Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

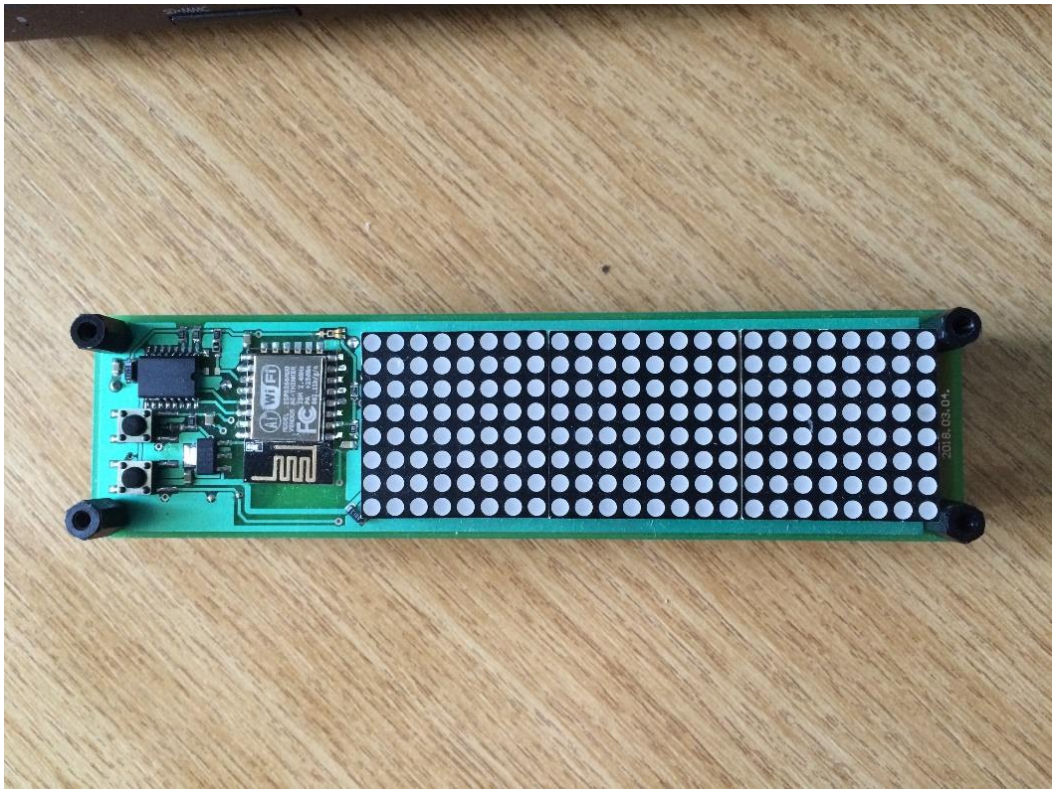
Földvári Dávid

MÁTRIX KIJELZŐS ÓRA ESP8266 MODULLAL

KONZULENS
Csorvási Gábor

BUDAPEST, 2018

Tartalom



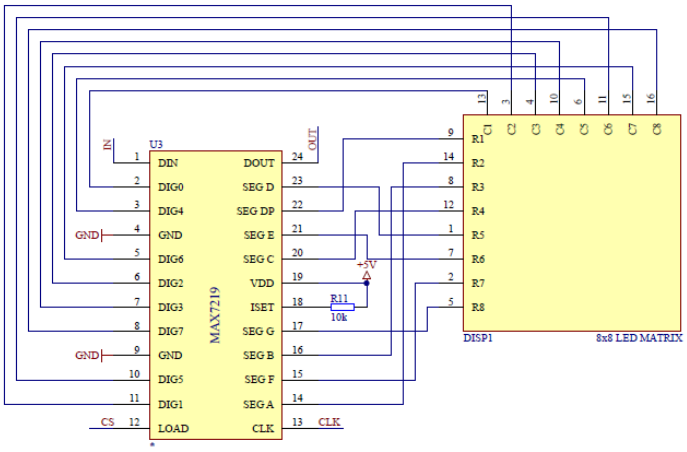
1. A motiváció.

A feladat szerint szükség volt egy órára, amely nem hagyományos hétszegmenses kijelzőn jeleníti meg az időt, hanem valami olyanon, amelyre mást is ki lehet jelezni. LCD kijelzőt nem szerettem volna alkalmazni, de a buszokról jól ismert mátrixos kijelzők mindig is tetszettek. Szerencsére léteznek erre a feladatra alkalmas modulok és meghajtó IC-k. A Maxim MAX7219 típusú SPI buszos meghajtójára esett a választás. A gombos időbeállítás helyett kényelmesebb megoldás kellett, így elkezdtem keresgélni az interneten. Sikerült találni egy WiFi kommunikációra képes modult (ESP8266-12F), amihez perifériákat illesztve megoldhatónak tűnt a feladat. Az volt a cél, hogy tápfeszültség nélkül se felejtse el az aktuális időt. Erre a feladatra a Maxim DS3231 hőmérséklet kompenzálású belső oszcillátoros real-time clock IC-jét választottam ehhez egy CR2032 gombelemet csatlakoztatva tápkimaradás esetén sem kell újra beállítani az időt. Tervezés közben merült fel az igény hőmérséklet mérésre alkalmas periféria csatlakoztatása. A legegyszerűbb megoldást választva egy LM75 típusú hőmérő IC-t választottam, amely ugyanarra az I2C buszra csatlakozik, mint az RTC. Adatlapok olvasása közben sikerült megtudni, hogy a MAX7219 képes 32 lépcsőben állítani a LED-ek fényerejét, ezért szükség volt egy analóg fényérzékelő szenzorra. Erre egy TEMT6000 típusú érzékelőt választottam, amely az ESP8266 analóg bemenetére kapcsolódik.

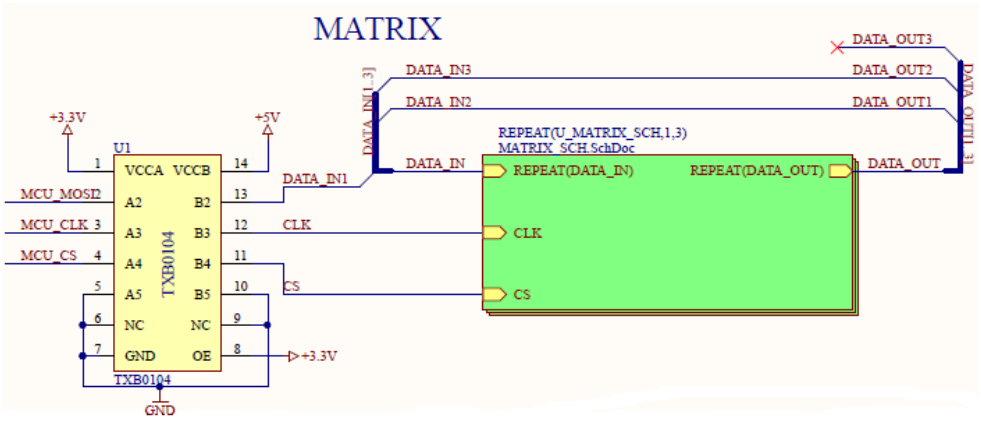
2. A kapcsolási rajz.

Miután sikerült összeválogatni a használni kívánt alkatrészeket, muszáj volt gyorsan meg is rendelni, mert ebay-ről töredék áron lehet hozzájutni majdnem mindenhez. Miközben az alkatrészek úton voltak elkészítettem a kapcsolási rajzot, illetve rajzokat.

A kijelző meghajtása:

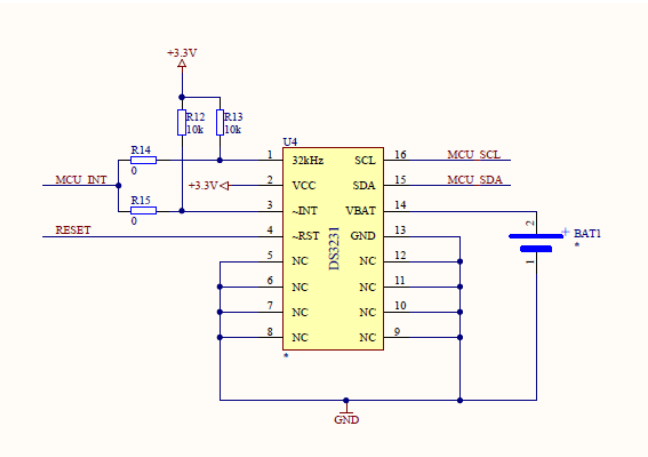


A meghajtó IC adatlapi ajánlása szerint kötöttem be a LED mátrix modult. Fontos erre figyelni, ha nem akarjuk, hogy később a programban a kijelezni kívánt adatokat bitenként kelljen összepakolni. Az R11 ellenállással állítható be a maximális áram, amit a kimenet a LED-ekre kapcsolhat. Egy ilyen LED modul 15-20mA-es áram mellett már maximum fényerővel világít. A kapcsolás kaszkásodítható: DIN_2 ->$DOUT_1$. A CS (Chip Select) jelet minden meghajtó közösen kapja meg, ez a MAX7219-nél LOAD-dal van jelölve. A CLK (SPI clock) jelet szintén minden modul megkapja közösen. 3 kijelzőt és 3 meghajtót használtam fel. Az utolsó meghajtó kimenetét szabadon kell hagyni. Az első meghajtónál viszont felmerült egy probléma. Az ESP8266 3.3V-os tápfeszültségen üzemel, a MAX7219 pedig 5V-on. Szükség volt egy illesztése 3.3V-ról 5V-ra. Erre a Texas Instruments TXB0104 típusú erre a feladatra kitalált IC-jét használtam fel.



A nem használt bemeneteket ajánlott GND-re kötni. Szerencsére erre csak ezen az egy helyen volt szükség. Az LM75 és a DS3231 is képes 3.3V-os tápfeszültségről üzemelni, így ott nem volt szükség szintillesztésre.

Az RTC bekötése is adatlapi ajánlás szerint, történt. Tervezésnél még nem tudtam, hogy a 32kHz-es kimenet generáljon megszakítást vagy az ~INT kimenet. Az egyik egy 32kHz-es négyszögjelet állít elő a másik pedig 1Hz-es megszakítást tud generálni. Végül sem az R14 sem az R15 nem került beépítésre. A program részletes magyarázatánál kitérek rá, hogy miért.



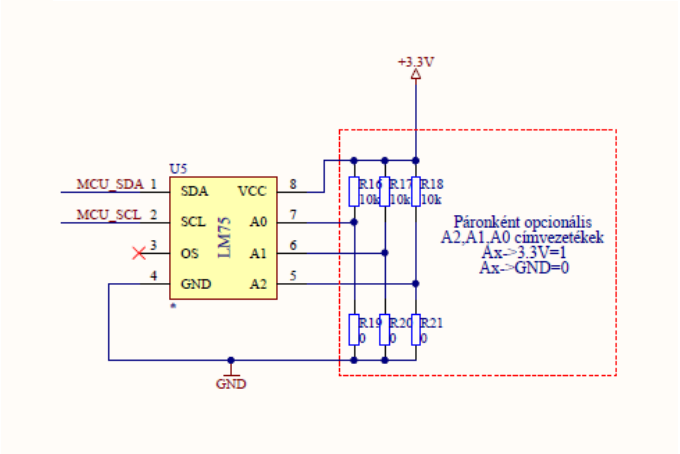
Az RTC fix I2C címmel rendelkezik az LM75-nek viszont van 3 címbeállító bemenete. Fontos volt figyelembe venni a buszra csatlakoztatott eszközök címeit.

A DS3231 báziscíme:

address byte contains the 7-bit DS3231 address, which is 1101000, followed by the direction bit (R/$\overline{W}$), which is 0 for a write. After receiving and decoding the

Az LM75 I2C báziscíme:

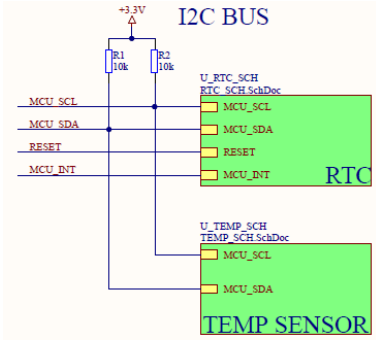
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
1	0	0	1	A2	A1	A0	R/$\overline{W}$



Teljesen tetszőleges, hogy hova kötjük az A0,1,2 címbeállító bemeneteket. Én az R16, R17, R18 ellenállásokat forrasztottam be, hogy a két cím minél több bitben térjen el egymástól.

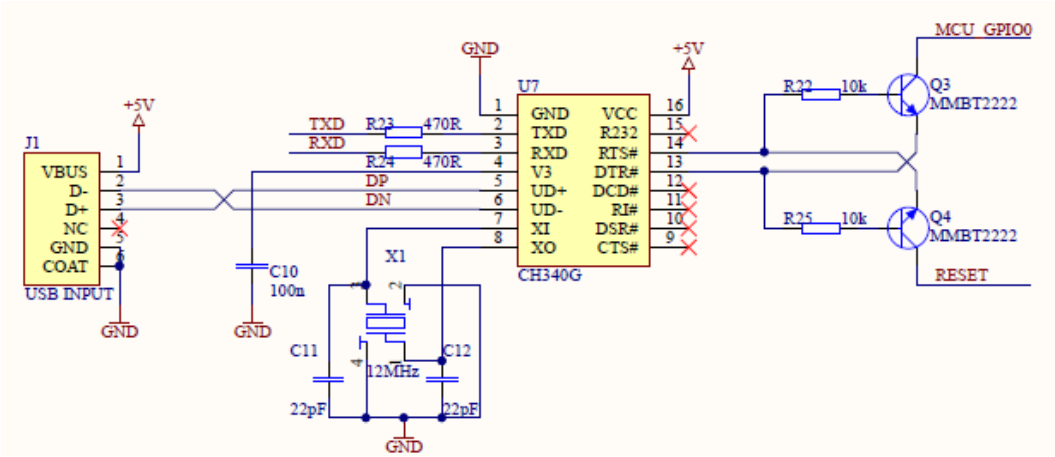
Így a DS3231 az 1101000[R/W] címmel, míg az LM75 az 1001111[R/W] címmel szerepel a buszon.

Az RTC és a hőmérő IC egy közös felhúzóellenállással csatlakoznak a pozitív tápfeszültségre. Erre mindenképp szükség van az I2C busz OC (OD) volta miatt. Az RTC megkapja az ESP RESET jelét minden RESET gomb megnyomásakor. A fentebb említett MCU_INT végül nem kapott semmilyen szerepet.

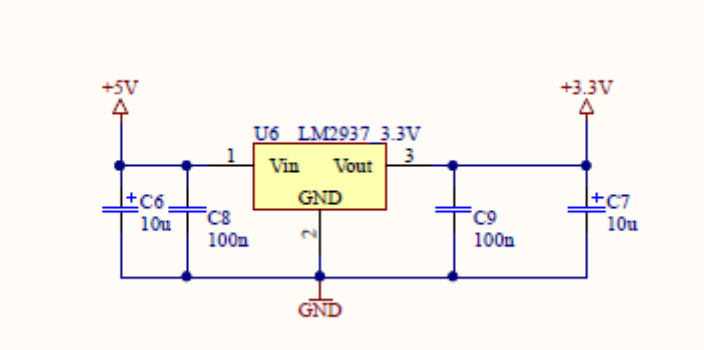


Az LM75 csak a két I2C vezetékével csatlakozik a mikrokontrollerhez.

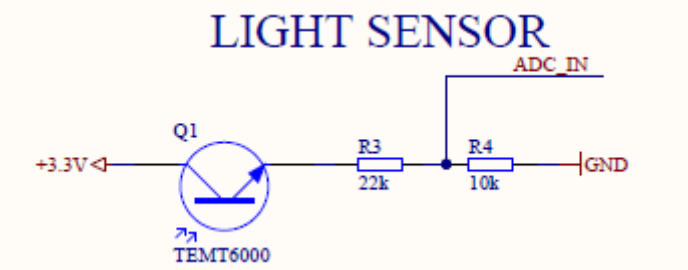
A programozáshoz az ESP UART kommunikációját lehet használni. Mivel hagyományos soros port már elég ritka a számítógépekben ezért egy USB-UART átalakítót terveztem a kapcsolásra. Az miniUSB csatlakozó egyrészt megoldja a polaritáshelyes +5V-os tápellátást és a CH340 típusú UART átalakító felé a kommunikációt. A kapcsolás megegyezik a NodeMCU nevű fejlesztőkártyán találhatóval.

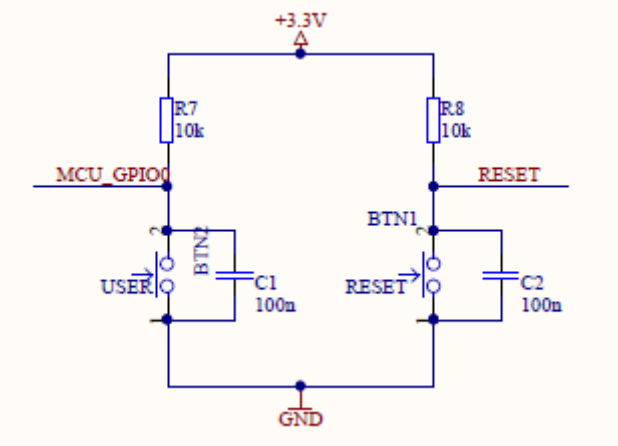


A 3.3V-os tápellátást az 5V-os tápfeszültségből egy LDO-val állítom elő. A ki és bemenet egyaránt kapott egy 100nF-os szűrő és egy 10uF-os pufferkondenzátort.

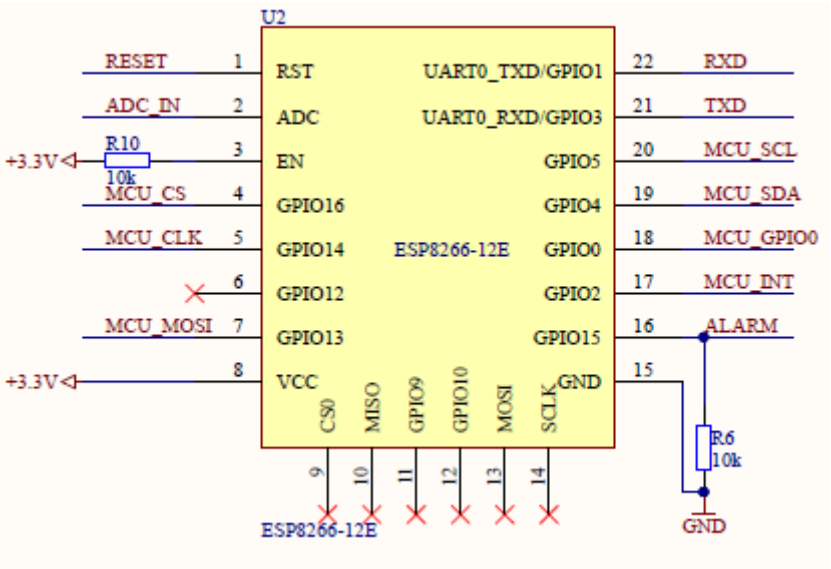


A fényerősség mérése analóg módon történik. Egy közel lineáris karakterisztikájú fényszenzorból és egy ellenállás osztóból áll. Erre azért van szükség mert adatlap szerint az ESP8266 modul analóg bemenete maximum 1V-os feszültséget kaphat. Az R4, R3 ellenállások 1/3 arányban osztják le a 3.3V-os tápfeszültséget az ADC_IN bemenetre. 1V feletti feszültséget még maximális fényerő mellett sem mérhetünk az ADC_IN bemeneten mert teljes megvilágítás mellett is esik ~0.5V a fényszenzor kollektor és emittere között.



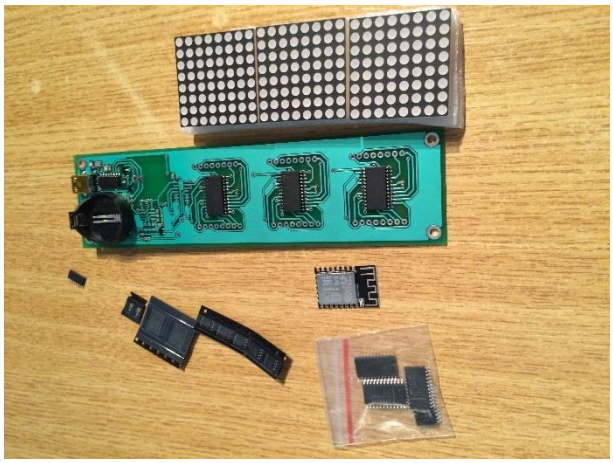


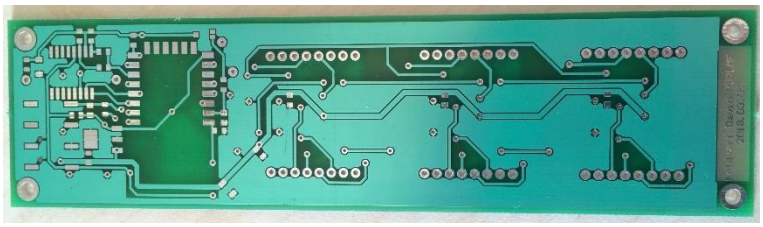
Az áramkörön elhelyeztem 2 nyomógombot. Az egyik egy RESET gomb, amely az ESP8266 és a DS3231 resetjét váltja ki. A másik nyomógomb a GPIO0-ra csatlakozik, ennek kettős szerepe van. Programozás esetén program módba lépéskor szükséges, program futtatása közben pedig a kijelzési módok között lehet váltani vele, erről bővebben később.



Programozáskor szükség van reset után arra, hogy a GPIO0 és GPIO15 logikai 0, a GPIO2 logikai 1 szinten legyen. Ekkor kerül a modul bootloader módba és lehetőség van a soros porton érkező firmware flashelésére. Az MCU_INT, mint fentebb említettem nem került bekötésre, helyette egy felhúzó ellenállással pozitív tápfeszültségre van kötve. Az ALARM kimenet sajnos nem használható, mivel program futtatása közben a GPIO15-nek folyamatosan GND-re húzva kell lennie. A működéshez szükség van az EN bemenetre pozitív tápfeszültséget kötni. Az ESP modulom verziója sajnos sok I/O kivezetést nem enged használni így muszáj voltam beérni azzal a kevéssel, amely használható. Mire mindent bekötöttem el is fogyott minden lehetőségem és sajnos az ALARM funkciótól búcsút kellett venni.

3. Az építés.





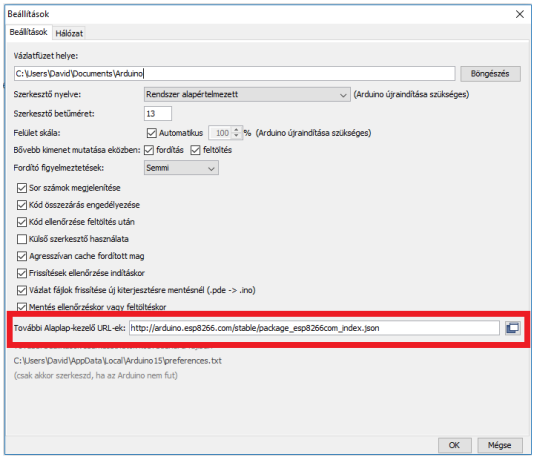
A megtervezett nyomtatott áramkör az ETT NYÁK gyártó üzemében került legyártásra. Miután kézhez kaptam az alkatrészeket is, beforrasztottam. Az áramkör felélesztését egy állítható áramkorlátú labortápegységről végeztem, mert az esetleges zárlatok tönkre tehetik az USB portokat. Miután meggyőződtem, hogy az áramkör nem zárlatos elkezdhettem a programmal foglalkozni.

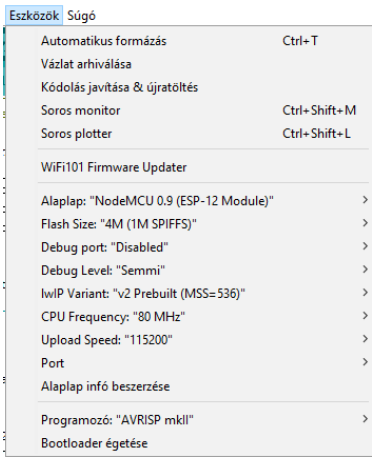
4. Az élesztés.

Az ESP8266 modulokat sajnos nem igazi fejlesztésekhez találták ki, sokkal inkább hobbisták használják. Én az Arduino környezetet választottam mert ha elakadok, akkor a legtöbb fórumon ehhez a környezethez található segítség.

Az első lépés beállítani az IDE-ben, hogy ESP8266 modullal szeretnénk foglalkozni.

Ha ez megtörtént az Arduino IDE letölti a fejlesztéshez szükséges könyvtárakat és



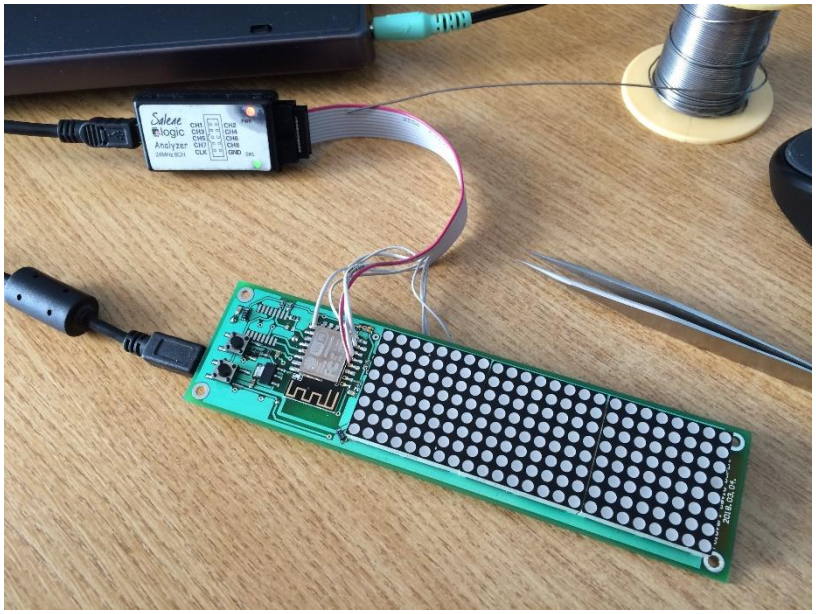


Majd az eszközök menü alatt az alábbi beállítások után kezdődhet a programozás.

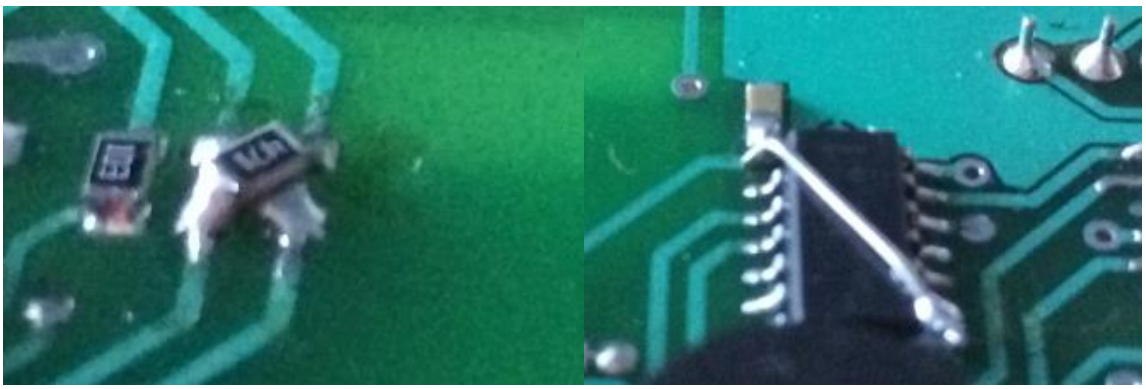
Rögtön ezt követően kipróbáltam, hogy működik-e a programozás. Nyitottam egy üres projektet és megpróbáltam flashelni, de nem sikerült feltölteni a programot az eszközre. A következő üzenet fogadott.

```
trying to connect
        flush start
        setting serial port timeouts to 1 ms
        setting serial port timeouts to 1000 ms
        flush complete
        espcomm_send_command: sending command header
        espcomm_send_command: sending command payload
        read 0, requested 1
warning: espcomm_sync failed
error: espcomm_open failed
error: espcomm_upload_mem failed
```

Nem tudtam, hogy mi a probléma ezért egy logikai anailizátorral megvizsgáltam az UART csomagokat, hogy egyáltalán megérkeznek-e az ESP lábára.



Kiderült, hogy elkövettem egy tervezési hibát, pontosabban kettőt. Az egyik, hogy a Tx-et a Tx_D-re kötöttem az Rx-et pedig az Rx_D-re kötöttem. Ezt úgy oldottam meg, hogy keresztben egymáson átívelve helyeztem el a 2 adatvezetékre csatlakozó ellenállást.



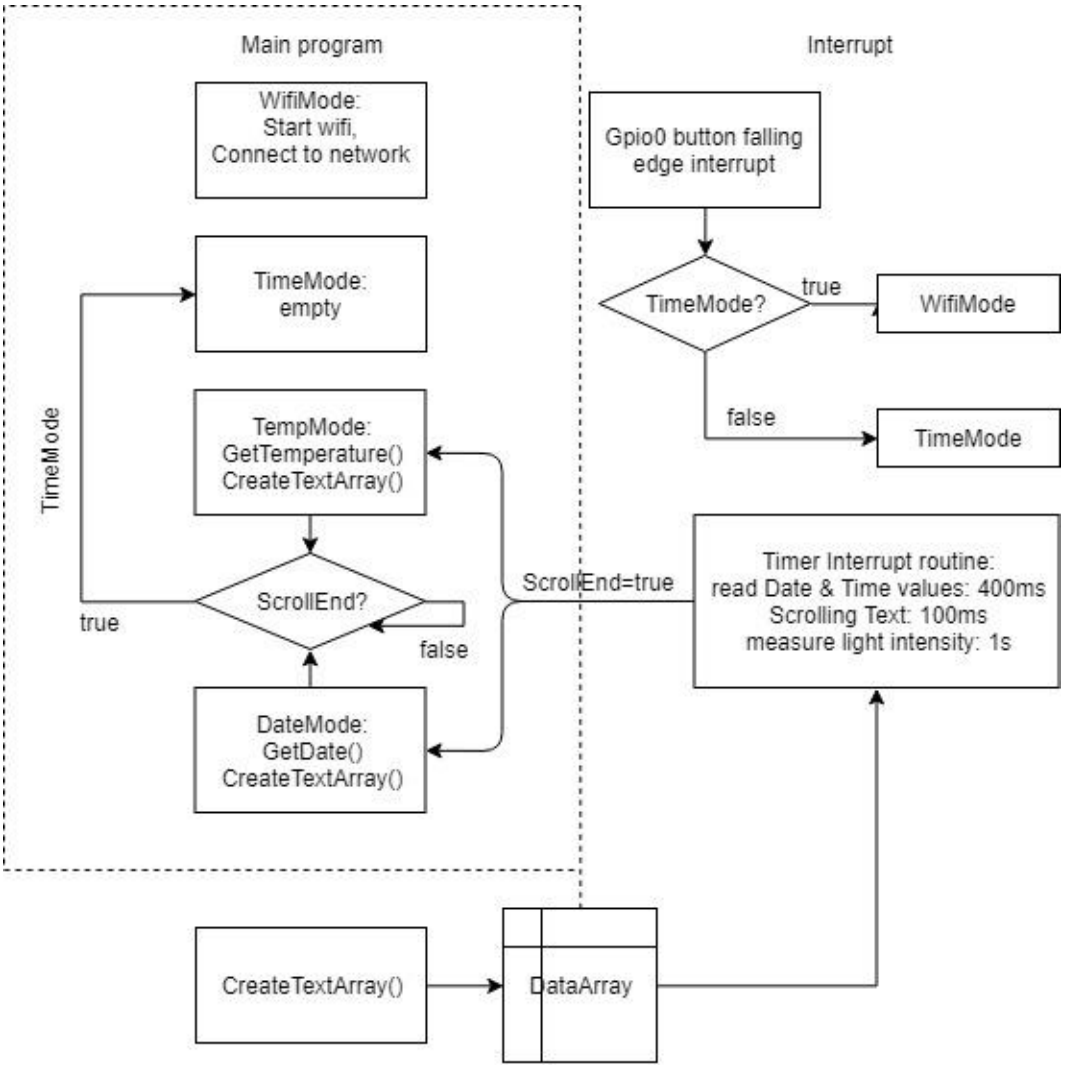
A másik hibát a logikai szintillesztő IC OE lábánál követtem el. GND helyett 3.3V-ra kell kötni, ezt egy alkatrész lábbal az alkatrész felett átívelően oldottam meg.

```
..................................................................................... [ 29% ]
..................................................................................... [ 58% ]
..................................................................................... [ 87% ]
................................                                                    [ 100% ]
starting app without reboot
        espcomm_send_command: sending command header
        espcomm_send_command: sending command payload
        espcomm_send_command: receiving 2 bytes of data
closing bootloader
        flush start
        setting serial port timeouts to 1 ms
        setting serial port timeouts to 1000 ms
        flush complete
```

A hiba kijavítása után már tökéletesen működött a programfeltöltés. Elkezdhettem írni a programot.

6. A működtető szoftver.

Az alap elképzelésem az volt, hogy a dátum és idő beállítása történjen egy HTML kérésen keresztül. Azt akartam, hogy az óra működjön egyszerű óraként, azaz amikor tápfeszültséget kap, akkor jelenítse meg az időt. A dátumot és a hőmérsékletet percenként egyszer jeleztetem ki felváltva. Mindent egyszerre kijelezni felesleges lenne és az idő mutatása fontosabb, mint a dátum vagy a hőmérséklet, így ezek az információk egyszer végigfutnak a kijelzőn és újra az időt mutatja.



A program blokkvázlata.

Az RTC és hőmérő IC-hez szerencsére sikerült találni könyvtárat, így azokból tudtam használni a számomra szükséges funkciókat.

Arduino IDE-ben az inicializáló kódot a setup blokkban kell megvalósítani. Minden külső és belső perifériát itt kell beállítani. Beállítok egy belső timer megszakítást 25milliszekundumra és egy GPIO0 lefutó éllel triggerelt megszakítást. Mivel egy timer megszakításom van, ezért elágazásokkal többfelé bontom. A fényszenzor analóg konverzióját másodpercenként végzem, szöveg futtatásakor a kijelző 0.1 másodpercenként lép a következő oszlopra, az RTC regisztereit pedig 0.4másodpercenként kérdezem le, így egy másodperc alatt három lekérdezés is megtörténik.

```
void ICACHE_RAM_ATTR onTimerISR() {
  scrollInterruptCounter += 1;
  adcInterruptCounter += 1;
  rtcInterruptCounter += 1;

  if (scrollInterruptCounter == 4) { //100ms megszakítás
  if (rtcInterruptCounter == 16) { //400ms megszakítás
  if (adcInterruptCounter == 40) { //1s megszakítás
  timer1_write(125000);//25ms intervallum
}
void handleInterrupt() {
  if (DisplayMode == WifiMode)
    DisplayMode = TimeMode;
  else DisplayMode = WifiMode;
  if (WiFiIsOn && DisplayMode == TimeMode) {
  }
}
```

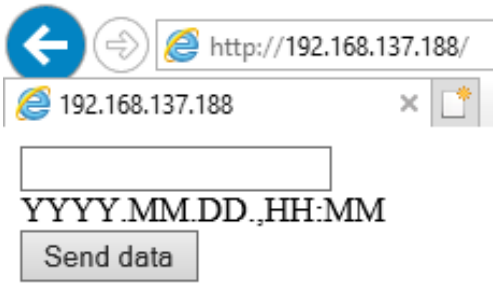
A megszakítás több részre bontása.

A kijelzési és programvégrehajtási állapotokat egy vagy több jelzőváltozóval oldottam meg. Erre mindenképp szükség volt a megszakítás és a főprogram közötti kommunikációhoz.

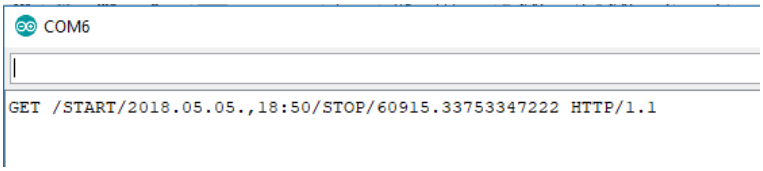
A főprogram végtelen ciklusa Arduino IDE-ben loop()-ként van jelölve. Jelen esetben a végtelen ciklusba négy elágazás került a négy lehetséges állapothoz (WifiMode, TimeMode, DateMode, TempMode). Alap esetben a program TimeMode, azaz idő kijelzéssel indul el. Minden perc 30. másodperce után egy futó szövegként vagy dátumot vagy a hőmérsékletet jeleníti meg, majd ha a szöveg véget ért, visszatér idő kijelzésére. A dátum és a hőmérséklet kijelzése felváltva történik percenként egyszer.

A környezeti fényerősség mérését másodpercenként végzem, majd ebből exponenciális szűréssel áll elő a kijelzőre adott fényerő nagysága. Egyszerű méréseknél az volt a tapasztalat, hogy nagyon sűrűn változik a fényerő ezért a konzulensem adott egy tanácsot, hogy végezzek rajta átlagoló szűrést. mintás szűrést.

Amikor megnyomjuk a GPIO0-ra csatlakozó lábat, akkor válthatunk idő és wifi mód között. Ez azt jelenti, hogy a megadott ssid azonosítóval és jelszóval csatlakozik a hotspotra és a kiosztott IP címet megjeleníti a kijelzőn futószövegként. Ezt beírva a böngészőbe egy egyszerű HTML oldal jelenik meg, amit az ESP-n futó szerver küld a böngészőnek.



Ebben a módban van lehetőségünk egy meghatározott formátum szerint a dátumot és az időt beállítani. A helyes formátum YYYY[pont]MM[pont]DD[pont][vessző]HH[kettőspont]MM. A feldolgozás egy egyszerű if-es szerkezetben vizsgálja, hogy az érkezett adat megfelel-e a formátumnak. Ha nem, nem történik semmi, hiszen hibás formátumot nem fogadunk el, és elképzelhető, hogy valaki figyelmetlenségből írta el. Ilyenkor újra próbálkozhatunk tetszőleges alkalommal. Ha megfelelő formátum érkezett, akkor az adatot fel kell dolgozni és el kell menteni az RTC memóriájába.



A kapott adatot egy keretbe rakom, hogy egyszerűbb legyen a feldolgozása. Azt úgy végzem, hogy a START/ után, a 11. karaktertől átmásolok 17 karaktert, így a tmpbuf-ban csak a dátum és idő karakterei leszek benne.

```
char tmpbuf[18];
tmpbuf[17]='\0';
for(uint8_t i=0;i<17;i++){
  tmpbuf[i]=request[11+i];
}
int year, month, day, hour, minute;
sscanf(tmpbuf,"%d.%d.%d.,%d:%d",&year,&month,&day,&hour,&minute);
DateTime SetDateTime(year,month,day,hour,minute);
rtc.adjust(SetDateTime);
```

Ezt sscanf() függvénnyel változókba írom, figyelve a szeparáló karakterekre. Majd a DS3231-hez letölthető Arduino könyvtár által biztosított objektumot létrehozom az adataimmal és az rtc.adjust() tagfüggvénnyel elmentem az RTC-be a beírt értéket.

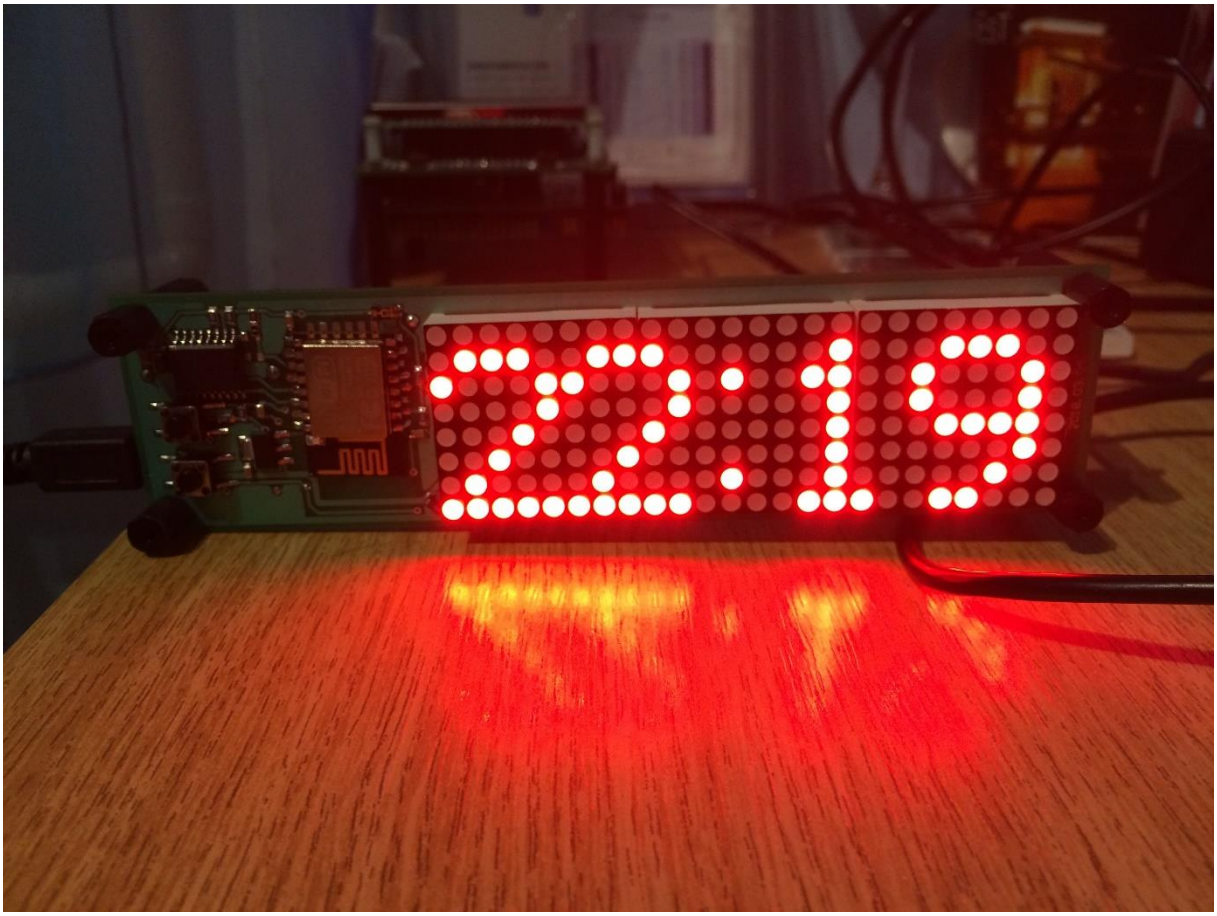
A szövegek kiírásáért a CreateTextArray() függvény felel.

```
void CreateTextArray(char *From) {
  ScrollText = false;
  TextLength = 0;
  while (From[TextLength] != '\0') {
    TextLength++;
  }
  for (uint8_t i = 0; i < TextLength; i++) {
    for (uint8_t j = 0; j < 6; j++) {
      DisplayDataArray[(i * 6) + j] = BitSwapping(characters[From[i]][j]);
    }
  }
  StartFrom = 0;
  ScrollEnd = false;
  ScrollText = true;
}
```

A megoldás lényege, hogy egyszerű szövegként lehet a kijelzőre írni szöveget, adatot, speciális karaktereket. Ezt úgy valósítottam meg, hogy a projekt tartalmaz egy characters.c és egy characters.h fájlt, amelyben a 8 bites ASCII karakterek karaktertáblázata található.

Minden karakter több oszlopból áll, maximálisan 256 karakteres szöveget lehet létrehozni. Ehhez a 256 karakteres szöveghosszúsághoz 1536 adatoszlop tartozhat, ez a DisplayDataArray. Ezeket a

program elején allokálom fix méretű char tömbként. A függvény, kommunikál a megszakítás scrollozásért felelős részével a ScrollText bool típusú változón keresztül. Ez után elkezdhetünk dolgozni a tömbön. A szöveghossznak megfelelően ki kell olvasni a karakterek oszlopainak értékét és eltárolni a DispayDataArrayben. Ehhez szükség van még a BitSwapping függvényre amely az oszlopadatok adattárolási módjából, (fentről lefelé:MSB…LSB) az SPI küldéshez igazítja a bájtokat. Ez egyszerű bitműveletekből áll, bitek sorrendjének cseréjéből. Az adattömb létrehozása után alaphelyzetbe állítjuk a scrollozásért felelős jelző és számláló változókat, amely után a megszakítás elvégzi a szöveg léptetését és kiírását.



Az elkészült áramkör működés közben.

7. Tapasztalatok.

A feladat megoldása során sok új dologgal ismerkedtem meg. Az ESP modulok eddig számomra nem voltak ismertek. Az SPI és I2C buszt ismertem ez előtt is ahogy az UART kommunikációt. Az Arduino IDE korlátoltságai miatt, logikai analizátoros sorosportos kiírásos debug módszereket kellett használnom, ami nagyon megnehezítette a hibák feltárását. Nem lehet regiszterek értékét olvasni vagy a kódot soronként léptetni. Lényegében az is egyfajta tapasztalat, ha a későbbiekben nem szeretném az Arduino környezetét használni.

8. Továbbfejlesztési lehetőségek.

A hardver hibáit természetesen kijavítani a terveken, a felesleges alkatrészeket leszedni róla, esetleg 3 kijelző helyett 4 kijelzővel megtervezni, hogy a számok ne érjenek össze. Az ESP8266 képes arra, hogy a weboldal HTML fájljait egy SD kártyáról olvassa be. Így szebb és komolyabb weboldalakat lehetne létrehozni.

9. Szükséges módosítások.

- az LM75 hőmérő szenzort nem a hőtermelő ESP8266 modul másik oldalára tervezném
- ESP8366 helyett inkább egy STM32 ARM mikrokontrollerre tervezném meg a kapcsolást, az ESP csak UART slave eszközként AT parancsokkal lenne használva.
- Az Arduino IDE-t valamilyen módon kikerülném, mert komolyabb fejlesztésekre nem alkalmas.

Földvári Dávid

2018