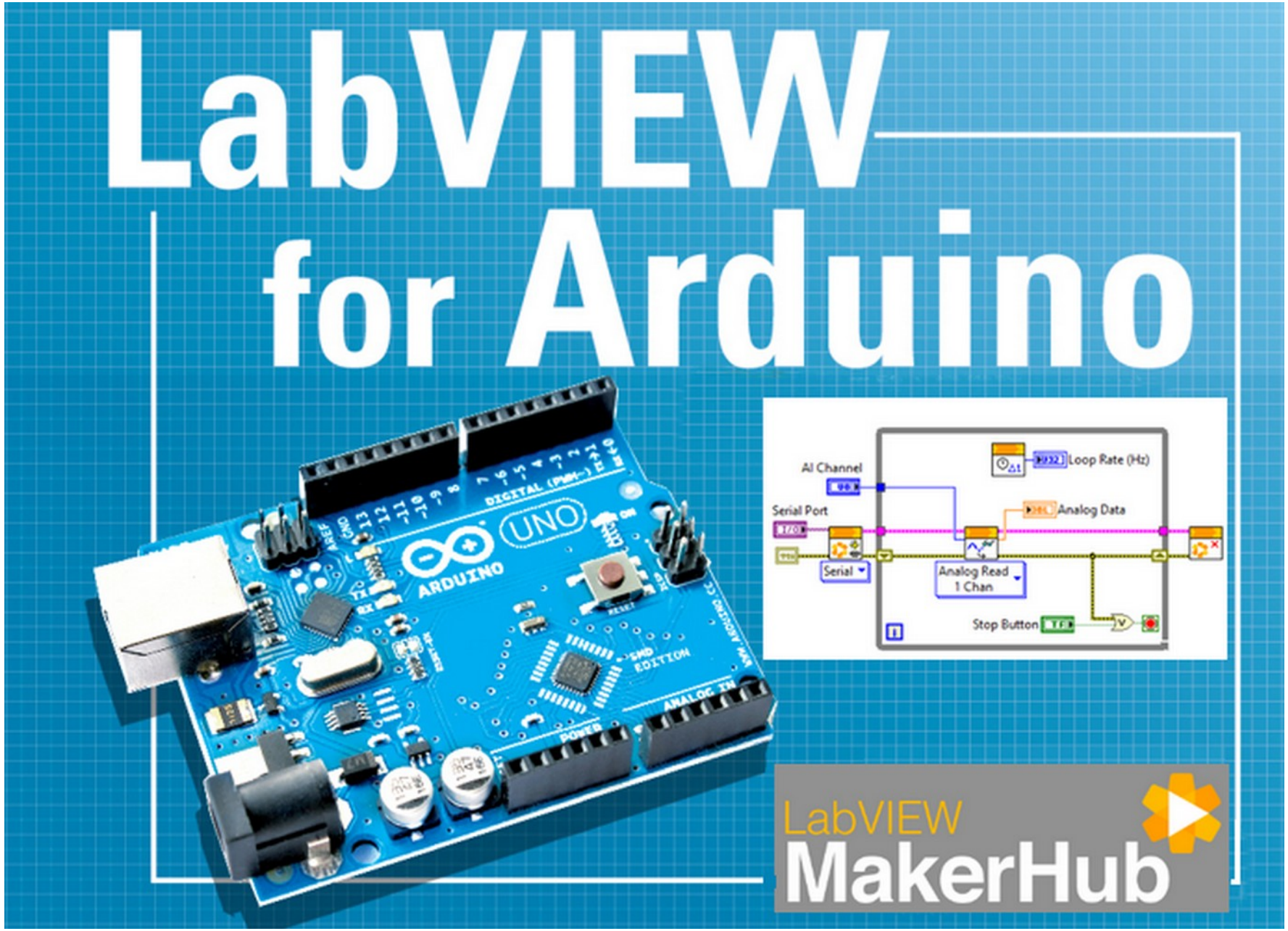
12. LabVIEW + LINX + Arduino - 3. rész

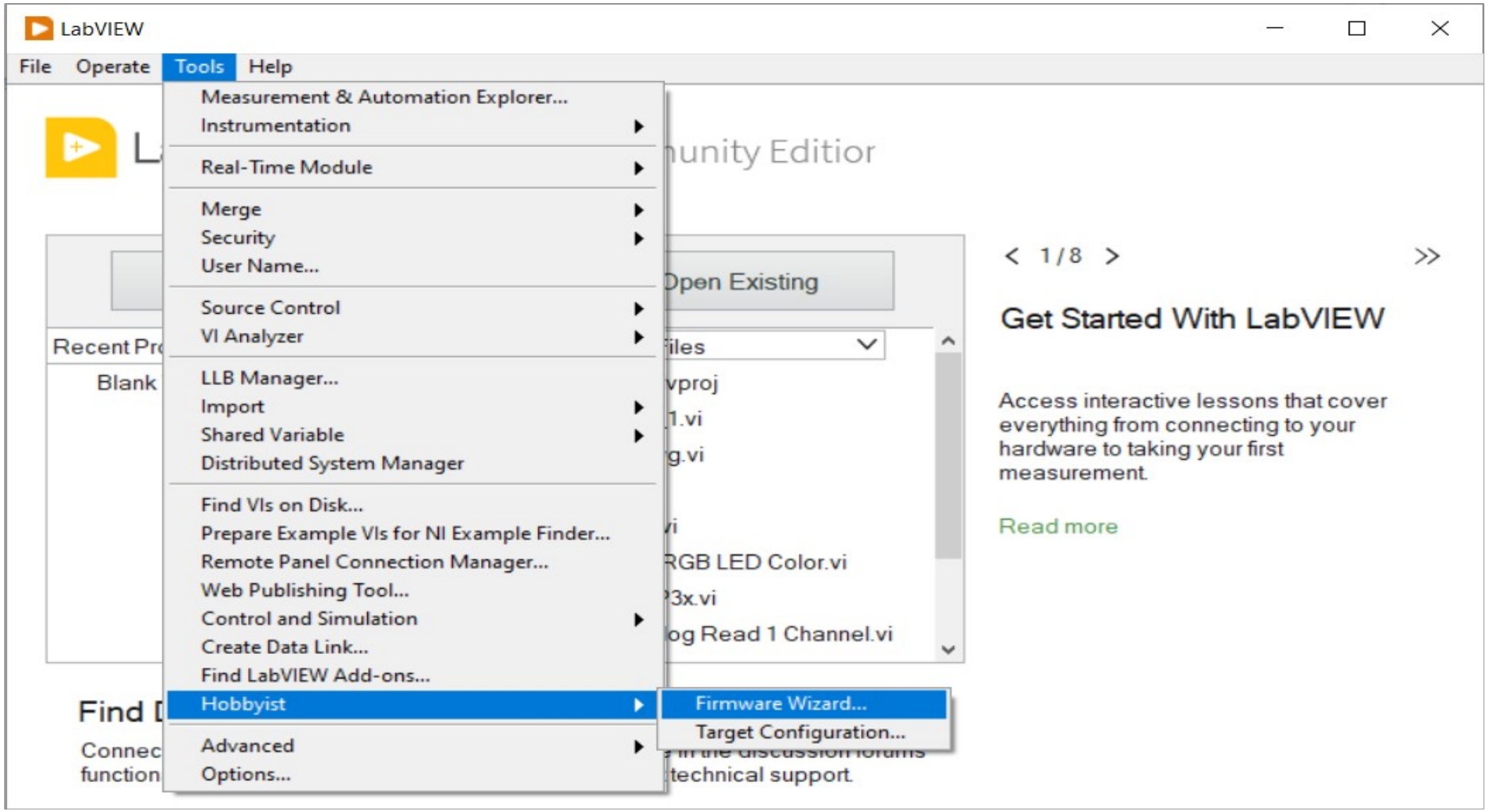


Felhasznált és ajánlott irodalom

- NI: Getting Started with Arduino and LabVIEW Community Edition
- NI: LabVIEW Documentation
- NI: Different Methods for Representing Data on an XY Graph
- NI: Types of Graphs and Charts
- Szabó Norbert: LabVIEW bevezető
- Jáger Attila: LabVIEW alapismeretek: 1. fejezet, 2. fejezet, 3. fejezet, 4. fejezet, 5. fejezet, 6. fejezet
- Friedl Gergely: LabVIEW segédlet
- Jeffrey Travis, Jim Kring: LabVIEW for Everyone (3rd Edition)
- Hans-Petter Halvorsen: LabVIEW LINX and Arduino
- SIN Consulting: LabVIEW-Basics (programgyűjtemény)
- @professionalengineer317: digital oscilloscope in STM32 and Labview

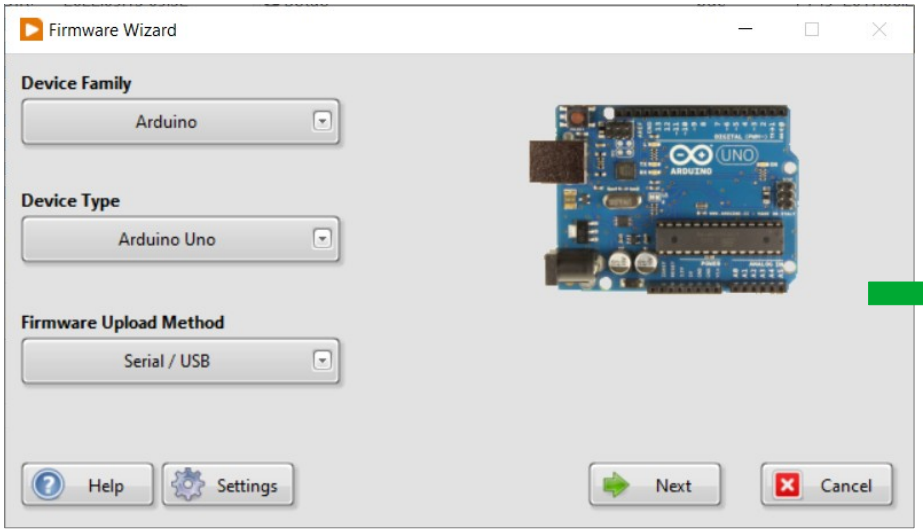
A LINX firmware újrafordítása 1. lépés

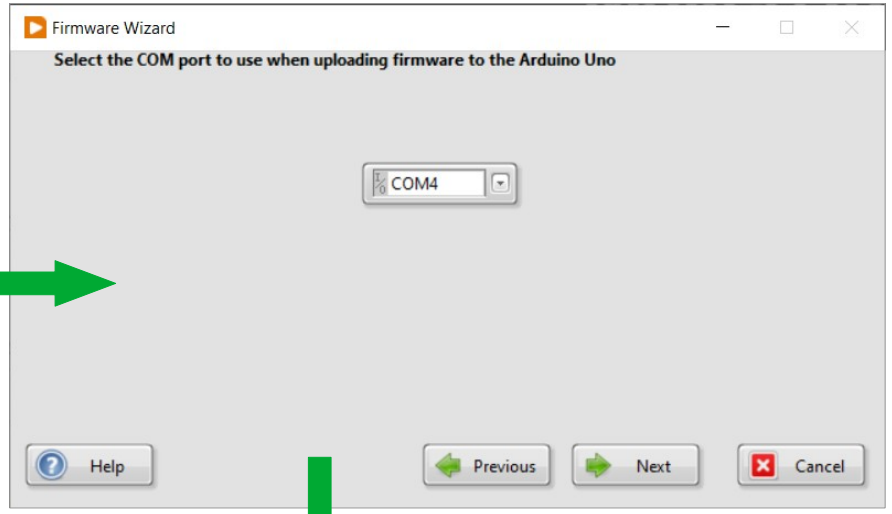
- Néha szükségünk lehet a „gyári” **LINX Arduino Uno** firmware módosítására, ekkor le kell fordítani azt, ennek lépéseit ismertetjükaz alábbiakban:
- Kattintsunk a LabVIEW **Tools→Hobbyist→Firmware Wizard** menüpontjára!

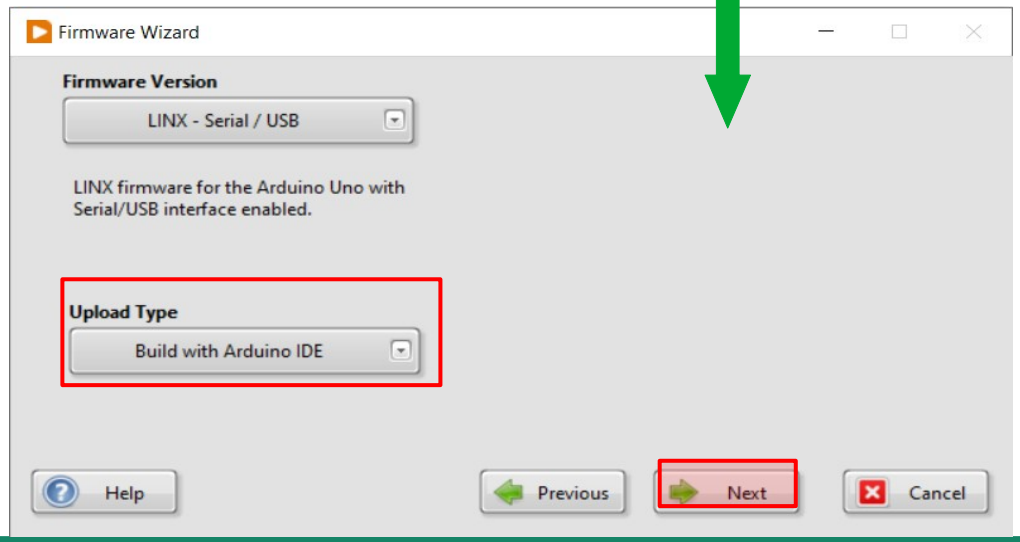


A LINX firmware újrafordítása 2. lépés

- A **Firmware Wizard**-ban válasszuk ki a kártyánk típusát, csatlakoztassuk a kártyát a PC-hez, válasszuk ki a soros portot, a feltöltés típusánál (*Upload Type*) pedig válasszuk a **Build with Arduino IDE** opciót!

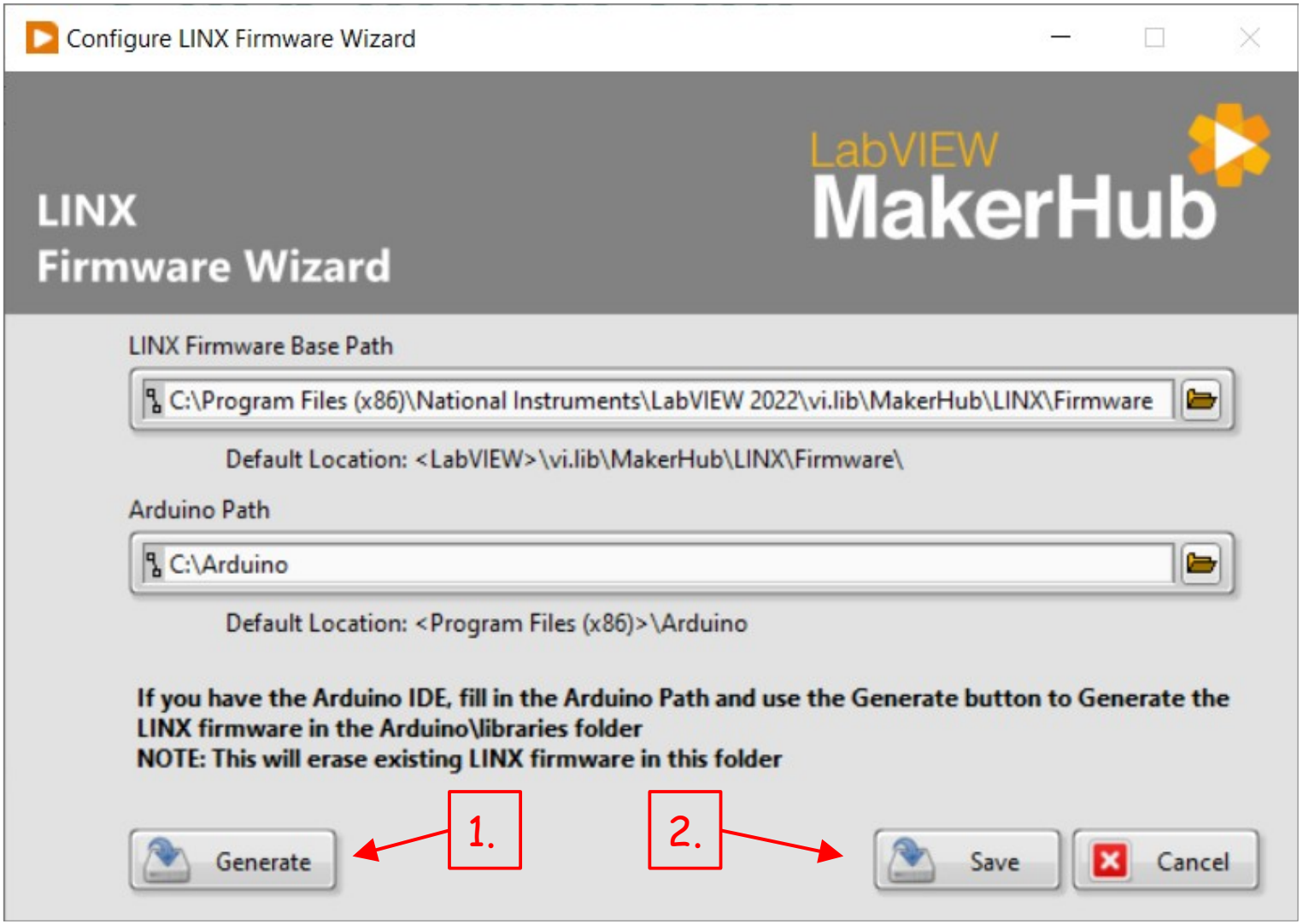






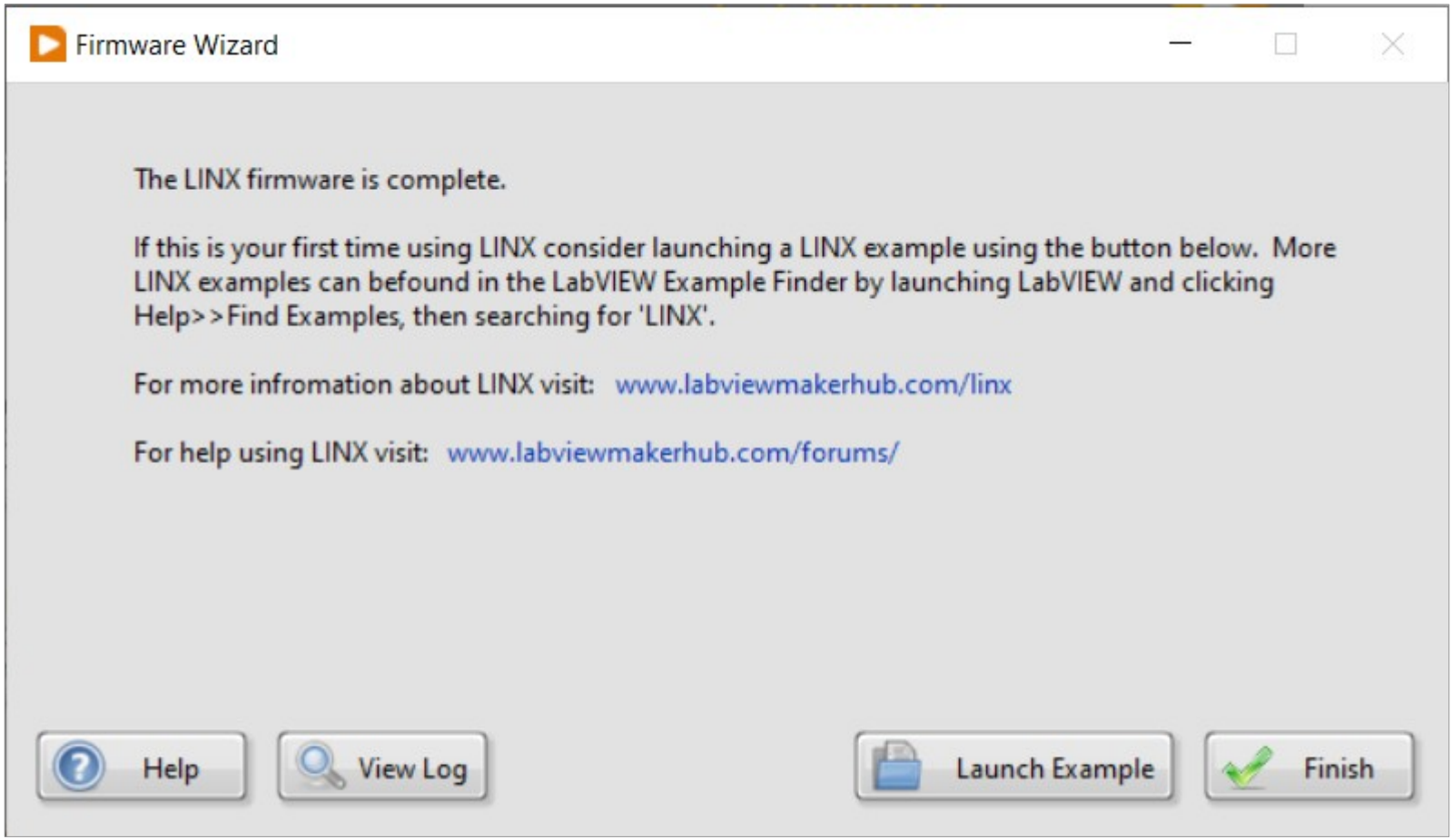
A LINX firmware újrafordítása 3. lépés

- Állítsuk be a **LINX firmware** és az **Arduino IDE** elérési útját, majd használjuk előbb a **Generate,** majd a **Save** gombot!



A LINX firmware újrafordítás vége

- Sikeres fordítás és programletöltés után a képen látható ablak jelenik meg. Itt a **Finish** gombra kattintva kiléphetünk, vagy a **Launch Example** gombra kattintva betölthetjük a **LINX - Blink (Simple).vi** mintapéldát



Az adatátviteli sebesség megváltoztatása

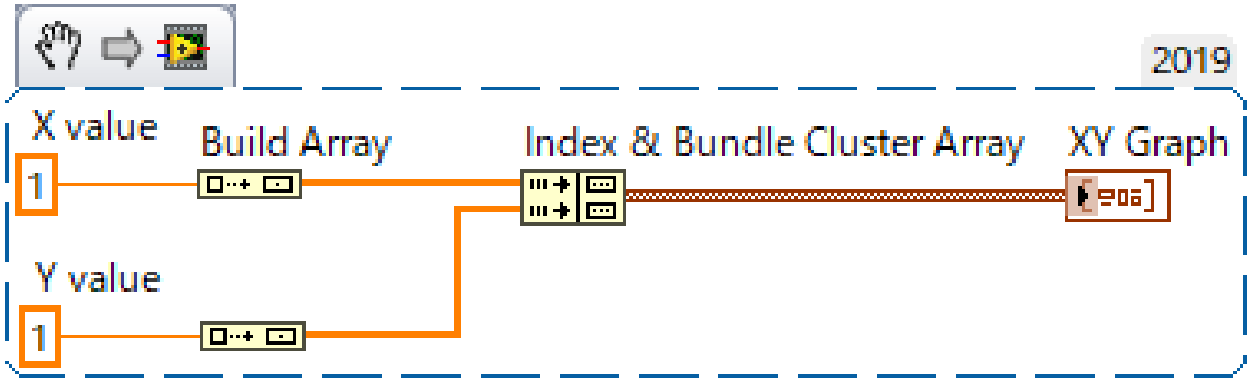
- A **LinxArduinoUno.cpp** nevű állomány több helyen is megtalálható, de ezek közül tapasztalataim szerint csak a **LabVIEW** telepítési helyén a **vi.lib\MakerHub\LINX\Firmware\Source\core\device** mappában található változat hatásos, ezt kell módosítani
- Keressük meg az alábbi sort és egészítsük ki:

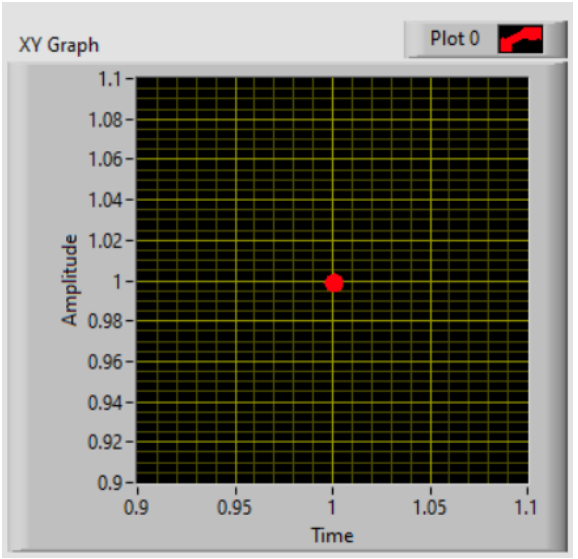
```
unsigned long LinxArduinoUno::m_UartSupportedSpeeds[NUM_UART_SPEEDS] = {
300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600, 115200,
230400, 2000000};
```

- Alapértelmezetten a legutolsó listaelem lesz kiválasztva, mint maximális megengedett adatátviteli sebesség
- A **LINX** leírása szerint a kapcsolat megnyitásakor 9600 bps sebességgel indul az adatforgalom, majd a megfelelő parancsok kiküldésével a maximális megengedett sebességre átállítva folytatódik az átvitel

Adatmegjelenítés az XY Graph modullal

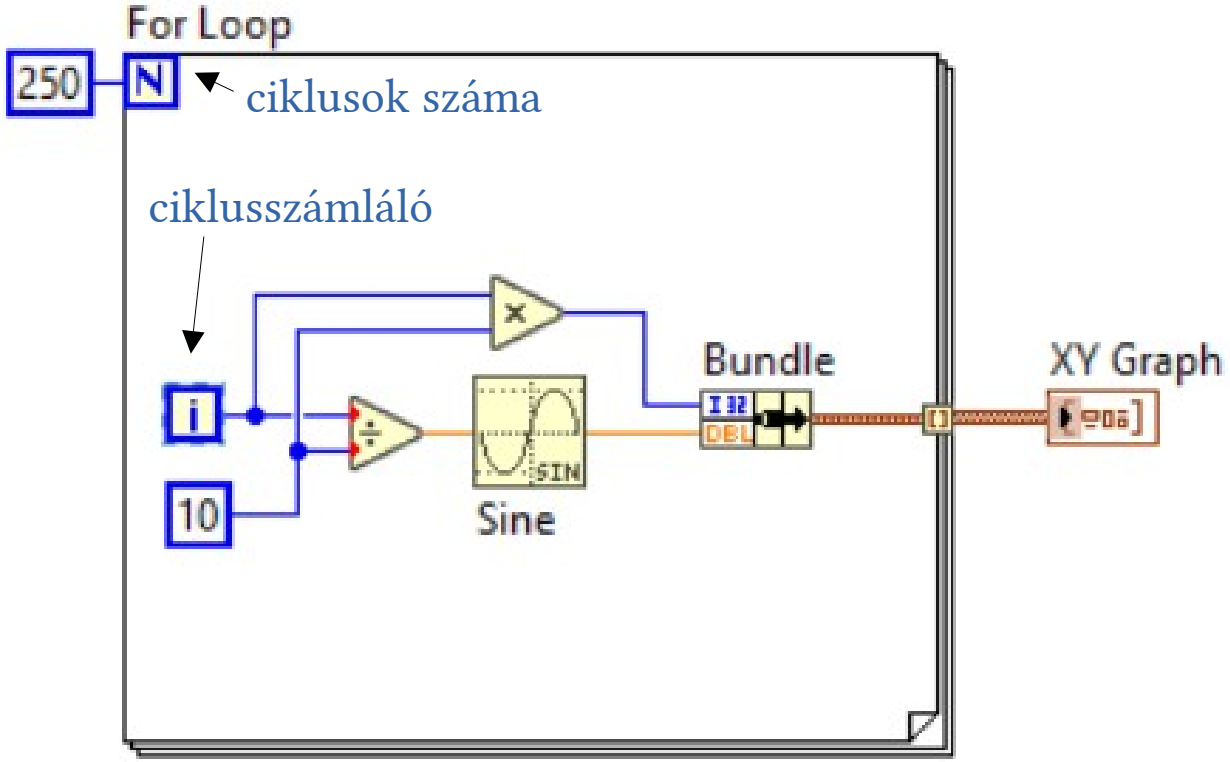
- Az **XY Graph** modul adatpárokat fogad, amelyek a megjelenítendő pontok X és Y koordinátáit jelentik
- Az NI: Different Methods for Representing Data on an XY Graph útmutatója többféle módszert mutat be az **XY Graph** használatára
- A legegyszerűbb ábra, egyetlen pont kirajzolásához a pont koordinátáit (itt egyelemű) tömbbé kell alakítani a **Build Array** függvénnyel, majd a két tömböt össze kell fogni egy **Bundle,** vagy egy **Index & Bundle Cluster Array** függvénnyel
- A **Build Array** függvénnyel a későbbi példáinkban természetesen több adatot is összefűzünk majd…

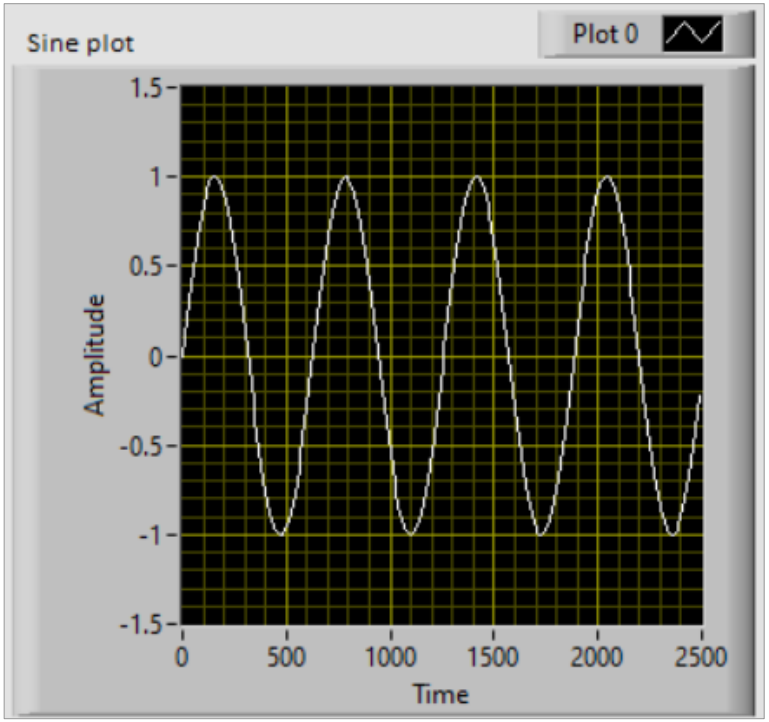




Adatmegjelenítés az XY Graph modullal

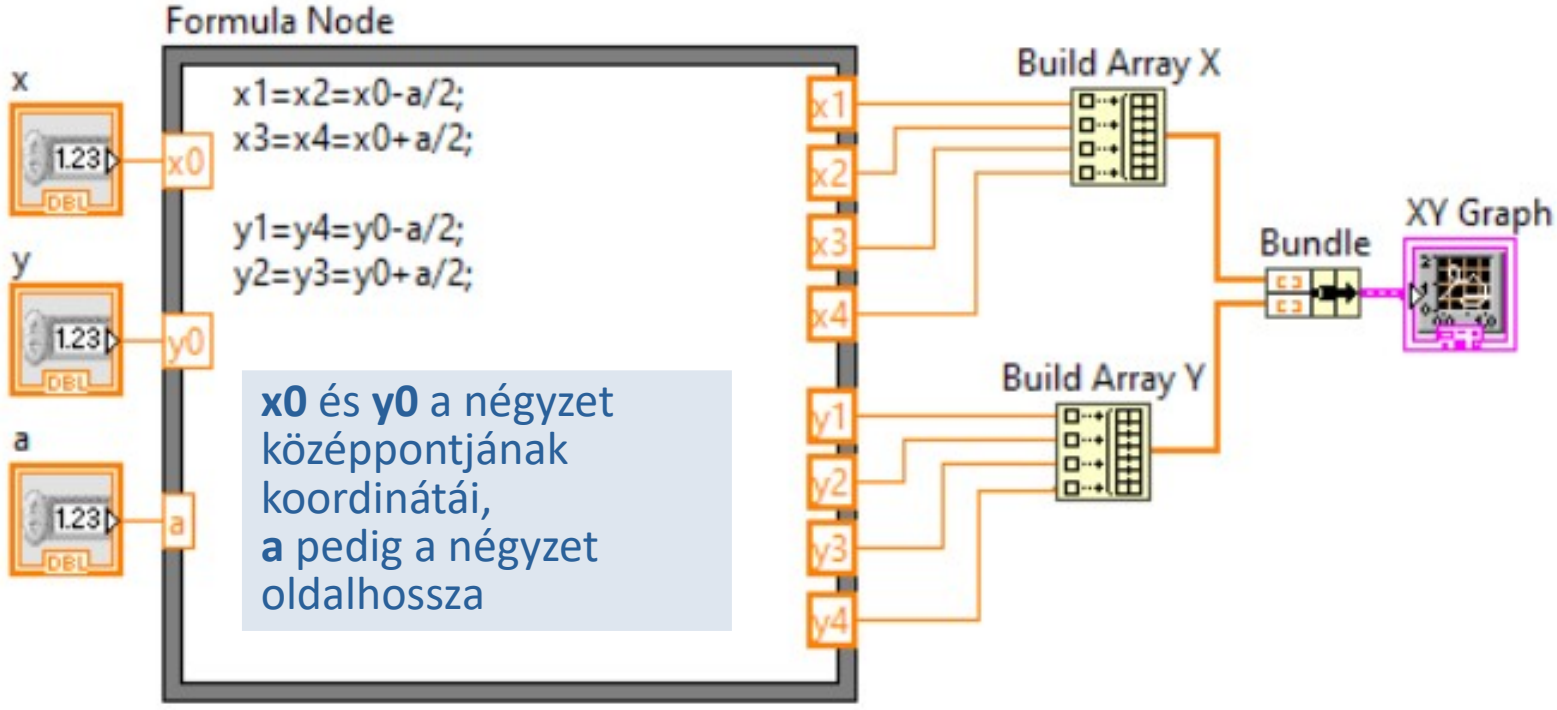
- Az alábbi példában (**sineplot.vi**) egy szinuszgörbét jelenítünk meg
- Itt előbb x-y adatpárokat képezünk, majd ezekből építünk tömböt ahol $x_i = i * 10$, $y_i = sin(i/10)$
- A 2023. február 23-i előadásban a **TPM36_avg.vi** alkalmazásban már láttuk, hogy a **for** ciklusból származó adatokat a **LabVIEW** egy autoindexelt tömbbe összesíti, amit közvetlenül átadhatunk az **XY Graph** modulnak, tehát most nincs szükség a **Build Array** függvény használatára





Negyzet.vi

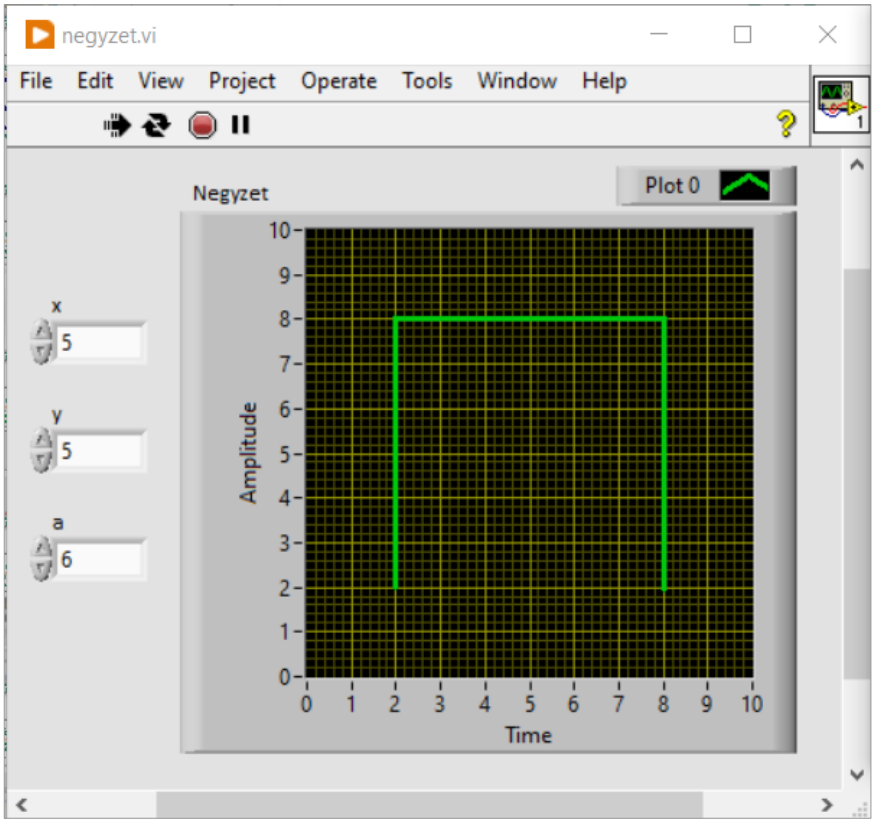
- Jáger Attila **jegyzetében** található ez a mintapélda, amelyben egy négyzet sarokpontjainak koordinátáit számítjuk ki, s ezeket adjuk át a kirajzolást végző **XY Graph** modulnak
- A számolást egy **Formula Node**-ba írjuk bele, s a bemenő, illetve kimenő adatokat nevezzük el az ábrán látható módon
- Az X és Y koordinátákból egy-egy tömböt alkotunk, amelyeket egy **Bundle** típusú modul kötegel össze az **XY Graph** számára

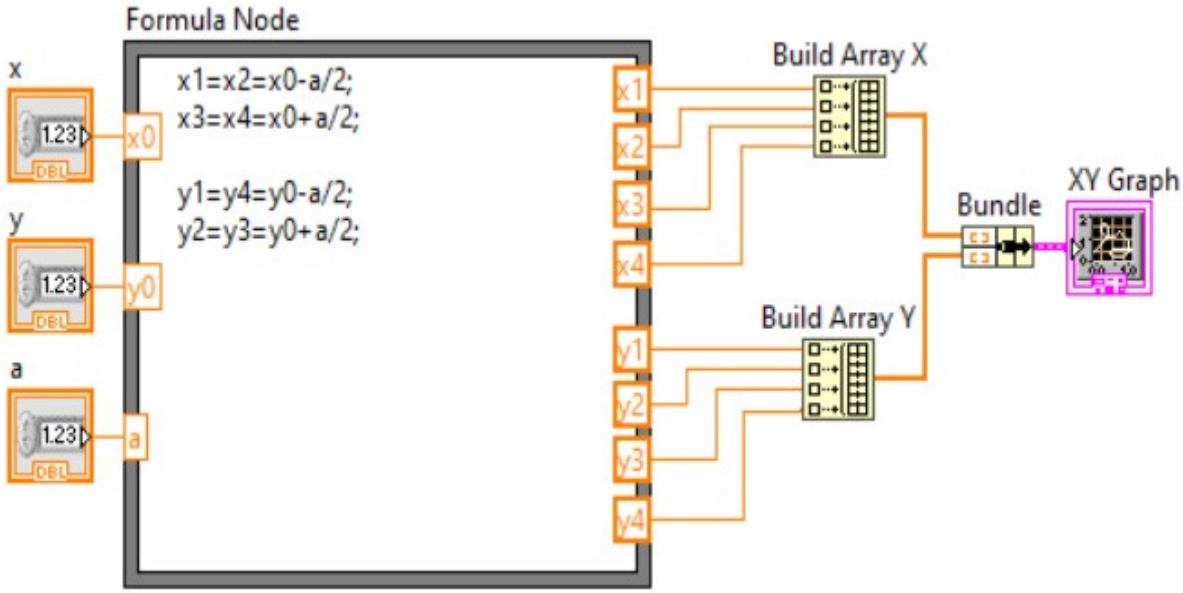


Negyzet.vi – így még hibás

„Ah de játszi fénye múl,/A midőn a nap sugára/Kék hegyek közé vonúl.” *(Kisfaludy K.)*

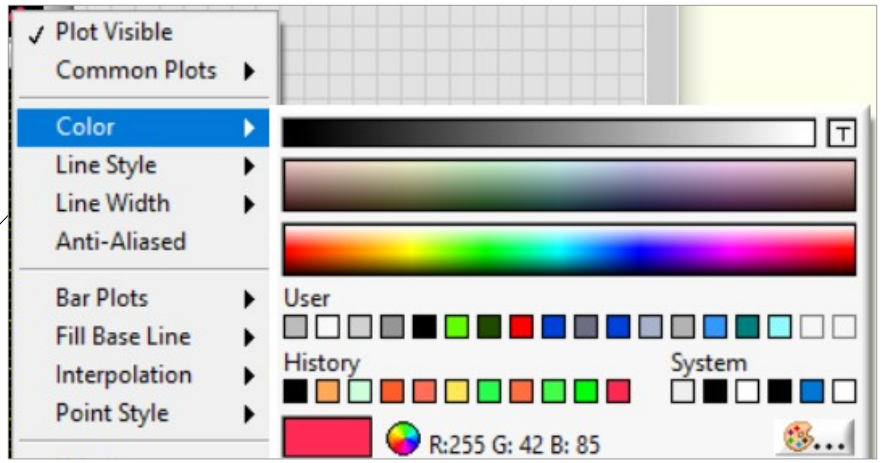
- Hiba csúszott az elgondolásba, mivel a négy oldal kirajzolásához öt adatpár kell, a kiindulás pont koordinátáit még egyszer meg kell adni! Ezt legegyszerűbebn a **Build Array** modulok kiterjesztésével és az ***x1,*** ill. ***y1***-ről jövő vezetékek átkötésével oldhatjuk meg

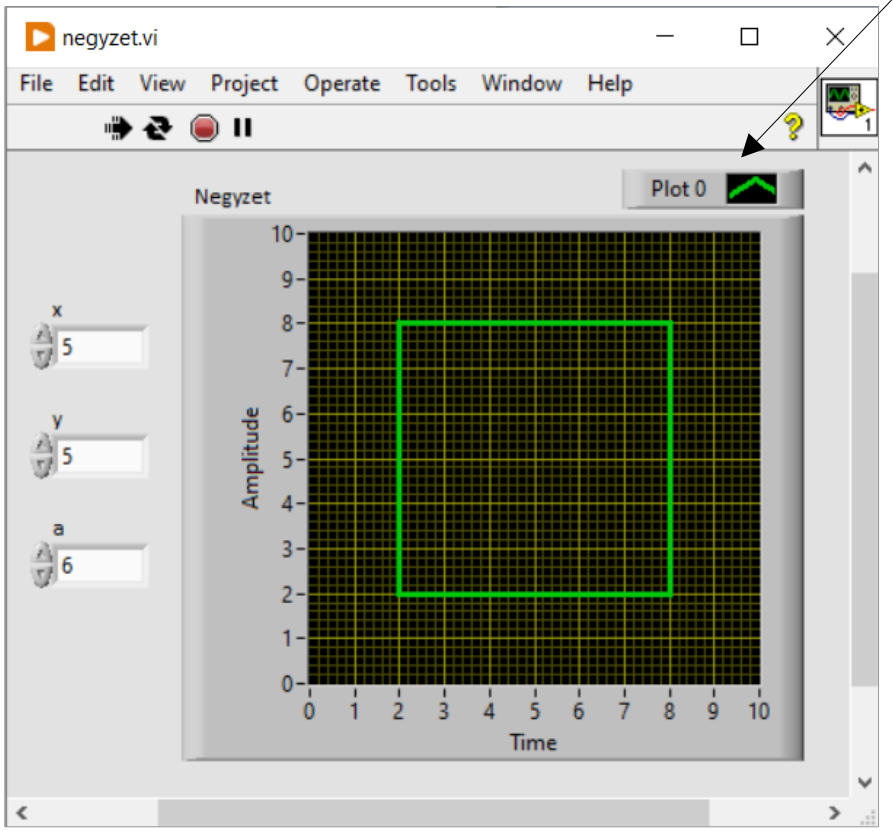


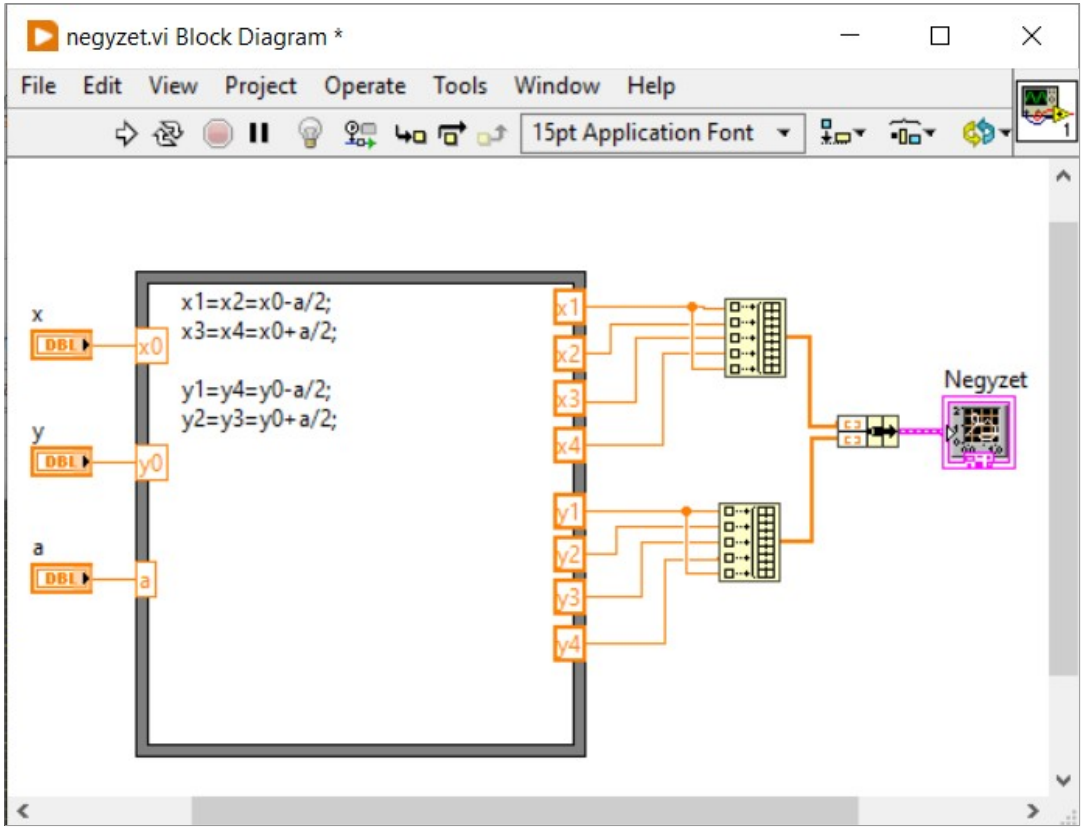


Negyzet.vi – javított változat

- Így már rendben van a kirajzolás
- **Az ábra tulajdonságait** a jobb felső sarokban látható sematikus ábrára kattintva állíthatjuk be

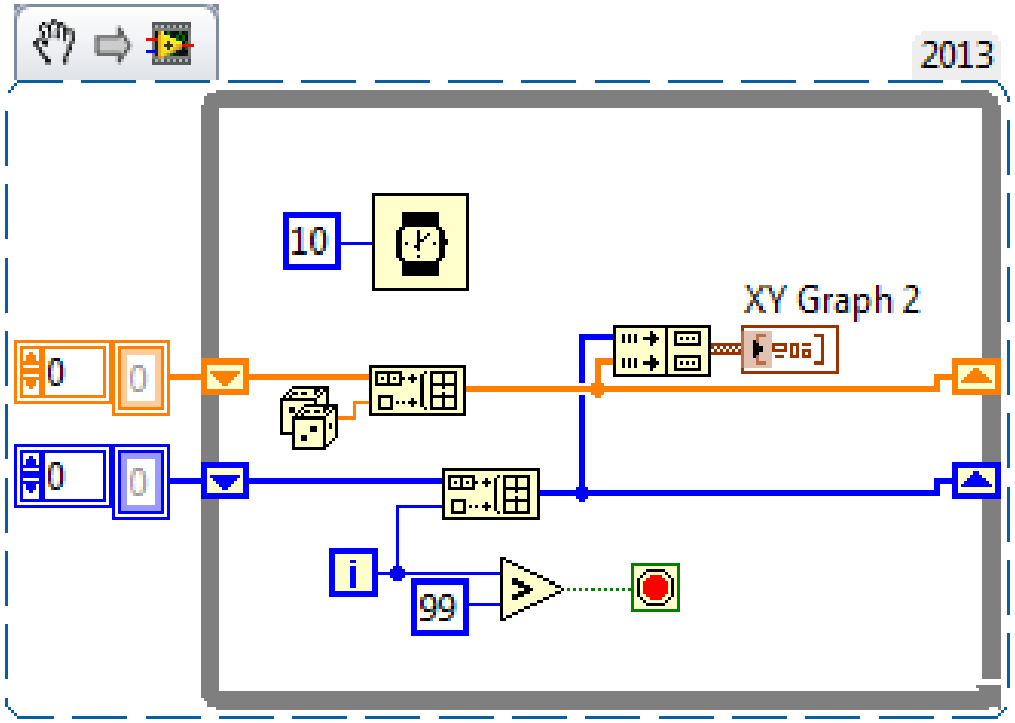


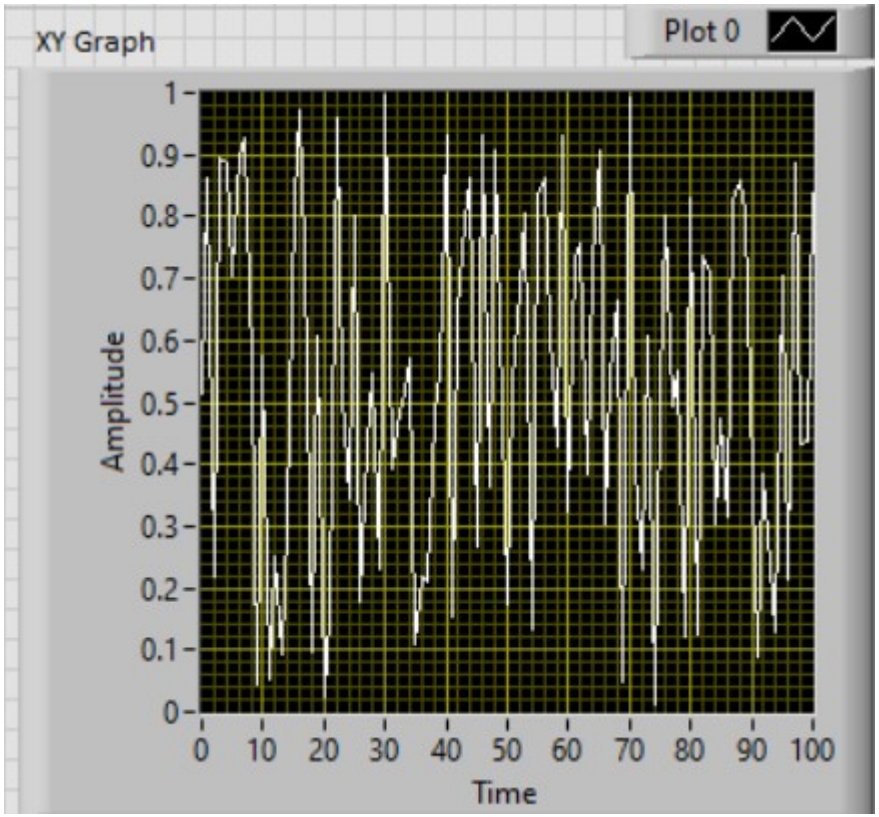




update_array.vi

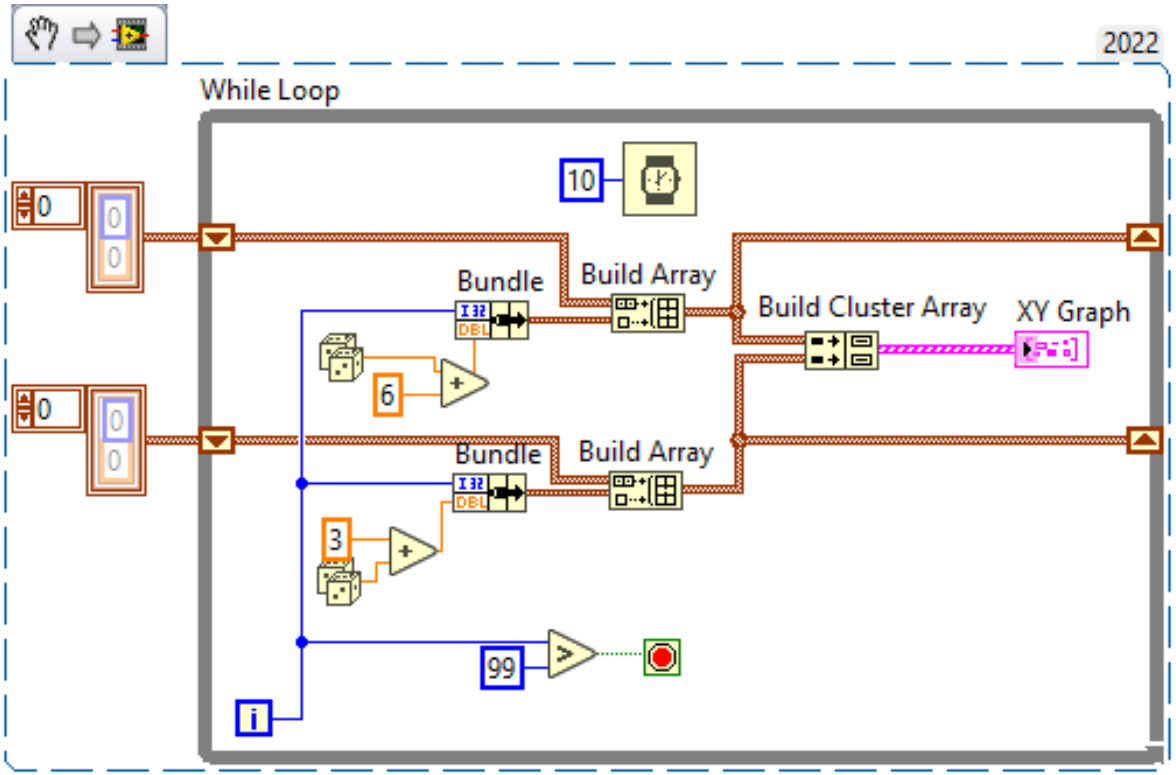
- Az NI: Different Methods for Representing Data on an XY Graph útmutatója következő mintapéldája azt mutatja be, hogy a kiindulási tömböket hogyan bővíthetjük pl. egy while ciklusban, s hogyan frissíthatjük egyidejűleg a diagramot is
- Itt most 100 db véletlenszámot generálunk és ezeket jelenítjük meg

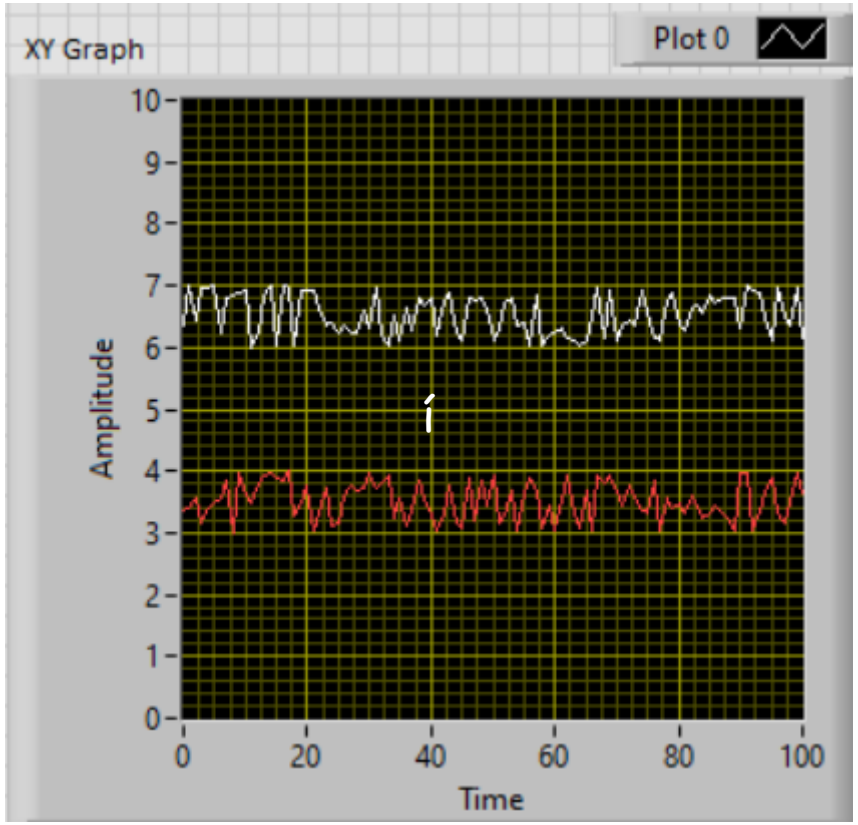




dual_plot.vi

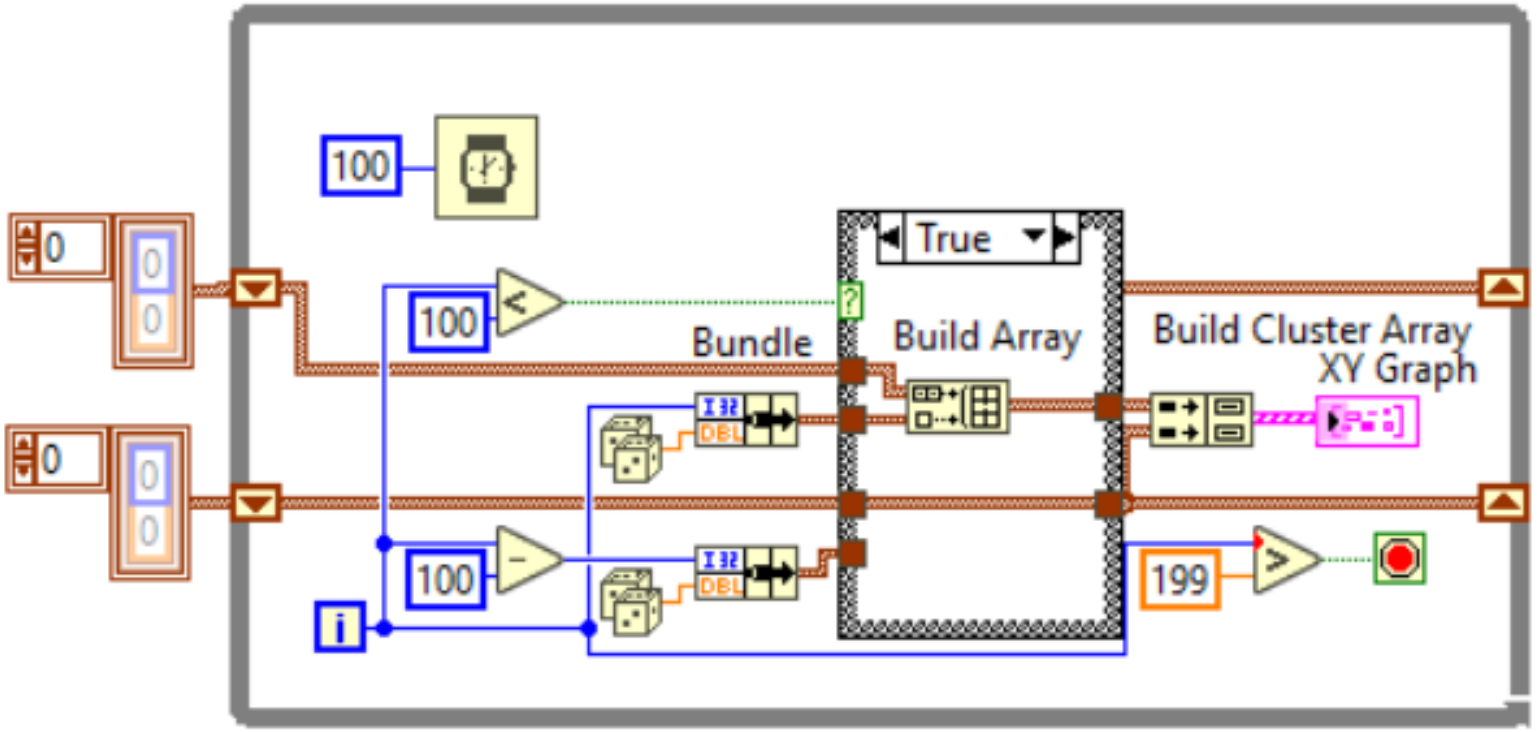
- Ha *x*,*y* pontpárokból álló tömböket bővítünk és ezeket kötegelve adjuk át az **XY Graph** modulnak, akkor két, vagy több adatsort is ábrázolhatunk egy közös diagramban
- Itt most 3, illetve 6 hozzáadásával egymáshoz képest eltoltuk a két görbét. Már esetben viszont lehet eltérő a cél, pl. annak ellenőrzése, hogy a két görbe mennyire fedi egymást



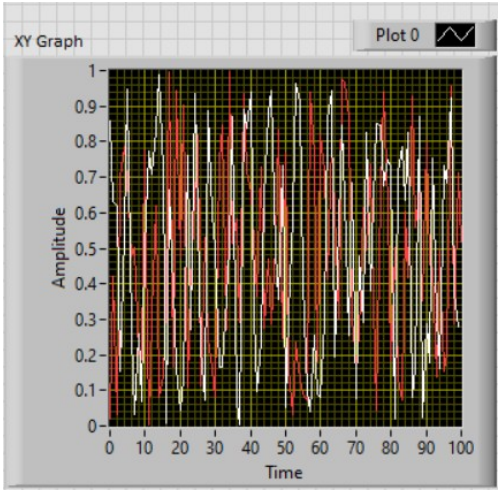


dual_plot_alternate.vi

- Az előző példához képest itt azzal bonyolítjuk a kirajzolást, hogy az első 100 ciklusban csak az első tömböt növeljük, a második 100 ciklusban pedig csak a második tömböt
- Az elágazást egy **if** blokk felhasználásával oldjuk meg

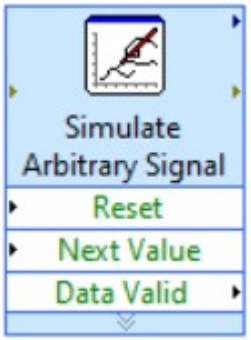


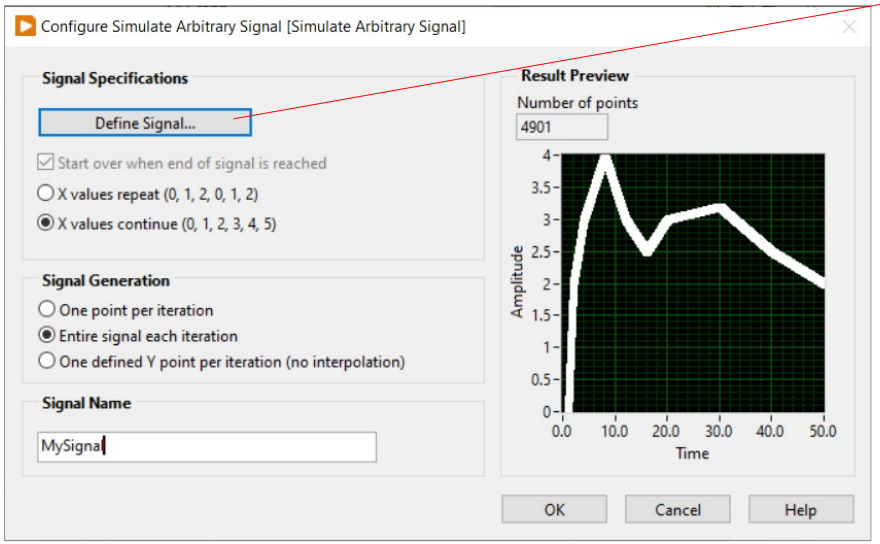
Két adatsor ábrázolása esetén természetesen a tulajdonságok külön beállíthatók

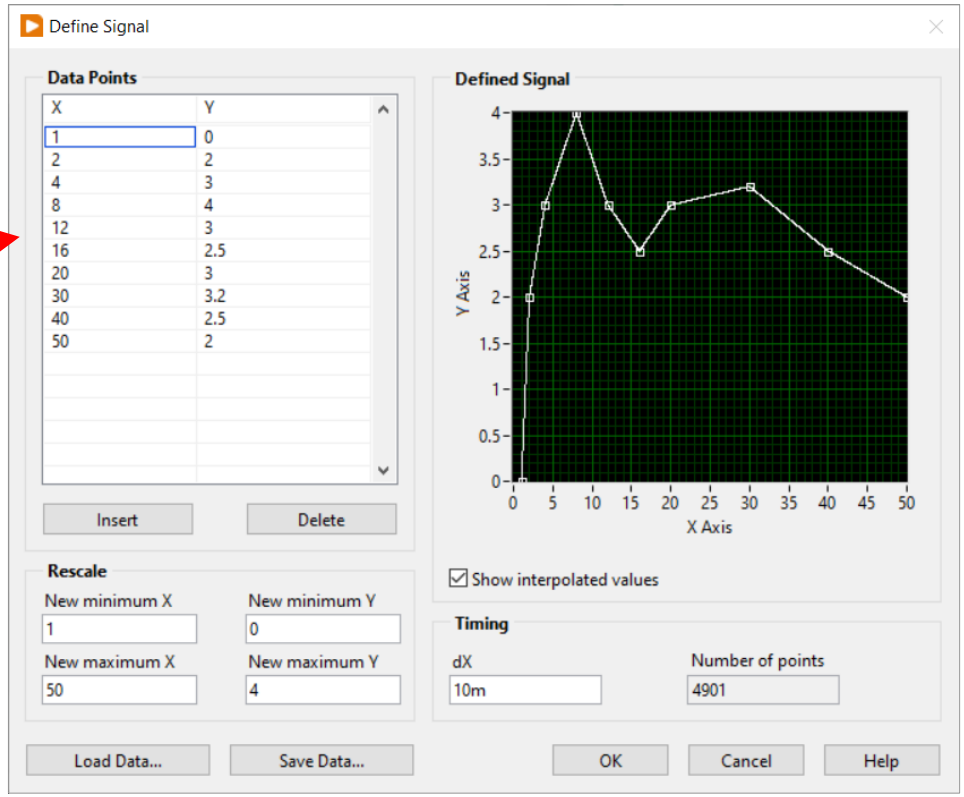


Express VI modulok

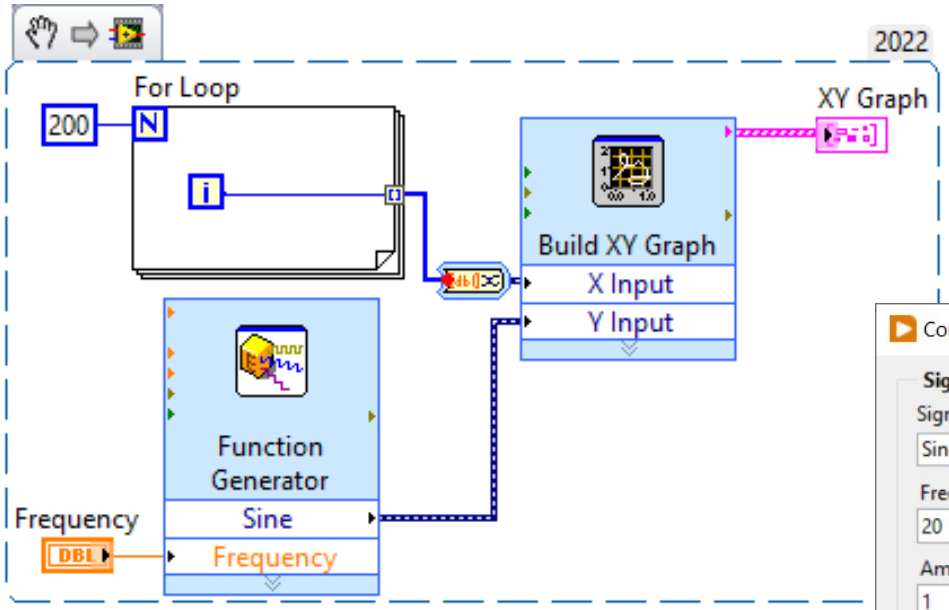
- Az **Express VI** olyan subVI, amelynek beállításait interaktívan konfigurálhatjuk egy párbeszédpanelen keresztül
- Az **Express VI**-ok kiterjeszthető modulként jelennek meg a blokkdiagramon, kék mezővel körülvett ikon formájában
- Az **Express VI-ok** fő előnye az interaktív konfigurálhatóság, könnyen használható VI-t készítünk velük
- Sajnos, futás közben nem hívható elő a dialógus ablak, s nem minden paraméter érhető el a Front panelről



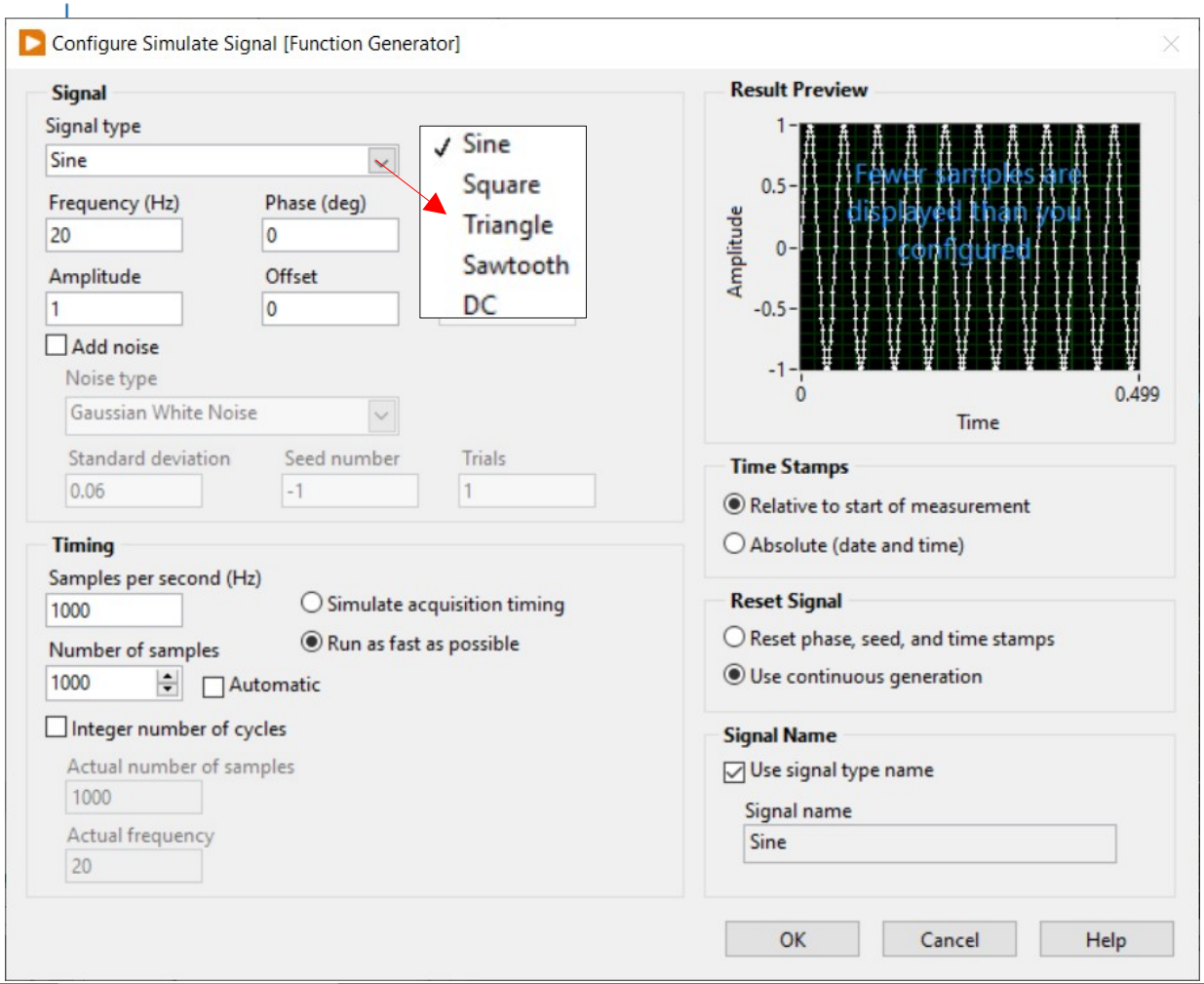




function_generator.vi



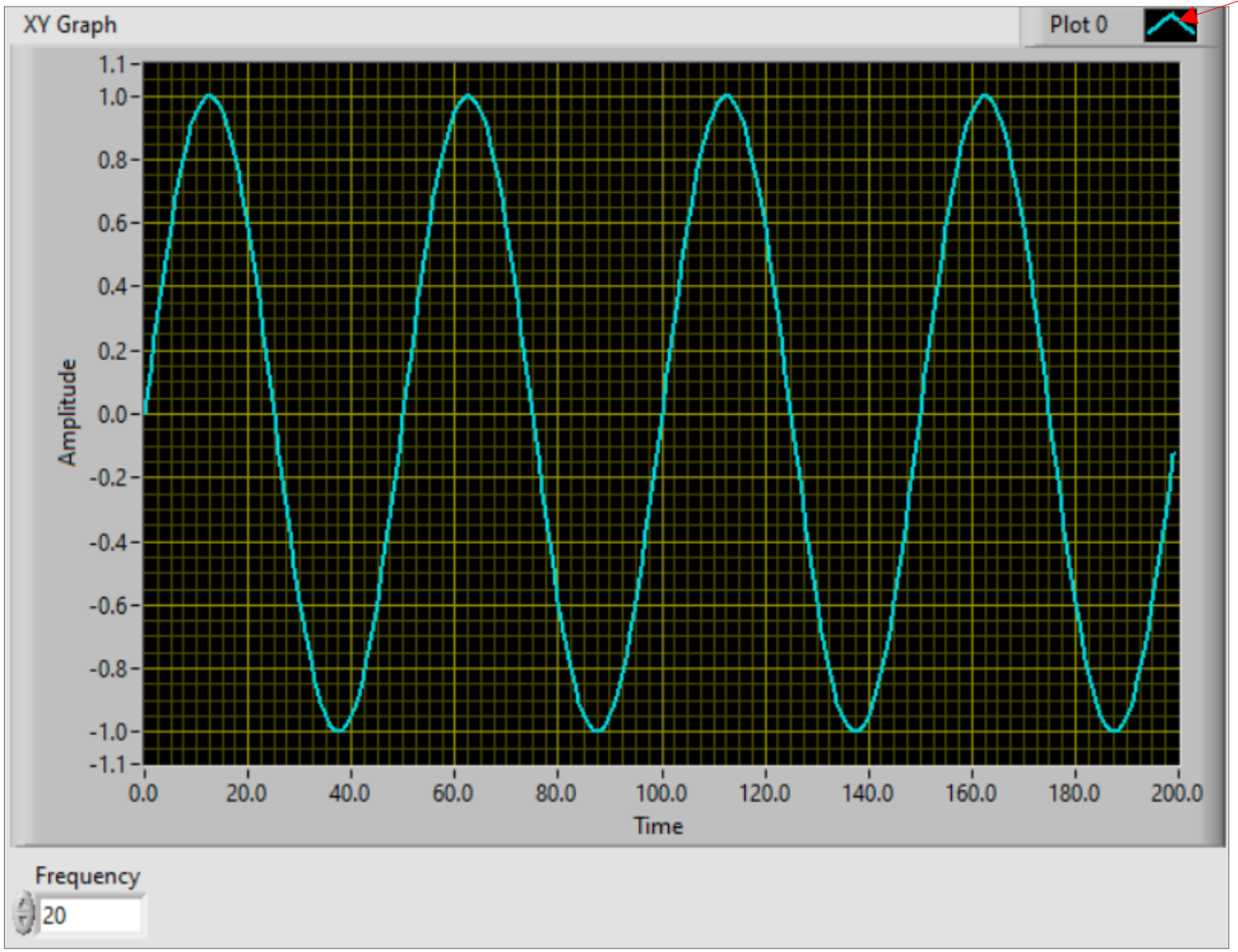
- **Express VI** modulokból például gyorsan összerakhatunk egy egyszerű jelgenerátor



- Az ***x*** értékeket itt egy **for** ciklusban állítjuk elő, a függvény-generátor pedig a dialógus dobozban konfigurálható
- A dialógus ablak dupla-kattintással nyitható meg
- A frekvencia kapott egy beállítót, így futás közben is állítható

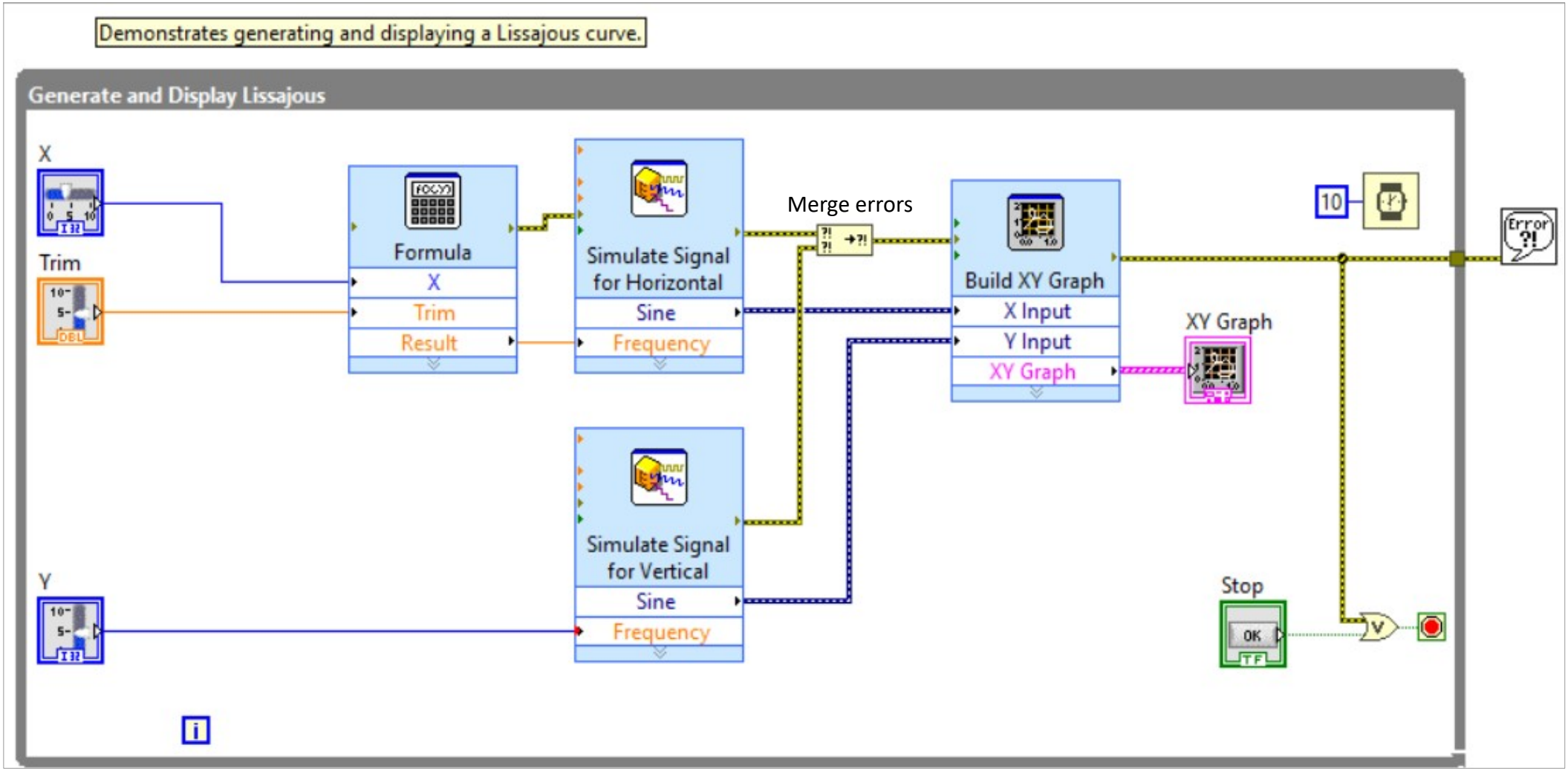
function_generator.vi

- A görbe megjelenéséét a jobb felső sarokra kattintva állíthatjuk be



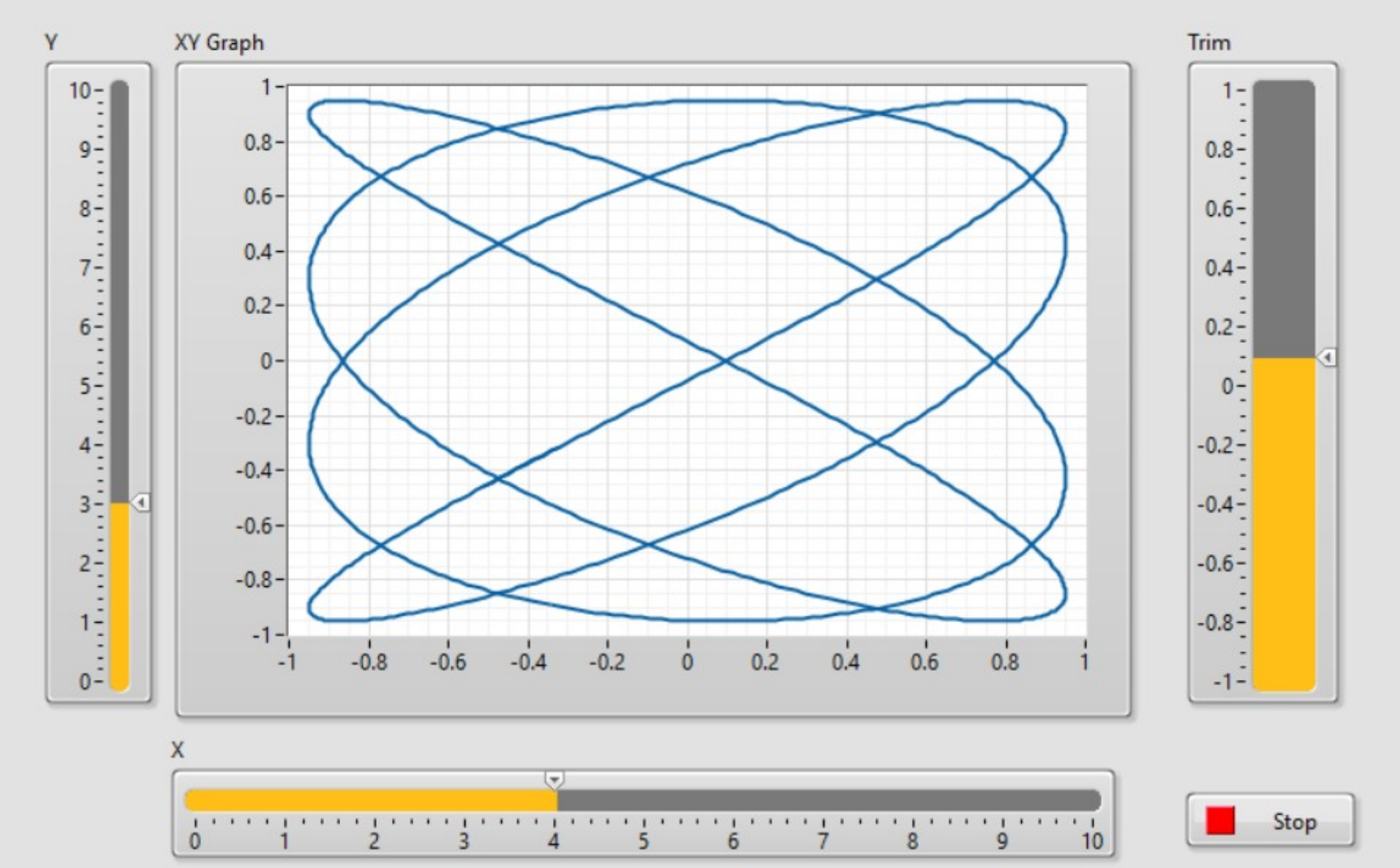
Lissajous with Express VIs.vi

- Ez a **LabVIEW** gyári mintapélda is Express VI modulokból van összerakva és a Lissajous-görbék kialakulását szemlélteti
- A Formula modul a Result = x + trim/20 kiszámítását végzi



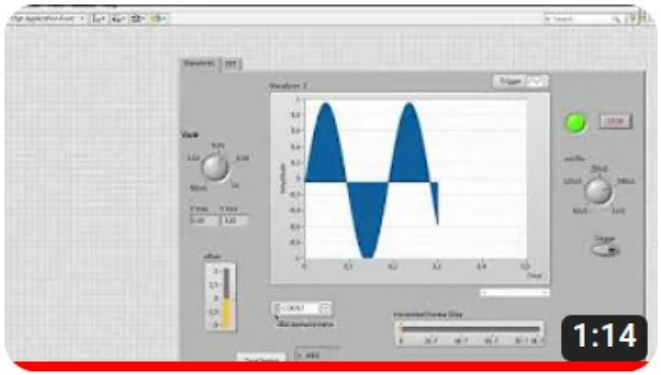
Lissajous with Express VIs.vi

- A csúszkákkal az ***x***, ill. ***y*** szinuszjelek frekvenciája állítható be



Oszcilloszkóp projekt STM32 kártyával

- A @professionalengineer317 YouTube csatornáján található a digital oscilloscope in STM32 and Labview című projekt ismertető videó, a projekt állományai pedig innen tölthetők le
- A projekt egy oszcilloszkóp-szerű virtuális műszert definiál, amely többletként a bemenő jel Fourier-transzformáltját is képes kiszámítani és megjeleníteni
- Az adatokat egy STM32 kártya küldi, melynek egyedi programja Mbed környezetben készült (a program közzétett verziója csak szimulált adatokat küld, de nem bonyolult ezeket valódi ADC konverziók eredményére cserélni (a programnak itt egy Mbed 6-ra átírt változatát mutatom be)
- A program tehát nem a LINX firmware-t használja, így az adatok karakterenkénti fogadását és számmá konvertálását a LabVIEW alkalmazásban kell megszervezni



digital oscilloscope in STM32 and Labview

mbed6-oscilloscope/main.cpp

- A kipróbáláshoz mesterségesen generált adatokat küldünk (az értelmes használathoz egy ADC konverzió eredményét kellene kiküldeni)
- Egy **STM32 NUCLEO-F446RE** kártyához **Mbed 6** projektet készítettünk az **Online Keil Studio** segítségével

```
#include "mbed.h"
#include "math.h"

#define WAIT_TIME_MS 10
float f = 0, g = 0, x = 0;
int i = 0;

int main() {
  while (1) {
    f = sin(i / 100.0);
    g = cos(i / 5.0);
    printf("%7.4f@", f/10.0 + g);
    thread_sleep_for(WAIT_TIME_MS);
    i++;
  }
}
```

Az adatokat egy-egy '@' jel terminálja

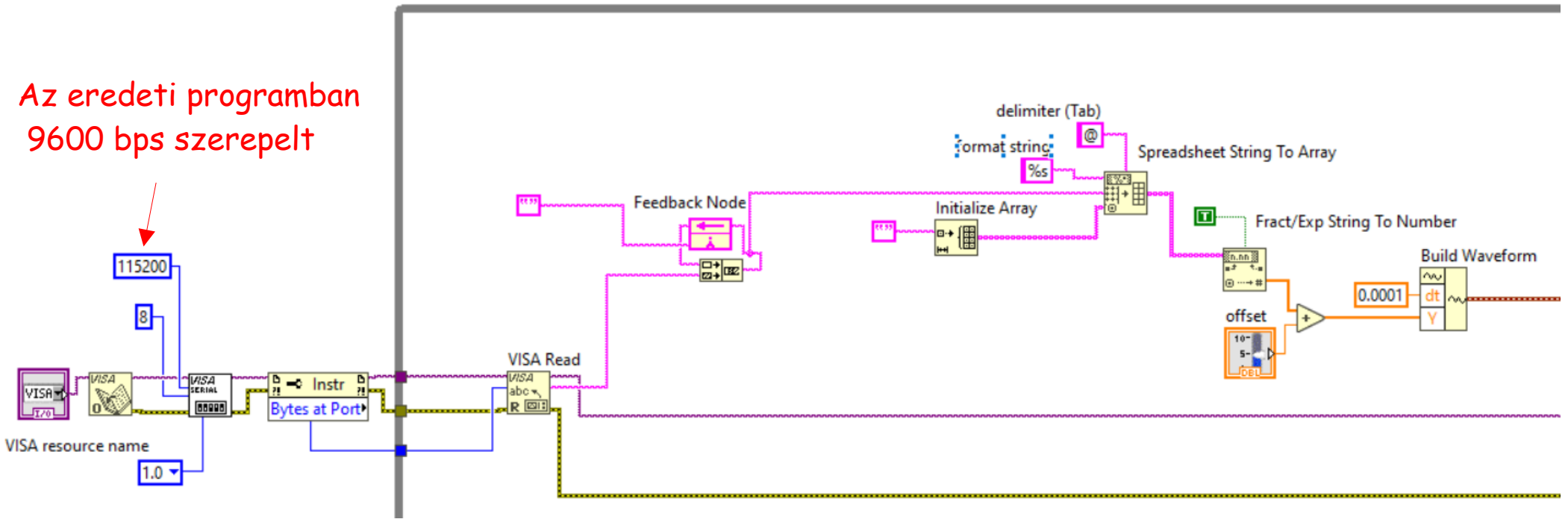
mbed6-oscilloscope/mbed-app.json

- A projektet az **mbed-os-example-blinky-baremetal** mintapéldából kiindulva hoztuk létre (tehát nincs belefordítva az RTOS)
- Az **mbed-app.json** konfigurációs állományban
  - Engedélyezni kellett a lebegőpontos kiírást
  - Az alapértelmezett 9600 bps helyett 115 200 bps átviteli sebességet állítottunk be

```
{
    "requires": ["bare-metal"],
    "target_overrides": {
      "*": {
        "target.c_lib": "small",
        "target.printf_lib": "minimal-printf",
        "platform.minimal-printf-enable-floating-point": true,
        "platform.stdio-minimal-console-only": true,
        "platform.stdio-baud-rate": 115200
      }
    }
}
```

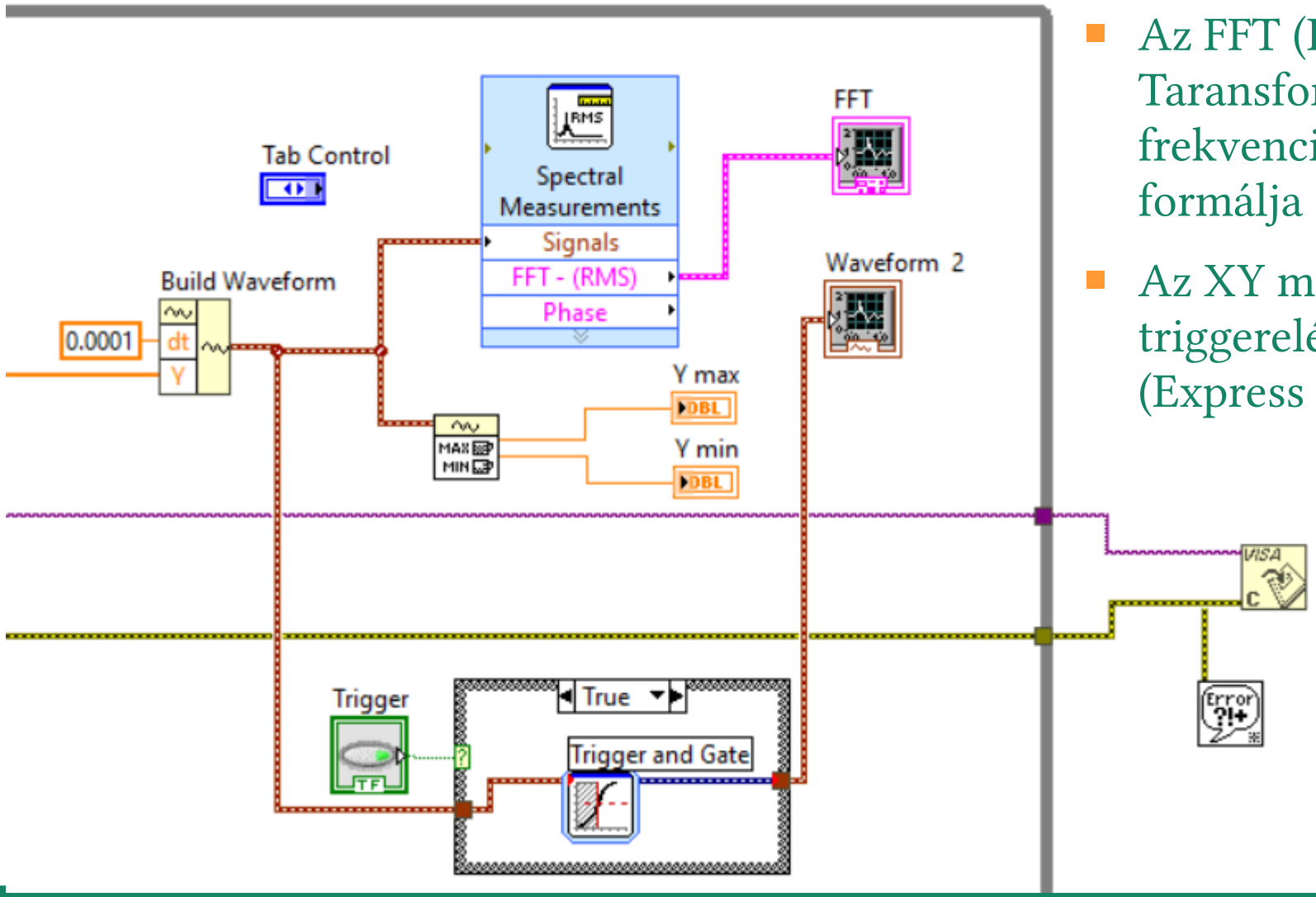
oscilloscope.vi - részlet

- **Az adatok beolvasása** VISA soros porton, karakterenként történik
- A karaktereket összefűzzük, majd a lezáró karakter (’@’) észlelésekor számmá alakítjuk (itt lehet eltolást hozzáadni)
- Ha a **use system decimal point** paraméter **T,** akkor a rendszer szerinti, ha pedig **F,** akkor mindenképp a pont lesz a tizedes elválasztó
- Az adatokból egy hullámtáblát készítünk, amit kétféle módon lehet megjeleníteni



oscilloscope.vi - részlet

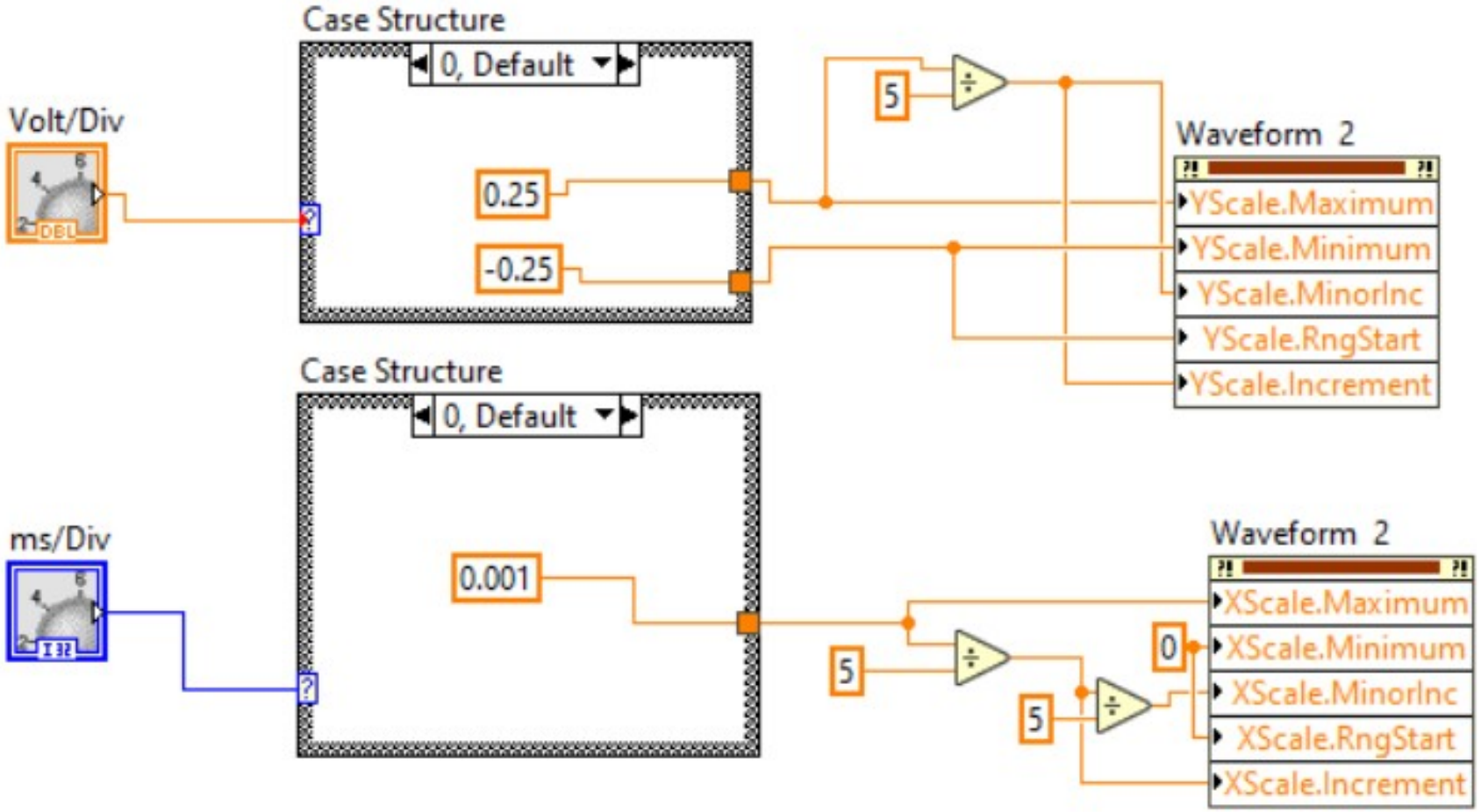
- **A kétféle megjelenítés** (XY Graph, illetve FFT) két lapon történik, melyek közül a Tab Control választja ki, hogy melyik látszódjon



- Az FFT (Fast Fourier Taransformation) a frekvencia térbe transz-formálja a hullámalakot
- Az XY megjelenítésnél triggerelési lehetőség van (Express VI funkció)

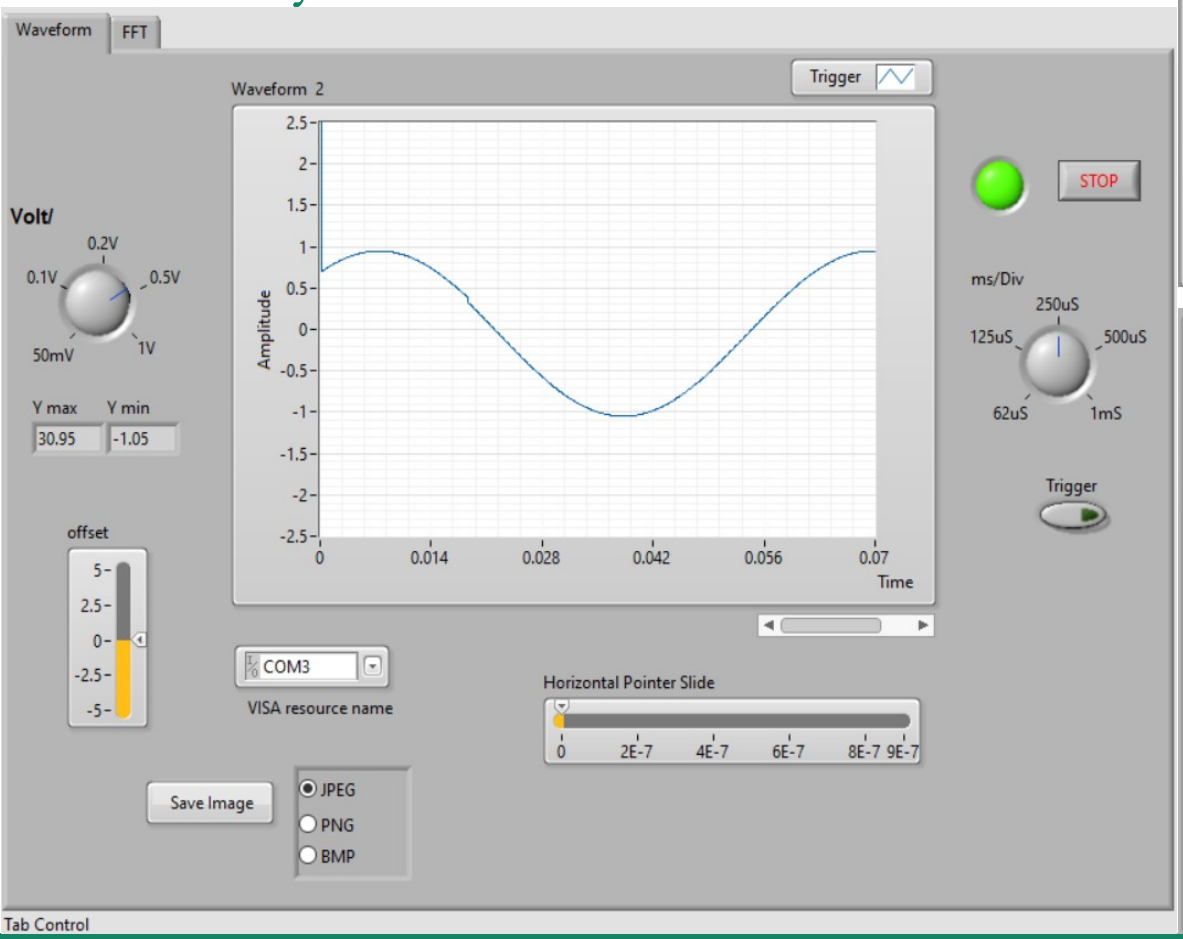
oscilloscope.vi - részlet

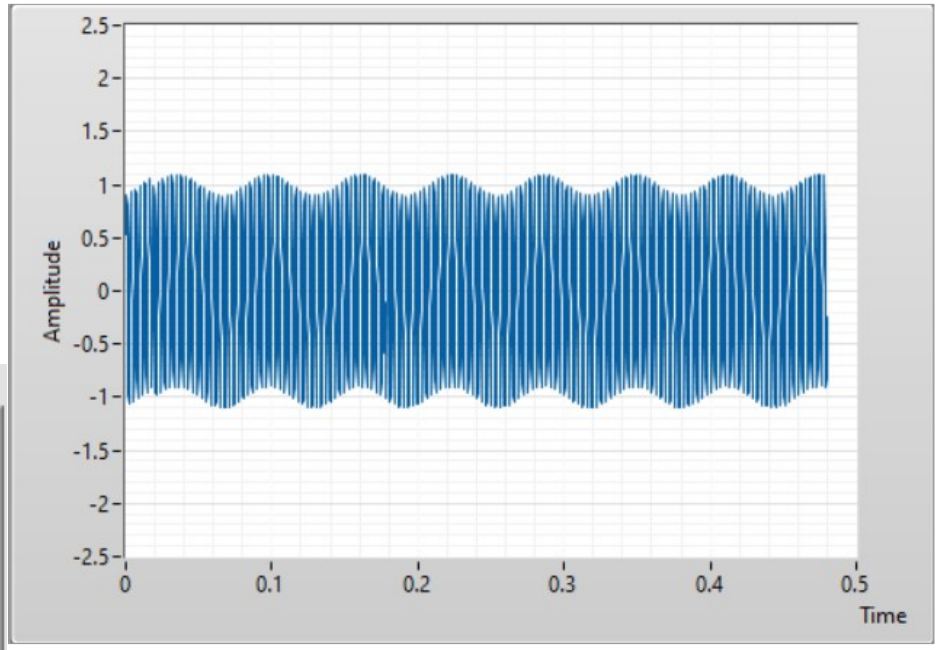
- **A megjelenítés „kezelőszervei”** ún. ***property node***-ok és ***case struktúrák*** segítségével állítják be a megfelelő paramétereket

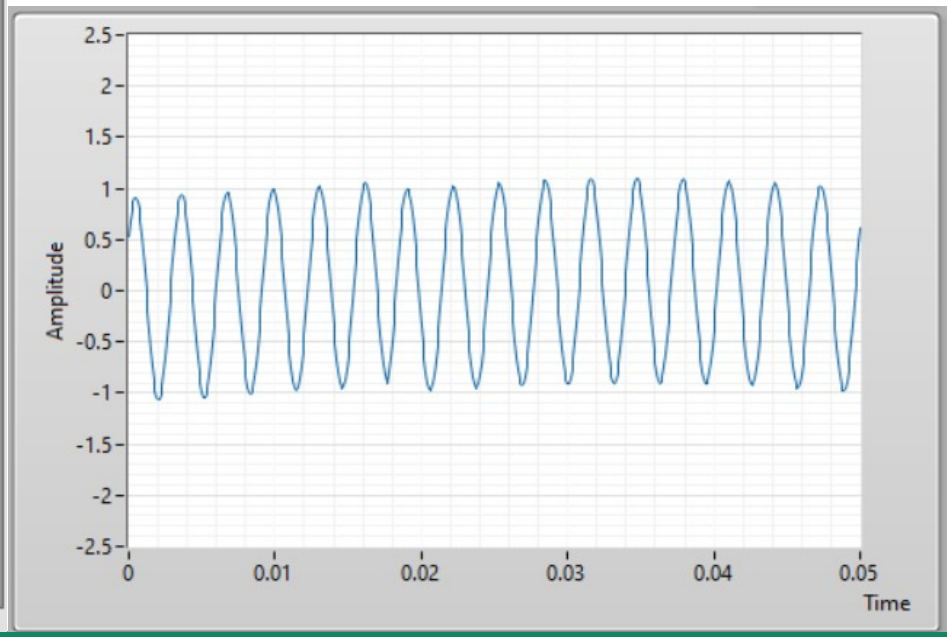


oscilloscope.vi – futási eredmény

- A beállításoktól függően az összetett jel szinusz vagy koszinusz összetevője dominál
- Az ***offset csúszka*** a függőleges eltolást szabályozza

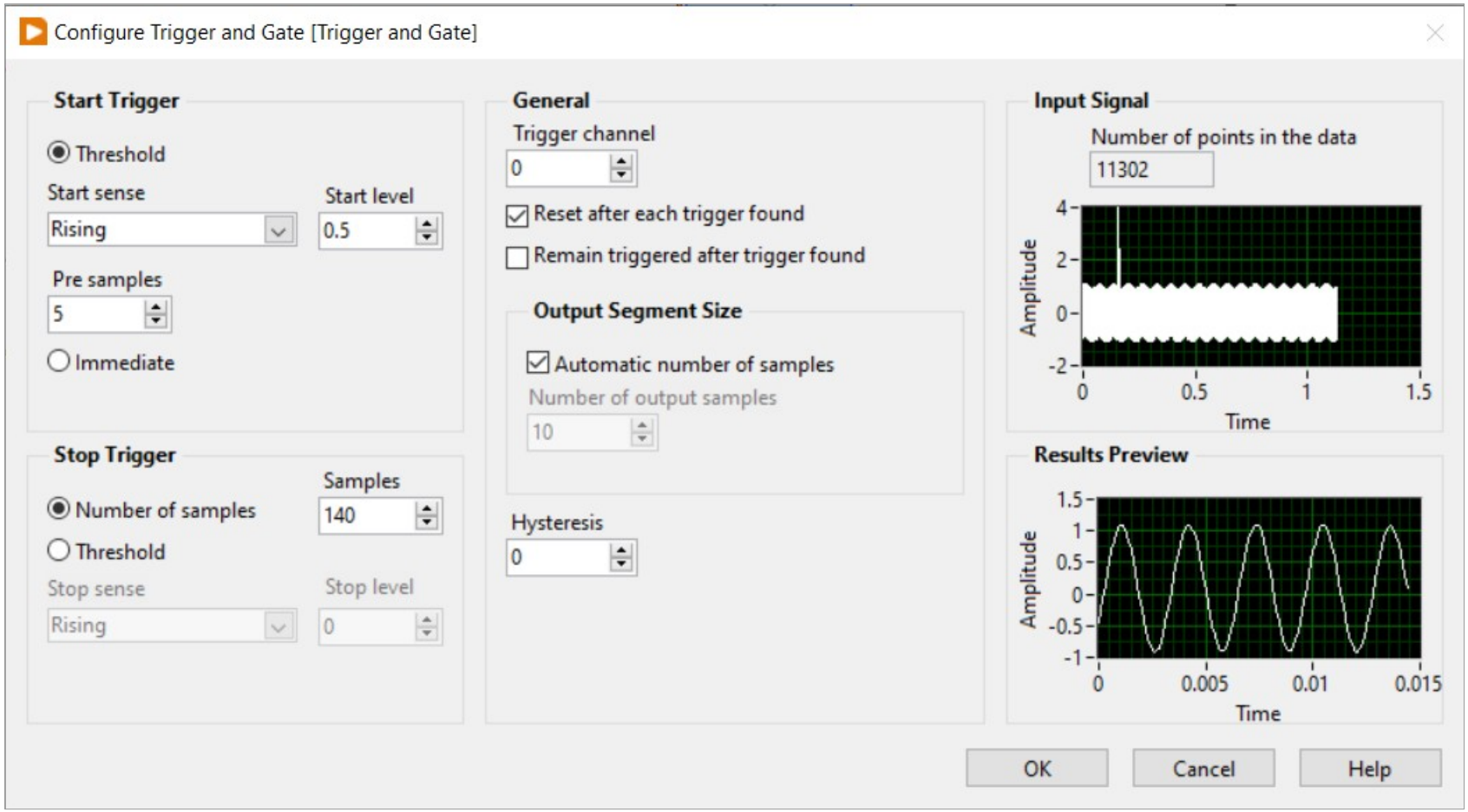






oscilloscope.vi – futási eredmény

- **A triggerelés paramétereit** csak akkro tudjuk módosítani, ha a futást leállítottuk
- A blokkdiagramban a Trigger modulra duplán kattintva nyílik meg az alábbi dialógusablak



oscilloscope.vi – futási eredmény

- **Az FFT lapon** az összetett jel frekvencia-spektruma látható (a képen a skála nem valós)