DEBRECENI EGYETEM INFORMATIKAI KAR
SZILÁRDTESTFIZIKAI TANSZÉK

VIRTUÁLIS OSZCILLOSZKÓP MEGVALÓSÍTÁSA PXI PLATFORMON LABVIEW-VAL

Konzulens:
Dr. Szabó István
Tanszékvezető egyetemi docens
valamint
Tóth István
Value Engineer, NI Hungary Kft.

Készítette:
Vas Dávid
Mérnök Informatikus B. Sc.
Mérés- és folyamatirányítás szakirány

Debrecen
2009.

Tartalomjegyzék

1. Bevezetés

1.1. Előszó

Az elektronikai eszközök gyártásának területén egyre nagyobb teret nyer magának az oszcilloszkóp, mint diagnosztikai eszköz használata. Széleskörű felhasználási körébe tartozik a különböző diagnosztikai mérések elvégzése, mely sokszor elengedhetetlen egy eszköz vagy egy áramköri lap funkcionális tesztjeinek elvégzéséhez, vagy egy eszközön hibás alkatrész megkereséséhez. A felhasználási köre szinte végtelen, az elektronikai fejlesztő- és gyártóiparban, autóiparban, kutatási területeken, egészségügyben, szolgáltatóiparban is alkalmazzák. Az elterjedtségének az oka az általános alkalmazhatóság, és a nagyfokú konfigurálhatóság. Ugyanúgy alkalmas általános célú mérésekre, mint egyetlen szituációhoz, vagy környezethez alkalmazkodó speciális mérésekhez (gondoljunk csak az EKG-készülékre). A népszerűségének a legfőbb oka a könnyen elsajátítható kezelés, és gyors beállítási lehetőség.

1.2. A diplomamunka célja

A szakdolgozat célkitűzése egy virtuális oszcilloszkóp kifejlesztése pontos felhasználói igények mentén, mely magába ötvözi a modern digitális oszcilloszkópok képességeit.

A program felhasználása annak elkészültét követően a National Instruments debreceni leányvállalatának gyártósorán történik majd, tesztelési és hibakeresési céllal. Ezért kiemelkedően fontos a modularitás, és a különböző eszközök támogatása. A virtuális műszerbe minden olyan funkció beépítésre kerül, amelyet bármely, NI oszcilloszkóp-kártya képes kihasználni, és a gyártásban szeretnének kihasználni, ezen túlmenően program jellege miatt, és a LabVIEW környezetnek köszönhetően nagyszerűen testre szabható hardver-specifikus beállításokkal is.

Mivel a dolgozat írása során a felhasználókkal való kapcsolattartásban a cég egyik dolgozója volt segítségemre, így folyamatosan figyelemmel kísérhette a munkát, és segítette azáltal, hogy minden igénynek megfeleljen a program. Mivel tesztelési és hibakeresési célokra lesz felhasználva a program, az elsődleges szempont a megvalósításnál az egyszerű kezelhetőség, és a gyors konfigurálhatóság volt. Erre azért van nagy szükség, mert a hibakeresés során nagyon sokféle jel előfordulhat.

A virtuális oszcilloszkóp előnye a digitális oszcilloszkópokkal szemben, hogy hatalmas mértékben skálázható, tehát képes működni egy asztali PC-n is egy USB-s szkópkártyával, vagy kihasználhatja a National Instruments PXI keretrendszerén keresztül az abba illeszthető nagysebességű digitalizálókat.

Mivel a modularitás a fő alapelv, ezért egy virtuális oszcilloszkóp jóval olcsóbb megoldás, főleg, mivel a cég maga gyártja a futtató hardvert, a program pedig továbbfejleszthető.

A program fejlesztése során az NI USB-5133 eszközét használtam, hordozhatósága miatt, mely az 1. ábrán látható.

1. ábra: NI USB-5133 oszcilloszkóp kártya

Ez a digitalizáló nem támogat minden lehetőséget, az ilyen funkciókat egy PXI keretrendszerbe foglalt PXI-5102-es kártyán teszteltem. Jelgenerátorként a számítógép hangkártyáját használtam.

1.3. Főbb oszcilloszkóp típusok

Tekintsük át a manapság használt és elterjedt főbb oszcilloszkóp típusokat.

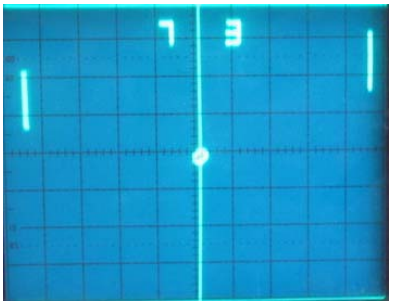
1.3.1. Analóg oszcilloszkóp

Az analóg oszcilloszkópot gyakorlatilag a katódsugárcsövek feltalálásával egy időben készítette el Karl Ferdinand Braun [1] német fizikus. Erre az alapokra építkezve készítette el 1930-ban egy brit vállalat az első kettős sugarú oszcilloszkópot, mely a második világháborúban a radar kifejlesztésében és működtetésében játszott jelentős szerepet. Mivel ezt az oszcilloszkópot nem kalibrálták, ezért csak rezgőkörök teljesítményének vizsgálatára voltak alkalmasak, tényleges mérésekre nem. Az első triggerelt oszcilloszkópot 1946-ban két amerikai tudós találta fel, mely az ismétlődő jelek vizsgálatában hozott előrelépést. A trigger egy bizonyos esemény, pl. egy megadott feszültséghatár felfelé történő átlépése, periodikus jeleknél ezért stabil lesz a kijelzés, ha mindig ugyanahhoz a feszültségértékhez tartozó ponton indul újra a vízszintes eltérítés)

Az analóg katódsugárcsöves oszcilloszkópokat még manapság is használják, bár ezeket egyre jobban leváltják a digitális tárolós oszcilloszkópok. Fő előnye a manapság használatos digitális oszcilloszkóppal szemben, hogy CRT kijelzője miatt a legapróbb jelugrások, és nagyfrekvenciás zavarok is könnyedén érzékelhetők vele. Mivel a felépítésük nem túl bonyolult, ezért manapság már viszonylag olcsón hozzá lehet jutni. Az analóg áramkörök miatt léteznek oszcilloszkópok, amelyek relatíve nagy feszültségtartományt képesek vizsgálni, ennek ellenére kis feszültségtartományokban is részletes felbontásúak.

A manapság használt típusok már mind kétsugaras változatok, melyekkel azonos időskálán két különböző jelet vethetünk össze, így például vizsgálhatjuk és összevethetjük bármilyen rendszer impulzusválaszát, vagy áramkörök bemeneti és kimeneti jeleit. Alapértelmezésként a sugár vízszintes eltérítése időben konstans növekvő (és a trigger esemény által indul újra), míg a függőleges eltérítést a mért feszültségváltozás befolyásolja.[6] A kétsugaras

oszcilloszkópoknál létezik még úgynevezett X-Y mód is, melynek használatakor a vízszintes eltérítést az egyik, a függőlegest a másik csatorna feszültségváltozása irányítja, így például szinuszjelek esetén a Lissajous-görbék alapján megállapítható a fáziskülönbség, vagy a frekvenciaszorzó. Érdekesség továbbá, hogy az első videojáték kijelzését az oszcilloszkóp X-Y módját használva valósították meg egy meghajtó áramkörrel (pong-játék), relékkel és potenciométerekkel működött. Újabb változata a 2. ábrán látható:



2. ábra: Pong játék

1.3.2. Digitális tárolós oszcilloszkóp

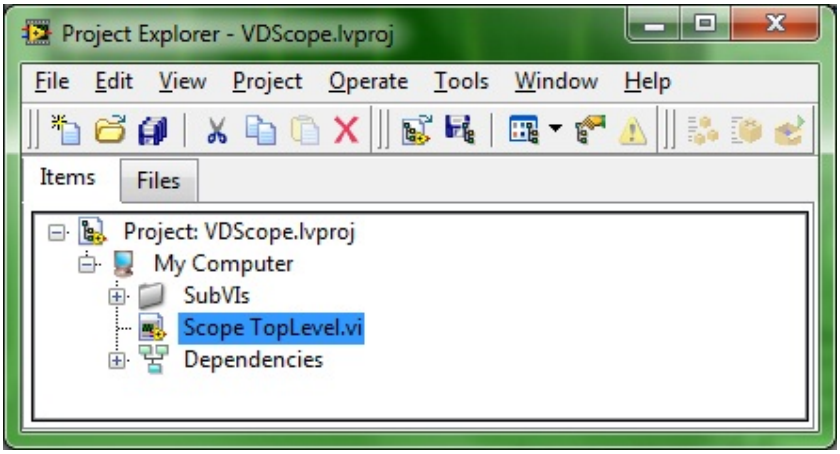
A digitális oszcilloszkópok az 1980-as évektől kezdve léteznek, Walter LeCroy alkotta meg az elsőt, méghozzá a svájci CERN számára magfizikai kutatások céljából. A digitális oszcilloszkópok minden csatorna jelét egy-egy ADC-vel (Analog-to-Digital Converter) digitalizálják, és a digitalizált jelet egy memóriában tárolják, majd megjelenítik. Az analóg oszcilloszkópokkal ellentétben a digitális oszcilloszkópok LCD kijelzőt használnak.

Mivel a digitális technikában nem lehet folyamatos jelet előállítani, ezért a digitális oszcilloszkópokra jellemző az úgynevezett mintavételezési frekvencia, vagy mintavételezési gyakoriság. Ez minél nagyobb, annál sűrűbben vesz mintát az oszcilloszkóp ADC-je az analóg vonalról, így annál nagyobb felbontású a jel. Mivel a mintavételezési frekvencia meghatározza a mérhető jel legnagyobb frekvenciáját, a nagyfrekvenciás zavarok detektálását viszonylag nehéz megoldani digitális oszcilloszkópokon. Az egyik erre született megoldás, hogy kézzel változtatni lehet a mintavételezési frekvenciát, vagy pedig egy bizonyos periódus jele kinagyítva megfigyelhető nagyobb mintavételezéssel.[3]

Az elektronikai eszközök hibakeresése során a digitális oszcilloszkópok vitathatatlan előnye, hogy az analógokkal szemben a trigger esemény előtti hullámforma is megtekinthető (pretrigger). Ezt úgy érik el, hogy az ADC folyamatosan digitalizálja a csatornáról származó adatokat, majd mikor a trigger esemény megtörténik, csak a memória felének (vagy felhasználó által beállított hányadának) méretéig történik a mintavételezés, így a trigger előtti hullámforma is a memóriában marad, és eltárolódik a trigger pozíciója is, amely meg is jelenik a kijelzőn a felhasználó számára. A digitális tárolós oszcilloszkópok előnye továbbá, hogy a vizsgált jelalak egy nem felejtő (jellemzően flash) memóriában eltárolható, így bármikor újra elővehető kiértékeléshez, ha nem ismételhető meg a mérés [4;5]. A modern digitális oszcilloszkópok a digitális technika fejlődésével már gigahertzes nagyságrendű tartományban is képesek mintát vételezni, megfelelő szabályok betartásával.

2. Az implementáció lehetőségei, a választott megvalósítás

Mivel a program, több programegységből fog állni, célszerű ezeket egyetlen projektbe foglalni. A LabVIEW 8.6 projekt kezelőjével (*Project Explorer, 3. ábra*) ez könnyedén megtehető, létre kell hozni egy projektet, elnevezni azt, majd minden létrehozott *VI* fájl ebbe a projektbe fog belekerülni. Ez abból a szempontból is előnyös, hogy elemzi a programegységek függőségeit, ezért másik számítógépre való átvitelkor tudjuk, milyen beépülő modulokra van még szükség.



3. ábra: Project Explorer

Innen könnyen megkereshetők a függőségek, de a program alegységei is egyszerűen megnyithatóak.

A program fő felépítésével kapcsolatban többféle változat merült fel, melyeknek előnyei és hátrányai is vannak egyaránt, ezeket szeretném bemutatni a következőkben.

2.1. Állapotgép-konstrukció

Az állapotgép egy olyan konstrukció, melynek többféle állapota van, és ezek között átmeneteket definiálunk. Mindenképpen létezik kezdő állapot, mely az elinduláskor fut le (inicializálás), és záró állapot, melyre a program befejeződése előtt kerül sor (lezárás). Programozástechnikai szempontból ez annyit jelent, hogy egy cikluson belül egy többszörös elágaztató utasítás található, mely elágazás feltétele az állapot, amelyet numerikus vagy szöveges változóban tárolunk. A ciklus kezdete előtt az állapotváltozót kezdőre állítjuk, így arra kerül a vezérlés, majd pedig ebből az állapotból a várakozó állapotba kerül a program. Ezen belül különböző események által más-más állapotba vihető át az állapotgép, melyek véget érésükkor ismét várakozó állapotba billentik az állapotgépet. A ciklusnak hátultesztelőnek kell lennie, mert amikor a végállapotra módosul az aktuális állapot, akkor még egyszer le kell futnia a ciklusmagnak, azon belül a lezáró állapotnak. A LabVIEW-ban az adatfolyam-elvű programozás révén ez könnyen megvalósítható.

Az állapotgépnek létezik továbbfejlesztett változata is, amelyben az állapotváltozó valójában egy végrehajtási sor, és egyszerre több „utasítás” is megadható, és azok az állapotok szekvenciálisan követik majd egymást.

A projekt szempontjából a program jelentős negatívuma, hogy az így létrehozott konstrukció magas állapotszám esetén meglehetősen átláthatatlan lesz, és a végrehajtás nem párhuzamosítható. Mivel a LabVIEW egyik legfontosabb előnye a párhuzamos programozás, ezért egy bonyolult programnál

ennek elvesztése nagy hatással lehet a teljesítményre. Párhuzamos végrehajtás megvalósítható, ha két párhuzamosan futtatandó „állapotot” egyetlen elágazásba írunk, de így ismételten csak a bonyolultsági fok nő.

2.2. Objektum-orientált megközelítés

A LabVIEW-nak is létezik beépített objektum-orientált támogatása, melyek során osztályokat, propertyket, eljárásokat hozhatunk létre, objektumokat példányosíthatunk. A projekt szempontjából a viszonylag könnyű programozhatóság az, ami előnyt jelentene, de mivel a szkópkártya vezérlő meghajtó beépülője (*NI-Scope*) referencia-alapon működik, és minden beállításhoz létezik lekérdező és/vagy beállító property, ezért erre még egy OO réteg „húzása” ismételten csak a performancia rovására menne. Ezen kívül a LabVIEW-ban minden osztály propertyjéhez beállító és lekérdező alprogramot (*subVI*) ír meg, valamint ezeknek hasonló kinézete is lesz. Így azon túl, hogy a forráskód méretét jelentősen növeli, a *call stack*-et (hívási sor) is sokkal jobban igénybe veszi, mely ismételten károsan befolyásolja a teljesítményt.

2.3. Eljárás- és eseményorientált koncepció

Az eljárásorientált programozás nagy programok esetében viszonylag átláthatatlan kódot eredményez. Ennek ellenére erre a módszerre esett a választásom. Mivel a teljesítmény az egyik legfőbb szempont, ezért előnyösebb ez a felépítés, és az *NI-Scope* driver sok propertyken keresztüli könnyű beállíthatósága miatt viszonylag könnyen követhető a program adatfolyama. Az adatfolyamon kívül a különböző felhasználói beavatkozásokra történő reagálás eseménykezelőkkel történik (*event structure*). A program három fő részből áll, amelyek az inicializálás, a fő vezérlés, és a lezárás.

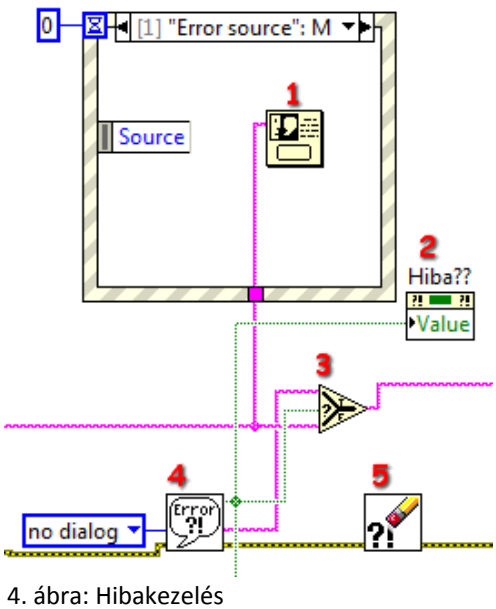
2.3.1. Az inicializáló rész

Az inicializáló rész feladata, hogy az eszközt alaphelyzetbe hozza, valamint megadja a beállítások kezdőértékeit.

Az inicializáló blokk két részből áll. Az első része egy ciklus, amelyben a program próbálja az eszközt inicializálni a kiválasztott eszközazonosító alapján, a ciklusmag mindaddig ismétlődik, amíg ezt nem sikerül végrehajtani. Hiba esetén egy előlapi LED kijelző jelzi, ha nem sikerült az inicializálás. Amint a driver inicializáló VI lefutott, a ciklus véget ér, és végrehajtódnak az alapbeállítások, majd rákerül a vezérlés a fő ciklusra.

2.3.2. A fő vezérlő rész

A fő vezérlő rész gyakorlatilag egyetlen while ciklus. Ezen belül vannak megvalósítva a beolvasási, hibakezelési, és eseményvezérlési egységek. A cikluson belül az egyik iterációról másikra történő adatátvitelt a shift regiszterek valósítják meg. Mivel a vezérlő rész ciklus szerkezetű, ezért a hibakezelés nem lenne felhasználóbarát, ha minden hiba észlelésénél egy felugró ablak figyelmeztetné a felhasználót. Ehelyett a felhasználó számára egy LED mutatja, ha éppen valamilyen hiba történt, és egy gomb megnyomásakor láthatóvá válik a hibaüzenet. Ezzel a módszerrel elkerülhető, hogy a program a hibaüzenetek miatt „beragadjon”, mivel az error adatvonal is átkerül egyik iterációról a másikra. Mivel a legutóbbi hiba szövegére is valahogyan „emlékeznie” kell a programnak, ezért az egy string típusú shift regiszterben tárolódik.



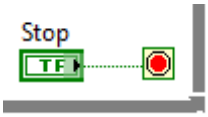
4. ábra: Hibakezelés

A 4. ábrán látható, hogy a string (rózsaszín vezeték) minden esetben a legutóbbi hiba szövegét fogja tartalmazni, a (3) útválasztó VI által, valamint hogy a hibavezeték (sárga) mindig nullázódik. Ha hiba történt, az bekapcsolja az ezt jelző LED-et egy *property node*-on keresztül (2). A (4) végzi a hiba lekezelését, az (5) pedig alaphelyzetbe állítja a hibavonalat (*No Errors*). A hiba kérésre megjelenítését a fenti esemény struktúrán belül az (1) végzi. Vegyük észre, hogy az

eseménystruktúra maximális időtúllépése 0 milliszekundum, ami azt jelenti, hogy ha nincs feldolgozandó esemény, a *timeout* keretet hajtja végre, és továbblép.

A fő ciklus futását a 

előlapi gombbal állíthatjuk le, mely a ciklusfeltételt igazra állítja (5. ábra).



5. ábra: Stop gomb a blokkdiagramon

A ciklus késleltetési ideje is beállítható, erre azért van szükség, mert ha nem lenne a benne késleltetés, akkor a program nagyon processzorintenzív lenne. Ezt a *Wait until next ms multiple* VI-al lehet beállítani, mely csak a következő óraütés többszörösére vár, tehát ha 250 a bemeneti értéke, de a ciklusmag az adott iterációban 100 ms alatt lefutott, akkor már csak 150 ms-ot fog várakozni.

2.3.3. A lezáró rész

A lezáró részben gyakorlatilag a kapcsolatot le kell zárni az eszközzel, és minden intézkedést meg kell tenni, hogy a kártya esetleges leválasztása biztonságos legyen. Ezt is egy NI-Scope beépített VI végzi, az *niScope Close*, melynek két bemenete van,

3. A program fejlesztésének folyamata

Miután elkészült a keretrendszer, ideje volt belefogni a „tényleges” fejlesztésbe. A fejezetben lépésről lépésre írom le a program fejlesztésének főbb szakaszait programegységek szerint, az azokhoz tartozó kutatási munkát, tanácsokat, esetleges zsákutcákat, vagy olyan részeket, amelyek a végső változatba nem kerültek bele. A fejlesztés folyamata rám nézve tanulságos volt, habár a fejlesztés előtt már elég jól ismertem a LabVIEW fejlesztőkörnyezetet, a munka során rengeteg újdonságot, hasznos tulajdonságot fedeztem fel benne.

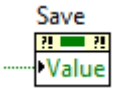
3.1. A fejlesztés menete, alkalmazott módszerek

A fejlesztés során habár megvolt a pontos követelmény, az interfész és a megvalósítás kialakításában szabad kezet kaptam. A program írásakor sok esetben eltértem a LabVIEW programozás preferált módszereitől [2], de ezt minden esetben a kód átláthatósága, és a későbbiekben való bővíthetősége miatt tettem.

3.1.1. Egy vezérlő értékének beállítása

A kód átláthatósága érdekében sok esetben alkalmaztam az úgynevezett *property node*-okat (7. ábra), egy-egy előlapi vezérlő értékének lekérdezésére, vagy módosítására. A property node-ok használata akkor javallott, amikor egy vezérlő értékének módosításakor meg szeretnénk hívni az ahhoz esetlegesen tartozó eseményvezérlőket. Amennyiben erre nincs szükség, a vezérlő értékére hivatkozó lokális változót hoztam létre, és azon keresztül állítottam be az értéket. Ennek teljesítménybeli vonatkozása nincs, ugyanis mindkét módszer hasonló sebességgel működik. Az ok ismét az átláthatóságban keresendő, a *local variable* (6. ábra) valamivel átláthatóbb kódot eredményez, mint a *property node*, a fő különbség ott van, hogy a lokális változóval ellentétben a property node a hibákat helyben lekezeli. Egy adott objektumhoz tartozó property node-okat, és lokális változókat egyszerűen megkereshetjük az objektumon jobb gombbal kattintva, és a *Find* menü almenüit használva (*Property nodes; Local variables*).

6. ábra: Local variable



7. ábra: Property node

3.1.2. Kódrészletek kiemelése, újrafelhasználhatóság

Ahogy növekszik egy kód, úgy lesz egyre átláthatatlanabb. A kód átláthatóságát a szöveges eljárásorientált programozási nyelvekben eljárások és függvények létrehozásával lehet növelni. Ehhez hasonló lehetőség a LabVIEW-ban is létezik, ha kijelölünk egy kódrészletet, és az *Edit* menü *Create SubVI* parancsot választjuk, akkor a kijelölt kódrészletet kiemeli egy külön VI-ba, és automatikusan hozzárendeli a be- és kimeneteket. A dolog szépsége, hogy amennyiben property node-ot akarunk kiemelni subVI-ba, abban az esetben az adott vezérlőhöz referenciát hoz létre, és a property node dinamikus lesz, miáltal a bemenetként megkapott referenciájú objektum tulajdonságait fogja lekérdezni / beállítani.

3.1.3. Eseménykezelés

A kód írása során az események kezelésének megoldása is felmerült bennem. A LabVIEW programozás során a preferált megoldás az [2], hogy programblokkonként egyetlen eseményvezérlőt (*event structure*) célszerű alkalmazni, ezen belül minden eseményhez, vagy több összetartozó eseményhez keretet (*event case*) létrehozni. Ezzel szemben én az összetartozó események kezelését végző programrészleteket külön-külön eseményvezérlőbe tettem, mert így átláthatóbb lesz a kód azáltal, hogy nem kell minden eseményhez tartozó kimeneti vonalhoz alapértelmezett értéket rendelni, hanem minden keretben „értékes” adatot rendelhetünk hozzájuk. A LabVIEW képes több eseménystruktúra kezelésére egyszerre, tehát nem veszik el egyetlen várakozó (*pending*) esemény sem, ha egy struktúra nem kezeli le azokat.

Ezzel szemben előlapi gombok megnyomását sok esetben nem eseményvezérlővel kezeltem le, hanem feltételes elágazással (*case structure*).

Ehhez viszont az szükséges, hogy a gomb állapota „lenyomva” maradjon, amíg rá nem kerül a vezérlés annak kezelésére. A logikai változót beállító vezérlőknél a LabVIEW-ban meg lehet adni az érték megjelenését felhasználói interakciótól függően:

- Switch when pressed: a kontroll értéke lenyomáskor változik meg, és a következő lenyomásig abban az állapotban marad.
- Switch when released: az érték a vezérlő felengedésekor változik meg, és úgy is marad a következő „felengedésig”
- Switch until released: az érték a vezérlő lenyomva tartásáig változik meg.
- Latch when pressed: az érték lenyomáskor egy pillanatra megváltozik, majd ismét vissza.
- Latch when released: az érték felengedéskor változik meg egyetlen időpillanatra
- Latch until released: hasonló mint a „Switch until released”, de az érték olvasása itt aszinkron módon történik.

Általánosságban elmondható, hogy a latch működés eseménykezeléshez alkalmasabb, a switch pedig nyomógombként való használatkor feltételes utasítással való kiolvasáshoz, vagy kapcsolóként való használathoz. Ha az értéket egy *Case* struktúrával szeretnénk kiolvasni, és csak a lenyomásra vagyunk kíváncsiak, a struktúra true értékhez tartozó keretében kell megírni, és a gombon „Switch when pressed” vagy „Switch when released” mechanikus akciót választva annak értékét a true frame-ben egyúttal false-re kell állítani, így nem marad „lenyomva” a gomb.

3.1.4. Reagálás az érvénytelen bevitelre

Újabb kérdés a vezérlők értékeinek beolvasása, ha hibás adatot ír be a felhasználó. Amennyiben numerikus vezérlőbe szöveges értéket írunk, a LabVIEW ezt automatikusan lekezeli, és a vezérlőt az alapértékére állítja. Ezt az alapértéket jobb gombbal a vezérlőn kattintva állíthatjuk be. Először állítsuk be

alapértelmezett értékre, majd jobb gomb, *Data Operations → Make current value default*. A vezérlő a VI megnyitásakor már az alapértelmezett értéket fogja mutatni.

Amennyiben egy vezérlőnek egy megadott értékhatáron belül kell értékeket elfogadnia, abban az esetben két lehetőségünk van. Az első, hogy ezt programmatikusan lekezeljük. Ebben az esetben property node-okkal kell még az érték kiolvasása előtt megfelelőre változtatnunk a vezérlő értékét, így ez a megoldás elég körülményes. Szerencsére a LabVIEW erre is kínál megoldást, a vezérlő tulajdonságai között a *Data Entry* lapfülön megadható az adott vezérlő megengedett minimális, illetve maximális értéke, és szükséges még beállítani, hogy a tartományon kívül eső bevitelt korrigálja (*Response to value outside limits* legördülőmezőben *Coerce* (kényszerítés) választása). Ezen kívül ezeket az értékeket programból is beállíthatjuk property node-okon keresztül, így dinamikusan állítható egy vezérlő tartománya.

Az értékek bevitelekor sokszor szükség van mértékegységek, és SI jelölésrendszer használatára, ennek beállítására az objektum tulajdonságainál a *Display Format* fülön van lehetőség. Itt automatikusra is állíthatjuk a kijelzést, de formátumsztring is megadható. Ennek formájára a későbbiekben részletesen is kitérünk, a kurzorok fejezetben.

3.2. A kijelzés

Minden oszcilloszkópnak szüksége van kijelzőre, amelyen a jeleket vizsgálhatjuk. A LabVIEW-ben hullámformák kijelzésére több lehetőségünk nyílik, vannak erre dedikált kijelzők, de saját magunk is hozhatunk létre, egy általános képmegjelenítő objektumot használva. Ez a lehetőség viszonylag bonyolult, és lassú is, mert főleg statikus grafikonok kijelzésére van kitalálva. Ezért mindenképp a LabVIEW kijelzői közül kellett választanom.

3.2.1. Waveform chart és waveform graph

Mivel a két vezérlő nagyon hasonló, ezért ezeket együtt értékelem. Mindkét vezérlő alapértelmezett vízszintes skálája az időskála, de a ráköthető bemeneti adatok tekintetében különbözőek. A *waveform chart*-ra ráköthetünk skaláris adatot, melynek hatására egyetlen pont értéke jelenik meg, egy- vagy kétdimenziós tömböt (ebben az esetben a tömb minden „sora" egy-egy jelalakként fog megjelenni), valamint az úgynevezett WDT adattípust (*Waveform Data Type*). Ez gyakorlatilag egy struktúra (*cluster*), melynek három eleme van:

- t_0: a kezdeti időpillanatot adja meg
- dt: két érték közötti időeltérés, Δt (másodpercben)
- Y: egydimenziós tömb, a jelalakot tárolja, két érték között időben pontosan Δt különbség van

A *waveform graph* bemeneti lehetőségei között nem szerepel a skaláris adat, minden másban megegyezik a waveform chart-tal, így elmondható, hogy a chart statikus adatok kijelzésére, míg a graph főleg dinamikus adatok kijelzésére való. Mivel az NIScope driver képes WDT adattípusba adatot beolvasni, ezért a waveform graph egy ideális választás lenne. Ne döntsünk túl gyorsan, nézzünk meg inkább még egy lehetőséget!

3.2.2. XY Graph

Az XY graph a nevéből adódóan X-Y értékpárok kijelzésére alkalmas leginkább. Míg a waveform graph-nál, vagy a waveform chart-nál nincs lehetőség egy időpillanathoz több értéket rendelni, az XY Graph vízszintes tengelye nem időtengely, így ez megoldható. A kijelző bemenetére kétféle adattípus köthető, az egyik egyetlen „ábra" (*plot*) kijelzésére használható, ebben az esetben az összetartozó X-Y értékpárokat egy struktúrában tároljuk, melynek két eleme van, mindkettő valós adatokat tartalmazó tömb. A tömbök elemszámának meg kell egyeznie. Az összetartozó értékpárok az azonos indexű tömbelemek. A

kijelzőn több plot megjelenítése ilyen struktúrák egydimenziós tömbbe rendezésével érhető el.

Végül ezt a kijelzési formát választottam, mivel a fő kielzőn nem feltétlenül csak jelalakokat fogunk megjeleníteni, hanem a két csatorna értékeit külön tengelyen ábrázolva azok X-Y grafikonját.

Ez azzal a hátránnyal jár, hogy amennyiben jelalakokat szeretnék megjeleníteni a kijelzőn, úgy a kapott WDT adattípust át kell konvertálni, a kijelző által érzékelhető formátumba. Ezt egy külön subVI-ba emeltem ki, mivel a konverziót mindkét csatorna jeleire el kell végezni. A konverzió úgy történik, hogy a beolvasott WDT adattípus Y értékeihez egy ciklus X értékeket rendel hozzá, méghozzá Δt időközönkénti lépéssel. Kezdőértéknek nem nullát választottam, hanem a program dinamikusan számolja ki a jelalak hosszából a kezdőértéket az (1) képlet alapján, hogy a 0 pont mindig pontosan a jel közepére essen.

$$\frac{n \cdot dt}{-2} \qquad (1)$$

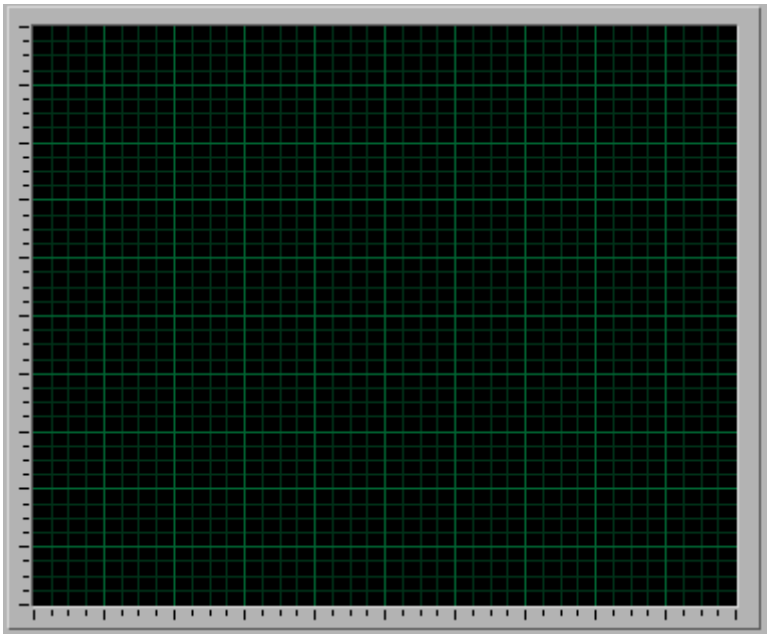
Az alprogramból kapott kimeneti struktúrát a kijelzőre kötve láthatjuk az adott csatorna jelalakját. A látható eredmény azonban meglehetősen elszomorító, ugyanis az XY Graph minden esetben a jelalakhoz igazítja a skálát, és nagyon sok hasznos képernyőterületet elfoglal. Ezen kívül a jel vizsgálatát is megnehezíti változó jelnél, a kijelző állandó újraskálázásával, ráadásul így a jelalak skálázására és eltolására sincs lehetőség.

Ennek kiküszöbölésére szerettem volna a kijelzőnek olyan megjelenést kölcsönözni, mint amilyen bármelyik oszcilloszkópon látható, hogy a skála osztása adott, és nem változik. Ahhoz, hogy ezt megtehessem, az XY graph tulajdonságai között mindkét tengelyre az *AutoScale* tulajdonságot ki kellett kapcsolnom, majd pedig az értékek és tengelyfeliratok kijelzését letiltani. Mivel ez nem elég, meg kellett adnom a skála minimális és maximális értékét is,

melynek a könnyű programozhatóság érdekében -5 alsó, és +5 felső határt adtam. Ezen kívül még statikusra kell állítani a skála vezetővonalainak gyakoriságát, hogy ez is mindig automatikus legyen, és jelalakok kiértékelésénél egységes skálával számolhassunk. Ezeket a beállításokat minden egyes ciklus iterációnál végre kell hajtani, mivel a LabVIEW képes őket „elfelejteni", így ez is belekerült egy külön subVI-ba, a beállításokat property node-okon keresztül lehet elvégezni. A megfelelő property node: XScale.Range és YScale.Range. Mindkettő ugyanaz a struktúra, melynek elemei:

- Increment: a fő vezetővonalak gyakorisága
- Maximum: a skála maximuma
- Minimum: a skála mimimuma
- Minor increment: a másodlagos vezetővonalak gyakorisága
- Start: a skála rajzolásának kiindulópontja

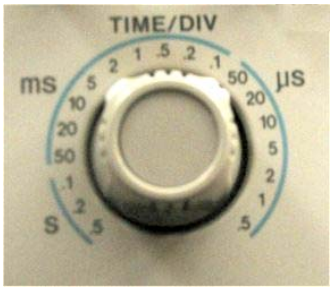
A vezetővonalak színének beállítása után (az alapértelmezett sárga nagyon zavaró), az így létrehozott kijelző meglehetősen hasonlít az oszcilloszkópoknál megszokottra (8. ábra)



8. ábra: Az oszcilloszkóp fő megjelenítője

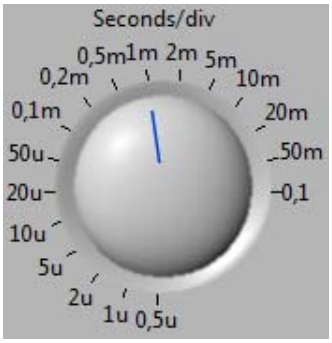
3.3. Az időskála beállításai

Miután elkészült a kijelző, meg kellett valósítani a vízszintes skála (időskála) skálázását. Ezt az oszcilloszkópoknál tipikusan egy időosztás-állító tekerőgomb teszi lehetővé, amint az a 9. ábrán látható.



9. ábra: időosztás-állító tekerőgomb egy analóg oszcilloszkópon

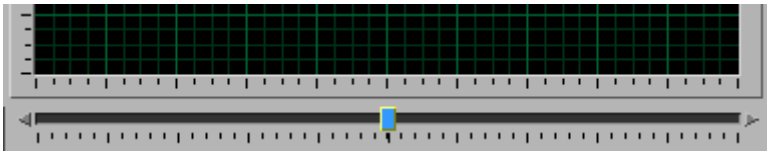
Ilyen tekerőgombok a LabVIEW-ben is léteznek, azonban van egy baj az alapbeállításukkal, hogy lineáris a skálájuk, és nem diszkrét értékekre állíthatóak be. Kezdetben próbáltam a vezérlő tulajdonságai között logaritmikus skálázásúra állítani, de ettől függetlenül nem sikerült elérni, hogy logaritmikus időközönkénti diszkrét értékeket vehessen fel. Szerencsére találtam egy lapfület, amellyel szöveges címkékkel megjelölt diszkrét értékeket lehet hozzárendelni a skálához. Ezt a *Dial* vezérlő tulajdonságlapjának *Text Labels* nevű lapfülén találtam meg, így felvettem a fenti képen is látható időosztás-értékeket. Ezek az értékek abból a szempontból előnyösek, hogy könnyű velük számolni.
A tekerőgomb állása azt adja meg, hogy a kijelző két fő vezetővonala között mekkora az időkülönbség. A létrehozott vezérlő a 10. ábrán látható.



10. ábra: Feszültségosztás beállítása LabVIEW-ban (Dial)

Ez a vezérlő fogja állítani a jel időosztását, de valahogyan időben a jelet el kell tudni tolni, hogy a kijelző méreténél nagyobb időtartamra is tudjuk vizsgálni azt.

A vízszintes eltolás megvalósítását legkönnyebben egy gördítősáv-szerű vezérlővel lehet megoldani. A görgetés megvalósítása a kijelző alatt elhelyezett *Pointer slide* segítségével lehetséges, melyet úgy valósítottam meg, hogy alaphelyzetben a kijelző közepe alatt álljon, és elmozdításkor a jel pontosan annyit mozdul, amennyit a mutatón mozdítunk. Így gyakorlatilag két „kijelzőnyi" jel megfigelésére van lehetőség egy adott időosztással. A vezérlőt a kijelző alá illesztve egészen természetesnek hat annak kezelése (11. ábra).



11. ábra: Vízszintes eltolás vezérlője az előlapon

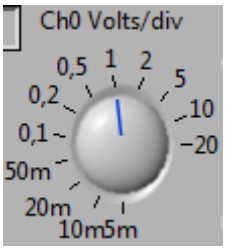
A vezérlő skálázása teljes mértékben megegyezik a kijelzőével, így nagyon egyszerűen programozható.

A programban a jel vízszintes skálázása és eltolása közvetlenül a jel WDT formátumból X-Y tömbbé konvertálása után valósul meg. Az X értékeket megszorozzuk az időosztás-állító aktuális értékének reciprokával, majd pedig hozzáadjuk a csúszka értékét. Mivel a kijelző két fő vezetőrácsa közötti különbség 1 (az XY graph szempontjából), ezért más átalakítási teendőnk az időskála mentén nincs. Ez meggyorsítja a számítást is, nem kell fölösleges konstansokkal szorozni.

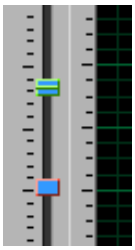
3.4. A feszültségskála beállításai

Az időskála beállításai már készen vannak, a következő lépés hasonló beállításokat eszközölni a függőleges skálán, a feszültségskálán is. Míg egy adott mérés során egyetlen időskála létezik, feszültségskálának annyinak kell léteznie, ahány csatorna jelét vizsgálni kívánjuk.

A feszültségosztás beállításának megoldása szinte ugyanolyan vezérlővel történik, mint az időskála esetében, ennek létrehozásakor már nem ütköztem nagyobb nehézségekbe, mivel a legjobb beállítási gyakorlatot az időosztás-állítás során elsajátítottam. A feszültségskála beállítása a 12. ábrán látható vezérlővel történik. Ebben az esetben is azt adhatjuk meg a tekerőgombbal, hogy a kijelzőn két szomszédos osztás távolsága hány Voltos feszültségnek feleljen meg.



12. ábra: Feszültségskála beállítása Dial vezérlővel



13. ábra: Függőleges eltolás vezérlője

Hasonló módszerrel, mint ahogyan azt az időskála mentén való eltolásnál alkalmaztuk, ugyanazt a vezérlőt munkára fogva (jobban mondva annak 90°-kal elforgatott változatát) megvalósítottam a függőleges eltolás lehetőségét. Ez többek között akkor jön jól, ha nem szeretnénk, hogy a két csatorna jele egymást fedje. A vezérlő ismételten hozzá lett illesztve a kijelzőhöz, ám akadt egy kis probléma. Nevezetesen az, hogy egynél több csatorna jelét kell tudnunk kezelni, de nekünk sajnos csak egy vezérlőnk van. Megpróbáltam a vezérlőt lemásolni, és a kettőt egymás mellé illeszteni, de amellett hogy túl sok helyet foglal el a kijelzőn, nem a leginkább felhasználóbarát megoldás. Jelen esetben kiemelten fontos, hogy egy ilyen alapvető beállítási lehetőség felhasználóbarátan legyen megvalósítva. Ennek a problémának a kiküszöbölése meghökkentően könnyű volt, de míg nem jöttem rá, elég sok időt eltöltöttem a különböző beállítások kipróbálásával. A megoldás még egy csúszka hozzáadása a vezérlőhöz, ami annyiban változtatja meg a vezérlő programozását, hogy a két csúszka értékét nem két külön skalárként kell kiolvasnunk, hanem egyetlen struktúra két elemeként. A vezérlő kinézete láttán mindenkiben egyből világossá válik, hogy hogyan is kéne ezt kezelni (13. ábra).

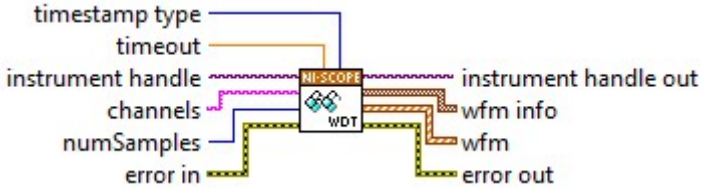
3.5. A beolvasás életre keltése

Eddig csak a megjelenítéssel foglalkoztunk, itt az ideje elkezdeni a jel beolvasását. Minden eddigi jelalak kijelzés és transzformálás beállítását egy express VI által generált jellel ellenőriztem. Ez több szempontból sem volt jó tesztkörnyezet, de biztos akartam lenni abban, hogy egy hiba esetén annak forrása nem a beolvasásban leledzik.

A LabVIEW-ban az niScope eszközpalettán több VI is van, amelyik a beolvasást hivatott megvalósítani. Nem kellett sokáig keresnem, mivel szerencsére a LabVIEW minden kiegészítőjéhez bőségesen rendelkezésre áll számos példaprogram. A példaprogramok keresése az alkalmazás bármely tervezőablakából a *Help* menü *Find examples…* parancsával érhető el. A példakereső azért is jó, mert online az ni.com-on is képes keresni, így mások által elkészített példaprogramok között is böngészhetünk.

3.5.1. A beolvasás programegységei

Egy idevágó példa kapcsán rátaláltam az eszközpalettán az *niScope Read (poly)* VI-ra. A VI nevében a „poly” szócska azt jelenti, hogy ez egy polimorf VI, és a működése attól függően lesz más és más, hogy melyik példányt választjuk ki (*polymorphic instance*). Léteznek olyan alprogramok is, amelyeknél a bemenetre kötött adattípus határozza meg a meghívott példányt. Ennek a VI-nak az esetében négyféle választási lehetőségünk van, ebből kettő kizárva, mivel azok egyszerre csak egyetlen jelalakot tudnak beolvasni. A maradék két példány közül az *1D Waveform Data Type* nevűt választottam, melynek kimenete egy WDT adattípusokból álló tömb, amelyben minden tömbelembe egy-egy csatorna hullámformája kerül. A VI be- és kimenetei a 14. ábrán láthatóak.



14. ábra: A beolvasást végző alprogram paraméterezése

A VI az *instrument handle*-ben kapja meg az eszközazonosító referenciát, ezt az *niScope Initialize* VI hozza létre az eszköz neve alapján az inicializációs részben. Minden NIScope VI-nak létezik ilyen be- és kimenete, mely arra szolgál, hogy ezek mentén összekötve a különböző programegységeket azok biztosan sorban legyenek végrehajtva. A *timestamp type* bemenet határozza meg, hogy a WDT struktúra kezdőpillanatát abszolút vagy relatív időben adja meg. Az abszolút idő mindig ugyanattól a ponttól kezdődik, a relatív idő pedig az éppen aktuális rendszeridő a beolvasás megkezdésekor. Mivel nekünk a kezdőpontra nincs szükségünk, így nem kell semmilyen bemenetet rákötni a vezetékre, hogy az alapértelmezett értéket vegye figyelembe. A *timeout* vonalon azt adhatjuk meg, hogy beállított trigger esetén hány másodpercet várjon a VI a trigger eseményre, miután rá kerül a vezérlés. Mivel nekünk még nincs konfigurálva trigger a szkópkártyához, egyelőre ez a beállítás is maradhat alapértelmezett értéken.

A channels bemenő string azt adja meg, hogy éppen melyik csatorna adatait szeretnénk mintavételezni. Ennek megadása formátumsztringgel lehetséges, melynek leírása a súgóban megtalálható. Innen kiderítve ennek kezdetben konstans „0,1” bemenő értéket adtam, hogy mindkét csatorna jelét egyszerre lehessen vizsgálni. Mivel a csatornák jelalakjának „kikapcsolhatónak” kell lenniük, kerestem egy olyan beállítást, amellyel „üres” csatorna adható meg, tehát azt akartam elérni, hogy a kimeneti tömb minden esetben kételemű legyen. Mivel ilyen beállítást nem találtam, ezért a csatornák kapcsolását úgy oldottam meg, hogy két egymásba ágyazott case struktúra a kiválasztott csatornáknak megfelelő értéket továbbít erre a bemenetre. Amennyiben csak az egyik csatorna van kiválasztva, a másik helyére üres WDT konstanst szúrok, így a megjelenítő színeit nem kell dinamikusan állítani. Képzeljük csak el, milyen összezavaró lenne, hogyha az első csatorna kikapcsolásakor a második csatorna jelének színe az elsőével egyezne meg.

A VI következő bemenő paramétere, a *numSamples* az adott mintavételezés hosszát adja meg, méghozzá a pontok számában. Ezt a beállítást is alapértelmezetten hagyva egyelőre megfelelő lesz.

Általában minden VI-on megtalálhatóak az *Error in* és *Error out* adatvezetékek. Az Error out arra szolgál, hogy a végrehajtás során keletkező hibákat programmatikusan kezeljük, az Error in szerepe pedig abban van, hogy ezek mentén sorbakötve az alprogramokat, amennyiben az egyikben hiba történik, az meggátolja a többi VI futását, így megelőzve a hibák „felhalmozódását", és elősegítve a vezérlés mielőbbi rákerülését a hibakezelő részre.

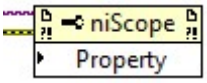
A kimenetek közül a *wfm* kimeneten jelenik meg az egy- vagy több beolvasott jel, mint ahogyan azt feljebb tárgyaltuk. A *wfm info* különböző, a jelre vonatkozó információkat ad át, mint például a mintavételezési gyakoriság, a beolvasás kezdetének időkülönbsége a trigger detektálásához képest, és még pár paraméter. Ezek közül nekünk egyikre sem lesz szükségünk, ugyanis a WDT struktúrából megtudható minden lényeges információ.

3.5.2. További beállítások

Amint a VI be- és kimenetei hozzá vannak rendelve a megfelelő adatvonalakhoz, ideje elkezdeni egy tesztelést. Rákötöttem egy jelforrást az oszcilloszkóp kártyára, és szépen működött minden, mindaddig, amíg egy hiba nem történt. Amint megtaláltam a hiba forrását, és kijavítottam a felületen, a kártya a jelet továbbra sem volt hajlandó olvasni, csak ha újraindítottam a programot. Ez elég bosszantó tud lenni, így utánaolvastam a témának, és azt találtam, hogy amennyiben egy hiba történik a beolvasás során, úgy a jelalak beolvasását az eszköz nem szünteti meg, és addig nem kezdhetünk új beolvasást, amíg az előző folyamatban van. Ennek kiküszöbölésére a program hibakezelő részébe beépítettem az *niScope Abort* VI-t, amelyre csak akkor kerül a vezérlés,

ha a legutóbbi hibát tároló shift regiszter értéke hibát jelez. Ekkor megszakítja az aktuális beolvasást, és a következő Read VI már le tud futni.

A jelek beolvasásához még számos alapbeállítást kell végezni, melyek nagy része az inicializációs szekcióban foglal helyet, néhányat pedig dinamikusan frissítünk. Ezeket a beállításokat property node-okon keresztül is elvégezhetjük, ugyanis egy üres property node-ot az eszközpalettáról a blokkdiagramra helyezve nem statikus lesz, így az *instrument handle* adatvonalat a referencia bemenetére kötve egyből felveszi annak az osztálynak a tulajdonságát, és a referenciában megkapott eszközt fogja vezérelni (15. ábra). Itt a szokásos módon választhatjuk ki a beállítandó tulajdonságokat, és ha egynél több property beállítására és/vagy lekérdezésére van szükség egyszerre, könnyedén átméretezhetjük, így újabb elemmel bővül a lista. Mivel a program inicializációs részében megtörténik az automatikus eszközbeállítás, ezért néhány dolgot a fő részen belül kell beállítanunk, így elősegítve, hogy a megjelenített jel paraméterei determinisztikusak legyenek. Minden ilyen beállítás elvégezhető property node-okon keresztül, így nem kell a programnak VI hívásokkal foglalkoznia, hanem egy lépésben beállítja az ahhoz tartozó driver paramétert.

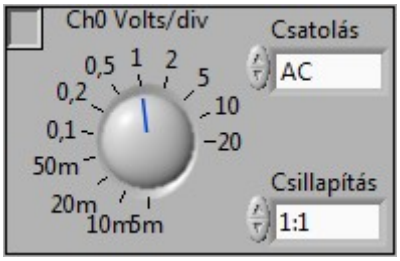


15. ábra: niScope Property Node, referencia alapú eszközhivatkozással

Első ilyen beállítás a csatornák csatolása, melyet mindkét csatornára külön-külön el kell végezni. A csatorna csatolásának három különböző értéke lehetséges, *AC*, *DC*, és *Ground*. AC csatolásban az oszcilloszkóp figyelmen kívül hagyja a jel esetleges feszültségoffszetjét, tehát ha egy szinuszjel 1V és 2V közötti értékeket vesz fel, akkor annak virtuális „0” értéke éppen 1,5V-nél van. Úgy is mondhatjuk, hogy a jel 1,5V-tal el van tolva. DC csatolásnál az oszcilloszkóp figyelembe veszi a jel offszetjét, tehát ha van benne egy 1,5V-os eltolás, abban az esetben a jel ténylegesen eltolva jelenik meg a kijelzőn, míg AC csatolásban a virtuális 0 pont a valódi 0 pont helyén jelenik meg. A *ground* csatolásban a jel

értéke a földpotenciállal lesz egyenlő, ezt legtöbb esetben kalibrációra, vagy a jel pontos pozícionálására használják.

A vertikális skálán még van lehetőség az úgynevezett csillapítás, vagy erősítés használatára. Ennek értéke 1:100 arány és 100:1 arány között nagyságrendenként mozog, melynek túl alacsony, vagy túl magas jelnek a bemenetre kapcsolása esetén van értelme. Ez is felhasználó által kiválaszthatóvá lett alakítva, így a függőleges skála vezérlői elérték végső formájukat (16. ábra).



16. ábra: A függőleges skála vezérlői

A függőleges beállítások közül még egy hátra van, mégpedig a feszültségtartomány (*voltage range*) beállítása. A függőleges feszültségtartománynak azért van értelme, hogy minél nagyobb felbontással szemlélhessük a vizsgálni kívánt jeleket. Ha például 10 Voltra állítjuk a maximális feszültségtartományt, és a kártyánk ADC-jeinek felbontása 8 bit, akkor a 10 Voltnak megfelelő feszültségérték a decimális 255-tel lesz egyenlő. Így például az 5 V-nak megfelelő feszültségérték decimális 127 lesz. Azonban ha a kártyára az aktuális maximális feszültségtartományánál nagyobb feszültséget kapcsolunk, a jel tetejét és alját „levágja”. Ez az ADC működésének alapelve miatt van így. Egy ADC-nek szüksége van meghajtó feszültségre, a konvertálandó feszültségre, és egy referenciafeszültségre, amelyet a konvertálandóval összehasonlítva adja meg annak értékét. Így a maximális feszültségtartomány gyakorlatilag az ADC referenciafeszültségét szabályozza. Egyelőre ezt a beállítást egyelőre a felhasználóra bízzuk, a finomhangoláskor még visszatérünk rá.

Még egy beállítás maradt hátra, a mintavételezési frekvencia (*sampling rate*). Ez az a gyakoriság, amellyel az ADC digitalizálja az aktuális bemenő analóg feszültségszintet. Ennek értéke minél magasabb, annál részletesebb lesz időben a jel. Ha értékét a vizsgált jel frekvenciájának legalább kétszeresére nem állítjuk, akkor kisebbnek fogjuk detektálni a frekvenciát, mint amennyi az valójában. Ahhoz, hogy az oszcilloszkóp megjelenítőjén a lehető legpontosabban legyen ábrázolva a jel, optimálisan az aktuális időosztás kijelzőn pixelben mért értékére kell állítanunk ((2) képlet).

$$\frac{1}{t_d}p \qquad (2)$$

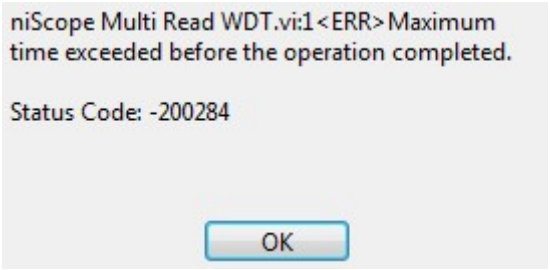
ahol t_d az aktuális időosztást jelöli, p pedig a pixelek számát időosztásonként. Mivel a pixelek számát időosztásonként csak bonyolult módszerekkel lehetne megtudni, és ez értékes időt venne el a feldolgozástól, ezért p értékét konstanssal állítottam be, úgy, hogy minden nagyításon megfelelően pontos jelet mutasson.

3.6. Triggerelés

Miért is van szükség triggerelésre? Ha nem létezne trigger, akkor csak abban az esetben látnánk stabil jelet a kijelzőn, ha a mért jel frekvenciája egész számú többszöröse lenne a beolvasási frekvenciának. Ezért ha stabil jelet szeretnénk látni, minden esetben a jel frekvenciájának megfelelően kellene beállítani a beolvasási gyakoriságot. Ennek a megoldására született meg a triggerelés. Az analóg oszcilloszkópoknál a trigger esemény bekövetkezte indította el a jelnek a megjelenítését (vagyis az állította alapállapotba a vízszintes eltérítő lemezt, a *horizontal sweep*-et). A digitális oszcilloszkóponkál ezzel szemben a beolvasás folyamatosan történik, majd az oszcilloszkóp megjegyzi, hogy a memóriában hol található a trigger esemény. Ezt a kijelzőn is megjelenítik, és általában felhasználó által pozícionálható, hogy hol legyen ennek helye a memóriában (ezt nevezzük helytelenül előtriggerelésnek).

Az NI oszcilloszkóp kártyák nem végeznek folyamatos beolvasást, hanem a read VI, vagy pedig a program explicit módon hívja meg a beolvasási folyamat

elindítását. Az NIScope driver úgy végzi a beolvasást, hogy annak elindításakor a trigger memóriában elfoglalt pozíciójáig (*trigger reference position*) olvassa az adatokat, és azt a memóriaterületet, mint egy FIFO sort használ, amíg a felhasználó által beállított trigger esemény be nem következik. Amint ez megtörténik, a memória hátralévő területének megfelelő mennyiségű adatot még mintavételez, és átadja a programnak a beolvasott adatot. A trigger pozícióját szükségtelen átadni, mivel azt a felhasználó állította be, így ezt a beállítást olvasva a kijelzőn megjeleníthető a triggerpozíció. A gyakorlatban az eszköznek nem minden esetben kell beolvasnia a teljes memóriaterületének és az aktuális mintavételezési frekvenciának megfelelő jelhosszt, ilyenkor a *Record Length* (olvasás hossza) paraméter által beállított értékre fog vonatkozni a trigger relatív pozíciója. Felmerülhet bennünk a kérdés, hogy mi történik akkor, ha sosem következik be a beállított trigger esemény? Ezt az niScope read VI úgy oldja meg, hogy bemenő paraméterként kér egy timeout változót, amellyel másodpercben megadhatjuk számára, hogy az előtriggerelési puffer megtelte után hány másodpercig várakozzon trigger esemény bekövetkeztére. Amennyiben ez nem történik meg, a 17. ábrán látható keletkezik.



17. ábra: Trigger detektálásának időtúllépési hibaüzenete

Az NI oszcilloszkóp kártyák többféle triggerelési típust támogatnak, melyek a következők:

- Immediate: ez a beállítás azt adja meg, hogy ne történjen triggerelés, hanem a beolvasást a lehető legnagyobb gyakorisággal végezze a fetchelő algoritmus. Ez a gyakorlatban azt jelenti, hogy nincs trigger esemény. Ha a drivert erre a triggerelési beállításra állítjuk, nincs semmilyen bemenő paraméter.

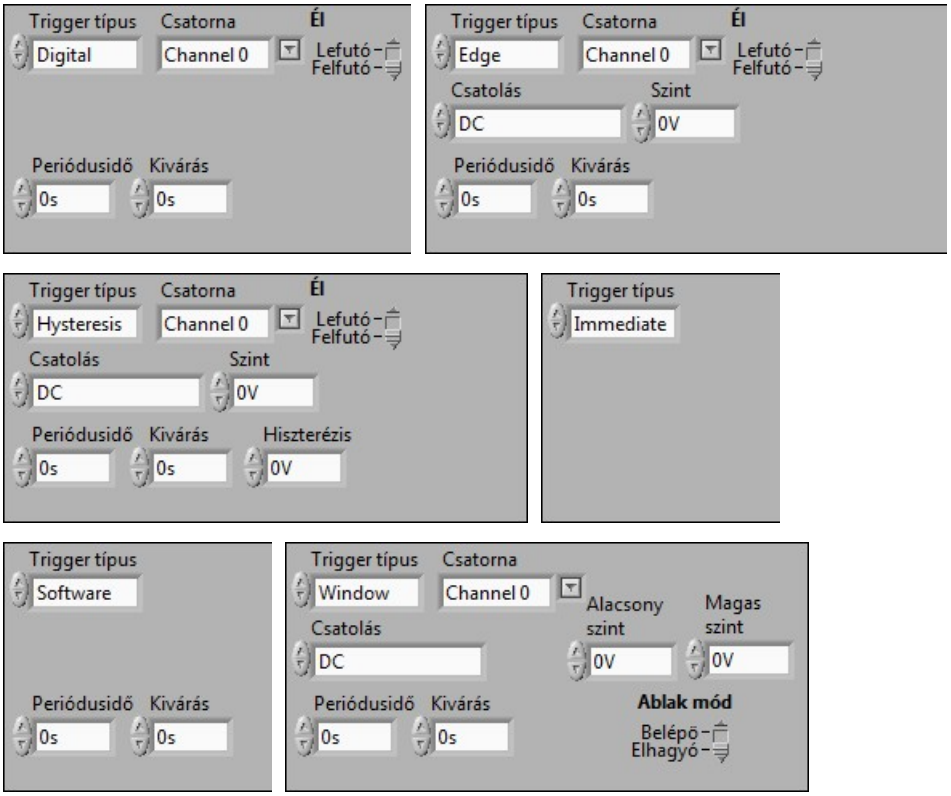
- Edge: ez magyarul „éltriggerelést” jelent, hosszabb nevén *Analog edge ref trigger*. Ez az esemény akkor következik be, ha a jel feszültségszintje valamilyen irányban átlép egy megadott feszültségszintet. Bemenő paraméterként meg kell adnunk, hogy melyik csatornán figyelje a trigger eseményt, valamint azt, hogy az adott csatornán milyen csatolással figyelje az eseményt. A csatolási lehetőségeket az előző fejezetben tárgyaltuk. Ezen kívül bemenő paramétere még, hogy milyen feszültségszintet figyeljen, és hogy ezt a jelalak felfutó, vagy lefutó élén tegye. Van még két paramétere, a *holdoff* és a *delay*, mindkettő azt szolgálja, hogy ha több felharmonikus található egy jelen, akkor mindig ugyanarra triggereljen. A holdoff értéke azt adja meg, hogy a trigger detektálása után mennyi ideig ne figyelje a trigger eseményt, a delay pedig azt, hogy a trigger esemény után mennyi idővel kezdje el beolvasni a jelet. A két paramétert például egy állóhullám esetén úgy célszerű beállítani, hogy a holdoff értéke az állóhullám alapfrekvenciájának reciproka legyen, így a vivőfrekvencia jelváltozásaira a trigger nem fog aktiválódni, vagyis egy állóhullám periódusán belül mindig ugyanarra az élre aktiválódjon.
- Hysteresis: hosszabb nevén *Analog Hysteresis Ref Trigger.* A hiszterézis trigger használata főleg zajos jelek esetében jön jól. Amennyiben nem szeretnénk, hogy a zaj által keltett fluktuáció aktiválja a trigger eseményt, ezt a trigger típust kell alkalmaznunk. Lényegében ugyanúgy működik mint az éltriggerelés, azzal a különbséggel, hogy a trigger esemény bekövetkezte után egy megadott szintnek megfelelőre kell emelkednie vagy süllyednie a feszültségnek, ellenkező esetben a trigger eseményt érvénytelennek nyilvánítja az eszköz. Ennek egyetlen plusz paramétere van az edge triggerhez képest, a hysteresis, mely Voltban adja meg a feszültségváltozás minimális értékét.
- Digital: ez a triggertípus digitális jelek vizsgálatánál jön jól. Hosszabb nevén *Digital Edge Ref Trigger.* Ugyanazt tudja, mint az analóg edge trigger, de itt nem kell beállítani a feszültségszintet, mert a kifejezetten nagy feszültségugrásokat figyeli. A digitális triggernek ebből adódóan

eggyel kevesebb bemenő paramétere van, a feszültségszint nem beállítható. Minden más beállításban az analóg éltriggernél megszokott beállításokkal dolgozhatunk. Ezt a trigger típust azonban nem minden oszcilloszkóp kártya támogatja, ebben az esetben hibaüzenetben értesít minket, hogy az adott funkció nem támogatott.

- Window: hosszú nevén *Analog Window Ref Trigger*. Magyar névváltozatot nem találtam hozzá. Tükörfordításban „Ablak" triggernek lehetne nevezni. A többi triggernél megszokott *Channel, Coupling, Holdoff, Delay* paramétereken kívül van még három csak rá jellemző paraméter. A *low level*, a *high level*, és a *window mode*. Ez a trigger típus akkor aktiválódik, ha a jel belép, majd elhagyja a két paraméterrel megadott feszültségablakot. Tehát ha a feszültségszint eléri a low level paraméter által meghatározott értéket, de nem éri el a high level feszültségszintjét, hanem hamarabb csökkenni kezd, akkor a trigger esemény nem váltódik ki. A window mode paraméter határozza meg, hogy az ablakba belépésre, vagy annak elhagyására aktiválódjon a trigger esemény (vagyis a trigger pozíciója a memóriában a belépésnél vagy az elhagyásnál legyen). Könnyen belátható, hogy ez a trigger típus hasonló a Hysteresis triggerhez.
- Software: hosszab nevén *Software Ref Trigger.* Ez a trigger esemény nem a feszültségszinttől vagy a jelalak változásától függő, hanem programozható. Az *niScope Send Software Edge* VI-t meghívva a programból ez a trigger aktiválódik. Ez akkor jöhet jól, ha a felhasználó szeretné aktiválni – akár egy másik – programból a triggert, így az teljes mértékben konfigurálhatóvá válik. A szoftveres triggerelési típusnak két paramétere van, a *holdoff* és a *delay*, melyek ebben az esetben is ugyanazt jelentik, mint a többi trigger típus esetében.
- Video: ezt a trigger típust a programba nem implementáltam le, mert nem volt rá felhasználói szükséglet. Ez a trigger a PAL vagy NTSC szabványú videojelek detektálására van kihegyezve. Ezt főleg televíziók, vagy tv-s telekommunikációs berendezések hibakeresésekor vagy tesztelésekor használják. Azért készítettek erre is külön trigger típust, mert ezeknek a

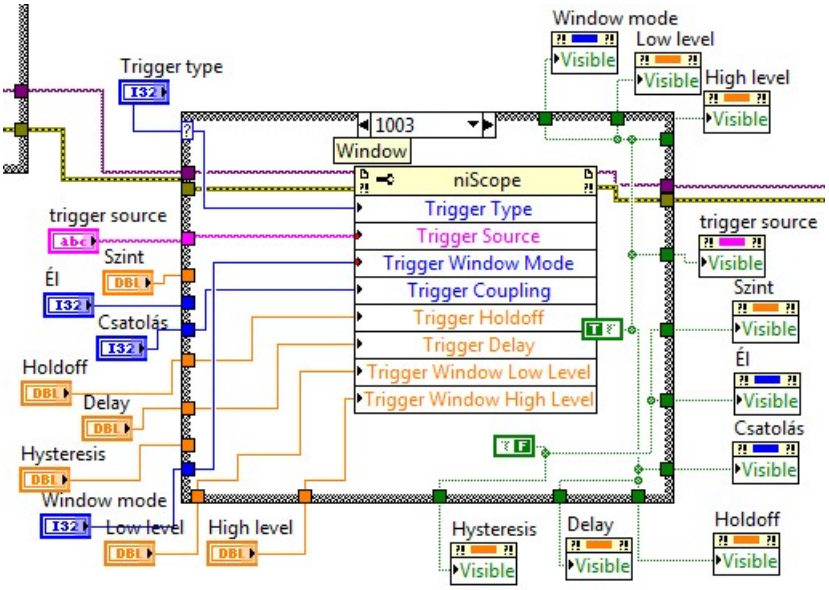
jeleknek a stabil detektálása más trigger típusokkal csak nagyon nehezen, vagy egyáltalán nem megvalósítható.

A programból az oszcilloszkóp kártyához tartozó trigger beállításokat szintén property node-okon keresztül végzem. A trigger típusától függően különbözőképpen jelennek meg a megfelelő beállító vezérlők (18. ábra).



18. ábra: Trigger beállítási lehetőségek annak típusától függően

A programból mindezt egy, a *Trigger típus* legördülőmező értékétől függő feltételes elágaztatásban végzem (a 19. ábrán a „window” eset látható):

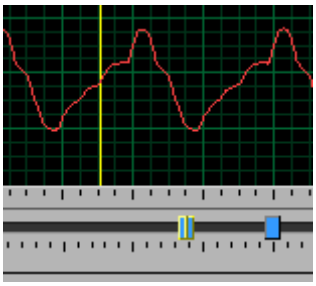


19. ábra: Trigger típusának esetválasztása

A különböző vezérlők láthatóságát is ez a struktúra szabályozza, méghozzá mindegyik elem *Visible* propertyjén keresztül. Ezt a programrészt sajnos nem lehet kiemelni subVI-ba, mert túl sok bemenettel és kimenettel rendelkezne, így szinte lehetetlen lenne hibát keresni benne, vagy újabb funkciókat hozzáadni.

A triggereléssel kapcsolatban még két megvalósítandó dolog maradt hátra. Az első a trigger relatív pozíciójának a beállítási lehetősége. Ezt a *Trigger Reference Position*, double típusú propertyn keresztül tehetjük meg. Ennek értéke 0 és 100 között lehet, ebből már sejthető, hogy a trigger relatív pozícióját százalékban kell megadni a vételezett jel tartományán. 0% a jel kezdete, 100% a vége. Ismét kérdéses volt, hogy ezt milyen vezérlővel oldjam meg, de ugyanazt a módszert választottam, mint a függőleges eltérítés esetén: egy újabb csúszka hozzáadását a vízszintes *Pointer slide*-hoz. A csúszka tartományát 0-100 közé kellett állítanom, és mivel a LabVIEW nem teszi lehetővé egy vezérlőn két csúszka külön skálára állítását. A csúszka tartománya -5 és +5 között van, ezért az aktuális értékhez hozzá kell adni 5-öt, és megszorozni 100-zal. Így ez a trigger referencia pozíció propertyjére kötve felhasználóbaráttá válik annak kezelése.

A trigger aktuális pozícióját is szeretnénk kijelezni a kijelzőre. Az XY graph képes kurzorokat hozzáadni a megjelenítőhöz, de ezek kezelése a vezérlő skálájához van igazítva, ezért nem lenne túl egyszerű megvalósítani. Az az ötletem támadt, hogy mi lenne, ha a trigger pozícióját is egy *plot*ként jeleníteném meg az XY grafikonon, a jelenlegi csatornák tömbjéhez még egy elemet hozzáadva. Ezekután ezt a jelet ugyanúgy görgethetem, mint a jelet, tehát mindig a „jelalakon marad". Ezt a megoldást egy külön subVI-ban programoztam le, ami bemenetként megkapja a trigger referencia pozícióját, és a vízszintes eltolás értékét, majd a kettőből kirajzolja a trigger pozíciót egy sárga vonal formájában, ahogyan az a 20. ábrán látható.
A trigger pozícióját állító csúszka szegélyét sárga színűre állítottam be, így könnyen megkülönböztethető a jel eltolását állító csúszkától.



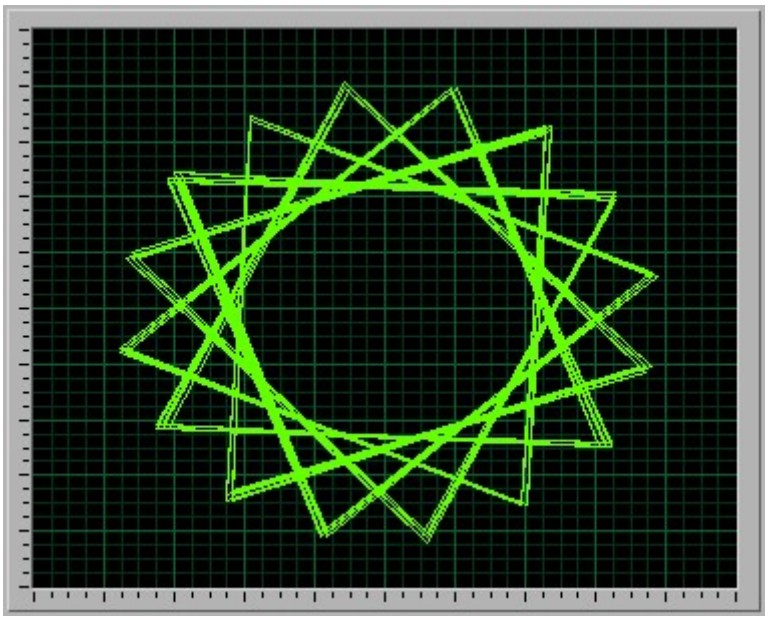
20. ábra: A trigger pozíciójának kirajzolása, és beállítása a csúszkával

A következő triggereléssel kapcsolatos beállítás a beolvasást végző VI-on történt, mégpedig a *Timeout* értékének beállítása. Mivel ennek beállítása minden esetben a vizsgált jeltől függ, ezért ezt a felhasználóra bíztam, egy létrehozott numerikus vezérlő formájában.

3.7. X-Y nézet

Egy valamire való oszcilloszkópban megtalálható ez a megjelenítési lehetőség. Ennek során nem a feszültségszintet ábrázoljuk az idő függvényében, hanem az egyik csatorna feszültségszintjét a másik függvényében.[4] Ennek gyakorlati jelentősége kevés területen létezik, de a nagyon egyszerű megvalósítás miatt kevés időt kell fordítani a fejlesztésére.

Ennek megvalósítását is külön subVI-ban végeztem, egyetlen bemenő és kimenő paramétere van. A programegységet a jelek transzformálása (nyújtás, eltolás) után kapcsoltam az adatvonalra, és egy feltételes elágaztatás vált a két megjelenési mód között. Így a jel minden transzformációja alkalmazódik az X-Y megjelenítésre is. Az időskála megváltoztatása az X-Y nézetre nincs hatással, mert az időskálától nem függ az X-Y diagram. A 21. ábrán látható két egymáshoz képest 90°-os fáziskülönbségű, azonos frekvenciájú jel XY grafikonja, amikor a jel alulmintavételezett (a mintavételezési frekvencia kisebb, mint a jel frekvenciájának kétszerese).



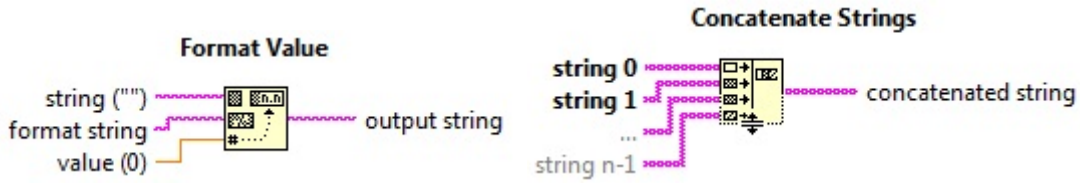
21. ábra: Két alulmintavételezett szinuszjel X-Y megjelenítése

3.8. Mérések végzése a jelen

Már van megjelenítőnk, és van skálánk is. Ezek alapján már viszonylag nagy pontossággal meg tudjuk mondani egy jelről annak frekvenciáját, vagy amplitúdóját. Azonban nem mindig elég az, ha az osztások alapján le tudjuk olvasni a megfelelő adatokat, az is jól jöhet, hogy pontosan megtudjuk egy jelről, hogy milyen pontokban milyen értéket vesz fel. Ehhez különböző méréseket kell elvégezni (a jel frekvenciája, amplitúdója), vagy pedig egy bizonyos pontban meg kell tudnunk mondani, hogy mennyi az aktuális feszültségérték (kurzorok).

3.8.1. Frekvencia és amplitúdó kijelzése

Az alapvető mérések megvalósításához a LabVIEW tartalmaz egy beélpített VI-t, amely az *Extract single tone information*. Ennek a VI-nak bemenő paraméterként meg kell adnunk a jelet, és kiszámítja belőle a jel frekvenciáját, és amplitúdóját. A következő kérdés ennek az adatnak a megjelenítése. Ennek külön kijelzőkre való kijelzésével ismét csak csökkenne a program felhasználóbarát mivolta. Mindenképpen olyan megoldást kell kitalálni, amellyel a megjelenítőre közvetlenül írhatunk szöveget, hogy minél egyszerűbb legyen a kezelőfelület. Ezt egyetlen propertyn keresztül tehetjük meg, a neve *Annotation List*. Az annotation (magyarázat) egy, a grafikonon megjeleníthető szöveg, általában tengelyek feliratozására, vagy grafikonrészletekhez megjegyzés fűzésére használják. Mivel pozicionálható, és sok tulajdonsága megadható, ezért tökéletesen alkalmas a nekünk szükséges adatok megjelenítésére. Azonban bemenő paraméterként sztringet igényel, amelyet nekünk kell felépítenünk a nyers adatokból, a 22. ábrán látható két VI segítségével:



22. ábra: Értékek szövegbe írására használt programegységek

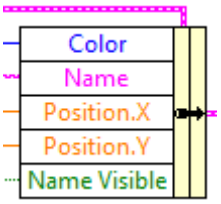
A format value első paramétere egy string, amelyhez hozzáfűzi a formátumsztringből és értékből előállított sztringet.

Én minden esetben a „%#.2p” formátumsztringet használtam, melynek elemei:

- %: a formátum megadás kezdetét jelzi
- #: azt jelzi, hogy a végső nullákat el kell hagyni (tört esetén)
- .2: azt jelzi, hogy az értéket 2 tizedes jegy pontossággal kell konvertálni

- p: azt adja meg, hogy az érték konvertálását a megfelelő SI előtagokkal végzi, így például a 0,05-ös értéket „50m”-re konvertálja, melyhez hozzáfűtve a mértékegységet jól értelmezhetően jeleníthetjük meg azt.

A megjegyzések használatánál ismét shift regisztereket alkalmaztam, ez azért előnyös, mert így nem kell mindig újra konstruálni a clustert, amelyet a property használ, hanem csak a megfelelő értékeket kicserélni benne. Ezt egy *Bundle by Name* nevű programozási struktúrával lehet megtenni (23. ábra).



23. ábra: Bundle by Name

Ezt használva nem kötelező minden részét kicserélnünk a struktúrának, amelyiket nem adjuk meg, az ugyanazon az értéken marad. A vezérlőt jobban széthúzva több elemnek is értéket adhatunk. Ezek használatával már könnyen kijelezhető a megjelenítőre a felirat: 

Így a felhasználónak nem kell számolnia, hogy megtudja ezeket az értékeket. Ezen kívül hozzáadásra került még egy jelölőnégyzet is, amivel megadhatjuk, hogy ezt az információt szeretnénk-e látni a kijelzőn, vagy sem.

3.8.2. Kurzorok kezelése

Van még egy módszer, amellyel különböző méréseket végezhetünk a jeleken, ez pedig a kurzorok kezelése. Az XY grafikonnak a kurzorok kezelésére is van megfelelő megoldása. Lehetőségünk van hozzáadni kurzorokat, módosítani azok idejét, lekérdezni az ahhoz tartozó feszültségértékeket. Ezzel csak egyetlen gond van, hogy ezek a kurzorok a megjelenítőnek a skálájához vannak rögzítve, tehát minden alkalommal, amikor átskálázzuk a megjelenített jelet, a kurzorok megjelenítését is módosítani kellene. Ehelyett ugyanazt a trükköt alkalmaztam a kurzorok megjelenítésénél, mint a trigger helyének megjelenítése esetében: újabb jelalakokat létrehozva.

Ennek első lépéseként felhasználóbarát módon kell megoldani a kurzorok hozzáadását, és törlését. Ez úgy a legfelhasználóbarátabb, hogy a kijelzőn való kattintásokat érzékeljük. Mivel a későbbiekben szeretnénk az oszcilloszkóp programablakát átméretezhetővé tenni, a lehető legdinamikusabban kell ezt megoldanunk, a kijelző méretének és egyéb paramétereinek lekérdezésével. A kattintás esemény kezelésekor paraméterben megkapjuk a kattintás pontos helyét, így már csak a kijelző paramétereket kell lekérdeznünk, és felhasználnunk hozzá az aktuális vízszintes transzformáció paramétereit. Ezekből a paraméterekből ki lehet számolni, hogy a jelen a mintavételezés középpontjához képest hol kattintottunk, a (3) képlettel:

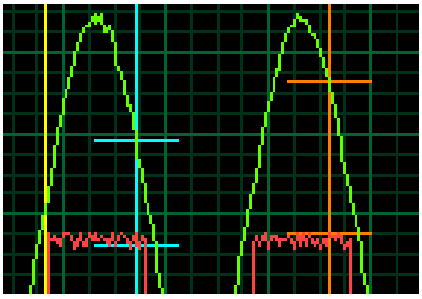
$$x = \frac{\frac{c-l}{r-l} \cdot 10 - 5 - o}{m} \qquad (3)$$

ahol x az az időpont a jelen, ahová kattintottunk, c a kattintás vízszintes pozíciója, l a rajzterület baloldali pozíciója, r a rajzterület jobb oldali pozíciója, o az aktuális vízszintes eltolás értéke, m pedig az aktuális időosztás reciproka. Az eseménystruktúra úgy van beállítva, hogy amennyiben egyszeres kattintást érzékelt, úgy a megadott helyhez hozzáadja a kurzort (a kurzorok helyét egy kételemű tömbben adjuk át az iterációk között egy shift regiszreren keresztül). Dupla kattintás esetén pedig törli a kurzort. Két kurzor kezelésére nyílik lehetőség, az egyiket az egér bal, míg a másikat a jobb gombjával vezérelhetjük.

A kurzorok megjelenítése, mint ahogyan azt fent említettem, a kijelzőre történik, újabb jelalakként. A kijelzéshez fel kell használni az előző lépésben létrehozott, valós adatokból álló tömböt, valamint szintén a jel transzformációjára vonatkozó adatokat, a csatornák jelinformációit, és a trigger relatív pozícióját. Ez utóbbi nem szükséges a megjelenítéshez, de a kurzorok adatainak megjelenítését is ugyanez az alprogram végzi majd, és ott a trigger eseményhez mért időkülönbséget kell majd kiírnunk. Ezen kívül kíváncsiak vagyunk a kurzor időpillanatában a két csatorna feszültségértékére. Erre azért is szükség van, hogy megfelelően ki tudjuk rajzolni a kurzort. A kurzornak a kinézete olyan lesz, hogy egy hosszú függőleges vonalat rajzolunk ki, és a

feszültségértéknél egy rövidebb vízszintes vonallal keresztezzük. A jelek transzformációjakor a kurzorokkal követnünk kell minden átalakítást, tehát a kurzorokra is alkalmazzuk az összes vízszintes és függőleges transzformációt.

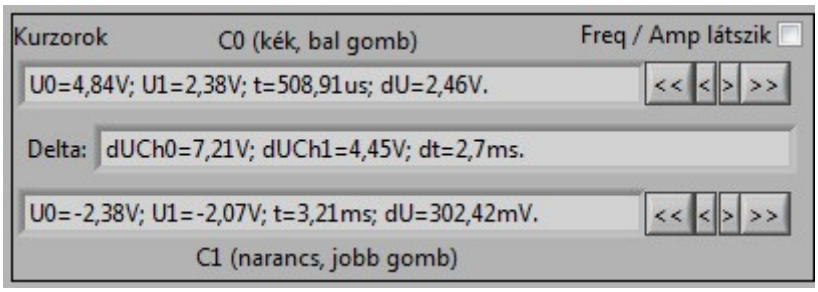
A megjelenítéshez különválasztjuk az X és Y tömböket, majd pedig a kurzor helyéhez tartozó feszültségértéket kikeressük. Ezt ciklussal kell megtennünk, mert ha a kurzort nem egy mintavételezési ponton helyeztük el, abban az esetben a kattintáshoz legközelebb eső pontot kell választanunk, és ehhez tudnunk kell az X értékeket is. A ciklus végeztével annak kimeneti csatornáján nem az aktuális X érték jelenik meg, hanem a tömbnek az az indexe, amelynél a megadott értéket találtuk. Ez alapján könnyedén kikereshető a csatornák Y értékéből a csatornához tartozó aktuális feszültségérték a kurzor idejénél. Amint ez megvan, már csak meg kell jelenítenünk, ehhez pedig külön-külön létrehozni a kurzor rajzának X-Y tömbjét. Az Y tömböt úgy kell létrehozni, hogy annak legalacsonyabb, és legmagasabb értékén ne legyen érvényesítve a függőleges irányú transzformáció, így az minden esetben „kilóg" majd a megjelenített területből. Amint ezzel készen vagyunk, a kijelzőn jobb illetve bal gombbal kattintva létrehozhatók a kurzorok, ezeknek kinézete a 24. ábrán látható.



24. ábra: A kurzorok képe a kijelzőn

Mivel a kurzorok hozzáadásával és megjelenítésével készen vagyunk, annyi van hátra, hogy megjelenítsük az azokhoz tartozó információkat. Ezek az információk már rendelkezésre állnak, mivel a megjelenés során kiszámoltuk őket. Annyi a teendőnk, hogy megfelelő formázás mellett megjelenítsük őket. Ezt először magán a kijelzőn tettem, és ott jelenítettem meg az értékeket megjegyzésekkel, ahol éppen a kurzor volt. Ez azért nem volt jó, mert ha nem

volt stabil a jel, akkor a felirat is „ugrált”, és képtelen voltam leolvasni, másrészről pedig le kellett volna kezelni, ha a felirat egy része a képernyőn kívül esik, és mivel a hossza változó, így ezt nem tehettem volna meg teljes bizonyossággal. Ezért úgy döntöttem, hogy külön megjelenítőt hozok létre a kurzor információinak. A kurzorok fejlesztése során felmerült egy olyan felhasználói igény is, hogy ne csak a megjelenítőn való kattintással lehessen egy kurzort elmozdítani, hanem legyenek lassabb és gyorsabb léptetést elősegítő gombok, mindkét irányba. Ezek a gombok magát a kurzor helyét állították be az ezt eltároló tömbben, így a megjelenítő részen nem kellett változtatni. Így kialakult a kurzorok kezeléséhez szükséges végleges felület, ide csoportosítottam a frekvencia és amplitúdó megjelenítését engedélyező jelölőmezőt is. A kialakított felület a 25. ábrán látható.

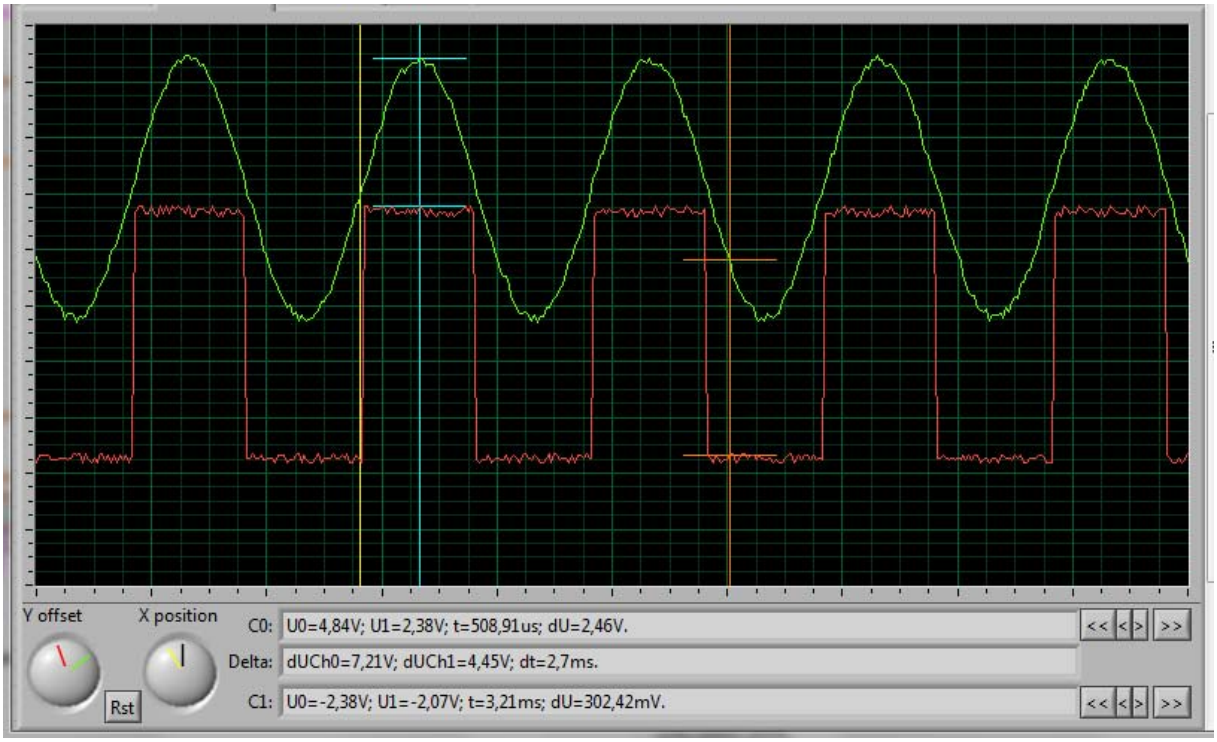


25. ábra: A kurzorok információit megjelenítő felület

Amint az látható, mindkét kurzornál megjelenik a két csatornán mért aktuális feszültségérték (U_0, U_1), ezek különbsége (ΔU), és a triggerhez képest relatív idő (t). Ezen kívül, ha két kurzor van hozzáadva, megjelennek a megfelelő csatornákra, és az időpontra mért különbségértékek. A képen is látható, hogy a kurzorok adatainak megjelenítése mellett található azoknak a finom léptetését szolgáló kezelőgombok. A finomabb kezelőgomb a nagy időosztás negyvened részével, vagy ha ez nagyobb, mint a mintavételezési gyakoriság, akkor a mintavételezési gyakoriságnak megfelelő értékkel lépteti arrébb a kurzort.

3.9. Nagy nézet

A felhasználói igények között szerepelt egy olyan nézeti lehetőség is, amely csak minimális beállítási lehetőségeket támogat, de a programablak felületének nagy részét egy ugyanolyan kijelző tölti ki, mint a normál kijelző, de nagyobb méretben. Így a jelek részletesebb vizsgálatára is lehetőség nyílik. A „Nagy nézet" és a normál nézet közötti átváltási lehetőséget egy úgynevezett *Tab control*-lal valósítottam meg. A felület kinézete a 26. ábrán látható.

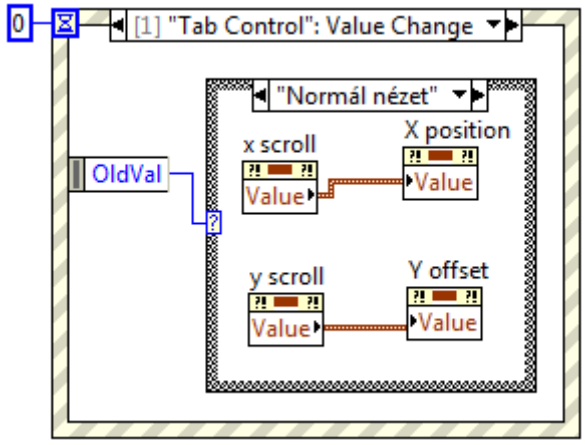


26. ábra: Nagy nézet

Amint az látható, a kezelőfelületen csak a legegyszerűbb vezérlők találhatóak meg, ezért úgy kell használni, hogy a jelalak skálázását a normál nézetben beállítjuk, és átváltunk nagy nézetre. Itt ugyanúgy megtaláljuk a kurzorok kezeléséhez szükséges vezérlőket, valamint a jelek eltolását elősegítő kontrollokat. Mivel a kurzorok hozzáadását dinamikusan oldottuk meg, így elég a megfelelő eseménystruktúrához hozzáadnunk ennek a kijelzőnek az esemény detektálását, ezáltal a nagy kijelzőn is van lehetőségünk kurzorok manipulálására egérrel. Mivel az eseménystruktúra nem statikus property node-okkal dolgozik, hanem megkapja a struktúrától paraméterként a referenciát, az

esemény kiváltódásától függő kijelző adatai alapján számolja ki a program a kurzor helyét. Ebből könnyen belátható, hogy ideális lenne, ha lehetne méretezni is az ablakot, ennek megoldásáról a „Finomhangolás” fejezetben írok bővebben.

A jel eltolását szolgáló vezérlőket gördítősávokról tekerőgombokra cseréltem, melyekhez szintén rendelhető két mutató. Ha egy vezérlő beállításaival szeretnénk létrehozni egy újat, másoljuk le a régi vezérlőt, majd az így kapott másolaton jobb gombbal kattintva, és a *Replace* lehetőséget választva hasonló kontrollra lecserélhető, az eredeti beállításainak megtartásával. Így azt nem kell újraskálázni, és nincs szükség semmilyen más paraméter megadására sem. Azonban az így létrehozott vezérlőkhöz is hozzá kell rendelnünk az eredeti vezérlő eseményeit, és property node-jait. Ez utóbbiakat úgy kell megtennünk, hogy a lapfüles vezérlő aktuális lapjától függően feltételes elágazással állítjuk be, vagy kérdezzük le a megfelelő vezérlők tulajdonságait. Azt is le kell kezelnünk, hogy mi történik akkor, ha átváltunk egy másik lapfülre. Szerencsére ennek kezelésére létezik egy esemény (*Value Change*), és az azt kezelő eseménystruktúra paraméterként megkapja mind a régi és az új értéket is. Így például egyik fülről a másikra váltáskor aktualizálhatjuk az egymásnak megfelelő vezérlők állásait, a 27. ábrán látható módon.

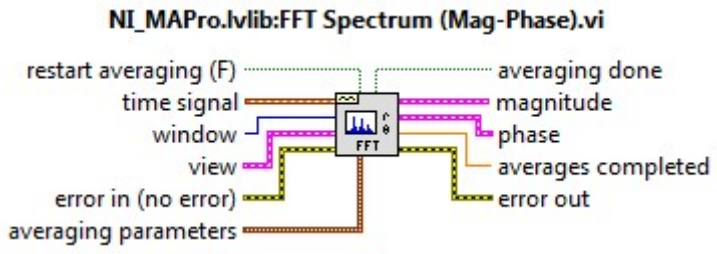


27. ábra: Lapfüles vezérlő lapváltási eseményének kezelése

Összességében állíthatjuk, hogy a nagy nézet sokkal alkalmasabb a jeleken lévő zavarok vagy zajok detektálására, és elemzésére.

3.10. Gyors Fourier transzformáció

A jelalakok vizsgálata során a felhasználónak gyakorta szüksége lehet nem csak a fő vivőfrekvencia, hanem az esetleges felharmonikusok frekvenciájának detektálására is. Ennek egyik nagyon elterjedt módja a gyors Fourier transzformáció (*Fast Fourier Transformation, FFT).* Ez egy olyan algoritmus, amellyel az idő-intenzitás függvényű jelalakokban a vízszintes időskálát frekvenciaskálával váltjuk le, így láthatóvá válik, hogy milyen frekvenciákból áll össze az adott jel.



28. ábra: Gyors Fourier Transzformációt megvalósító VI paraméterezése

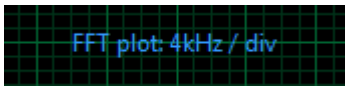
Ennek megvalósításához szerencsére létezik egy LabVIEW-ban előre megírt VI, így annak konfigurálásával könnyen elvégezhetjük az FFT-t (28. ábra).

Ennek mindössze egyetlen bemenetét, és egy kimenetét használva elérhető a kívánt hatás. A *time signal* bemenetre kötve a bemenő jelet megkapjuk annak frekvencia-bázisúra transzformált változatát a *magnitude* kimeneten. A kimeneten megjelenő jel pontosan annyi pontból fog állni, mint a bemenő jel, így az minél pontosabb, annál pontosabb lesz a frekvenciaskála is. A VI ezen kívül még képes átlagolni különböző módszerekkel több iteráción keresztül a kimenő jelet, így az egyre pontosabb lehet. Ennél a VI-nál ennek nincs sok jelentősége, de van egy másik VI (*FFT Spectrum (Real-Im)*), amely logaritmikus skálán ábrázolja a frekvenciaskálát, így az jóval pontatlanabb. Én az előbbit használom, mivel a mintavételezés hosszával, és a mintavételezési frekvenciával egyszerű módon állítható a kapott jel frekvenciatartománya, és pontossága. A frekvenciatartomány minden esetben az időalapú jel mintavételezési

frekvenciájának fele lesz, a pontosságot pedig a mintavétel hossza határozza meg. Az FFT módra váltást egy rádiógomb-szerű működéssel építettem be az alkalmazásba, mellyel lehet választani a normál megjelenítés, az X-Y kijelzés, és az FFT között: 

A kijelzőt FFT módba kapcsolva azonnal láthatóvá válnak az aktuális domináns frekvenciaértékek (29. ábra)

29. ábra: Egy zajos jel frekvenciaspektruma

A csúcsokhoz tartozó frekvenciaértékek kijelzésének megoldását úgy akartam megoldani, hogy minden csúcshoz tartozó annotation-ban jelöltem volna az arra vonatkozó értékeket. Ez egy, a fent látható, sok felharmonikussal rendelkező jel esetén olvashatatlanná tette a kijelzést, így más módszert alkalmaztam: minden esetben kiszámítottam, hogy egy fő osztásközhöz mekkora frekvenciaugrás tartozik, és ezt az adatot jelenítem meg: 

Ennek a megvalósítása is szintén a *Format value* VI segítségével történt, a megfelelő SI prefixumokat alkalmazó formátumsztringgel.

3.11. Jelalak mentése és visszatöltése

Külön felhasználói igény volt az is, hogy a jelalakok elmenthetők és visszatölthetők legyenek – mint ahogyan azt a digitális tárolós oszcilloszkópoknál megszokhattuk. Ez esetben a mentés formátumával szemben kikötés volt, hogy ember által is olvasható (*human-readable*) legyen, hogy esetleg azt más programokban is fel lehessen használni.

A LabVIEW maga biztosít egy-egy express VI-t hullámformák mentésére és betöltésére, így ezekkel kezdtem a kísérletezgetést. Az express VI-t az különbözteti meg a „nem express" VI-októl, hogy a beállítások nagy részét nem kötelező adatvezetéken keresztül átadni az alprogramnak, hanem a VI-ra duplán kattintva megnyílik az express VI konfigurálóablaka.

Mivel nekem jelalak elmentésére és betöltésére is szükségem van, ezért a *Write waveforms to file* és a *Read waveforms from file* express VI-okkal próbálkoztam. Ezeknek a beállító dialógusablakában többféle fájltípust is meg lehet adni, én a szöveges, tabulátorokkal elválasztott fájltípust választottam. Ki kellett jelölnöm, hogy minden adatot mentsen a bemenő WDT adattípusból, és befejeztem a konfigurálást. Mivel az express VI-ok közvetlenül nem kezelik a WDT adattípusokat, hanem időben változó jelek továbbítására úgynevezett *Dynamic Data* adattípust használnak, ezért a kettő között konverziót kell végrehajtani, melyet a *Convert to Dynamic Data* VI-al lehet megtenni. Ennek kimenetét az express VI bemenetére kötve a Mentés gomb lenyomásakor lefutó eseményvezérlőben elvileg készen is vagyunk. Leteszteltem, a létrejövő .lvm (*LabVIEW Measurement*) fájl tényleg tartalmazta a két csatorna feszültségértékeit.

Hasonló módszerrel megvalósítottam a jel beolvasását is. Ehhez azonban valahogy váltanom kellett a beolvasott jel, és a szkópkártyáról olvasott jel megjelenítése között. Ezt nem volt túl nehéz megoldanom, mivel az aktuális beolvasott jelet egy shift regiszteren keresztül továbbvittem a következő

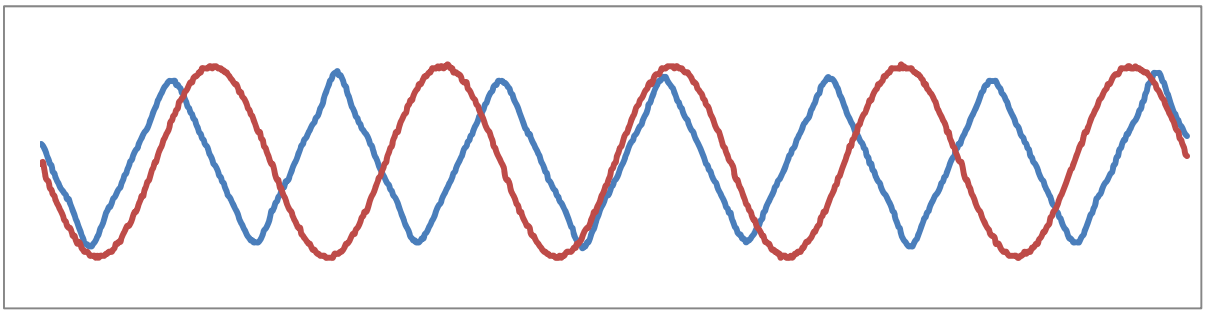
iterációra. Így csak azt kellett megvalósítanom, hogy egy kapcsolónak a lenyomásakor a jelet ne az oszcilloszkópból olvassa, hanem a shift regiszterből. Ezért az egész beolvasási folyamatot egy case struktúrába foglaltam, és ha a *Pause* gomb lenyomott állapotban van (*Switch when pressed* mechanikus működés), akkor az összes niScope driverhez tartozó VI-t kihagyja, és így egy statikus jel látható a kijelzőn. Mivel a jeltranszformációk a beolvasás után vannak megvalósítva, ezért egy statikus jelet is skálázhatunk, eltolhatunk, tehetünk rá kurzorokat, sőt FFT vagy X-Y nézetbe is kapcsolhatjuk a megjelenítőt.

Miután ezzel készen voltam, a jelalak fájlból való beolvasásának sikeressége után nem volt más dolgom, mint hogy programból egy lokális változón keresztül a *Pause* gombot lenyomott állapotba helyeztem, így elősegítve, hogy a kijelzőn a beolvasott jelalak statikus képe legyen látható. Ezzel készen voltam, így leteszteltem a beolvasást is. Hiba történt, a kijelzőn csak két vízszintes vonal volt látható. Először az elmentett jelre gyanakodtam, de miután leellenőriztem, igazolódott, hogy a hiba máshol keletkezik. Csináltam gyorsan egy tesztesetet egy külön VI-ba az express VI-okból, és szerencsémre a jelalakot megjelenítő vezérlőn be volt kapcsolva az automatikus skálázás. Az exportált .lvm fájlt tüzetesebben megvizsgálva jöttem rá, hogy a *Write waveforms to file* VI a fájlba nem írt bele semmilyen időzítésre vonatkozó információt, azaz beleírta, de a Δt minden esetben 1 s volt.

Mivel a fájlból olvasó VI úgy is konfigurálható, hogy szöveges adatokkal dolgozzon, én pedig szerettem volna megtartani az X értékeket, ezért magam írtam meg a jelalak fájlba exportálását végző alprogramot. A mentés gomb megnyomásakor így egy fejléccel ellátott, tabulátorokkal tagolt fájlba történik az összes mintavételezési pont kiexportálása. Ezt a műveletet a LabVIEW beépített fájlkezelő VI-okon keresztül sikerült leprogramoznom. Mint minden programozási nyelv, a LabVIEW is referenciaként kezeli a fájlváltozót, így a fájlkezelésnek itt is ugyanazok a szokásos műveletei. A *Format into file* VI

különböző adatok fájlba való írását engedélyezik formátumsztring alapján. Ez a formátumsztring a C-ben megszokott változat, tehát nem a numerikus adatok szövegbe való formázását segíti elő, de többféle adattípust, és az escape szekvenciákat is támogatja (például „%d%s\t%f\n”).

Szerencsére az importáló express VI beállító felületén megadható, hogy LabVIEW fejléc nélküli fájlból is tudjon olvasni, valamint, hogy a fájl első „oszlopa” az időtengelyt jelentse, így ennek beállításával több dolgom nem is volt, következhetett a tesztelés. Egy elmentett két csatornás jelalkot visszaolvastam, és tökéletesen működött, majd Excelbe is beolvastam (30. ábra). A legfőbb előnye a jelalakok visszaolvasásának, hogy azokon minden olyan műveletet végrehajthatunk, amelyet egy „aktuális” jelen tennénk, azzal a kivétellel, hogy a jel statikus lesz.



30. ábra: A kiexportált jel Microsoft Office Excelben megjelenítve

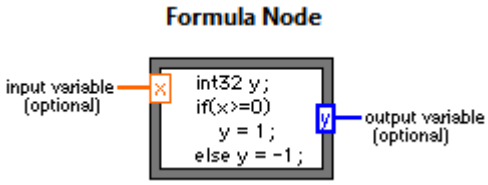
3.12. Formulák

Egy oszcilloszkóp sokat ér, de vannak funkciók, amelyeknek kifejlesztése egyszerű, az eszköz értékén mégis sokat növel.

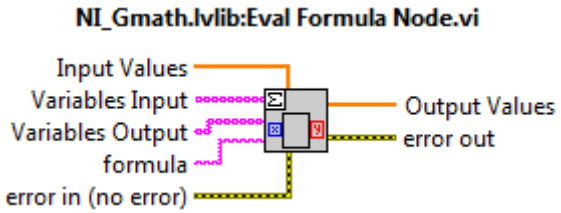
Ilyen képesség a jelalakok transzformációjának lehetősége is. Sok esetben szükség lehet arra, hogy valamilyen okból transzformációkat végezzünk a jeleken. Ilyen lehet például két digitális jel kivonása egymásból, jel invertálása, vagy éppen két jel szorzása.

A felhasználói igény mindössze annyiból állt, hogy alapvető transzformációkra képes legyen a program, például a két csatorna jelének összeadására, jel

invertálására, stb. Ezzel szemben én szerettem volna egy kicsit rugalmasabb beállítási lehetőséget biztosítani. A LabVIEW lehetővé teszi kódon belül különböző scriptek vagy műveletek leprogramozását szövegesen, C nyelvhez hasonló szintaxissal, ez a *formula node* (31. ábra). Amint azt az ábrán is láthatjuk, lehetőség van bejövő és kimenő változókat létrehozni, amelyekre hivatkozhatunk a formulán belülről. Ezt a struktúrát már alkalmaztam a kurzorok pozíciójának kiszámításához.

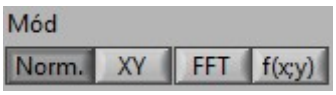


31. ábra: Formula Node programozási struktúra



32. ábra: A Formula Node paraméterezhetőbb változata

Most nekem valami másra lenne szükségem, mivel a *formula node* szöveges mezőjének az előlapról nem tudok értéket adni. Ennek megoldására született meg a LabVIEW-ban az *Eval formula node* VI (32. ábra). Ez a blokkdiagramon elhelyezhető struktúra VI változata, így a formula is dinamikusan paraméterezhető. A *Variables Input* sztring tömbben azt adjuk meg, hogy hogyan nevezzük a bemenő változókat, a *Variables Output* sztring tömb pedig a kimenő változókat. Ezeknek 1:1 megfeleltetése az *Input Values* és az *Output values* tömb, melyekbe/ből a sztring tömbben megadott értékeket tesszük/várjuk. A *Formula* szöveges paraméterben adjuk át a kódot, amit szeretnénk, hogy kiértékeljen a VI.

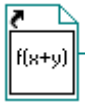


33. ábra: A módválasztó rádiógombok végleges kialakítása

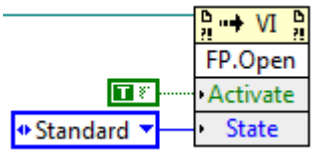
A lehetőség programba integrálásán sokat gondolkodtam, míg végül arra jutottam, hogy külön alprogramba helyezem el, melynek előlapját megjelenítem, amikor a módválasztó gombok közül a felhasználó lenyomja a gombot (33. ábra).

Ezzel viszont kis nehézségbe ütköztem. Ha egy VI-t beágyazunk egy programba, akkor lehetőségünk van megadni, hogy amikor rá kerül a vezérlés, akkor megjelenítse annak előlapját is. Ez ciklusba szervezett programnál gondot okoz, mivel minden iterációnál szeretnénk interakciót a beágyazott VI ablakával. Ennek megoldására jobb gombbal a beágyazott VI-on kattinva, majd a *Call Setup…* opciót választva, itt a *Load and retain on first call* opciót kell kiválasztani, és így megjelenik a VI, amint rá kerül a vezérlés, de nem villog az ablak. Ezzel a megoldással az a probléma, hogy ilyen esetben, ha be szeretnénk zárni, majd újra megnyitni az ablakot, már nem fog működni, mert ez esetben már nem az első hívás lesz az adott alprogramra nézve.

Ezért ki kellett dolgoznom egy olyan módszert, amellyel a VI megnyitása egy gombnyomással megtehető, majd annak bezárásakor a megjelenítési mód visszaálljon „Normál" módba. Ezen kívül azt is el kell érni, hogy a megnyílt előlap ne legyen modális, azaz szabadon lehessen bármelyik előlapot használni. Erre az adott VI előlapján a *File → VI Properties…* opciót választva, ott a *Window Appearance* kategóriában a *Customize…* gombra kattintva ki lehet választani, hogy modális legyen, vagy ne. Ahhoz, hogy meg tudjuk nyitni a VI előlapját, a *property node*-hoz hasonló szerkezetet kell használnunk, az *Invoke node*-ot (35. ábra). Ez annyiban tér el a property node-tól, hogy nem az objektum egy tulajdonságának beállítására való, hanem annak valamilyen tagfüggvényének a meghívására. Ezeknek akár több be- és kimeneti értékei lehetnek. Hasonlóan, mint a property node-nál, az invoke node-hoz is kell egy referencia. Ezt VI-ok esetében a *Static VI Reference* „konstanssal" tudjuk legkönnyebben beállítani, annyi a dolgunk, hogy ennek keretébe belehúzzuk a hivatkozott alprogram ikonját (34. ábra). Az így létrejövő referenciát már használhatjuk az invoke node-ban az előlap megnyitásához.

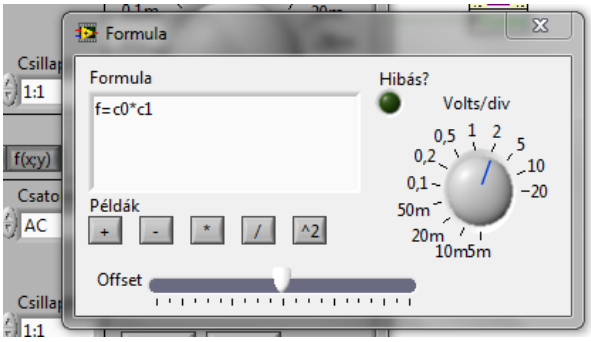


34. ábra: Statikus VI hivatkozás



35. ábra: Invoke node

Programból megnyitva egyből megjelenik a már megtervezett előlap (36. ábra)

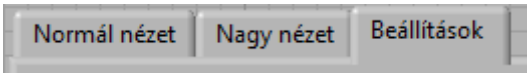


36. ábra: A formula beállító ablaka

A szövegmezőbe írhatjuk a formulát, akár egészen bonyolultakat is, melyek több utasításból is állhatnak. Ha a beírt formula hibás, akkor azt a LED jelzi. A szokásos vezérlőket ide is hozzáadtam a jelalak nyújtásához / eltolásához. Az előlapra különböző példák is be vannak építve, hogy aki először látja a programot, ki tudja következtetni a formula szintaxisát. Jelen esetben c0 és c1 jelöli a csatornákat, f pedig a formula jelét (kimenet). Mialatt a formula ablak nyitva van, ugyanúgy használhatóak a főprogram vezérlői is.

3.13. Finomhangolás

Mivel végeztem a felhasználók által kért funkciók megvalósításával, egy kicsit barátságosabbá, felhasználóbarátabbá szerettem volna tenni a programot. Minden olyan tevékenység, amellyel a felhasználónak kell bajlódnia, hasznos időt vesz el tőle.



37. ábra: A lapfüles vezérlő végleges elrendezése

Első apróbb módosítás, hogy a lapfüles vezérlőhöz hozzáadtam egy *Beállítások* fület, ahol minimális beállítási lehetőséget szeretnék biztosítani (37. ábra)

Ide egyelőre kijelzésként az aktuális relatív triggerpozíciót tettem, és az eszköz aktuális hőmérsékletének kijelzését. Hozzáadásra került még egy *Slow* gomb is a *Pause* gomb mellé, amellyel lelassítható a program fő ciklusa, például kurzorok könnyebb kezeléséhez. Ezen kívül a beállítások fülre tettem még egy *Program késleltetés* beállítómezőt, amely szüneteket iktat a fő ciklus iterációjába, hogy ne terhelje annyira a processzort.

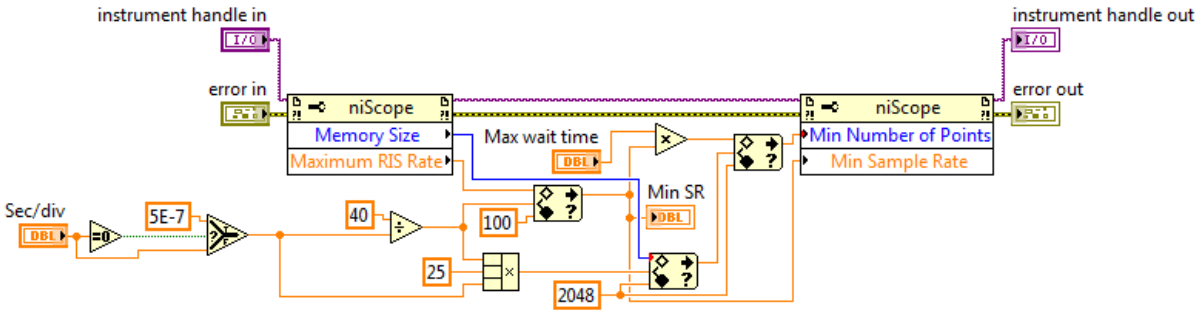
3.13.1. Feszültségtartomány

Az niScope driver 3.8-as verziója sajnos nem ad lehetőséget a kártya maximális feszültségtartományának lekérdezésére. Mivel szerettem volna egy automatikus feszültségtartomány-állítást a programba építeni, erre a paraméterre mindenképpen szükségem volt. A feszültségtartomány beállítása nem fokozatmentesen történik az oszcilloszkóp-kártyákon, hanem lépcsőkben. Ezért nem egy maximális feszültségtartományt detektáltam, hanem az összes lehetséges feszültségtartományt. Ezt úgy teszem, hogy beállítom a lehető legalacsonyabbra, majd lekérdezem, majd növelem, ismét lekérdezem, és így tovább. Így a driver gyakorlatilag „végigugrál" a lehetséges feszültségtartomány-beállításokon. Az inicializációs szekcióban ez a VI átadja a kapott lehetséges értékeket egy tömbnek, és a *Beállítások* fülön lévő vezérlőnek. Ebből kézzel is ki lehet választani feszültségtartományt, ha az automatikus módszer valamiért nem megfelelő.

Az automatikus feszültségtartomány-állítás a fő ciklusban van megvalósítva, egy külön subVI-ba kiemelve. Ez úgy működik, hogy megnézi, mennyi volt az előző iteráció amplitúdója, és ha ez megközelíti az aktuális feszültségtartományt, akkor növelni fogja azt a következő ugrásig, mindaddig, amíg el nem éri a maximális tartományt. Ennek a megoldásnak egyetlen hátránya van, hogy ha lassú az iteráció, a feszültségtartományt is lassan állítja be. Felvetettem annak a lehetőségét is, hogy ne alulról, hanem felülről közelítse a jelet, de ez nem kivitelezhető, mivel nem tudnánk eldönteni, hogy mikor változik a jel.

3.13.2. Mintavételezés

Mivel a jelből egy adott pillanatban csak kevés látható, ezért érdemes úgy megválasztani a mintavételezés hosszát, hogy ne statikus legyen minden időosztásnál, főleg, mert a kisebb időosztásnál a jeleket nagyobb mintavételezési frekvenciával olvassuk. Ezért a mintavételezés hosszát, és a mintavételezési frekvenciát is minden esetben az időosztástól függően kell beállítanunk. Ennek megvalósítása is külön VI-ba került (38. ábra). A kódrészleten látható, hogy minden időosztásnál negyvenszer történik mintavételezés, vagy legfeljebb a maximális RIS gyakorisággal. A RIS a *Random Interleaved Sampling*, azaz a véletlenszerű időközönként „összefésült” mintavételezés, ami annyit tesz, hogy ténylegesen nem a megadott frekvenciával történik a mintavételezés, de ismétlődő jeleknél többször végez mintavételezést, így pontosabb jelet láthatunk. A driveren az *Enforce Realtime* propertyt hamisra állítva engedélyezzük ezt a mintavételezési módot. Így ha a mintavételezési frekvencia átlépi a valós idejű mintavételezési határt, automatikusan RIS módba kapcsol.



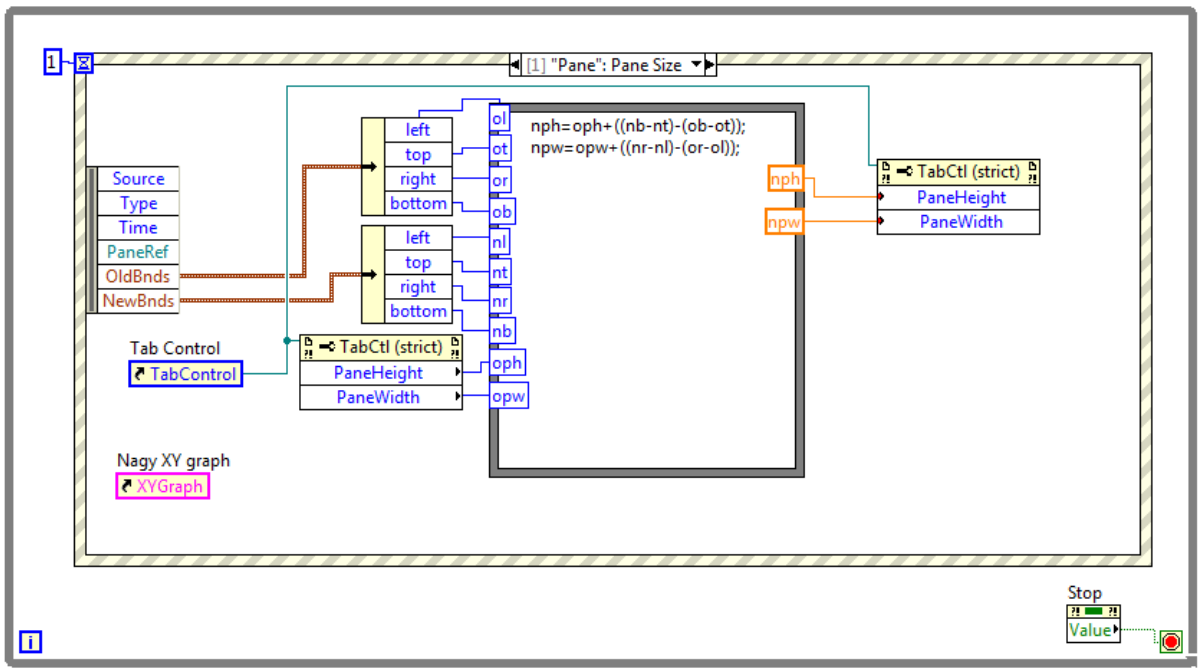
38. ábra: Mintavételezés hosszát és gyakoriságát beállító alprogram

A mintavételezés hosszát 2048 pontban minimalizáltam, ez 2 KB-ot jelent mindkét csatornára (8 bites vertikális felbontás esetén). Ennél rövidebb jel vételezésére soha nincs szükség, így a hibákat is könnyen elkerüljük. A mintavételezés hosszát ezen kívül az aktuális időosztás 25-szörösére állítom be, ez egyrészt biztosítja, hogy a görgető vezérlőkkel soha ne tudjunk „üres területre” gördíteni, másrészt kisebb időosztásnál kiküszöböli, hogy az oszcilloszkóp a kisebb frekvenciás jelet kiszűrje AC csatolású módban.

Ezen kívül létrehoztam a *Beállítások* fülön egy beállítómezőt, amellyel megadható, hogy maximum mennyi ideig várjon a program a jel beolvasásával. Ez azt jelenti, hogy ha ezt a mezőt kisebbre állítjuk, és az aktuális időosztás 25-szöröse hosszabb ennél az időtartamnál, akkor csak a felhasználó által beállított időnek megfelelő jelhosszt olvas be. Ennek a beállításnak az értéke befolyásolja a jel fetchelését végző VI *timeout* paraméterét is.

3.13.3. Automatikus méretezés

Főleg a nagy nézet miatt érdemesnek éreztem leprogramozni, hogy az ablak átméretezésekor bizonyos objektumok az ablakkal együtt méreteződjenek. Ezt egy, a fő ciklustól különálló, azzal párhuzamosan futó ciklusban kellett megvalósítani, hogy minden esetben megtörténjen a méretezés. Először csak két objektumot szerettem volna méretezni a program ablakával, a lapfüles vezérlőt, és a nagy nézet megjelenítőjét (39. ábra).



39. ábra: A programból végül kikerült automatikus átméretezés megoldása

A képen nem látható a nagy nézet megjelenítőjének az átméretezését végző programrész. Miután ezt megvalósítottam, döbbenve tapasztaltam, hogy az átméretezést rettenetesen lassú sebességgel hajtotta végre a program, mivel a

méretezés alatt végig látható volt az ablak tartalma. Szerencsére a VI beállításai között megtaláltam azt a lehetőséget, hogy minden objektum az ablak átméretezésével együtt méreteződjön, és ezt valamivel gyorsabban hajtja végre a program, bár még így is viszonylag lassú.

3.13.4. Dokumentáció

A programhoz külön dokumentáció készítésére nem volt igény. Mivel a programot használóknak nem fog papír alapú dokumentáció a rendelkezésükre állni, ezért a programba kell beépíteni azt. Minden előlapi objektumhoz lehet megjegyzést fűzni, a tulajdonságlapján a *Description* fülön. Ezen a fülön HTML kódokkal lehet a megjelenő szöveget formázni. Az ide beírt szöveg a Ctrl-H billentyűkombináció, vagy a *Help→Show context help* menüre kattintva hozható elő, és az egeret a kérdéses objektum fölé mozgatva megjelenik a leírás.

4. A program futtatásához szükséges környezet

- Minimális LabVIEW verziószám: 8.6.
- NI-Scope driver: minimum a 3.51-es verzió
    - ingyenesen letölthető az ni.com-ról
- A program a VDScope.lvproj projektből indítható,
    - ezen belül a Scope TopLevel.vi tartalmazza a programot
- A program standalone is futtatható, az NI-Scope driver megléte esetén, az App könyvtárban található, a VDScope.exe nevű fájl

5. Összegzés

A program fejlesztése során hatalmas segítséget jelentett az, hogy felhasználói oldalról folyamatosan visszajelzéseket kaptam, és így tökéletesen meg tudtam felelni az elvárásoknak. A program funkciói tekintetében nem kaptam szabad kezet, de azoknak a kialakítása során mindig a könnyű kezelhetőséget tartottam szem előtt.

A program már mindenben megfelel a felhasználói igényeknek, azonban vannak még lehetőségek, amelyeket be lehet építeni a programba.

Az egyik ilyen lehetőség testre szabható szűrők hozzáadása. Az NI-Scope driver által támogatott eszközök nagy része támogatja a kártyán történő jelfeldolgozást (*Onboard Signal Processing*), ezek segítségével különböző szűrőket adhatunk hozzá a vételezett jelekhez, ha szeretnénk kiszűrni a jelekből pl. a nagyfrekvenciás zajokat.

Másik lehetőség a továbbfejlesztésre egy „automatikus" beállító gomb létrehozása, melynek hatására az aktuális jelfrekvenciából és az amplitúdóból a program beállítja a vízszintes és függőleges beállításokat.

Több nyelv kiválasztásának lehetősége elősegíthetné azt, hogy a program felhasználói interfészét a magyarul nem tudó kollégák is tudják használni. Erre azért nem volt igény, mert az oszcilloszkópokon megszokott kezelőszervek egyértelműen vannak elnevezve, és külföldi dolgozó esetén őket megfelelően betanítják azok használatára.

Rengeteg más megvalósítható fejlesztés jöhet még elő felhasználói igényként, a lényeg az, hogy a LabVIEW-nek köszönhetően ezek a módosítások könnyedén beleintegrálhatóak lesznek a programba, míg egy fizikai oszcilloszkóp esetén csak drágább típusok tudnak újabb funkciókat.

6. Köszönetnyilvánítás

A dolgozat megírásában, és a program létrehozásában, vagy egyéb, a dolgozat időbeli elkészültének elősegítésében a következők nagy segítséget nyújtottak, ezét külön köszönet illeti őket:

- Tóth István, Value Engineer – NI Hungary Kft.
  A felhasználói igények feltérképezésében, valamint a program megírásának folyamatos koordinálásában nyújtott segítséget
- Dr. Szabó István, tanszékvezető egyetemi docens – Debreceni Egyetem Szilárdtestfizikai Tanszék
  A LabVIEW programozás alkalmazásszintű elsajátításában, és a szakdolgozat írásában, annak felépítésében nyújtott segítséget
- NI Hungary Kft. – A National Instruments Corpotation debreceni leányvállalata
  Rendelkezésemre bocsátotta a dolgozat megírásához szükséges hardvereszközöket, és a LabVIEW fejlesztőkörnyezetet

7. Irodalomjegyzék

[1]: Csepregi-Horváth Kázmér: Oszcilloszkóp méréstechnika. Műszaki Könyvkiadó, Budapest, 1976.

[2]: LabVIEW User Manual. *April 2003 Edition.* Part number: 320999E-01. National Instruments Corporation, Austin, U. S. A. (Texas), 2003.

[3]: User's and Service Guide. *3000 Series Oscilloscopes.* Publication number: D3000-97000. Agilent Technologies, Colorado Springs, U. S. A. (Colorado) 2005.

[4]: Digital oscilloscopes: For best results, understand how they work – http://www.edn.com/archives/1996/070496/14dfcov.htm, 2009. Október 16.

[5]: Digital Storage Scope.FAQ – http://www.morgenstierne.no/dsofaq.htm, 2009. November 2.

[6]: The Oscilloscope – http://www.upscale.utoronto.ca/GeneralInterest/Harrison/Oscilloscope/Oscilloscope.html, 2009. Május 16.