

LabVIEW I. mérés

Mérést végezte: Paulicsek Ádám Imre

Mérés dátuma: 2021.02.25. 11:15-14:00

Mérés helye: Budapest

paulicsek.adam.imre@hallgato.ppke.hu

Mérőeszköz adatai: LabVIEW software

I. LABVIEW

A LabView egy grafikus programfejlesztő, amely elsősorban méréstechnikai és a hozzákapcsolódó jelfeldolgozási feladatok megoldására szolgál, de alkalmas más, pl. szimulációs munkákra is. A grafikus programozás egy látványos, könnyen követhető programozási módot jelent, amelyet könnyen sajátíthatnak el a hagyományos programnyelveket nem ismerő szakemberek is. Mindazonáltal meg kell jegyezni, hogy a programozás alapvető jellemzői (változók deklarálása, ciklusok) teljes mértékben a C programnyelvhez hasonlóan, illetve azzal analóg módon történnek, ezért bármilyen hagyományos programnyelv alapszintű ismerete sokat segít a kezdeti lépésekben.

II. MÉRÉSI FELADATOK

A. Első feladat

(Az előlapon elhelyezett tetszőleges típusú, nyomógombot bekapcsolva gyulladjon meg egy négyzet alakú sárga LED. A m/s mértékben beadott sebességet írja ki és mutassa meg egy tetszőleges formájú kijelző km/h egységben.)

A Front Panel megnyitása után, választottam egy nyomógombot (push button) a controls palette-ből, majd ezután egy LED-et (square LED). A LED-nek a színét, a tulajdonságainál (properties) változtattam meg egy sárga árnyalatú színre. Ennek befejeztével, átváltottam a Block Diagramhoz, ahol összekapcsoltam a két azonos boolean típusú rendeltetű kapcsolót, és LED-et. Az izzó, akkor villan fel, amikor a bekapcsológombra rákattintok, ezzel a kapcsoló értéke megváltozik false-ról, és true értékkel fog rendelkezni, amit a vezetékek tárol el, és adja tovább az izzónak. Mivel igaz értéket kap az izzó, ezért felvillan, és addig fog világítani, amíg a nyomógomb értékét meg nem változtatjuk hamisra.

A feladat másik részében, hogy egy m/s értéket kell átváltani km/h. Ehhez használnunk kell a két érték között egy matematikai összefüggést, hogy $1 \text{ méter per szekundum} = 3,6 \text{ kilométer per órával}$. A Front Panel-en kiválasztottam egy numeric control-t, és egy numeric indicatort a controls palette-ből. Következőben a Block Diagramban, a function palette segítségével hozzáadtam egy numeric control-lal azonos numeric constant-ot, aminek az értékét 3,6-ra állítottam, a matematikai összefüggés felhasználásával. Ezek után egy szorzás függvényt adtam hozzá, aminek a két bemenete a constant, és a control, a kimenete pedig az indikátor.

Ez a két része a feladatnak, teljesen egymástól függetlenül működnek, nem befolyásolják egymás működését, mivel nincsenek összekapcsolva. Nem változik meg az átváltás kimenő értéke azzal, hogy fel van-e kapcsolva az izzó, vagy nincsen. És ugyanez fordítva.

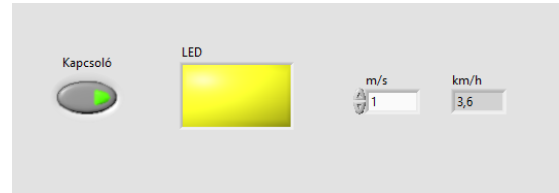


Fig. 1. Első feladat (Front Panel)

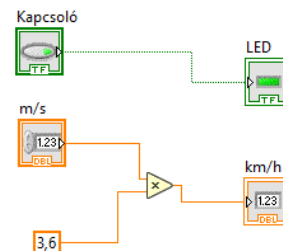


Fig. 2. Első feladat (Block Diagram)

B. Második feladat

(Alakítsa át és mentse el új néven az 1. pont feladatában a nyomógombot kapcsolóra, majd módosítsa a programot olyanra, hogy csak akkor történjen mértékegység átszámítás, ha a kapcsoló ki van kapcsolva. Tehát kikapcsolat kapcsoló esetén az eredményt km/h-ban bekapcsolt kapcsoló esetén m/s-ban kell kiírni. A mértékegység átszámítás be/kikapcsolt helyzetéről jelenjem meg információt az előlapon.)

Az előző feladatot fogom tovább módosítani, és alakítani a feladat leírása szerint. Először a nyomógombra jobb egérgépet után, a replaceban kiválasztottam egy kapcsolót (slide switch). A következő lépésben arra kell megoldást találni, hogy hogyan tudnám, csak akkor átalakítani a mértékegységet, amikor a kapcsoló ki van kapcsolva. Abban tér el az előző feladattól, hogy itt összekötjük ezt a két különböző eseményt. Erre szerintem a legjobb módszer, ha egy case structure-t alkalmazok. Ezt a C++ programozásban elágazásnak nevezzük, vagyis van egy igaz, és van egy hamis ága. A struktúra két bemenettel fog rendelkezni, de hogy melyik ága fog "lefutni", a kapcsoló értéke fogja eldönteni. Mindkét esetben összekötöm a kapcsolót az izzóval, hogy kapjak valami információt arról, hogy mikor milyen állapottal rendelkezik a kapcsoló. A másik numeric control bemenetelt, pedig már másképpen fogom kezelni a két különböző esetben. Az első esetben, amikor hamis, akkor a bemenetet megszorozom egy 3,6 numeric constant-al, és az eredményt összekötöm a kimenettel. A másik esetben, amikor igaz, akkor csak simán összekötöm a bemenetet, a kimenettel.

Ekkor, ha kikapcsolt állapotban van, akkor a m/s bemenetet, átváltja km/h-ba, viszont ha bekapcsolt állapotban van, akkor a m/s bemenet, egyben a kimenet is lesz.

Emellett a feladat kéri, hogy jelenjen meg valami információ az átváltással kapcsolatban, ezért mindkét esetben létrehoztam egy string constant-ot, amit összekötöttem egy string indicator-ral. Ez jelzi, hogy éppen a kimeneti érték, pontosan milyen mértékegységgel rendelkezik.

A mértékegység átváltást egy subVI-ben kell elhelyezni. A subVI egy program csomag, vagy modulnak is szoktuk nevezni. Ennek azzal a jelentőséggel bír, hogy az egész modul egyetlen egy ikonból fog állni, és nem fogom látni a szerkezetét. Vagyis egyszerűen, amit szeretnék egy modulba tenni, csak kijelölöm egy négyzettel, aztán az edit fülben kiválasztom, hogy csináljon belőle egy subVI.

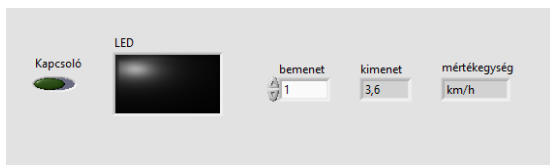


Fig. 3. Második feladat (Front Panel)

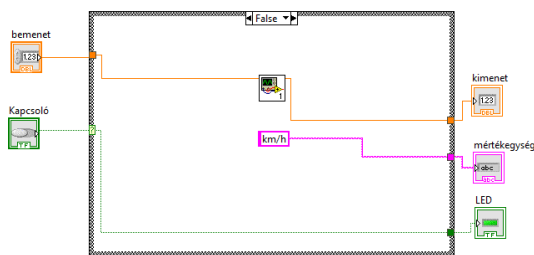


Fig. 4. Második feladat (Block Diagram)

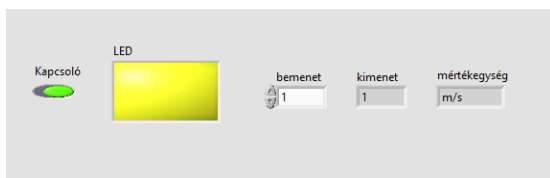


Fig. 5. Második feladat (Front Panel)

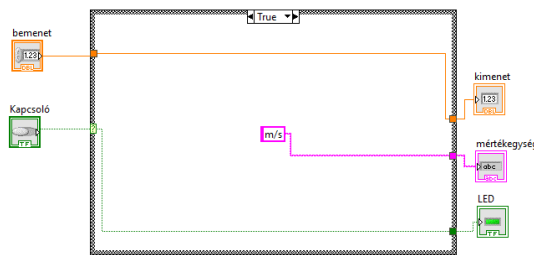


Fig. 6. Második feladat (Block Diagram)

C. Harmadik feladat

(Készítsen egy kockajáték szimulációt mely egy nyomógombot segítségével hozható működésbe. A

nyomógomb megnyomására egyszer kell három független kockával dobni és az eredményeket kijelezni. Ha az eredmények összege 10 akkor gyulladjon ki egy kör alakú zöld LED.)

Először a Front Panel-en választok egy nyomógombot (OK button), ebben az esetben egy olyat választottam, amelyre tudok szöveget írni a könnyebb kezelhetőség érdekében. Emellett még egy numeric indicator-t is hozzáadtam, hogy lássam, hogy a dobásoknak az összege pontosan mennyi is lesz. Ez a megjelenítés, nem csak esztétikai, hanem ellenőrzés szempontjából is nagy jelentősége lesz, hogy jól működik-e a mérőeszközöm. Mint az előző feladatban, itt is egy case structure-ot fogok létrehozni a Blokk Diagram-ban. Ebben az esetben nem a hamis ág, hanem az igaz ág fog nagyobb szerepet betölteni. Ezekután a dobókockát kezdtem el elkészíteni. Hozzáadtam egy véletlenszám generátort, de ez csak [0,1) intervallum közötti számokat generál. Erre a problémára, olyan megoldást találtam ki, hogy az eredmény megszorozom hattal, de ez még nem teljesen oldotta meg a problémámat, mert nem egész számok jöttek létre így. Ezt a problémát úgy orvosoltam, hogy a kapott eredmény lefelé kerekítettem. Ezzel egy 0 és 5 közötti egész számokat tudok generálni. Ehhez, hogy ne legyen hamis a dobókockám, még hozzáadok egyet, és így elkészült az 1 és 6 közötti egész számaimat generáló kocka. Minden egyes lehetőség, ugyanakkora valószínűsége van, mivel a véletlenszám generátor, a 0 és az 1 közötti számokat, ugyanakkora eséllyel generálja, és ezt egy konstans tag beszorzásával, és hozzáadásával nem változik meg a valószínűsége. Ezek után ezt a methodot egy subVI-ba fogtam össze, és az elkészült subVI másoltam le még kétszer, amivel létrejött három különálló dobókocka. Ezek után az eredményeket összeadtam, az összeadás függvény segítségével, ami csak két bemenete lehet, ezért kétszer kellett alkalmaznom. A végeredményt összekötöttem az elején létrehozott numeric indicator-ral, ami kiírja a három dobás összegét.

A feladat második része az, hogy az eredményt vizsgáljuk meg, és ha az pontosan 10, akkor gyulladjon ki egy kör alakú zöld LED. Az előzőleg elkészült case structure-ot fogom tovább módosítani úgy, hogy létrehozok a casen belül egy numeric constants-ot, aminek az értéke 10 lesz. Emellé létrehozok egy egyenlőség függvényt, aminek a két bemenetéről eldönti, hogy egyenlők-e az értékek. Ezt összekötöm a konstanssal és az eredménnyel, majd ennek a kimenetére hozzákötöm a LED-et, amit előbb hozzáadok a Front Panel-ben. A case structure másik elágazását nem fogom használni, mert ha nem dobok a kockákkal, akkor nem történik semmi. A program lefuttatása során észrevettem, hogy nem egyszer, fut le, hanem végtelenszer. Aztán ezt a problémát úgy küszöböltem ki, hogy a gombnak a mechanikáját átállítottam, mert előtte olyan mechanikára volt rakva, hogy a gomb megnyomása után, a gomb addig fog igaz értéket adni, amíg nem nyomjuk meg még egyszer. Ezt állítottam át arra a mechanikára, hogy a gomb megnyomása után le is kapcsolja saját magát.



Fig. 7. Harmadik feladat (Front Panel)

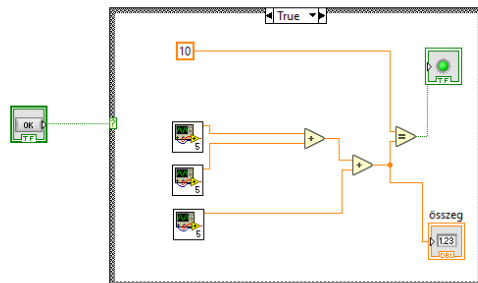


Fig. 8. Harmadik feladat (Block Diagram)

D. Negyedik feladat

(A 3. pontban elkészített dobókocka szimulációt subVI-ként felhasználva készítsen programot mely legalább 10000-szer dob a három kockával és megjeleníti az eredmények gyakoriságát, azaz, azt, hogy hány alkalommal lett az eredmény 3, 4, ... 18.)

Az előző feladatban elkészült dobókocka szimulációt fogom felhasználni ebben a feladatban subVI-ként, ami egy-egy kocka dobását fogja szimulálni. Ebben az esetben egy meghatározott számszor kell megismételni a szimulációt, és ennek az adatait utána eltárolni, hogy később fel tudjam majd használni. Egy ismert alkalomsszor kell ismételni valamit, arra a legjobb megoldás, ha a For Loop-ot használom. Ezt másképpen nevezhetjük egy for ciklusnak is, és ami ennek a belsejében van, az annyiszor fog lefutni, ahányszor megadjuk hogy lefusson, ami ebben az esetben 50000-szer kell. Ennek a belsejébe fogom a dobókocka szimulációkat tenni, amiket összeadok két darab összeadó függvényrel, aztán ezeket eltárolom. Ekkor ütköztem egy problémába, hogy hogyan lehetne a legjobban ezeket az adatokat eltárolni. Végül az internet segítségével megtaláltam, hogy van hisztogram beépített modul, ezért nagyon könnyű dolgom volt, mert nem nekem kellett elkészíteni ezt a modult. Ennek a hisztogramnak két bemenete van, az egyik a dobások összege, a másik pedig, hogy mennyi legyen a hisztogramnak a beosztása. Ezt egy numeric constant segítségével adtam meg, aminek az értékét 16 állítottam be, mert a legmagasabb összege a három kockának 18 lehet, és a legkisebb pedig 3, és egymásból kivonva őket, megkapjuk a szükséges beosztást. Végül ennek a hisztogramnak, a gráf kimenetére kattintottam jobb egérklickkel, és hozzáadtam egy hisztogram diagrammot.

Lefutottva a programot, láthatjuk az elkészült ábrán, hogy a három dobókocka összege, milyen gyakorisággal dobtuk. Ezeknél a diagramoknál azt vehetjük észre, hogy úgy néz ki, mint egy piramis. És egyben azt jelenti, hogy a szimuláció, tényleg sikeres volt, mert az értékek egyenletesen oszlanak el a tartományon.

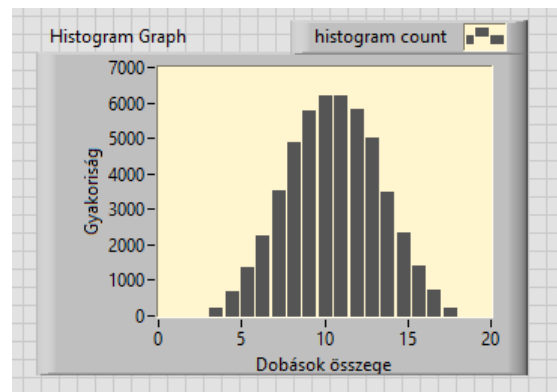


Fig. 9. Negyedik feladat (Front Panel)

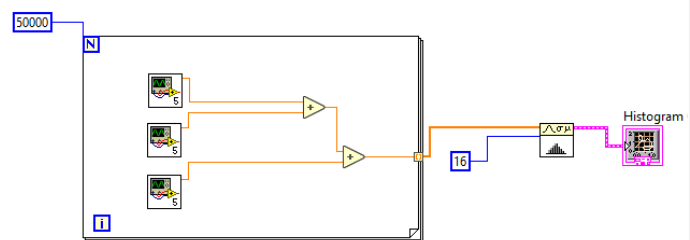


Fig. 10. Negyedik feladat (Block Diagram)

E. Ötödik feladat

(A 4.pont szerinti feladatot végezze el úgy, hogy a dobókocka 1..6 értéke helyett folytonos 0..1 intervallumot használ és a gyakoriságot legalább 20 független érték összegéből képezi.)

A negyedik feladatot alakítottam tovább, a feladat leírása szerint. Az előző feladattól annyiban különbözik, hogy ebben nem változtattam meg a véletlenszám generátor által kapott értékeket, hanem azzal dolgozok tovább. És így kapok 0 és 1 közötti értékeket. A szemléletesség érdekében, csak 3 darab szám összegének a gyakoriságát fogom elvégezni, és az eredményt huszas felbontásba jeleníttem meg. Ehhez a megoldáshoz szükségem lesz két For Loop-ra, vagy két darab for ciklusra, amiben a belső ciklus fogja a 3 dobást generálni, és a külső ciklus fogja a kapott értékeket összeadni. Ezt a műveletet a ciklus 50000-szer fogja elvégezni.

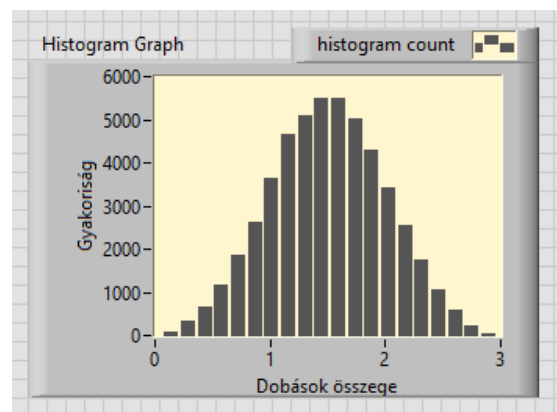


Fig. 11. Ötödik feladat (Front Panel)

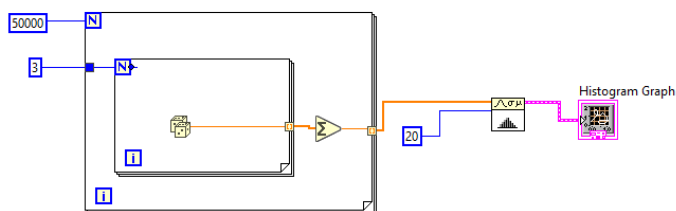


Fig. 12. Ötödik feladat (Block Diagram)

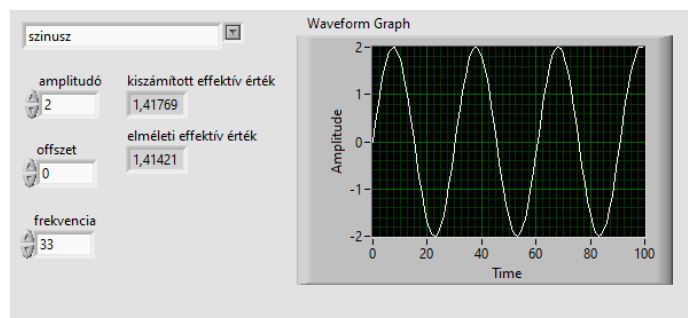


Fig. 13. Hatodik feladat (Front Panel)

F. Hatodik feladat

(Szimuláció segítségével választható módon állítson elő szinuszt, négyszög, fűrészt és háromszög alakú jeleket. Az előállított jelek paraméterei (amplitudó, offset, frekvencia) legyenek beállíthatók. Az előállított jeleket értelmezze úgy mintha egy feszültségforrás jele lenne. A számítás során a jelalakot mintánként egy adott tömbben kell elhelyezni, és annak értékeit kell ábrázolni. A megjelenítés során ügyeljen arra, hogy futtatás közben sem ugrálhat össze vissza az ábra)

Először a Front Panel-ben létrehozok egy combo box-ot, hogy egy legördülő listából kitudjam majd választani, az éppen általam akart szimuláció elkészítéséhez szükséges típusjelet. Ebbe felvittem egyesével a 4 lehetőséget. Aztán, ahogy a feladat is kérte, három numeric control-t hoztam létre, egy amplitúdónak, egy offsetnek, és egy frekvenciának. És mindezek mellé, még egy hullám gráfot is hozzá adtam. Ezekután, a Block Diagram-ban létrehoztam négy különböző ággal rendelkező case structure elágazást. Aztán abban a problémába ütköztem, hogy nem futott le a szimuláció, ahogy én akartam, és később kiderült, hogy volt egy karakter eltérés a combo box, és az elágazásnak a nevében, és ezért nem futott le. Utána létrehoztam mindegyikben egy jelszimulációt, aminek beállítottam a megfelelő alakú jelet, és hozzákötöttem a numeric bemeneteket, a megfelelő helyekre. A szimulációból kikerkező dinamikus adatokat, átkonvertáltam double tömbbé, a Convert From Dynamic Data beépített modul segítségével. Ennek a kimenetét összekötöttem a hullám gráffal, amiben az adatok nem mozognak az idő változásával, ezért nem fog ugrálgatni az ábrán, a szimulációnak a jele.

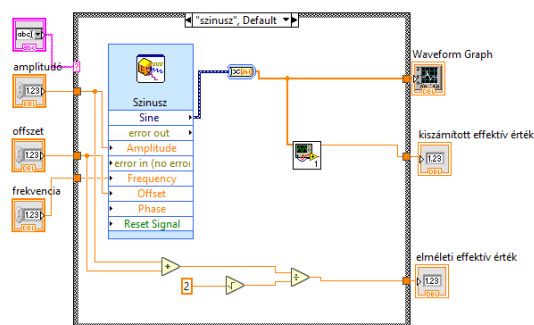


Fig. 14. Hatodik feladat (Block Diagram)

REFERENCES

- [1] Labview,
http://www.electro.uni-miskolc.hu/~elkszabo/Oktatas/LabView_bevezeto.pdf

Emellett kérte a feladat, hogy számítsuk ki a feszültségforrásnak az elméleti, és a számított effektív értékét. Mivel feszültségforrásról beszélünk, ezért a legnagyobb kitérés lesz ebben az esetben, a maximális feszültség. Az elméleti effektív értéke különbözik a jelek között. A szinuszt feszültségnek a maximumát, el kell osztani $\sqrt{2}$ -vel, a háromszögének és a fűrésznek $\sqrt{3}$ -mal, viszont a négyszög esetében megegyezik. A számított effektív értékét úgy kapjuk meg, hogy a tömbbeli adatokat négyzetre emeljük, utána a kapott értékeket összeadom, és elosztom az adatok darabszámával. Az így kapott eredményt gyök alá vonjuk, és megkapjuk az effektív értékét. A két eredmény nem egyezik meg, és nem is fog sosem egyenlő lenni, mivel a számított értéknek a határértéke lesz, az elméleti érték.