
**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar
Méréstechnikai és Információs Rendszerek Tanszék

Koren Zoltán

# ESP32-S2 ÉS KÜLSŐ GPS SZOFTVERES ILLESZTÉSE

Önálló Laboratórium

KONZULENS

Dr. Kovácsházy Tamás

BUDAPEST, 2021

# Tartalomjegyzék

# 1 Bevezetés

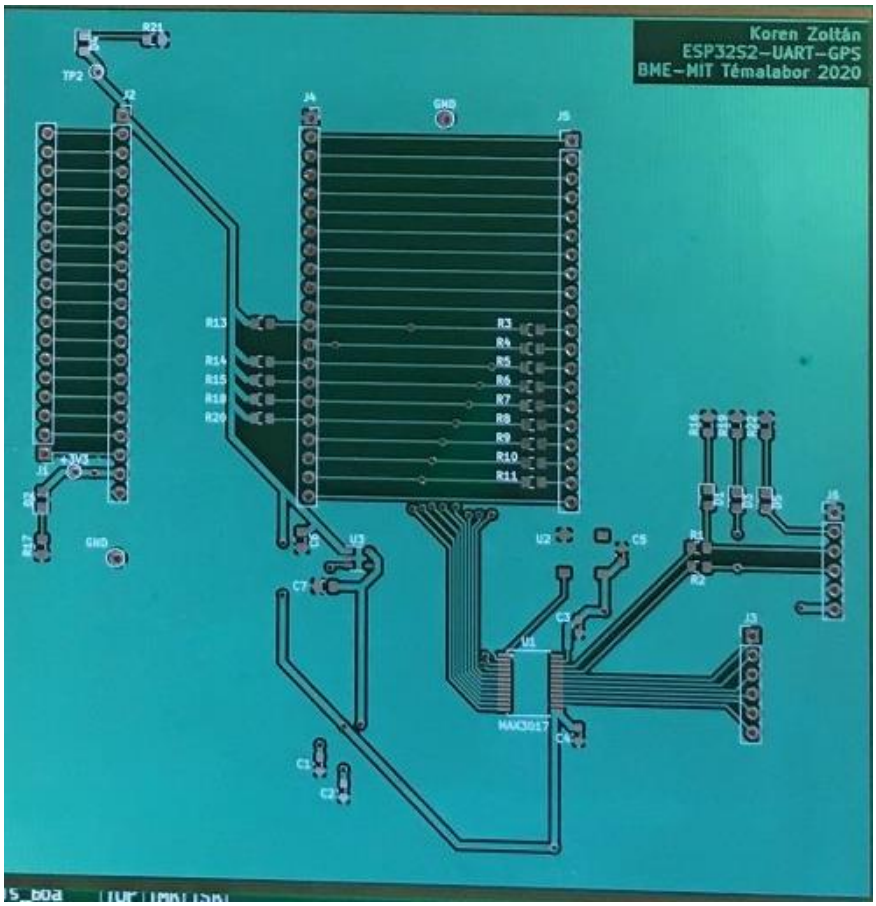
A témalabor során elkészült a soros kommunikáción keresztül a GPS-t az ESP32-S2 fejlesztő kártyához csatlakoztató hardver. Az önálló labor során a feladatom, hogy elkészítsem és teszteljem a rendszert működtető szoftvert.

## 1.1 Témalaboratóriumi munka

A témalabor folyamán a feladatom egy olyan NYÁK-nak az elkészítése volt, ami képes egy soros interfész kialakítására az ESP32-S2-Saola-1-es kártya[1], illetve egy külső GPS között. Az ESP32-S2 csak egy UART chippel rendelkezik, amit elhasznál arra, hogy a PC-vel USB-s kapcsolatot létesítsen (ezen keresztül lehet a kártyát felprogramozni), így kénytelen voltam másik módszert alkalmazni a kapcsolat kialakítására.

**1. ábra: ESP32-S2-Saola-1**

A választás végül a Maxim Integrated MAX3107 UART [2] chipjére esett, amivel az ESP32-S2 I2C és SPI módon is képes kommunikálni. A témalaboros munkám eredménye a 2. ábrán látható áramkör, az önálló labor során ezen a NYÁK-on dolgoztam tovább.




**2. ábra: Témalabor NYÁK**

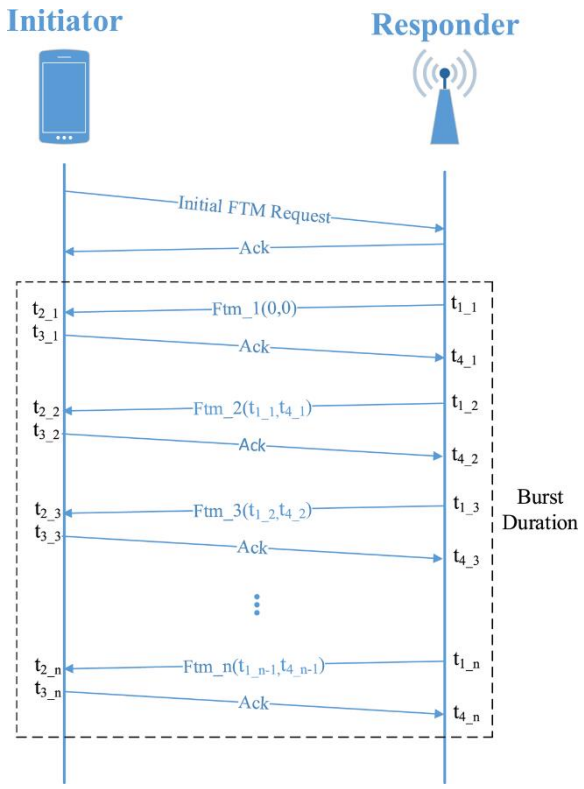
# 1.2 Célok

## 1.2.1 GPS modulok

A GPS modulok nagyon sok információt tudnak szolgáltatni. Köztük talán a legfontosabbak a helyadatok, illetve a kisugárzási idők. Ezeknek az adatoknak több felhasználási módja is létezik: beágyazott rendszerekben a szenzorhálózatokon belül egyes egységeknek az óraszinkronizációját lehet ezzel elvégezni, hiszen ilyenkor egy Master egység órájához szinkronizálnak az egyes egységek. A GPS-eknek más felhasználási módja is lehet, például autókban a navigáción kívül a sebesség figyelését is elvégezhetik. Ez azért egy kedvező felhasználás, hiszen, ha arra leszünk figyelmesek, hogy a sebesség hirtelen lecsökkent 0-ra, akkor feltételezhetjük, hogy valamilyen baleset történt. Ha megállapítottuk a sebességnek ilyen hirtelen lecsökkenését, akkor a mentők, és egyéb szervek azonnal értesülnek, így életmentő perceket lehetne nyerni ilyen helyzetekben.

### 1.2.2 WiFi Fine Time Measurement [3]

A cél, hogy ezekkel a GPS-es modulokkal olyan csomópontokat alakítsak ki egy mikrokontrollerrel, ami a vett adatokat feldolgozva képes méréseket végezni. Az IEEE 802.11-2016 tartalmazza a WiFi Fine Time Measurement (továbbiakban: FTM) protokolt, ami egy olyan mérési módszer, ahol azt mérjük, hogy egy WiFi jel mennyi idő alatt ér el egy Access Pointhoz (továbbiakban: AP), egy Stationtől (továbbiakban: STA). Az idők ismeretében már a távolságok is kiszámíthatóak (~3.3 ns/m).




**3. ábra: FTM mérés**

Ha a mérést több modullal is elvégezzük, akkor a kiadódó időkből (és távolságokból) kiszámítható egy „*initiator*” pozíciója. A feladat során ennek a környezetnek az alappilléreit szándékozom elkészíteni: a csomópontokat.

A fejlesztő kártya megválasztása is ezért volt fontos, ugyanis az ESP32-S2 rendelkezik egy WiFi modullal, így képes a mérés elvégzésére. Az Espressif saját fejlesztőkörnyezete továbbá biztosít példakódokat is, köztük az FTM-es mérésekhez is található kód. Az ESP32-S2 támogatja tehát az FTM-es méréseket, képes STA és AP módban is működni.

# 2 Önálló Laboratóriumi munka

## 2.1 Hardver

Ugyan az önálló laboratóriumi témám a szoftveres illesztést nevezni meg célnak, nem lehet elmenni amellett, hogy a távolléti oktatás miatt a témalaboros munkámat nem tudtam befejezni a jelenléti lehetőségek hiányában. Így a hátramaradó feladatok elvégzését az önálló laboratórium során kellett megtennem.

### 2.1.1 Forrasztás

Önálló laboratórium előtt még soha nem volt lehetőségem forrasztani, így eleinte kihívásként tekintettem a műveletre. Amikor a félév elején hozzá kezdtem a NYÁK már elkezdett oxidálódni, ami megnehezítette a beforrasztást. Az alkatrészek nagyrésze mind felületszerelt volt, az IC-k beforrasztása a kis lábméret miatt nehezebbnek bizonyult, mint ahogyan azt előzetesen felmértem.

### 2.1.2 Bemérés

Miután a forrasztással végeztem, elkezdődött az áramkör tesztelése, hogy kiderüljenek a tervezési, illetve forrasztási hibák, mielőtt a szoftver írásába neki kezdek. Az áramkör tesztelése során több súlyos probléma ütköztem, amik ellehetetlenítették az előrehaladást, így megoldás(oka)t kellett találnom.

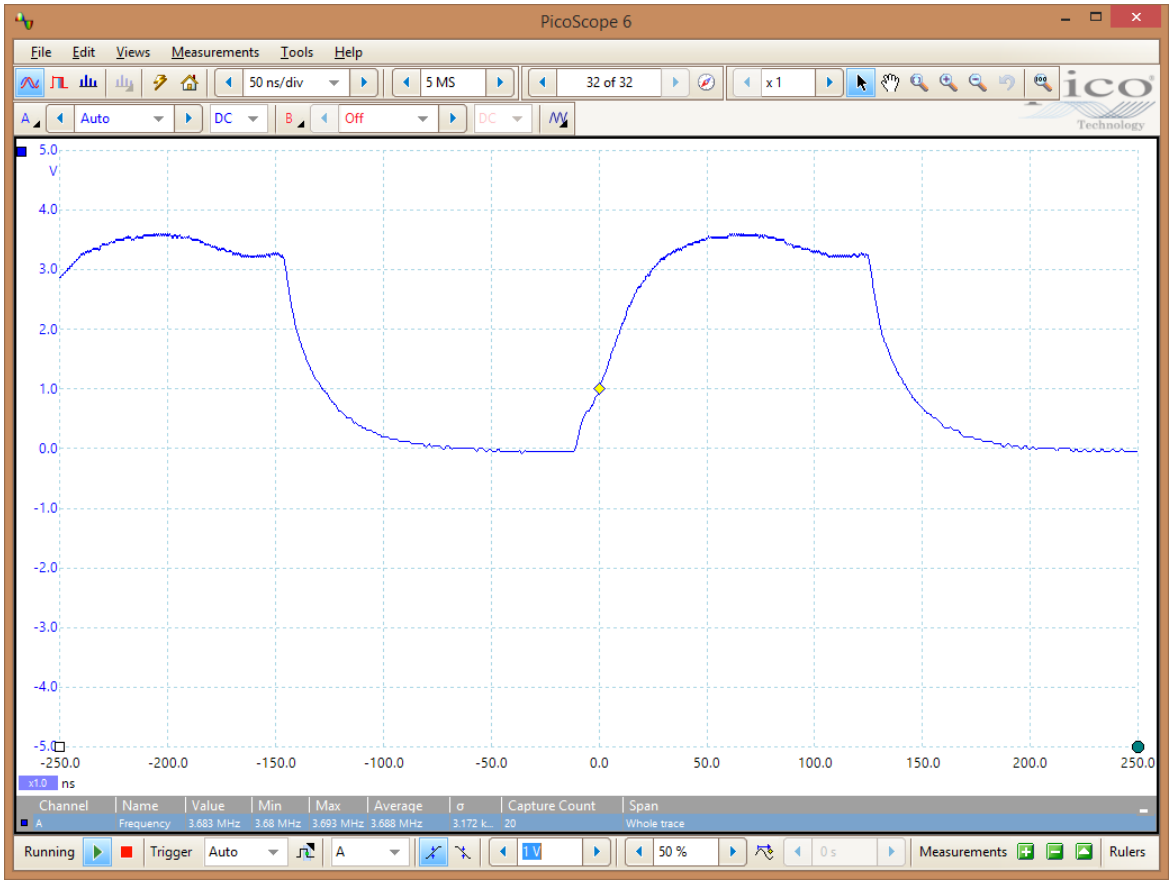
#### 2.1.2.1 Áramfelvétel

Miután a tápot rácsatlakoztattam a NYÁK-ra, és beállítottam a tápfeszültséget, egy nagyon furcsa hibára lettem figyelmes. Az áramkör áramfelvétele 10-szer többnek bizonyult, mint ahogyan az eredetileg tervezve lett, továbbá egyes helyeken rövidzárlatot is jelzett a műszer. A MAX3107 adatlapja szerint csak 1-2 mA szükséges a működéshez, illetve az egész áramkör a tervek szerint körülbelül 10 mA körüli áramfelvételt igényelne. Ennek ellenére, a mért áram 150 mA körüli érték volt. Az UART chip érintésre nagyon melegedett, ami mindenképpen a MAX3107-re terelte a gyanút.

#### 2.1.2.2 Oszcillátor

Egy másik probléma az volt, hogy az UART működéséhez szükséges kristályoszcillátor nem szolgáltat a kimenetén semmilyen jelet. Leszedve a NYÁK-ról,

és külön megmérve, viszont a megfelelő frekvenciájú négyszög jelet kaptunk. A lassú fel- és lefutások a mérés körülményeiből adódtak.




**4. ábra: Oszcillátor jele az oszcilloszkópon**

Erről arra következtettem, hogy a hiba nem a konkrét áramköri elemben keresendő, hanem a NYÁK-on van valami hiba, ami nem teszi lehetővé, hogy az oszcillátor kimenetén bármilyen feszültség megjelenjen.

#### 2.1.2.3 LDO Voltage Regulator

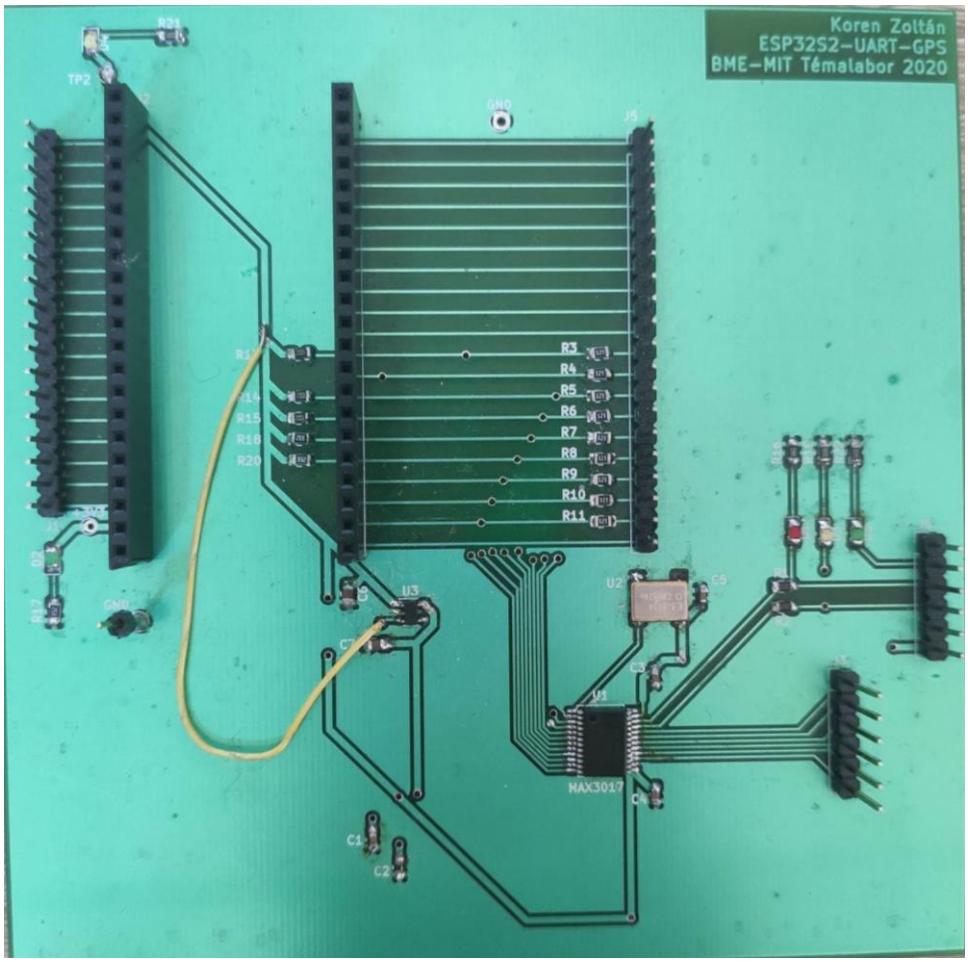
Az utolsó felmerülő hiba az LDO Voltage Regulatorrel kapcsolatos. Az UART chipnek szükséges 1.8V-ot szolgáltatni a bemeneténél, amit ezzel az elemmel érünk el. A kimeneten azonban nem jelenik meg az 1.8V, ami újabb kérdéseket vetett fel.

### 2.1.3 Megoldás a problémákra

A hibakeresés folyamán két konkrét hibát sikerült találni, ami végül megoldást jelentett a felmerülő problémákra. Az első egy tervezési hiba, amit még az előző félév folyamán követtem el: az LDO Voltage Regulator engedélyező bemenetét rosszul állítom elő, nem a tápfeszültséget kapja, hanem az ESP32-S2 egyik lábához van kötve. Bár a

problémát megoldaná az, hogy „éles” helyzetben az ESP32-S2 azon lába mindig magas logikai szintre van húzva, de ez nem célszerű megoldás. A vezetékezést elvágtam, és egy drót segítségével a tápfeszültséget odavezettem az engedélyező lábhoz.

A másik probléma egy gyártási hiba volt. Valószínűleg, amikor az egész áramkör először kapott tápot, a MAX3107 chip azonnal elsült, és emiatt okozhatott rövidzárlatot, továbbá ilyen hatalmas áramfelvételt az áramkörben. Kicserélésével normalizálódott a NYÁK áramfelvétele, és a melegedés is megszűnt. Az oszcillátor működése is megfelelő volt a 2 problémás alkatrész javítása után.



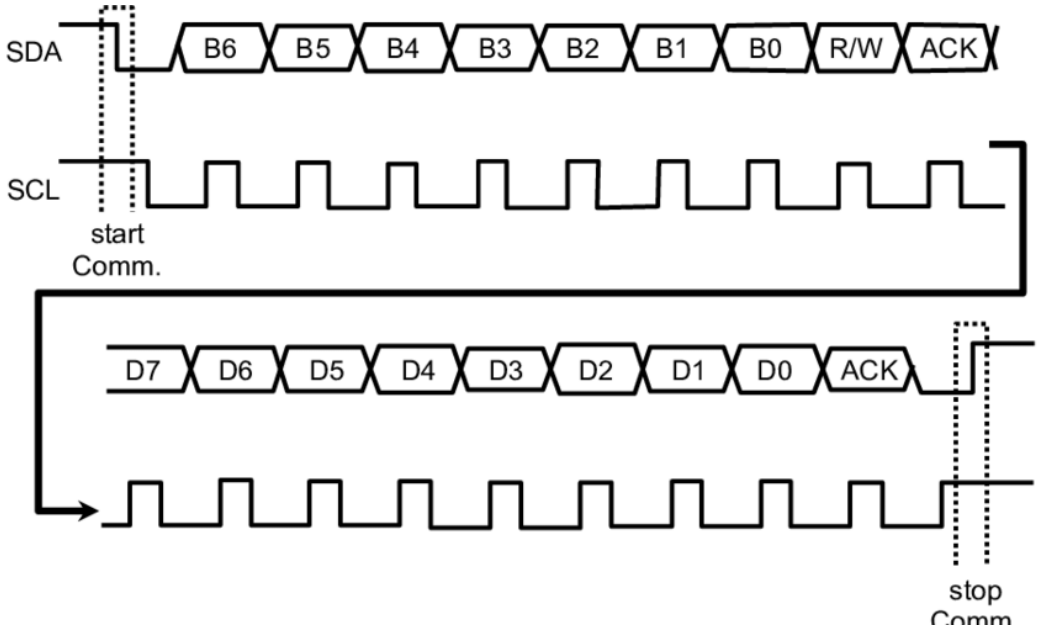

**5. ábra: A kész NYÁK**

## 2.2 Szoftver

Miután elkészültem a NYÁK-al, hozzáláttam a szoftver megírásához. Fejlesztőkörnyezetnek az Arduino IDE-t használtam, ugyanis a célnak pont megfelelő, továbbá egyszerű a használata. Az ESP32-S2 alapvetően nem támogatott, de internetes útmutatók alapján sikerült összehozni a két rendszert.

### 2.2.1 I2C

Az I2C egy olyan soros kommunikáció protokol, ahol az adatok bitről bitre kerülnek átadásra egy vezetéken (SDA) multi-master, illetve multi-slave elrendezésben. Az USART vagy SPI protokollokhoz hasonlóan az I2C-hez is szükséges órajel (SCL), ami azonos a vételi és adó oldalon. [4]




**6. ábra: I2C kommunikáció**

A 6.ábrán látható egy átlagos I2C üzenet. Ha nincsen kommunikáció, az I2C busz magas logikai szinten van, az I2C üzenet kezdetét meg a START bit jelöli, ami lehúzza alacsony logikai szintre a buszt. Ezután következik a címet jelölő biteknek a továbbítása. A cím 7-10 bites szokott lenni, ami azonosít egy slave egységet. Minden slavenek van egy egyedi azonosítója (cím), az I2C kommunikáció ezen szakaszában szólítjuk meg a konkrét slave egységet, a R/W bittel jelöljük, hogy olvasási vagy írási műveletet akarunk végezni, majd a slave egység visszajelez (ACK), ha a buszon rajta van és sikerült megtalálni. Ezután következnek az adatbitek, amik mindig 8 bit hosszúságúak. Minden adatot tartalmazó keret után a slave visszajelez, hogy vette-e az adást (ACK), vagy valamilyen hiba következett (NACK). Több adatkeret is elküldésre kerülhet, az utolsó adatkeret végén egy STOP bitnek kell elhelyezkednie, ami az SDA vonalat alacsony logikai szintről magas logikai szintre viszi.

## 2.2.2 Megvalósítandó feladatok

### 2.2.2.1 I2C Scan

Mielőtt komolyabb kódot írtam volna, egy I2C Scant futtattam le, hogy kiderítsem a MAX3107 UART címét. Ugyan a datasheet is tesz róla említést, de meg akartam bizonyosodni a teszt segítségével is.

**Table 5. I2C Address Map**

| DIN/A1 | $\overline{CS}$/A0 | READ/ WRITE | $I^2C$ ADDRESS |
|---|---|---|---|
| 0 | 0 | W | 0x58 |
| | | R | 0x59 |
| 0 | 1 | W | 0x5A |
| | | R | 0x5B |
| 1 | 0 | W | 0x5C |
| | | R | 0x5D |
| 1 | 1 | W | 0x5E |
| | | R | 0x5F |

**7. ábra: MAX3107 lehetséges címei**

A megfelelő lábakat úgy állítottam be az UART-on, hogy a címnek 0x58-nek kell lennie (hiszen az I2C scan során olvasni egy Write műveletet végzünk). A futtatás után a következő eredményt kapjuk:

```
14:58:45.436 -> Scanning...
14:58:45.483 -> I2C device found at address 0x2C
14:58:45.583 -> Device found!
14:58:45.583 -> Done
14:58:45.583 ->
```

**8. ábra: MAX3017 címe**

Arra a furcsaságra figyelhetünk fel, hogy a címek nem egyeznek meg. Ha felírjuk a két számot binárisan, az alábbiakat kapjuk:

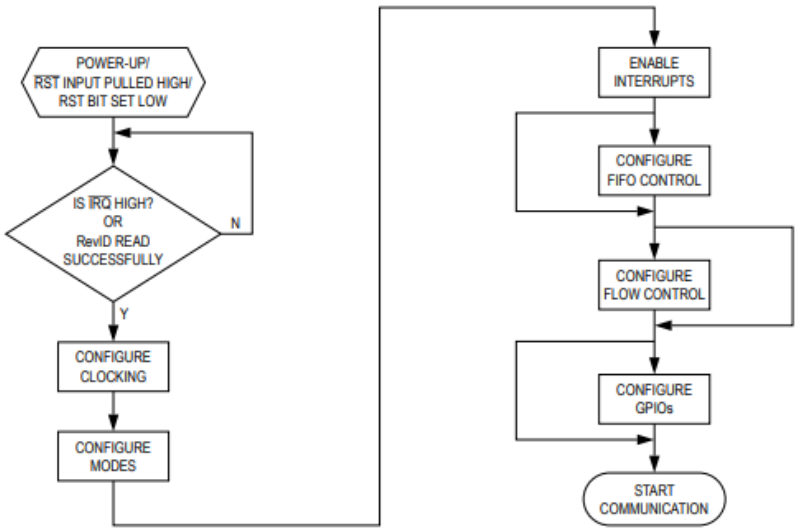
$$(0x58)_{HEX} = (0101\ 1000)_{BIN}$$

$$(0x2C)_{HEX} = (0010\ 1100)_{BIN}$$

Látható, hogy az adatlapi érték már magába foglalja a R/W bitet is az LSB-nél, míg az Arduino által szolgáltatott cím nem. Ha mért értéket egyszer balra shifteljük, akkor látható, hogy a címek megegyeznek.

### 2.2.2.2 Loopback

A vétel, illetve az adás ellenőrzésére egy loopback tesztet szerettem volna elkészíteni, ami azt jelenti, hogy az UART RX és TX vonalait össze szándékozom kötni. Így a küldött adatokat azonnal vissza tudom olvasni, és amennyiben nincsenek eltérések, jól működik egyrészt a chip, másrészt a program. A loopback teszthez már be kell állítani megfelelően az UART chipet, a megfelelő inicializálási lépéseken végig kell menni.




**9. ábra: UART inicializálás folyamatábrája**

Az inicializálási lépéseken végig menve érhetjük el azt, hogy az UART megfelelően működjön a későbbiekben. Miután a hardveres, illetve szoftveres reseteket elvégeztük, és megbizonyosodtunk róla, hogy feléledt a chip, elkezdődhet az UART belső regisztereinek írása.

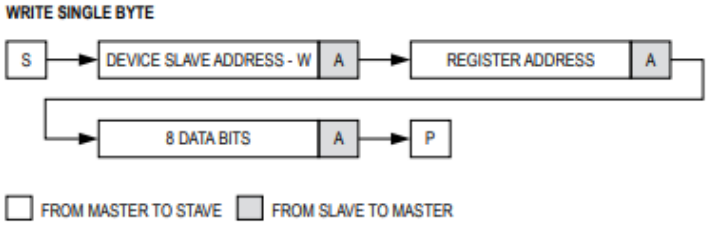
Az órajelhez köthető regisztereknél jelezni kell az UART számára, hogy egy külső oszcillátor szolgáltatja az órajelet, továbbá az adatlapon található információknak megfelelően a BaudRate-et is be kell állítani. Mivel a GPS modul 9600 BaudRate-el fog majd adni, ezért érdemes már most az UART-ot így felprogramozni.

$$D = \frac{f_{REF}}{16 \cdot BaudRate}$$

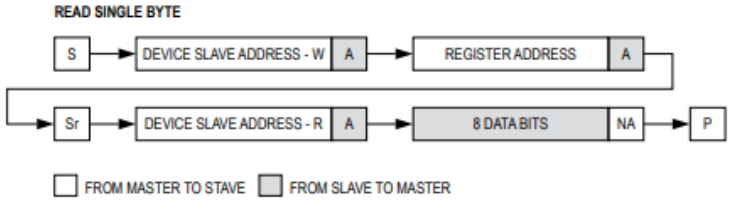
$$D = \frac{3.6864\ MHz}{16 \cdot 9600} = 24$$

Mivel 3.6864 MHz-es a külső oszcillátor által szolgáltatott órajel, ezért annyit kell a képletbe behelyettesíteni. D értéke 24 lett, így az UART megfelelő regiszterébe 24-et kell írni.

Flow Controlra, illetve a GPIO lábakra nincsen szükség a feladat megvalósítása során, így ezeket az inicializálási lépéseket ki lehet hagyni.

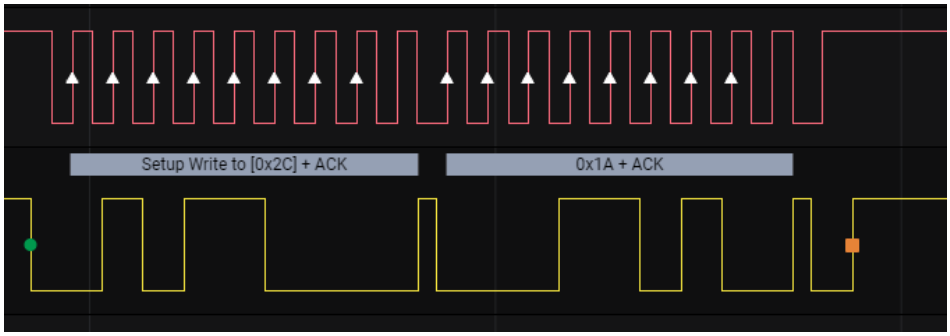



**10. ábra: I2C írási művelet**

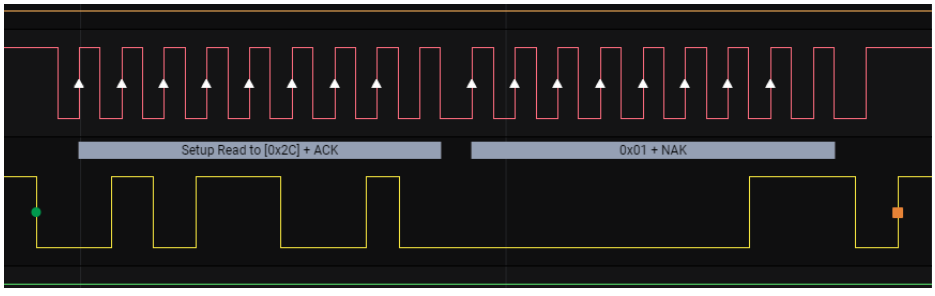



**11. ábra: I2C olvasási művelet**

Az I2C kommunikációt a 10. és 11. ábrának megfelelően készítettem el. Miután megcímzem az UART egységet, a következő keretnek az UART belső regiszterét kell tartalmaznia. Adás-vétel esetén ez a 0x00, de más regiszterek is elérhetőek kívülről. Ezek után jöhetnek az adatkeretek.

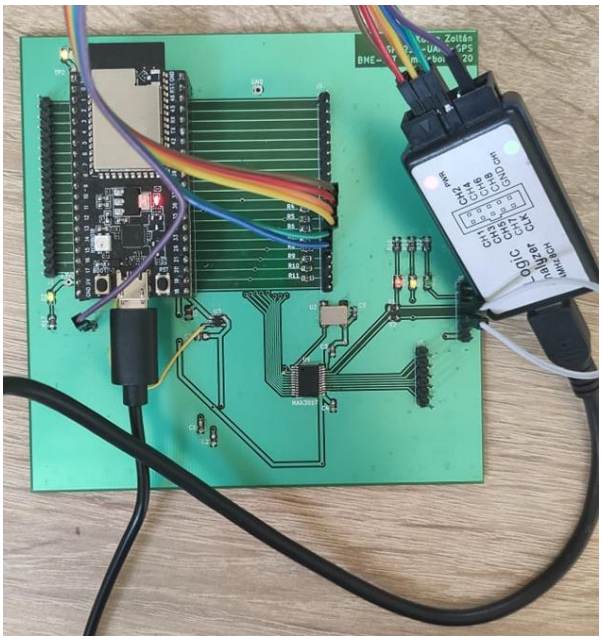



**12. ábra: PLLConfig regiszter megcímzése**




**13. ábra: PLLConfig regiszter tartalmának olvasása**

Ahogyan az a logikai analizátorral felvett ábrákon is látszik, sikeresen tudtam az UART belső regiszterét kiolvasni, ugyanis PLLConfig regiszter értékén nem változtattam a felprogramozás során, és az alapértelmezett értéket tudtam kiolvasni.



**14. ábra: Mérési elrendezés a loopbackhez**

```
17:02:32.920 -> Sending: 170
17:02:32.920 -> I2C_Write Running
17:02:32.920 -> Received:
17:02:32.920 -> 170
```

**15. ábra: Sikeres loopback**

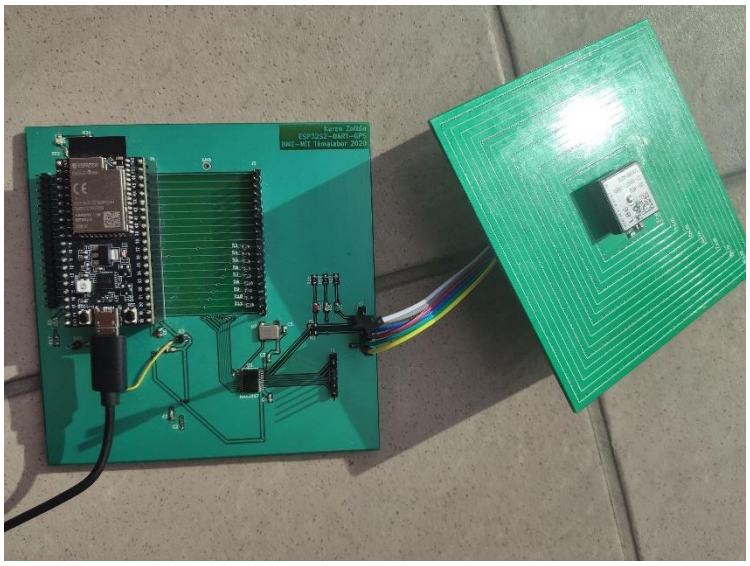
A Loopbacket elvégeztem szoftveresen (Mode regiszter írásával lehet szoftveres loopbacket használni), illetve hardveresen is (vezetékkel összekötöttem RX és TX vonalakat) és a vártaknak megfelelően működött.

### 2.2.2.3 GPS illesztés

A projekt következő fázisa a konkrét illesztés megvalósítása. Eleinte azt hittem, hogy elegendő lesz a már megírt függvények használata, azonban ez nem így lett. Miután a GPS modult rákötöttem az UART RX, TX vonalaira nem volt semmi vétel, így kénytelen voltam az eddig írt függvényeket átírni. Végül már sikerült adatokat kapnom, azonban csak a számok jöttek át, a karakterek már nem.

```
18:40:01.272 -> $□□□□,164000.000,□,4727.4421,□,019,1*,,0□.28.,3,81□,812,61,6,2□898,,030,□4.041□2
18:40:02.267 -> $□□□□,164001.000,□,4727.4421,□,019□5□□0□14.2,9,,8*□,8*2964,7,,,663923,11
18:40:03.163 -> $420*0□*
18:40:03.263 -> $□□□□,164002.000,□,4727.4421,□,0190,,□,,
```

**16. ábra: Hibás GPS adatok**



**17. ábra: Mérési elrendezés a GPS illesztéshez**

Eleinte a BaudRate hibára gyanakodtam, így megtettem az ezzel kapcsolatos lépéseket.


**18. ábra: Blokkdiagram a BaudRate megméréséhez**

| | Type | Start | Duration | data |
|---|---|---|---|---|
| ■ | data | 600.77625 ms | 989.583 µs | $ |
| ■ | data | 601.818167 ms | 989.583 µs | G |
| ■ | data | 602.860125 ms | 989.583 µs | P |
| ■ | data | 603.902042 ms | 989.583 µs | R |
| ■ | data | 604.943958 ms | 989.583 µs | M |
| ■ | data | 605.985917 ms | 989.583 µs | C |
| ■ | data | 607.027833 ms | 989.583 µs | , |
| ■ | data | 608.06975 ms | 989.583 µs | 1 |
| ■ | data | 609.111708 ms | 989.583 µs | 4 |
| ■ | data | 610.153625 ms | 989.583 µs | 4 |
| ■ | data | 611.195542 ms | 989.583 µs | 7 |
| ■ | data | 612.2375 ms | 989.583 µs | 3 |
| ■ | data | 613.279417 ms | 989.583 µs | 6 |
| ■ | data | 614.321333 ms | 989.583 µs | . |

**19. ábra: Logikai analizátor által mért adatok**


**20. ábra: Bitidő**

Mivel tudjuk, hogy a bitidő reciproka a BaudRate:

$$\frac{1}{104.208\mu s} = 9596.1922$$

Névlegestől való eltérés:

$$\frac{9600 - 9596.1922}{9600} \cdot 100 = \ 0.0397\ \%$$

Látszik, hogy az eltérés minimális, így elhanyagolható. Végül a megoldást nem a BaudRate-ben kellett keresni, hanem az UART regisztereiben. Az UART alapértelmezetten 7 bit hosszúságú szavakat várt, így ezért nem lehetett a karaktereket megkapni, mert az utolsó bitet mindig levágta a vételkor. Miután az UART megfelelő regiszterében jeleztem, hogy 8 bit hosszúságú szavakat kell mintavételeznie, már a karaktereket is megkaphattam a vételi oldalon.

```
23:06:23.221 -> $GNRMC,2000010623.000,A,4727.4452,N,01902.8904,E,0.92,315.90,210521,,,A*7A
23:06:23.287 -> $GPVTG,315.90,T,,M,0.92,N,1.71,K,A*3F
23:06:23.321 -> $GPGGA,210623.000,4727.4452,N,01902.8904,E,1,6,2.53,113.0,M,41.1,M,,*53
23:06:23.420 -> $GNGSA,A,3,06,17,19,09,,,,,,,,,2.70,2.53,0.96*13
23:06:23.454 -> $GNGSA,A,3,67,81,,,,,,,,,,,2.70,2.53,0.96*1A
23:06:23.520 -> $GPGSV,3,1,11,04,88,057,,03,58,095,,09,53,242,23,06,38,307,20*7E
```

**21. ábra: GPS adatok**

A vett adatok még nem teljesen tökéletesek. Vannak szekvenciák, amik többnyire jók, azonban vannak olyan adatok, amiknél egy-egy bit felcserélve érkezik meg az ESP32-S2 számára. Az önálló laboratórium keretein belül már nem volt lehetőségem rájönni, hogy mi a hiba, ennek ellenére a jelenlegi állapot is már egészen kedvező eredményeket mutat.

A 21. ábrán látható NMEA [5] adatokból már konkrét dolgok leolvashatóak, például a dátum (GNRMC üzenet vége fele 210521), helyadatok (GNRMC üzenet

közepe: 4727.4452 North, 01902.8904 East), a pontos idő UTC-ben (GPGGA üzenet eleje: 210623).

# 3 A jövő

Az önálló laboratóriumi feladat sokféleképpen továbbfejleszthető, akár szakdolgozat formájában is.

## 3.1 MAX3107 könyvtár

A szoftver elkészítése során arra lettem figyelmes, hogy még senki sem írt ehhez a chiphez könyvtárat, ami I2C kommunikációt tett volna lehetővé. A terveim közt szerepel az, hogy egy ilyet elkészítsek, és mindenki számára elérhetővé tegyek.

## 3.2 NYÁK újra tervezése

A bemérés során több tervezési hibára lettem figyelmes (vannak, amik a működést nem befolyásolják, azonban lényeges szempontok: például a NYÁK mérete), amiknek a kijavítása elengedhetetlen ahhoz, hogy a projektet bármilyen formában a jövőben folytatni lehessen.

## 3.3 FTM

Miután megtörtént az újra tervezés, és a további csomópontok legyártása elkezdődhet az FTM-es méréseknek a lebonyolítása, pozíciólokalizációs mérések elkészítése.

# Irodalomjegyzék


[1] https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/hw-reference/esp32s2/user-guide-saola-1-v1.2.html

[2] https://www.maximintegrated.com/en/products/interface/controllers-expanders/MAX3107.html

[3] https://www.researchgate.net/figure/FTM-Protocol-Overview_fig2_328327674

[4] https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[5] https://www.trimble.com/OEM_ReceiverHelp/V4.44/en/NMEA-0183messages_MessageOverview.html

# Függelék

### i2c_scan.c

```
#include <Wire.h>
#define SDA_PIN 33
#define SCL_PIN 26
#define I2C 35
#define A0 21
#define A1 20


void setup() {
  pinMode(I2C,OUTPUT);
  pinMode(A0,OUTPUT);
  pinMode(A1,OUTPUT);
  digitalWrite(I2C,LOW);
  digitalWrite(A0,LOW);
  digitalWrite(A1,LOW);
  Wire.begin(SDA_PIN,SCL_PIN);
  Serial.begin(115200);
  while(!Serial);
  Serial.println("\nI2C Scanner");
}

void loop() {
  byte error, address;
  bool device = false;
  Serial.println("Scanning...");
  for (address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if(error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if(address<16)
      {
        Serial.print("0");
      }
      Serial.println(address,HEX);
      device = true;
    }
    else if (error == 4)
    {
      //Serial.print(error, DEC);
      Serial.print("Unknown error at address 0x");
      if(address<16)
      {
        Serial.print("0");
      }
      Serial.println(address,HEX);
    }
  }

  if(device==true)
```

```
  {
    Serial.println("Device found!");
  }
  else
  {
    Serial.println("No I2C devices found");
  }
Serial.println("Done\n");
delay(5000);
}
```

**loopback.c**

```
#include <Wire.h>
#define SDA_PIN 33
#define SCL_PIN 26
#define I2C 35
#define A0 21
#define A1 20
#define RST 37
#define IRQ_PIN 19

/* Register map definitions */
#define MAX3107_RHR           (0x00)
#define MAX3107_THR           (0x00)
#define MAX3107_IRQEN         (0x01)
#define MAX3107_ISR           (0x02)
#define MAX3107_LSRINTEN      (0x03)
#define MAX3107_LSR           (0x04)
#define MAX3107_SPCLCHRINTEN (0x05)
#define MAX3107_SPCLCHARINT   (0x06)
#define MAX3107_STSINTEN      (0x07)
#define MAX3107_STSINT        (0x08)
#define MAX3107_MODE1         (0x09)
#define MAX3107_MODE2         (0x0a)
#define MAX3107_LCR           (0x0b)
#define MAX3107_RXTIMEOUT     (0x0c)
#define MAX3107_HDPLXDELAY    (0x0d)
#define MAX3107_IRDA          (0x0e)
#define MAX3107_FLOWLVL       (0x0f)
#define MAX3107_FIFOTRGLVL    (0x10)
#define MAX3107_TXFIFOLVL     (0x11)
#define MAX3107_RXFIFOLVL     (0x12)
#define MAX3107_FLOWCTRL      (0x13)
#define MAX3107_XON1          (0x14)
#define MAX3107_XON2          (0x15)
#define MAX3107_XOFF1         (0x16)
#define MAX3107_XOFF2         (0x17)
#define MAX3107_GPIOCONFG     (0x18)
#define MAX3107_GPIODATA      (0x19)
#define MAX3107_PLLCONFIG     (0x1a)
#define MAX3107_BRGCONFIG     (0x1b)
#define MAX3107_DIVLSB        (0x1c)
#define MAX3107_DIVMSB        (0x1d)
#define MAX3107_CLKSOURCE     (0x1e)
#define MAX3107_REVID         (0x1f)

/*********************************************************
** Startup with SW and HW reset
```

```
**
** Arguments: -
**
** return value:  -
*********************************************************/

void MAX3107_RST_Process()
{
  Serial.println("Reset process running");
  /*
    Wait for the MAX 3107 to come out of reset
    Wait for IRQ PIN to come up, as this
    indicates that the MAX3107 reset is complete
  */

  while (IRQ_PIN == 0);

  /* Perform a hardware reset */

  digitalWrite(RST, LOW);
  delay(1);
  digitalWrite(RST, HIGH);
  while (IRQ_PIN == 0);

  /* Perform a software reset */

  MAX3107_I2C_Write(MAX3107_CLKSOURCE, 0b00011000);
  MAX3107_I2C_Write(MAX3107_MODE2, 0x01);
  MAX3107_I2C_Write(MAX3107_MODE2, 0x00);
  while (IRQ_PIN == 0);

}

/*********************************************************
** Configuring Modes and Clocking
**
** Arguments:
** loopback: TRUE, if loopback should be enabled
**
** return value:  -
*********************************************************/

void MAX3107_I2C_Init(bool loopback)
{
  Serial.println("Initialization process running");
  /* Configure clocking */

  MAX3107_I2C_Write(MAX3107_CLKSOURCE, 0b00011000);
  MAX3107_I2C_Write(MAX3107_DIVLSB, 24);
  MAX3107_I2C_Write(0x0B, 3);


  /*Configure modes*/

  if(loopback)
    {
      MAX3107_I2C_Write(MAX3107_MODE2, 0b10100000); // Loop back
    }
  else
```

```
    {
      MAX3107_I2C_Write(MAX3107_MODE2, 0b10000000);
    }

  /*Enable Interrupts*/

  MAX3107_I2C_Write(MAX3107_IRQEN, 0b11111111);

  // Flow control - not needed
  // GPIOs - not needed
}

/*********************************************************
** Write one byte to the specified register in the MAX3107
**
** Arguments:
** port: MAX3107 register address to write to
** val: the value to write to that register
**
** return value:  TRUE – register successfully written
**                FALSE – I2C protocol error of some kind
**
*********************************************************/

bool MAX3107_I2C_Write(byte port, byte val)
{
  byte error;
  while (!Serial);
  Serial.println("\nI2C_Write Running");

  Wire.beginTransmission(byte(0x2C)); // Datasheet + I2C scan
  Wire.write(port);
  Wire.write(val);
  error = Wire.endTransmission();

  if (error != 0)
  {
    Serial.println("I2C communication failed");
    return false;
  }
  return true;
}

/*********************************************************
** Read one byte from the specified register in the MAX3107
**
** Arguments:
** port: MAX3107 register address
**
** return value:  Register containments
**
*********************************************************/

byte MAX3107_I2C_Read(byte port)
{
  uint8_t req;
  byte reading = 0;
  byte error;
  Wire.begin(SDA_PIN, SCL_PIN); // Wire comm. begin
```

```
  while (!Serial);

  Wire.beginTransmission(byte(0x2C)); // Datasheet + I2C scan
  Wire.write(port);
  error = Wire.endTransmission();

  if (error != 0)
  {
    Serial.println("I2C communication failed at #1");
    return 0b01010101;
  }

  Wire.requestFrom(44,1);
  if (1 <= Wire.available())
  {
    reading = Wire.read();
    Serial.println("Received: ");
    Serial.println(reading);
  }

  error = Wire.endTransmission();

  return reading;
}
/**************************************************************************
***************************************************************/
void setup() {
  /*
     PIN Definitions
  */
  Serial.begin(115200);
  Serial.println("I2C communication: Start");
  Wire.begin(SDA_PIN, SCL_PIN); // Wire comm. begin

  pinMode(I2C, OUTPUT);
  pinMode(A0, OUTPUT);
  pinMode(A1, OUTPUT);
  pinMode(RST, OUTPUT);
  digitalWrite(I2C, LOW);
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);

  MAX3107_RST_Process();
  MAX3107_I2C_Init(false);

}

void loop() {

  Serial.print("Sending: ");
  Serial.print(0xAA, DEC);
  MAX3107_I2C_Write(MAX3107_THR, 0xAA);
  MAX3107_I2C_Read(0x00);
  Serial.println();
}
```

**i2c_gps.c**

```
#include <Wire.h>
#include <inttypes.h>


#define SDA_PIN 33
#define SCL_PIN 26
#define I2C 35
#define A0 21
#define A1 20
#define RST 37
#define IRQ_PIN 19

/* Register map definitions */
#define MAX3107_RHR          (0x00)
#define MAX3107_THR          (0x00)
#define MAX3107_IRQEN        (0x01)
#define MAX3107_ISR          (0x02)
#define MAX3107_LSRINTEN     (0x03)
#define MAX3107_LSR          (0x04)
#define MAX3107_SPCLCHRINTEN (0x05)
#define MAX3107_SPCLCHARINT  (0x06)
#define MAX3107_STSINTEN     (0x07)
#define MAX3107_STSINT       (0x08)
#define MAX3107_MODE1        (0x09)
#define MAX3107_MODE2        (0x0a)
#define MAX3107_LCR          (0x0b)
#define MAX3107_RXTIMEOUT    (0x0c)
#define MAX3107_HDPLXDELAY   (0x0d)
#define MAX3107_IRDA         (0x0e)
#define MAX3107_FLOWLVL      (0x0f)
#define MAX3107_FIFOTRGLVL   (0x10)
#define MAX3107_TXFIFOLVL    (0x11)
#define MAX3107_RXFIFOLVL    (0x12)
#define MAX3107_FLOWCTRL     (0x13)
#define MAX3107_XON1         (0x14)
#define MAX3107_XON2         (0x15)
#define MAX3107_XOFF1        (0x16)
#define MAX3107_XOFF2        (0x17)
#define MAX3107_GPIOCONFG    (0x18)
#define MAX3107_GPIODATA     (0x19)
#define MAX3107_PLLCONFIG    (0x1a)
#define MAX3107_BRGCONFIG    (0x1b)
#define MAX3107_DIVLSB       (0x1c)
#define MAX3107_DIVMSB       (0x1d)
#define MAX3107_CLKSOURCE    (0x1e)
#define MAX3107_REVID        (0x1f)


/*********************************************************
** Startup with SW and HW reset
**
** Arguments: -
**
** return value:  -
*********************************************************/
```

```
void MAX3107_RST_Process()
{
  Serial.println("Reset process running");
  /*
    Wait for the MAX 3107 to come out of reset
    Wait for IRQ PIN to come up, as this
    indicates that the MAX3107 reset is complete
  */

  while (IRQ_PIN == 0);

  /* Perform a hardware reset */

  digitalWrite(RST, LOW);
  delay(1);
  digitalWrite(RST, HIGH);
  while (IRQ_PIN == 0);

  /* Perform a software reset */

  MAX3107_I2C_Write(MAX3107_CLKSOURCE, 0b00011000);
  MAX3107_I2C_Write(MAX3107_MODE2, 0x01);
  MAX3107_I2C_Write(MAX3107_MODE2, 0x00);
  while (IRQ_PIN == 0);

}

/*************************************************************
** Configuring Modes and Clocking
**
** Arguments:
** loopback: TRUE, if loopback should be enabled
**
** return value:  -
*************************************************************/

void MAX3107_I2C_Init(bool loopback)
{
  Serial.println("Initialization process running");
  /* Configure clocking */

  MAX3107_I2C_Write(MAX3107_CLKSOURCE, 0b00011000);
  MAX3107_I2C_Write(MAX3107_DIVLSB, 24);
  MAX3107_I2C_Write(0x0B, 3);

  /*Configure modes*/

  if (loopback)
  {
    MAX3107_I2C_Write(MAX3107_MODE2, 0b10101000); // Loop back
  }
  else
  {
    MAX3107_I2C_Write(MAX3107_MODE2, 0b10001000);
  }

  /*Enable Interrupts*/

  MAX3107_I2C_Write(MAX3107_IRQEN, 0b11111111);
```

```
  // Flow control - not needed
  // GPIOs - not needed
}

/*************************************************************
** Write one byte to the specified register in the MAX3107
**
** Arguments:
** port: MAX3107 register address to write to
** val: the value to write to that register
**
** return value:  TRUE – register successfully written
**                FALSE – I2C protocol error of some kind
**
*************************************************************/

bool MAX3107_I2C_Write(byte port, byte val)
{
  byte error;

  while (!Serial);

  Wire.beginTransmission(byte(0x2C)); // Datasheet + I2C scan
  Wire.write(port);
  Wire.write(val);
  error = Wire.endTransmission();

  if (error != 0)
  {
    return false;
  }
  return true;
}

/*************************************************************
** Read one byte from the specified register in the MAX3107
**
** Arguments:
** port: MAX3107 register address
**
** return value:  Register containments
**
*************************************************************/

void MAX3107_I2C_Read(byte port)
{
  uint8_t req;
  char reading;
  byte i2c_error;
  Wire.begin(SDA_PIN, SCL_PIN); // Wire comm. begin
  while (!Serial);

  Wire.beginTransmission(byte(0x2C)); // Datasheet + I2C scan
  Wire.write(port);
  i2c_error = Wire.endTransmission();

  if (i2c_error != 0)
  {
```

```
    Serial.println("I2C communication failed at #1");
  }

  static bool checksum = false;
  static uint8_t checksumcnt = 0;
  Wire.begin(SDA_PIN, SCL_PIN);
  Wire.requestFrom(44, 8);
  while (1 <= Wire.available())
  {
    reading = Wire.read();
    if (reading == '$')
    {
      Serial.println();
      checksum = false;
      checksumcnt = 0;
    }
    if(reading == '*')
    {
      checksum = true;
    }
    if(checksum)
    {
      checksumcnt = checksumcnt + 1;
    }
    if(!((reading == 10 | reading == 13) | (checksum == 3)))
    {
      Serial.print(reading);
    }
  }
  i2c_error = Wire.endTransmission();
}


/**************************************************************************
**************************************************************/
void setup() {
  /*
     PIN Definitions
  */
  Serial.begin(115200);
  Serial.println("I2C communication: Start");
  Wire.begin(SDA_PIN, SCL_PIN); // Wire comm. begin

  pinMode(I2C, OUTPUT);
  pinMode(A0, OUTPUT);
  pinMode(A1, OUTPUT);
  pinMode(RST, OUTPUT);
  digitalWrite(I2C, LOW);
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);

  MAX3107_RST_Process();
  MAX3107_I2C_Init(false);

  Wire.begin(SDA_PIN, SCL_PIN); // Wire comm. begin
  while (!Serial);
  Wire.beginTransmission(byte(0x2C)); // Datasheet + I2C scan
  Wire.write(0x00);
  byte error;
```

```
  error = Wire.endTransmission();
  if (error != 0)
  {
    Serial.println("I2C communication failed at #1");
  }
}

void loop() {

  MAX3107_I2C_Read(0x00); // Ezt úgy kéne, hogy csak akkor, ha van FIFO-ban
adat
}
```