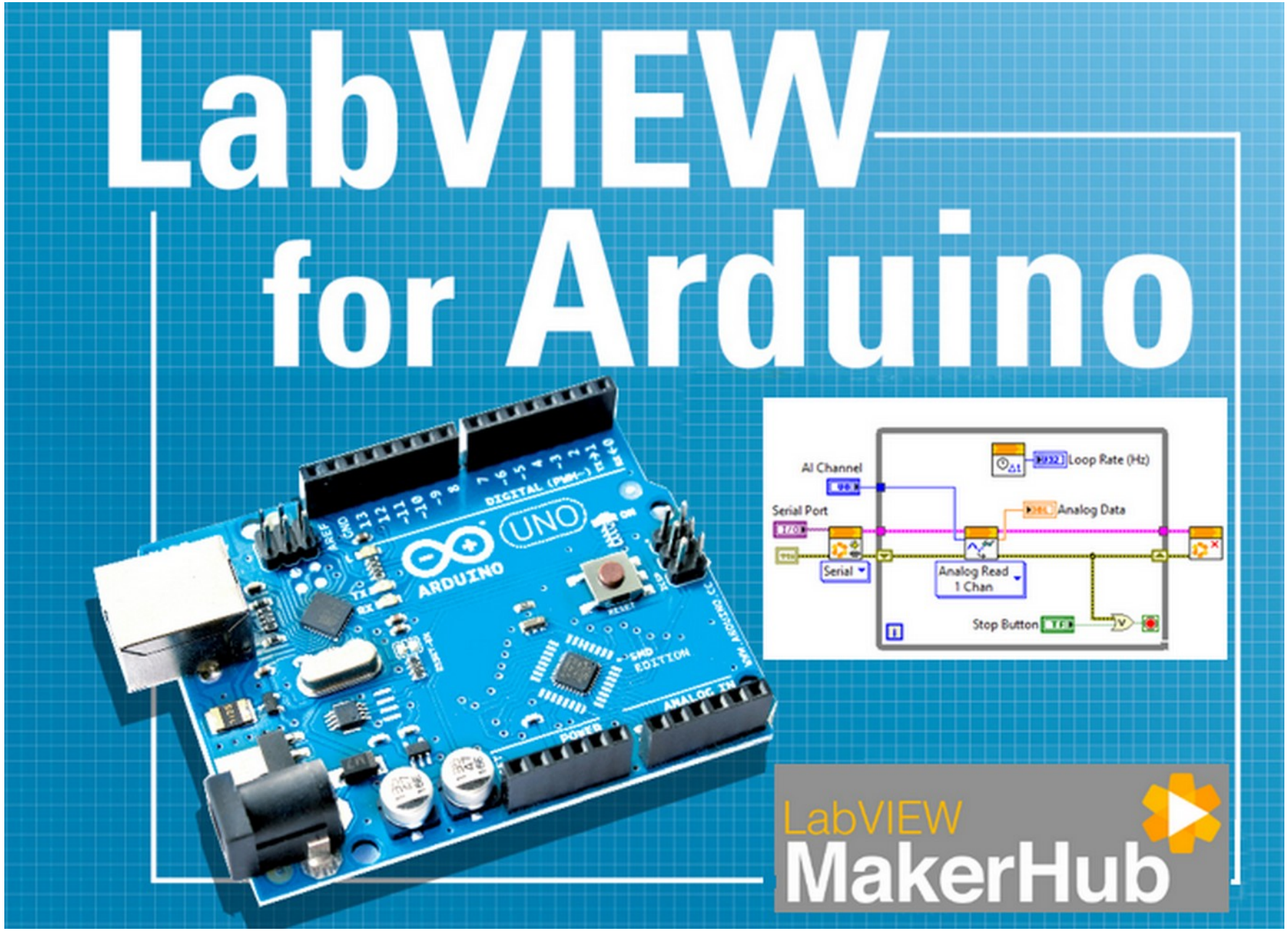
16. LabVIEW + LINX + Arduino - 6. rész



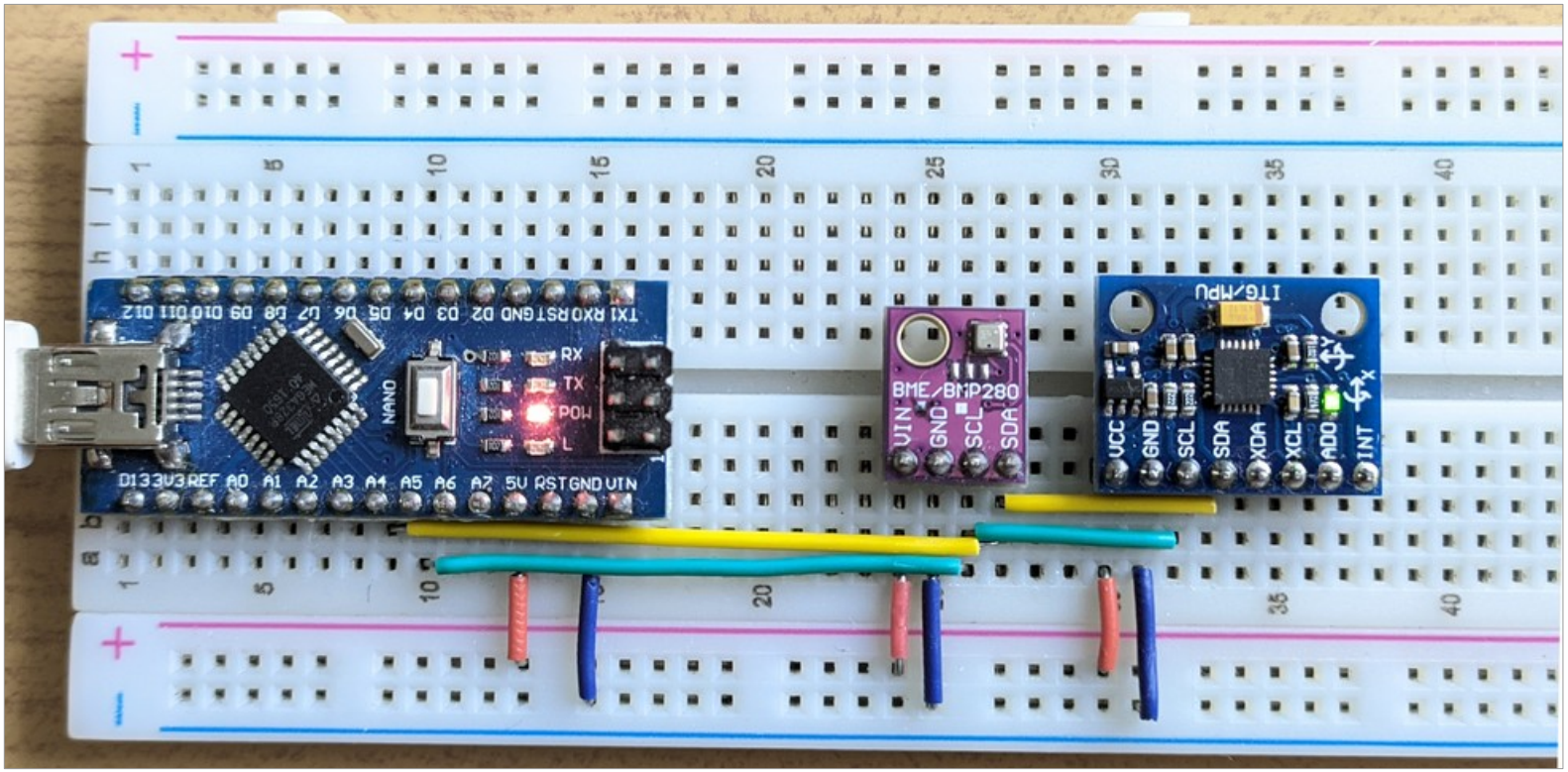
Felhasznált és ajánlott irodalom

- NI: Getting Started with Arduino and LabVIEW Community Edition
- NI: LabVIEW Documentation
- 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)

- Asper S.r.l.: BME280 LabVIEW code for Raspberry Pi
- Sudharsan Sukumur: How to Use I2C in LabVIEW
- Sudharsan Sukumar: Instructables: Plug and Play Pmods Using LabVIEW
- Bosch Sensortec: BME280 adatlap
- InvSense-TDK: MPU-6050 datasheet
- InvSense-TDK: MPU-6050 Register Map

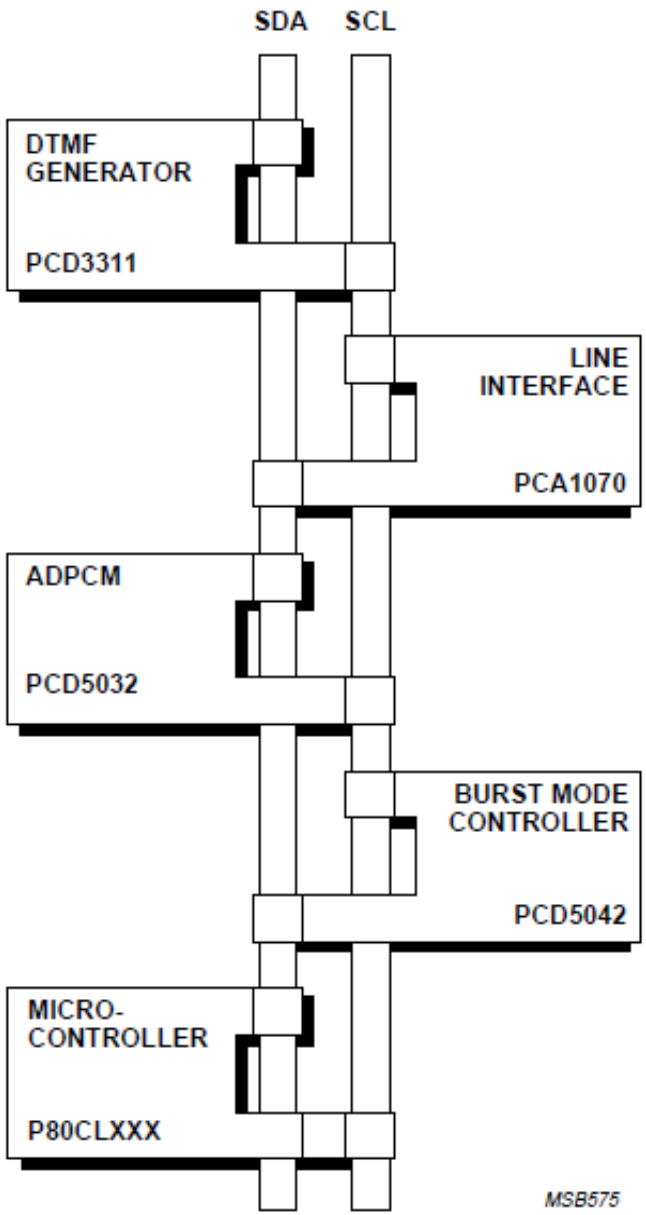
Tartalomjegyzék

A kísérleti elrendezés

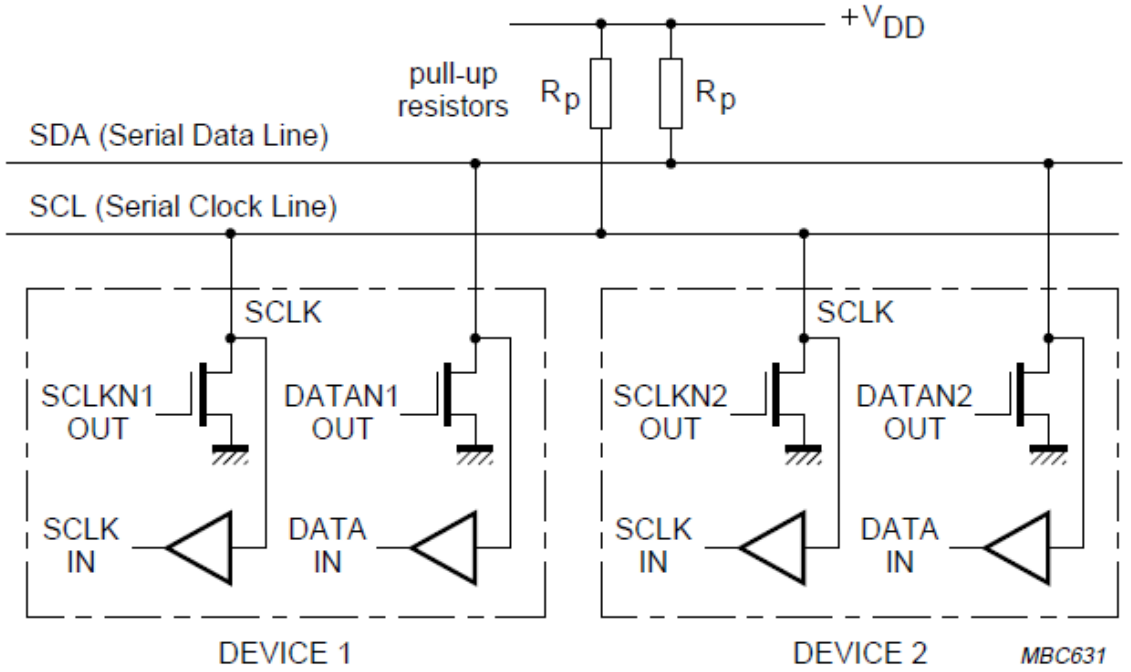


Az I2C busz

- „Inter-Integrated Circuit” busz – vagy „kétvezetékes busz”, amelyet eredetileg a Philips cég dolgozott ki 1982-ben.
- **Több eszköz** (master és slave) **fűzhető fel a buszra**
- A buszt a **mester (master) eszközök** vezérlik és kezdeményezik az adatforgalmat. **A szolga (slave) eszköz** akkor válaszol, ha címzéssel megszólítják
- **Az I²C busz két jelvezetéket használ**
  - **SCL:** szinkronizáló órajel
  - **SDA:** soros adat
- **A részletes leírás** az ”Az I²C-busz specifikációja és használata” című dokumentumban olvasható

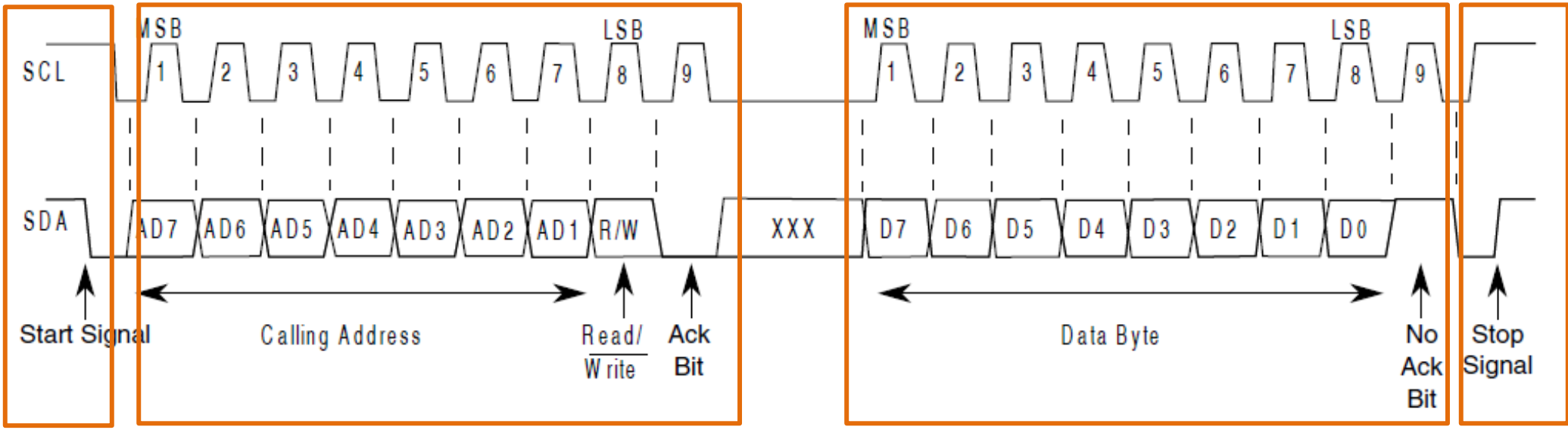


Az I2C busz vezérlése



- A jelvezetékeket ellenállások húzzák fel tápfeszültségre V_{DD}
- Nyitott nyelőelektródás FET-ek húzzák le a vonalakat alacsony szintre
- A buszt vezérlő mester eszköz állítja elő az SCL órajelet
  - normál mód: 100 kHz
  - gyors mód: 400 kHz
  - nagysebességű mód: 1 MHz, vagy több (aktív felhúzással)

I2C üzenetformátum



Üzenet-orientált adatátvitel négy felvonásban:

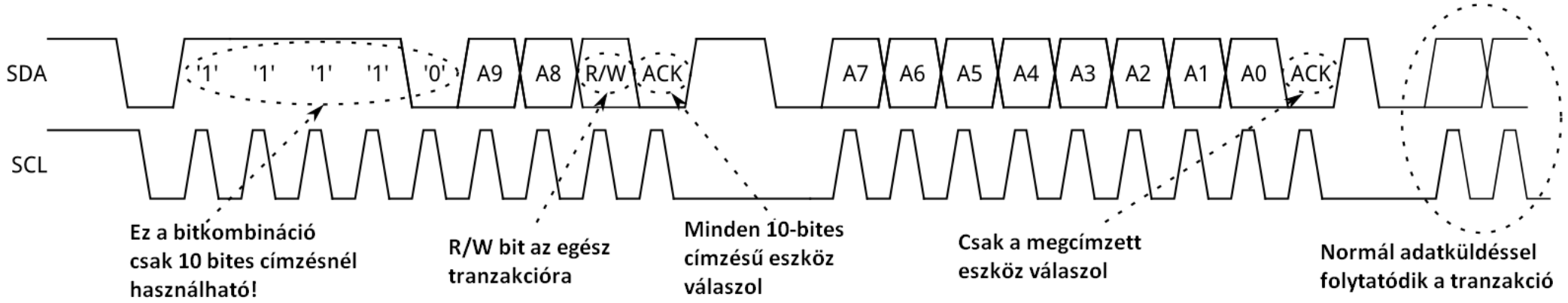
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

Nyugtázás (ACK): a 9. órajel impulzus tartamára a vevő alacsony szinten tartja az **SDA** jelvezetéket.

Negatív nyugtázás (NAK): a 9. órajel impulzus idején senki sem húzza le az **SDA** jelvezetéket – az magas szinten marad.

10-bites címzés az I2C buszon

- A 10-bites címzés az eredetileg 7-bites címzést használó I2C protokoll bővítése.
- A 10 bites címet csak két részletben tudjuk kiküldeni.
- Az első rész egy speciális bitsorozattal (11110) kezdődik. Ez a bitkombináció a 10 bites címzés számára fenntartott, tehát 7 bites címzésnél ez a bitsorozat nem használható eszközcímként.
- Az első rész kiküldése után minden 10-bites címzést használó eszköz küld nyugtázást.
- A második bájtban a cím többi bitjeit (A0…A7) küldjük ki. Erre már csak az az eszköz válaszol, amelynek mind a 10 címbitje megegyezik az általunk kiküldöttekkel.
- Az adatküldés a továbbiakban ugyanúgy folyik tovább, mint a 7-bites címzésnél.

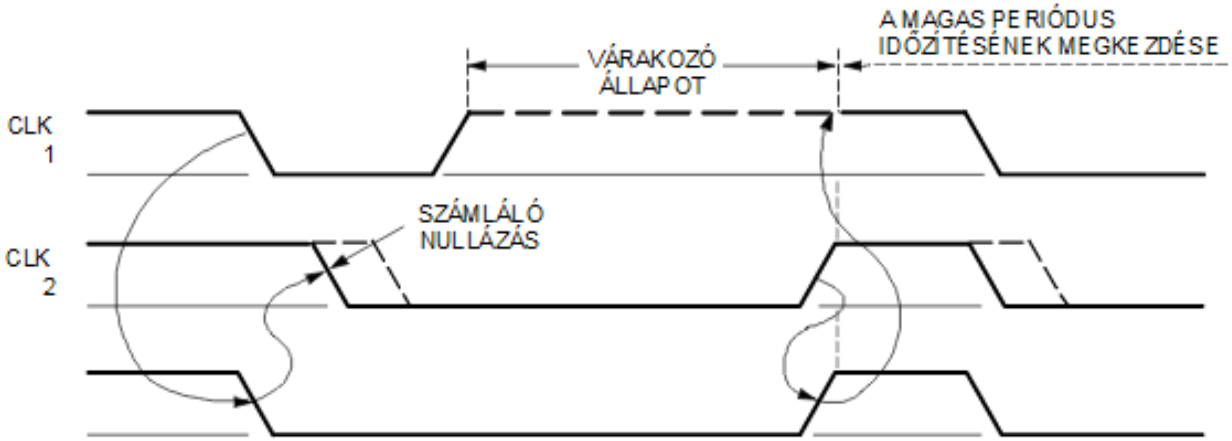


Órajel szinkronizálás

Minden master a saját órajelét generálja az **SCL** vezetéken ahhoz, hogy üzenetet vigyen át az I²C-buszon.

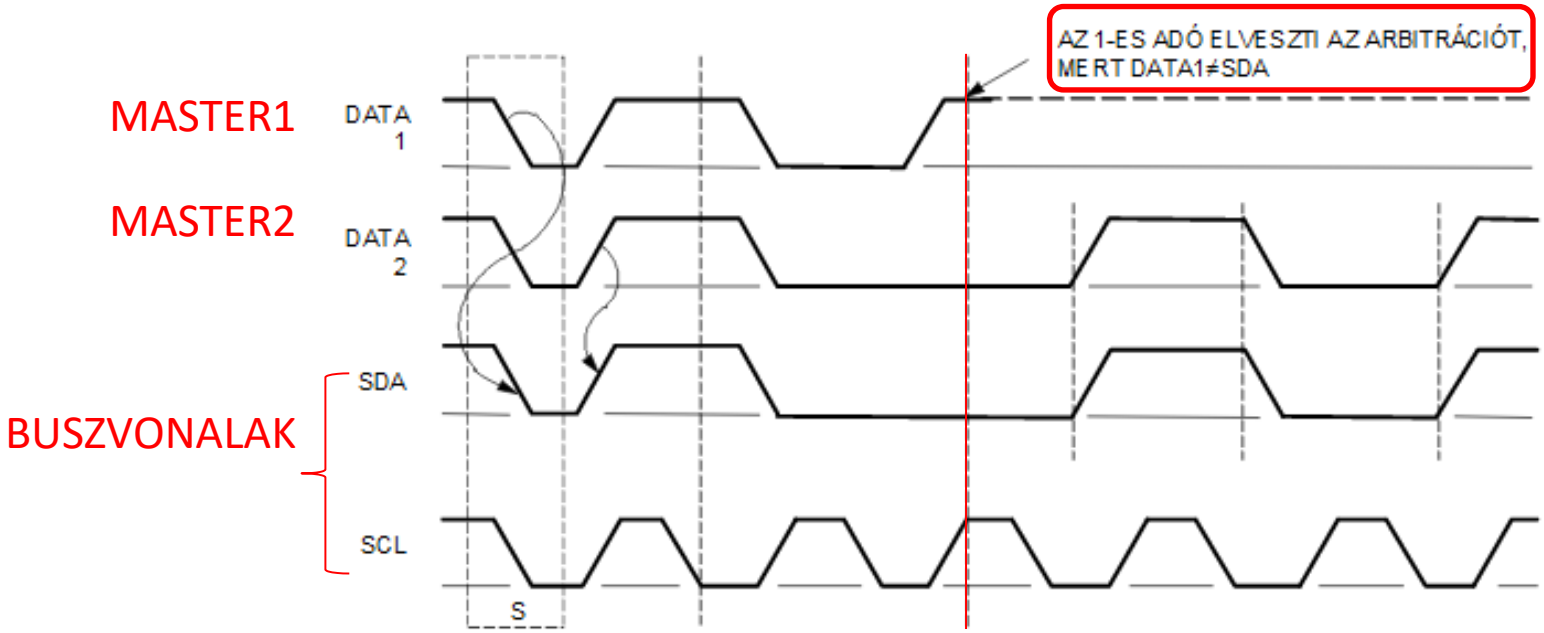
Az órajelszinkronizáció az I²C interfészek huzalozott **ÉS** kapcsolatával megy végbe (a vezetéken csak akkor lesz magas szint, ha minden eszköz magas szintre vált).

- Az **SCL** vezetéken egy **magas-alacsony átmenet** az érintett eszközöknél az alacsony periódusuk időzítésének megkezdését eredményezi. Ha egy eszköz órajele alacsonyra váltott egészen addig alacsony állapotban tartja az SCL vezetéket, amíg az órajele magas periódusához nem ér.
- Az **SCL** vezetéket a leghosszabb alacsony periódusú eszköz tartja alacsonyan. Erre az időre a rövidebb alacsony periódussal rendelkező eszközök **magas szintre várakozó állapotba** kerülnek.



Arbitráció

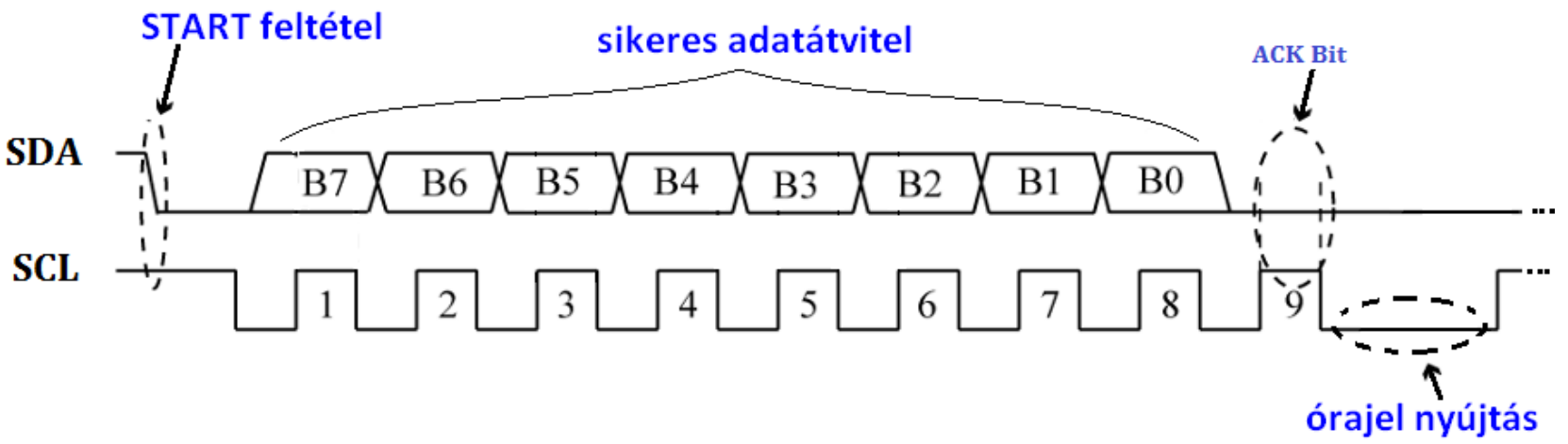
- Multi-master környezetben csak akkor kezdhetünk átvitelt, ha a busz szabad.
- Két, vagy több master generálhat egy start feltételt a START feltétel minimális tartási idején belül. Amig az SCL vezeték magas szinten van az SDA vezetéken végbemegy az arbitráció.
- Az a master veszít, amelyik magas szintet küld, miközben egy másik master alacsonyat. A vesztes master lekapcsolja az adatkimeneti fokozatát, mert a busz jelszintje nem egyezik meg a saját jelszintjével.
- Az arbitráció több biten keresztül folytatódhat



Órajel megnyújtása

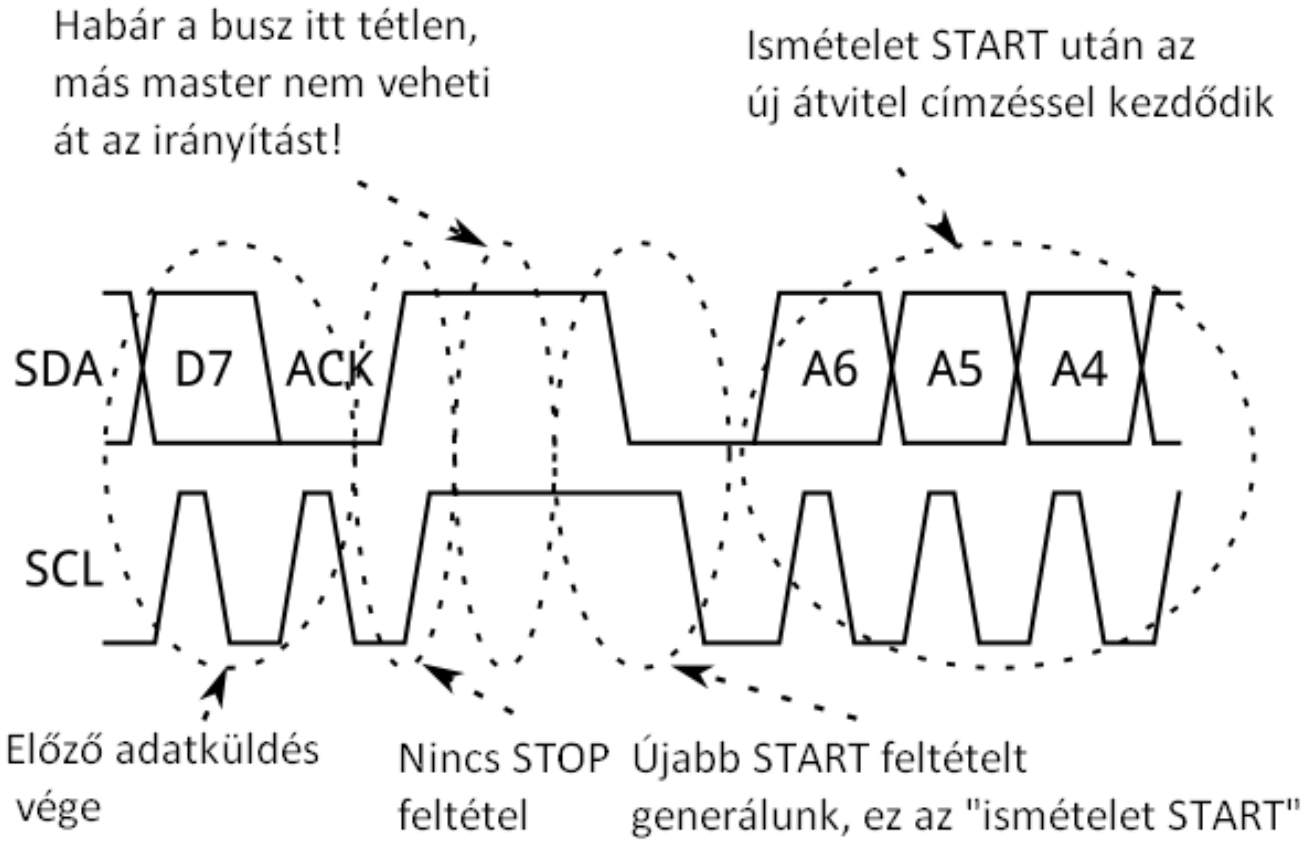
Az órajelet a mester generálja, de előfordulhat, hogy egy slave eszköz nem képes együttműködni a diktált tempóval, s „időt kér”. A már említett órajelszinkronizációs mechanizmus használható fel erre. Ez azt jelenti, hogy egy eszköz képes lehet az adatbájtok gyors fogadására, de az adat eltárolására vagy átvitelhez való előkészítéséhez több időre van szüksége. A slave ilyenkor a fogadás és a nyugtázás után **alacsony szinten tarthatja** az **SCL** vezetéket, hogy a mastert várakozási állapotba kényszerítse, amíg a slave kész nem lesz a következő bájt átvitelére, mint egy kézfogásos típusú eljárásban.

Nomál és gyors (fast) módban az adatátvitel bármelyik fázisában megnyújthatja a slave az órajelet. Nagysebességű (high-speed) módban azonban, ahol az órajelek fehúzásában aktív elemek is részt vesznek, csak a nyugtázó bit után és az azt követő adatbit előtt megengedett az SCL vonal lehúzása.



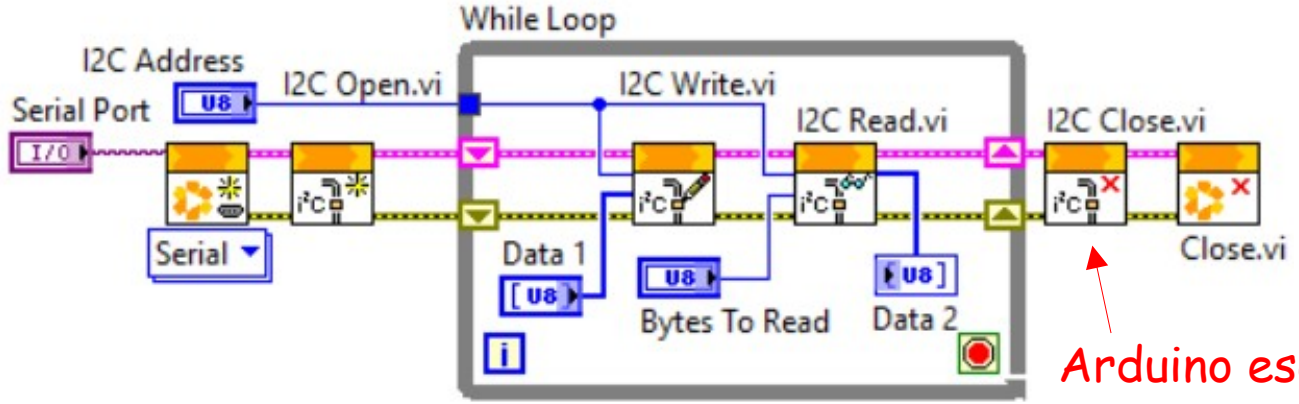
Ismételt START feltétel

Gyakran szükség van arra, hogy egy összetett tranzakciót egy master egy menetben végigvihessen. Ilyen eset lehet például egy eszköz valamely regiszterének vagy memóriaterületének megcímzése, majd onnan adatok beolvasása. Ha az adatküldés és adatfogadás között nem generálunk **STOP** feltételt, akkor a master magánál tudja tartani a busz feletti vezérlést. A **STOP** nélkül generált újabb **START** feltételt **ismételt START**-nak (repeated START) nevezzük. Az **ismételt START** feltételt újabb cím/parancs bájt küldése kell, hogy kövesse.



I2C eszközök kezelése LabVIEW környezetben

- **LabVIEW** környezetben az **Arduino** kártyához kapcsolt **I2C** eszközök kezelésére az alábbi lehetőségek kínálkoznak:
  - I2C-hez **egyedi Arduino firmware,** LabVIEW-val UART kapcsolat
  - **LINX firmware bővítése** egyedi I2C kiszolgáló parancsokkal
  - LINX firmware, **LabVIEW-ban szervezett I2C** tranzakciókkal
- Az első két megoldás előnye a gyors és hatékony **I2C** kiszolgálás, hátránya viszont a nehezebb hordozhatóság, az egyedi firmware
- A harmadik megközelítés, amellyel a mai előadásban foglalkozunk, fő előnye a hordozhatóság és hogy nem kell Arduino programot írni. Hátránya viszont a lassúbb kiszolgálás (az I2C tranzakciók adatait a soros porton keresztül is át kell küldeni, a LINX protokollal)

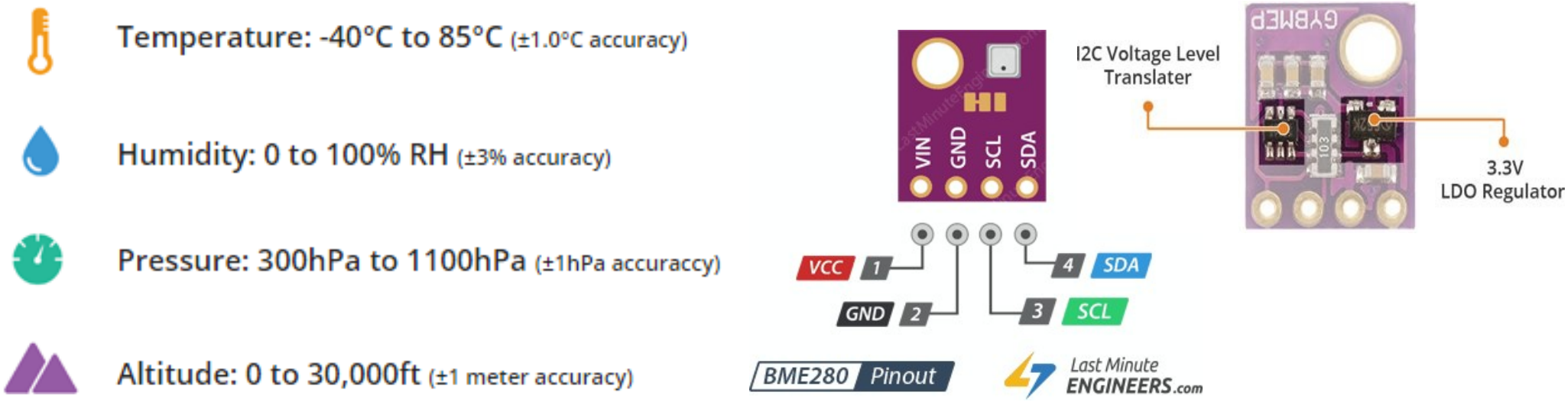


Egy tipikus **LabVIEW** alkalmazás szerkezete I2C szenzor rendszeres Kiolvasására, standard LINX firmware-rel

Arduino esetén ez nem kell!

A BME280/BMP280 I2C szenzor használata

- A Bosch gyártmányú **BME280** szenzor nyomást, hőmérsékletet és relatív páratartalmat mér, a **BMP085, BMP180, BMP183** utódja
- A **BMP280** is hasonló felépítésű, csak páratartalmat nem mér
- A szenzor **SPI** és **I2C** interfésszel rendelkezik, mi egy 5 V-tal és 3,3 V-tal egyaránt használható, **I2C** bekötésű modult használtunk
- Az **I2C** cím átforrasztással változtatható: **0x76** vagy **0x77**



Az ábrák forrása: lastminuteengineers.com/bme280-arduino_tutorial

BME280 szenzor főbb jellemzői

Key features

- Package — 2.5 mm x 2.5 mm x 0.93 mm metal lid LGA
- Digital interface — I²C (up to 3.4 MHz) and SPI (3 and 4 wire, up to 10 MHz)
- Supply voltage — V_{DD} main supply voltage range: 1.71 V to 3.6 V; V_{DDIO} interface voltage range: 1.2 V to 3.6 V
- Current consumption — 1.8 µA @ 1 Hz humidity and temperature; 2.8 µA @ 1 Hz pressure and temperature; 3.6 µA @ 1 Hz humidity, pressure and temperature; 0.1 µA in sleep mode
- Operating range — -40…+85 °C, 0…100 % rel. humidity, 300…1100 hPa
- Humidity sensor and pressure sensor can be independently enabled / disabled
- Register and performance compatible to Bosch Sensortec BMP280 digital pressure sensor
- RoHS compliant, halogen-free, MSL1

Key parameters for humidity sensor

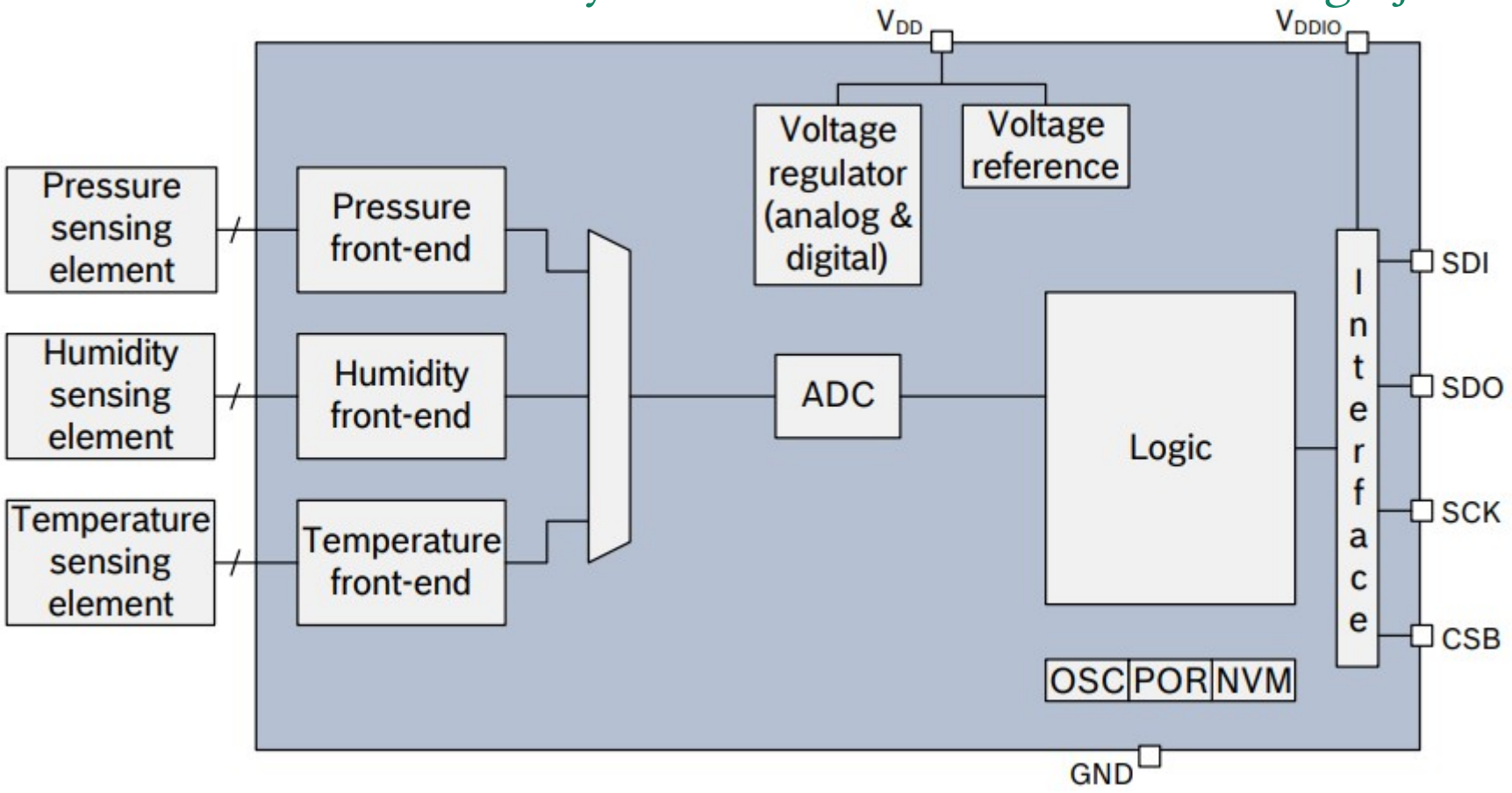
- Response time ($\tau_{63\%}$) — 1 s
- Accuracy tolerance — ±3 % relative humidity
- Hysteresis — ±1% relative humidity

Key parameters for pressure sensor

- RMS Noise — 0.2 Pa, equiv. to 1.7 cm
- Offset temperature coefficient — ±1.5 Pa/K, equiv. to ±12.6 cm at 1 °C temperature change

A BME280 szenzor felépítése

- Az ábrán a szenzor felépítésnek vázlata látható
- Az NVM jelzésű „nem felejtő” memória tartalmazza a nyers adatok átszámításához szükséges kalibrációs adatokat
- A **BME280** szenzor elvileg **SPI** és **I2C** üzemmódban is használható, de az általunk használt kártya csak az **I2C** üzemmódot támogatja



Kalibrációs konstansok

- A szenzorból kiolvasott adatok az ADC konverziók nyers eredményei, amelyeket még korrigálni kell az adatlapban megadott **algoritmus** és a szenzorból kiolvasható **kalibrációs adatok** felhasználásával
- A mellékelt táblázat azt mutatja be, hogy a kalibrációs konstansokat milyen regisztercím melyik bitcsoportjából lehet kiolvasni (a H1...H6 konstansok nem folytonosan helyezkednek el)

Register Address	Register content	Data type
0x88 / 0x89	dig_T1 [7:0] / [15:8]	unsigned short
0x8A / 0x8B	dig_T2 [7:0] / [15:8]	signed short
0x8C / 0x8D	dig_T3 [7:0] / [15:8]	signed short
0x8E / 0x8F	dig_P1 [7:0] / [15:8]	unsigned short
0x90 / 0x91	dig_P2 [7:0] / [15:8]	signed short
0x92 / 0x93	dig_P3 [7:0] / [15:8]	signed short
0x94 / 0x95	dig_P4 [7:0] / [15:8]	signed short
0x96 / 0x97	dig_P5 [7:0] / [15:8]	signed short
0x98 / 0x99	dig_P6 [7:0] / [15:8]	signed short
0x9A / 0x9B	dig_P7 [7:0] / [15:8]	signed short
0x9C / 0x9D	dig_P8 [7:0] / [15:8]	signed short
0x9E / 0x9F	dig_P9 [7:0] / [15:8]	signed short
0xA1	dig_H1 [7:0]	unsigned char
0xE1 / 0xE2	dig_H2 [7:0] / [15:8]	signed short
0xE3	dig_H3 [7:0]	unsigned char
0xE4 / 0xE5[3:0]	dig_H4 [11:4] / [3:0]	signed short
0xE5[7:4] / 0xE6	dig_H5 [3:0] / [11:4]	signed short
0xE7	dig_H6	signed char

Regiszter térkép

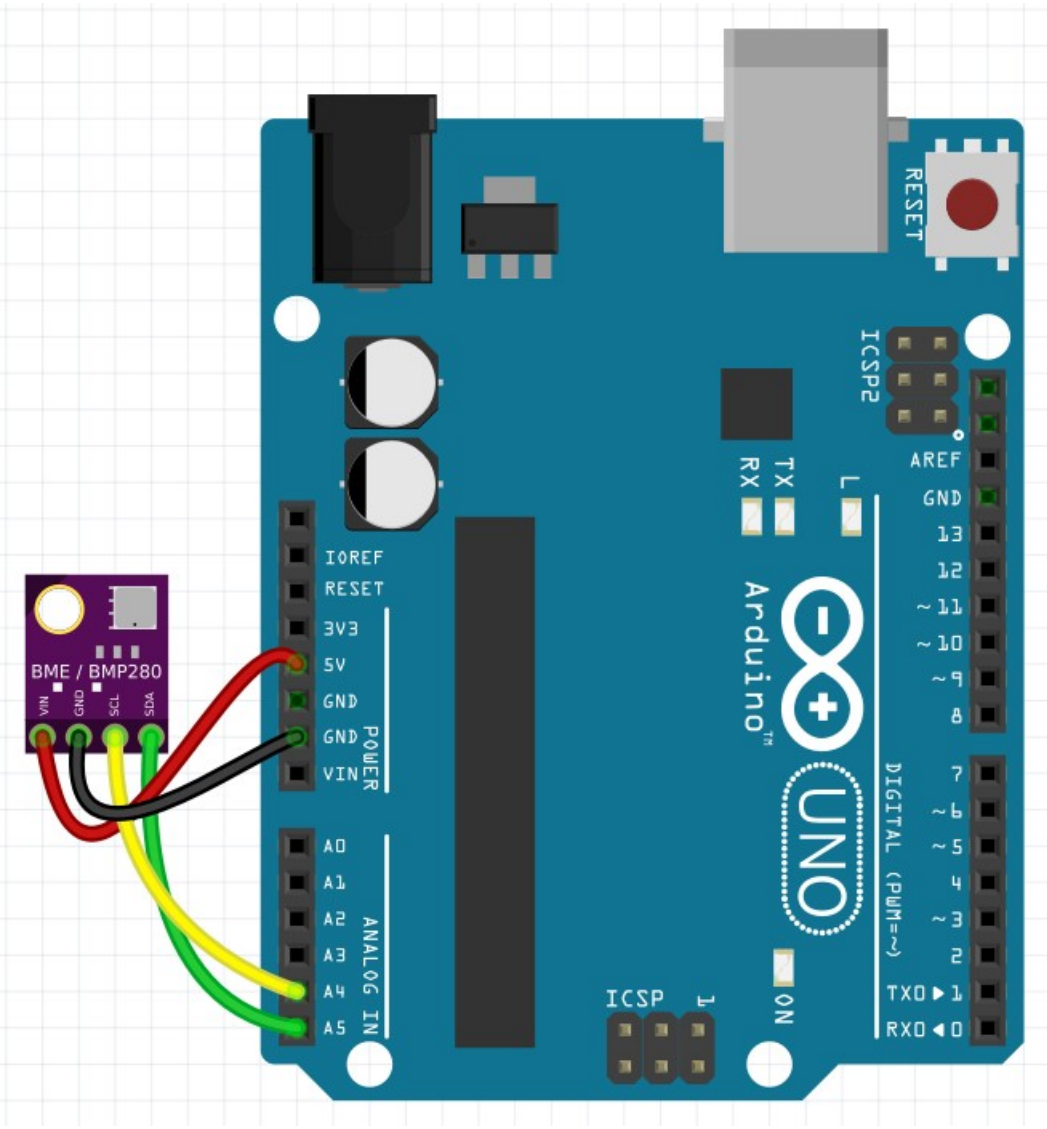
- A nyomás és a hőmérséklet 20 bites, a páratartalom 16 bites adat
- Az ***osrs_x*** bitek a túlmintavételezés megadására szolgálnak
- Az **0xE0** regiszterbe írt **0xB6** érték újraindítja a szenzort (***soft reset***)
- A **0xF5** vezérlő regiszter 0. bitjét ne írjuk 1-be, mert SPI módba kapcsol!

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
hum_lsb	0xFE	hum_lsb<7:0>								0x00
hum_msb	0xFD	hum_msb<7:0>								0x80
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
ctrl_hum	0xF2						osrs_h[2:0]			0x00
calib26..calib41	0xE1…0xF0	calibration data								individual
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x60
calib00..calib25	0x88…0xA1	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Chip ID	Reset
Type:	do not change	read only	read / write	read only	read only	read only	write only

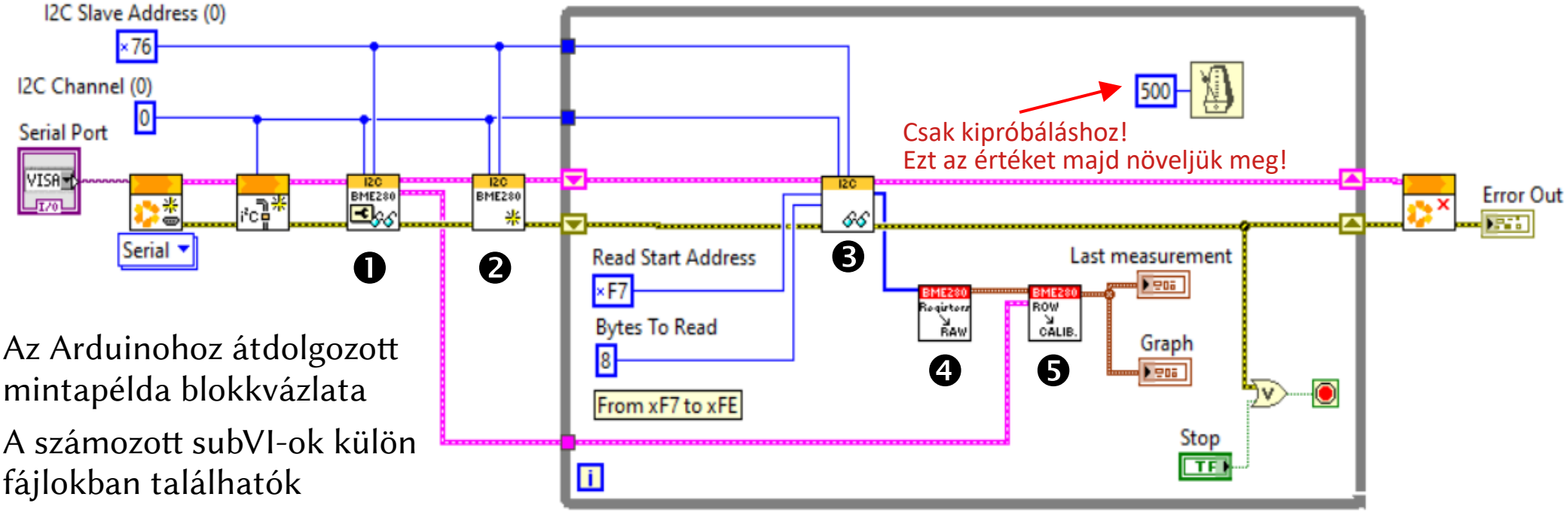
Olvassuk ki a BME280 szenzor adatait!

- A **BME280** szenzort egy **LINX** firmware-rel ellátott **Arduino** kártyához kötjük
- A **BME280** szenzor 3,3 V-os és 5V-os kivitelű kártyán is kapható (utóbbi esetben saját feszültség-stabilizátor biztosítja a 3,3 V-os tápfeszültséget), eszerint kössük **VCC**-t az Arduino kártya 3,3 V-os, vagy 5 V-os kimenetére
- **GND**-t az Arduino **GND**-re
- Az **SDA** kivezetést az Arduino **A4** jelzésű kivezetésével, az **SCL** kivezetést pedig az Arduino **A5** jelzésű kivezetésével kössük össze



Arduino_I2C_BME280_read_example.vi

- A **GitHub**on található egy szépen kidolgozott **LabVIEW** mintapélda: BME280 LabVIEW code for Raspberry Pi, aminek az Arduino kártyára adaptált, módosított változatát mutatjuk be
- A fő különbség, hogy itt nem lokális **I2C** csatornát kezelünk, hanem a soros porton keresztül **LINX** távkapcsolattal kezeljük az I2C eszközt
- További eltérés, hogy **Arduino** esetén csak 0. sorszámú **I2C** csatorna van és az **I2C Close** funkció nem támogatott (nincs is rá szükség)

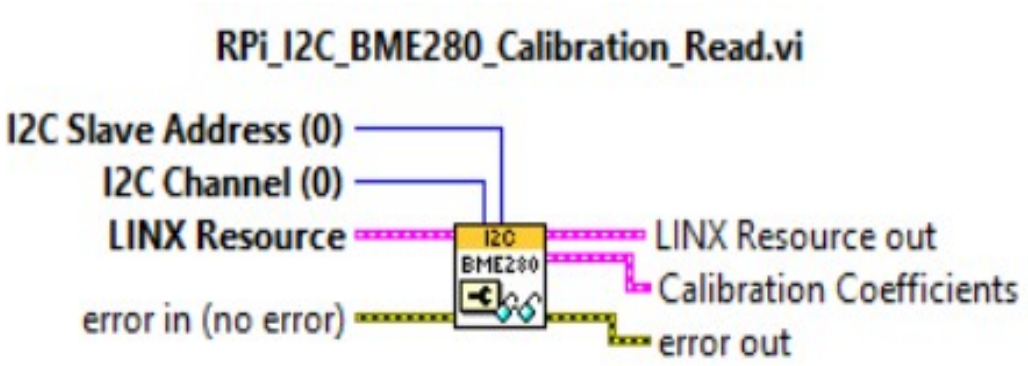


Az Arduinohoz átdolgozott mintapélda blokkvázlata

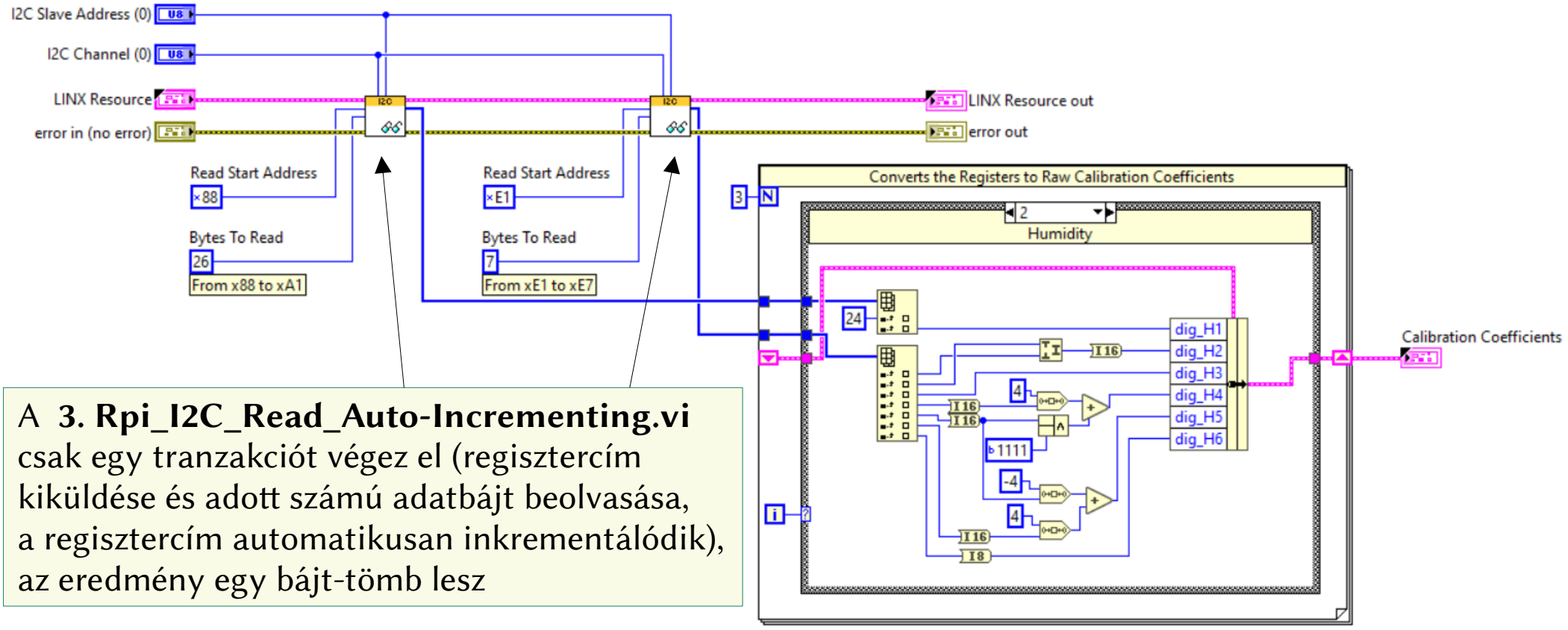
A számozott subVI-ok külön fájlokban találhatók

1. Rpi_I2C_BME280_Calibration_Read.vi

- A kalibrációs adatokat a 0x88–0xA1 és az 0xE1–0xE7 regisztercím területekről kell kiolvasni és három csoportba szétválogatni (T1-T3, P1-P9, H1-H6)



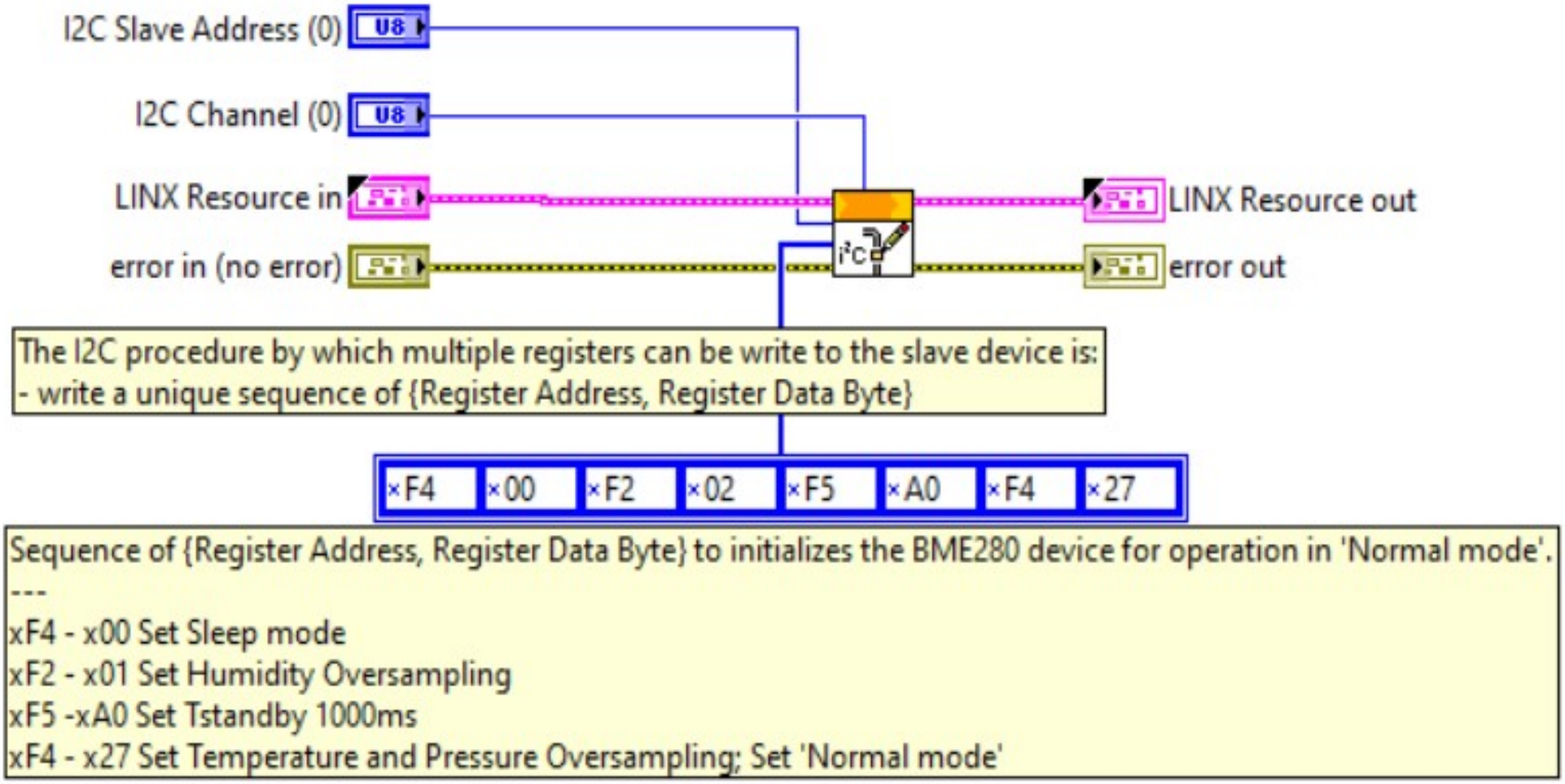
- Az adatok felhasználására majd az Rpi_BME280_Raw_Measures_to_Calibrated_Measures_Conversion.vi subVI-ban kerül sor



A **3. Rpi_I2C_Read_Auto-Incrementing.vi** csak egy tranzakciót végez el (regisztercím kiküldése és adott számú adatbájt beolvasása, a regisztercím automatikusan inkrementálódik), az eredmény egy bájt-tömb lesz

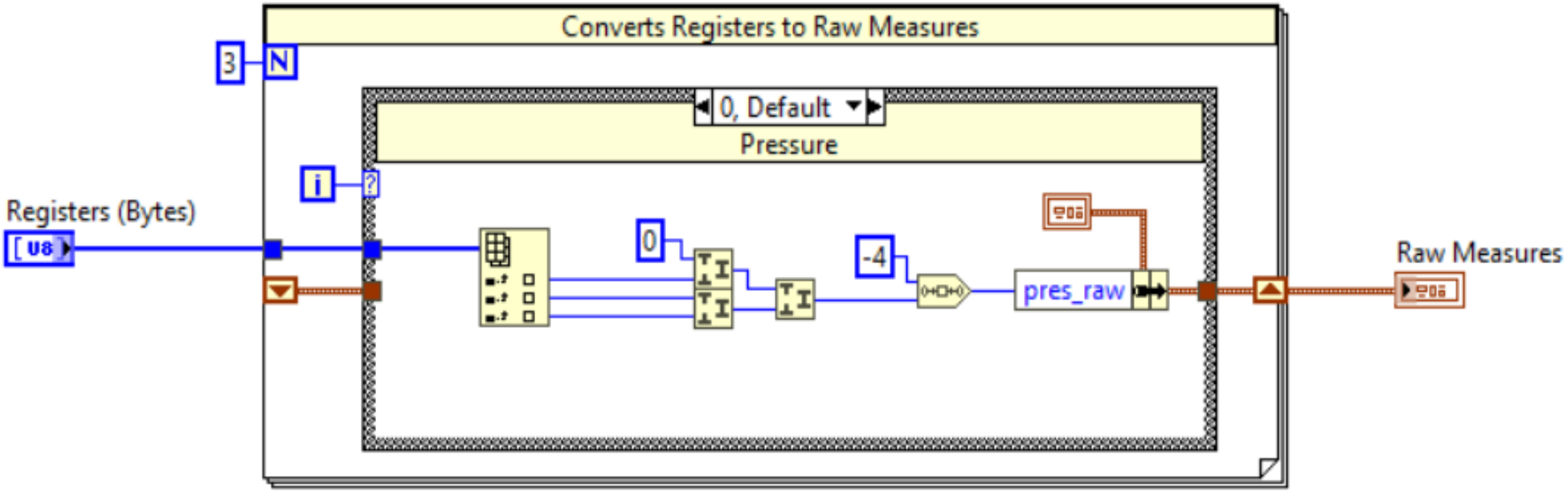
2. RPi_I2C_BME280_Device_Intialize.vi

- Ez a *subVI* a **BME280** szenzor üzemmódjának beállítására szolgál
- Szükség esetén módosíthatjuk az adatokat…



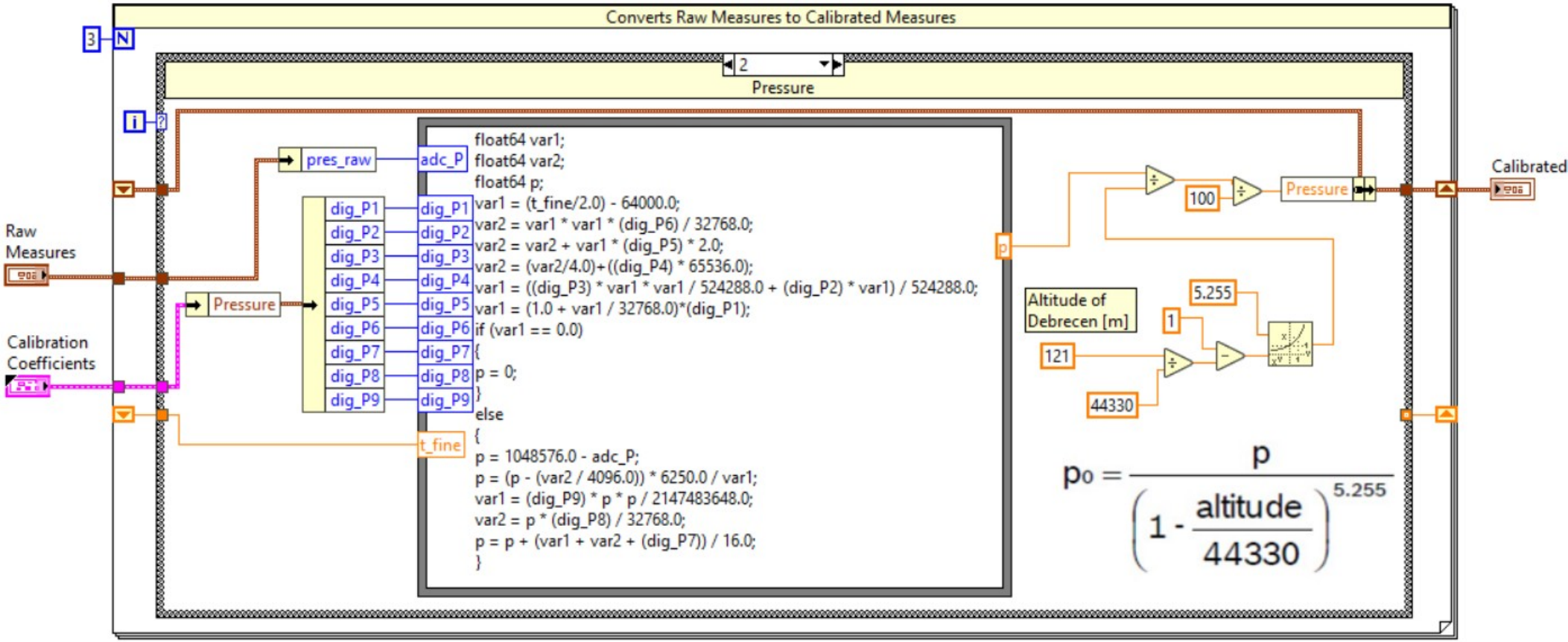
4. RPi_BME280_Registers_to_Raw_Measures_Conversion.vi

- Az adatok kiolvasása (8 bájt az 0xF7 regisztercímtől kezdődően) az **Rpi_I2C_Read_Auto-Incrementing.vi** modullal történik, amelyet korábban már említettünk
- A kiolvasásból kapott bájtok közül három-három a nyomás és a hőmérséklet adatát tartalmazza, a két utolsó bájt pedig a relatív páratartalom értékét adja meg, ezeket gyűjti ki a Raw Measures tömbbe ez a *subVI*, az alábbiak szerint



5. Rpi_BME280_Raw_Measures_to_Calibrated_Measures_Conversion.vi

- A nyers adatokból a kalibrációs konstansokkal az adatlapban található algoritmus segítségével számoljuk ki a tényleges értékeket (**t_fine** az első ciklusban kiszámolt paraméter)
- A nyomásértéknél beleírtuk a tengerszintre történő átszámítást is (Debreceni magasságadattal), bár ezt nem itt és nem így kellene…



Helyi légnyomás átszámítása tengerszintre

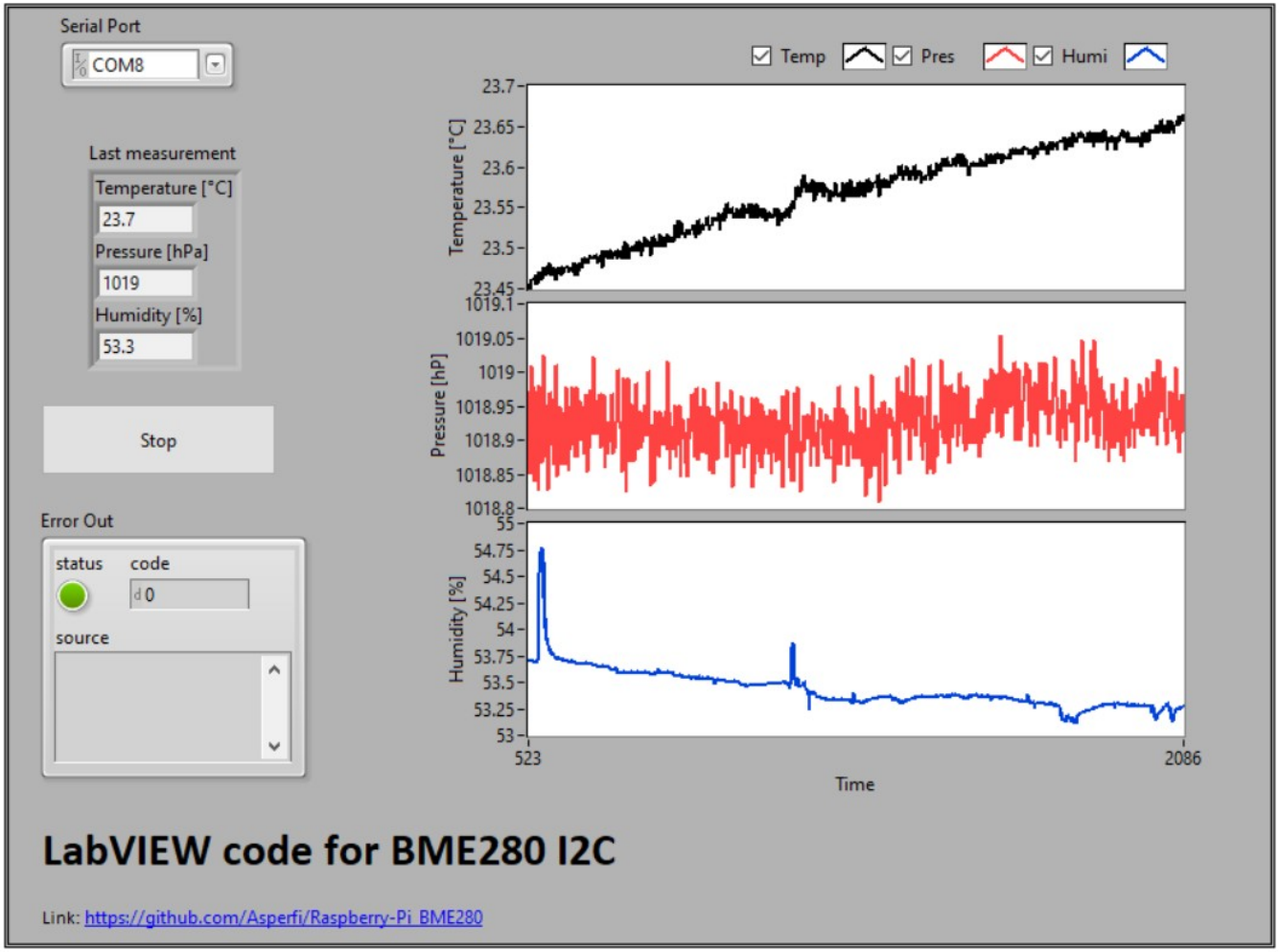
- Tudnunk kell a helyi tengerszint feletti magasságot (altitude) és a szenzorral meg kell mérnünk az abszolút helyi nyomás értékét (p).
- A tengerszintre átszámított légnyomás (p_0) a következő képlettel számítható ki:

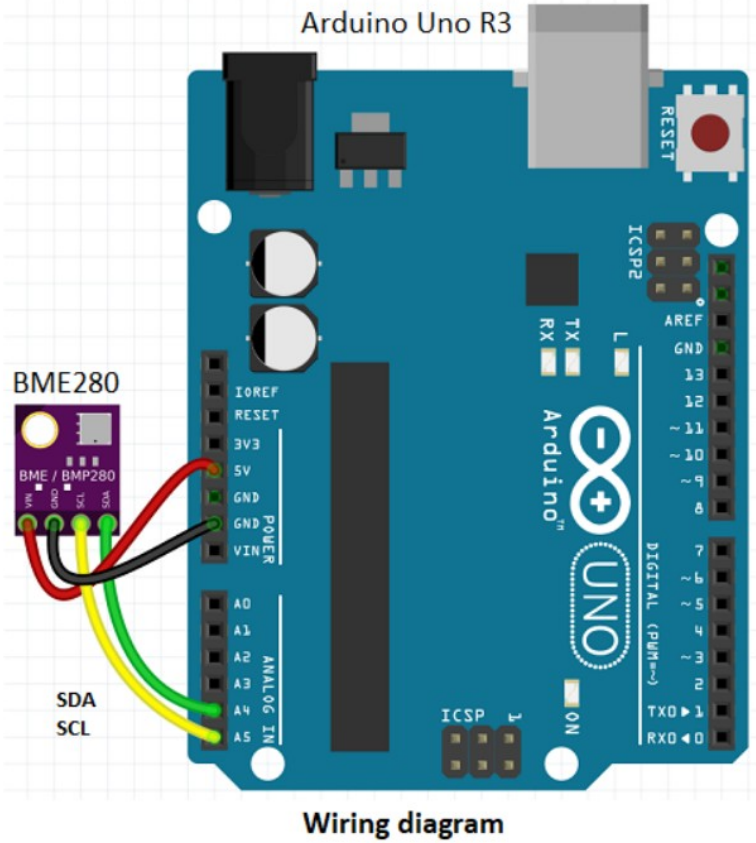
$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

- **Egyszerű(bb) közelítések, fix helyen:**
  - Debrecenben (kb. 120 m) 1440 Pa-t hozzáadunk a mért légnyomáshoz
  - Debrecenben (kb. 120 m) a mért értéket megszorozzuk 1.014346-tal

Arduino_I2C_BME280_read_example.vi futtatása

- A program egy (nem tipikus) futási eredménye az ábrán látható
- Megfigyelhetjük, hogy az automatikusa skálázás nem mindig kedvező
- A mintavételezések idejét a gyakorlatban hosszabbra kell venni



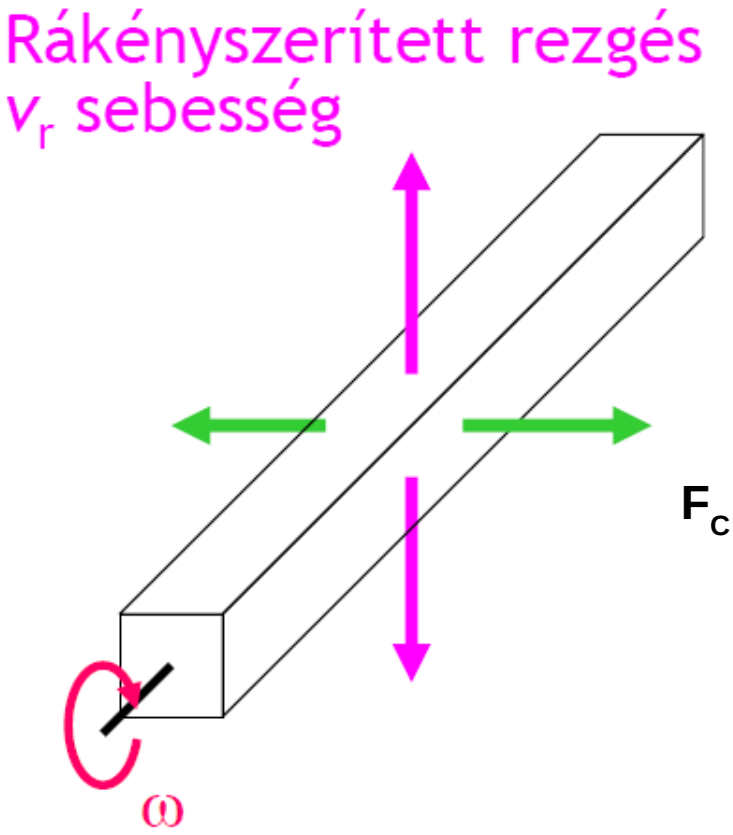


Wiring diagram

Rezgőelemes giroszkóp

- Alapelv: rugalmas rúd kényszerrezgése, Coriolis erő
- ω szögsebességű forgás esetén a fellépő $\mathbf{F}_c$ Coriolis erő a rákényszerített rezgésre merőleges deformációt okoz

  ⟶ Mérhető

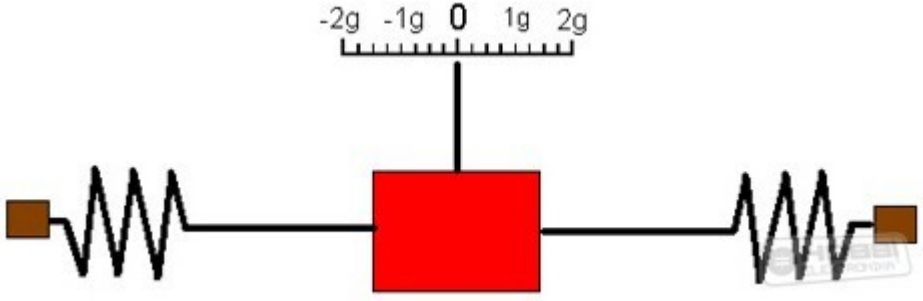


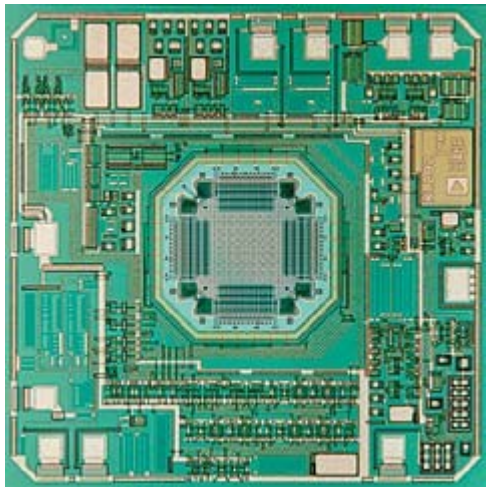
- Mérés elve szerint:
  - Piezoelektromos hatás
  - Kapacitív elvű elmozdulás-mérés
- Az alkalmazott technológia szerint:
  - Piezokeramikus kristály
  - **MEMS** – Micro ElectroMechanical System

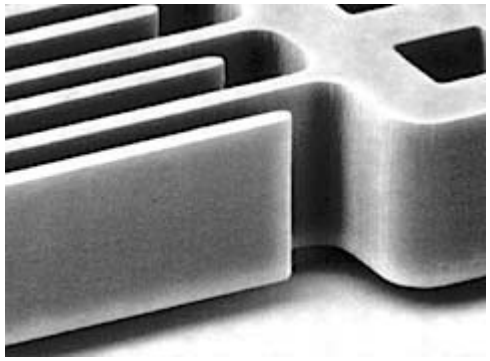
$$F_c = -m\left(\omega \times v_r \cdot L\right)$$

Gyorsulás érzékelők

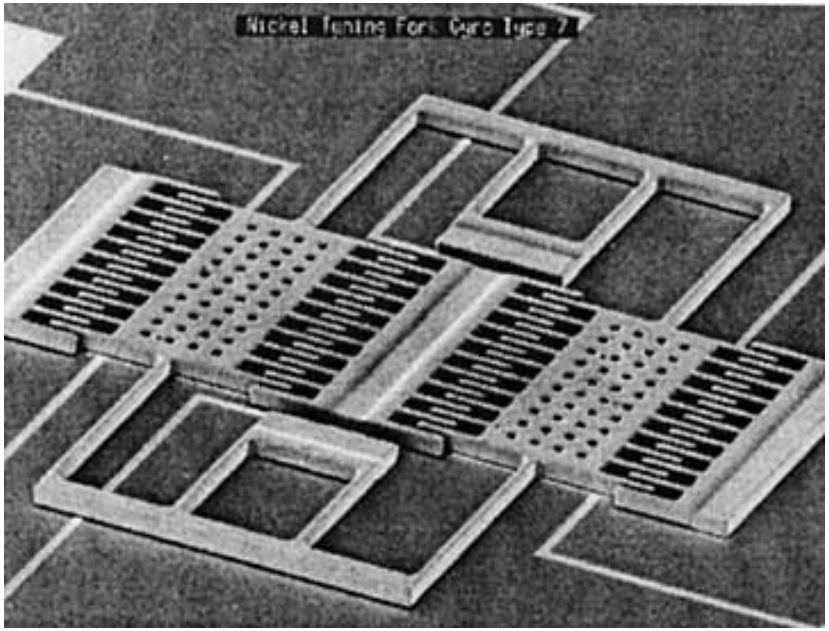
- Fizikai elv: rugó – tömeg együttes
- Érzékelés elve
  - Piezokeramikus
    Érzékelés piezoelektromos elven
  - **MEMS**
    - Kapacitív
    - Termikus







20 µm-es Si fésű struktúra (Draper Lab)



A Draper Lab fésű struktúrával kialakított szenzora

Analog Devices ADXL 50 gyorsulásmérő

Gyorsulás érzékelők

- A gyorsulás vektoriális egyenlete Newton törvénye alapján:

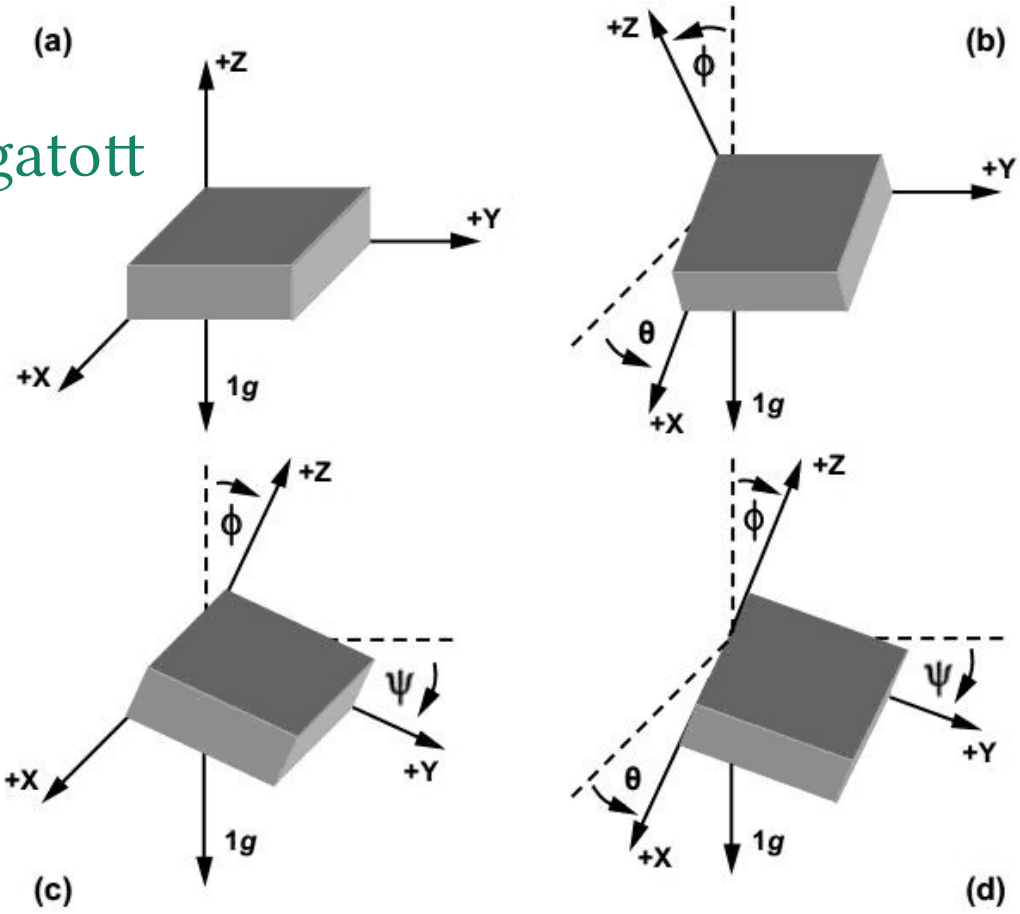
$$\mathbf{a}_m = \frac{1}{m}(\mathbf{F} - \mathbf{F}_g),$$ ideális esetben **F** = 0 (csak a gravitáció hat)

ahol $\mathbf{a}_m$ a gyorsulás, **m** a tömeg, **F** a tömegre ható erők eredője, $\mathbf{F}_g$ pedig a gravitációs erő

- A nehézségi erő a szenzor elforgatott rendszerében

$$\mathbf{F}_g = R_I^B \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = \begin{pmatrix} -mg\sin(\theta) \\ mg\cos(\theta)\sin(\phi) \\ mg\cos(\phi)\cos(\theta) \end{pmatrix}$$

$$\mathbf{a}_m = \begin{pmatrix} g\sin(\theta) \\ -g\cos(\theta)\sin(\phi) \\ -g\cos(\phi)\cos(\theta) \end{pmatrix}$$



Gyorsulás érzékelők

- A gyorsulás mért értékeiből meghatározhatjuk a θ emelkedési szöget (pitch) és a ϕ orsózási szöget (roll)

$$\hat{\theta}_{accel} = \arcsin\left(\frac{a_{m,x}}{g}\right),$$

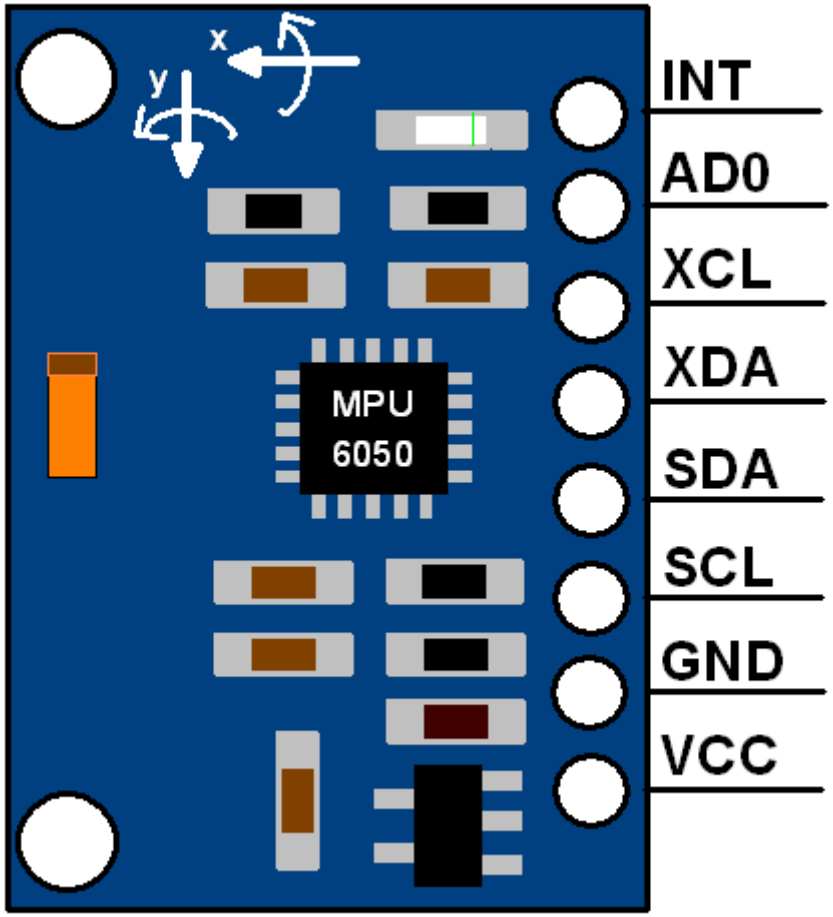
$$\hat{\phi}_{accel} = \arctan\left(\frac{a_{m,y}}{a_{m,z}}\right)$$

A ^jel itt azt jelzi, hogy becsült értékről van szó, melynek meghatározásába beleszól minden zavar, rázkódás, zaj

- A fenti képletekkel meghatározott értékek segítségével korrigálhatjuk a giroszkóp adataiból számolt szögeket az elcsúszás (drift) kordában tartására
- Az **Inerciális Mérőegységek** (IMU) egy tokban giroszkópot és gyorsulásmérőt is tartalmaznak

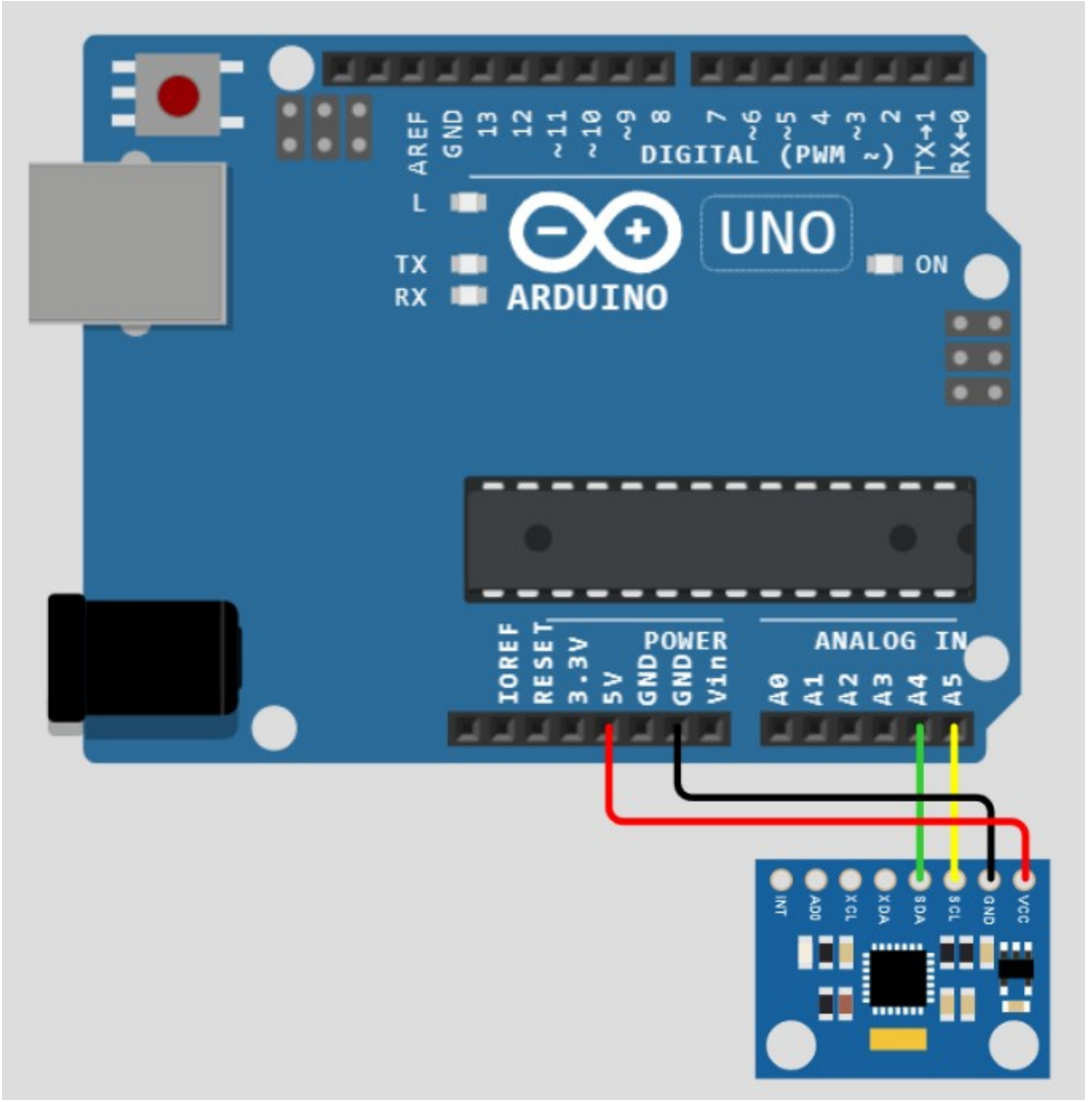
Az MPU6050 modul

- Gyorsulásmérőt és szögsebesség-mérőt tartalmaz (6-tengelyű)
- 16 bites regiszterek, I2C illesztő
- VCC = 5V, GND = közös pont
- SCL, SDA = I2C órajel és adat
- AD0 címvonal (0 vagy 1)
- INT megszakításkérő jel (jelez, ha van kiolvasható adat)
- XCL és XDA a szenzor kiterjesztését teszi lehetővé egy mágneses szenzor csatlakoztatásával



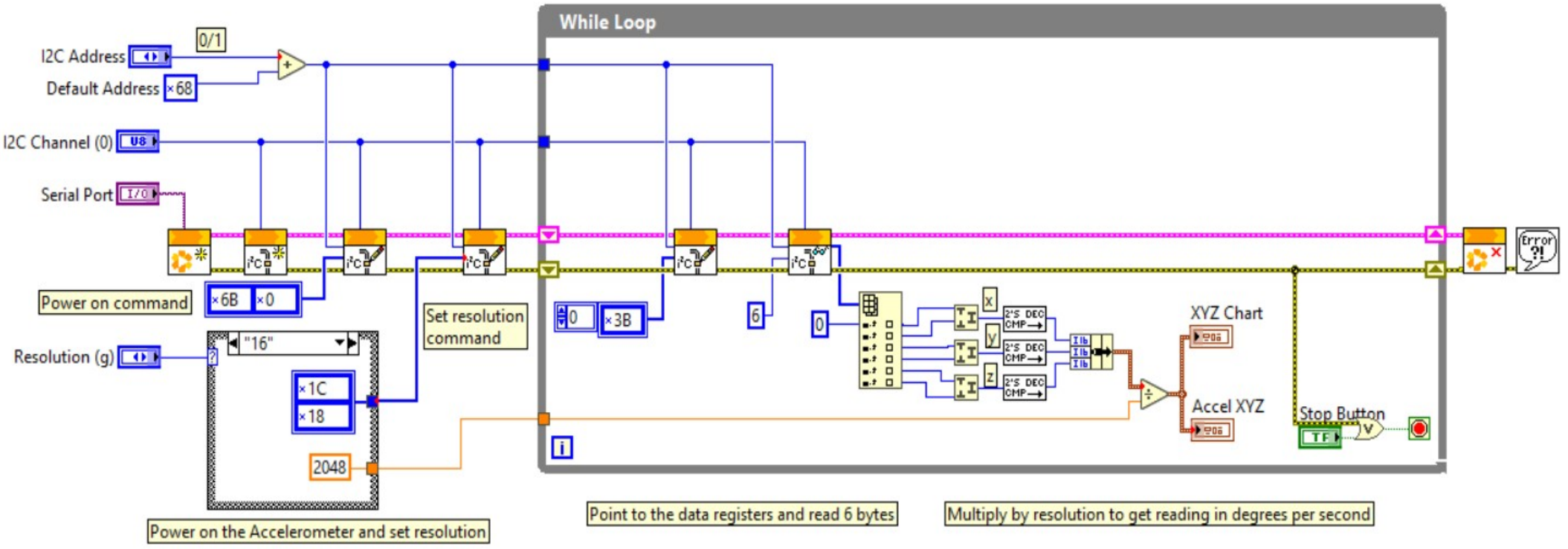
Bekötési vázlat

- Az **MPU6050** modul egy **LINX** firmware-rel ellátott **Arduino** kártyához kötjük
- Az **MPU6050** modul **VCC** kivezetését az Arduino 5 V-os kimenetére kötjük
- **GND**-t az Arduino **GND**-re
- Az **SDA** kivezetést az Arduino **A4** jelzésű kivezetésével, az **SCL** kivezetést pedig az Arduino **A5** jelzésű kivezetésével kössük össze



MPU6050_accel_test.vi

- A program eredeti változata az Instructables oldalán található: Sudharsan Sukumur How to Use I2C in LabVIEW
- Mivel az eredeti program más kártyához és más szenzorhoz készült, néhány kisebb módosításra volt szükség
- Itt most csak a gyorsulásmérő adatait olvassuk ki és inicializálásnál ennek érzékenységét állítjuk be (±2g, ±4g, ±8g, illetve ±16g választható)

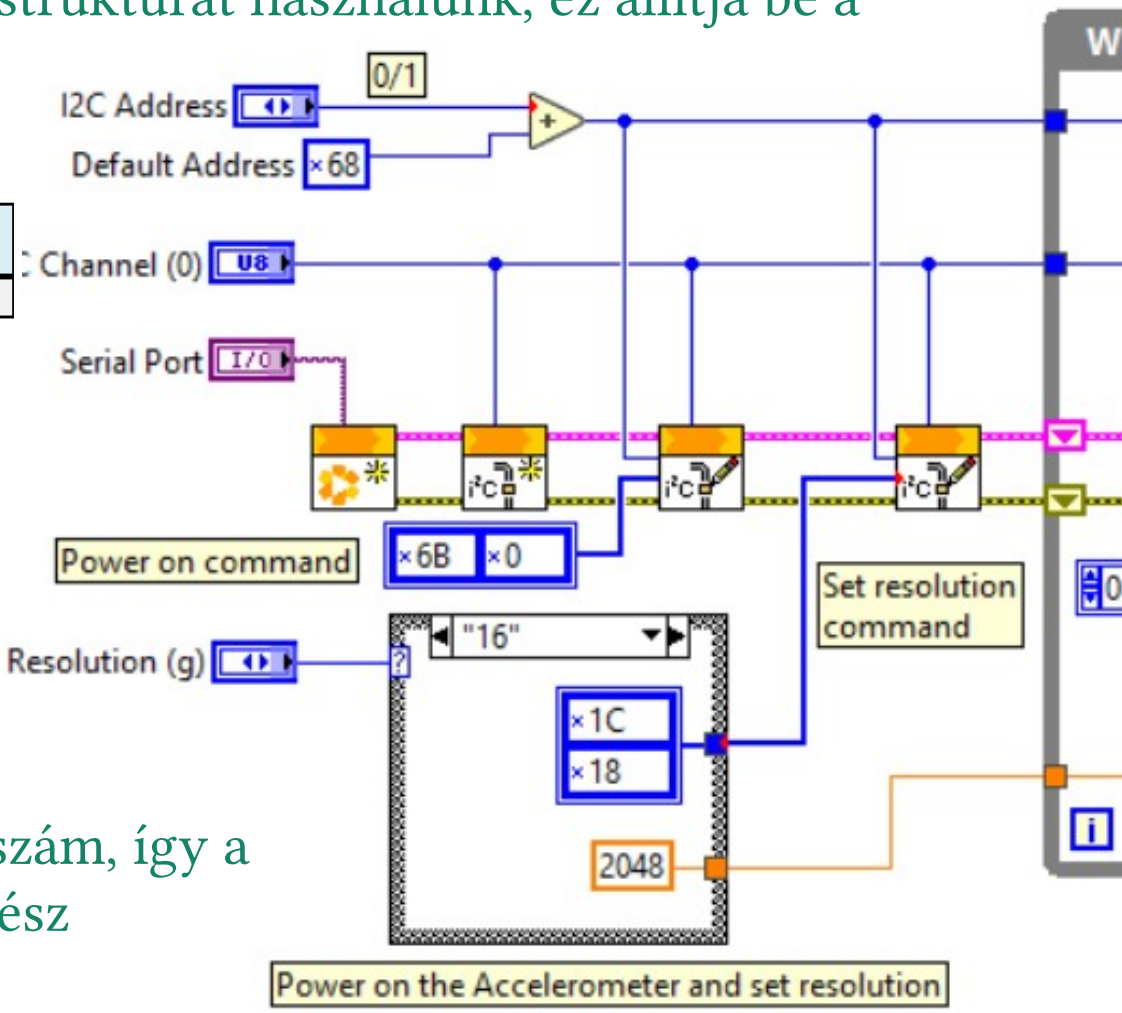


Az előkészítő rész

- **Az I2C cím** 0x68 vagy 0x69 lehet. Az előlapi választó csak egy 0/1 értéket ad vissza, ehhez tehát hozzá kell adni 0x68-at
- Az I2C busz inicializálása után bekapcsoljuk a szenzort **0x6B, 0x00** kiküldésével
- A felbontás beállításához egy Case struktúrát használunk, ez állítja be a kiküldendő parancs és a kalibráló konstans értékét

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

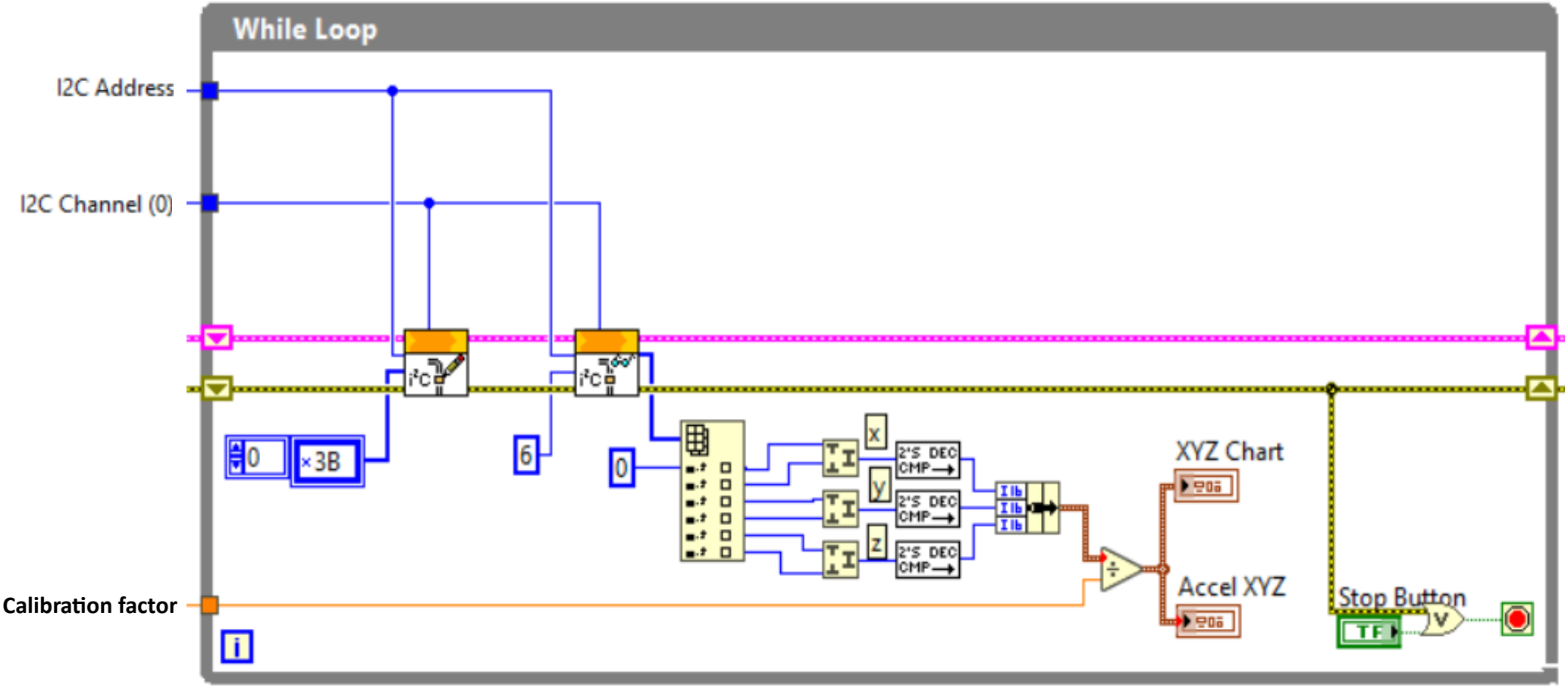
AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*



- A kiolvasott érték 16 bites előjeles szám, így a *±16g* méréshatár esetén 2048 skálarész jelent 1 g gyorsulást

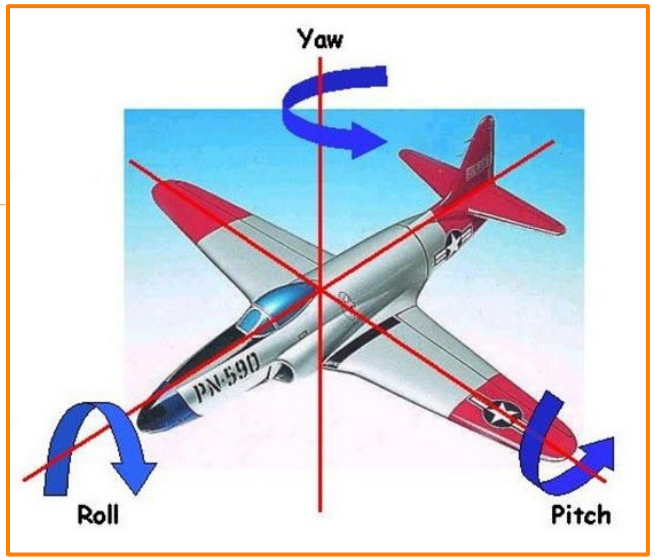
A ciklikus rész

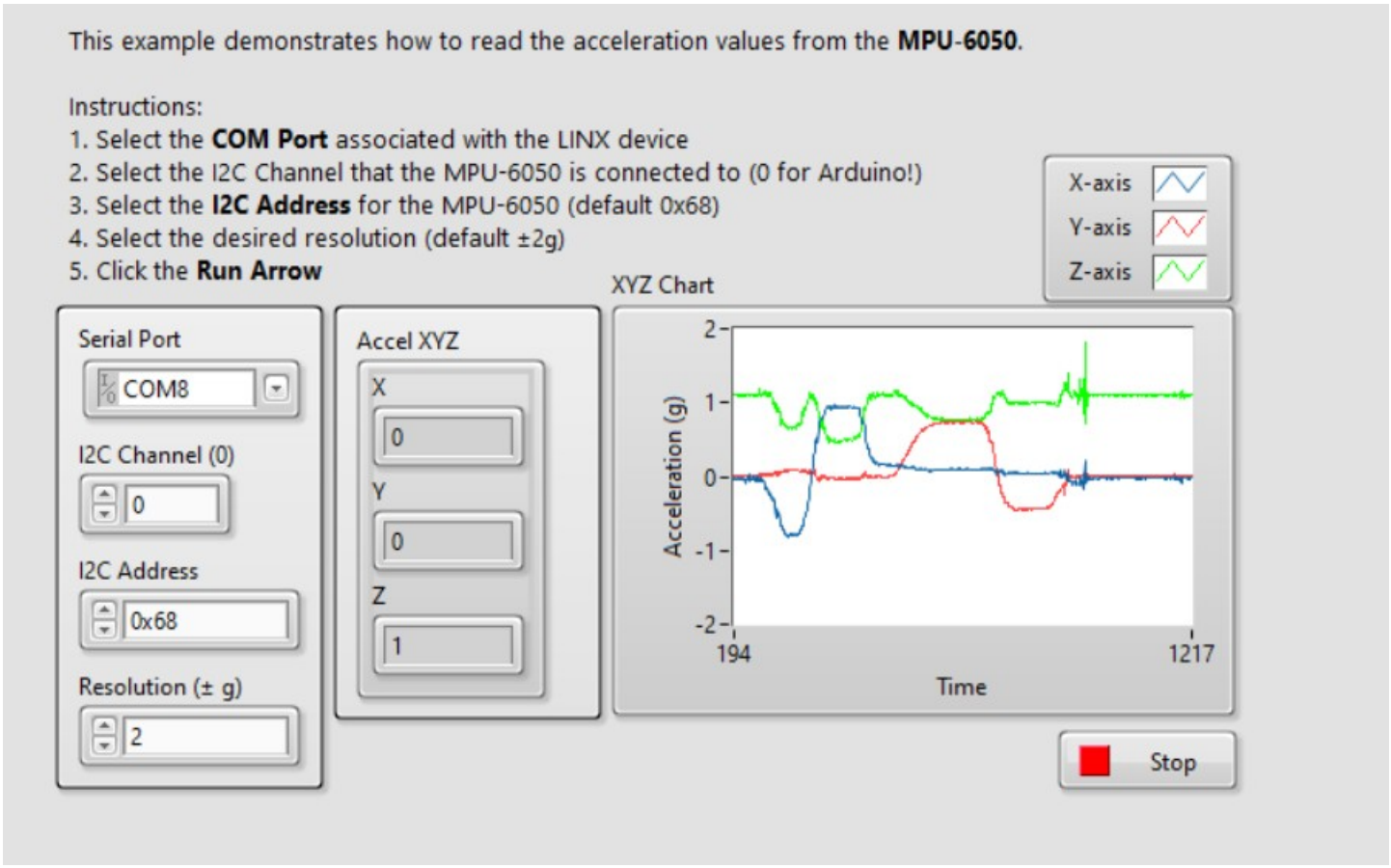
- Az adatok kiolvasásához a **0x3B** címtől kezdődően 6 bájtot kell lekérni, majd páronként egyesíteni és előjeles számmá konvertálni
- Az így kapott X, Y, Z komponenseket még el kell osztani a méréshatárhoz tartozó kalibrációs konstanssal, így g egységekben kapjuk meg az értékeket



MPU6050_accel_test.vi

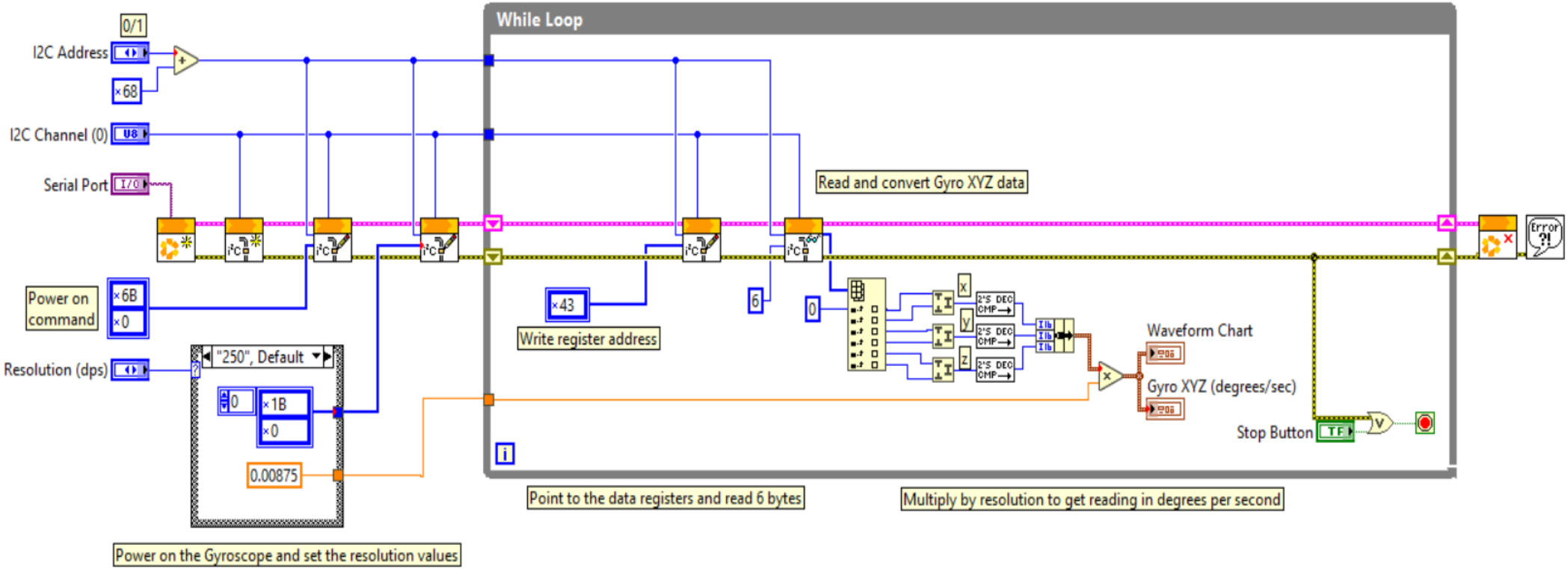
- Az ábrán forgatás (*roll*) és billentés (*pitch*) eredménye látható
- A ***z*** tengely körüli forgatás (*yaw*) nem mutat változást, mert az elfordulás nem változtat a gravitáció X, Y és Z komponenseinek értékén





MPU6050_gyro_test.vi

- Az előző program apró változtatásával most az **MPU6050** szenzor szögelfordulási sebesség X, Y, Z komponenseit tartalmazó regisztereket olvassuk ki
- Inicializálásnál az 0x1B regiszterben kell beállítani a kívánt érzékenységet (250, 500, 1000, vagy 2000 fok/s)

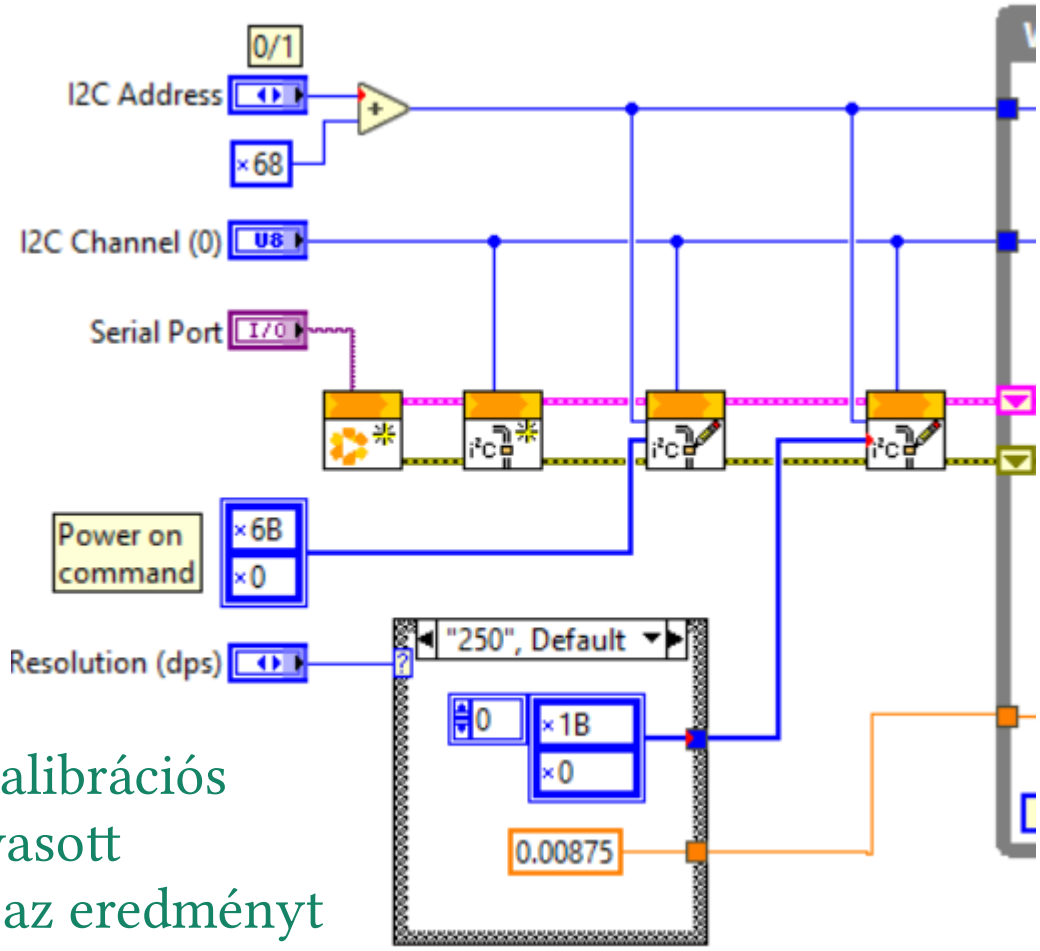


Az előkészítő rész

- **Az I2C cím** 0x68 vagy 0x69 lehet. Az előlapi választó csak egy 0/1 értéket ad vissza, ehhez tehát hozzá kell adni 0x68-at
- Az I2C busz inicializálása után bekapcsoljuk a szenzort **0x6B, 0x00** kiküldésével
- Az érzékenység beállításához egy Case struktúrát használunk, ez állítja be a kiküldendő parancs és a kalibráló konstans értékét

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

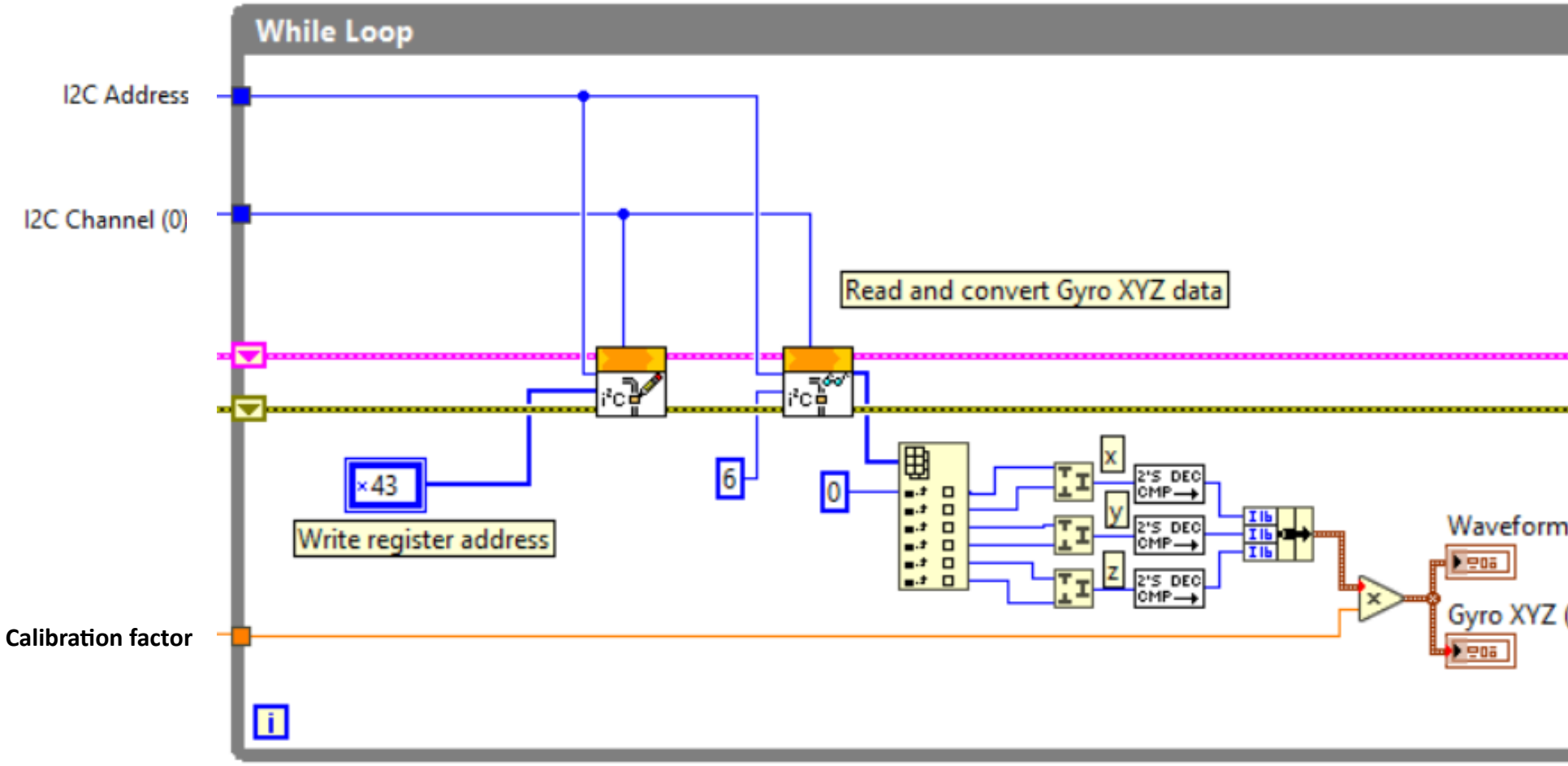
FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s



- A kiválasztott méréshatárhoz tartozó kalibrációs értékkel majd meg kell szorozni a kiolvasott adatokat, hogy °/s egységekben lássuk az eredményt

A ciklikus rész

- Az adatok kiolvasásához most a **0x43** címtől kezdődően kell 6 bájtot lekérni, majd páronként egyesíteni és előjeles számmá konvertálni
- Az így kapott X, Y, Z komponenseket meg kell szorozni a méréshatárhoz tartozó kalibrációs konstanssal, így °/s egységekben kapjuk meg az értékeket



MPU6050_gyro_test.vi

- A program egy futási eredménye az alábbi ábrán látható
- Ahhoz, hogy értékelhető szögelfordulási értéket kapjunk, a kapott szögsebesség értékeket még integrálni kellene…