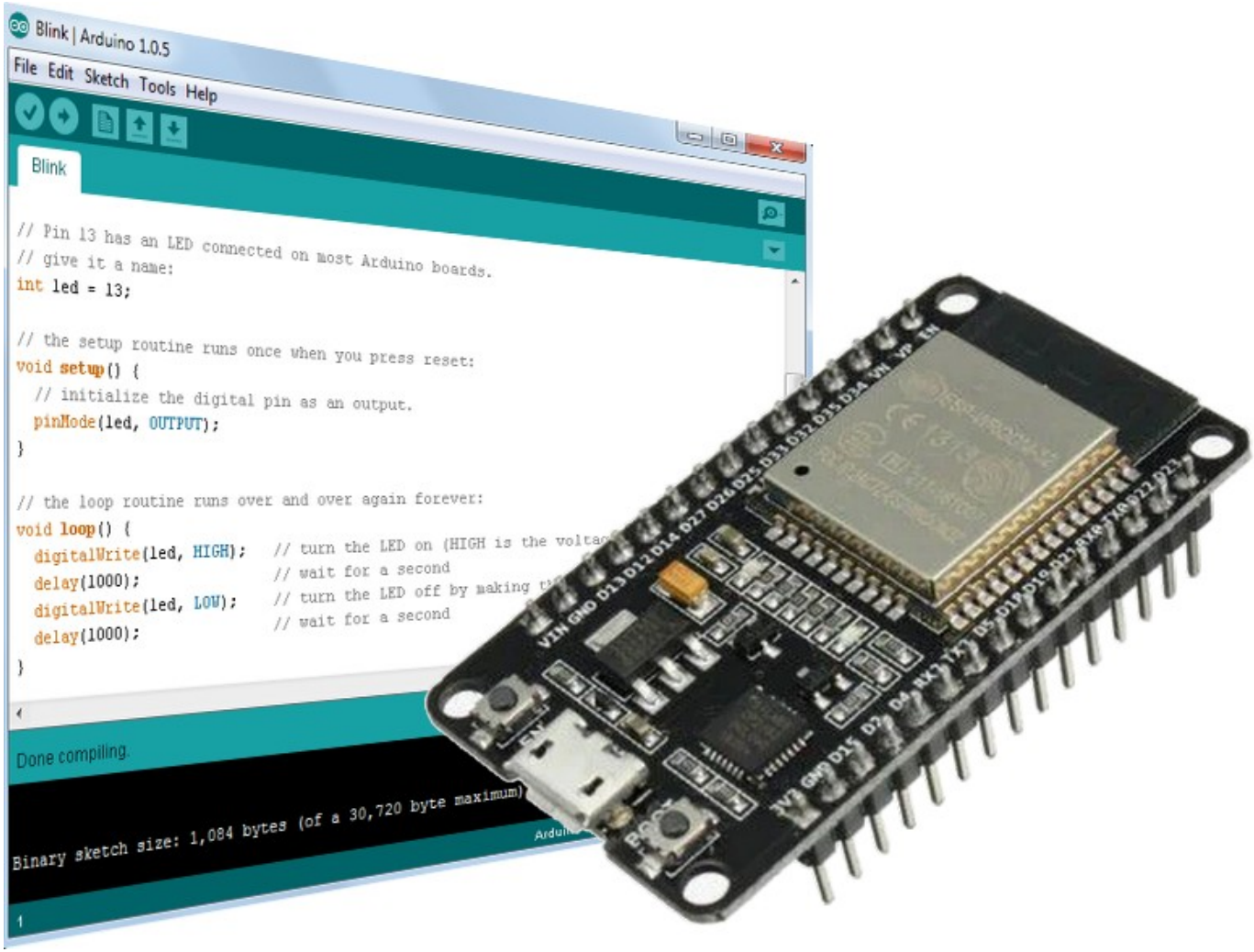
ESP32 mikrovezérlők programozása Arduino környezetben

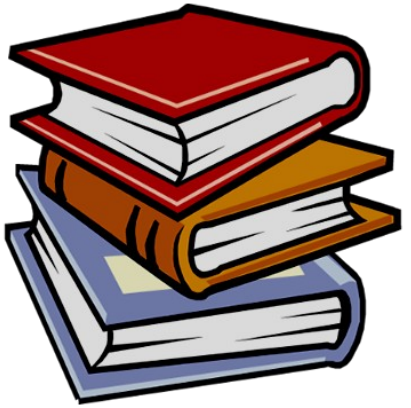


ARDUINO™

4. Analóg I/O, analóg szenzorok

Felhasznált és ajánlott irodalom

- Harsányi Réka – Juhász Márton András:
  Fizikai számítástechnika – elektronikai alapok és Arduino programozás
- Henry Cheung: github.com/e-tinkers/esp32-adc-calibrate
- Khaled Magdy: ESP32 Temperature Sensor LM35 Interfacing
- **Processing IDE:** processing.org
- Bill Kujawa: Meter (Processing library)
- Espressif: ESP32 Technical Reference Manual



Példaprogramok

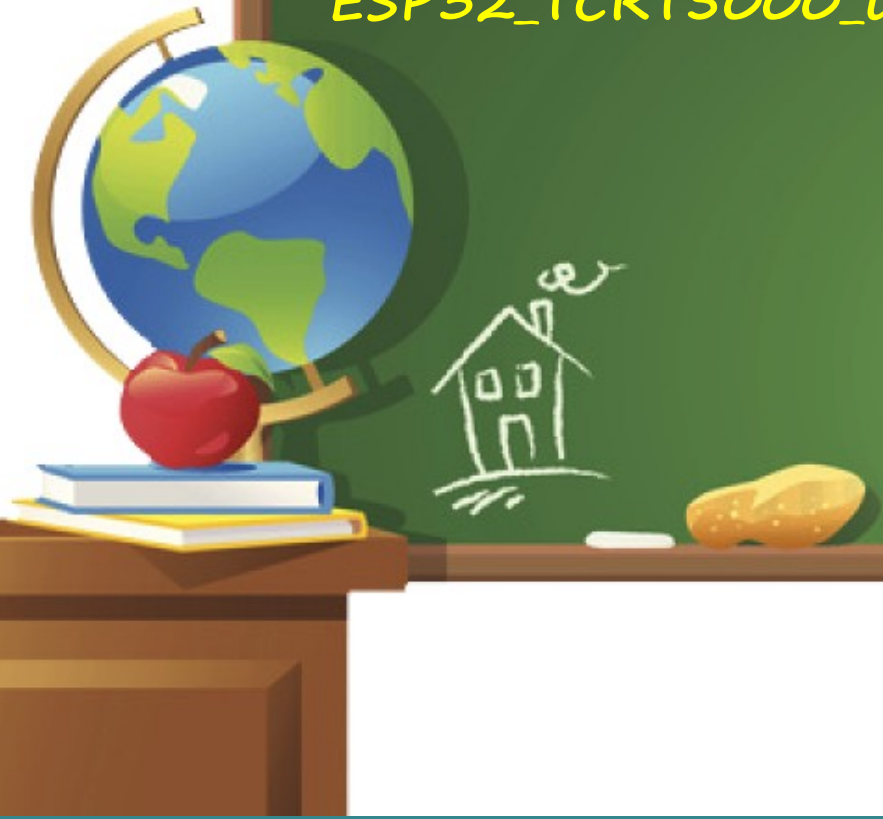
ESP32_ADC_calibrate – ADC kalibrálás

ESP32_analog_thermometer – hőmérő szenzor

ESP32_LDR_test – egyszerű fényérzékelés

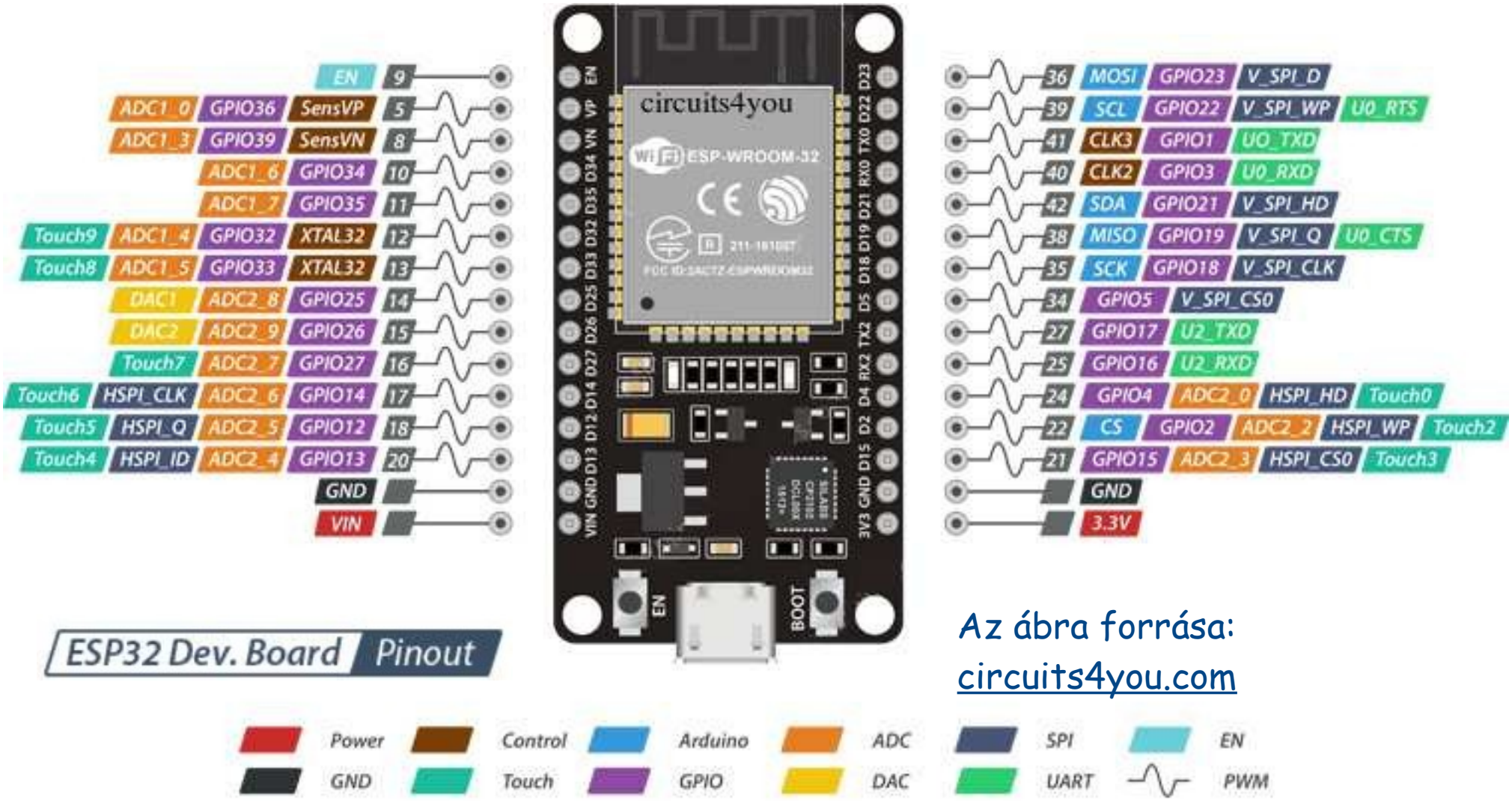
ESP32_twighlight_switch – szürkületkapcsoló

ESP32_TCRT5000_test – optikai közelségérzékelő



Emlékeztető: Analóg be- és kimenetek

- Két ADC van, de nem mindegyik bemenet érhető el, s **ADC2** csak a WiFi letiltott állapotában használható



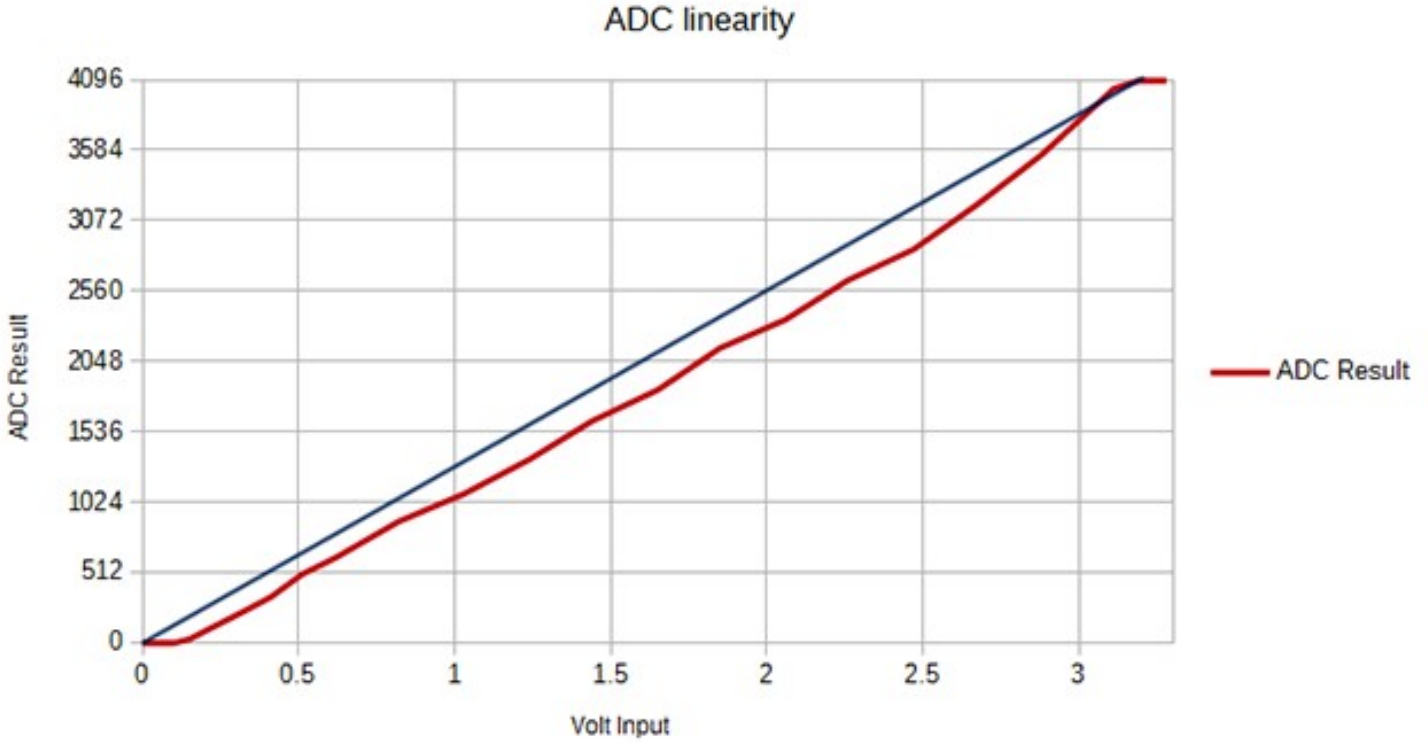
Az ábra forrása:
circuits4you.com

Emlékeztető: Analóg I/O függvények

- **analogRead(*pin*)** – megméri a megnevezett bemeneten a feszültséget és visszaad egy számot (alapértelmezetten 12 bit a felbontás és kb. 3,3 V a méréshatár)
- **analogReadResolution(*resolution*)** – beállítja a felbontást (9 –12 bit, alapértelmezetten: 12 bit)
- **analogSetAttenuation(*attenuation*)** – méréshatár beállítása az összes bemenetre vonatkozóan (**ADC_0db:** 1 V, **ADC_2_5db:** 1,5 V, **ADC_6db:** 2 V, **ADC_11db:** 3,3 V)
- **analogSetPinAttenuation(*pin, attenuation*)** – a méréshatár beállítása egy adott lábra
- **dacWrite(*pin, value*)** – beállítja a megadott **DAC** kimenet feszültségét, ahol
  pin – a DAC kimenethez tartozó kivezetés (**DAC1:** 25, **DAC2:** 26)
  value – 0 – 255 közötti szám

Az ADC kalibrálása

- Ahogy az első előadásban láttuk, az analóg bemeneti csatornák linearitása komoly problémákat vet fel, amire Henry Cheung (github.com/e-tinkers/esp32-adc-calibrate) olyan megoldást mutat be, amelyben a DAC segítségével ellenőrizzük a linearitást
- A kalibráláshoz egy keresőtáblát állítunk össze (ehhez az eltérő felbontás miatt interpolálni kell)



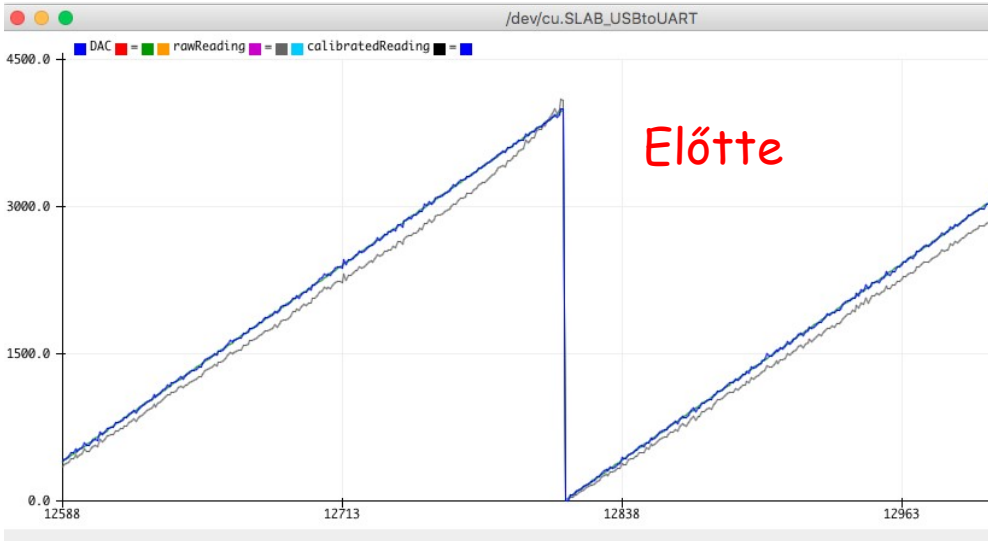
ESP32_ADC_calibrate.ino (részlet)

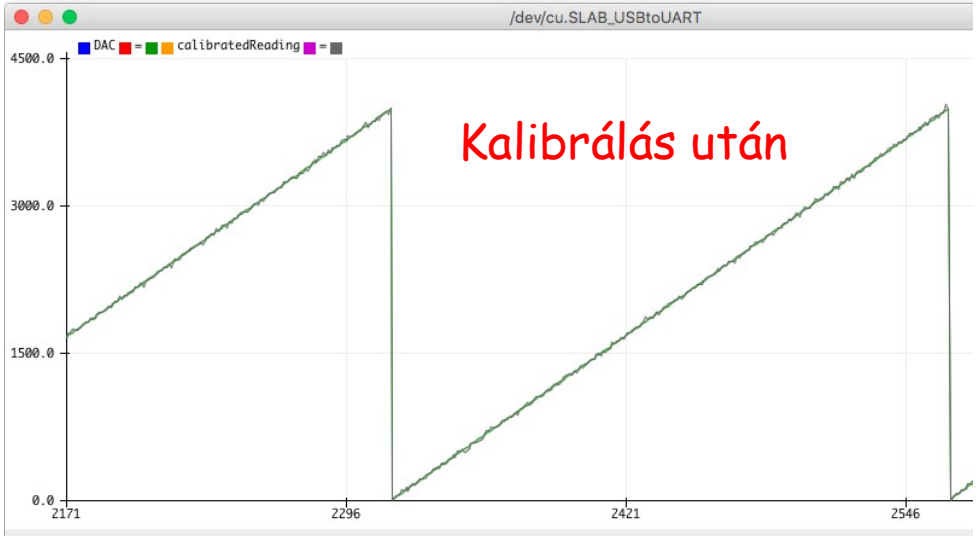
- A kalibrálást és a táblázat (LUT) összeállítását végző szoftvernek csak a konfigurálandó részét mutatjuk be: kérhetünk *float* vagy *int* táblázatot, s kérhetünk megjelenítést a **Serial Plotter** ablakban

```
#include <Arduino.h>
#include <driver/dac.h>

//#define GRAPH       // uncomment this for print on Serial Plotter
//#define FLOAT_LUT   // uncomment this if you need float LUT
#define ADC_PIN 35    // GPIO 35 = A7, uses any valid Ax pin as you wish

float Results[4097];
float Res2[4096*5];

void setup() {
    dac_output_enable(DAC_CHANNEL_1);    // pin 25
    dac_output_voltage(DAC_CHANNEL_1, 0);
    analogReadResolution(12);
    Serial.begin(115200);
    delay(1000);
}
. . .
```

Az ESP32_ADC_calibrate program használata

- Kössük össze a **GPIO25** (**DAC1**) és a **GPIO35** (**A7**) kivezetéseket!
- Döntsük el, hogy *float* vagy *int* táblázatot kérünk (utóbbi esetben maradjon megjegyzésben a `#define FLOAT_LUT` sor!
- Fordítsuk le és futtassuk a programot!
- Amikor a program leállt, másoljuk ki és mentsük el a kiírt táblázatot (pl. egy LUT.h nevű állományba) a Soros Terminál ablakból és ezt csatoljuk be majd a programjainkba
- A LUT.h korrekciós tábla használatát a következő példaprogramban mutatjuk meg





Mit kezdjünk a LUT.h állománnyal?

- Mivel több programban is használni fogjuk, célszerű a vázlatfüzet (Sketchbook) mappán belül a **libraries** almappában létrehozni egy **LUT** nevű könyvtárat, ebbe bemásolni a **LUT.h** állományt
- A használatához a programba be kell csatolni:

```
#include "LUT.h"
```

- A nyers adatot indexként használva kapjuk a korrigált értéket:

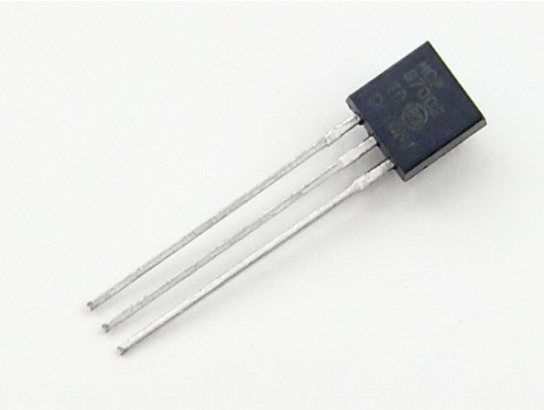
```
int rawData = analogRead(ADC_PIN);
int calibratedData = ADC_LUT[rawData];
```

- A **LUT.h** állomány lényegében egy 4096 elemű tömböt definiál:

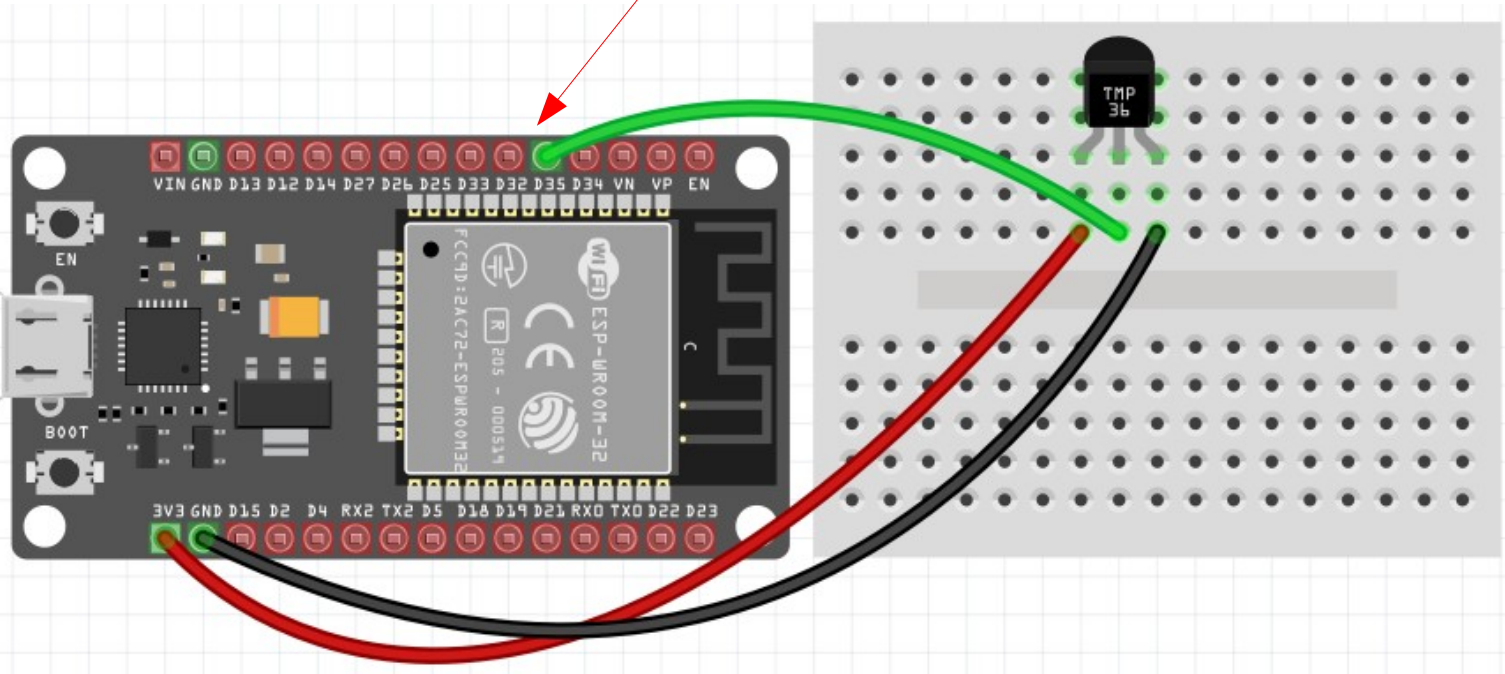
```
const int ADC_LUT[4096] = { 0,
64,66,68,69,71,73,75,77,79,80,82,83,84,85,87,
88,89,90,92,93,94,95,96,97,98,99,100,101,102,103,
103,104,105,106,107,108,109,110,111,111,113,114,116,117,119,
 . . .
4072,4073,4073,4074,4074,4075,4075,4076,4076,4077,4077,4078,4078,4079,4080,
4080,4081,4082,4084,4085,4086,4087,4088,4089,4090,4091,4092,4093,4094,4095
};
```

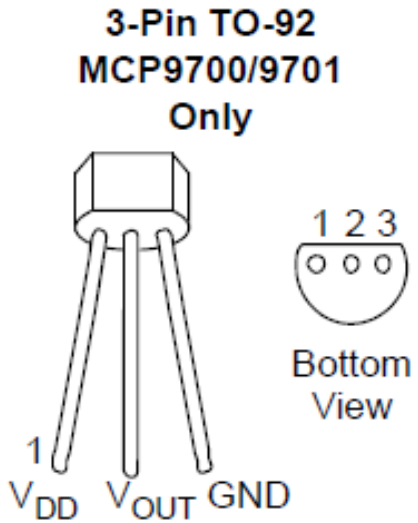
Hőmérséklet mérése analóg hőmérővel

- Microchip **MCP9700**
  - VDD = 2,5 – 5,5 V
  - Mérési tart.: –40 – 150 °C
  - Érzékenység: 10 mV / °C
  - Nullapont: 500 mV @ 0 °C



Most az **A7 (GPIO35)** bemenetre kötöttük
a hőmérőt, de más analóg bemenetet is használhatunk!



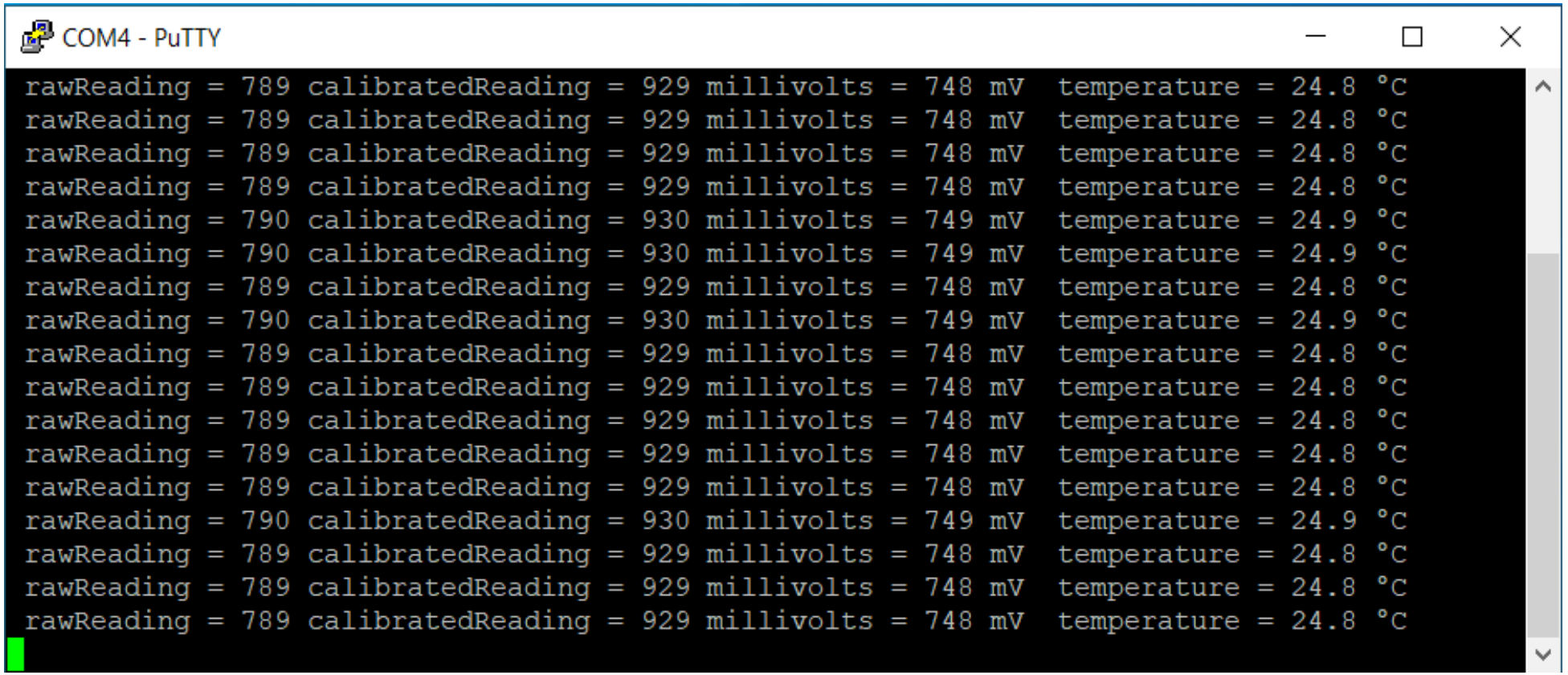


```ino
#include <Arduino.h>
#include "LUT.h"                                  // Kalibrációs táblázat
#define ADC_PIN 35                                // pin 35 lesz a bemenet

void setup() {
  analogReadResolution(12);                       // 12 bites felbontást
  analogSetAttenuation(ADC_11db);                 // Méréshatár 0-3,3 V
  Serial.begin(115200);                           // baudrate = 115200 bit/s
  while (!Serial) {}                              // Várunk a soros portra
}

void loop() {
  uint32_t rawReading = 0 ;
  for (int i = 0; i < 1024; i++) {                // 1024 mérést átlagolunk
    rawReading += analogRead(ADC_PIN);            // az ADC kiolvasása
  }
  rawReading = rawReading >> 10;                  // osztás 1024-gyel
  int calibratedReading = ADC_LUT[rawReading];    // a kalibrált érték előkeresése
  Serial.print(F(" rawReading = "));              // Nyers adat kiírása
  Serial.print(rawReading);
  Serial.print(F(" calibratedReading = "));       // Kalibrált adat kiírása
  Serial.print(calibratedReading);
  int milliVolts=calibratedReading*3300/4096;     // Átszámítás feszültségre (mV)
  float tempC = (milliVolts - 500) / 10.0;        // Átszámítás Celsius fokokra
  Serial.print(F(" millivolts = "));              // Feszültség kiírása
  Serial.print(milliVolts);
  Serial.print(F(" mV  temperature = "));         // Hőmérséklet kiírása
  Serial.print(tempC, 1);
  Serial.println(F(" °C"));
  delay(2000);
}
```

ESP32_analog_thermometer.ino programlista

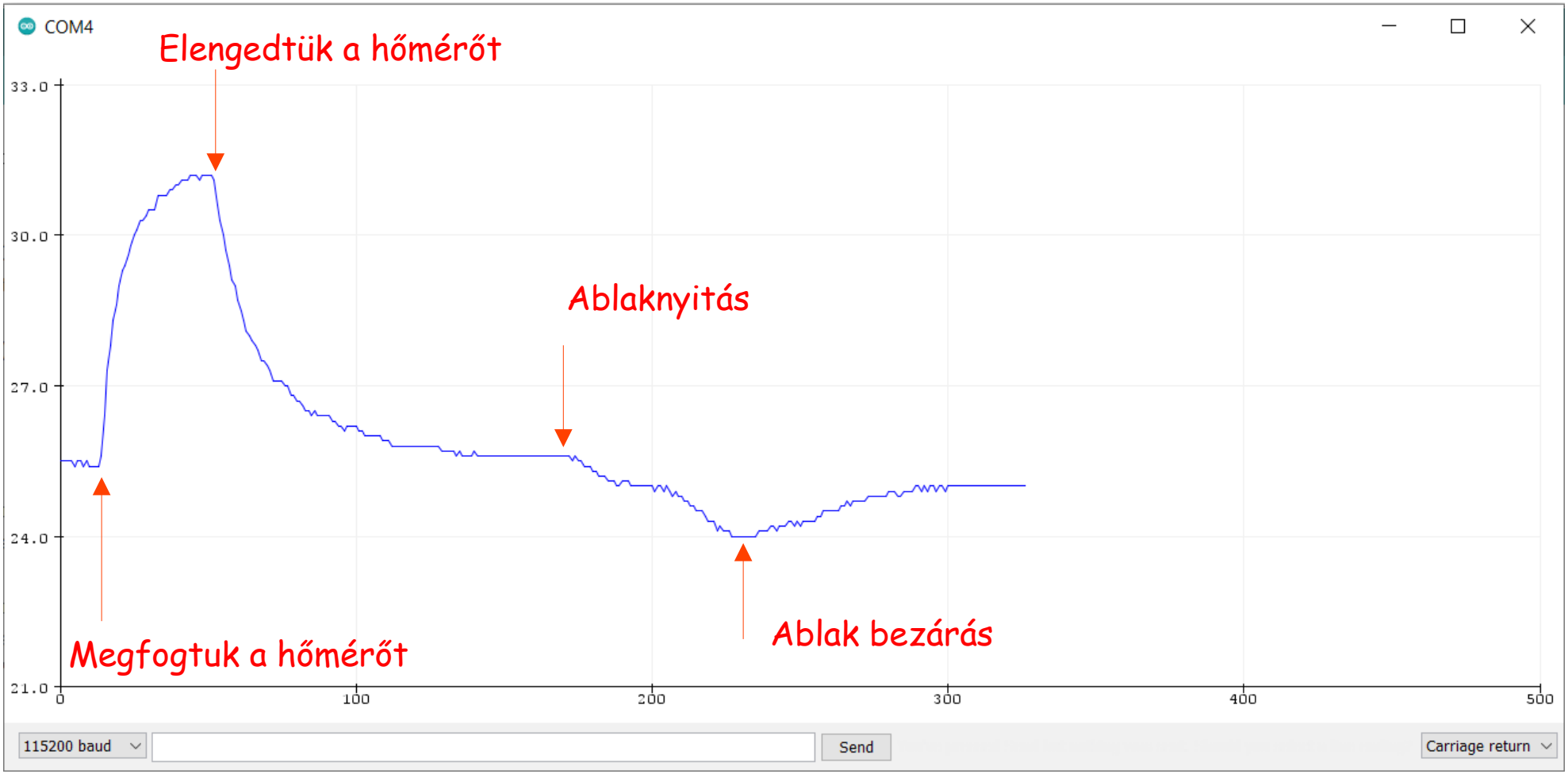
ESP32_analog_thermometer futási eredmény

- Az alábbi ábrán egy futási eredmény látható
- Ebben a tartományban jelentős a korrekció mértéke
- Szobahőmérővel összehasonlítva ez az eredmény reálisabb, mint a korábbi, lineáris korrekcióval kapott (lásd az első előadásban)



Fokozzuk az élményt!

- Hagyjuk el a fölösleges kiírásokat, csak a hőmérséklet számszerű értékét írjuk ki! (lásd: **ESP32_analog_thermometer2.ino**)
- A *Serial Plotter* ablakban kövessük a hőmérséklet időbeli változását!



Hőmérő grafikus kijelzéssel

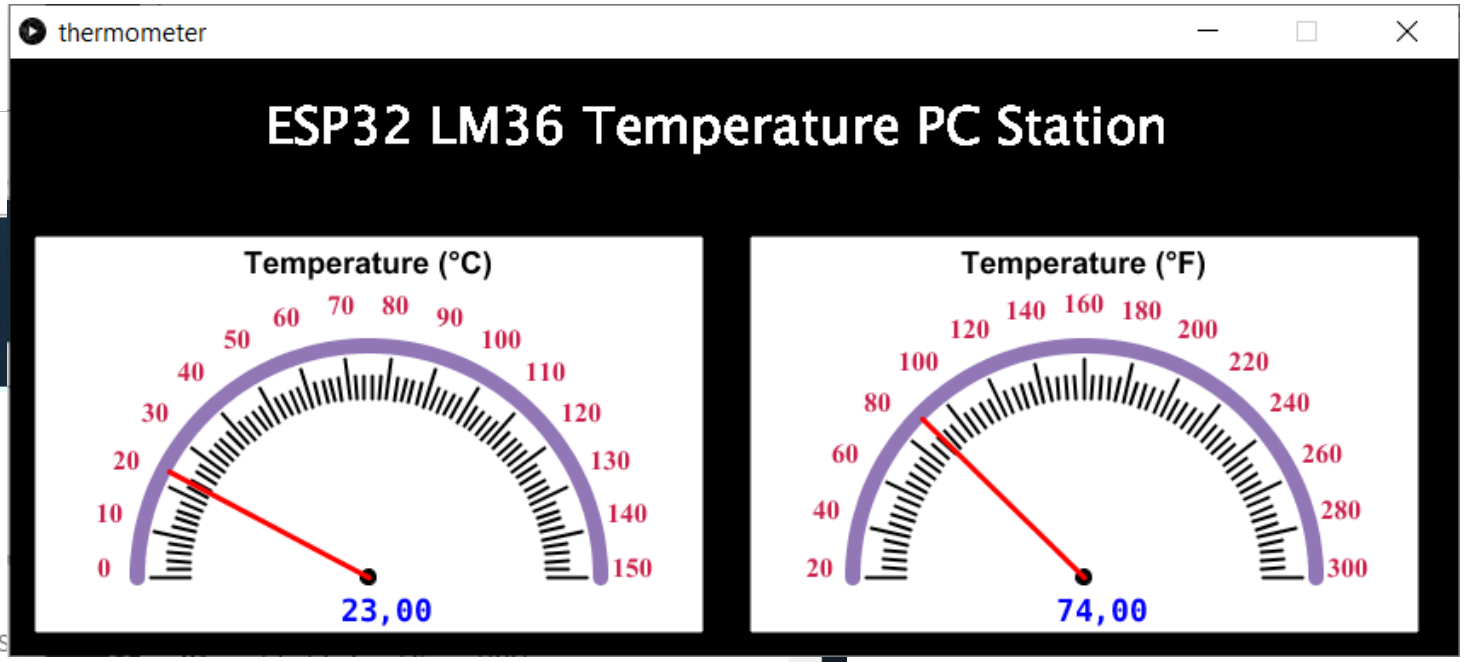
- Khaled Magdy: ESP32 Temperature Sensor LM35 Interfacing c. projektjében egy **Processing** alkalmazást is ismertet, melynek felhasználásával az **ESP32_analog_thermometer** program eredményét grafikusan jeleníthetjük meg a PC képernyőjén
- Ehhez telepíteni kell a **Processing IDE**-t (3.5.4 verzió)
- Nyissuk meg az ezen előadás mintaprogramjai között található **thermometer.pde** Processing vázlatot!
- A **Processing IDE** ***Sketch*** menüjében válasszuk ki az ***Import library/Add library*** menüpontot, majd keressük meg és importáljuk a **Meter** nevű kiegészítő könyvtárat (a letöltéshez Internet kapcsolat szükséges)
- Ha egynél több (virtuális) soros portunk van, és nem az **ESP32** kártya a legkisebb sorszámú, akkor módosítsuk az alábbi sort:

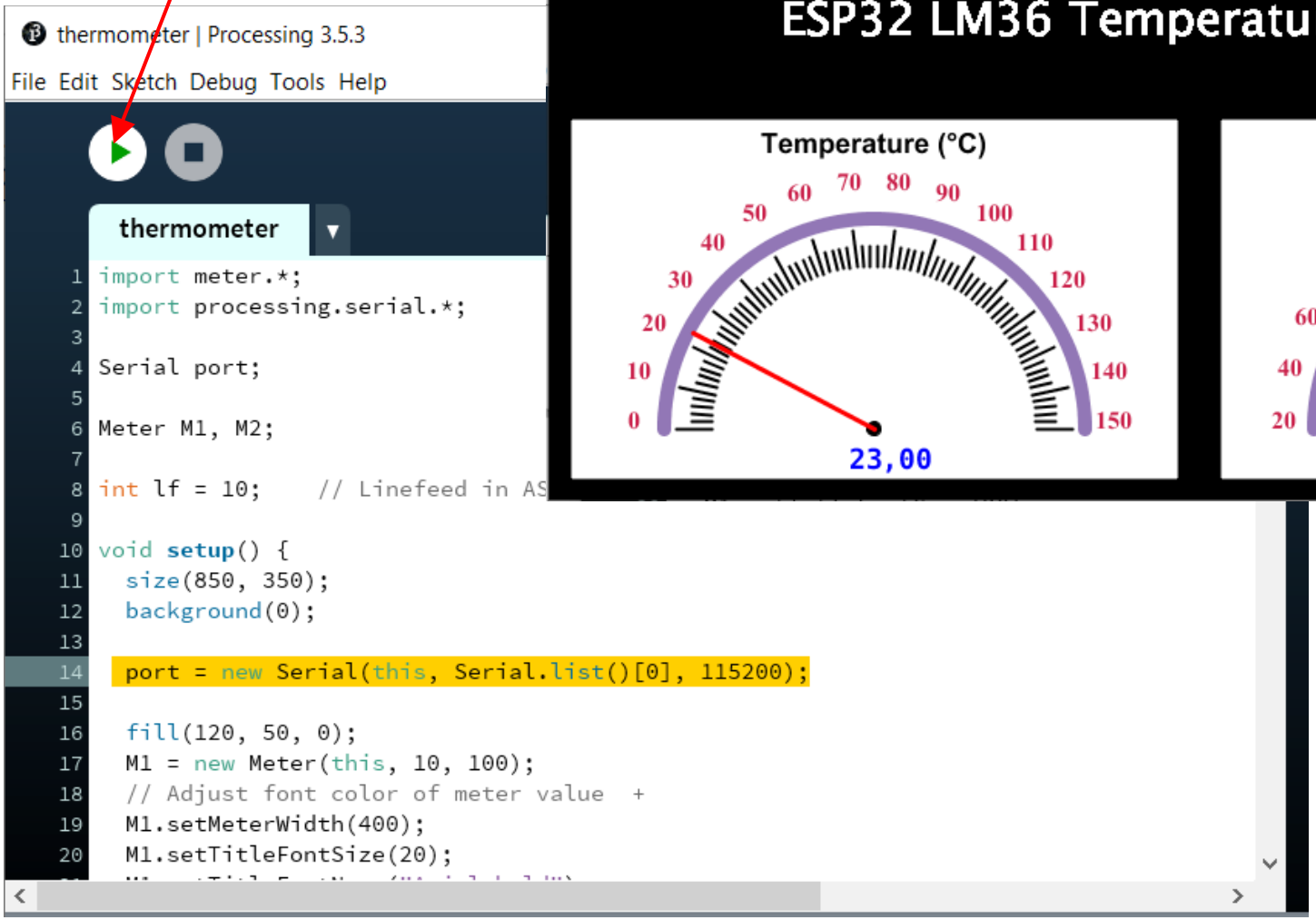
```
port = new Serial(this, Serial.list()[0], 115200);
```

(Itt `Serial.list()[0]` helyett írható `Serial.list()[1]`, vagy pl. `"COM5"`)

Hőmérő grafikus kijelzéssel

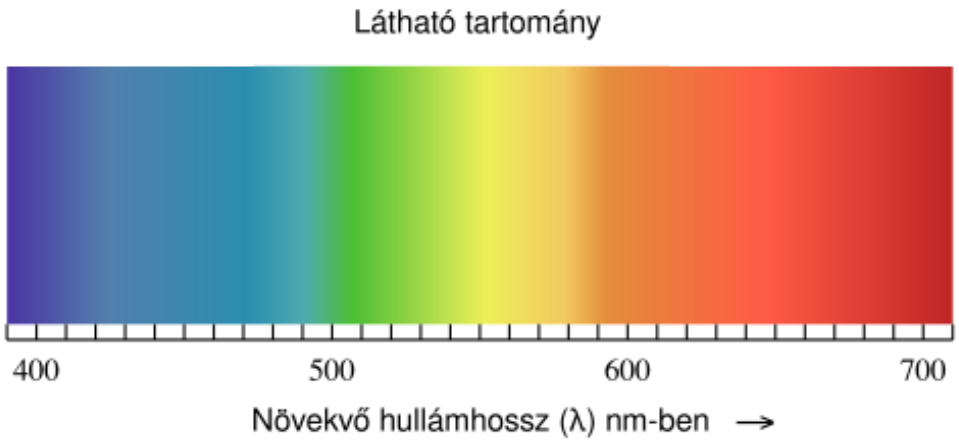
- A "lejátszás" gombra kattintva elindul a program, s a soros porton keresztül fogadja az adatokat az **ESP32**-től

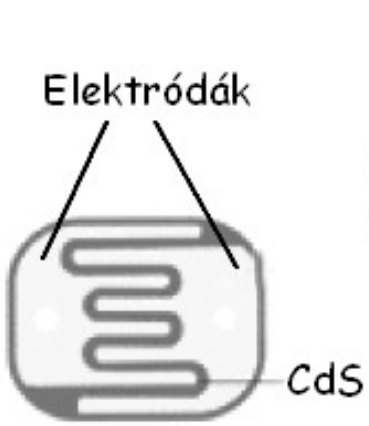


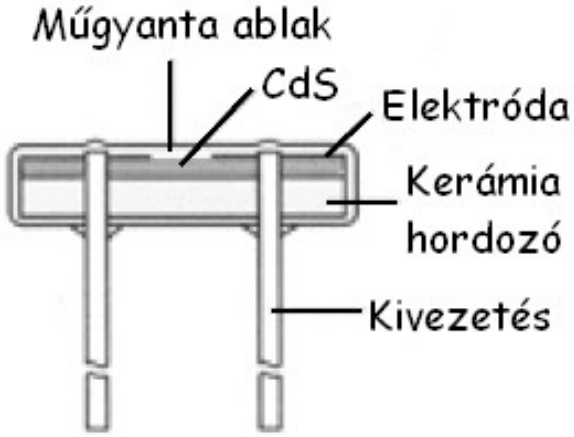


CdS fényérzékeny ellenállás

A GL55 típusú kadmiumszulfid (CdS) anyagú ellenállások vezetőképessége a fény hatására növekszik.
A spektrális érzékenység maximuma 540 nm (sárgászöld).







Típus	Sötét-ellenállás	Ellenállás @ 10 lx	γ	Válaszidő
GL5528	1 MΩ	10 – 20 kΩ	0.6	20 – 30 ms

$$\gamma = lg\left(\frac{R10}{R100}\right)$$

R10 és R100 a 10 és 100 lux-nál mért ellenállás értékek.

Fotometriai alapfogalmak

Fényforrás fényerőssége: az egységnyi térszögbe kisugárzott fényteljesítmény (ref 555 nm). Mértékegysége: 1 cd (= 1W/683 sr), kb. 1 gyertya fényerőssége. Egy 100 W izzó fényerőssége kb. 120 cd.

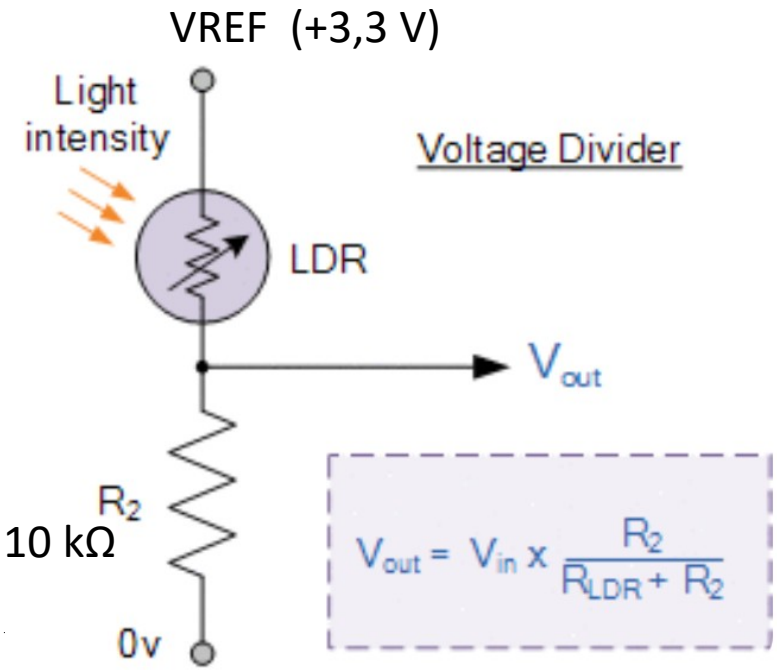
Fényáram: a fényerősség és a kisugárzási térszög szorzata. Mértékegysége: 1 lm (= 1 cd · 1 sr) Egy 100 W izzólámpa fényárama kb. 1380 lumen.

Megvilágítás erőssége: az adott felületre eső fényáram és a felület nagyságának hánydosa. Mértékegysége: 1 lx (1 lux a megvilágítása annak a felületnek, amelynek 1 négyzetméterére merőlegesen és egyenletesen 1 lumen fényáram esik). Egy 100 W izzó 1,5 m távolságból kb. 100 lx megvilágítást eredményez.

Környezet	Megvilágítás (lux)
Iroda	500
Folyosó	50
Napfény nyáron	100 000
Napfény télen	10 000
Telihold	0,2
Színérzékelés határa	3

ESP32_LDR_test.ino

- Az LDR és a 10 kΩ-os ellenállás feszültségosztót alkot. Az osztó feszültségét az ADC segítségével megmérve, az LDR ellenállása meghatározható

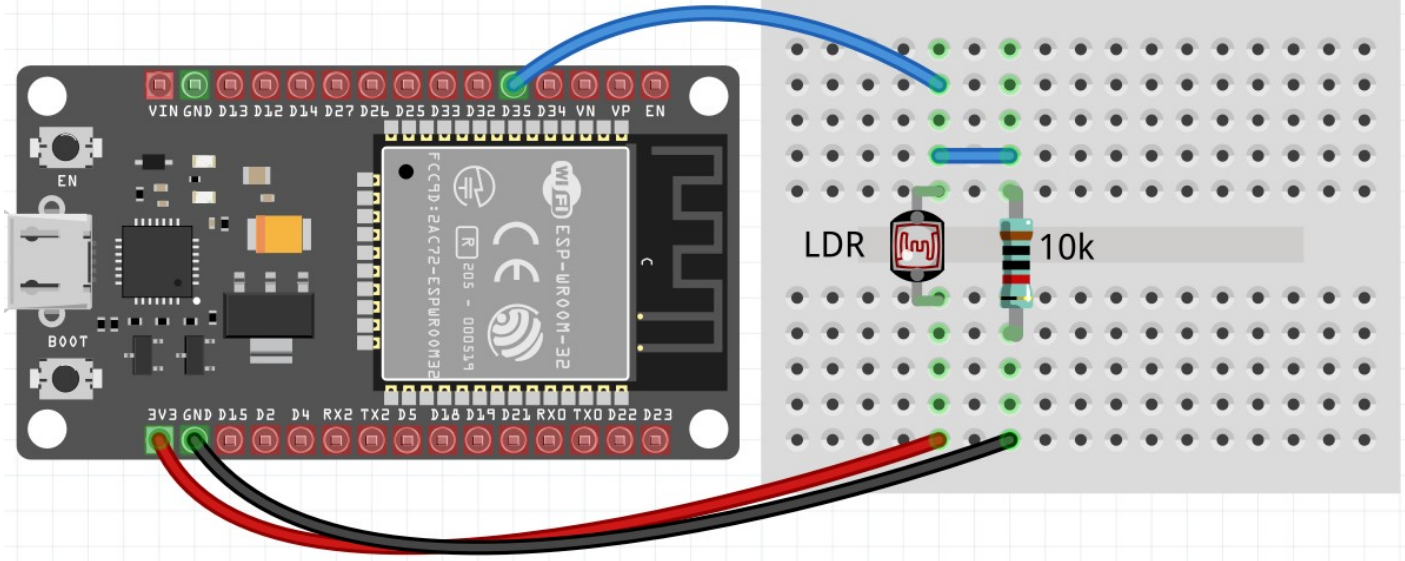


$$V_{OUT} = V_{REF} \cdot \frac{R_2}{R_{LDR} + R_2}$$

$$V_{OUT} = N_{ADC} \cdot V_{REF} / 4096$$

$$\frac{N_{ADC}}{4096} = \frac{R_2}{R_{LDR} + R_2}$$

$$R_{LDR} = \frac{R_2 \cdot 4096}{N_{ADC}} - R_2$$



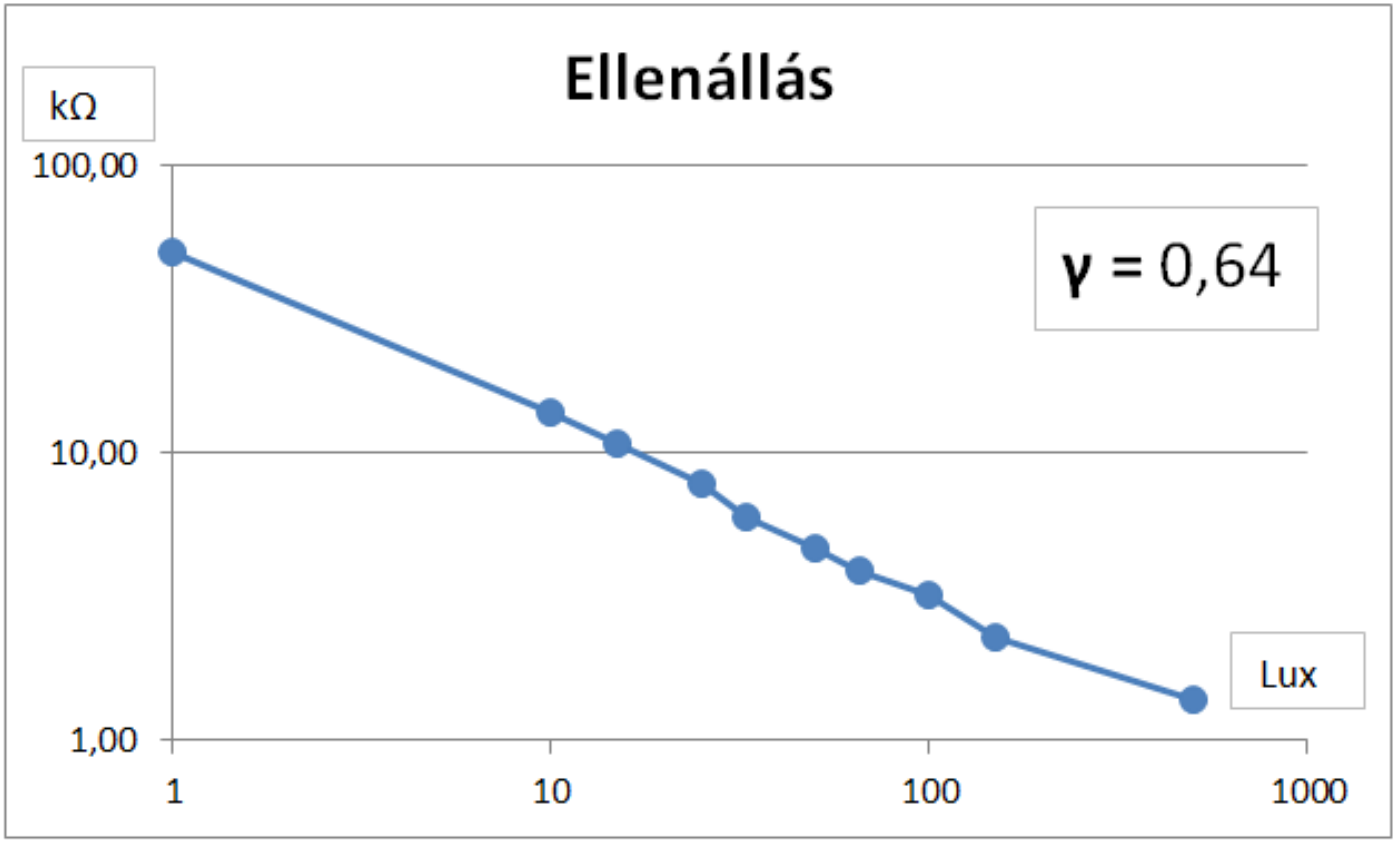
ESP32_LDR_test.ino

```
#include <Arduino.h>
#include "LUT.h"                                  // Kalibrációs táblázat
#define ADC_PIN 35                                // pin 35 lesz a bemenet
#define VREF   3.30                               // Referencia feszültség
void setup() {
  Serial.begin(115200);                           // baudrate = 115200 bit/s
}
void loop() {
  uint32_t rawReading = 0 ;
  for (int i = 0; i < 1024; i++) {                // 1024 mérést átlagolunk
    rawReading += analogRead(ADC_PIN);            // az ADC kiolvasása
  }
  rawReading = rawReading >> 10;                  // osztás 1024-gyel
  int calibratedReading = ADC_LUT[rawReading]; // a kalibrált érték előkeresése
  float voltage = calibratedReading*VREF/4096; // az osztási pont feszültsége
  Serial.print("Voltage = ");
  Serial.print(voltage,3);
  float Rx = 10000.0;                             // Kiszámítjuk az ellenállást
  if(calibratedReading > 0) {                     // ha nem nulla az ADC eredménye
    Rx = 40960.0f/calibratedReading - 10.0f;    // Rx + 10k = 10k * 4096/NADC
  }
  Serial.print(" V  Resistance = ");
  Serial.print(Rx,1);
  Serial.println(" kOhm");
  delay(2000);                                    // Pihenünk egy kicsit...
}
```

A fotoellenállás kalibrálása

Feladat: változtassuk a megvilágítást, és mérjük meg az LDR ellenállását a megvilágítás függvényében! (Luxmérő alkalmazást találunk az okostelefonokhoz is)

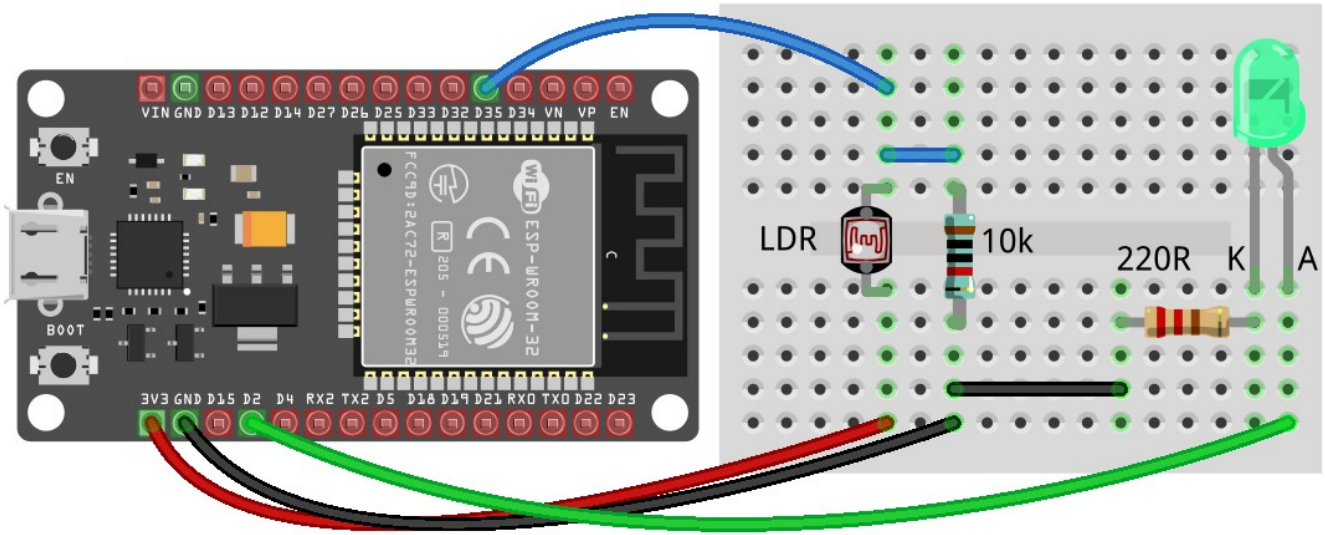
Megvilágítás [lx]	Ellenállás [kΩ]
1	50,00
10	14,00
15	10,90
25	7,80
33	6,00
50	4,70
65	3,90
100	3,20
150	2,30
500	1,40



Mindkét skála logaritmikus!

Szürkületkapcsoló

- Egészítsük ki az előző kapcsolást egy LED-del, amit most a **GPIO2** kivezetésre kötünk!
- Határozzuk meg azt a szintet, ahol be akarjuk kapcsolni a LED-et! A közzétett példában 2500 a kapcsolási szint, az ADC kiolvasás ennél kisebb értékeinél felkapcsoljuk, nagyobbaknál lekapcsoljuk a LED-et.

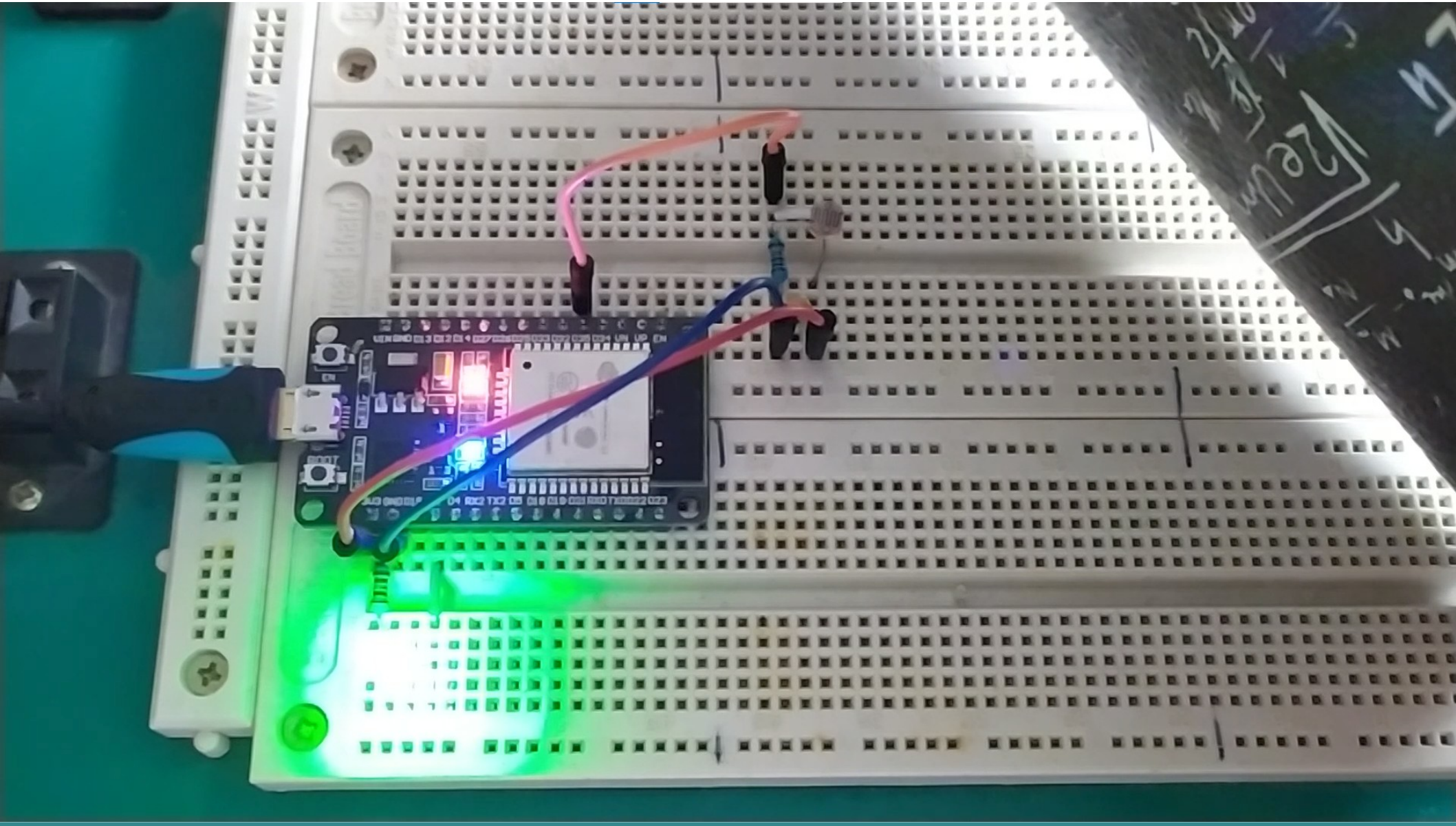


fritzing

ESP32_twighlight_switch.ino

```
#include <Arduino.h>
#include "LUT.h"                                  // Kalibrációs táblázat
#define ADC_PIN    35                             // GPIO35 lesz a bemenet
#define LED_PIN     2                             // GPIO2 a digitális kimenet
#define VREF     3.30                             // Referencia feszültség
#define SW_LEVEL 2500                             // Switch level
void setup() {
  analogReadResolution(12);                       // 12 bites felbontást
  analogSetAttenuation(ADC_11db);                 // Méréshatár 0-3,3 V
  pinMode(LED_PIN, OUTPUT);                       // legyen digitális kimenet
}

void loop() {
  uint32_t rawReading = 0 ;
  for (int i = 0; i < 1024; i++) {                // 1024 mérést átlagolunk
    rawReading += analogRead(ADC_PIN);            // az ADC kiolvasása
  }
  rawReading = rawReading >> 10;                  // osztás 1024-gyel
  int calibratedReading = ADC_LUT[rawReading]; // a kalibrált érték előkeresése
  if (calibratedReading < SW_LEVEL) {
    digitalWrite(LED_PIN,HIGH);                   // Lámpa felkapcsolása
  } else {
    digitalWrite(LED_PIN,LOW);                    // Lámpa lekapcsolása
  }
  delay(100);                                     // Pihenünk egy kicsit...
}
```

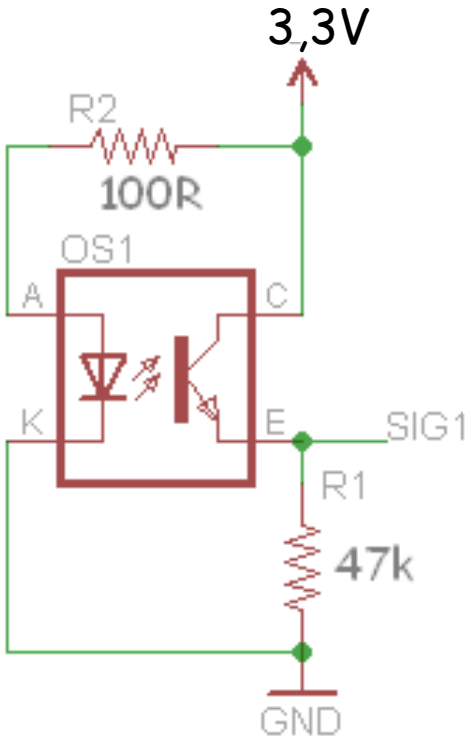
ESP32_twighlight_switch futási eredmény

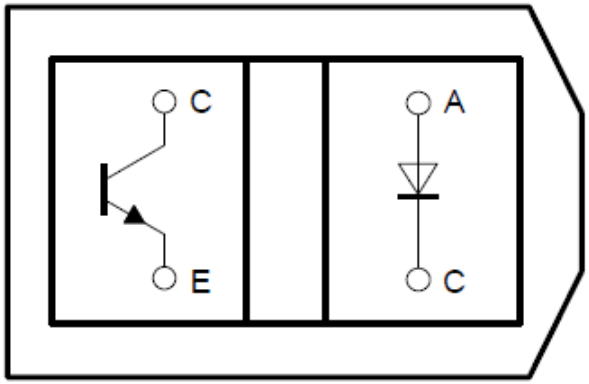


TCRT5000 reflektív optikai érzékelő

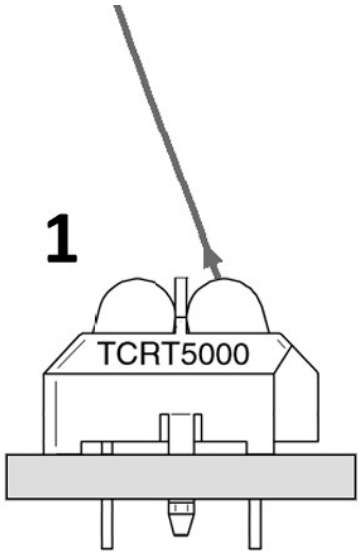
Egy IR LED-et és egy fototranzisztort tartalmaz.

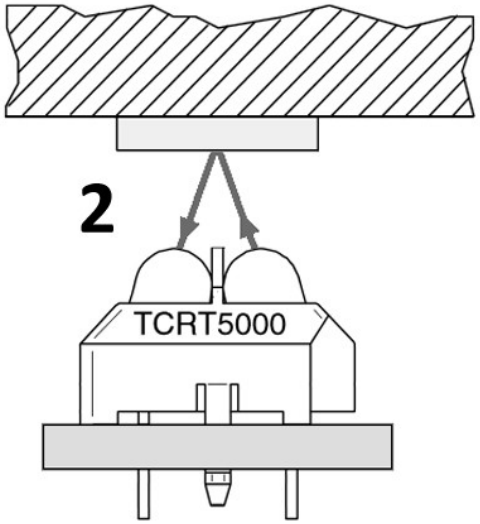
Működési elv: ha nincs akadály, a fény nem verődik vissza, a tranzisztor nem érzékel fényt.
Visszaverődés esetén a távolságtól és a reflexiós tényezőtől függ a visszaverődési arány.





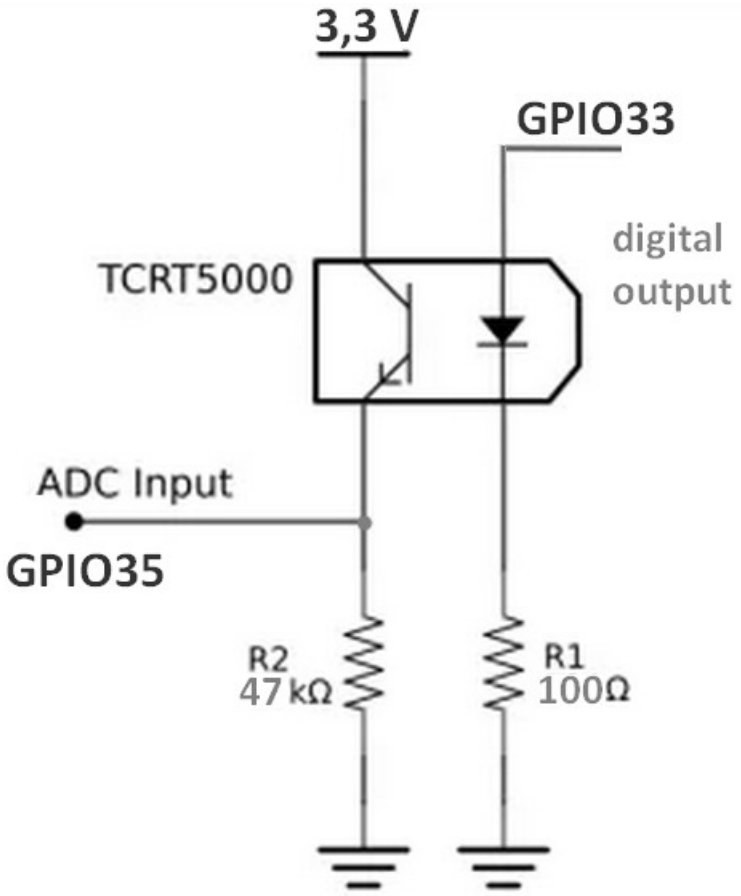
felülnézet

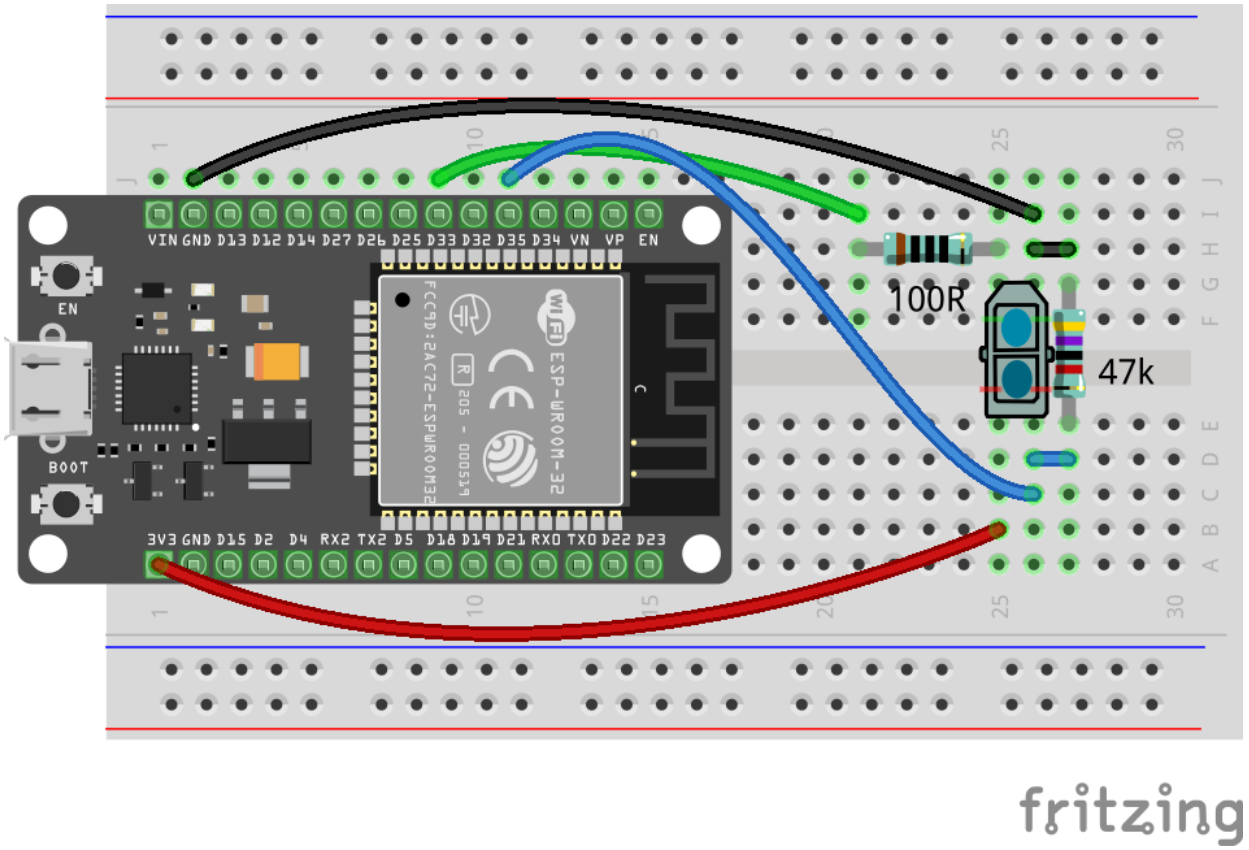




Bekötési vázlat

Fototranzisztornál mindegy, hogy kollektor- vagy emitter oldalra kerül a munkaellenállás. Mi most emitter oldalra tettük, így a nagyobb feszültség nagyobb megvilágítottságot jelent.





ESP32_TCRT5000_test.ino 2/1.

- Az alábbi programban felkapcsolt és lekapcsolt LED mellett is mérjük a beeső fényt, s a kettő különbségét tekintjük a visszavert fénynek (a többi a szórt környezeti fény hatása)
- Minden ciklusban 1024 mérést végzünk és a kapott adatokat átlagoljuk

```
#include <Arduino.h>
#include "LUT.h"                                    // Kalibrációs táblázat
#define ADC_PIN    35                               // GPIO35 lesz a bemenet
#define LED_PIN    33                               // GPIO33 digitális kimenet
#define VREF     3.30                               // Referencia feszültség

void setup() {
  analogReadResolution(12);                         // 12 bites felbontást
  analogSetAttenuation(ADC_11db);                   // Méréshatár 0-3,3 V
  pinMode(LED_PIN, OUTPUT);                         // legyen digitális kimenet
  Serial.begin(115200);                             // baudrate = 115200 bit/s
  while (!Serial) {}                                // Várunk a soros portra
}
```

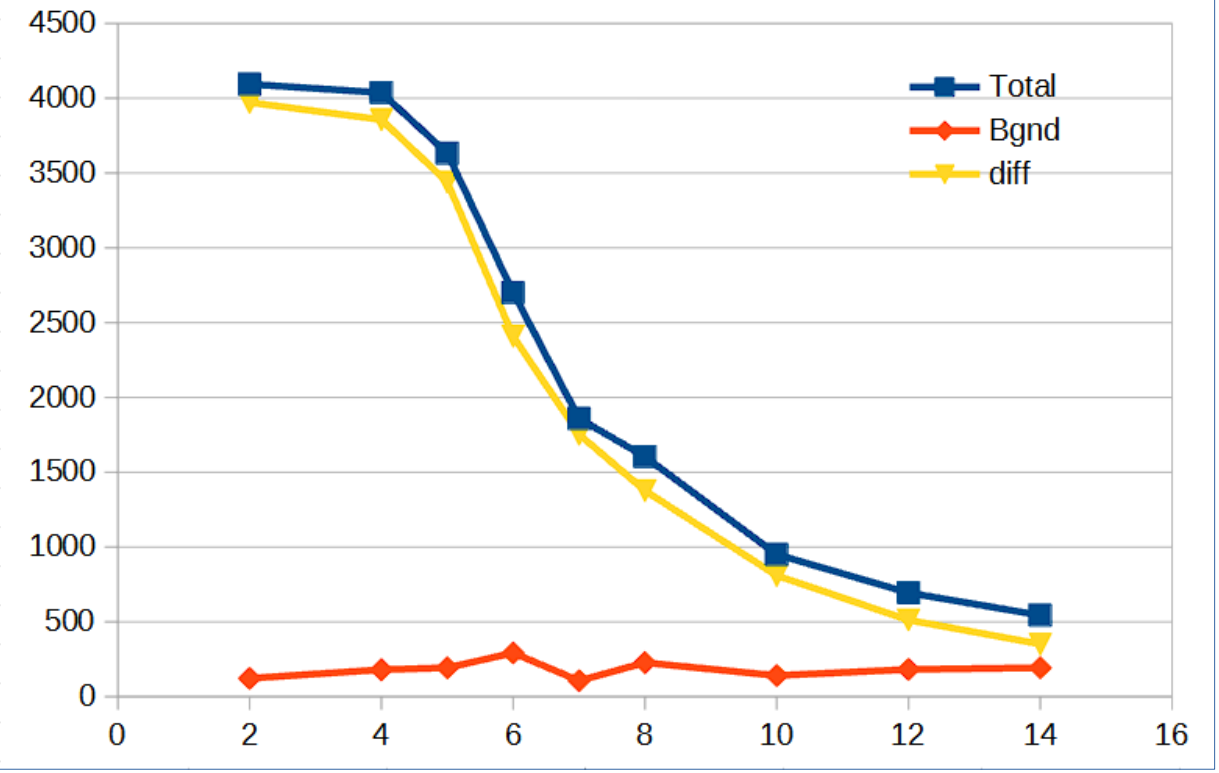
ESP32_TCRT5000_test.ino 2/2.

```
void loop() {
  int sum0 = 0 ;                                  // Háttérfény mérés, LED ki
  digitalWrite(LED_PIN,LOW);
  for (int i = 0; i < 1024; i++) {                // 1024 mérést átlagolunk
    sum0 += analogRead(ADC_PIN);                  // az ADC kiolvasása
  }
  sum0 = sum0 >> 10;                              // osztás 1024-gyel
  sum0 = ADC_LUT[sum0];                           // a kalibrált érték előkeresése

  int sum1 = 0 ;                                  // Visszavert fény mérése, LED be
  digitalWrite(LED_PIN,HIGH);
  for (int i = 0; i < 1024; i++) {                // 1024 mérést átlagolunk
    sum1 += analogRead(ADC_PIN);                  // az ADC kiolvasása
  }
  digitalWrite(LED_PIN,LOW);
  sum1 = sum1 >> 10;                              // osztás 1024-gyel
  sum1 = ADC_LUT[sum1];                           // a kalibrált érték előkeresése
  int diff = sum1 - sum0;
  if(diff < 0) diff = 0;                          // Negatív érték kiszűrése
  Serial.print(" Összes: ");                      // Eredmények kiírása
  Serial.print(sum1);
  Serial.print(" Háttér: ");
  Serial.print(sum0);
  Serial.print(" Visszavert: ");
  Serial.println(diff);
  delay(2000);                                     // Pihenünk egy kicsit...
}
```

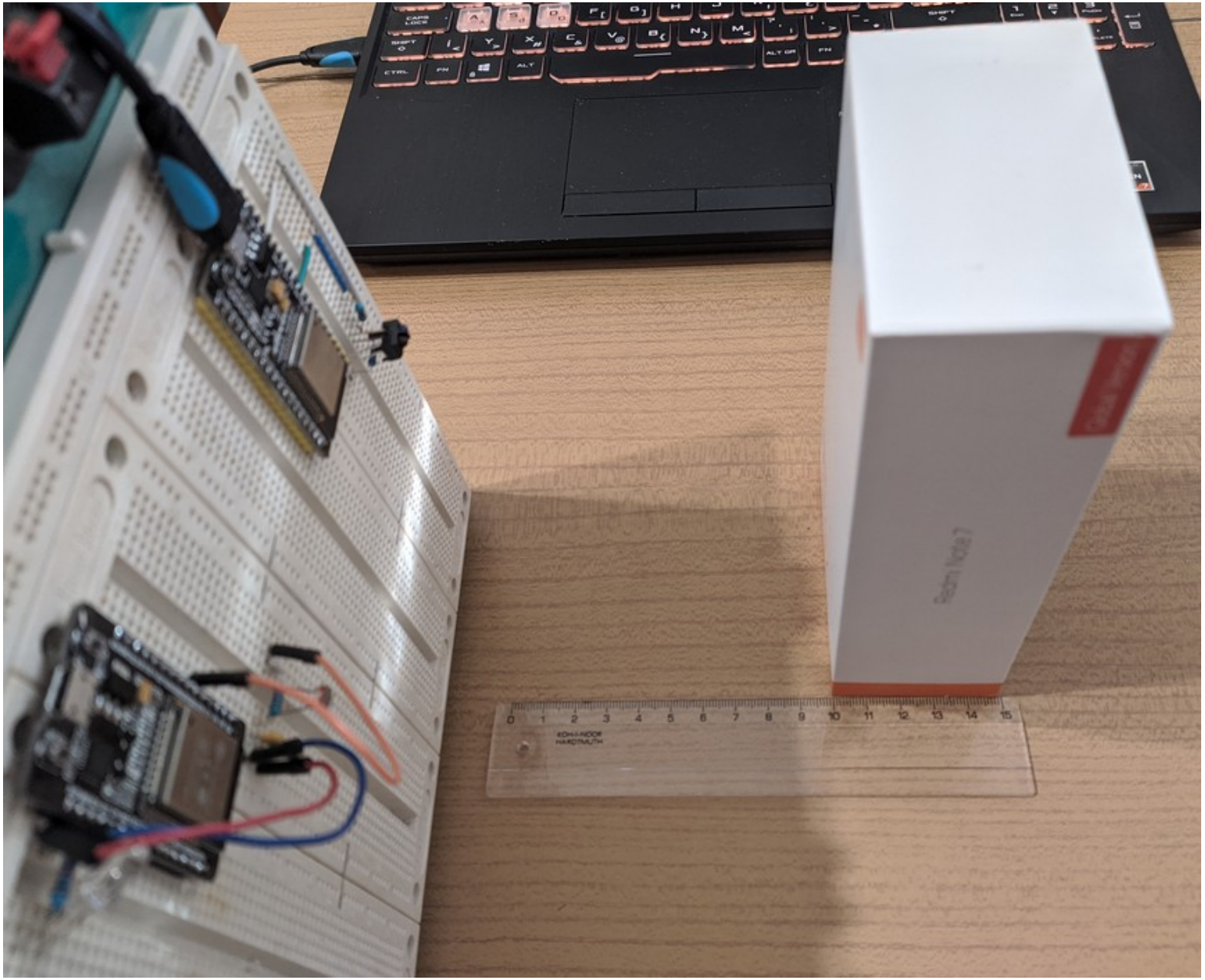
ESP32_TCRT5000_test eredménye

- Az alábbi ábrán az összes beeső fény, a szórt háttér és ezek különbsége látható (ezt a különbséget tekinthetjük a visszevert fénynek)
- Az érzékenységet a munkaellenállás cseréjével (22 kΩ) csökkentettük, és az ablakon beeső fényt is leárnyékoltuk

d [cm]	Total	Bgnd	diff
2	4094	122	3972
4	4037	180	3857
5	3632	193	3439
6	2701	292	2409
7	1859	106	1753
8	1604	227	1377
10	950	141	809
12	694	182	512
14	544	193	351



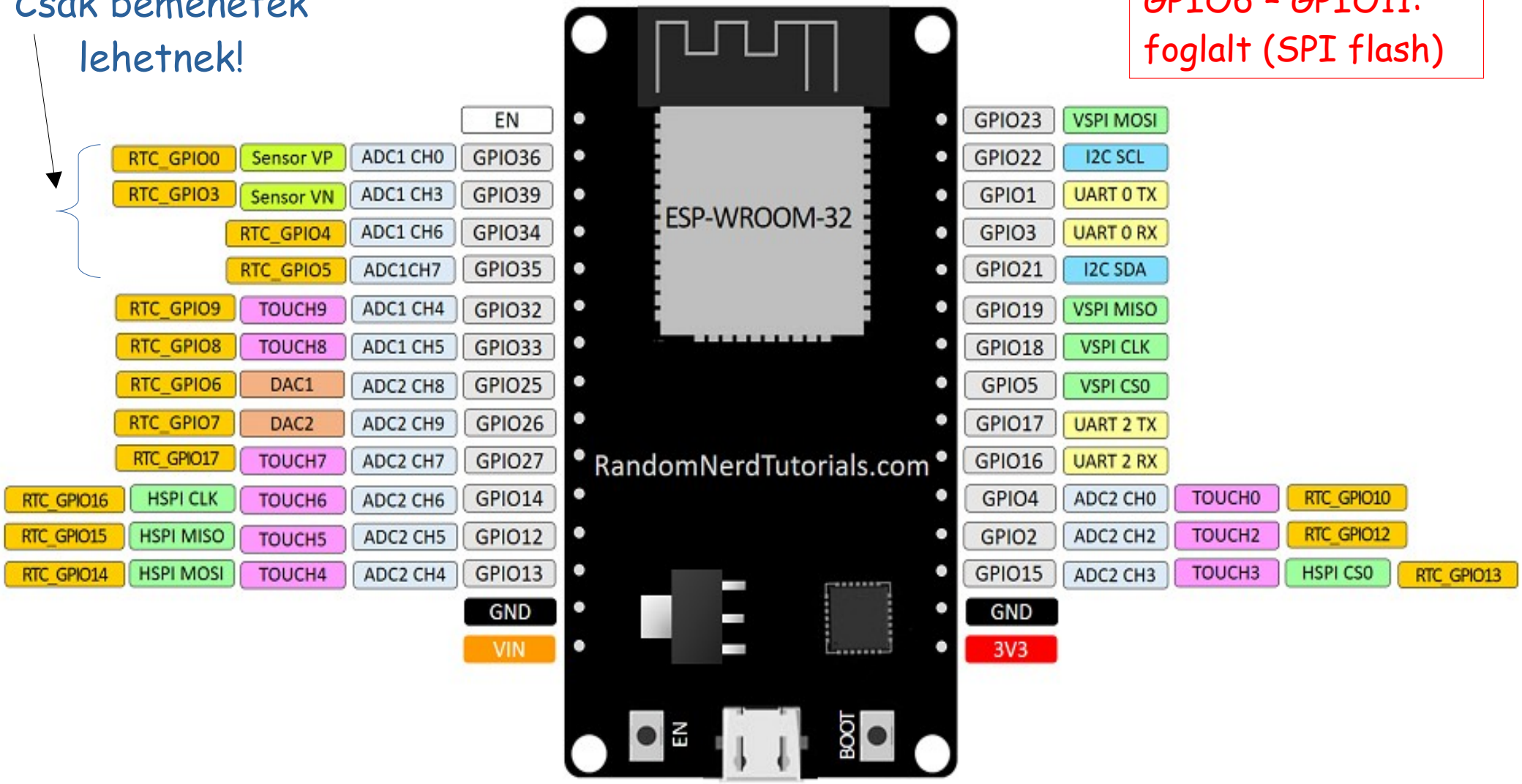
Mérési elrendezés



A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 – GPIO11: foglalt (SPI flash)



- **Forrás:** randomnerdtutorials.com/getting-started-with-esp32/

Ellenállás színkódok