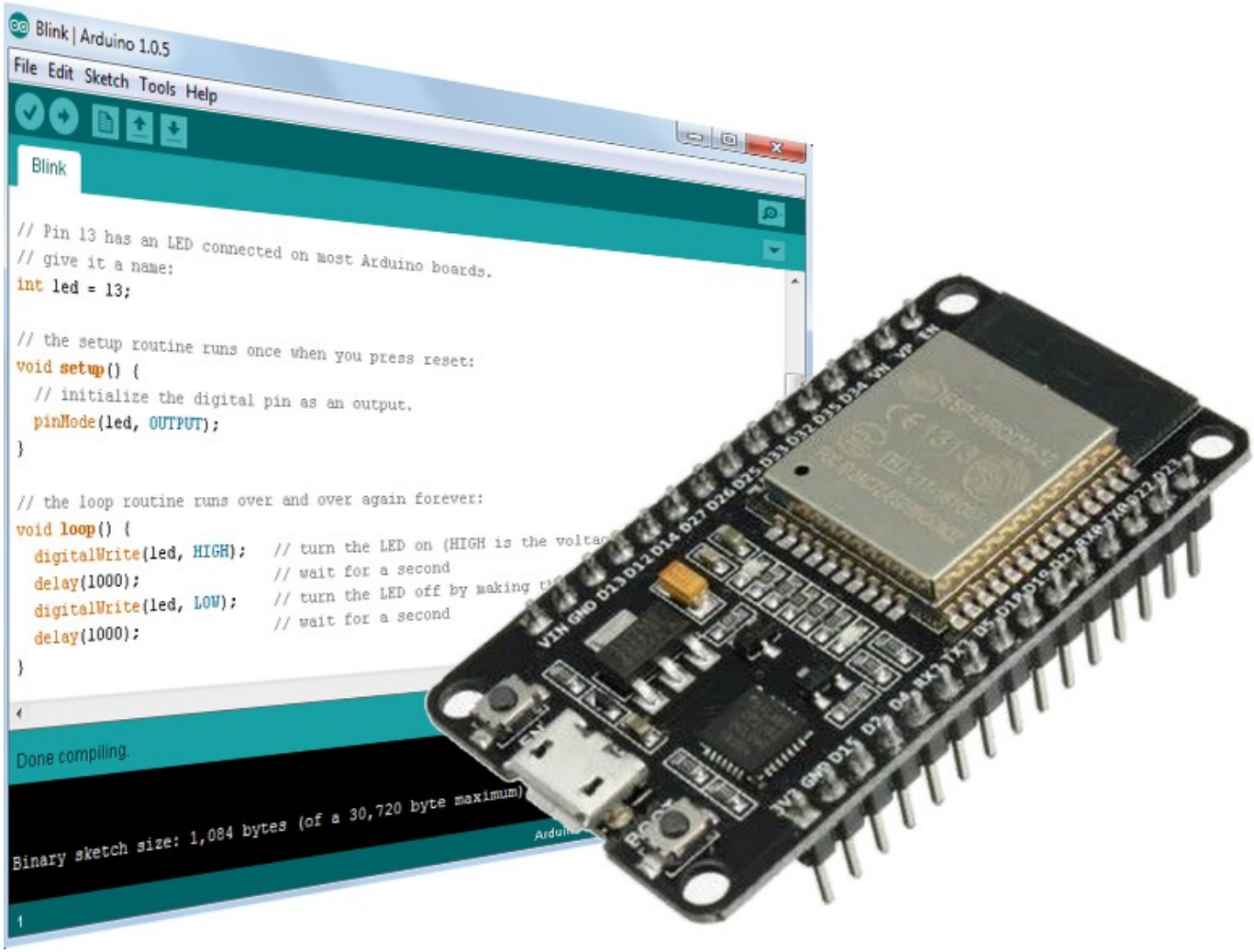
ESP32 mikrovezérlők programozása Arduino környezetben

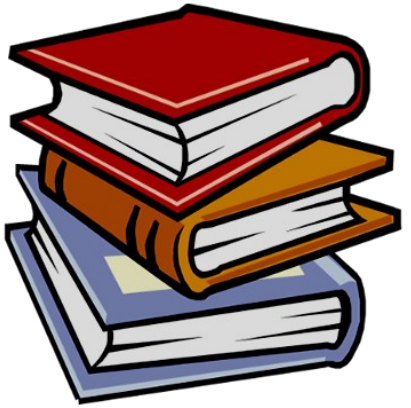


ARDUINO™

6. I2C kommunikáció, WiFi (S)NTP óraprojekt

Felhasznált és ajánlott irodalom

- Last Minute Engineers: Getting Date & Time From NTP Server With ESP32
- Maxim Integrated: DS3231 datasheet (PDF)
- Andrew Wickert: DS3231 Arduino Library
- Arduino Libraries: NTPClient library
- Cplusplus.com: C time library documentation
- Wikipédia: Hálózati Idő Protokoll (NTP és SNTP)
- ESPRESSIF: ESP32 Arduino Core Documentation
- ESPRESSIF: ESP32 Technical Reference Manual



Példaprogramok

ESP32_DS3231_read – I2C RTC kiolvasása

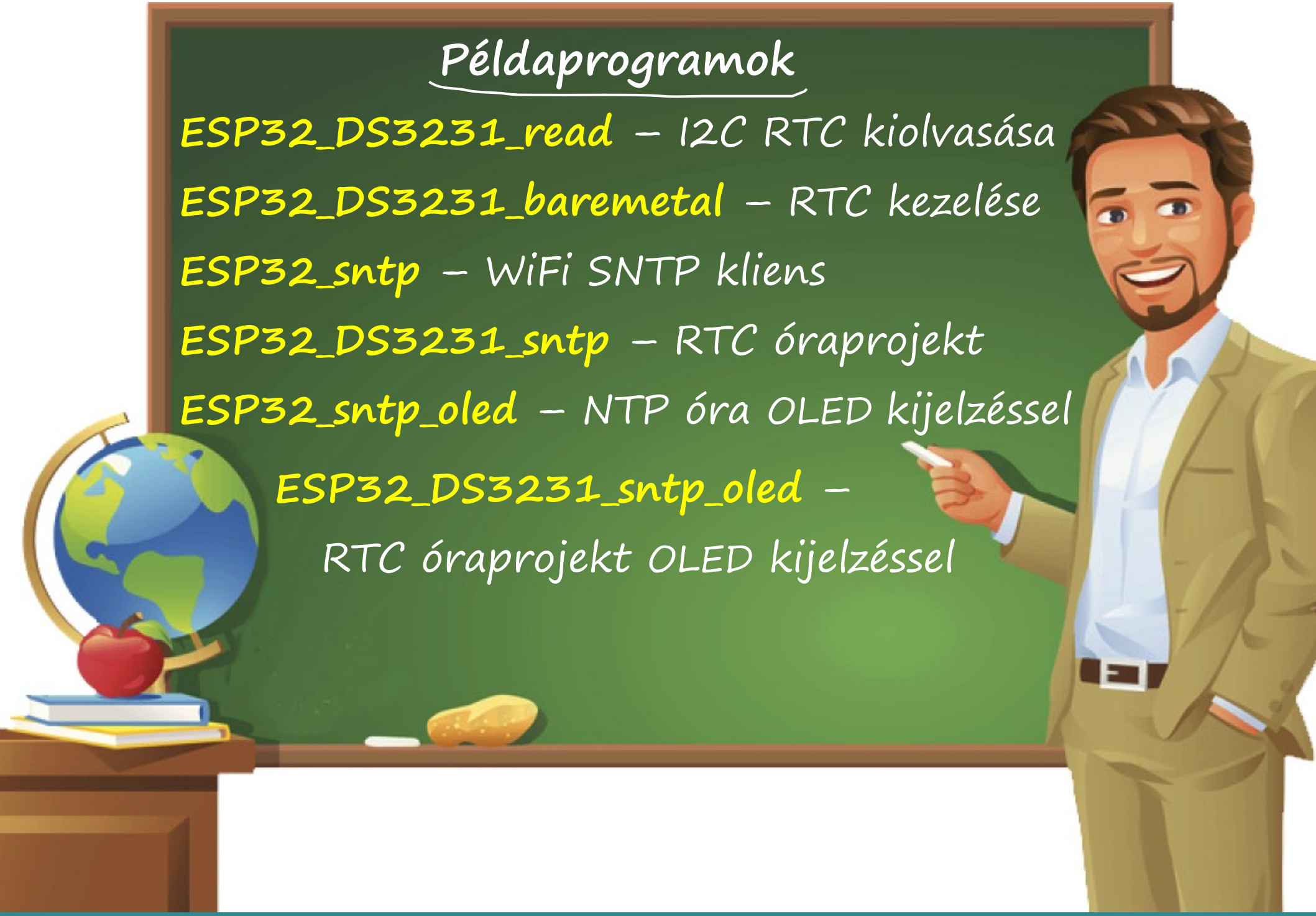
ESP32_DS3231_baremetal – RTC kezelése

ESP32_sntp – WiFi SNTP kliens

ESP32_DS3231_sntp – RTC óraprojekt

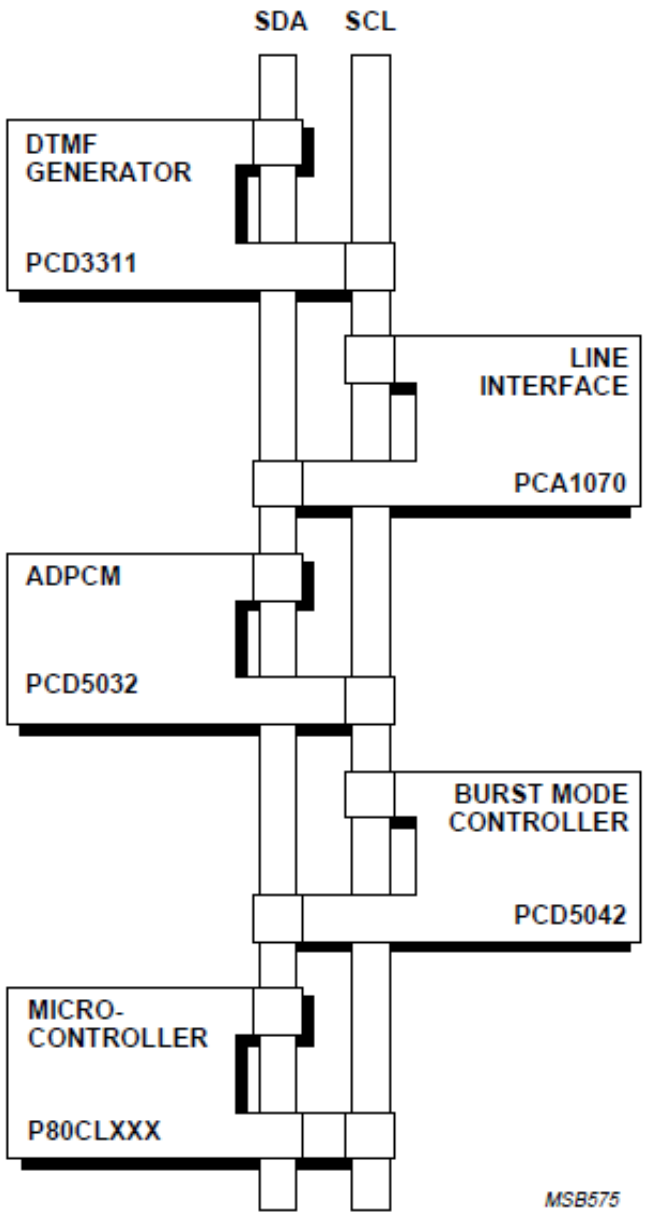
ESP32_sntp_oled – NTP óra OLED kijelzéssel

ESP32_DS3231_sntp_oled – RTC óraprojekt OLED kijelzéssel



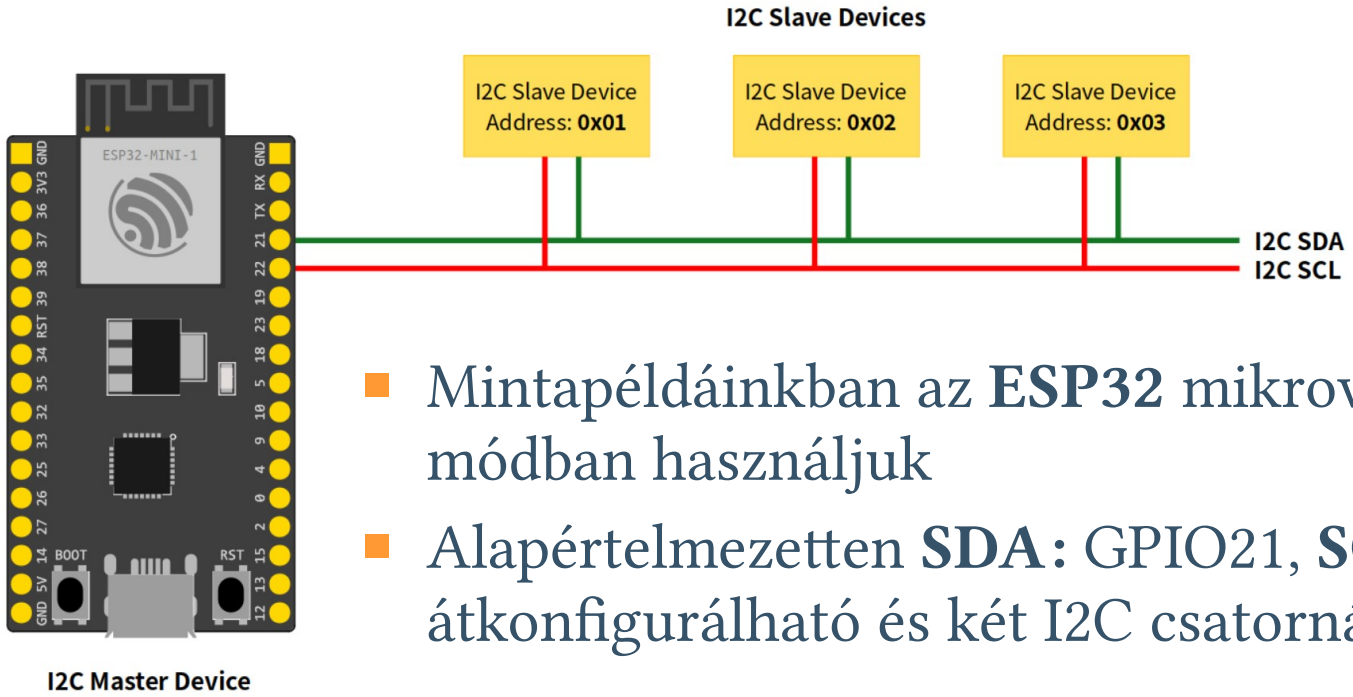
Az I2C busz

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



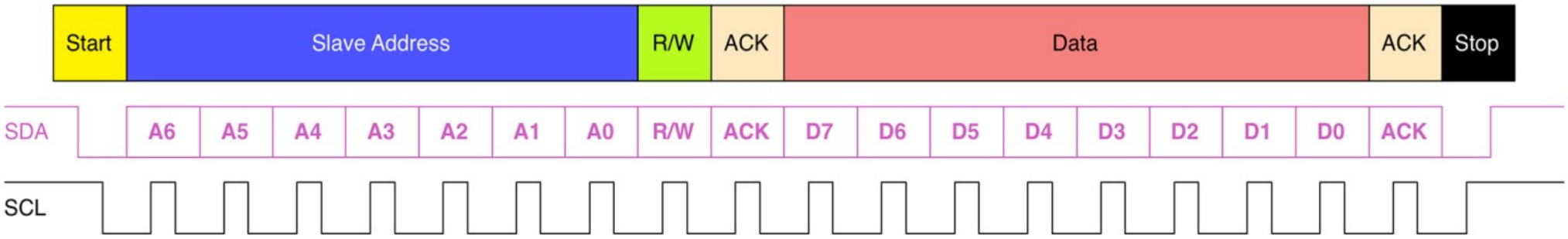
Az I2C busz

- Részletes információ a korábbi előadásainkban található:
- Az I2C kommunikációs csatorna (2020. február 6.)
  előadásvázlat mintaprogramok
- Az I2C kommunikációs csatorna – 1. rész (2021. február 4.)
  előadásvázlat mintaprogramok video
- Az I2C kommunikációs csatorna – 2. rész (2021. február 18.)
  előadásvázlat mintaprogramok video



- Mintapéldáinkban az **ESP32** mikrovezérlőt master módban használjuk
- Alapértelmezetten **SDA:** GPIO21, **SCL:** GPIO22, de ez átkonfigurálható és két I2C csatornát is nyithatunk

I2C üzenetformátum



(Az ábra forrása: Carmine Noviello – Mastering STM32)

Üzenet-orientált adatátvitel négy felvonásban:

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

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

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

Bitsorrend: A parancsbájt és az adatbájtok bitjeinek kiküldési sorrendje: **MSB first,** azaz a legnagyobb helyiértékű bit az első kiküldött bit.

Az ESP32 I2C library függvényei

- Az ESP32 I2C API az **Arduino Wire** könyvtárán alapul, s a *Wire.h* fejléc állományt kell becsatolni az **I2C** kommunikációt használó programokba
- *bool* **begin()** – *true* értékkel tér vissza, ha sikeres az inicializálás
- *bool* **end()** – lezárja a kommunikációt, felszabadítja az erőforrásokat
- *bool* **setPins**(*sdaPin, sclPin*) – az **I2C** kivezetések átirányítása
- *bool* **setClock**(*frequency*) – a buszfrekvencia megadása (def. 100 000 Hz)
- *bool* **setTimeOut**(*tout_ms*) – a „türelmi idő” megadása (def. 50ms)
- **beginTransmission**(*address*) – tranzakció indítása master módban
- **write**(*data*) – egy adatbájt kiírása a bufferbe
- *size_t* **write**(*uint8_t*, size_t*) – több bájt csoportos kiküldése
- *uint8_t* **endTransmission**(*sendStop*) – tranzakció vége (kimenet: hibakód)
- *uint8_t* **requestFrom**(*addr, size, sendStop*) – adatlekérés (*size:* bájtok száma)
- *uint8_t* **readBytes**(*data*, size*) – adatok kiolvasása a bufferből

Az I2C busz(ok) konfigurálása

- Az **ESP32** mikrovezérlő két fizikai **I2C** csatornával rendelkezik, s ezek kezeléséhez két **TwoWire** objektumot már példányosított a rendszer **Wire** és **Wire1** néven
- Az **ESP32** GPIO mátrix segítségével a kivezetések átirányíthatók

```
#include <Wire.h>
#define SDA_0 33            // I2C0 bus
#define SCL_0 32
#define SDA_1 18            // I2C1 bus
#define SCL_1 19

void setup() {
  Wire.setPins(SDA_0 , SCL_0);
  Wire.setClock(400000);  // Fast speed (400 kHz)
  Wire.begin();
  Wire1.setPins(SDA_1 , SCL_1);
  Wire1.begin();          // Standard speed (100 kHz)
}

void loop() {
}
```

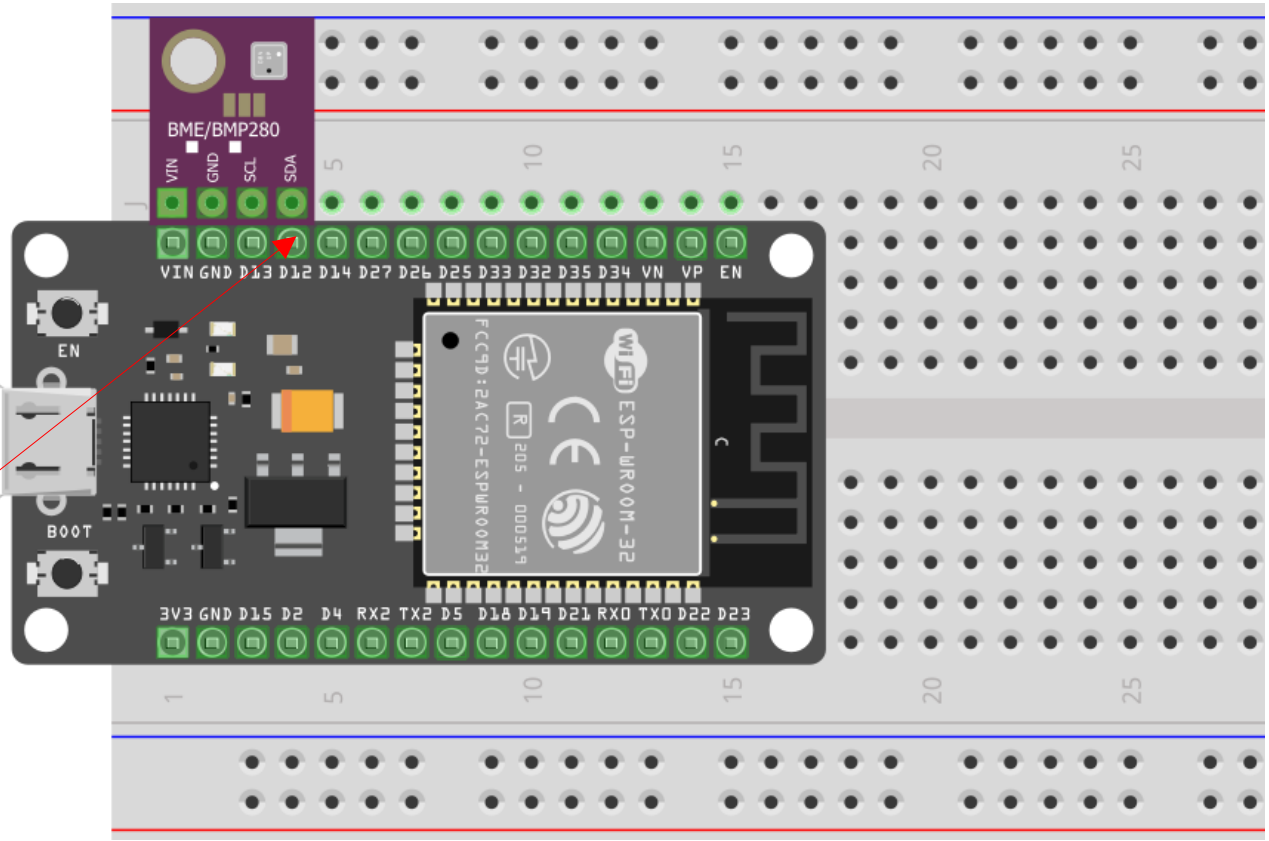
ESP32_two_i2c.ino
Példa két I2C busz konfigurálására

Az I2C busz(ok) konfigurálása

- Ha az a zseniális ötletünk támadna, hogy a képen látható módon csatlakoztassuk az előző előadásban már használt **BME280** szenzort, akkor vegyük figyelembe, hogy rendszerindításkor egyes kivezetéseknek a mellékelt táblázatban levő szinten kell lennie (itt a **GPIO12** kivezetés lehúzásáról kell gondoskodni induláskor)

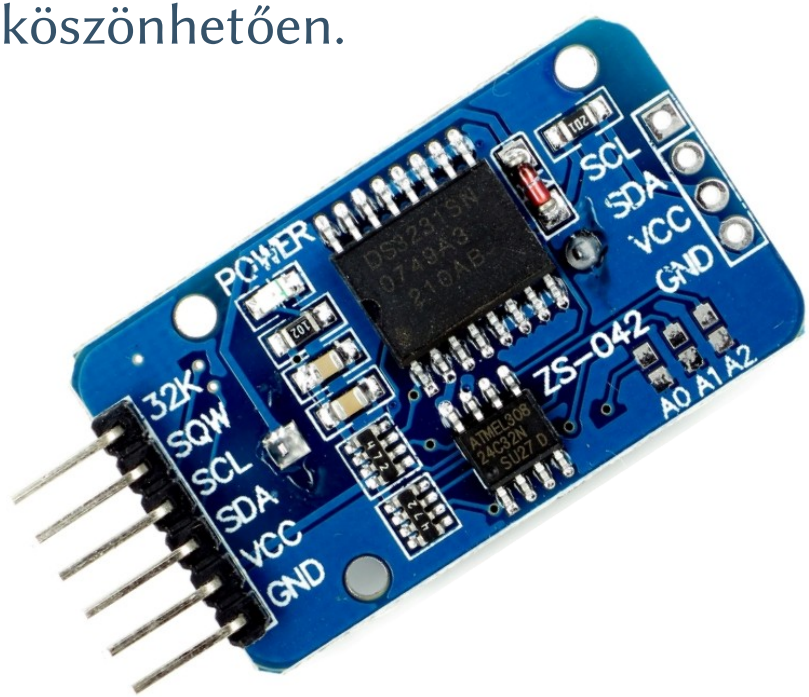
Boot feltételek indításhoz, (illetve program-letöltéshez)

PIN	Level
GPIO0	High (Low)
GPIO2	Low
GPIO5	High
GPIO12	Low
GPIO15	High



DS3231 Real-time óra modul

- Oszcillátor, óra és naptár egy tokban. A tápfeszültség megszűnésekor a hátoldalán elhelyezett telepről üzemel tovább.
- A modul többé-kevésbé cserekompatibilis a DS1307-tel, egy 4 kB EEPROM is tartalmaz, de van néhány eltérés:
  - pontosabb óra (±2 ppm a –40 – 80 °C tartományban) a beépített hőkompenzált oszcillátornak köszönhetően.
  - két riasztási időpont is megadható
  - van beépített hőmérője
  - nincs belső RAM
  - Vbat 4.2 V-os Li akkuval is táplálható!
- EEPROM I2C címe: **0x57** (állítható)
- DS3231 I2C címe: **0x68**

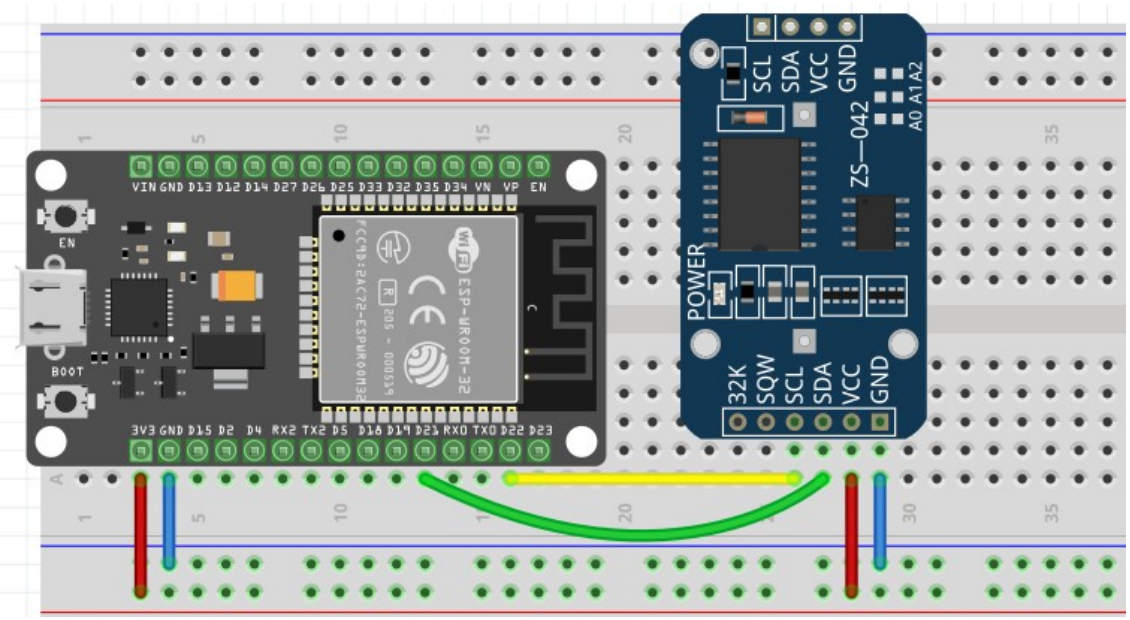


- Adatlap: datasheets.maximintegrated.com/en/ds/DS3231.pdf

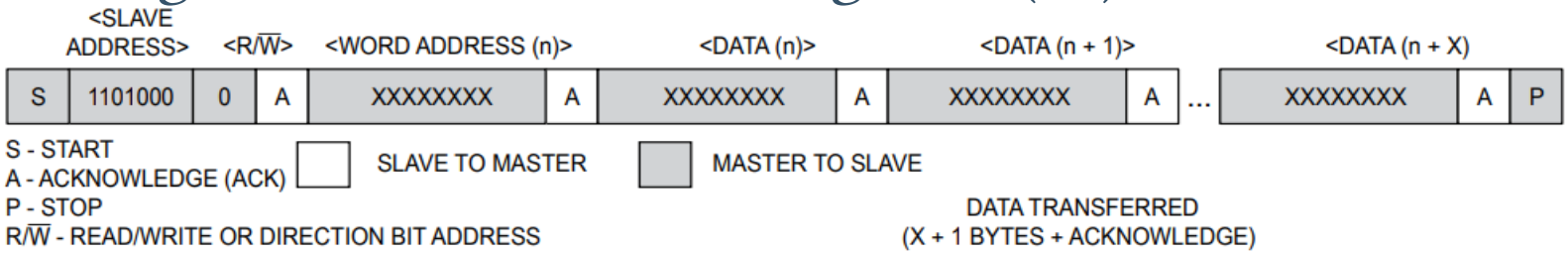
A DS3231 bekötése, használata

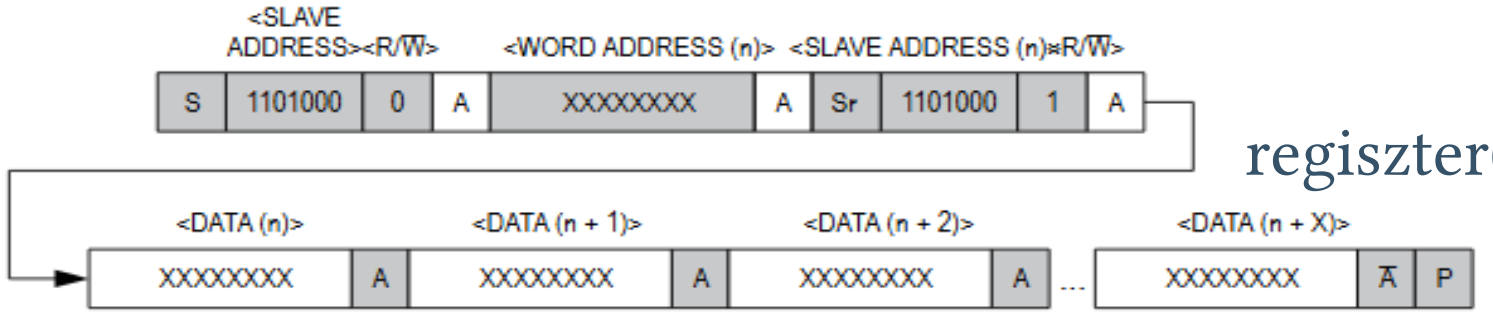
- Az áramköri elrendezés:

ESP32	**DS3231**
GPIO22	SCL
GPIO21	SDA
3V3 v. Vin	VCC
GND	GND



- Adatforgalom az I2C buszon: regiszter(ek) írása





regiszter(ek) olvasása

DS3231 regiszterek

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

ESP32_DS3231_read.ino

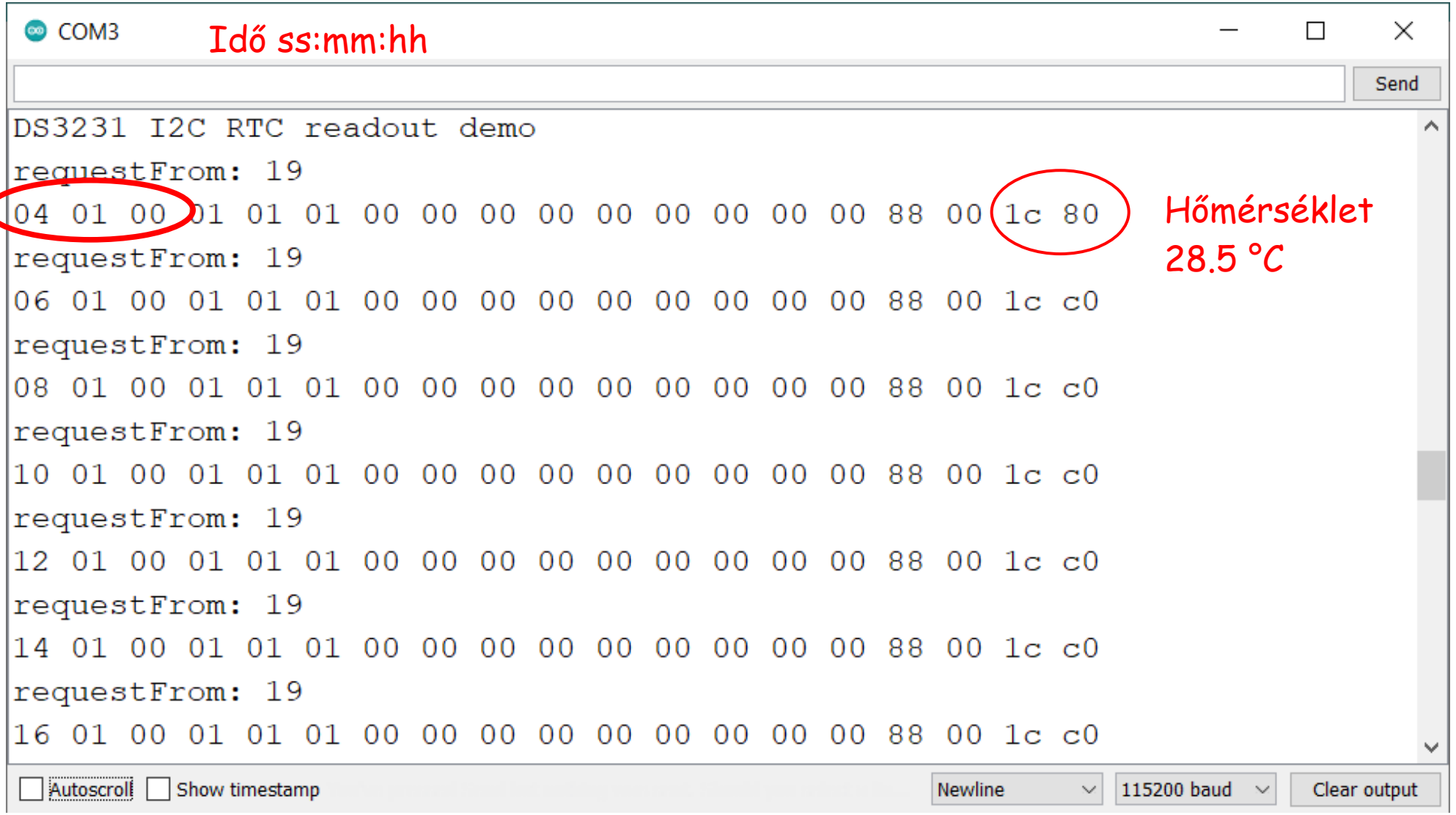
```
#include "Wire.h"
uint8_t cmd[20];                // data buffer

void setup() {
  Serial.begin(115200);
  Wire.begin();
  cmd[0] = 0x0E;                // Pointer to CONFIG register
  cmd[1] = 0x00;                // Data for CONFIG (enable clock)
  Wire.beginTransmission(0x68);
  Wire.write(cmd, 2);
  Wire.endTransmission(true);
}
void loop() {
  int nbytes = 0; delay(2000);
  cmd[0] = 0x00;                 // Pointer to first register
  Wire.beginTransmission(0x68);
  Wire.write(cmd[0]);            // Write register address
  Wire.endTransmission(false);
  nbytes = Wire.requestFrom(0x68, 19, true); // Request 19 bytes from slave
  if (nbytes) {
    Wire.readBytes(cmd, nbytes); // Read data into cmd[] array
    for (int i = 0; i < nbytes; i++) {
      Serial.printf("%02x ", cmd[i]);        // write data in hexadecomal format
    }
    Serial.println("");
  }
}
```

Kiolvassuk és a soros porton kiíratjuk a DS3231 RTC regisztereit

I2C address = 0x68

ESP32_DS3231_read.ino

- A program futási eredménye az alábbi ábrán látható



C Time Library

- A **C Time Library** az időadatok kényelmes kezeléséhez különféle adattípusokat és függvényeket definiál (itt nem soroljuk fel mindet)
- **Adattípusok:**
  - **time_t** – egy adott időpont óta eltelt, másodpercekben mért idő (pl. UNIX időbélyeg: 1970. jan 1. 00:00 GMT óta eltelt idő)
    Megjegyzés: Az idő ábrázolása implementációfüggő, így nem közvetlenül a kapott adatot használjuk fel, hanem pl. a **localtime** függvény segítségével
  - **struct tm** – időadat, elemekre bontva
- **Függvények:**
  - **mktime** – a ***tm*** struktúrát ***time_t*** típusú időbélyeggé konvertálja
  - **localtime** – a ***time_t*** típusú időadatot ***tm*** struktúrává alakítja, a helyi időzóna szerint
  - **strftime** – egy ***tm*** struktúrában megadott idő szöveggé konvertálása, megadott formátum szerint
- A **C Time Library** az **ESP32 Arduino Core** csomagnak is része, a következő programokban mi is ezt fogjuk használni az időadatok kezeléséhez vagy a konvertáláshoz

A tm struktúra

- A **C Time Library** használatához a **time.h** fejléc állományt kell becsatolni, s ez definiálja azt a struktúrát is (**struct tm**), amelyben az időt elemekre bontva tárolhatjuk (lásd: struct tm leírása)
- Sajnos, nincs 100 %-os megfeleltethetőség a **tm** struktúra és a **DS3231** RTC regiszterei között, ahogy az alábbi táblázatban is láthatjuk, az RTC emellett BCD számábrázolást használ
- Az eltéréseket programjainkban majd figyelembe kell venni

```
struct tm {
  int   tm_sec;
  int   tm_min;
  int   tm_hour;
  int   tm_mday;
  int   tm_mon;
  int   tm_year;
  int   tm_wday;
  int   tm_yday;
  int   tm_isdst;
};
```

Member	Type	Meaning	Range	DS3231
tm_sec	int	seconds after the minute	0-60*	0-59
tm_min	int	minutes after the hour	0-59	0-59
tm_hour	int	hours since midnight	0-23	0-23
tm_mday	int	day of the month	1-31	1-31
tm_mon	int	months since January	0-11	1-12
tm_year	int	years since 1900	0-199	0-99
tm_wday	int	days since Sunday	0-6	1-7
tm_yday	int	days since January 1	0-365	--
tm_isdst	int	Daylight Saving Time flag		--

years since 2000

ESP32_DS3231_baremetal.ino

- Nézzük meg, hogy hogyan tudjuk megszervezni a **DS3231** eszközből az időadatok kiolvasását (és beírását)!
- Letölthető és kényelmesen használható **DS3231** programkönyvtár helyett okulásul magunk szervezzük meg az **I2C** kommunkációt és az adatkonverziót (erre utal a *baremetal* kifejezés)
- Használjuk viszont a beépített **Wire** és **C Time** könyvtárakat, amelyekről az előzőekből már szó volt
- Az alábbi függvényeket fogjuk definiálni:
  - **setTime()** – **DS3231** inicializálása egy ***tm*** struktúra adataiból
  - **getTime** – **DS3231** kiolvasása és egy ***tm*** struktúra feltöltése
  - **decToBcd** – segédlet **setTime()**-hoz (adat BCD-be konvertálása)
  - **bcdToDec** – segédlet **getTime()**-hoz (BCD adat konvertálása)

ESP32_DS3231_baremetal.ino - 3/1.

- A főprogram roppant egyszerű, szinte triviális, az ördög majd a definiálandó függvények részleteiben bújik meg...
- Kiíratásnál az alábbi varázslat helyett az **strftime()** függvényt is használhatnánk az időadatok karakterfüzérré történő konvertálására

```
#include "Wire.h"           // I2C programkönyvtár
#include "time.h"           // C Time programkönyvtár

#define RTC_ADDRESS 0x68    // I2C slave cím
struct tm timeinfo;         // Időadatokat tároló struktúra

void setup() {
  Serial.begin(115200);     // Soros porton lesz kiíratás
  Wire.begin();             // Alapértelmezett I2C busz inicializálás
  //setTime(&timeinfo);     // Valamikor be kell állítani az órát...
}

void loop() {
  getTime(&timeinfo);       // Idő és dátum kiolvasása
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S"); // Kiíratás varázslat
  delay(1000);              // Várunk egy másodpercet
}
```

ESP32_DS3231_baremetal.ino - 3/2.

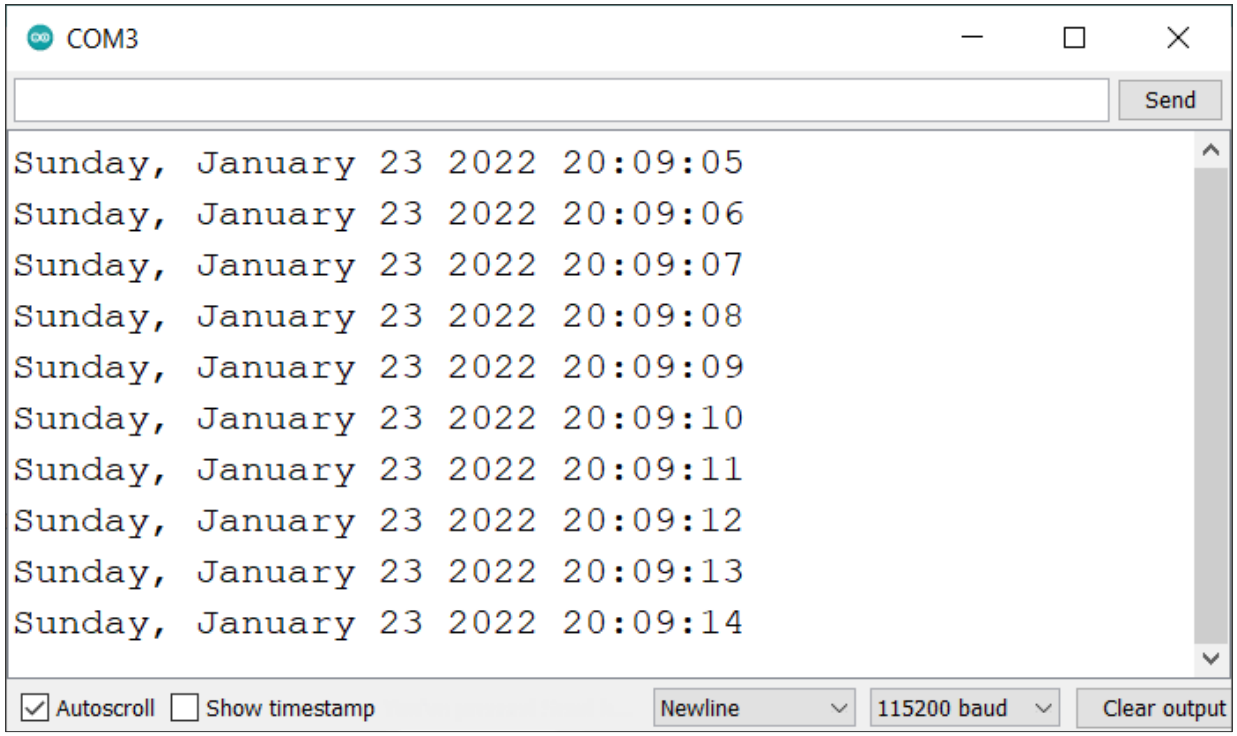
```
//--- Convert decimal numbers to BCD ----------------------
byte decToBcd(byte val) {
  return ( (val / 10 * 16) + (val % 10) );
}
//--- Convert BCD numbers to decimal ----------------------
byte bcdToDec(byte val) {
  return ( (val / 16 * 10) + (val % 16) );
}

//--- Set RTC time/date -----------------------------------
void setTime(struct tm* time) {
  Wire.beginTransmission(RTC_ADDRESS);
  Wire.write(0); // set register pointer to 00h
  Wire.write(decToBcd(time->tm_sec));   // set seconds
  Wire.write(decToBcd(time->tm_min));   // set minutes
  Wire.write(decToBcd(time->tm_hour));  // set hours
  Wire.write(time->tm_wday + 1);        // set day of week (1=Sun, 7=Sat)
  Wire.write(decToBcd(time->tm_mday));  // set date (1 to 31)
  Wire.write(decToBcd(time->tm_mon) + 1); // set month
  Wire.write(decToBcd(time->tm_year - 100)); // year from 2000 (0 to 99)
  Wire.endTransmission();
}
```

ESP32_DS3231_baremetal.ino - 3/3.

```
//--- Read time/date from RTC -----------------------------
void getTime(struct tm* time) {
  Wire.beginTransmission(RTC_ADDRESS);
  Wire.write(0);                        // a kiolvasás kezdőcímének beállítása
  Wire.endTransmission(false);
  Wire.requestFrom(RTC_ADDRESS, 7);  // Hét bájt kiolvasása (time/date)
  time->tm_sec = bcdToDec(Wire.read() & 0x7f);  // Másodpercek (0-59)
  time->tm_min = bcdToDec(Wire.read());         // Percek (0 - 59)
  time->tm_hour = bcdToDec(Wire.read() & 0x3f); // Órák (24h kijelzéshez)
  time->tm_wday = bcdToDec(Wire.read() - 1);    // Hét napja (0 - 6)
  time->tm_mday = bcdToDec(Wire.read());        // hónap napja (1 - 31)
  time->tm_mon = bcdToDec(Wire.read() - 1);     // hónap sorszáma (0 - 11)
  time->tm_year = bcdToDec(Wire.read()) + 100;  // 1900-tól eltelt évek
}
```

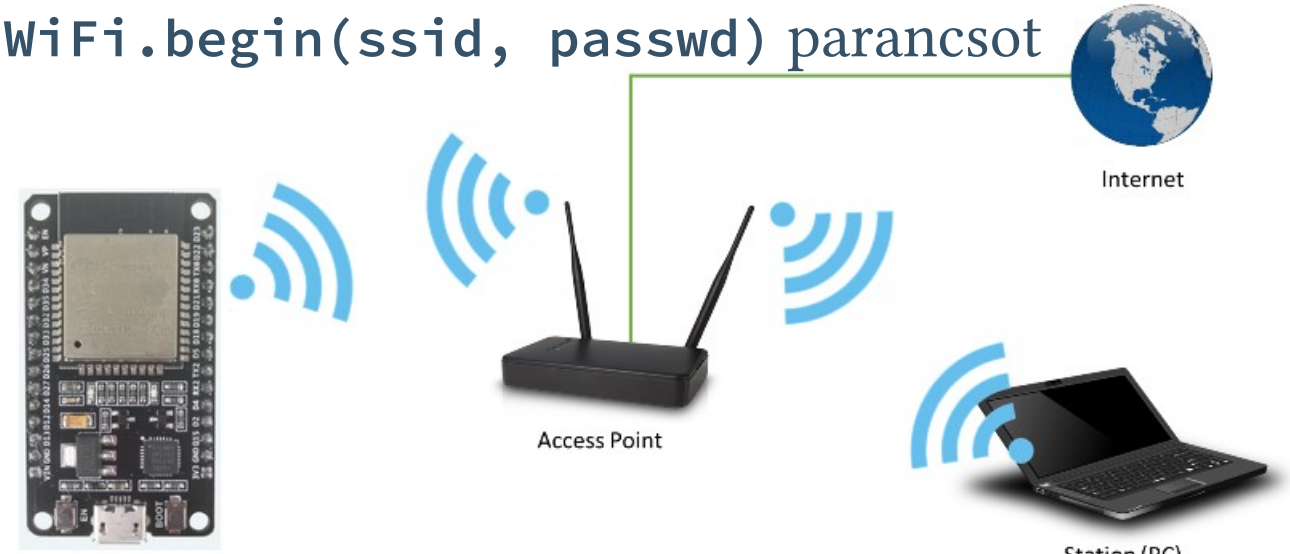
„Apu, hod med be?”

- Az **ESP32_DS3231_baremetal.ino** programot futtatva szépen működik, de nem biztos, hogy a valódi időt mutatja – ehhez legalább egyszer be kell állítani a **DS3231** regisztereit!
- Hogyan tehetjük ezt meg, hogyan állíthatjuk be a pontos időt?
- Egy kézenfekvő lehetőség erre az **ESP32 WiFi** perifériájának és a nyilvános **NTP** vagy **SNTP** szolgáltatás használata



Kapcsolódás az Internetre

- Az **ESP32 Wi-Fi** perifériája és az **ESP32 Arduino Core** részét képező WiFi programkönyvtár lehetővé teszi, hogy a helyi hálózatra csatlakozzunk (**STA** módban, kliensként, vagy **AP** módban, elérési pontként, vagy egyidejűleg mindkét módot használva)
- Mi most kliens alkalmazást készítünk, így **STA** (station) módban a helyi routerhez (vagy elérési ponthoz) fogunk csatlakozni, amely az internet elérést biztosítja számunkra
- A programjainkba ehhez becsatoljuk a **WiFi.h** fejléc állományt és kiadjuk a `WiFi.begin(ssid, passwd)` parancsot



ESP32 Wi-Fi STA módban

Csatlakozás a WiFi hálózathoz

- Az **ESP32** kliensként történő használatához csatlakoztatnunk kell a Wi-Fi hálózatra, a belépéshez meg kell adnunk a WIFI_SSID és WIFI_PASS adatokat
- A kiíratás természetesen opcionális, a **Serial** kezdetű sorokat elhagyhatjuk

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

void setup () {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
}
```

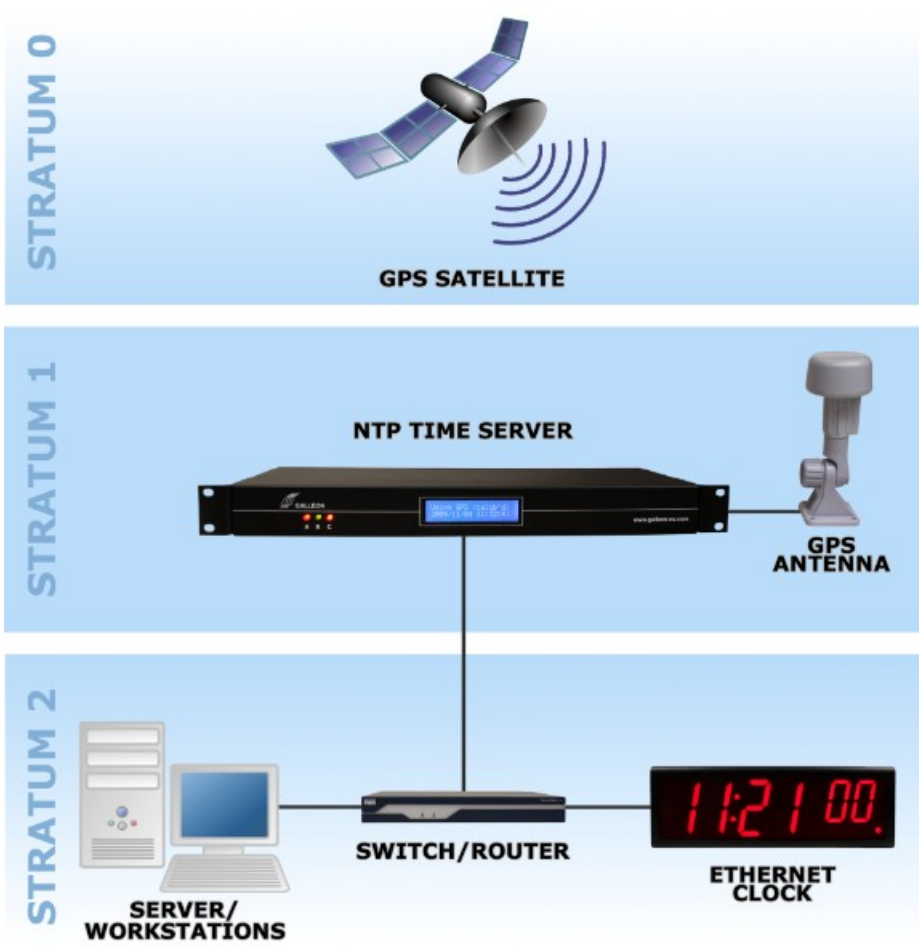
- A személyes adatokat kiszerveztük egy fejléc állományba, amelyet a **Vázlatfüzet** (Sketchbook) mappa **libraries/secrets** almappájában helyeztünk el

secrets.h

```
#define WIFI_SSID MY_SSID
#define WIFI_PASS MY_PASSWORD

String OPENWEATHERMAP_APPID =
"**************************";

String THINGSPEAK_WRITE_APIKEY =
"xxxxxxxxxxxx";
```

Pontos idő lekérdezése NTP kéréssel

- A világot behálózó **NTP** (Network Time Protocol) szerverek ún. UDP csomagokkal kommunikálnak
- **NTP** szerver lehet helyi vagy távoli
- A nyilvános szervereket az **pool.ntp.org** fogja össze (lásd: https://www.ntppool.org/en/)
- Az **NTP** szerverek általában a 123-as portot használják
- Az üzenetcsomag formátumát (amely többnyire 48 bájt) és a protokollt eredetileg az **RFC958,** ma (NTP v4) az **RFC5905** írja le
- A legegyszerűbb **NTP** kérelem: 0x1B és 47 db nulla



Kép forrása: www.galsys.co.uk/news/

ESP32_sntp.ino

- Az **SNTP** az **NTP** protokol egyszerűsített változata, beágyazott rendszerek számára. A **WiFi** programkönyvtár már tartalmazza

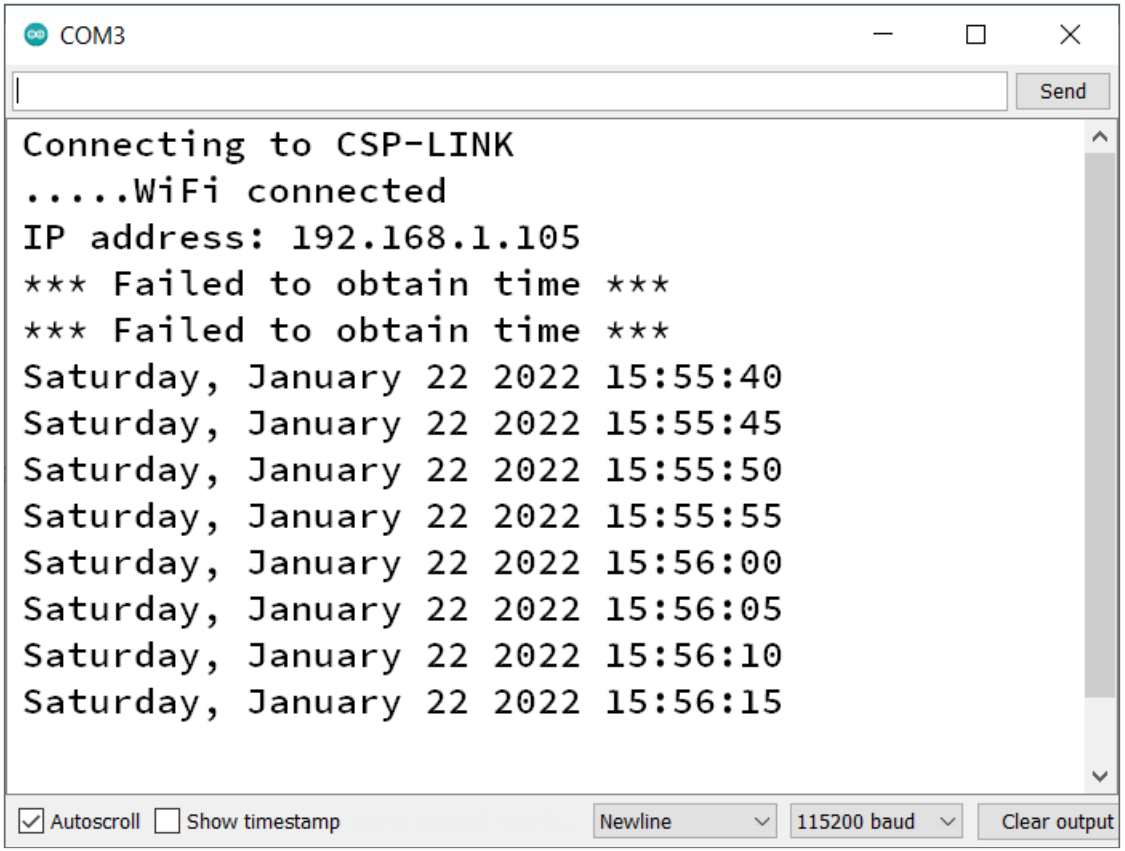
```
#include <WiFi.h>
#include "time.h"
#include "secrets.h"
const char* time_zone = "CET-1CEST,M3.5.0,M10.5.0/3";  // Europe/Budapest
const char* ntpServer = "hu.pool.ntp.org";             // Regional NTP net

void setup() {
  Serial.begin(115200);
  setup_wifi();                                 // Connecting to WiFi AP
  configTzTime(time_zone, ntpServer);  // Configure local Time Zone
}

void loop() {
  struct tm timeinfo;
  delay(5000);
  if (getLocalTime(&timeinfo)) {
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
  }
  else { Serial.println("*** Failed to obtain time ***"); }
}
```

Időzóna és nyári időszámítás megadása
lásd: https://leo.leung.xyz/wiki/Timezone

A **configTime**, **configTzTime** *és* **getLocalTime** *függvényeket* **Arduino.h** *deklarálja és* **esp32-hal.time.c** *definiálja*

ESP32_sntp.ino

- A program a helyi **WiFi** hálózatra a *secrets.h* állományban megadott paraméterekkel jelentkezik be (közben 0,5 s-onként kiír egy pontot)
- Ezután egy SNTP szerverről próbálja lekérni az időt, ez eltarthat egy ideig (eközben kapjuk 5 s-onként a „*Failed…*” kezdetű kiírásokat)



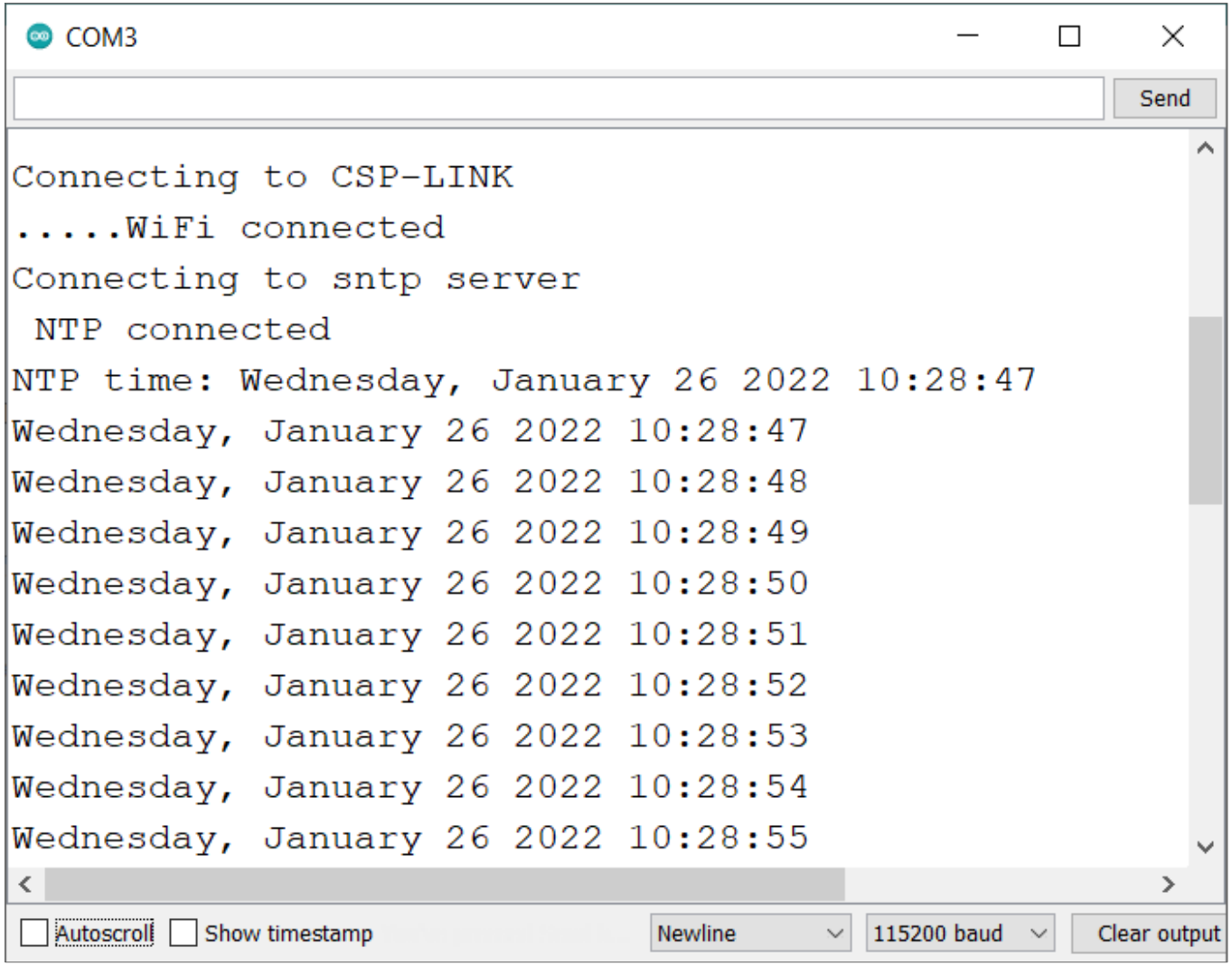
ESP32_DS3231_sntp.ino

- Az előző program tapasztalatai alapján már könnyen ki tudjuk egészíteni a korábbi **ESP32_DS3231_baremetal** programunkat
- A kiegészített program neve **ESP32_DS3231_sntp.ino** lesz
  - Induláskor kapcsolódjon **Wi-Fi**-n keresztül a helyi hálózatra
  - Kérje le egy **SNTP** szervertől az aktuális időt
  - Írja be az időtt a **DS3231 RTC**-be – szerencsére ehhez már az előzőekben megírtuk a setTime() függvényt
  - Ezután a **Wi-Fi** kapcsolatra nincs tovább szükség, le is kapcsolhatjuk
- A fenti kiegészítéssel megoldjuk a DS3231 óra automatikus beállítását induláskor, s onnan kezdve már tényleg az aktuális időt mutatja
- A következő oldalon csak a kiegészített program releváns részletét mutatjuk be, a többi részletben megegyezik a korábban bemutatott **ESP32_DS3231_baremetal.ino** programunkkal

ESP32_DS3231_sntp.ino (részlet)

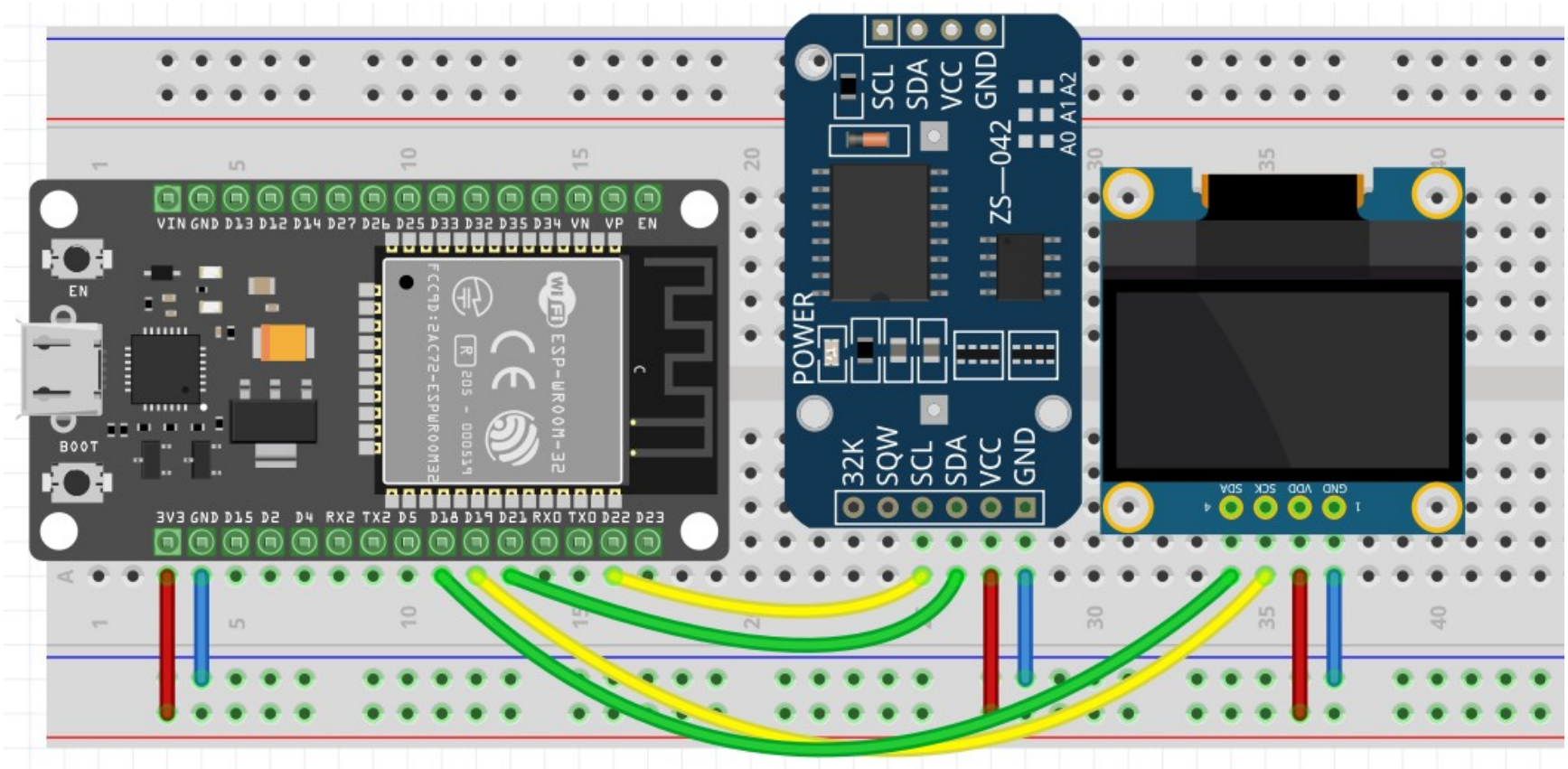
```
#include "Wire.h"           // I2C programkönyvtár
#include "time.h"
#include <WiFi.h>
#include "secrets.h"        // WiFi SSID & password
#define RTC_ADDRESS 0x68    // I2C eszköz címe
struct tm timeinfo;         // Időadatok tárolására

void setup() {
  Serial.begin(115200);
  Wire.begin();             // I2C busz inicializálás
  setup_wifi();             // Connecting to WiFi AP
  setup_sntp();             // Requesting time from NTP server
  WiFi.disconnect(true);    // disconnect WiFi
  WiFi.mode(WIFI_OFF);      // as it's no longer needed
}

void setup_sntp() {
  const char* ntpServer = "hu.pool.ntp.org";              // regionális NTP hálózat
  const char* time_zone = "CET-1CEST,M3.5.0,M10.5.0/3";   // Europe/Budapest időzóna
  configTzTime(time_zone, ntpServer);                     // Időzóna megadása
  Serial.println("Connecting to sntp server");
  while (!getLocalTime(&timeinfo)) {Serial.print(".");}   // Pontosidő lekérése
  setTime(&timeinfo);                                     // A DS3231 RTC beállítása
  Serial.println(" NTP connected");
  Serial.println(&timeinfo, "NTP time: %A, %B %d %Y %H:%M:%S");
}
```

ESP32_DS3231_sntp.ino eredménye



NTP óra OLED kijelzéssel

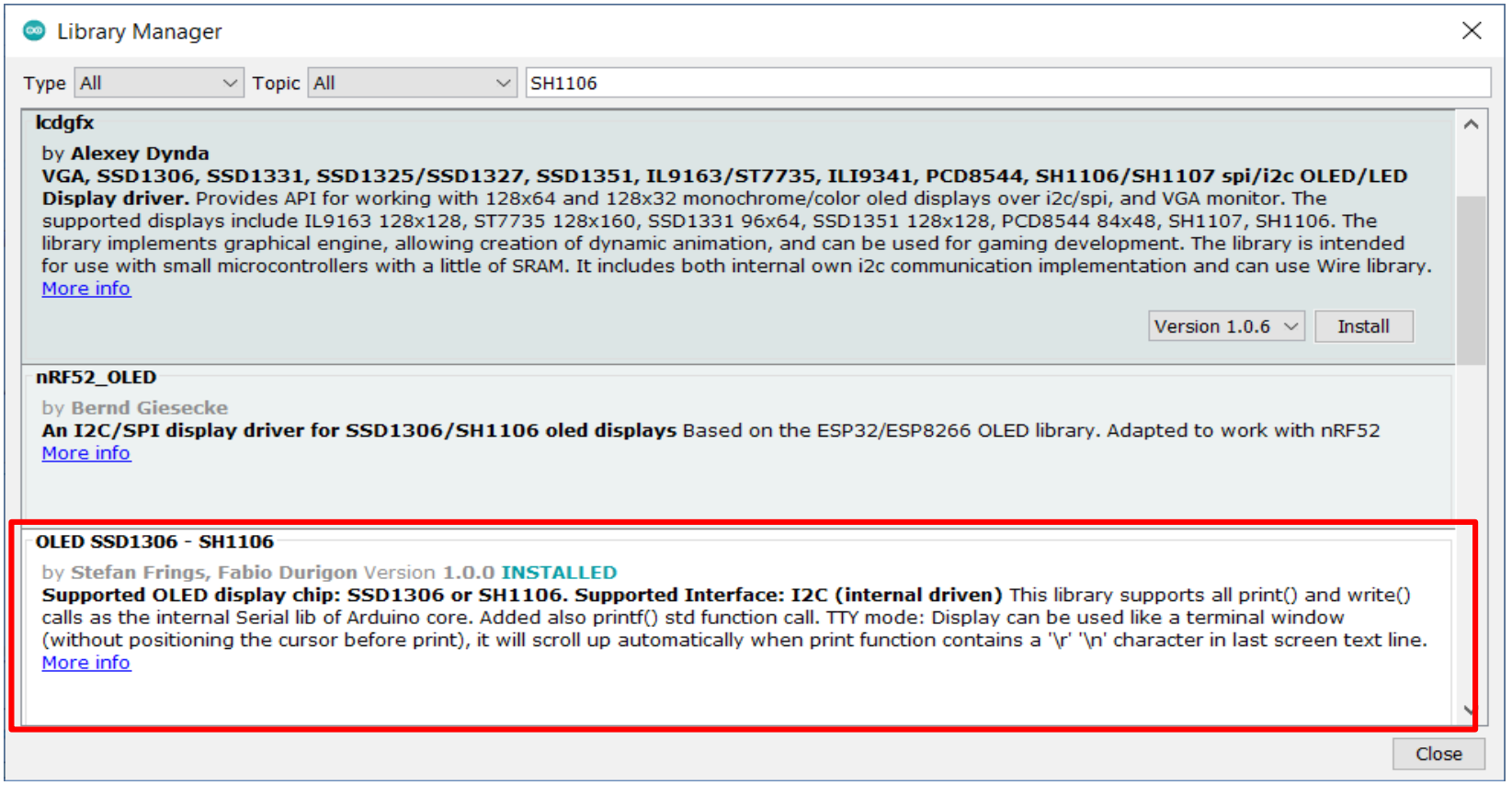
- Ha függetleníteni akarjuk az óránkat a számítógéptől (pl. elemes táplálás, hordozható kivitel), akkor szükségünk lesz egy kijelzőre
- Az **I2C** buszra csatlakoztatható, **SSD1306,** vagy **SH1106** vezérlővel ellátott OLED kijelzők egyszerű megoldást kínálnak ehhez



Az OLED programkönyvtár

- Az **OLED SSD1306 – SH1106** könyvtárat választottam, de az **oled.cpp** állományban a **display()** metódust **SH1106**-hoz javítani kell:

```
if (isSH1106) {
    i2c_send(0xB0 + page); // set page
    i2c_send(0x02);        // lower columns address =2
    i2c_send(0x10);        // upper columns address =0
}
```



A kijelző inicializálása

- Az OLED kijelző I2C címe 0x3C (esetleg (0x3D)
- **Megjegyzés:** ez az oled programkönyvtár **szoftveres I2C** kezeléssel van megírva, nem közösíthető a hardveres **I2C** busszal!
- A kivezetéseket, az **I2C** címet és az egyéb paramétereket (grafikai felbontás, van-e RESET kivezetés, SH1106 vagy SSD1306 vezérlő) a konstruktornak kell megadni, például:

```
OLED display=OLED(18,19,NO_RESET_PIN,0x3C,128,32,false);
```

SDA pin SCL pin RESET pin I2C address resolution in pixels isSH1106?

A fenti példa egy SSD1306 128 x 32 pixel felbontású kijelzőt definiál

SDA
SCL
VCC
GND

ESP32_sntp_oled.ino

- Most először az **ESP32_sntp.ino** programot egészítjük ki a kijelzővel, s majd csak utána foglalkozunk a **DS3231** RTC-vel

```
#include <WiFi.h>
#include <Wire.h>
#include <time.h>
#include "oled.h"
#include "secrets.h"

const char* time_zone = "CET-1CEST,M3.5.0,M10.5.0/3";
const char* ntpServer = "hu.pool.ntp.org";
struct tm timeinfo;
static char msg[20];        // character buffer

OLED display = OLED(21, 22, NO_RESET_PIN, 0x3C, 128, 32, false); // SSD1306
//OLED display = OLED(21,22,NO_RESET_PIN,0x3C,128,64, false);    // SSD1306
//OLED display = OLED(21,22,NO_RESET_PIN,0x3C,128,64, true);     // SH1106

void setup() {
  Serial.begin(115200);
  setup_wifi();             // Connecting to WiFi AP
  configTzTime(time_zone, ntpServer);
  display.begin();
  display.clear();
  display.display();
}
```

Ebben a programban nem használjuk a hardveres I2C buszt ezért használhattuk itt a 21, 22 kivezetéseket az OLED kijelzőhöz...

A soros porti kiírásokat csak a program ellenőrzése céljából hagytuk meg, a végleges programban erre már nincs szükség!

ESP32_sntp_oled.ino

- A kiíratást a **display.draw_string()** függvény, az idő/dátum adatok szöveggé alakítását az **strftime() függvény** segítségével végezzük
- A megjelenítéshez a **display.display()** függvényt is meg kell hívni

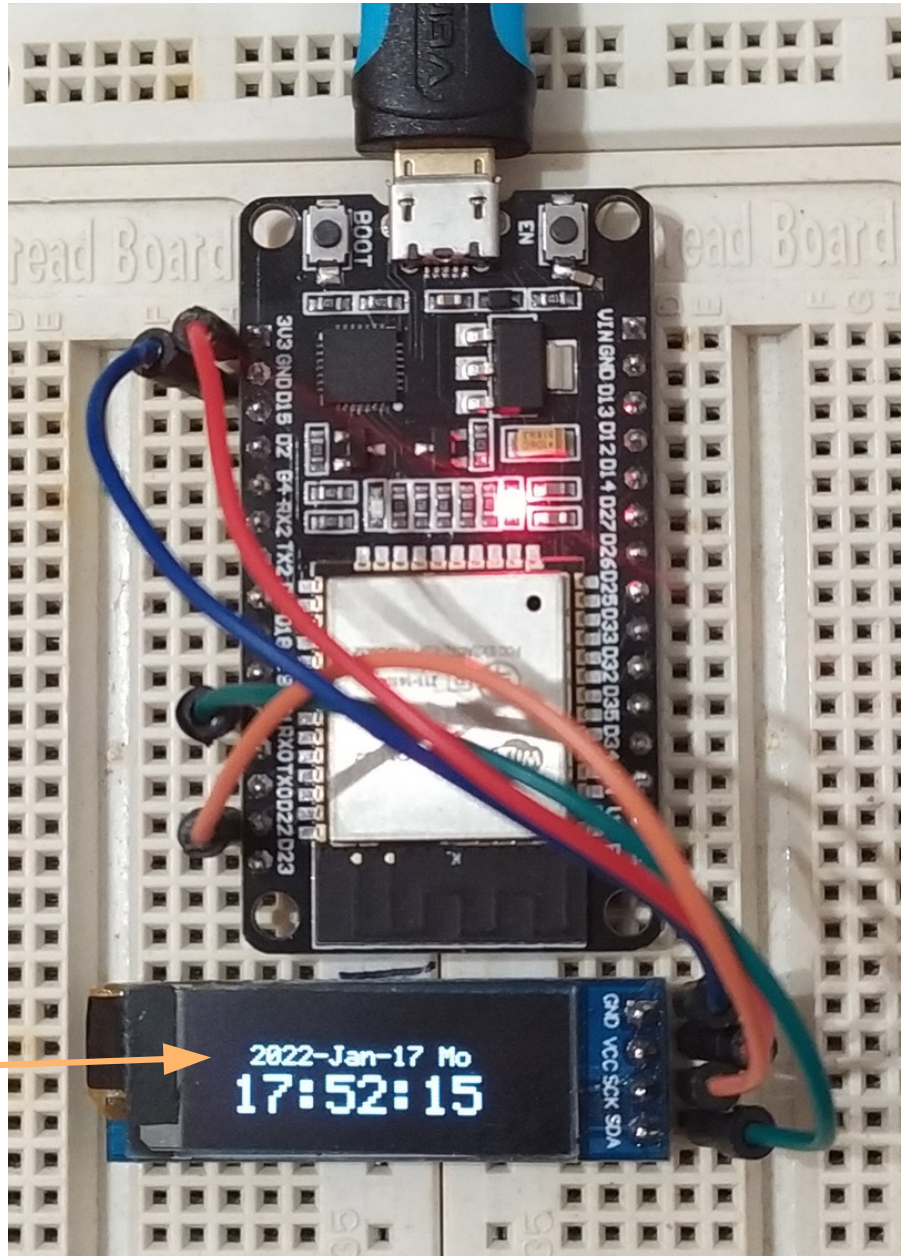
```
strftime(msg, maxsize, ”format”, &timeinfo);
```

Buffer max. text size Date/Time format struct tm*

```
void loop() {
  delay(1000);
  if (getLocalTime(&timeinfo)) {
    Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
    display.clear();
    strftime (msg, 15, "%Y-%b-%d %a ", &timeinfo); // 2022-Jan-27 Thu format
    display.draw_string(16, 1, msg);               // Display date
    strftime (msg, 10, "%T ", &timeinfo);          // Display time as 17:00:00
    display.draw_string(8, 12, msg, OLED::DOUBLE_SIZE);
    display.display();                             // Refresh screen
  }
  else {
    Serial.println("*** Failed to obtain time ***");
  }
}
```

ESP32_sntp_oled.ino

- A program futási eredménye az alábbi képen látható
- A kép állása az **oled.cpp** állományban az **A0/C0** vagy **A1/C8** inicializáló parancsokkal választható ki.
  Erről bővebb információ a 2021. február 18-i előadásban található (STM32 tanfolyam)



Ez az A0/C0 parancsok által kiválasztott állás

ESP32_DS3231_sntp_oled.ino

- Egészítsük ki az **ESP32_DS3231_sntp.ino** programot az előző programból átvett OLED kijelző kezeléssel!
- Az OLED kijelző kezeléssel kibővített új programunk neve **ESP32_DS3231_sntp_oled.ino** lesz, s induláskor csatlakozik a Wi-Fi hálózatra, SNTP protokollal lekéri a pontos időt, ezzel az adattal feltölti a **DS3231** RTC modul regisztereit, majd lekapcsolódik a hálózatról, és a továbbiakban az RTC-ből kiolvasott időt jeleníti meg az OLED kijelzőn
- A soros port kiíratásokat egyelőre meghagyjuk (a program nyomkövetéséhez nagy segítséget jelent)
- A hardveres és a szoftveresen kezelt **I2C** buszok szétválasztása miatt az OLED kijelzőt a **GPIO18** (SDA) és **GPIO19** (SCL) kivezetésekre kötöttük, a 30. oldalon látható bekötési rajz szerint

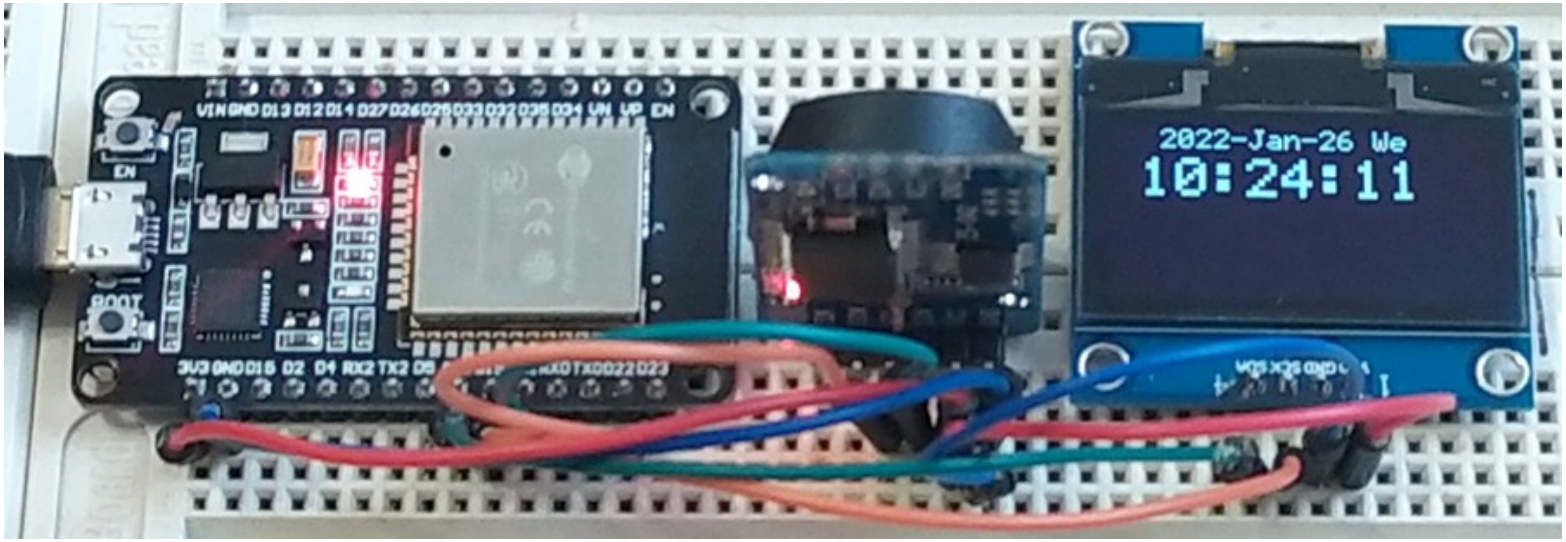
ESP32_DS3231_sntp_oled.ino (részlet)

```
#include "Wire.h"           // I2C library
#include "oled.h"           // https://github.com/durydevelop/arduino-lib-oled
#include <WiFi.h>           // Wi-Fi and network library
#include "time.h"           // https://www.cplusplus.com/reference/ctime/
#include "secrets.h"        // WiFi SSID & password

#define RTC_ADDRESS 0x68    // I2C eszköz címe
struct tm timeinfo;
static char msg[20];       // character buffer

OLED display = OLED(18, 19, NO_RESET_PIN, 0x3C, 128, 32, false);  // SSD1306
//OLED display = OLED(18, 19, NO_RESET_PIN,0x3C,128,64, false);   // SSD1306
//OLED display = OLED(18, 19, NO_RESET_PIN, 0x3C, 128, 64, true); // SH1106

void setup() {
  Serial.begin(115200);
  Wire.begin();
  display.begin();
  display.clear();
  display.draw_string(4, 8, "RTC clock", OLED::DOUBLE_SIZE);
  display.display();
  setup_wifi();             // Connecting to WiFi AP
  setup_sntp();             // Requesting time from NTP server
  WiFi.disconnect(true);    // disconnect WiFi
  WiFi.mode(WIFI_OFF);      // as it's no longer needed
}
```

ESP32_DS3231_sntp_oled.ino (részlet)

```
void loop() {
  getTime(&timeinfo);       // display time/date
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
  display.clear();
  strftime (msg, 15, "%Y-%b-%d %a ", &timeinfo);
  display.draw_string(16, 1, msg);            // Display date
  strftime (msg, 10, "%T ", &timeinfo);       // Display time
  display.draw_string(8, 12, msg, OLED::DOUBLE_SIZE);
  display.display();                          // Refresh screen
  delay(1000);              // every second
}
```

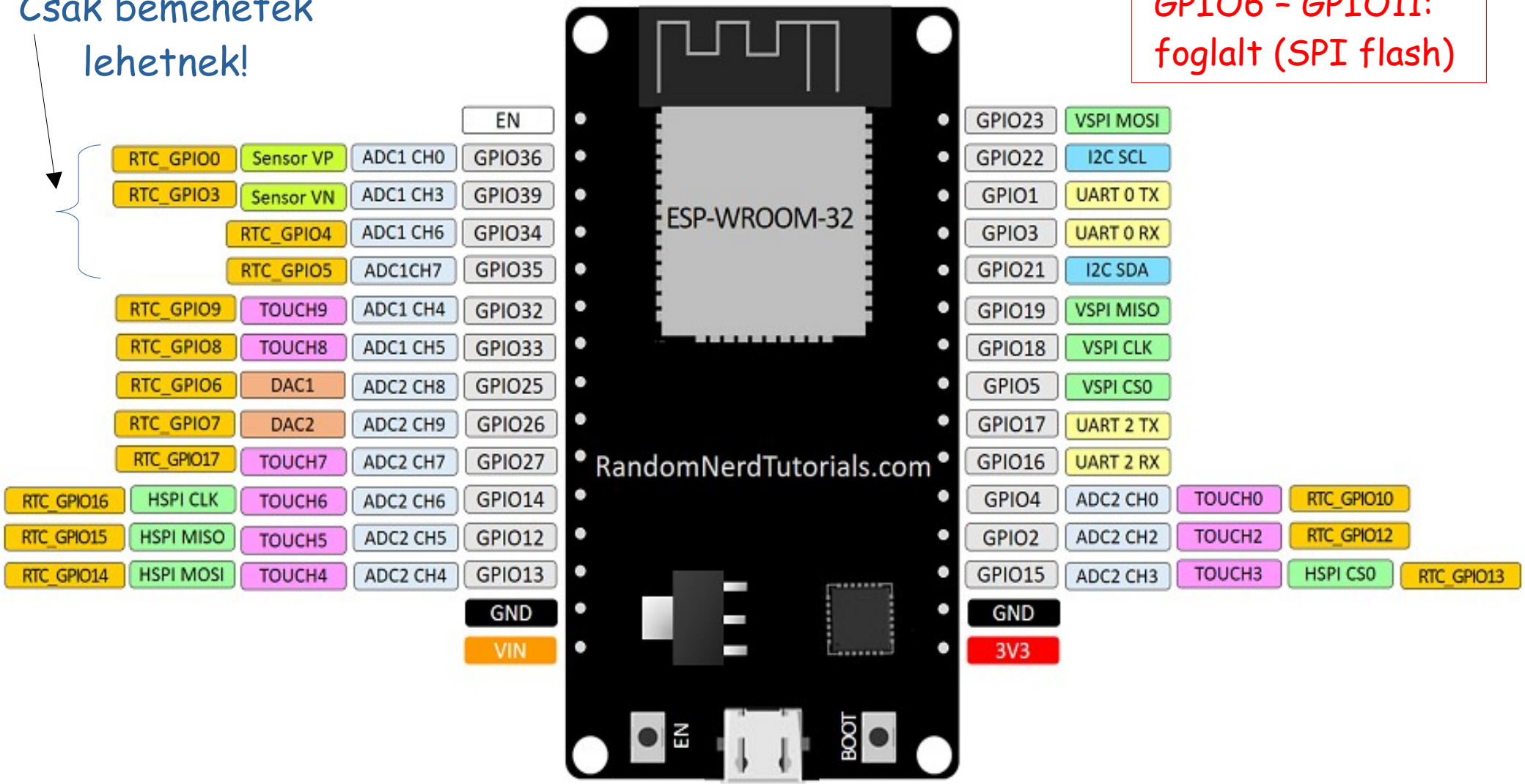
- A program többi része megegyezik az ESP32_DS3231_sntp.ino programmal



A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



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

Ellenállás színkódok