VIRTUÁLIS MŰSZEREK ÉS A LABVIEW

VIRTUAL INSTRUMENTS AND THE LABVIEW

Disznós Imre, fandi@externet.hu
Debreceni Egyetem Informatika Kar

1. Bevezetés

A mérés és automatizálás terén mára piacvezető szerepet betöltő National Instruments Corporation 1986-ban vezetett be egy olyan szoftvert, melynek segítségével mérnökök és kutatók ú.n. virtuális műszereket tudtak készíteni. Ennek továbbfejlesztett változatát – a LabVIEW 5.0-t – az 1999-es év tesztprogramjának is nevezték. Jelenleg a legújabb verzió a LabVIEW 7.1, amely az évek során számos eszközzel bővült, és kvázi szabvánnyá vált a virtuális műszerek világában [1].

A virtuális műszerek nemcsak az iparban, hanem az oktatásban is hatékonyan alkalmazhatóak. Segítségükkel helyettesíthetők a hagyományos műszerek, és kihasználhatóak a PC nyújtotta előnyök.

Cikkemben a LabVIEW-t programozói szemszögből szeretném bemutatni, kitérve a G nyelv eszközeire és specialitásaira. Továbbá megemlítek néhány tipikus, a nyelv jellegzetességeiből adódó buktatót. Végül összegzem a LabVIEW-val való munkám során szerzett tapasztalataimat.

A LabVIEW egy grafikus fejlesztő eszköz, amely egy adatfolyam nyelvre épül, melynek a neve G. A LabVIEW külön érdekessége, hogy benne nem szövegalapú forráskódot írunk, hanem egy úgynevezett blokk diagramot rajzolunk. Ez lesz a „forráskód”.

Ezt az eszközt kifejezetten mérnökök (méghozzá villamosmérnökök) számára mérési, automatizálási és folyamatirányítási célokra fejlesztették ki. Segítségével úgynevezett virtuális műszereket (Virtual Instruments röviden: VI) hozhatunk létre.

2. Virtuális műszerek

A virtuális műszer egy olyan program, amely egy fizikai műszer külső megjelenését és működését modellezi.

Miért használnak virtuális műszereket? Képzeljünk el például egy gyártósort, ahol az egyes gyártási fázisokban a termékek fizikai paramétereinek tesztelésére számos mérőműszert használnak, amelyek ára elég magas. Ráadásul sok helyet foglalnak el. Ha valamelyik alkatrészük meghibásodik, azt ki kell cserélni (vagy rosszabb esetben új berendezést kell vásárolni). Egyszóval az ilyen megoldásnak sok hátránya van. Erre a problémára adnak választ a virtuális műszerek. Azaz fogjunk egy erős számítógépet, tegyünk bele néhány digitalizáló, illetve jelgeneráló kártyát, telepítsük fel rá egy alkalmas fejlesztőeszközt, és máris elkészíthetjük a megfelelő virtuális műszereinket.

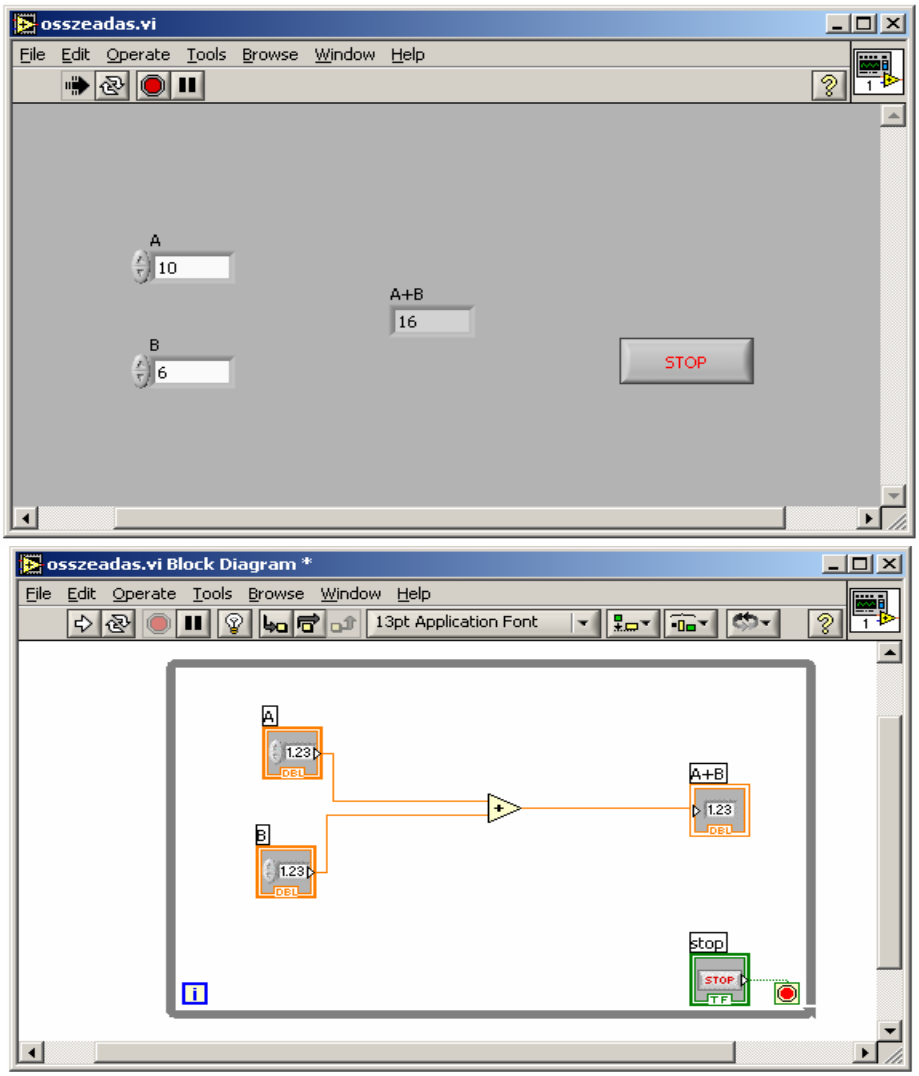
További érv a virtuális műszerek mellett, hogy velük kihasználhatóak a számítógép nyújtotta előnyök (kommunikáció, naplózás stb.)

Virtuális műszer számos fejlesztőeszközzel létrehozható. Cikkem témájául a LabVIEW-t mint fejlesztőeszközt azonban nem csak azért választottam, mert az egyik legelterjedtebb a virtuális műszerek világában, hanem azért is, mert programozási szempontból számos érdekességet tartalmaz.

3. A fejlesztői környezet

Maga a fejlesztői környezet két fő részből áll, melyek külön ablakokban jelennek meg. Az egyik az előlap (Front Panel) nevet viseli. Ezen hozható létre a felhasználói interfész, melynek kialakításakor számos komponensből válogathatunk. A másik fő rész a már fentebb említett blokk diagram (Block Diagram).

Az előlapi komponenseknek két nagy csoportja van: kontrollok és indikátorok. Az előbbiek az input adatok bevitelére szolgálnak, míg az utóbbiak az outputot jelenítik meg. Ezeket a komponenseket a kontroll palettán találjuk. Itt találhatóak még a díszítő elemek is, melyekkel szebbé tehetjük a programunk felhasználói felületét.



1. ábra
Az előlap és a blokk diagram

Amikor LabVIEW-ban programozunk szinte mindig az egeret kell használnunk, ezért meg kell ismerkednünk az egérkurzor funkcióival. Az ezek közötti váltásra szolgál az eszközök paletta. Az eszközök neve föntről lefelé és balról jobbra a következő: automatikus eszközválasztás, működtető eszköz, pozícionáló/átméretező, címkéző/feliratozó, huzalozó, gyorsmenü eszköz, görgető eszköz, megszakítási pont, próba pont, színmásoló, színező.

4. A LabVIEW nyelve a G

Amint már a bevezetőben említettem a LabVIEW-ban adatfolyam nyelven programozunk, melynek eredményeként egy blokk diagram jön létre. A blokk diagramban a csomópontok egyszerű vagy összetett műveletek végrehajtására szolgálnak. Több fajtájuk van: függvények, struktúrák és subVI-ok. Ezeket a függvény palettán találjuk. Egy csomópontnak nulla, egy vagy több bemenete és nulla, egy vagy több kimenete lehet. A csomópontokat huzalok kötik össze, melyek az adatáramlást biztosítják. A csomópontok bemeneti és kimeneti pontjait termináloknak nevezzük.

Egy csomópont akkor hajtódik végre, ha minden adat elérhető a bemeneti terminálokon. Ha adott pillanatban több ilyen csomópont is van, akkor azok elvileg párhuzamosan futnak le. A gyakorlatban ezek sorrendje indeterminisztikus. Mindezek miatt a LabVIEW-ban nagyon könnyű megvalósítani a párhuzamos programozást, viszont a futási sorrend biztosítására extra erőfeszítéseket kell tennünk.

A blokk diagram az adatáramlási sorrendben hajtódik végre, azaz az adat egy csomópont kimeneti termináljáról az adott csomóponttal közvetlen összehuzalozott másik csomópont bemeneti termináljára érkezik. Vagyis tévhit, hogy az adatáramlás balról-jobbra irányú, ez utóbbi csak egy konvenció, ami bár nem kötelező, de módszertanilag helyes ezt az irányzatot követni.

Az adatok típusát a vezetékek színe és vastagsága jelzi. Helyhiány miatt sajnos nincs rá mód, hogy felsoroljam a G nyelv adattípusait, de az érdeklődő olvasó a hivatkozott irodalmakban bőséges információt talál róluk.

A csomópont tehát a VI végrehajtási egysége. A legegyszerűbb csomópont a függvény, amely a blokk diagram alapvető építőeleme. Nem rendelkezik előlappal és blokk diagrammal, szemben a subVI-al. Ez utóbbi az alprogramnak felel meg. Alkalmazásuknak számos előnye van. Modulárissá, jól átláthatóvá teszi a programunkat, amely megkönnyíti a tesztelést. Emellett az újrafelhasználhatóságot is támogatja.

A függvényekhez hasonlóan a LabVIEW-ban számos beépített subVI létezik, azonban mi is könnyedén hozhatunk létre újakat.

Itt kell megemlítenem a 7.0-s verzióban bevezetett expressVI-t, amely a fejlesztő dolgát könnyíti meg azzal, hogy nemcsak huzalozással, hanem interaktív dialógusablakkal is konfigurálható.

A csomópontok harmadik csoportját a struktúrák alkotják. Ezek a G nyelv vezérlőszerkezetei. A blokk diagramon úgy jelennek meg, mint egy keret. Egy struktúra által körülhatárolt diagram részt szubdiagramnak nevezünk. A struktúrák termináljait két nagy csoportra oszthatjuk. Az egyik csoportba olyan speciális terminálok tartoznak, amelyek egy adott struktúra működését befolyásolják, illetve arról valamilyen információt szolgáltatnak. A másik csoportba a bemeneti és a kimeneti terminálok tartoznak, melyeket *tunnel*-nek nevezünk.

A vezérlőszerkezetek első csoportját a szekvenciát megvalósító struktúrák alkotják. Közös jellemzőjük, hogy egy vagy több egymást követő keretből állnak, és az ezekben lévő szubdiagramok szekvenciálisan hajtódnak végre. Két fajtája van: a *Stacked Sequence Structure* és a *Flat Sequence Structure*. Az előbbiben a keretek – mint a kártyapakliban a lapok – egymást fedve helyezkednek el, míg az utóbbiban egymás mellett – mint a filmszalagon a képkockák – sorakoznak. Működésük: ha a vezérlés az adott szekvenciára kerül, akkor először az első keretben lévő szubdiagram hajtódik végre, majd a másodikban lévő és így tovább. A szekvencia befejezi működését, ha az utolsó keretében lévő szubdiagram végrehajtása is befejeződött. Egy *Flat Sequence Structure* szomszédos keretei közötti adatok átadása *tunnel*-ek segítségével történik. A *Stacked Sequence Structure* szerkezetéből adódóan itt az előbb említett módszer nem használható. Helyette az ú.n. *sequence local* terminál használatos.

A második, szelekciót megvalósító csoportba egyetlen struktúra tartozik. Neve: *Case Structure*. Szintén egy vagy több keretből áll, és az elhelyezkedésük a *Stacked Sequence Structure*-hoz hasonlatos. Működése: a szelektor terminálon megjelenő értéktől függően hajtódik végre valamelyik szubdiagram.

A harmadik csoportot az iterációt megvalósító struktúrák alkotják. Ezek egyetlen keretből állnak, melyben elhelyezkedő szubdiagram a ciklusmag. Előírt lépésszámú ciklust a *For Loop*-al hozhatunk létre. A számláló terminájára a kívánt lépésszámot kell huzaloznunk, az iterációs terminálja pedig ciklusszámlálóként szolgál. A lépésszámot azonban úgy is megadhatjuk, hogy egy tömb típusú bemenetre engedélyezzük az indexelést (*Enable Indexing*). Ekkor a tömbből lejön egy dimenzió, vagyis 1D tömbből skalár érték lesz (a tömb i. eleme), 2D tömbből pedig 1D tömb (a tömb i. sora). Az előbbi esetben minden elemre, az utóbbiban pedig minden sorra lefut a ciklus. Ugyanezt el lehet játszani a kimenettel is, ekkor a kimenet plusz egy dimenziós tömbként kerül ki. Azonban megtehetjük azt is, hogy egy adott *For Loop*-nak több különböző méretű tömböt adunk bemenetként, engedélyezve az indexelést, ráadásul a számláló terminált is inicializálhatjuk. Ekkor a ciklus lépésszámát a tömbméretek illetve a számláló terminál értéke közül a legkisebb határozza meg.

A *While Loop*-nak szintén két terminálja van. Az egyik a már ismertetett iterációs terminál, a másik a feltételes terminál, amely egy logikai értéket kap, és ez alapján állítja le a ciklust. Beállíthatjuk, hogy a ciklust igaz vagy hamis érték esetén állítsa-e le. Ezen ciklus be-, és kimenetein szintén engedélyezhetjük az indexelést, azonban ez nem határozza meg a lépésszámot.[2] [3]

A 7.1-es verzió újdonsága a *Timed Loop*, amely az egyes iterációs lépéseket meghatározott időközönként hajtja végre.

A ciklusok alkalmazásakor gyakran szükség van olyan tárolóra, amelyben a ciklus minden egyes végrehajtásakor ki tudjuk olvasni annak az előző végrehajtáskori értékét, és a módosított értékét bele tudjuk írni. Erre két eszköz áll rendelkezésünkre: a shift regiszter és a visszacsatoló csomópont. A két eszköz működése egymással teljesen kompatibilis, és egymásra tetszőlegesen felcserélhetőek.

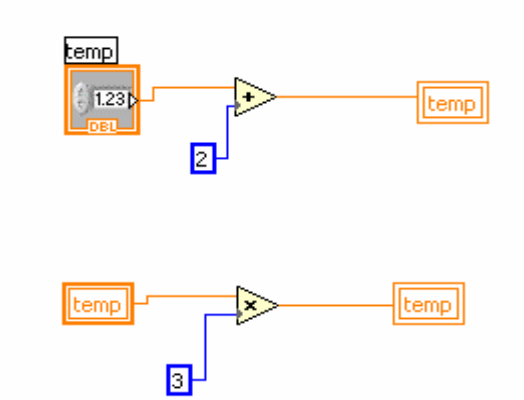
A visszacsatoló csomópontnál érdekességként megemlítem, hogy alapesetben a bemeneti terminál a jobb, míg a kimeneti a bal oldalán található (tehát a „balról-jobbra szabály” itt sem érvényesül). Azonban lehetőség van a terminálok felcserélésére. Ezt úgy érhetjük el, hogy fordítva kötjük be az input és output vezetéket, majd az egérrel egy kicsit elmozgatjuk a csomópontot.

A 7.0-es verzióban a korábbi verziók „fapados” eseménykezelési lehetőségeit kibővítették egy hatékony eseménykezelő eszközrendszerrel, amelynek legfontosabb darabja az *Event Structure*. Működése: ha bekövetkezik egy esemény, és az *Event Structure*-ben van hozzá kapcsolódó keret, akkor az abban lévő szubdiagram végrehajtódik.

A LabVIEW-ban a változó fogalma eltér a hagyományos nyelvekben megszokottól. Megkülönböztetünk lokális és globális változót.

A lokális változó mindig valamely előlapi kontrolhoz vagy indikátorhoz (a továbbiakban összefoglaló néven előlapi komponens) kapcsolódik, segítségével a hozzá kapcsolt komponens értékét tudjuk írni vagy olvasni. Ha egy lokális változó nincs előlapi komponenshez kapcsolva, akkor ez szintaktikai hibát eredményez. Továbbá az is szintaktikai hibához vezet, ha nem használjuk fel a változót a blokk diagramon, azaz nem írjuk vagy olvassuk.

A lokális változó gyakori használata nem javasolt, túlzott alkalmazása nehezen átlátható kódot és olykor indeterminisztikus működést eredményezhet. Az utóbbi szemléltetésére álljon itt egy példa. Tegyük fel, hogy van egy előlapi komponensünk, és azt lokális változók segítségével írjuk és olvassuk. Ha nem gondoskodunk a műveleti sorrend betartatásáról – a G nyelv jellegzetességéből adódóan – a végrehajtási sorrend véletlenszerű lesz.



2. ábra
A lokális változók használatának veszélyei

A globális változó tulajdonképpen egy blokk diagram nélküli előlap, és különböző VI-ok közötti kommunikációra használatos. A fent említett okok miatt túlzott használata szintén nem javasolt. További hátránya, hogy alkalmazása monolitikus programot eredményez, amely jelentősen rontja a VI karbantarthatóságát és a részeinek újrafelhasználhatóságát.[4]

5. Futtatás és hibakeresés

Az elkészült programokat az eszköztár futtatás gombjával tudjuk elindítani. Ha a programunk nem tartalmaz szintaktikai hibát, akkor ezen a gombon egy fehér nyíl látható, amely futás közben szaggatott fekete nyílra vált. Ellenkező esetben egy töredezett nyíl ábrája lesz látható. Ha ekkor a gombra kattintunk, megjelenik egy ablak a hibák listájával.

Teszteléskor jó szolgálatot tehet még az állandó futtatás gomb, melynek hatására a VI addig fut, amíg a végrehajtás leállítása gombbal le nem állítjuk, vagy a leállítás/folytatás gombbal átmenetileg felfüggesztjük a program végrehajtását. Ez a funkció főleg akkor hasznos, ha egy VI futását többször egymás után, különböző inputokra akarjuk megfigyelni.

Az egyik leghatékonyabb tesztelési módszer a végrehajtás nyomkövetése. Ha aktiváljuk, akkor a vezetékeken kis „buborékok” fogják jelezni az adatáramlást, és a csomópontok által kiadott értékeket is megjeleníti.

Mint a komolyabb fejlesztői környezetekben, itt is lehetőség van a program lépésenkénti végrehajtására, amit három gombbal szabályozhatunk.

Mindezen felül a vezetékeken elhelyezhetünk próba- és megszakítási pontokat is, melyek szintén nagymértékben megkönnyítik a hibakeresést.

Végül, de nem utolsó sorban itt kell megemlítenem egy fontos adattípust az error clustert, amely kifejezetten futás idejű hibák lekezelésére alkalmas.

6. Súgó és felhasználói támogatás

A LabVIEW-ban a felhasználót számos eszköz segíti a program megismerése során. Több tucat példaalkalmazás között szemezgethetünk, és a súgó is elég részletes leírást nyújt. Ha ez nem lenne elég, akkor rendelkezésünkre áll egy komplett kézikönyv is pdf formátumban.

Külön ki kell emelnem a környezetfüggő súgót. Ha aktiváljuk, és az egérkurzort valamely blokk diagram objektum fölé visszük, akkor egy kis ablakban megjelenik annak rövid leírása egy hivatkozással a súgóban lévő részletesebb ismertetőre.

A programmal kapcsolatos legfrissebb információkat a National Instruments honlapján találjuk.

7. Végszó

Végezetül álljon itt néhány személyes gondolat a LabVIEW-ról. A hagyományos nyelvekhez szokott programozó számára eleinte szokatlan ez a grafikus adatfolyamnyelv. Mindenképpen egyfajta szemléletváltást igényel a nyelv megismerése. Azonban a befektetett erőfeszítés hamar megtérül. A kezdeti buktatók után szinte élvezetessé válik a fejlesztés.

A LabVIEW-val való munka során legjobban a program nyomkövetési lehetőségeivel és a környezetfüggő súgóval voltam megelégedve.

Az elektronikus dokumentációkon erősen érződik, hogy a program főleg mérnököknek készült. A nyelv programozói szemszögből történő megközelítése ezen dokumentációkban sajnos eléggé szűkre szabott.

8. Irodalomjegyzék

[1] www.ni.com

[2] LabVIEW User Manual

[3] National Instruments LabVIEW Introduction Course

[4] Jorgen Jehander: Graphical Object-Oriented Programming in LabVIEW