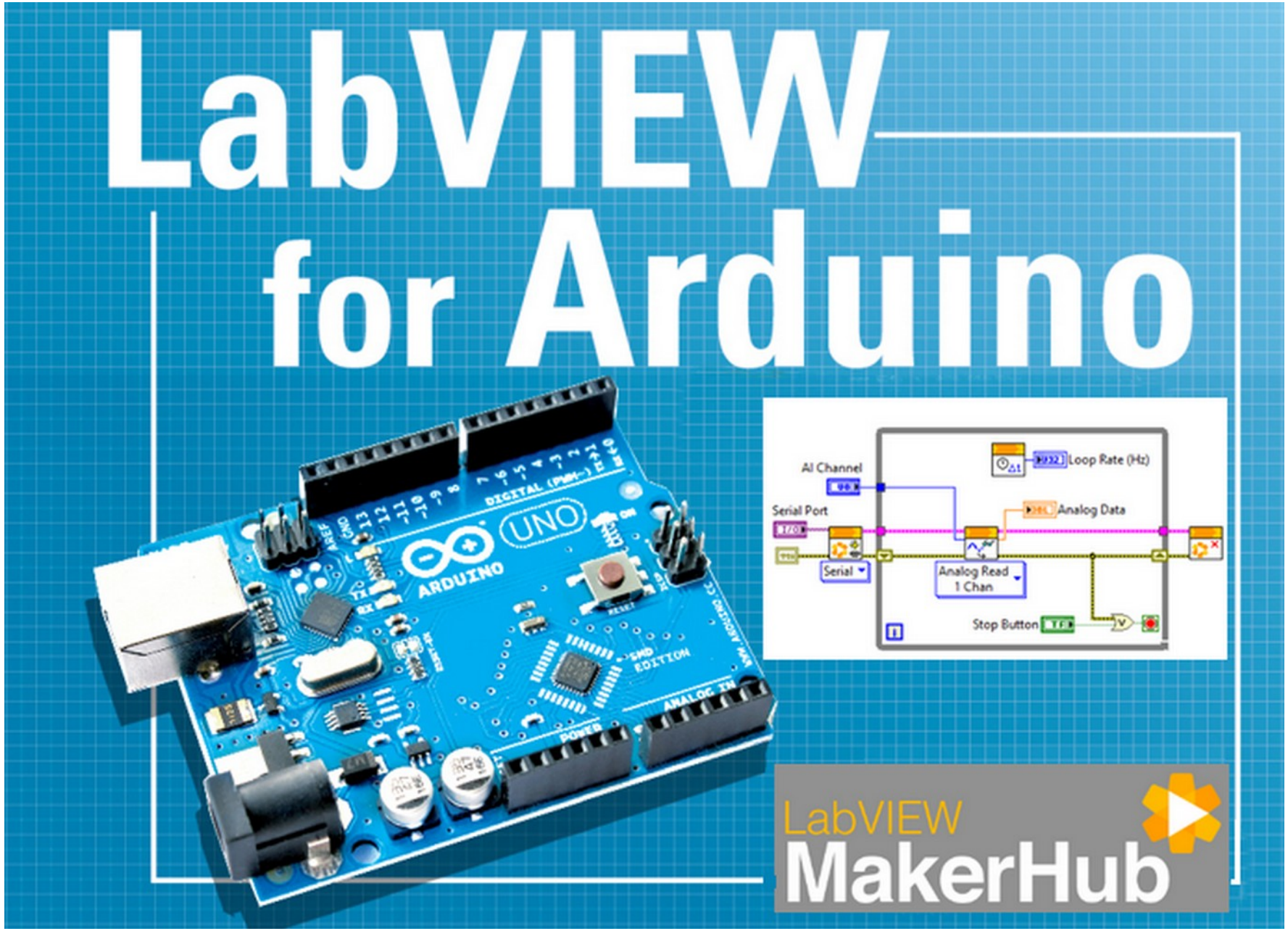
15. LabVIEW + LINX + Arduino - 5. rész

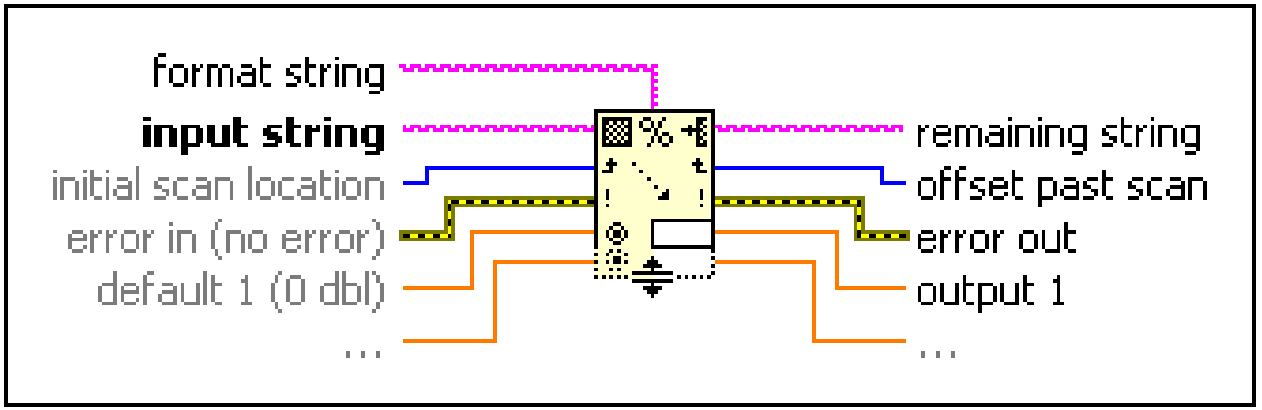


Felhasznált és ajánlott irodalom

- NI: Getting Started with Arduino and LabVIEW Community Edition
- NI: LabVIEW Documentation
- NI: Optimization VIs
- NI: Overview of Curve Fitting Models and Methods in LabVIEW
- NI: Curve Fitting Express VI
- 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)
- Nicholas J: Arduino& Labview With 2 DS18B20
- Linear Technology (Analog Devices): LTC2400 24-bit ADC datasheet

Adatok beolvasása string változóból

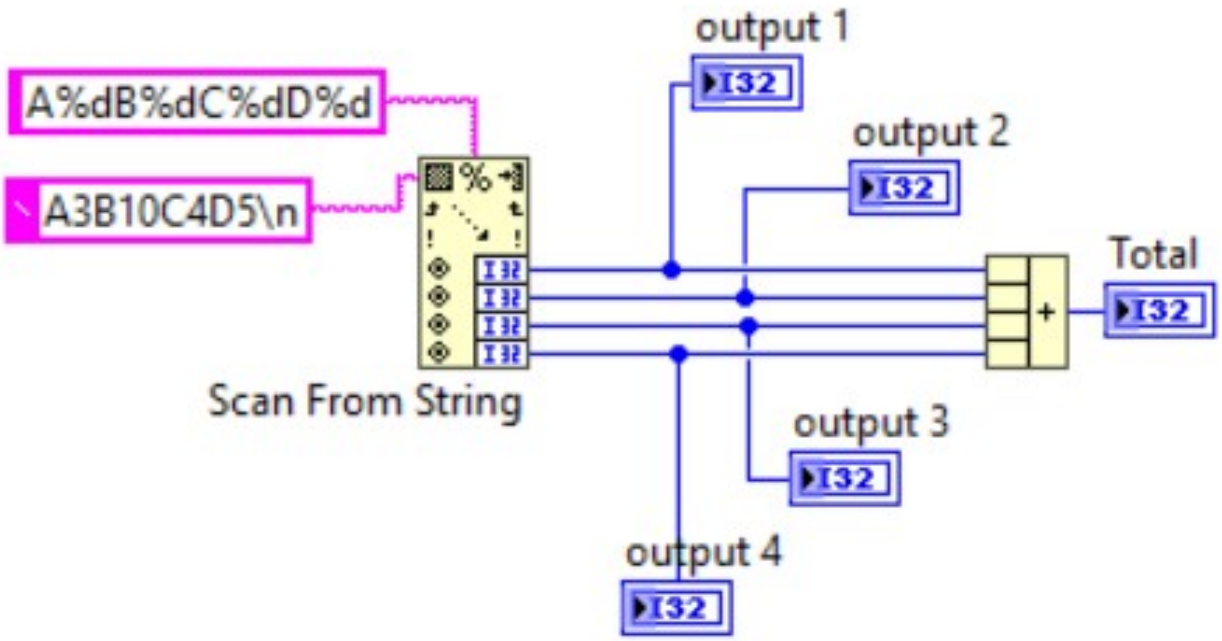
- Az adatok kinyerésere szövegből (származzon az fájlból vagy soros porti beolvasásból) többféle módszer is van, amelyek közül a **Scan From String** a C nyelvben használt **scanf()** függvényhez hasonlóan működik: a beérkező szöveget egy formátum-szöveggel hasonlítja össze
- Az adatok típusát vagy a **formátum szövegből** (ha konstans) vagy az **alapértelmezett** (default) értékekből határozza meg
- Ha a teljes stringet nem dolgoztuk fel, a maradék szöveg, illetve a hozzá tartozó offset értéke megjelenik a kimeneteken

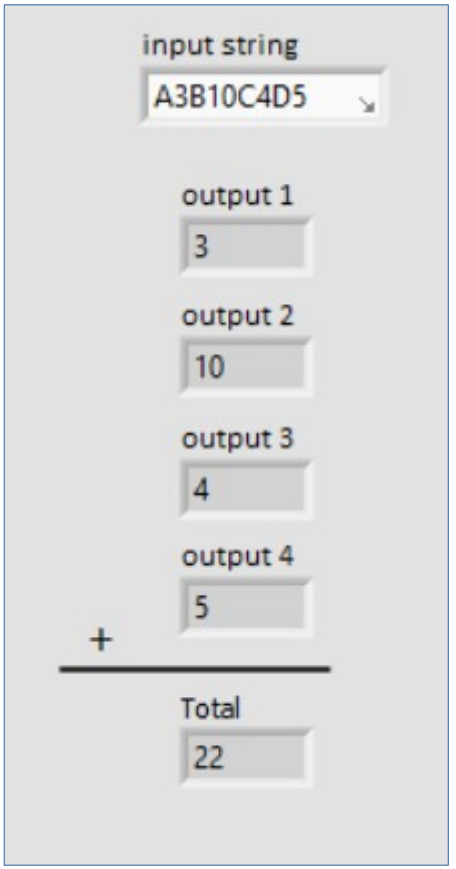


- Mivel ezt a módszert több programban is használni fogjuk, ismerkedjünk meg vele egy egyszerű példán keresztül!

scanf.vi

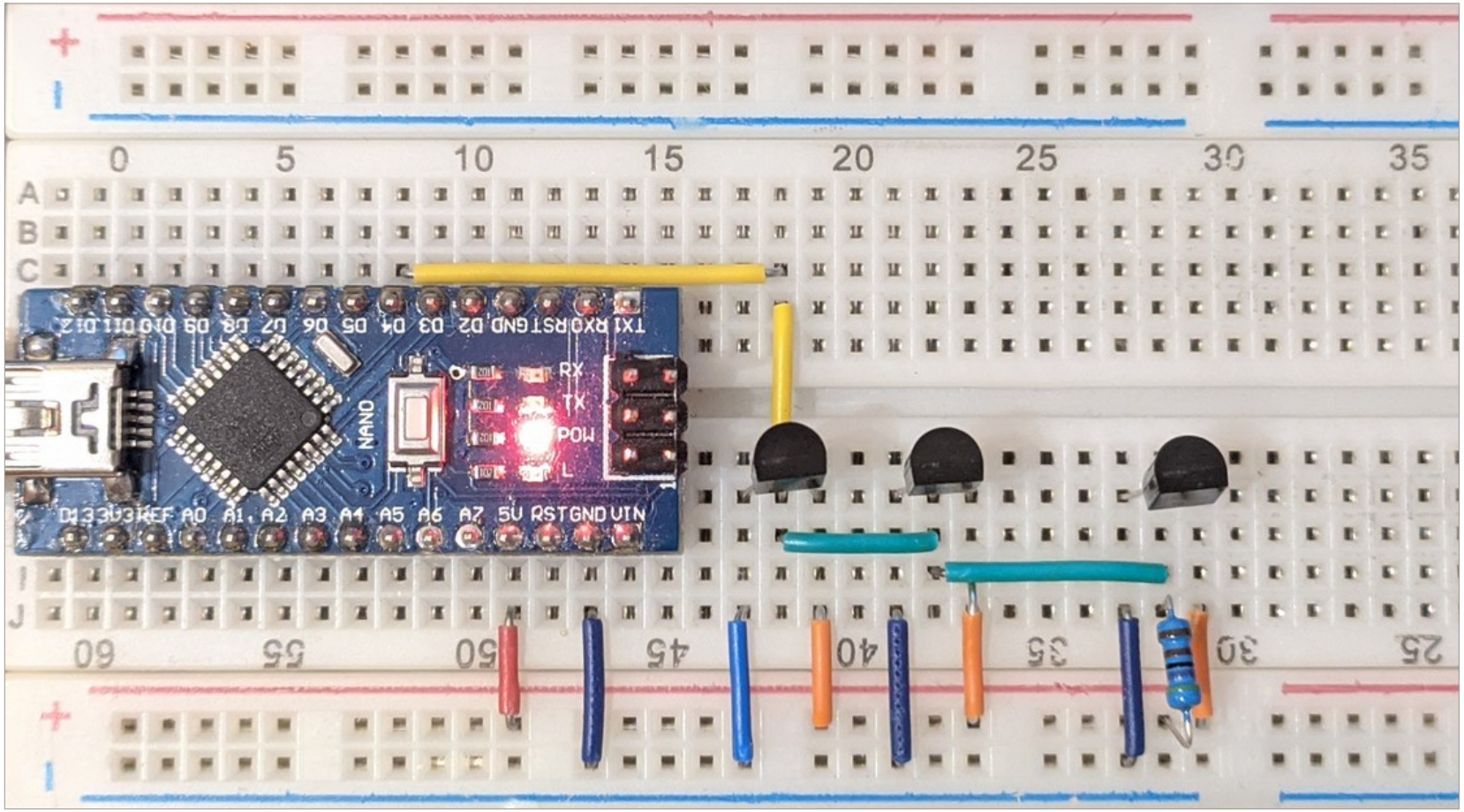
- Ebben a példában négy egész számot várunk, amelyeket A,B,C,D betűk előznek meg, ezeket szeretnénk kibontani
- A formátum: `A%dB%dC%dD%d`
- Egy bemeneti minta: `A3B10C4D5\n`
- A kinyert adatokat itt össze is adtuk, de ennek most nincs jelentősége





Hőmérés DS18B20 hőmérőkkel

- A **Scan From String** függvényt használjuk ebben a programban is, ahol három **DS18B20** hőmérő jelet fogadjuk a soros porton
- A hőmérőket egy **Arduino nano** olvassa ki (a **D4** lábra van kötve az 1-wire busz) és küldi ki az adatokat a soros porton 9600 bps sebességgel



thermometers.ino

- Az **Arduino** programban a legegyszerűbb lekérdezési módot használtuk. A **loop** függvényben még egy időzítés is felférne...

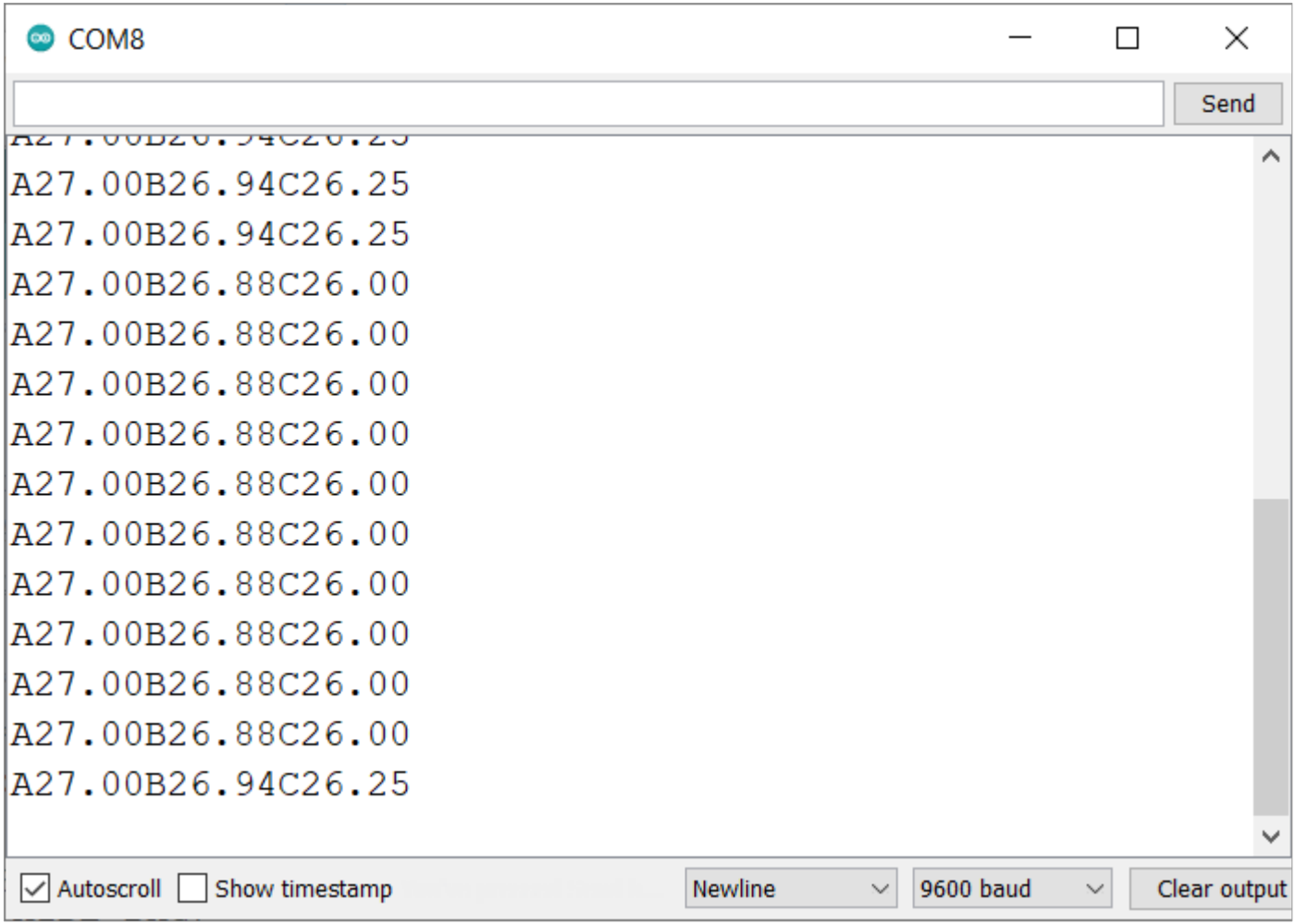
```
#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 4   // A D4 lábra kötjük az 1-wire buszt
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

void setup(void) {
  Serial.begin(9600);
  sensors.begin();
}

void loop(void) {
  sensors.requestTemperatures();
  String temp1 = String(sensors.getTempCByIndex(0),2);
  String temp2 = String(sensors.getTempCByIndex(1),2);
  String temp3 = String(sensors.getTempCByIndex(2),2);
  String buffer = "A"+temp1+"B"+temp2+"C"+temp3;
  Serial.println(buffer);
}
```

Két tizedesre kérjük a kiírást

Összefűzzük a kiírandó szöveget

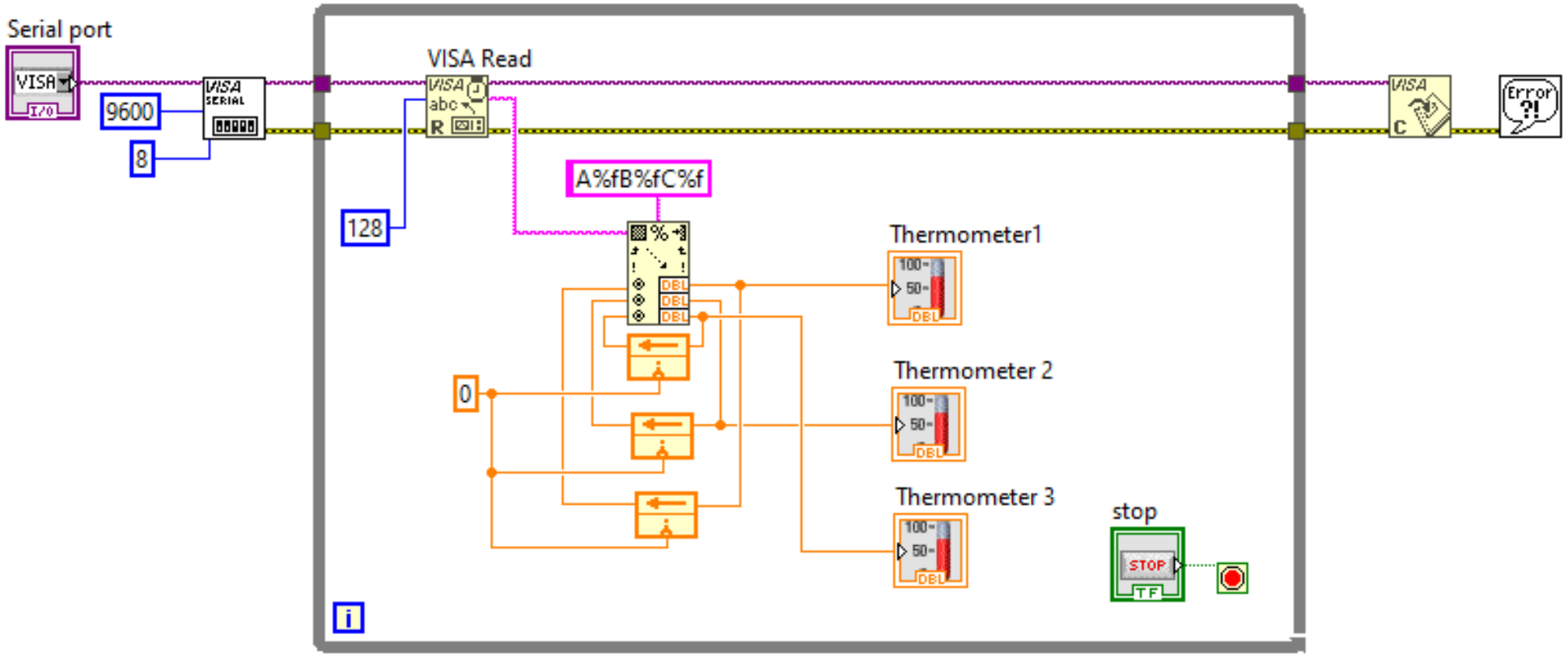
thermometers.ino futtatása

- Az adatokat két tizedesjeggyel írjuk ki, s a szenzorok adata elé egy-egy azonosító betűt is fűztünk



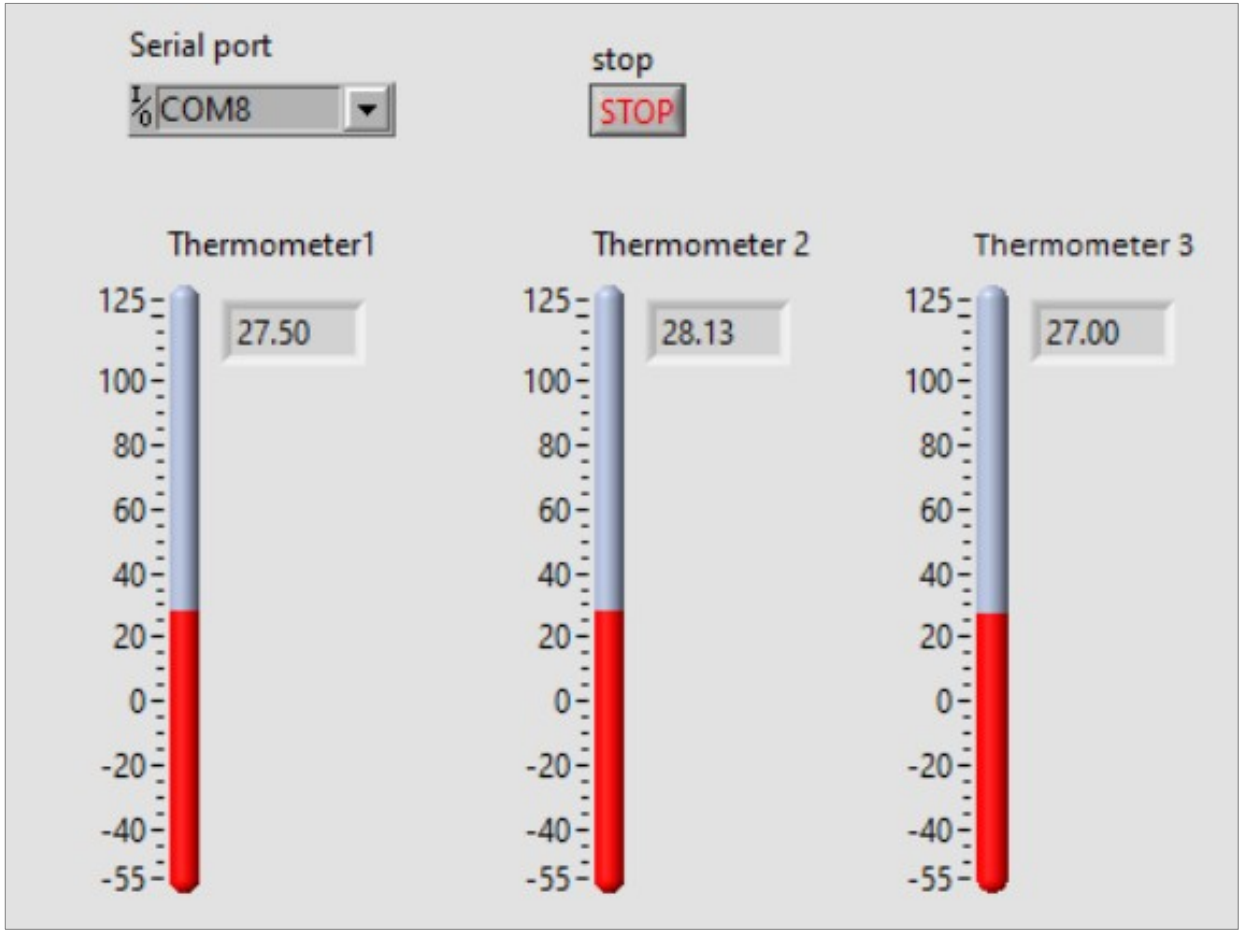
thermometers.vi

- A beolvasást a **VISA Read** függvénnyel végezzünk, a sorvége jelig, ami alapértelmezetten a soremelés karakter (a bufferméret itt 128)
- A szövegsor értelmezését visszacsatolással (feedback) is megbonyolítottuk, ami ciklusokon átívelve megőrzi az előző értéket (itt a **default bemenetekhez** használtuk fel ezeket)



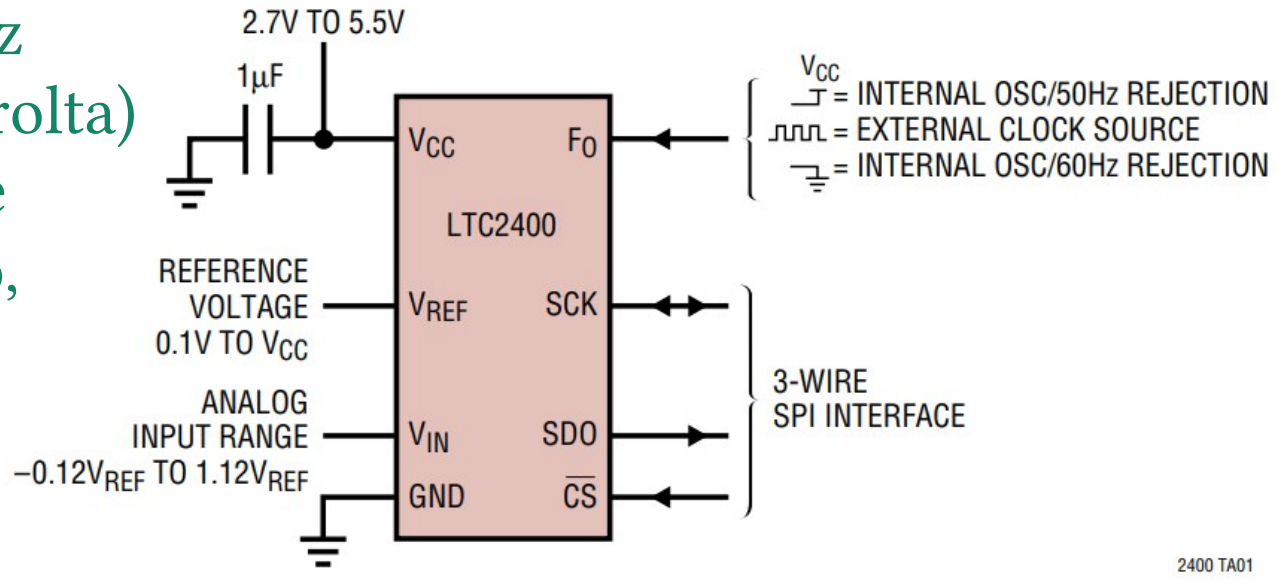
thermometers.vi futtatása

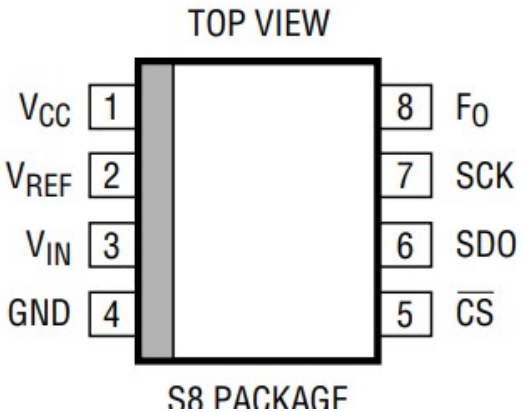
- A A program indítása előtt csatlakoztatni kell az **Arduino** kártyát és ki kell választani a soros portot a **Serial port** nevű vezérlőben
- Töredéksor esetében a program hibára fut, ezt még javítani kellene



Az LTC2400 24-bites ADC

- A Linear Technology (az Analog Devices felvásárolta) LTC2400 típusú ADC-je egyszerűen használható, nagy felbontású ADC-t kínál





LINEAR TECHNOLOGY

LTC2400

24-Bit µPower
No Latency ΔΣ™ ADC in SO-8

FEATURES

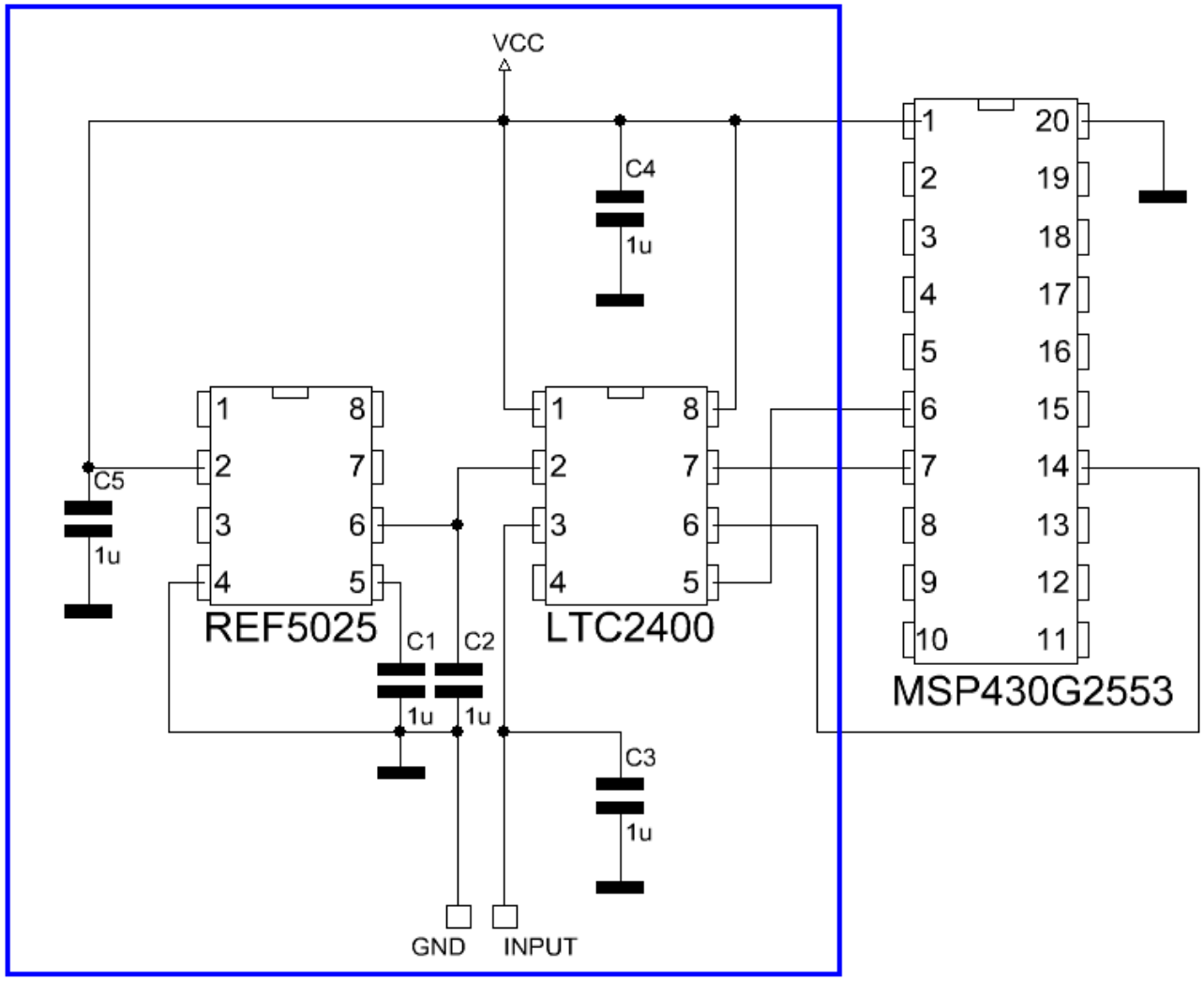
- 24-Bit ADC in SO-8 Package
- **4ppm INL, No Missing Codes**
- **4ppm Full-Scale Error**
- **Single Conversion Settling Time for Multiplexed Applications**
- **0.5ppm Offset**
- **0.3ppm Noise**

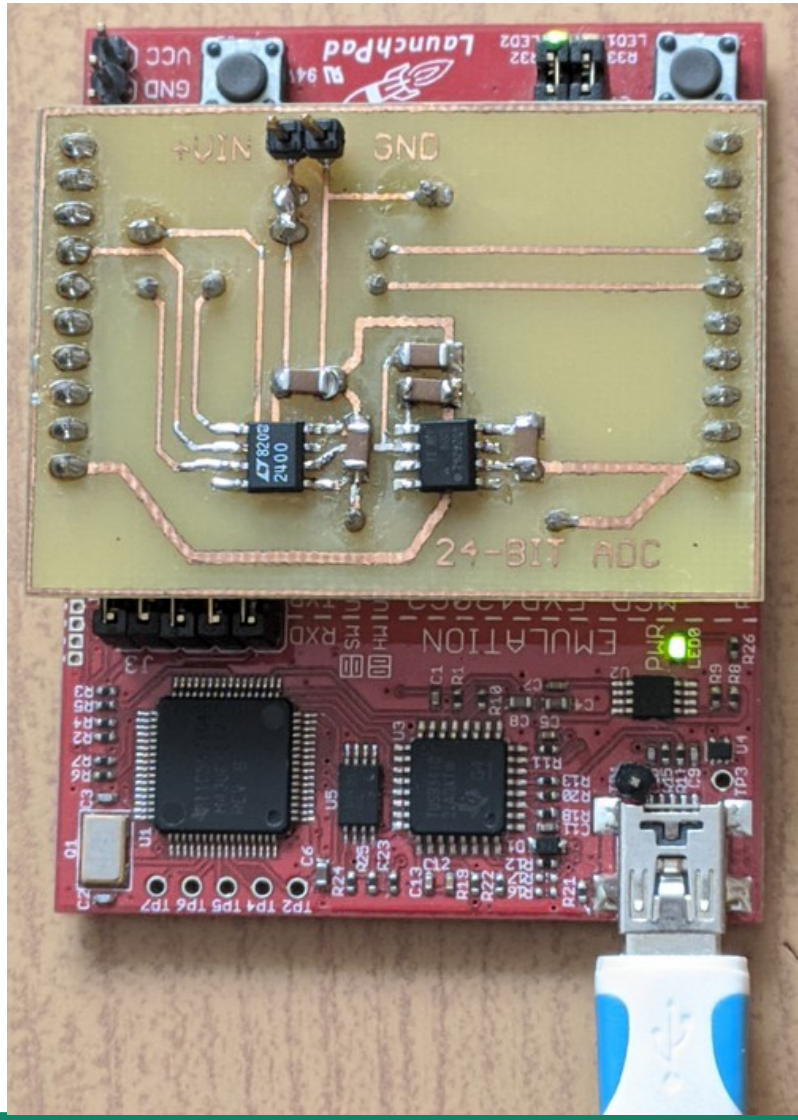
DESCRIPTION

The LTC®2400 is a 2.7V to 5.5V micropower 24-bit converter with an integrated oscillator, 4ppm INL and 0.3ppm RMS noise. It uses delta-sigma technology and provides single cycle settling time for multiplexed applications. Through a single pin the LTC2400 can be configured for better than 110dB rejection at 50Hz or 60Hz ±2%, or it can be driven by an external oscillator for a user

ADC24 fedlap MSP430 Launchpad kártyához

- Az LTC2400 ADC és egy 2.5 V-os referencia egy MSP430 Launchpad kártyához csatlakozik (VCC = 3.5V, Vref = 2.5 V)





ADC24bit/main.c (részlet)

A program IAR Embedded Workbench for MSP430 fejlesztői környezetben készült 2012-ben

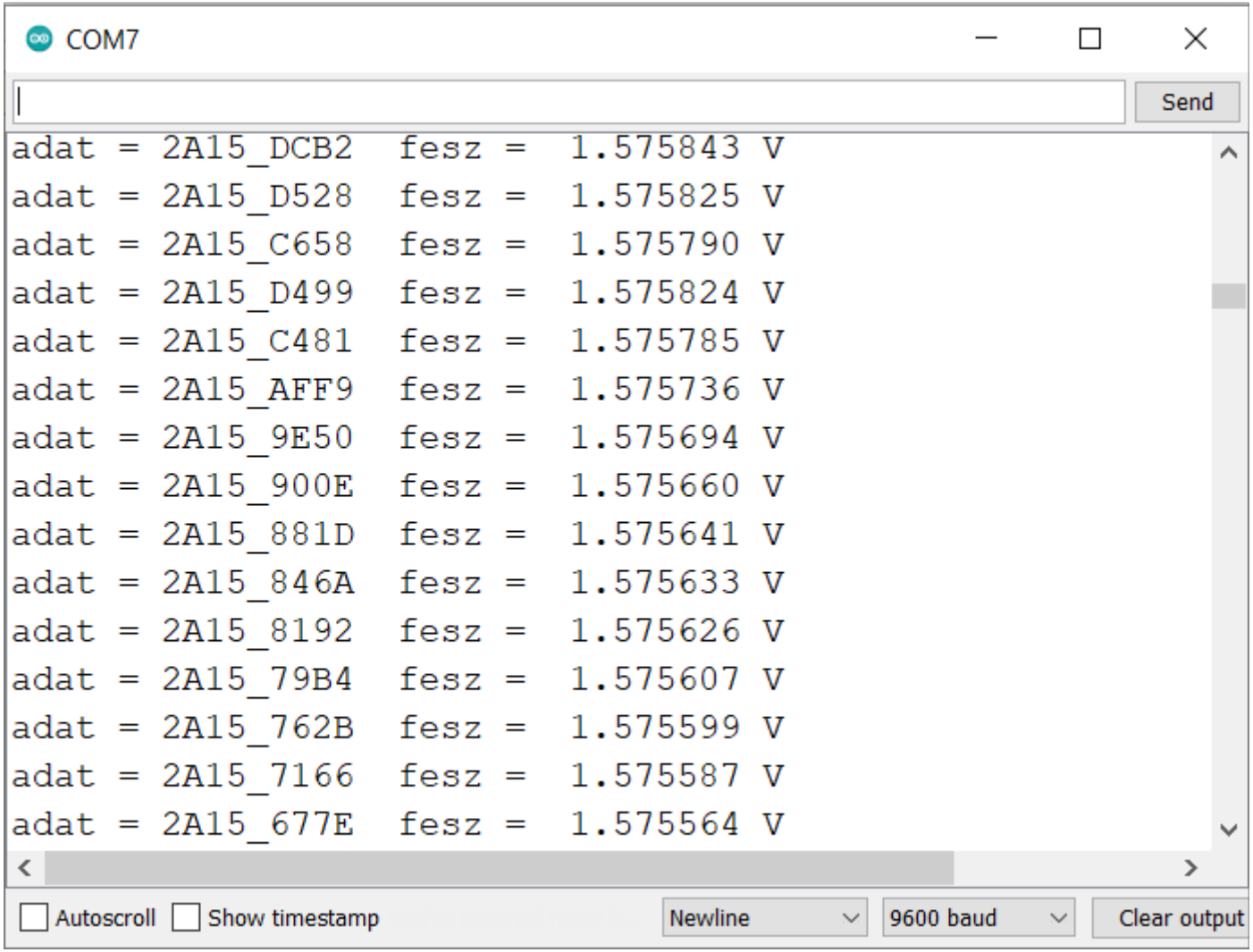
```
#include <msp430.h>
#include <stdint.h>
#define LED1       BIT0
#define TXD        BIT1                 // TXD a P1.1 lábon
#define CSEL       BIT4                 // SPI Chip Select P1.4
#define SCK        BIT5                 // SPI Clock       P1.5
#define SDI        BIT6                 // SPI Data input  P1.6
uint32_t data;
uint16_t d1,d2;
uint8_t counter,
        adc_sign,                       // elõjel bit tárolója
        adc_ext;                        // kiterjesztett tartomány jelzõje
int32_t u;
uint64_t w;

int main( void ) {
  WDTCTL = WDTPW + WDTHOLD;             // Letiltjuk a watchdog idõzítõt
  DCOCTL = CALDCO_1MHZ;                 // DCO a gyárilag kalibrált
  BCSCTL1 = CALBC1_1MHZ;                // 1 MHz-es frekvenciára
  P1DIR |= TXD+CSEL+SCK;                // TXD, CSEL és SCK digitális kimenet
  P1DIR &= ~(SW2+SDI);                  // SW2 és SDI legyenek bemenetek
  P1OUT |= TXD+SW2;                     // TXD=mark, SW2 felhúzás
  P1REN |= SW2;                         // Belsõ felhúzás engedélyezés
```

ADC24bit/main.c (részlet)

```
  while(1)  {
    P1OUT &= ~BIT4;                     // CSEL aktíválás
    while(P1IN & SDI);                  // Várakozás, ha az ADC foglalt
    Data = 0; counter = 32;
    while(counter--) {
      data = data << 1;                 // adatbitek léptetése balra
      P1OUT |= SCK;                     // órajel aktiválás
      if(P1IN & SDI) data++;            // soron következõ adat bit beolvasása
      P1OUT &= ~SCK;                    // órajel alaphelyzetbe állítás
    }
    P1OUT |= CSEL;                      // CSEL alaphelyzetbe állítás
    d1 = data>>16;
    d2 = data & 0xFFFF;
    adc_sign = !(d1&0x2000);            // Elõjel a 29 bit negáltja
    adc_ext = (d1>>12)&1;               // Extrém érték jelzése
    w = data & 0x1FFFFFFFUL;            // 28+1 bit a konverzió eredménye
    if(adc_sign) { w |= 0xFFFFFFFE0000000UL;  }
    w = w*2500000UL;                    // 2 500 000 uV
    u = (w>>28);                        // divide by 2^28 (ADC resolution)
    sw_uart_puts("adat = ");    sw_uart_out4hex(d1);
    sw_uart_putc('_');          sw_uart_out4hex(d2);
    sw_uart_puts("  fesz = "); sw_uart_outdec(u,6);
    sw_uart_puts(" V");
    if(adc_ext) sw_uart_puts(" ***");
    sw_uart_puts("\n");
  }
}
```

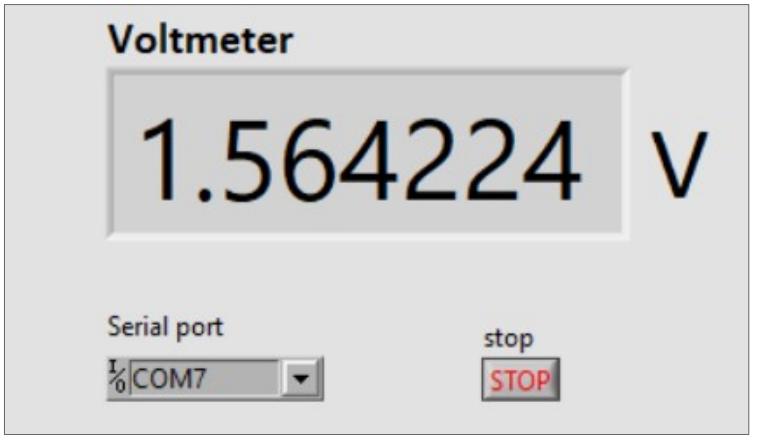
ADC24bit futási eredmény

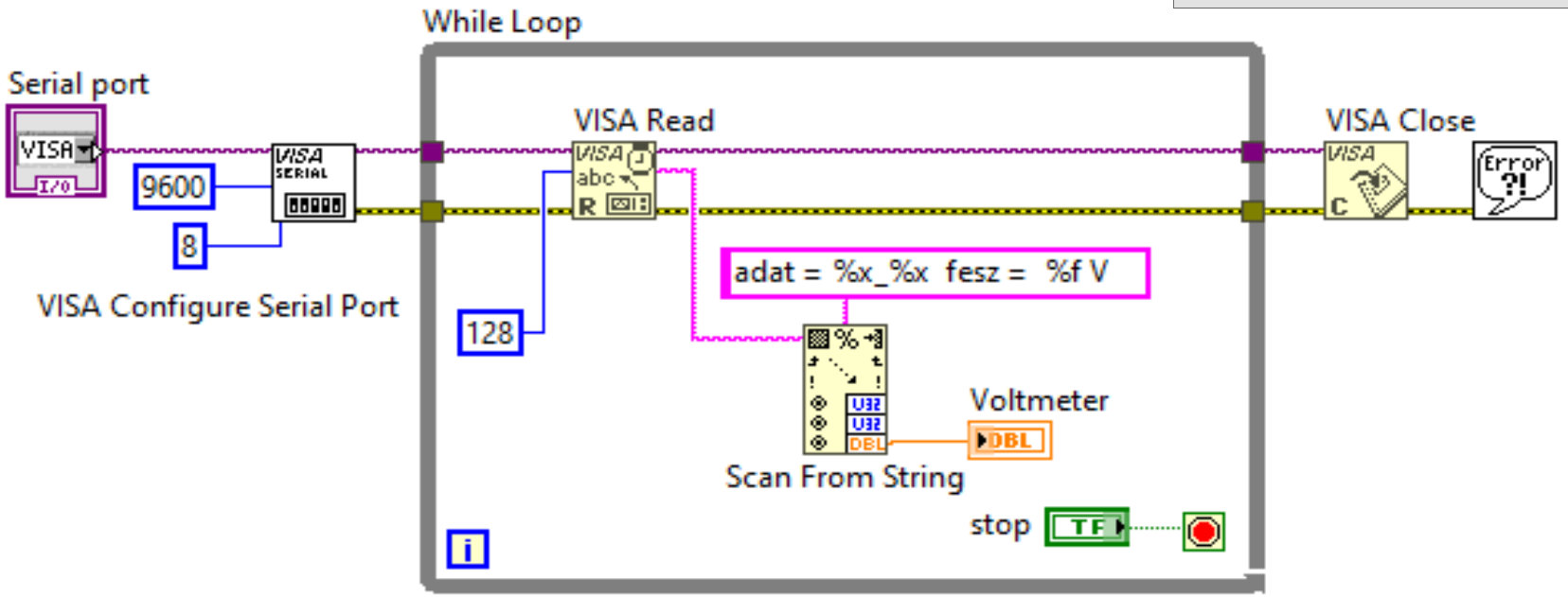
- Egy tipikus futási eredmény az ábrán látható



ADC24.vi: virtuális voltmérőt készítünk

- A **Thermometers.vi** programnál látott funkciókkal és modulokkal könnyen összerakhatunk egy voltmérőt is, az előző oldalakon bemutatott 24-bites ADC adatait fogadva
- A kijelzőt nagy fontméretben és 6 tizedesre konfiguráljuk (mikrovoltos felbontás)



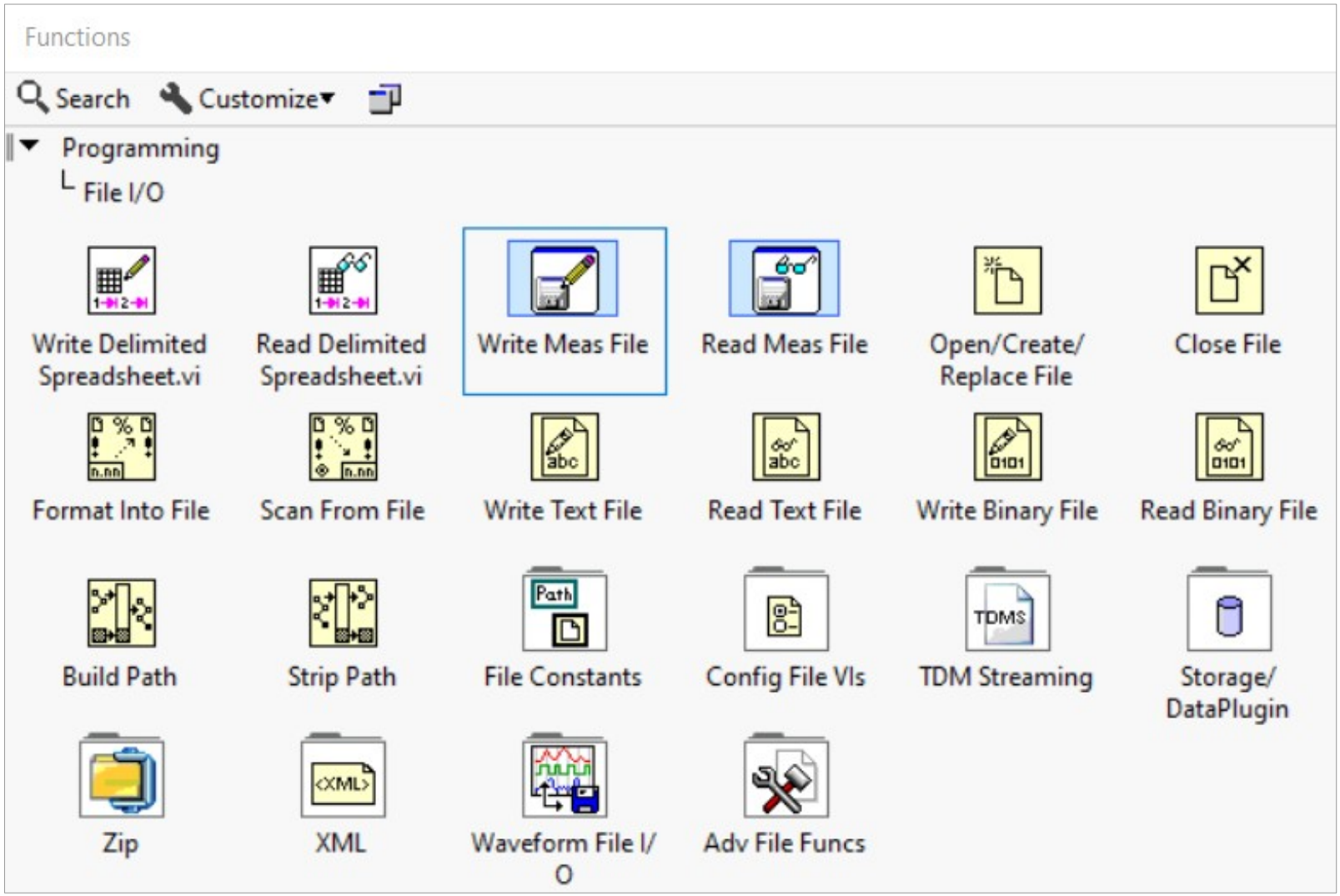


File I/O

- A fájlműveletek során adatokat olvasunk fájlból, vagy írunk fájlba, s ez többnyire három lépésben történik:
  - Fájl megnyitása
  - Adatok olvasása vagy írása
  - Fájl bezárása
- A **LabVIEW File I/O** palettája és azok alpalettái sokféle VI-t kínálnak a fájlműveletek megvalósításához, amelyek három különböző szinten biztosítják a műveletek megvalósítását
  - A magas szintű VI-ok többnyire csak egy fájltípust írnak/olvasnak, de mindhárom fenti lépést elvégzik, s ha kell, fájl dialógusablakot is nyitnak
  - A középszintű fájlkezelő VI-ok esetén az egyes műveletek VI-ait nekünk kell elhelyzni a blokkdiagramban, cserébe a fájlműveletek finomabb behangolására van lehetőségünk
  - A haladó szintű fájlkezeléshez használható VI-okat az **Advanced File Functions** alpalettán találjuk meg

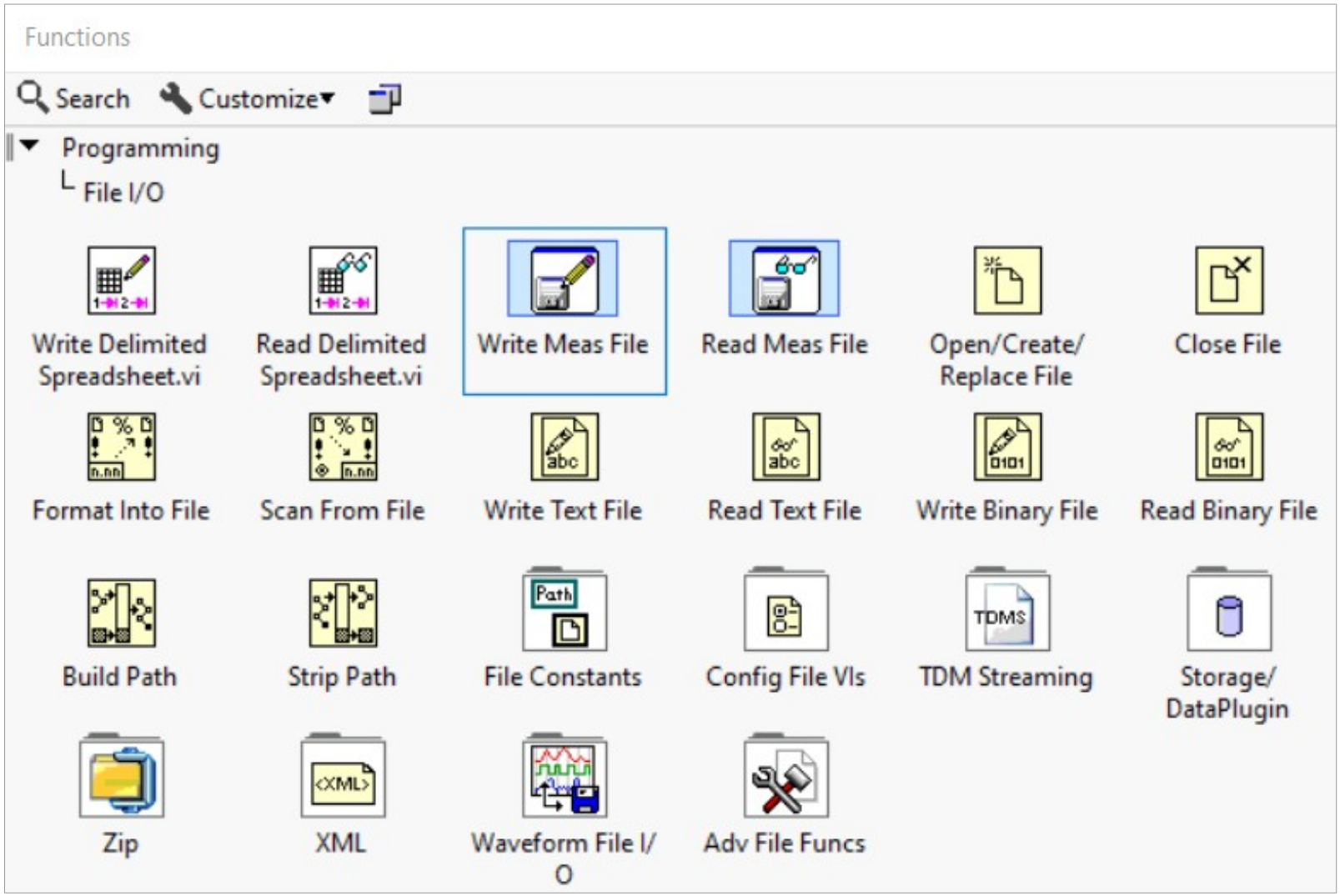
A File I/O paletta

- A fehér hátterű ikonok magasszintű, a sárga ikonok közepes szintű műveleteket képviselnek, a kék hátterűek pedig **Express VI**-ok



A File I/O paletta

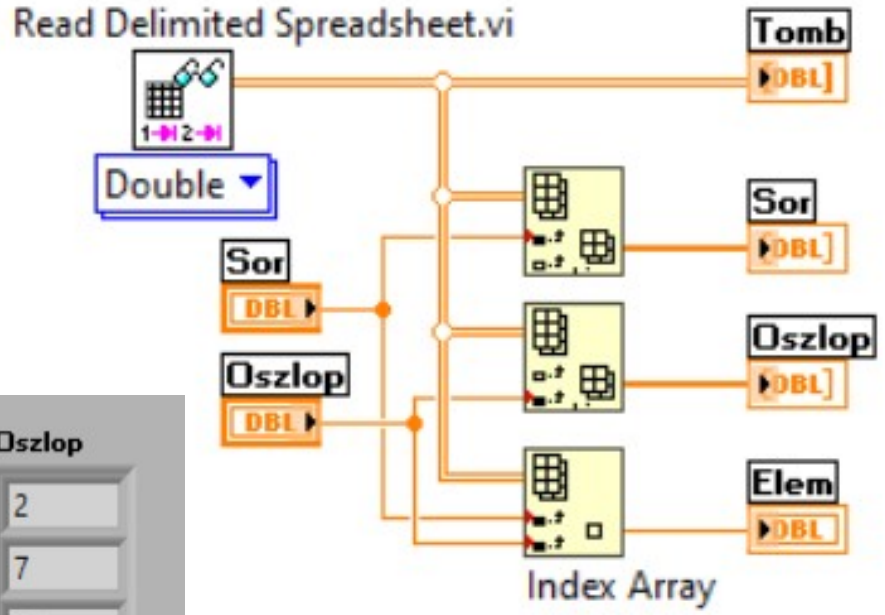
- A fehér hátterű ikonok magasszintű, a sárga ikonok közepes szintű műveleteket képviselnek, a kék hátterűek pedig **Express VI**-ok

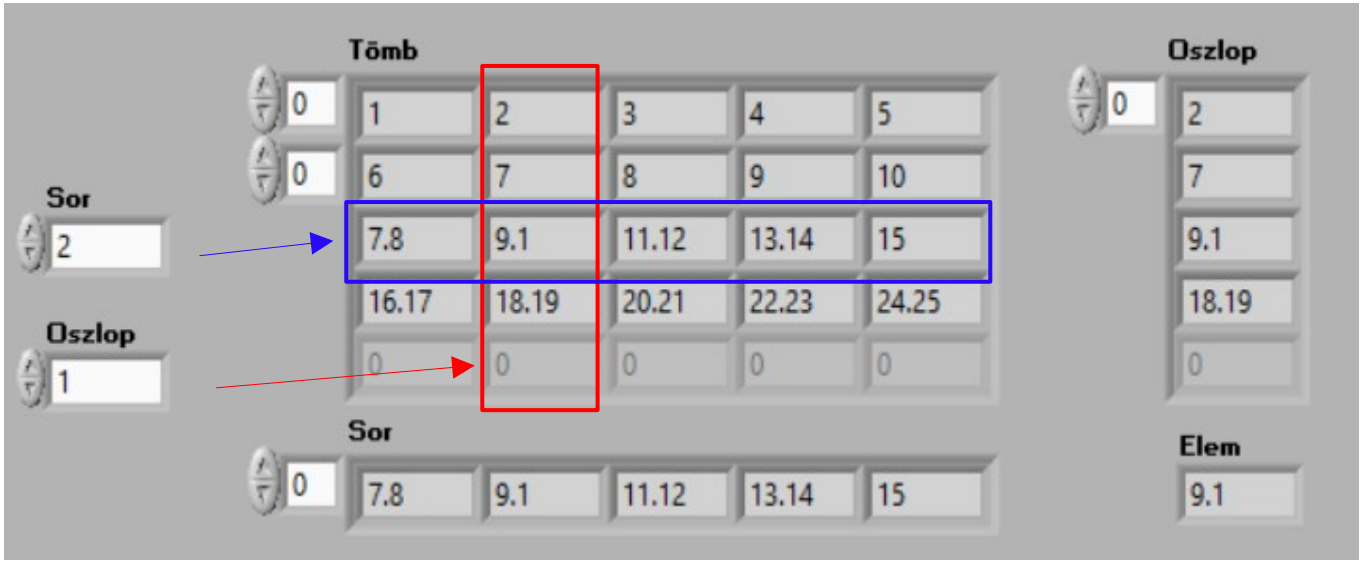


Tableread.vi: tabulált adatok beolvasása

- Ebben a mintapéldában egy magasszintű fájlkezelő VI segítségével fogunk adatokat beolvasni egy tabulátorral szeparált adatokat tartalmazó szövegfájlból (Jáger Attila: LabVIEW alapismeretek 3. fejezet)
- A mellékelt **table.txt** fájl tartalma:

```
1       2       3       4       5
6       7       8       9       10
7.8     9.10    11.12   13.14   15.0
16.17   18.19   20.21   22.23   24.25
```

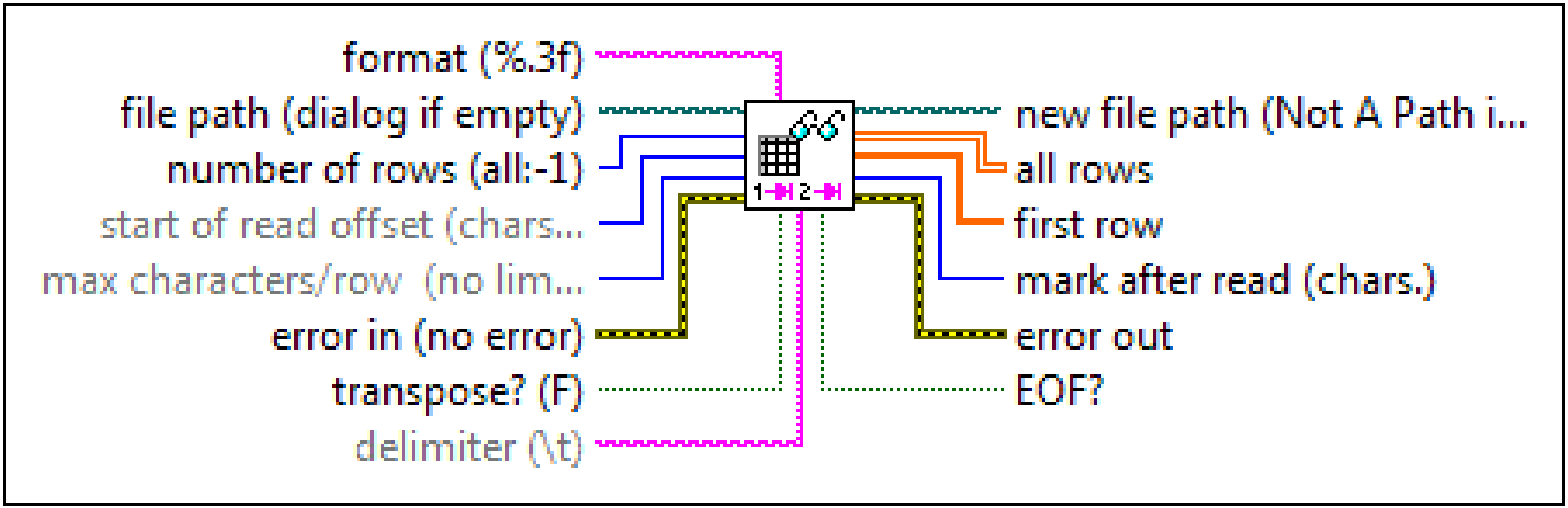




A program indításakor a felugró dialógus ablakban kell a beolvasandó fájlt (itt **table.txt**) kiválasztani

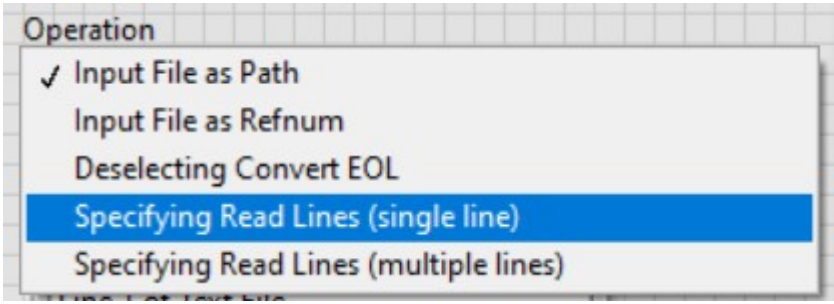
Tableread.vi: tabulált adatok beolvasása

- A **Read Delimited Spreadsheet** VI lebegőpontos (double), egész (integer) vagy szöveg (string) típusú adatfájlok sorait olvassa be és konvertálja 2D táblázattá
- Alapértelmezetten a tabulátor jel (**\t**) az elválasztó, **%.3f** a formátum és fájl végéig tart a beolvasás (a beolvasandó sorok száma = –1)
- A **transpose** paraméter átállításával a sorok és oszlopok felcserélhetők
- Ha nincs megadva fájlnév, egy párbeszédablak automatikusan megnyílik



Szövegfájl írása/olvasása

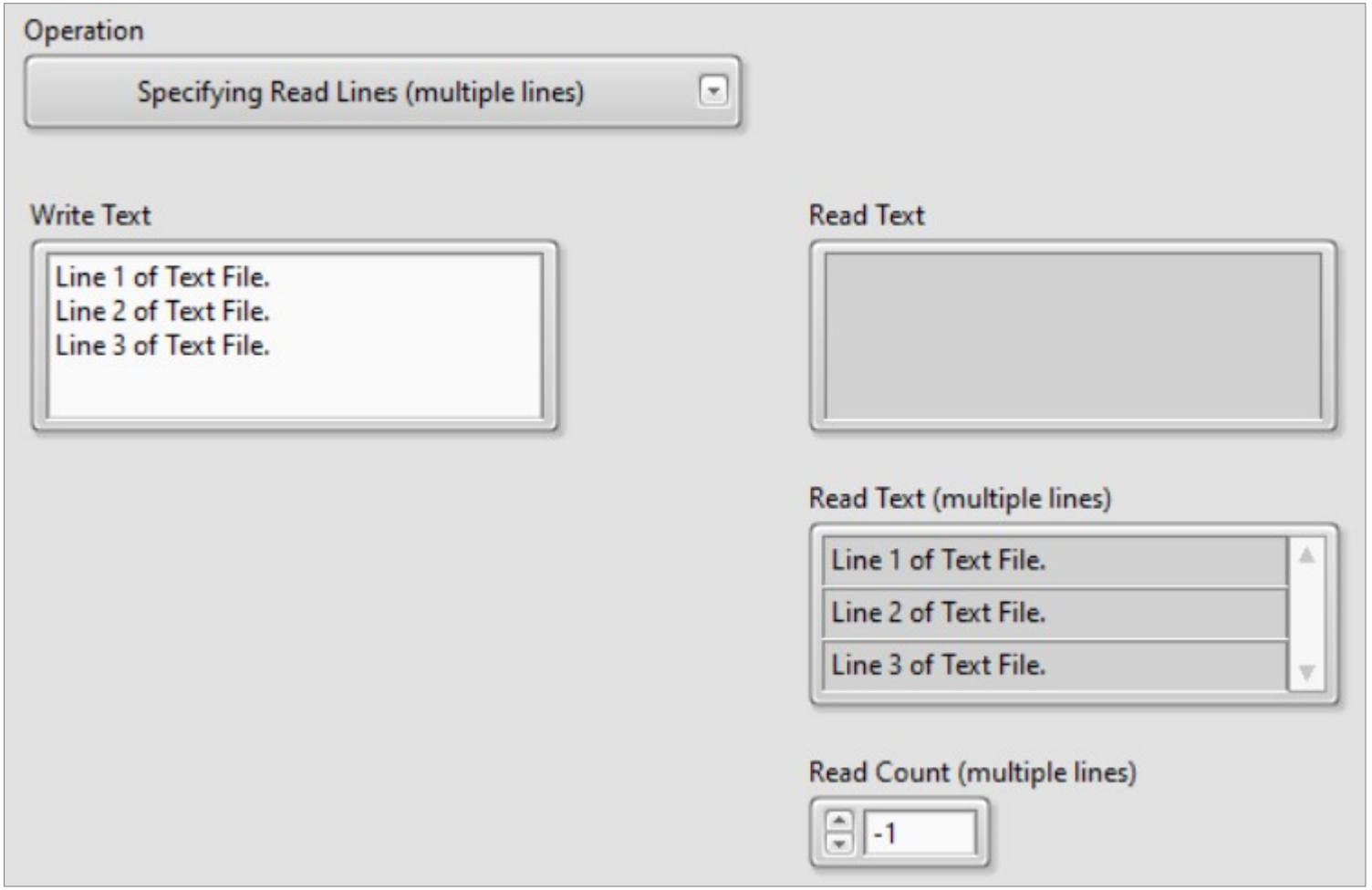
- A **LabVIEW examples\File IO\Text (ASCII)** almappájában található `Write to Text File and Read from Text File.vi` mintapélda a szövegfájlok egyszerű kezelését mutatja be
- Az üzemmódot az előlapi legördülő menüben választhatjuk ki



- **Input File as Path** – elérési úttal és fájlnévvel hivatkozunk a fájlra, beolvasás karakter módban
- **Specifying Read Lines** (single line) – soronként beolvasás, csak egyetlen sort olvas be
- **Specifying Read Lines** (multiple lines) – soronként beolvasás, több sort olvas be, vagy minden sort (–1 megadása esetén)
- A beolvasást írás előzi meg: kiírja a a beíróablakban található szöveget a felhasználó **Documents\LabVIEW Data** mappájába a **Example Write Read Text File.txt** nevű állományba

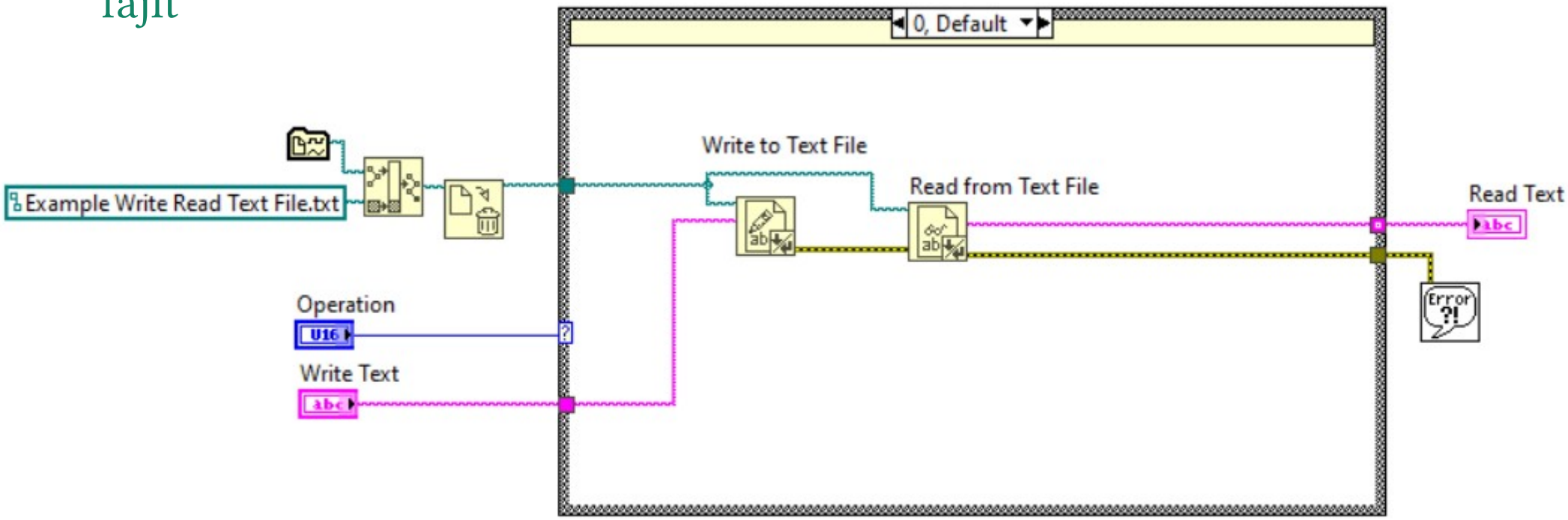
Write to Text File and Read from Text File.vi

- Többsoros módban a **Read Count** paraméter szabja meg, hogy hány sort olvassunk be (–1 esetén fájl végéig olvasunk), a többi módnál ennek nincs szerepe



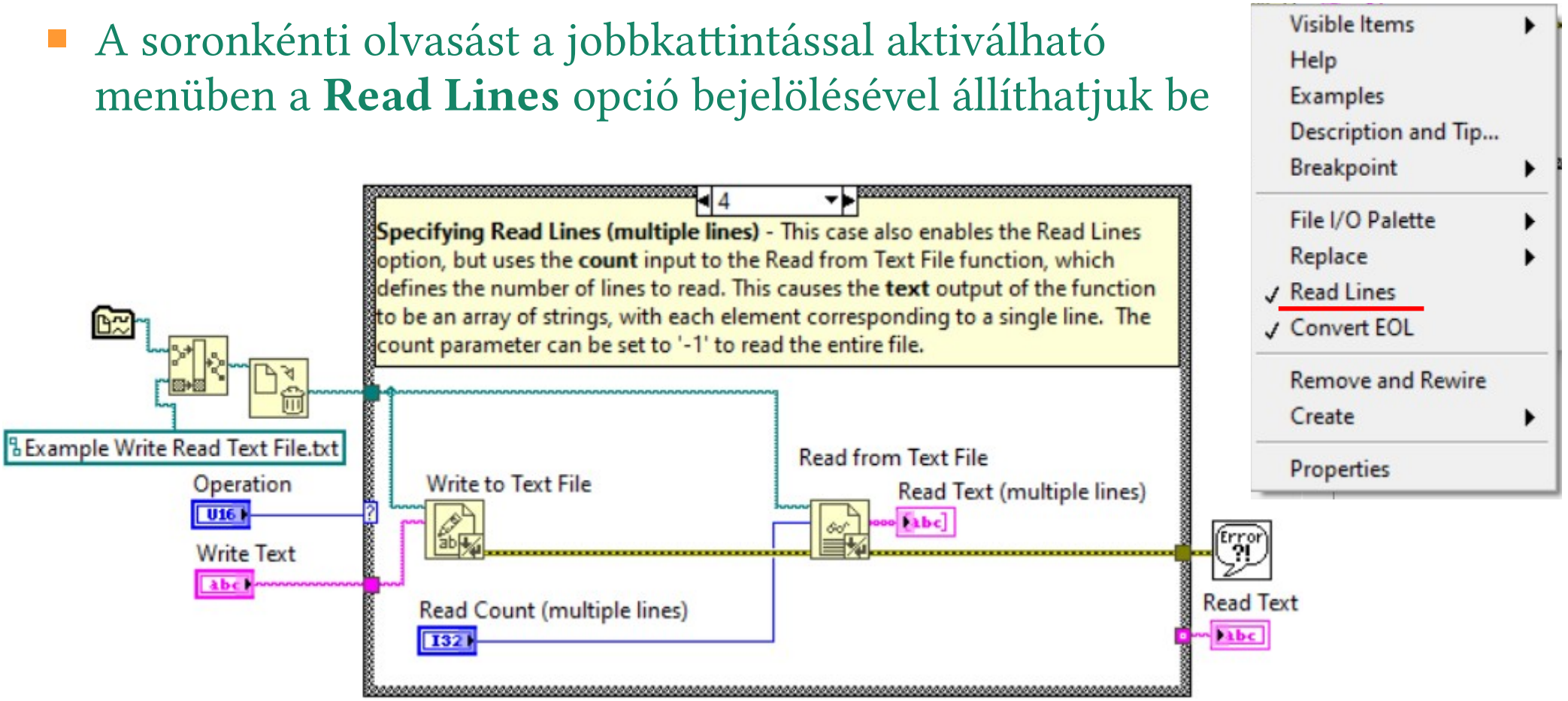
Write to Text File and Read from Text File.vi

- A program első lépésben összefűzi az alapértelmezett mappát (ami a felhasználó **Documents\LabVIEW Data** mappája) a rögzített fájlnévvel és **törli, ha már van ilyen fájl.**
- **Alapértelmezett módban** ezután a **Write to Text File** modul megnyitja az új fájlt, kiírja a beíróablak tartalmát, majd lezárja a fájlt
- Ezt követően a **Read from Text File** modul megnyitja a fájlt, karakteres módban beolvassa a tartalmát, kiírja a Read Text ablakba, majd lezárja a fájlt



Write to Text File and Read from Text File.vi

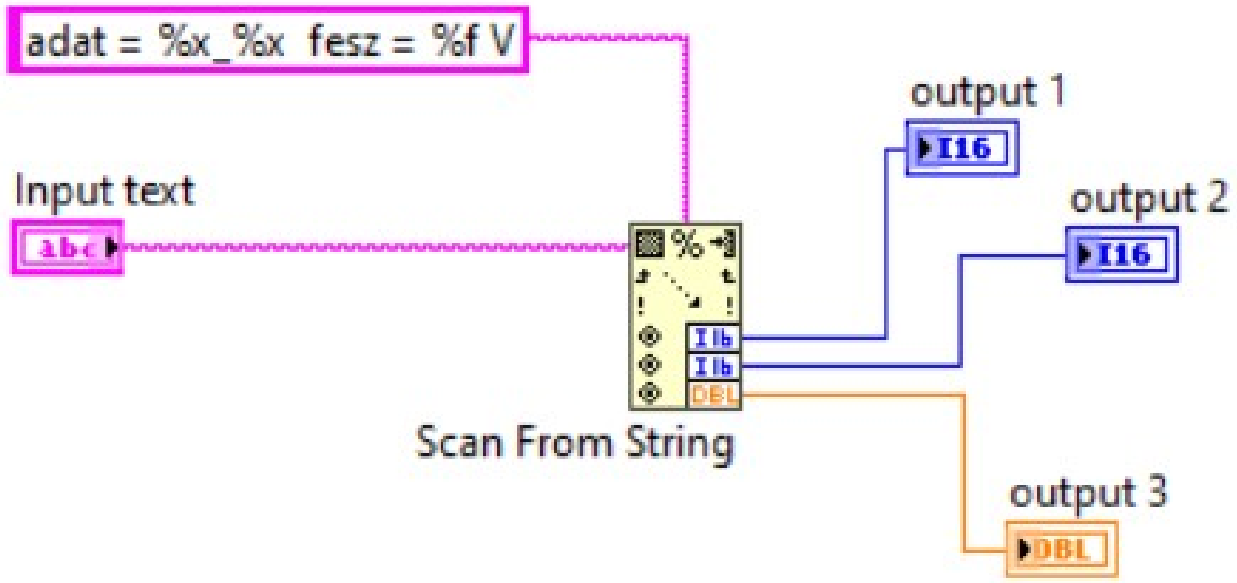
- **Többsoros módban** a kiírás ugyanúgy történik, mint az előbb
- A **Read from Text File** modul itt soronkénti olvasásra van beállítva (ezt jelzi az ikon bal alsó sarkében a vonalkázás) a beolvasandó sorok számát pedig a **Read Count** paraméter szabja (–1 esetén fájl végéig olvasunk)
- A soronkénti olvasást a jobbkattintással aktiválható menüben a **Read Lines** opció bejelölésével állíthatjuk be



file_to_histogram_express.vi

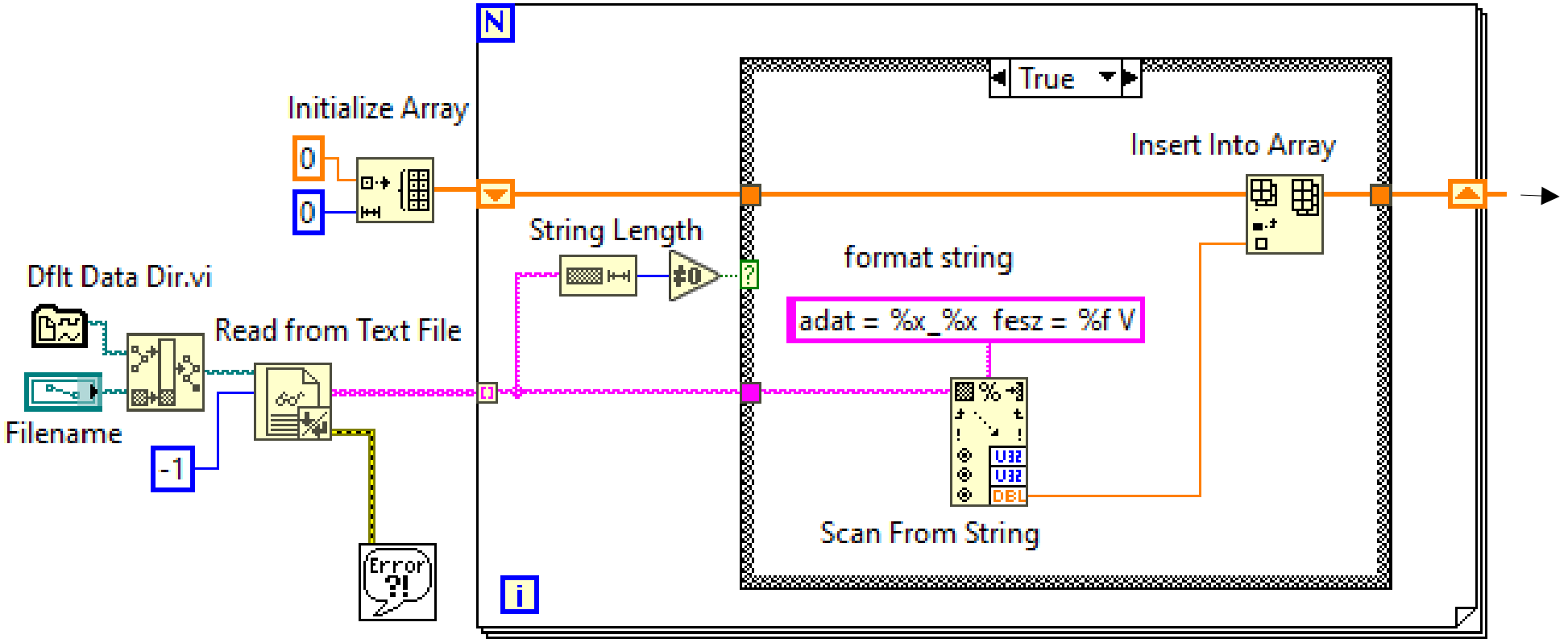
- **Feladat:** Olvassunk be adatfájlokat, amelyek az MSP430 Launchpad kártya és a 24 bites ADC feltét segítségével mért adatokat tartalmazzák, s hisztogram, illetve statisztika készítésével elemezzük az adatsorokat!
- A soronkénti beolvasás az előző program mintájára megoldható. Sajnos, üres sorok is lehetnek ezeket majd ki kell szűrni!
- A fájlok formátuma így néz ki:
- Most csak a feszültség értékek lesz szükségünk, amelyeket a **Scan From String** függvény segítségével nyerhetünk ki a korábban már bemutatott módon

```
adat = 2665_F5B6  fesz =  0.999731 V
adat = 2666_0490  fesz =  0.999766 V
adat = 2666_2A93  fesz =  0.999857 V
adat = 2665_E51F  fesz =  0.999691 V
```



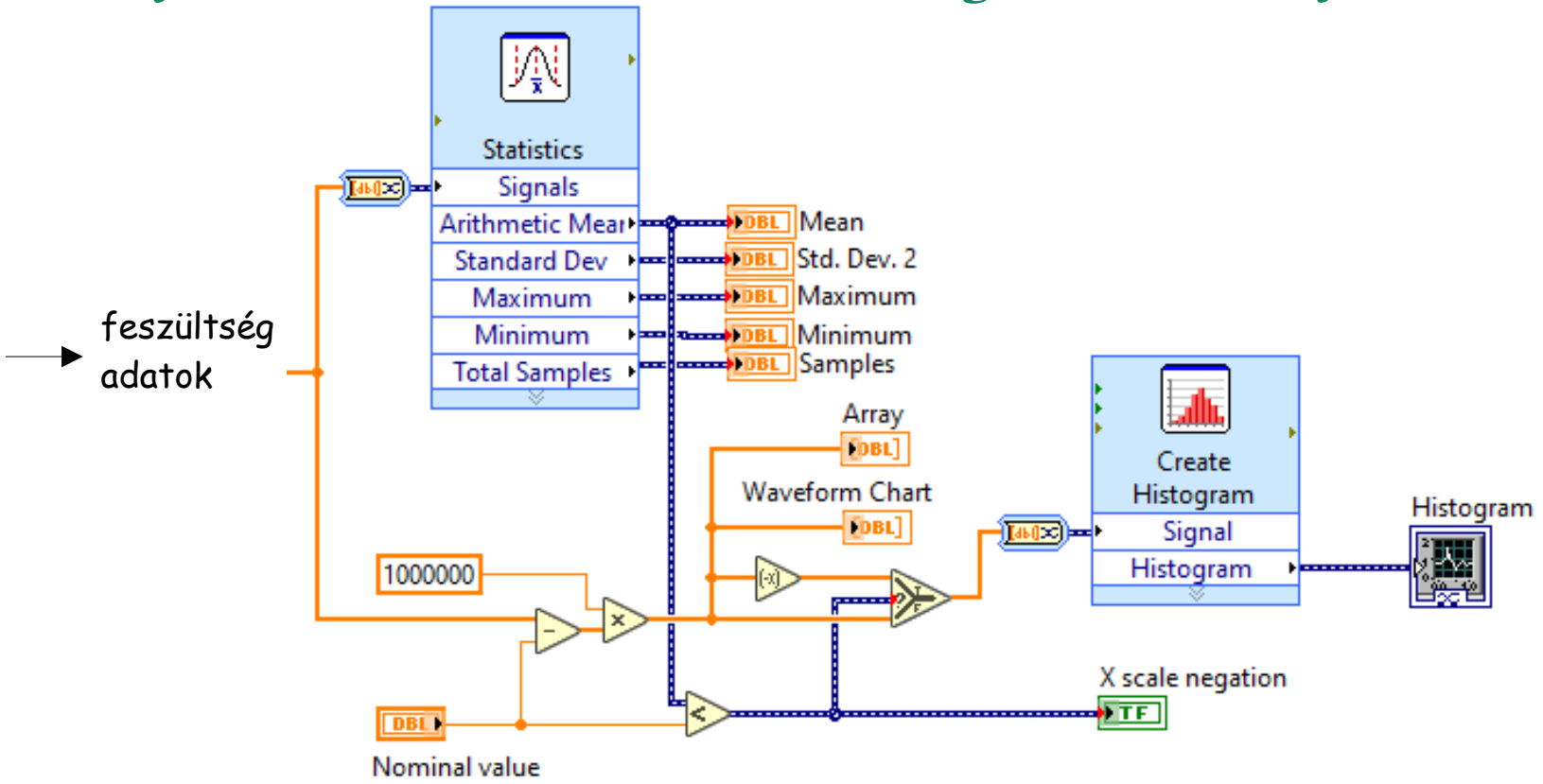
file_to_histogram_express.vi – 1. rész

- **Filename** itt egy beíróablak, ahol a fájlnév (szükség esetén relatív elérési úttal együtt) megadható
- Az üres sorok kiszűrésére a kezdeti üres tömbhöz csak akkor adunk új elemet, amikor a sorhosszúság nem nulla
- A feldolgozandó adattömb a **Shift regiszter** kimeneténél (⟶) áll elő



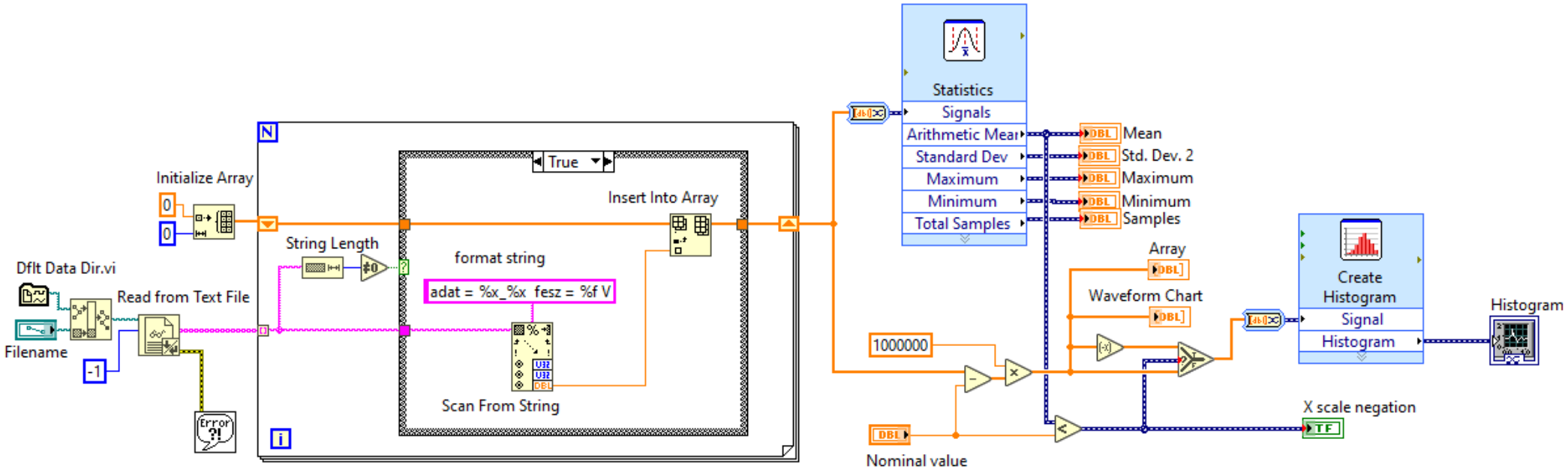
file_to_histogram_express.vi – 2. rész

- Az **Express VI** modulokat konfigurálása a dupla kattintással előhívható párbeszédablakban történik
- A hisztogramhoz a **névleges feszültségtől** való eltérések értékét használtuk, mikrovoltokra átskálázva. A hisztogramok megjelenítése negatív értékeknél megzavarodik, ezért szükség esetén automatikusan előjelet váltunk, amit az **X scale negation** LED kijelez az előlapon

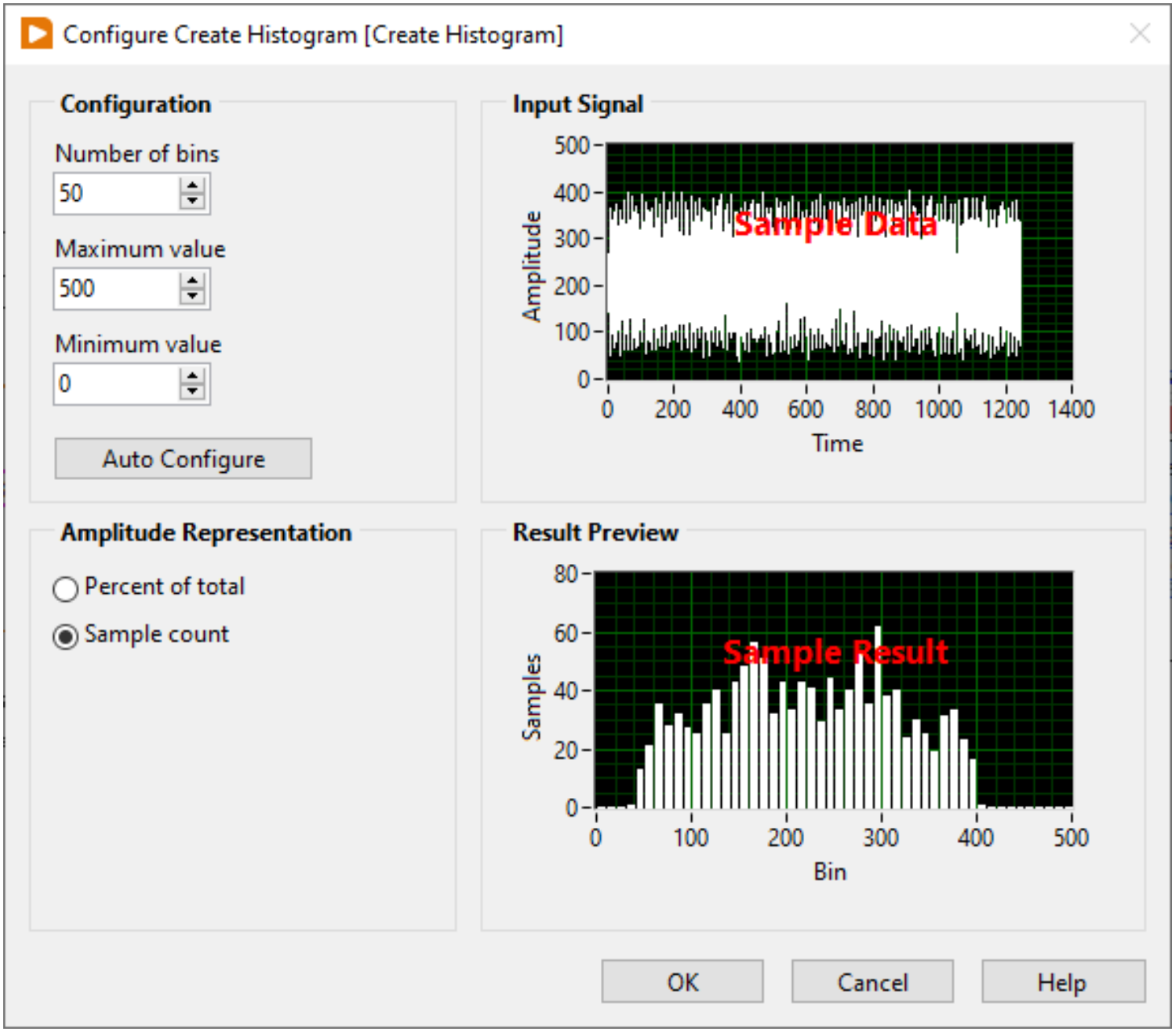


file_to_histogram_express.vi

- A két részletben ismertetett program az alábbi ábrán látható egészben
- Az **If stuktúra** false ága csak egyetlen átkötést tartalmaz (a shift regiszter segítségével a korábbi tartalmat őrzi)
- A program indítása előtt beírandó a beolvasni kívánt **fájl neve** (szükség esetén a már említett **LabVIEW Data** mappához képesti relatív elérési útjával együtt), és meg kell adni a mért feszültség **névleges értékét** (**Nominal value**)



A Histogram modul konfigurálása



A Statistics modul konfigurálása

Configure Statistics [Statistics]

Statistical Calculations

- [x] Arithmetic mean
- [ ] Root mean square (RMS)
- [ ] Median
- [x] Standard deviation
- [ ] Mode
- [ ] Variance
- [ ] Sum of values
- [ ] Kurtosis
- [ ] Skewness

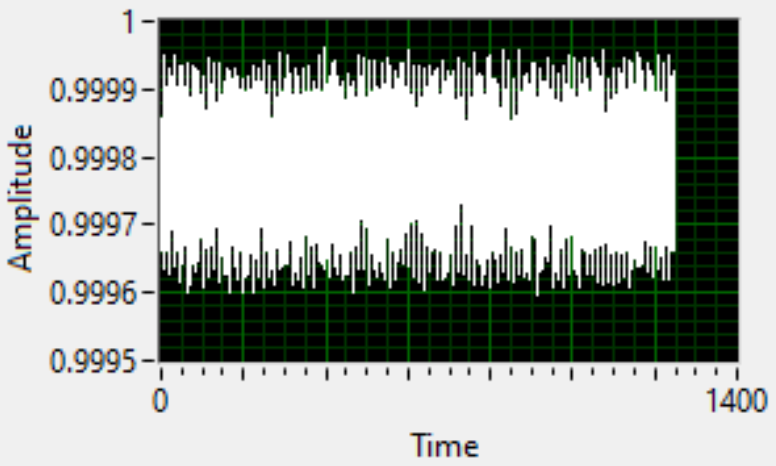
Extreme Values

- [x] Maximum
- [ ] First time
- [ ] Time of maximum
- [ ] First value
- [ ] Index of maximum
- [ ] Last time
- [x] Minimum
- [ ] Last value
- [ ] Time of minimum
- [ ] Index of minimum
- [ ] Range (maximum - minimum)

Sampling Characteristics

- [x] Total number of samples
- [ ] Time between samples (dt)

Input Signal



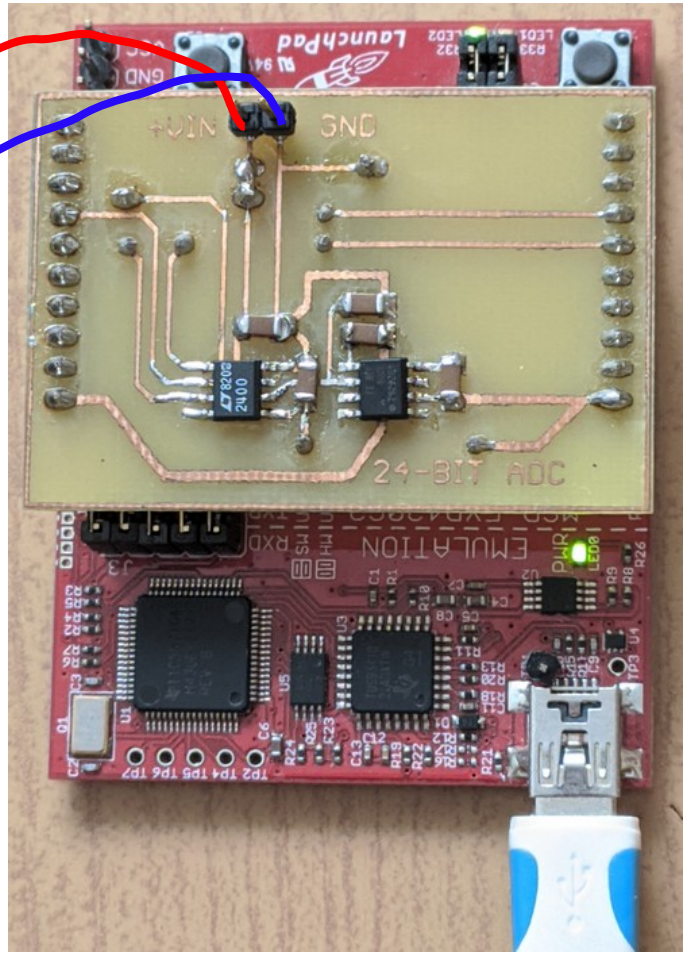
Results

Statistic	Result
Arithmetic Mean	0.999781
Standard Dev	9.359318E-5
Maximum	0.999961
Minimum	0.999597
Total Samples	1247

OK Cancel Help

1. kísérlet: mérések DC standard tápegységgel

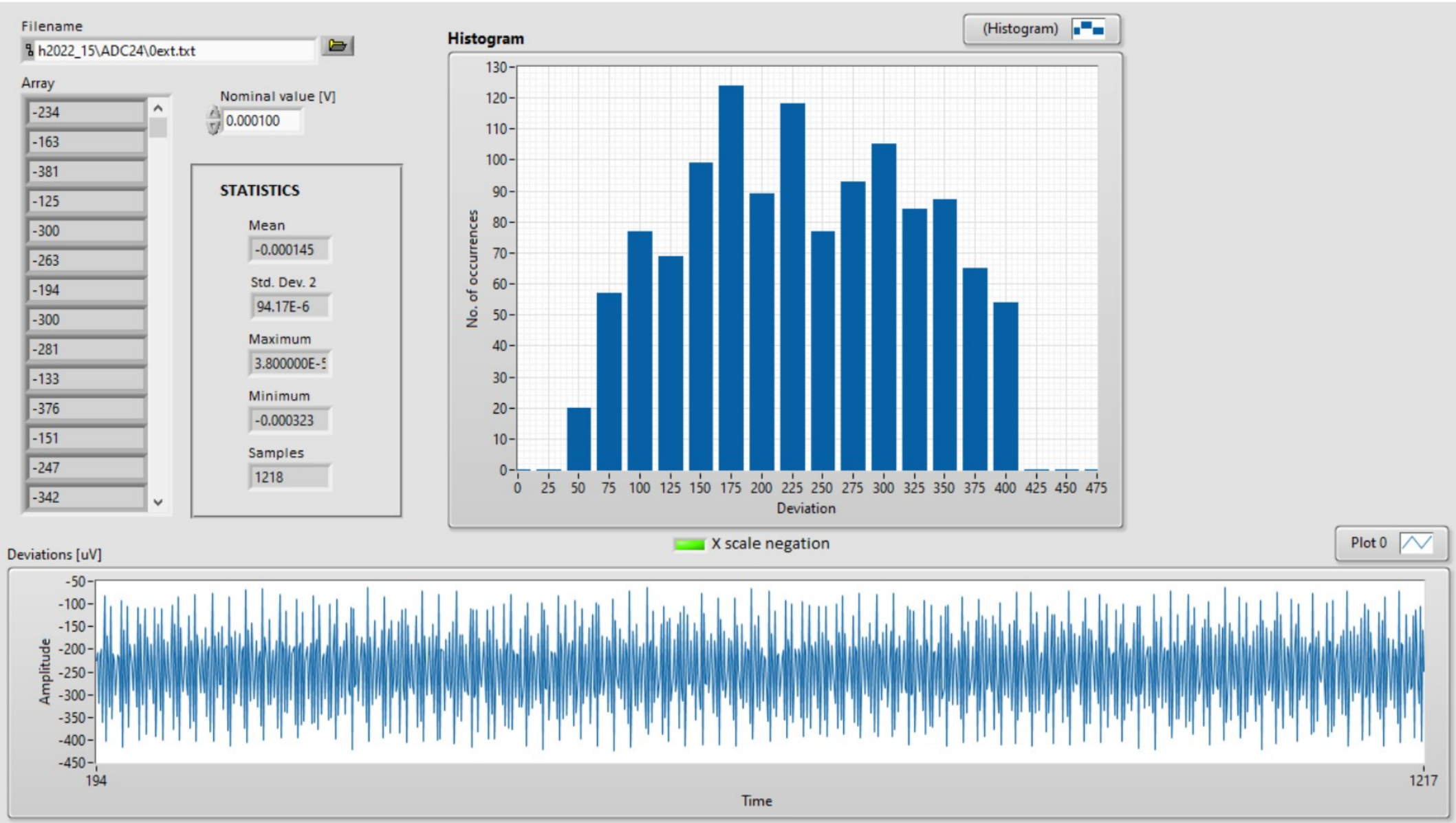
- Az első kísérletnél egy **Takeda Riken** gyártmányú **TR-6120** típusú DC standard (precíziós egyenfeszültségű tápegység) feszültségét mértük



Névl. feszültség	Adatfájl neve	Mérések átlaga
0.00 V	0ext.txt	-0.000145 V
0.50 V	0_5ext.txt	0.499824 V
1.00 V	1_0ext.txt	0.999781 V
1.50 V	1_5ext.txt	1.499744 V
2.00 V	2_0ext.txt	1.999717 V
2.50 V	2_5ext.txt	1.999717 V

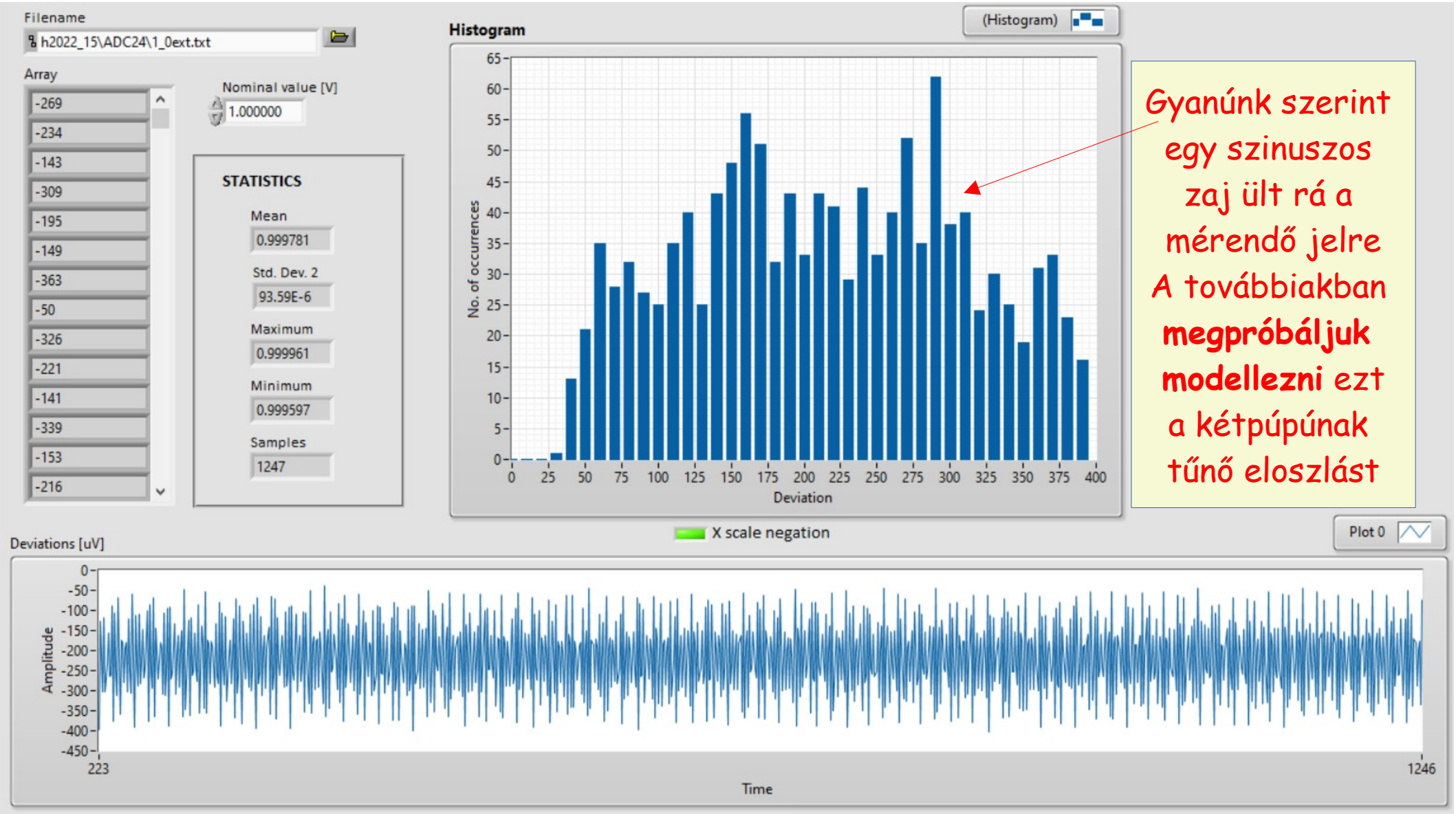
file_to_histogram_express.vi futtatása

- A **0ext.txt** adatsor elemzése, a hibaeloszlás **nem a normális eloszlást** követi!



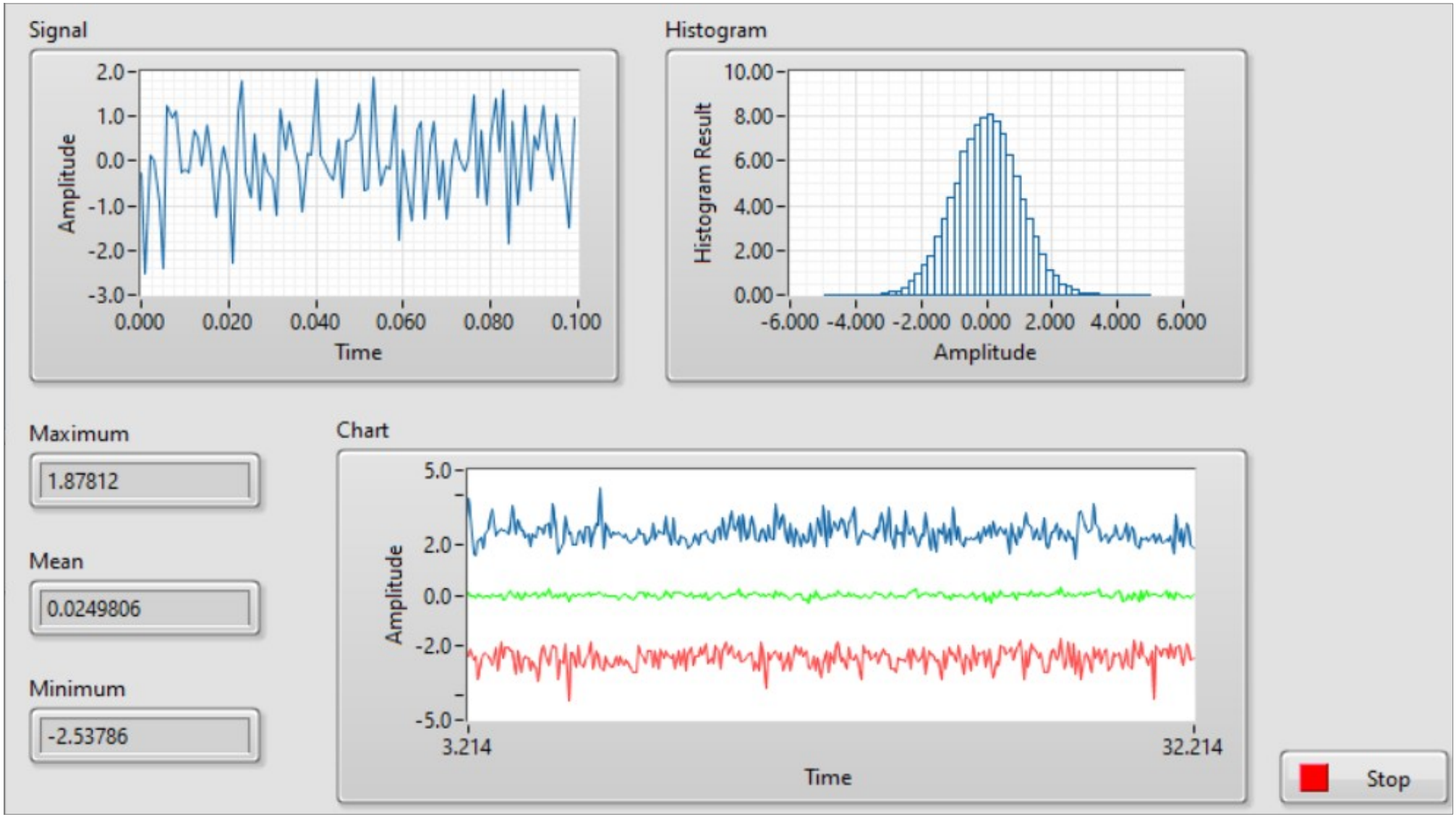
file_to_histogram_express.vi futtatása

- Az **1_0ext.txt** adatsor elemzése, a hibaeloszlás nem a normális eloszlást követi!



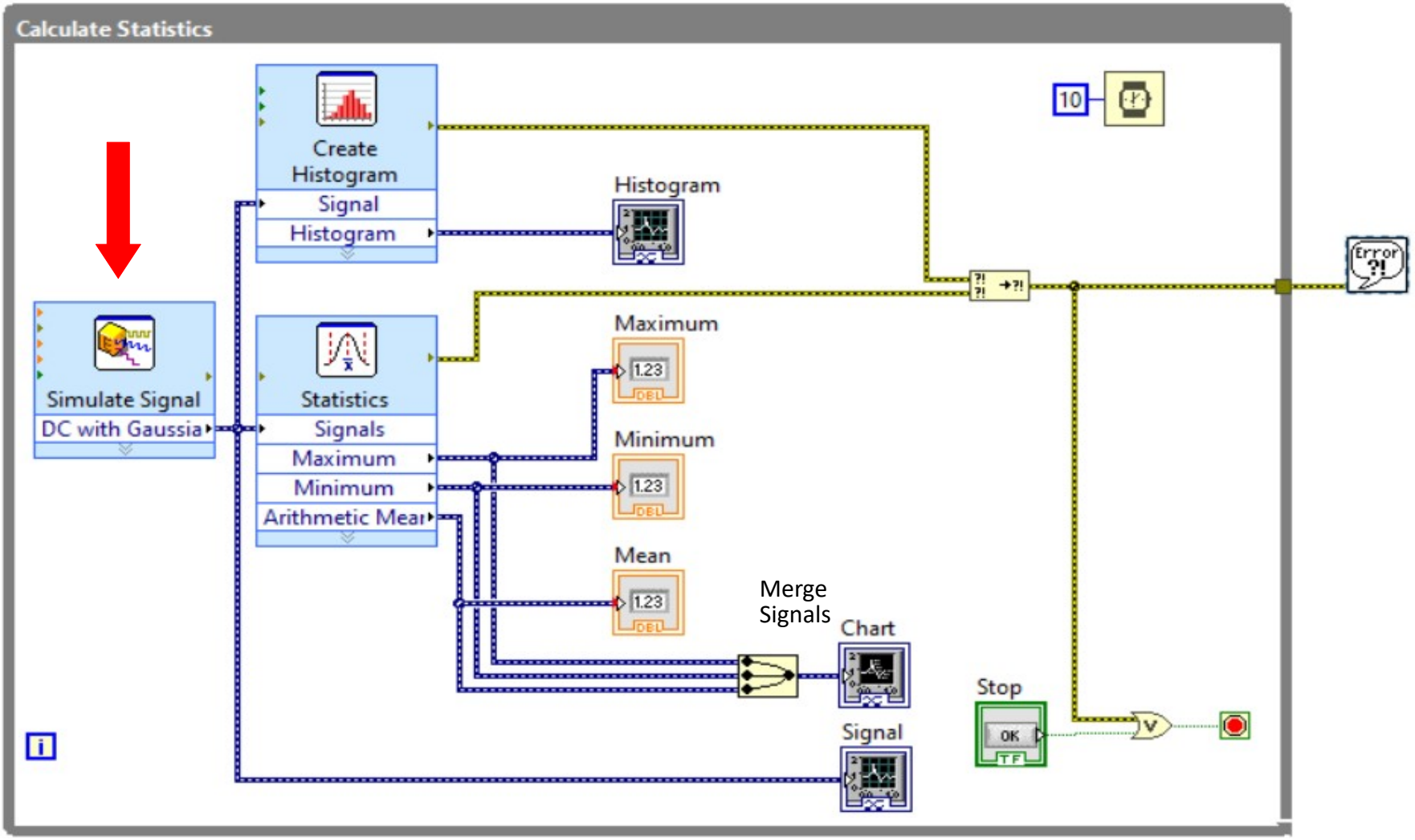
Emlékeztető: Express VI - Statistics.vi

- A LabVIEW **examples\Express Vis** mappájában található mintapélda véletlenszám csomagokat generál, s ezek statisztikai jellemzőit jeleníti meg



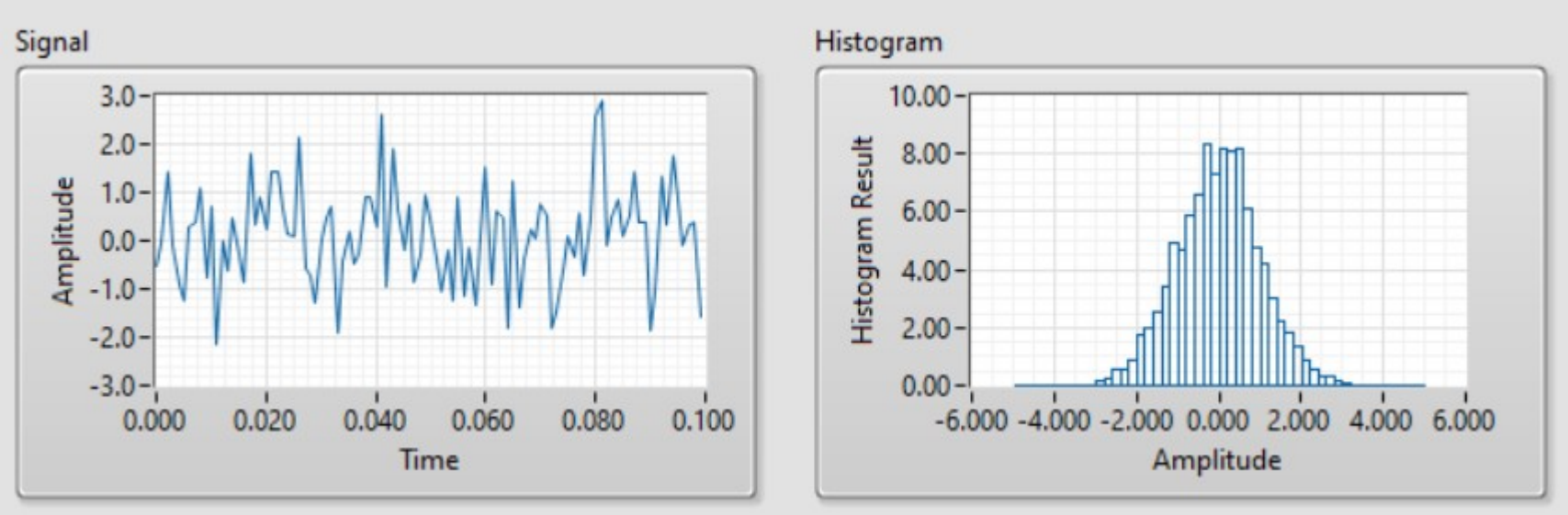
Express VI - Statistics.vi

- A jelszimulációt végző **Simulate Signal** modul beállításával fogunk játszadozni: normál eloszlású zaj, szinusz és zajjal terhelt szinusz hisztogramjainak összehasonlítására próbálunk ki különféle beállításokat

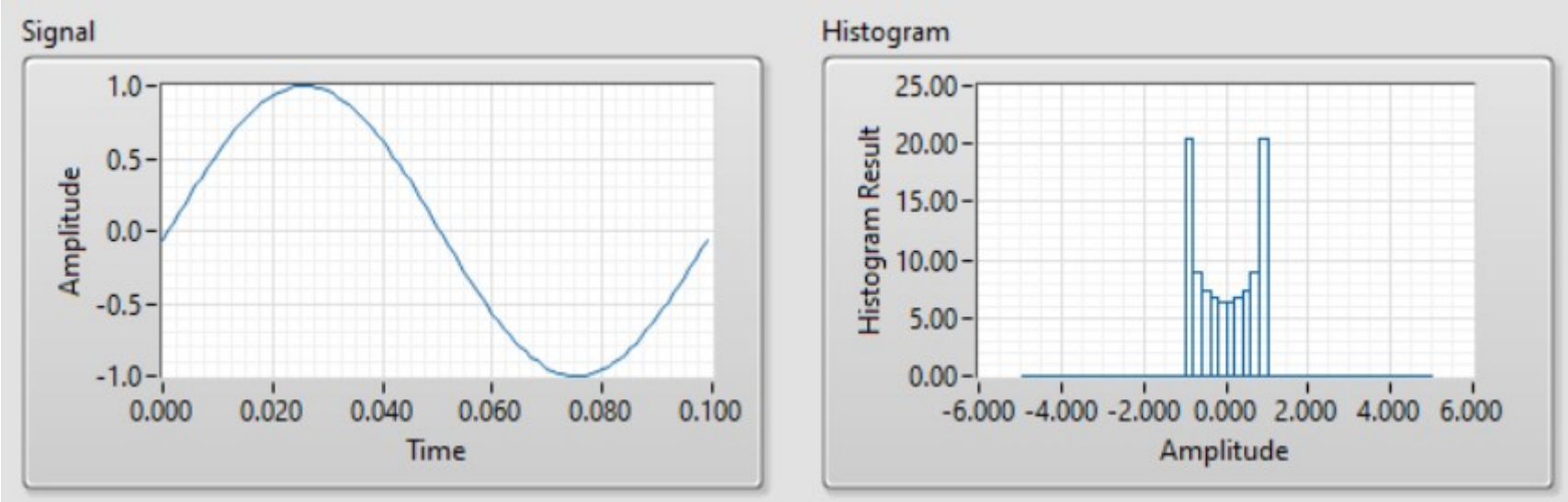


Express VI - Statistics.vi

- DC + Gaussian white noise

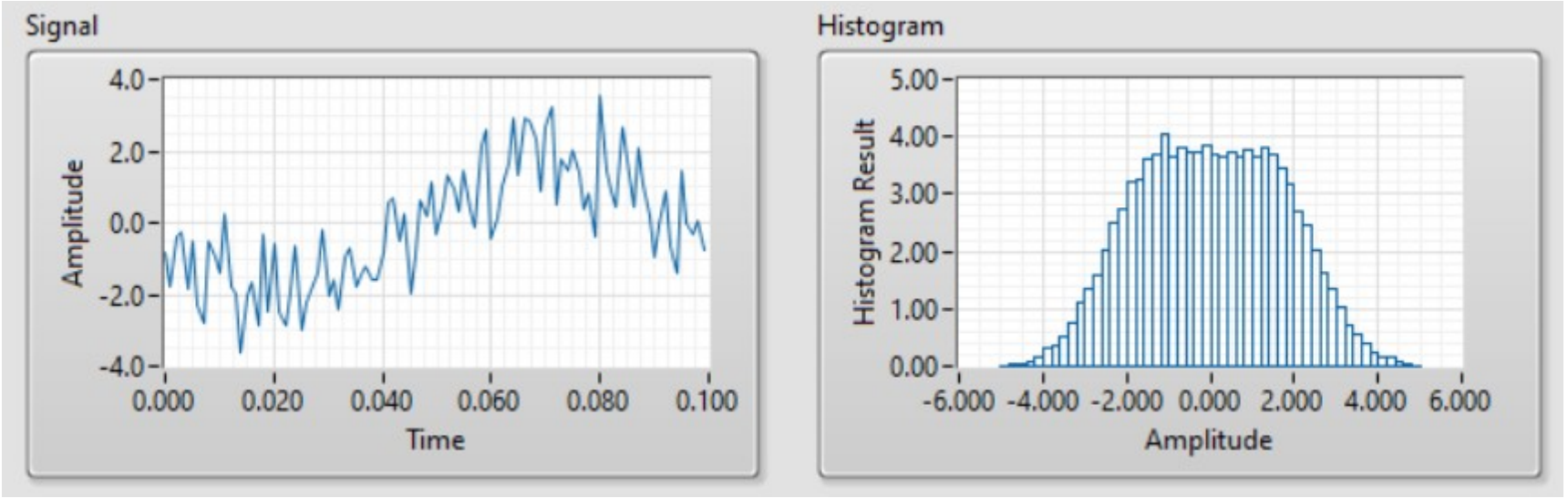


- Sine (without noise)

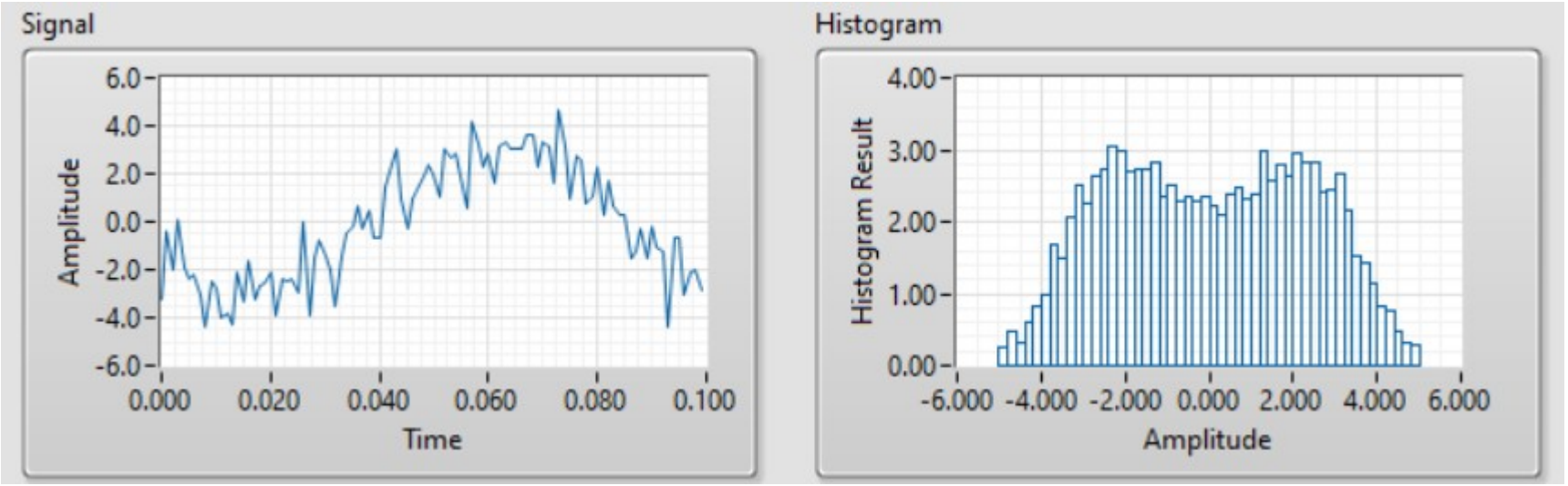


Express VI - Statistics.vi

- 2 x Sine + Gaussian white noise

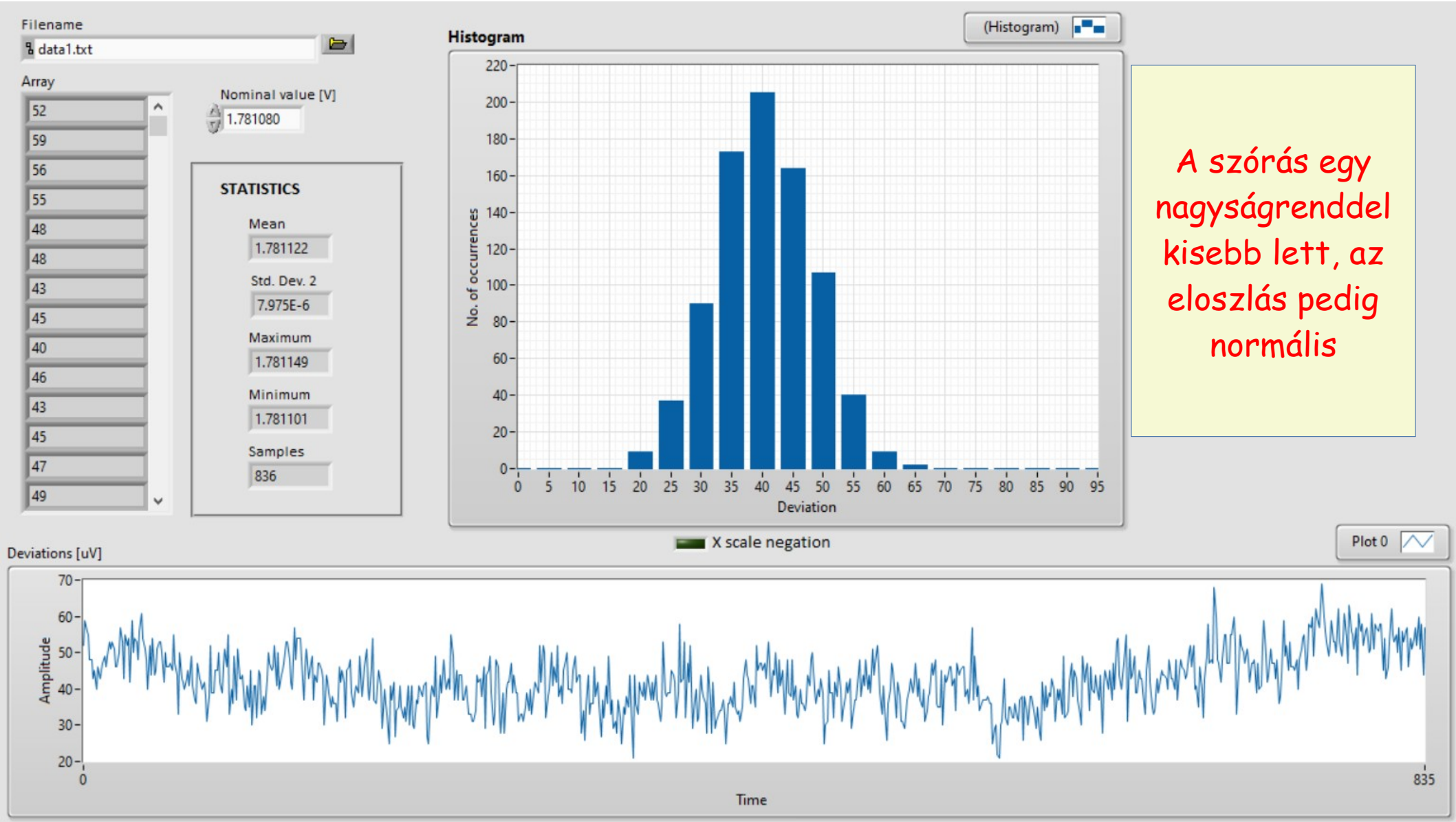


- 3 x Sine + Gaussian white noise – *„valóban, görög van a falóban!”*



2. kísérlet: csökkentsük a zajt!

- **data1.txt** – az **MSP430** 3.5 V-os tápfeszültségét mértük egy 2x10 kΩ osztóval



A szórás egy nagyságrenddel kisebb lett, az eloszlás pedig normális

3. kísérlet: csökkentsük a zajt!

- **data2.txt** – 9 V-os elemmel táplált 100 kΩ + 10 kΩ osztón mértük a feszültséget