# LabVIEW használata

Heiszman Henrik
Neptun kód: ENV2R9
Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
1083 Budapest, Práter utca 50/A
heiszman.henrik@hallgato.ppke.hu

***Téma*–A LabVIEW grafikus programozói környezet segítségével különböző matematikai és programozói feladatok illetve problémák megoldása olyan programokkal, melyeket a felhasználó egyszerűen, számára is érthető módon tud használni. Mérés során a National Instruments által fejlesztett LabVIEW programot használtam.**

## I. A JEGYZŐKÖNYVBEN HASZNÁLT FOGALMAK

Front Panel: ez az általunk elkészített, virtuális műszer kezelőfelülete, ide helyezhetők el a különböző vezérlő és kijelző egységek, valamint különféle design elemek.

Block Diagram: itt az alkalmazás grafikus programozása folyik, ez a virtuális műszer belseje.

Controls Palette: itt találhatóak a Front Panel-en elhelyezhető vezérlő és kijelző elemeket (illetve szokás még kontrolloknak és indikátoroknak nevezni őket).

## II. A LABVIEW ELŐKÉSZÍTÉSE

A LabVIEW ikonra kattintva elindítottam a programot, majd létrehoztam egy új „Blank VI” projectet. A tanultak alapján a „ctrl+T” billentyűkombinációval egymás mellé illesztettem a képernyőn a Front Panelt és a Block Diagramot. Ezzel a pár lépéssel elő is állítottam a két tiszta oldallal rendelkező munkafelületet.

## III. SEBESSÉG ÁTVÁLTÁSA

Az általam elkészített programnak feladata lesz, hogy hiba nélkül, a felhasználó által meghatározott, $\frac{m}{s}$ mértékegységű sebességet átváltsa $\frac{km}{h}$ -ba, majd az így kiszámolt értéket jelenítse meg egy általam tetszőlegesen megválasztott kijelzőn. Valamint a program képes lesz egy, a felhasználó által üzemeltetett gomb hatására egy sárga színű, négyzet alakú LED-et felkapcsolni.

A program elkészítése során több fajta inputra (a felhasználó által kezelt bemenetre) is szükségem lesz. Kelleni fog egy nyomógomb, amely majd üzemelteti a LED, továbbá egy numerikus bemenet, amely segítségével az üzemeltető megadhatja az átváltani kívánt sebesség mértékét ($\frac{m}{s}$-ban).

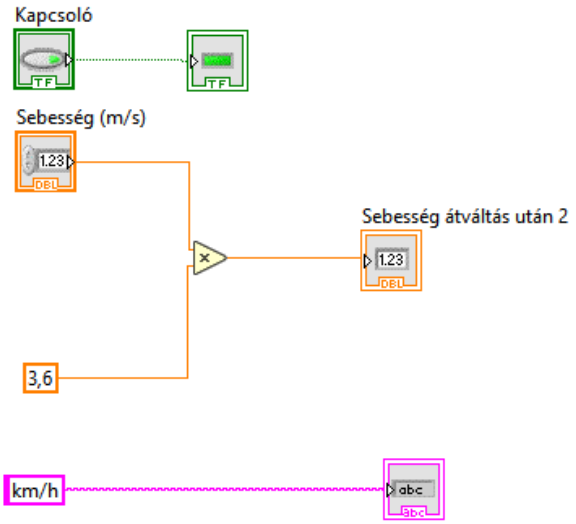
Ezeket a bemeneteket egyszerűen elhelyezem a Front Panel. A használni kívánt inputot a következőképpen hoztam létre: az egér jobb gombjával kattintottam a Front Panelen, amelynek köszönhetően megjelent Controls Palette. Ebből az ablakból kiválasztottam az általam használni kívánt inputokat: egy Boolean típusú nyomógombot és egy numerikus bemenetet. (Boolean típusú gombnak nevezzük azt a bemenet, amely igaz vagy hamis érték bevitelére szolgál.) Ezek után szinten a Controls Paletten „Boolean” menüpont alatt kiválasztottam a négyszögletes LED-et, majd elhelyeztem a Front Panelen. Elvárás, hogy az a dióda négyzet alakú legyen és sárgán világítson. A méretét egyszerűen a LED egyik szélére kattintva, húzással méretre lehet szabni. A dióda színének beállításához a jobb kattintással lenyíló menüsor „Properties” alpontját kell megnyitni. Itt az „Appearance” fülön a „Colors” alpontban beállítható az On/Off állásban lévő LED színe. Én az On állapot színét a feladatnak megfelelően a 255; 255; 100 és az Off állapot színét pedig a 100; 100; 0; RGB kódú sárga árnyalatra állítottam. Végül elhelyeztem egy numerikus és egy string típusú kimenetet is, amelyen keresztül a program képes lesz megjeleníteni az átváltott sebesség mértékét és mértékegységét a felhasználó számára.

Az vizuális kellékek elkészítése után elkezdtem a feladat software-es részét elkészíteni Block Diagramon.

Első lépésben a nyomógomb kimenetét összekötöttem a LED bemenetével, ezzel biztosítva, hogy a dióda kapcsolható legyen a felhasználó igénye szerint.

Második lépésben a mértékegységváltó helyes működését szolgáló programot készítettem el. A Block Diagramon létrehoztam egy numerikus konstanst, amelynek a 3,6-et adtam értékül, majd elhelyeztem a szorzás elvégzésére szolgáló függvényt. Ennek a függvénynek a bemenetéra rákötöttem a numerikus bementet és a konstans számot, majd a kimenetét pedig összekötöttem a numerikus kimenettel. Ezek után a Block Diagramon létrehoztam egy string típusú konstanst is, amelybe „$\frac{km}{h}$”-ot írtam. Ez az állandó felelős azért, hogy az üzemeltető lássa az átváltott szám mértékegységét. Végül összekötöttem a string állandót a string kimenettel.

E két lépés elvégzésével a program elkészült és az ellenőrző futtatás során hibátlannak bizonyult: a nyomógomb kapcsolja a lámpát és a bemenetre érkező az adatott átváltás során helyesen meg is jeleníti. (*1. ábra*)


*1. ábra*
Elkészült átváltó

Meg kell még említeni a program matematikáját is. Fizikából tanultak alapján tudjuk, hogy $1\frac{m}{s} = 3{,}6\frac{km}{h}$ . Ebből adódóan volt szükségem egy konstans számmal (3,6) való szorzásra.

Mivel a program készítése során egymástól független működésre kalibráltam az mértékegységváltót és a LED kapcsolóját, így az applikáció használata során ez a két funkció egymást semmiben sem befolyásolja.

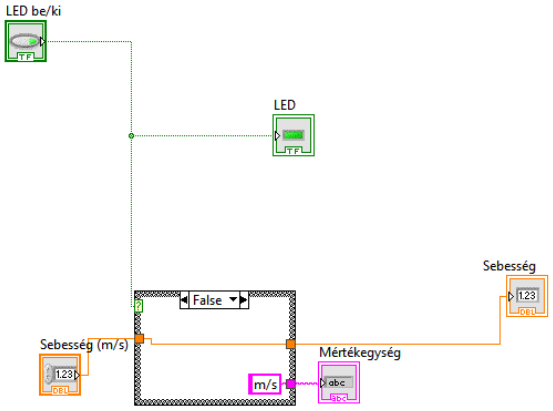
## IV. VÁLASZTHATÓ ÁTVÁLTÁS

Ebben a részben átalakítottam a már meglévő (1. feladatban létrehozott) programomat úgy, hogy a felhasználó saját maga tudja meghatározni, hogy át szeretné-e váltani a megadott sebességet vagy sem.

Annak érdekében, hogy a program el tudja végezni a feladatát, két esetet kell elkülöníteni egymástól, és mind a két szcenáriót egymástól különböző módon kell kezelni. Az egyik eset az, ha a felhasználó nem szeretne átváltást, tehát a kapcsoló az „OFF” állásban van. Ekkor a programtól azt várjuk el, hogy egyszerűen kiírja a bemeneti értéket és a mértékegység kijelzőn a „$\frac{m}{s}$” jelenjen meg. A másik esetben a felhasználó bekapcsolja az átváltó funkciót, tehát a kapcsoló az „ON” állásban van. Ekkor azt várom el a programtól, hogy helyesen átváltsa a bemeneti adatot és a mértékegység kijelzőn a „$\frac{km}{h}$” szerepeljen.

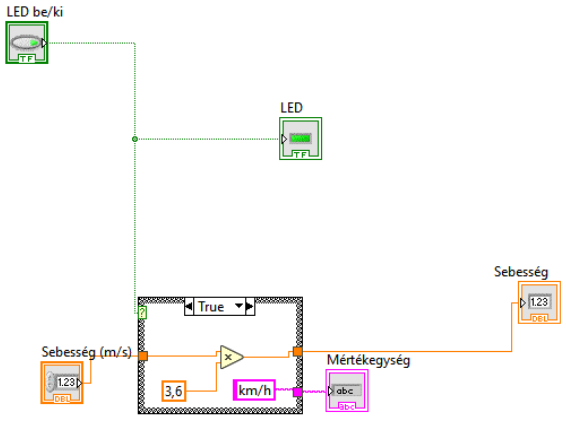
Első lépésben a Block Diagramon létre kell hozni egy „Case Structure ” nevű függvényt, amely arra szolgál, hogy különböző eseteket lehessen megkülönbözetni (jelen esetben ez az „OFF” / „ON” (igaz/hamis) állása a kapcsolónak.), és össze kell kötni a kapcsoló kimenetével.

Második lépésben meg kell tervezni azt az esetet, amely során a kapcsoló a hamis, tehát kikapcsolt állásban van. Ekkor egyszerűen a „Case Structure” False állásánál át kell kötnünk a sebesség bemenetet a kimenettel és a string konstanst $\frac{m}{s}$ állapotban be kell kötni a string kimenetre. Ezt a következő ábrán szemléltetem. (*2. ábra*)



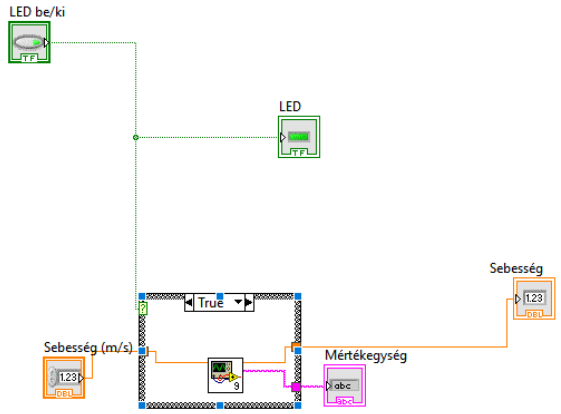

*2. ábra*
False ág

Harmadik lépésben megterveztem azt az esetet, amikor a kapcsoló az igaz, tehát a bekapcsolt állapotban van. Ekkor a „Case Structure” True állásában be kell húzni az első feladat során elkészített átváltó program szorzó és mértékegységet kiíró részét. Ezt szemlélteti a következő ábra. (3. *ábra*)




*3. ábra*
True ág

Feladat része volt továbbá az, hogy használjak úgynevezett SubVI-t a program elkészítése során. A SubVI arra szolgál, hogy a programon belül egy-egy részletet össze lehet sűríteni, így az elkészült program jobban átláthatóvá és könnyebben kezelhetővé válik. SubVI létrehozásához ki kell jelölni a tömörítendő elemeket a Block Diagramon és az „Edit” menüpont alatt a „Create SubVI” alpontot kell kiválasztani. A következő ábrán látható, hogy hogyan jelzi a LabVIEW az elkészült SubVI-t. (*4. ábra*)




*4. ábra*
SubVI használata

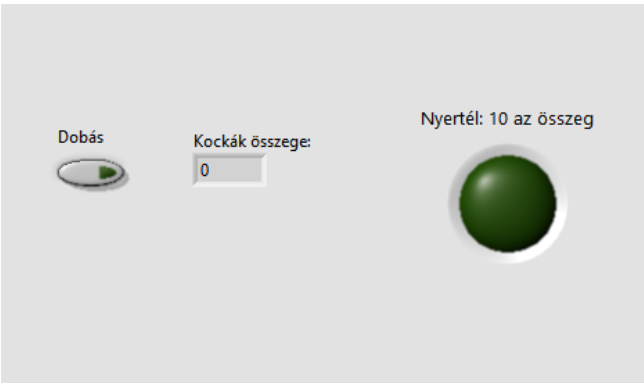
A programban továbbra is él az a funkció, hogy a kapcsoló segítségével vezérelni lehet a LED-et. Itt ez hasznos volt mert, így tudja a felhasználó a dióda állapota alapján, hogy be van- e kapcsolva az átváltás.

Az előbb ismertetett lépések elvégzése során a program elkészült. Az ellenőrző futtatás során hibátlannak bizonyult.

## V. KOCKAJÁTÉK

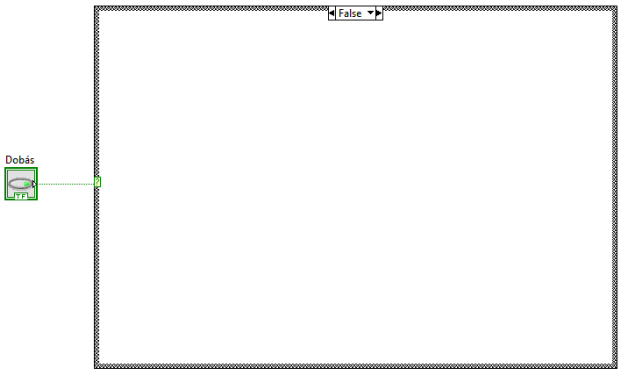
Ez a program egy kockajátékot fog szimulálni, amely egy nyomógomb segítségével üzemeltethető. A gomb megnyomása után a program egyszerre fog „dobni" három hatoldalú dobókockával és abban az esetbe, ha a dobókockákon szereplő számok összege pontosan megegyezik tízzel, akkor elkezd világítani zölden egy LED. Fontos részlet, hogy ne legyenek hamisak a dobókockák, tehát minden dobás során az egyes kockák értéke 1 és 6 között azonos valószínűséggel forduljanak elő.

Első lépésben, a már megszokott módon, elhelyeztem a Front Panelen az üzemeltetéshez elengedhetetlen eszközöket: a nyomógombot, a zöld színű kerek LED-et és egy numerikus kijelzőt, hogy a felhasználó lássa, hogy mennyi az adott dobás során, a kockákon szereplő értékek összege. (*5. ábra*)



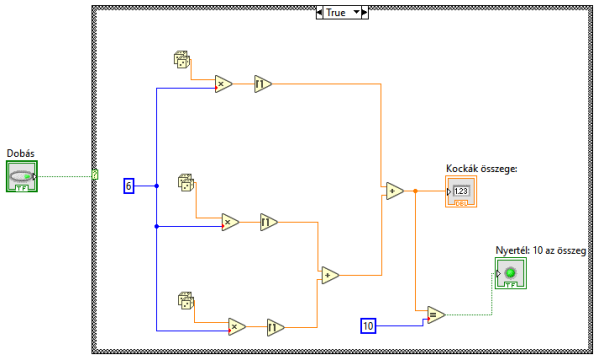

*5. ábra*
Kockajáték Front Panel

Második lépésben nekiláttam elkészíteni a program sofware-es hátterét. A korábbi példához hasonló, itt is két esetet kell megkülönböztetni. Egyik eset, amikor a gombot nem nyomták le, a másik esetben pedig, amikor a felhasználó elindította az adott kört, tehát lenyomta a gombot. Az első esetben nem kell semmit csinálnia a programnak, hiszen ilyenkor a játékos még nem indította el a játékot. Ez számomra azt jelenti, hogy a Block Diagramon létrehozott, a nyomógomb által vezérelt esetvizsgáló függvény „false" állásában nem szabad, hogy bármi is szerepeljen. *(6. ábra)*




*6. ábra*
False állás

Harmadik lépésben leprogramoztam annak az esetnek a kezelését, amikor a felhasználó megnyomja a gombot. Ekkor le kell generálni három, egymástól független, 1-6 közötti, egész számot, majd ezeknek az összegét ki kell írni a kijelzőre, valamint be kell kapcsolni az izzót abban az esetben, ha ez az összeg megegyezik tízzel. Ez az alábbi módon néz ki a LabVIEW-ban. *(7. ábra)*



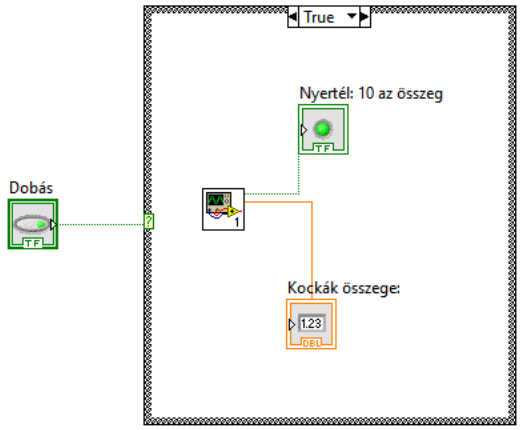

*7. ábra*
True állás

LabVIEW-ban a random szám generálására létrehoztam az esetvizsgálón belül három „Random Number (0-1)" nevű függvényt (ezek reprezentálják a három dobókockát). Ezek a függvények legenerálnak egy 0 és 1 között lévő számot. Nyilván való, hogy ez számomra csak részben jó, ezzel valamilyen matematikai műveletet kell végezni, hogy 1-től 6-g valamilyen egész számot kapjak. Erre tökéletes megoldás, hogy a generált számot megszorzom hattal, amely így 0 és 6 közötti tört szám lesz, majd ezt egy újabb függvény segítségével felfelé kerekítem. Ezzel a módszerrel a program

legenerál egytől hatig egy random számot, amely során az egyes számok az valószínűséggel fordulnak elő.

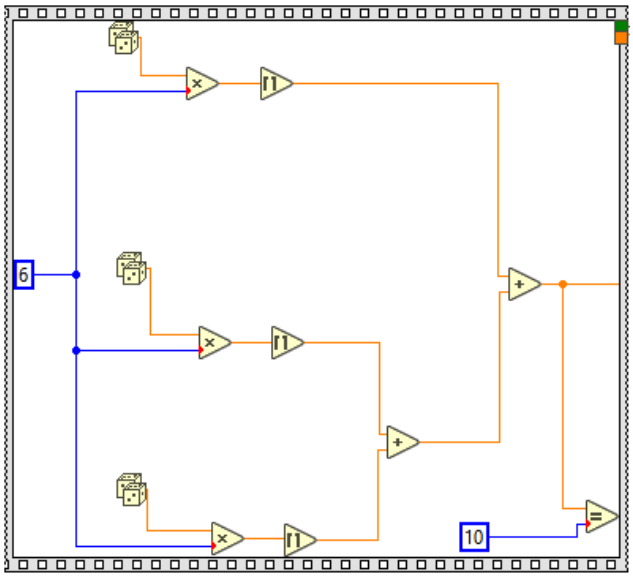
Ezt a random szám generálást a program háromszor végzi el egymás után és az így kapott három számot összegzi. Az összeadás elvégzése során az eredmény megjeleníti a kijelzőn. A programba fel kellett használ egy egyenlőséget vizsgáló függvényt, amely két bemenetére a kockák összegét és egy konstanst kötöttem. Ennek az állandónak az értéke 10, hiszen azt szeretnénk vizsgálni, hogy az összeg értéke megegyezik-e tízzel. A függvény kimenetét, amelyen vagy igaz, vagy hamis érték szerepel a működés során, a zöld LED-re kötöttem, amellyel azt értem el, hogy az egyenlőséget vizsgáló függvény értéke határozza meg az állapotát.

Utolsó lépésben az esetvizsgáló függvény belsejében található elemeket SubVI-ba rendeztem. *(8.ábra)*



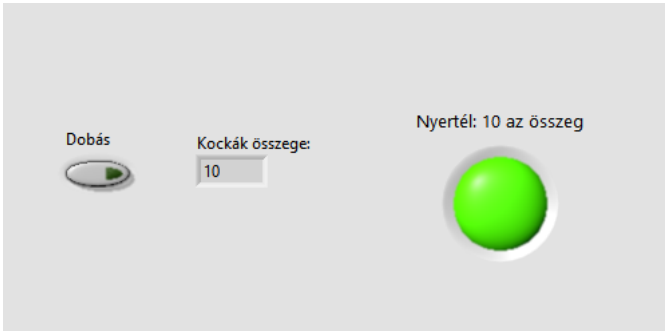

*8. ábra*
SubVI-ba rendezett függvények

A SubVI tartalma a következő ábrán látható. *(9. ábra)*




*9. ábra*
SubVI tartalma

A fent leírt lépések elvégzésével a program elkészült és a futtatások során hibátlannak bizonyult. *(10. ábra)*



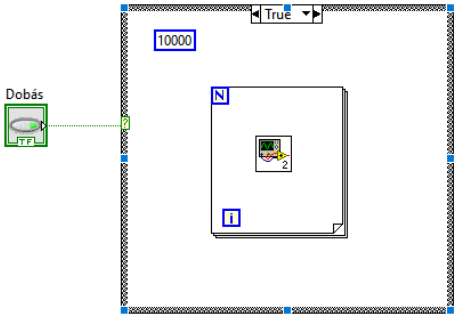

*10. ábra*
A program futás közben

## VI. KOCKAJÁTÉK GYAKORISÁGA

Ez a program a már V. pontban látott kockajáték módosított verziója lesz. Ez a software ebben az esetben minimum 10 000-szer fogja elvégezni a három kockával egyszerre való dobást és az értékek összegzését, majd ezeknek az értékeknek a gyakoriságát megjeleníti. (Hány alkalommal lett az összeg 3; 4; … 18.)

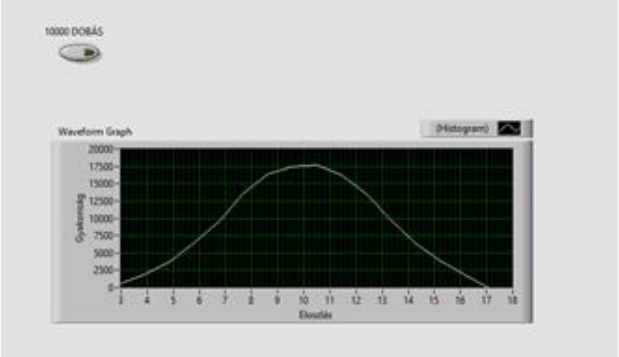
Első lépésben, a kockajáték programomból eltávolítottam az aktuális összeget monitorozó numerikus kijelzőt. Erre az eszközre többé már nincs szükség, hiszen a software a másodperc tört része alatt végzi el a 10 000 összeg kiszámolását és jeleníti azt meg, így a felhasználó csak az utolsó értéket érzékelné a kijelző, ami ebben az esetben szükségtelen. Hasonló indokkal távolítottam el a kerek LED-et is, amely abban az esetben világított, ha a dobott összeg értéke megegyezett tízzel.

Második lépésben leprogramoztam, hogy a kockák dobása és összegzése, a gomb lenyomását követően, 10 000-szer történjen meg. Ezt a LabVIEW-ba beépített „for” ciklussal valósítottam meg úgy, hogy a már meglévő SubVI-t belehelyeztem a ciklusba. A for ciklusnak az a tulajdonsága, hogy addig ismétel egy adott feladatot, amíg egy előre meghatározott érték el nem ér egy általunk választott értéket. Ebben az esetben a „N”-et futtatjuk 1-től 10 000-ig. N=1 a első kör után, N=2 a második kör után, és így tovább addig, ami el nem éri a 10 000-dik kört, ami után a program leáll és megjeleníti a kért értékeket. LabVIEW-ban az N végső értékének megadását egy numerikus konstanssal kell megtenni. A második lépést a következő ábra szemlélteti. *(11. ábra)*

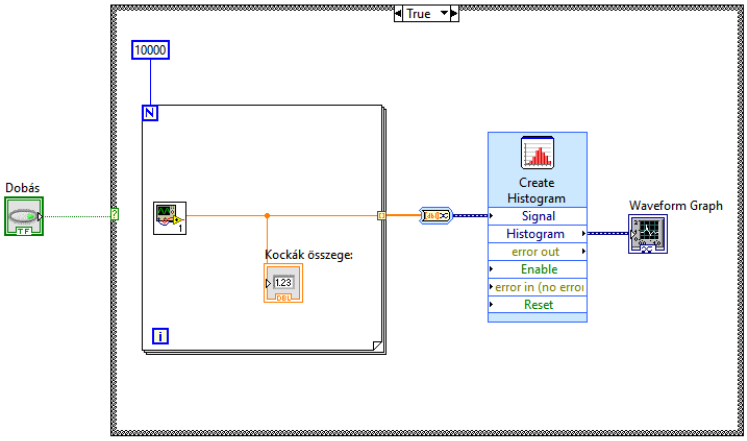



*11. ábra*
For cikklus

Harmadik lépésben létrehoztam egy hisztogram előállító függvényt, amelynek „Signal” nevezetű bementére a SubVI numerikus kimenetét kötöttem. Miután ezzel végeztem létrehoztam a felhasználó számára egy panelt, amely grafikusan szemlélteti a dobott értékek eloszlását. Ennek a gráfnak a vízszintes tengelyén a dobott értékek láthatóak, függőleges tengelyén pedig az egyes értékek száma (10 000-ből hányszor szerepelt a dobott érték). *(12-13. ábra)*

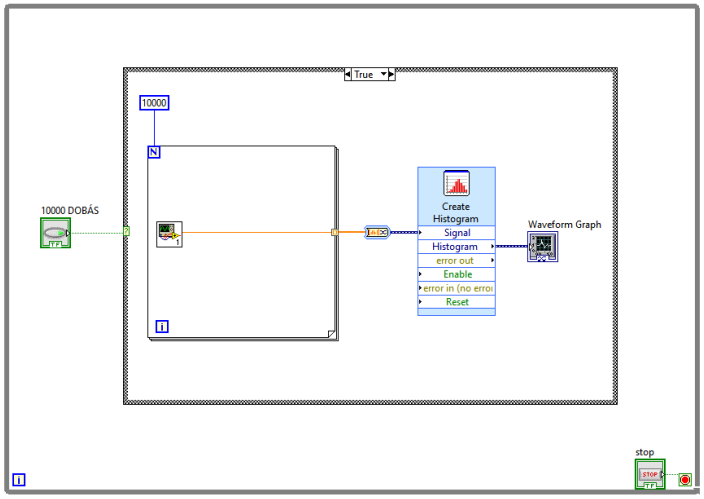



*12. ábra*
Front Panel a gráffal




*13. ábra*
Gráf software-s része

Negyedik lépesben az eddig elkészült programot belehelyeztem egy „while” ciklusba. While ciklus általában addig fut, amíg egy adott feltétel nem teljesül. Az én programomban ez a feltétel a felhasználói felületen elhelyezett STOP gomb megnyomása, tehát a program barmikor megszakítható futás közben, ha a felhasználó úgy akarja. Az elkészült program a következő ábrán látható. *(14. ábra)*



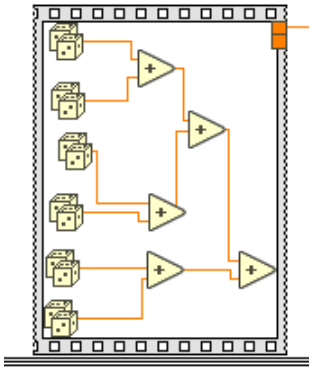

*14. ábra*
While ciklussal kiegészített program

Az elkészült program futtatása után láthatjuk az eloszlást a front panelen elhelyezett kijelzőn (lásd 12. ábra). A grafikonból látható, hogy az értékek egy úgy nevezett normális, más néven Gauss-eloszlást követnek. Ennek képe egy haranggörbe. Ez abból adódik, hogy a három dobókockával dobható összegek nem azonos valószínűséggel fordulnak elő. Ennek oka, hogy vannak összegek, amelyeket több féle kombinációval lehet kidobni, mint másokat. Például hármat csak egyféleképpen lehet kidobni, ha mind a három kockával egyest dobunk, míg tízet sokkal több módon. Ez az eloszlás ideális esetben egy szimmetrikus görbét alkot, ehhez nagyon jól közelít az én programom által rajzolt grafikon is. Ebből láthatjuk, hogy a program a feljebb taglalt lépésekkel elkészült és az elvárt feladatot hibamentesen elvégzi.

## VII. GYAKORISÁG SZÁMOLÁSA

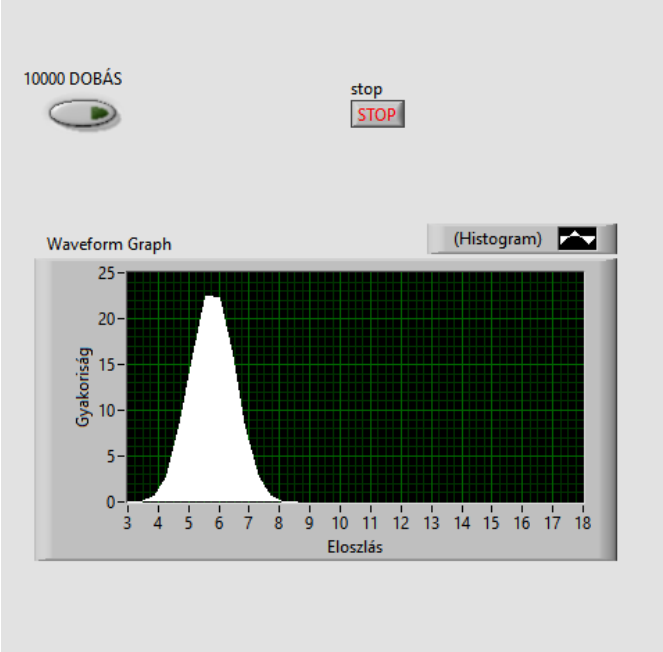
Ez a program az előző program egy variánsa, melyben a dobókocka 1-6 értéke helyett folytonos 0-1 intervallumot használ és a gyakoriságot legalább 20 független értékből képezi.

Ehhez a program elkészítéséhez felhasználtam a VI. pontban elkészített, hibátlanul futó programot. a meglévő programon az alábbi módosítást végeztem. Az előzőleg elkészített programban a SubVI tartalmazott egy számolást, amely hattal való szorzással és kerekítéssel a 0-1 intervallumú számot 1-6 intervallumú számmá alakította. Erre a lépesre a mostani programban nem volt szükség, hiszen számomra az a cél, hogy 0-1 intervallumon adjak össze véletlen számokat. számokat. *(15. ábra)*

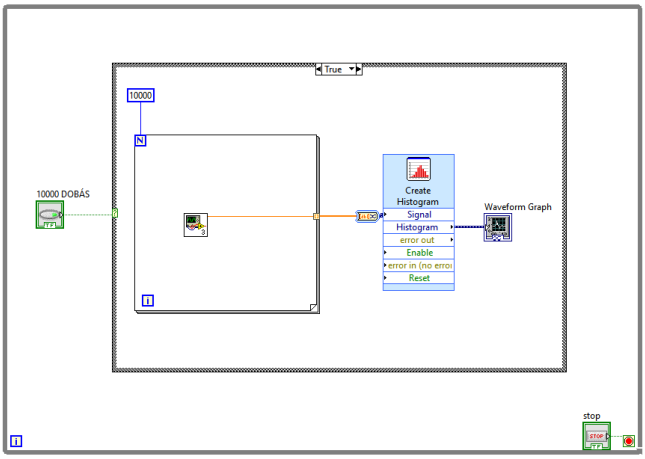


*15. ábra*
A SubVI tartalma

A programot lefuttattam és számomra megfelelőképpen futott. *(16-17. ábra)*




*16. ábra*
Elkészült program: Front Panel




*17. ábra*
Elkészült program:Block Diagram

## VIII. VÁLASZTHATÓ JEL SZIMULÁLÁSA

Ennek a programnak az lesz a feladat, hogy a felhasználó igénye szerint tudjon előállítani és megjeleníteni sinus, négyszög, fűrész valamint háromszög alakú jelet. Emellett, a felhasználó kedve szerint tudja majd állítani a jel három paraméterét: amplitúdó, offset, frekvencia, melyekből a program ki fogja számolni az effektív feszültség értékét és azt egy numerikus kijelzőn meg fogja jeleníteni.

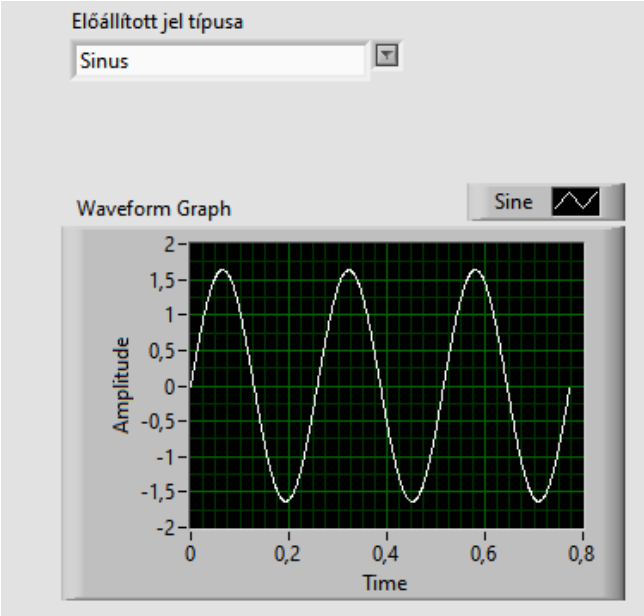
Első lépésben létrehoztam egy „while” ciklust a Block Diagrammon. Erre azért volt szükség, hogy később a felhasználó bármikor le tudja állítani egy gomb lenyomásával a program futását. Innentől kezdve minden felhasznált függvény valamint fizikai eszköz a while cikluson belül került programozásra.

Második lépésben a Front Panelen elhelyeztem a használathoz elengedhetetlen eszközöket. Négy inputot raktam le a felhasználói felületre. A „Combo Box” nevezetű input biztosítja, hogy a felhasználó ki tudja választani a kívánt alakzatot, míg három egyenként nullától tízig állítható csuszka pedig a grafikon paramétereinek állítására szolgál (amplitúdó, offset, frekvencia). A negyedik vezérlő pedig egy STOP gomb, amellyel a felhasználó leállíthatja a program futását az előbb taglalt módon. Ezeken kívül elhelyeztem két numerikus kijelzőt melyek egyike a számolt a másik pedig az elvárt effektív feszültséget fogja kijelezni. Végül létrehoztam a panelt, amelyen kirajzolja a program a választott grafikont.

Harmadik lépésben a létrehoztam egy függvényt („Case Structure”), amely segítségével szét lehet választani a négy különböző, a felhasználó által kiválasztott grafikonrajzolási módot. A függvény a korábban már létrehozott „Combo Box” állásának megfelelő módon fog futni. A Case Structure függvénynek három bemenete van minden állásban: amplitúdó, offset, frekvencia és három kimenete. Két numerikus, amelyek az effektív feszültséget jelenítik majd meg és egy, a grafikonok megjelenítésére szolgáló panel. Ezeket már a második lépésben elhelyezetem a Front Panelen.

Negyedik lépésben leprogramoztam az egyes esetek során a kirajzoláshoz szükséges elemeket az alábbi módon. Minden esetnél elhelyeztem egy-egy a jel szimulálására szolgáló függvényt, melynek bemeneteire az amplitúdó, a frekvencia és az offsetet állító csúszkák kimenetét kötöttem. A szimulátor megfelelő típusú kimenetét (sinus esetén sine stb.) pedig rákötöttem a grafikonrajzoló panelre.

Ezek után lefuttattam a programot, hogy ellenőrizni tudja, hogy az eddig elkészített funkciók megfelelően működnek-e. A futás során nem tapasztaltam semmiféle rendellenességet *(18. ábra)*

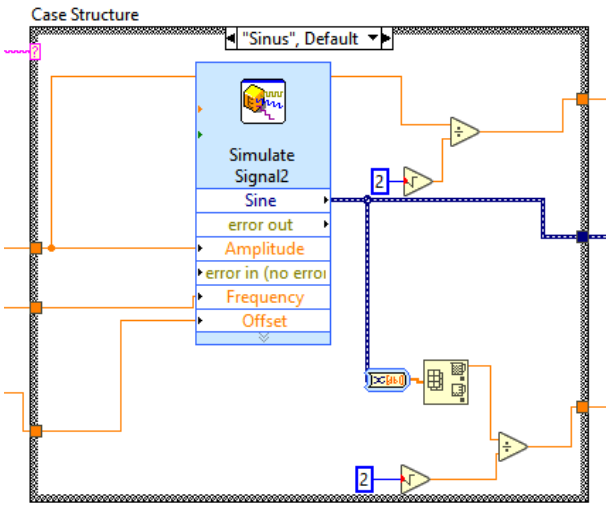



*18. ábra*
A kirajzolt grafikon

Utolsó lépésben elkészítettem az effektív feszültségek számolására szolgáló programot. Ezt minden görbetípusra azonos módon tettem meg, így csak egy (sinus) görbe esetén írom le a lépéseket, de ezek mindegyik esetre igazak és helyesen működnek. Matematikai összefüggésből tudom az effektív érték az alábbi képlettel számítható ki:

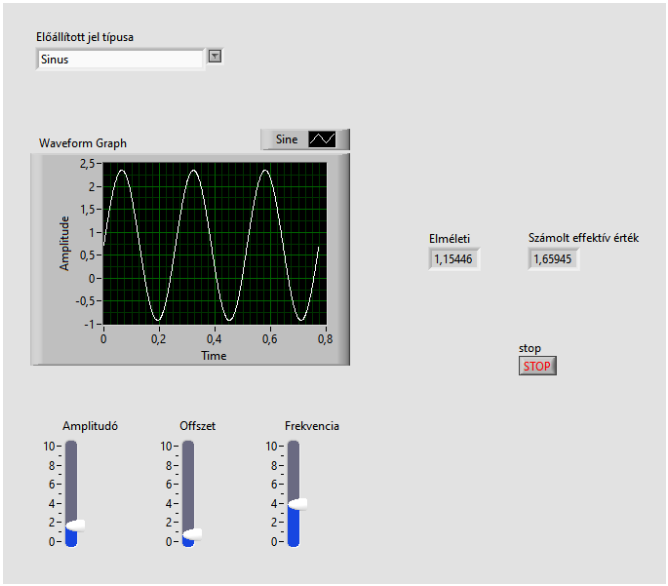
$$U_{\text{eff}} = \frac{X}{\sqrt{2}}, \qquad \text{x csúcsérték eseten.}$$

A sinusgörbéből, az adatokat eltároltam egy tömbben, ebből a tömbből lekértem maximális értéket, amelynek hányadosát vettem $\sqrt{2}$ –vel, így kiszámítva az effektív értéket. A tömb maximuma ebben az esetben a generált jel csúcsértékét adja vissza. Az elméleti effektív érték kiszámítást hasonlóan programoztam le kivéve, hogy itt a csúcsértéket egyenlővé tettem az amplitúdó, mivel definíció szerint az amplitúdó időben változó mennyiségek legnagyobb eltérése az egyensúlyi állapottól (csúcsérték). *(19. ábra)*

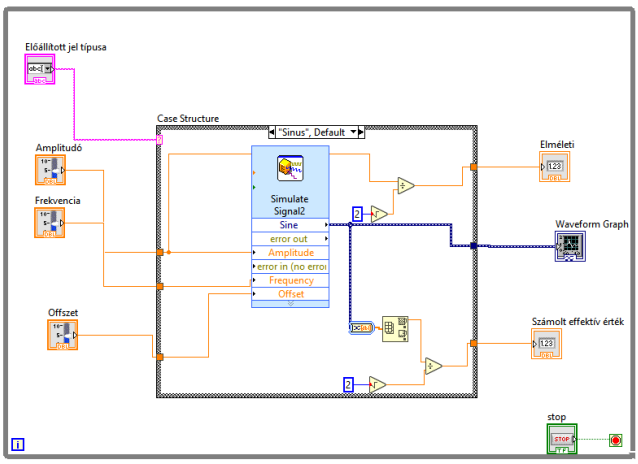



*19. ábra*
Effektív értéket számoló program

Az elkészült program futtatása során látható, hogy hibamentesen fut, viszont egyes paramétereknél a számolt és az elméleti effektív értékek eltérnek. Ez abban az esetben fordul elő, ha az offset paraméter nem nulla. Ennek az az oka, hogy míg az elméleti érték számolása során az amplitúdóval számol a program, addig a számolt érték előállítása során a tömb maximumával számol, amely nem minden esetben egyezik meg a csúcsértékkel. Ezen kívül a program tökéletesen fut és elvégzi az elvárt feladatokat. *(20-21. ábra)*




*20. ábra*
Elkészült program: Front Panel




*21. ábra*
Elkészült program: Block Diagram

# Felhasznált források


Online LabVIEW Help Resources

LabVIEW mérési utasítás

LabVIEW segédlet

Normális eloszlás

Effektív érték számolása