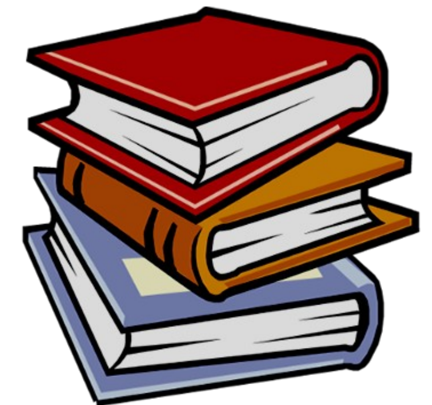
ESP32 mikrovezérlők programozása Arduino környezetben

3. Érintésérzékelés, impulzusszélesség-moduláció

Felhasznált irodalom

- Microcontrollerslab: ESP32 touch sensor
- Random nerd tutorials: ESP32 Capacitive Touch Sensor Pins with Arduino IDE
- Khaled Magdy: ESP32 PWM Tutorial & Examples
- Ravi Teja: In-depth ESP32 PWM Tutorial
- Joshua Hrisko: Arduino Breathing LED Functions
- Espressif: ESP32 Technical Reference Manual



Példaprogramok

ESP32_touch_test – érintésérzékelés próba

ESP32_touch_ledswitch – LED vezérlés szenzorral

ESP32_touch_slider – érintésérzékelő csúszka

ESP32_pwm_ledfade – LED fényerő vezérlése

ESP32_pwm_ledfade2 – „lélegző” LED

ESP32_pwm_rgbled – színkeverés RGB LED-del és három potméterrel

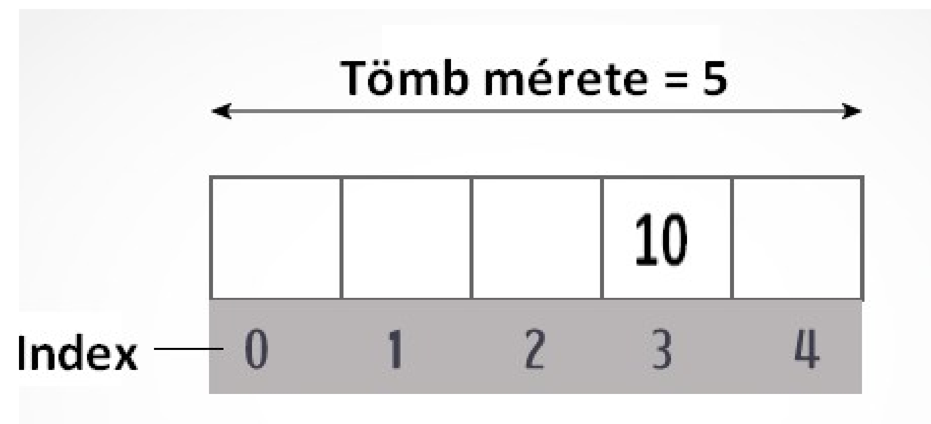
ESP32_pwm_music – egyszerű hangkeltés

Tömbök

- A tömbváltozó közös névvel ellátott értékek/számok gyűjteménye, melyeket az index (sorszám) szerint érhetünk el. A tömbök indexelése a 0 sorszámmal kezdődik.
- A tömböt - a többi változóhoz hasonlóan - használat előtt deklarálni kell és lehetőség szerint a kezdőértékkel feltölteni, például:

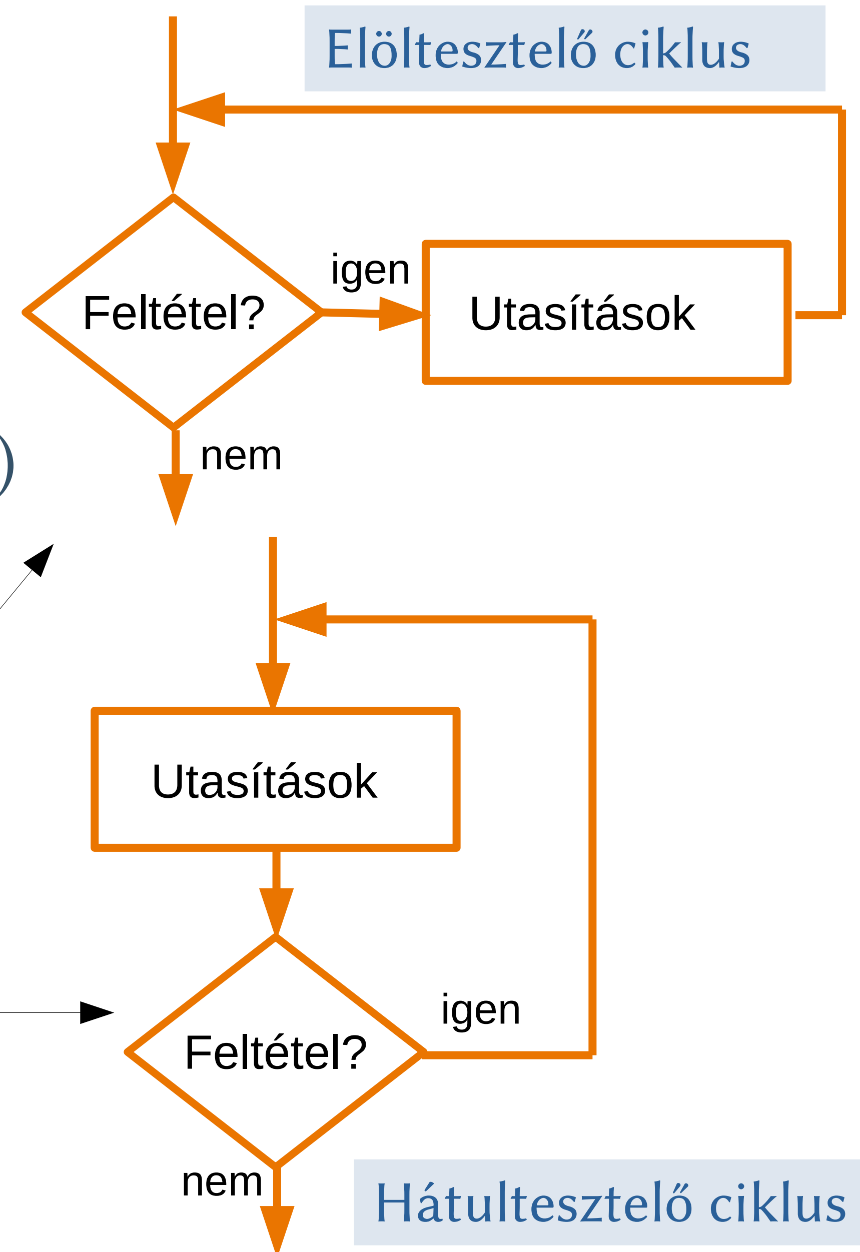
```
int a[5];        // Egy 5 elemű, int típusú tömböt deklarálunk
a[3] = 10;       // Az a tömb 4. eleme 10 lesz
int x = a[3];    // Az x változó értéke 10 lesz
```

- A tömbök egyik haszna az, hogy programciklusban egy számlálót növelgetve végig tudjuk futtatni a tömbelemek sorszámain (lásd majd a **for** ciklusnál!)
- A tömbök másik haszna az, hogy függvényeknek történő paraméter-átadásnál elegendő a tömb nevét és az elemek számát megadni



Programciklusok szervezése

- Ismétlődő feladatok esetén az utasításainkat „újrahasznosíthatjuk” **ciklusok** szervezésével, amikor egy utasításblokkot többször lefuttatunk
- Nem (csak) végtelen ciklust akarunk, ezért kell bennmaradási (vagy kilépési) **feltétel**
- A feltételt az utasításblokk végrehajtása előtt vagy után is vizsgálhatjuk:
  - Előltesztelő ciklus: **while, for**
  - Hátultesztelő ciklus: **do … while**



Ciklusszervezés `for` és `while` utasítással

```
for (int i=0; i<5; i++) { utasítások }
```

- A **`for`** utasítás sorszámozott elöltesztelő ciklust szervez
  1) Az első kifejezésben szereplő változó felveszi a kezdőértéket
  2) Kiértékelésre kerül a második kifejezés, s teljesülés (nem nulla érték) esetén végrehajtásra kerül a ciklus törzse
  3) A ciklustörzs lefutása után végrehajtásra kerül a harmadik kifejezésben szereplő művelet, majd visszatérünk a 2. ponthoz
- A **`while`** utasításnál csak a kiértékelendő feltétel adható meg

Kezdőérték feltétel ciklusváltozó léptetés

```
for(int i = 0; i < 5; i++) {
  digitalWrite(LED,HIGH);
  delay(50);
  digitalWrite(LED,LOW);
  delay(500);
}
```

FOR ciklus

```
int i = 0;            // Kezdőérték
while (i < 5) {       // Feltétel
  digitalWrite(LED,HIGH);
  delay(50);
  digitalWrite(LED,LOW);
  delay(500);
  i++;                // Ciklusváltozó léptetés
}
```

WHILE ciklus

Átlagolás a for ciklus használatával

- Több mérés eredményét összeadjuk, majd elosztjuk az adatok számával (számtani átlag)
- Az alábbi példában a beépített Hall szenzorral végzünk 1000 db mérést és ezek eredményeinek számtani átlagát vesszük

```
void setup() {
  Serial.begin(115200);                 // Soros port inicializálása
}

void loop() {
  int32_t val = 0;                      // val lesz az összegző változó
  for (int i = 0; i < 1000; i++) {    // Ciklust szervezünk, 1000-szer fut le
    val += hallRead();                  // Minden új mérés eredményét hozzáadjuk
  }
  val = val / 1000;                     // Az összeget osztjuk a mérések számával
  Serial.println(val);                  // Kiíratjuk a végeredményt
}
```

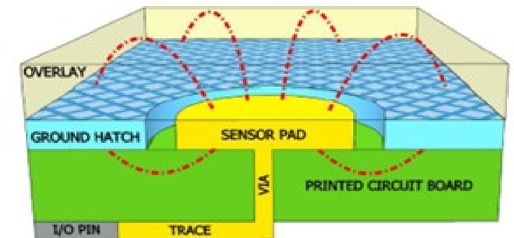
break utasítás – kilépés a ciklusból

- A **break** utasítás használatával kiléphetünk a **for, while** vagy **do…while** ciklusból, függetlenül a normál kilépési feltételtől. A **switch case** utasításokból is a **break** segítségével léphetünk ki
- Példa **for** ciklusból történő kilépésre: fokozatosan addig növelgetjük egy teljesítményszabályozó kimeneten a PWM jel kitöltését (pl. fűtés), amíg az analóg bemenőjel (pl. hőmérő) jele el nem ér egy előre megadott szintet (treshold)

```
int threshold = 40;
int sens;
for (int x = 0; x < 255; x++) {
  analogWrite(PWMpin, x);
  sens = analogRead(sensorPin);
  if (sens > threshold) {       // kiugrás határérték túllépésnél
    x = 0;
    break;
  }
  delay(50);
}
```

Kapacitív érintésérzékelés

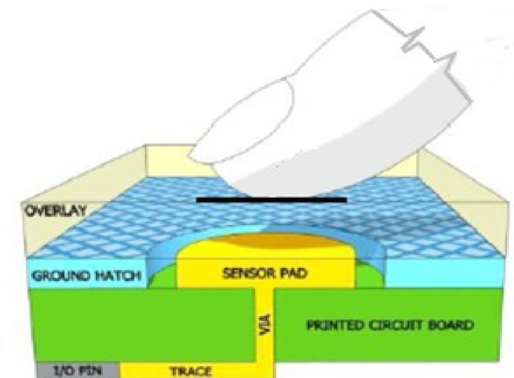
- Az érintésérzékelés megvalósítási lehetőségei: optikai, rezisztív, illetve kapacitív érzékelés – mi most az utóbbival foglalkozunk
- A kapacitív érintésérzékelés lényege: az ujjunkkal megérintett elektróda és a földpont közötti kapacitás megnövekedik, s ez a változás valamilyen módszerrel detektálható:
  - A feltöltött elektróda feszültsége töltésmegosztás miatt érintéskor lecsökken
  - Relaxációs oszcillátorba kötve a kapacitást, az érintés a frekvenciát megváltoztatja
  - Állandó árammal töltve/kisütve, érintéskor a töltési idő megváltozik



- A legegyszerűbb esetben az érintőgomb a nyomtatott áramkörön kialakított fóliadarab, amelyet a forrasztásvédő lakk elszigetel

$$c_X = c_P + c_F \qquad c_F = \varepsilon_0 \cdot \varepsilon_r \cdot A/d$$

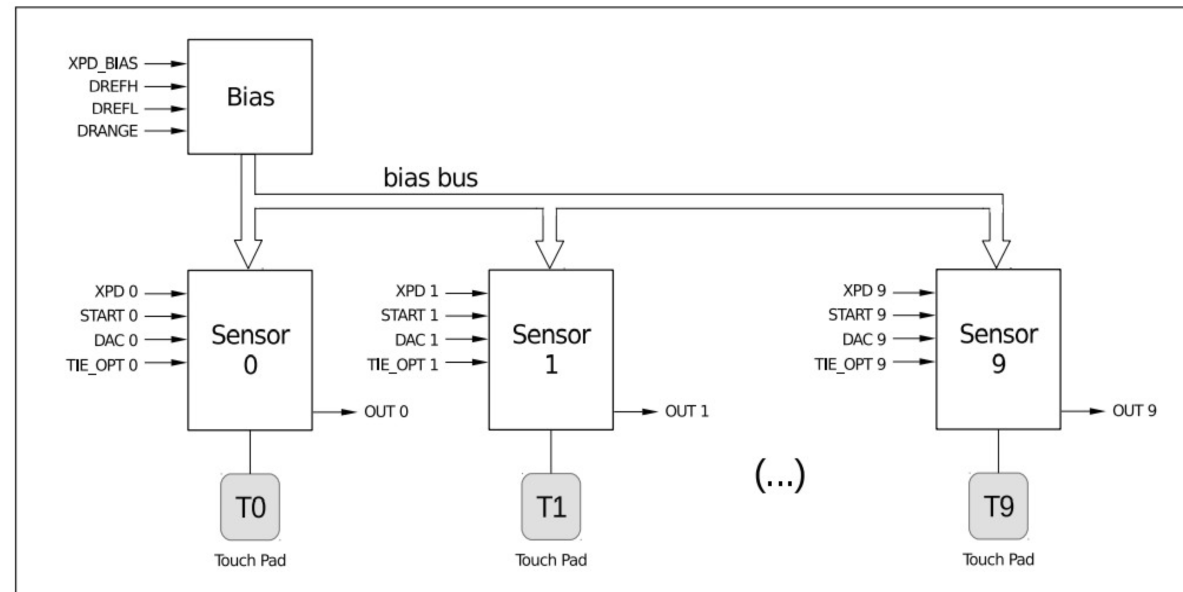
ahol A a felület, d a fedőréteg vastagsága

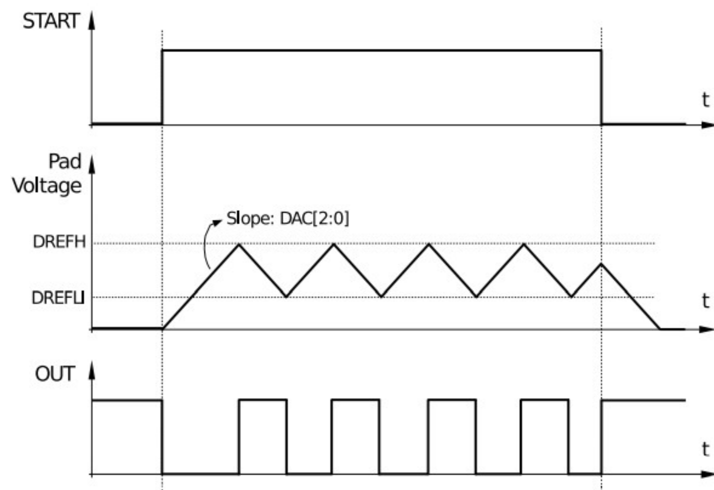


Érintésérzékelés az ESP32 mikrovezérlőben

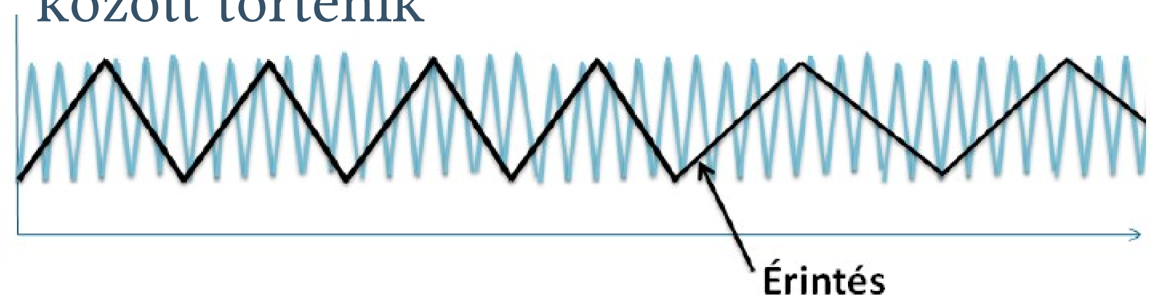
- 10 csatorna áll rendelkezésre. Ezekben periodikus feltöltés és kisütés zajlik, a periódusokat adott ideig számláljuk, az érzékelést egy véges állapotgép (FSM) vezérli

T0	GPIO4	T5	GPIO12
T1	GPIO0	T6	GPIO14
T2	GPIO2	T7	GPIO27
T3	GPIO15	T8	GPIO33
T4	GPIO13	T9	GPIO32



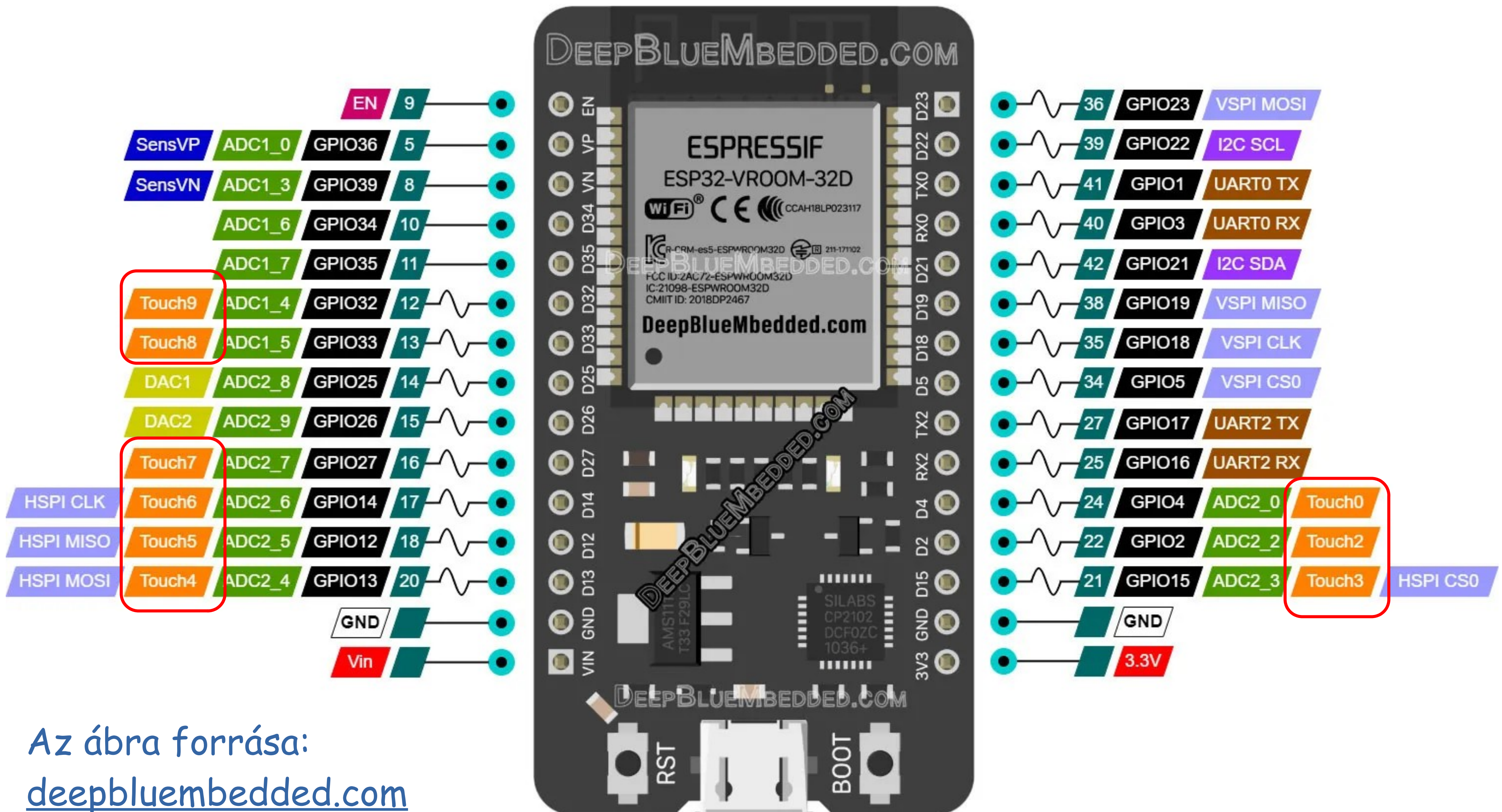


A feltöltés és kisütés a megadott alsó és felső feszültséghatárok (DREFH, DREFU) között történik



Érintésérzékelés az ESP32 mikrovezérlőben

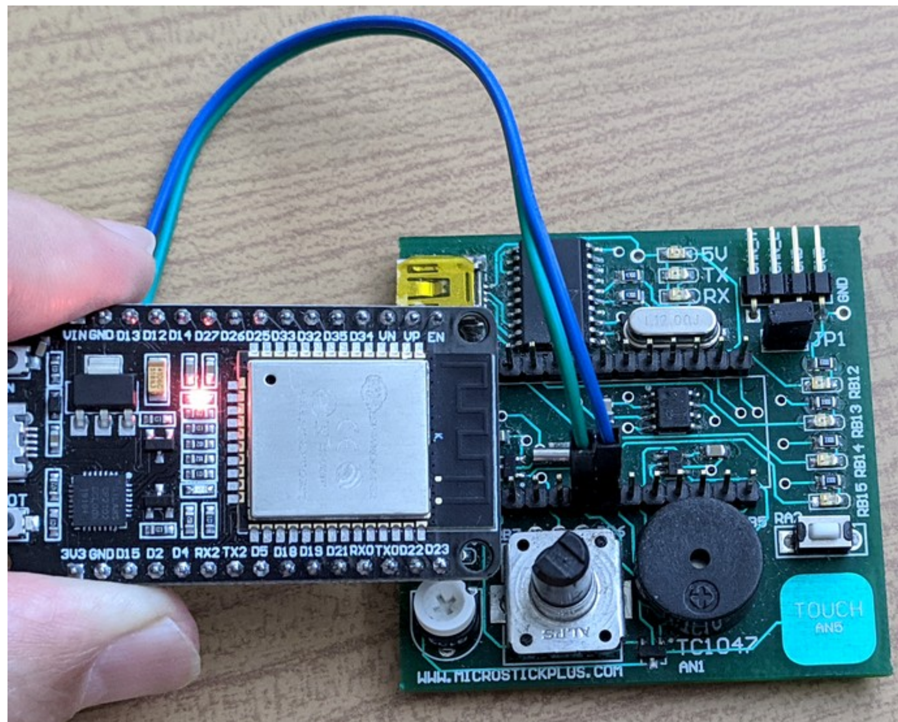
- Az **ESP32 Devkit-V1** kártya érintésérzékelő bemenetei az ábrán láthatók. A 30 tüskés kártyán a **Touch1** bemenet nincs kivezetve



Az ábra forrása: deepbluembedded.com

ESP32_touch_test.ino

- Egy kb. 1 cm^2-es fólia „gombot” kötöttünk a **T4 (GPIO13)** bemenetre (külön kártya esetén a GND pontokat is kössük össze)
- A kiolvasás a **touchRead**(*pin*) függvénnyel történik




```
#define TOUCH_PIN 13  //T4-et is írhatunk
void setup() {
  Serial.begin(115200);
}

void loop() {
  int touch_value = touchRead(TOUCH_PIN);
  Serial.print("touch sensing value = ");
  Serial.println(touch_value);
  delay(1000);
}
```

```
Testing the T4 (GPIO13) touch sensing input
touch sensing value = 41
touch sensing value = 48
touch sensing value = 48
touch sensing value = 48
touch sensing value = 48
touch sensing value = 48
touch sensing value = 47
touch sensing value = 18
touch sensing value = 18
touch sensing value = 20
touch sensing value = 2
touch sensing value = 18
```

Elengedett állapot

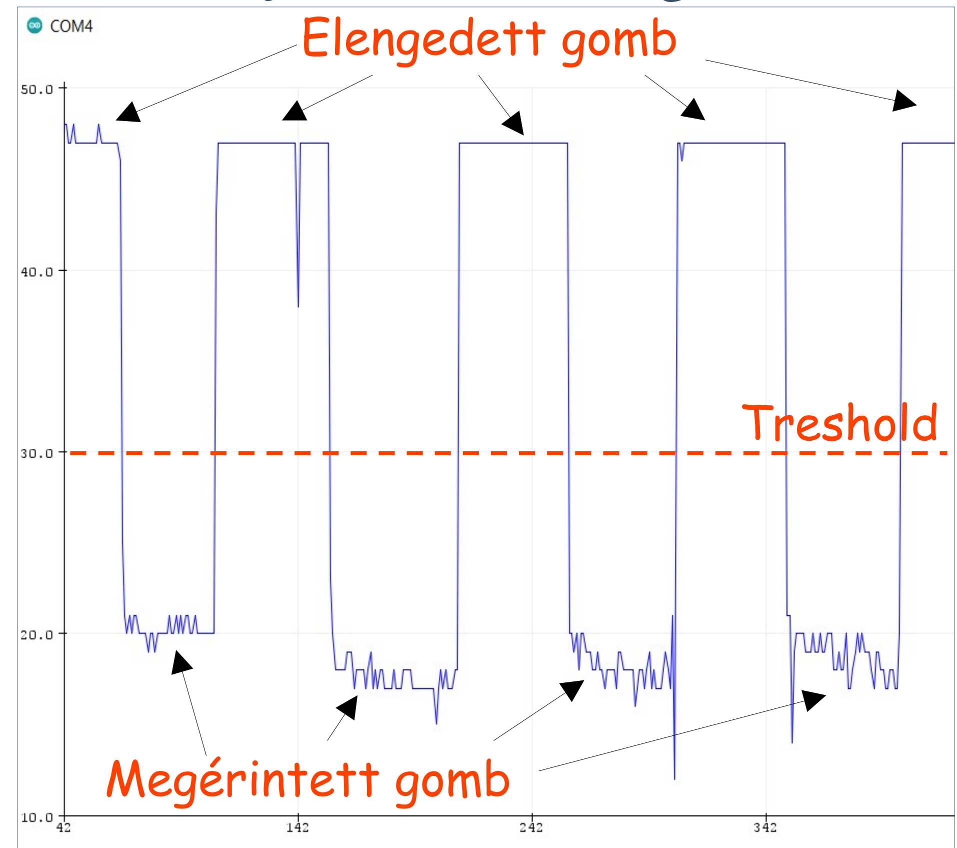
Megérintett állapot

ESP32_touch_ledswitch.ino

- A **T4 (GPIO13)** bemenetre kötött érintőgombbal vezéreljük a beépített LED-et (GPIO2): a LED akkor világít, amikor a gombot megérintjük
- A kvázi analóg jelleg miatt egy küszöbszintet (threshold) jelölünk ki

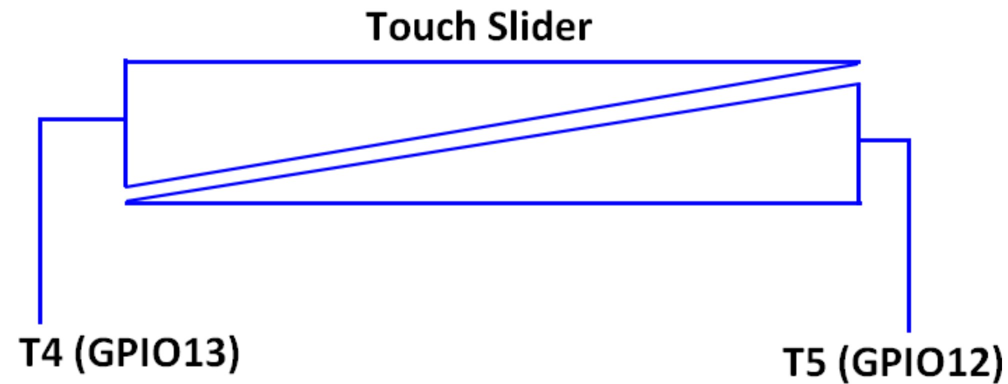
```
#define TOUCH_PIN 13
#define LED_PIN   LED_BUILTIN
#define THRESHOLD  30

void setup() {
  Serial.begin(115200);
  pinMode (LED_PIN, OUTPUT);
}

void loop() {
  int touch_value = touchRead(TOUCH_PIN);
  Serial.println(touch_value);
  if (touch_value < THRESHOLD) {
    digitalWrite(LED_PIN, HIGH);
  }
  else {
    digitalWrite(LED_PIN, LOW);
  }
  delay(100);
}
```

A kiírt adatokat a **Serial Plotter** ablakban jelenítettük meg

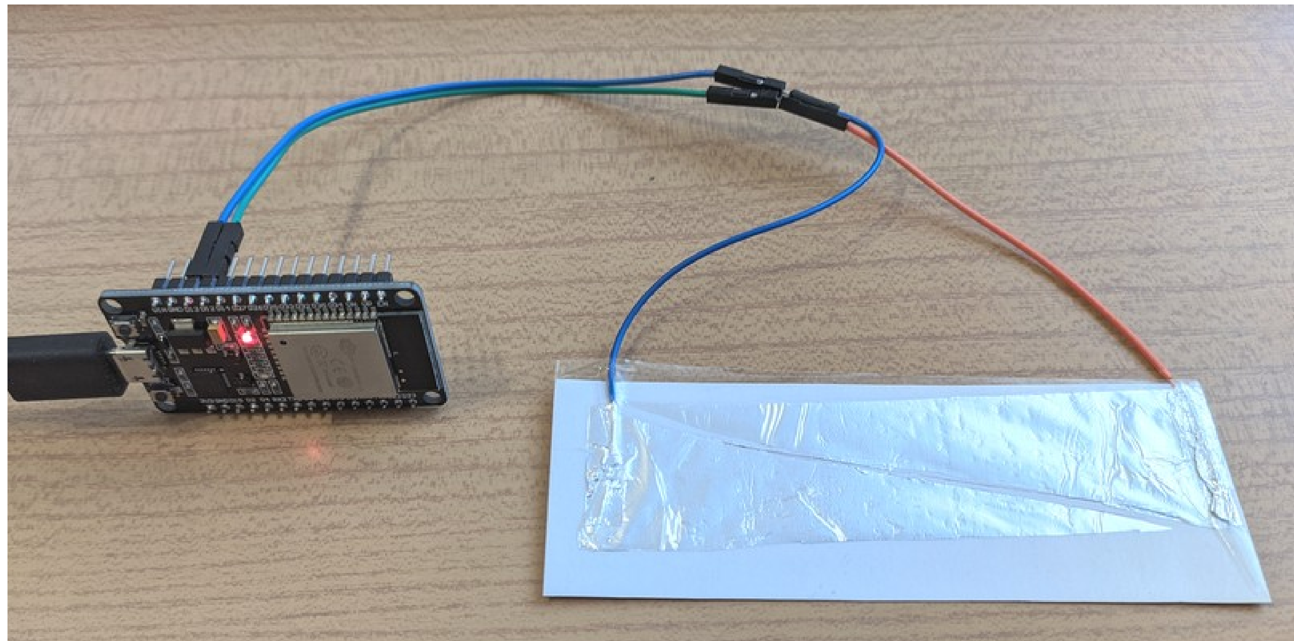


Érintőcsúszka készítése

- Az érintésérzékelés lehetőségeit felhasználva potméter helyettesítő csúszkát is készíthetünk, amelyhez két bemenetet használunk fel
- A két, háromszög alakú elektródát alumíniumfóliából vágtuk ki és celluxszal rögzítettük a karton alaplaphoz



- Ha az ujjunkat végighúzzuk a csúszkán, a kapacitás a két bemenet esetében inverz módon változik, s a kettő aránya jelzi, hogy hol érintettük meg a csúszkát



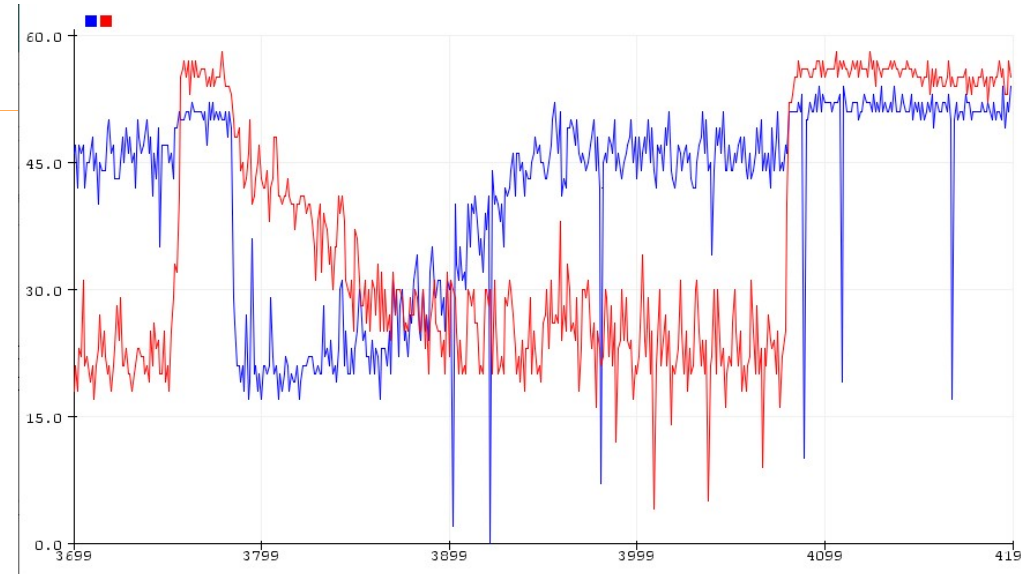
ESP32_touch_slider.ino

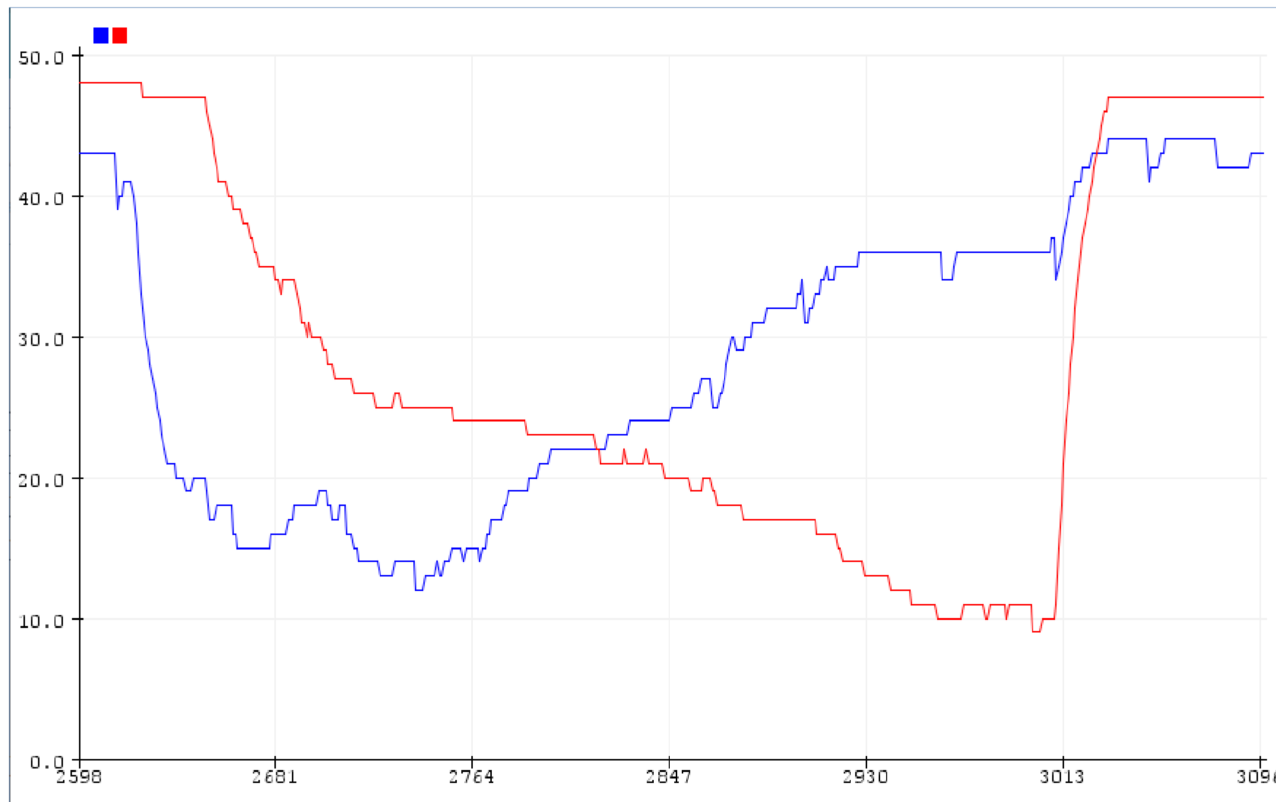
- Az alábbi programban az érintésérzékelők kiolvasásán és kiíratásán kívül egy ún. gördülő simítást is végzünk, az adatok szórása miatt

```
#define TOUCH_PIN_1 T4
#define TOUCH_PIN_2 T5
int touch_value1 = 40;
int touch_value2 = 40;

void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println("Touch slider on T4 and T5");
}

void loop() {
  touch_value1 = (9*touch_value1 + touchRead(TOUCH_PIN_1))/10;
  touch_value2 = (9*touch_value2 + touchRead(TOUCH_PIN_2))/10;
  Serial.print(touch_value1);
  Serial.print(" ");
  Serial.println(touch_value2);
  delay(50);
}
```

ESP32_touch_slider.ino

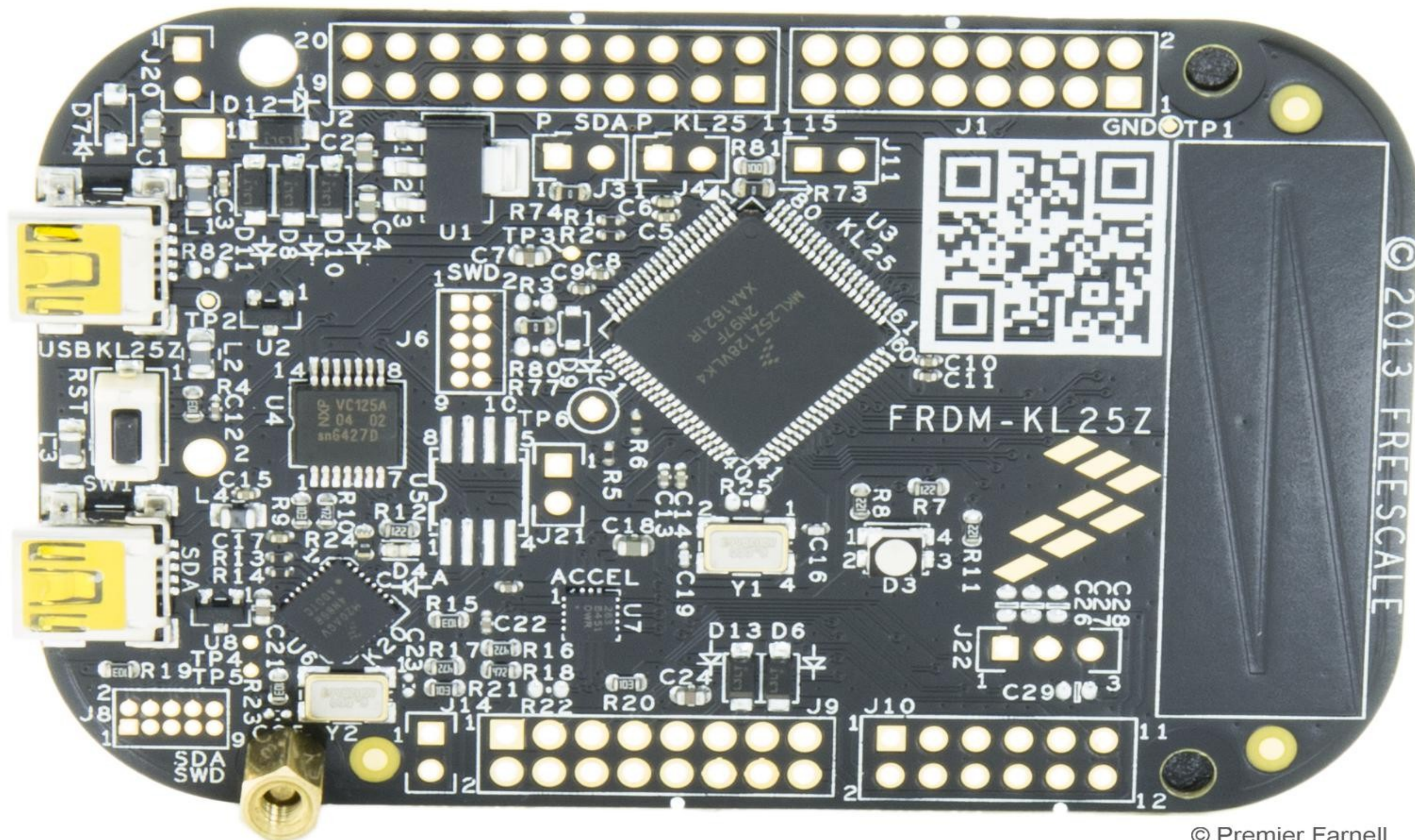
- Az ábrákon a simítás nélküli, illetve a simított adatok láthatók
- Az ujjunkat végighúzva a görbék láthatóan keresztezik egymást, de a kivitelezés még tökéletlen





Egy tökéletesebb csúszka

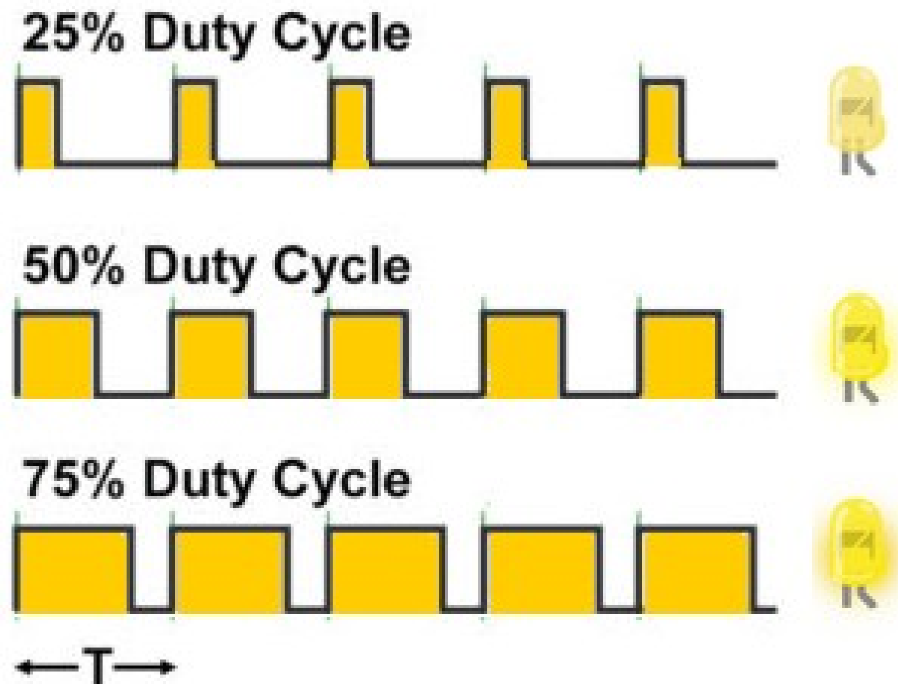
- A **FRDM-KL25Z** kártya kapacitív csúszkája fésűszerűen egymásba nyúló elektródái jobb pozíciófelismerést biztosítanak (kisebb a „mellényúlás” esélye)



© Premier Farnell

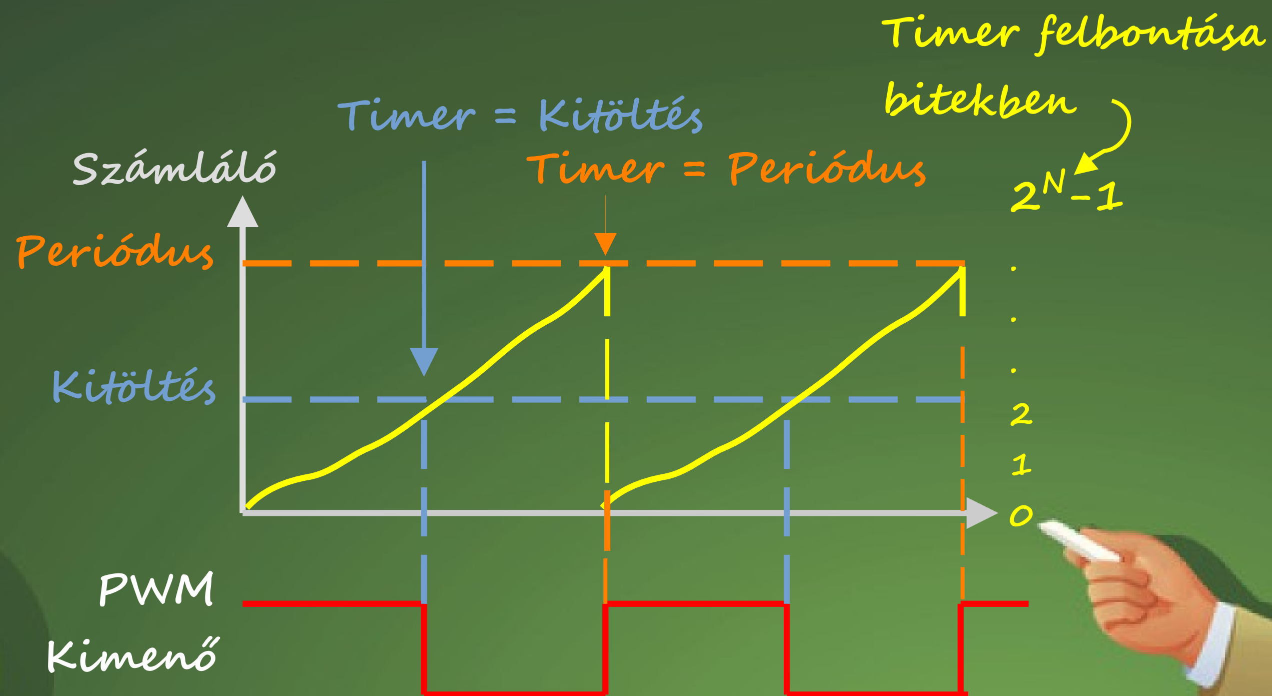
PWM: impulzus-szélesség moduláció

- **PWM** = pulse width modulation (impulzus-szélesség moduláció)
- Szerepe, hogy a ***frekvencia*** (vagy a periódusidő) megválasztása után a ***kitöltés*** (duty cycle) változtatásával pl. a fogyasztóra eső teljesítményt megváltoztassuk
- A frekvenciát úgy kell megválasztani, hogy a szemünket ne zavarja a villódzás *(f > 100 Hz)*
- 25 %-os kitöltésnél a LED csak ideje 1/4-ében világít
- 50 %-os kitöltésnél a fele időben világít
- 75 %-os kitöltésnél a LED az ide ¾ részében világít



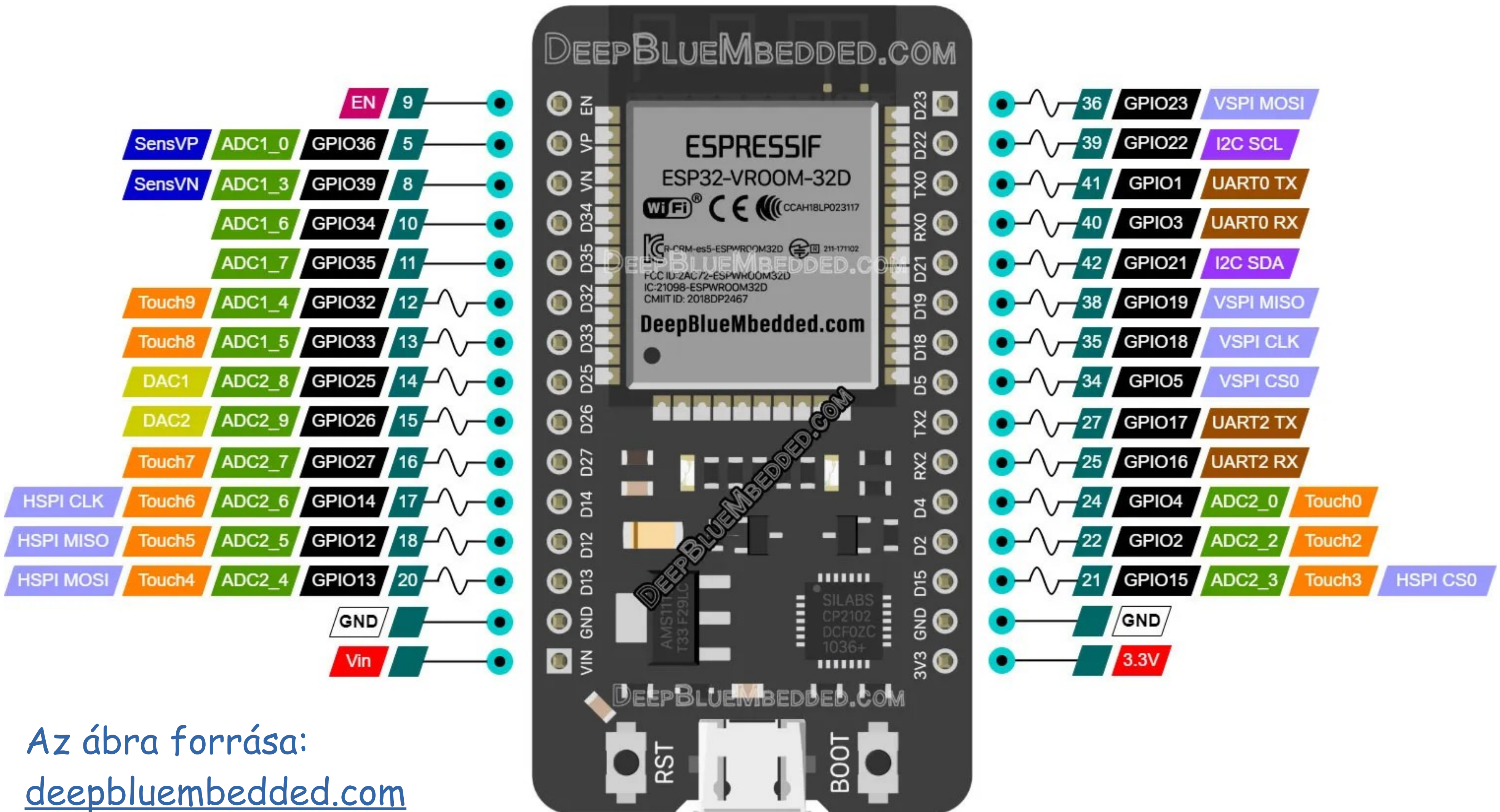
Megjegyzés: a szemünk nem lineáris függvény szerint érzékel!

Hogy működik a PWM?



ESP32 PWM csatornák

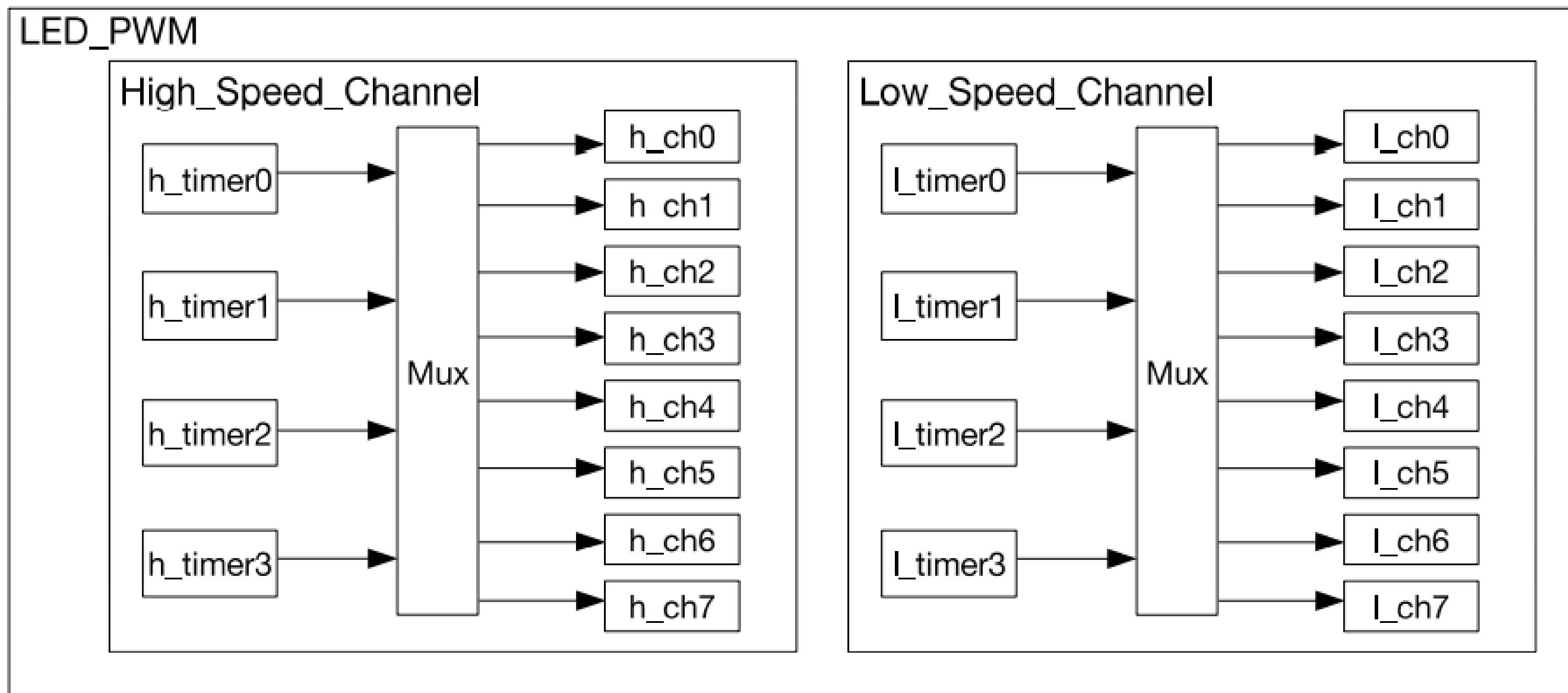
- Az ESP32 16 db LED vezérlő **PWM** csatornával rendelkezik, amelyek bármelyik ***digitális kimenethez*** hozzárendelhetők



Az ábra forrása: deepbluembedded.com

ESP32 PWM csatornák

- Két-két csatorna közös időzítőhöz kapcsolódik, így a frekvenciájuk is közös (csak a kitöltés különbözhet)
- Négy időzítő nagyfrekvenciás órajel bemenetű (80 MHz)
- Négy további időzítő alacsonyfrekvenciás bemenetű (8 MHz)



ESP32 PWM csatornák

- A **PWM** csatornák felbontása változtatható (1-16 bit), s a választott felbontás szabja meg a szabályozhatóság finomságát (8-bites felbontás esetén 0 – 255 közötti lehet a kitöltés értéke, 12-bitesnél 0 – 4095, és így tovább ...
- Megjegyzés: a **PWM** frekvencia és a felbontás szorzata nem lehet nagyobb a bemenő órajel frekvenciájánál
- Az alábbi táblázatban a leggyakrabban használt frekvenciákat és a hozzájuk tartozó maximális felbontást tüntettük fel

LEDC Clock Source	LEDC Output (PWM) Frequency	Highest Resolution
APB_CLK (80 MHz)	1 kHz	1/65536 (16 bit)
APB_CLK (80 MHz)	5 kHz	1/8192 (13 bit)
APB_CLK (80 MHz)	10 kHz	1/4096 (12 bit)
RTC8M_CLK (8 MHz)	1 kHz	1/4096 (12 bit)
RTC8M_CLK (8 MHz)	8 kHz	1/512 (9 bit)
REF_TICK (1 MHz)	10 kHz	1/512 (9 bit)

ESP32 PWM kezelő függvények

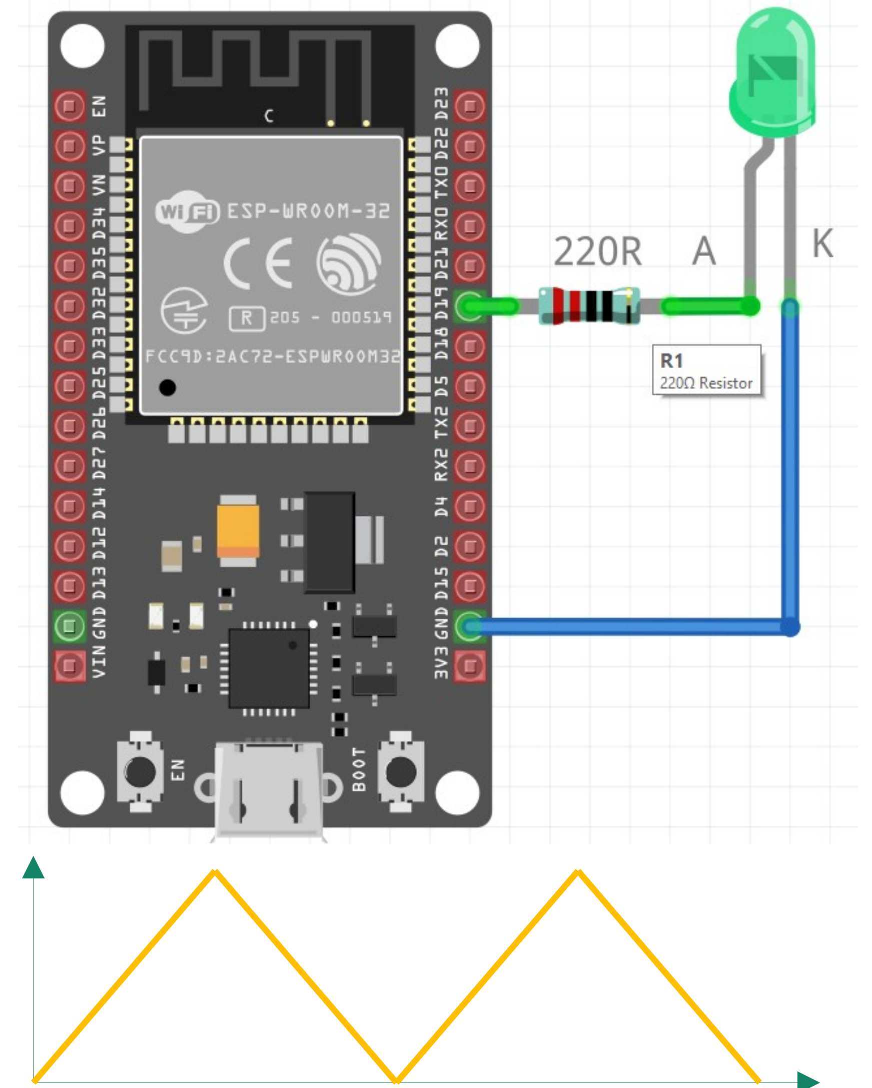
Az ESP32 programkönyvtár az alábbi függvényeket definiálja:

- **ledcSetup**(*channel, frequency, resolution*) – PWM csatorna konfigurálása
- **ledcAttachPin**(*pin, channel*) – PWM csatorna kimenethez rendelése
- **ledcWrite**(*channel, duty*) – PWM kitöltés beállítása
- **ledcRead**(*channel*) – PWM kitöltés visszaolvasása
- **ledcWriteTone**(*channel, frequency*) – adott frekvenciájú hang keltése
- **ledcWriteNote**(*channel, note, octave*) – hangkeltés temperált skálán
- **ledcReadFreq**(*channel*) – visszaolvassa a beállított frekvenciát
- **ledcDetachPin**(*pin*) – a kivezetést leválasztja a PWM csatornákról

ESP32_pwm_ledfade.ino

- Változtassuk folyamatosan a **GPIO19** kimenetre kötött LED fényerejét!

```
#define LED_GPIO   19
#define PWM1_Ch    0
#define PWM1_Res   8
#define PWM1_Freq  1000
int PWM1_DutyCycle = 0;

void setup() {
  ledcAttachPin(LED_GPIO, PWM1_Ch);
  ledcSetup(PWM1_Ch, PWM1_Freq, PWM1_Res);
}

void loop() {
  while(PWM1_DutyCycle < 255) {
    ledcWrite(PWM1_Ch, PWM1_DutyCycle++);
    delay(10);
  }
  while(PWM1_DutyCycle > 0) {
    ledcWrite(PWM1_Ch, PWM1_DutyCycle--);
    delay(10);
  }
}
```



ESP32_pwm_ledfade2.ino

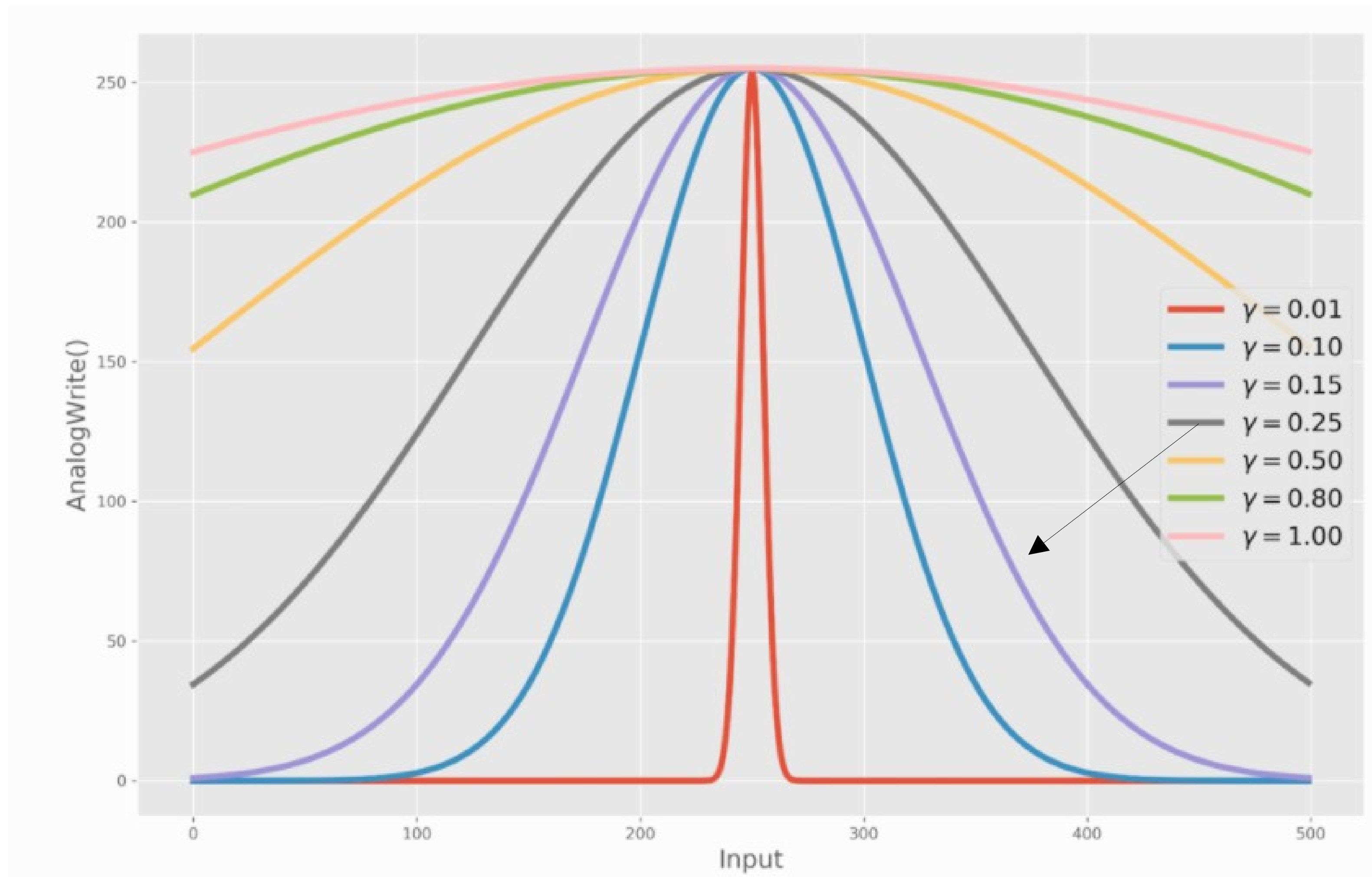
- Szebben „lélegzik” a LED, ha a fel és lefutás menete nem lineáris, hanem Gauss-függvényt követ. A programot innen adaptáltuk: makersportal.com/blog/2020/3/27/simple-breathing-led-in-arduino

```
#define LED_GPIO   19
#define PWM1_Ch    0
#define PWM1_Res   12
#define PWM1_Freq  1000
float smoothness_pts = 500; //larger=slower change in brightness
float gamma_par = 0.14;     // affects the width of peak
float beta_par = 0.5;       // shifts the gaussian to be symmetric

void setup() {
  ledcAttachPin(LED_GPIO, PWM1_Ch);
  ledcSetup(PWM1_Ch, PWM1_Freq, PWM1_Res);
}
void loop() {
  for (int i=0; i<smoothness_pts; i++){
    float pwm_val =
      4095.0*(exp(-(pow(((i/smoothness_pts)-beta_par)/gamma_par,2.0))/2.0));
    ledcWrite(PWM1_Ch, int(pwm_val));
    delay(5);
  }
}
```

$$y = 4095 \cdot e^{-\frac{(x/N - 0.5)^2}{2\gamma^2}}$$

ESP32_pwm_ledfade2.ino

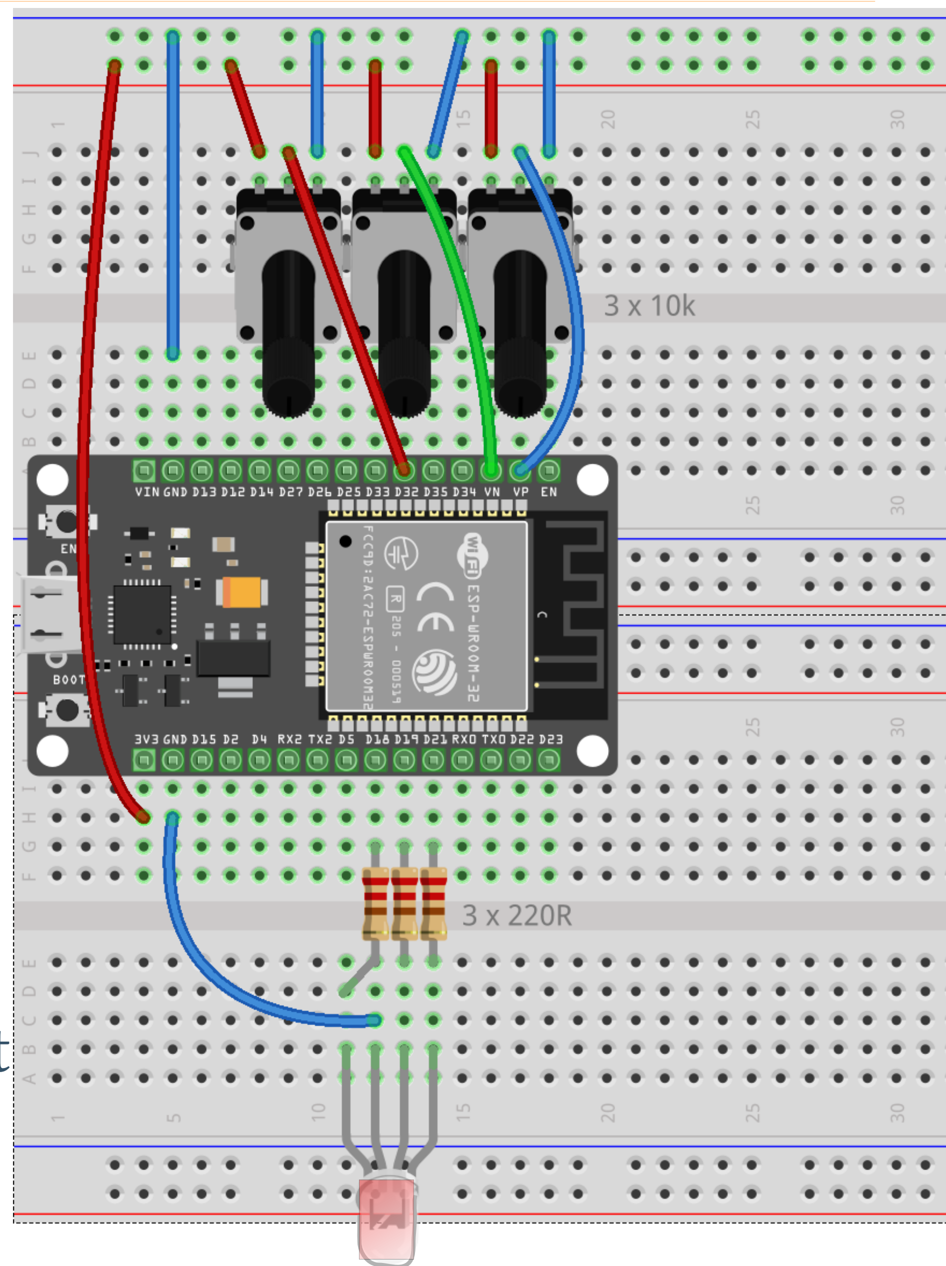
- Az előző programban **gamma_par** 0.14-os értéke a lila görbéhez hasonló lefutást eredményez



Forrás: makersportal.com/blog/2020/3/27/simple-breathing-led-in-arduino

ESP32_pwm_rgbled

- Színkeverést játszunk: három potméterrel külön-külön változtatjuk az RGB LED vörös, zöld és kék színösszetevőit
- A potméterek csúszkái az **A0, A3** és **A4** (VP, VN és D32) analóg bemenetekre csatlakoznak
- Az RGB LED anódjait a **D18, D19, D21** digitális kimenetek vezérlik
- Áramkorlátozásra egy-egy 220 Ω-os ellenállást használunk
- Az ESP32 kártya szélessége miatt két (vagy dupla) próbapanelre lesz szükségünk



ESP32_pwm_rgbled.ino 2/1. oldal

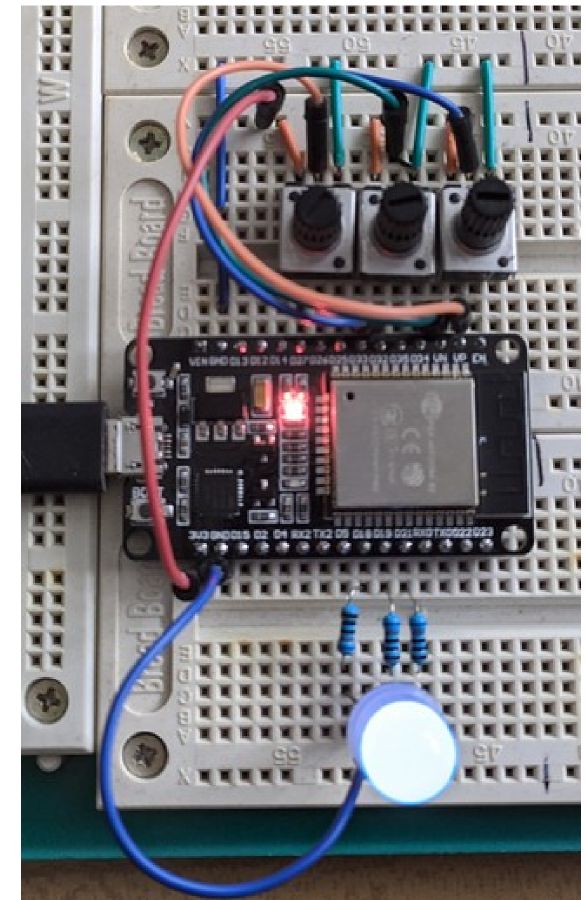
```
// LED anódok vezérlését végző kimenetek
const int redLEDPin   = 18;    // GPIO18
const int greenLEDPin = 19;    // GPIO19
const int blueLEDPin  = 21;    // GPIO21
// Kitöltési tényezők tárolója
uint16_t redDutyCycle;
uint16_t greenDutyCycle;
uint16_t blueDutyCycle;

// PWM csatornák paraméterei
const int redPWMFreq = 5000;   /* 5 KHz */
const int redPWMChannel = 0;
const int redPWMResolution = 12;
const int RED_MAX_DUTY_CYCLE = (int)(pow(2, redPWMResolution) - 1);

const int greenPWMFreq = 5000; /* 5 KHz */
const int greenPWMChannel = 2;
const int greenPWMResolution = 12;
const int GREEN_MAX_DUTY_CYCLE = (int)(pow(2, greenPWMResolution) - 1);

const int bluePWMFreq = 5000;  /*  5 KHz */
const int bluePWMChannel = 4;
const int bluePWMResolution = 12;
const int BLUE_MAX_DUTY_CYCLE = (int)(pow(2, bluePWMResolution) - 1);

const int ADC_RESOLUTION = 4095; /* ADC felbontása: 12-bit */
```

A felhasznált program eredetijének forrása:
https://www.electronicshub.org/esp32-pwm-tutorial/

ESP32_pwm_rgbled.ino 2/2. oldal

A felhasznált program forrása: https://www.electronicshub.org/esp32-pwm-tutorial/
Érdemes megnézni a program eredetijét a fenti linkre kattintva. Abban az egyes csatornákat különböző felbontással és frekvenciával konfigurálták

```
void setup() {
  /* PWM csatornák konfigurálása (frekvencia, felbontás) */
  ledcSetup(redPWMChannel, redPWMFreq, redPWMResolution);
  ledcSetup(greenPWMChannel, greenPWMFreq, greenPWMResolution);
  ledcSetup(bluePWMChannel, bluePWMFreq, bluePWMResolution);
  /* A LED PWM csatornákat GPIO kiemenetekhez rendeljük */
  ledcAttachPin(redLEDPin, redPWMChannel);
  ledcAttachPin(greenLEDPin, greenPWMChannel);
  ledcAttachPin(blueLEDPin, bluePWMChannel);
}

void loop() {
  /* A három analóg bemenet kiolvasása ADC-vel */
  redDutyCycle = analogRead(A0);
  greenDutyCycle = analogRead(A3);
  blueDutyCycle = analogRead(A4);
  /* A PWM kimenetek beállítása a kívánt kitöltéssel */
  ledcWrite(redPWMChannel, redDutyCycle);
  ledcWrite(greenPWMChannel, greenDutyCycle);
  ledcWrite(bluePWMChannel, blueDutyCycle);
  delay(50);
}
```



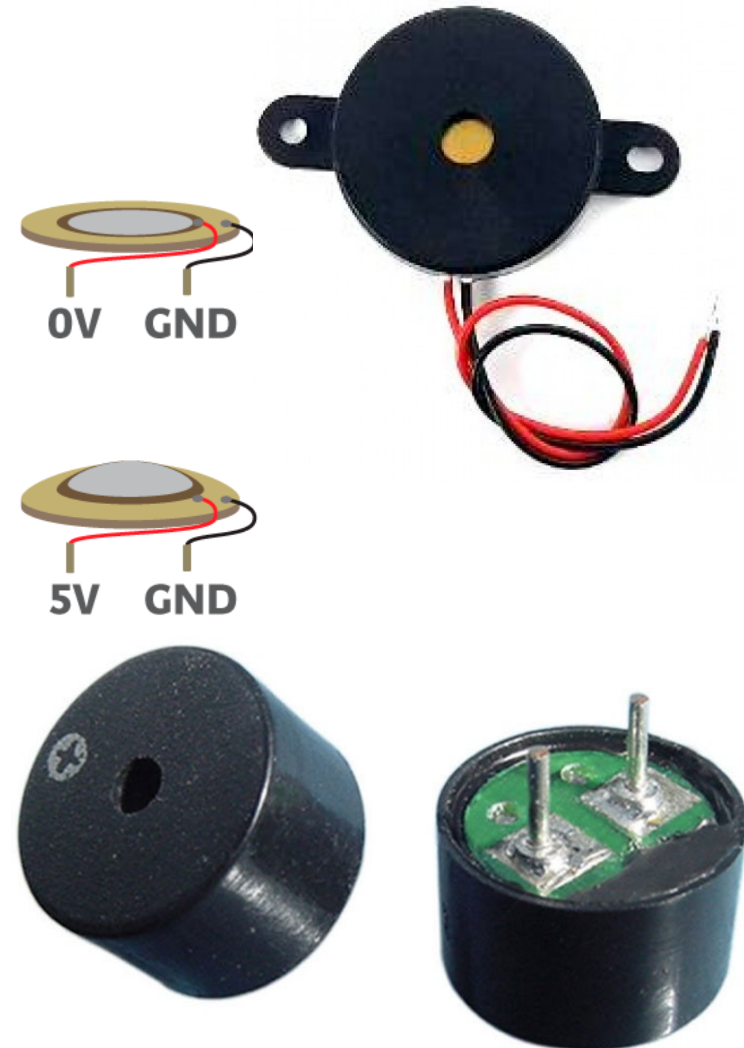
ESP32_pwm_music.ino

- Játsszuk le az alábbi dallamot egy PWM csatorna felhasználásával, a hangot egy passzív piezo csipogó segítségével szólaltassuk meg!

negyed nyolcad fél

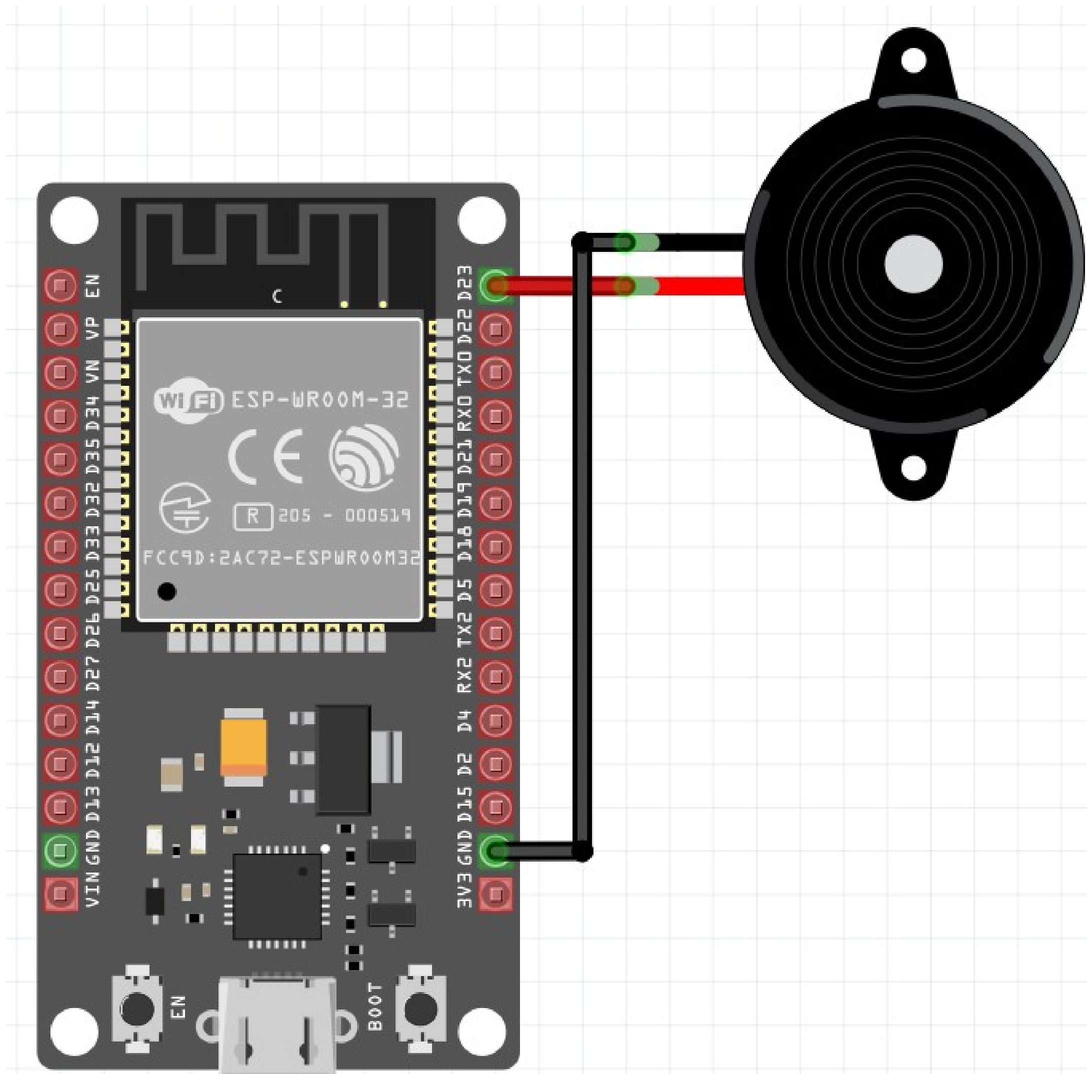
Oran-ges and le-mons, says the bells of St. Cle-men's

Szó/Szótag	Hang	Frekvencia (Hz)	Hossz
oran	E	659	1
ges	C#	554	1
and	E	659	1
le	C#	554	1
mons	A	440	1
says	B	494	½
the	C#	554	½
bells	D	587	1
of	B	494	1
st	E	659	1
clem	C#	554	1
en's	A	440	2



ESP32_pwm_music.ino

- Kössük a csipogó megjelölt (+ jel, vagy piros vezeték) pólusát a **GPIO23** kivezetéshez, a másik pólusát a GND-re!



ESP32_pwm_music.ino

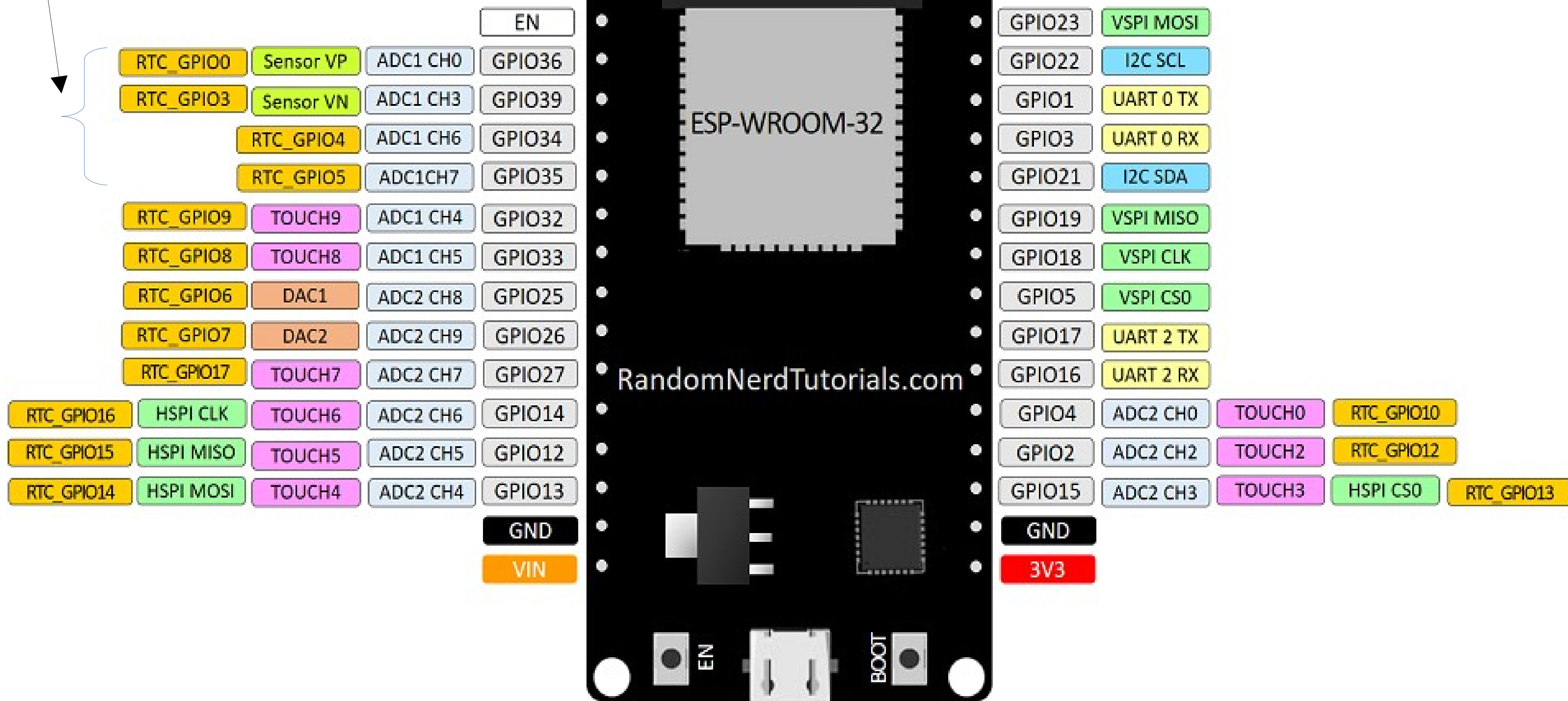
- A programban két tömböt definiálunk és ezekben tároljuk a lejátszandó hangok frekvenciáját és hosszuk reciprokát

```
#define buzzer_pin 23    // A GPIO23 kimenetet használjuk
#define buzzer_chan 0    // 0. PWM csatornát használjuk hangkeltésre

int frequency[] = {659,554,659,554,550,494,554,587,494,659,554,440};
int beat[] = {4, 4, 4, 4, 4, 8, 8, 4, 4, 4, 4, 2};

void setup() {
  // Itt nincs tennivalónk
}

void loop() {
  ledcAttachPin(buzzer_pin, buzzer_chan);
  for (int i = 0; i < 12; i++) {
    ledcWriteTone(buzzer_chan, frequency[i]);  // a frekvencia beállítása
    delay(2000 / beat[i]);                     // tartjuk a hangot
  }
  ledcDetachPin(buzzer_pin);                   // elnémítjuk a hangot
  delay(2000);
}
```

A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 – GPIO11: foglalt (SPI flash)



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

Ellenállás színkódok