

# Témalabor-I40-2022

---

## Bevezető

---

**Feladatot elvégezték:** Nemes Attila, Nemes Axel Roland

**Konzulens:** Kovács László

## Wiring Diagram

---

Opto-kapu bekötése:



Hőmérő bekötése:



Árammérő bekötése:



MPU-9250 bekötése:



Mikrofon bekötése:



## Szenzorok

---

### MLX90614

Használt könyvtár: [Adafruit MLX90614](#)

[Dokumentáció](#) hozzá.

### Előnyök

- Kontaktmentes hőmérséklet mérés
- Celsius és Fahrenheitot is képes mérni
- Kifejezetten pontos

### Hátrányok

- Nem tapasztaltam

## Tapasztalat

- Egyszerű volt használni a könyvtárral.

## Fordulatszám mérő (Opto kapu)

### Előnyök

- Nem kellett külön könyvtárat használni hozzá
- Interrupt függvénnyel megbízható, helyes eredményt ad

### Hátrányok


- Nem tapasztaltam

## Megjegyzés

Érdekes interrupt megoldással figyelni azt, hogy jelez-e a szenzor így nem/nehezen lehet lemaradni egy-egy jelzésről.

## Árammérő

Házilag készített mérő berendezés. Mind a három fázis mérésére alkalmas.

 Árammérő kapcsolási rajza

A képen látható **mérhető feszültség**ből 3 darab van, minden fázisnak 1-1 db. Ezeket anaolg lehet olvasni, hogy mekkora feszültség van rajta és abból könnyen vissza lehet számolni. Az 1.65V-os kimenet nem mindig pontosan ennyi (nekem kb. 1.49V-1.55V között mozgott). Ezt is érdemes mérni és ezzel korrigálni a mérést.

A fentebb látható diagrammoknál lehet részletesebben látni, hogy melyik színű kábel pontosan mit is jelent.

## MPU-9250

Elérhető a dokumentáció a szenzorhoz ezen az [oldalon](#) keresztül.

A regiszter adatlap pedig ezen az [oldalon](#) érhető el.

### Előnyök

- Tartalmaz gyorsulásmérőt
- A gyorsulásmérő 4000Hz-es mintavételezésre is képes
- i2c és SPI kommunikáció is lehetséges
- Sok könyvtár található hozzá

### Hátrányok

- Nem tapasztaltam

## Könyvtárak

Három különböző könyvtárat is kipróbáltam, és mindegyikről más-más véleményem lett:

- [asukiaaa](#) git hub felhasználó könyvtárával barátkoztam meg a leginkább. Egyszerű használni és viszonylag jó dokumentáció tartozik hozzá. Viszont hátránya, hogy nem paraméterezhető fel tetszés szerint a szenzor.

- [hideakitai](#) git hub felhasználó könyvtárát sem nehéz használni, és lehetséges a szenzor felparaméterezése is. Viszont nem vettem volna észre a paraméterek állítgatásával a változásokat a szenzor adatgyűjtésén. Legfontosabb pedig hogy **lassabban tud adatot gyűjteni mint az asukiaaa könyvtar által létrehozott paraméterezés**
- [bolderflight](#) git hub felhasználó könyvtára nagyon komplex, sok lehetőséget biztosít. Viszont nekem nem sikerült elérnem vele a szenzort, többszöri próbálkozásra sem. Így nem tudok véleményt mondani róla.

## Mikrofon

### Előnyök

- Egyszerű használni
- Nem kell külön könyvtár a használatához

### Hátrányok

- Nem tapasztaltam.

### Tapasztalat

Egyszerű volt dolgozni vele, és eredményes.

## ESP-32

### Arduino IDE board config leírása

Nem biztos hogy le kell tölteni, de az esp32 felismeréséhez szükség van egy USB-driverre. Ez esetben [ezen a lineken](#) le van írva, hogyan lehet letölteni.

Ahhoz hogy lehessen ESP-32-re is programozni szükséges arduino IDE-ben egy beállítás.

Lépések:

- **file** fül lenyitása
- **Preferences** kiválasztása, ekkor megjelenik egy új ablak
- az **Addition boards manager** edit textbe be kell szúrni a következő linket: [https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json). Ha már lenne ott egyéb link, akkor egy vesszővel el kell választani egymástól a két linket.

Ezek után hogy beállítsuk a megfelelő boardra és portra a következőket kell tenni:

- board beállítása: **tools -> Board -> esp32**, lépésekkel megjelenik egy lista a létező esp32 board típusokkal. Ezek közül a megfelelőt ki kell választani.
- port beállítása: **tools -> Port**, ekkor megjelenik egy lista a portokkal, melyből a megfelelőt ki kell választani. **Szükséges csatlakoztatni az eszközt, hogy lássuk a megfelelő portot!**

### EN/BOOT button

- Amikor programot töltesz fel az esp32-re az Arduino IDE **Output** felületén a **connecting...** jelenik meg, akkor a boot gombot egy ideig lenyomva kell tartani. (Az USB port mellett lévő nem EN gomb a BOOT gomb.)

## Librarby hozzá

- Nem szükséges külön könyvtárat letölteni a programozásához.

## Egyéb könyvtárak

---

### MQTT Kliens

Használt könyvtárak:

- [EspMQTTClient](#): Az **Axi** féle kódokban ez volt használva. Könnyű a használata, hatékony és eléggé jó dokumentáció tartozik hozzá.
- [PubSubClient](#): Az **Attila** féle kódokban ez volt használva.

## Szenzorokkal egyszerű kommunikáció (I2C)

Használt könyvtár: [Wire](#)

Néhány esetben ez könnyebé tette egy-egy szenzor elérését I2C-n keresztül.

## Mosquitto

---

### Telepítése

Az alábbi [oldalról](#) kell letölteni. Telepítésnél elég végig a nextre kattintgatni, majd végül az install-ra.

### Elindítása

Készíteni kell egy **.conf** fájlt amit elmentesz a mosquitto telepési helyére. Tartalmazza a következőket:

```
listener 1883

allow_anonymous true
```

Ezután rendszergazdaként futtatva a parancssort el kell navigálni a mosquitto telepítési helyére és a következő utasítást kiadni, hogy elindítsuk az mqtt szerverünket:

```
mosquitto -v -c (config fájl neve amit létrehoztunk, kiterjesztéssel együtt)
```

A felugró ablakon engedélyezzük a kapcsolatot (csak első alkalommal kell ezt megtennünk).

## Csatlakozás MQTT Explorer-el

Ehhez szükségünk van a szerver **ip címére** amit egy új parancssorban kérdezzük le az **ipconfig** parancs kiadásával. Nekünk vagy az **ethernet adapter ethernet** IPv4 címe kell, amennyiben ethernet kapcsolatunk van. Vagy a **wireless LAN adapter wi-fi** IPv4 címe kell, ha wifi-re vagyunk csatlakozva.

A telepített MQTT Explorer-t megnyitva egy új **connection**-t létre hozva adjuk meg a szerver nevét (mindegy mi), majd hostnak adjuk meg a megszerzett IP címet. *(Felhasználó névnek és jelszónak nem kell megadni semmit!)*

Megjegyzés: Telepíteni az MQTT Explorert a következő [oldalról](#) lehet.

## Attila Tapasztalatok

---

### FFT

A projekteben én a microfonnal és a rezgésmérővel foglalkoztam. Melyhez szükséges volt megértetnem az FFT működését. Mivel ezelőtt még nem foglalkoztam vele, több mint egy hetig eltartott a megértése. Megnéztem milyen lépések kellenek, hogy jó eredményeket kapjunk az algoritmussal. Ennek a megértéséhez ajánlom az alábbi [oldalt](#). Ezek után elkezdtem keresgélni hogy milyen implementációk vannak. Két különböző megközelítést találtam:

- Az egyik **rekurzív függvény hívásokkal** dolgozik, melynek csak az a hátránya, hogy a rekurzió miatt lassabb lesz a végrehajtás ideje. Viszont mindenhol ezt mutatják be, tanítják, így sok információ érhető el róla.
- Egy másik megközelítés a **DIT (Decimation-in-time) FFT** algoritmus mely iteratívan oldja meg az algoritmust. Ennek megértéséhez az alábbi [oldalt](#) ajánlom.

Érdekesnek megjegyezném, hogy már fejlesztés alatt van egy gyorsabb algoritmus mint az FFT, ez az úgy nevezett **SFFT (Sparse FFT)**.

### ESP32

#### Ismerkedés

Ezután az esp32-vel ismerkedtem meg. A programozása az Arduino IDE-vel nem volt nehéz. Az első alkalommal viszont be kellett konfigurálni az IDE-t mely egy ideig eltartott (fentebb a Szenzorok alatti ESP32 bekezdésnél le van írva részletesen). Majd mikor kódot szerettem volna feltölteni akkor rá kellett jönnöm, hogy feltöltés alatt a boot gombot lenyomva kell tartani. A továbbiakban viszont minden szépen és jól ment.

#### Rendszer felépítése

A végső rendszert az alábbi megfontolások végett alakítottuk ki:

- Az FFT algoritmushoz kettő az n.-en mennyiségű adat kell. Mivel a maximálisan elérhető mintavételezés frekvencia 4kHz ezért **két 4096 double-t** tartalmazó tömb szükséges (egyik a valós a másik az imagináris értékhez), hogy másodpercenként legyen eredményünk.
- Az esp32 statikus memória területébe **csak 13566 double-t** tartalmazó tömb fér, **és ekkor más kódrészlettelől még nincs is szó!** Így nem tudtuk megoldani azt, hogy kétszálon futtassuk a programot (egy adatgyűjtő szálat szerettem volna, meg egy adat feldolgozót), mivel szálanként kellett volna két darab 4096 double-t tartalmazó tömb.
- Az előző bekezdésben említett probléma miatt a mikrofont és az MPU-9250 szenzor adatait feldolgozó kódot sem tudtuk egy esp32-re rátenni, hanem szükség volt két különbözőre.
- A harmadik esp32-re pedig azért van szükség, mert a forgási sebességet mérő **optokapu érzékeny az időbeli késleltetésre** a számítások végett. Mivel pedig a előző két esp32-n is időérzékeny műveleteket végzünk nem tudtuk egyikre sem rátenni.

Összességében tehát 3 db esp32-re volt szükségünk, amikre az alábbi szenzorokat raktuk:


1. gyorsulásmérő (MPU-9250-en van rajta)
2. mikrofon
3. opto kapu, hőmérő (MLX90614), árammérő

## Mikrofon

Ezek után a mikrofonnal foglalkoztam. Tetszett a szenzorral való munka, mivel egyszerű volt használni és gyorsan elértem az eredményeket (adatot gyűjteni, majd azt feldolgozni). Használatához nem volt szükségem külön könyvtárra.

- adatgyűjtés: csak a megfelelő pin-ről kellett analogReadet használni
- adat feldolgozás: ez alatt értem azt, hogy a begyűjtött idő tartományos jelet kellett frekvencia tartományba át alakítani, hogy a legdominánsabb frekvenciákat megkaphassuk. Ehhez az **arduinoFFT** könyvtárat használtam.

Végül az alábbi futási időket kaptam (az eredmények mikroszekundumban vannak):

 Eredmények

### Megjegyzések:

- A **MajorPeaks** függvényt is az arduinoFFT könyvtár kódjából vettem, csak átalakítottam, hogy ne csak egy frekvenciát adjon vissza, hanem tetszőleges számút.
- A MajorPeaks függvényhez szükséges a **mintavételezési sebesség** megadása, melyet közvetlen a meghívása előtt ki is számítottam a következőképpen: az adatgyűjtés idejét átváltom másodpercbe, majd a gyűjtött adatmennyiségét elosztom ezzel az idővel.

## Mpu-9250

Ezen szenzorral való munka tetszett a legkevésbé. Első problémám már az elérésével kezdődött. Nem tudtam, hogy a **Wire.begin()** függvénybe meg kell adjam az adat és órajel pint, mellyen keresztül az esp32 a szenzorral kommunikál. Majd 3 különböző könyvtárat is kipróbáltam, mely a szenzorhoz készült (ezeket a Szenzorok alatti MPU-9250 bekezdés alatt részletezem). Melyek kisebb-nagyobb sikerrel működtek. Viszont egyik könyvtárral sem sikerült elérnem az elviekben maximálisan elérhető 4000Hz-es mintavételezést.

### Megjegyzések

- A mikrofonnál lévő megjegyzések ide is vonatkoznak.
- **Asukiaaa** könyvtárával az alábbi futási eredményeket értem el (az eredmények mikroszekundumban vannak):

 Eredmények

- **Hideakitai** könyvtárával pedig az alábbi futási eredményeket értem el (az eredmények mikroszekundumban vannak):

 Eredmények

- **Bolderflight** könyvtárával nem tudtam elérni a szenzort, és nem sikerült megoldani ezen problémát.

## Wifi-n keresztüli kommunikáció

Erre a beépített **Wifi** könyvtárat használtam. Problémám az csak a wifi-re való csatlakozással volt, mert kollégiumban nem lehet új eszközzel regisztráció nélkül rácsatlakozni a hálózatra. Miután viszont saját mobilneten keresztül próbálkoztam hiba nélkül ment.

# Mosquitto

A konzulens ajánlására próbáltam ki ezt az MQTT szerveret. Nekem tetszett a mosquitto, habár nincs grafikus megjelenítése, csak parancssori.

Az esp32 kódjában én a **PubSubClient** könyvtárat használtam, melyet egyszerűen tudtam kezelni.

## MQTT Explorer

Habár a konzulens nekünk a **NodeRed** használatát javasolta, én emelett döntöttem. Az Explorert egyszerűen tudtam kezelni és nem láttam szükségét a másiknak, mivel az MQTT szerver a lényeg.

## Axi Tapasztalatok

---

Feladatom az árammérő, hőmérséklet mérő és a fordulatszám mérő szenzorok elkészítése volt. Mivel a hardver komponensek készen voltak így nekem csak a kódolt kellett megírni. Nehézségek voltak, hogy nem volt pontosan ledokumentálva, hogy pontosan milyen topológia szerint vannak összekötve a szenzorok. Viszont miután végig követtem a kábeleket onnantól fogva egyszerű volt a történet.

Minden szenzornál más és más volt a bökkenő, hiszen semmi sem ment egyből tökéletesen természetesen... Még mielőtt neki kezdtem volna az egyes méréseknek először az alap dolgokkal foglalkoztam kicsit. Így az első 1-2 hétben csak ismerkedtem a könyvtárakkal és főleg az MQTT-vel.

**Javaslatom** az, hogy csak egy-egy dologgal ismerkedj. Próbáld ki csak magában az MQTT klienszerver kapcsolatot, próbáld ki több könyvtárat és válaszd ki ami a legszimőatikusabb. Én is így tettem és ha egyszer az elején csinálsz egy jó alapot onnantól fogva a későbbiek során sokkal egyszerűbb dolgod lesz.

## MQTT kapcsolat

Mivel a projekt alapja volt egy megbízható és hatékony kommunikációs protokoll ezért ezt próbáltam meg minél jobban megismerni. A projekt alatt MQTT-t használtunk ami szerintem kifejezetten hatékony ahhoz képest, hogy milyen egyszerű a használata.

## MQTT Kliens

Több könyvtár kipróbálása után én [ezt](#) a könyvtárat választottam. Elfogadható dokumentáció és példa volt hozzá ahhoz, hogy ki tudjak belőle indulni. Ezt használtam a többi szenzornál is amit én csináltam.

Amikor kiismertem a könyvtárat előre csináltam egy sablonos MQTT kliens kódot amit már csak fel kellett használnom a későbbi szenzoroknál és így akkor már nem kellett az MQTT kliens kóddal többet foglalkozni.

## MQTT Szerver

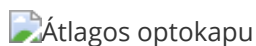
Itt történt meg az első zsácutca a projekt során. Először a **Beywise** szerveret használtam főleg azért, mert sok fórumon top 3 legjobb MQTT szerver között volt illetve, mert ingyenes. Sajnos már az ismerkedés elején kibúj a szög a zsákból. Borzalmas volt az egész. Nem volt hatékony, lassú és folyamatosan lefagyott. Hostol a szerver egy webes felületet ahol lehet kezelni az egész szervert ami kecsegtetően hagzik de borzalmas az egész. Egy óra alatt nem bírtam csinálni egy olyan diagramot amin 3 adatot egyszerre meg tudtam volna jeleníteni (pedig volt rá lehetőség, tálcán kínálta a UI...) csak éppen működni nem akart szegény. Ezért **NEM AJÁNLOM A HASZNÁLATÁT.**

Másik nekifutásra a konzulensünk által javasolt **mosquitto MQTT szerver**t használtuk. Ez nagyon megbízható volt, gyors és robosztus meglátásom szerint. Egy hátránya volt a Beywise-hoz képest, hogy nem járt hozzá webes megjelenítő így külön kliens programot kellett használni, hogy áttekinthetően lehessen látni a topic-okat és az adatokat.

## Fordulatszám mérő

Első nehézség abban volt hogy az előre elkészített fordulatszám mérő berendezés előre be volt kötve és nem volt ledokumentálva, hogy melyik ESP mely vezetékeken kommunikál az opto kapuval. Viszont miután megtaláltam onnantól fogva egyszerűbbé vált minden.

A mérést egy opto kapu segítségével oldottam meg. Ami nagyjából így néz ki.



Működési elve, hogy a rés között fény halad. Egyik irányból kibocsátja, másik irányból érzékeli azt. Feszültség változással tudja jelezni, azt ha képes mérni a fényt azaz nem gátolja semmi a fény terjedését illetve azt is ha valami gátolja a terjedést. Ehhez egy tárcsát használtam amin előre meghatározva, egyenlő távolságban 6 darab lyuk van rajta. Ezeknél a lyukaknál fog érzékelni a kapu. Két érzékelési idő közötti idővel pedig vissza lehet számolni a sebességet azaz az RPM-et amire kíváncsiak vagyunk.

A számolást először folyamatos átlagolással csináltam meg. Az arról szólt, hogy egy kör alatt számolt lyukak közötti időt átlagoltam így kaptam egy átlagos körülfordulási időt, majd ebből számoltam ki az RPM-et. Ezzel az volt a baj, hogy túlságosan finom jelet kaptam így a nagyobb kilengéseket, eltéréseket nehezen (vagy egyáltalán nem) lehetett mérni.

Második nekifutásra ablakos módszerrel folytattam. Egy 6 elemű tömbbe írtam mindig a mért lyuk különbséget és az egész tömböt átlagolom. Persze minden egyes tömbbe írás után a mutató értékét növeltem, ha pedig túl csordult volna akkor a tömb elejére raktam. (Hasonló mint OPre-ben az órás megoldás memóriai kezelésnél). Ezzel a megoldással ugyan finom jelet kaptunk viszont az ingadozásokat is jól lehetett mérni vele.

## Interrupt

Ennél a feladatnál nagyon fontos, hogy interrupt függvény segítségével oldjuk meg. Én is így tettem. Ennek a lényege, hogy amikor a lábon felfelé/lefelé futó él jelenik meg vagy él változás ([beállítás függő](#)) akkor a megadott függvény le fog futni. Így nem fogunk lemaradni egy mérésről sem.

## Lecsengés

Mivel a megoldás alapja egy interrupt függvény használata ezért el lehet gondolkodni mi van akkor ha leáll az eszterga? Mivel nem fog forogni a kerék így nem fog interrupt generálódni és így nem fog frissülni a tömbb ezért úgymond be fog fagyni/akadni a tömb tartalma. Így nem fog lenullázódni és leállni.

Ennek orvoslása érdekében folyamatosan figyelem, hogy bekövetkezik-e ez a "befagyás" jelenség. Ha igen akkor nullázom a tömböt és a jelet kétféleképpen nullázom:

- *FINOM\_EJTES = true*  
Ekkor a jel fokozatosan fog lecsengeni ezzel "szimulálva" a motor leállását hiszen az is szépen lassan fog leállni.
- *FINOM\_EJTES = false*  
Ekkor az RPM értékét egyből lenullázza.



Ez persze kis módosítás vagy extra de legalább ezzel lehet állítani a preferált módot ha erre később szükség lenne.

## Hőmérő

Ez egyszerű mérésnek tűnt és az is volt. Ebben az egyetlen nehézség az az volt, hogy megkeressem mi is az adott szenzor pontos neve. Mivel nem volt ledokumentálva előző csapatok által és még a szenzoron sem volt jelezve a neve ezért kutakodni kellett. Szerencsére Bixby Vision-el sikerült megtalálni a szenzort és egy nagyon egyszerű és kiváló könyvtárat. A könyvtárhoz dokumentáció [itt](#) is elérhető. Ezzel kényelmesen lehet nem csak Celsiusban, hanem Fahrenheitban mérni is.

Végül már csak annyi dolgom volt, hogy MQTT szervernek elküldjem a mért adatokat.

Érdekessége ennek a mérőnek ami nagyon meglepett, hogy érintkezés nélkül eléggé pontosan mér kifejezetten gyorsan.

## Árammérő

Ez volt a legizgalmasabb mérésem a projekt alatt, nagyon élveztem elkészíteni. Maga az elektronika készen volt viszont ahhoz, hogy ezt meg tudhassam csinálni meg kellett értenem. Ezután pedig kezdtem is neki a mérés elkészítésének. Mind a három fázist kellett mérni amit a mért 0-3.3V feszültségből lehetett vissza számolni. Ehhez analóg olvasást kellett végre hajtani de szerencsére az ESP32-n rengeteg analóg láb van.

A programból is látszódik, hogy vissza kellett számolni a feszültséget mivel az analóg olvasás 0-4096 tartományban mér. Ezután használni kellett az Ohm törvényt meg figyelembe kellett venni a tekercs általi szorzót. Miután ezeket elkészítettem vízmelegítővel és hajszárítóval ki is próbáltam.

## Adatok gyűjtése

Egyszerűnek hangzik de sajnos mégsem az. A baj az, hogy a mért adatokat amiket a soros portról olvasunk azokat kéne átmásolni excel-ben, hogy maradandó grafikont készíthessünk. (Persze lehetne soros plotterről is de az nagyon kiszámíthatatlanul képes ugrálni).

Arduino IDE-ben nem lehet soros monitorról rendesen másolni így más megoldást kellett választanom. A [Device Monitoring Studio](#)t választottam. Ingyenes viszont cserébe 10 napos próba verziót kapunk. Erre a kis, gyors mérésre viszont ez tökéletes. (Az arduino értékeit legegyszerűbben a **Console View/Consol View (legacy)** típusú monitoring móddal lehet kimásolni).

Miután mértem adatokat a következő grafikonok születtek meg:

 Árammérő első mérés

Ez a legelső, kezdeti mérés volt még csak egy fázisra de már itt is szépen kirajzolódik a szinusz görbe.

 Árammérő második mérés

Ez a hajszárító mérése ugyan úgy csak egy fázis lett mérve. Kicsit csúnya a jel de ez a Device Monitoring Studio hibája. Sajnos nem megbízható eszköz...

 Árammérő harmadik mérés

Ez pedig a végleges mérés amit egy vízforralón végeztem el, most már mind a három fázisra. Szépen látható ahogy két fázis (kék, zöld) egymást váltják míg a harmadik, sárga színű fázis nulla közelben marad hiszen az a föld.

Ez a mérés tetszett a legjobban mert kevés baj volt vele és mert itt tanultam a legtöbbet az áramméréssel kapcsolatban. Nem mellesleg nagyon szép grafikonokat lehetett készíteni.