(De találtam egy Hosyond cég neve alatt megjelentetett készletet,
a hozzá csatolt leírás ugyanez a dokumentum, ugyanezekkel a hibákkal.
Természetesen a felfedezett hibákat kijavítottam.

A fordító megjegyzése.)

ESP32 alapszintű kezdőkészlet

Fordította: Maczák András

andras@maczak.eu

Az eredeti dokumentáció, a példaprogramok kódjai és a CP2101 illesztőprogram Windowshoz innen letölthetők:

`https://www.dropbox.com/sh/osi3jtv5ncuyby2/AADkuo03wGXK8JiwwbV3zPc6a?dl=0`

Az általam lefordított megjegyzéseket tartalmazó és hibajavított programkódok pedig:

`https://maczak.eu/arduino/kitts/esp32projektek.zip`

Tartalomjegyzék

A csomag tartalma

ESP32 Fejlesztő kártya 1db

0,96 hüvelykes OLED 1db

830 pontos bekötő kártya 1db

Fényérzékelő ellenállás modul 1db

DHT11 hőmérséklet és páratartalom modul 1db

HC-SR501 PIR mozgásérzékelő 1db

Mikro USB kábel 1db

Ellenállás (220/1k/10k) 30db

Potenciométer (10k) 1db

Kapcsoló gomb 6db

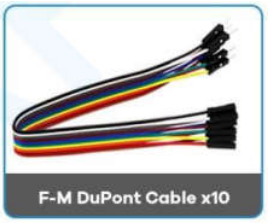
Passzív berregő 1db

5V 2-csatornás relé modul 1db

Akadálykerülő modul 1db

Aktív berregő 1db

RGB LED 2db



F-M Dupont kábel 10db

F-F Dupont kábel 10db

M-M Dupont kábel 10db

LED piros 5db

LED sárga 5db

LED zöld 5db

ESP32 Bevezető

Új ESP32? Kezd itt! Az ESP32 az Espressif által kifejlesztett, alacsony költségű és alacsony fogyasztású System on a Chip (SoC) mikrokontrollerek sorozata, amelyek Wi-Fi és Bluetooth vezeték nélküli képességeket és kétmagos processzort tartalmaznak. Ha ismered az ESP8266-ot, az ESP32 az utódja, rengeteg új funkcióval.

ESP32 specifikációk

Ha kicsit technikásabbra és specifikusabbra szeretnél szert tenni, tekintsd meg az ESP32 alábbi részletes specifikációit (forrás: http://esp32.net/) – további részletekért tekintsd meg az adatlapot:

`https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`

- Vezeték nélküli kapcsolat WiFi: 150,0 Mbps adatsebesség HT40-nel
- Bluetooth: BLE (Bluetooth Low Energy) és Bluetooth Classic
- Processzor: Tensilica Xtensa kétmagos 32 bites LX6 mikroprocesszor, 160 vagy 240 MHz-en
- Memória:
    - ROM: 448 KB (a rendszerindításhoz és az alapvető funkciókhoz)
    - SRAM: 520 KB (adatokhoz és utasításokhoz)
    - RTC gyors SRAM: 8 KB (adattároláshoz és a fő CPU-hoz az RTC rendszerindítás során mélyalvó módból)
    - RTC lassú SRAM: 8 KB (a társprocesszor eléréséhez mély alvó módban)

    - eFuse: 1 Kbit (ebből 256 bitet használ a rendszer (MAC-cím és chip-konfiguráció), a fennmaradó 768 bit pedig az ügyfélalkalmazások számára van fenntartva, beleértve a Flash-titkosítást és a chip-azonosítót)

- Beágyazott flash: belsőleg csatlakoztatott flash IO16, IO17, SD_CMD, SD_CLK, SD_DATA_0 és SD_DATA_1 az ESP32-D2WD és ESP32-PICO-D4 rendszereken.
    - 0 MiB (ESP32-D0WDQ6, ESP32-D0WD és ESP32-S0WD chipek)
    - 2 MiB (ESP32-D2WD chip)
    - 4 MiB (ESP32-PICO-D4 SiP modul)
- Alacsony fogyasztás: biztosítja, hogy továbbra is használhatja az ADC-konverziókat, például mélyalvás közben.
- Periféria bemenet/kimenet:
    - periféria interfész DMA-val, amely kapacitív érintést is tartalmaz
    - ADC-k (analóg-digitális konverter)
    - DAC-ok (digitális-analóg konverter)
    - I²C (Belsőleg Integrált áramkör)
    - UART (univerzális aszinkron vevő/adó)
    - SPI (soros periféria interfész)
    - I²S (Integrált Interchip hang)
    - RMII (Csökkentett Média-független Interfész)
    - PWM (impulzusszélesség-moduláció)
- Biztonság: hardveres gyorsítók AES és SSL/TLS számára

ESP32 fejlesztő kártyák

Az ESP32 a csupasz ESP32 chipre utal. Az „ESP32” kifejezést azonban az ESP32 fejlesztőkártyákra is használják. Az ESP32 csupasz chipek használata nem egyszerű sem praktikus, különösen tanulás, tesztelés és prototípuskészítés során. Legtöbbször az ESP32 fejlesztői kártyát szeretnéd használni.

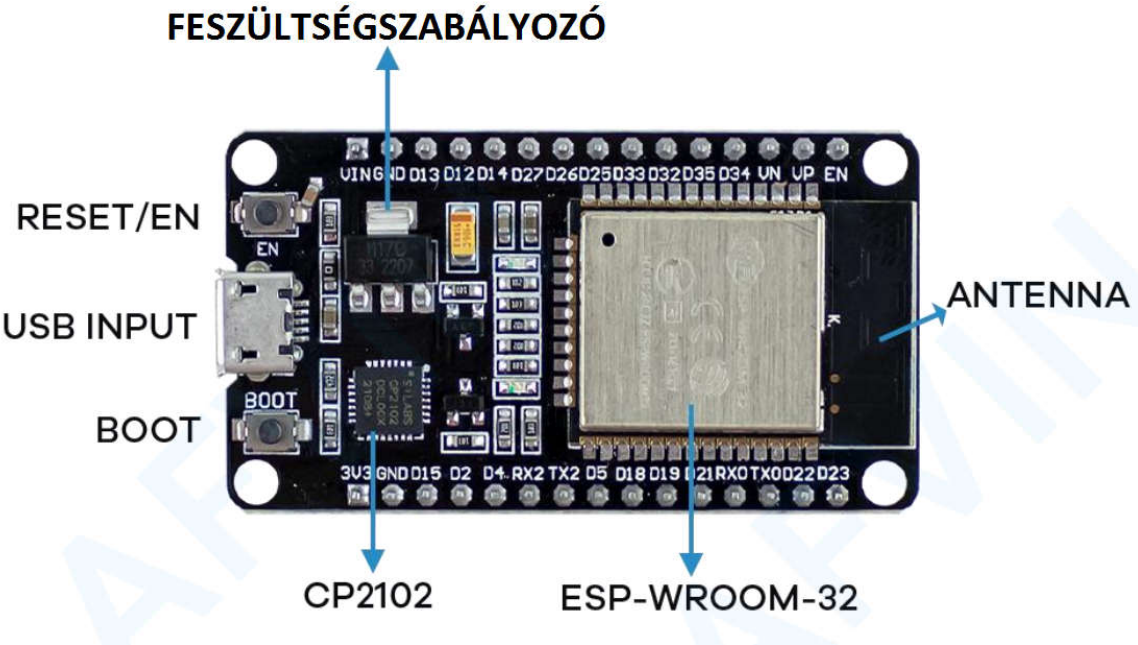
Referenciaként az ESP32 DEVKIT V1 kártyát fogjuk használni. Az alábbi képen az ESP32 DEVKIT V1 kártya látható, 30 GPIO tűvel.

Műszaki adatok – ESP32 DEVKIT V1

Az alábbi táblázat összefoglalja az ESP32 DEVKIT V1 DOIT kártya jellemzőit és specifikációit:

Magok száma	2 (kétmagos)
Wi-Fi	2,4 GHz 150 Mbit/s-ig
Bluetooth	BLE (Bluetooth kis energiájú) és a régi Bluetooth
Felépítés	32 bites
Órajel frekvencia	240 MHz-ig
RAM	512 KB
Tűk	30 (modelltől függően)
Perifériák	Kapacitív érintés, ADC (analóg-digitális átalakító), DAC (digitális-analóg átalakító), I2C (Inter-Integrated Circuit), UART (univerzális aszinkron vevő/adó), CAN 2.0 (Controller Area Network), SPI (soros periféria interfész) , I2S (Integrated Inter-IC Sound), RMII (Reduced Media-Independent Interface), PWM (impulzus-szélesség-moduláció) és még sok más.
Beépített gombok	RESET és BOOT gombok

Beépített LED-ek	beépített kék LED GPIO2-re csatlakoztatva; beépített piros LED, amely jelzi, hogy a tábla áram alatt van
USB az UART-hoz	CP2102

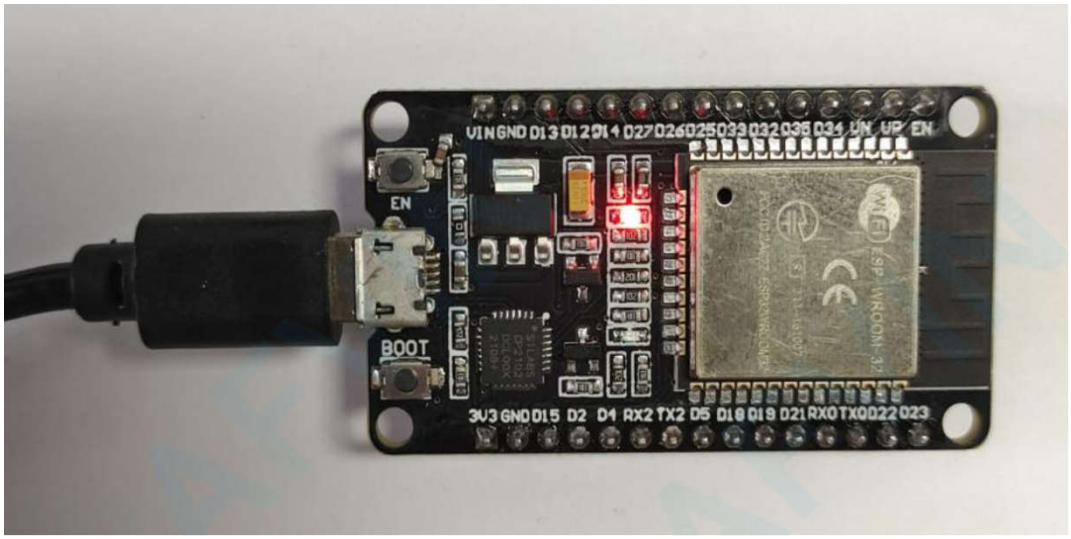


Egy microUSB interfésszel rendelkezik, amellyel a kártyát a számítógéphez csatlakoztathatod a kód feltöltéséhez vagy az áramellátáshoz.

A CP2102 chipet (USB-tól UART-ig) használja a számítógéppel való kommunikációhoz egy COM-porton keresztül, soros interfészen keresztül. Egy másik népszerű chip a CH340. Ellenőrizd, hogy milyen az USB-UART chip átalakító a kártyán, mert telepítened kell a szükséges illesztőprogramokat, hogy a számítógéped kommunikálni tudjon az alaplappal (erről az útmutató későbbi részében olvashatsz bővebben).

Ezen az alaplapon található még egy RESET gomb (lehet, hogy EN felirattal is) az alaplap újraindításához, valamint egy BOOT gomb, amely flash üzemmódba helyezi a kártyát (a kód fogadására használható). Vedd figyelembe, hogy egyes kártyákon előfordulhat, hogy nincs BOOT gomb.

Egy beépített kék LED-del is rendelkezik, amely belsőleg csatlakozik a GPIO 2-höz. Ez a LED hasznos a hibakereséshez, hogy valamilyen vizuális fizikai kimenetet adjon. Van egy piros LED is, amely akkor világít, amikor árammal látod el a táblát.



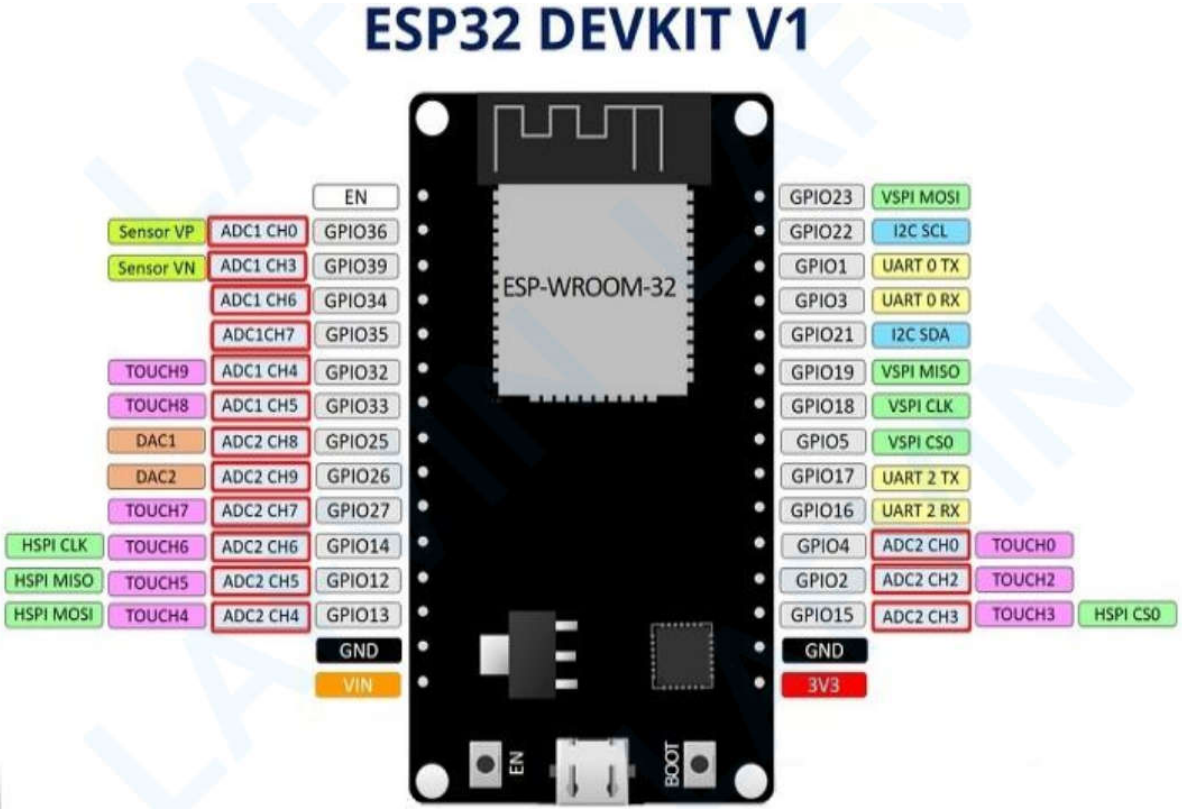
Az ESP32 tűkiosztása

Az ESP32 perifériák a következőket tartalmazzák:

- 18 Analóg-digitális konverter (ADC) csatorna
- 3 SPI interfész
- 3 UART interfész
- 2 I2C interfész
- 16 PWM kimeneti csatorna
- 2 digitális-analóg konverter (DAC)
- 2 I2S interfész
- 10 kapacitív érzékelő GPIO

Az ADC (analóg-digitális konverter) és DAC (digitális-analóg konverter) funkciók meghatározott statikus érintkezőkhöz vannak hozzárendelve. Eldöntheti azonban, hogy mely érintkezők UART, I2C, SPI, PWM stb. – csak hozzá kell rendelni őket a kódban. Ez az ESP32 chip multiplexelési funkciója miatt lehetséges.

Bár a tűk tulajdonságait meghatározhatod a szoftverben, a következő ábrán látható, hogy alapértelmezés szerint hogyan vannak hozzárendelve a tűk.



Ezenkívül vannak olyan tűk, amelyek speciális jellemzőkkel rendelkeznek, amelyek alkalmassá teszik őket egy adott projekthez. A következő táblázat bemutatja, hogy mely érintkezőket a legjobb bemenetként, kimenetként használni, és melyek azok, amelyekkel óvatosan kell bánni.

A zölddel kiemelt tűk felhasználhatóak. A sárgával kiemeltek használhatók, de oda kell figyelni, mert főként indításkor előfordulhat váratlan viselkedésük. A pirossal kiemelt tűket nem ajánljuk bemenetként vagy kimenetként használni.

GPIO	Input	Output	Megjegyzések
0	FELHÚZOTT	OK	PWM jelet ad ki rendszerindításkor, LOW-nak kell lennie a flash módba lépéshez
1	TX tű	OK	Hibakeresési kimenet indításkor
2	OK	OK	A fedélzeti LED-hez van csatlakoztatva, flash üzemmódba lépéshez lebegésben vagy LOW-ban kell hagyni
3	OK	RX tű	HIGH indításkor
4	OK	OK	
5	OK	OK	PWM jelet ad ki a rendszerindításkor, strapping* tű
12	OK	OK	a BOOT meghiúsul, ha magasra húzzák, strapping tű
13	OK	OK	
14	OK	OK	PWM jelet ad ki rendszerindításkor
15	OK	OK	PWM jelet ad ki a rendszerindításkor, strapping tű
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	

22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		Csak input
35	OK		Csak input
36	OK		Csak input
39	OK		Csak input

*A srapping szónak nem találtam magyar megfelelőjét. Tulajdonképpen azt jelenti, hogy ezeknek a tűknek rendszerindításkor különleges szerepük van, a leírást lásd kicsit később...

Folytasd az olvasást az ESP32 GPIO-k és funkcióinak részletesebb és mélyebb elemzéséhez.

Csak input tűk

A 34-39 közötti GPIO-k GPI-k – csak bemeneti tűk. Ezeknek a tűknek nincs belső felhúzó vagy lehúzó ellenállása. Nem használhatók kimenetként, ezért ezeket a lábakat csak bemenetként használd:

- GPIO 34
- GPIO 35
- GPIO 36
- GPIO 39

Az ESP-WROOM-32-be integrált SPI flash

A GPIO 6–GPIO 11 elérhető néhány ESP32 fejlesztői kártyán. Ezek a tűk azonban az ESP-WROOM-32 chip integrált SPI flash-hez csatlakoznak, és nem ajánlottak más célra. Tehát ne használja ezeket a tűket a projektjeiben:

- GPIO 6 (SCK/CLK)
- GPIO 7 (SDO/SD0)
- GPIO 8 (SDI/SD1)
- GPIO 9 (SHD/SD2)
- GPIO 10 (SWO/SD3)
- GPIO 11 (CSC/CMD)

Kapacitív érintős GPIO-k

Az ESP32 10 belső kapacitív érintésérzékelővel rendelkezik. Ezek érzékelik a változásokat bármiben, ami elektromos töltést tartalmaz, például az emberi bőrben. Így észlelni tudják a GPIO-k ujjal történő megérintésekor előidézett eltéréseket. Ezek a tűk könnyen integrálhatók kapacitív párnákba, és helyettesíthetik a mechanikus gombokat. A kapacitív érintőtűkkel az ESP32 felébreszthető a mély alvásból.

Ezek a belső érintésérzékelők ezekhez a GPIO-khoz csatlakoznak:

- T0 (GPIO 4)
- T1 (GPIO 0)
- T2 (GPIO 2)
- T3 (GPIO 15)
- T4 (GPIO 13)
- T5 (GPIO 12)
- T6 (GPIO 14)
- T7 (GPIO 27)
- T8 (GPIO 33)
- T9 (GPIO 32)

Analóg-digitális átalakító (ADC)

Az ESP32 18 x 12 bites ADC bemeneti csatornákkal rendelkezik (míg az ESP8266 csak 1 x 10 bites ADC-vel rendelkezik). Ezek a GPIO-k, amelyek ADC-ként és megfelelő csatornaként használhatók:

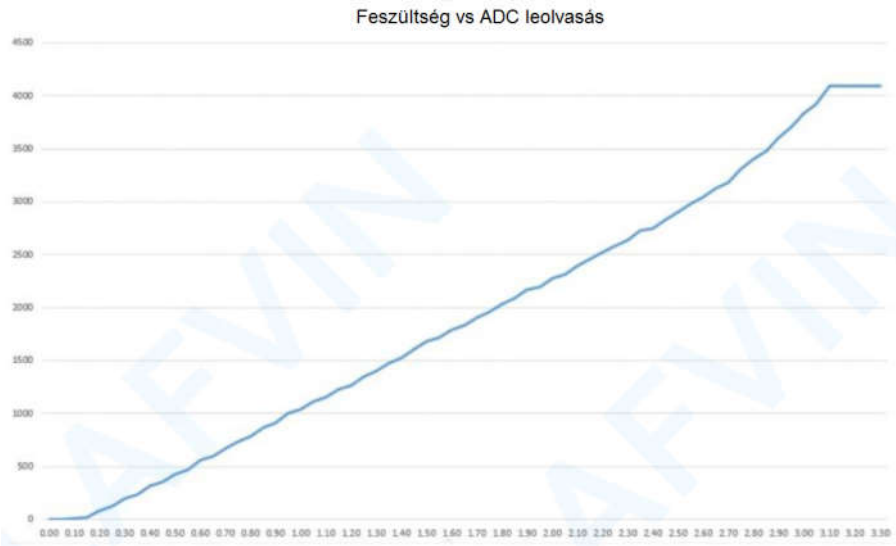
- ADC1_CH0 (GPIO 36)
- ADC1_CH1 (GPIO 37)
- ADC1_CH2 (GPIO 38)
- ADC1_CH3 (GPIO 39)
- ADC1_CH4 (GPIO 32)
- ADC1_CH5 (GPIO 33)
- ADC1_CH6 (GPIO 34)
- ADC1_CH7 (GPIO 35)
- ADC2_CH0 (GPIO 4)

- ADC2_CH1 (GPIO 0)
- ADC2_CH2 (GPIO 2)
- ADC2_CH3 (GPIO 15)
- ADC2_CH4 (GPIO 13)
- ADC2_CH5 (GPIO 12)
- ADC2_CH6 (GPIO 14)
- ADC2_CH7 (GPIO 27)
- ADC2_CH8 (GPIO 25)
- ADC2_CH9 (GPIO 26)

Megjegyzés: Az ADC2 tűk nem használhatók Wi-Fi használata esetén. Tehát, ha Wi-Fi-t használsz, és problémáid vannak az ADC2 GPIO értékének lekérésével, fontold meg az ADC1 GPIO használatát. Ennek meg kell oldania a problémádat.

Az ADC bemeneti csatornák felbontása 12 bites. Ez azt jelenti, hogy 0 és 4095 közötti analóg értékeket kaphatsz, amelyekben a 0 a 0 V-nak, és a 4095 a 3,3 V-nak felel meg. A kódon és az ADC tartományon is beállíthatod a csatornáidnak a felbontását.

Az ESP32 ADC érintkezőknek nem lineáris a viselkedésük. Valószínűleg nem fogsz tudni különbséget tenni 0 és 0,1 V, illetve 3,2 és 3,3 V között. Ezt szem előtt kell tartanod az ADC érintkezők használatakor. A következő ábrán láthatóhoz hasonló viselkedést fogsz kapni.



Digitális-analóg konverter (DAC)

Az ESP32-n 2 x 8 bites DAC csatorna található a digitális jelek analóg feszültségű jelkimenetekké alakításához. Ezek a DAC csatornák:

- DAC 1 (GPIO 25)

- DAC 2 (GPIO 26)

RTC GPIO-k

Az ESP32 rendelkezik RTC GPIO támogatással. Az RTC alacsony fogyasztású alrendszeréhez irányított GPIO-k akkor használhatók, ha az ESP32 mélyalvásban van. Ezekkel az RTC GPIO-kkal fel lehet ébreszteni az ESP32-t mélyalvásból, amikor az Ultra Low Power (ULP) társprocesszor fut. A következő GPIO-k külső ébresztési forrásként használhatók.

- RTC_GPIO0 (GPIO36)
- RTC_GPIO3 (GPIO39)
- RTC_GPIO4 (GPIO34)
- RTC_GPIO5 (GPIO35)
- RTC_GPIO6 (GPIO25)
- RTC_GPIO7 (GPIO26)
- RTC_GPIO8 (GPIO33)
- RTC_GPIO9 (GPIO32)
- RTC_GPIO10 (GPIO4)
- RTC_GPIO11 (GPIO0)
- RTC_GPIO12 (GPIO2)
- RTC_GPIO13 (GPIO15)
- RTC_GPIO14 (GPIO13)
- RTC_GPIO15 (GPIO12)
- RTC_GPIO16 (GPIO14)
- RTC_GPIO17 (GPIO27)

PWM

Az ESP32 LED PWM vezérlő 16 független csatornával rendelkezik, amelyek konfigurálhatók különböző tulajdonságú PWM jelek generálására. Minden kimenet, amely kimenetként működhet, használható PWM lábként (a 34-39 GPIO-k nem tudnak PWM jelet generálni).

A PWM jel beállításához a következő paramétereket kell megadnod a kódban:

- A jel frekvenciája;
- Üzemi ciklus;

- PWM csatorna;
- GPIO, ahol a jelet ki akarod adni.

I2C

Az ESP32 két I2C csatornával rendelkezik, és bármelyik érintkező beállítható SDA vagy SCL-ként. Ha az ESP32-t Arduino IDE-vel használja, az alapértelmezett I2C érintkezők a következők:

GPIO 21 (SDA)

GPIO 22 (SCL)

Ha más tűket szeretnél használni a vezetékkönyvtár használatakor, csak hívnod kell:

```
Wire.begin(SDA, SCL);
```

SPI

Alapértelmezés szerint az SPI tű-leképezése a következő:

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

Megszakítások

Minden GPIO megszakításként konfigurálható.

Strapping tűk

Az ESP32 chipnek minden indításkor vagy újraindításakor specifikus konfigurációs paraméterekre van szüksége, amelyek meghatározzák a rendszerindítási módot, a feszültségbeállításokat, ROM-üzenetek esetleges nyomtatását, JTAG jelforrás beállítását, stb.

Egyes érintkezők egyedi funkcióval rendelkeznek az ESP32 indításakor. Ezeket Strapping tűknek nevezik. A Strapping tűk magas, alacsony vagy lebegő állapotok kombinációi határozzák meg az ESP32 rendszerindítási viselkedését.

Az ESP32 chip a következő strapping tűkkel rendelkezik:

- GPIO 0 (LOW-nak kell lennie a rendszerindítási módba lépéshez)
- GPIO 2 (rendszerindításkor lebegőnek vagy LOW-nak kell lennie)
- GPIO 4
- GPIO 5 (HIGH-nak kell lennie a rendszerindítás során)
- GPIO 12 (LOW-nak kell lennie a rendszerindítás során)
- GPIO 15 (HIGH-nak kell lennie a rendszerindítás során)

Ezekkel az ESP32 bootloader vagy flash módba kerül. A legtöbb beépített USB/soros fejlesztői kártyán nem kell aggódnod ezen érintkezők állapota miatt. A tábla megfelelő állapotba hozza a tűket flash vagy rendszerindítási módhoz. További információ az ESP32 rendszerindítási mód kiválasztásáról itt található.

Ha azonban perifériák vannak csatlakoztatva ezekhez a érintkezőkhöz, gondot okozhat az új kód feltöltése, az ESP32 új firmware-lel való flash-elése vagy az alaplap visszaállítása. Ha néhány periféria van csatlakoztatva a rögzítőtüskéihez, és problémába ütközik a kód feltöltése vagy az ESP32 flash, ennek oka lehet, hogy ezek a perifériák megakadályozzák, hogy az ESP32 a megfelelő módba lépjen. Olvasd el a Boot Mode Selection dokumentációját, amely elvezet téged a helyes irányba. Az alaphelyzetbe állítás, flash vagy rendszerindítás után ezek a tűk a várt módon működnek.

HIGH tűk a rendszerindításkor

Egyes GPIO-k állapotukat HIGH-ra változtatják, vagy PWM-jeleket adnak ki rendszerindításkor vagy alaphelyzetbe állításkor. Ez azt jelenti, hogy ha kimeneteid vannak csatlakoztatva ezekhez a GPIO-khoz, akkor váratlan eredményeket kaphatsz az ESP32 alaphelyzetbe állításakor vagy elindulásakor.

- GPIO 1
- GPIO 3
- GPIO 5
- GPIO 6-tól GOIO 11-ig (csatlakozik az ESP32 integrált SPI flash memóriához – nem javasolt a használata).
- GPIO 14
- GPIO 15

Engedélyezés (EN)

Az engedélyezés (EN) a 3,3 V-os szabályozó engedélyező érintkezője. Fel van húzva, ezért csatlakoztassa a földhöz a 3,3 V-os szabályozó letiltásához. Ez azt jelenti, hogy ezt a nyomógombhoz csatlakoztatott tűt használhatja például az ESP32 újraindításához.

GPIO áramfelvétel

A GPIO-nként felvett abszolút maximális áram 40 mA az ESP32 adatlap „Ajánlott működési feltételek” szakasza szerint.

ESP32 beépített Hall effektus érzékelő

Az ESP32 beépített Hall effektus érzékelővel is rendelkezik, amely érzékeli a környezetében lévő mágneses tér változásait.

ESP32 Arduino IDE

Van egy kiegészítő az Arduino IDE-hez, amely lehetővé teszi az ESP32 programozását az Arduino IDE és annak programozási nyelve segítségével. Ebben az oktatóanyagban megmutatjuk, hogyan telepítheted az ESP32 kártyát az Arduino IDE-ben, függetlenül attól, hogy Windows, Mac OS X vagy Linux operációs rendszert használsz.

Előfeltételek: Arduino IDE telepítve

A telepítési eljárás megkezdése előtt telepítened kell az Arduino IDE-t a számítógépedre. Az Arduino IDE két verziója telepíthető: az 1-es és a 2-es verzió.

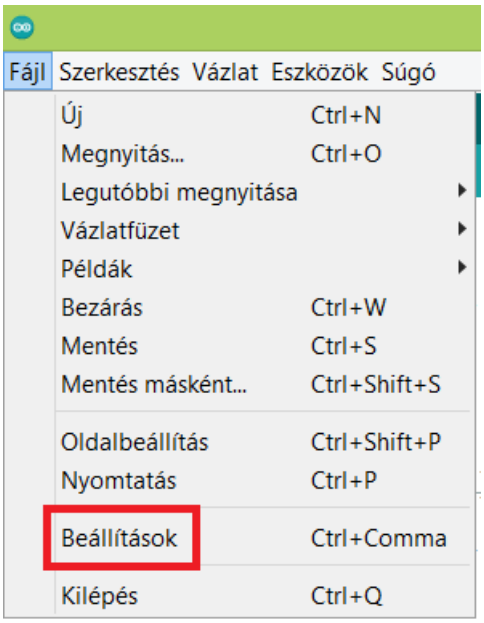
Az Arduino IDE letöltéséhez és telepítéséhez kattints a következő linkre:
https://www.arduino.cc/en/software

Melyik Arduino IDE verziót ajánljuk? Jelenleg az ESP32-hez van néhány beépülő modul (például az SPIFFS fájlrendszer-feltöltő beépülő modul), amelyeket még nem támogat az Arduino 2. Tehát, ha a jövőben használni kívánod az SPIFFS bővítményt, javasoljuk, hogy telepítsd a régebbi 1.8.X-as verziót. Csak le kell görgetned az Arduino szoftver oldalán, hogy megtaláld.

Az ESP32 bővítmény telepítése az Arduino IDE-ben

Az ESP32 kártya Arduino IDE-be való telepítéséhez kövesd az alábbi utasításokat:

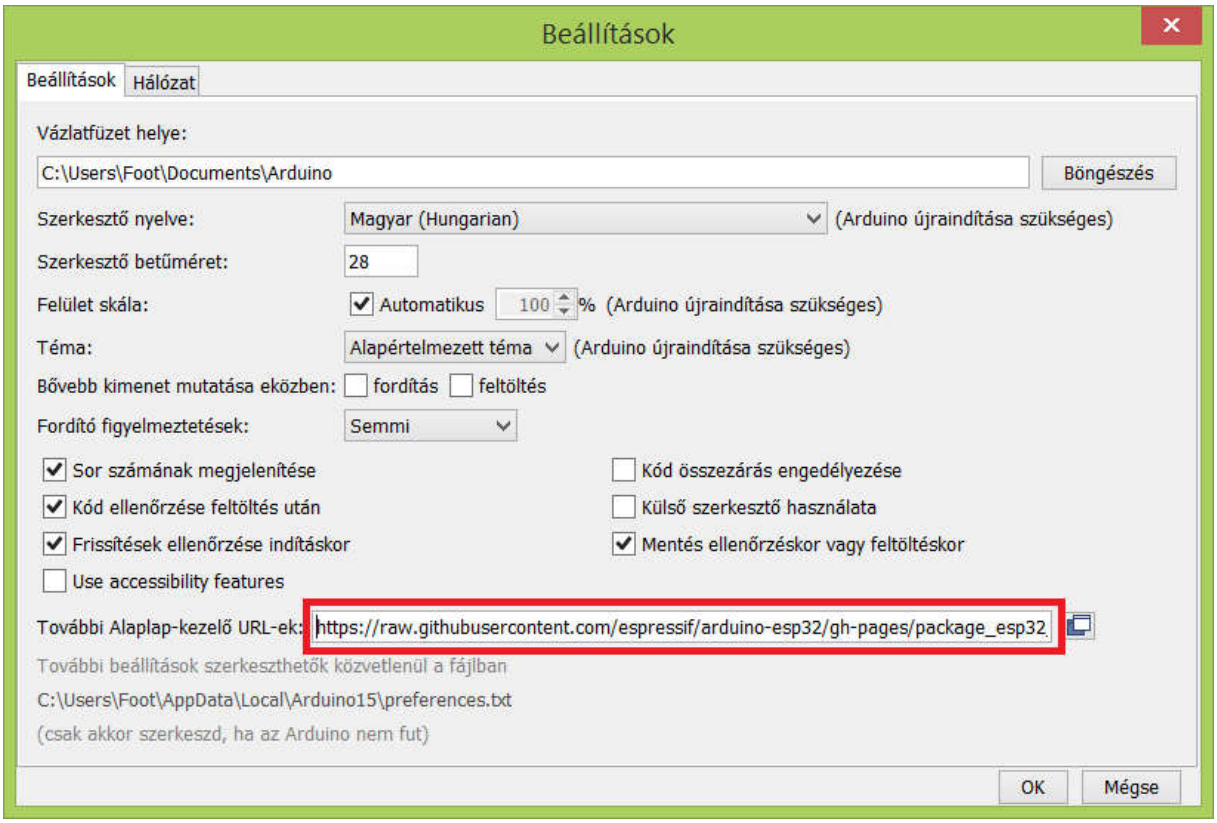
1. Az Arduino IDE-ben lépj a **Fájl> Beállítások** (**File> Preferences**) menüpontra



2. Írd be a következőket a „További Alaplap-kezelő URL-ek:” mezőbe:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
```

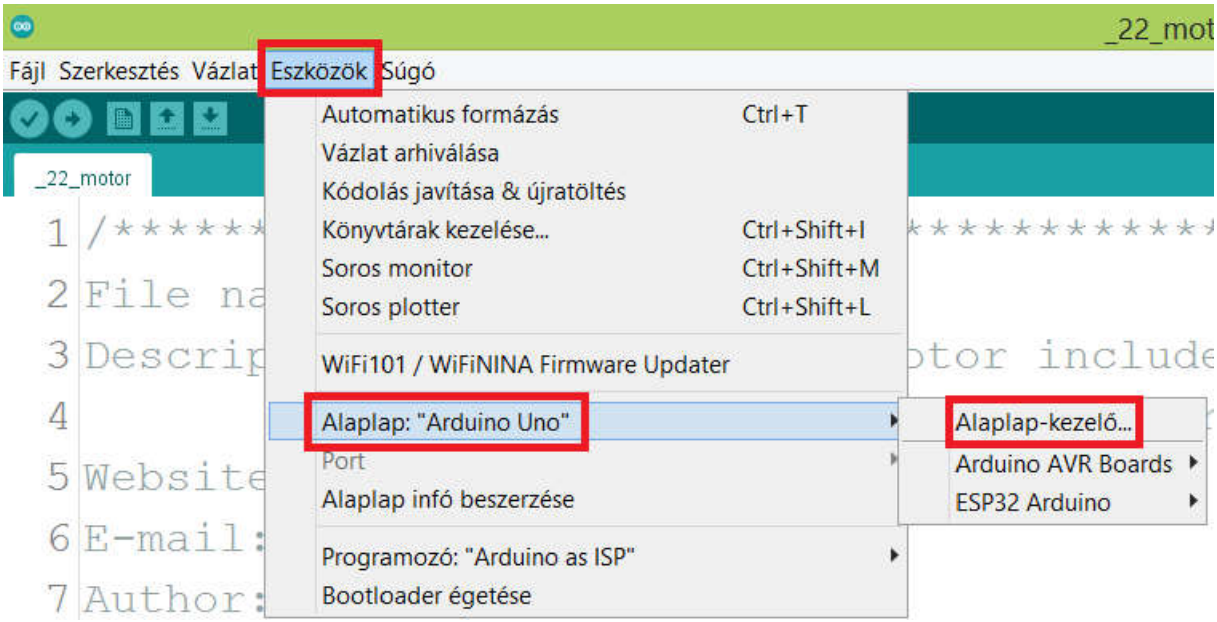
Ezután kattints az „OK” gombra:



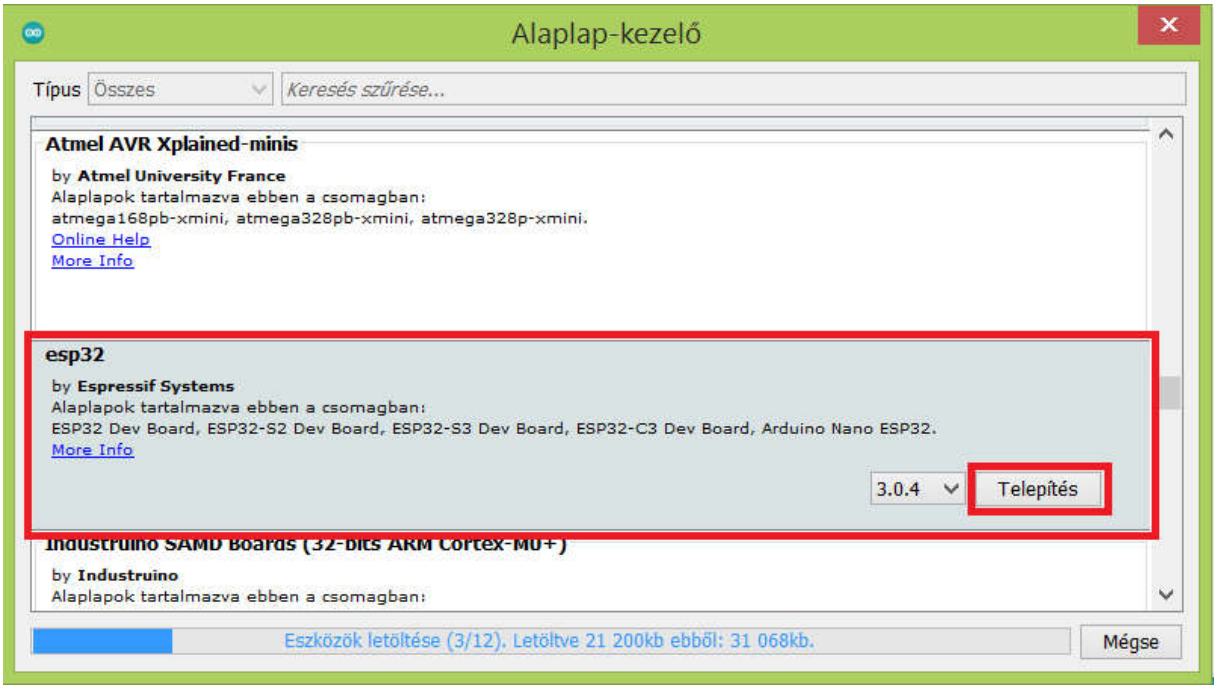
Megjegyzés: ha már rendelkezel az ESP8266 kártya URL-címével, az URL-eket vesszővel választhatod el az alábbiak szerint:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-
pages/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

3. Nyisd meg az Alaplap-kezelőt (Boards Manager...). Lépjen az **Eszközök > Alaplap > Alaplap-kezelő...** (**Tools > Board > Boards Manager…**) menübe



4. Keresd meg az **ESP32** kifejezést, és nyomd meg a telepítés gombot az „**ESP32 by Espressif Systems**” számára:

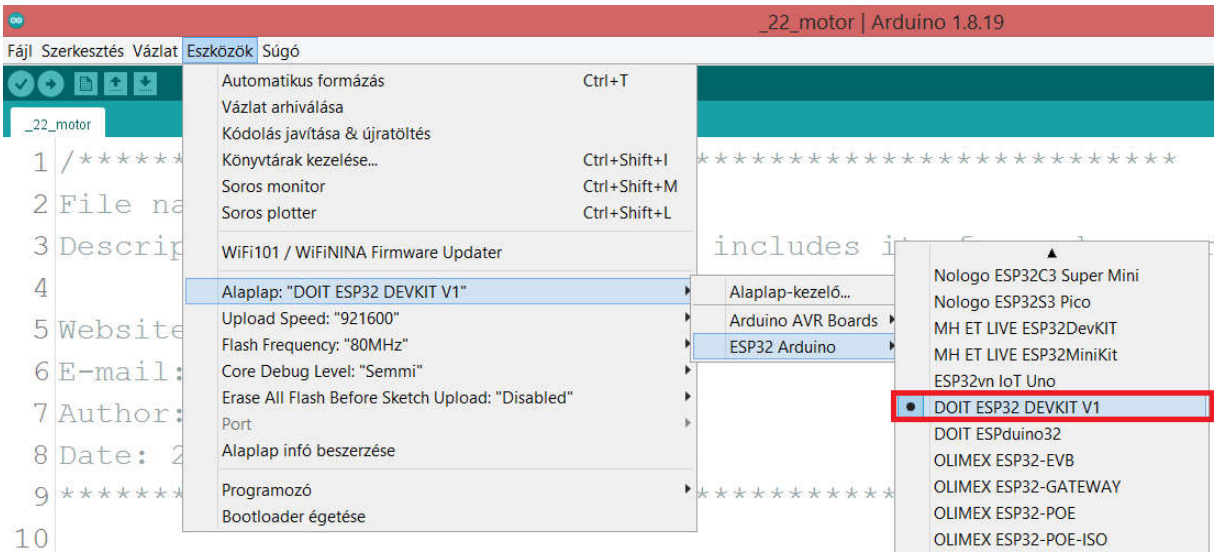


5. Néhány másodperc múlva települnie kell.

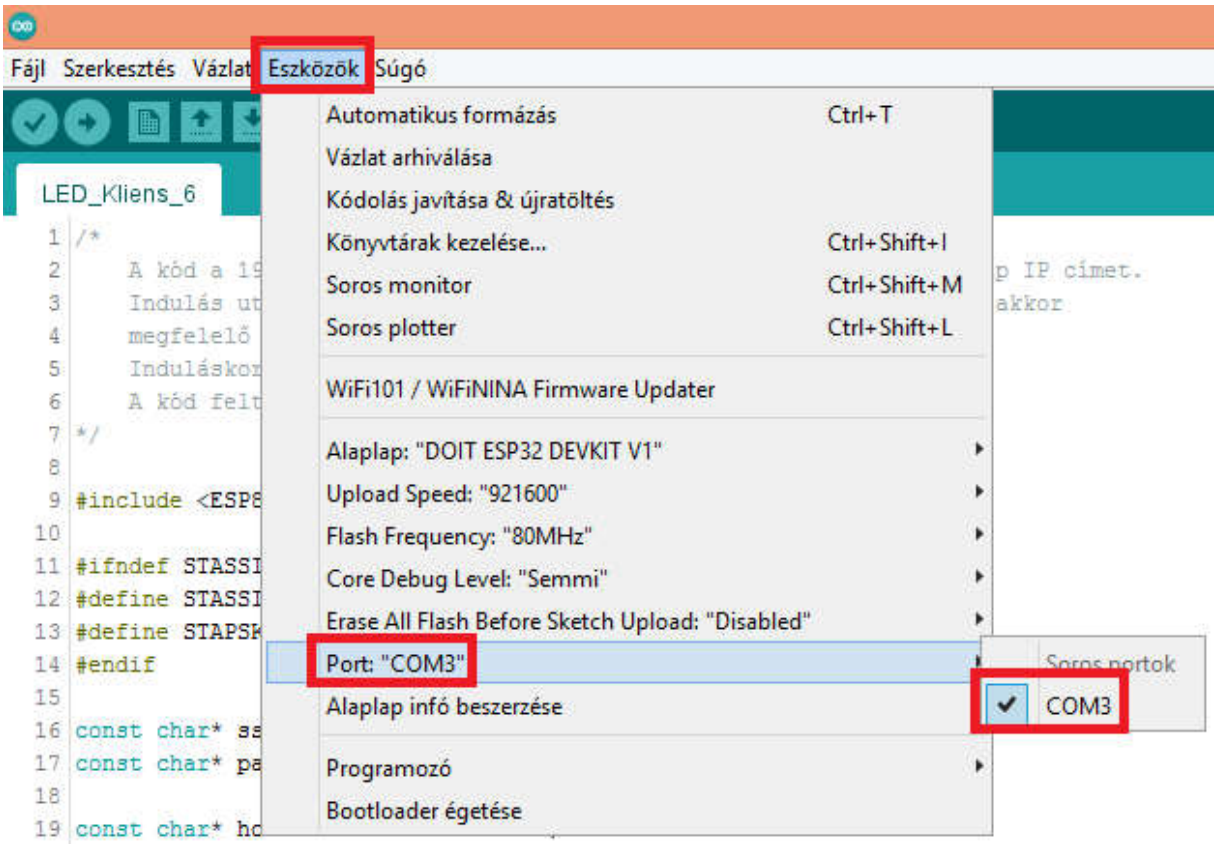
Töltsd fel a tesztkódot

Csatlakoztasd az ESP32 kártyát a számítógéphez. Ha az Arduino IDE meg van nyitva, kövesd az alábbi lépéseket:

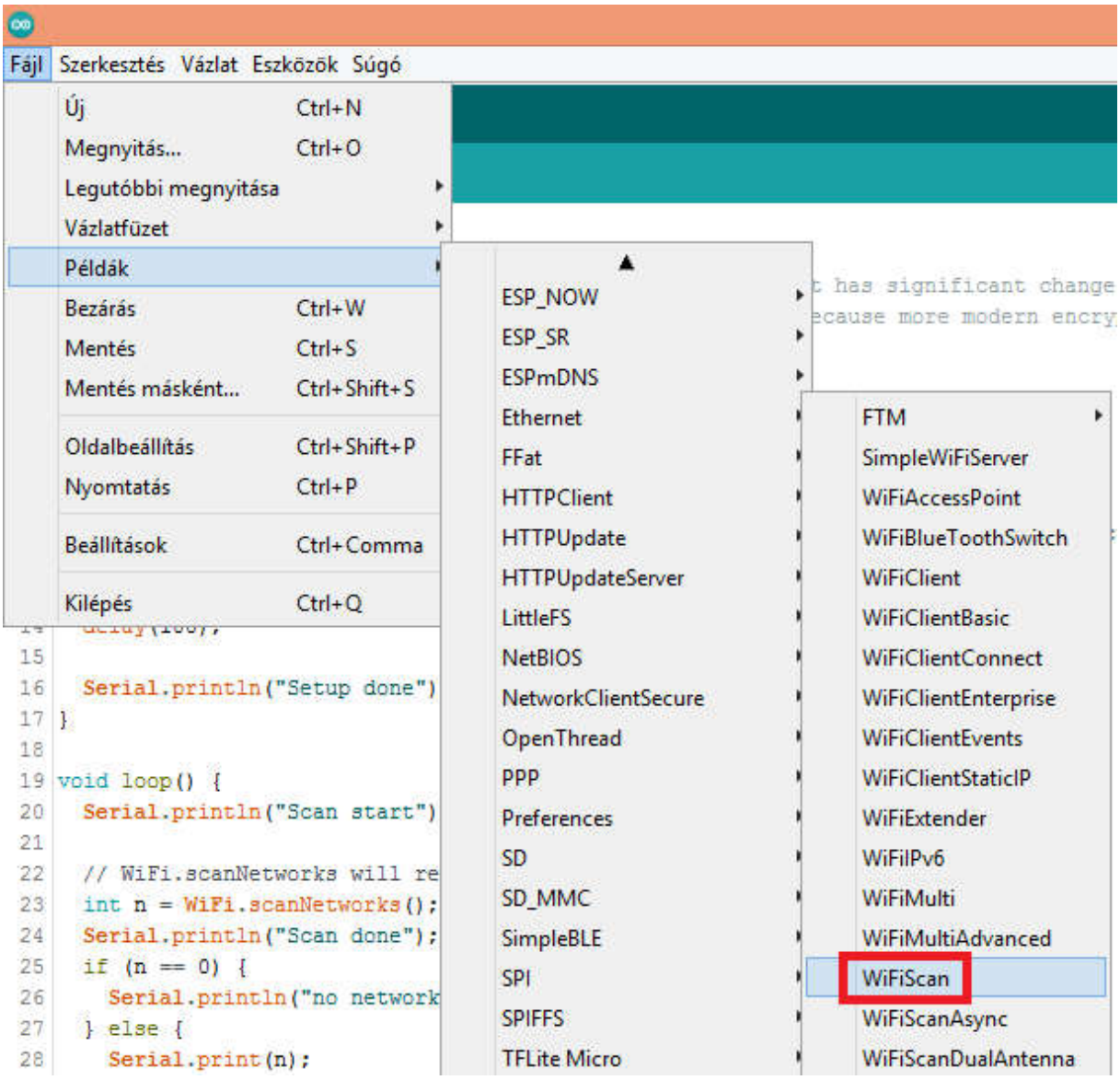
1. Válaszd ki a táblát az **Eszközök > Alaplap** (**Tools > Board**) menüben (esetemben ez a **DOIT ESP32 DEVKIT V1**)



2. Válaszd ki a portot (ha nem látod a COM-portot az Arduino IDE-ben, telepítened kell a CP210x USB-t az UART Bridge VCP-illesztőprogramjaihoz):



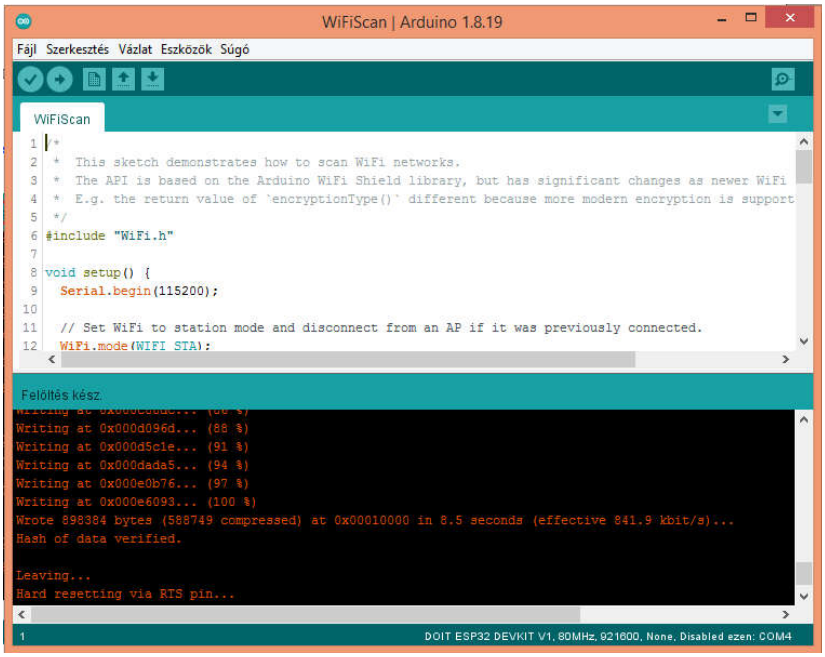
3. Nyisd meg a következő példát a **Fájl > Példák > WiFi > WiFiScan** (**File > Examples > WiFiScan**)



4. Egy új vázlat nyílik meg az Arduino IDE-ben:

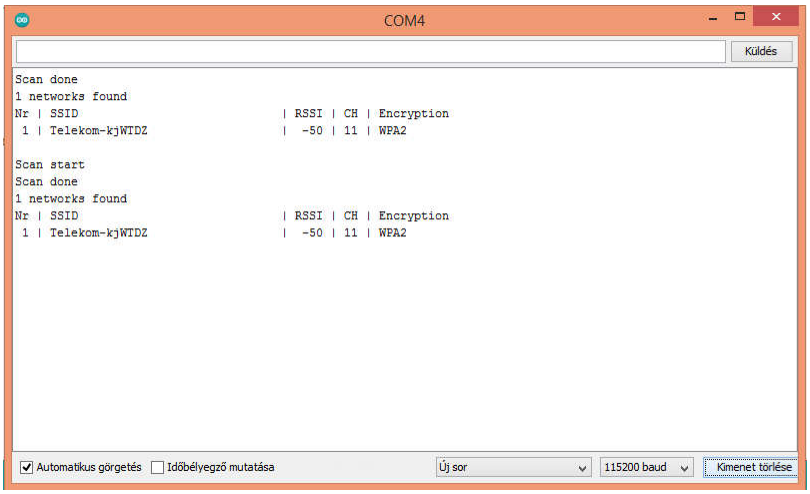
5. Nyomd meg a Feltöltés gombot az Arduino IDE-ben. Várj néhány másodpercet, amíg a kód összeáll és feltöltődik a táblára.

6. Ha minden a várt módon történt, a „Feltöltés kész” üzenetnek kell megjelennie.



7. Nyisd meg az Arduino IDE soros monitort 115200 adatátviteli sebességgel:

8. Nyomd meg az ESP32 fedélzeti Engedélyezés gombot, és látnod kell az ESP32 közelében elérhető hálózatokat:

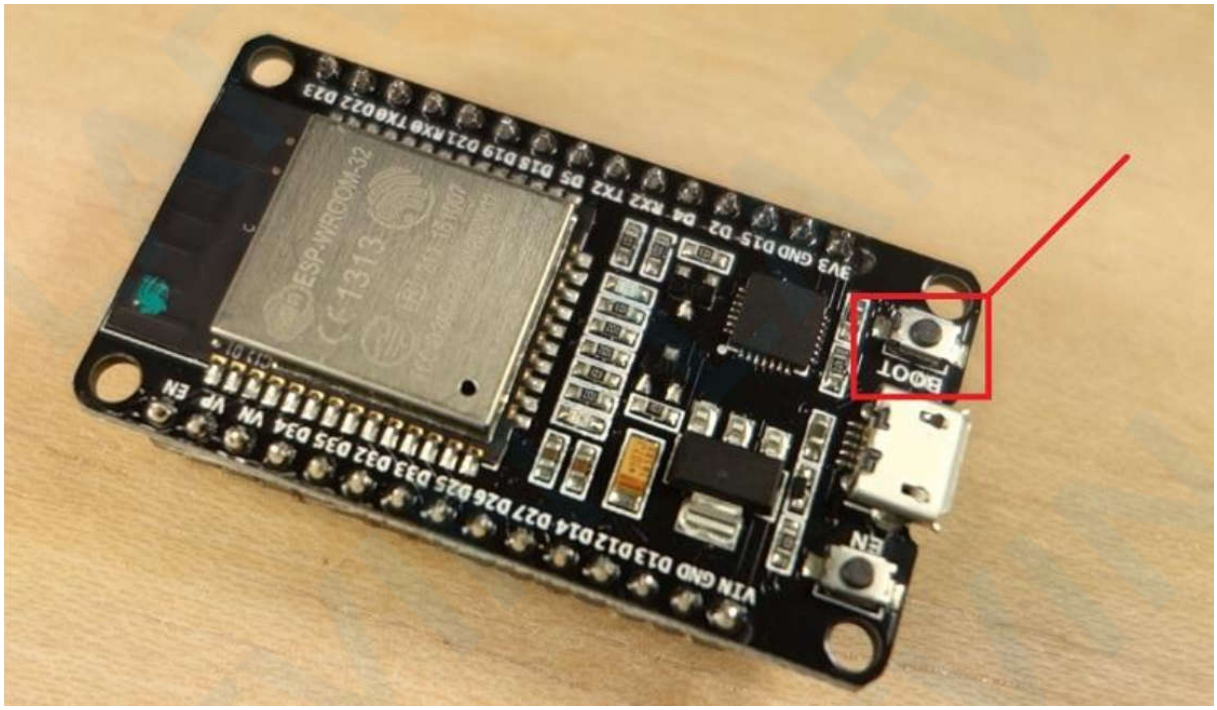


Hibaelhárítás

Ha új vázlatot próbálsz feltölteni az ESP32-re, és ezt a hibaüzenetet kapod: „Végzetes hiba történt: Nem sikerült csatlakozni az ESP32-hez: Időtúllépés… Csatlakozás…”. Ez azt jelenti, hogy az ESP32 nincs flash/feltöltés módban.

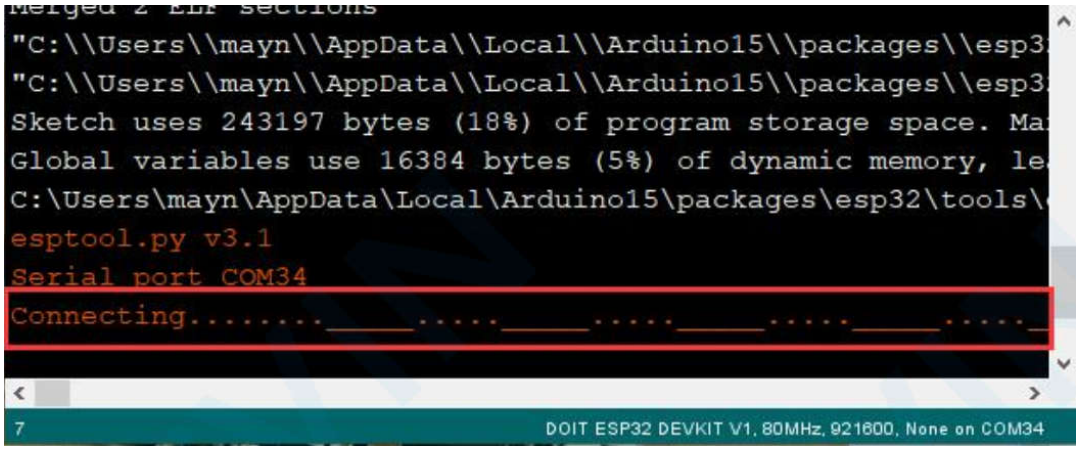
A megfelelő kártyanév és COM-port kiválasztása után kövesd az alábbi lépéseket:

- Tartsd lenyomva a „BOOT” gombot az ESP32 kártyán



- Nyomd meg a „Feltöltés” gombot az Arduino IDE-ben a vázlat feltöltéséhez:

- Miután megjelenik a „Connecting…” üzenetet az Arduino IDE-ben, engedd el az ujjadat a „BOOT” gombról:



- Ezt követően látnod kell a „Feltöltés kész” üzenetet, ennyi.

Az ESP32-n futnia kell az új vázlatnak. Nyomd meg az „ENABLE” gombot az ESP32 újraindításához és az új feltöltött vázlat futtatásához.

Ezt a gombsort kell megismételned minden alkalommal, amikor új vázlatot szeretnél feltölteni.

1. projekt ESP32 bemenetek, kimenetek

Ebből az első lépések útmutatóból megtudhatod, hogyan olvasd be a digitális bemeneteket, például egy kapcsológombot, és hogyan vezéreld a digitális kimeneteket, például egy LED-et az ESP32 és az Arduino IDE segítségével.

Előfeltételek

Az ESP32-t Arduino IDE segítségével programozzuk. Tehát a folytatás előtt győződj meg arról, hogy telepítetted az ESP32 kártyák bővítményét:

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

ESP32 vezérelt digitális kimenetei

Először is be kell állítanod a vezérelni kívánt GPIO-t OUTPUT-ként. Használd a `pinMode()` függvényt az alábbiak szerint:

```
pinMode(GPIO, OUTPUT);
```

A digitális kimenet vezérléséhez csak a `digitalWrite()` függvényt kell használnod, amely argumentumként fogadja, a hivatkozott GPIO-t (int szám), valamint a HIGH vagy LOW állapotot.

```
digitalWrite(GPIO, STATE);
```

Minden GPIO használható kimenetként, kivéve a 6–11. GPIO-t (az integrált SPI flash-hez csatlakoztatva) és a 34., 35., 36. és 39. GPIO-t (csak bemeneti GPIO-k);

Tudj meg többet az ESP32 GPIO-król: Az ESP32 tűkiosztása

ESP32 Digitális bemenetek olvasása

Először is be kell állítanod a vezérelni kívánt GPIO-t INPUT-ként. Használd a `pinMode()` függvényt az alábbiak szerint:

```
pinMode(GPIO, INPUT);
```

A digitális bemenet, például egy gomb olvasásához, a `digitalRead()` függvényt kell használni, amely argumentumként fogadja az általad hivatkozott GPIO-t (int szám).

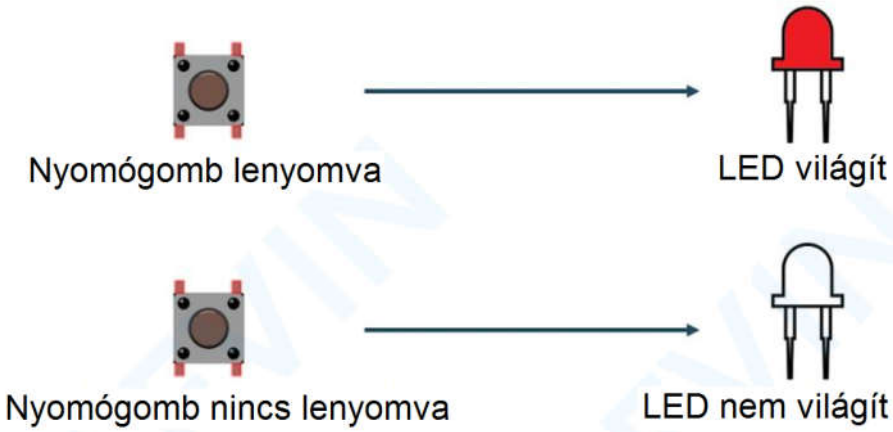
```
digitalRead(GPIO);
```

Minden ESP32 GPIO használható bemenetként, kivéve a 6-tól 11-ig terjedő GPIO-kat (az integrált SPI flash-hez csatlakoztatva).

Tudj meg többet az ESP32 GPIO-król: Az ESP32 tűkiosztása

Projekt példa

A digitális bemenetek és a digitális kimenetek használatának bemutatásához egy egyszerű projekt-példát készítünk nyomógombbal és LED-del. Leolvassuk a nyomógomb állapotát, és ennek megfelelően kapcsoljuk a LED-et, ahogy az a következő ábrán látható.



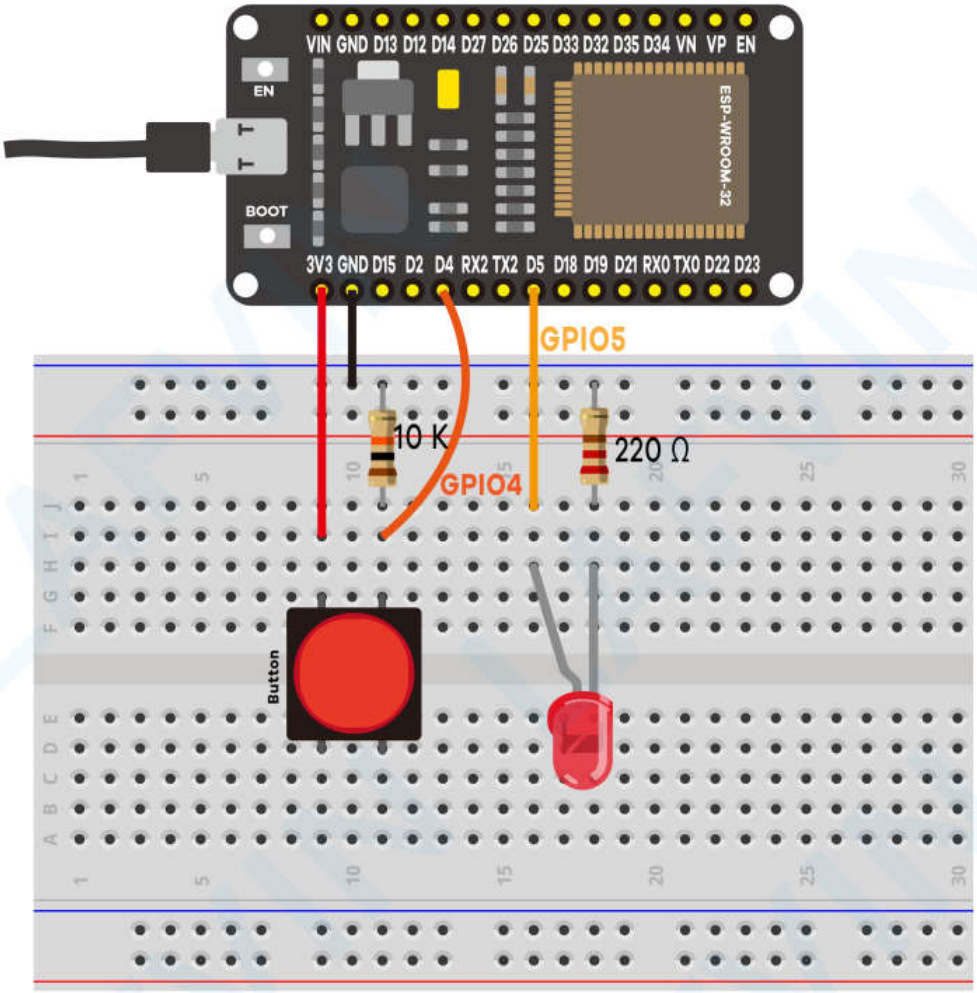
Szükséges alkatrészek

Íme az áramkör felépítéséhez szükséges alkatrészek listája:

- ESP32 DEVKIT V1
- 5 mm LED
- 220 Ohmos ellenállás
- Nyomógomb
- 10 kOhmos ellenállás
- Bekötőkártya
- Jumper vezetékek

Vázlatos diagram

A folytatás előtt össze kell állítanod egy áramkört LED-del és nyomógombbal. A LED-et a GPIO 5-höz, a nyomógombot pedig a GPIO 4-hez csatlakoztatjuk.



A kód

Nyisd meg a `Project_1_ESP32_Inputs_Outputs.ino` kódot az arduino IDE-ben.

```
// set pin numbers
const int buttonPin = 4;  // the number of the pushbutton pin
const int ledPin =  5;    // the number of the LED pin

// variable for storing the pushbutton status
int buttonState = 0;

void setup() {
  Serial.begin(115200);
  // initialize the pushbutton pin as an input
  pinMode(buttonPin, INPUT);
  // initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the state of the pushbutton value
```

```
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH
  if (buttonState == HIGH) {
    // turn LED on
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off
    digitalWrite(ledPin, LOW);
  }
}
```

Hogyan működik a kód

(A fordító megjegyzése: a fedőlapon található webcímről letölthető a magyarított változat, ahol a megjegyzéseket is lefordítottam.)

A következő két sorban változókat hozunk létre a tűk hozzárendeléséhez:

```
const int buttonPin = 4;
const int ledPin = 5;
```

A gomb a GPIO 4-hez, a LED pedig a GPIO 5-höz csatlakozik. Ha az Arduino IDE-t ESP32-vel használjuk, a 4 a GPIO 4-nek, az 5 pedig a GPIO 5-nek felel meg.

Ezután hozzunk létre egy változót a gomb állapotának megtartásához. Alapértelmezés szerint 0 (nincs lenyomva).

```
int buttonState = 0;
```

A `setup()`-ban inicializáljuk a gombot INPUT-ként, a LED-et pedig OUTPUT-ként. Ehhez használd a `pinMode()` függvényt, amely fogadja az általad hivatkozott tű módját: INPUT vagy OUTPUT.

```
pinMode(buttonPin, INPUT);
pinMode(ledPin, OUTPUT);
```

Az `loop()`-ban olvashatjuk le a gomb állapotát, és ennek megfelelően beállíthatjuk a LED-et. A következő sorban beolvassa a gomb állapotát, és elmenti a `buttonState` változóba. Ahogy korábban láttuk, a `digitalRead()` függvényt használja.

```
buttonState = digitalRead(buttonPin);
```

A következő `if` utasítás azt ellenőrzi, hogy a gomb állapota HIGH-e. Ha igen, akkor a `digitalWrite()` függvény segítségével bekapcsolja a LED-et, amely argumentumként fogadja a ledPin-t és a HIGH állapotot.

```
if (buttonState == HIGH)
{
```

```
 digitalWrite(ledPin, HIGH);
}
```

Ha a gomb állapota nem HIGH, akkor kikapcsoljuk a LED-et. Csak beállítjuk a LOW értéket második argumentumként a `digitalWrite()` függvényben.

```
else
{
digitalWrite(ledPin, LOW);
}
```

A kód feltöltése

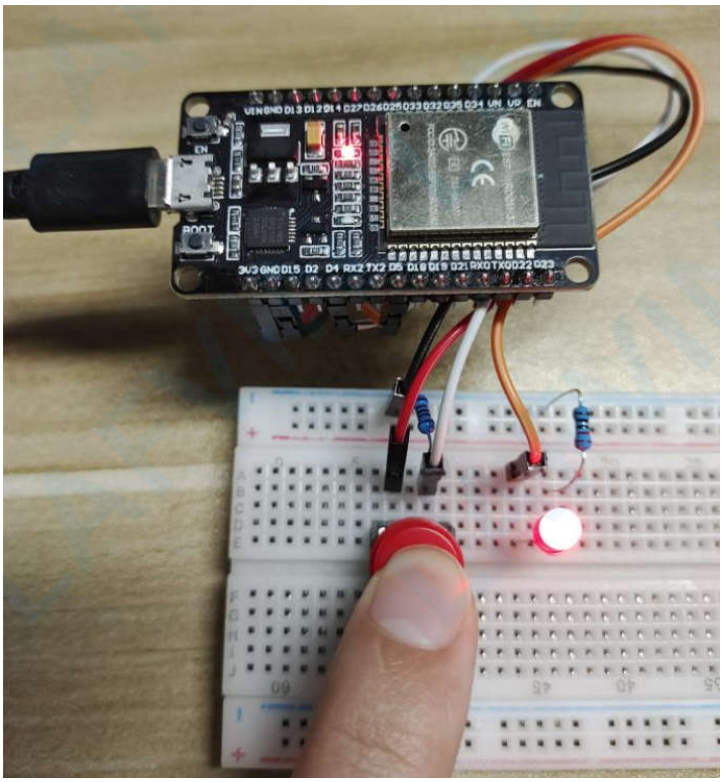
Mielőtt a feltöltés gombra kattintanál, lépj az **Eszközök > Alaplap** elemre, és válaszd ki a DOIT ESP32 DEVKIT V1 kártyát.

Válaszd az **Eszközök > Port** lehetőséget, és válaszd ki azt a COM-portot, amelyhez az ESP32 csatlakozik. Ezután nyomd meg a feltöltés gombot, és várd meg a „Feltöltés kész” üzenetet.

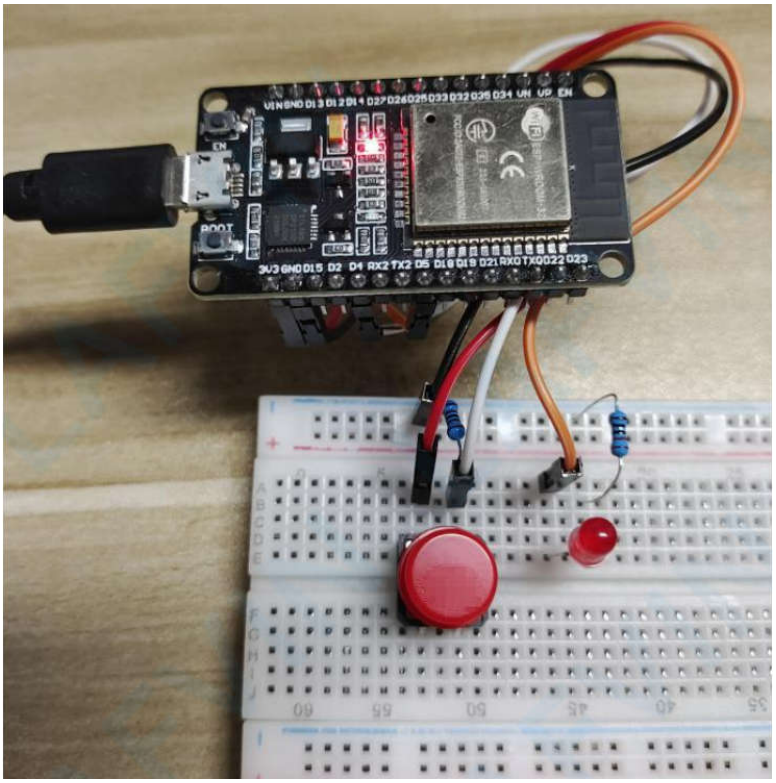
Megjegyzés: Ha sok pontot lát (csatlakozás…__…__) a hibakereső ablakban, és a „Nem sikerült csatlakozni az ESP32-hez: Időtúllépés a csomagfejlécre várakozva” üzenetet, az azt jelenti, hogy meg kell nyomnia az ESP32 fedélzeti BOOT gombját, miután a pontok megjelennek. Hibaelhárítás.

Demonstráció

A kód feltöltése után teszteld az áramkört. A LED-nek világítania kell, amikor megnyomod a nyomógombot:



És kikapcsol, amikor elengeded:



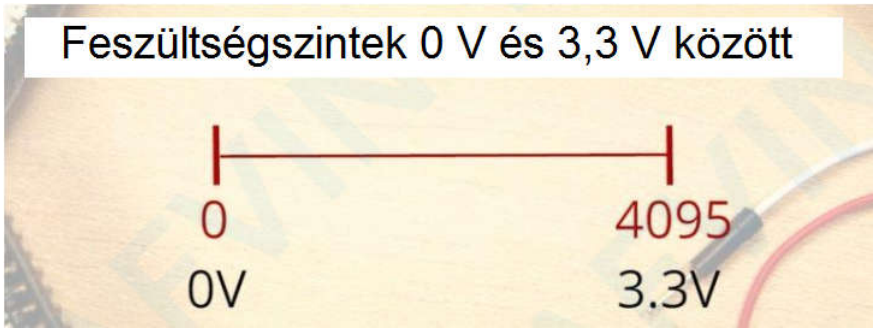
2. projekt ESP32 analóg bemenetek

Ez a projekt bemutatja, hogyan kell az analóg bemeneteket olvasni az ESP32-vel az Arduino IDE használatával. Az analóg leolvasás változó ellenállásokból, például potenciométerekből vagy analóg érzékelőkből származó értékek olvasásához használható.

Analóg bemenetek (ADC)

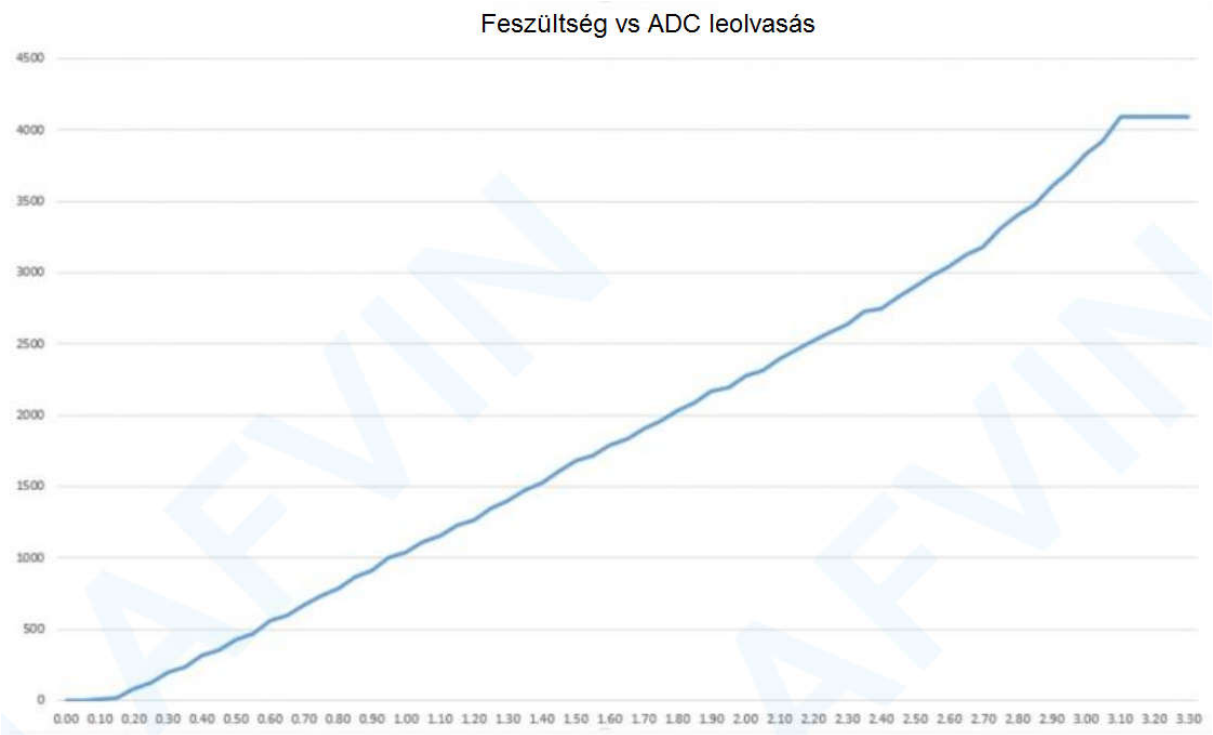
Az analóg érték ESP32-vel történő leolvasása azt jelenti, hogy 0 V és 3,3 V között változó feszültségszinteket mérhetsz.

A mért feszültséget ezután egy 0 és 4095 közötti értékhez rendeljük, amelyben 0 a 0 V-nak, 3,3 V pedig 4095-nek felel meg. A 0 V és 3,3 V közötti feszültségek között a megfelelő értéket kapjuk.



Az ADC nem lineáris

Ideális esetben lineáris viselkedés várható az ESP32 ADC érintkezők használatakor. Azonban nem ez adódik. A következő diagramon látható viselkedést fogsz kapni:



Ez azt jelenti, hogy az ESP32 nem képes megkülönböztetni a 3,3 V-ot a 3,2 V-tól. Mindkét feszültségnél ugyanazt az értéket adja: 4095.

Ugyanez történik a nagyon alacsony feszültségértékeknél is: 0 V és 0,1 V esetén ugyanazt az értéket adja: 0. Ezt szem előtt kell tartanod az ESP32 ADC érintkezők használatakor.

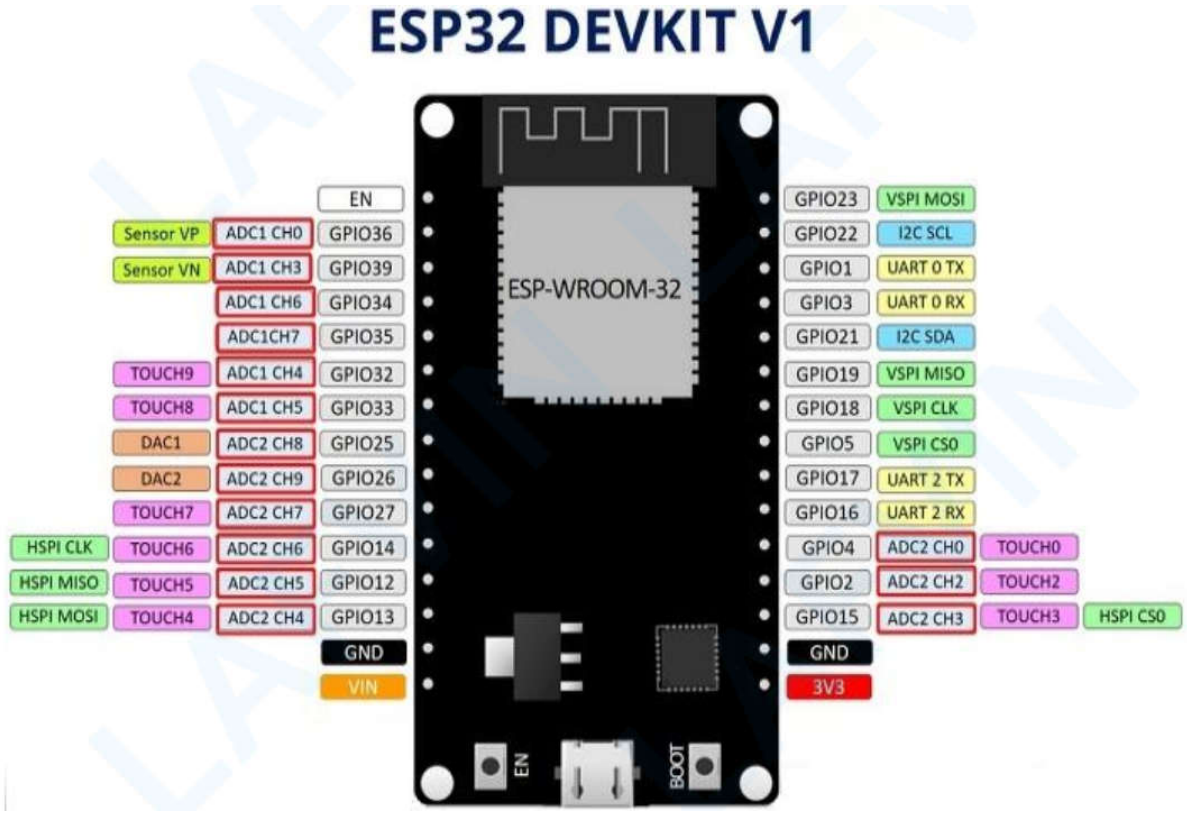
analogRead() függvény

Az analóg bemenet olvasása az ESP32-vel az Arduino IDE használatával olyan egyszerű, mint az `analógRead()` függvény használata. Argumentumként az olvasni kívánt GPIO-t fogadja:

```
analogRead(GPIO);
```

Csak 15 analóg bemenet érhető el a DEVKIT V1 kártyán (30 GPIO-val rendelkező verzió).

Nézd meg az ESP32 kártya kivezetését, és keresd meg az ADC érintkezőit. Ezek az alábbi ábrán piros kerettel vannak kiemelve.



Ezek az analóg bemeneti érintkezők 12 bites felbontással rendelkeznek. Ez azt jelenti, hogy amikor egy analóg bemenetet olvasol, annak tartománya 0 és 4095 között változhat.

Megjegyzés: Az ADC2 érintkezők nem használhatók Wi-Fi használata esetén. Tehát, ha Wi-Fi-t használsz, és problémáid vannak az ADC2 GPIO értékének lekérésével, fontold meg egy ADC1 GPIO használatát, amely megoldja a problémát.

Hogy lássuk, hogyan kapcsolódik minden egymáshoz, egy egyszerű példát mutatunk be egy analóg érték kiolvasására egy potenciométerről.

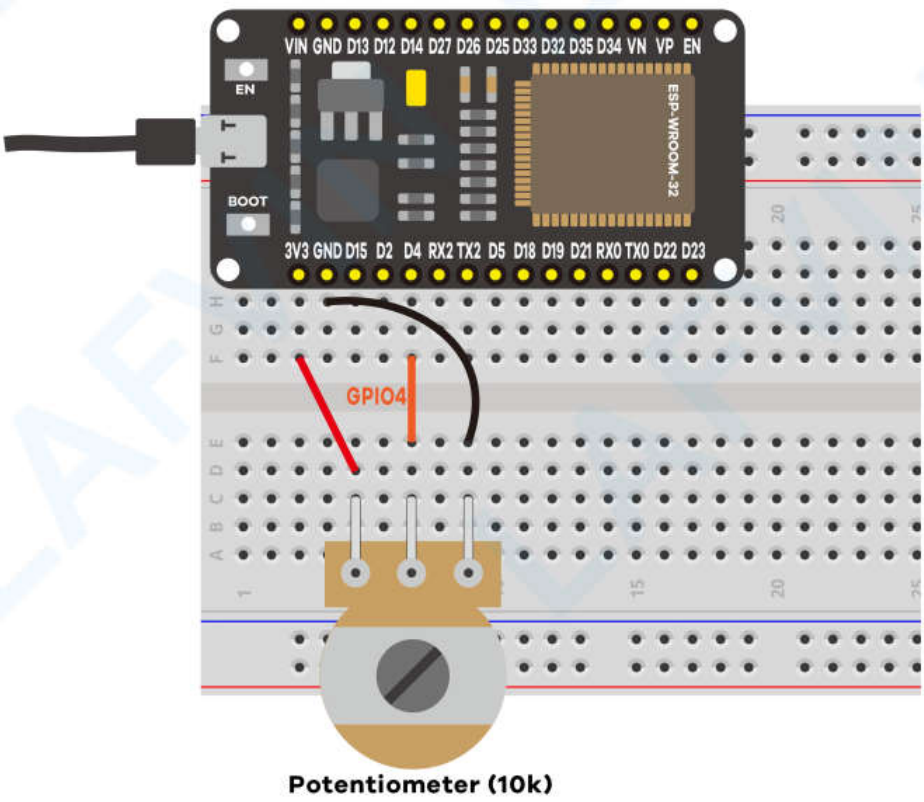
Szükséges alkatrészek

Ehhez a példához a következő alkatrészekre van szükséged:

- ESP32 DEVKIT V1 Kártya
- Potenciométer
- Bekötőkártya
- Jumper vezetékek

Vázlat

Csatlakoztass egy potenciométert az ESP32-hez. A potenciométer középső érintkezőjét a GPIO 4-hez kell csatlakoztatni. Referenciaként használhatod a következő sematikus ábrát.



Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

Nyisd meg a project_2_ESP32_Inputs_Outputs.ino kódot az arduino IDE-ben.

```
// Potentiometer is connected to GPIO 4 (Analog ADC2_CH0)
const int potPin = 4;
```

```
// variable for storing the potentiometer value
int potValue = 0;

void setup() {
  Serial.begin(115200);
  delay(1000);
}

void loop() {
  // Reading potentiometer value
  potValue = analogRead(potPin);
  Serial.println(potValue);
  delay(500);
}
```

Ez a kód egyszerűen kiolvassa az értékeket a potenciométerről, és kiírja ezeket az értékeket a soros monitorra.

A kódban először meghatározod azt a GPIO-t, amelyhez a potenciométer csatlakozik. Ebben a példában ez a GPIO 4.

```
const int potPin = 4;
```

A `setup()`-ban inicializálunk egy soros kommunikációt 115200 adatátviteli sebességgel.

```
Serial.begin(115200);
```

A `loop()`-ban az `analogRead()` függvényt használjuk az analóg bemenet olvasásához a `potPin`-től.

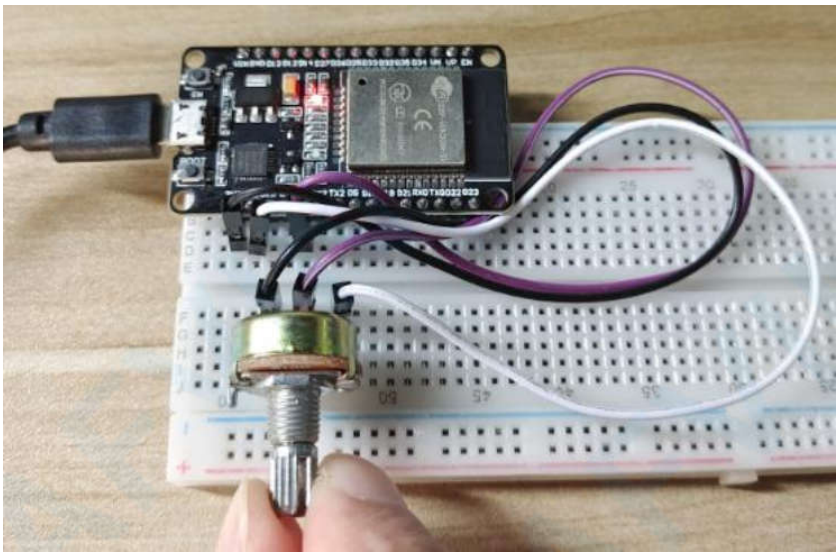
```
potValue = analogRead(potPin);
```

Végül kinyomtatjuk a potenciométerről leolvasott értékeket a soros monitoron.

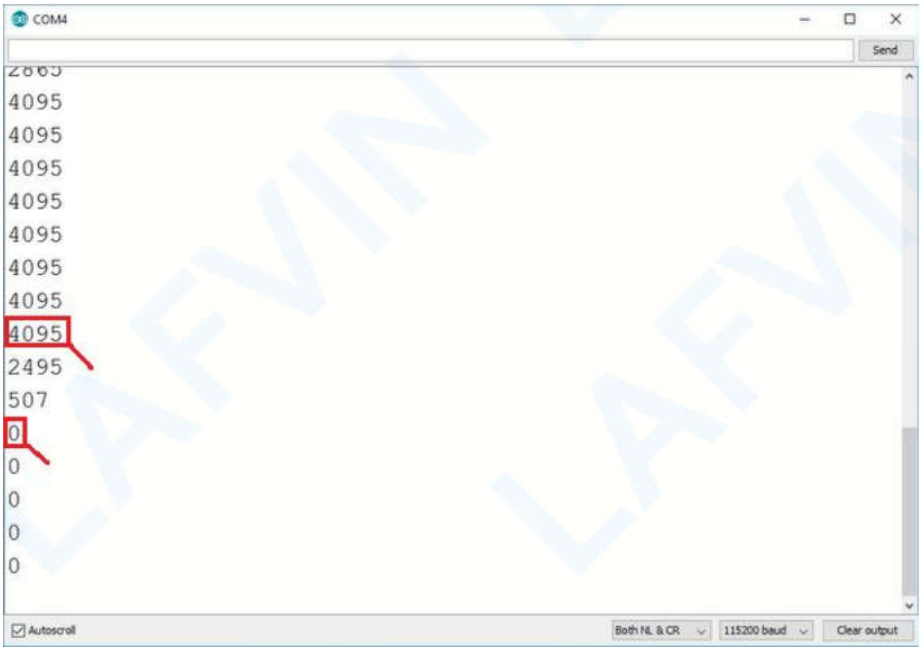
Töltsd fel a kapott kódot az ESP32-re. Győződj meg arról, hogy a megfelelő kártya és COM port van kiválasztva az **Eszközök** menüben.

A példa tesztelése

A kód feltöltése és az ESP32 reset gomb megnyomása után nyisd meg a soros monitort 115200 adatátviteli sebességgel. Forgasd el a potenciométert, és nézd meg az értékek változását.



A maximális érték 4095, a minimális érték pedig 0.



Kibontás

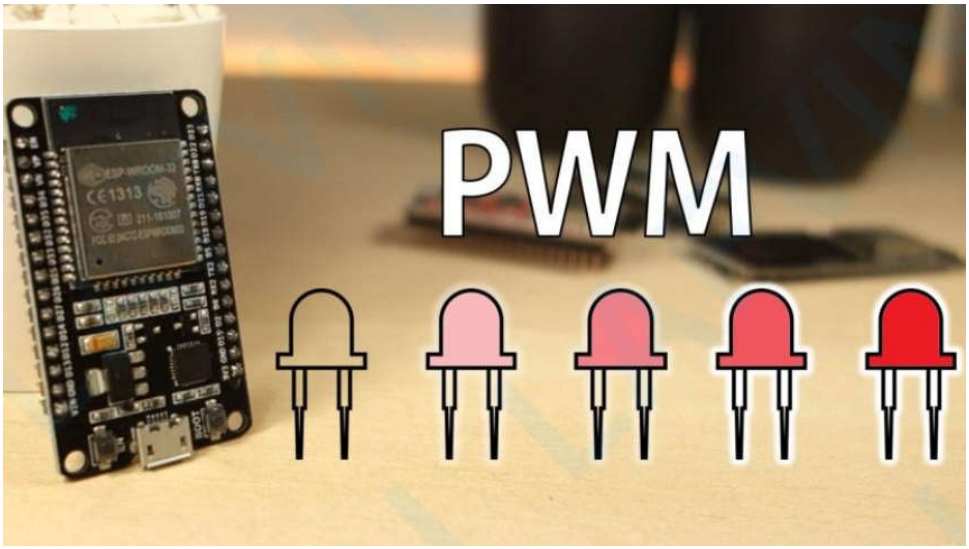
Ebből a cikkből megtanultad, hogyan olvasd le az analóg bemeneteket az ESP32 használatakor az Arduino IDE-vel. Összefoglalva:

- Az ESP32 DEVKIT V1 DOIT kártya (30 tűs változat) 15 ADC tűvel rendelkezik, amelyek segítségével az analóg bemeneteket leolvashatod.
- Ezek a tűk 12 bites felbontással rendelkeznek, ami azt jelenti, hogy 0 és 4095 közötti értékeket kaphatsz.
- Egy érték leolvasásához az Arduino IDE-ben egyszerűen az `analogRead()` függvényt használjuk.

- Az ESP32 ADC érintkezők nem lineáris viselkedésűek. Valószínűleg nem fogsz tudni különbséget tenni 0 és 0,1 V, illetve 3,2 és 3,3 V között. Ezt szem előtt kell tartanod az ADC érintkezők használatakor.

3. projekt ESP32 PWM (analóg kimenet)

Ebben az oktatóanyagban megmutatjuk, hogyan generálhatsz PWM jeleket az ESP32 segítségével az Arduino IDE használatával. Példaként megépítünk egy egyszerű áramkört, amely az ESP32 LED PWM vezérlőjével változtatja a LED fényerejét.



ESP32 LED PWM vezérlő (a fordító által módosított fejezet)

Az ESP32 LED PWM vezérlővel rendelkezik, 16 független csatornával, amelyek különböző tulajdonságú PWM jelek generálására konfigurálhatók.

Különféle függvények használhatók a PWM jelek generálására, és ugyanazokat az eredményeket érhetjük el. Használhatjuk az `analogWrite()` függvényt (mint az Arduino táblákban), vagy használhatjuk a `LEDC` függvényeket. Ez utóbbi azonban az EPS32-es csomagból a 3.x.x verzió óta el lett távolítva, vagyis nem használhatók. Mivel az ESP32 kitthez mellékelt mintapéldákban a LEDC függvényeket használatával oldják meg a feladatot, mindkét módszert bemutatom, és közzéteszem az `analogWrite()` függvénnyel történő megoldást is.

ESP32 LED PWM vezérlő LEDC függvényekkel

Íme a lépések, amelyeket követned kell a LED PWM-mel történő fényerő változtatásához az Arduino IDE használatával:

1. Először is ki kell választanod egy PWM csatornát. 16 csatorna van 0 és 15 között.

2. Ezután be kell állítanod a PWM jel frekvenciáját. LED-eknél az 5000 Hz-es frekvencia megfelelő.

3. Be kell állítanod a jel terhelhetőségi felbontását is: 1 és 16 bit közötti felbontások vannak. 8 bites felbontást használunk, ami azt jelenti, hogy a LED fényerejét 0 és 255 közötti értékkel szabályozhatod.

```
ledcSetup(GPIO, freq, resolution);
```

4. Ezután meg kell adnod, hogy melyik GPIO-n vagy GPIO-kon jelenjen meg a jel. Ehhez a következő függvényt kell használni:

```
ledcAttachPin(GPIO, channel)
```

Ez a függvény két argumentumot fogad. Az első a GPIO, amely a jelet adja ki, a második pedig a csatorna, amely a jelet generálja.

5. Végül a LED fényerejének PWM segítségével történő szabályozásához használd a következő függvényt:

```
ledcWrite(channel, dutycycle)
```

Ez a függvény argumentumként fogadja a PWM jelet generáló csatornát és a munkaciklust.

ESP32 LED PWM vezérlő analogWrite() függvénnyel

1. A `setup()`-ban a használandó GPIO-t kimenetre kell állítani.

```
pinMode(GPIO, OUTPUT);
```

2. A programban a GPIO kimenetének megfelelő értékre való beállítását az `analogWrite()` függvénnyel lehetséges.

```
analogWrite(GPIO, dutyCycle);
```

Ha szükséges lenne itt is be lehet állítani a PWM jellemzőit. A frekvencia beállítása:

```
analogWriteFrequency(GPIO, frequency);
```

Illetve a felbontás beállítása:

```
analogWriteResolution(GPIO, resolution);
```

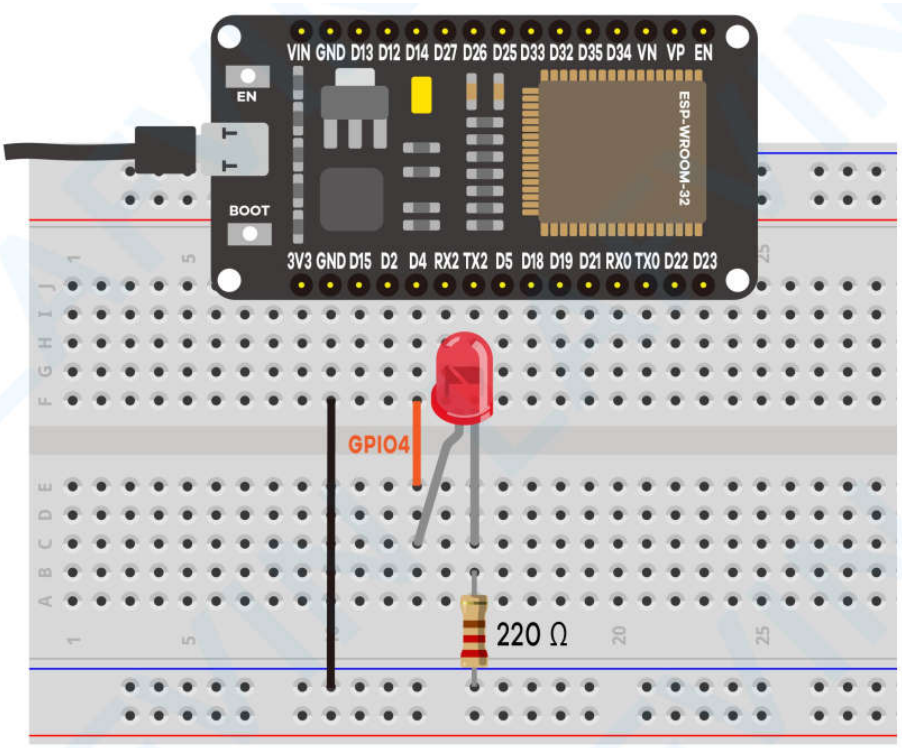
Szükséges alkatrészek

Az oktatóanyag követéséhez a következő alkatrészekre lesz szükséged:

- ESP32 DEVKIT V1 Kártya
- 5 mm-es LED
- 220 Ohmos ellenállás
- Bekötőkártya
- Jumper vezetékek

Vázlat

Csatlakoztassunk egy LED-et az ESP32-hez az alábbi kapcsolási rajz szerint. A LED-et a GPIO 4-hez kell csatlakoztatni.



Megjegyzés: tetszőleges tűt használhatsz, feltéve, hogy kimenetként működik. Minden kimenet, amelyik kimenetként működhet, használható PWM tűként. Az ESP32 GPIO-kkal kapcsolatos további információkért olvasd el a következőt: Az ESP32 tűkiosztása: Melyik GPIO érintkezőket érdemes használni?

Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

Nyisd meg a project_3_ESP32_PWM.ino kódot az arduino IDE-ben.

(A fordító megjegyzése: újabb ESP32 könyvtár használatakor a kód nem fog működni, a fentebb leírt okok miatt. A fedőlapon megtalálható címről a javított változat letölthető.)

```
// the number of the LED pin
const int ledPin = 4;  // 4 corresponds to GPIO4

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

void setup(){
  // configure LED PWM functionalitites
  ledcSetup(ledChannel, freq, resolution);
```

```
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(ledPin, ledChannel);
}

void loop(){
  // increase the LED brightness
  for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }

  // decrease the LED brightness
  for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }
}
```

Kezdjük azzal, hogy meghatározzuk azt a tűt, amelyhez a LED csatlakozik. Ebben az esetben a LED a GPIO 4-hez van csatlakoztatva.

```
const int ledPin = 4; // 16 corresponds to GPIO4
```

Ezután beállíthatjuk a PWM jel tulajdonságait. Meghatározhatjuk az 5000 Hz-es frekvenciát, kiválasztjuk a 0-s csatornát a jel generálásához, és beállítjuk a 8 bites felbontást. Ezektől eltérő tulajdonságokat is választhatnánk, hogy különböző PWM jeleket generáljunk.

```
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;
```

A `setup()`-ban be kell állítani a LED PWM-et azokkal a tulajdonságokkal, amelyeket korábban definiáltunk a `ledcSetup()` függvény segítségével, amely argumentumként fogadja a `ledChannel`-t, a `freq`-t és a `resolution`-t, az alábbiak szerint:

```
ledcSetup(ledChannel, freq, resolution);
```

Ezután ki kell választani azt a GPIO-t, amelyről a jelet kapjuk. Ehhez használjuk a `ledcAttachPin()` függvényt, amely fogadja argumentumként azt a GPIO-t, ahol a jelet szeretné megkapni, és a jelet előállító csatornát. Ebben a példában a `ledPin` GPIO-ban kapjuk meg a jelet, amely megfelel a GPIO 4-nek. A jelet generáló csatorna a `ledChannel`, amely a 0-s csatornának felel meg.

```
ledcAttachPin(ledPin, ledChannel);
```

A ciklus során a munkaciklust 0 és 255 között kell változtatni a LED fényerejének növelése érdekében.

```
for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
// changing the LED brightness with PWM
ledcWrite(ledChannel, dutyCycle);
delay(15); }
```

Majd 255 és 0 között a fényerő csökkentéséhez.

```
for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
// changing the LED brightness with PWM
ledcWrite(ledChannel, dutyCycle);
delay(15);
}
```

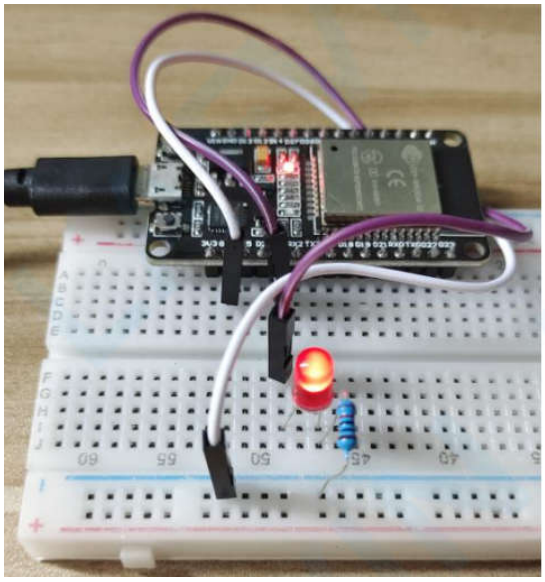
A LED fényerejének beállításához csak a `ledcWrite()` függvényt kell használni, amely argumentumként fogadja a jelet generáló csatornát és a munkaciklust.

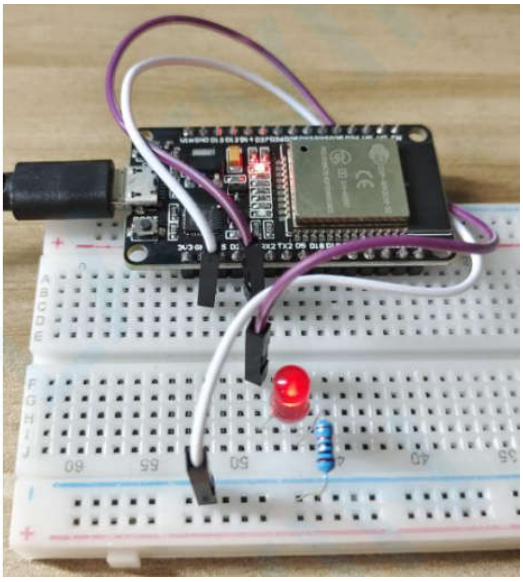
```
ledcWrite(ledChannel, dutyCycle);
```

Mivel 8 bites felbontást használunk, a munkaciklust 0 és 255 közötti értékkel szabályozzuk. Vedd figyelembe, hogy a `ledcWrite()` függvényben azt a csatornát használjuk, amely a jelet generálja, és nem a GPIO-t.

A példa tesztelése

Töltsd fel a kódot az ESP32-re. Győződj meg arról, hogy a megfelelő kártya és COM port van kiválasztva. Nézd meg az áramkörödet. Lesz egy világító LEDed, amely növeli és csökkenti a fényerőt.





4. projekt ESP32 PIR mozgásérzékelő

Ez a projekt bemutatja, hogyan lehet mozgást észlelni az ESP32-vel PIR mozgásérzékelő segítségével. A hangjelző riaszt, ha mozgást észlel, és leállítja a riasztást, ha egy előre beállított ideig (például 3 másodpercig) nem érzékel mozgást.

Hogyan működik a HC-SR501 mozgásérzékelő

A HC-SR501 érzékelő működési elve a mozgó tárgy infravörös sugárzásának változásán alapul. Ahhoz, hogy a HC-SR501 érzékelő észlelje, az objektumnak két követelménynek kell megfelelnie:

- Az objektumnak infravörös sugárzást kell kibocsátania.

- A tárgynak mozognia vagy remegnie kell.

Így:

- Ha egy tárgy kibocsátja az infravörös sugarat, de NEM mozog (például egy személy mozdulatlanul áll), az érzékelő NEM érzékeli.

- Ha egy tárgy mozog, de NEM bocsát ki infravörös sugarat (például robot vagy jármű), az érzékelő NEM érzékeli.

Bemutatjuk az időzítőket

Ebben a példában az időzítőket is bemutatjuk. Azt akarjuk, hogy a LED egy előre meghatározott számú másodpercig világítva maradjon a mozgás érzékelése után. Ahelyett, hogy egy `delay()` függvényt használnánk, amely blokkolja a kódot, és nem engedi meg, hogy mást csináljon meghatározott számú másodpercig, másik időzítőt kell használnunk.

A `delay()` függvény

Ismerned kell a `delay()` függvényt, mivel széles körben használják. Ez a funkció meglehetősen egyszerűen használható. Egyetlen int számot fogad argumentumként. Ez a szám azt az időt jelenti ezredmásodpercben, amelyet a programnak várnia kell, amíg a következő kódsorra lép.

```
delay(idő_ezredmásodpercban)
```

Ha a `delay(1000)`-et alkalmazod, a programod 1 másodpercre megáll ezen a soron.

A `delay()` egy blokkoló függvény. A blokkoló funkciók megakadályozzák, hogy a program bármi mást tegyen, amíg az adott feladat be nem fejeződik. Ha egyszerre több feladatra van szükséged, akkor nem használhatod a `delay()` függvényt.

A legtöbb projektnél kerülni kell a `delay()` -t, helyette másik időzítőt kell használni.

A `millis()` függvény

A `millis()` nevű függvény visszaadja a program első indítása óta eltelt ezredmásodpercek számát.

```
millis()
```

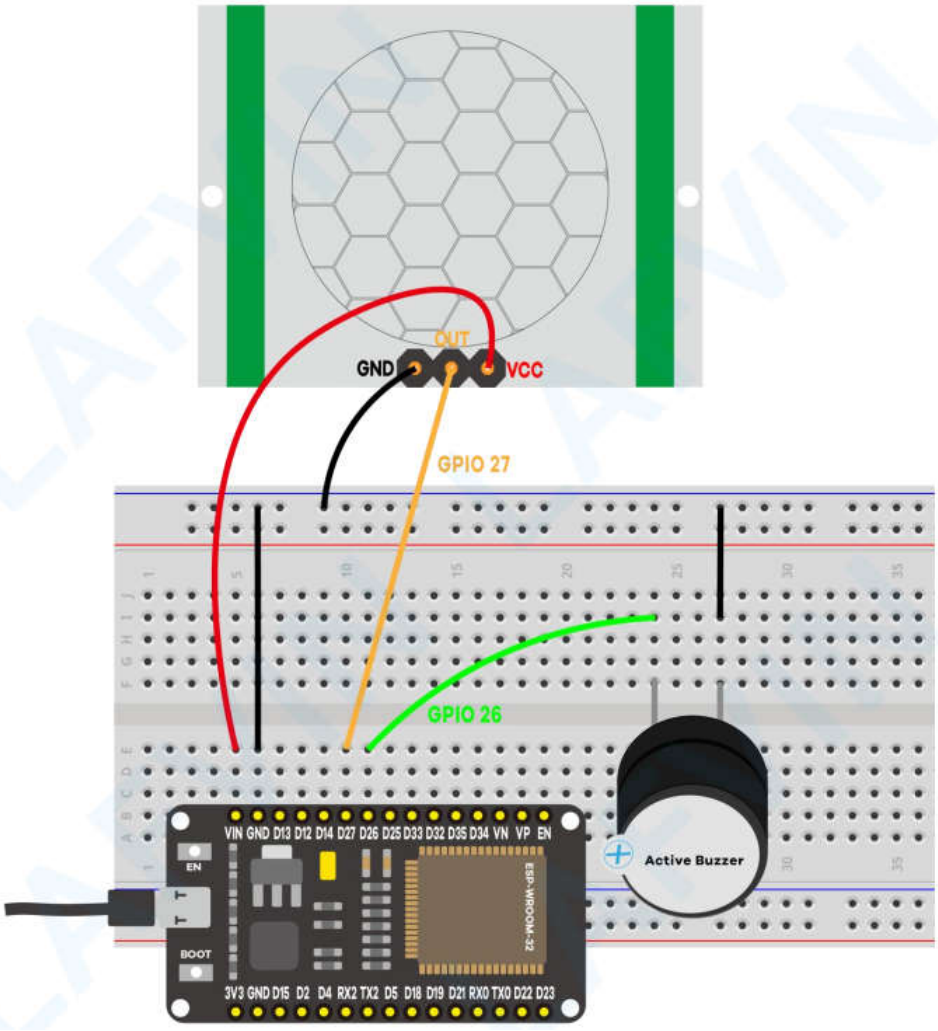
Miért hasznos ez a funkció? Mert némi matematikai számítással könnyen ellenőrizheted, hogy menynyi idő telt el anélkül, hogy blokkolnád a kódot.

Szükséges alkatrészek

Az oktatóanyag követéséhez a következő alkatrészekre van szükséged

- ESP32 DEVKIT V1 Kártya
- PIR mozgásérzékelő (HC-SR501)
- Aktív berregő
- Bekötőkártya
- Jumper vezetékek

Vázlat



Megjegyzés: A HC-SR501 üzemi feszültsége 5 V. Használd a Vin tűt a tápellátáshoz.

Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

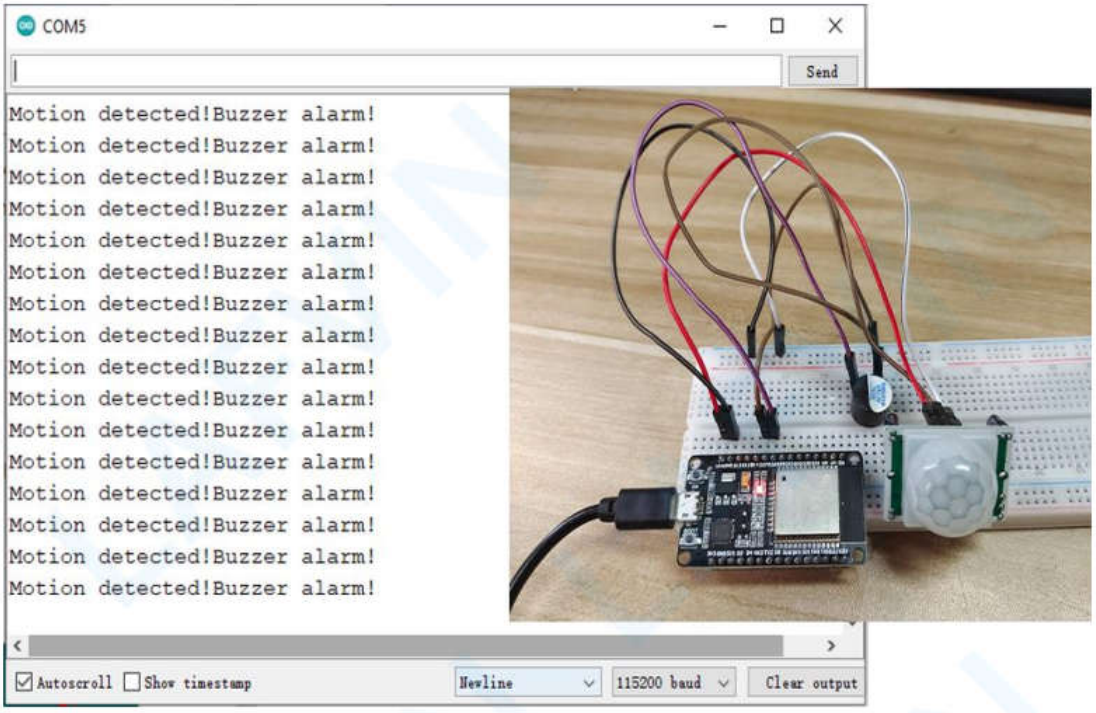
Nyisd meg a project_4_ESP32_PIR_Motion_Sensor.ino kódot az arduino IDE-ben.

Demonstráció

Töltsd fel a kódot az ESP32-re. Győződj meg arról, hogy a megfelelő kártya és COM port van kiválasztva. Töltsd fel a tesztkódot

Nyisd meg a soros monitort 115 200 adatátviteli sebességgel.

Mozgasd a kezed a PIR érzékelő elé. A hangjelzésnek be kell kapcsolnia, és a soros monitoron megjelenik a „Motion detected!Buzzer alarm!” üzenet. 4 másodperc elteltével a hangjelzésnek ki kell kapcsolnia.



5. projekt ESP32 kapcsoló webszerver

Ebben a projektben egy önálló webszervert hozunk létre ESP32-vel, amely az Arduino IDE programozási környezet segítségével vezérli a kimeneteket (két LED). A webszerver mobilra érzékeny, és bármely eszközzel elérhető, amely böngészőként működik a helyi hálózaton. Lépésről lépésre megmutatjuk, hogyan kell létrehozni a webszervert, és hogyan működik a kód.

A projekt áttekintése

Mielőtt közvetlenül a projektbe kezdenénk, fontos felvázolni, hogy mit fog csinálni webszerverünk, hogy a későbbiekben könnyebben követhessük a lépéseket.

- Az elkészített webszerver két LED-et vezérel, amelyek az ESP32 GPIO 26-hoz és a GPIO 27-hez csatlakoznak;
- Az ESP32 webszervert úgy érheted el, hogy beírod az ESP32 IP-címét a helyi hálózat böngészőjébe;
- A webszerver gombjaira kattintva azonnal módosíthatod az egyes LED-ek állapotát.

Szükséges alkatrészek

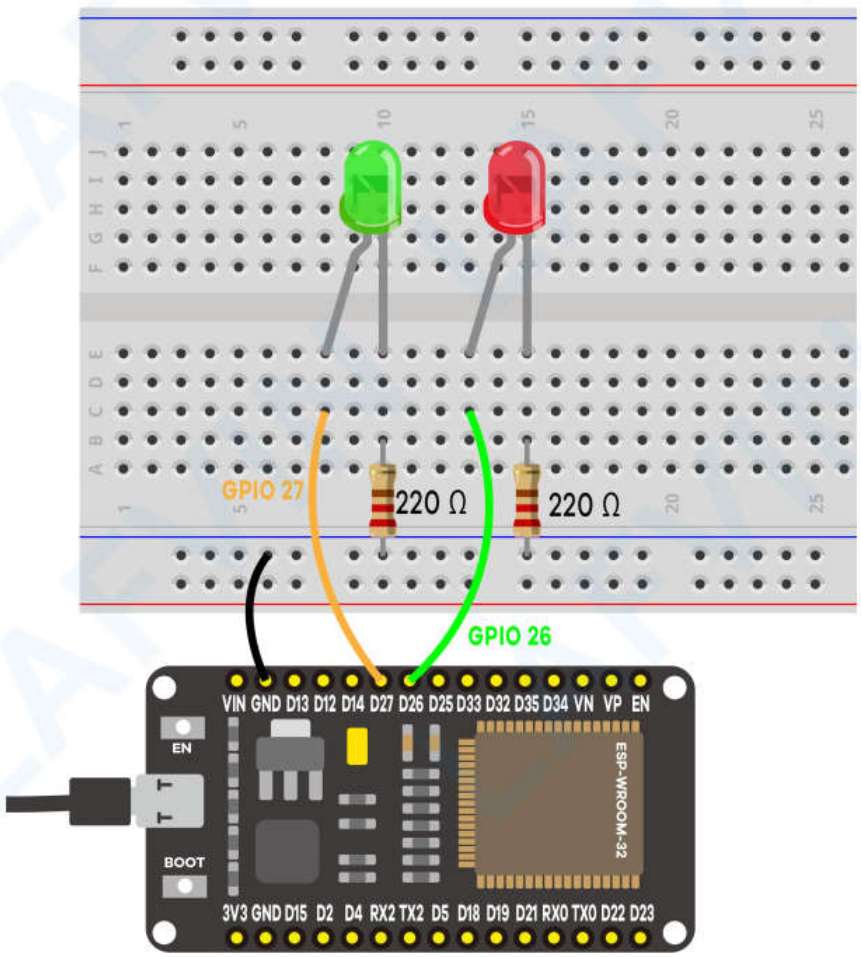
Ehhez az oktatóanyaghoz a következő alkatrészekre lesz szükséged:

- ESP32 DEVKIT V1 Kártya
- 2x 5 mm-es LED
- 2x 220 Ohmos ellenállás
- Bekötőkártya
- Jumper vezetékek

Vázlat

Kezd az áramkör felépítésével. Csatlakoztass két LED-et az ESP32-höz az alábbi kapcsolási rajz szerint – az egyik LED a GPIO 26-hoz, a másik pedig a GPIO 27-hez csatlakozik.

Megjegyzés: 30 tűs ESP32 DEVKIT DOIT kártyát használunk. Az áramkör összeszerelése előtt hogy ellenőrizd a használt kártya kivezetéseit.



Kód

Itt megadjuk az ESP32 webszervert létrehozó kódot. Nyisd meg a Project_5_ESP32_Switch_Web_Server.ino kódot az arduino IDE-ben, de még ne töltd fel. Néhány változtatást kell végrehajtanod, hogy az működjön.

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

A hálózati hitelesítési adatok beállítása

A következő sorokat kell módosítanod a hálózati hitelesítő adataiddal: SSID és jelszó. A kód jól mutatja, hogy hol kell módosítani.

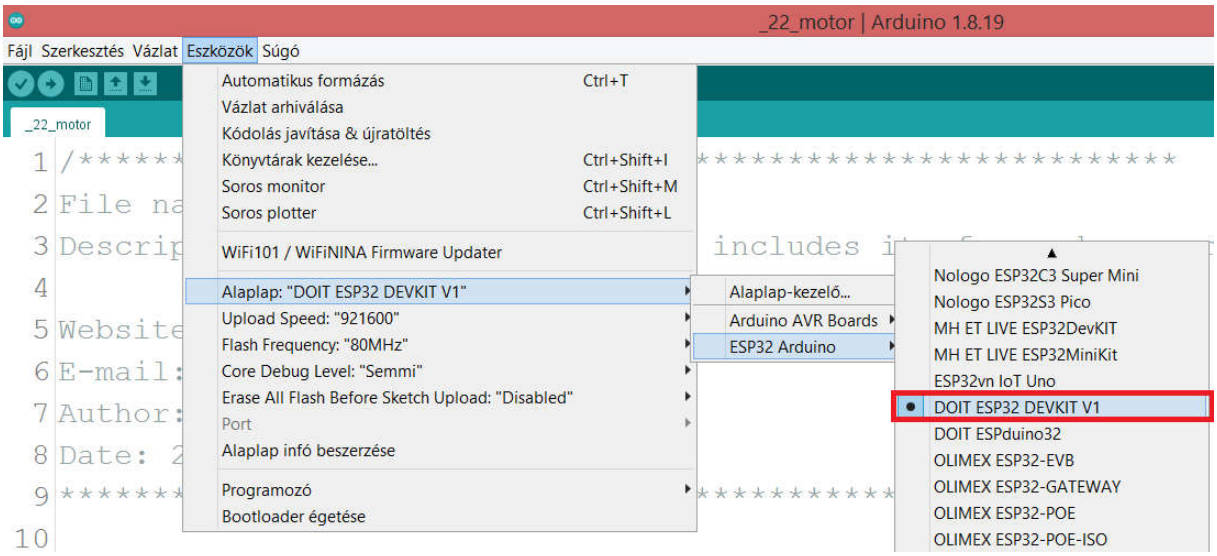
```
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

A kód feltöltése

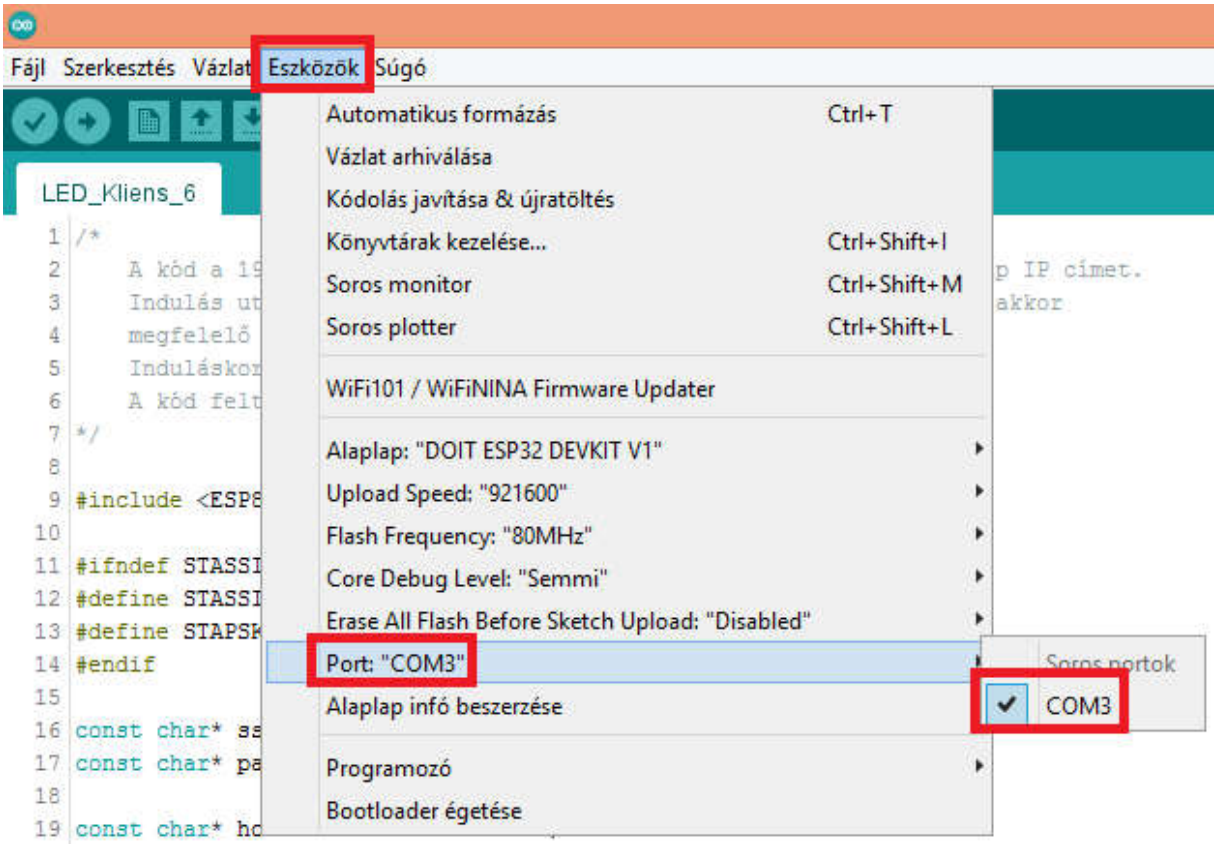
Most feltöltheted a kódot, és a webszerver azonnal működni fog. Kövesd a következő lépéseket a kód ESP32-re való feltöltéséhez:

1) Csatlakoztasd az ESP32 kártyát a számítógéphez;

2) Az Arduino IDE-ben válaszd ki a táblát az **Eszközök > Alaplap** menüpontban (esetünkben az ESP32 DEVKIT DOIT kártyát használjuk);



3) Válaszd ki a COM-portot az **Eszközök > Port** menüpontban.



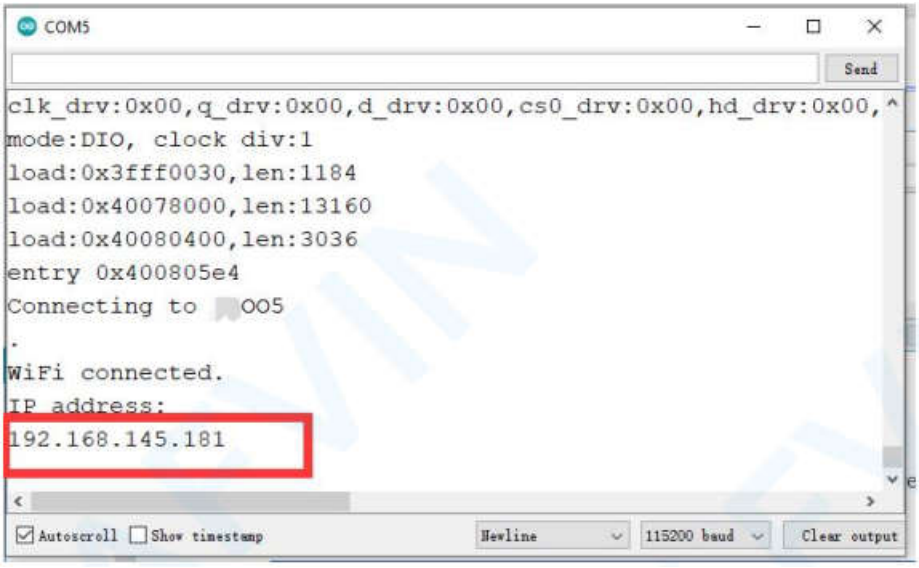
4) Nyomd meg a Feltöltés gombot az Arduino IDE-ben, és várj néhány másodpercet, amíg a kód összeáll és feltöltődik a táblára.

5) Várd meg a „Feltöltés kész” üzenetet.

Az ESP IP-címének megkeresése

A kód feltöltése után nyisd meg a soros monitort 115200 adatátviteli sebességgel.

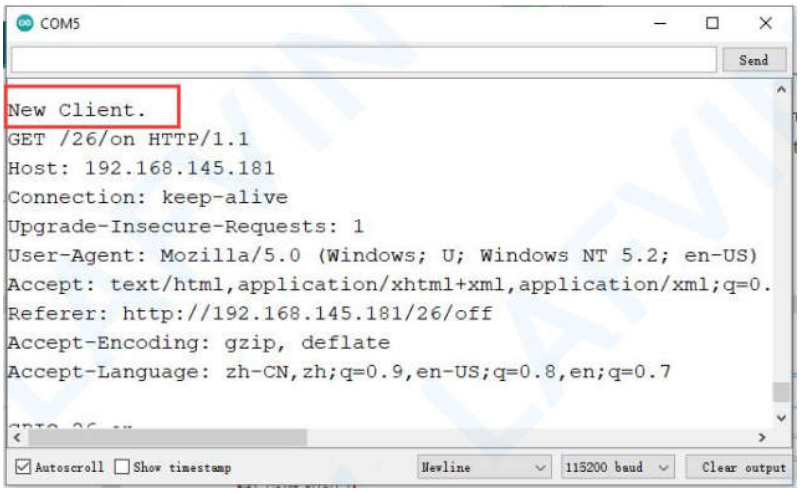
Nyomd meg az ESP32 EN gombot (reset). Az ESP32 csatlakozik a Wi-Fi-hez, és kiadja az ESP IP-címét a soros monitoron. Másold ki az IP-címet, mert szükséged lesz rá az ESP32 webszerver eléréséhez.



A webszerver elérése

A webszerver eléréséhez nyisd meg a böngészőt, illeszd be az ESP32 IP-címét, és a következő oldal jelenik meg.

Ha megnézed a soros monitort, láthatod, mi történik a háttérben. Az ESP HTTP kérést kap egy új klienstől (jelen esetben a böngészőtől).

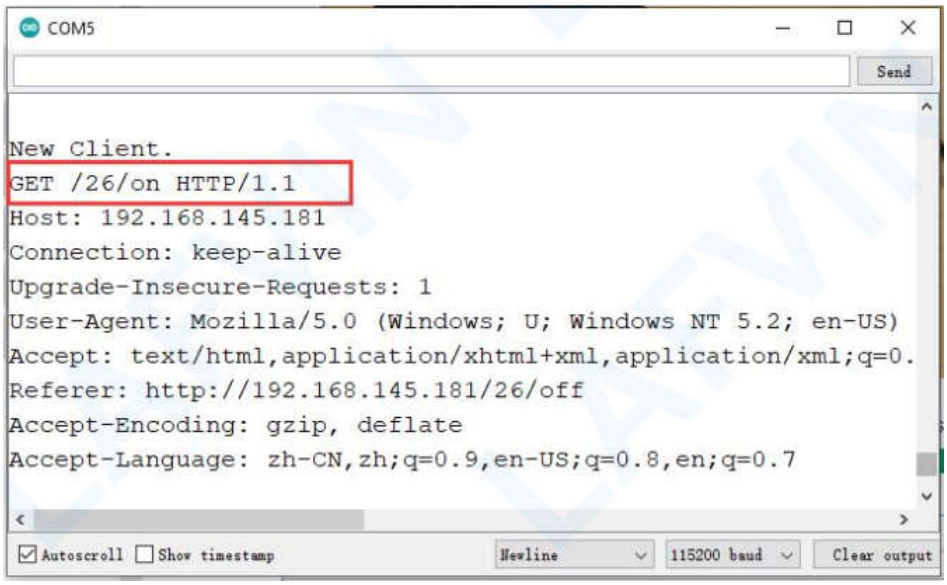


További információkat is láthatsz a HTTP-kérésről.

Demonstráció

Most tesztelheted, hogy a webszerver megfelelően működik-e. Kattints a gombokra a LED-ek vezérléséhez.

Ugyanakkor megtekintheted a soros monitort, hogy megnézd, mi történik a háttérben. Például, amikor a GPIO 26 BE gombra kattintasz a bekapcsolásához, az ESP32 kérést kap a /26/on URL-en.



Amikor az ESP32 megkapja ezt a kérést, bekapcsolja a GPIO 26-hoz csatlakoztatott LED-et, és frissíti az állapotát a weboldalon.

A GPIO 27 gombja hasonló módon működik. Teszteld, hogy megfelelően működik-e.

Hogyan működik a kód

Ebben a részben közelebbről megvizsgáljuk a kódot, hogy megtudjuk, hogyan működik.

Az első dolog, amit meg kell tennünk, hogy betöltődjön a WiFi könyvtár.

```
#include <WiFi.h>
```

Mint korábban említettük, az ssid-t és a jelszót a következő sorokba kell beírnod az idézőjelbe.

```
const char* ssid = "";
const char* password = "";
```

Ezután beállítjuk a webszervert a 80-as portra.

```
WiFiServer server(80);
```

A következő sor létrehoz egy változót a HTTP-kérés fejlécének tárolására:

```
String header;
```

Ezután létrehozunk segédváltozókat a kimenetek aktuális állapotának tárolására. Ha több kimenetet szeretnél hozzáadni, és el szeretnéd menteni az állapotukat, több változót kell létrehoznod.

```
String output26State = "off";
String output27State = "off";
```

Ezenkívül minden kimenethez hozzá kell rendelni egy GPIO-t. Itt a GPIO 26-ot és a GPIO 27-et használjuk. Bármilyen más megfelelő GPIO-t is használhatsz.

```
const int output26 = 26;const int output27 = 27;
```

setup()

Most menjünk a `setup()`-ba. Először soros kommunikációt indítunk 115200 adatátviteli sebességgel hibakeresési célból.

```
Serial.begin(115200);
```

A GPIO-kat OUTPUT-ként határozzuk meg, és LOW-ra állítjuk.

```
// Initialize the output variables as outputs
pinMode(output26, OUTPUT);
pinMode(output27, OUTPUT);
// Set outputs to LOW
digitalWrite(output26, LOW);
digitalWrite(output27, LOW);
```

A következő sorok a `WiFi.begin(ssid, password)` paranccsal kezdi a Wi-Fi kapcsolatot, megvárják a sikeres csatlakozást, és kinyomtaják az ESP IP-címét a soros monitorban.

```
// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
```

loop()

A loop()-ban programozzuk, hogy mi történik, ha egy új kliens kapcsolatot létesít a webszerverrel.

Az ESP32 mindig a következő sorral figyeli a bejövő ügyfeleket:

```
WiFiClient client = server.available();
// Listen for incoming clients
```

Amikor egy ügyféltől kérés érkezik, a beérkező adatokat elmentjük. Az ezt követő `while` ciklus mindaddig fut, amíg az ügyfél kapcsolatban marad. Nem javasoljuk a kód következő részének megváltoztatását, hacsak nem tudod pontosan, mit csinálsz.

```
if (client) { // If a new client connects,
    Serial.println("New Client."); // print a message out in the serial
                                   // port
```

```
    String currentLine = ""; // make a String to hold incoming data from
                             // the client
    while (client.connected()) { // loop while the client's connected
        if (client.available()) { // if there's bytes to read from the
                                  // client,
            char c = client.read(); // read a byte, then
            Serial.write(c); // print it out the serial monitor
            header += c;
            if (c == '\n') { // if the byte is a newline character
            // if the current line is blank, you got two newline
            // characters in a row.
            // that's the end of the client HTTP request, so send a
            // response:
                if (currentLine.length() == 0) {
                // HTTP headers always start with a response code
                // (e.g.HTTP/1.1 200 OK)
                // and a content-type so the client knows what's coming,
                // then a blank line:
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-type:text/html");
                    client.println("Connection: close");
                    client.println();
```

Az `if` és `else` utasítások a következő részben ellenőrzik, hogy melyik gombot nyomták meg a web-oldalon, és ennek megfelelően szabályozza a kimeneteket. Amint azt korábban láttuk, a megnyomott gombtól függően különböző URL-ekre küldünk kérést.

```
// turns the GPIOs on and off
if (header.indexOf("GET /26/on") >= 0){
    Serial.println("GPIO 26 on");
    output26State = "on";
    digitalWrite(output26, HIGH);
}
else if (header.indexOf("GET/26/off") >= 0) {
    Serial.println("GPIO 26 off");
    output26State = "off";
    digitalWrite(output26, LOW);
}
else if (header.indexOf("GET/27/on") >= 0) {
    Serial.println("GPIO 27 on");
    output27State = "on";
    digitalWrite(output27, HIGH);
}
else if (header.indexOf("GET/27/off") >= 0) {
    Serial.println("GPIO 27 off");
    output27State = "off";
```

```
    digitalWrite(output27, LOW);
}
```

Például, ha megnyomtad a GPIO 26 ON gombot, az ESP32 kérést kap a **/26/ON** URL-en (látjuk, hogy ez az információ a soros monitor HTTP fejlécében található). Így ellenőrizhetjük, hogy a fejléc tartalmazza-e a **GET /26/on** kifejezést. Ha tartalmazza, akkor az `output26state` változót ON-ra állítjuk, és az ESP32 bekapcsolja a LED-et.

Ez hasonlóan működik a többi gombnál is. Tehát, ha további kimeneteket szeretnél hozzáadni, módosítanod kell a kód ezen részét, hogy tartalmazza azokat.

A HTML weboldal megjelenítése

A következő dolog, amit meg kell tennünk, a weboldal létrehozása. Az ESP32 választ küld a böngészőnek néhány HTML kóddal a weboldal elkészítéséhez.

A weblap a `client.println()` kifejezés használatával kerül elküldésre az ügyfélnek. Argumentumként meg kell adnod, hogy mit szeretnél küldeni az ügyfélnek.

Az első dolog, amit el kell küldenünk, mindig a következő sor, amely azt jelzi, hogy HTML-t küldünk.

```
<!DOCTYPE HTML><html>
```

Ezután a következő sor minden webböngészőben érzékennyé teszi a weboldalt.

```
client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
```

A következőket pedig arra használjuk, hogy megakadályozzuk a kéréseket a faviconban. - Nem kell aggódnod emiatt a sor miatt.

```
client.println("<link rel=\"icon\" href=\"data:,\">");
```

A weboldal stílusának kialakítása

Ezután van néhány CSS-szövegünk a gombok és a weboldal megjelenésének stílusához. Kiválasztjuk a Helvetica betűtípust, meghatározzuk a megjelenítendő tartalmat blokként és középre igazítva.

```
client.println("<style>html { font-family: Helvetica; display:
inline-block; margin: 0px auto; text-align: center;}");
```

Gombjainkat #4CAF50 színnel, szegély nélkül, fehér színű szöveggel és ezzel a béleléssel: 16px 40px. A szövegdekorációt is none-ra állítjuk, a betűméretet, a margót és a kurzort pointerre állítjuk.

```
client.println(".button { background-color: #4CAF50; border: none;
color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;}");
```

Meghatározzuk egy második gomb stílusát is, a gomb összes tulajdonságával, amit korábban definiáltunk, de más színnel. Ez lesz a kikapcsoló gomb stílusa.

```
client.println(".button2 {background-color:#555555;}</style></head>");
```

A weboldal első címsorának beállítása

A következő sorban beállítjuk weboldalának első címsorát. Itt található az „ESP32 webszerver”, de ezt a szöveget tetszés szerint módosíthatjuk.

```
// Web Page Heading
client.println("<h1>ESP32 Web Server</h1>");
```

A gombok és a megfelelő állapot megjelenítése

Ezután írjunk egy bekezdést a GPIO 26 aktuális állapotának megjelenítéséhez. Amint láthatod, az `output26State` változót használjuk, így az állapot azonnal frissül, amikor ez a változó megváltozik.

```
client.println("<p>GPIO 26 - State " + output26State + "</p>");
```

Ezután a GPIO aktuális állapotától függően megjelenítjük a be vagy a ki gombot. Ha a GPIO aktuális állapota ki van kapcsolva, akkor az ON gombot, ha nem, akkor az OFF gombot.

```
if (output26State=="off") {
client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
}
else
{
client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
}
```

Ugyanezt az eljárást alkalmazzuk a GPIO 27 esetében is.

A kapcsolat lezárása

Végül, amikor a válasz véget ér, töröljük a fejlécváltozót, és leállítjuk a kapcsolatot az ügyféllel a `client.stop()` segítségével.

```
// Clear the header variable
header = "";
// Close the connection
client.stop();
```

Lezárás

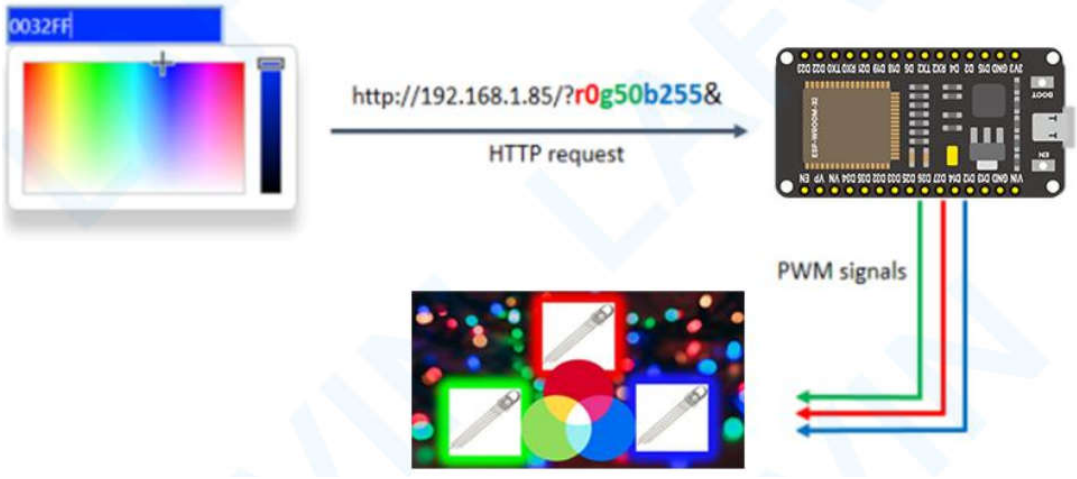
Ebben az oktatóanyagban megmutattuk, hogyan építhetsz webszervert az ESP32 segítségével. Mutattunk egy egyszerű példát, amely két LED-et vezérel, de az ötlet az, hogy ezeket a LED-eket relével vagy bármilyen más vezérelni kívánt kimenettel helyettesítsük.

6. projekt RGB LED webszerver

Ebben a projektben megmutatjuk, hogyan lehet távolról vezérelni egy RGB LED-et ESP32 kártyával egy színválasztóval ellátott webszerver segítségével.

A projekt áttekintése

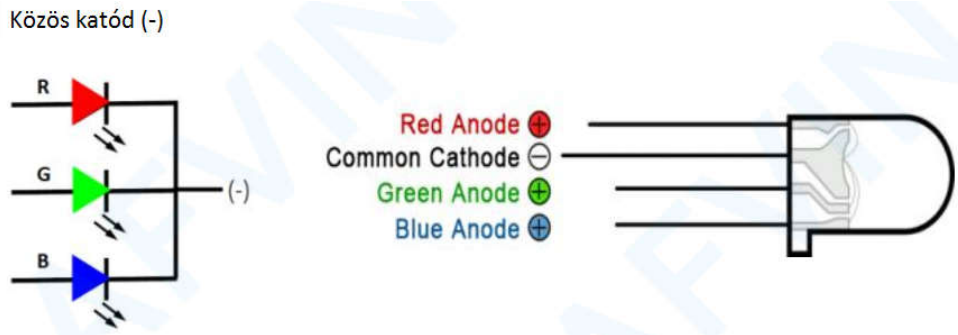
Mielőtt elkezdenénk, nézzük meg, hogyan működik ez a projekt:



- Az ESP32 webszerver megjelenít egy színválasztót.
- Amikor kiválasztunk egy színt, a böngésző kérést küld egy URL-re, amely tartalmazza a kiválasztott szín R, G és B paramétereit.
- Az ESP32 fogadja a kérést, és felosztja az értéket minden szín paraméterhez.
- Ezután a megfelelő értékű PWM jelet küld az RGB LED-et vezérlő GPIO-knak.

Hogyan működnek az RGB LED-ek?

Egy közös katódú RGB LED-ben mindhárom LED negatív csatlakozáson (katódon) osztozik. A készlet mindegyike közös katódú RGB.



Hogyan készítsünk különböző színeket?

RGB LED-del természetesen piros, zöld és kék fényt is előállíthatunk, illetve az egyes LED-ek intenzitásának konfigurálásával más színeket is előállíthatunk.

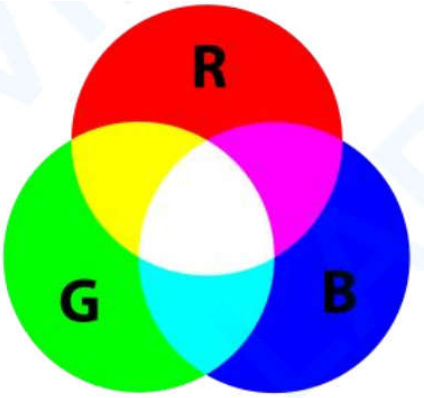
Például a tisztán kék fény előállításához a kék LED-et a legmagasabb intenzitásra, a zöld és a piros LED-eket pedig a legalacsonyabb intenzitásra kell állítani. Fehér fényhez mindhárom LED-et a legmagasabb intenzitásra kell állítani.

Színek keverése

Más színek előállításához a három színt különböző intenzitással kombinálhatja. Az egyes LED-ek intenzitásának beállításához PWM jelet használhatsz.

Mivel a LED-ek nagyon közel vannak egymáshoz, szemünk inkább a színkombináció eredményét látja, nem pedig a három színt külön-külön.

Ha ötletet szeretne a színek kombinálására, tekintse meg a következő táblázatot. Ez a legegyszerűbb színkeverési táblázat, de ötletet ad a működéséről és a különböző színek előállításáról.

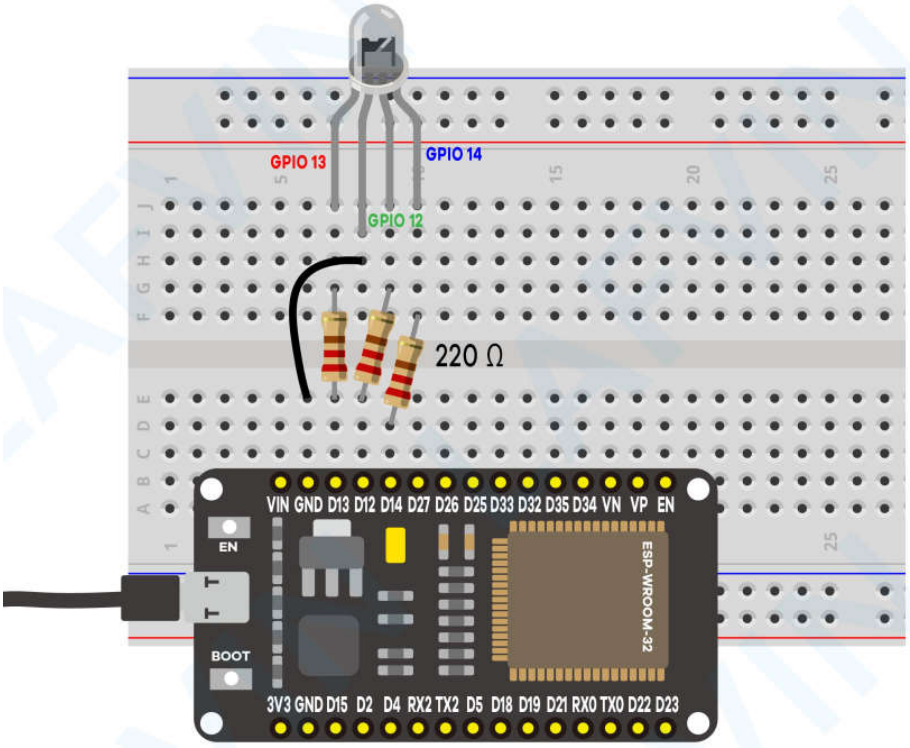


Szükséges alkatrészek

Ehhez az oktatóanyaghoz a következő alkatrészekre lesz szükséged:

- ESP32 DEVKIT V1 Kártya
- RGB LED
- 3x 220 Ohmos ellenállás
- Bekötőkártya
- Jumper vezetékek

Vázlat



Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

Az áramkör összeállítása után nyisd meg a project_6_RGB_LED_Web_Server.ino kódot az arduino IDE-ben.

A kód feltöltése előtt ne felejtsd el beírni a hálózati hitelesítő adatait, hogy az ESP csatlakozhasson a helyi hálózathoz.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Hogyan működik a kód

Az ESP32 vázlat a `WiFi.h` könyvtárat használja.

```
#include <WiFi.h>
```

A következő sorok karakterlánc-változókat határoznak meg a kérelem R, G és B paramétereinek tárolására.

```
String redString = "0";
String greenString = "0";
String blueString = "0";
```

A következő négy változó a HTTP-kérés későbbi dekódolására szolgál.

```
int pos1 = 0;
int pos2 = 0;
int pos3 = 0;
int pos4 = 0;
```

Hozzunk létre három változót a GPIO-khoz, amelyek az R, G és B sáv paramétereit vezérlik. Ebben az esetben a GPIO 13-at, a GPIO 12-t és a GPIO 14-et használjuk.

```
const int redPin = 13;
const int greenPin = 12;
const int bluePin = 14;
```

Ezeknek a GPIO-knak PWM jeleket kell kiadniuk, ezért először a PWM tulajdonságait kell konfigurálnunk. Állítsuk a PWM jel frekvenciáját 5000 Hz-re. Ezután minden színhez társítsunk egy PWM-csatornát.

```
const int freq = 5000;
const int redChannel = 0;
const int greenChannel = 1;
const int blueChannel = 2;
```

És végül állítsuk a PWM csatornák felbontását 8 bitesre.

```
const int resolution = 8;
```

A `setup()`-ban rendeljük hozzá a PWM-tulajdonságokat a PWM-csatornákhoz.

```
ledcSetup(redChannel, freq, resolution);
ledcSetup(greenChannel, freq, resolution);
ledcSetup(blueChannel, freq, resolution);
```

A következő kódrész megjeleníti a színválasztót a weboldalon, és kérést küld a kiválasztott szín alapján.

```
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\"content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");
client.println("<link
rel=\"stylesheet\"href=\"https://stackpath.bootstrapcdn.com/bootstrap/4.3.
1/css/bootstrap.min.css\">");
client.println("<scriptsrc=\"https://cdnjs.cloudflare.com/ajax/libs/jscolo
r/2.0.4/jscolor.min.js\"></script>");
```

```
client.println("</head><body><div class=\"container\"><div
class=\"row\"><h1>ESP Color Picker</h1></div>");
client.println("<a class=\"btn btn-primary btn-lg\" href=\"#\"
id=\"change_color\" role=\"button\">Change Color</a> ");
client.println("<input class=\"jscolor
{onFineChange:'update(this)'}\" id=\"rgb\"></div>");
client.println("<script>function update(picker)
{document.getElementById('rgb').innerHTML =
Math.round(picker.rgb[0]) + ', ' + Math.round(picker.rgb[1]) + ',
' + Math.round(picker.rgb[2]);");
client.println("document.getElementById(\"change_color\").href=\"?
r\" + Math.round(picker.rgb[0]) + \"g\" + Math.round(picker.rgb[1])
+ \"b\" + Math.round(picker.rgb[2]) +
\"&\";}</script></body></html>");// The HTTP response ends with
another blank line
client.println();
```

Amikor kiválaszt egy színt, a következő formátumú kérést kapja.

```
/?r201g32b255&
```

Tehát fel kell osztanunk ezt a karakterláncot, hogy megkapjuk az R, G és B paramétereket. A paraméterek `redString`, `greenString` és `blueString` változókban vannak elmentve, és értékük 0 és 255 között lehet.

```
pos1 = header.indexOf('r');
pos2 = header.indexOf('g');
pos3 = header.indexOf('b');
pos4 = header.indexOf('&');
redString = header.substring(pos1+1, pos2);
greenString = header.substring(pos2+1, pos3);
blueString = header.substring(pos3+1, pos4);
```

A sáv ESP32-vel történő vezérléséhez használjuk a `ledcWrite()` függvényt, hogy PWM-jeleket generáljon a HTTP-kérésből dekódolt értékekkel.

```
ledcWrite(redChannel, redString.toInt());
ledcWrite(greenChannel, greenString.toInt());
ledcWrite(blueChannel, blueString.toInt());
```

Megjegyzés: Tudj meg többet a PWM-ről ESP32-vel: 3. projekt ESP32 PWM (analóg kimenet)

A sáv ESP8266-tal, illetve az ESP32 könyvtár 3.x.x vagy annál újabb verziójával történő vezérléséhez az `analogWrite()` függvényt kell használnunk PWM-jelek generálására a HTTP-kérésből dekódolt értékekkel. Lásd: ESP32 LED PWM vezérlő

```
analogWrite(redPin, redString.toInt());
analogWrite(greenPin, greenString.toInt());
analogWrite(bluePin, blueString.toInt())
```

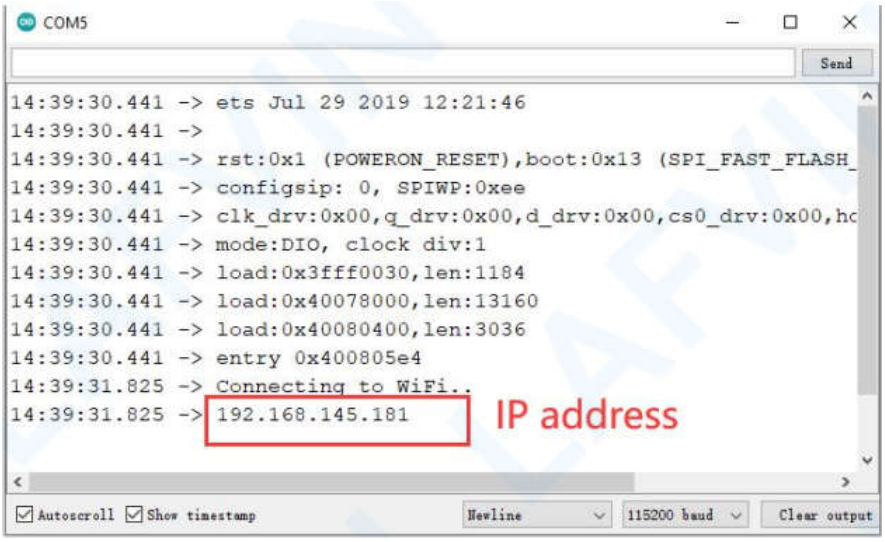
Mivel az értékeket egy string változóban kapjuk meg, a `toInt()` metódus segítségével egész számokká kell konvertálnunk.

Demonstráció

A hálózati hitelesítő adatok behelyezése után válaszd ki a megfelelő kártyát és COM-portot, és töltsd fel a kódot az ESP32-re. Emlékeztető: Töltsd fel a tesztkódot

Feltöltés után nyisd meg a soros monitort 115200 adatátviteli sebességgel, és nyomd meg az ESP engedélyezése/visszaállítása gombot. Meg kell kapnod a kártya IP-címét.

Megjegyzés: A böngészőnek és az ESP32-nek ugyanahhoz a helyi hálózathoz kell csatlakoznia.

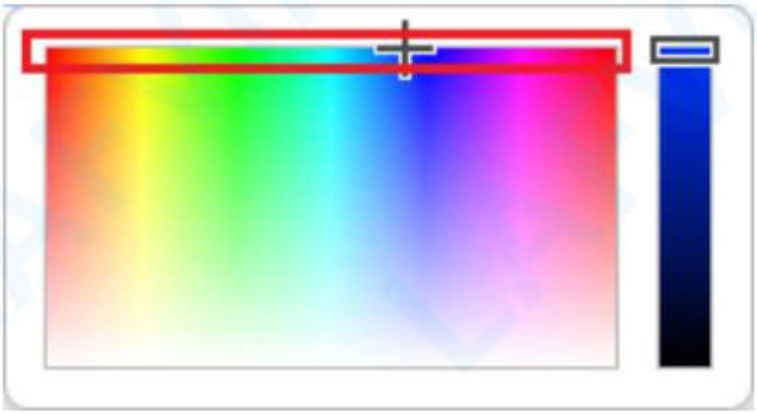


Nyisd meg a böngészőt, és írd be az ESP IP-címét. Most használd a színválasztót az RGB LED színének kiválasztásához.

Ezután meg kell nyomnod a „Szín módosítása” gombot, hogy a szín érvénybe lépjen.

Az RGB LED kikapcsolásához válaszd ki a fekete színt.

A legerősebb színek (a színválasztó tetején) azok, amelyek jobb eredményeket produkálnak.



7. projekt ESP32 Relé webszerver

A relé használata az ESP32-vel nagyszerű módja az AC háztartási készülékek távoli vezérlésének. Ez az oktatóanyag elmagyarázza, hogyan lehet vezérelni egy relémodult az ESP32-vel. Megnézzük, hogyan működik a relémodul, hogyan lehet a relét az ESP32-höz csatlakoztatni, és hogyan építhetünk fel egy webszervert a relé távoli vezérléséhez.

A relék bemutatása

A relé egy elektromosan működtetett kapcsoló, és mint minden más kapcsoló, ez is be- és kikapcsolható, átengedi az áramot vagy sem. Alacsony feszültséggel vezérelhető, mint például az ESP32 GPIO-k által biztosított 3,3 V, és lehetővé teszi a magas feszültségek, például 12 V, 24 V vagy hálózati feszültség (Európában 230 V és az Egyesült Államokban 120 V) vezérlését.

A bal oldalon két készlet három csatlakozó található a nagyfeszültségek csatlakoztatásához, a jobb oldali érintkezők (alacsony feszültség) pedig az ESP32 GPIO-khoz csatlakoznak.

Hálózati feszültség csatlakozások

Az előző képen látható relémodul két csatlakozóval rendelkezik, mindegyik három aljzattal: közös (COM), normál zárt (NC) és normál nyitott (NO).

- **COM:** csatlakoztassa a szabályozni kívánt áramot (hálózati feszültség).
- **NC (Normally Closed):** az alaphelyzetben zárt konfigurációt használja, ha azt szeretné, hogy a relé alapértelmezés szerint zárva legyen. Az NC COM érintkezők csatlakoztatva vannak, ami azt jelenti, hogy az áram folyik, hacsak nem küld jelet az ESP32-től a relémodulhoz, hogy megnyissa az áramkört és leállítsa az áram folyását.
- **NO (Normally Open):** a normál nyitott konfiguráció fordítva működik: nincs kapcsolat a NO és a COM érintkezők között, így az áramkör meg van szakítva, hacsak nem küld jelet az ESP32-től az áramkör lezárására.

Vezérlőcsapok

A kisfeszültségű oldalon egy négy érintkezőből és egy három érintkezőből álló készlet található. Az első készlet a VCC-ből és a GND-ből áll a modul bekapcsolásához, valamint az 1-es bemenetből (IN1) és a 2-es bemenetből (IN2) az alsó és felső relék vezérléséhez.

Ha a relémodulnak csak egy csatornája van, akkor csak egy IN érintkezője lesz. Ha négy csatornája van, akkor négy bemeneti tűje lesz, és így tovább.

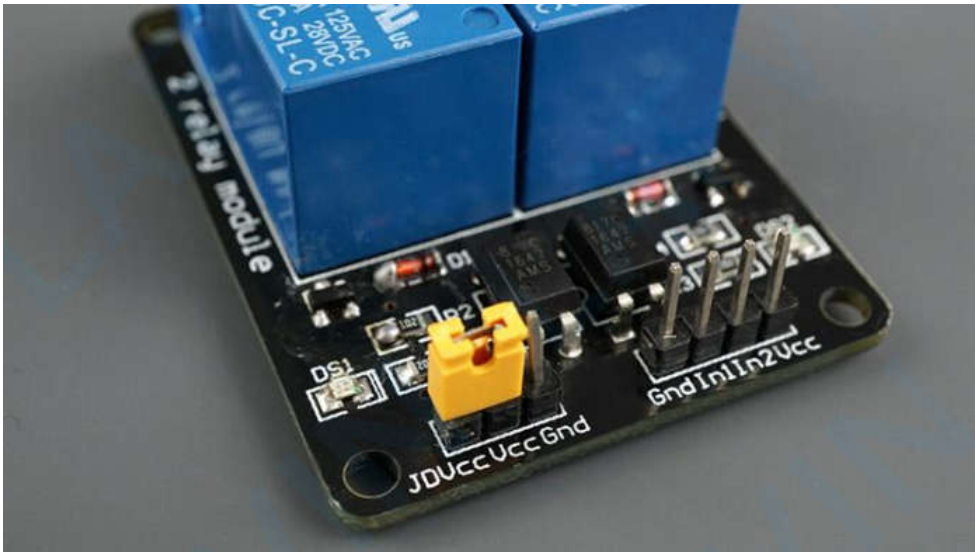
Az IN érintkezőkre küldött jel határozza meg, hogy a relé aktív-e vagy sem. A relé akkor aktiválódik, ha a bemenet kb. 2V alá esik. Ez azt jelenti, hogy a következő forgatókönyvek lesznek:

- Normálisan zárt konfiguráció (NC):
    - HIGH jel – áram folyik
    - LOW jel – az áram nem folyik
- Normálisan nyitott konfiguráció (NO):
    - HIGH jel – az áram nem folyik
    - LOW jel – áram folyik

Normálisan zárt konfigurációt kell használni, amikor az áramnak legtöbbször folynia kell, és csak időnként szeretnéd leállítani.

Használj normál nyitott konfigurációt, ha azt szeretnéd, hogy az áram időnként folyjon (például időnként kapcsoljon fel egy lámpát).

Tápegység kiválasztása



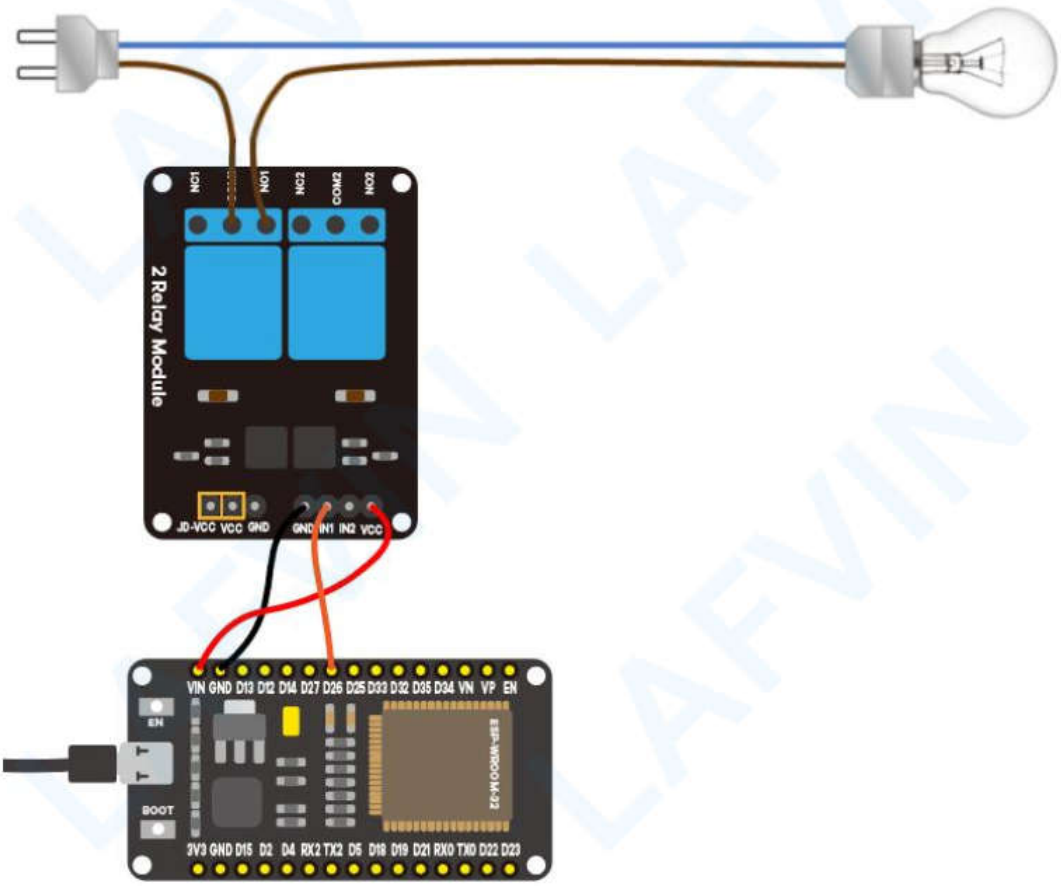
A második érintkezőkészlet GND, VCC és JD-VCC érintkezőkből áll.

A JD-VCC érintkező táplálja a relé elektromágnesét. Figyeld meg, hogy a modul egy áthidaló jumperrel rendelkezik, amely összeköti a VCC és a JD-VCC érintkezőket; az itt látható sárga, de lehet, hogy a tiéd más színű.

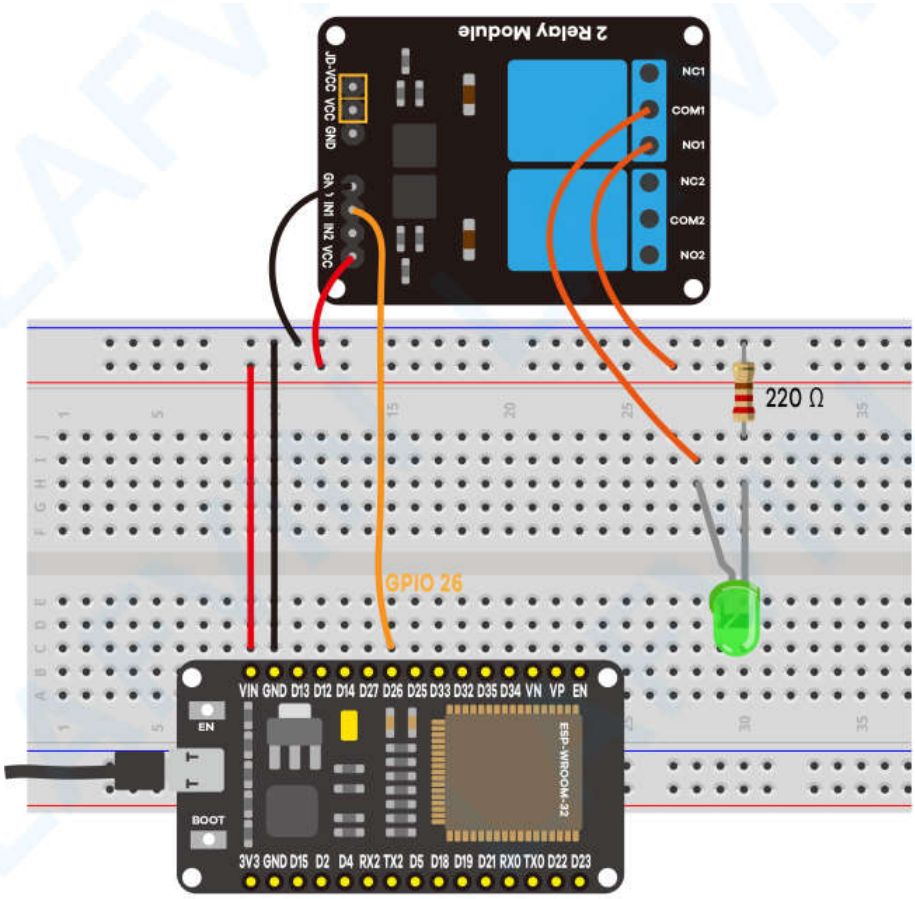
Az áthidaló sapkával a VCC és a JD-VCC érintkezők csatlakoztatva vannak. Ez azt jelenti, hogy a relé elektromágnes közvetlenül az ESP32 tápérintkezőjéről táplálkozik, így a relémodul és az ESP32 áramkörök nincsenek fizikailag elszigetelve egymástól.

Az áthidaló jumper nélkül független áramforrást kell biztosítania a relé elektromágnesének a JD-VCC tűn keresztül történő bekapcsolásához. Ez a konfiguráció fizikailag leválasztja a reléket az ESP32-ről a modul beépített optocsatolójával, amely megakadályozza az ESP32 károsodását elektromos tüskék esetén.

Vázlat



Figyelmeztetés: A nagyfeszültségű tápegységek használata súlyos sérüléseket okozhat. Ezért a kísérletben a nagy tápfeszültségű izzók helyett 5 mm-es LED-eket használnak. Ha nem ismeri a hálózati feszültséget, kérjen meg valakit, aki segít. Az ESP programozása vagy az áramkör bekötése közben győződjön meg arról, hogy minden le van választva a hálózati feszültségről.



Az ESP32 könyvtárának telepítése

A webszerver felépítéséhez az ESPAsyncWebServer könyvtárat és az AsyncTCP könyvtárat használjuk.

Az ESPAsyncWebServer könyvtár telepítése

Kövesd a következő lépéseket az ESPAsyncWebServer könyvtár telepítéséhez:

1. Kattints ide az ESPAsyncWebServer könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kell kapnod egy ESPAsyncWebServer-master mappát.

3. Nevezd át a mappát ~~ESPAsyncWebServer-master~~ -ről ESPAsyncWebServer névre.

4. Helyezd át az ESPAsyncWebServer mappát az Arduino IDE telepítési könyvtárak libraries mappájába.

Alternatív megoldásként az Arduino IDE-ben megnyithatja a **Vázlat > Könyvtár tartalmazása > .ZIP könyvtár hozzáadása…** menüpontot, és kiválaszthatja az éppen letöltött könyvtárat.

Az ESP32 AsyncTCP könyvtárának telepítése

Az ESPAsyncWebServer könyvtár működéséhez az AsyncTCP könyvtárra van szükség. Kövesd a következő lépéseket a könyvtár telepítéséhez:

1. Kattints ide az AsyncTCP könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kell kapnod egy AsyncTCP-master mappát.

3. Nevezze át a mappát ~~AsyncTCP-master~~-ről AsyncTCP-re.

4. Helyezd át az AsyncTCP mappát az Arduino IDE telepítési könyvtárak libraries mappájába.

5. Végül nyisd meg újra az Arduino IDE-t.

Alternatív megoldásként az Arduino IDE-ben megnyithatja a **Vázlat > Könyvtár tartalmazása > .ZIP könyvtár hozzáadása…** menüpontot, és kiválaszthatja az éppen letöltött könyvtárat.

Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

A szükséges könyvtárak telepítése után nyisd meg a project_7_ESP32_Relay_Web_Server.ino kódot az arduino IDE-ben.

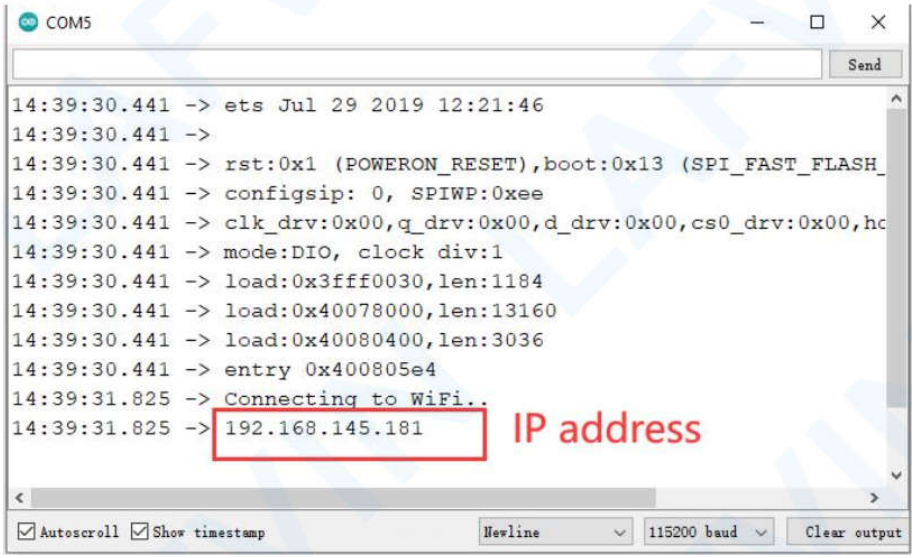
A kód feltöltése előtt ne felejtsd el beírni a hálózati hitelesítő adatait, hogy az ESP csatlakozhasson a helyi hálózathoz.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Demonstráció

A szükséges módosítások elvégzése után töltse fel a kódot az ESP32-re. Emlékeztető: Töltsd fel a tesztkódot

Nyisd meg a soros monitort 115200 adatátviteli sebességgel, és nyomd meg az ESP32 EN gombot az IP-cím megszerzéséhez. Ezután nyisd meg a böngészőt a helyi hálózaton, és írd be az ESP32 IP-címét, hogy hozzáférj a webszerverhez.



Megjegyzés: A böngészőnek és az ESP32-nek ugyanahhoz a helyi hálózathoz kell csatlakoznia.

A kódban megadott relék számának megfelelően két gombbal a következőket kell kapnia.

Mostantól a gombokkal vezérelheted a reléket az okostelefonoddal.

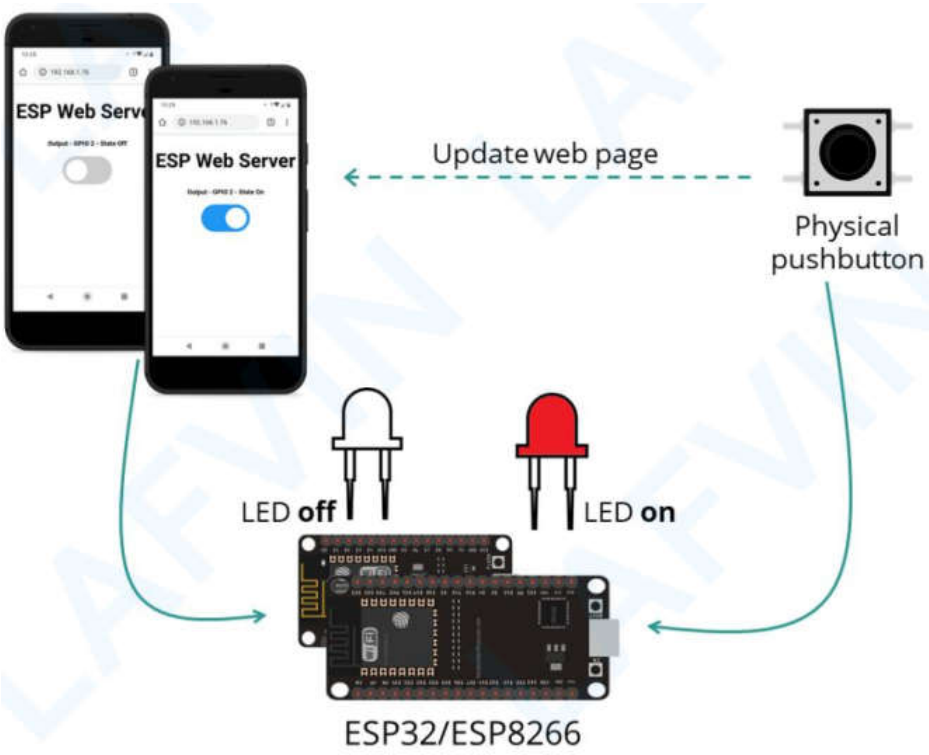

14:43
192.168.145.181
ESP Web Server
Relay #1 - GPIO 26
Relay #2 - GPIO 27

8. projekt Kimeneti állapot szinkronizálása Webszerverrel

Ez a projekt bemutatja, hogyan lehet vezérelni az ESP32 vagy ESP8266 kimeneteket egy webszerver és egy fizikai gomb egyidejű használatával. A kimenet állapota frissül a weboldalon, akár fizikai gombon, akár webszerveren keresztül változtatja meg.

A projekt áttekintése

Vessünk egy pillantást a projekt működésére.



- Az ESP32 vagy ESP8266 egy webszervert tartalmaz, amely lehetővé teszi a kimenet állapotának szabályozását;
- Az aktuális kimeneti állapot megjelenik a webszerveren;
- Az ESP egy fizikai nyomógombhoz is kapcsolódik, amely ugyanazt a kimenetet vezérli;
- Ha a kimeneti állapotot a fizikai nyomógombbal módosítja, akkor az aktuális állapota is frissül a webszerveren.

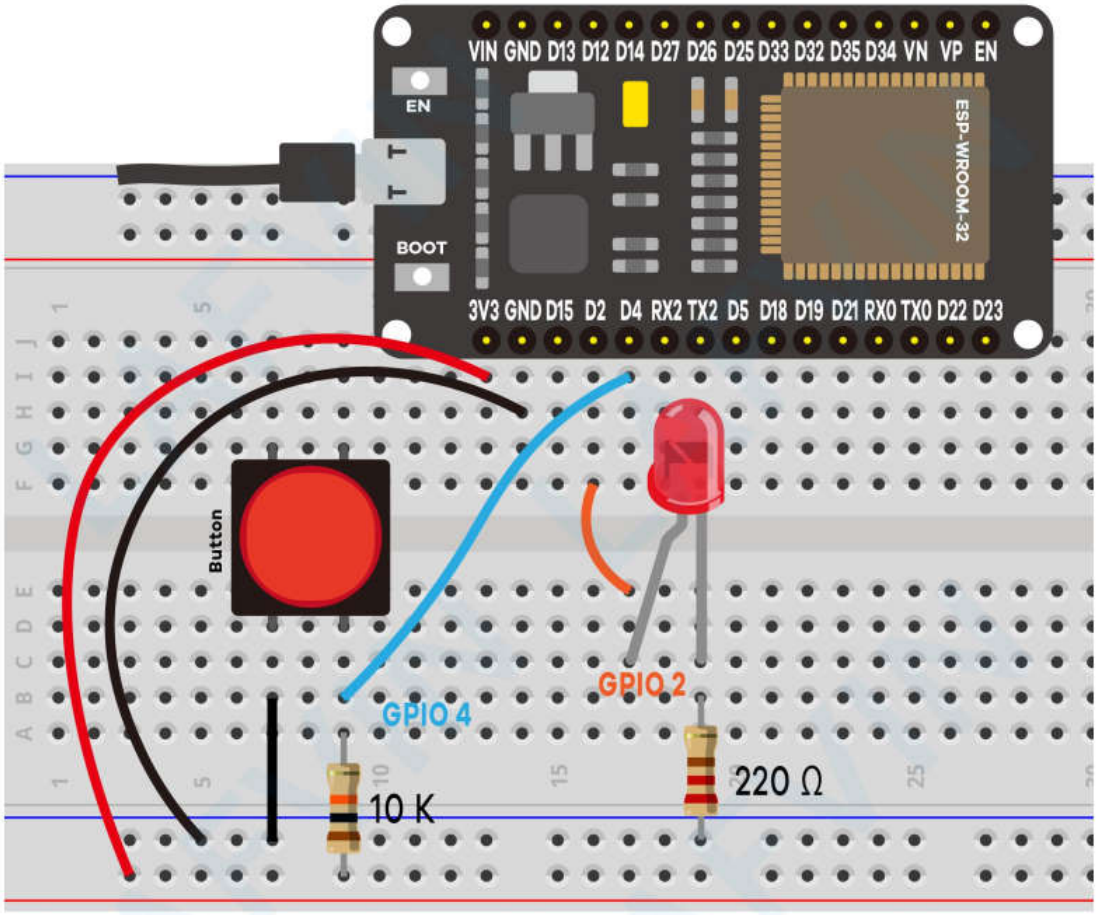
Összefoglalva, ez a projekt lehetővé teszi ugyanazt a kimenetet egy webszerver és egy nyomógomb egyidejű vezérlésével. Amikor a kimeneti állapot megváltozik, a webszerver frissül.

Szükséges alkatrészek

Ehhez az oktatóanyaghoz a következő alkatrészekre lesz szükséged:

- ESP32 DEVKIT V1 Kártya

- 5mm LED
- 220 Ohmos ellenállás
- Nyomógomb
- 10 kOhmos ellenállás
- Bekötőkártya
- Jumper vezetékek



Az ESP32 könyvtárának telepítése

A webszerver felépítéséhez az ESPAsyncWebServer könyvtárat és az AsyncTCP könyvtárat használjuk.

Az ESPAsyncWebServer könyvtár telepítése

Kövesd a következő lépéseket az ESPAsyncWebServer könyvtár telepítéséhez:

1. Kattints ide az ESPAsyncWebServer könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kell kapnod egy ESPAsyncWebServer-master mappát.

3. Nevezd át a mappát ~~ESPAsyncWebServer-master~~-ről ESPAsyncWebServer névre.

4. Helyezd át az ESPAsyncWebServer mappát az Arduino IDE telepítési könyvtárak libraries mappájába.

Alternatív megoldásként az Arduino IDE-ben megnyithatja a **Vázlat > Könyvtár tartalmazása > .ZIP könyvtár hozzáadása…** menüpontot, és kiválaszthatja az éppen letöltött könyvtárat.

Az ESP32 AsyncTCP könyvtárának telepítése

Az ESPAsyncWebServer könyvtár működéséhez az AsyncTCP könyvtárra van szükség. Kövesd a következő lépéseket a könyvtár telepítéséhez:

1. Kattints ide az AsyncTCP könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kell kapnod egy AsyncTCP-master mappát.

3. Nevezze át a mappát ~~AsyncTCP-master~~-ről AsyncTCP-re.

4. Helyezd át az AsyncTCP mappát az Arduino IDE telepítési könyvtárak libraries mappájába.

5. Végül nyisd meg újra az Arduino IDE-t.

Alternatív megoldásként az Arduino IDE-ben megnyithatja a **Vázlat > Könyvtár tartalmazása > .ZIP könyvtár hozzáadása…** menüpontot, és kiválaszthatja az éppen letöltött könyvtárat.

Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

A szükséges könyvtárak telepítése után nyisd meg a project_8_Output_State_Synchronization_Web_Server.ino kódot az arduino IDE-ben.

A kód feltöltése előtt ne felejtsd el beírni a hálózati hitelesítő adatait, hogy az ESP csatlakozhasson a helyi hálózathoz.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Hogyan működik a kód

A gomb állapota és a kimeneti állapota

A `ledState` változó tárolja a LED kimeneti állapotát. Alapértelmezés szerint a webszerver elindulásakor LOW.

```
int ledState = LOW; // the current state of the output pin
```

A `buttonState` és `lastButtonState` segítségével megállapítható, hogy a nyomógombot megnyomták-e vagy sem.

```
int buttonState; // the current reading from the input pin
int lastButtonState = HIGH; // the previous reading from the inputpin
```

Gomb (webszerver)

Nem adtuk meg a HTML-kódot az `index_html` változó gombjának létrehozásához. Ez azért van így, mert azt szeretnénk, hogy a LED aktuális állapotától függően módosíthassuk, amit a nyomógombbal is módosíthatunk.

Létrehoztunk tehát egy helyőrzőt a %BUTTONPLACEHOLDER% gombhoz, amelyet HTML-szövegre cserélünk, hogy később létrehozzuk a gombot a kódban (ez a `processor()` függvényben történik).

```
<h2>ESP Web Server</h2>
%BUTTONPLACEHOLDER%
```

processor()

A `processor()` függvény a HTML-szöveg helyőrzőit tényleges értékekkel helyettesíti. Először is ellenőrzi, hogy a HTML-szövegekben van-e helyőrző `%BUTTONPLACEHOLDER%`.

```
if(var == "BUTTONPLACEHOLDER"){
```

Ezután hívja meg az `outputState()` függvényt, amely visszaadja az aktuális kimeneti állapotot. Mentjük az `outputStateValue` változóba.

```
String outputStateValue = outputState();
```

Ezt követően használja ezt az értéket a HTML-szöveg létrehozásához, hogy a megfelelő állapotú gomb jelenjen meg:

```
buttons+= "<h4>Output - GPIO 2 - State <span
id=\"outputState\"><span></h4><label class=\"switch\"><input
type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"output\" "
+ outputStateValue + "><span class=\"slider\"></span></label>";
```

HTTP GET kérés a kimeneti állapot megváltoztatására (JavaScript)

A gomb megnyomásakor a `toggleCheckbox()` függvény meghívódik. Ez a funkció különböző URL-eken kéri a LED be- vagy kikapcsolását.

```
function toggleCheckbox(element) {
    var xhr = new XMLHttpRequest();
    if(element.checked){ xhr.open("GET", "/update?state=1", true); }
    else { xhr.open("GET", "/update?state=0", true); }
    xhr.send();}
```

A LED bekapcsolásához kérést küld az **/update?state=1** URL-címen:

```
if(element.checked){ xhr.open("GET", "/update?state=1", true); }
```

Ellenkező esetben kérést küld az **/update?state=0** URL-re.

HTTP GET kérés frissítési állapothoz (JavaScript)

A webszerver kimeneti állapotának frissítése érdekében a következő függvényt hívjuk meg, amely másodpercenként új kérést küld a **/state** URL-re.

```
setInterval(function ( ) {
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
var inputChecked;
var outputStateM;
if( this.responseText == 1){
inputChecked = true;
outputStateM = "On";
} else {
inputChecked = false;
outputStateM = "Off";
} document.getElementById("output").checked = inputChecked;
document.getElementById("outputState").innerHTML =
outputStateM;
}
};
xhttp.open("GET", "/state", true);
xhttp.send();}, 1000 ) ;
```

A kérések kezelése

Ezután kezelnünk kell, mi történik, amikor az ESP32 vagy ESP8266 kéréseket kap ezeken az URL-eken.

Amikor egy kérés érkezik a gyökér / URL címre, elküldjük a HTML oldalt, valamint a processzort.

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
request->send_P(200, "text/html", index_html, processor);});
```

A következő sorok ellenőrzik, hogy kapott-e kérést a következő napon: az **/update?state=1** vagy **/update?state=0** URL és a változik a `ledState` ennek megfelelően.

```
server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request)
{
String inputMessage;
String inputParam;
// GET input1 value on <ESP_IP>/update?state=<inputMessage>
if (request->hasParam(PARAM_INPUT_1)) {
```

```
inputMessage = request->getParam(PARAM_INPUT_1)->value();
inputParam = PARAM_INPUT_1;
digitalWrite(output, inputMessage.toInt());
ledState = !ledState;
} else {
inputMessage = "No message sent";
inputParam = "none";
} Serial.println(inputMessage);
request->send(200, "text/plain", "OK");});
```

Amikor egy kérés érkezik a **/state** URL-re, elküldjük az aktuális kimeneti állapotot:

```
server.on("/state", HTTP_GET, [] (AsyncWebServerRequest *request) {
request->send(200, "text/plain",
String(digitalRead(output)).c_str());});
```

loop()

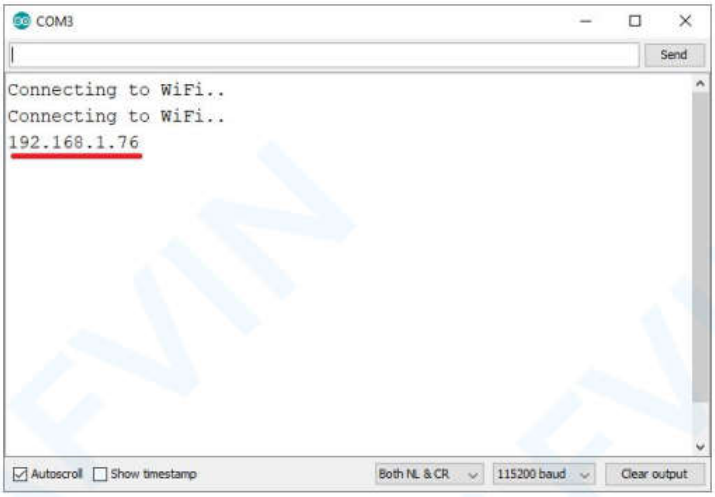
A `loop()`-ban visszafordítjuk a nyomógombot, és a `ledState` változó értékétől függően be- vagy kikapcsoljuk a LED-et.

```
digitalWrite(output, ledState);
```

Demonstráció

Töltsd fel a kódot az ESP32-re. Emlékeztető: Töltsd fel a tesztkódot

Ezután nyisd meg a soros monitort 115200 adatátviteli sebességgel. Nyomd meg a fedélzeti EN/RST gombot az IP-cím megszerzéséhez.



Nyiss meg egy böngészőt a helyi hálózaton, és írd be az ESP IP-címét. Az alábbiak szerint hozzá kell férj a webszerverhez.

Megjegyzés: A böngészőnek és az ESP32-nek ugyanahhoz a helyi hálózathoz kell csatlakoznia.

A webszerveren lévő gomb megnyomásával bekapcsolhatod a LED-et.

Ugyanezt a LED-et a fizikai nyomógombbal is vezérelheted. Állapota mindig automatikusan frissül a webszerveren.

9. projekt ESP32 DHT11 webszerver

Ebben a projektben megtudhatod, hogyan építhetsz aszinkron ESP32 webszervert a DHT11 segítségével, amely megjeleníti a hőmérsékletet és a páratartalmat az Arduino IDE segítségével.

Előfeltételek

Az általunk elkészített webszerver automatikusan frissíti a leolvasásokat, anélkül, hogy frissíteni kellene a weboldalt.

Ezzel a projekttel megtudhatod:

- Hogyan lehet leolvasni a hőmérsékletet és a páratartalmat a DHT-érzékelőkről;
- Létrehozunk egy aszinkron webszervert az ESPAsyncWebServer könyvtár használatával;
- Frissítse az érzékelő leolvasásait automatikusan anélkül, hogy frissítened kellene a weboldalt.

Aszinkron webszerver

A webszerver felépítéséhez az ESPAsyncWebServer könyvtárat használjuk, amely egyszerű módot biztosít egy aszinkron webszerver felépítésére. Az aszinkron webszerver felépítésének számos előnye van, amint azt a könyvtár GitHub oldalán említettük, például:

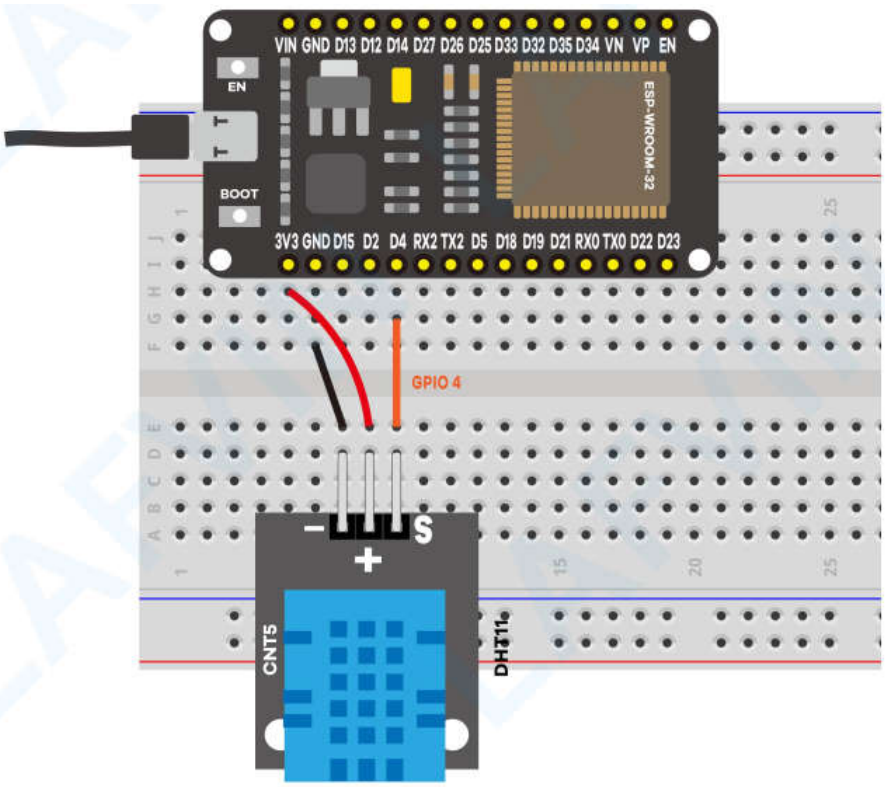
- „Egynél több kapcsolat kezelése egyszerre”;
- „A válasz elküldésekor azonnal készen áll a többi kapcsolat kezelésére, miközben a szerver a háttérben gondoskodik a válasz elküldéséről”;
- „Egyszerű sablonfeldolgozó motor a sablonok kezelésére”;

Szükséges alkatrészek

Ehhez az oktatóanyaghoz a következő alkatrészekre lesz szükséged:

- ESP32 DEVKIT V1 Kártya
- DHT11 modul
- Bekötőkártya
- Jumper vezetékek

Vázlat



Könyvtárak telepítése

Ehhez a projekthez telepítened kell néhány könyvtárat:

- A DHT és az Adafruit Unified Sensor Driver könyvtárak a DHT érzékelőből való olvasáshoz.
- ESPAsyncWebServer és Async TCP könyvtárak az aszinkron webszerver felépítéséhez.

Kövesd a következő utasításokat a könyvtárak telepítéséhez:

A DHT Sensor Library telepítése

Ha az Arduino IDE használatával szeretnél olvasni a DHT-érzékelőből, telepítened kell a DHT-sensor könyvtárát. Kövesd a következő lépéseket a könyvtár telepítéséhez.

1. Kattints ide a DHT Sensor könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kapsz egy DHT-sensor-library-master mappát.

3. Nevezd át a mappát ~~DHT-sensor-library-master~~-ről DHT_sensor-ra.

4. Helyezd át a DHT_sensor mappát az Arduino IDE telepítési könyvtárak mappájába.

5. Végül nyisd meg újra az Arduino IDE-t.

Az Adafruit Unified Sensor Driver telepítése

A DHT érzékelővel való együttműködéshez telepítened kell az Adafruit Unified Sensor Driver könyvtárat is. Kövesd a következő lépéseket a könyvtár telepítéséhez.

1. Kattints ide az Adafruit Unified Sensor könyvtárának letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kapsz egy Adafruit_sensor-master mappát.

3. Nevezd át a mappát ~~Adafruit_sensor-master~~-ről Adafruit_sensor-ra.

4. Helyezd át az Adafruit_sensor mappát az Arduino IDE telepítési könyvtárak mappájába.

5. Végül nyisd meg újra az Arduino IDE-t.

Az ESPAsyncWebServer könyvtár telepítése

Kövesd a következő lépéseket az ESPAsyncWebServer könyvtár telepítéséhez:

1. Kattints ide az ESPAsyncWebServer könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kell kapnod egy ESPAsyncWebServer-master mappát.

3. Nevezd át a mappát ~~ESPAsyncWebServer-master~~-ről ESPAsyncWebServer névre.

4. Helyezd át az ESPAsyncWebServer mappát az Arduino IDE telepítési könyvtárak libraries mappájába.

Az ESP AsyncTCP könyvtárának telepítése

Az ESPAsyncWebServer könyvtár működéséhez az AsyncTCP könyvtárra van szükség. Kövesd a következő lépéseket a könyvtár telepítéséhez:

1. Kattints ide az AsyncTCP könyvtár letöltéséhez. A Letöltések mappában kell lennie egy .zip állománynak.

2. Csomagold ki a .zip állományt, és kell kapnod egy AsyncTCP-master mappát.

3. Nevezze át a mappát ~~AsyncTCP-master~~-ről AsyncTCP-re.

4. Helyezd át az AsyncTCP mappát az Arduino IDE telepítési könyvtárak libraries mappájába.

5. Végül nyisd meg újra az Arduino IDE-t.

Kód

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

A szükséges könyvtárak telepítése után nyisd meg a project_9_ESP32_DHT11_Web_Server.ino kódot az arduino IDE-ben.

A kód feltöltése előtt ne felejtsd el beírni a hálózati hitelesítő adatait, hogy az ESP csatlakozhasson a helyi hálózathoz.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Hogyan működik a kód

A következő bekezdésekben elmagyarázzuk, hogyan működik a kód. Folytasd az olvasást, ha többet szeretnél megtudni, vagy ugorj a Demonstráció részre a végeredmény megtekintéséhez.

Könyvtárak importálása

Először importáljuk a szükséges könyvtárakat. A webszerver felépítéséhez a WiFi, az ESPAsyncWebServer és az ESPAsyncTCP szükséges. Az Adafruit_Sensor és a DHT-sensor könyvtárak szükségesek a DHT11 vagy DHT22 érzékelők olvasásához.

```
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include <ESPAsyncTCP.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

A változók definiálása

Meghatározzuk azt a GPIO-t, amelyhez a DHT adattű csatlakozik. Ebben az esetben a GPIO 4-hez csatlakozik.

```
#define DHTPIN 4 // Digital pin connected to the DHT sensor
```

Ezután válasszuk ki a használt DHT-érzékelő típusát. Példánkban a DHT11-t használjuk. Ha más típust használsz, csak törölnöd kell az érzékelő megjegyzését, és megjegyzést kell fűznie az összes többihez.

```
#define DHTTYPE DHT11 // DHT 11
```

Példányosítsunk egy DHT-objektumot a korábban meghatározott típussal és tűvel.

```
DHT dht(DHTPIN, DHTTYPE);
```

Hozzunk létre egy AsyncWebServer objektumot a 80-as porton.

```
AsyncWebServer server(80);
```

Olvassuk a hőmérséklet- és páratartalom-függvényeket

Két függvényt hoztunk létre: az egyiket a hőmérséklet leolvasására (`readDHTTemperature()`), a másikat pedig a páratartalom olvasására (`readDHTHumidity()`).

```
String readDHTTemperature() {
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  //float t = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}

String readDHTHumidity() {
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return "--";
  }
  else {
    Serial.println(h);
    return String(h);
  }
}
```

Az érzékelő leolvasása olyan egyszerű, mint a `readTemperature()` és `readHumidity()` metódusok használata a `dht` objektumon.

```
float t = dht.readTemperature();
float h = dht.readHumidity();
```

Van egy olyan feltételünk is, amely két kötőjelet (–) ad vissza arra az esetre, ha az érzékelő nem kapja meg a leolvasást.

```
if (isnan(t)) {
Serial.println("Failed to read from DHT sensor!");
return "--";}
```

A leolvasások karakterlánc-típusként kerülnek visszaadásra. A float karakterlánccá alakításához használjuk a `String()` függvényt.

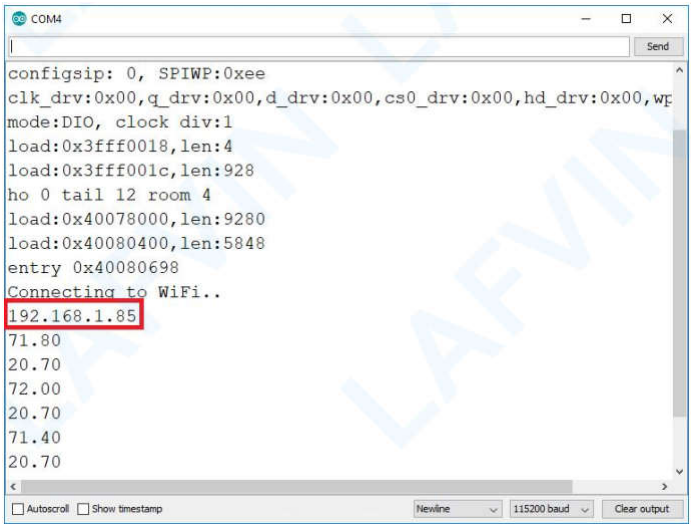
```
return String(t);
```

Alapértelmezés szerint a hőmérsékletet Celsius-fokban mérjük. Ha a hőmérsékletet Fahrenheit-fokban szeretnéd megkapni, írd megjegyzésbe a hőmérsékletet Celsius-ban, és törölje a Fahrenheit-fokban, így a következőket kapja:

```
float t = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
//float t = dht.readTemperature(true);
```

Most töltd fel a kódot az ESP32-re. Győződj meg arról, hogy a megfelelő kártyát és COM-portot választottad ki. Emlékeztető: Töltsd fel a tesztkódot

Feltöltés után nyisd meg a soros monitort 115200 adatátviteli sebességgel. Nyomd meg az ESP32 reset gombot. Az ESP32 IP-címe ki lesz nyomtatva a soros monitorra.



Demonstráció

Nyisson meg egy böngészőt, és írja be az ESP32 IP-címét. A webszervernek a legfrissebb szenzorértékeket kell megjelenítenie.

Megjegyzés: A böngészőnek és az ESP32-nek ugyanahhoz a helyi hálózathoz kell csatlakoznia.

Figyeld meg, hogy a hőmérséklet- és páratartalom-értékek automatikusan frissülnek anélkül, hogy frissíteni kellene a weboldalt.

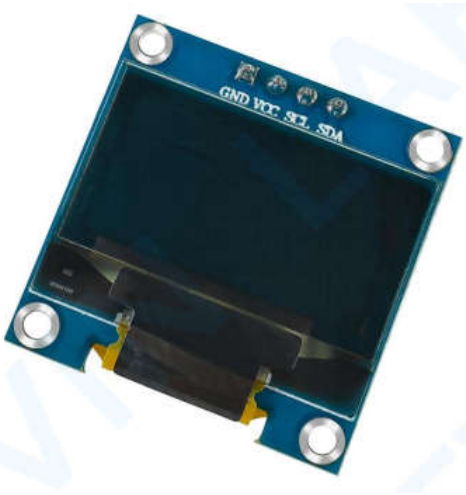

5:16
192.168.1.85
ESP32 DHT
Server
Temperature 21.40 °C
Humidity 63.90 %

10. projekt ESP32 OLED kijelző

Ez a projekt bemutatja, hogyan kell használni a 0,96 hüvelykes SSD1306 OLED kijelzőt ESP32-vel Arduino IDE használatával.

Bemutatjuk a 0,96 hüvelykes OLED kijelzőt

Az OLED-kijelző, amelyet ebben az oktatóanyagban használunk, az SSD1306 modell: egyszínű, 0,96 hüvelykes kijelző 128 × 64 pixeles, amint az a következő ábrán látható.

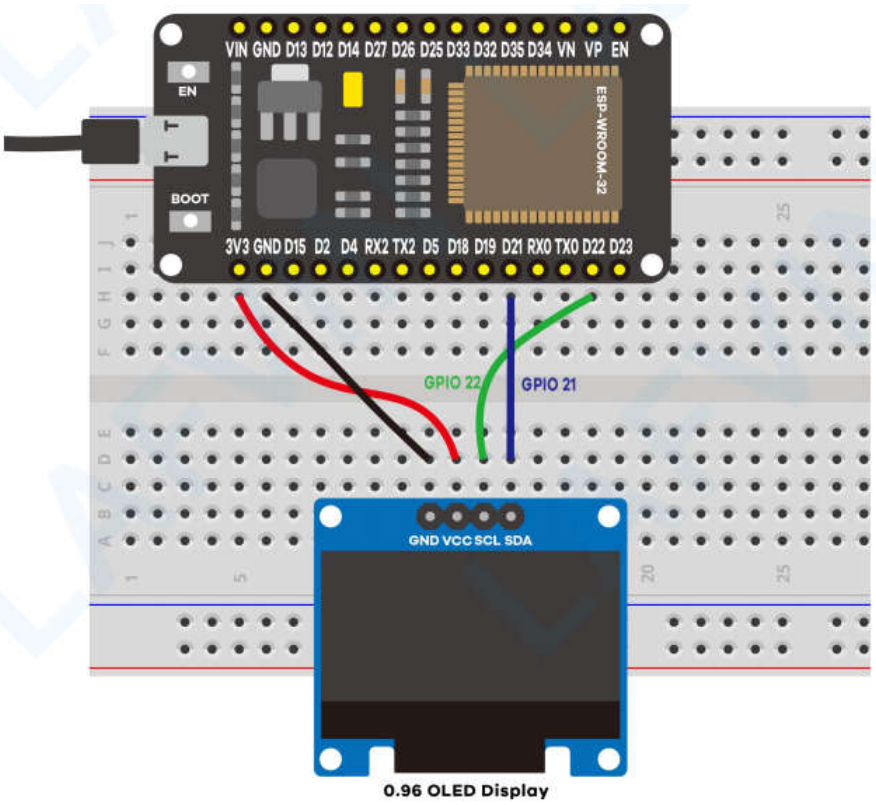


Az OLED kijelző nem igényel háttérvilágítást, ami nagyon szép kontrasztot eredményez sötét környezetben. Ráadásul a pixelei csak bekapcsolt állapotban fogyasztanak energiát, így az OLED-kijelző kevesebb energiát fogyaszt, mint más kijelzők.

Mivel az OLED kijelző I2C kommunikációs protokollt használ, a vezetékezés nagyon egyszerű. Az alábbi táblázatot referenciaként használhatod.

OLED tű	ESP32
Vin	3,3 V
GND	GND
SCL	GPIO 22
SDA	GPIO 21

Vázlat



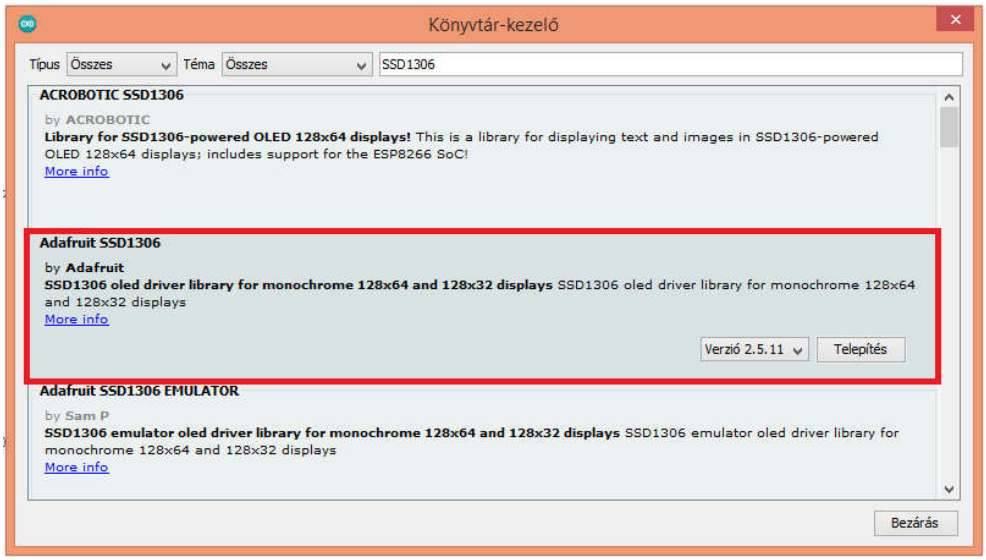
SSD1306 OLED könyvtár telepítése – ESP32

Számos könyvtár áll rendelkezésre az OLED-kijelző ESP32 segítségével történő vezérléséhez. Ebben az oktatóanyagban két Adafruit könyvtárat fogunk használni: Adafruit_SSD1306 könyvtárat és Adafruit_GFX könyvtárat.

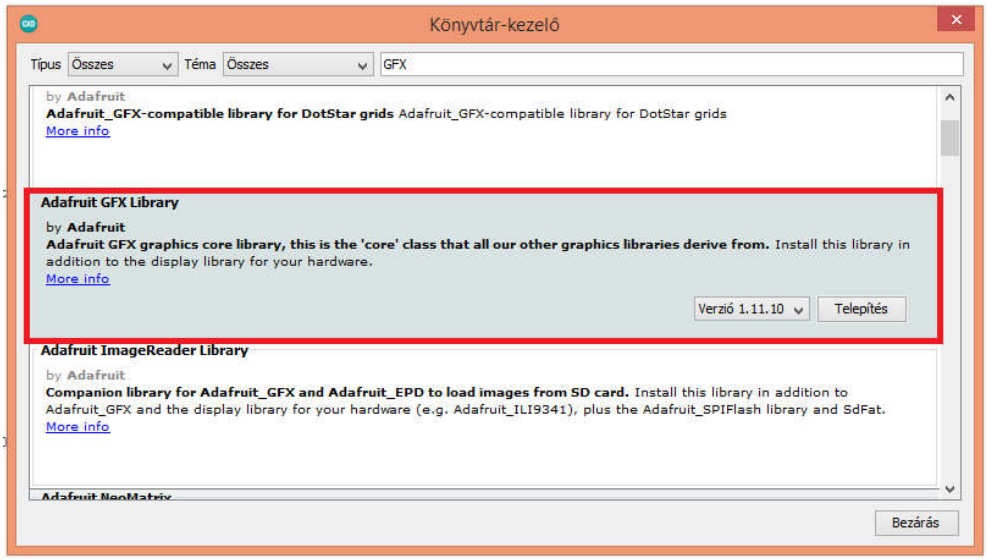
Kövesd a következő lépéseket a könyvtárak telepítéséhez.

1. Nyisd meg az Arduino IDE-jét, és lépj a **Vázlat > Könyvtár tartalmazása > Könyvtárak kezelése** menüpontra. A Könyvtárkezelőnek meg kell nyílnia.

2. Írd be az „**SSD1306**” kifejezést a keresőmezőbe, és telepítsd az SSD1306 könyvtárat az Adafruittól.



3. Miután telepítetted az SSD1306 könyvtárat az Adafruitból, írd be a „GFX” kifejezést a keresőmezőbe, és telepítsd a könyvtárat.



4. A könyvtárak telepítése után indítsd újra az Arduino IDE-t.

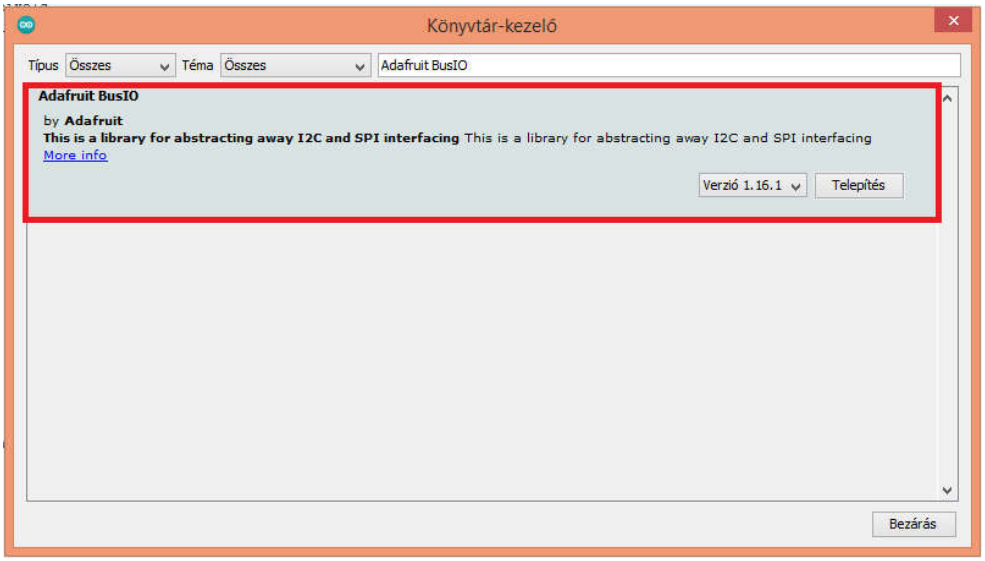
Adafruit_BusIO könyvtár telepítése (a fordító hozzáfűzése)

Nekem a fordításkor az Arduino IDE **hibaüzenetet** küldött, mégpedig, hogy nem találja az Adafruit_I2CDevice.h fájlt. Ezért telepítenem kellett az Adafruit_BusIO könyvtárat is.

Ha nálad is ez a hibaüzenet jelent meg, kövesd a következő lépéseket a könyvtár telepítéséhez, egyébként ez a lépés kihagyható.

1. Nyisd meg az Arduino IDE-jét, és lépj a **Vázlat > Könyvtár tartalmazása > Könyvtárak kezelése** menüpontra. A Könyvtárkezelőnek meg kell nyílnia.

2. Írd be az „**Adafruit BusIO**” kifejezést a keresőmezőbe, és telepítsd a könyvtárat.



3. A könyvtár telepítése után indítsd újra az Arduino IDE-t.

Kód

A szükséges könyvtárak telepítése után nyisd meg a Project_10_ESP32_OLED_Display.ino kódot az arduino IDE-ben.

Az ESP32-t Arduino IDE segítségével programozzuk, ezért a folytatás előtt győződj meg arról, hogy telepítve van az ESP32 kiegészítő: (Ha már megtetted ezt a lépést, ugorhatsz a következő lépésre.)

- Lásd: Az ESP32 bővítmény telepítése az Arduino IDE-ben

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
-1);void setup() {
Serial.begin(115200);
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D
for 128x64
Serial.println(F("SSD1306 allocation failed"));
for(;;);
} delay(2000);
display.clearDisplay();
display.setTextSize(2);
display.setTextColor(WHITE);
display.setCursor(0, 30);
// Display static text
display.println("LAFVIN");
display.display();
```

```
delay(100);
}
void loop() {
// Scroll in various directions, pausing in-between:
display.startscrollright(0x00, 0x0F);
delay(7000);
display.stopscroll();
delay(1000);
display.startscrollleft(0x00, 0x0F);
delay(7000);
display.stopscroll();
delay(1000);
}
```

Hogyan működik a kód

Könyvtárak importálása

Először is importálni kell a szükséges könyvtárakat. A Wire könyvtár az I2C használatához és az Adafruit könyvtárak a kijelzőre való íráshoz: Adafruit_GFX és Adafruit_SSD1306.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Inicializáljuk az OLED kijelzőt

Ezután meghatározzuk az OLED szélességét és magasságát. Ebben a példában 128 × 64-es OLED-kijelzőt használunk. Ha más méretet használsz, ezt módosíthatod a SCREEN_WIDTH és a SCREEN_HEIGHT változókban.

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

Ezután inicializálunk egy megjelenítési objektumot az I2C kommunikációs protokollal (`&Wire`) korábban meghatározott szélességgel és magassággal.

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

A (-1) paraméter azt jelenti, hogy az OLED-kijelzőn nincs RESET tű. Ha az OLED-kijelzőnek van RESET tűje, akkor azt egy GPIO-hoz kell csatlakoztatni. Ebben az esetben paraméterként add meg a GPIO-számot.

A `setup()`-ban hibakeresés céljából inicializáljuk a soros monitort 115200 átviteli sebességgel.

```
Serial.begin(115200);
```

Inicializáljuk az OLED-kijelzőt a `begin()` metódussal az alábbiak szerint:

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
Serial.println("SSD1306 allocation failed");
for(;;); // Don't proceed, loop forever}
```

Ez a részlet a soros monitoron is kinyomtat egy üzenetet arra az esetre, ha nem tudnánk csatlakozni a kijelzőhöz.

```
Serial.println("SSD1306 allocation failed");
```

Ha másik OLED-kijelzőt használsz, előfordulhat, hogy módosítanod kell az OLED-címet. Esetünkben a cím 0x3C.

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
```

A kijelző inicializálása után adjunk hozzá két másodperces késleltetést, hogy az OLED-nek elegendő ideje legyen inicializálódni, mielőtt szöveget írna:

```
delay(2000);
```

Töröljük a kijelzőt, beállítjuk a betűméret, a színt és kiírjuk a szöveget

A kijelző inicializálása után töröljük a puffert a következővel: a `clearDisplay()` metódussal:

```
display.clearDisplay();
```

Szöveg írása előtt be kell állítanunk a szöveg méretét, színét és azt, hogy a szöveg hol jelenjen meg az OLED-en.

Beállítjuk a betűméretet a `setTextSize()` metódussal:

```
display.setTextSize(1);
```

Beállítjuk a betűszínt a `setTextColor()` metódussal:

```
display.setTextColor(WHITE);
```

A WHITE fehér betűtípust és fekete hátteret állít be.

Határozzuk meg a szöveg kezdőpontját a `setCursor(x,y)` metódussal. Ebben az esetben a szöveget úgy állítjuk be, hogy a (0,0) koordinátákkal kezdődjön – a bal felső sarokban.

```
display.setCursor(0,0);
```

Végül elküldhetjük a szöveget a kijelzőre a `println()` metódussal, az alábbiak szerint:

```
display.println("Hello, world!");
```

Ezután meg kell hívni a `display()` metódust, hogy ténylegesen megjelenítse a szöveget a képernyőn.

```
display.display();
```

Az Adafruit OLED könyvtár hasznos módszereket kínál a szöveg egyszerű görgetéséhez.

- `startscrollright(0x00, 0x0F)`: a szöveg görgetése balról jobbra
- `startscrollleft(0x00, 0x0F)`: a szöveg görgetése jobbról balra
- `startscrolldiagright(0x00, 0x07)`: szöveg görgetése a bal alsó sarokból a jobb felső sarokba
- `startscrolldiagleft(0x00, 0x07)`: a szöveg görgetése a jobb alsó sarokból a bal felső sarokba

Töltsd fel a kódot

Most töltd fel a kódot az ESP32-re. Emlékeztető: Töltsd fel a tesztkódot

A kód feltöltése után az OLED görgetett szöveget jelenít meg.