Mérés és adatgyűjtés
Laboratóriumi jegyzet

Mingesz Róbert, Gingl Zoltán

2014

A tananyag a TÁMOP-4.1.2.A/1-11/1-2011-0104 "A felsőfokú informatikai oktatás minőségének fejlesztése, modernizációja" c. projekt keretében a Pannon Egyetem és a Szegedi Tudományegyetem együttműködésében készült.

ÚJ SZÉCHENYI TERV

MAGYARORSZÁG MEGÚJUL

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Mérés és adatgyűjtés
Laboratóriumi jegyzet

Írták:

Dr. Mingesz Róbert Zoltán és Dr. Gingl Zoltán

Lektorálta:

Dr. Szabó István

Kulcsszavak:

LabVIEW, méréstechnika, adatgyűjtés, adatok feldolgozása, A/D konverzió, D/A konverzió, műszerek, távmérés, szenzorok

Összefoglalás:

A Mérés és adatgyűjtés laboratóriumi gyakorlat célja, hogy az elméleti alapok elsajátítása után a hallgatók a gyakorlatban is kipróbálják a tudásukat és tapasztalatot szerezzenek a különböző mérési eljárások, mérési adatok feldolgozása valamint a mérő és adatfeldolgozó szoftverek terén. E jegyzet célja, hogy hasznos segédanyagot biztosítson a hallgatóknak a laboratóriumi gyakorlat teljesítéséhez.

Tartalomjegyzék

Bevezetés

A Mérés és adatgyűjtés laboratóriumi gyakorlat célja, hogy az elméleti alapok elsajátítása után a hallgatók a gyakorlatban is kipróbálják a tudásukat és tapasztalatot szerezzenek a különböző mérési eljárások, mérési adatok feldolgozása valamint a mérő és adatfeldolgozó szoftverek terén.

A laboratóriumi gyakorlatok végrehajtása során jelentős szerepet kap a LabVIEW programozási környezet, ez egy megbízható és jól kezelhető hátteret biztosít a mérési szoftverek megvalósításához.

Jelen jegyzet szorosan kapcsolódok a gyakorlat tematikájához, az egyes laboratóriumi feladatsorokhoz biztosítja a szükséges háttértudást. Minden egyes fejezet végén van néhány gyakorló példa. Ezek némely részlete szándékosan nincs teljesen specifikálva, ugyanis a laboratóriumban lévő eszközök időben változatnak.

Bár a jegyzet elsősorban a Szegedi Tudományegyetem Mérnök informatikus BSc szak tantervében szereplő Mérés és adatgyűjtés laboratóriumi gyakorlat tanmenetét követi, bízunk benne, hogy további hallgatóknak, esetleg már az iparban dolgozó szakembereknek is hasznos információkat nyújthat. A szerzők szívesen fogadnak bármilyen visszajelzést, amelyek segíthetik, hogy a jegyzet újabb kiadásai a lehető legjobb és legfrissebb tudást nyújtsa olvasóinak.

A jegyzet a TÁMOP-4.1.2.A/1-11/1-2011-0104 A felsőfokú informatikai oktatás minőségének fejlesztése, modernizációja program keretében történt, a szerzők köszönik a jegyzet elkészítéséhez nyújtott támogatást.

1. A LabVIEW programozási környezet

A LabVIEW egy olyan grafikus programozási környezet és programozási nyelv, melyet bárki gyorsan és könnyen megtanulhat. Mérnöki és tudományos feladatokra van optimalizálva: széleskörű beépített analíziskönyvtárral rendelkezik, segíti az adatok megfelelő vizualizációját, számos műszer és eszköz vezérelhető segítségével és rendkívül gyorsan fejleszthetők vele színvonalas alkalmazások.

A LabVIEW egy grafikus nyelv (ezért „G”-nyelvként is hivatkoznak rá), amely adatfolyam vezérelt. Utóbbi azt jelenti, hogy az egyes műveletek végrehajtását nem azok helyzete határozza meg a kódban, hanem hogy mikor áll rendelkezésre a végrehajtáshoz szükséges összes adat.

A LabVIEW programozási környezetet a National Instruments fejleszti. A jegyzetben a 2012-es verziónak megfelelő funkciókat írjuk le.

A LabVIEW legfontosabb előnyei:

- Könnyű megtanulni és használni. Bárki megtanulhatja, nem szükségesek programozási alapismeretek. Elsősorban tudományos és mérnöki felhasználásra van optimalizálva, egyszerűen vizualizálhatók segítségével a mérési eredmények, valamint a vezérlési szerkezetek.
- Gyors fejlesztést tesz lehetővé, ezáltal nő a produktivitás valamint csökkenthetők a költségek.
- Teljes funkcionalitást biztosít, számos beépített matematikai analízis funkció valamint jelfeldolgozási könyvtár áll rendelkezésre. Különböző ipari vezérlési és szabályozási valamint képfeldolgozási algoritmusok állnak rendelkezésre. Támogatja a többszálú végrehajtást, az eseményvezérlést, az objektumorientált programozást. Számos platform programozható egyetlen egy nyelven keresztül (számítógépek, beágyazott rendszerek, valós idejű rendszerek, FPGA-k, mikrovezérlők
- Ipari szabvány, rengeteg kompatibilis hardver, mérő és adatgyűjtő műszer áll rendelkezésre, széleskörűen alkalmazzák automatizált tesztrendszerekben.
- Tipikus felhasználások:
    - Mérés, adatgyűjtés, adatok elemzése, megjelenítése
    - Ipari vezérlés
    - Egyedi rendszerek, prototípusok fejlesztése
    - Komplex tudományos mérőrendszerek vezérlése (*Big Physics*)
    - Oktatás

Bevezetés a programozási környezetbe

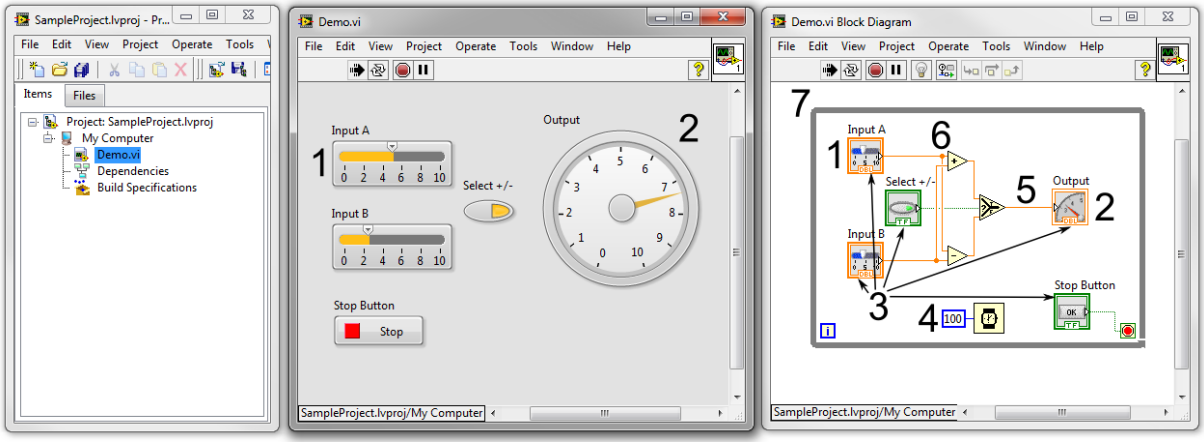
A LabVIEW indulásakor megjelenő induló képernyőn kezdhetünk neki egy új projektnek, vagy egy egyszerű program létrehozásának, de akár megnyithatjuk a már létező projekteket vagy programokat is.

A LabVIEW-ban a programokat *Virtual instrumentnek* hívják (*VI*, virtuális műszer), utalva ezzel a programnyelv eredetére, amikor is elsősorban műszereket valósítottak meg a segítségével, a műszerfunkciók jelentős részét maga a szoftver végezte. Egy *VI* három részből áll: a *Front Panel*, a *Block Diagram* valamint az Icon/Connector Pane (ikon/csatlakozó mező), utóbbinak az alprogramok (*subVI*) létrehozásánál van fontos szerepe (lásd később).

A *Front panel* (a program előlapja) az a része a programnak, amit a felhasználók látnak. Ezen foglalnak helyet a beviteli mezők (*Control*), amelyek adatokat fogadnak a felhasználótól (pl.

nyomógombok, potenciométerek), és a kimeneti mezők (*Indicator*), amelyek a felhasználót tájékoztatják (pl. kijelzők, grafikonok).

A *Block diagram* határozza meg a program által végrehajtott műveleteket. Minden egyes előlapi elemnek a *Block diagram*on megfelel egy *Terminal*. Az adatok áramlását a diagramon lévő vezetékek (*Wire*) határozzák meg, az elvégzendő műveleteket pedig a csomópontok (*Node*). Ez teljesen megfelel egy valódi műszer felépítésének, ahol az előlapi kezelőszervek és kijelzők a belső áramkörökkel vannak összeköttetésben, ahol a megfelelő funkciókat vezetékekkel összekötött elektronikai elemek végzik, melyek működése természetesen matematikai összefüggésekkel írható le. A LabVIEW-ban az utasítások végrehajtásának alapelve az adatfolyam alapú programozás, mely lényege, hogy egy csomópont végrehajtása akkor következik be, ha az összes bemeneti adata rendelkezésre áll. A blokkdiagramon a végrehajtási sorrendet a csomópontokat körbefoglaló programvezérlési struktúrákkal lehet módosítani (pl. elágazás, ciklus, szekvencia).

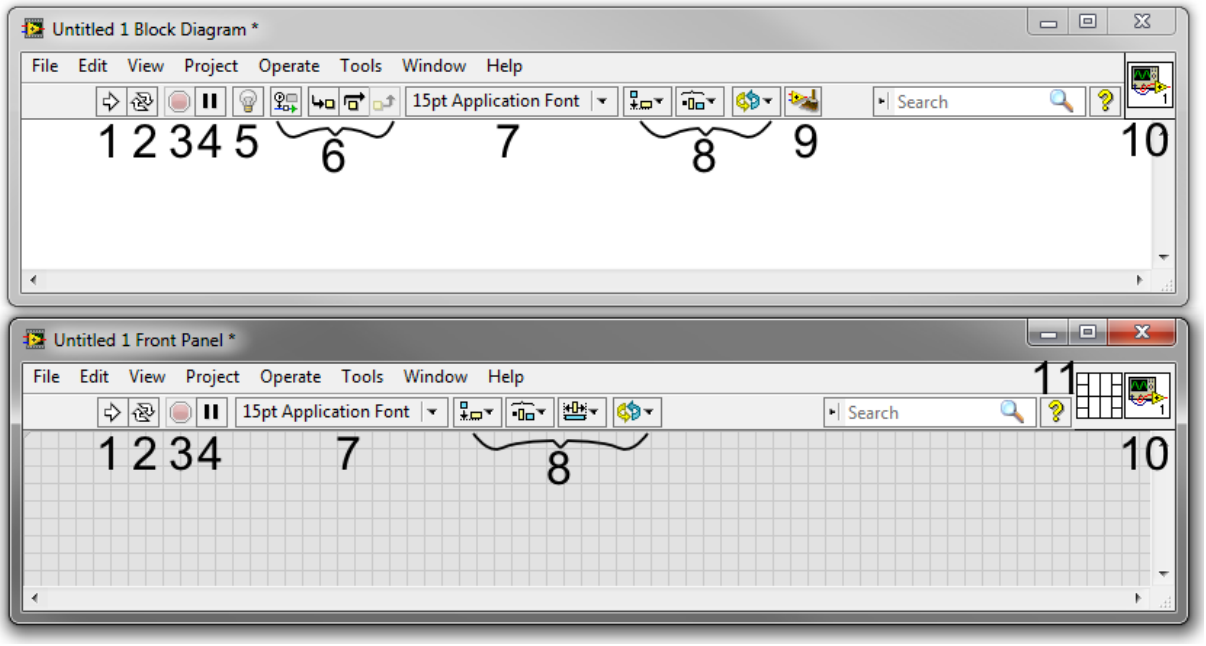


1.1. ábra: A projekt ablak, a program előlapja valamint a program blokk diagramja. A Front panelen található beviteli mezők: Control (1) és Indicator (2), melyek megfelelői a Block Diagramon is láthatóak, ezeket összefoglalva Terminalnak (3) nevezzük. Itt található még Constant (4), Wire (5), Node (6) és egy While structure (7).

A fenti programnak megfelelő C-kód (a bemenetek és kimenetek kezelése nélkül):

```
do {
   Output = Select ? InputA + InputB : InputA - InputB;
   waitms(100);
} while (!stop)
```

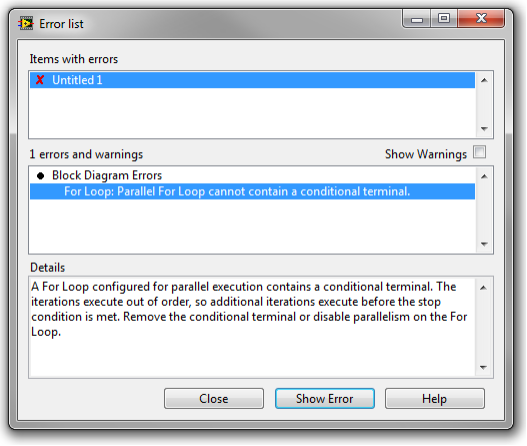
A projekt a más programozási környezetek esetén megszokott módon lehetővé teszi az állományok rendezését egy virtuális könyvtárstruktúra segítségével.. Bizonyos mérési és kommunikációs feladatok pedig csak akkor hajthatók végre, ha létrehozunk egy projektet. Az összetartozó funkciójú *VI*-okból llb kiterjesztésű könyvtár készíthető. Nagyobb programok, projektek esetén különös figyelmet kell fordítani a megfelelő modularitásra, a megfelelő tervezési minták követésére, különben a kódok átláthatatlanok lesznek, és nehéz lesz őket karbantartani.



1.2. ábra: Menüsor és eszköztár

Mind minden programozási környezet esetén a LabVIEW-ban is menüsor és eszköztár segíti a szerkesztést. Az előlap és a blokk diagram eszköztára némileg eltérő. A legfontosabb gombok:

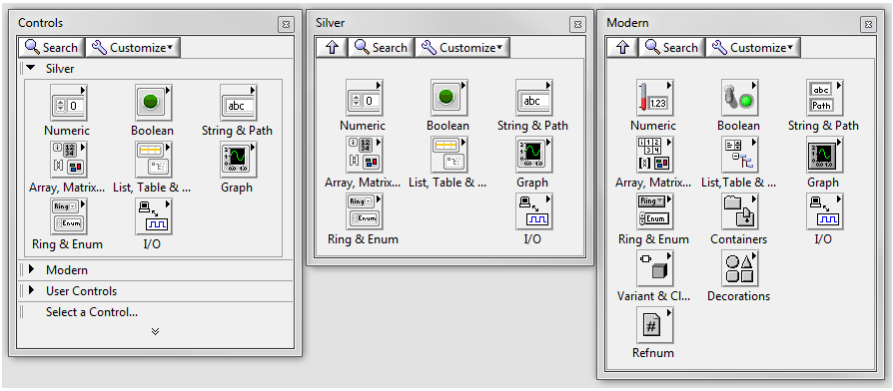
1) *Run* – egyszeri futás. Ha a program nem futtatható, a diagram hiányos, vagy hibás, akkor a nyíl egy törött nyíllá változik (*List Errors*). Rákattintva megjelenik az *Error list* nevű ablak, mely felsorolja a hibákat. Egy hibára kattintva részletesebb leírás jelenik meg a hibáról, mely segítheti a hiba kijavítását, duplán rákattintva pedig megmutatja a hiba helyét.
2) *Run Continuously* – folyamatos futás, a program befejezésekor újrakezdi a futtatást.
3) *Abort Execution* – program futásának megszakítása (általában nem ajánlott ennek a gombnak a használata, különösen, ha külső eszköz vezérlését végzi a program)
4) *Pause* – végrehajtás ideiglenes felfüggesztése, elsősorban diagnosztikai célokra, hibakeresésre használjuk
5) *Highlight Execution* – kiemeli az adatok áramlását a programban, hibakeresésre használhatjuk
6) További hibakeresést segítő eszközök (pl. *probe*, változók tartalmának megjelenítése futás közben)
7) Formázást segítő eszközök
8) Elemek rendezését segítő eszközök
9) *Cleanup Diagram/Selection* – a teljes diagramot vagy a kijelölt részt rendezi
10) A *Connector Pane* valamint a VI ikonja



1.3. ábra: A hibák listáját és magyarázatát tartalmazó ablak

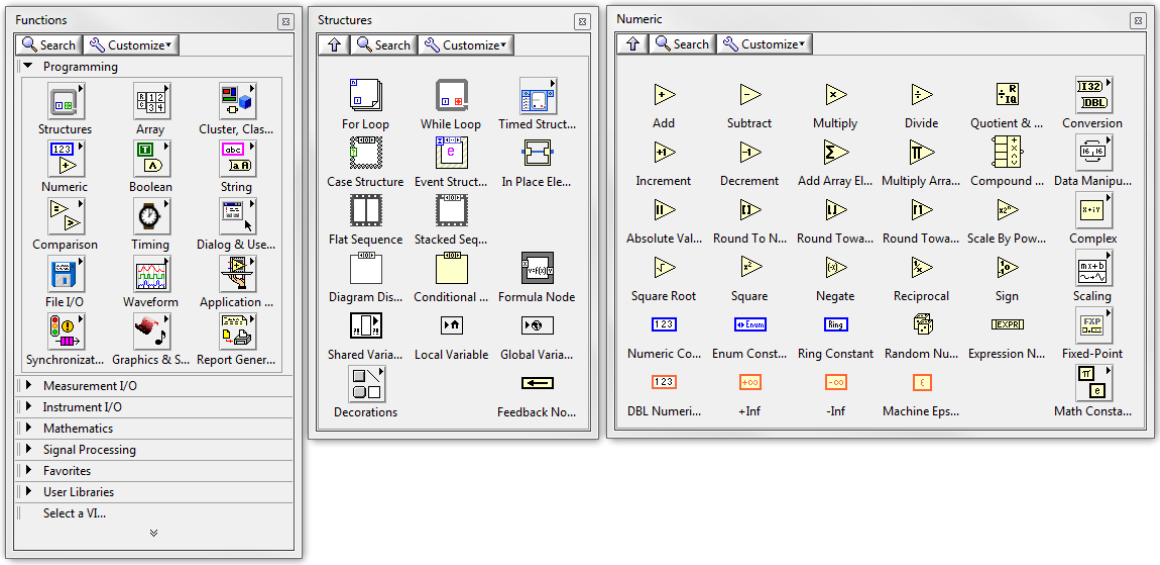
A *Front panel*-beli elemeket a *Control* palettáról választhatjuk ki és helyezetjük el az előlapon. (A programot csak akkor szerkeszthetjük, hogy ha az nem fut.) A paletta csak az előlapon érhető el, ha nem látható, akkor jobb gombbal üres helyre kattintva megjelenik. A palettát bármikor rögzíthetjük a gombostű megnyomásával.

Az előlapi elemek típus szerint csoportosított palettákba rendezve érhetők el. Egyes előlapi elemek többféle stílusban is megtalálhatók (*Modern, Silver, Classic, System*). Ha valamelyik alpalettát nem látjuk, a lefelé mutató két nyíllal megjeleníthetjük őket. A *Change visible palettes…* menüvel kiválaszthatjuk, hogy mely palettákat szeretnénk alapértelmezésben látni. Hasonlóan át is rendezhetjük a palettát.



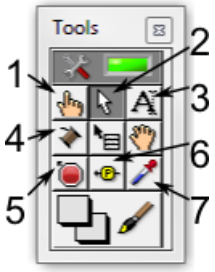
1.4. ábra: A *Control* paletta

A diagramon elhelyezhető elemek a *Functions* palettán érhetők el. Ha tudjuk egy *node* vagy *subVI* nevét, a *Search* segíthet megtalálni az adott eszközt (a keresőablak a CTRL+szóköz billentyűkombinációval is előhívható). Ha nem tudjuk biztosan, hogy mit keresünk, a palettákon való böngészés segíthet.



1.5. ábra: *Functions* paletták

Az egérkurzor által végrehajtható funkciók függenek az aktuális pozíciótól (néha egy-egy pixelen múlik az aktuális funkció). A végrehajtható funkciók a *Tools* paletta segítségével manuálisan is kiválaszthatók (ha ki van kapcsolva az *Automatic Tools Selection*, akkor a tabulátor vagy a szóköz lenyomásával is válthatunk az egyes eszközök között).

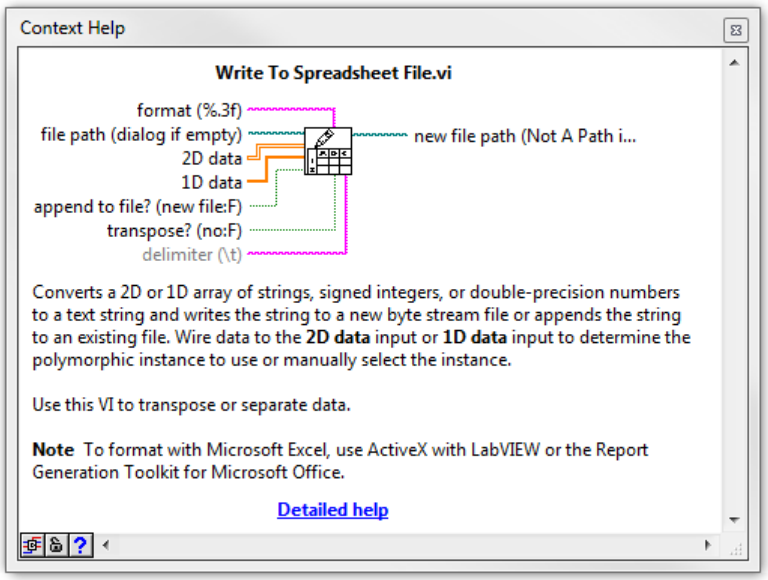


1.6. ábra: *Tools* paletta

A kurzor legfontosabb funkciói:

1) adatok módosítása, manipulálása (*Operate Value*)
2) kijelölés és szerkesztés (*Position/Size/Select*)
3) szöveg módosítása (*Edit Text*)
4) vezetékezés (*Connect wire*)
5) *breakpoint* beszúrása, a program végrehajtásának felfüggesztéséhez egy adott ponton
6) *Probe* eszköz, egy vezetéken levő adat értékének figyeléséhez
7) Színminta vételezése és átszínezés

Szerkesztés közben az egyik leghasznosabb segédeszköz a sugó *Context help* funkciója. Ez menet közben egy kis ablakban hasznos információkat közöl a kurzor alatt lévő előlapi elemekről, a diagrambeli csomópontokról, VI-okról és magukról a vezetékekről is. A *Context help* elérhető a menüből, a toolbar kérdőjel alakú gombját megnyomva valamint a CTRL+H billentyűkombináció segítségével. Az ablak alján van három gomb: az első megnyomásával kicsit részletesebb lesz a súgó, a másodikkal rögzíthetjük a kijelzett elemet, a harmadikkal pedig megnézhetjük az adott elem részletes súgóját. Ha a saját programjainkat/alprogramjainkat megfelelően dokumentáljuk, akkor azok leírása is megjelenik a *Context Help*-ben.



1.7. ábra: Példa a *Context help* tartalmára

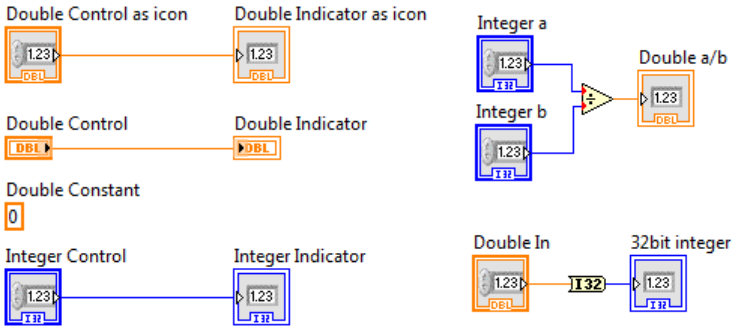
Adattípusok, adatfolyam programozás

Az adatfolyam programozás során az adatok tárolása az adatvezetékhez és az előlapi elemekhez kapcsolódik. A vezeték adattípusát a kimeneti változó adattípusa határozza meg. A bementi változó adattípusának általában meg kell egyezni a vezeték adattípusával. Az adattípus konverzióra a LabVIEW számos függvénnyel rendelkezik. Bizonyos esetekben automatikus konverzió történhet, ezt a bemeneti csatlakozón egy piros pont jelzi. Az automatikus típuskonverziók néha hibához vezethetnek vagy hibát jelezhetnek, ezért célszerű az automatikus konverziót explicitté tenni. A terminálok és a vezetékek színe utal az aktuálisan használt adattípusra, a *Context help* pedig részletesebb leírást is ad a típusról.

Numerikus adattípusok

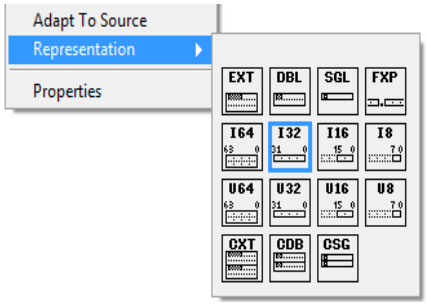
A LabVIEW-ban előforduló szám típusú adattípusok (*Numeric*):

- Egész számok (kék)
  számolás közben túlcsordulhatnak
  - előjel nélküli számok 8, 16, 32 és 64 bitesek
  - előjeles számok, 8, 16, 32 és 64 bitesek
- Lebegőpontos számok (narancssárga)
  szimpla pontosságú, dupla pontosságú, kiterjesztett pontosságú
- Komplex számok (narancssárga, a lebegőpontos számoknak megfelelő választható pontossággal)
- Fixpontos számok esetén külön meg lehet határozni a szám teljes méretét bit-ben valamint, hogy az egész rész hány bitet foglalhat el. Elsősorban korlátozott erőforrások használata esetén (pl. FPGA) van jelentősége.



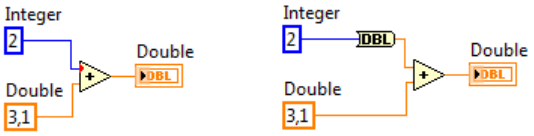
1.8. ábra: Numerikus terminálok normál és ikonnézetben. Automatikus és explicit típuskonverzió.

Egy numerikus *Control* vagy *Indicator* esetén egyszerűen tudunk változtatni a szám típusán (reprezentációján), a helyi menüben (az elemre a jobb egérgombbal kattintva jön elő) a *Representation* menüelemet választva. Egy *Control* vagy *Indicator* további tulajdonságait a *Properties* menüelem segítségével tudjuk módosítani



1.9. ábra: Adattípus (reprezentáció) megváltoztatása

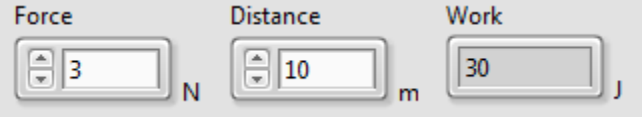
A számokkal végzett alapvető műveletek a *Programming/Numeric* palettán találhatók. További műveletek (szögfüggvények, exponenciális függvények) a *Mathematics* palettán találhatók.



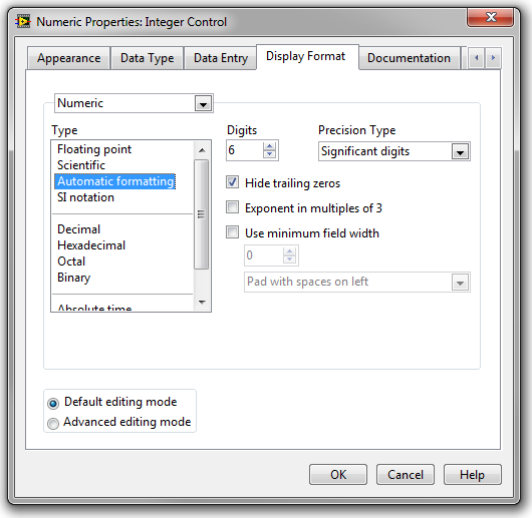
1.10. ábra: Balra: automatikus típuskonverzió az őt jelölő piros ponttal. Jobbra pedig explicit típuskonverzió.

A numerikus típusok között a többi programozási nyelvhez hasonlóan létezik automatikus típuskonverzió. A LabVIEW grafikusan mutatja, ha egy adaton konverzió történik, egy kis piros pont jelenik meg a vezeték végén, ami egy műveletvégző rész bemenetére csatlakozik. Az automatikus típuskonverziót jelölő pontra figyelni kell, az esetek egy részében ugyanis hibás lehet a típuskonverzió eredménye, másik részében pedig programozási hibára is felhívhatja a figyelmet.

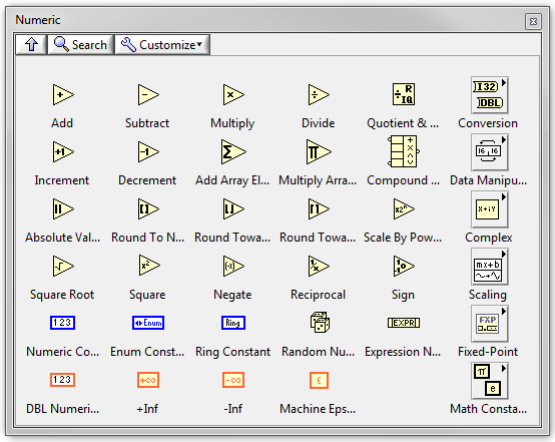
Numerikus adatokhoz mértékegységet is rendelhetünk, így a program jobban megfelelhet a valós jelek, mennyiségek kezelésének. Ez biztosítja, hogy például csak olyan mennyiségeket összegezhetünk, melyek azonos mértékegységgel rendelkeznek.



1.11. ábra: Mértékegységek megadása. A LabVIEW már a program írása közben ellenőrzi, hogy helyesek-e a mértékegységek.



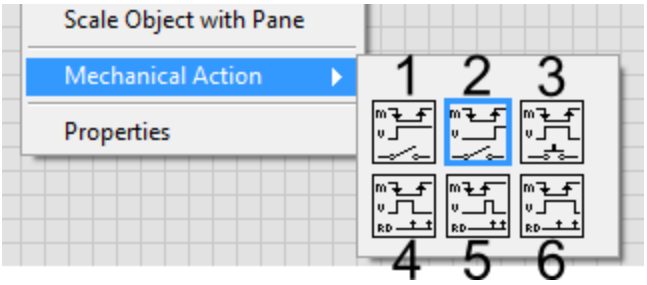
1.12. ábra: Egy *Control* vagy *Inticator* által megjelenített szám formázása



1.13. ábra: A numerikus műveleteket tartalmazó paletta. A műveletek döntő többsége *polimorf*, vagyis a bemenetektől függ hogy milyen adattípusokon dolgozik és milyen adattípus lesz az eredmény.

Boolean típusok

Egy boolean változó igaz (true) és hamis (false, alapértelmezett érték) értéket vehet fel. A boolean elemek zöld színűek. Az előlapon az ilyen változók tipikusan LED-ként, kapcsolóként vagy nyomógombként jelennek meg. Egy *Boolean control*nak különböző üzemmódjai lehetnek.

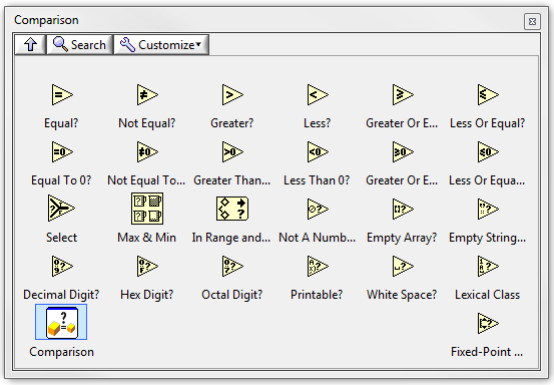


1.14. ábra: Egy *Boolean Control* lehetséges üzemmódjai (mechanical action)

Lehetséges üzemmódok:

1) *Switch when pressed* – a gombot átkapcsoljuk egyik állapotból a másikba (egérgomb lenyomásakor)
2) *Switch when released* – a gombot átkapcsoljuk egyik állapotból a másikba (egérgomb felengedésekor, egész addig a felhasználó meggondolhatja magát és elhúzhatja az egeret). Az előbbi két módot tipikusan billenőkapcsolóként használjuk.
3) *Switch until released* – csak addig változik meg az állapota, amíg lenyomjuk. Tipikus nyomógomb. Ha a lenyomás ideje alatt a program nem kérdezi le a nyomógomb állapotát, a gombnyomás elvész.
4) *Latch when pressed* – lenyomás után tárolja az állapotát egész addig, míg a program ki nem olvassa az értékét, utána visszatér eredeti állapotába
5) *Latch when released* – az egérgomb felengedésekor változtatja meg az állapotát, és addig tárolja, míg a program ki nem olvassa az értékét. Ez az alapértelmezett nyomógombtípus pl. Stop gombok implementálására, valamint minden olyan esetben, amikor valami egyszer végrehajtandó utasítást akarunk kiadni a program számára.
6) *Latch until released* – addig változik meg az állapota, amíg lenyomjuk, viszont biztosítva van, hogy felengedés után még egyszer ki lesz olvasva az állapota. A program így biztosan észreveszi a gomb lenyomását.

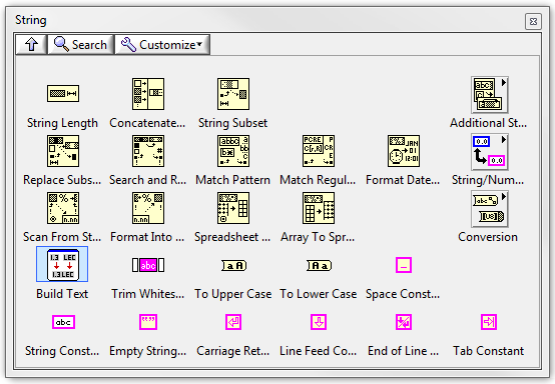
Programozás közben boolean értékek legtöbbször az összehasonlítások eredményeként keletkeznek. Összehasonlítani nem csak számokat tudunk, hanem szinte bármilyen adattípust.



1.15. ábra: Összehasonlításokat tartalmazó paletta

String és path

A string típusú változó nyolc bites karakterek sorából áll. A LabVIEW-ban egyrészt szöveges információk tárolására használjuk, másrészt pedig bináris adatfolyamok (stream) esetén. A string elemek rózsaszín színűek.

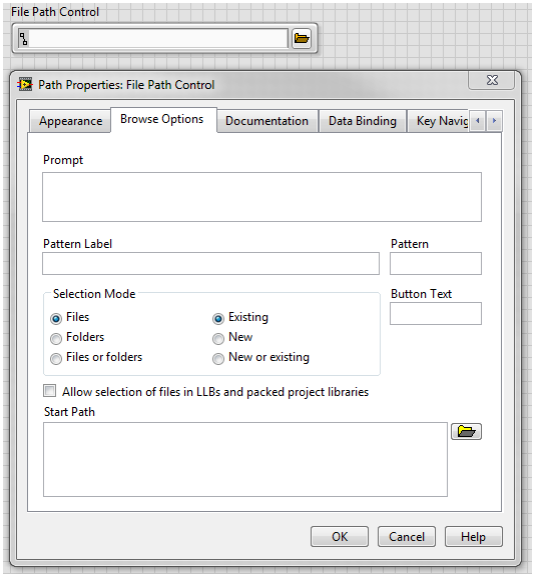


1.16. ábra: String műveletek

A legfontosabb string műveletek:

- *String Length* – hossz lekérdezése
- *Concatenate Strings* – string-ek egyesítése
- *String Subset* – string egy részének kiolvasása
- *String/Number conversion* – szám sztring-é formázása és szöveg számmá alakítása

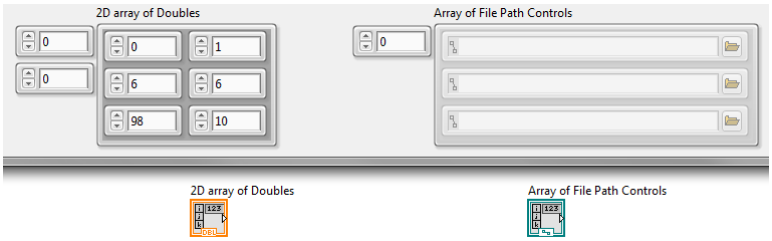
A *File Path* szolgál arra, hogy egy-egy fájlra platformfüggetlen módon hivatkozhassunk. Egy *File Path Control* esetén megválaszthatjuk, hogy milyen fájlokat és/vagy könyvtárakat választhatunk ki a program számára.



1.17. ábra: *File Path Control* és beállításai

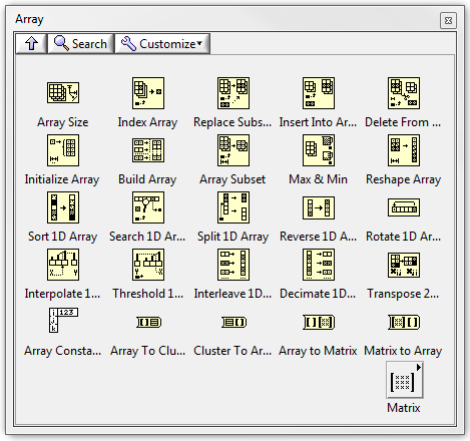
Tömbök

A tömbök olyan struktúrák, amelyek azonos típusú elemeket tartalmazhatnak, úgy, hogy azok sorrendje meghatározott. LabVIEW-ban a tömbön kívül bármilyen adatstruktúrát tartalmazhatnak. A LabVIEW-ban tetszőleges dimenziószámú tömböt definiálhatunk. Tömb *Control*-t vagy *Indicator*-t úgy hozhatunk létre, hogy az előlapon lerakjuk a tömb keretét (*Array Shell*), majd ebbe belehúzzuk a kívánt típusú elemet.



1.18. ábra: Minták tömbökre. A bal oldali egy 2D tömb, ami számokból áll, a jobb oldali pedig egy 1D tömb, ami fájlútvonalakat tartalmaz. Mindkét tömb bal felső sarkában található az *index display*, mely segítségével navigálhatunk az elemek között. Ha itt nulla szerepel, akkor a nulladik elemtől kezdve jelennek meg az elemek, ha 10 szerepel, akkor a 10 indexű elem van legfelül kijelezve.

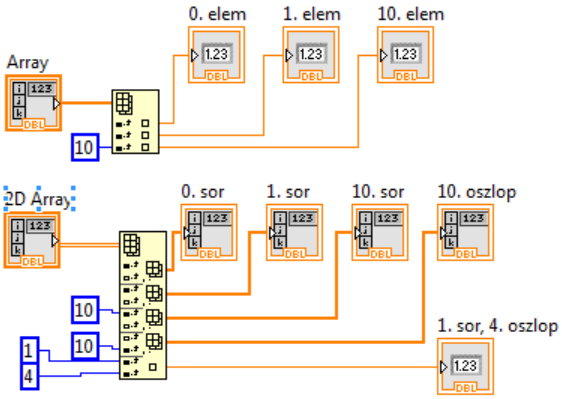
Az egyszerre megjelenő elemek számát, valamint a tömb dimenziószámát egyszerűen tudjuk módosítani a szerkesztéskor megjelenő kék pöttyök „húzásával". Az *Index Display* segítségével választhatjuk ki az elsőre megjelenő elem sorszámát. A tömbben elhelyezett első elem indexe nulla.



1.19. ábra: Tömbműveleteket tartalmazó paletta

A legfontosabb tömbműveletek:

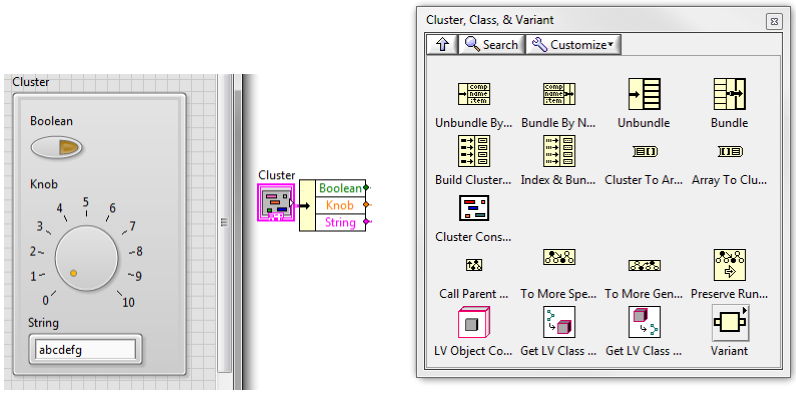
- *Array Size* – Tömb méretének lekérdezése. 1D tömbök esetén ez egy skalár, más esetben egy tömböt kapunk vissza, amely tartalmazza a tömb egyes dimenzióiban való méretét.
- *Index Array* – a tömb egy elemének, esetleg sorának, oszlopának kiolvasása
- *Delete From Array* – a tömb egy elemének vagy tartományának törlése
- *Initialize Array* – megadott méretű, megadott egyforma elemekből álló tömb létrehozása
- *Build Array* – egyéni elemekből való tömb építése vagy tömbök egymás után fűzése (*Concatenate Inputs*), vagy magasabb dimenziós tömbök építése alacsonyabb dimenziószámú tömbökből (az egyes opciók közül a helyi menüből választhatjuk ki a számunkra megfelelőt)
- *Array Subset* – a tömb egy tartományának kiolvasása



1.20. ábra: Példa a tömb elemeinek/sorainak/oszlopainak kiolvasására

Clusterek

A *Cluster*ek egy összetett adatstruktúra, egy a felhasználó által létrehozott adatstruktúra, melyek különböző típusú elemeket tartalmazhatnak. Mindegyik elemhez hozzá van rendelve egy címke, valamint az elemek sorrendje is rögzített. A *Cluster* megfelel a C struktúra típusának. Egy *Clustert* úgy hozhatunk létre, hogy lerakjuk a keretét, majd pedig egymás után belehúzzuk a kívánt elemeket Ha ugyanazt a struktúrát a programban többször használni akarjuk, célszerű létrehozni belőle egy egyedi kontrolt. Ezt a klaszter kiválasztása után az előlapi *Edit/custumize* control menű segítségével tehetjük meg. Válasszuk a *typedef* vagy *a strickt typedef* opticót, és mentsük el az újonnan készített controlt a projekt állományai közé. Ha a későbbiekben módosítjuk a típusdefiníciót az összes példány frissülni fog. A szigorú típusdefiníció „*Strict typedef*” esetén a kontrolok kinézete is meg lesz határozva



1.21. ábra: *Cluster* használata és a *Cluster* paletta

A LabVIEW-ban egy speciális struktúra az *Error Cluster*, melyet a hibák kezelésére használnak. A hibaklaszter *status* logikai értéke jelzi a hiba bekövetkezését. A hiba *code* numerikus értéke a beépített függvények esetén a hibát azonosító kód. A *source* string a hiba forrását jelöli, azt a *VI*-t. ahol a hiba keletkezett. A hibakezelést biztosító *VI*-ok egy be és kimeneti hiba klasztert tartalmaznak. Az egymást követő *VI*-ok hibavezetékeinek összekötésével biztosítható a hibakezelés. *Error Cluster* esetén az *Explain Warning/Error* opció segítheti a felhasználót vagy programozót megfejteni a hiba okát.

Adatfolyam alapú programozás

A LabVIEW-ban az egyes utasítások végrehajtása nem attól függ, hogy azok egymáshoz képest milyen sorrendben (egymás alatt vagy egymás mellett) helyezkednek el, hanem attól,

hogy az adatok hogyan „áramlanak” a program végrehajtása során. Ezt adatfolyam alapú programozásnak hívjuk (*Data Flow Programming*). A következő két szabály határozza meg a végrehajtás sorrendjét:

- egy csomópont akkor hajtódik végre, amikor a bemenetén az összes adat a rendelkezésére áll
- a csomópont kimenetén akkor jelennek meg az adatok, amikor az befejezte az összes feladatát

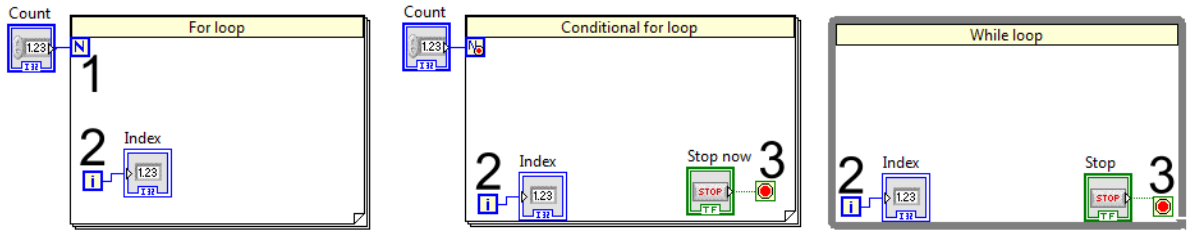
A LabVIEW támogatja a párhuzamos végrehajtást. Amennyiben a fenti két szabály alapján nem határozható meg, hogy mely csomópont hajtódik végre először és melyik másodjára, akkor azok akár párhuzamosan is végrehajtódhatnak (több számítógépmag esetén ez fizikailag is megtörténhet). Az, hogy párhuzamosan végrehajtható műveletek közül melyik fog először megvalósulni, nem határozható meg előre, ráadásul futásról-futásra is változhat. Ha a futás sorrendje fontos, akkor azt megfelelő vezetékezéssel vagy megfelelő struktúrákkal (*Sequence structure*, lásd később) oldhatjuk meg. Ha egy változó értékét több párhuzamosan futó programrészlet is írhatja és olvashatja, a futtatás eredménye a végrehajtás sorrendjétől függhet. Ekkor a párhuzamos programozás versenyhelyzet (*Race condition*) kialakulásához is vezethet, ezekre érdemes figyelni programozás közben. Megfelelő erőforrás lefoglalási módszerekkel a versenyhelyzetek megszüntethetők.

Bár a program működését az elemek és vezetékek elrendezése nem befolyásolja, a program olvashatóságára azonban hatással vannak. A program írása közben a következő szabályokat érdemes betartani:

- Az adatok a vezetékeken mindig balról jobbra haladjanak
- Az egyes programblokkokat is úgy érdemes elhelyezni, hogy az adatok balról jobbra áramoljanak
- Ha a fenti szabály nem valósítható meg, akkor az adatok lehetőleg fentről lefelé haladjanak
- Lehetőség szerint a teljes program férjen el egy képernyőn. Ha ez nem lehetséges, akkor lehetőleg csak egy görgetősávot kelljen használni
- A programunkat megfelelően kommenteljük, hogy később is meg lehessen érteni a működését
- A program strukturálását, a áttekinthetőségét jelentősen növeli a subVI-ok alkalmazása.
- Hasonló szerepe van egyedi adatvezeték klaszterek, típusokba rendezésének.

Programozási struktúrák

A LabVIEW-ban számos olyan programozási struktúra elérhető, mely a hagyományos programnyelvekbe is megtalálható. E mellett van számos olyan struktúra, mely a grafikus programozáshoz köthető.



1.22. ábra: *For* ciklus és *While* ciklus

Ha az utasítások egy csoportját többször szeretnénk végrehajtani, akkor ciklusokat használunk. A LabVIEW-ban két ciklus is rendelkezésre áll, a *For loop* és a *While loop*. A diagramon való elhelyezéshez kiválaszthatjuk a kívánt ciklust a palettán, majd pedig körberajzoljuk vele a diagramunknak egy részét. A körberajzolt elemek automatikusan belekerülnek a ciklusba. A ciklusban lévő diagramrészt nem csak egyszer hajtódik végre, hanem többször, a megadott feltételeknek megfelelően. A fő különbség a két ciklus között:

- A *For* ciklus az N terminálba bekötött számszor hajtódik végre, akár nullaszor is.
- A *While* ciklus a megadott feltétel teljesüléségi hajtódik végre, akár végtelenszer, de mindenképp legalább egyszer (a feltétel kiértékelése a ciklus végén történik)

A ciklusokban előforduló elemek:

1) *N* – ismétlések száma (*For* ciklus esetén)
2) *i* – iteráció sorszáma (nulláról indul, *For* ciklus esetén *N*-1-ig megy el)
3) Feltétel vizsgáló terminál (*Condition terminal*) – elsősorban *While loop* esetén használjuk, a ciklus végrehajtásának befejezését határozhatjuk meg. Két opció áll rendelkezésre melyet a helyi menüből választhatunk ki: a *Stop if true* valamint a *Continue if true*. A terminálba *Error Cluster*-t is beköthetünk, ekkor a ciklus akkor áll le, amikor hiba keletkezett.

A fenti ciklusok a következő *C* kódnak felelnek meg:

- *For loop*

```
int i;
int N = count;
for (i = 0; i < N; i++) {
   // ciklusmag
}
```

- *Conditional for loop*

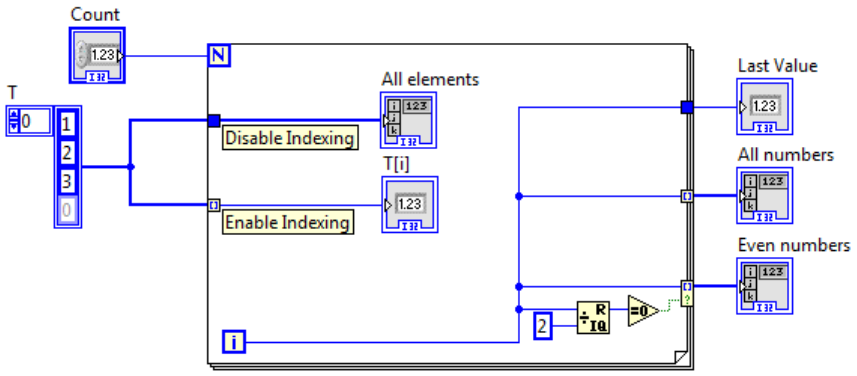
```
int i;
int N = count;
int stop = 0;
for (i = 0 ; i < N; i++) {
   //ciklusmag
   if (stop) {
          break;
   }
}
```

- *While loop*

```
int i = 0;
int stop = 0;
do {
```

```
    //Ciklusmag
    i++;
} while (!stop);
```

Amikor egy vezetéket bevezetünk egy struktúrába vagy kivezetünk belőle, akkor ott egy kis négyzet jelenik meg (*Tunel*). A struktúra típusától függően ennek az elemnek különböző lehetőségei vannak, ezt általában a helyi menüben tudjuk kiválasztani. Ciklusok esetén a tömbökkel való interakció a legfontosabb opció. Lehetőség van arra, hogy a bemenetként felhasznált tömböt automatikusan kiindexeljünk, a ciklus minden egyes iterációja során egy-egy újabb elemet kiolvasva (ez megfelel más programozási nyelvek foreach utasításának). For ciklus esetén ilyenkor a *Loop Count* változó bekötése felesleges, a ciklus annyiszor fog végrehajtódni amekkora a tömb mérete. Több, különböző méretű tömb esetén a legkisebbnek a mérete fog érvényesülni, kivéve, ha be van kötve a *Loop Count* és az ennél kisebb.

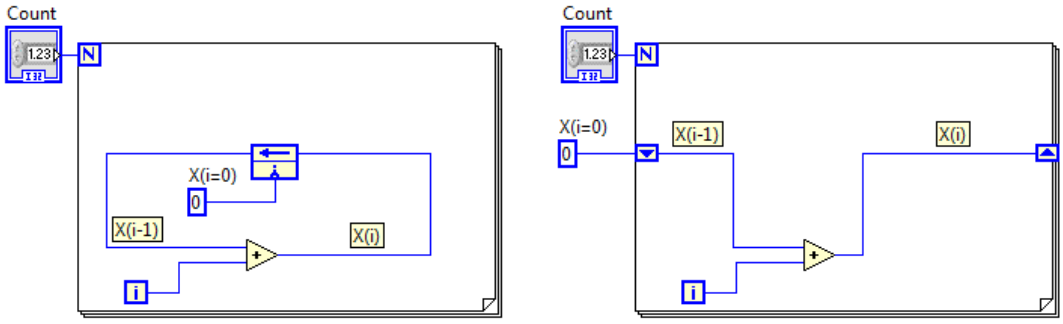


1.23. ábra: Automatikus indexelés. A ciklus legfeljebb háromszor fog lefutni a bekötött *T* tömb miatt.

Ha egy ciklusból kivezetünk egy skalárt, szintén több lehetőségünk van, ezt a *Tunel Mode* helyi menüből tudjuk kiválasztani: megkaphatjuk csak az utolsó értéket (*Last value*), az összes elemet egy egy dimenziós tömbben (*Indexing*). Utóbbi esetén egy feltételhez is köthetjük, hogy egy elem hozzá legyen-e adva a tömbhöz vagy nem (*Conditional*, *Indexing*). Ha egy tömböt vezetünk ki, akkor kivezethetjük az utolsó végrehajtás eredményét, építhetünk belőle magasabb dimenziós tömböt (*Indexing*), vagy megtartva a tömb dimenzióját egymás után fűzhetjük az eredményt (*Concatenating*). For ciklus esetén az alapértelmezett beállítás az indexelés (vagyis ha egy skalárt kivezetünk, abból egyből egy tömb lesz), *while* ciklus esetén pedig az utolsó értéket kapjuk meg.

Ciklusok esetén szükség lehet egy korábbi iteráció eredményének felhasználása egy későbbi iteráció során. Erre a *Feedback Node* vagy a *Shift Register* szolgál. A két megoldás lényegében ekvivalens, de az elkészített program jobban követhető, ha *Shift Registert* használunk.

A *Feedback Node* elérhető a *Structures* palettán, de ha egy hurkot készítünk a vezeték eszközzel, akkor is létrejön. A *Shift Register*t a ciklus helyi menüjében érhetjük el, ha a ciklus szélére kattintunk a *Create Shift Register* opcióval.



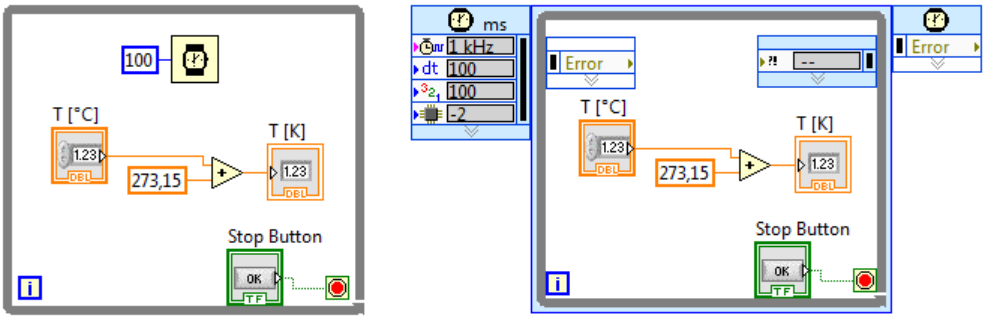
1.24. ábra: A *Feedback Node* és a *Shift Register*. A két kód pontosan ugyanazt hajtja végre.

A *Feedback Node*-nak megfelelő *C* kód:

```
int i;
int N = count;
int x = 0;
for (i = 0; i < N; i++) {
   x = x + i;
}
```

A legtöbb program nem csak egyszer hajt végre egy műveletet, hanem egy ciklusban addig, amíg le nem állítjuk. Erre tipikusan egy *while* ciklust használunk, mely leállításáról egy Stop gomb gondoskodik.

Ciklusok időzítését elsősorban két okból szoktuk megtenni. Az egyik ok, az erőforrásokkal, pl. processzoridővel való takarékosság, amikor egy műveletet, pl. egy bemenet állapotának figyelését, nem kell a lehető legnagyobb sebességgel végezni. Másik lehetőség, amikor egy mérési vagy vezérlési folyamatot ütemezni szeretnénk. A ciklusok időzítését több módon is megtehetjük az alábbi ábrának megfelelően.

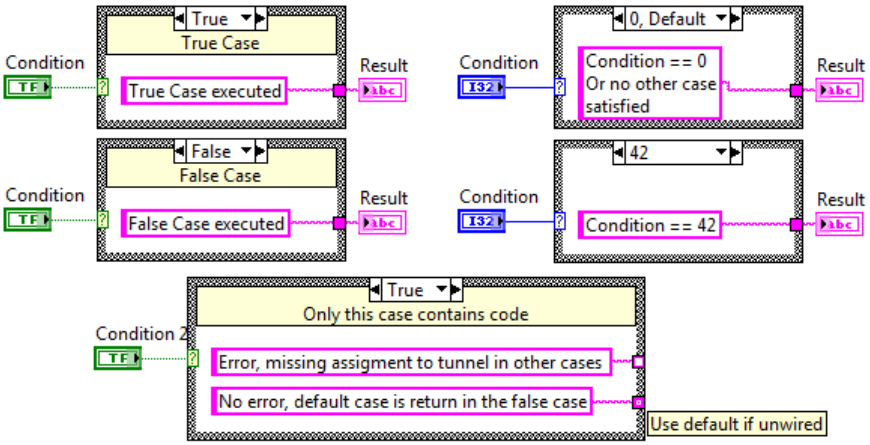


1.25. ábra: Várakozás megvalósítása *while* ciklus esetén. Bal oldalt a *Programming/Timing/Wait (ms)* node felhasználásával, jobb oldalt pedig a *Programming/Structures/Timed Structures/Timed Loop* felhasználásával.

Case struktúra

A program egy részének a feltételes végrehajtására a *Case Structure* szolgál. A struktúra több lapot is tartalmaz, az a diagramrész hajtódik végre, mely megfelel a feltétel kiértékelő terminálba bekötött értéknek. Ha egy boolean a bemenet, akkor két lehetséges eset van, a *True* és a *False*. Az egyes lapok között a fejlécre kattintva válthatunk. Ha egész számot kötünk be, akkor már sokszámú lehetőségűnk van. A *Default case* akkor hajtódik végre, amikor a program nem talál a bemeneti értéknek megfelelő lapot. Lebegőpontos számot bekötve az egész számmá konvertálódik. A *Case* struktúra string értékeket is elfogad, valamint *enumerate* típusú változókat. Utóbbi előnye, hogy a *case* struktúra ágainak elnevezése követi

a felsorolt eseteket, így jól áttekinthetővé válik az ágak feladata. További lehetséges bemenet *Error Cluster*, ekkor két lapunk lesz, az *Error* (piros) és a No *Error* (zöld).



1.26. ábra: Case struktúra

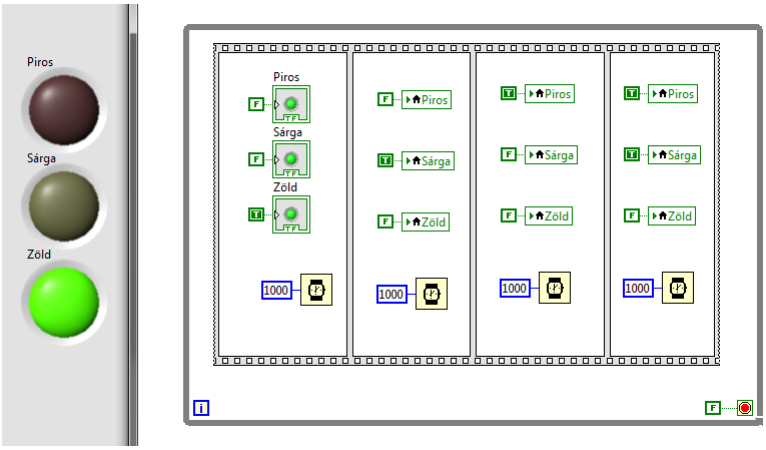
Egy értéket kivezetve a *Case* struktúrából ott egy *tunel* keletkezik. A kivezetést a *Case* struktúra összes lapján be kell kötni, különben hibás lesz a programunk. Ekkor a kivezetés nem teli négyzet, hanem üres négyzet. Ha nem kívánjuk az összes esetben megadni, a helyi menüből kiválaszthatjuk a *Use default if unwired* opciót.

Sorrendiséget biztosító struktúrák

Ha fontos a műveletek végrehajtásának sorrendje, de azt a megfelelő vezetékezéssel nem lehet biztosítani, akkor a legegyszerűbb választás *Flat Sequence Structure* vagy a *Stacked Sequence Structure*. Utóbbi használatát kerülni kell, mivel nehezen követhetővé teszi a programot. Mindkét esetben az egymás utáni *keret*ekben lévő diagramrészeket a LabVIEW egymás után hajtja végre.

Egy-egy kimenteti terminálhoz csak egy-egy adatforrást köthetünk hozzá. Ha a programunkban több helyen szeretnénk változtatni egy kimeneti mező értékét, akkor azt a *Local Variable* segítségével végezhetjük el. Érdekesség, hogy bemeneti mezők (*Control*) értékét is módosíthatjuk ezzel a módszerrel, valamint bármelyik mezőt nem csak írni, hanem olvasni is tudjuk. Lokális változót legegyszerűbben a terminál helyi menüjének *Create/Local Variable* opciójával tudunk létrehozni, de a *Programming/Structures* menüben is megtalálható. A lokális változókhoz hasonlóan globális változókat is használhatunk (*Global Variable*), ezeket tetszőleges *VI*-ból vagy *subVI*-ból elérhetjük. A globális változó egy speciális VI, mely csak *Front panelt* tartalmaz.

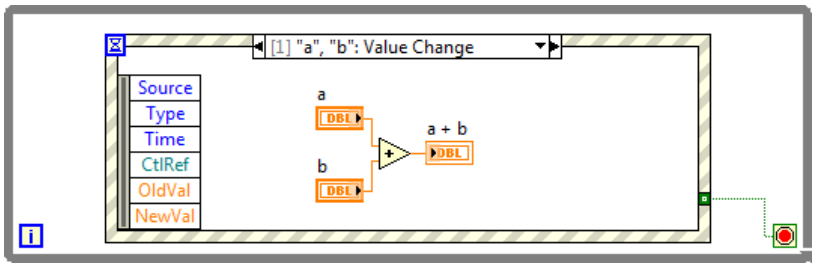
Az összes *Control* és *Indicator* olyan objektum, mely számos tulajdonsággal rendelkezik, amelyek a program futtatása során is módosíthatók a terminálhoz rendelt *Property Node* segítségével. Például letiltható egy kontrol működése az „*enabled*” állapot módosításával.



1.27. ábra: Egyszerű jelzőlámpa-alkalmazás, mely tartalmaz egy végtelen ciklust, *Flat Sequence Structure*-t, *Local Variable*-t valamint várakozást.

Ha egy struktúrát úgy törlünk, hogy kijelöljük, majd a delete gombot megnyomjuk, a struktúra tartalma is elvész. Ha meg akarjuk tartani a tartalmát, akkor a helyi menü *Remove Case Structure* opcióját választhatjuk (illetve a többi struktúra esetén a neki megfelelőt). Több lapot tartalmazó struktúrák esetén a LabVIEW figyelmeztet minket, hogy az épp nem látott lapokon lévő tartalom elvész.

Eseménykezelés: Event structure



1.28. ábra: Eseményvezérelt programozás az *Event structure* segítségével

Az előlapi elemek változását egy programból két módon detektálhatjuk. Az egyik módszer, amikor egy ciklusban rendszeresen beolvassuk az aktuális értékét és összehasonlítjuk a korábbival (pooling mode). A módszer folyamatosan használja a számítógép erőforrásait, ezért használata nem célszerű. A másik módszer, melyet más programozási nyelvek is támogatnak (pl. Java, C#), az eseményvezérelt programozás. Ekkor az operációs rendszer vagy a futtatókörnyezet figyeli a felhasználó tevékenységét, és amennyiben az egy megfelelő eseményt generál, pl. megnyom egy gombot, végrehajtja az eseményhez hozzárendelt kódot. LabVIEW-ban az *Event Structure* használható a funkció megvalósítására. A struktúra több *frame*-t is tartalmazhat, egy-egy *frame*-hez pedig több eseményt is hozzárendelhetünk. Ameddig egy figyelt esemény sem következik be, a struktúra erőforrások pazarlása nélkül várakozik. Amikor az esemény bekövetkezik, az adott *frame* tartalma végrehajtódik, a program pedig továbblép a struktúrából. Ha a kódot nem csak egyszer szeretnénk végrehajtani, hanem folyamatosan szeretnénk figyelni az eseményeket, az *Event Structure*-t egy while ciklusba kell beilleszteni.

SubVI – alprogramok

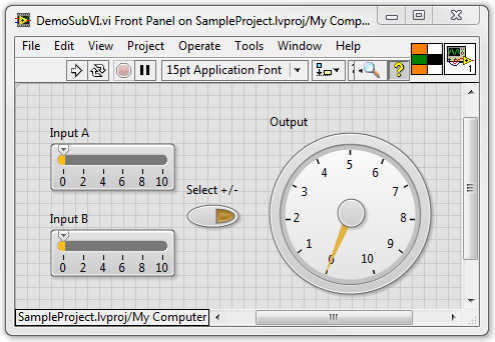
A SubVI-ok alapvető fontosságúak a LabVIEW programozásban. Segítségükkel oldható meg a megírt programrészek újrahasznosítása, valamint jelentősen egyszerűsíthetünk a programjaink kinézetén. Bármelyik VI-ból tudunk SubVI-t készíteni, mindössze néhány

dologra kell figyelni. Ezek után az alprogramunk ugyanúgy használható lesz, mint a Function Paletta többi eleme.

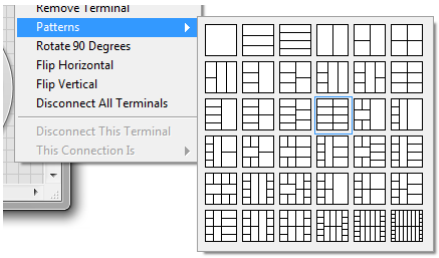
A front panel jobb felső sarkában két négyzetet találunk. A baloldali a terminálok bekötését szimbolizálja, a jobb oldali pedig a SubVI ikonját mutatja.

A terminálok (be és kimeneti változók) azok a pontok, ahova majd vezetéket tudunk húzni a SubVI-t felhasználó VI-ban. Ahhoz, hogy egy terminált hozzárendeljünk az előlapi elemhez, először kattintsunk az egérrel egy szabad terminálra, ezt követően pedig a bemenetre vagy kimenetre. Ekkor a terminál színe átváltozik az előlapi elem adattípusának megfelelően. Bár a LabVIEW nem jelez hibát, de a beviteli mezőket mindig a bal oldalra kössük be a kiviteli mezőket pedig a jobb oldalra.

Minden egyes bemeneti változó esetén három lehetőséget adhatunk meg a helyi menüből kiválasztva: *Optional*, *Recomended* és *Required*. Utóbbi esetben a *subVI* adott bemenetére kötelező bekötni valamilyen adatot, különben a *VI* nem lesz futtatható. Az *Optional*, *Recomended* bemenetek esetén a *subVI*-t felhasználó programozó dönthet úgy, hogy nem köt be semmilyen adatot, ekkor a *subVI* az adott bemenet alapértelmezett értékét fogja megkapni. Az alapértelmezett bármikor megváltoztathatjuk az aktuálisan kijelzett értékre, ha a helyi menüben a *Data Operations*/*Make Current Value Default* opciót választjuk.



1.29. ábra: Terminálok bekötése a Front Panelen. A terminálokat mutató négyzet fölé navigálva az egérrel, az egér huzalozó eszközzé változik. Ezek után beköthetjük a bemeneti és kimeneti mezőket a kiválasztott helyekre.



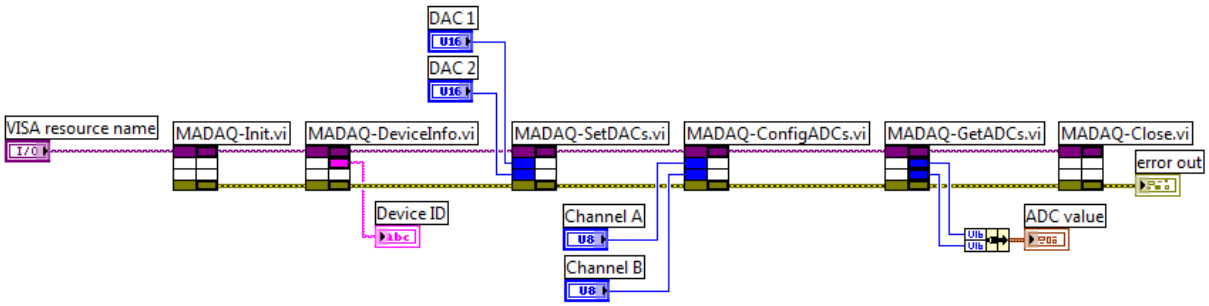
1.30. ábra: Terminálok elrendezésének kiválasztása

A terminálok lehetséges elrendezését a helyi menü *Patterns* opciójával tudjuk kiválasztani. Általában érdemes a 4+4-es elrendezést használni, még akkor is, hogy ha nem használjuk ki az összes terminál helyet, így az alprogramunk funkcionalitása könnyen bővíthető lesz. E mellett az egymás után következő SubVI-ok szépen illeszkedhetnek egymáshoz.

Ha egy felhasznált SubVI elrendezését módosítjuk, az őt felhasználó VI-ok nem fognak futni egész addig, míg újra nem linkeljük a SubVI-t (a helyi menü *Relink to SubVI* opcióját használva).

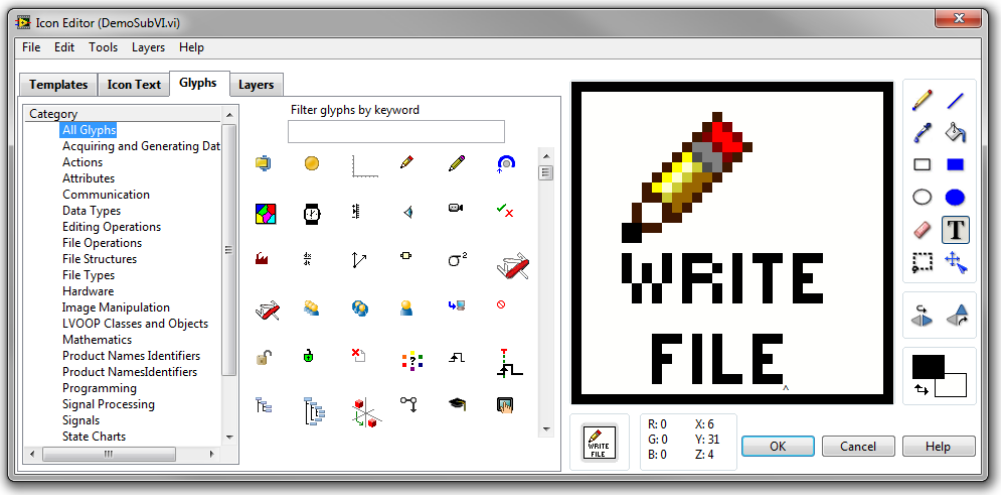
Az esetek nagy részében szükség van megfelelő hibakezelésre, illetve a sorrendiség megfelelő biztosítására. Ekkor az alsó két sarokban lévő terminálokat az *Error Cluster* továbbvezetésére

használjuk, a SubVI-okat pedig egymás után felfűzzük rá. Ez a tervezési mintázat rendszeresen előfordul, amikor egy külső erőforráshoz kell hozzáférni (pl. fájl írása, műszer kezelése). Ilyen esetekben a két felső saroknak is speciális szerepe van, az adott erőforrásra hivatkozó referencia van bevezetve, illetve kivezetve minden VI-ból.



1.31. ábra: Minta program egy műszer kezelése esetén. A felső lila vezeték a műszer elérhetőségét adja meg, az alsó mustársárga pedig az Error Cluster. Ha valamely SubVI-ban hiba keletkezik, akkor a többi nem fut le, kivéve a műszer lezárását végző utolsó elem.

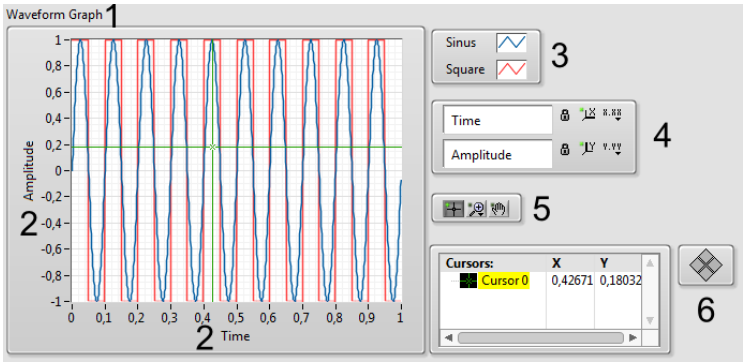
A SubVI ikonját minden esetben célszerű megváltoztatni, hogy az az aktuálisan végrehajtott funkciót tükrözze. Ez az ikon fog látszódni, amikor egy VI-ban felhasználjuk. A *File* menü *VI Properties* almenüjét kiválasztva a SubVI számos tulajdonságát megváltoztathatjuk. Ezek közül a SubVI későbbi felhasználását segíti, hogy ha helyesen kitöltjük a *Documentation* kategóriát. Az itt megadott leírások a *Context Help*-ben is megjelennek. Hasonlóan, az egyes beviteli és kimeneti mezők dokumentációját is érdemes megfelelően kitölteni. Az elkészített *subVI*-okat a *Functions Palette Select a VI...* menüjében választhatjuk ki és helyezhetjük el a diagramon.



1.32. ábra: Az ikonszerkesztő ablak.

Adatok grafikus megjelenítése

LabVIEW-ban az adatok grafikus ábrázolásának számos módja van, jellemzője ezeknek, hogy más programozási nyelvekhez képest nagyon egyszerűen beilleszthetők a programba, és olyanra alakítható a kinézetük, amilyenre szeretnénk.



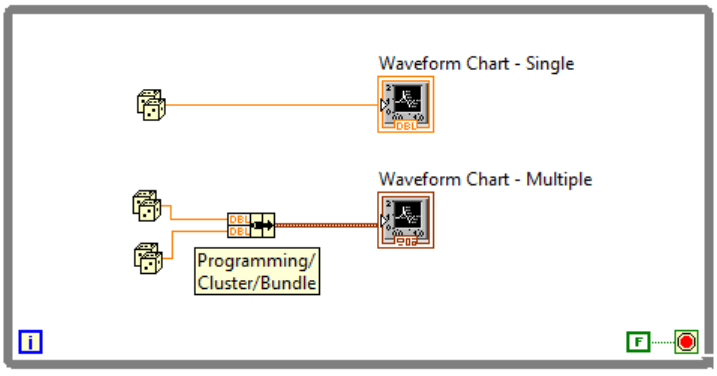
1.33. ábra: Tipikus grafikon. Az elemek többségét a *Visible Items* helyi menüből tudjuk láthatóvá tenni

A grafikonok fő elemei:

1) Grafikon neve (hasonlóan használható a *Caption* is)
2) Tengelyek felirata és a skála
3) *Plot Legend* – az egyes görbék neve. Itt változtatható meg a görbéknek a kinézete
4) *Scale Legend* – a skálákat lehet egyszerűen manipulálni a segítségével
5) *Graph Palette* – a grafikont lehet nagyítani, kurzorokat mozgatni…
6) *Cursor Legend* – kurzorokat hozhatunk létre, melyek értékeit kiolvashatjuk

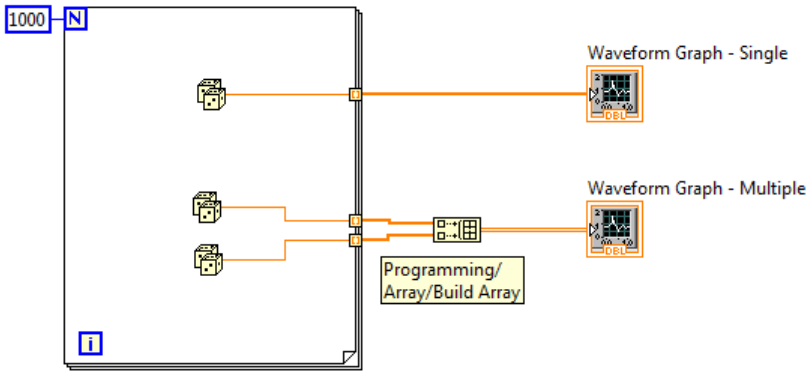
Attól függően, hogy milyen adatokat szeretnénk megjeleníteni, különböző grafikontípusokat érdemes választanunk.

A *Waveform Chart* működése során az aktuálisan megjeleníteni kívánt adatpontot hozzáfűzi a korábbi elemekhez, amit a saját belső memóriájában eltárol. A grafikonon a legutóbbi N elemet jeleníti meg, ahol az N számot mi tudjuk megadni. Ha az adatokat egy tömbbé fűzzük össze, akkor a tömb összes elemét hozzáfűzi a korábbi adatsor végére. Ha több görbét szeretnénk megjeleníteni, akkor egy *Cluster*t kell létrehozzunk. A vízszintes skála beosztását mind a *Properties* helyi menü segítségével, mind pedig a *Property Node*-on keresztül meg tudjuk változtatni (*X-scale Ofset* és *Multiplier*).



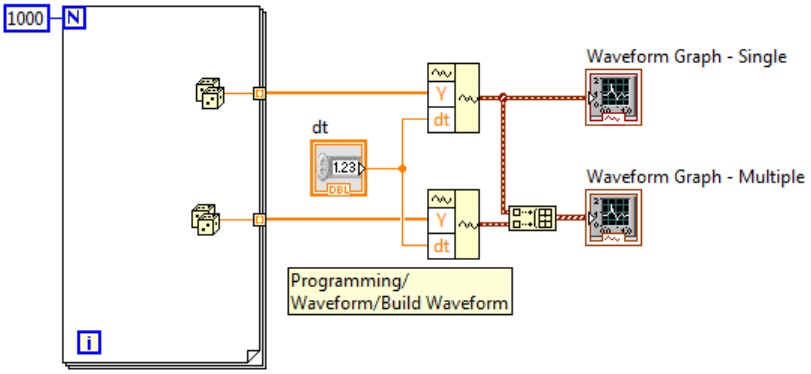
1.34. ábra: Adatok megjelenítése Waveform Chart-on

Hasonlóan jelennek meg az adatok a Waveform Graph esetén is, viszont ekkor az összes adatot egyszerre kapja meg a grafikon egy tömbben, a korábbi adatokat nem tárolja. Több grafikon megjelenítéséhez két dimenziós tömböt használhatunk.



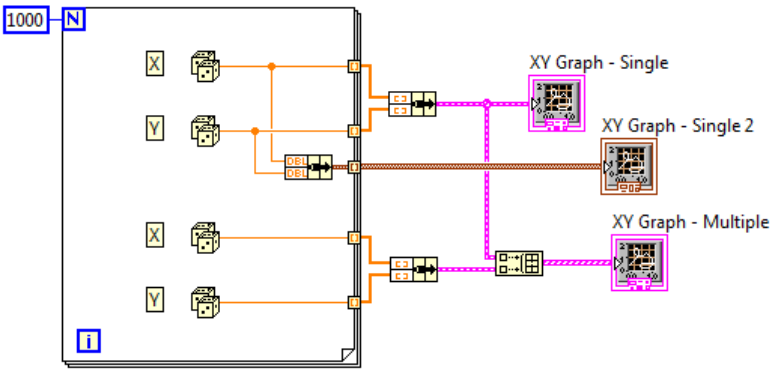
1.35. ábra: Adatok megjelenítése Waveform Graph segítségével

A LabVIEW-ban a mintavételezett adatok megjelenítéséhez a *Waveform* típusú változókat használhatjuk. A hullámforma adattípus egy klaszter, mely a mintavételezett jelértékek tömbje mellett a kezdőidőpontot és a mintavételi időközt is tartalmazza.

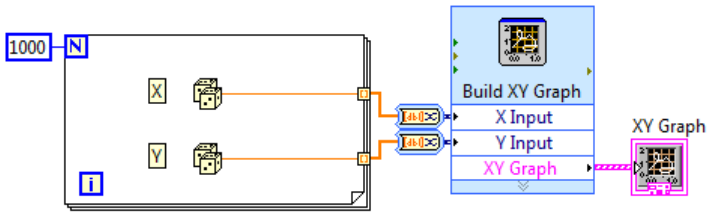


1.36. ábra: *Waveform*-ok megjelenítése grafikonon

Az *XY Graph*-on egymás függvényében ábrázolhatunk két jelet. Több módszer is van, mely segítségével ez elérhető, a legkézenfekvőbb, ha két tömböt (az x értékek és az y értékek) fűzünk össze egy két elemű *Cluster*-é. Ha több görbét szeretnénk látni, akkor ebből a *Cluster*ből építhetünk tömböt.



1.37. ábra: Az *XY Graph* használata

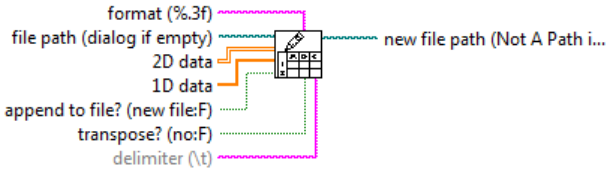


1.38. ábra: Az Express *XY Graph* használata

Adatok betöltése, exportálása

A mérési adatokat számos esetben el kell tárolnunk a számítógép merevlemezén, úgy, hogy azokat később is fel tudjuk használni, akár LabVIEW programokból, akár más programokból. Az egyik legkönnyebben használható formátum a szöveges formában tárolt táblázat. Bár ez a formátum nagyobb helyet foglal el, mintha binárisan tárolnánk az adatokat, viszont könnyen olvasható bármilyen program számára. Az egyetlen dolog, ami rendszeresen problémát okozhat, az a tizedesvessző-tizedespont probléma. A számítógép nyelvi beállításaitól függ, hogy melyiket értelmezik a programok, szükség esetén ezt a beállítást felül kell írni.

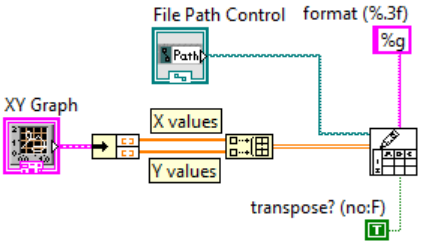
A szükséges fájlműveleteket a *Programming/File I/O* palettán találhatjuk, ezek közül a *Write to Spreadsheet File* és a *Read From Spreadsheet File* a legegyszerűbben használható.



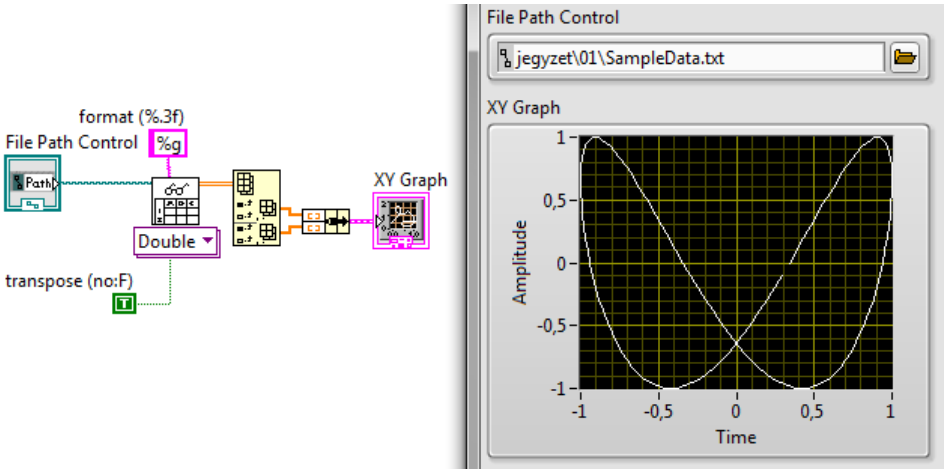
1.39. ábra: A *Write to Spreadsheet File* VI termináljai

Write to Spreadsheet File paraméterei:

- *file path* – itt tudjuk megadni, hogy hova írjon. Ha nincs meghatározva vagy bekötve, egy párbeszédablakban lehet kiválasztani a célt
- *2D data* – ha több oszlopot szeretnénk kiírni, akkor ezt a bemenetet kell használni
- *1D data* – ha csak egy adatsort akarunk fájlba menteni
- *append to file?* – hozzáfűzhetjük a korábbi fájl végére
- *transpose* – az alapbeállítás az, hogy az egyes adatsorokat sorokba menti. Ez eltér attól, hogy a legtöbb felhasználásban, mi oszlopokban szeretnék látni az adatokat (pl. az első oszlop az idő, a második a kitérés). Ezért ide érdemes bekötni egy *True* konstanst.
- *delimiter* - az elválasztó karakter meghatározása. Az alapértelmezett a tabulátor.
- *format* – a formátumot meghatározó *format string*. A C *printf* függvényéhez hasonlóan működik. A LabVIEW súgó *Format Specifier Syntax* oldalán olvashatunk részletesen a lehetőségekről. Az alapértelmezett beállítás (három tizedesjegy) sokszor nem elegendően pontos. Kis számok esetén pedig nem vált át automatikusan tudományos megjelenítésre, ezáltal az összes szám nulla lehet. Ezen okok miatt a „%g” (automatikus formázás) vagy a „%.6e” (tudományos formázás hat tizedejeggyel) sokkal jobb választás. Lehetőség van eltérni a rendszer tizedeskarakter beállításától is, ha biztosítani akarjuk, hogy tizedespontot használjon a program, akkor be kell szúrni a „%.;” karaktereket a formátumstring elé (pl. „%.;%.6e”)



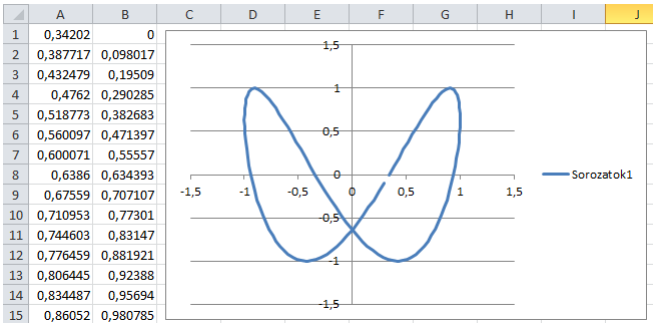
1.40. ábra: Adatok mentése fájlba a *Write to Spreadsheet File* segítségével. Az X tengely adatsora lesz az első oszlop, az Y tengely adatsora pedig a második.



1.41. ábra: A korábban kiírt adatok beolvasása és megjelenítése XY grafikonon.

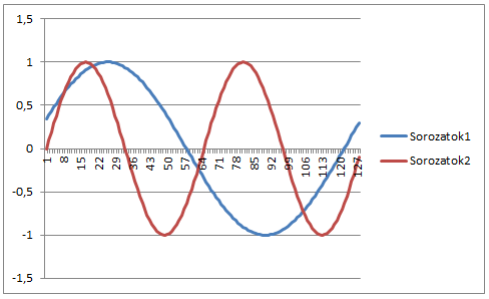
Az adatok beolvasása a *Read From Spreadsheet File* segítségével történik. Ugyanazokat a beállításokat kell használni, mint korábbi írás esetén. Amennyiben nem vagyunk biztosak abban, hogy milyen beállításokkal tudjuk beolvasni a kívánt adatokat, érdemes egy egyszerű tömbként kijelezni az összes beolvasott adatot (A *all rows* kimenet *Create/Indicator* helyi menüjét választva), így hibás beolvasás esetén könnyebb megtalálni a hiba okát. A leggyakoribb lehetséges hibák: a *transpose* nem megfelelő beállítása, a tizedeskarakter nem megfelelő beállítása, valamint az elválasztó karakter (*delimiter*) nem megfelelő beállítása.

A szöveges fájlba elmentett adatok táblázatkezelő programokkal is megnyithatók. Amennyiben az elválasztó karakter vagy a tizedeskarakter nem felel meg a program alapbeállításainak, akkor az adatokat megfelelő módon importálni kell. Excel esetén a következő menüből érhetjük el ezt az opciót: Adatok / Külső adatok átvétele / Szövegből. Itt megválaszthatjuk az elválasztó karaktert és a tizedesjegynek megfelelő karaktert. Amennyiben hibásan importáljuk be az adatokat, akkor az egyes cellákat nem számként, hanem szövegként fogja felismerni. Ez onnan ismerhető fel, hogy a szöveg bal oldalra van igazítva, és nem jobb oldalra, mint a számok.



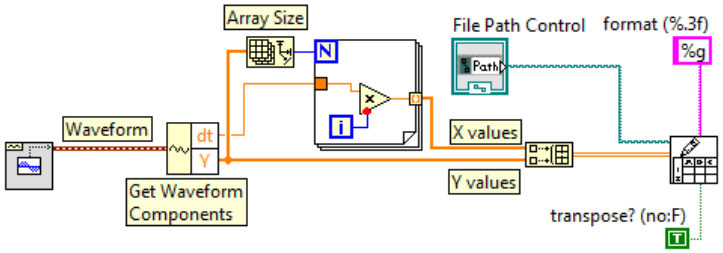
	A	B
1	0,34202	0
2	0,387717	0,098017
3	0,432479	0,19509
4	0,4762	0,290285
5	0,518773	0,382683
6	0,560097	0,471397
7	0,600071	0,55557
8	0,6386	0,634393
9	0,67559	0,707107
10	0,710953	0,77301
11	0,744603	0,83147
12	0,776459	0,881921
13	0,806445	0,92388
14	0,834487	0,95694
15	0,86052	0,980785

1.42. ábra: Az Excelbe beolvasott adatok, és azok Pontdiagramon ábrázolva



1.43. ábra: Hibásan, vonaldiagramon ábrázolt adatok. A két adatsor nem egymás függvényében van ábrázolva, az X tengely pedig a pont sorszámát tartalmazza.

Ha *Waveform*-ot szeretnénk menteni, akkor a mintavételi adatokat is el kell tárolnunk. Amennyiben más programokban szeretnénk megjeleníteni az adatokat, célszerű létrehozni azt a tömböt is, amely a mintavételek idejét tárolja. Sokszor nincs szükség a kezdőidőpontra, ekkor a következő ábrán bemutatott programmal célszerű menteni az adathalmazt. Az így elmentett adatokat Excelben pontdiagramon ábrázolhatjuk (a vonaldiagram szintén hibás megoldás).



1.44. ábra: Tipikus megoldás a *Waveform* mentésére, ahol a mintavétel időpontjait tartalmazó tömböt egy for ciklussal hozzuk létre

Alkalmazások

Differenciálegyenlet megoldása Euler-módszerrel

Differenciálegyenletek numerikus megoldásának legegyszerűbb módszere az Euler-módszer. Legyen a következő egyenlet amit meg szeretnénk oldani:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(t, x(t))$$

A megoldáshoz először meg kell adnunk a kezdőfeltételeket:

$$t_0 = t0$$

$$x_0 = x0$$

Majd pedig lépésről lépésre kiszámolhatjuk a megoldást:

$$t_{n+1} = t_n + dt$$

$$x_{n+1} = x_n + dt \cdot f(t_n, x_n)$$

A dt adja meg a szimuláció lépésközét, és alapvető hatással van a szimuláció pontosságára. Ha csökkentjük az értékét, a szimuláció egyre pontosabb lesz, annak árán, hogy hosszabb ideig fog futni a program.

Példa: van egy rugóra felfüggesztett testünk, melynek rögzítési pontját mozgathatjuk (gerjesztés). A testre lineáris légellenállás hat. Készítsünk olyan programot, mely kiszámítja a test mozgását és azt egy grafikonon ábrázolja.

Megoldás:

A mozgást leíró egyenletek:

$$F = -D \cdot (x + E) - C \cdot v$$

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{F}{m}$$

ahol x a test kitérése, E a gerjesztőjel, v a sebesség, m a test tömege, D a rugóállandó, C pedig a légellenállást jellemző konstans.

A megoldás során a kezdőfeltételek:

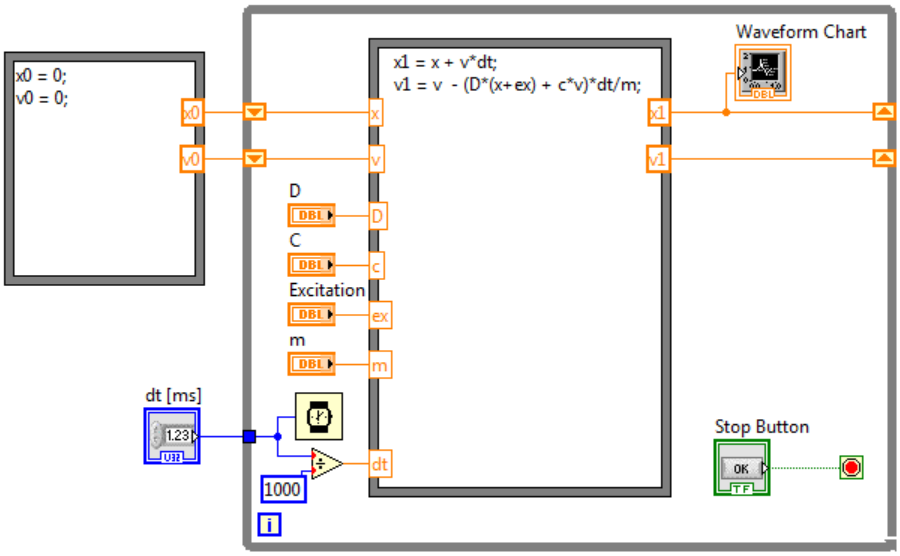
$$x_0 = 0$$

$$v_0 = 0$$

Az iterációban szereplő egyenletek pedig:

$$x_{n+1} = x_n + v_n \cdot dt$$

$$v_{n+1} = v_n - (D \cdot (x_n + ex_n) + C \cdot v_n) \cdot \frac{dt}{m}$$



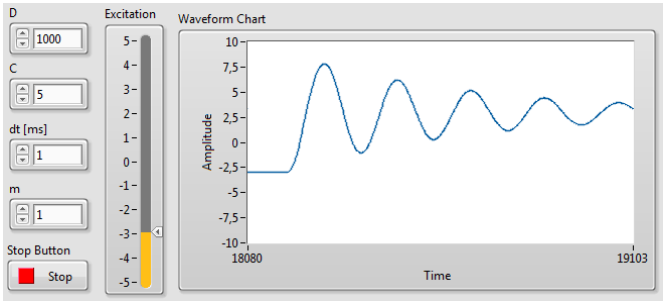
1.45. ábra: Egy rugóra függesztett test mozgását numerikusan megoldó program Block diagramja

A megoldást kiszámoló program egy *Formula Node*-val kezdődik. Ennek a megoldásnak az előnye, hogy akár bonyolultabb egyenleteket is beírhatunk szövegesen (C jellegű szintaxissal), ezzel jelentősen leegyszerűsíthetjük a programunkat. A *Formula Node*-ban az összes felhasznált változót definiálnunk kell, ha mást nem úgy, hogy azt bemenetként vagy kimenetként megadjuk. Ez a funkció a *Formula Node* szélén elérhető helyi menüben található meg (*Add Input*, *Add Output*). Itt mindenképp meg kell adnunk a változó nevét. Fontos, hogy a *Formula Node*-ba beírt összes sort pontos vesszővel kell zárni.

Az egyes iterációk egy ciklusban zajlanak le, ahol szintén egy *Formula Node*-dal számoljuk ki az egyenletek eredményét. A korábbi változók egy *Shift Register*-en keresztül érhetők el. Az idő lépésközét a dt változó adja meg. Ugyanez adja meg a várakozást az egyes ciklusok között,

így a szimuláció valós idejű lesz. A test aktuális helyzetét egy *Waveform Chart* grafikonon jelenítjük meg pontról-pontra. Mivel az *Excitation* változó a cikluson belül van, annak változása azonnal hatással van a szimulációra. Hasonlóan, a *D*, *C* és *m* változókat is menet közben módosíthatjuk. A *dt [ms]* időközt viszont nem.

Érdekesség: amennyiben a légellenállást nullának adjuk meg, a rezgés amplitúdója folyamatosan növekszik. Ez a szimuláció pontatlanságát mutatja, a dt csökkentésével javítható rajta (viszont a *Wait (ms)* rutin 1 ms-nál kisebb várakozást nem tesz lehetővé).



1.46. ábra: A rugóra felfüggesztett test mozgása gerjesztés hatására

Feladatok

1. feladat

Adjon össze két komplex számot majd jelenítse meg a végeredményt!

2. feladat

a) Hozzon létre olyan programot, mely kiszámolja egy másodfokú egyenlet gyökeit!

b) Mikor nincs megoldása az egyenletnek? Hogyan számolhatjuk ki ilyenkor is a gyököket?

c) Hozzon létre olyan SubVI-t, mely másodfokú egyenlet gyökei számolja ki, és más programokba is beilleszthető!

3. feladat

a) Számolja ki egy egész szám faktoriálisát!

b) Legfeljebb mekkora szám faktoriálisát számolhatja ki pontosan különböző reprezentáció esetén?

4. feladat

Jelenítse meg az egydimenziós véletlen bolyongás folyamatát egy Waveform Chart-on!

A mozgást leíró egyenlet:

$$x_{i+1} = x_i + \xi_i$$

ahol ξ_i egy -1 és $+1$ közötti véletlenszám.

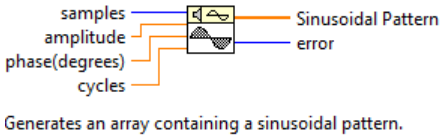
5. feladat

Hozzon létre olyan programot, mely egy jelzőlámpa működését szimulálja le!

6. feladat

a) Rajzoljon ki Lissajous-görbéket egy XY-Graph segítségével.

b) A Lissajous-görbék változzanak időben, mint a régi scifik-ben!



01-600 – A program megvalósításához ajánlott rutin, mely létrehoz egy tömböt, melyben egy megadott periodusszámú (*cycles*) és fázisú (*phase*) szinusz mintázat van

7. feladat

a) Készítsen olyan programot, mely a ferde hajítás útvonalát ábrázolja XY grafikonon!

A mozgást leíró differenciálegyenletek:

$$\frac{\mathrm{d}\vec{r}(t)}{\mathrm{d}t} = \vec{v}(t)$$

$$\frac{\mathrm{d}\vec{v}(t)}{\mathrm{d}t} = \vec{a}(t) = \frac{\overrightarrow{F_{\text{eredő}}}}{m} = \frac{m \cdot \vec{g}}{m}$$

A kezdőfeltételek: $\overrightarrow{r(0)} = 0$ és $\overrightarrow{v(0)} = \overrightarrow{v_0}$.

Megjegyzés: a megoldás során a vektorokat fel kell bontani x és y irányú értékekre, így skalárokkal már számolható a megoldás.

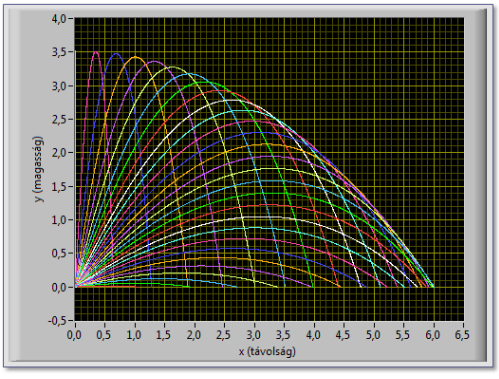
b) Vegye figyelembe a levegő négyzetes légellenállását!

A légellenállással az eredő erő fog módosulni a következőnek megfelelően:

$$\overrightarrow{F_{eredő}} = m \cdot \vec{g} - c \cdot \overrightarrow{v^2(t)} \cdot \hat{v}$$

Megjegyzés: v^2 számolásakor nem elég csak az egyik tengelyt figyelembe venni, az sebességvektor teljes hosszára szükség van. $\hat{v}$ a sebesség irányába mutató egységvektor.

c) Ábrázolja egy grafikonon, hogyan függnek a mozgás pályái a kezdőszög függvényében!



1.47. ábra: Példa eredmény a c) feladatra

d) Ábrázolja egy grafikonon a pályákat különböző légellenállások esetén!

e) Ábrázolja egy XY grafikonon a hajítás távolságát a kezdőszög függvényében!

f) Ábrázolja egy XY grafikonon a légellenállás függvényében, hogy mely kezdőszöghöz tartozik a legnagyobb megtett távolság!

2. Multiméterek használata

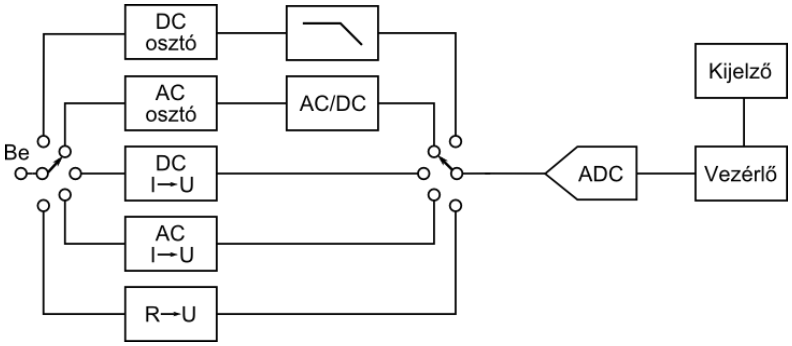
A multiméter egy olyan eszköz, mellyel különböző elektromos mennyiségeket (feszültség, áramerősség, ellenállás) mérhetünk meg. Míg a múltban elsősorban analóg, mutatós multimétereket használtak, ma már egyeduralkodók a digitális multiméterek. A multiméterek lassan változó mennyiségek mérésére alkalmasak, ha nagyobb frekvenciájú jelek vannak jelen az áramkörben, hibás eredményre juthatunk.

2.1. ábra: Tipikus multiméterek

A multiméterek fő elemei:

- Kijelző – itt olvashatjuk le a mért értéket. Elsősorban számokat jelenít meg, de típustól függően további hasznos információt is megjelenhetnek (mérésmód, méréshatár, maximum/minimum, analóg kijelzés, az elem töltöttsége). A kijelzőn lévő számjegyek száma határozza meg a felbontást, ezt digitben fejezzük ki. A 3,5 digites kijelző azt jelöli, hogy van 3 teljes értékű digit, továbbá a legnagyobb helyiértékű digit amely -1 és +1 között változhat.
- Üzemmódválasztó tárcsa – ennek segítségével választhatjuk ki a kívánt mérési üzemmódot. Az multiméterek egy részénél ugyanezzel a tárcsával választhatjuk ki a méréstartományt is, valamint ennek segítségével kapcsolhatjuk ki a multimétert. Vannak automatikus méréshatárú multiméterek, ahol a méréstartományt maga a multiméter választja ki a mért értéknek megfelelően.
- Mérővezetékek, mérőcsúcsok – ezek segítségével vezetjük el a mért jeleket a multiméterbe. Tipikusan egy fekete (közös vagy föld) és egy piros (jel) mérővezetéket használhatunk. A mérőcsúcs végének kiképezése feladattól függően más és más lehet (pl. banándugó, hegyes csúcs, csipesz)
- Bemenő csatlakozók – ezek speciális szigeteléssel ellátott banándugó bemenetek. A legtöbb mérőműszeren négy bemenet, melyeket a használt funkció függvényében kell használatba venni. Van olyan multiméter, amely figyelmeztet hibás bekötés esetén, pl. ha árammérés üzemmódban van a műszer, de a piros vezeték a „V/Ω” bemenetre van bekötve. A „COM” csatlakozó a közös pont, melyet mindegyik mérési üzemmód esetén használni szoktunk, ide kell csatlakoztatni a fekete mérőfejet. A „mA” jelölésű bemenetet kis áramok mérése esetén használhatjuk, és minden esetben biztosítékkal van védve. Az „20A” jelölésű csatlakozót nagy áramok mérésekor használhatjuk, ez azonban sokszor nincs biztosítékkal védve. A „V/Ω” bemenetet feszültség és ellenállás mérés esetén használjuk, illetve minden más mérési üzemmód esetén (frekvencia mérése, szakadásvizsgálat…). Némelyik multiméter tartalmazhat további

csatlakozókat is (tranzisztor vizsgálata, kapacitás mérése, hőmérséklet mérése, négy pontos ellenállásmérés, hátlapi csatlakozók)



2.2. ábra: Egy tipikus multiméter blokkvázlata

A digitális multiméterek többsége egy olyan A/D konvertert tartalmaz, amely -200 és +200 mV közötti feszültségek mérésére alkalmas, és kettős integrálás elvén működik. A bemeneti osztók és átalakítók feladata, hogy a mért elektromos jelet átalakítsa, majd leossza, hogy ebbe a feszültségtartományba kerüljön. A megjelenítésről a vezérlőáramkör és a kijelző gondoskodik.

A hordozható, kézi multiméterekkel szemben az asztali multiméterek általában sokkal nagyobb felbontással és pontossággal rendelkeznek, alkalmasak automatizált mérésekre, és számítógépről vezérelhetők. Ehhez használhatunk USB interfészt, soros interfészt, Ethernetet vagy ipari interfészeket (GPIB) is.

2.3. ábra: Egy asztali multiméter képe

Egy multiméter vásárlásakor számos szempontot kell figyelembe venni a döntés előtt. Az egyik legfontosabb kérdés a műszer minősége., hiszen egy rossz minőségű multiméter jelentősen akadályozhatja a munkát. A napi használat során a csatlakozók és kapcsolók állapota kritikus, gyengébb minőségű multiméterek esetén ezek eloxidálódnak, ezáltal bizonytalan lesz az érintkezésük és a műszer által kijelzett érték is.

A következő két legfontosabb paraméter a felbontás és a pontosság. Bár általában van közük egymáshoz, nem egyeznek meg. A felbontás azt adja meg, hogy legalább mekkora kell legyen két mért mennyiség között a különbség, hogy azokat a műszer meg tudja különböztetni. A felbontást többféleképpen is megadhatják, egyrészt a digitek számával (pl. 3,5 digites), és a legnagyobb kijelezhető számmal+1-el (*count*, pl. 2000). A legáltalánosabban elterjedt multiméterek 3,5 digitesek, jellemzőek még a 4,5 digites multiméterek is. Ennél nagyobb felbontással már leginkább speciális igényű, nagy pontosságú multiméterek rendelkeznek.

A műszer pontossága azt adja meg, hogy legfeljebb mekkora lehet az eltérés a valódi és a mért érték között. A műszerek esetén ez mindig megadott feltételek között igaz csak (méréstartomány, hőmérséklet, páratartalom, rendszeres kalibrálás…). Példa a pontosság

megadására: $\pm 1\% + 3d$. Az első rész adja meg a hibának azt a részét, mely arányos a mért értékkel. A második tag digitben van megadva, és a mért értéktől független. 1 digit az utolsó számjegy eggyel való változására utal, és a felbontással egyezik meg. A multiméterek hibájának nagy része determinisztikus, így az átlagolás nem segít (kivéve a nagy felbontású multiméterek esetén, ahol sokszor maguk a multiméterek adnak olyan lehetőséget, hogy átlagolhassuk az egymás utáni mérések eredményét).

Mivel a multiméterek elemről működnek, csak akkor tekinthető a mérés az adatlapban megadott pontosságúnak, ha az elem megfelelő töltöttségű, amit a multiméternek jeleznie kell. Olcsóbb multiméterekben ez a funkció gyakran hiányzik, ebben az esetben megbízható mérés végzésére alkalmatlanok, hiszen sosem bízhatunk meg a kijelzett értékben.

Példa hibaszámolásra

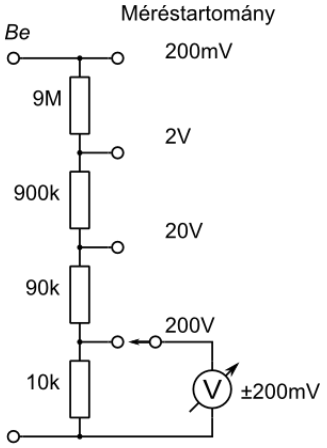
Tegyük fel, hogy feszültséget mérünk egy 3,5 digites multiméterrel 20 V-os méréstartományban, az adatlap szerint a műszer pontossága $\pm 1{,}5\% + 3d$. A műszer által kijelzett érték 4,35 V. Mekkora a mérés hibája?

A műszer által kijelzett értékek $-19{,}99$ és $+19{,}99$ között változhatnak, a felbontás, vagyis egy digit 0,01 V. A hiba így:

$$H = 4{,}65\ \text{V} \cdot 1{,}5/100 + 3 \cdot 0{,}01\ \text{V} = 0{,}09975\ \text{V}$$

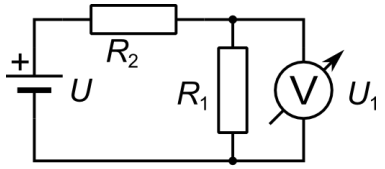
Feszültség mérése multiméterrel

Feszültség mérése esetén a bemenet egy feszültségosztóra van rávezetve. A feszültségosztó kimenetét méri meg az A/D konverter, majd kijelzi az eredményt. A feszültségosztó úgy van megvalósítva, hogy méréstartománytól függetlenül mindig állandó a műszer impedanciája. Az impedancia tipikusan 10 MΩ, gyenge minőségű multiméterek esetén elterjedt az 1 MΩ-os impedancia, speciális, pl. asztali multiméterek esetén bizonyos méréstartományokban ugyanakkor 10 GΩ feletti impedancia is választható, mely sokszor nélkülözhetetlen nagy pontosságú mérések kivitelezéséhez.



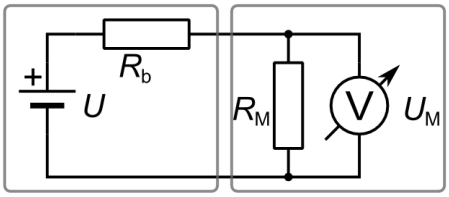
2.4. ábra: A bemeneti feszültségosztó felépítése

Feszültség mérését úgy végezzük, hogy a műszert párhuzamosan kötjük be az áramkörbe azzal az alkatrésszel melyen feszültséget kívánunk mérni. E közben figyelni kell arra, hogy a mérőfejekkel ne okozzunk rövidzárlatot az áramkörben. A méréshez nem kell megszakítani az áramkört. A multiméter akkor jelez ki pozitív értéket, ha a piros mérőfej potenciálja nagyobb a fekete mérőfej (COM) potenciáljánál. Egyébként negatív értéket mutat.



2.5. ábra: Feszültség mérése multiméterrel. Az R_1 ellenálláson lévő feszültséget mérjük.

Ha a mérendő feszültségforrás impedanciája túlságosan nagy, akkor már figyelembe kell venni a multiméter impedanciáját is.



2.6. ábra: A multiméter bemenő impedanciájának (R_M) hatása egy feszültségforrás feszültségének mérésére. A feszültségforrás belső ellenállása R_b

Például, ha $R_\mathrm{b} = 100\ \mathrm{k\Omega}$, $R_\mathrm{m} = 1\ \mathrm{M\Omega}$, $U = 10\ \mathrm{V}$

$$U_\mathrm{M} = I \cdot R_\mathrm{M} = \frac{U}{R_\mathrm{b} + R_\mathrm{M}} \cdot R_\mathrm{M} = \frac{10\ \mathrm{V}}{100\ \mathrm{k\Omega}+1\ \mathrm{M\Omega}} \cdot 1\ \mathrm{M\Omega} = 9{,}09\ \mathrm{V}$$

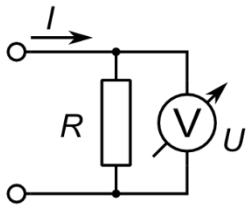
A mérés relatív hibája: $h = 9{,}09\ \%$

Váltakozó jelek mérése során legtöbbször a feszültség (vagy az áramerősség) effektív értékére vagyunk kíváncsiak. Az ehhez szükséges négyzetes átlag mérése komolyabb elektronikát igényel, ezért a legtöbb műszer egyenirányítás után egyszerű közepet mér, majd a mért értéket átskálázza úgy, hogy szinuszos jel esetén pont az effektív érték legyen a végeredmény. Nem szinuszos jelek esetén az eredmény szisztematikusan eltér a valódi RMS értéktől. Magasabb kategóriájú műszerek ezzel szemben a valódi effektív értékét mérik meg, ezt azzal jelölik, hogy *True RMS*-t mérnek. Az effektív értékről bővebben a 6. fejezetben lehet olvasni.

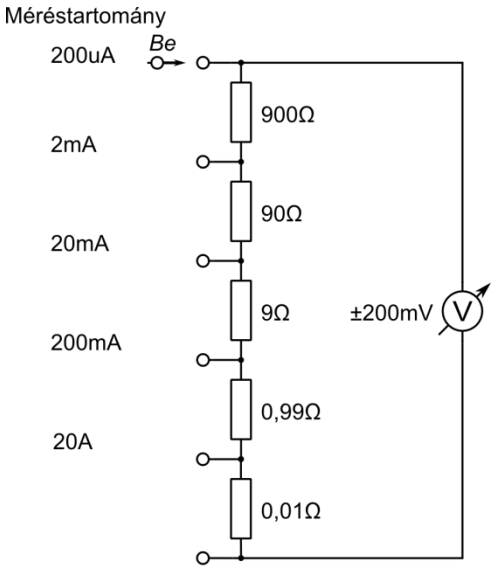
Váltakozó jelek esetén fontos a műszernek a sávszélessége. Az átlagos multiméterek csak néhány száz Hz-ig használhatók, e felett már pontatlanok lesznek. Ha magasabb frekvencián szeretnénk pontosan mérni, mindenképp meg kell vizsgálni az adott multiméter specifikációját.

Áram mérése multiméterrel

Az áramerősség méréséhez először át kell alakítani a mérendő áramerősséget feszültséggé, melyet a multiméter belső A/D konvertere ezek után már meg tud mérni. Az egyik legegyszerűbb megoldás, hogy ha az áram egy ellenálláson keresztül folyik át, majd mérjük az ezen eső feszültséget (ez 200-300 mV nagyságrendű is lehet). Az ellenállás nagysága adja a műszer belső impedanciáját, mely befolyásolja a mért áramerősség nagyságát, gyakorlatilag az egész áramkör működését módosítja. Kis áramerősségek esetén van lehetőség aktív áramerősség-feszültség konverzióra is, ekkor a feszültségesés már csak néhány mV. Nagyobb áramerősségek esetén az áram mágneses terét is felhasználhatjuk az áramerősség mérésére.

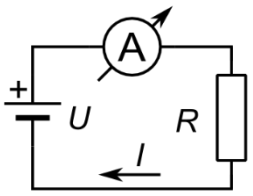


2.7. ábra: Az áramesősség-feszültség konverzió egy ellenállás segítségével



2.8. ábra: Az áramerősség mérése egy multiméterben

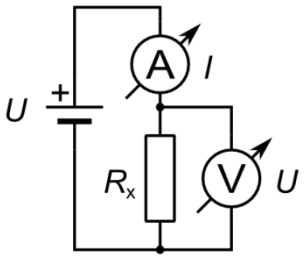
Az áramerősség méréséhez először meg kell szakítani az áramkört, majd pedig sorosan beépíteni az árammérőt (gyakori hiba, hogy az árammérőt párhuzamosan építik az áramkörbe: ez rövidzárlatot okoz!). A beépített árammérő a belső ellenállása miatt jelentősen módosíthatja az áramkör működését, ráadásul a belső ellenállás a méréstartománytól is függ. Áramerősség méréséhez a mA vagy az A bemeneteket kell használni, ez a csatlakozó biztosítékkal van védve. A 20A-es bemenő csatlakozó legtöbbször nincs biztosítékkal védve, valamint huzamos használata esetén (> 20 s) a műszer túlmelegedhet.



2.9. ábra: Az árammérő bekötése az áramkörbe. Az áramkört minden esetben először meg kell szakítani. A fenti esetben a műszer akkor mutat pozitív értéket, ha a piros vezeték van a tápfeszültség pozitív vége felé. Ekkor az áram a műszer piros bemenetén befelé, a fekete (COM) bemenetén pedig kifelé folyik.

Mivel az árammérőt legtöbbször kényelmetlen beépíteni az áramkörbe, rendszeresen alkalmazunk olyan megoldást, hogy az áramkörbe már beépített, ismert értékű ellenálláson mérünk feszültséget, és a feszültségesésből számoljuk ki az ellenálláson átfolyó áramot.

Ellenállás mérése multiméterrel



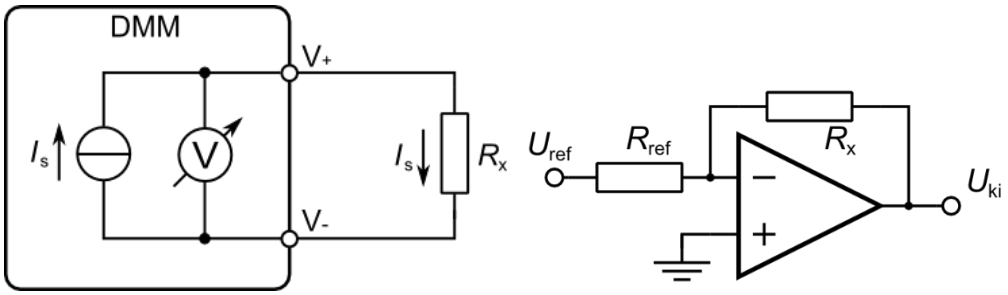
2.10. ábra: Ellenállás mérésének alapelve

Az Ohm-törvény alapján egy ellenállás értékét a legkönnyebben úgy mérhetjük meg, hogy rákötünk egy feszültségforrást, majd pedig megmérjük az ellenálláson eső feszültséget, és az ellenálláson átfolyó áramot.

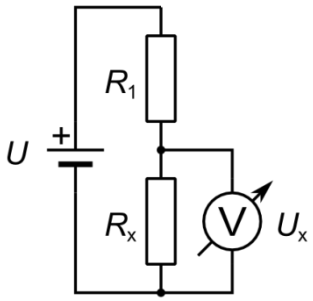
$$R = \frac{U}{I}$$

A fent vázolt alapmérés helyett általában olyan elrendezést szoktunk alkalmazni, ahol csak egy mennyiséget kell megmérni. Az egyik ilyen megoldás, ahol egy áramgenerátort kötünk az ellenállás két végére, és mérjük az ellenálláson eső feszültséget. A másik elterjedt megoldás pedig, amikor egy feszültségosztó egyik tagja az ismeretlen ellenállás. Ekkor a kimenő feszültség:

$$U_\mathrm{x} = \frac{R_\mathrm{x}}{R_1 + R_\mathrm{x}} \cdot U$$



2.11. ábra: Ellenállás mérése áramgenerátorral. Bal oldalon egy egyszerű alapkapcsolás, a jobb oldalon pedig egy műveleti erősítővel megvalósított verziója



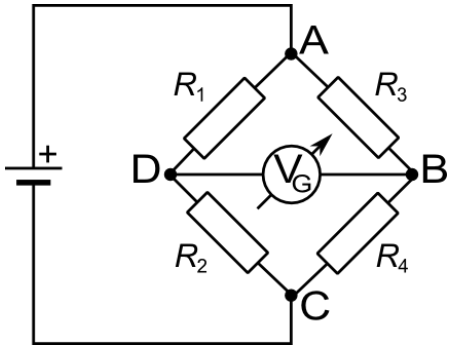
2.12. ábra: Ellenállásosztóval megvalósított ellenállás mérési mód

Szenzorok ellenállásának mérése során egyik elterjedt módszer a Wheatstone-híd. Ha a híd egyensúlyban van, vagyis $V_\mathrm{G} = 0$:

$$R_\mathrm{x} = \frac{R_3 \cdot R_2}{R_1}$$

Általában pedig V_G-t így tudjuk kiszámítani:

$$V_G = \left(\frac{R_x}{R_3 + R_x} - \frac{R_2}{R_1 + R_2}\right)$$

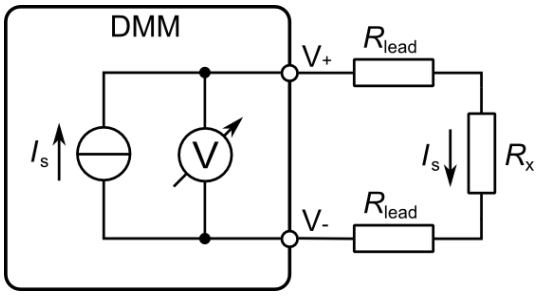


2.13. ábra: A Wheatstone-híd

A kapcsolás kifejezetten alkalmas kis ellenállás-változások mérésére. Ebben az elrendezésben sokkal jobban ki tudjuk használni a méréstartományt, mint az egyszerű ellenállásosztó esetén. A mérőkapcsolást gyakran használják platina hőmérsékletszenzorok, nyúlásmérő bélyegek és nyomásmérők esetén.

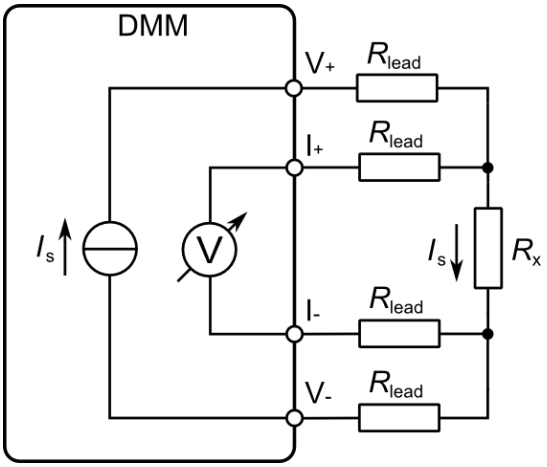
Egy ellenállás mérése közben a következőkre kell figyelni:

- A mérendő ellenállást ki kell venni az áramkörből. Ellenkező esetben az áram egy része nem az ellenálláson keresztül fog átfolyni, így hibás mérésre jutunk. Feszültség alatt lévő áramkörben tilos ellenállást mérni.
- Egy ellenállás mérése közben oda kell figyelni, hogy legfeljebb csak az egyik oldalát foghatjuk meg kézzel. Különben az emberi test párhuzamosan lesz kötve a mérendő ellenállásnál, ami nagyobb ellenállások esetén (kΩ felett) már jelentős hibát okozhat. Pontos mérésekhez egyáltalán ne fogjuk meg az ellenállást!
- Alacsony értékű ellenállások esetén (kΩ alatt) a vezetékek valamint a kontaktusok ellenállása már számottevő arányt képviselhetnek, így a pontos mérésekhez ezek ellenállását is figyelembe kell venni, illetve kiküszöbölni.



2.14. ábra: Két pontos ellenállásmérés. A vezetékek ellenállása hozzáadódik a mért ellenálláshoz.

Ha ki akarjuk küszöbölni a vezetékek és a kontaktusok hatását, akkor négypontos ellenállásmérést alkalmazhatunk. Itt két vezeték szolgál az ellenállás áramának biztosításához, másik kettő (*sense*) pedig a feszültség méréséhez. Utóbbi két vezetéken minimális áram folyik, így a vezetékeken gyakorlatilag nem esik feszültség. Négypontos ellenállásmérést asztali multiméterekkel közvetlenül is tudunk végezni.



2.15. ábra: Négy pontos ellenállásmérés

Egy speciális multiméter-család az úgynevezett *Source Meter*. Hagyományos multimétereknél az áramgenerátor árama többnyire fix, nem lehet szabályozni legfeljebb a méréstartomány változtatásával. *Source Meter* esetén viszont nagy pontossággal szabályozható akár az ellenállásra kapcsolt áram, akár az ellenállásra kötött feszültség, és ez a szokásos mA-ek helyett akár ampereket is elérhet (az ellenállásra kötött feszültség pedig akár a 100 V-ot). Segítségével a feszültség-áram karakterisztika külön tápegység nélkül is kimérhető.

További mérések multiméterrel

Az ellenállásméréshez kötődik a multiméter folytonosságvizsgáló funkciója. Ekkor a multiméter sípol, amennyiben rövidzárlatot érzékel (vagyis a mért ellenállás kisebb nagyjából 10 Ω-nál). Ezt a funkciót általában hibakereséshez használjuk. Némelyik multiméter azonnal reagál a rövidzárlatra, némelyik esetén viszont meg kell várjuk az ellenállásmérés eredményét, így nagyjából fél másodperc múlva kapunk csak eredményt.

Diódateszteléskor a dióda nyitóirányú feszültségét mérhetjük meg, ez tipikusan 0,5 V és 0,8 V közé esik.

Kapacitás méréséhez a multiméterek egy részének külön csatlakozója van, ahova be kell illeszteni a mérendő kapacitást. Kis kapacitások mérésénél figyelni kell arra, hogy mekkora a műszer (és a vezetékek) belső parazita kapacitása, amely akár 10 pF nagyságrendű is lehet.

A multiméter nem alkalmas induktivitások mérésére, erre speciális RLC műszert használhatunk. Induktivitás mérését speciális RLC mérő műszerekkel végezhetünk. Ez kifejezetten ellenállás, kapacitás és induktivitás mérésére van kiképezve, és nem csak a konkrét induktivitást/kapacitást méri meg, hanem az alkatrészek nem ideális voltához köthető további paramétereket (pl. ohmikus tag).

A multiméterek nagy része alkalmas frekvencia merésére. Ilyenkor figyelni kell arra, hogy milyen amplitúdójú jelek fogadására alkalmas, általában legalább 200 mV amplitúdójú kell legyen a mérendő jel.

A multiméterek egy része hőmérsékletet is tud mérni, ehhez szükség van egy megfelelő típusú (általában K-típusú) termoelemre.

Multiméterekkel mérhető a bipoláris tranzisztorok áramerősítési tényezője (β, h_{FE}). Erre a célra a multiméteren egy speciális csatlakozósor szolgál, ’B’, ’C’, ’E’ feliratokkal. Ha a

multiméter által mutatott érték jelentősen eltér az adatlap által megadottól, előfordulhat, hogy a tranzisztor meghibásodott, de az is, hogy rosszul állapítottuk meg az eszköz bekötését.

Feladatok

1. feladat

Egy multiméter 22 kΩ méréstartományában a felbontás 0,001 kΩ, a pontossága pedig $\pm(0{,}5\% + 10)$. Ha ebben a méréstartományban azt kapjuk, hogy az ellenállás értéke 13,621 kΩ, mekkora lehet a mérés hibája?

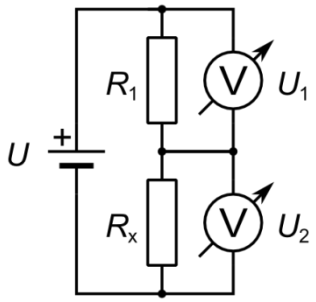
2. feladat

Egy 100 kΩ-os belső ellenállású feszültségforrás feszültségét mérjük egy 10 MΩ-os bemenő impedanciájú multiméterrel. Mekkora relatív hibával mérjük a feszültséget? Mekkora a hiba, hogy ha a feszültségforrás belső ellenállása 1 MΩ illetve 10 MΩ?

3. feladat

Az alábbi elrendezés segítségével mérjük egy R_x ellenállás nagyságát. $R_1 = 1$ MΩ, $U_1 = 7{,}3$ V, $U_2 = 4{,}2$ V, a feszültségmérők belső ellenállása 10 MΩ.

Mekkora az ismeretlen ellenállás értéke? Mekkora relatív hibát követünk el, ha nem vesszük figyelembe a műszerek belső ellenállását?



2.16. ábra: A 3. feladat elrendezése

4. feladat

Egy 0,6 V-os feszültségforrásra sorosan kapcsolunk egy 25 V-os ellenállást és egy árammérőt. Mekkora áram folyik az áramkörben az árammérő különböző állásaiban, és mit mutat az árammérő?

Az árammérő belső ellenállása:

Mérőállás	**2 A**	**200 mA**	**20 mA**	**2 mA**
Belső ellenállás [Ω]	0,1	1	10	100

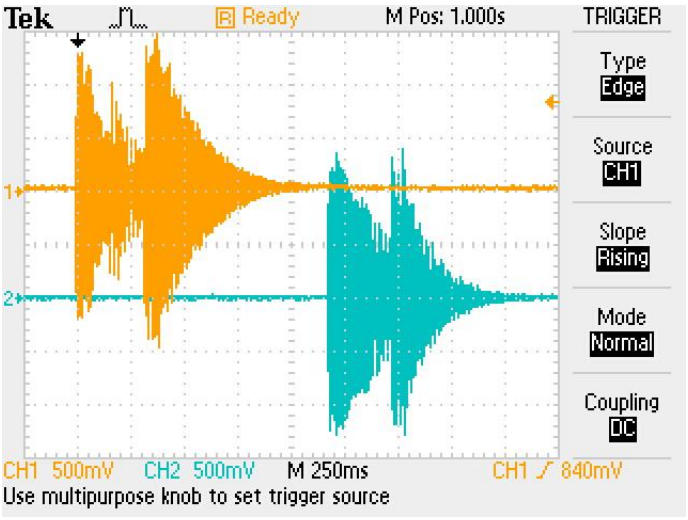
5. feladat

Vizsgálja meg a kiadott tápegység esetén a kimenő feszültséget valamint a kimenő áramerősséget a terhelés függvényben!

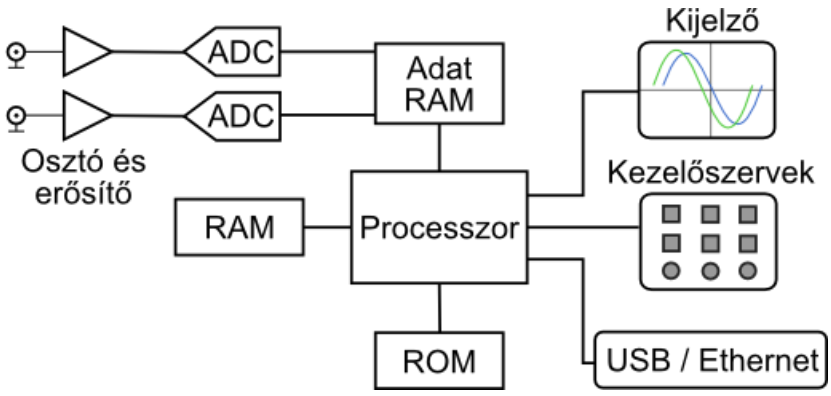
3. Oszcilloszkópok használata

Az oszcilloszkóp egy olyan műszer, mely váltakozó jelek (elsősorban feszültség) mérésére és grafikus megjelenítésére alkalmas. Megkülönböztetünk analóg és digitális oszcilloszkópokat. A jegyzet keretein belül a digitális oszcilloszkópokkal foglalkozunk, mivel ma már egyre elterjedtebbek. A kifutónak tekinthető analóg oszcilloszkópokról bővebben a Michailovits L. által írt jegyzet 8. fejezetében lehet olvasni [19].

Az oszcilloszkópon megjelenő grafikon vízszintes tengelye jelenti az időt, a függőleges tengely pedig a feszültséget. Ahhoz, hogy az oszcilloszkóp szinkronban legyen a mérendő jelenséggel, illetve a kívánt eseményt mérjük meg, az indítójel (trigger jel) megfelelő beállítása szükséges.



3.1. ábra: Példa egy mért jelre (a számítógép hangkártyájának teszt jele)



3.2. ábra: Egy digitális oszcilloszkóp egyszerűsített blokkvázlata

Egy digitális oszcilloszkóp egy nagy sebességű (tipikusan GHz-es, 8-10 bit-es) A/D konverter segítségével vesz egyenletes időközönként mintákat a jelből, majd ezt egy gyors memóriában eltárolja. A mintavételezett jelet nagy sebességű digitális elektronikával feldolgozza, majd egy LCD kijelzőn megjeleníti az eredményt. Mindegyik digitális oszcilloszkóp alkalmas a mért jelalakok tárolására, valamint az indítójel előtti mintavételezésre is (*pretrigger*). Már ez a két funkció önmagában is jelentős előnyt nyújt egy hagyományos analóg oszcilloszkópokkal szemben.

Digitális oszcilloszkópból elsősorban három fajtát különböztetünk meg:

- Asztali oszcilloszkóp. A kisebb tömeg miatt már ezek is sokkal könnyebben hordozhatók az analóg társaikkal szemben.

- Kézi oszcilloszkóp. Ezek multiméter nagyságú oszcilloszkópok, melyek saját nagyméretű kijelzővel rendelkeznek és a hagyományos multiméter funkciók mellett alkalmasak váltakozó jelek mérésére is.
- PC bázisú oszcilloszkópok. Ezek nem rendelkeznek saját kezelőszervekkel és kijelzővel, a számítógépről vezérelhetjük őket. A számítógéphez való csatlakozás történhet USB porton, Ethernet interfészen vagy ipari interfészen keresztül.

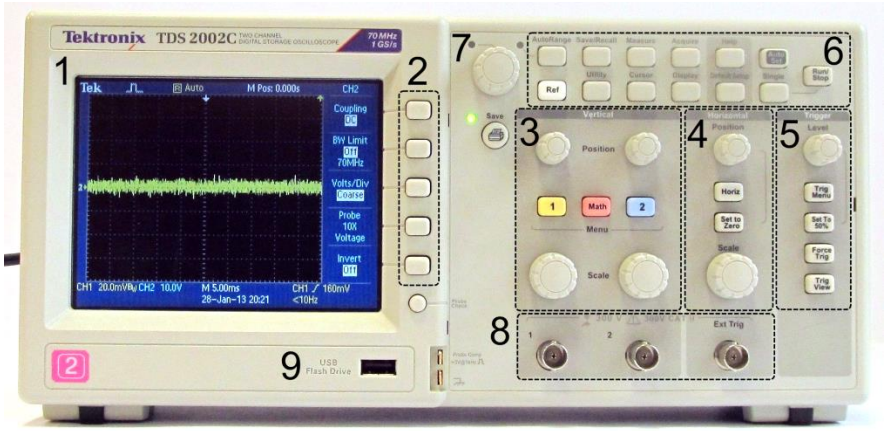
3.3. ábra: Példa PC-oszcilloszkópra [20]

A digitális oszcilloszkópok egy része alkalmas logikai jelek vizsgálatára is (*mixed-signal oscilloscope*). Ekkor 2-4 analóg jel mellett 8-16 kétállapotú logikai jelet is mérhetünk egyidejűleg, majd ezeket együtt megjeleníthetjük a kijelzőn. A digitális jeleket fel is lehet dolgozni, pl. 8 csatornából egy 8 bit-es számot is megjeleníthetünk.

Az oszcilloszkópok legfontosabb paraméterei:

- Analóg sávszélesség (az oszcilloszkóp analóg elektronikájának 3dB-es csillapításhoz tartozó frekvenciája)
- Mintavételi sebesség, adat/s vagy frekvenciaegységekben
- Függőleges felbontás
- Az adattároló memória mérete
- Az analóg csatornák száma
- A digitális csatornák száma

Az oszcilloszkóp képességeit alapvetően meghatározza a rajta futó szoftver. A szoftver bővítésével új funkciók lesznek elérhetők (ezeket általában külön kell megvásárolni). Érdekesség, hogy néhány gyártó bizonyos típusokból csak egy fajta hardvert gyárt, a sávszélességet pedig szoftveresen lehet „bővíteni”. E mellett akár multimétert és jelgenerátort is beépítenek az oszcilloszkópba, melyet szintén a megfelelő modul megvásárlásával aktiválhatunk.



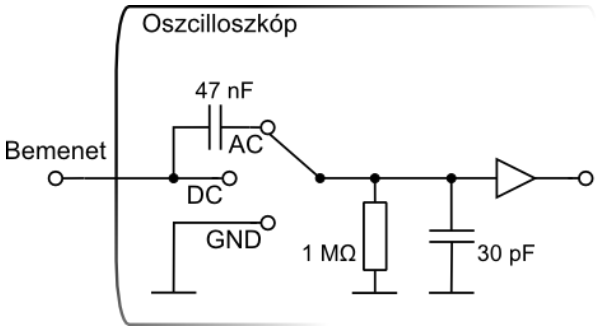
3.4. ábra: Asztali digitális oszcilloszkóp

Egy digitális oszcilloszkópon az összes kezelőszerv a szoftveren keresztül befolyásolja a méréseket, így az aktuális funkciójuk a beállításoktól is függhet. A kijelző mellett van egy gombsor (*Options Buttons*) melynek a funkciója a kijelzőn van megjelenítve. Minden egyes csatornához tartozik egy gomb (általában színes), mellyel az adott csatornához tartozó menüt jeleníthetjük meg. A kiválasztott csatornát ugyanezen gomb ismételt megnyomásával tudjuk inaktiválni, ekkor eltűnik a kijelzőről. Az oszcilloszkópok egy részén van csatornánként két forgatógomb, mellyel bármikor tudjuk állítani a függőleges érzékenységet (*Scale*) valamint a csatorna aktuális pozícióját (*Position*).

Az időalap beállításait a *Horizontal* kezelőszerv-csoportban érhetjük el, itt tudjuk előhozni a hozzá tartozó menüt, valamint bármikor átállíthatjuk a skálát és a vízszintes pozíciót. Az indítójelet a *Trigger* beállításoknál tudjuk konfigurálni. E mellett számos további menüt hozhatunk elő, az oszcilloszkóp aktuális tulajdonságainak megfelelően. Mivel az oszcilloszkóp gyártók folyamatosan új funkciókkal jönnek elő, ezért mindig érdemes az oszcilloszkóphoz adott kezelési útmutatót elolvasni, hogy kihasználhassuk oszcilloszkópunk minden képességét. A jegyzetben részletezett funkciók elsősorban a Tektronix TDS2002C [21]típusú oszcilloszkópra vonatkoznak.

Az oszcilloszkóp bemenete és a mérőfejek

Az oszcilloszkópra a jeleket egy BNC csatlakozón keresztül vezethetjük, vagy koaxiális kábel, vagy oszcilloszkóp mérőfej segítségével. A multiméterekhez hasonlóan az oszcilloszkóp bemenete sem ideális, bemenő impedanciája befolyásolhatja a mérendő jeleket.



3.5. ábra: Az oszcilloszkóp bemenetének helyettesítő képe

Alacsony frekvenciák esetén a bemenet ohmikus ellenállása számít. Ez általában 1 MΩ. Nagy sebességű oszcilloszkópok esetén van lehetőség 50 Ω-os bemenő impedancia választására is. Ez pontosan megegyezik a BNC kábel hullámimpedanciájával, így elkerülhetők a jelvisszaverődések. Ahogy nő a frekvencia úgy nő a kapacitív tag szerepe is a bemeneti impedanciában.

3.6. ábra: Oszcilloszkóp mérőfej

Az oszcilloszkóp mérőfej fő szerepe, hogy a mérendő jelet jeltorzulás nélkül elvezesse az oszcilloszkóphoz. Vége egy kampóban végződik, mely segítségével könnyű a rögzítése. Szükség esetén ez a kampó eltávolítható, ekkor egy hegyes csúcs áll rendelkezésre. Minden mérésnél fontos a mérőfej megfelelő földelése, erre szolgál a kábelből oldalirányba kivezető krokodilcsipesz.

A mérőfej leoszthatja a mért jelet, ezáltal növelni tudja azt az impedanciát, amivel csökkenti a jelforrás terhelését. Egy 10x osztás esetén az impedancia 10 MΩ lesz, valamint a kapacitás is lecsökken, így a sávszélesség nő. Cserében a mért jel is tizedére csökken. Nagyobb frekvenciájú méréseknél, valamint érzékeny pontok mérésénél mindenképp érdemes 10x vagy nagyobb osztást használni. A mérőfejek jelentős része egy kapcsolót is tartalmaz, mellyel a mérőfej osztása módosítható. Ha a mért jelet leosztjuk, az oszcilloszkóp bemenetén lévő kapacitás miatt torzulhat a jelátvitel. Hogy ezt kompenzáljuk, a mérőfejet kalibrálni kell, vagyis a benne lévő kompenzáló változtatható kapacitásokat a megfelelő értékűre kell hangolni. A mérőfej kalibráció pontos menete az oszcilloszkóp, valamint a mérőfej útmutatójában található meg.

Nagy frekvenciájú méréseknél kritikus hogy milyen frekvenciáig lehet használni a mérőfejet. Az egyszeres osztású mérőfejek általában néhány MHz-ig használhatók csak, sok oszcilloszkóp a névleges sávszélességet csak a mellékelt mérőfejek 10x osztási aránya mellett biztosítja.

Magasabb kategóriájú oszcilloszkópok esetén használhatunk aktív mérőfejeket is, melyek egy beépített elektronikával biztosítják a megfelelő méréstartományt és jelátvitelt. Ezek egy speciális csatlakozóval csatlakoznak az oszcilloszkóphoz, mely a szokásos BNC kábel mellet extra érintkezőkön keresztül biztosítja a mérőfej tápellátását valamint a mérőfej és az oszcilloszkóp közötti kommunikációt. Minden egyes oszcilloszkóp gyártó más szabványt használ a mérőfejei számára (gyakran egy gyártón belül is vannak különbségek), így azok egymással nem kompatibilisek.

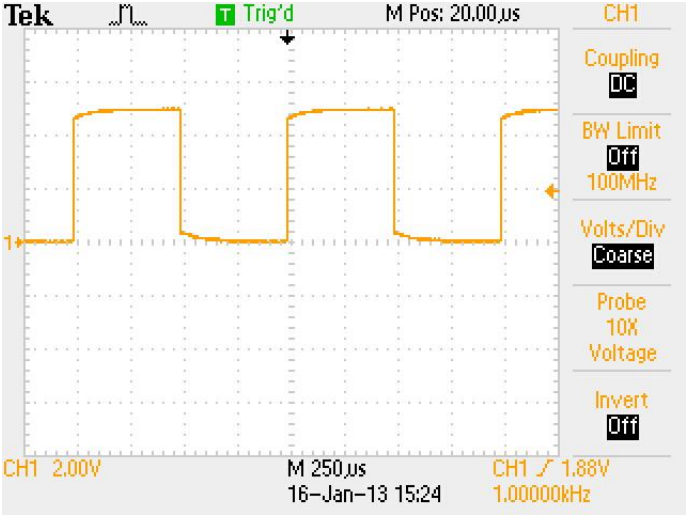
3.7. ábra: Az oszcilloszkóp képernyője mérés közben

Üzem közben az oszcilloszkóp képernyőjén számos információ jelenik meg. Ezek közül a legfontosabbak:

1. A mért jelek. A görbe színe megfelel a kiválasztott csatorna színének. Célszerű ennek megfelelően kiválasztani a mérőfejeket is, melyeket szintén meg lehet jelölni egy-egy színes gyűrűvel.
2. Az adott csatornához tartozó nulla szint
3. A trigger állapota (triggerelve, várakozik a triggerre, nincs trigger, mérés szüneteltetése)
4. Az aktuális trigger feszültség
5. Az egyes csatornák beállításai (érzékenyég: mekkora egy nagy négyzetnek az értéke, ez a képernyő nyolcada szokott lenni)
6. A vízszintes eltérítés beosztása (ennyi idő egy vízszintes beosztás, ez a teljes kijelzett idő tizede szokott lenni)
7. Trigger forrása és beállítása
8. Helyi menü

A digitális oszcilloszkópok egy hasznos szolgáltatása az automatikus beállítás (*Auto Set*), mely segítségével az esetek többségében helyesen beállítja az oszcilloszkópot, hogy jól láthassuk a kiválasztott jeleket. Az esetek egy részében (ritkán látható események, különleges jelek) ez a funkció nem vezet eredményre. Az *Auto Set* gomb mellett van általában a *Run/Stop* gomb, mellyel felfüggeszthetjük a mintavételezést, ekkor az utolsó mérés eredményét elemezhetjük, valamint a *Single* gomb, mellyel csak egy esemény mérését végezzük el, utána az oszcilloszkóp megáll.

Függőleges erősítő beállításai



3.8. ábra: A függőleges erősítő menüje

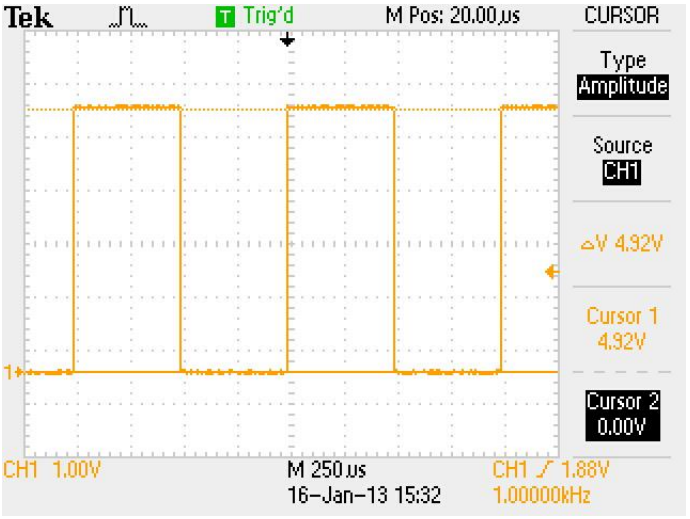
Az kiválasztott csatornához tartozó gombot megnyomva megjelenik a függőleges erősítő menüje. Újból megnyomva a gombot ki lehet kapcsolni az adott csatornát.

A menüben lévő elemek:

- *Coupling*: a bemenő erősítő csatolása. DC választás esetén a bemenő jel változtatás nélkül kerül a függőleges erősítőbe. AC csatolás esetén az egyenfeszültségű komponens leválasztásra kerül. Ezt az opciót ritkán használjuk, leginkább csak akkor, amikor a DC választás esetén nem látjuk a kívánt jeleket, például, ha egy kis méretű váltakozó jelet kívánunk megvizsgálni egy nagy egyenfeszültségű jelen. AC csatolás esetén a lassú jelek torzulnak.
- *BW limit*: sávszélesség korlátozása. A sávszélesség csökkentésével csökkenthető a jelen lévő zaj.
- *Volts/Div*: A skála forgatógomb funkcióját változtathatjuk meg a durva beállítás (alapbeállítás) és a finom beállítás között.
- *Probe*: mérőfej beállításának kiválasztása. Ez a beállítás meg kell, hogy feleljen az aktuális mérőfej beállításnak, különben hibás lesz a leolvasott érték.
- *Invert*: jel invertálása (-1-el való szorzása)

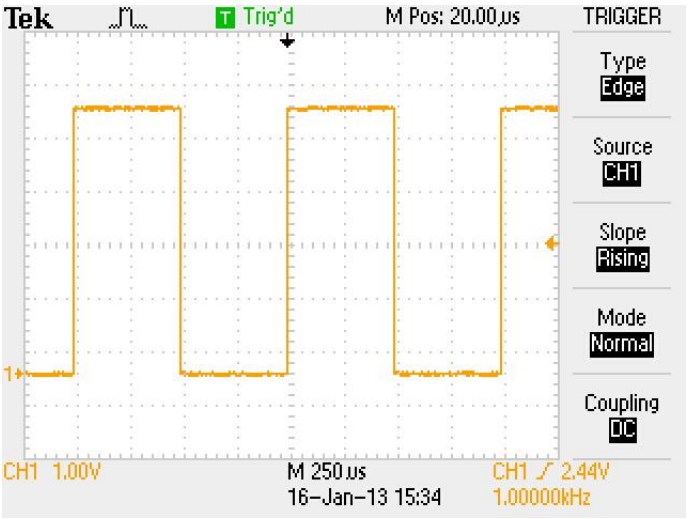
Amennyiben helyes a mérőfej beállítása az oszcilloszkóp kijelzi a függőleges beosztás mértékét. Ez alapján egyszerűen leolvashatjuk a mért feszültségeket. AC állás esetén nem ismerjük, hogy hol van a jel nulla szintje.

Szükség esetén aktiválhatunk kurzorokat is, így könnyebben le tudjuk olvasni a kívánt értékeket.



3.9. ábra: Feszültség leolvasása kurzorok segítségével

A trigger beállítások és a vízszintes eltérítés



3.10. ábra: A trigger menü

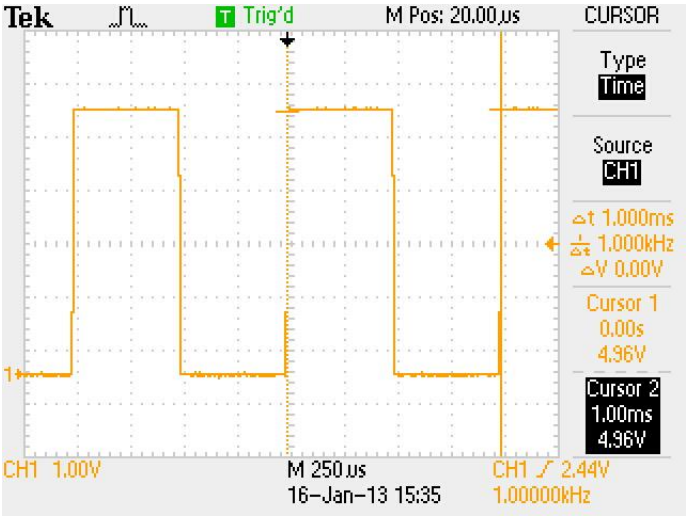
Hogy a számunkra fontos eseményeket regisztrálhassuk és megjeleníthessük, a trigger beállítása alapvető fontosságú. A trigger menü a következő opciókat tartalmazza (beállításoktól függhet, hogy mely opciók aktívak):

- *Type* – trigger típusa
  - *Edge* – élre triggerelt (alapértelmezett érték)
  - *Video* – szabványos videojelek vizsgálatakor alkalmazzuk
  - *Pulse* – impulzusra triggerelt
  - *Egyéb* – megfelelő oszcilloszkóp esetén további eseményekre is van lehetőség triggerelni, pl. akár logikai eseményekre is
- *Source* – triggerjel forrása
- *Slope* – felfutó vagy lefutó élre triggereljen az oszcilloszkóp
- *Mode* – trigger üzemmód
  - *Auto* – ha nincs triggerjel, akkor az oszcilloszkóp belső időzítése vezérli a mérést. Nagyon lassú mérések esetén az oszcilloszkóp átkapcsol *Scan* üzemmódba, és a trigger áramkör ilyenkor inaktív

    - *Normal* – csak akkor történik mérés, ha van is triggerjel. Ritkán előforduló események esetén ezt érdemes választani
- *Coupling* – a triggerjel csatolása. A szokásos opciók mellett itt van lehetőség különböző szűrésekre is.

További hasznos funkciók:

- *Set to 50%* – a trigger szintet beállítja a jel közepére, ezáltal stabilizálhatjuk a kijelzett képet
- *Force Trig* – mérés manuális indítása
- *Trig View* – a triggerjel megjelenítése
- *Holdoff* – a Horizontal menüben elérhető beállítás, megadja, hogy mennyi ideig várjon két mérés között

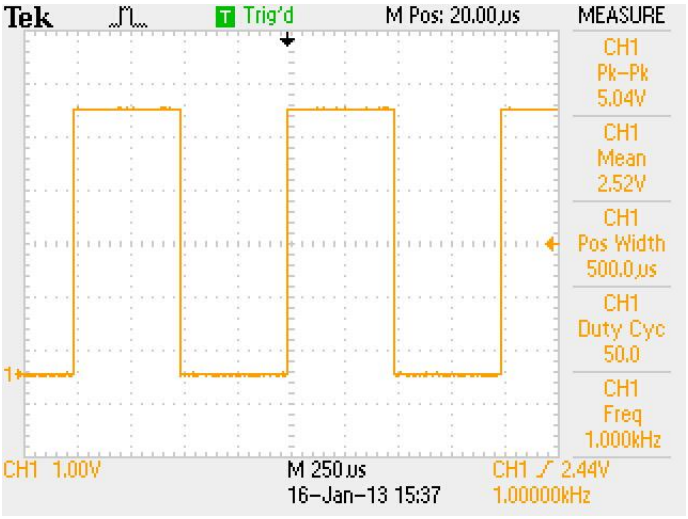


3.11. ábra: Idő leolvasása kurzorok segítségével

Automatizált mérések

A digitális oszcilloszkópok képesek arra, hogy automatikus méréseket végezzenek el a mért jelen, és ezeket megjelenítsék. Egyszerre több ilyen mérést elvégezhetünk. Ezt a funkciót a *Measure* menüben érhetjük el. Tipikus mérések:

- *Freq* – frekvencia
- *Period* – periódusidő
- *Mean* – a jel középértéke
- *Pk-Pk* – csúcstól csúcsig amplitúdó (a zajt is belemérve)
- *Cyc RMS* – egy jelperiódusra mért effektív érték
- *Min, Max* – minimum és maximum érték
- *Phase* – fázisszög mérése. Fázisszöget két kiválasztott jel között mérhetünk



3.12. ábra: Példa mérésre

További funkciók

Digitális oszcilloszkópok esetén különböző mintavételezési beállítások állnak rendelkezésre (*Aquire* menüpont). Az egyik hasznos opció, ami a zaj csökkentésére alkalmazhatunk, az az átlagolás. Ha pedig nagyon gyors jeleket (*glitch*) szeretnénk detektálni akkor a *Peak Detect* opció hasznos számunkra. Ilyenkor az oszcilloszkóp a lehető leggyorsabban mintavételez, függetlenül a beállított mintavételezési frekvenciától, ezáltal képes regisztrálni a gyors, rövid ideig tartó csúcsokat is.

A matematikai menüben (*Math*) különböző műveleteket hajthatunk végre a mért jelek között, és ezeket megjeleníthetjük. Itt tudunk spektrumot is számolni.

A *Save/Recall* menü segítségével tudjuk elmenteni a mért adatokat. Lehetőségünk van elmenteni a mért jelet eltárolni különböző formátumokban, a beállításokat, valamint a kijelzett képet is. Utóbbinál választhatunk festéktakarékos üzemmódot is. E mellett lehetőség van jelalakok (*Reference*), valamint a beállítások mentésére és betöltésére is.

Hasonlóan az analóg oszcilloszkópokhoz XY módot is ki tudunk választani (*Display/Format*). Oszcilloszkóptól függően XY módban különböző korlátozások léphetnek érvénybe, pl. nem feltétlenül lehet állítani a mintavételi sebességet, vagy nem lehet elmenteni az adatokat.

Oszcilloszkópos mérések esetén a következőkre kell figyelni:

- A helyes méréshez az oszcilloszkóp mérőfejét megfelelően földelni kell. A földelés csatlakozását körültekintően kell megválasztani, különben könnyen okozhatunk olyan rövidzárlatot, mely vagy a vizsgált berendezés, vagy az oszcilloszkóp meghibásodásához vezet.
- Az oszcilloszkópok bemenetei nem differenciálisak, nem lehet feszültségkülönbséget mérni (kivéve, ha speciális, differenciális mérőfejet használunk).
- Mivel az oszcilloszkóp mintavételezi a mért jelet, lehetséges a mintavételi tétel megsértése. Ekkor a mért jel akár teljesen más frekvenciájúnak is tűnhet, mint amilyen az valójában. Az oszcilloszkópok *peak detect* üzemmódja segíthet elkerülni a mintavételi tétel sérüléséből adódó hibákat.

Feladatok

1. feladat

Az oszcilloszkóp útmutatójának megfelelően kalibrálja az oszcilloszkóphoz adott mérőfejeket!

2. feladat

Vizsgálja meg a kalibráló jelet, majd mentse el az összes adatot pendrive-ra! Importálja az adatokat, majd jelenítse meg Excelben! (Az excel alapértelmezett beállításai nem feltétlenül megfelelőek.)

3. feladat

Vizsgálja meg, hogy milyen jel jön ki a hangkártya kimenetén a hangszórók tesztelésekor! A teljes jelet jelenítse meg (látszódjon mindkét csatorna, a jel elejétől a végéig, ez kb. 2,5 másodpercig tart)! Milyen trigger beállításokat kell használni?

4. feladat

Készítsen LabVIEW-ban olyan programot, mely két azonos frekvenciájú, de különböző amplitúdójú és fázisú jelet ad ki (ehhez a *Generate Sound* példaprogram ad segítséget)!

Mérje meg a jelet oszcilloszkóppal! Az oszcilloszkóp „*Measure*” lehetőségeit felhasználva jelezze ki a jelek frekvenciáját, amplitúdóját és fázisszögét!

Mentse el a jeleket, majd ábrázolja a jeleket Excelben!

5. feladat

Vizsgálja meg a mintavételi tétel megsértésének következményeit oszcilloszkópon! Ehhez használjon jelgenerátort vagy a hangkártya kimenetét!

6. feladat

Egy négyszögjel segítségével vizsgálja meg hogyan hat a mérőfej 1x és 10x állása a sávszélességre!

Hogyan hat a jelforrás impedanciája a sávszélességre? (Ehhez használjon egy soros ellenállást!)

4. Műszerek vezérlése LabVIEW környezetből

A digitális műszerek, valamint az informatika fejlődésével egyre gyakrabban merül fel az az igény, hogy a műszereinket számítógéphez kössük, ezáltal gyorsíthassunk az adatok feldolgozásának folyamatán. A műszerek számítógépre való kötésével egyben lehetőségünk van automatizált tesztrendszereket is létrehozni, melyek automatikusan, külső beavatkozás nélkül végeznek el akár több ezer mérést a vizsgált rendszeren. Az interfészek és internet segítségével akár távméréseket is végezhetünk tetszőleges távolságokból is.

A teljesség igénye nélkül a következő műszereket vezérelhetjük számítógépről:

- Hagyományos mérőműszerek számítógépes interfész opcióval, ilyenek pl. a multiméter, oszcilloszkóp, tápegységek. A legelterjedtebb interfészek: USB port, Ethernet valamint GPIB port.
- Adatgyűjtők és univerzális műszerek: több analóg bemenettel és kimenettel rendelkeznek, valamint digitális I/O vonalakkal. A megvalósított funkciót a számítógépen futó program határozza meg. A leggyakrabban USB interfésszel vannak ellátva.
- Számítógépes mérőkártyák: ahogy nevük mondja a számítógép PCI vagy PCIe portjába helyezhetők. Előnyük a nagy adatátviteli sebesség.
- Moduláris műszerek esetén a műszer funkcionalitását elsősorban a műszerbe épített modulok határozzák meg. Ezek lehetnek analóg bemenetek és kimenetek, digitális I/O, valamint összetettebb műszerfunkciókat megvalósító kártyák, pl. multiméterek, jelgenerátorok, vektor-jel analizátorok, adattárak. A műszerplatformok egy része saját adatfeldolgozó egységgel (processzorral) rendelkezik, melyek számítógéptől független mérést és vezérlést tesznek lehetővé (pl. cRIO, PXI platform). Leggyakoribb interfészek: USB, Ethernet.

Ahhoz, hogy egy műszert LabVIEW programból kezeljünk, számos eszközre van szükségünk. Egyrészt, szükségünk van egy meghajtóprogramra (*driver*) ahhoz, hogy az operációs rendszer felismerje a műszert, és lehetőséget teremtsen a vele való kommunikációra. E mellett szükségünk van egy külön meghajtóprogramra (*instrument driver*) ami lehetővé teszi, hogy a LabVIEW programok hozzáférjenek, és magas szinten vezéreljék a műszert.

A műszer gyártójától és típusától függően számos megoldás van a műszer elérésére.

Legjobb esetben a gyártó egy LabVIEW kompatibilis *instrument driver*t biztosít, így azt már kész megoldásként felhasználhatjuk a saját programjainkban. A driver használatához általában biztosítanak példaprogramokat is, melyek bemutatják, hogyan kell használni az adott műszert. A driverek egy része szabványos kezelést tesz lehetővé, így könnyebb biztosítani a műszerek közötti kompatibilitást. A driverek egy része csak egy LabVIEW *library* fájlból (llb-ből) és néhány kiegészítő fájlból áll, komolyabb driverek már csak telepítőprogrammal érhetők el.

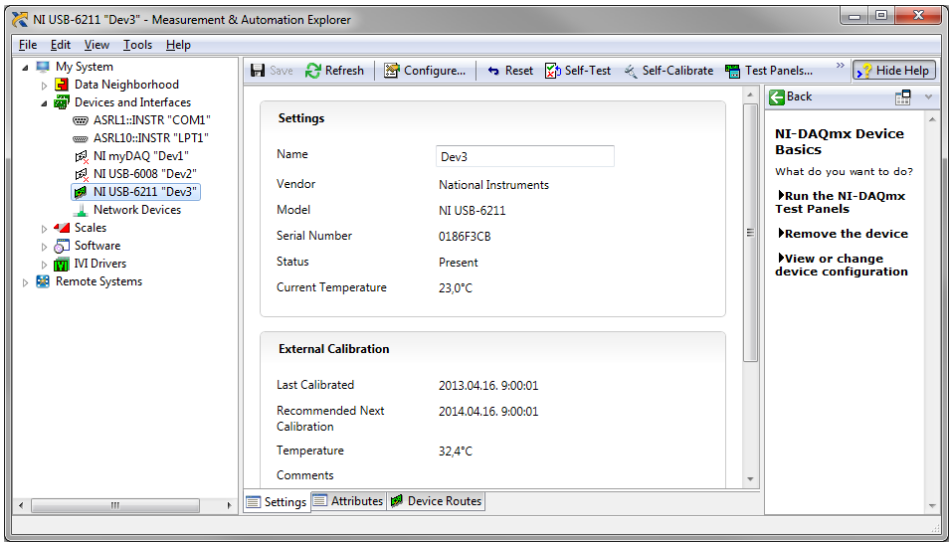
Ha nincs LabVIEW driver, még mindig lehet, hogy a gyártó biztosított megfelelő programozói interfészt más programnyelvek számára (C++, C#). Az ezekhez a függvénykönvtárakhoz tartózó függvényeket a *Call Library Function* segítségével meghívhatjuk LabVIEW-ból, így saját drivert hozhatunk létre.

Az esetek egy részében a gyártó nem biztosít semmilyen drivert, viszont a kommunikációs protokoll részletes leírása elérhető. Így saját magunk tudjuk elkészíteni a megfelelő meghajtóprogramot, bár ez már komolyabb tudást és gyakorlatot igényel.

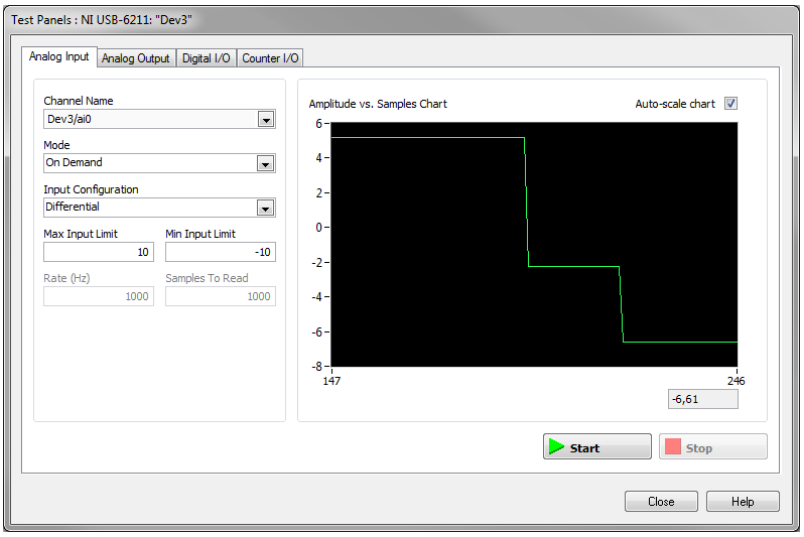
Sajnos vannak olyan esetek is, amikor nincs dokumentálva a kommunikáció, és az adott eszköz csak a hozzáadott egyedi szoftverrel vezérelhető. Ezeket az eszközöket érdemes elkerülni, mivel nem fogjuk tudni beintegrálni a saját szoftvereinkbe.

Számos elterjedt szabvány létezik a műszerek számítógépről való kezelésére (pl. a *VISA* szabvány). Azonban ezekkel érdemes vigyázni, ugyanis előfordulhat, hogy az egyes gyártók különböző módon implementálják ezeket a szabványokat, így műszereik nem lesznek teljesen kompatibilisek egymással. Ezért a driver telepítése során oda kell figyelni, hogy milyen forrásból származnak a driverek. Amennyiben LabVIEW környezetben szeretnénk használni a műszereket, akkor sok esetben az összes *instrument driver*-t a LabVIEW honlapjáról érdemes letölteni [11], így várhatóan kevesebb inkompatibilitási probléma lép fel. Amennyiben nehézségekbe ütközünk célszerű felkeresni a műszer gyártóját vagy forgalmazóját.

A számítógéphez csatlakoztatott eszközöket és műszereket az *Measurement & Automation Explorer* (*NI MAX*) segítségével lehet áttekinteni. A program segítségével tudjuk azonosítani és konfigurálni az egyes műszereinket, valamint tesztelhetjük is őket a *Test Panels* eszköz segítségével.



4.1. ábra: A *Measurement & Automation Explorer* (*MAX*) ablaka



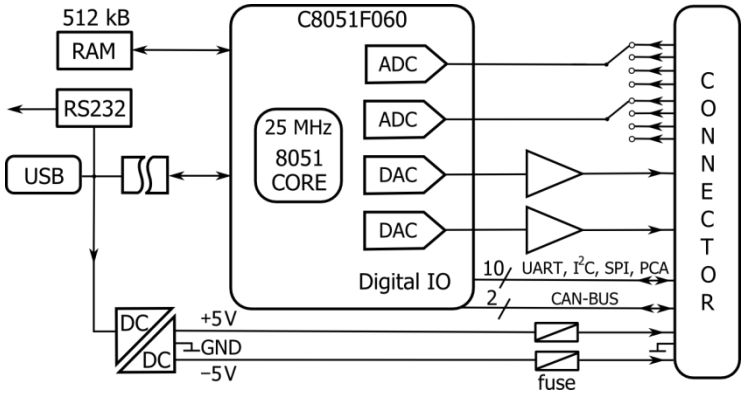
4.2. ábra: Egy műszer tesztelése a *MAX Test Panels* segítségével

A (virtuális) soros port és a rá kapcsolt műszerek vezérlése

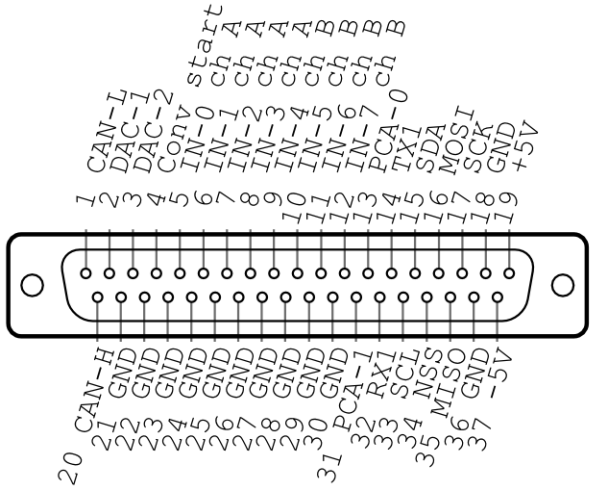
Az USB port megjelenése előtt egy periféria számítógéphez csatlakoztatásának legegyszerűbb módja az RS232 soros port volt. Az USB port megjelenésével ezeket a műszereket nagyon könnyen lehet adaptálni az új szabványhoz, a művelet mindössze egy kiegészítő áramkört igényel (USB-soros konverter), így a korábbi műszereket ugyanúgy tovább lehet használni. Bár a számítógép már nem egy soros porton keresztül kapcsolódik a műszerhez, a felhasználók továbbra is soros portként hivatkozhatnak rá, ugyanúgy kezelhetik a programokból, ezt hívjuk virtuális soros portnak.

Ahhoz, hogy egy virtuális soros portos műszert használjunk, szükség van az USB-soros konverter meghajtóprogramjára (ezt az operációs rendszer igényli), a LabVIEW-hoz szükséges *VISA* driverre és opcionálisan egy megfelelő LabVIEW driverre (ami lehet egyetlen egy llb is). A kommunikáció során két adatfolyam van, az egyik, amit az eszköznek küldünk, a másik, amit az eszköztől kapunk. A kommunikáció lehet bináris is és szöveges is, a LabVIEW rutinok String-eket használnak a rutinok bemeneteként és kimeneteként.

A LabVIEW-ban a soros port közvetlenül is manipulálható, így akár magunk is készíthetünk LabVIEW drivert. A következőben egy példát mutatunk be a gyakorlat során alkalmazott nyílt forrású MA-DAQ műszert felhasználva [24].



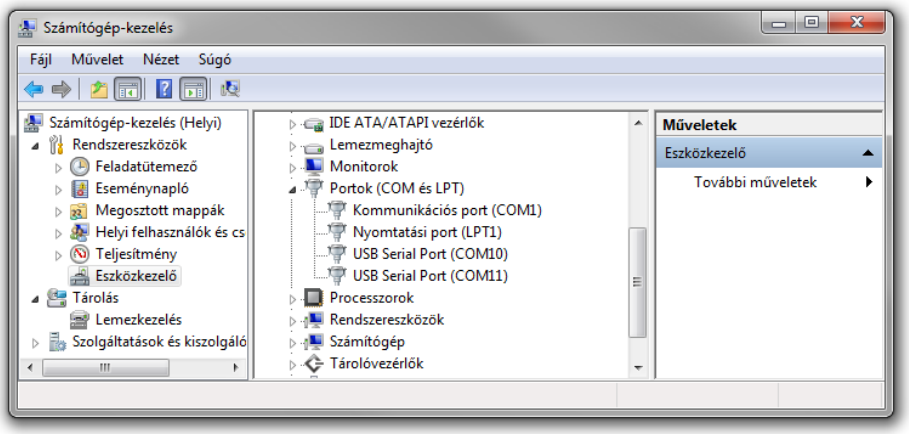
4.3. ábra: MA-DAQ blokkvázlata



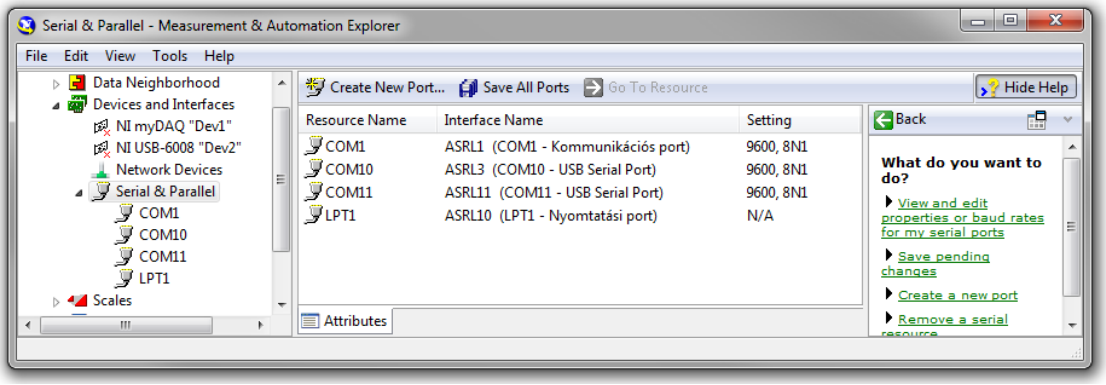
4.4. ábra: A MA-DAQ csatlakozójának bekötése

Kommunikáció inicializálása

A MA-DAQ az FTDI által gyártott FT2232HL integrált áramkört használ USB-soros interfészként [25]. Amennyiben korábban nem volt ilyen eszköz csatlakoztatva a számítógéphez fel kell telepíteni a Windows drivert. Ezután csatlakoztatva a készüléket a Windows fel fogja ismerni, és megjelenik az eszközkezelőben. Minket elsősorban az érdekel, hogy milyen azonosítót rendel hozzá a Windows az aktuális eszközhöz. A MA-DAQ esetén két virtuális soros port is megjelenik, az egyiken keresztül kommunikálhatunk a műszerrel, a másik pedig felhasználható szabványos RS-232 portként egy tetszőleges külső műszerrel való kommunikációhoz. Sajnos a két port sorrendje nem meghatározott, véletlenszerűen lesz az egyik a kisebbik sorszámú, a másik pedig a magasabb sorszámú (az esetek nagy részében maga a műszer lesz az alacsonyabb sorszámú). A MA-DAQ kommunikációja bináris, ezért a LabVIEW környezetben is ennek megfelelően kell beállítani a port kezelését.

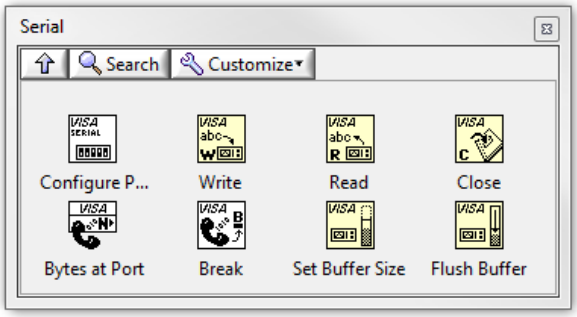


4.5. ábra: A két virtuális soros port az eszközkezelőben



4.6. ábra: A felismert soros portok az *NI Measurement & Automation Explorer* (*MAX*) ablakában

A soros portokhoz a LabVIEW is hozzárendel egy saját azonosítót (*VISA resource name*), mely segítségével a *VISA* driveren keresztül használható. Ez az esetek döntő többségében megegyezik az eszközkezelőben megjelenített sorszámmal, de előfordulhat, hogy a számozás elcsúszik. Korábbi verziókban más jelölést használtak a soros portok jelölésére, így a COM10 ARSL10 ként jelent meg.

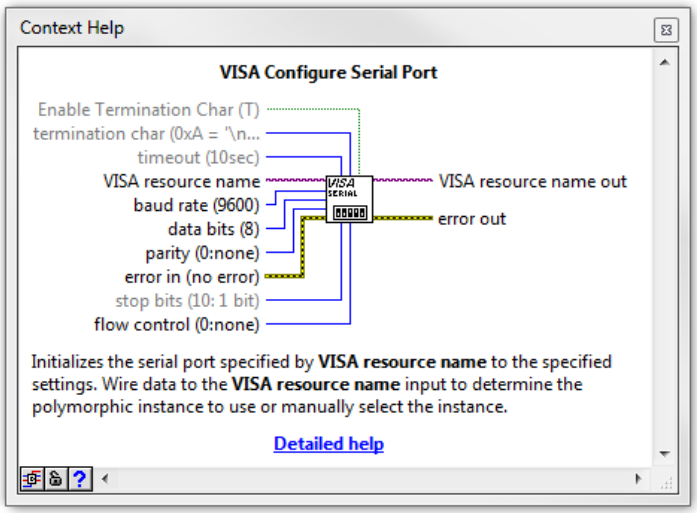


4.7. ábra: A soros port kezelésére szolgáló rutinok

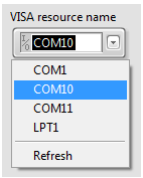
A LabVIEW-ban a soros portot kezelő rutinok az *Instrument IO / Serial* palettán érhetők el.

A soros portot a használat előtt megfelelően be kell állítani majd megnyitni. A MA-DAQ-hoz szükséges beállítások (részletesebben a soros portról a 8. fejezetben):

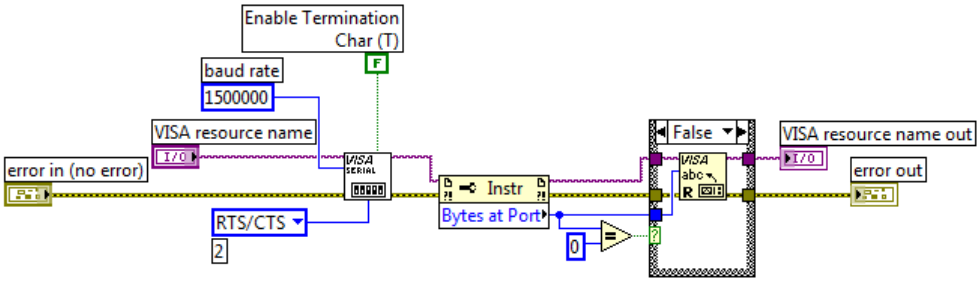
- *Enable Termination Char* – false érték (a true érték soronkénti szöveges kommunikáció beolvasásánál hasznos, bináris kommunikációnál hibához vezet)
- *VISA resource name* – a soros port azonosítója
- *timeout* – a 10 s-os alapbeállítás megfelelő. Ha 10 másodpercig nem érkeznek meg a várt adatok, akkor az olvasás hibával tér vissza.
- *baud rate* – a kommunikáció sebessége: 1 500 000 bit/s
- *data bits* – a 8 bit-es alapbeállítás megfelelő
- *parity* – a MA-DAQ nem használ paritásbitet
- *stop bits* – az alapbeállítás megfelelő
- *flow control* – a szükséges beállítás RTS/CTS



4.8. ábra: A soros port konfigurálása



4.9. ábra: A megfelelő soros port kiválasztása. Ha az adott port nem látszik (pl. mert az eszköz a LabVIEW indítása után lett csatlakoztatva) a *Refresh* gomb segítségével lehet a listát frissíteni. Ez a funkció nem feltétlenül működik helyesen, ha a program fut.



4.10. ábra: Soros port inicializálása

A szükséges beállítások majd a soros port megnyitása után illik törölni a soros port bufferében esetleg meglévő adatokat. Ha korábban megszakadt volna a kapcsolat, és a műszer küldött volna még adatokat, azok ekkor törlődni fognak, így van esély a kommunikáció helyreállítására. Ha a műszer véletlenül nem megfelelő állapotban van (pl. egy mérést végez), akkor ez önmagában nem elegendő (el kell küldeni a megfelelő leállító parancsot, majd akár többször újra próbálkozni a kapcsolatfelvétellel). A problémák elkerülése érdekében a kommunikációt mindig megfelelően állítsuk le a programjaink végén, valamint sose használjuk az *Abort* gombot a programok leállítására.

Ha a soros port már konfigurálva van és meg van nyitva, a program többi részében használható. A soros port lezárásáig más program nem fér hozzá a porthoz. Ezért célszerű azt lezárni a programunk végén a *Close* VI segítségével

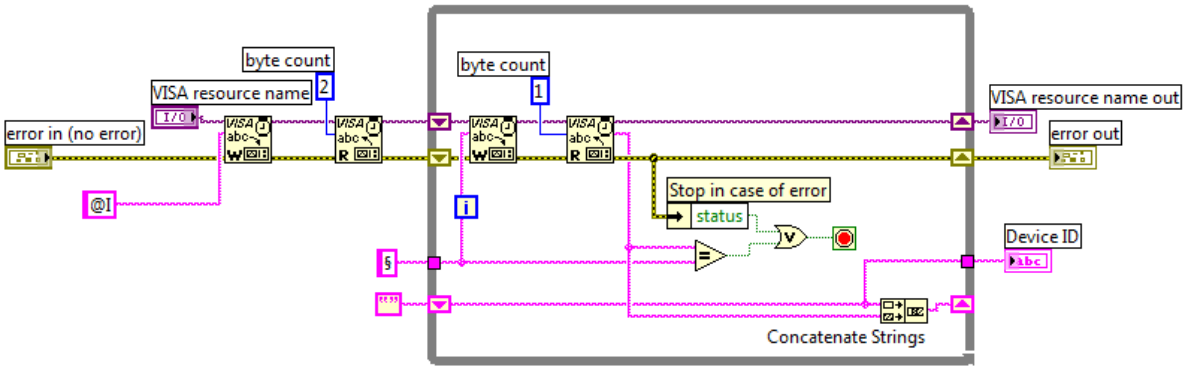
Alacsony szintű kommunikáció

A MA-DAQ számára különböző parancsokat küldhetünk, melyeket azonnal végrehajt. Minden parancs egy ’@’ karakterrel kezdődik, majd következik a parancs azonosítója (kis vagy nagy betű), majd a parancs paraméterei binárisan. A MA-DAQ az összes kapott bájtot visszaküldi a számítógépnek, innen is tudhatjuk, hogy az az adatokat feldolgozta, és kész a következő parancs fogadására. Ha a MA-DAQ alapállapotban van és kész fogadni a parancsokat minden egyes fogadott karaktert visszaküld, akkor is, ha az nem érvényes parancs.

Eszköz azonosító szöveg beolvasása

A MA-DAQ kérésre elküldi a saját azonosító szövegét. Az ehhez használt parancs a ’@I’. A parancs leküldése után a MA-DAQ minden egyes további karakter után felküld egyet az azonosító szövegéből, egész addig, míg el nem fogy. Ezek után a leküldött karaktert változás nélkül megkapjuk.

Az azonosító szöveg beolvasásához tehát le kell küldenünk a parancsot, visszaolvassuk a parancs két bájtját, majd pedig addig küldünk egy kiválasztott karaktert (ami nem lehet benne az azonosító szövegben), míg ugyanazt a karaktert vissza nem kapjuk. A kapott karaktereket összefűzzük egy szöveggé, ez lesz az azonosító szöveg.



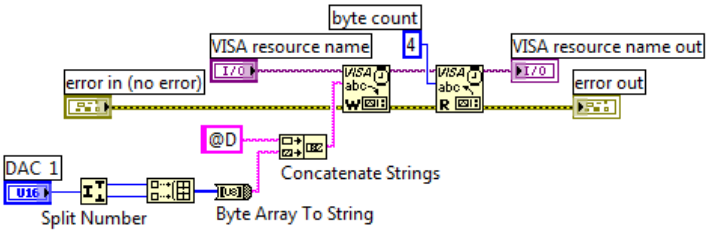
4.11. ábra: Az eszközazonosító szöveg beolvasása

Kimenő feszültség beállítása

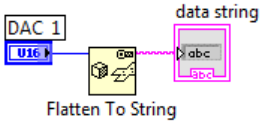
Az analóg (DAC) kimeneteket a következő két paranccsal lehet beállítani:

- '@D[B_1][B_0]' – DAC-1 feszültségének beállítása
- '@d[B_1][B_0]' – DAC-2 feszültségének beállítása

A [B_1][B_0] egy 16 bit-es előjel nélküli egész szám mely a kimenő feszültséget meghatározza. A kód 0 és 65535 között változhat, a 0 felel meg a -5 V-nak, a 65535 pedig az 5 V-nak (a kimenő feszültségek kis mértékben különbözhetnek ettől). Ahhoz, hogy egy 16 bit-es számot el tudjunk küldeni a soros porton, először fel kell daraboljuk 8 bit-es számokra, tömböt készíteni majd pedig a tömböt átalakítani szöveggé. A művelet során kritikus a változók típusa, különben hibás eredményt kapunk. A parancs leküldése után vissza kell olvassuk az összes leküldött adatot.



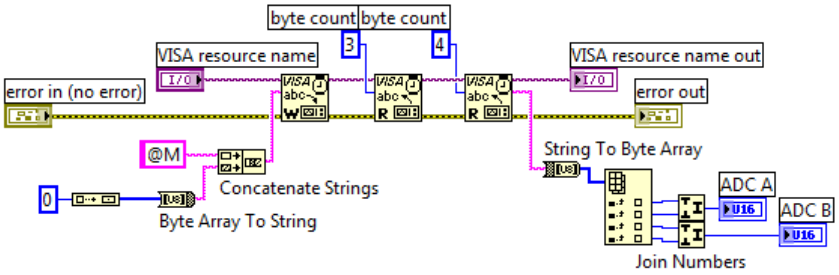
4.12. ábra: A DAC-1 feszültségének beállítása



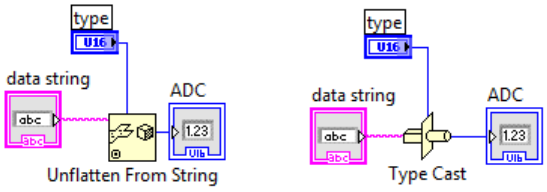
4.13. ábra: Alternatív módszer egy tetszőleges adat bináris string-gé konvertálásának. A módszer hátránya, hogy kevesebb lehetőségünk van az adatok manipulálására, vagy pl. 24 bit-es számok használatára.

Mért feszültség beolvasása

A MA-DAQ mérés során egyszerre két kiválasztott csatornát tud mintavételezni. A méréshez kiküldött parancs: '@M[A_0]', ahol A_0 az átlagok száma, válaszként pedig a következőket kapjuk: [CH-A_1][CH-A_0][CH-B_1][CH-B_0], vagyis a két bemenő csatorna 16 bit-es adatait kapjuk. A méréshez először el kell küldjük a parancsot, vissza kell olvassuk a leküldött parancs karaktereit, majd pedig a négy adatbájtot. Utóbbiakból lehet összerakni az ADC-k kódjait.



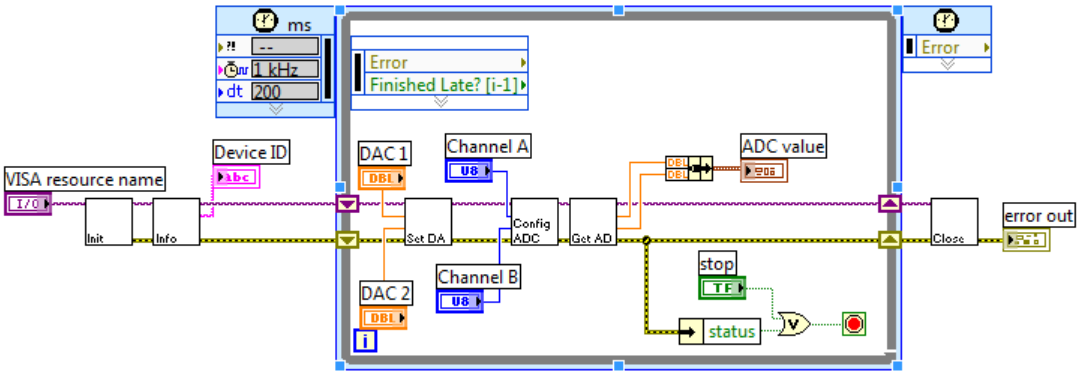
4.14. ábra: Az analóg bemenetek értékeinek beolvasása



4.15. ábra: Alternatív módszer az adatok konverziójára

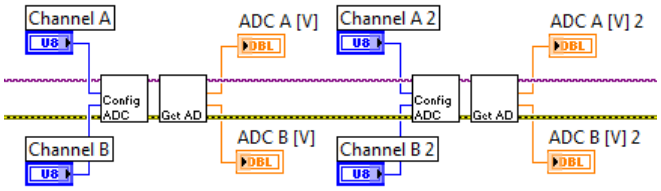
MA-DAQ library használata

A MA-DAQ-hoz letölthető a MA-DAQ.llb, melynek köszönhetően már nem kell alacsony szintű kommunikációhoz folyamodni a műszer használatához. Az llb-ben lévő példaprogram bemutatja hogyan lehet használni az alap funkciókat. Az eszközt először inicializálni kell, ekkor történik a soros port megfelelő konfigurációja és megnyitása. Ezután következik az eszközazonosító szövegének beolvasása. Ha ez a művelet sikertelen lenne, jelentheti azt, hogy nem a megfelelő soros portot választottuk ki. Ez után következik a mérést végző ciklus, ami *Timed Loop*-ot használ fel. Itt először megtörténik a kimeneti feszültségek beállítása, majd kiválasztunk két mért csatornát, melyeket beolvasunk, majd az eredményt megjelenítjük egy *Waveform Chart*-on. Hiba esetén vagy a stop gomb megnyomásakor a program leáll, ez után történik a műszer lezárása, jelen esetben a soros port lezárása.



4.16. ábra: MA-DAQ példaprogram

A példaprogramban jól látható, hogy a műszerre hivatkozó referencia sehol sem ágazik el, az egyes VI-ok egymás után vannak felfűzve. Ez a megoldás biztosítja a végrehajtás megfelelő sorrendiségét. Amennyiben ezt nem biztosítanánk, a műszer párhuzamosan kaphatna több parancsot és a kommunikáció felborulna.



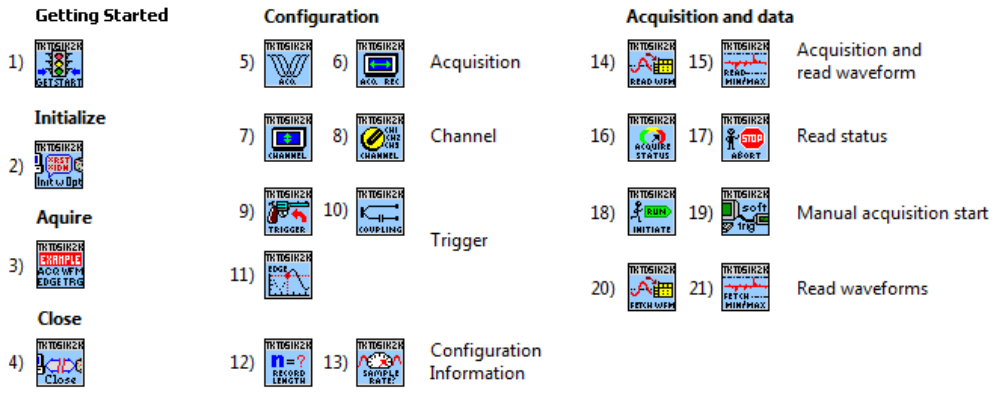
4.17. ábra: Ha több csatornát szeretnénk beolvasni, akkor egymás után kell végrehajtani a csatornaváltást és a feszültség beolvasást. A soros portra hivatkozó referencia biztosítja a végrehajtás megfelelő sorrendjét.

Oszcilloszkópok vezérlése LabVIEW környezetből

A legtöbb modern digitális oszcilloszkópot már számítógépről is lehet vezérelni. A következőben röviden a laboron használt TDS2002C oszcilloszkóp használatát mutatjuk be. Megjegyzés: a LabVIEW és a driver verziójától függően eltérések lehetnek az itt bemutatott példától.

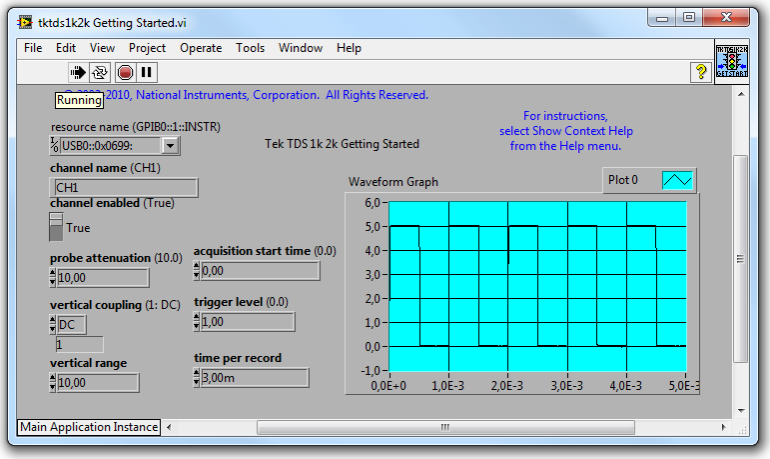
Ahhoz, hogy az oszcilloszkópot vezérelni tudjuk, szükség van a megfelelő meghajtóprogramokra. Az NI honlapján az *Instrument Driver Network* több találatot is ad megfelelő driverekre [11]. Ezek más és más szolgáltatásokat nyújtanak. A legtöbb funkcionalitást az *IVI instrument driver* szolgáltatja. Ahogy a driver oldalán meg van adva, a driver telepítése előtt további szoftvereket is telepíteni kell, az *IVI Compliance Package* megfelelő verzióját valamint a *VISA* drivereket. A driverek telepítése előtt érdemes rendszer-visszaállítási pontot készíteni, így szükség esetén újra lehet próbálkozni a telepítéssel.

Az oszcilloszkópokhoz mellékelt CD-ken lévő szoftvereket nem kell telepíteni (sőt, tapasztalatunk szerint a két CD-n lévő két különböző szoftver egymással inkompatibilis). A driver elemei elérhetők az Instruments Drivers palettán, illetve a következő helyen is: *C:\Program Files\National Instruments\LabVIEW 201x\instr.lib\tktds1k2k*. A könyvtár legfontosabb elemei a 4.11. ábrán vannak bemutattva. Az első oszlopban lévő 1) VI egy példaprogram (*tktds1k2k Getting Started.vi*), mellyel egyrészt lehet ellenőrizni, hogy sikerült-e kapcsolatot teremteni, másrészt a programból kiindulva elkészíthetjük saját alkalmazásunkat.

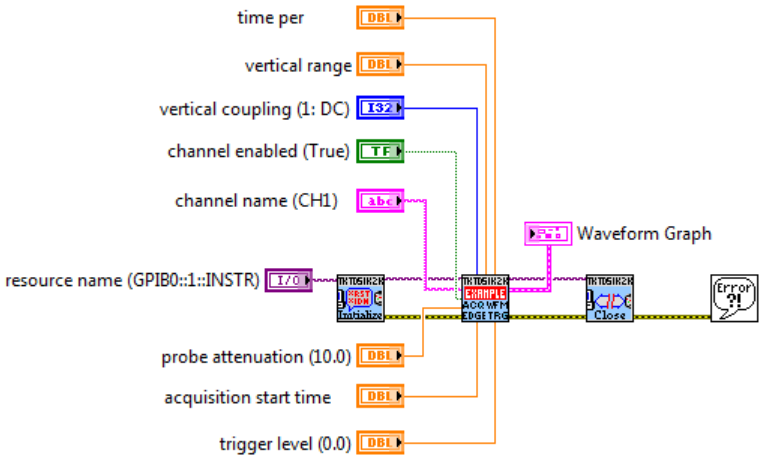


4.18. ábra: Az oszcilloszkóp kezelésére szolgáló legfontosabb VI-ok. Az alfejezetben lévő számozások erre az ábrára vonatkoznak.

A példaprogram futtatásához helyesen kell beállítani a paramétereket. Egyrészt ki kell választani az oszcilloszkópot (*resource name*), mivel USB-n keresztül csatlakozik az eszköz, azt az opciót válasszuk ki, ami USB-vel kezdődik. Az *Measurement & Automation Explorer*-ben lehet megnézni az azonosító-eszköz hozzárendeléseket, valamint emberibb *alias*-okat is rendelhetünk hozzájuk.

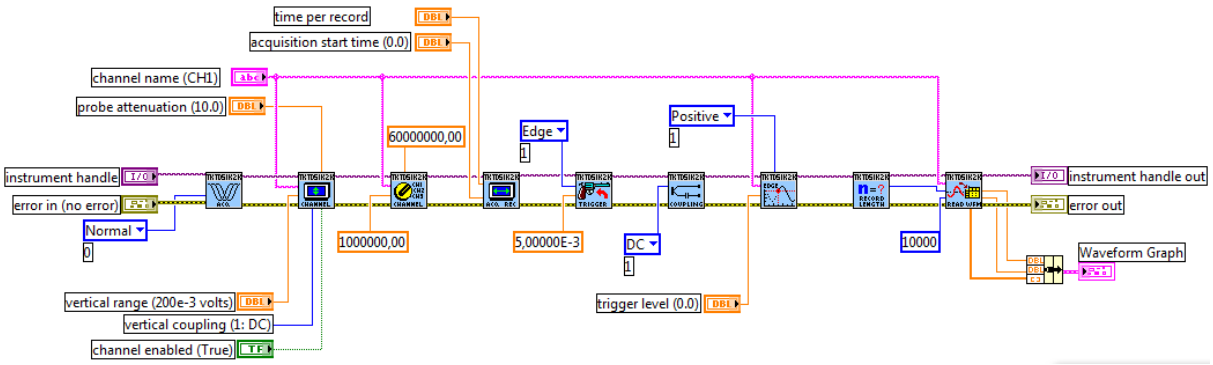


4.19. ábra: A példaprogram előlapja a kalibrálójel vizsgálata esetén



4.20. ábra: A példaprogram diagramja

Ahogy a 4.20. ábrán látható példaprogram először megnyitja a megfelelő erőforrást és inicializálja az oszcilloszkópot (2). Ezt követően elvégez egy mérést (3), majd a legvégén lezárja az eszközt (4). Érdekesség, hogy az inicializálás nem mindig sikeres, néha nem tudja alapállapotba hozni az oszcilloszkópot. Ezt az opciót kikapcsolva, a következőkben már helyesen lefut.



4.21. ábra: A mérést végző VI

A méréshez a program (3) vi belsje) először beállítja az oszcilloszkóp minden funkcióját, ezek után következik a triggerelt mérés. Mivel minden beállítást elvégez, az egész folyamat nagyjából nyolc másodpercet vesz igénybe. Ha ennél gyorsabban szeretnénk mérni, akkor.

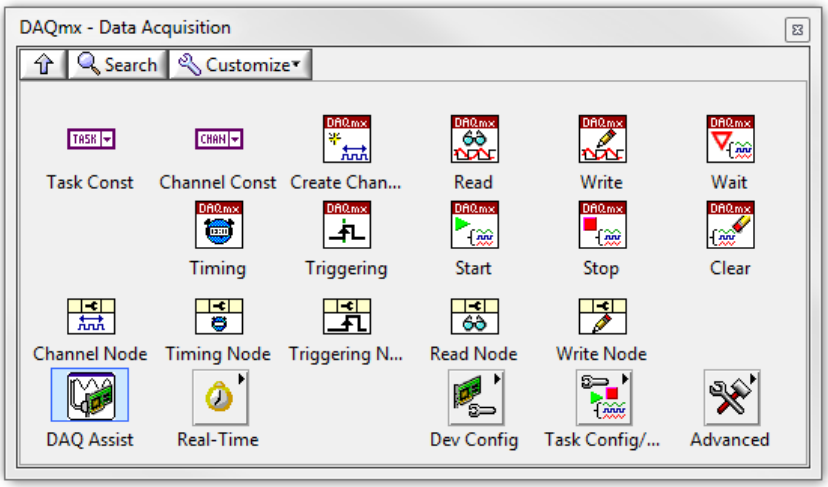
A mérést végző VI (3) először konfigurálja a mérés típusát (5) majd beállítja a mért csatorna szokásos mérési paramétereit (*vertical range*, *vertical offset*, engedélyezés, *probe attenuation* és *coupling*)(7) illetve a csatorna bemenő impedanciáját és a maximális bemeneti frekvenciát (8). A triggerelés konfigurációja után (9) be van állítva a trigger csatolása (10) valamint forrása, szintje és az él iránya (11). A (12) VI adja meg számunkra hogy egy-egy mérés hány pontból fog állni. A Read waveform (14) élesíti az oszcilloszkópot, mely ez után várakozni fog a megfelelő trigger jelre. Ezen VI futtatásakor nincs Auto trigger üzemmód, amennyiben nem következik be esemény a megadott időtartam alatt (*timeout*) hiba keletkezik. Egyébként beolvassa a mért adatsort.

A példaprogram egyetlen egy csatornát mintavételez egyszer, a teljes folyamat nagyjából nyolc másodpercig tart. Ha több mérést szeretnénk egymás után végezni, nagyobb frissítési rátával, akkor olyan programot kell készítsünk, ahol a mérést végző ciklusban csak a mérést végző subVI van benne (14). Ha több csatornánk szeretnénk mérni, akkor egyrészt minden csatornát konfigurálni kell, a mérést követően a többi csatorna által mért adatokat a Fetch waveform (20) segítségével olvashatjuk ki. Az adatgyűjtés élesítését manuálisan is végezhetjük (18), állapotát a (16) segítségével kérdezethetjük le. Triggerjel hiányában megszakíthatjuk a mérést (17), vagy generálhatunk egy szoftveres trigger eseményt (19).

DAQmx műszerek

A National Instruments által gyártott műszerek nagy részét egy egységes felületen keresztül lehet használni, a *DAQmx* könyvtár segítségével. Az, hogy egy-egy műszer pontosan milyen feladatok végrehajtására alkalmas, a műszer specifikációjában megtalálható. Néhány speciálisabb funkciót viszont ki kell próbálni használat előtt, hogy az adott műszer támogatja-e olyan formában, ahogy használni szeretnénk.

Számos további gyártó is úgy készíti a driverét, hogy az hasonlóan használható legyen, mint a DAQmx.



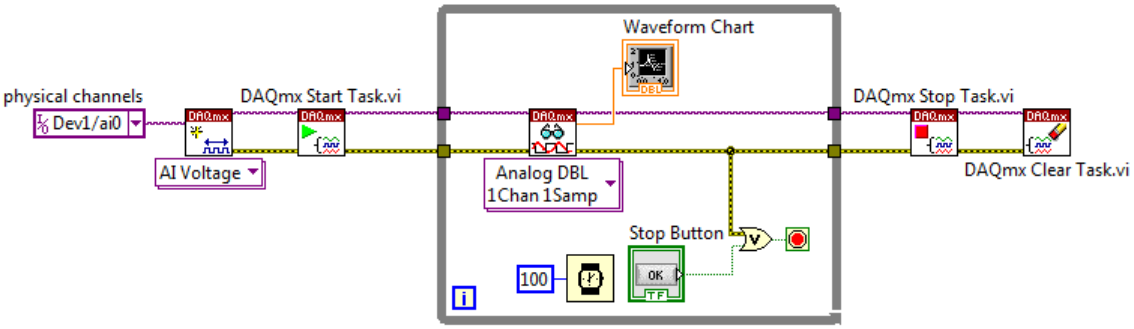
4.22. ábra: A *DAQmx* paletta (*Measurement IO* része)

A két legfontosabb fogalom:

- *Task* – egy mérési feladat, amit a műszer végrehajthat. Egy műszer akár több *Task* párhuzamos végrehajtására is alkalmas.
- *Channel* – egy fizikai vagy logikai csatorna, ami lehet analóg vagy digitális bemenet vagy kimenet. Ezzel határozzuk meg, hogy mely bemenetet vagy kimenetet szeretnénk mérni. Egy *Task*-hoz több fizikai csatornát is hozzárendelhetünk.

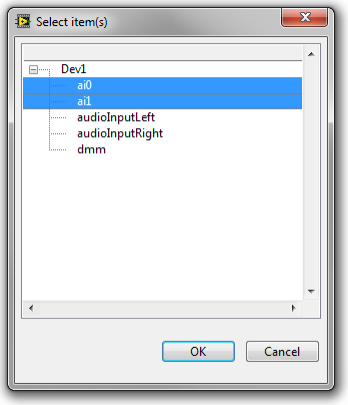
A mintavételezés időzítését tekintve a mérés lehet:

- *On demand / software timed*: a számítógépen futó program végzi az időzítést. A mérés időpontja nincs pontosan definiálva.
- *Hardware timed*: a mérőeszköz végzi a mintavételezés pontos ütemezését. Ennek köszönhetően a mintavételezési időpontok távolsága jól definiált. A mintavételi órajelet a mérés előtt lehet beállítani, ekkor konfigurálhatjuk, hogy folyamatos mérést szeretnénk (*Continuous Samples*), vagy véges mintákat szeretnénk mérni, akár többször egymás után, de a mintavételezések között szünetet tarthatunk (*Finite Samples*). A mintavételezésről bővebben a 6. fejezetben lehet olvasni.



4.23. ábra: Példa mérőprogramra

Egy mérés végrehajtásához először létre kell hozni és konfigurálni a kívánt csatornát. Ezek után a feladatot el kell indítani, majd akár egy ciklusban el lehet végezni a mért értékek beolvasását. A program végén le kell állítani az összes feladatot, valamint törölni kell őket. A VI-ok egy része polimorf, saját igényeinknek megfelelően választhatjuk ki a kívánt funkciót. Ha egyszerre több fizikai csatornán szeretnénk mérést végezni, akkor azt a '*Browse...*' opcióra kattintva végezhetjük el, ahol a CTRL gomb megnyomásával egyszerre több csatornát is kiválaszthatunk. Ekkor beolvasáskor valamelyik nekünk megfelelő *N Chan* opciót kell kiválasztani.



4.24. ábra: Több fizikai csatorna kiválasztása

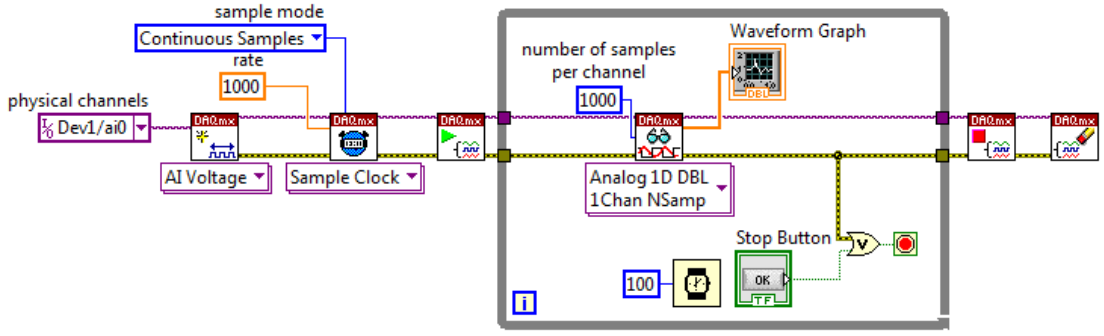
A csatorna kiválasztása mellett megadhatjuk a bemenő/kimenő feszültségtartományt is. A megadott értékek alapján, amennyiben a hardver alkalmas rá, megfelelő előerősítést is beállíthat. Analóg bemenetek esetén az egyik fontos beállítás a bemenet csatolása, a következő lehetőségek közül választhatunk:

- *Differential*: a műszer két vezeték közötti feszültségkülönbségét méri. Mindkét vezeték feszültsége a műszer bemenő feszültségtartományán belül kell hogy legyen. Lebegő (földfüggetlen) jelek esetén a bemenetek szaturálódhatnak valamely

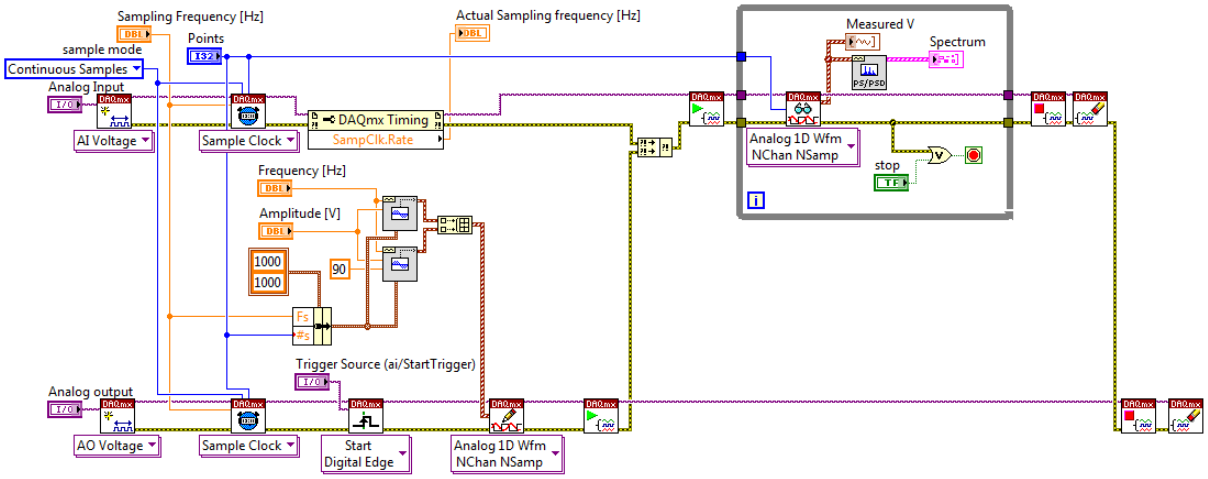
tápfeszülségnél, ezt úgy akadályozhatjuk meg, hogy ha nagy értékű ellenállásokon keresztül a földre kötjük egyik vagy mindkét vezetéket. Földelt jelek esetén ezzel a méréssel kiküszöbölhető a műszer és a külső föld potenciálja közötti eltérések hatása. Amennyiben a műszer földfüggetlen (galvanikusan leválasztott), célszerű a műszer földjét (*AI GND* vagy *COM*) összekötni a mérendő rendszer földjével.

- *Non-Referenced Single-Ended (NRSE)*: a mért jel egyik vezetéke az *AI-Sense* bemenetre van kötve, a műszer ehhez a referenciaponthoz képest méri a bemenő feszültséget. *AI-Sense* bemenetből csak egy van műszereként, egyébként a mérés megfelel a differenciális mérésnek.
- *Referenced Single-Ended (RSE)*: a bemeneti feszültséget a műszerföldhöz képes mérjük (*AI GND*).
- *Default*: a műszer adott bemenetére jellemző alap-konfiguráció. Egyetlen műszer esetén is különbözhet bemenetről-bemenetre, pl. NI 6211 műszer esetén az AI 0 alapértelmezett beállítása a differenciális (az AI 0 és AI 8 bemenet közötti feszültségkülönbséget méri), a közvetlenül mellette lévő AI 8 alapértelmezett beállítása pedig a *Referenced Single-Ended*.

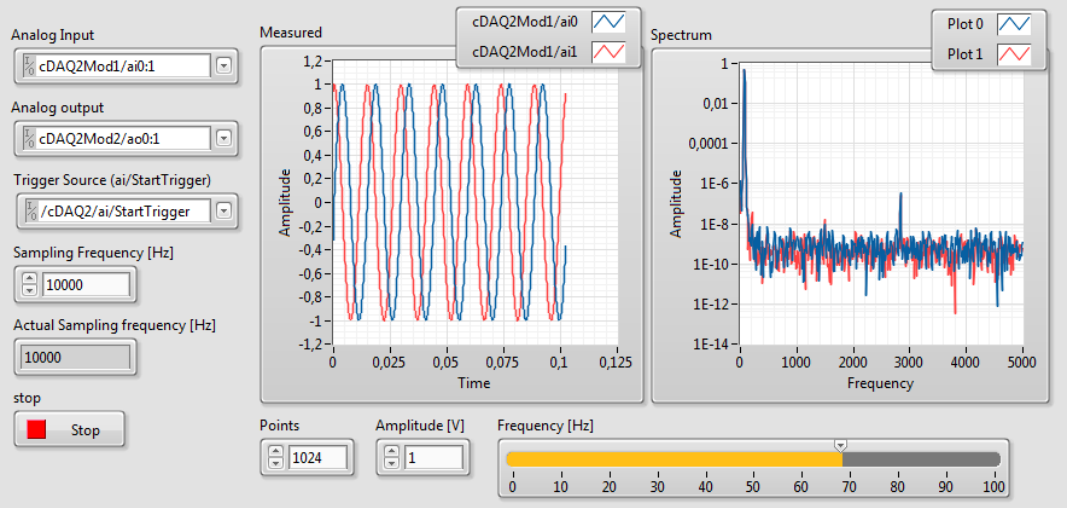
Ha megadott időzítéssel szeretnénk jeleket mérni, akkor meg kell adni a mintavételi frekvenciát is. Szükség esetén megfelelő triggerjeleket felhasználva egymáshoz lehet szinkronizálni az egyes feladatokat. Így szinkron méréseket is végre lehet hajtani, amennyiben a hardver is támogatja.



4.25. ábra: Mintavételezéses mérés a mintavételi frekvencia beállításával



4.26. ábra: Mérőprogram szinkron gerjesztéssel és méréssel két-két csatornán



4.27. ábra: A program előlapja

Feladatok

1. feladat

Hozzon létre olyan SubVI-okat, melyek a következő feladatokat látják el a MA-DAQ műszer esetén:

- Kommunikáció inicializálása, azonosító szöveg lekérdezése
- Feszültség kiadása valamely D/A konverter segítségével. Mi az összefüggés a DAC kódja és a kimenő feszültség között? Ábrázolja az összefüggést grafikonon is! Hozzon létre olyan VI-t, melynél V-ban kell megadni a kimenő feszültséget és nem bináris kódban!
- Kösse össze a DAC kimenetét valamely szabadon választott ADC bemenettel! Hozzon létre olyan programot, mely az ADC kódját be tudja olvasni! Vizsgálja meg mi az összefüggés az ADC kódja és a bemenő feszültség között, ábrázolja az eredményt! Hozzon létre olyan VI-t, mely V-ban jelzi ki a mért feszültséget!

2. feladat

Helyezze üzembe az oszcilloszkópot és vizsgálja meg a mérőfej kalibráló jelet! A *tktds1k2k Getting Started.vi* segítségével vizsgálja meg a kalibráló jelet a számítógép segítségével!

3. feladat

Hangkártyával játsszon le egy tetszőleges zeneszámot, majd vizsgálja meg a jelet oszcilloszkóp segítségével számítógépről! Módosítsa a példaprogramot, hogy az a lehető legsűrűbben végezzen mérést! Ábrázolja a zeneszám spektrumának időbeli változását *Intensity Chart*-on!

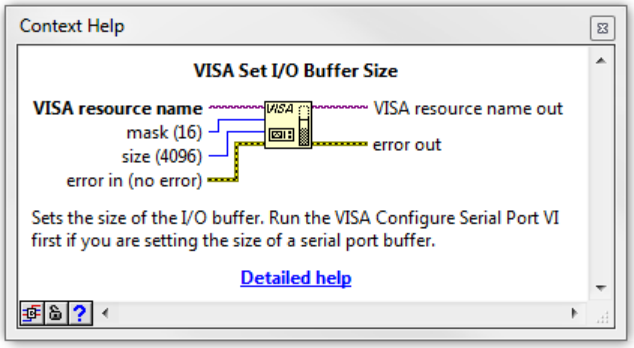
4. feladat

Hozzon létre egy függvénygenerátort egy NI műszert felhasználva! Melyik műszer alkalmas erre a feladatra, és melyik nem?

5. Mérések végzése és a mérési adatok feldolgozása LabVIEW környezetben

A személyi számítógépeket általában nem arra készítik, hogy velük valós idejű mérési és vezérlési feladatokat lehessen végezni. Ezért, ha számítógéppel végzünk méréseket számos szempontra kell odafigyelni:

- Ahhoz, hogy a mért jelek eljussanak a feldolgozó szoftverig, akár több 100 ms-ra is szükség lehet. Nagyságrendileg ugyanennyi idő kell, míg a szoftver által meghatározott parancsok eljutnak a műszerig.
- Az operációs rendszer illetve a többi program bármikor több másodpercig lefoglalhatja az erőforrásokat. Addig a szoftver nem tud sem adatokat fogadni, sem pedig beavatkozni.
- Szükség van az adatok megfelelő pufferelésére. E nélkül az adatok elveszhetnek. Lehetőség van hardveres pufferelésre, FIFO-k alkalmazására magában a műszerben, illetve az eszköz meghajtó-programjában, még az operációs rendszer szintjén. Minél nagyobb a kommunikáció sebessége, annál nagyobb méretű pufferekre lesz szükség.
- Ha túl sok feldolgozást végez a program, előfordulhat, hogy nem végez időben, mire a következő adatcsomagot fel kellene dolgozni. Érdemes két, egymástól független szálban végezni a mérést és az adatok feldolgozását. Így is csak a legszükségesebb adatfeldolgozásokat végezzük el mérés közben.



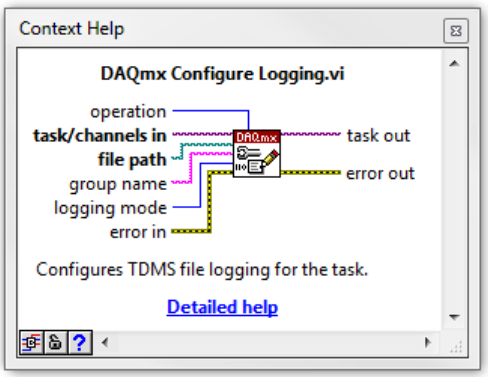
5.1. ábra: A soros port pufferének mérete szükség esetén növelhető. Így kisebb az esélye az adatok elvesztésének.

Természetesen a számítógépek használatának számos előnye is van, pl. nagy számítási sebesség érhető el, valamint rengeteg tárterület is rendelkezésre áll. E mellett rendelkezésre állnak speciális ipari számítógépek, valós idejű operációs rendszerrel, melyek biztosítják a mérésekhez szükséges megbízható reakcióidőt. Nagy sebességű adatfeldolgozáshoz és vezérléshez pedig FPGA alapú hardvereket is használhatunk. A LabVIEW alkalmas mindezek programozására is.

Mérési adatok mentése

Mérés közben szükség lehet az adatok mentésére is. Ha nagy mennyiségű adatunk van, szintén több tényezőre kell odafigyelni. Ha egyszerre több fájlba írunk, a merevlemez fejének mozgása időveszteséget okozhat. Nagyon nagy adatforgalmak esetén ez már problémát jelenthet, ezért célszerű inkább egy fájlba menteni, minél kevesebb adatfeldolgozással az adatokat, amiket később, utólag át lehet alakítani a kívánt formátumba. Néha maguk a műszerdriverek is támogatják az adatok biztonságos, valós idejű mentését. Ha NI mérőeszközöket használunk DAQmx driverrel, akkor például beállíthatjuk, hogy az adatokat

közvetlenül a driver mentse el a merevlemezre, akár úgy is, hogy a program azokat nem olvassa be. Ehhez az NI speciális adatformátumát használja, a TDMS-t [13].



5.2. ábra: Adatok mentése driver szinten TDMS fájlformátumban. Használata megtalálható a LabVIEW példakeresőben (*Find examples…*)

Amennyiben kevés adat keletkezik a mérés során, akkor nem annyira kritikus, hogy azokat milyen formátumban mentjük. Használhatunk szöveges fájlokat (pl. *Spreadsheet*), melyek előnye, hogy könnyen olvashatók bármilyen programmal. Használhatunk bináris formátumot is, de ekkor biztosítanunk kell egy programot, mely majd beolvassa az adatokat. A TDMS (technical data management streaming) a mérési adatok mellett további, a méréshez kapcsolódó adat is eltárolható. Egy modul segítségével az adatok közvetlenül is beolvashatók Excel táblázatba. E mellett követhetünk további szabványos fájlformátumokat is (pl. WAV, WDQ).

LabVIEW-ban van lehetőségünk a kijelzett adatok megjegyzésére is. Ehhez az *Edit* menüből válasszuk ki *Make Current Values Default* menüt, ezt követően a VI mentésekor minden kijelzett adat és beállítás elmentődik. A VI újbóli megnyitásakor ezek lesznek az alapértelmezett értékek. Ha csak egy *Control* vagy *Indicator* értékét szeretnénk alapértelmezetté tenni, akkor ugyanezt az opciót a helyi menü *Data Operations* almenüjében találhatjuk. A módszer hátránya, hogy a VI fájlja ezek után tárolni fogja az összes adatot, és megnövekszik a mérete.

Alternatív módszer a *Data Logging*, mely az *Operate* menüben érhető el. Ekkor az adatok egy külön fájlba mentődnek el, ráadásul több adathalmazt is elmenthetünk. Viszont az adott fájl csak addig lesz használható, míg nem módosítjuk a programunk előlapját. Hogy elkerüljük az adatok elvesztését, mindig készítsünk biztonsági másolatot a programunk aktuális állapotáról. A grafikonokon kijelzett adatok a helyi menüből könnyen exportálhatók szöveges, excel vagy kép formátumba.

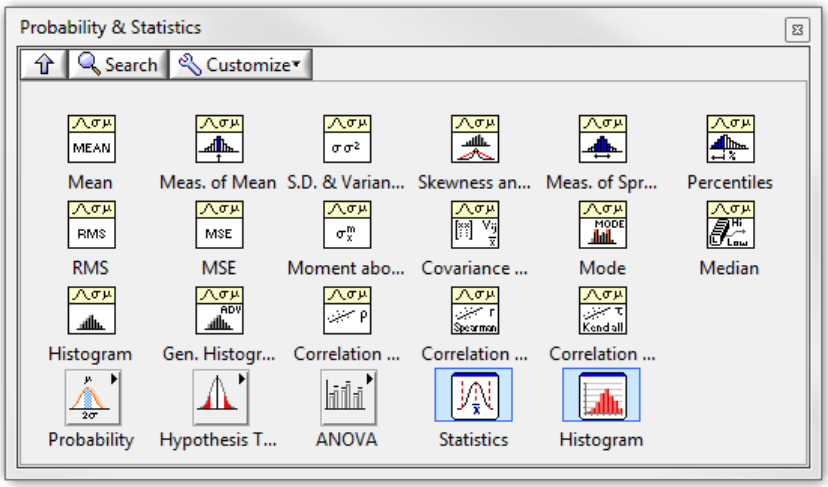
A mérési adatok feldolgozása

A mérési adatokat számos elemzésnek vethetjük alá. Számolhatunk különböző statisztikákat, illeszthetünk rájuk különböző görbéket, valamint különböző időtartománybeli és frekvenciatartománybeli vizsgálatokat is végezhetünk.

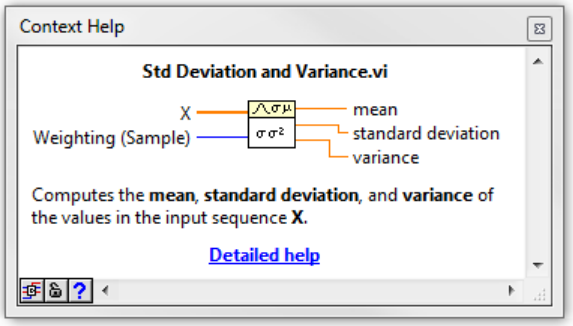
A két leggyakrabban használt statisztikai jellemző az adatok középértéke valamint a szórása. A *Standard Deviation and Variance.vi* megadja a középértéket, a szórást, valamint a varianciát (szórásnégyzet). Ha a *Weighting* bemenet *Sample* értékre van állítva (alapbeállítás) a korrigált empirikus szórást kapjuk meg, egyébként a korrigálatlant. A korrigált empirikus szórás képlete:

$$s_N = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{X_N})}{N-1}}$$

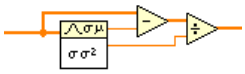
$$\overline{X_N} = \frac{\sum_{i=1}^{N} x_i}{N}$$



5.3. ábra: A statisztikai számolások palettája (a *Mathematics* paletta része)

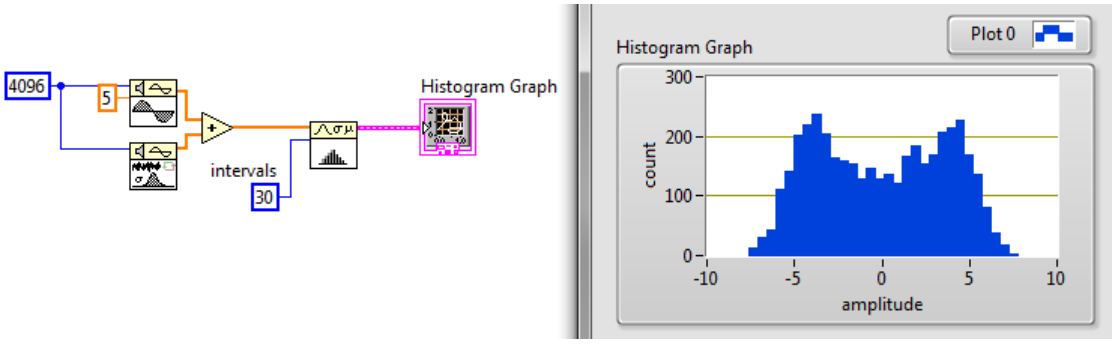


5.4. ábra: Átlag és szórás számolása

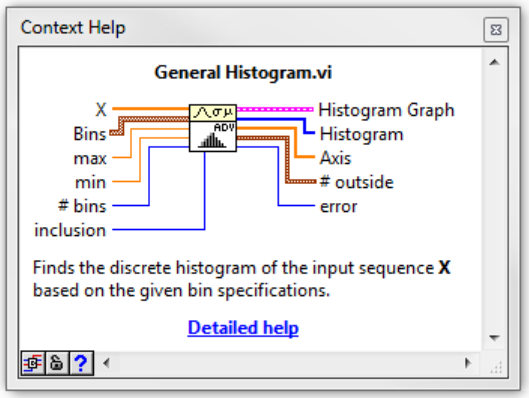


5.5. ábra: Egy jel normálása. Ezek után a középértéke nulla, szórása pedig egységnyi lesz.

Ha kíváncsiak vagyunk egy jel eloszlására, akkor hasznos segédeszköz a hisztogram. Segítségével ábrázolható egy jel sűrűségfüggvénye. A *Histogram.vi* a legegyszerűbb módszer egy hisztogram elkészítésére. Megadhatjuk neki az adattömböt és az intervallumok számát, ez alapján ő kitalálja, hogy az egyes intervallumok hogy helyezkedjenek el. Kimenetként kapunk egy XY grafikont, megkapjuk a hisztogram értékeit (hány darab adat van egy-egy intervallumban, az y tengely értékei), valamint az intervallumok közepét (ez lesz az x tengely). Néha hátrányt jelent, hogy az intervallumokat automatikusan skálázza, így az egyes hisztogramokat körülményes összehasonlítani. Erre ad megoldást a *General Histogram.vi*, ahol sokkal több opciónk van: meghatározhatjuk, hogy mely tartományt akarjuk felosztani adott számú intervallumra, de akár egyenként is megadhatjuk az egyes intervallumok határait.

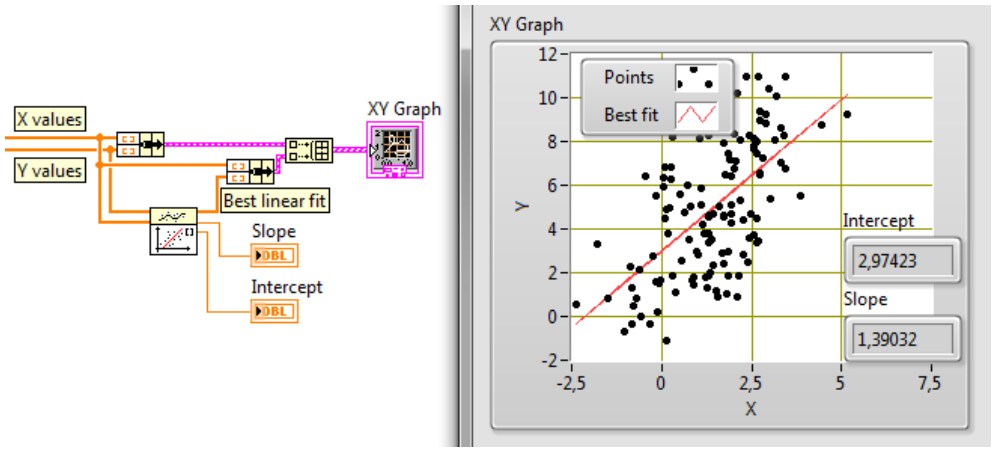


5.6. ábra: Egyszerű hisztogram készítése



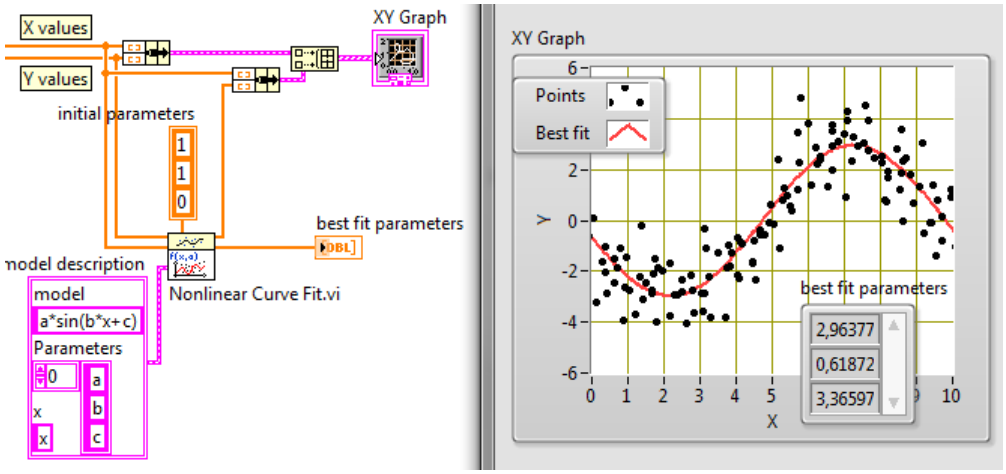
5.7. ábra: A General Histogram.vi sokkal több beállítási lehetőséget biztosít

Az adatok feldolgozása során sűrűn van szükség egyenes illesztésére a különböző paraméterek megállapítására (lineáris regresszió). Erre legegyszerűbben a *Linear Fit.vi* használható, mely segítségével megállapítható az egyenes meredeksége (*Slope*) és tengelymetszete (*Intercept*).



5.8. ábra: Egyenes illesztése, valamint az eredmények megjelenítése egy XY grafikonon

Számos lehetőség van nemlineáris illesztésre is. Ezek közül a leguniverzálisabb a Nonlinear Curve Fit.vi, ahol mi magunk adhatjuk meg a görbe képletét, vagy ha nem elérhető képlet, akkor azt a VI-t, ami kiszámolja a függvény értékét. Szükséges még megadni az illesztés kezdő értékeit is. Az algoritmus fokozatosan közelít a legjobb illesztéshez. Amennyiben nem megfelelőek a bemenő paraméterek, az illesztés nem konvergál az optimális megoldáshoz..



5.9. ábra: Nemlineáris illesztés

Feladatok

1. feladat

A bemenet GND állásba kapcsolásával vizsgálja meg, hogy az oszcilloszkóp különböző érzékenységénél mekkora az oszcilloszkóp zaja. Az LSB hányszorosa a szórás?

2. feladat

Készítsen olyan programot, mely a megadott *Minta.05.02.dat* adatfájlt beolvassa [28], majd megjeleníti. Az adatfájl első oszlopa az idő, a második pedig a kitérés.

Illesszen egyenest az adatsorra, jelenítse meg az illesztés paramétereit, valamint ábrázolja egyszerre az adatpontokat és az illesztett egyenest!

Vonja ki az előbb kapott trendet az adatsorból, így megkapja az egyenesre szuperponált véletlenszerű jelet (zaj). Adja meg a zaj szórását. Készítsen hisztogramot a zajról, majd jelenítse meg!

3. feladat

Olvassa be a Minta.05.03.dat adatfájlt [28], és jelenítse meg egy XY grafikonon! Hozzon létre két kurzort a grafikonon, majd Property Node segítségével olvassa ki a két kurzor X koordinátáját. A kurzorok segítségével vágja ki a 0 körüli csúcsot az adatsorból, majd jelenítse meg külön grafikonon a kapott részletet. Az így kapott görbére illesszen Lorenzi-görbét, és jelenítse meg őket egy közös grafikonon.

A görbe képlete: $y = \frac{a^2}{((x-b)^2+c^2)}$

A kurzorok listáját a grafikon *CrsrList* nevű tulajdonsága tartalmazza. Ez egy tömb, mely *Cluster*-eket tartalmaz. A *Cluster Position.X* eleme tartalmazza a kurzorok X koordinátáját. Az adott X értékhez tartozó tömbindexet a *Threshold 1D array* tömbművelettel lehet megállapítani.

4. feladat

A kiadott műszer segítségével mérje meg, majd ábrázolja a következő alkatrészek feszültség-áramerősség karakterisztikáját:

- Dióda
- Zéner dióda
- Zöld LED
- Sárga LED
- Piros LED

Mi az összefüggés az egyes LED-ek karakterisztikája és színe között? Számolással is igazolja az összefüggést!

5. feladat

Az adatgyűjtő és a termisztor segítségével készítsen hőmérőt. A hőmérséklet időbeli változását egy *Waveform Chart*-on jelezze ki.

6. Váltakozó áramú mérések

A váltakozó áram paraméterei

Az elektronikában előforduló egyik leggyakoribb jel a szinuszos váltakozó feszültség vagy áram. Ennek időbeli értéke:

$$U(t) = U_{\max}\sin(\omega \cdot t + \varphi)$$

ahol $U_{\max}$ az amplitúdó, ω a jel körfekvenciája (egysége 1/s), φ a fázisszög. Fáziskülönbséget mindig egy megadott referenciához képest tudunk értelmezni. Szinusz helyett ugyanúgy használhatunk koszinuszt is az egyenleteinkben. A jel frekvenciája: $f = \omega/2\pi$ (egysége: Hz), a periódusidő pedig $T = 1/f$ (egysége: s).

Váltakozó jeleket számos további paraméterrel is jellemezhetjük:

Egyszerű középérték

$$U_{\mathrm{k}} = \frac{1}{T}\int_0^T U(t)\mathrm{d}t$$

Tiszta szinuszos jel esetén a középérték nulla.

Effektív középérték Az effektív érték annak az egyenfeszültségnek vagy egyenáramnak az erősségének felel meg, melynek teljesítménye egy ohmikus ellenálláson megegyezik az egyen vagy váltóáram teljesítményével.

$$U_{\mathrm{eff}} = \sqrt{\frac{1}{T}\int_0^T U(t)^2\mathrm{d}t}$$

Tiszta szinuszos jel esetén: $U = U_{\max}/\sqrt{2}$

Abszolút középérték (az egyenirányított jel középértéke)

$$U_{\mathrm{a}} = \frac{1}{T}\int_0^T |U(t)|\mathrm{d}t$$

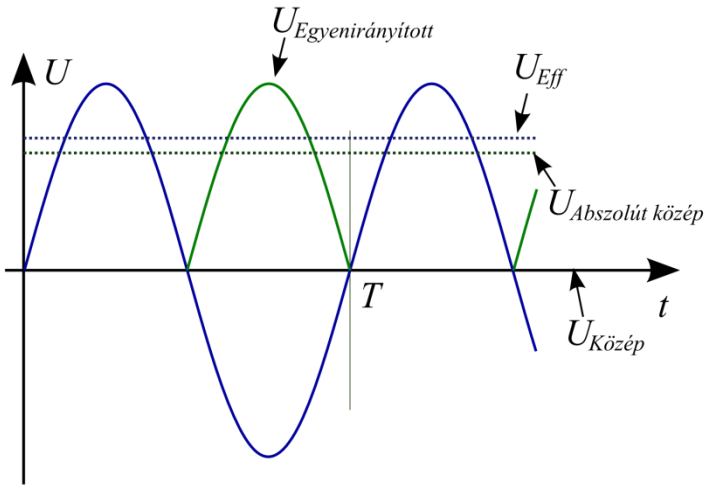
Szinuszos jel esetén $U_{\mathrm{a}} = 2/\pi \cdot U_{\max}$

Csúcsérték

Szinuszos jel esetén maga az amplitúdó, zajok esetén tipikusan három szórás.

Csúcstól csúcsig amplitúdó

Az amplitúdó kétszerese, a legnagyobb és a legkisebb érték különbsége.



6.1. ábra: A váltakozó feszültség paraméterei szinuszos jel esetén

Egy váltakozó jel pillanatnyi teljesítményét a következőképp tudjuk meghatározni:

$$P(t) = U(t) \cdot I(t)$$

A fogyasztó által felvett teljesítmény folyamatosan változik. Amennyiben kíváncsiak vagyunk a *hatásos teljesítményre*, egy periódusidőre átlagolnunk kell a pillanatnyi teljesítményt. Szinuszos jel esetén ez a következő:

$$P_{\text{eff}} = U_{\text{eff}} \cdot I_{\text{eff}} \cdot \cos(\varphi)$$

ahol φ a feszültség és az áramerősség közötti fázisszög. Az energia egy része folyamatosan ingázik az erőmű és a fogyasztó között, ez a meddő teljesítmény:

$$P_{\text{m}} = U_{\text{eff}} \cdot I_{\text{eff}} \cdot \sin(\varphi)$$

Mintavételezés, mintavételi tétel, spektrum

A periodikus jelek előállíthatók szinuszos és koszinuszos függvények összegeként:

$$x(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} [a_k cos(k \cdot \omega_0 \cdot t) + b_k \sin(k \cdot \omega_0 \cdot t)]$$

ahol: $x(t) = x(t + T_0)$ a periodikus függvény, T_0 a periódusidő, k pedig a felharmonikus sorszáma. Az a_k és a b_k az egyes komponensek amplitúdója, melyet könnyen kiszámolhatunk.

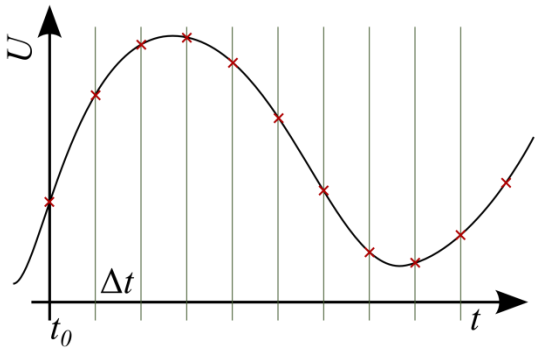
Folytonos, nem periodikus jelek esetén a Fourier-transzformációt használhatjuk:

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot \mathrm{e}^{-\mathrm{i}\omega t} \mathrm{d}t$$

$$x(t) = \int_{-\infty}^{\infty} X(f) \cdot \mathrm{e}^{\mathrm{i}\omega t} \mathrm{d}\omega$$

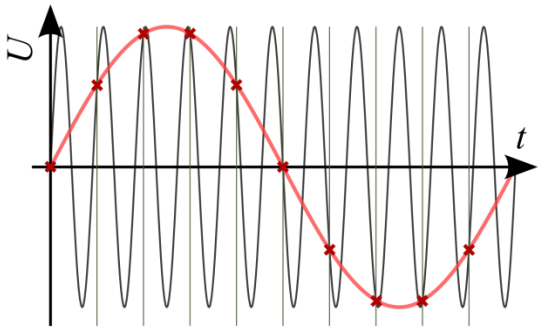
Itt $x(t)$ a jel időtartománybeli reprezentációja, $X(f)$ pedig a neki megfelelő frekvenciatartománybeli reprezentáció, vagyis spektrum. A spektrum komplex számokból áll, ahol a komplex szám szöge megadja a fázisszöget az adott frekvencián. Az imaginárius egységet (i) az elektronikában néha j-vel jelölik.

Amikor egy időfüggő jelet mérünk, csak véges számú adatot tárolhatunk, a jelből csak meghatározott időközönként veszünk mintát. Ezt az eljárást hívjuk mintavételezésnek. A mintavételezés legtöbbször periodikus, a mintavételi időköz reciproka a mintavételi frekvencia: $f_m = 1/\Delta t$.



6.2. ábra: A mintavételezett jel. A LabVIEW-ban *Waveform* adattípusként célszerű tárolni.

Mintavételi tétel: Ha a jelben előforduló legnagyobb frekvenciájú komponens frekvenciája kisebb, mint a mintavételi frekvencia fele, a mintavételezés nem okoz információveszteséget. A mintavételi tétel megsértése esetén a magasabb frekvenciájú komponensek megjelennek a spektrumban a 0 és $f_m/2$ közötti tartományban. Ezt a jelenséget hívjuk *aliasing* zajnak Az aliasing elkerülésére a mintavételezés előtt a jelből ki kell szűrni a mintavételezési frekvencia felénél nagyobb frekvenciájú komponeseket.



6.3. ábra: A mintavételi tétel megsértésének következményei: a magas frekvenciájú szinusz helyett egy sokkal alacsonyabb frekvenciájú szinuszt kapunk

Ha egy mérést végzünk, akkor mind mintavételezzük a jelet, mind pedig kiválasztunk a jelből egy véges méretű mintát. Ebben az esetben a Fourier-transzformáció helyett a diszkrét Fourier-transzformációt használjuk (DFT). Ekkor a spektrum csak diszkrét pontokban van értelmezve (mint a Fourier-sor esetén), viszont csak a mintavételi frekvencia feléig van értelmezve. Ezen a tartományon kívül a spektrum redundáns, és periodikusan ismétlődik. Az időbeli jel a mintavételezés miatt diszkrét időpontokban van értelmezve, a mérési időn kívüli rész pedig periodikusan ki lesz terjesztve. A periodikus kiterjesztés miatt törés következhet be a jelben - ha a jel periódusának nem egész számú többszöröse a mérési időtartam -, így a spektrum megváltozik, nem éles, hanem szélesebb csúcsokat fogunk kapni (spektrális szétfolyás). Ezt az effektust megfelelő ablakfüggvényekkel csökkenthetjük.

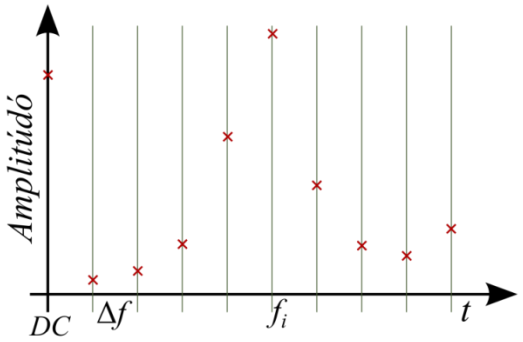
A diszkrét Fourier-transzformáció egyenletei:

$$X_k = \sum_{j=0}^{N-1} x_j \cdot \mathrm{e}^{-\mathrm{i}\cdot 2\pi \cdot \frac{j\cdot k}{N}}$$

$$x_j = \frac{1}{N} \sum_{j=0}^{N-1} X_k \cdot \mathrm{e}^{\mathrm{i}\cdot 2\pi \cdot \frac{j\cdot k}{N}}$$

Ahol $x_j = x(j \cdot \Delta t)$ a mintavételezett jel. Az így kapott spektrumot kétoldalas spektrumnak is hívjuk. A $k = 0 \cdots \frac{N}{2} - 1$ közötti rész a tartalmazza a lényeges információt, a $k = \frac{N}{2} \cdots N - 1$ közötti rész pedig a korábbi rész komplex konjugáltja, $x_k = x_n - k^*$. A nulladik indexű elem a jel átlaga (DC komponens), a spektrum felbontása pedig:

$$\Delta f = \frac{1}{T_{\text{mérés}}} = \frac{1}{N \cdot \Delta t}$$



6.4. ábra: Példa spektrumra. Az *i*. komponens frekvenciája: $f_i = i \cdot \Delta f$

A k-adik frekvenciához (k>0) tarozó periodikus komponens amplitúdója $2 \cdot |X_k|$.

A teljesítménysűrűség segítségével jellemezhetjük egy adott frekvenciatartományba eső teljesítményt. A DFT eredményéből a következőképp számolhatjuk ki:

$$PSD = \frac{2 \cdot |X_k|^2}{\Delta f}$$

A gyors Fourier-transzformáció (FFT) pontosan ugyanazt számolja ki, mint a DFT, azonban hatékonyabban van implementálva az algoritmus. Így a műveletigénye N^2 helyett $N \log_2 N$. Fontos feltétel viszont, hogy az adatpontok száma kettő hatvány kell legyen. Nagyobb adatmennyiségek esetén célszerű az FFT-t választani a spektrum számolásához.

A spektrum ábrázolása során a függőleges tengely sokszor logaritmikus (decibelskála), ugyanígy a frekvencia tengely is lehet logaritmikus. A aktuális feladatnak megfelelően célszerű megválasztani az ábrázolást, hogy a lehető legjobban megfigyelhető legyen a kívánt jelenség.

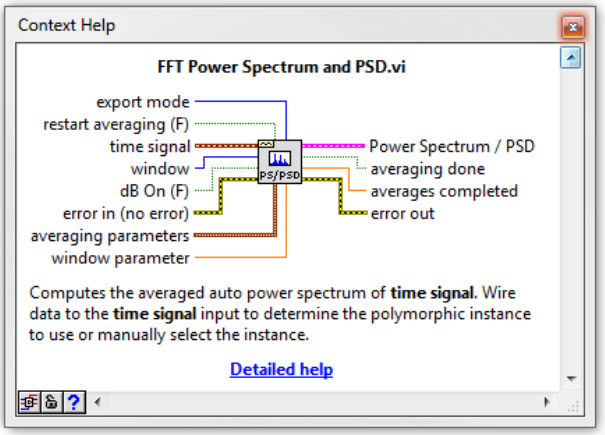
Négyzetes jelek esetén (pl. teljesítmény) a decibelt a következő képlettel határozhatjuk meg:

$$P_{db} = 10 \log\left(\frac{P}{P_0}\right)$$

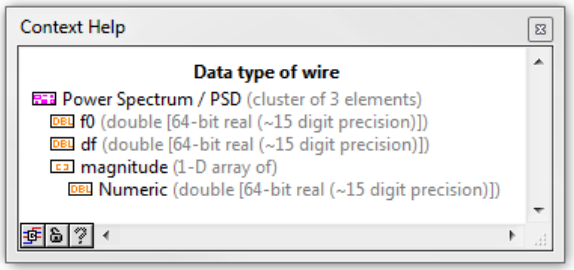
Lineáris jel esetén (pl. áramerősség) a decibelérték:

$$I_{db} = 20 \log\left(\frac{I}{I_0}\right)$$

A LabVIEW-ban egy jel teljesítménysűrűség-spektrumát a *FFT Power Spectrum and PSD.vi* segítségével határozhatjuk meg (Signal Processing / Waveform Measurements). Ha a vizsgált adatsor nem kettő hatvány, akkor FFT helyett automatikusan DFT algoritmust alkalmaz.



6.5. ábra: Teljesítménysűrűség-spektrum számolása. A VI számos paramétert fogad, megadható az ablakfüggvény, az átlagolások száma, valamint az, hogy decibel skálán ábrázoljuk az eredményt. Megjegyzés: a teljesítménysűrűség spektrum számolásához az *export mode* bemenetet *Power Spectral Density* értékre kell állítani.



6.6. ábra: A teljesítménysűrűség-spektrum számolás eredménye. Tartalmazza a spektrum összes paraméterét, *Waveform Graph*-on közvetlenül ábrázolható. Az *f0* mindig nulla, a *df* pedig a frekvenciafeloldás. A magnitude az a tömb, ami a spektrumot tartalmazza.

Feladatok

1. feladat

Vizsgálja meg egy háromszögjel és négyzetének az amplitúdó-spektrumát. Magyarázza meg a különbségeket!

2. feladat

A mellékelt SampleSignal.06.02.vi előállít egy időbeli jelet [28]. Készítsen olyan programot, mely kiszámolja a jel teljesítménysűrűség-spektrumát! Vonjon le következtetéseket a spektrum alapján! A jelet darabolja fel 1024 mintákból álló darabokra, majd számolja ki ezen darabok spektrumát és ábrázolja *Intensity Graph*-on (időbeli spektrum)! Magyarázza meg a látott spektrumot!

Ábrázolja egy grafikonon az 1024 Hz-en mérhető teljesítményt, és vonjon le következtetéseket!

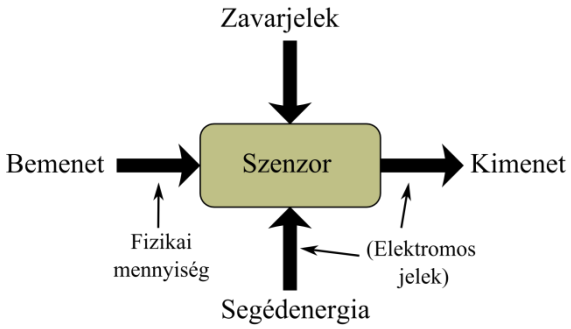
3. feladat

Mérje meg a kiadott áramkör átviteli függvényét úgy, hogy változó frekvenciájú szinusszal gerjeszti, majd pedig megméri a válaszjelet. Ábrázolja az átviteli függvényt!

Mérje meg úgy az átviteli függvényt, hogy egyszerre több szinuszos összetevővel gerjeszti a rendszert!

7. Szenzorok és távadók vizsgálata

A mérendő fizikai vagy kémiai jeleket szenzorokkal alakíthatjuk át elektromos jelekké. Az így létrejött elektromos jelet kondicionáljuk (zajszűrés, erősítés), digitalizáljuk, majd pedig szoftveresen feldolgozzuk.



7.1. ábra: A szenzorok működése során történő energiaátalakítások

A szenzorokat számos tulajdonsággal jellemezhetjük:

- Mért fizikai mennyiség (pl. hőmérséklet, nyomás, elmozdulás, kémhatás, ...)
- Mérési tartomány: mely tartományban teljesíti a szenzor a specifikációkat
- Felbontóképesség: melyik a legkisebb mennyiség, amit még meg tud különböztetni
- Mérés hibája (nullpont-hiba, érzékenység-hiba, hiszterézis, linearitás-hiba)
- Hőmérsékletfüggés, érzékenység a környezeti hatásokra (pl. nedvesség)
- Beállási idő, frekvenciakarakterisztika
- Kialakítás, működés elve
- Zaj
- Szükséges segédenergia (pl. tápfeszültség)

Szenzor interfész áramkörök

A szenzorok különböző kimenetekkel rendelkezhetnek, ez meghatározza, hogy milyen módszerrel tudjuk elvégezni a megfelelő jelkondicionálást illetve a digitalizálást.

Feszültség kimenetű szenzorok

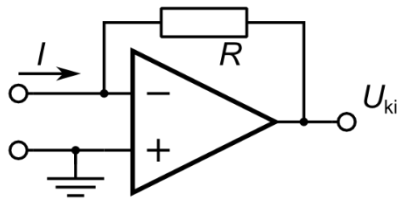
A jelszinttől függően a következőkre lehet szükség:

- Nagy bemenő ellenállású erősítők
- Differenciálerősítő
- Sávszűrők
- Szintillesztés
- ADC meghajtása

Ezeket a funkciókat megfelelő műveleti erősítőkkel, műszererősítőkkel és aktív szűrőkkel valósíthatjuk meg. Feszültség kimenetű szenzor pl. a termoelem és a fényelem.

Áram kimenetű szenzorok

Ezen szenzorok jelét többnyire feszültséggé alakítjuk, az így kapott feszültséget pedig kondicionáljuk (zajszűrés, erősítés), majd pedig digitalizáljuk. Áram kimenetű szenzor pl. a fotódióda.



7.2. ábra: Példa áram-feszültség konverzióra

Ellenállás kimenetű szenzorok kezelése

Ellenállások értékének mérésére számos módszer van (lásd a 2 fejezetet). Kis ellenállású szenzorok esetén (1 kΩ alatt), már számottevő a vezetékek ellenállása, ezért négypontos ellenállásmérést célszerű használni. Nagy ellenállású szenzorok esetén (100 kΩ felett) pedig a mérőműszerek bemenő impedanciája is fontos lehet, így megfelelően kell kiválasztani a feldolgozó elektronikát. Az ellenállás kimenetű szenzorok egy részét híd kapcsolásba kötjük, így az egészen kicsi változások is jól mérhetők. Ilyen szenzorok többek között a nyúlásmérő bélyegek, nyomásszenzorok egy része valamint a PT100-as hőmérsékletszenzorok.

TEDS szenzorok

Ezek a szenzorok egyrészt tartalmazzák magát az érzékelő elemet, esetleg valamilyen fokú előerősítést, de e mellett tartalmaznak egy külön memóriát, mely elektronikus formában tartalmazza a szenzor részletes adatlapját (Transducer Electronic Data Sheet). Ez tartalmazza a szenzor leírását, valamint a kalibrációs adatokat is. Ezáltal a mérőműszer azonnal, automatikusan mindent tud a csatlakoztatott szenzorról.

Digitális szenzorok

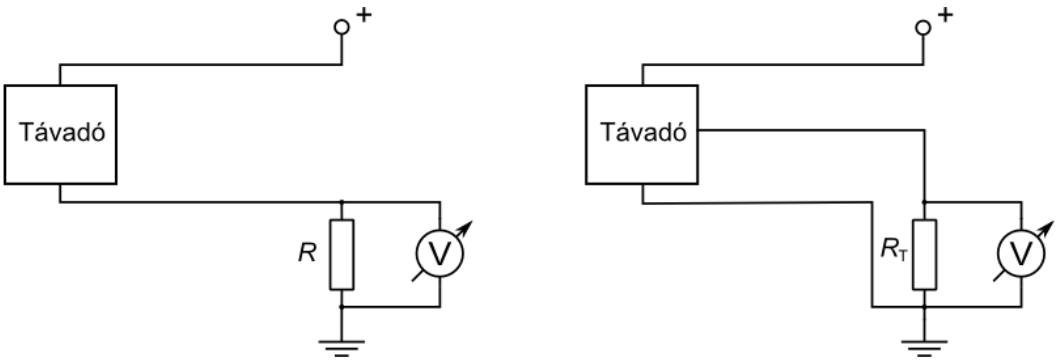
Jelenleg egyre inkább elterjednek azok a szenzorok, amelyek nem csak magát az érzékelő elemet tartalmazzák, hanem a jelfeldolgozó elektronikát valamint az A/D konvertert is. A szenzort valamilyen szabványos digitális interfészen kapcsolhatjuk hozzá a beágyazott rendszerhez, ami általában egy mikrovezérlő. A legelterjedtebb interfészek az SPI, és az I2C, sokszor egy buszra számos szenzor is felfűzhető.

A digitális szenzorok előnye, hogy a szenzor tartalmazza a kalibrációs adatokat, a mikrovezérlő számára küldött kód pedig lineáris függvénye a mért fizikai mennyiségnek. Hátrányuk lehet, hogy a mintavételezés időpontját nem mindig tudjuk időzíteni, így ha fontos a megfelelően szinkronizált mintavételezés, akkor nem mindig alkalmazhatók.

Távadók

Az iparban a szenzorokat és a feldolgozó elektronikát egy robosztus házba építik be, a mérő és vezérlőrendszerhez pedig szabványos interfészeken keresztül csatlakoztathatjuk. A legtipikusabb interfészek:

- 4-20 mA-es távadók: a szenzor egy áramjelet ad ki, amely 4 és 20 mA között változhat, és általában lineáris függvénye a mért jelnek. A 4 mA-es minimum áram elegendő energiát szolgáltathat a szenzor elektronikájának működtetéséhez, így bizonyos szenzorok nem is igényelnek külön tápfeszültséget.
- 0-10 V-os távadók: a mért tartományt ebbe a jól kezelhető feszültségtartományba alakítják át, így az egyszerűen kezelhető a PLC-k A/D konverterei által.



7.3. ábra: 4-20 mA-es távadók bekötése

Szenzorok alkalmazása

Hőmérséklet mérése

A legtöbb folyamat, fizikai és kémiai tulajdonság hőmérsékletfüggő. Éppen ezért a hőmérséklet az egyik leggyakrabban mért paraméter. A hőmérséklet mérésére számos lehetőségünk van, a mérés elve az, hogy hőmérsékletváltozás hatására változás áll be a szenzorban, és ezt a változást mérjük. A méréshez biztosítani kell a megfelelő hőátadást, mivel a szenzor hőegyensúlyba kell kerüljön a mérendő test hőmérsékletével. A hőmérsékletváltozás általában lassú folyamat, ezért másodpercenként néhány mintavétel többnyire elég a megfelelő időfelbontáshoz.

Az egyik leggyakrabban alkalmazott és legolcsóbb hőmérséklet-mérő szenzor a termisztor, ha -90°C és 130°C között kell hőmérsékletet mérni. Karakterisztikája alapján NTC hőmérőnek is hívják: a hőmérséklet növekedésével csökken a szenzor ellenállása, mégpedig egy exponenciális függvény szerint:

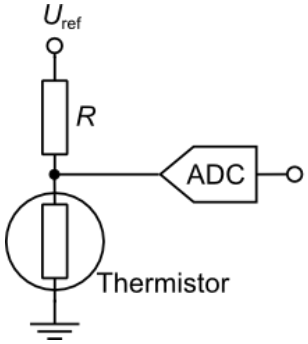
$$R(T) = R_{\text{ref}} \cdot \mathrm{e}^{A+\frac{B}{T}+\frac{C}{T^2}+\frac{D}{T^3}+\cdots}$$

ahol A, B, C, D a termisztorra jellemző paraméterek, melyeket az adatlapban megadnak. Fontos, hogy a hőmérséklet Kelvinben van. Az előbbi bonyolult formula helyett általában egy egyszerűbbet használunk, mely az alkalmazások nagy részében elegendően pontos:

$$R(T) = R_{25} \cdot \mathrm{e}^{\frac{B_{25/85}}{T}-\frac{B_{25/85}}{T_{25}}}$$

$$T(R) = \frac{1}{\frac{1}{T_{25}} + \frac{1}{B_{25/85}} \cdot \ln\frac{R}{R_{25}}}$$

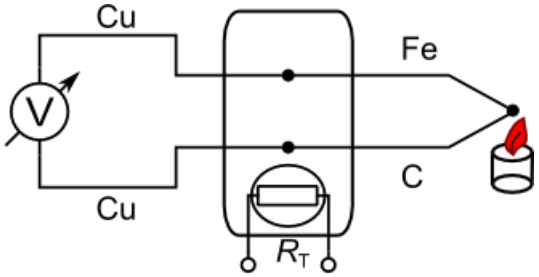
A T_{25} a 25°C-hoz tartozó hőmérséklet Kelvinben, R_{25} a 25°C-hoz tartozó ellenállásérték (pl. 10 kΩ), a $B_{25/85}$ az adott termisztorra jellemző paraméter és az adatlapjában megtalálható. Értéke 3500 és 4500 K között van.



7.4. ábra: A termisztor tipikus bekötése: az ellenállásosztó kimenete megváltozik a hőmérsékletváltozás hatására

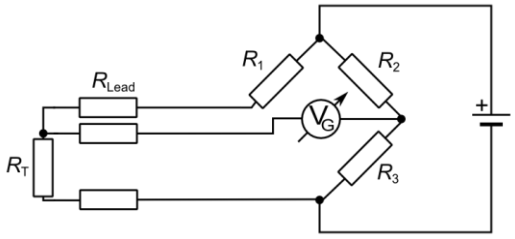
7.5. ábra: Egy termisztor képe

Ha a termisztornál nagyobb mérési tartományra van szükségünk, alkalmazhatunk termoelemeket. Ezek tipikusan -200°C és 1350°C között használhatók. A termoelemek (*thermocouple*) működése a Seebeck-effektuson alapul, hőmérsékletkülönbség hatására a pólusain feszültségkülönbség lép fel. Ez a feszültségkülönbség nagyon kicsi, a termoelemeket fogadó műszerek bemenő feszültségtartomány 100 mV alatt van. Ennek megfelelően nagy az előerősítés mértéke, valamint a nagy felbontású mérésekhez gyakran 24 bit-es A/D konvertert használnak. Mivel a kapott feszültség a mérési pont és a referenciapont közötti hőmérsékletkülönbségtől függ, fontos az adott pont hőmérsékletének pontos ismerete. Ennek figyelembevételét hívjuk hidegpont-kompenzációnak.



7.6. ábra: Hidegpont-kompenzálás

A platina szenzorok segítségével (pl. PT-100, RTD) a termisztornál nagyobb pontosság és alacsonyabb drift érhető el széles mérési tartomány mellett (-260°C és 850°C között). Ugyanúgy, mint a termisztor esetén itt is a szenzor ellenállásának változását használjuk fel a hőmérséklet mérésére, de itt az ellenállás nő a hőmérséklet emelkedésekor, valamint a karakterisztika is közel lineáris. Ugyanakkor a hőmérséklet méréséhez nagyon pontosan kell mérni az ellenállás változását. Ehhez tipikusan egy ellenállásmérő hídkapcsolást alkalmazunk.



7.7. ábra: PT100-as szenzor mérése a vezeték ellenállásának figyelembe vételével

Gyorsulásszenzorok, giroszkópok és iránytűk

Manapság a legtöbb telefon tartalmaz inercia szenzorokat, melyek arra szolgálnak, hogy a telefon követhesse saját helyzetének, pozíciójának változását. E mellett ezek a szenzorok fontos szerepet kapnak a különböző robotok (kerekes robotok, repülő robotok) navigálásában. Mindhárom szenzortípusból gyártanak analóg és digitális interfésszel rendelkezőt is.

A gyorsulás érzékelése általában egy tehetetlen tömeg elmozdulásának mérésén alapul. Mikro méretű megmunkálással ezek a szenzorok ugyanakkora méretűek, mint egy átlagos integrált áramkör (MEMS technológia), és három egymásra merőleges tengelyre mérik a gyorsulást. A gyorsulásszenzorok alkalmazásánál fontos, hogy nem csak a test sebességének változását méri, hanem a földi gravitáció hatását is. Utóbbinak köszönhetően szögmérésre is használhatók a szenzorok. A gyorsulásjelből integrálással elvileg meg lehetne állapítani az aktuális sebességet és pozíciót is, azonban a nullponthibák, a drift, és a függőlegessel bezárt szög változása miatt az eredmény túl pontatlan lesz akár becslésekhez is.

7.8. ábra: Gyorsulásmérő és giroszkóp MEMS szenzorok

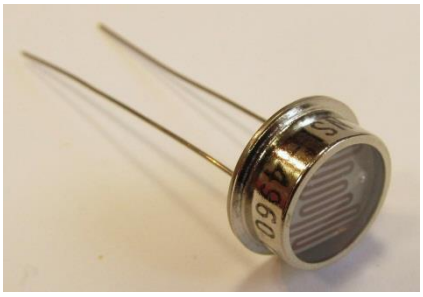
A giroszkópok a Coriolis-erőt kihasználva mérik a szenzor szögsebességét. A szenzorok szintén a MEMS technológián alapulnak, és egy, két vagy három egymásra merőleges tengely irányában mérik a szögsebességet. Az aktuális pozíció megállapításához integrálni kellene a jelet, azonban a pontatlanságok miatt az eredmény nehezen használható.

A Hall szenzor segítségével a mágneses tér erősségét mérhetjük, feszültség kimenetű szenzor. Különböző feladatokra használhatjuk, pl. közelség érzékelésére (végállás-kapcsolók), motorok vezérlésére, szög meghatározására.

A térbeli irányok megfelelő követéséhez az iránytű a legalkalmasabb szenzor, mely a Föld mágneses terét méri. Ugyanakkor alkalmazásakor figyelni kell a szenzor közelében lévő mágneses anyagokra.

Fény érzékelése

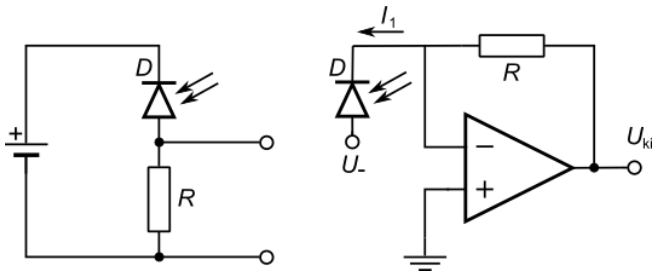
A fényellenállás olyan félvezető eszköz, mely ellenállása a fényerősség növekedésével csökken. A termisztorhoz hasonlóan könnyen illesztetjük a mérőelektronikához.



7.9. ábra: Fényellenállás

A fotódiódák sokkal gyorsabban reagálnak a fény változására, így akár adatátvitelre is használhatók. A rajtuk átfolyó áram egyenesen arányos a fényintenzitással. Spektrális érzékenységük megfelelő optikai szűrővel módosítható, elterjedtek az infra-fotódiódák, melyekből optikai kapukat, közelségérzékelőket készítünk.

7.10. ábra: Fotódióda



7.11. ábra: Fotódiódák illesztésére alkalmas áramkörök. A jobb oldali áramkör az előfeszítés értékétől függően sokkal gyorsabb, valamint a kimenet is lineáris függvénye a fényintenzitásnak

Feladatok

1. feladat

- Hány ohmos egy termisztor 50°C-on? $R_{25} = 10\ \text{k}\Omega$, $B_{25/85} = 3977\ \text{K}$.
- Egy K-típusú termoelem feszültsége 15 mV. Mekkora a hőmérsékletkülönbség?

2. feladat

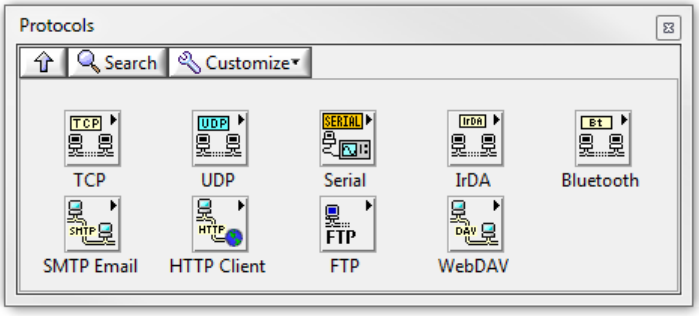
Kapcsoljon az adatgyűjtőhöz egy termisztort a megfelelő módon, majd készítsen olyan programot, mely regisztrálja és kijelzi a hőmérsékletet °C-ban. Vegye fel egy test lehűlési karakterisztikáját!

3. feladat

A megadott szenzor/távadó adatlapja alapján készítsen egy illesztő áramkört az adatgyűjtő eszközhöz, majd mérje meg és jelezze ki a szenzor jelét!

8. Kommunikációs protokollok a méréstechnikában

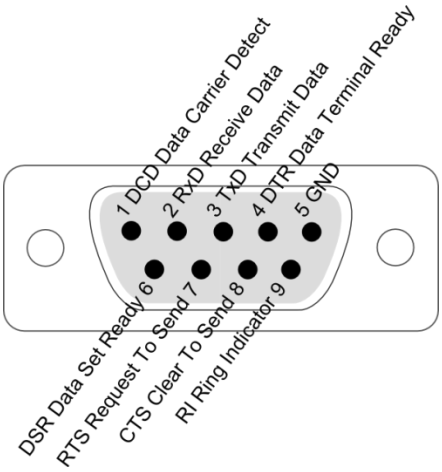
A méréstechnikában és az iparban számos kommunikációs protokoll terjedt el. A legtöbb ilyen protokoll soros adatátvitelre épül, a legfontosabb különbségek az adatvonalak logikai szintjei és szerepe, valamint a fizikai rétegre épülő további rétegekben rejlenek.



8.1. ábra: A legelterjedtebb protokollok LabVIEW-ban

Az RS232 szabvány

Az RS-232 szabvány elég régi (első verziója 1962-ben született), elsősorban az volt a célja, hogy a számítógépekhez különböző perifériákat tudjunk hozzákötni (pl. modemet). A nevezéktan is ezt tükrözi, valamint a rendelkezésre álló vonalak. A DTE eszköz (*Data Terminal Equipment*) felel meg a számítógépnek, a DCE (Data Circuitterminating Equipment) pedig a perifériának.



8.2. ábra: A soros port lábbekötése

Az egyes vonalak szerepe:

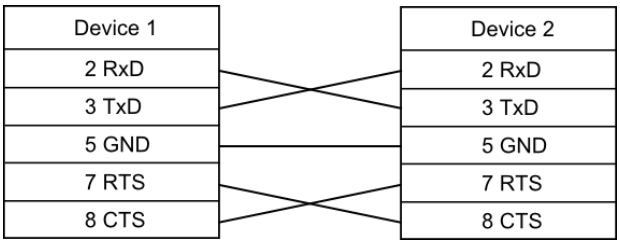
Vezeték	Funkció	Irány
TxD	Adatok küldése	DTE → DCE
RxD	Adatok olvasása	DCE → DTE
GND	Referenciapont (föld)	
RTS	A számítógép adatot kér az eszköztől	DTE → DCE
CTS	Az eszköz kész az adatok fogadására	DCE → DTE
DTR	A számítógép kész a kommunikációra	DTE → DCE
DSR	Az eszköz kész a kommunikációra	DCE → DTE
DCD	A modem csatlakozva van a telefonos	DCE → DTE

	hálózathoz	
RI	Bejövő hívás érkezett	DCE → DTE

A vonalak jelszintje:

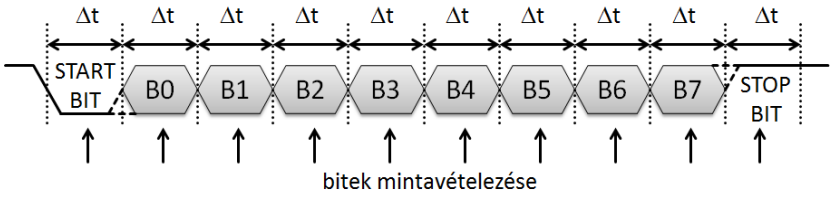
Feszültségszint a kimeneteken	Logikai állapot	Vezérlőjel állapot
-5 .. -25 V	1	kikapcsolva
+5 .. +25 V	0	bekapcsolva

Jelenleg az RS-232 szabványt már ritkán használjuk modemek vezérlésére; leginkább az iparban fordul elő, ahol berendezéseket és műszereket vezérel. Ekkor az egyes vonalak más szerepet kaphatnak. A feladatok egy részében a GND mellett csak a TxD és RxD vonalakat használják a kommunikációra, így három vezeték elég a kapcsolat fenntartásához. Ha fontos az adatátvitel ütemezése (biztosítani, hogy az egyik vagy a másik fél alkalmas az adatok fogadására), akkor *handshaking* vonalakat is alkalmaznak, ezek közül az RTS/CTS a legelterjedtebb. Az eszközök egy része magát DTE-nek tekinti, így ha ilyen eszközöket kötünk össze, akkor a vezetékekben keresztbe kell kötni az egyes vonalakat. A jelszintek sem feltétlenül követik a szabványban megadottat, elterjedt, hogy TTL jelszinteket alkalmaznak a kommunikációra.



8.3. ábra: A kommunikációra szolgáló vezetékek keresztbekötése

A kommunikáció során az RxD és a TxD vonalakon keresztül mennek át az adatok. A kommunikáció aszinkron, vagyis nincs olyan vezeték a rendszerben (*clock*), ami biztosítaná az adatátvitel ütemezését. Mindkét fél saját órajelgenerátorral dolgozik, mely meghatározza a kommunikáció sebességét. Utóbbit *Baud Rate*-nak hívjuk, és meg kell egyezzen mindkét félnél. A vezeték alapállapota az 1 logikai szint. A kommunikáció indulásánál egy bitnyi ideig alacsony szintűre vált, ez a start bit. Ezek után következnek egymás után az egyes bitek, a legalacsonyabb helyi értékűvel kezdve. Az átvitt bitek száma 5 bit és 9 bit között változhat. A legutolsó bit lehet paritásbit is. Az átvitel végén következik a stop bit, mely biztosítja, hogy a berendezések alapállapotba kerülhessenek. A stop bit lehet 1, 1,5 és 2 bit hosszúságú.



8.4. ábra: A kommunikáció idődiagramja

Példái ipari buszokra

RS-422 és RS-485

Az RS-232 szabvány nem alkalmas nagy sebességű adatátvitelre, nagyobb távolságokra. Erre a feladatra találták ki az RS-422 és RS-485 szabványokat, amelyek már differenciálisan viszik át a jeleket (akár 10 MBit/s, 1500 m). Az RS-422 szabvány esetén két vezetékpár viszi át a jeleket, külön-külön mindkét irányba, az RS-485 szabvány esetén egy vezetékpár van csak, maguk a kommunikáló felek döntik el, hogy melyik az adó, és melyik a fogadó fél. Ezekre a szabványokra számos magasabb szintű protokoll épül rá, pl. a *HART protocol*, *Profibus*.

GPIB (IEEE-488)

Ez egy széleskörűen elterjedt busz, elsősorban műszerek és automatizált mérőrendszerek vezérlésére. Az adatok átvitele párhuzamos, a csatlakozó pedig speciális. Egyszerre akár több eszközt is felfűzhetünk egy GPIB buszra. Hátránya, hogy költséges.

CAN-Busz

Alacsony költségű, elsősorban járműelektronikára tervezett busz. A CAN-Busz a prioritások kezelésével megvalósuló biztonságos szinkronizált kommunikáció. Négy vezetéket használ fel: differenciális adatátvitelt, földet valamint egy tápfeszültséget. Egy vezetékre számos eszközt lehet illeszteni. Az üzenetek prioritása az üzenet azonosítójába van kódolva, ha egyszerre több eszköz is megpróbálna üzenetet küldeni, a magasabb prioritású üzenet jut célba, az alacsonyabb prioritású, „recesszív” biteket tartalmazó üzenet küldése pedig megszakad.

A CAN (*controller area network*) szabványra számos magasabb szintű réteg épül, például a *DeviceNet*, *VSCP*.

PROFINET

Az Ethernet szabványra épül, valós idejű kommunikációt is lehetővé tesz (akár 1 ms alatti ciklusidővel).

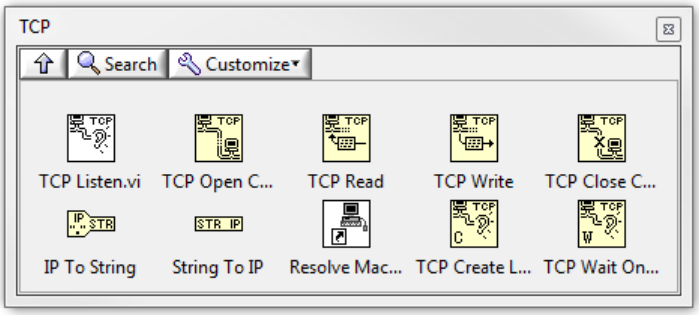
SCPI

Az SCPI (*Standard Commands for Programmable Instruments*) protokollt műszerek vezérlésére tervezték, és számos fizikai rétegre épülhet (GPIB, RS-232, Ethernet, USB) [26]. A parancsok szöveges üzenetekből állnak (pl. „MEASURE:VOLTAGE:DC?<nl>”). Az egyes parancsok egy fa struktúrába rendezhetők, a szabvány különböző műszerosztályokat definiál, melyeknek szabványos parancsaik vannak.

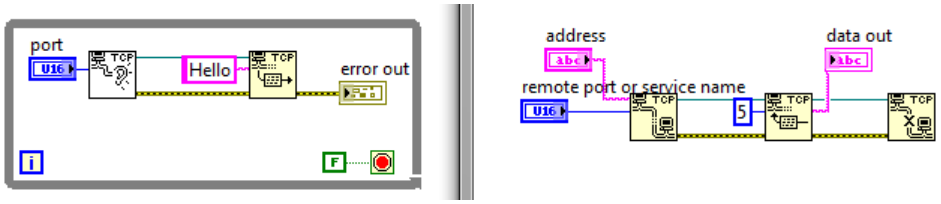
Kommunikáció TCP protokoll segítségével

A TCP protokoll az egyik leggyakrabban használt internetes kommunikációs protokoll, mely biztosítja, hogy az adatok hibamentesen és sorrendhelyesen jussanak el egyik alkalmazástól a másikig. Ennek segítségével könnyen valósíthatók meg olyan alkalmazások, melyek egymással megbízhatóan cserélnek adatokat.

A kommunikáció során van egy szerver, aki várakozik a kliens csatlakozására. A csatlakozáshoz a kliensnek ismernie kell a szerver IP címét, valamint azt a portszámot, ahova kapcsolódhat. A kapcsolat során az adatok áramlása két irányú, mind a szerver, mind a kliens tud adatokat fogadni és küldeni egy-egy csatornán. Az adatok megfelelő átviteléhez szükség van az erre a rétegre épített további protokollra, ahol megvan, hogy az egyes adatok milyen méretűek, és hogyan kell továbbítani a kommunikációs csatornán.



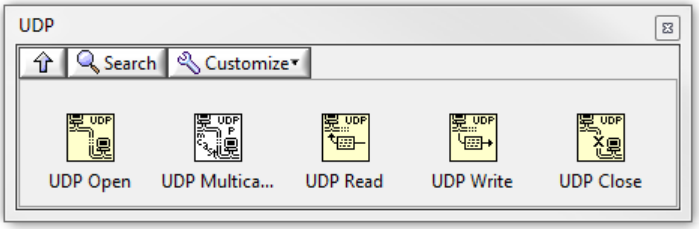
8.5. ábra: A TCP protokoll kezelésére szolgáló paletta



8.6. ábra: Egyszerű példa TCP kommunikációra. A bal oldali a szerver, a jobb oldali pedig a kliensoldal. A szerver által létrehozott csatlakozás megszűnik, amikor a kliens lezárja a kapcsolatot, de komolyabb alkalmazások esetén illik azt megfelelően a szerver oldalon is megfelelően lezárni, a felmerülő eseményeket, hibákat kezelni.

Az UDP protokoll tulajdonságai

Az UDP protokoll egy egyszerű adatküldési modell, mely segítségével üzeneteket vagy adatcsomagokat küldhetünk egyik gépről a másikra. A kommunikációs során nincs szükség állandó kapcsolatra a küldő és a fogadó között, ráadásul lehetőség van egyszerre több kliens számára is üzenetet küldeni egy (Multicast). A TCP kommunikációval ellentétben nincs szükség kapcsolat felépítésére, handshake-ing-ra, valamint az adatok rendezésére, így használata előnyös valós idejű alkalmazások fejlesztésénél. Ugyanakkor UDP protokoll esetén nincs biztosítva, hogy az adatcsomag valóban megérkezik-e rendeltetési helyére, az elveszett csomagok nincsenek automatikusan újraküldve. Ha biztosítani kell az adatok biztonságos megérkezését, azt az UDP-re épülő alkalmazásnak kell biztosítania (megfelelő timeout-ok, visszajelzések és újraküldések segítségével).



8.7. ábra: Az UDP protokoll kezelésére szolgáló paletta

Feladatok

1. feladat

Vizsgálja meg az RS-232 kommunikáció jeleit oszcilloszkópon! Határozza meg a Baud-rate-t az mérés alapján! Azonosítsa a jel egyes részeit!

2 feladat

Hozzon létre chatprogramot, ahol az egyik gépen beírt szöveg megjelenik a másik gépen és viszont!

3. feladat

Továbbítsa egy mérés adatait az egyik gépről a másikra TCP majd UDP protokoll segítségével! Hasonlítsa össze a két megoldást!

4. feladat

Hozzon létre egy egyszerű HTTP szervert, mely beágyazza a mérés eredményeit egy egyszerű weboldalba! Tesztelje a szerver működését egy másik gépről!

9. Távmérés és DataSocket technológia

A mérés és vezérléstechnikában sűrűn előfordul, hogy a műszereket vezérlő számítógép, és a mérések kiértékelése és vezérlése különböző helyeken van. Hasonlóan, számos mérést vezérlő eszközön LabVIEW vagy hasonló szoftver fut, mely a vezérlő számítógéppel csak interneten keresztül kommunikál. Ilyenkor szükség van a megfelelő protokollokra ahhoz, hogy az adatok áramlása megfelelő legyen. Biztosítani kell a megfelelő átviteli sebességet, a kommunikáció biztonságát, és megfelelő szabványokhoz való illesztését. A LabVIEW számos eszközt biztosít ahhoz, hogy minél kevesebb időbefektetéssel ezeket megoldjuk.

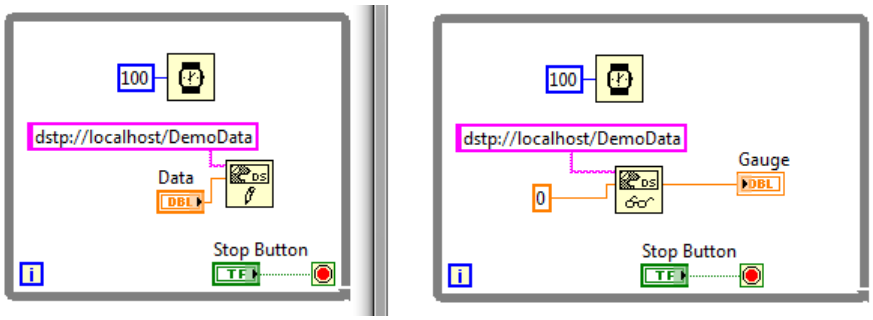
A legegyszerűbb módszer a LabVIEW segítségével való távmérésre a LabVIEW *Remote Panels*. Ekkor lényegében a szervergépen futó LabVIEW program előlapját érhetjük el egy távoli gépről. A szerver gépen a *Tools* menü *Web Publishing Tool..,* opciót választva konfigurálhatjuk a program megosztását, e mellett pedig elindíthatjuk a webszervert. A *Web Application Server*-t a *Tools/Option* menüben is konfigurálhatjuk A kliensgépen egy böngészőre illetve a LabVIEW futtatókörnyezetre van szükség a *Remote Panel* megnyitására. A böngészőben a *Request Control of VI* valamint a *Release Control of VI* szerezhetjük meg az irányítást vagy szakíthatjuk meg a kapcsolatot.

Ha adatokat akarunk cserélni két gép között, kézenfekvő megoldás a TCP vagy UDP-n alapuló adatkapcsolat. Ha ezekre a megoldásokra akarjuk alapozni a kommunikációt, fontos tudni, hogy a kapcsolat bármikor megszakadhat, az UDP csomagok pedig eltűnhetnek. Programunknak kezelnie kell ezeket a problémákat. E mellett küldés előtt az adatainkat adatainkat string-é kell alakítani (pl. Flatten to String), olvasáskor pedig visszaalakítani az eredeti formátumba. A TCP vagy UDP protokollok előnye, hogy könnyen tudunk olyan nem LabVIEW alkalmazásokat készíteni, melyek kommunikálnak LabVIEW programjainkkal.

A DataSocket technika

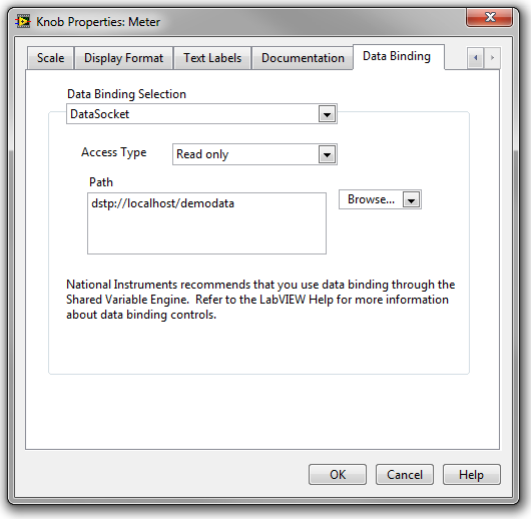
A *DataSocket* technika lehetőséget ad arra, hogy egyszerűen megosszunk és olvassunk adatokat a különböző programok között [14]. A kommunikáció a TCP/IP adatkapcsolatokon alapul és különböző protokollokra épülhet (*DSTP*, *OPC*, *LOOKOUT*, HTTP, FTP, lokális fájl).

DSTP esetén szükség van egy *DataSocket* serverre, mely biztosítja a kommunikációt az adat szolgáltatója (*publisher*) és a kliens között. Ez futhat az aktuális, vagy egy távoli gépen is. Ahhoz, hogy használni lehessen, szükséges a szolgáltatás engedélyezése a tűzfalban valamint a *DataSocket* server futására. A szerver, valami konfigurációja a *Start menü/National Instruments/Datasocket* mappában érhető el.



9.1. ábra: Adatok megosztása és beolvasása a DataSocket technikával

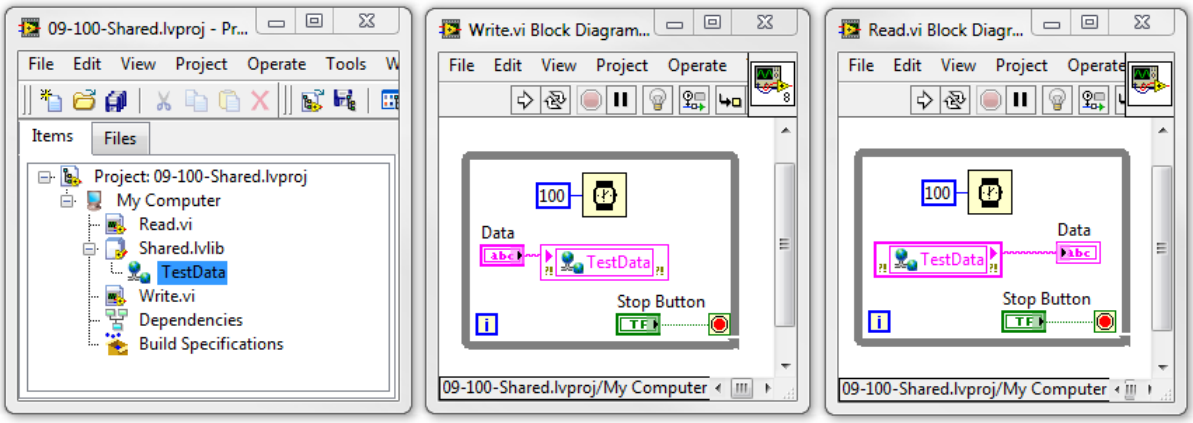
Lehetőség van arra is, hogy egy *Control* vagy *Indicator*, valamint egy *DataSocket* között közvetlenül megadjuk a kapcsolatot (*Data Binding*). Írni is és olvasni is tudjuk az adott változót. A kijelző mellett egy kis zöld ikon jelzi a kapcsolat létrejöttét.



9.2. ábra: Adatkötés egy *Control/Indicator* és a *DataSocket* változó között

Megosztott változók: Shared variables

A *shared variables* technológiát hasonló célokra használhatjuk, mint a *DataSocket* technológiát, azonban sokkal több funkciót biztosít számunkra [15]. Ahhoz azonban hogy használni tudjuk, szükséges, hogy alkalmazásunkat egy projekten belül készítsük el. Magát a megosztott változót ezen a projekten belül hozhatjuk létre. Számos adattípus közül választhatunk, e mellett megválaszthatjuk, hogy szeretnénk-e bufferelni az adatokat, vagy valós idejű FIFO-t hozzárendelni. Utóbbiak arra szolgálnak, hogy ideiglenesen tárolják az adatokat, így elkerülhetők az adatveszteségek.



9.3. ábra: A shared variables használata

A *DataSocket* technológiához hasonlóan a *shared variables* esetén is van lehetőség arra, hogy egy előlapi objektumot hozzákössünk egy elosztott változóhoz. Fontos, hogy ahhoz, hogy jól működjön ez a megoldás, a LabVIEW-nak be kell töltenie az előlapot, vagyis SubVI-ok esetén nem célszerű ennek a megoldásnak a használata.

Bár a *shared variables*-t elsősorban különböző műszerekkel való kommunikációra találták ki, használható számítógépek közötti kommunikációra is [16].

Feladatok

1. feladat

Készítsen olyan megoldást, ahol az adatgyűjtő műszert egy távoli gépről vezérli. A távoli gépen lehessen kimenő feszültséget állítani, valamint beolvasni két csatorna mért értékét. Utóbbiak legyenek megjelenítve egy *Waveform Chart*-on is.

10. Valós idejű rendszerek programozása

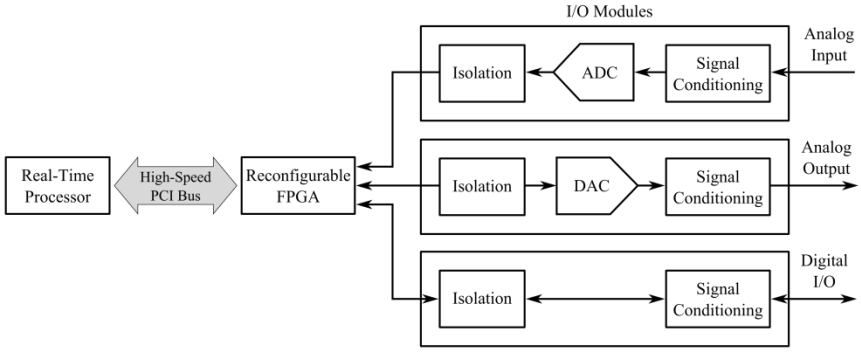
Ebben a fejezetben a National Instruments által gyártott cRIO és sbRIO valós idejű platformok programozását tekintjük át röviden. Valós idejű rendszerek esetén fontos követelmény, hogy bizonyos eseményekre (külső jelek vagy megadott időzítés) megbízhatóan, mindig a megadott időkorláton belül szülessen meg a válasz. Hagyományos operációs rendszerek esetén (pl. Windows) ez nem garantálható, a háttérben futó feladatok (pl. vírusirtók) akár igen hosszú ideig lefoglalhatják a gépet, miközben az nem reagál az eseményekre, a rövid válaszidőt nagy teljesítményű gépek esetén sem lehet garantálni. A determinisztikus válaszidő mellett a valós idejű rendszereknél fontos szempont, hogy azok hosszú ideig, akár évtizedekig, megszakítás nélkül megbízhatóan működjenek. Ennek megfelelően a hardver komponensek is nagyon magas minőséggel és megbízhatósággal rendelkeznek. Szabályozások és vezérlések megvalósítására elsősorban valós idejű rendszereket használnak mind az iparban, mind a kutatásban.

Legfontosabb fogalmak:

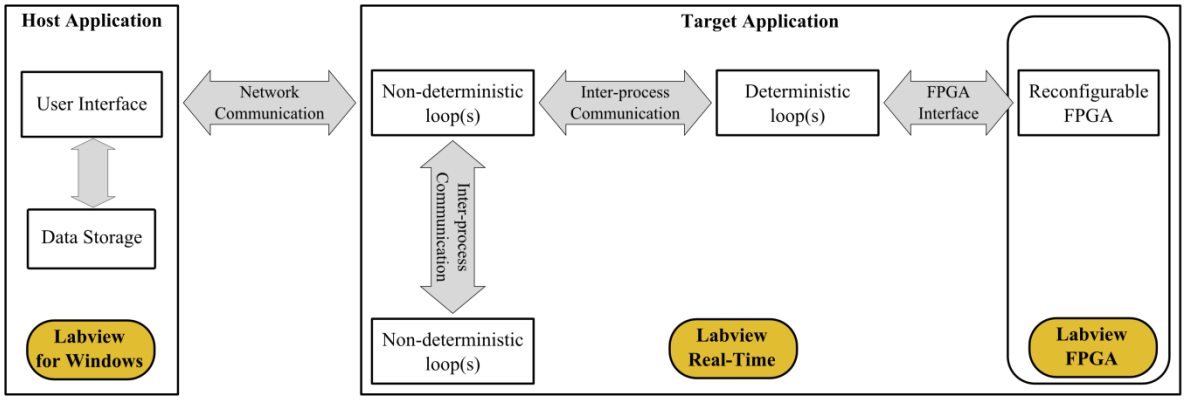
- Válaszidő (*response time*) – mennyi idő alatt válaszol a rendszer egy külső ingerre
- Ciklusidő (*loop cycle time*) – egy-egy szabályozási/vezérlési ciklus végrehajtásának ideje
- Jitter – a válaszidő vagy ciklusidő véletlenszerű eltérése a kívánt értékektől
- Determinisztikus feladat – egy feladat, mely mindig a megadott időkeretek között végre kell hajtódjon
- Prioritás – az egyes feladatok elsőbbsége a többi feladathoz képest. Valós idejű operációs rendszerekben a magas prioritású feladatok lesznek először végrehajtva.

cRIO rendszerek felépítése

A cRIO rendszer egy valós idejű processzorból, egy FPGA-t tartalmazó hátlapból (*chassis*) és különböző *c-series* modulokból áll. A valós idejű rész tartalmaz egy processzort, memóriát és háttértárat, valamint ezen fut a valós idejű operációs rendszer. Ez a processzor a LabVIEW Real-Time moduljával programozható. Az FPGA egy programozható logikai áramkör, mely akár több százezer logikai kaput tartalmaz és számos feladatot képes végrehajtani párhuzamosan, nagy sebességgel. Nem fut rajta operációs rendszer és a LabVIEW FPGA moduljával programozható. Az FPGA közvetlenül kezeli az egyes modulokat, melyen egy SUBD-15 csatlakozón keresztül csatlakoznak a hátlaphoz. A cRIO modulokat működés közben is ki lehet húzni a rendszerből valamint újra csatlakoztatni (hot-swappable). A modulok nagy része galvanikusan le van választva a vezérlőtől.



10.1. ábra: cRIO rendszer felépítésének blokkvázlata



10.2. ábra: cRIO szoftver architektúra

Egy elkészített cRIO alkalmazás tipikusan három fő részből áll. Az számítógépen futó alkalmazás (*host application*), valósítja meg a felhasználóval való kommunikációt valamint a nagyobb erőforrások igénylő off-line adatfeldolgozást és adatmentést. A valós idejű rendszeren fut a *target application*, mely számos különálló szálat futtathat egy időben, ezek közül legtöbbször egy ciklus valós idejű vezérlést hajt végre (*deterministic loop*). Ezek mellett a nagy sebességű valós idejű feldolgozásért az FPGA-n futó kód felelős (*FPGA application*). Az egyes programok és szálak különböző módon küldhetnek egymásnak adatokat, erről egy későbbi alfejezetben lesz bővebben szó.

A példaként bemutatott hardver egy egybeintegrált cRIO-9076-os vezérlőn alapul, mely a 400 MHz-es processzor mellett egy Xilinx Spartan-6 LX 45-ös FPGA-t tartalmaz. A processzor számára 512 MB háttértár és 256 MB RAM áll rendelkezésre. Az FPGA 27 288 logikai egységet tartalmaz 2 MBit RAM mellett. A modulok számára 4 *slot* áll rendelkezésre.

10.3. ábra: Példa cRIO hardver cRIO-9076 vezérlővel és a következő modulokkal: NI 9215, NI 9263, NI 9401 és NI 9403

Az **NI 9215** modul 4 csatornát tud szimultán mintavételezni legfeljebb 100 kHz mintavételi sebességgel, 16 bit felbontás mellett. Mindegyik bemenet differenciális. Két kivitelben készül a modul. A BNC csatlakozókkal ellátott modul esetén a BNC-k külső érintkezője egy 1 MΩ-os ellenálláson van a referenciapontra (COM) kötve. A sorkapoccsal ellátott modul esetén mindkét érintkező nagyimpedanciás, ahhoz, hogy a mérés megfelelő legyen, a COM csatlakozót a külső áramkör referenciapontjával (föld) össze kell kötni.

Az **NI 9263** modul 4 darab 16 bit-es A/D konvertert tartalmaz, ezeket 100 kHz-es frekvenciával szimultán lehet frissíteni. Egy-egy csatornából legfeljebb 1 mA áramot lehet kivenni, e felett már nem teljesíti a specifikációkat. Ha ennél nagyobb kimenő áramra van szükség külső követő erősítőket kell alkalmazni.

Az **NI 9401** modul 8 digitális TTL vonalat tartalmaz, ezek frissítési ideje 100 ns. A vonalak irányát négyes csoportokban lehet megadni, bemenetként vagy kimenetként. Az FPGA segítségével az IO vonalak speciális feladatokat is elláthatnak, pl. PWM jelek előállítása, kvadratúra enkóderek jelének feldolgozása, különböző számláló üzemmódok. Fontos, hogy az összes vonalat csak egyféleképp lehet konfigurálni (pl. mindegyik PWM kimenet), az egyes vonalak beállításai (PWM periódusidő) viszont egyedileg megadhatók. Az egyes kimenetek legfeljebb 2 mA-es árammal terhelhetők.

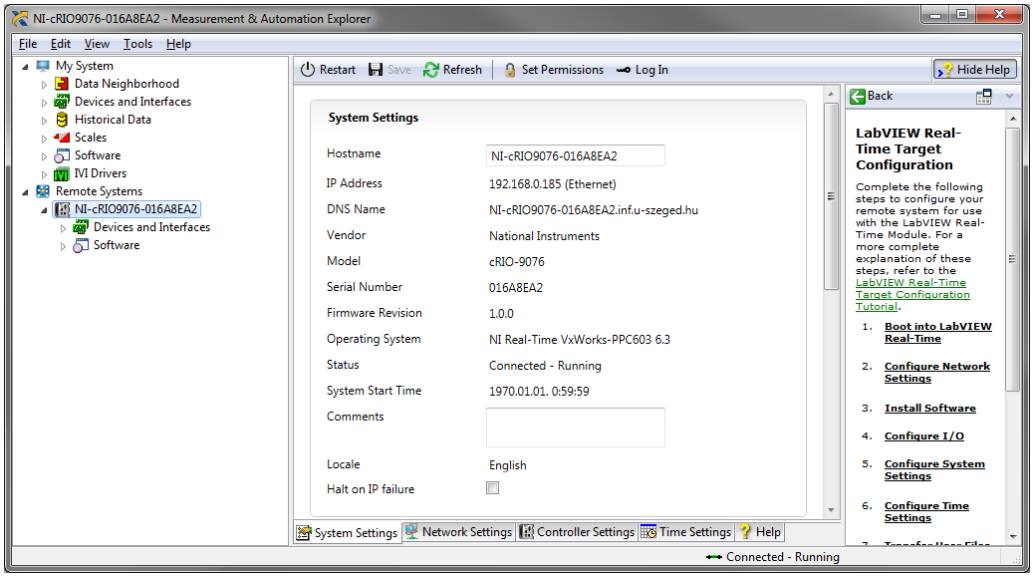
Az **NI 9403** modul 32 digitális TTL vonalat tartalmaz. Az egyes vonalak irányát egyesével lehet meghatározni. A vonalak frissítési ideje 7-8 µs, az irányváltási ideje pedig 18 µs.

cRIO eszközök konfigurálása

A csatlakoztatott eszközök felderítésére a *Measurement & Automation Explorer* (*MAX*) használható. Hacsak nincsenek speciális igények, célszerű, hogy ha mind a fejlesztő számítógép, mind pedig a cRIO eszköz ugyanazon alhálózaton belül van, e mellett pedig az IP címek kiosztása DHCP protokollal történik. E feltételek biztosítását laborjainkon egy routerrel biztosítjuk, melyhez mind a fejlesztő számítógépet, mind a cRIO eszközt csatlakoztatni lehet. A helyes működéshez elengedhetetlen, hogy a tűzfal átengedje a szükséges hálózati kommunikációt, Windowsban az alhálózat legyen Otthoni vagy Munkahelyi hálózat, a tűzfal összes kérdésére pedig az engedélyezés opciót kell kiválasztani. Ezen beállítások rendszergazdai jelszót igényelhetnek, elmulasztásuk esetén a fejezetben bemutatott lépések nem fognak működni.

A MAX segítségével többek között a következő műveleteket hajthatjuk végre a cRIO eszközön:

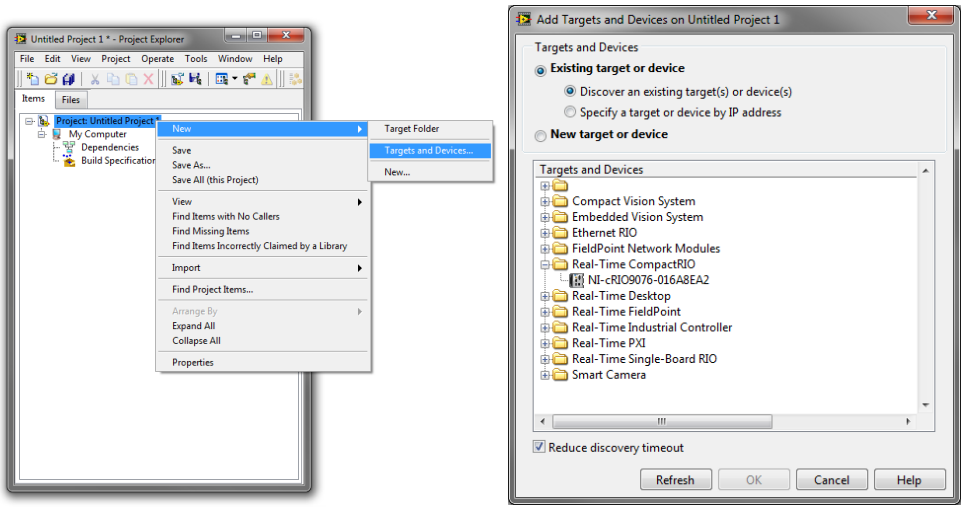
- Hálózati beállítások konfigurálása, elsősorban akkor célszerű, hogy ha nem használhatunk DHCP protokollt vagy statikus IP címet kell hozzárendeljünk az eszközhöz.
- Eszköz újraindítása.
- Formázás segítségével törölhetjük az eszközön lévő összes szoftvert és beállítást. Ezt követően telepíteni kell a működéshez szükséges szoftvereket.
- Szoftverek telepítése: itt választhatjuk ki a szükséges valós idejű szoftvereket (*LabVIEW Real-Time*, *NI compactRIO*), valamint a szükséges komponenseket (*add-ons*). Utóbbiak nélkülözhetetlenek számos funkció használatához. Fejlesztés közben, ha még nem tudjuk mely funkciókra lesz szükség, célszerű az összeset feltelepíteni, végleges program esetén viszont az erőforrásokkal való takarékosság jegyében csak a szükségeseket. Ha a számítógépen futó fejlesztőkörnyezet nem kompatibilis a cRIO-n lévővel, akkor formázás után lehet újratelepíteni az aktuális verziót.



10.4. ábra: cRIO adatai az NI MAX programban

cRIO eszköz programázásához létre kell hoznunk egy projektet. A LabVIEW projekt példákat mintákat (*Template*) is felajánl (pl. LabVIEW Real-Time Control on CompactRIO , LabVIEW FPGA Project). A következőkben azt mutatjuk be, hogy egy üres projektből kiindulva hogyan készíthetjük el a projektünket.

Az üres projekt létrehozását követően a projekt gyökerére jobb gombbal kattintva a *New / Target and Devices* opciót kell választani. Amennyiben az eszközünket el tudjuk érni hálózaton keresztül, az *Existing target or device* opciót válasszuk, majd keressük meg az eszközünket a *Real-Time CompactRIO* mappában. Amennyiben az eszközünk aktuálisan nem érhető el a *New target or device* opcióval létrehozhatunk egyet, melyet a későbbiekben manuálisan konfigurálhatjuk, hozzáadhatunk új modulokat.



10.5. ábra: Projekt létrehozása

Létező eszköz kiválasztása esetén a LabVIEW rákérdez, hogy milyen programozási módban kívánjuk használni az eszközt (a kérdés elmaradása hibára utal). A következő programozási módokban használhatjuk cRIO eszközeinket:

- *Scan interface* – a modulokat közvetlenül programozhatjuk a *Real-Time* programból, kimenetek és bemenetek elérését a *Scan Engine* biztosítja. Ez a legegyszerűbb módja a cRIO használatához, elsősorban akkor célszerű használni, ha a ciklusidő nem kell

gyorsabb legyen mint 1 ms. A *Scan Engine* frissítési frekvenciáját a cRIO target *Properties* menüjében tudjuk állítani, 1 ms alatti érték a rendszer stabilitása nem biztosított.

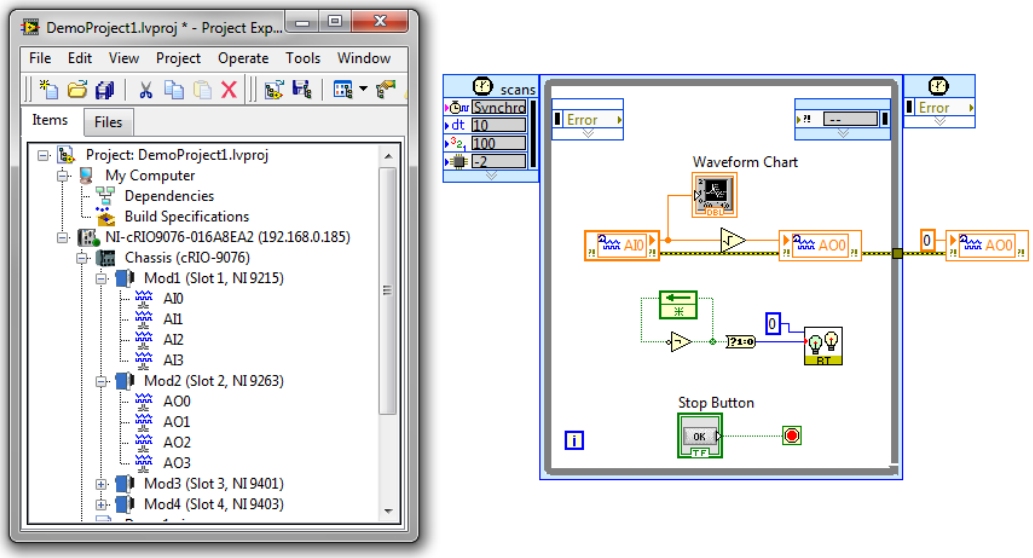
- *LabVIEW FPGA Interface* – a modulokat közvetlenül az FPGA kódból vezéreljük. Ezzel az üzemmóddal érhető el a leggyorsabb válaszidő, mely akár 10 ns-os nagyságrendű is lehet. Hátránya, hogy mind az FPGA-t mind pedig a valós idejű processzort programoznunk kell
- *Mixed mode* – Az egyes modulokat a *Scan inferface* segítségével érjük el, míg a többit az FPGA-ból programozzuk közvetlenül.

A VI-ok projektben elfoglalt helye határozza meg, hogy azok hol hajtódnak végre. A következő opciók közül választhatunk:

- *My Computer* – a kód a számítógépen fog futni (*Host Application*). Fő feladatai többek között a felhasználói interfész megvalósítása, adatok mentése, off-line analízis.
- *RT CompactRIO Target* – a kód a valós idejű processzoron fut (*Target Applicaiton*).
- FPGA Target – a kódot az NI FPGA modul segítségével készíthetjük el. Ahhoz, hogy a kódot az FPGA-n futtassuk, le kell fordítani. Ez a folyamat hosszú időt vehet igénybe (akár órákat), éppen ezért lehetőség van a kód működésének számítógépen való szimulálására.

Real-time processzor programozása

A 10.6. ábrán egy egyszerű példaprogram látható, mely egy mért feszültségérték gyökét előállítja az egyik feszültségkimeneten. E mellett a cRIO user LED-jét a végrehajtással szinkronban villogtatja.



10.6. ábra: Egyszerű példaprogram

Az I/O változókat a projekt ablakból drag-and-drop módszerrel tudjuk a programunkba berakni. A program futását a Run gomb megnyomásával kezdeményezhetjük, ekkor a LabVIEW először letölti a céleszközre, majd ha nem történt hiba, elindítja a programot. A program előlapját a számítógépről elérhetjük, a PC és a valós idejű processzor közötti kommunikációt a LabVIEW a háttérben megoldja.

A kódok a számítógéptől függetlenül is futtathatók. Ehhez először létre kell hozni a *Build Specification* elemben egy új *Real-Time Application*-t, ahol többek között specifikálhatjuk,

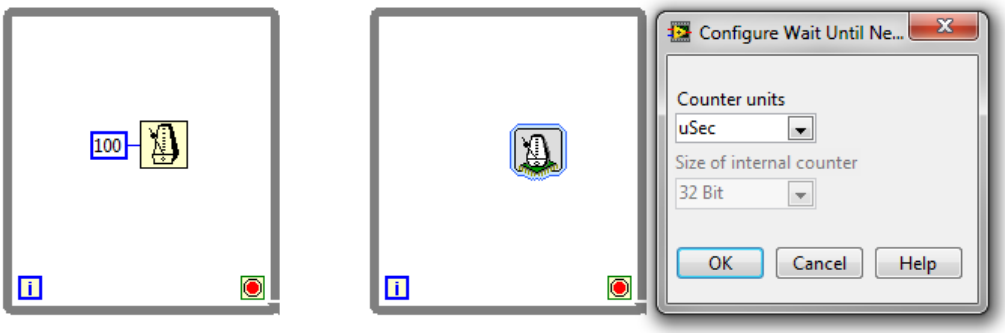
hogy mely VI lesz a fő alkalmazás (*Top-level VI*). Sikeres *Build*-et követően pedig megadhatjuk, hogy a kiválasztott alkalmazás induljon el automatikusan a cRIO bootolásakor (*Set as startup*), majd el is indíthatjuk (*Run as startup*).

Prioritások cRIO rendszerekben

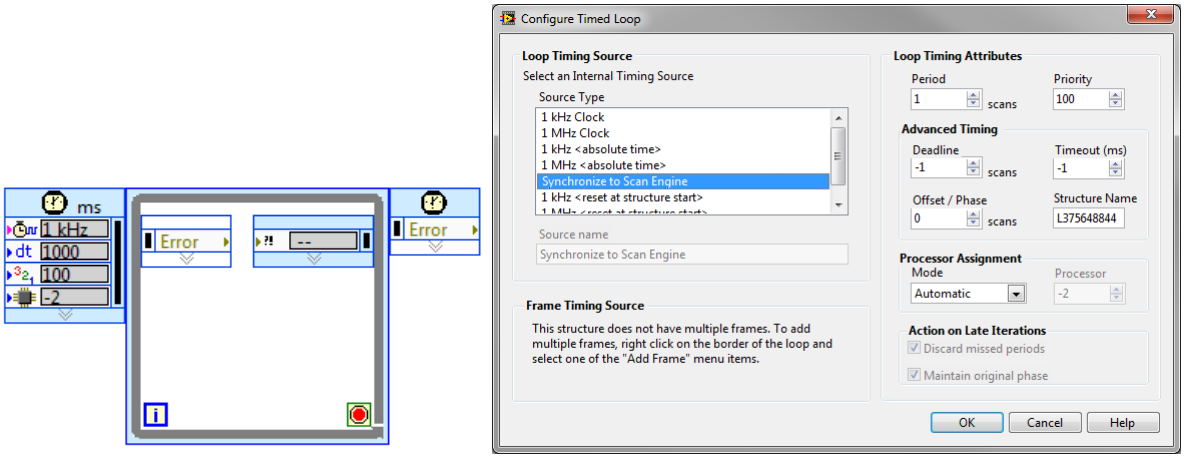
A jegyzetben korábban már szerepelt a while ciklus valamint a *Timed Loop*. A valós idejű processzor programozása közben ezek szerepe jelentősen eltér.

Hagyományos while ciklus esetén az időzítést a ciklusba rakott várakozással tudjuk meghatározni. Használhatjuk a *Wait (ms)* valamint a *Wait Until Next ms Multiple* vi-okat ms feletti időzítések beállítására. Valós idejű platformok esetén az *RT Timing* palettán lévő időzítő rutinok esetén sokkal nagyobb a rugalmasságunk, használhatunk µs-os időzítést, illetve a processzor órajelének többszörösét is megadhatjuk az időzítés alapjául (*Ticks*). Az így létrehozott ciklusok végrehajtása nem determinisztikus, nincs biztosítva a megadott időben való végrehajtás.

Determinisztikus végrehajtáshoz *Timed Loop* használata szükséges. Ezen ciklusok végrehajtása magasabb prioritással történik, így elsőbbséget végeznek a többi kód végrehajtásához képest. Különböző órajelforrásokhoz lehet szinkronizálni a végrehajtást, amennyiben I/O változókat használunk, akkor célszerű a *Scan Engine* végrehajtásához időzíteni.



10.7. ábra: Nem determinisztikus while ciklus időzítése



10.8. ábra: *Timed Loop* és konfigurációja

Amennyiben egyszerre több végrehajtási szál (pl. párhuzamos ciklusok) versenyeznek az erőforrásokért, akkor a prioritás az, ami meghatározza a végrehajtás elsőbbségét. A LabVIEW-ban a következő prioritási szintek vannak definiálva:

- Above time critical – legmagasabb prioritás, az operációs rendszer szintje, illetve ezzel a prioritással rendelkezik a Scan Engine.
- *Time critical* – legmagasabb prioritás ami egy VI-hoz rendelhető. Nem javasolt ezen opció használata.
- *Timed Loop*-ok szintje – a ciklusok két prioritási szint közötti prioritással rendelkeznek. A ciklusok prioritása e mellett azok *Priority* tulajdonságával egymás alá-fölé rendezhetők.
- *High*
- *Above normal*
- *Normal* – alap beállítás, ilyen prioritással rendelkezik többek között a beépített FTP szerver valamint a fejlesztői kapcsolat.
- *Background* – legalacsonyabb prioritási szint

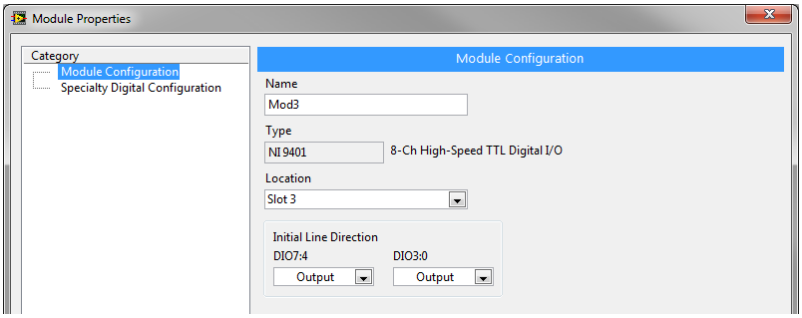
Egy VI prioritása annak *Properties* / *Execution* menüjében módosítható, amennyiben feltétlenül indokolt. Valós idejű rendszerekben a magas prioritású szálak végrehajtása elsőbbséget élve az alacsonyabb prioritású szálakhoz képest. Amennyiben a magas prioritású szál az összes processzoridőt elhasználja az alacsonyabb prioritású szálak egyáltalán nem hajtódnak végre (éhezés, *starving*). Pl. ha egy *Timed Loop* több időt igényelne, mint amennyi egyébként számára rendelkezésre áll, akkor folyamatosan futni fog, miközben a fejlesztői kapcsolat nem hajtódik végre, megszakad a kapcsolat a fejlesztőgép és a cRIO között.

Determinisztikus programok fejlesztése közben többek között a következő szempontokra kell figyelni:

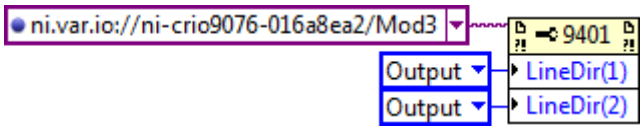
- Egy processzormag egyetlen determinisztikus feladat végrehajtására alkalmas, több determiniszikus szál esetén azok versenyezni fognak az erőforrásokért
- A processzor kihasználtság ne legyen 80 % felett, így mindig van némi tartalék a feladatok végrehajtására. A processzor kihasználtságát a *Tool* / *Distributed System Manager alkalmazás* segítségével lehet követni.
- Megfelelő hardver választása, melynek teljesítménye alkalmas a kívánt feladat végrehajtására
- Megosztott erőforrások kerülése (lokális / globális változók, semafor-ok, memóramenedzsment)
- Nem szükséges funkciók (pl. debuggolás) kikapcsolása
- Alacsony szintű funkciók használata, Express VI-ok kerülése

I/O modulok konfigurációja

A I/O modulok egy részénél (pl. digitális I/O modulok, univerzális analóg bemeneti modul) a bemenetek és kimenetek különböző módon konfigurálhatók. A konfigurálást két úton lehet megtenni. Egyrészt a modulra kattintva a *Properties* helyi menü segítségével. Másrészt pedig a LabVIEW kódból is el lehet végezni a modul tulajdonságainak (*Properties*) módosításával vagy a modulhoz tartozó függvények (*Methods*) meghívásával.



10.9. ábra: Az NI 9401 digitális modul konfigurációja a *Module Properties* ablak segítségével

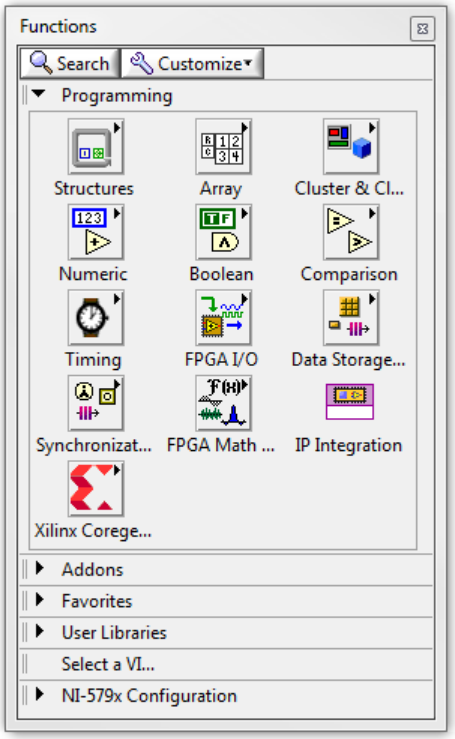


10.10. ábra: Az NI 9401 modul vonalainak kimenetként való konfigurálása LabVIEW kódból

FPGA programozása

A cRIO platform összes képességét az FPGA közvetlen programozásával lehet elérni. Legfontosabb előnye a nagy sebesség és gyors válaszidő (alapbeállítások mellett 25 ns). E mellett akár 200 MHz-es ciklusidőt is el lehet érni. A platform támogatja a teljesen párhuzamos feldolgozást, egyszerre több determinisztikus ciklus is futhat egymástól teljesen függetlenül. Az FPGA-n nem fut semmilyen operációs rendszer, ami bármilyen jittert okozhatna, e mellett pedig nagyon megbízható (nem fagy le). Közvetlen hozzáférést nyújt az egyes I/O modulokhoz, azok a lehető legnagyobb sebességgel kezelhetők, a frissítési gyakoriságot nem korlátozza a *Scanning engine* 1 ms-os ciklusideje. Bizonyos modulokat pedig csak az FPGA programozásával lehet kezelni (pl. az NI 9223 1 MHz-es A/D konverter modul).

Az FPGA programozása során viszont hamar szembesülünk azzal, hogy a valós idejű rendszerhez képest erősen korlátozottak az erőforrások, komolyabb kódok esetén szükség van az átgondolt tervezésre. Mint ahogy a 10.11. ábrán látható a felhasználható funkciók is erősen korlátozottak. Többek között nem lehet dupla pontosságú lebegőpontos számokat használni, helyette elsősorban egész és fixpontos számokkal célszerű számolni. Programozás közben csak 1D tömbök használhatók. Az FPGA paletta a 10.11. ábrán látható



10.11. ábra: Az FPGA paletta

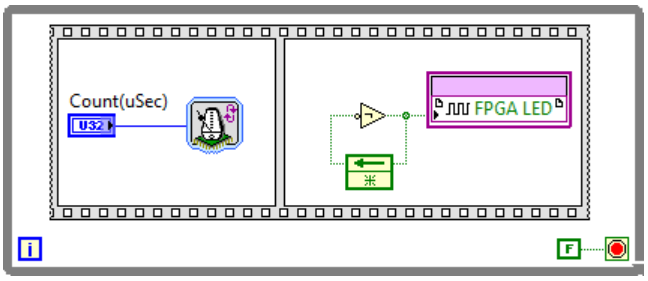
Az FPGA programot futtatás előtt le kell fordítani az FPGA által értelmezni tudott *bitfile*-ba. A fordításhoz először ki kell választani a megfelelő *Compile Server*-t. Ez lehet helyi, az aktuális fejlesztőgépen futó, hogy ha fel van telepítve a *XILINX compilation tools* modul, lehet egy távoli gép, vagy, megfelelő hozzáférés esetén a *LabVIEW FPGA Compile Cloud Service*. Fordításkor a LabVIEW legenerálja az ideiglenes fájlokat, mely alapján a XILINX fordító elvégzi a feladatot. Fordítás során a *Compilation Status* ablakban lehet követni a folyamatot valamint az előzetes jelentéseket. A fordítási folyamat hosszú időt vehet igénybe, akár órákat is. Éppen ezért célszerű a kód működését számítógépen való szimulált futással ellenőrizni (*FPGA Target* / *Properties* / *Debugging* / *Execure VI on Development Computer with Simulated I/O*).

FPGA programozás közben a „hagyományos” LabVIEW programozástól eltérő irányelveket kell követni:

- A *Top-Level VI* előlapi elemei a számítógéppel vagy a valós idejű processzorral való kommunikációra is szolgálnak, e miatt komoly erőforrásokat igényel implementációjuk. E miatt célszerű ezen elemek számát korlátozni, valamint nem előnyös nagy tömböket és clustereket megjeleníteni.
- Ahol csak lehet, törekedni kell a legkisebb adattípusok használatára (pl. ha csak 100-ig számolunk el, csak I8 változót használjunk)
- Osztás, maradékos osztás kerülése.
- Fixpontos számok használata. Fixpontos számoknál figyelni kell, hogy azok mérete a műveletek során ne növekedjen túlságosan meg.
- FPGA-ra optimalizált műveletek használata (*FPGA Math*)
- Hibavezetékek kerülése, a végrehajtás sorrendiségét *Flat Sequence Structure* segítségével lehet biztosítani

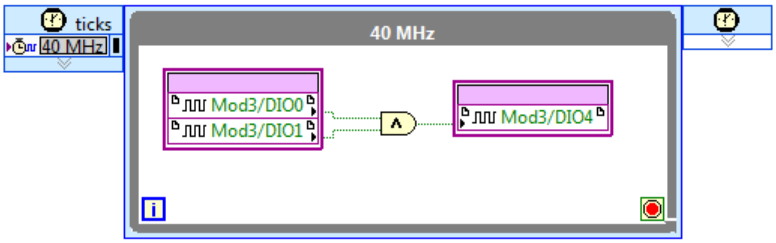
FPGA programozása közben az I/O változókat hasonló módon lehet lerakni, mint a valós idejű processzor programozása közben, a projekt ablakban lehet kiválasztani majd lerakni a

program diagramjába. A 10.12. ábrán egy egyszerű példaprogram látható, mely a cRIO előlapján lévő FPGA LED-et villogtatja a megadott gyakorisággal. A program futtatása hasonló is hasonló módon történik, mint a valós idejű rendszer esetén, a Run gomb megnyomásával elindul a folyamat, szükség esetén újra lesz fordítva a kód, majd letöltés után elindul az FPGA-n, miközben a számítógépen követni lehet az előlapi elemek állapotát, illetve módosítani a bemeneti változók értékét.



10.12. ábra: Egyszerű FPGA példaproram

FPGA esetén a Timed Loop speciális szerepet tölt be: a benne lévő elemek egyetlen órajelciklus alatt hajtódnak végre (Single Cycle Loop, SCL). További előnye, hogy a benne megvalósított műveletek sokkal kevesebb erőforrást igényelnek, mivel az adatok kapuzása tárolása (latch) csak a ciklus végén történik meg. Fontos, hogy az SCL estén az összes művelet végre kell tudjon hajtódni a megadott, általában rövid idő alatt. Amennyiben a fordító nem képes ennek megfelelő FPGA konfigurációt létrehozni, fordítási hiba jön létre. Az SCL ciklusokban használható műveletek száma erősen korlátozott, valamint csak bizonyos digitális I/O vonalak használhatók.



10.13. ábra: Egyciklusos while loop használata

Kommunikáció a különböző szálak között

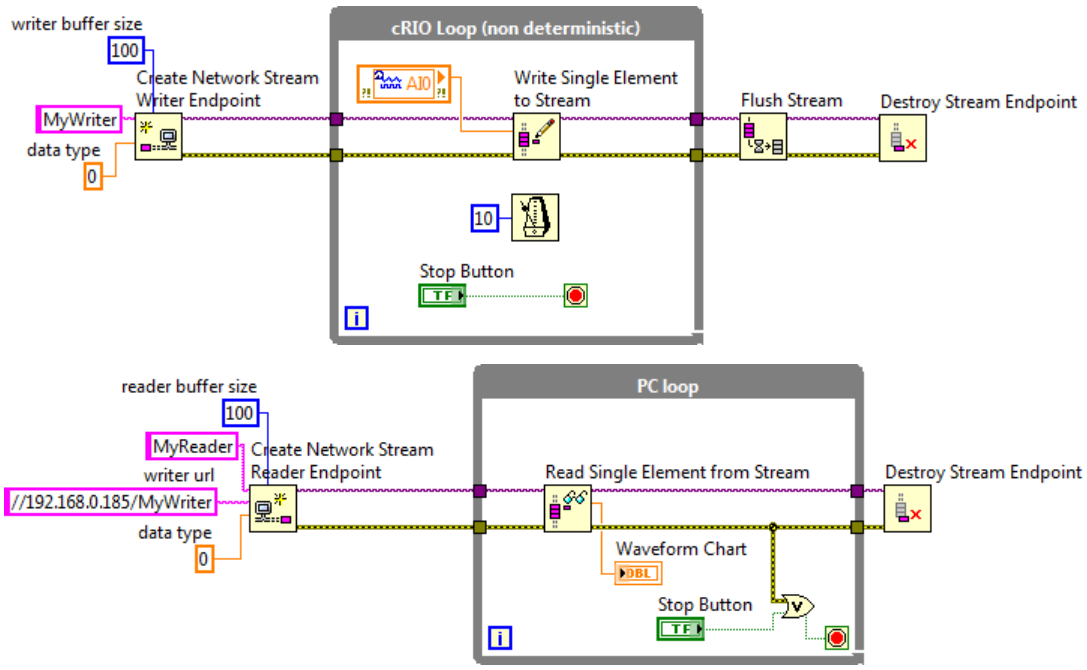
Valós idejű rendszerek programozása esetén számos szál és program fut párhuzamosan, ezek között pedig különböző adatokat kell cserélni, miközben biztosítjuk egyrészt az adatok integritását, másrészt pedig azt, hogy a determinisztikus ciklusok ne veszítsenek megbízhatóságukból. Ezen jegyzet keretein belül csak a kommunikáció alapvető módjai kerülnek szóba, a lehetséges megoldásokról bővebben a [29] irodalomban lehet olvasni. A kommunikációnak többek között a következő esetei különböztetjük meg:

- Aktuális érték megosztása, a korábbi értékekre nincs szükség
- Frissítések: aktuális érték megosztása csak változás esetén, megszakítások
- Adatfolyamok (stream): nagy mennyiségű adatot kell átvinni az egyik helyről a másikra, adatvesztés nélkül. Az adatok megérkezésének ideje nem kritikus.
- Üzenetek vagy parancsok: az adatok minél kevesebb látenciával kell megérkezzenek a célhoz, adatvesztés nem történhet.

Számítógép – cRIO kommunikáció

A számítógép és a valós idejű processzor közötti kommunikációt meg lehet valósítani alacsony szintű internetes protokollok segítségével (TCP/IP, UDP). A megoldás előnye a rugalmasság, a számítógépen tetszőleges operációs rendszerek és szoftverek futhatnak. Hátrányuk viszont, hogy ezekre a protokollokra egy saját protokollt kell építsünk, mely biztosítja az adatok megbízható átvitelét, valamint problémák esetén a kapcsolat újraépítését. Utóbbi problémák leküzdésére a LabVIEW számos beépített megoldással rendelkezik.

A 9. fejezetben bevezetett megosztott változók (*Shared Variable*) elsősorban az aktuális értékek valamint a frissítések átadására alkalmas. Megjegyzés: az I/O változokat is el lehet érni *Shared Variable*-ként (*I/O Variable*) a PC-ről, alapesetben ugyanis engedélyezve van hálózati megosztásuk (*Enable Network Publishing*). Amennyiben PC-ről közvetlenül szeretnénk elérni valós idejű ciklusokat, a változókban be kell kapcsolni a *RT FIFO* opciót. A FIFO hibakimenete jelzi a FIFO állapotát (*overflow*, *underflow*). *Single Element RT FIFO* esetén csak az utolsó adat van tárolva. Az *RT FIFO* csak a valós idejű processzoron belüli megbízható kommunikációt szolgálja, amíg az adatok át nem adódnak a hálózati kommunikációért felelős programmoduloknak. A hálózati átvitel bufferelésére szolgál a *Network / Use Buffering* opció, mely biztosítja, hogy hálózat esetleges lelassulása ne akadályozza a megbízható adatátvitelt.



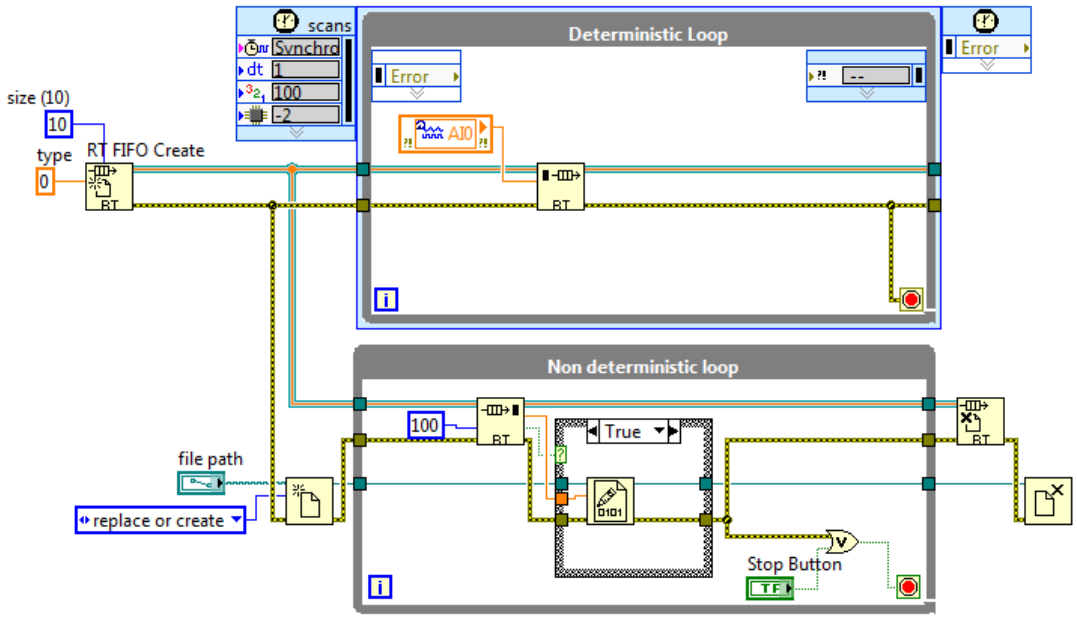
10.14. ábra: Adatok írása és olvasása *Network Stream* esetén

Nagyobb mennyiségű adat átvitelét a Network Stream megoldással érhetjük el, erről egy egyszerű példa a 10.14. ábrán látható.

Inter-Process kommunikáció

A megosztott változók a valós idejű processzoron futó szálak közötti kommunikációra is használhatók. Ekkor célszerű a *Single-Process shared variables* opciót választani. Amennyiben determinisztikus szál is érintett a kommunikációban, a *RT FIFO* opciót be kell kapcsolni.

Nagyobb mennyiségű adat megbízható átvitelére az *RT FIFO* eszköz szolgál (nevével ellentétben nem egyezik meg a korábban említett *RT FIFO* opcióval). A következő ábrán szerepel egy egyszerű példa a használatáról:



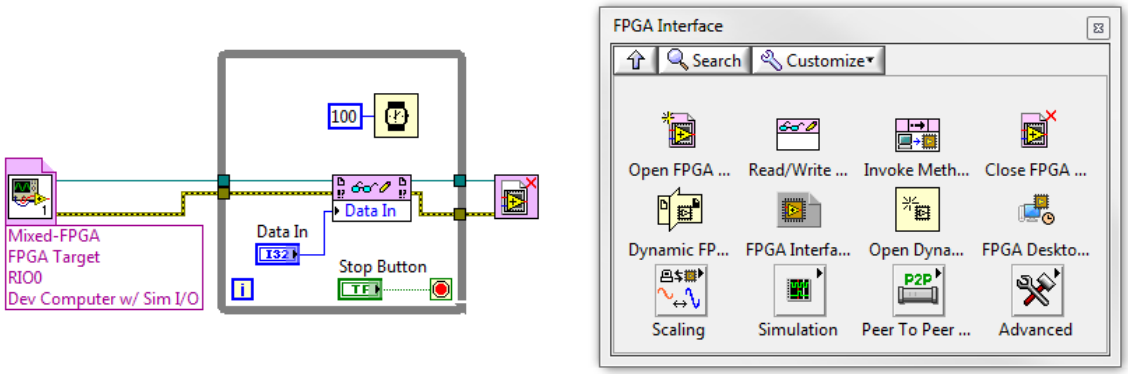
10.15. ábra: Egyszerű példa az *RT-FIFO* használatára

Kommunikáció megvalósítása az FPGA kódban

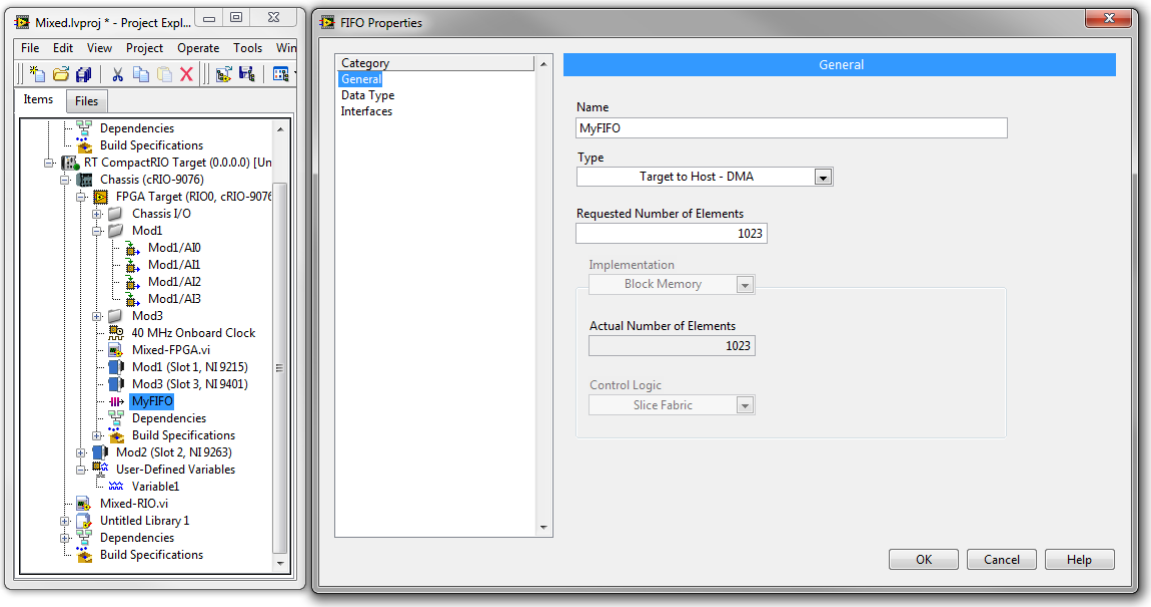
Az FPGA különböző programelemei közötti kommunikációra számos lehetőség van:

- Lokális változók: utolsó adatok megosztása (*Top-Level VI* esetén nagyobb erőforrásigény)
- Globális változók: utolsó adatok megosztása
- *Memory item*: utolsó adat megosztása az FPGA memóriája segítségével
- *Register items*: utolsó adat megoszása
- *Occurence*: ciklusok szinkronizálása
- FIFO: adatok pufferelt átvitele. A puffermérettől függően különböző opciókat célszerű választani: 100 adatig: *Flip-Flop*, 100-300 adat között: *Look-Up Table*, 300 adat felett: *Block Memory*.

A PC vagy a valós idejű processzorból az FPGA az *FPGA Interface* segítségével érhető el. Az *Open FPGA VI Reference* segítségével választható ki a megfelelő FPGA Top VI (helyi menü / *Configure* ...). Ezt követően a VI előlapi elemeit írni és olvasni is lehet.



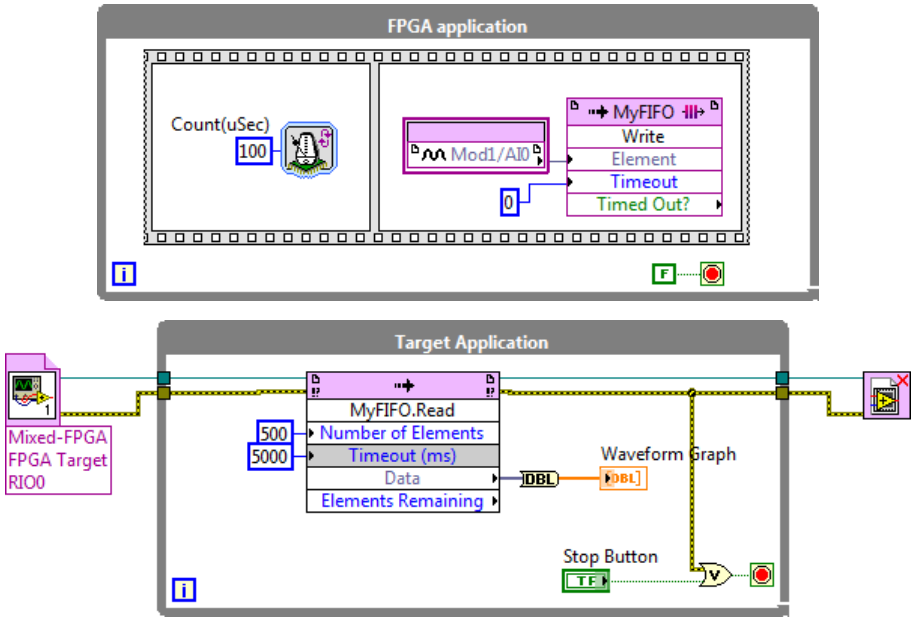
10.16. ábra: Az FPGA Interface használata előlapi eleme értékének módosítására



10.17. ábra: A DMA FIFO konfigurálása

Amennyiben a *Scanning engine* rendelkezésre áll, akkor létre lehet hozni saját I/O változókat (*User Defined I/O variables*). Ezek segítségével ugyanúgy lehet kommunikálni a FPGA kóddal, mint a modulok I/O változóival. A változókat a projekt ablakban a *Chassis* helyi menüjében a *New / User Defined Variable* menüben valósítható meg

Nagy mennyiségű adat veszteségmentes átvitelére a *DMA FIFO* használható. Megjegyzés: a cRIO szabad DMA csatornáinak száma korlátozott, amennyiben a *Scanning Engine* fut, az két csatornát is felhasznál.



10.18. ábra: Egyszerű példa a DMA FIFO használatára

Feladatok

1. feladat

Helyezze üzembe a cRIO eszközt!

Találja meg az eszközt a MAX segítségével!

Formázza a cRIO merevlemezét!

Telepítse a szükséges* szoftvereket!

2. feladat

Olvassa be egy potenciométer állását majd jelenítse meg állapotát egy LED-soron!

3. feladat

Vezérelje egy léptetőmotor mozgását a valós idejű processzor segítségével. A sebességet egy a cRIO analóg bemenetére kötött potenciométer segítségével szabályozza!

Valósítsa meg ugyanezt a feladatot az FPGA használatával!

Köszönetnyilvánítás

A jegyzet a TÁMOP-4.1.2.A/1-11/1-2011-0104 „*A felsőfokú informatikai oktatás minőségének fejlesztése, modernizációja*” program keretében történt.

A szerzők köszönetüket fejezik ki Dr. Szabó Istvánnak a Debreceni Tudományegyetem Szilárdtest Fizikai Tanszék vezetőjének a jegyzet gondos lektorálásáért valamint a konstruktív javaslataiért.

Szeretnénk megköszönni Makan Gergelynek, Nagy Tamásnak és Vadai Gergelynek a jegyzet ábráinak igényes elkészítéséért, hasznos tanácsaikért.

E mellett pedig szeretnénk megköszönni a National Instruments Hungary Kft.-nek, valamint képviselőjének, Litkei Mártonnak a mérőműszerek valamint a LabVIEW használatában nyújtott folyamatos támogatásukért.

Irodalomjegyzék

[1] Gerzson Miklós: *„Méréselmélet”*, 2011, Typotex Kiadó, Tankönyvtár, elérhető: http://www.tankonyvtar.hu/hu/tartalom/tamop425/0008_gerzson/

[2] Kemény Sándor, Deák András: *„Mérések tervezése és eredményeik értékelése”*, Műszaki Könyvkiadó, Budapest, 1990.

[3] Schnell László szerk.: *„Jelek és rendszerek méréstechnikája”*, Műszaki Könyvkiadó, Budapest, 1985.

[4] U. Tietze, Ch Schenk: *„Analóg és digitális áramkörök”*, Műszaki Könyvkiadó, Budapest, 1990.

[5] Ernest O. Doebelin: *“Measurement Systems Application and Design”*, McGraw-Hill Publishing Company, 1990

[6] Harry N. Norton: *“Handbook of Transducers”*, Prentice Hall PTR, New Jersey, 1989

[7] Lambert Miklós: *„Szenzorok – elmélet és gyakorlat”*, Invest – Marketing Bt., Budapest, 2009.

[8] Analog Devices: *“The Data Conversion Handbook”*, elérhető: http://www.analog.com/library/analogdialogue/archives/39-06/data_conversion_handbook.html

[9] National Instruments: *“Introduction to NI LabVIEW”*, elérhető: http://www.ni.com/gettingstarted/labviewbasics/

[10] National Instruments: Online LabVIEW és mérésadatgyűjtési tanfolyam 1. és 2., elérhető: http://zone.ni.com/wv/app/doc/p/id/wv-3220

[11] National Instruments: *“Instruments driver network”*, elérhető: http://search.ni.com/nisearch/app/main/p/ap/tech/lang/hu/pg/1/sn/ssnav:idr/

[12] National Instruments: *“Big physics”*, elérhető: http://www.ni.com/physics/

[13] National Instruments: *“The NI TDMS File Format”*, elérhető: http://www.ni.com/white-paper/3727/en

[14] National Instruments: *“DataSocket Tutorial”*, elérhető: http://www.ni.com/white-paper/3224/en

[15] National Instruments: *“Using the LabVIEW Shared Variable”*, elérhető: http://www.ni.com/white-paper/4679/en

[16] National Instruments: *“How Do I Set Up Shared Variables to Communicate Between Two PCs?”*, elérhető: http://digital.ni.com/public.nsf/allkb/7815BCE435DCC432862575DA006FEBF8

[17] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery: *„Numerical Recipes: The Art of Soft Computing”, Third Edition, Chapter 17.*, 2007, Cambridge University Press, elérhető: http://www.nr.com

[18] *“Digital Multimeter Kit”*, elérhető: http://www.electronickits.com/kit/complete/meas/m-2666k.pdf

[19] Michailovits Lehel szerk.: *„III. éves fizikai laboratóriumi gyakorlatok”*, JATEPress, Szeged, 2004.

[20] Pico Technology: *“PicoScope 5000 USB PC Oscilloscopes”*, elérhető: http://www.picotech.com/picoscope5200-pcos.html

[21] Tektronix: *“TDS2000C Digital Storage Oscilloscope”,* elérhető: http://www.tek.com/oscilloscope/tds2000-digital-storage-oscilloscope

[22] Rick Bittert et al: *“LabVIEW: Advanced Programming Techniques”*, CRC Press, 2006

[23] Peter A. Blume: *“The LabVIEW Style Book”*, Prentice Hall, 2007

[24] *„MA-DAQ mérőműszer dokumentációja”*, elérhető: http://www.noise.inf.u-szeged.hu/edudev/madaq/
[25] Future Technology Devices: *“FT2232H - Hi-Speed Dual USB UART/FIFO IC”*, elérhető: http://www.ftdichip.com/Products/ICs/FT2232H.htm
[26] IVI foundation, elérhető: http://www.ivifoundation.org/Default.aspx
[27] *“RXTX”*, elérhető: http://rxtx.qbang.org
[28] *Mellékletek a jegyzethez*, elérhetők: http://www.noise.inf.u-szeged.hu/Education/malj/
[29] *NI LabVIEW for CompactRIO Developer’s Guide*, elérhető: http://www.ni.com/compactriodevguide/