,3);

  while True do begin
    t2[i mod 16]:=32+i;
    LCD_Cmd(LCD_CLEAR);
    LCD_Cmd(LCD_CURSOR_OFF);
    LCD_Out(1,1,t1);
    LCD_Out(2,1,t2);
    i:=(i+1) mod 96;
    Delay_ms(500);
  end;
end.
```

Az LCD kijelző kezelésére a *mikroPascal* az *LCD Library*-t bocsájtja rendelkezésünkre. Az `LCD_Config` eljárással közöljük, hogy milyen hardveres összeállítást használunk. A fenti kapcsolási rajznak megfelelő paramétereket látjuk épp beírva a programba.

A példaprogram azt mutatja, hogyan lehet egy-egy karaktertömböt kiíratni az LCD kijelző első és második sorába. További lehetőségek, valamint az `LCD_Cmd` paramétereinek lehetséges értékei felől tájékozódjunk a *Súgó*ból.

5. A PIC AD konverterének használata

Mikrovezérlőnkbe egy 10 bites AD konvertert építettek, amelyet megelőz egy 5 bemenetű analóg multiplexer. Így akár 5 különböző analóg forrásból származó feszültséget is rendre az AD konverter bemenetére kapcsolhatunk. A multiplexer bemeneteit az A port A_0, A_1, A_2, A_3, A_4 bitjein érjük el. Az `ADCON1` regiszter tartalma határozza meg, hogy ezekből melyik analóg és melyik a már korábban megismert digitális bemenet. Azt is beállíthatjuk, hogy az AD konverzió során – az AD konverter egység – mit tekintsen referencia feszültségnek.

ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	Vss	2/1
011x	D	D	D	D	D	D	D	D	VDD	Vss	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

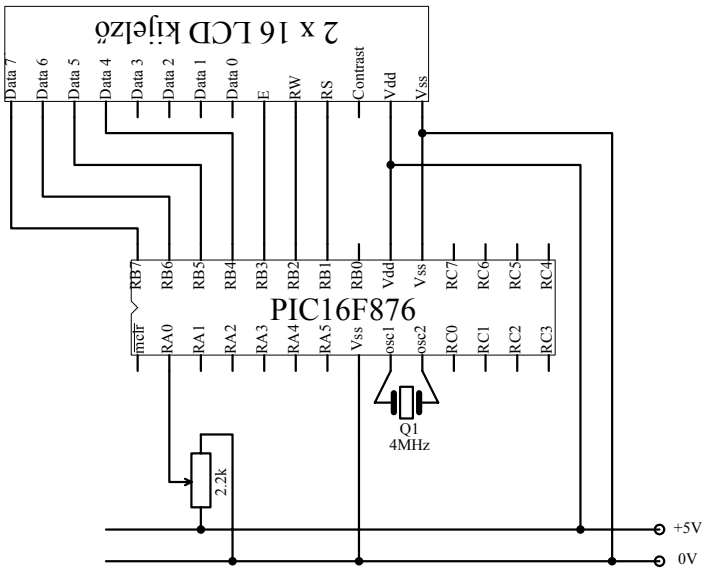
`PCFG3..PCFG0` bitek jelentését tanulmányozzuk a fenti táblázatban. Ha pl. a konverzió pontossága nem olyan fontos és megelégszünk, a mikrovezérlő tápfeszültségeivel referencia gyanánt, akkor használjuk nyugodtan a `PCFG3..PCFG0=0000` értéket. Igényes esetben viszont gondoskodjunk stabil V_{ref+} és V_{ref-} feszültségértékekről az A_3 A_2 bemeneteken és használjuk a `PCFG3..PCFG0=1000; PCFG3..PCFG0=1011 stb.` esetek egyikét!

Példaprogramunkban az LCD kijelző inicializálása után az A port irányregiszterét (az összes bitet) bemenetre állítjuk, majd az `ADCON1` regiszternek azt mondjuk, hogy az eredmény legyen jobbra igazított, az összes bemenet az A porton analóg, míg referencia feszültségek legyenek a mikrovezérlő tápfeszültségei.

```
program ADkonverter_teszt;
//Hardver:  az  alapösszeállításon  túl  PortA.0-hoz  egy  potenciométerrel  változtatható
feszültség kapcsolódik
//         az eredmény megjelenítésére az LCD kijelzőt használjuk
var ad:word;
    ad_str:string[5];
{------------------------------------------------------}
begin
  Lcd_Config(PORTB,7,6,5,4,PORTB,1,2,3);
  LCD_Cmd(LCD_CLEAR);
  LCD_Cmd(LCD_CURSOR_OFF);
  LCD_Out(1,3,'AD konverzio');
  TRISA:=$FF;
  ADCON1:=$80;
  while True do begin
    ad:=Adc_Read(0);
    Delay_ms(500);
    WordToStr(ad,ad_str);
    Lcd_Out(2,6,ad_str);
  end;
end.
```

A végtelen ciklusban, amit félmásodpercenként ismételgetünk elvégezzük az AD konverziót a 0.-ik csatornán (A_0 bemeneten), az eredményt `ad` változóban tároljuk és kiíratjuk az LCD kijelzőre. Mivel az AD konverter 10 bites, ezért egy 0..1023 közötti számot kapunk eredményül.

Használjuk az alábbi hardver kiegészítést:



Sok sikert mindenkinek a mikrovezérlők világában!