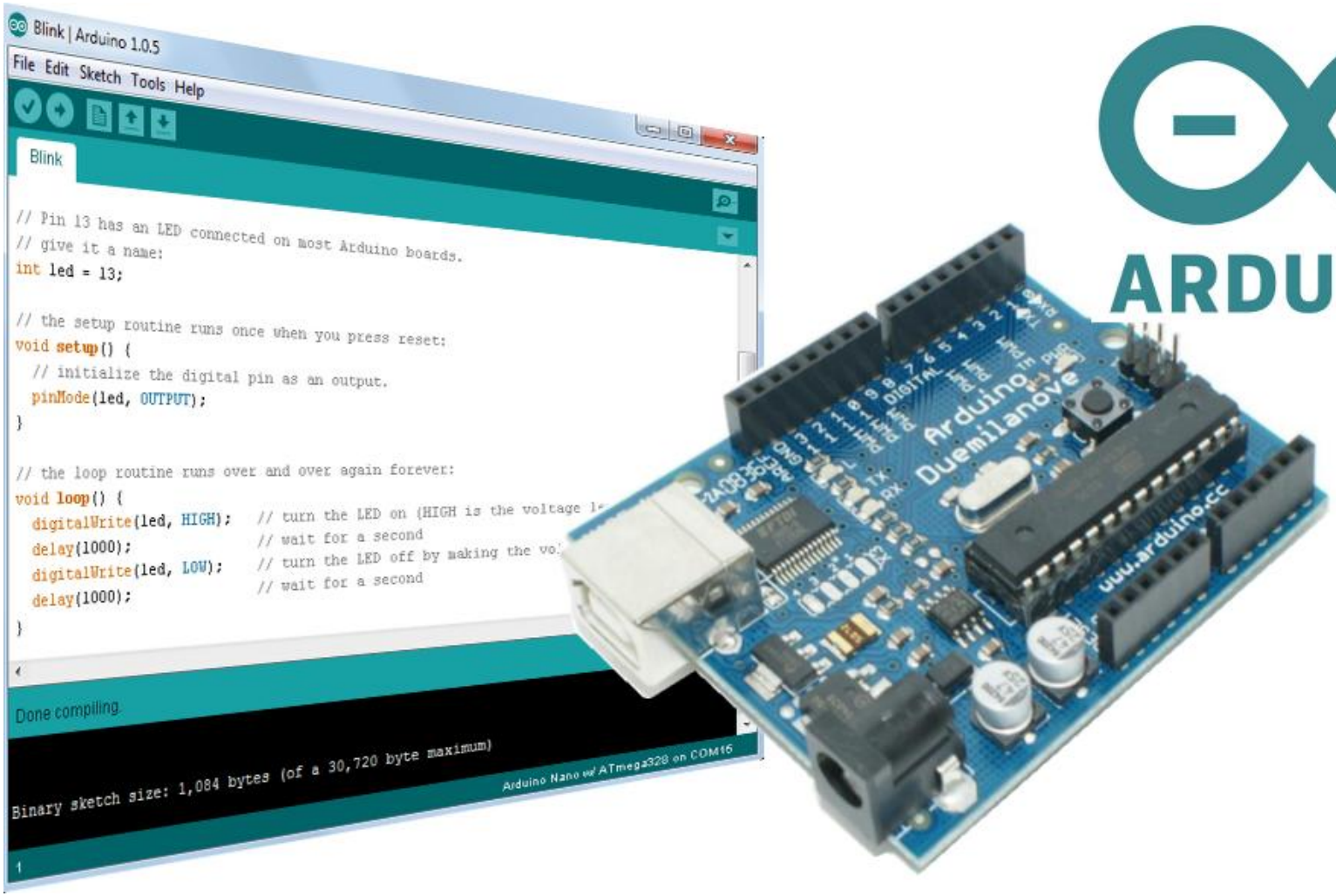


Bevezetés a mikrovezérlők programozásába:

Digitális szenzorok

Ajánlott irodalom

- Aduino LLC.: Arduino Language Reference
- ATMEL: ATmega328p mikrovezérlő adatlapja
- Brian W. Kernighan, Dennis Ritchie: A C programozási nyelv
- Cseh Róbert: Arduino programozási kézikönyv
- Ruzsinszki Gábor: Mikrovezérlős rendszerfejlesztés C/C++ nyelven I. – PIC mikrovezérlők
- Ruzsinszki Gábor: Mikrovezérlős rendszerfejlesztés C/C++ nyelven II. – Arduino

Lab 14 projektek

Sonar – Ultrahangos távolságmérés

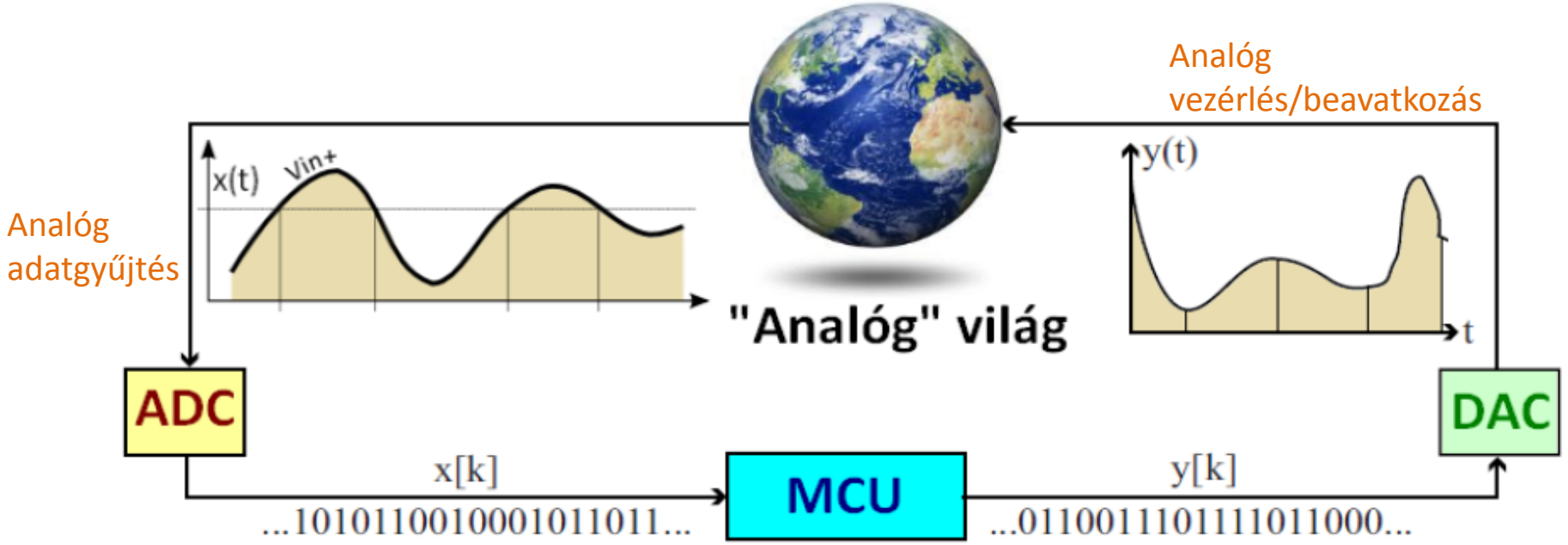
DHT22_test – Hőmérséklet és relatív páratartalom mérése DHT22 szenzorral

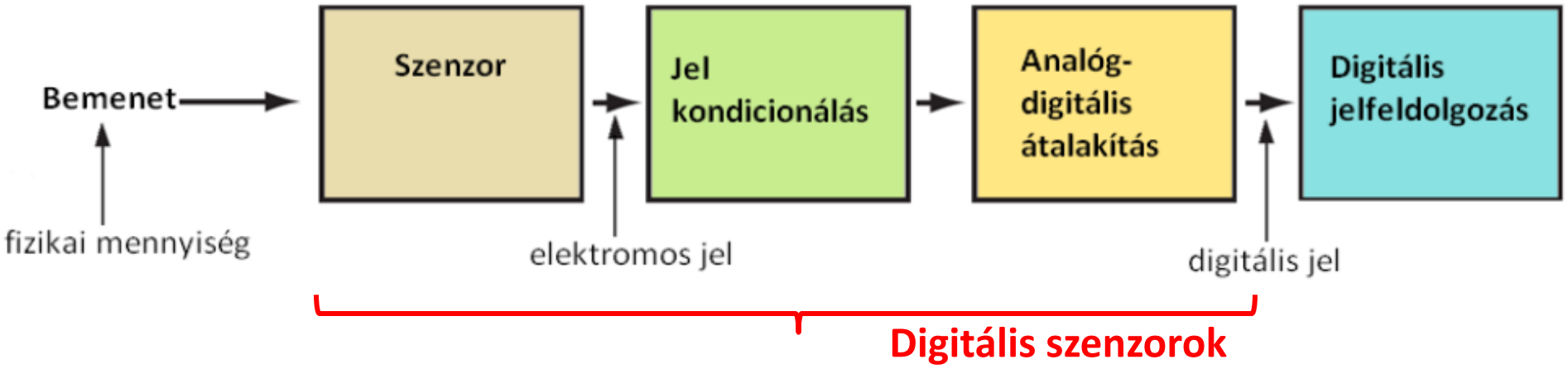
TRHlogger – Hőmérséklet és relatív páratartalom mérése, az eredmények megjelenítése és naplózása a PC-n

PressureSensor_hw – légnyomás mérése BMP180 szenzorral, hardveres I2C támogatással

BMP085test – légnyomás méréséből tengerszintfeletti magasság meghatározása

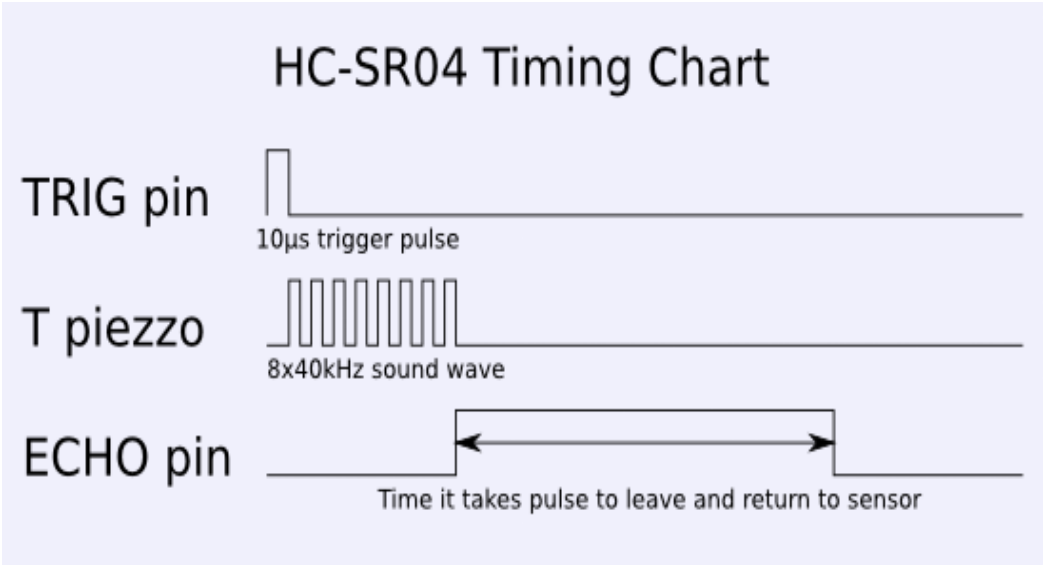
Digitális szenzorok





Ultrahangos távolságmérő

A **HC-SR04** modul piezo jeladója az indító impulzus hatására egy 40 kHz-es jelcsomagot sugároz ki. A modul digitális kimenő impulzusának szélessége megegyezik a visszaverődött hang terjedési idejével. Valójában ez tehát részben digitális, részben analóg szenzornak tekinthető…

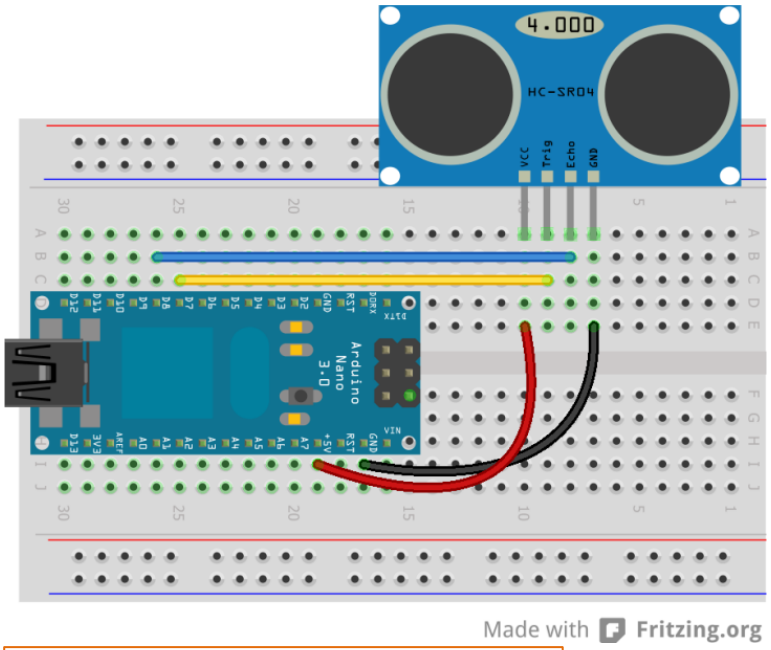


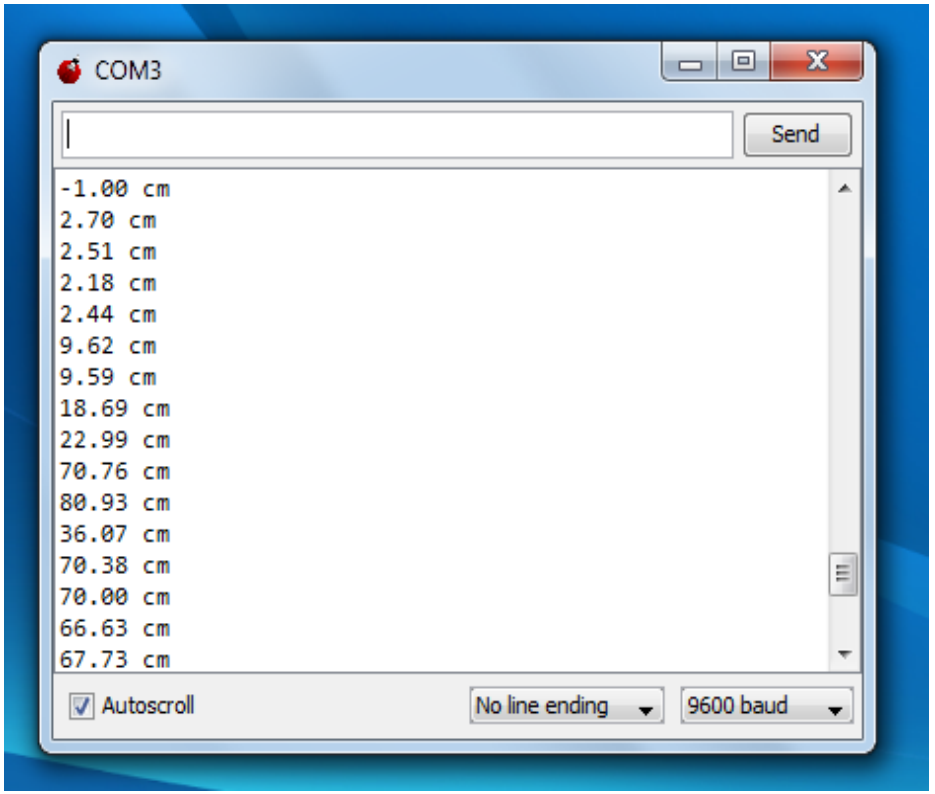
Főbb paraméterek

- Tápfeszültség: 4.5 V – 5.5 V
- Mérési tartomány: 2 cm – 4 m (gyakorlatban inkább 2 m)
- Érzékelési szögtartomány: ~ 16 °

Megjegyzés: Az MSP430 Launchpad kártya esetében a TP1 pontról vehetjük az 5 V-os tápfeszültséget, az ECHO jel fogadásánál pedig védenünk kell a bemenetet a túlfeszültségtől. Az Arduino kártya esetében természetesen nincs gond az illesztéssel!

Ultrahangos távolságmérő





	Arduino	MSP430
Trigger:	D7	P2_5
Echo:	D8	P2_4

unsigned long **pulseInt**(int pin, int value)

Meghatározza a beérkező impulzus szélességét (mikroszekundum egységekben).
pin – a vizsgált bemenet sorszáma
value – az vizsgálandó impulzus polaritása (HIGH vagy LOW)

Sonar.ino – 1/2. oldal

```
#define echoPin 8                 // Echo bemenet
#define trigPin 7                 // Trigger kimenet
#define RED_LED 13
```

Hardverfüggő rész

```
int maximumRange = 400;           // Legnagyobb távolság cm-ben
int minimumRange = 1;             // Minimális távolság cm-ben
long duration;                    // Időtartam [us]
float distance;                   // Távolság [cm]void

void setup() {
  Serial.begin(9600);             //Soros kapcsolat 9600 bit/s
  Serial.println("Sonar program");
  pinMode(echoPin, INPUT);        //Impulzus bemenet
  pinMode(trigPin, OUTPUT);       //Vezérlő kimenet
  digitalWrite(trigPin, LOW);     //Alaphelyzetben alacsony szint
  pinMode(RED_LED,OUTPUT);        //A beépített piros LED jelző funkciót lát el
  digitalWrite(RED_LED,HIGH);
}
```

Sonar.ino – 2/2. oldal

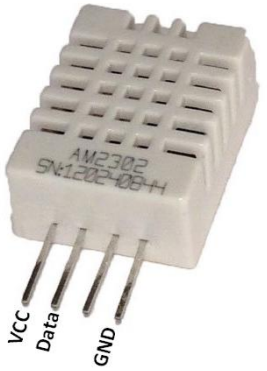
```
void loop() {
  delay(1000);                    //egy kis várakozás
  digitalWrite(RED_LED,HIGH);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH); //Bejövő impulzus szélességének meghatározása
  digitalWrite(RED_LED,LOW);
  distance = duration/58.82;    // Kiszámoljuk a távolságot
  if (distance >= maximumRange || distance < minimumRange) {
    distance = -1.0;
  }
  Serial.print(distance);       // A mért távolság kiíratása
  Serial.println(" cm");
}
```

Trigger impulzus előállítása

$$d = \frac{t * v}{2}$$

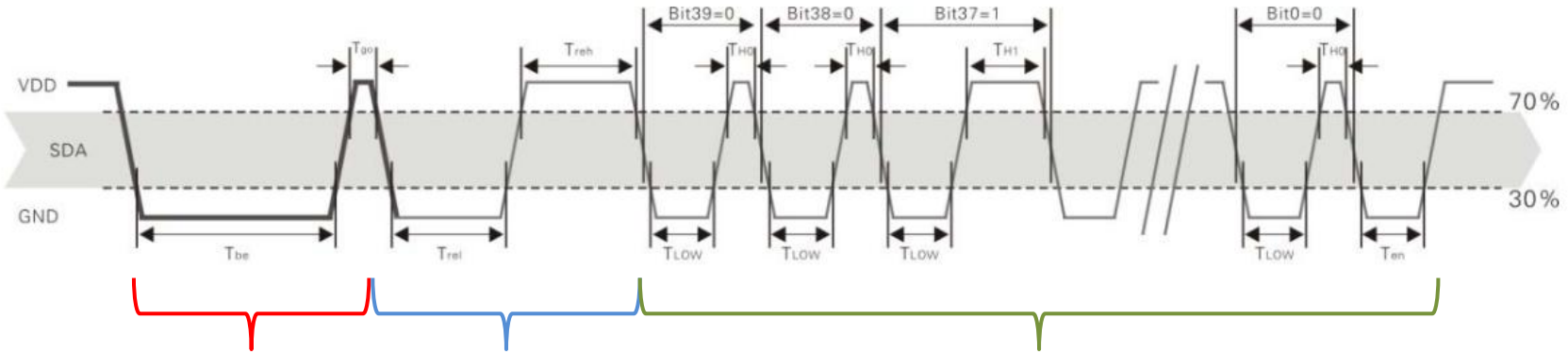
Ahol *d* a távolság, *t* az impulzus hossza, *v* a hang terjedési sebessége (~340 m/s). Mivel *t* értéke µs-ban adott, d-t pedig cm-ben mérjük, így d = t * 34000/2000000, azaz **d = t/58.82**

Hőmérséklet és relatív páratartalom mérése DHT22 szenzorral



AZ AM2302 (DHT22) SZENZOR FŐBB JELLEMZŐI

Felbontás: hőmérséklet 0.1 ^{o}C és rel. páratartalom 0.1 %
Kommunikáció: 1-wire, nem szabványos protokol, 4 bájt adat (nedvesség 2 bájt, hőmérséklet 2 bájt) + 1 bájt ellenőrző összeg, digitálisan szolgáltatja az adatokat.
Mintavételezési gyakoriság: 2 másodpercenként
Tápfeszültség: 3,5 – 5.5 V



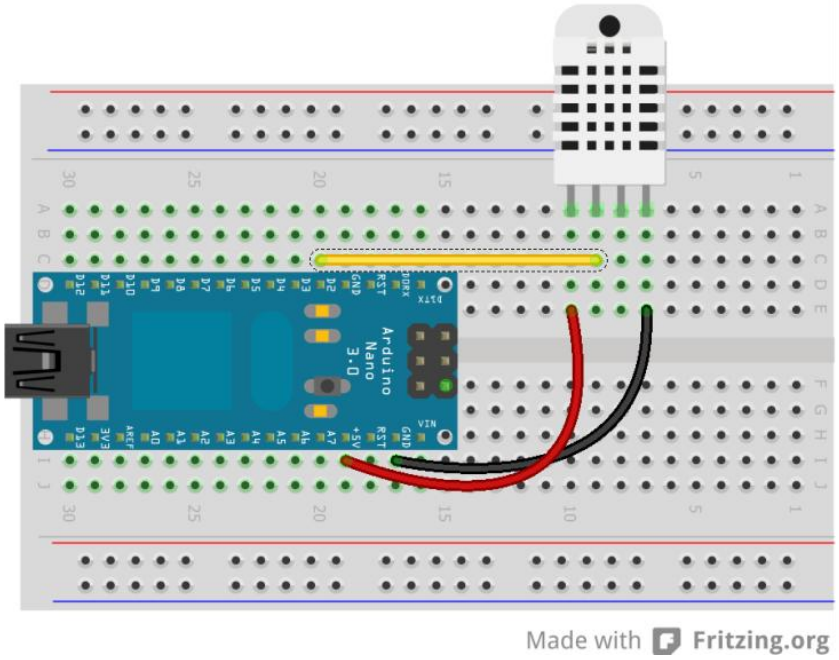
Host indítójel

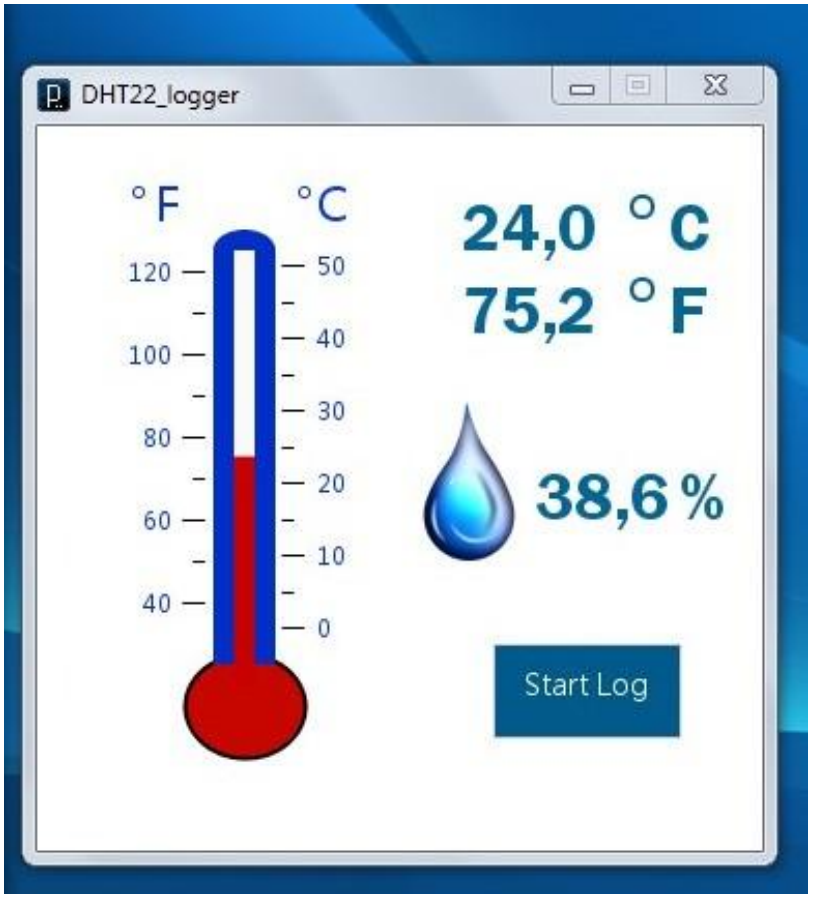
Szenzor nyugtázó jel

40 bitnyi adat
32 bit információ + 8 bit ellenőrző összeg

Összesen tehát 85 időzítést tartalmaz egy-egy tranzakció …

Kapcsolási elrendezés és a futtatási eredmény





DHT programkönyvtár

A szenzor kezelésére külön programkönyvtár áll rendelkezésre, melyet a Vázlatfüzet (Sketchbook) „libraries” mappájába kell bemásolni (DHT.h és DHT.cpp állományok). Az eredetileg az Arduinohoz készült, általam módosított könyvtár az Energiához is használható.

A DHT.h tartalma

```
class DHT {                    //A DHT objektum deklarálása
 private:
  uint8_t data[6];
  uint8_t _pin, _type;
  boolean read(void);
  unsigned long _lastreadtime;
  boolean firstreading;

 public:
  DHT(uint8_t pin, uint8_t type);        //Konstruktor
  void begin(void);                      //Inicializálás
  float readTemperature(bool S=false); //T kiolvasás
  float convertCtoF(float);              //C ➔ F konverzió
  float readHumidity(void);              //RH kiolvasás
};
```

A DHT programkönyvtár használata

Egyszerű példa a DHT objektum metódusainak hívására (feltételeztük, hogy a szenzor DHT22 típusú és az adatvonala az Arduino kártya D2 kivezetésére csatlakozik).

Megjegyzés: Nem teljes program, az értelmes használathoz kiegészítésre szorul.

```
DHT dht(2, DHT22);                          //A DHT osztály példányosítása

void setup() {
  dht.begin();                              //inicializálás
}

void loop() {
  float h = dht.readHumidity();             //Relatív páratartalom kiolvasása
  float t = dht.readTemperature();          //Hőmérséklet kiolvasása
  float f = dht.convertCtoF(t);             //Átszámítás Fahrenheit skálára
  //Kiíratás, kijelzés, stb.
  . . . .
}
```

DHT.cpp (a kritikus részlet)

```
// read in timings
 for ( i=0; i< MAXTIMINGS; i++) {
   counter = 0;
   while (digitalRead(_pin) == laststate) { //Állapotváltozás detektálása
     counter++;
//     delayMicroseconds(2);
     if (counter == 255) {
       break;
     }
   }
   laststate = digitalRead(_pin);
   if (counter == 255) break;
   if ((i >= 4) && (i%2 == 0)) {  // ignore first 3 transitions

// shove each bit into the storage bytes
     data[j/8] <<= 1;
     if (counter > DHTLEVEL)
       data[j/8] |= 1;
     j++;
   }
 }
```

Igazítás az eltérő CPU sebességhez

Kártya	DHTLEVEL
Arduino	6
MSP430	11
Tiva C	64

DHT22_test.ino

```
#include "DHT.h"
#define DHTPIN  2       //MSP430:P2_3
#define DHTTYPE DHT22   // DHT 22
DHT dht(DHTPIN, DHTTYPE);
void setup() {
 Serial.begin(9600);
 dht.begin();
}
void loop() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(t) || isnan(h)) {
    Serial.println("Failed to read from DHT");
  } else {
    Serial.print("Humidity: ");
    Serial.print(h,1);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(t,1);
    Serial.println(" *C");
    delay(2000);
  }
}
```

A mintaprogram célja a DHT22 szenzor működésének ellenőrzése, és az üzembiztos kiolvasáshoz szükséges időzítési paraméter optimalizálása.

Olvasási hiba esetén a visszatérési érték NaN lesz.
NaN = Not a Number (nem szám)

A DHT.h állományban vegyük ki a kommentből a DEBUG makró definiálását!

DHT22_test.ino futtatása

Megjegyzés: Ez a futtatás MSP430 Launchpad kártyán történt! **Arduino** esetén kisebb számokat kapnánk…

Összeolvasás:

A hexadecimálisan kiíratott adatok tizedfokban, illetve tizedszázalékban adják meg a mért értékeket, s a kétbájtos adatoknál a magasabb helyiértékű áll elől.

Például 1, 40, 1, D, 4F jelentése:

Humidity = (1*256 + 64)/10 = 32.0 %

Temp = (1*256 + 13)/10 = 26.9 °C

Kontrolösszeg = 1 + 40 + 1 + D = 4F

(hexadecimális számok!!!)

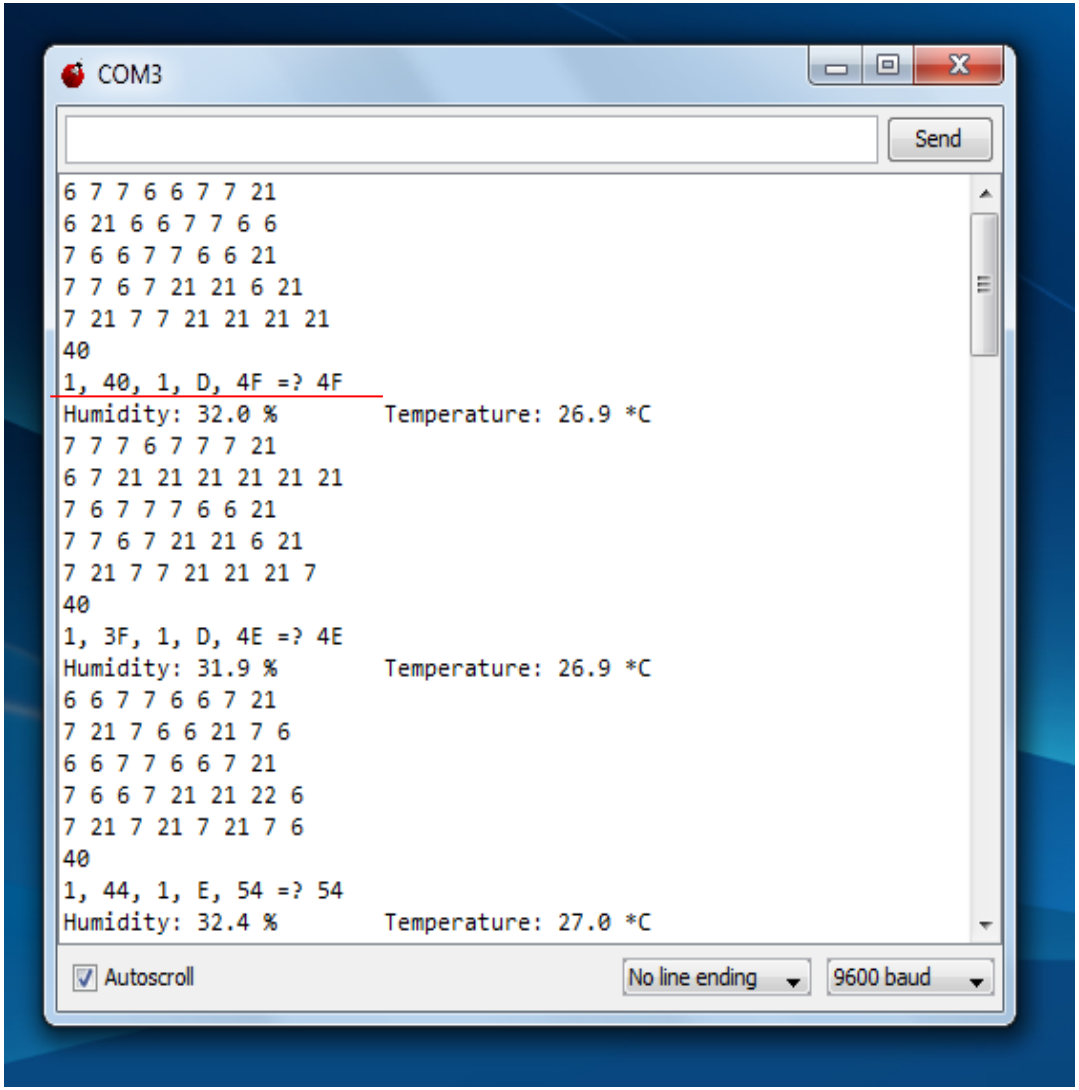
Időzítések:

6 – 7 ciklus ’0’ bit esetén

21 ciklus ’1’ bit esetén

Diszkriminációs szint:

10 – 15 közötti érték az optimális



TRHlogger.ino

A hőmérséklet és páratartalom kijelzése a PC képernyőjén, együttműködve a PC-n futó **dht22** (Processingben írt) alkalmazással.

```
#include "DHT.h"
#define DHTPIN  2         //MSP430 esetén: P2_3
#define DHTTYPE DHT22     // DHT 22 (AM2302)
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  delay(5000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(t) || isnan(h)) {
    //Serial.println("Failed to read from DHT");
  } else {
    Serial.print(h,1);
    Serial.print(" ");
    Serial.println(t,1);
  }
}
```

Fontos figyelmeztetés:
Ne felejtsük el kommentbe tenni a **DEBUG** makró definiálását a **DHT.h** állományban!
Most zavaró lenne a sok kiírás…

Lényegében csak a kiíratás formátumát változtattuk meg.
Például:
42.0 25.6
41.9 25.7
. . .

A dht22 program futtatása a PC-n

- ✓ Bontsuk ki a **dht22.zip** állományt valahová, s indítsuk el a **dht22\application.windows32** mappában található **dht22.exe** programot.
- ✓ A program működéséhez a gépen 32 bites Java futtatói környezet (JRE) szükséges, s az alkalmazás igényli a mellette található állományokat és mappát (**rxtxSerial.dll, waterdrop.jpg, lib** mappa, s kell hozzá az egy szinttel föntebb található **data** mappa is).
- ✓ Mellékeltük a forráskódot is (**dht22.pde**), ami a **processing.org** címről szabadon letölthető **Processing** fejlesztőrendszerrel futtatható, illetve fordítható újra.
- ✓ Ha egynél több soros port található a rendszerben, s nem Az Arduino a legkisebb sorszámú, akkor a **dht.pde** forrásfájlban írjunk a

```
Dev_Board = new Serial(this, Serial.list()[0], 9600);
```

sorban a 0 helyett 1-et!

I2C kommunikáció

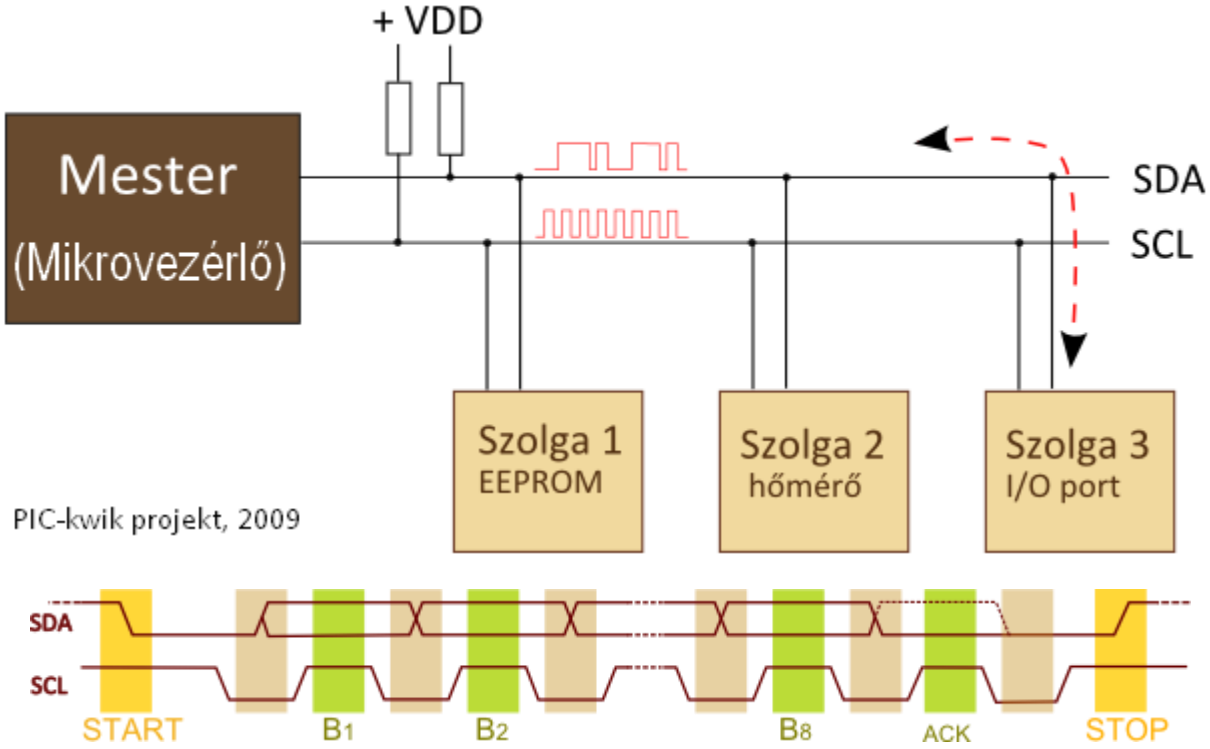
Az **I2C** (Inter-Integrated Circuit = integrált áramkörök közötti) kétvezetékes soros kommunikációs sínt a Philips fejlesztette ki. Az 1990-es évek közepétől számos versenytárs is fejlesztett ki I2C kompatibilis eszközöket, melyeket olcsón és egyszerűen vezérelhetünk a kétvezetékes kommunikációs buszon.

Az I2C busz néhány jellemzője:

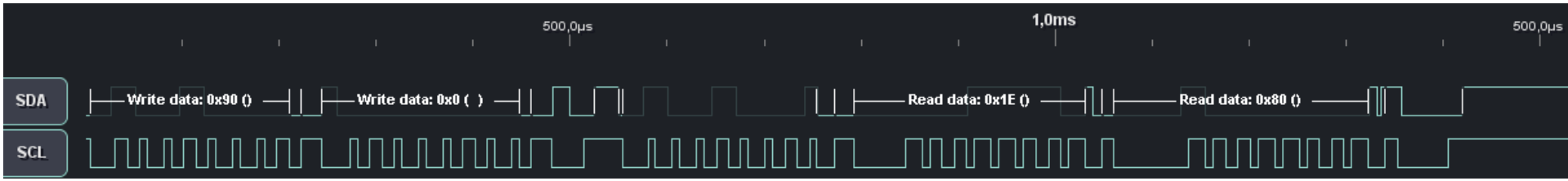
- Két vonalat használ: **SCL** a szinkronjel (órajel), **SDA** pedig kétirányú adat jel.
- A busz akkor szabad, ha egy **STOP** feltételt követően mind az **SDA**, mind az **SCL** vonal magas szinten van (nincs aktív kimenet).
- Soros, 8-bit-es, kétirányú adatforgalom, sebessége tipikusan max 100 kbit/s, vagy 400 kbit/s. Újabb eszközöknél előfordul max. 3,4 mbit/s sebességgel.
- Mindegyik csatlakoztatott eszköz címezhető egy egyedi címmel (7 v. 10 bit).
- Master/slave kapcsolat, melyben a master kezdeményez, vezérel és szolgáltatja az órajelet.
- Bonyolultabb esetekben több master is csatlakozhat a buszra (arbitáció!).

*Leírás: www.muszeroldal.hu/measurenotes/**i2c** hu.pdf*

Az I2C busz

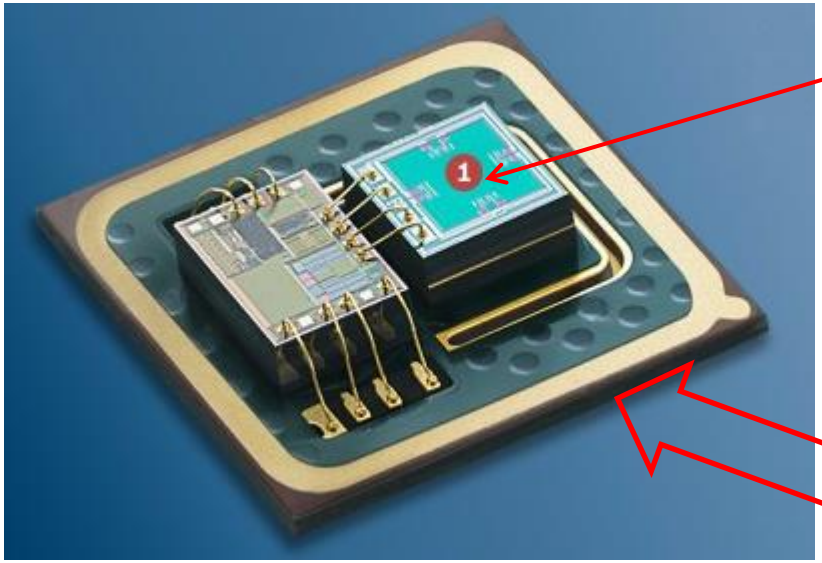


Tipikus események az I2C buszon



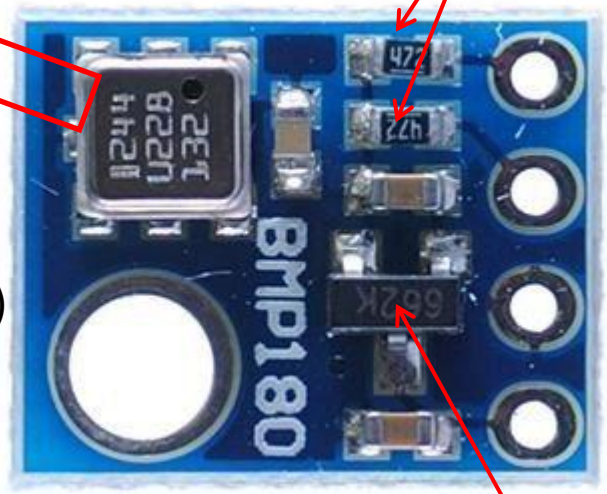
START Slave cím + W regisztercím RESTART Slave cím + R Két adatbájt beolvasása STOP

Légnyomás mérése BMP180 szenzorral



Piezorezisztív nyúlásmérő, amely a nyomás hatására bekövetkező deformációt érzékeli

Felhúzó ellenállások



Feszültségstabilizátor (3,3V)

Bosch SensorTec BMP180

Nyomásmérés: 300 – 1100 hPa (9000 - -300m)
Tápfeszültség: 1,8 – 3,6 V
Áramfelvétel: 5 µA (1 mintavétel/s esetén)
Kis zaj: 0.06hPa (0.5m) kisfogyasztású mód
0.02hPa (0.17m) nagyfelbontású mód
Jellemzők: Hőmérő, I2C felület, gyárilag kalibrált

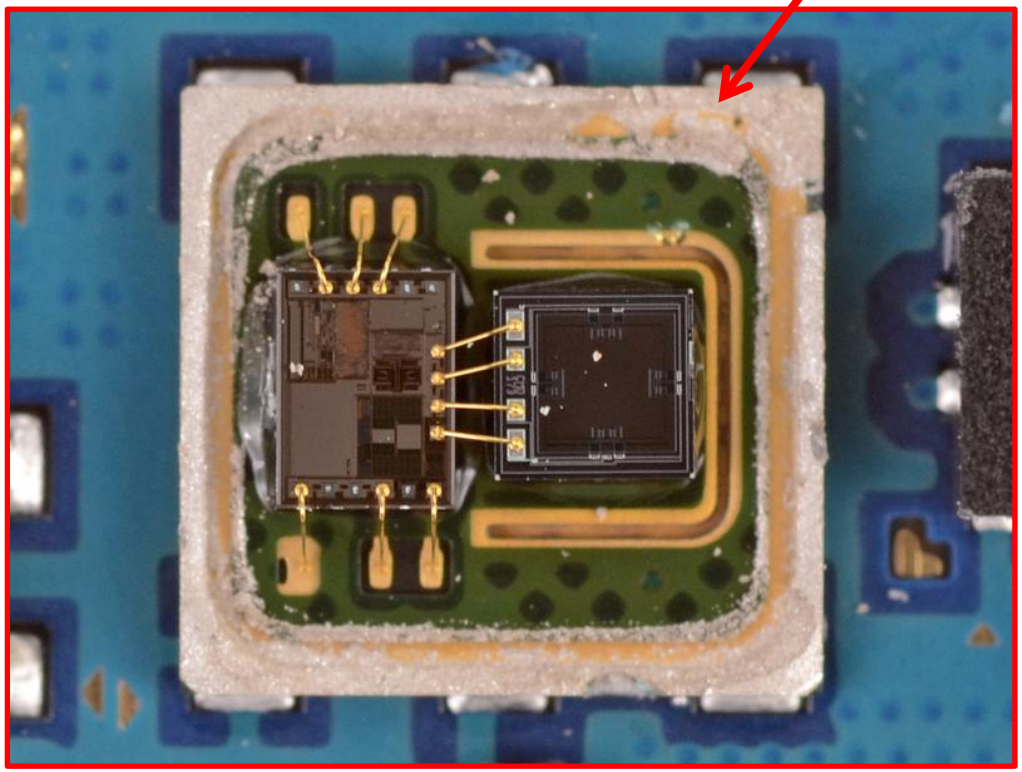
Alkalmazási területek

- GPS navigáció kiegészítése (magasságmérés)
- Kültéri és beltéri navigáció
- Sport és szabadidős tevékenység (pl. magasságmérés túrázás közben)
- Időjárás előrejelzés (a légnyomás helyi süllyedése vihar közeledtét jelzi)
- Függőleges sebesség ellenőrzése (emelkedési/süllyedési sebesség)

Példa: http://www.ifixit.com/Teardown/Samsung+Galaxy+Nexus+Teardown/7182

A szétszedett Samsung Galaxy Nexus belsejében is található egy Bosch BMP180 légnyomásmérő!

BMP180 szenzor

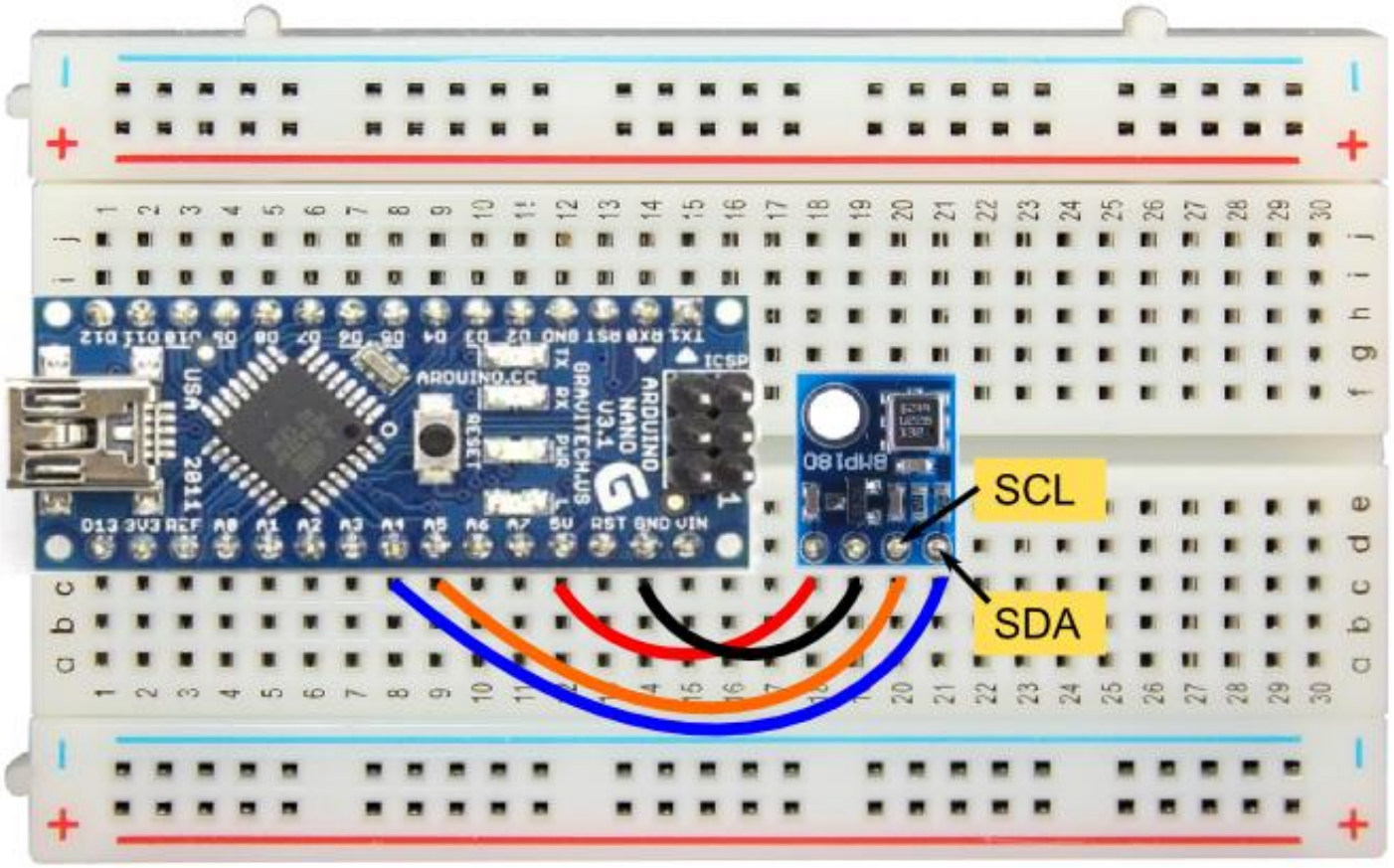


Kapcsolás, szerelés

Hardveres I2C támogatással

Arduino	BMP180
+5V	VIN
GND	GND
A5	SCL
A4	SDA

MSP Launchpad használatához a **2013/14 évad** anyagából a *talk10.pdf* és a *Lab10.zip* állományokat tanulmányozza!



PressureSensor_hw.ino

Felhasználjuk Adrian Struder BMP085 templát könyvtárát is (lásd: ../Arduino/libraries mappa)
https://github.com/astuder/BMP085-template-library-Energia

```
#include "Wire.h"                       // HW supported I2C library
#include "BMP085_t.h"                   // BMP085/BMP180 template library
BMP085<3> PSensor;                      // instantiate and define precision (0..3)

void setup() {
  Serial.begin(9600);                   //Soros kapcsolat 9600 bit/s
  Wire.begin();                         //I2C kapcsolat inicializálása
  PSensor.begin();                      //initalize pressure sensor
}

void loop() {
  PSensor.refresh();                    // read current sensor data
  PSensor.calculate();                  // calculate temperature and pressure
  float t = PSensor.temperature;        // temperature in 0.1 Celsius units
  float p = PSensor.pressure;           // pressure in Pascal units
  Serial.print("Temperature: ");
  Serial.print(t/10,1);                 // Print result with 1 decimal
  Serial.println(" C");
  Serial.print("Pressure: ");
  Serial.print((p+1440)/100,1);         // pressure in hPa with altitude correction
  Serial.println(" hPa");
  delay(5000);
}
```

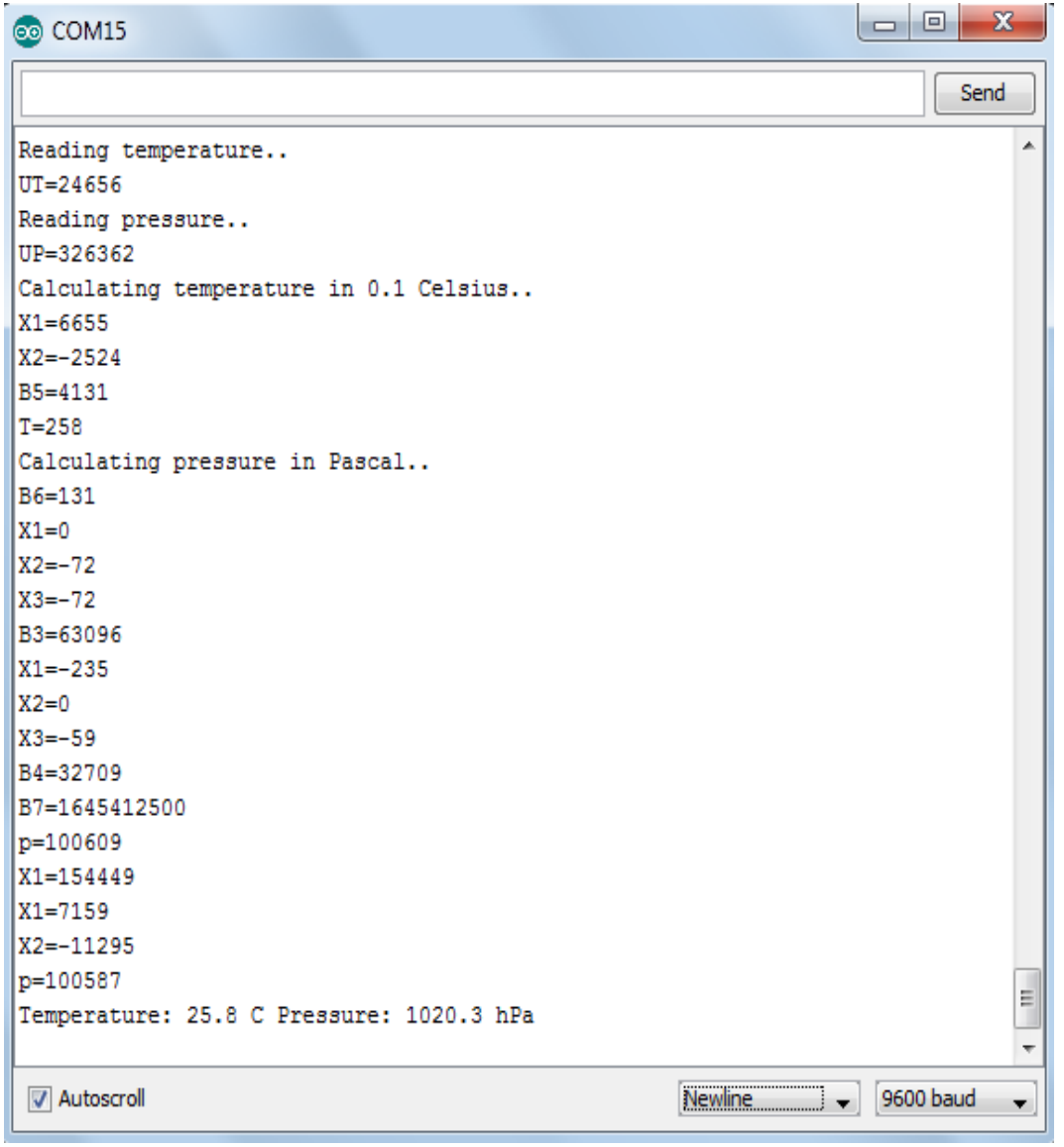
Ellenőrzés, nyomkövetés

A **BMP085_t.h** állományban definiáljuk a DEBUG_BMP085 makrót, ekkor aktiválódnak a nyomkövető kiíratások, amelyekkel ellenőrizhetjük a szenzorból kiolvasott nyers adatokat, valamint a hőmérséklet és a nyomás kiszámítását.

Tipp: Töröljük ki a komment jelet!

```
// #define DEBUG_BMP085
```

A kiszámítás képleteit a BMP180 szenzor adatlapja tartalmazza.
http://ae-bst.resource.bosch.com/media/products/dokumente/bmp180/BST-BMP180-DS000-09.pdf



A kiszámítás menete

Table 5: Calibration coefficients

	BMP180 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

UT és **UP** a nyers hőmérséklet és nyomás adat
oss – oversampling (0 – 3)

calculate true temperature

$X1 = (UT - AC6) * AC5 / 2^{15}$
$X2 = MC * 2^{11} / (X1 + MD)$
$B5 = X1 + X2$
$T = (B5 + 8) / 2^4$ [0,1 °C]

calculate true pressure

$B6 = B5 - 4000$
$X1 = (B2 * (B6 * B6 / 2^{12})) / 2^{11}$
$X2 = AC2 * B6 / 2^{11}$
$X3 = X1 + X2$
$B3 = ((AC1*4+X3) << oss + 2) / 4$
$X1 = AC3 * B6 / 2^{13}$
$X2 = (B1 * (B6 * B6 / 2^{12})) / 2^{16}$
$X3 = ((X1 + X2) + 2) / 2^2$
$B4 = AC4 * (unsigend\ long)(X3 + 32768) / 2^{15}$
$B7 = ((unsigned\ long)UP - B3) * (50000 >> oss)$
if (B7 < 0x80000000) { p = (B7 * 2) / B4 }
else { p = (B7 / B4) * 2 }
$X1 = (p / 2^8) * (p / 2^8)$
$X1 = (X1 * 3038) / 2^{16}$
$X2 = (-7357 * p) / 2^{16}$
$p = p + (X1 + X2 + 3791) / 2^4$ [Pa]

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 ($\mathbf{p_0}$) a következő képlettel számítható ki:

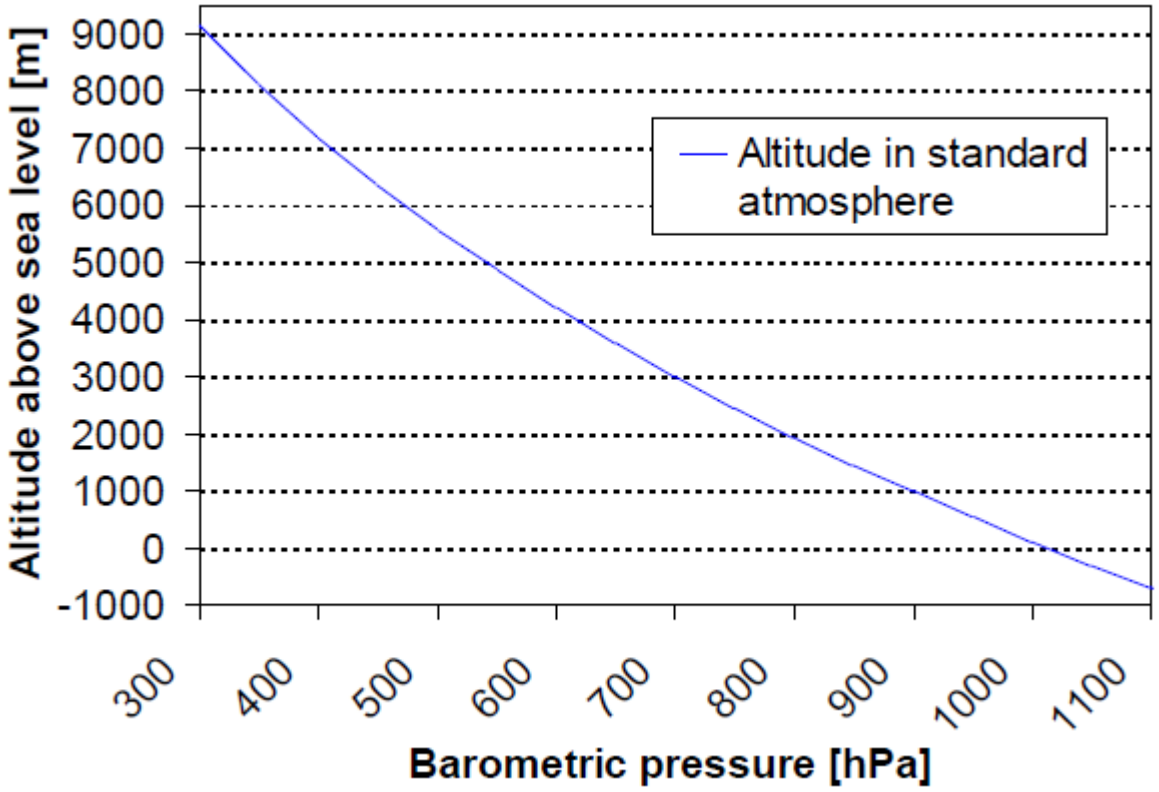
$$p_0 = \frac{p}{\left(1 - \frac{altitude}{44330}\right)^{5.255}}$$

Egyszerű(bb) közelítés:
Debrecenben (kb. 120 m) 1440 Pa-t hozzáadunk a mért légnyomáshoz

Magasság meghatározása

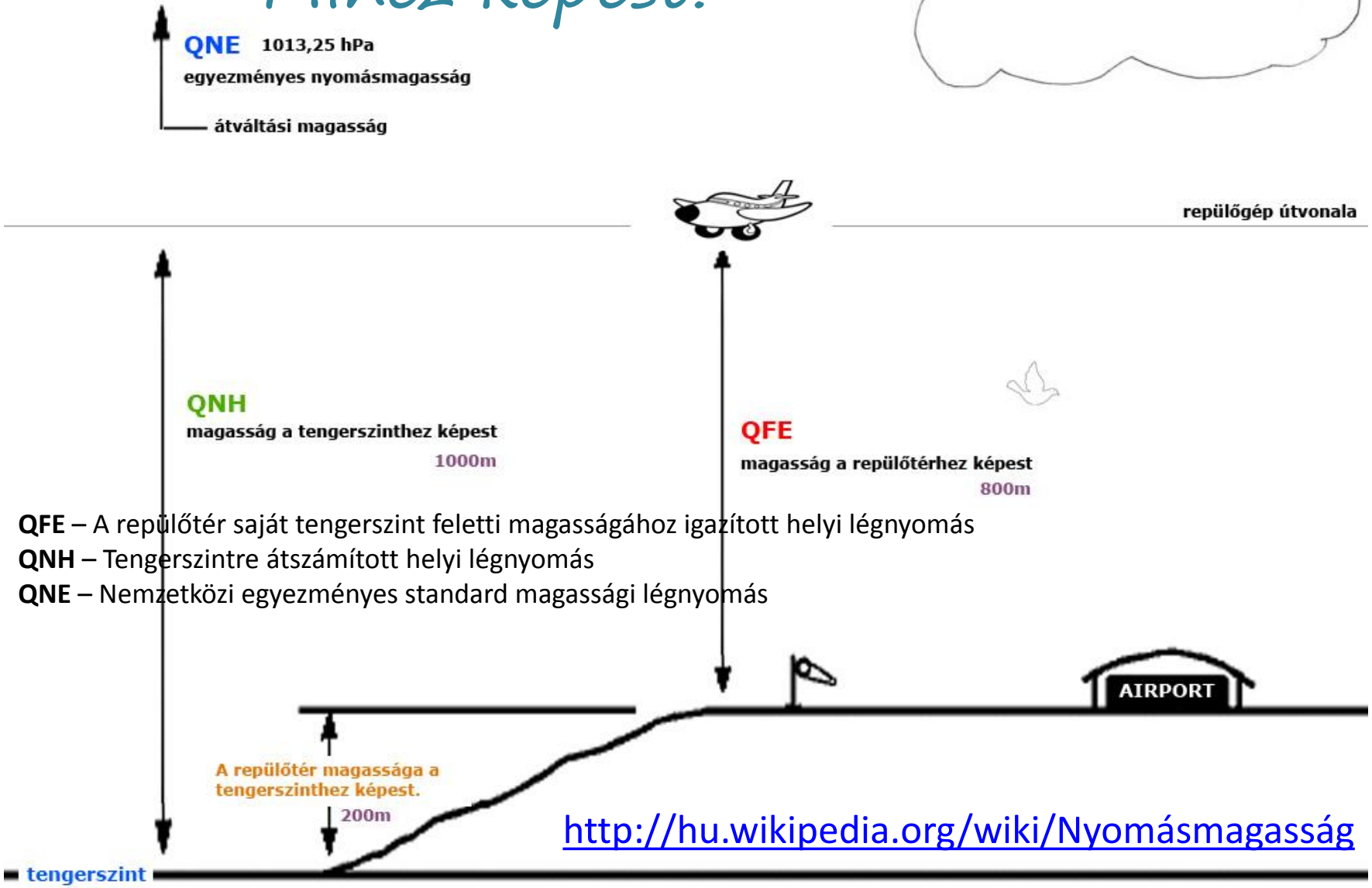
$$\text{magasság} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Ahol **p** az általunk mért nyomás, $\mathbf{p_0}$ pedig a tengerszinti nyomás (pl. 1013.25 hPa)



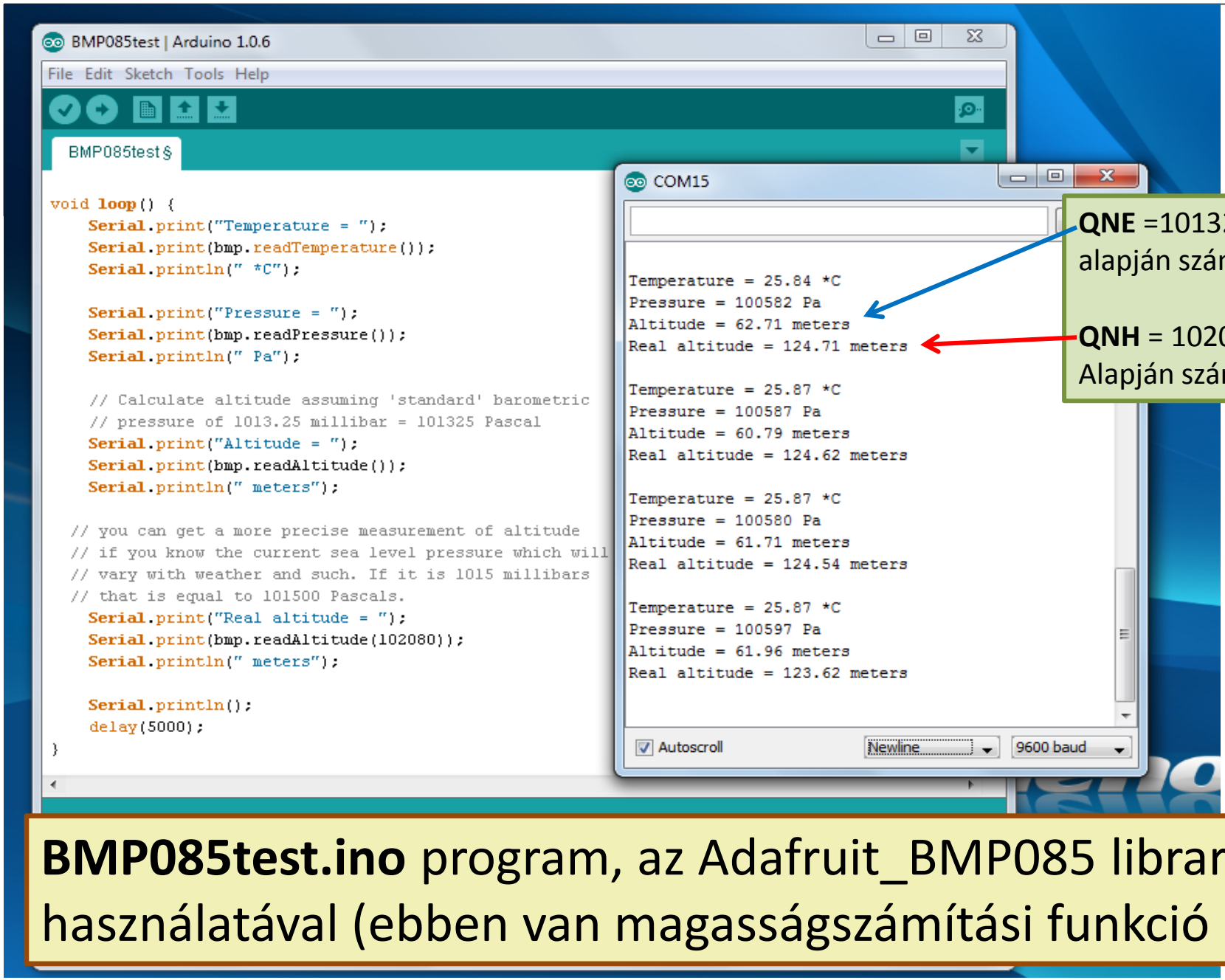
A gyakorlatban a nyomás nemcsak a magasságtól, hanem a meteorológiai viszonyoktól is függ (hőmérséklet, páratartalom, stb.)

– Milyen magasan repül a repülő?
– Mihez képest?



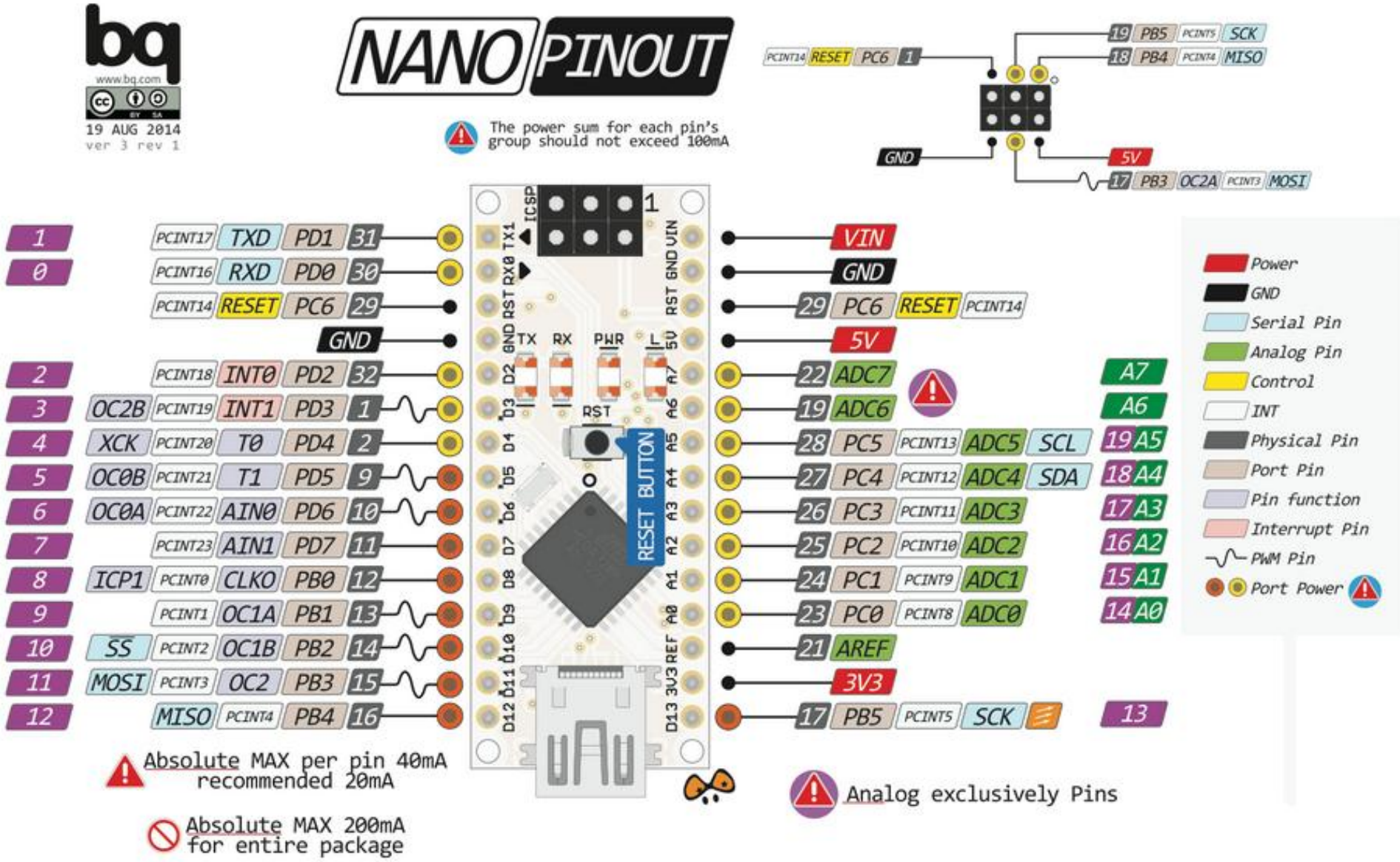
QFE – A repülőtér saját tengerszint feletti magasságához igazított helyi légnyomás
QNH – Tengerszintre átszámított helyi légnyomás
QNE – Nemzetközi egyezményes standard magassági légnyomás

http://hu.wikipedia.org/wiki/Nyomásmagasság



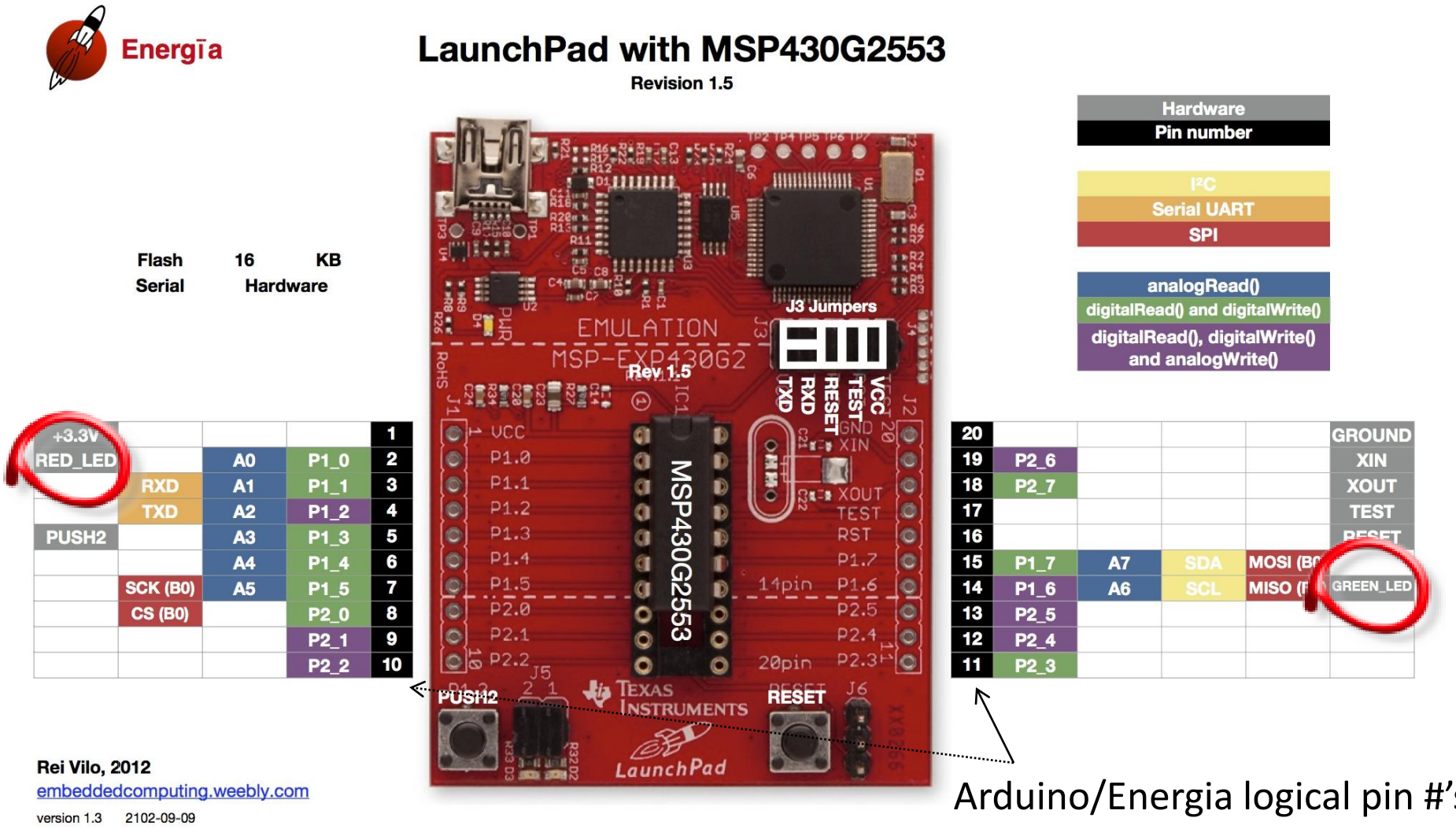
BMP085test.ino program, az Adafruit_BMP085 library használatával (ebben van magasságszámítási funkció is)

Emlékeztető: Arduino nano v3.0



MSP430 Launchpad : Energia Pinout

http://github.com/energia/Energia/wiki/Hardware



Rei Vilo, 2012
embeddedcomputing.weebly.com
version 1.3 2102-09-09

Arduino/Energia logical pin #'s