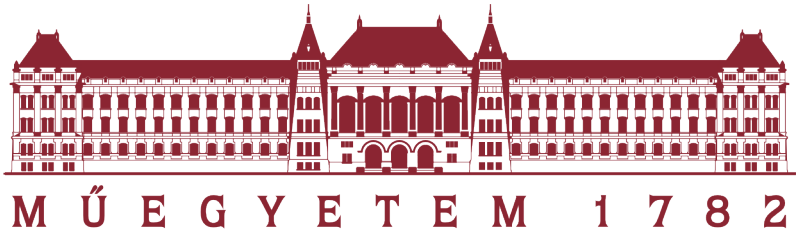


Szenzorhálózat okosotthon alkalmazásra Thread alapokon

Készítette:
Mihalik Márk

Konzulens:
Matolcsy Balázs

TDK dolgozat

2021

Kivonat

Ez a TDK munka betekintést nyújt az új generációs Thread hálózatok felépítésébe, működtetésébe és a Thread mesh hálózatokban rejlő potenciális alkalmazási lehetőségekre. A dolgozatban a nemrégiben megjelent Nordic NRF52840-dongle modulok és saját nyomtatott áramkörök segítségével mutatom be a Thread kommunikációt több modul és egy átjáró között. Az ilyen Thread hálózatok legnagyobb előnye a vetélytársakkal szemben a kiváló skálázhatóság, mesh topológia miatt a kiemelkedően jó területi lefedettség és a megfelelően gyors adatátvitel. A Thread hálózatok egyik kiemelkedő alkalmazási területe az okosotthon(ok). Ezen felhasználási területen sokféle Thread hálózat kompatibilis eszközt kapcsolunk egy nagy egységes mesh hálózatba, és a Thread hálózati végpontokon keletkezett szenzor/beavatkozó modul adatokat egy egységes adatbázisban tárolhatjuk, amely hozzásegít az okosotthonok szabályozási és monitoring rendszerének hatékony kialakításához. Dolgozatomban felépítek egy Thread alapú okosotthon platformot (hardver és szoftver szempontból is), amelyben egy központi egységet (Border Router) és több szenzorral ellátott végpontot (Endpoint) használok fel. Az adatok megjelenítéséhez egy grafikus felhasználói interfészt is készítettem, amely segítségével az egyes szenzoradatok megjeleníthetők, illetve az eszközök fel és lecsatlakoztathatók.

Abstract

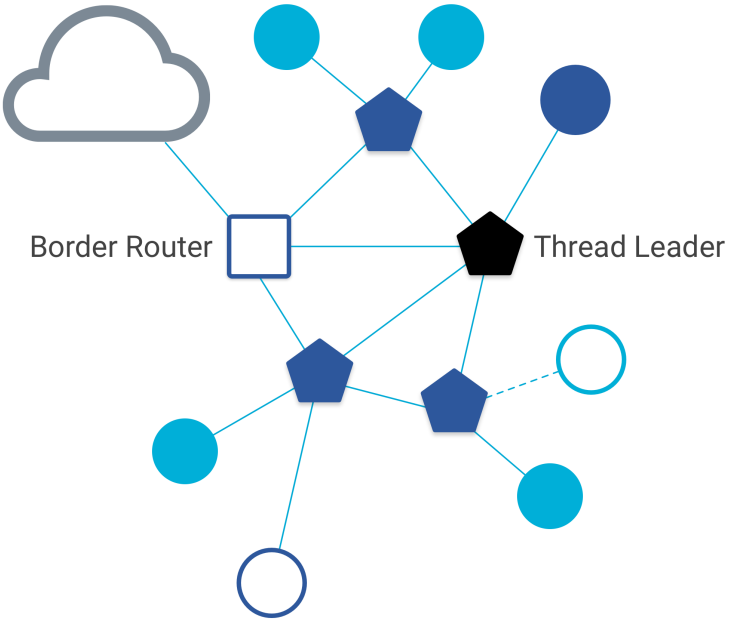
This TDK work provides insight into the design, operation and potential applications of the new generation of Thread networks. In this thesis, I demonstrate Thread communication between multiple modules and a gateway using the recently released Nordic NRF52840-dongle modules and my own printed circuit boards. The main advantage of such Thread networks over their competitors is the excellent scalability, mesh topology due to the outstandingly good area coverage and the sufficiently fast data transfer. One of the prominent applications of Thread networks is the smart home(s). In this application area, a large variety of Thread network compatible devices are connected to a large unified mesh network and the sensor/intervention module data generated at the Thread network endpoints can be stored in a unified database, which contributes to the efficient design of control and monitoring systems for smart homes. In my thesis, I build a Thread-based smart-home platform (both from hardware and software point of view) using a central unit (Border Router) and several sensor endpoints (Endpoint). I have also created a graphical user interface to display the data, and to connect and disconnect devices.

1. Thread

1.1. Bevezetés

Manapság egyre többször találkozhatunk a "Smart Home" kifejezéssel a médiában, vagy az interneten. A statista.com [1] felmérése szerint 2021-ben az okos otthonok részesedése a világpiacon el fogja érni az év végére a 100 millió amerikai dollárt, 2025-re pedig ez a szám ennek a kétszerese lesz. Az okosotthonok célja, hogy a tulajdonos, vagy az otthon élvezője egyszerűen tudja a házában lévő eszközöket egy okostelefonon vagy az interneten keresztül irányítani. Ilyen eszközök lehetnek az okos termosztátok, lámpa kapcsolók és konnektorok. Az előbb felsorolt példák csak kis halmazát képezik ezeknek az eszközöknek a széles palettáján, mivel a piacon számtalan más egyéb szórakoztató elektronikai, biztonsági vagy vezérlő elektronikával (például öntözőrendszer) találkozhatunk. Másfelől nagyon fontos az energiafelhasználás minimalizálása is, mivel a környezetvédelem egyre jelentősebb szerepet játszik a társadalomban, illetve nem mindenhol megoldott az, hogy egy termosztátnak hálózati feszültséget biztosítsunk. Erre a piacra és ezekre a problémákra nyújt megoldást a Thread. A Thread egy IPv6-ra épülő mesh hálózat (ellenben a Zigbee-vel, ami IPv4-re épül), amit alacsony fogyasztású IoT eszközök kommunikációjára fejlesztettek ki. Az IPv4 címzéssel szemben, ami egy 32 bites szám (4,3 milliárd különböző cím foglalható le), az IPv6 128 bites számokat használ a címzésre, így $3{,}4 \cdot 10^{38}$ különböző címet biztosít az eszközöknek és a valamilyen célra dedikált IP címeknek. Ezzel a módszerrel minden eszköz külön címet kaphat, így megkönnyítve az átjátszók (router) szerepét az internet világában. Másik nagy előnye abban rejlik, hogy az IEEE 802.15.4-es szabványra épül, így a hétköznapi Wi-Fi 2.4 GHz frekvencia sávjába esik. Ez a frekvencia-sáv minden országban a szabadfelhasználású ISM (Industrial, Scientific, Medical) frekvencia sávba esik, így nem kell érte licenc díjat fizetni. Továbbá biztonságos, mivel a csatlakoztatás lehetősége korlátozott ideig áll fent egy adott eszköznek (erről a folyamatról egy későbbi részben részletesen lesz szó), és a kommunikáció csakis az egy hálózaton lévő eszközök között megy végbe. Az hálózati topológia megbízhatónak tekinthető, mivel több eszköz több eszközzel is kapcsolatban van (mesh, szövevényes hálózati jelleg), így egy eszköznek a meghibásodása esetén nem romlik el a teljes alhálózat. Ez annak is köszönhető, hogy az eszközök dinamikusan választják a szerepüket a hálózatban. Az alacsony fogyasztású Thread eszközök képesek akár évekig üzemelni egy lítium gombelemről. Végül, de nem utolsó sorban több száz eszközt lehet egy Thread hálózatra csatlakoztatni, ami lehetséges megoldást nyújt az okos irodák eszközeinek az összekapcsolására is.

1.2. A Thread hálózat



1. ábra. Thread hálózati topológia
[2]

A 1. ábrán látható egy tipikus Thread hálózati topológia. Mindegyik kék hexagon egy Router-t, magyarul útválasztót jelöl. Ezekre az eszközökre tudnak csatlakozni az úgynevezett End Device-ok vagyis a végpontok. Fontos kitérni arra, hogy két féle eszköztípus került definiálásra a Thread protokollon belül. Az egyik a *Full Thread Device*, röviden **FTD** a másik pedig a *Minimal Thread Device*, röviden **MTD**. Az előbbit olyan környezetbe ajánlott tervezni, ahol megoldott az eszköz folyamatos energiaellátása (például 230 V-os táphálózat), mert csak is kizárólagosan az ilyen fajta eszközök képezhetik a hálózat gerincét (lásd hexagonok), mivel ezeknek az eszközöknek a feladata a IP cím topológia fenntartása, menedzselése, illetve az adatok küldése a Thread hálózaton belül. A Thread FTD eszközök RF adó-vevője folyamatosan bekapcsolt állapotban van, illetve minden routernek a multicast (olyan előre lefoglalt IP cím, amivel az eszközök egy csoportja érhető el) címére felíratkozik, így azokat a lekéréseket is kezelnie kell. Ez megnövekedett fogyasztást eredményez, számszerűen az áramfelvétel 12-15 mA-re adódik 5 V-os tápfeszültség mellett, amiből már látható, hogy azt a követelményt nem teljesítené az eszközünk, hogy egy gombelemről is évekig működőképes legyen. Erre a problémára adnak magoldásást a Thread MTD eszközök. Mivel ezek az eszközök csakis a

routerrel vagy más néven a Parent (szülő) eszközzel kommunikálhatnak. Ekkor ezeket az eszközöket Child-nak (gyermek) nevezzük és így csakis a Router felől érkező kéréseket kell kiszolgálnia, amivel tovább csökkenthető az energiafogyasztás. A következő felsorolások végén zárójelek között referálok a 1. ábrára az egyértelműbb megfogalmazás érdekében.
Az FTD eszközök három funkciót láthatnak el, ami lehet [3]

- Router (sötétkék kitöltött hexagon) — hálózati feladatokat ellátó eszköz,
- Router Eligible End Device - REED — képes routerként magát kinevezni (sötétkék kitöltött kör),
- Full End Device - FED — nem képes magát routerként kinevezni (sötétkék kitöltetlen kör).

Továbbá három speciális állapota lehet egy FTD eszköznek, amik pedig a

- Thread Leader (fekete kitöltött hexagon) — routerek menedzselését végző eszköz,
- Border Router (sötétkék négyzet) — Thread és más hálózat közötti átjáró/átjátszó,
- Commissioner (nincs jelezve a képen) — csatlakoztatást végző eszköz.

Az MTD eszközökhöz kettő speciális állapot tartozhat a

- Minimal End Device - MED — folyamatos kapcsolatban van a szülővel,
- Sleepy End Device - SED — általában alvó állapotban van és ébredéskor le kell kérdeznie a szülőt.

Ebben a részben megismerkedhettünk egy Thread hálózatban előforduló eszköz típusokról, viszont egy-egy fontosabb szerepet fontos kiemelni.

1.3. Thread hálózati szerepkörök részletes leírása

1.3.1. A Routerek szerepe

Ahogyan már az előző részben írtam, egy Router esetén az adó-vevőegység folyamatosan be van kapcsolva, mivel ezek az eszközök csak FTD típusúak lehetnek. A főbb feladatai közé tartozik, hogy továbbítsa a csomagokat az eszközök között és kapcsolatot tartson a végpontokkal (End Device). Kiemelt szerepeihez tartozik, hogy csak ezek az egységek képesek felcsatlakoztatni új eszközt a Thread hálózatra. Erre a folyamatra később a Commissioner esetében ki fogok térni.

1.3.2. Az End Device/Child szerepe

Itt eltérő működést láthatunk az MTD és FTD eszközök között, de egy pontban közös a működésük. Ezek az eszközök kizárólag egy Routerhez vannak csatlakoztatva és csakis ezekkel a Routerekkel kommunikálhatnak. Ebből következik, hogy más egységekre nem tudnak közvetlenül adatokat küldeni. MTD eszközök esetén az adó-vevő egység az energiafelhasználás csökkentésének érdekében opcionálisan kikapcsolhatóak.

1.3.3. A Thread Leader szerepe

Ezek az eszközök egyedülállóak egy Thread hálózatban és speciális feladatuk az, hogy a Router eszközöket irányítsák, és kezeljék az egész hálózatra vonatkozó konfigurációs információkat. Ezek az eszközök (Router-ek) saját magukat dinamikusan nevezik ki erre a feladatra, hogy ezzel is csökkentsék a hálózat meghibásodásának esélyét. Az eszköz-szerepek újra osztásának lehetősége általában 130 másodpercenként történik.

1.3.4. A Border Router szerepe

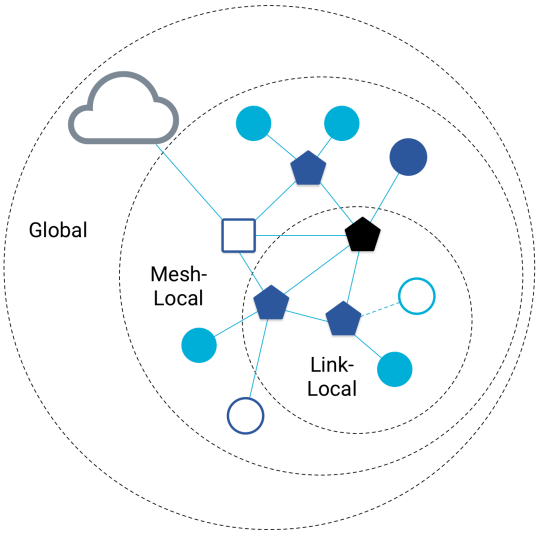
Ezek az eszközök képesek kapcsolatot teremteni egy Thread hálózat és egy külső hálózat között, vagyis egy átjáró szerepet valósítja meg. Külső hálózat lehet egy Ethernet hálózat, mint például, amit én valósítottam meg TDK dolgozatom keretein belül. Minden eszköz működhet Border Routerként, erre nincsen megkötés, viszont érdemes egy FTD eszközt választani erre a célra, mert így folyamatos kommunikáció lehetséges a külvilággal.

1.3.5. A Commissioner szerepe

Ahogyan már fentebb említettem, ez egy plusz feladat, amit csak a Router-ek vállalhatnak el. Ez a feladat nem más, mint új eszközök felcsatlakoztatása a hálózatra. Ez a felcsatlakoztatás egy egyedi chipben tárolt azonosítóval történik, amit eui64-nek neveznek. Amennyiben ezt az egyedi azonosítót megismeri a Commissioner, akkor a csatlakozni kívánt eszköz a hálózati jelszó segítségével fel tud csatlakozni a hálózatra. A csatlakozás problémájára a Thread Group, az OpenThread-et fejlesztő Google is kínál különböző megoldásokat, amik kész megoldásoknak tekinthetőek a fejlesztők és a felhasználók számára is.

1.4. IPv6 címzések egy Thread hálózaton belül

A 2. ábrán látható, hogy három fő részre lehet egy ilyen hálózatot osztani.[5]



2. ábra. Thread hálózat felosztása
[4]

– Link-Local - prefixe: fe80::/16 — Ebben a hálózatban azok az eszközök helyezkednek el, amik szomszédosak egymással, vagyis egy rádióadással elérhetőek. Általánosságban szomszédos eszközök feltérképezésére használják.

– Mesh-Local - prefixe: fd00::/8 — Ebben a hálózatban minden olyan eszköz szerepel, ami csatlakoztatva van az adott Thread hálózathoz.

– Global - prefixe: 2000::/3 — Ebben a hálózatban résztvevő eszközök elérhetőek a Thread hálózaton kívülről is.

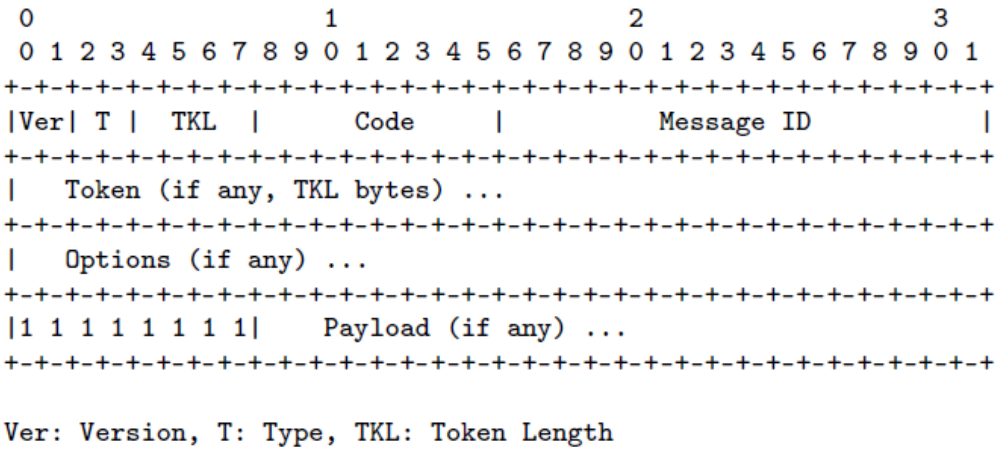
1.4.1. Mesh-Local cím szerepe

Ezeknek a címeknek már sokkal komplexebb feladata van és ezekkel a címekkel küldhetünk adatokat a hálózaton belül. Két csoportra oszthatóak fel, ami lehet

– Routing Locator,

– Mesh-Local EID.

A *Routing Locator*, más néven az **RLOC** címből lehet következtetni egy eszköz hálózatban elfoglalt helyére, illetve annak az eszköznek a szerepére is. A *Mesh-Local EID* (Endpoint Identifiers) címmel bármelyik eszköz elérhető ugyanazon a Thread hálózaton belül. A Mesh-Local címnek van a legnagyobb



3. ábra. Egy CoAP üzenet formátuma
[7]

szerepe a dolgozatban, mivel ezzel a címmel képesek vagyunk például a Border Router eszközről elérni minden hálózatra csatlakoztatott eszközt, így információkat küldhetünk, illetve fogadhatunk. Erre a későbbiekben láthatunk példát, hogy hogyan lehet egy weboldalról saját eszközeinket menedzselni, illetve az eszközökre adatot küldeni.

1.5. CoAP protokoll

A *Constrained Application Protocol* röviden **CoAP** egy kifejezetten IoT eszközökre kifejlesztett információátviteli szerver-kliens kapcsolati protokoll. Nagyon hasonló az internet világából ismert HTTP protokollra, viszont mivel az IoT eszközök sok esetben kis erőforással rendelkeznek, így ennek az implementálása sok erőforrást felhasználna és ezzel megnövelné az eszközök előállítási költségét. Erre a problémára jelent megoldást a CoAP, ami akár 10 kB RAM és 100 kB kódmemóriát foglal le az eszköz erőforrásából. [6] A 3. ábrán láthatjuk, hogy hogyan épül fel egy CoAP üzenet. Első két bájt a verzió számot jelöli, majd a következő két bájt az üzenet típusát foglalja magába, ami négy féle értéket vehet fel.

- Kérelem típusú üzenetek (Request)
  - 0 - CONfirmable — Ez az üzenettípus egy megerősítő (ACK) választ vár.
  - 1 - NON-confirmable — Semmilyen visszajelzést nem vár ez az üzenet típus.

– Válasz típusú üzenetek (Response)

  - 2 - ACKnowledgement — Megerősítő válaszüzenet (lásd. CONfirmable).
  - 3 - ReSeT — Ez az üzenet jelzi, hogy az üzenet beérkezett, de sikertelen a feldolgozás eredménye.

Az előbb megjelölt ábrán a TKL négy biten jelöli az opcionális szimbólum (Token) rész hosszát. Ezután a következő egy bájtos kódrész (Code) két részre osztható, egy három bites típusra (Class) és egy négy bites kódra (Code). Tegyük fel, hogy a kliens a szerver felé egy GET lekérést indítványozik. A lekérő metódusok a 0-ás osztályba tartoznak és speciálisan a GET metódust az 1 érték jelöli. Ha ezt papíron jelölni szeretnénk, akkor az osztályt és a kódot egy ponttal választjuk el, így a GET lekérést a 0.01 jelöli. Az üzenet azonosító kódja (Message ID - MID) egy 16 bites szám, amivel az esetleges üzenet duplikációkat lehet kiküszöbölni. A Token rész egy automatikus, kliens által generált a TKL-ben definiált hosszúságú szám, amit a szervernek módosítás nélkül kell visszaküldenie. A Token célja, hogy az lekérésre érkező választ a kliens pároztatni tudja. A beállítások rész az üzenettel kapcsolatos beállításokat tartalmazza, mint például az általam is használt cím (Uri-Path), amire a lekéréseket a tervezett eszközök végzik vagy az üzenet tartalmának a típusa, ami lehet akár egy UTF-8 kódolt szöveg vagy akár egy JSON formátum is. Abban az esetben, ha takarékoskodni szeretnénk a memóriával, akkor érdemes CBOR-t használni, ami a JSON formátumhoz hasonló felépítésű, viszont a karakterekkel kódolt adatot, tömörebb, kisebb méretű, viszont emberek számára nem olvasható bináris formátumba kódolja át. Ezek után egy, egy bájtos és 0xFF értékű rész választja el a tényleges adatoktól az eddig megismert részeket. A CoAP protokoll részletes specifikációja elérhető a datatracker.ietf.org oldalon.[8] Ebben a részben megismerkedhettünk a Thread hálózatban előforduló eszközökkel, a Thread hálózat topológiájával és az OpenThread által használt CoAP protokollal.

2. Zephyr Project

2.1. Bevezetés

Mivel a Thread kis fogyasztású, energiatakarékos eszközökre kifejlesztett protokoll, aminek célterületei az okosotthonok, ezért fontos, hogy olyan eszközöket tervezzünk, amik nagy számítási kapacitás mellett képesek az előbbi kritériumot tartani. Modern ARM Cortex-M magokkal ellátott mikrokontrollereket számos gyártó készít különböző perifériákkal, amik olykor kifejezetten

energiatakarékosak is. Vannak olyan megoldások is, amik egy chip-en belül integrálják a rádiós modult. Ezek a gyártók, mint például a Nordic, Silabs vagy STM különböző ingyenesen használható SDK-kat készít, hogy megkönnyítsék az általuk forgalmazott kontrollerek programozását. Az IEEE 802.15.4 szabványt használó protokollhoz a Nordic Semiconductors külön SDK-t biztosít, ezzel támogatva a Thread elérését is. Sok esetben viszont egy új SDK-nak a megtanulása holt időt jelent egy cég szempontjából, arról nem is beszélve, hogyha univerzális eszközöket tervezünk, amikben csak az eszközök által kínált lehetőségek egy részét használjuk, akkor minden eszközre módosítanunk kell a kódunkat. Ezekre a problémákra kínál megoldást a Zephyr.

2.2. A Zephyr Project

A Zephyr Projectre sokszor egy valósidejű operációs rendszerként hivatkoznak, aminek a célterülete a mikrokontrollerek és egyéb beágyazott rendszerekben használt architektúrájú processzorok, viszont ennek a felépítése sokkal bonyolultabb, mivel a valóságban az operációs rendszer csak egy kis rétege az egészet tekintve. Alapvetően nyílt forráskódú projekt, ami a Linux Foundation-höz köthető és amit úgy terveztek, hogy biztonságkritikus rendszerekbe lehessen őket implementálni. A Zephyr OS-t három fő részre lehet bontani

- Kernel — memória kezelés, megszakítás kezelés, időzítések kezelése stb.,
- Operating System Services — perifériák kezelése, hardveres modulok kezelése, hibakezelés stb.,
- Application Services — illesztett eszközök kezelése (pl.: Szenzorok), internetes protokollok kezelése (pl.: CoAP).

2.3. Zephyr RTOS tulajdonságai

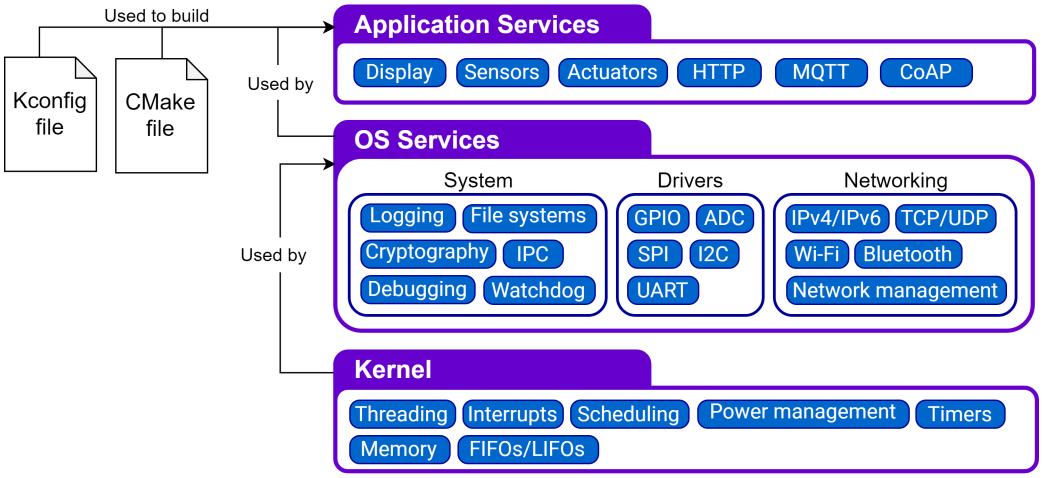
A *Zephyr Real-Time-Operating-System*, azaz **RTOS** a következő funkciókkal rendelkezik

- multithreading (többszálas végrehajtás),
- megszakítás kezelés,
- memória kezelés,
- szálak/threadek (Zephyr OS Task) közötti üzenetek küldése,
- energiakezelés,

– memória védelem (pl.: túlcsordulás ellen).

2.4. Zephyr Project célja

Egyik legfőbb tulajdonsága az RTOS mellett, hogy nagyon könnyen lehet több fajta eszközre lefordítani a megírt forráskódot úgy, hogy a forráskódot nem, csak a beállításokat módosítjuk. Másik lényeges tulajdonsága, hogy harmadik féltől származó könyvtárak is vannak a Zephyr projektbe implementálva, így ezeket egyszerűen lehet használni (ilyen könyvtár az OpenThread is).



4. ábra. Zephyr Project felépítése
[9]

2.5. Zephyr Project - További programok

2.5.1. DeviceTree

Továbbá fontos megemlíteni, hogy a Zephyr Project Cross-Platform, vagyis Linux, MacOS és Windows operációs rendszereken elérhető. A Zephyr a hardver leírásához és a hardveres hordozhatósághoz a **DeviceTree** programot használja. Lényegében ennek a programnak a konfigurációs fájljában bizonyos funkcióknak nevet lehet adni (például a LED1 vagy GOMB0), vagy le lehet írni, hogy melyik címen helyezkedik el egy adott regiszter (például I2C). Ez azért előnyös, mert ha a programunkat más hardverre szeretnénk port-olni, akkor csak a DeveceTree leíró fájlt (.dts kiterjesztéssel ellátott fájlok) kell módosítani. Például, egy STM32F4-es mikrokontrolleren a GPIO24-re elhelyezek egy gombot GOMB0 néven, majd ugyan ezt megteszem egy

nRF52-n is a P0.31-re, akkor nem kell a kódot úgy módosítanom, hogy közben a különböző HAL könyvtárakat megfeleltetem egymásnak, mert ezt majd a Zephyr megteszi a fejlesztő helyett. Az eszközöket egy Nordic nRF52840 chippel terveztem. A Nordic Semiconductor ehhez a chiphez két hivatalos hardvert készített, az egyik a fejlesztői kártya, amit Development Kit-nek neveznek (NRF52840DK), a másik pedig Dongle, aminek a célja a chip USB-n keresztüli programozása és használata az nRF Connect alkalmazással. Mind a két hardver dts leíró fájlja megtalálható előre megírva a Zephyr-ben, így 1 sor átírásával képes voltam a fejlesztői kártyáról a Dongle-re fordítani a teljes kódomat, amit a tervezés előtt fejlesztettem.

2.5.2. KConfig és CMake

A programkód szoftveres hordozhatóságához a Zephyr a **KConfig** programot használja, amivel fordítási időben fordítja bele a használt modulokat, drivereket a programkódba. Ez a program segít engedélyezni szolgáltatásokat és a kernelmodulokat a Linux kernel fordításakor is. A fejlesztéshez továbbá szükséges a **CMake**, ami a könyvtárak header és source fájljait adja hozzá elérési út alapján a projecthez.

2.5.3. A DeviceTree és KConfig összehasonlítása

Összefoglalva a DeviceTree program végzi a hardveres leírást, a hardver elemek engedélyezését vagy tiltását, megszakítások engedélyezését, illetve az órajel beállítását is, amiből boot időben be fogja állítani a mikrokontrollert vagy a processzort. A KConfig ezzel szemben a fordítási időben fogja a Zephyr-be fordítani az adott drivereket és egyéb könyvtárakat. Erre egy példát szeretnék hozni. Tegyük fel, hogy van két alkalmazásunk ugyan abban a hardveres környezetben, ami egy olyan kontrollerrel rendelkezik, amiben megtalálható egy I2C és egy UART periféria, illetve a hardveren elhelyezésre került egy LED, ami egy adott GPIO lábra van kötve. Az egyik alkalmazásban egy I2C-s BME280-as szenzort szeretnék olvasni, a másikban pedig egy LED-et szeretnénk villogtatni. Mivel ugyan azt a hardvert szeretnénk használni, ezért ugyan azt a DeviceTree leíró struktúrát kell alkalmaznunk, így azzal már nem lesz több dolgunk. Szoftveres szempontból az első alkalmazásban szükségünk van az I2C driverre, ezért a KConfignak jelezni kell ezt. Mivel másra nincs szükségünk, ezért fordítási időben a KConfig ezt nem fogja include-olni, így nem fog a végleges programkódba az UART driver belefordulni. A második programban ugyan ez fog történni, csak a programkód nem fogja tartalmazni az I2C drivert sem.

2.5.4. Példa a DeviceTree fájl felépítésére

```
{
        compatible = "nordic,nrf-timer";
        status = "okay";
        reg = < 0x40009000 0x1000 >;
        cc-num = < 0x4 >;
        interrupts = < 0x9 0x1 >;
        prescaler = < 0x0 >;
        label = "TIMER_1";
};
```

Ebben a DeviceTree fájl részletben egy időzítő definiálása történik. Látható, hogy ez egy Nordic chipbe épített nRF időzítő, ami engedélyezve van a rendszerben (status="okay";). Látható a regiszterblokk címe, ahol ez a timer van és az, hogy mekkora terület van ennek az időzítőnek lefoglalva. A CC regiszterek száma 4 darab. Ezek összehasonlító regiszterek, amiknek a feladata az, hogy ha az időzítő számlálója eléri az ebben a regiszterben tárolt számot, akkor megszakítást kezdeményezzen. A megszakítás külön van definiálva a fájlban, hogy mit jelentsenek ezek az értékek. Az utolsó előtti sorban pedig az látható, hogy ennek az időzítőnek nincs előosztása. Továbbá erre a timerre a zephyrben TIMER_1 ként fogunk tudni hivatkozni. Ha megnézzük ennek a Nordic chipnek a dokumentációját akkor az látható, hogy a timerhez ténylegesen ez a regiszter érték tartozik. Ha egy ilyen fájlt szeretnénk módosítani, vagy egy default értéket felül írni, akkor azt egy *.ovarlay fájlal tehetjük meg a programunk helyén, ahol hivatkozás után beállíthatjuk a kívánt értéket (például az okay-t false-ra állítjuk és ekkor ez az egység letiltásra kerül).

2.6. Nordic nRF52840

Az nRF52840 a Nordic Semiconductors 2017 év végén megjelent mikrokontrollere, amit a 2.4 GHz-es vezeték nélküli technológiák piacára terveztek. Teljes szoftveres SDK-t biztosít a cég Thread, Zigbee, Bluetooth LE (Low Energy) és Bluetooth Mesh hálózatok számára. A programkód futtatásáért egy ARM Cortex M4-es mag felel, amiben található egy lebegőpontos műveletvégző egység. 1 MB flash és 256 kB RAM memóriával rendelkezik, amit a komplex OpenThread kódja is csak 30% RAM és flash memóriát foglal le a Zephyr ebben a TDK dolgozatban használt szoftvereknek. Belső DC/DC konverterrel és egy LDO-val is rendelkezik, így meghajtható külső USB tápfeszültségéről. A külső, illesztett perifériáknak is biztosíthat tápfeszültséget a

DC/DC átalakító és LDO együttes használatával, mivel ezeknek az átalakítóknak a kimenete kivezetésre került. A maximum terhelhetősége az adatlap alapján a feszültség átalakítók kimenetének 25 mA. A kimeneti feszültség 1,8V és 3,3V között állítható két erre kijelölt regiszterrel. Megtalálható benne az ARM CryptoCell technológia, ami a különböző titkosítások elvégzésenek az idejét redukálja. Az IC továbbá számos kommunikációs perifériát tartalmaz, mint például az USB 2.0, QSPI, 32 MHz-es SPI, illetve számos más egyéb timert és ADC modult. Ezeknek a teljes listája az adatlap[10] második oldalán érhető el.

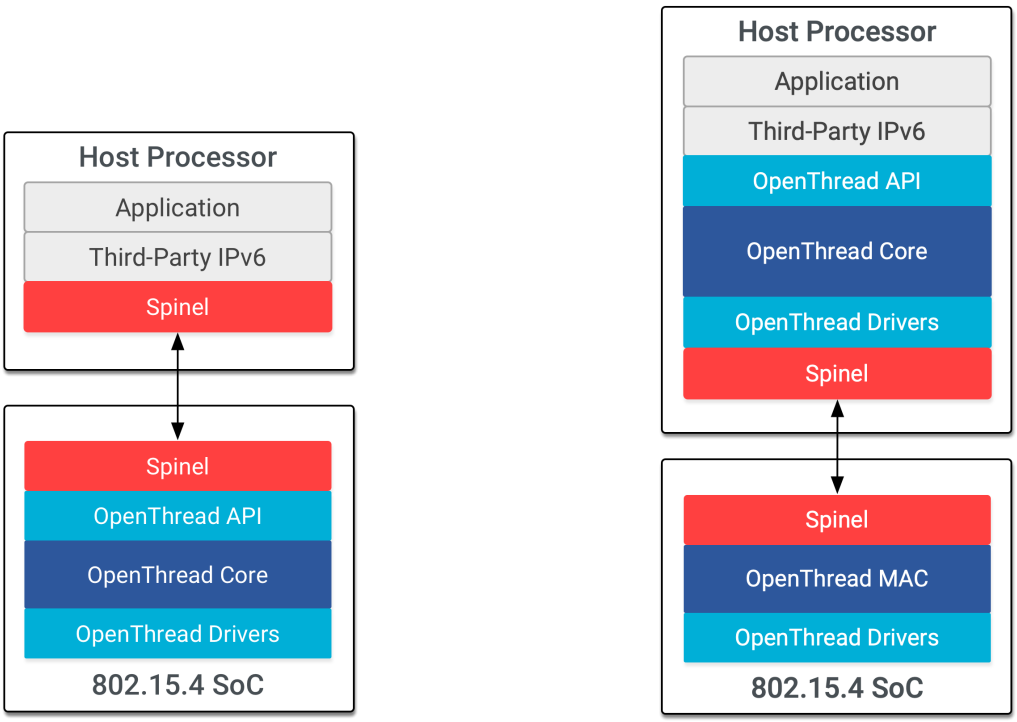
3. Border Router megvalósítása

3.1. Co-Processzorok

Az átjárót, angol nevén a *Border Router*-t egy társprocesszorral (*Co-Processzor*) lehet megvalósítani. Ehhez a Nordic a saját chipjeire, a Zephyr, illetve a OpenThread számtalan platformra nyújt megoldásokat. Itt fontos megismerkedni kettő fajta elképzeléssel a Co-Processzorok világában.

3.1.1. Network Co-Processzor

Az egyik ilyen fajta elképzelés a *Network Co-processzor*, röviden **NCP**, ami a 5. ábra bal oldalán látható. Ezt a design-t akkor érdemes használni, ha tehermentesíteni szeretnénk a fő erőforrásunkat (*Host processzor*) energia megtakarítás céljából, mivel így tétlen (*idle*) állapotban futhat a processzor, vagy akár alvó állapotban tarthatjuk azt. Előfordulhat az az eset is, hogy valamilyen erőforrásigényes tevékenység miatt fontos, hogy ne terheljük feleslegesen a processzorukat, mint például egy kamera képének a feldolgozása. Ebben az esetben a Thread API és a kommunikáció elvi kialakításához szükséges réteg (*Thread Core*) a Thread Rádió fizikai rétegével együtt fog egy jóval alacsonyabb fogyasztású, különálló processzoron futni. A Host processzor és a Co-Processzor UART, SPI vagy USB kommunikáción keresztül tud kommunikálni. Az ábrán ez a kommunikáció a "Spinel" rétegben valósul meg, ami egy olyan protokoll, amit kifejezetten erre a feladatra fejlesztett ki a Google, ami ebben[11] a dokumentációban lelhető fel. A bal oldali képről az a következtetés vonható le, hogy a Host-on csak az IPv6 kezelés és az alkalmazásunk fog futni.



(a) Network Co-Processzor

(b) Radio Co-Processzor

5. ábra. Network Co-Processzor vs. Radio Co-Processzor

3.1.2. Radio Co-Processzor

Ebben az esetben, ezen a *Radio Co-Processzor*-on, röviden **RCP**-n csak egy minimális kommunikációs réteg a *Thread MAC* (*Medium Access Control*) és a fizikai driver fut együtt. Ennek az előnye, hogy minden OpenThreaddel kapcsolatos réteg a jóval erősebb Host processzoron helyezkedik el. Mint ahogyan az előző esetben ez is valamilyen soros kommunikációval kommunikál.

3.2. A megvalósításhoz használt co-processzor design

A fentebb leírtak alapján az első verzióban NCP felhasználásával készítettem el a saját Border Routerremet, viszont az OpenThread folyamatos, gyors változásának köszönhetően egy olyan irány kezd kirajzolódni, hogy a két design közül a fejlesztési szakaszban inkább az RCP design fog érvényesülni. Jelenleg a Google is csak ezt támogatja, így a fejlesztést és a hardver tervezést ebbe az irányba folytattam tovább.

3.3. Raspberry Pi HAT - Border Router

3.3.1. Követelmények

A Border Routert egy Raspberry Pi 4-nek a felhasználásával valósítottam meg, amire egy HAT-et terveztem és programoztam fel. A **HAT** szó a *Hardware Attached on Top* angol kifejezésből származik. A Raspberry Pi 2014 óta fejleszti a HATs GitHub[12] könyvtárat, aminek az alapötlete, hogy a Raspberry Pi Model B tüskesorát felhasználva különböző eszközök jöhessenek létre. Ez a tüskesor 40 pin-ből áll a második széria óta és azóta ennek az alap kiosztása nem változott. Az évek során, ahogyan egyre bonyolultabb Broadcom chip-ek jelennek meg a Raspberry-kben, egy-egy tüskesorlábon egyre több funkció érhető el. Ezeken a HAT-eken, ha elhelyezünk egy I2C-n keresztül kommunikáló EEPROM-ot, akkor a Raspberry Pi képes boot idő alatt a GPIO lábakat ebben a tárolóban lévő adat alapján felkonfigurálni. Alapvető követelmény volt, hogy két (GPIO 0 és GPIO 1), az I2C EEPROM-nak dedikált lábakat nem lehet más feladatra használni. Továbbá két másik alapkövetelményt is felállítottak a fejlesztők. Az egyik, hogy ha a tápfeszültség csatlakoztatása lehetséges a HAT-ről, akkor egy ideálisnak tekinthető dióddával (*ideal diode*) védekezni kell a visszatáplálástól (*back powering*), mivel ha csatlakoztatva van a tápfeszültség a Raspberry Power portján keresztül, akkor különböző feszültségszintek esetén károkat okozhat táp modulban. A harmadik alapkövetelmény, hogy bizonyos lábakat védeni kell a rövidzártól, mivel ezeknek a boot idő alatt régebbi verziókban kimeneti állapotban konfigurálódnak fel, mivel boot idejű jelzésekre voltak használatosak. Csak akkor mondható egy eszköz HAT-nek, ha

- teljesíti a három alapkövetelményt,
- valós, feldolgozható adatot tartalmazó I2C-s EEPROM foglal helyet a HAT-en,
- 40 lábas tüskesorral tud csatlakozni a Raspberry-re,
- a mechanikai specifikációt követi (adott méret és lyuk átmérők),
- legalább 8 mm távolságra van a két NYÁK egymástól,
- képes legalább 1,3 A folytonos árammal táplálni az 5 V-os lábon keresztül.

A tervezés során ezeknek az irányelveknek megfeleltem, ezeket betartottam, így azt mondhatom, hogy az általam készített Thread átjáró nevezhető HAT-nek.

3.3.2. EEPROM és a tárolt adat

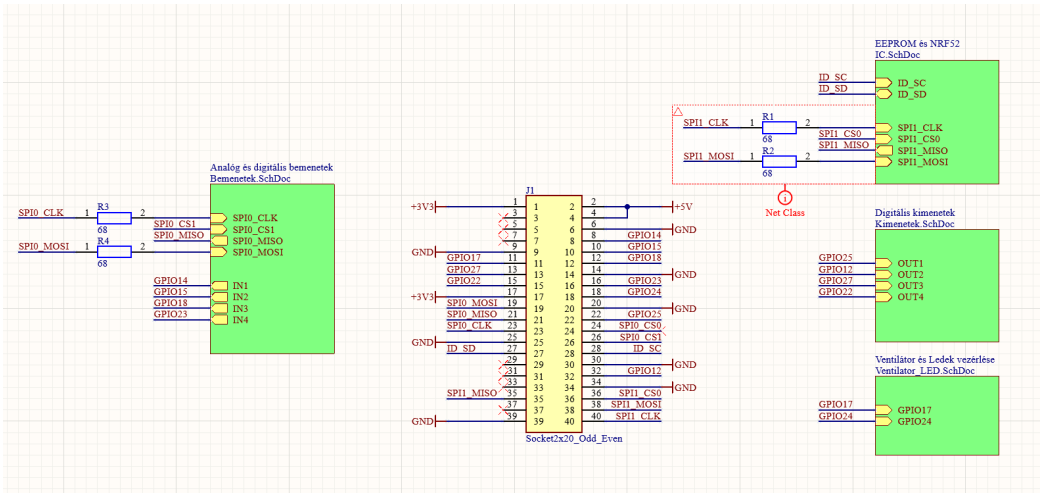
EEPROM-nak egy **CAT24Cxx** OnSemiconductor által gyártott 256 kByte-os tárolót választottam, mivel ez rendelkezik hardveres írás védelemmel (*Write Protection*) az egész memória területre, ellenben a Microchip által gyártott ugyan ilyen szériás IC-kel. Így az általam gyártott panelen íráskor egy jumpert kell eltávolítani, így nem történhet meg az, hogy a felhasználó véletlenül átírja az EEPROM tartalmát. Ebben a könyvtárban megtalálhatóak az EEPROM írásához és EEPROM tartalmának legenerálásához célspecifikus programok. A könnyebb konfiguráció készítéshez egy szövegfájlban kell beállítanunk, hogy melyik GPIO lábra, milyen funkciót szeretnénk beállítani, ami lehet bemenet, kimenet vagy alternatív funkció, amit az adott Raspberry Pi BCM chip-jének a dokumentációjában találhatunk meg. Továbbá lehetőség van a bementekként konfigurált lábak fel- vagy lehúzására. A Raspberry Pi

GPIO	Default Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0	High	SDA0	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
1	High	SCL0	SA4	DE	SPI3_MISO	RXD2	SCL6
2	High	SDA1	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
3	High	SCL1	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3

6. ábra. A Rasberry Pi 4 első négy GPIO funkciója
[13]

4 egy BCM2711-es chippel van ellátva, aminek az első négy GPIO lábának a funkciói a 6. ábrán vannak megjelenítve. Ebből a táblázatból az olvasható ki, hogy ha ezeket ALT0 funkciójú lábaknak konfiguráljuk fel, akkor két I2C perifériát kapunk. Másik esetben, ha egy UART perifériára van szükségünk, amin biztosítanunk kell az adatáramlás szabályozását (*Flow-Control*), akkor pedig ezeknek a lábaknak az ALT4 funkciót kell választanunk. Ha feltöltöttük a memóriát érvényes adattal, akkor a következő boot alkalmával ezt a Raspberry automatikusan kiolvassa és a lábakat ezek szerint fogja beállítani. Azt fontos megjegyezni, hogy a /boot partíción található egy config.txt fájl, amiben engedélyeznünk kell a speciális perifériáinkat, amit a *dtoverlay* program fog megtenni. Például az uart3 perifériánk az engedélyezését flow-control-lal a következő képpen lehet megtenni:

dtoverlay=uart3, ctsrts

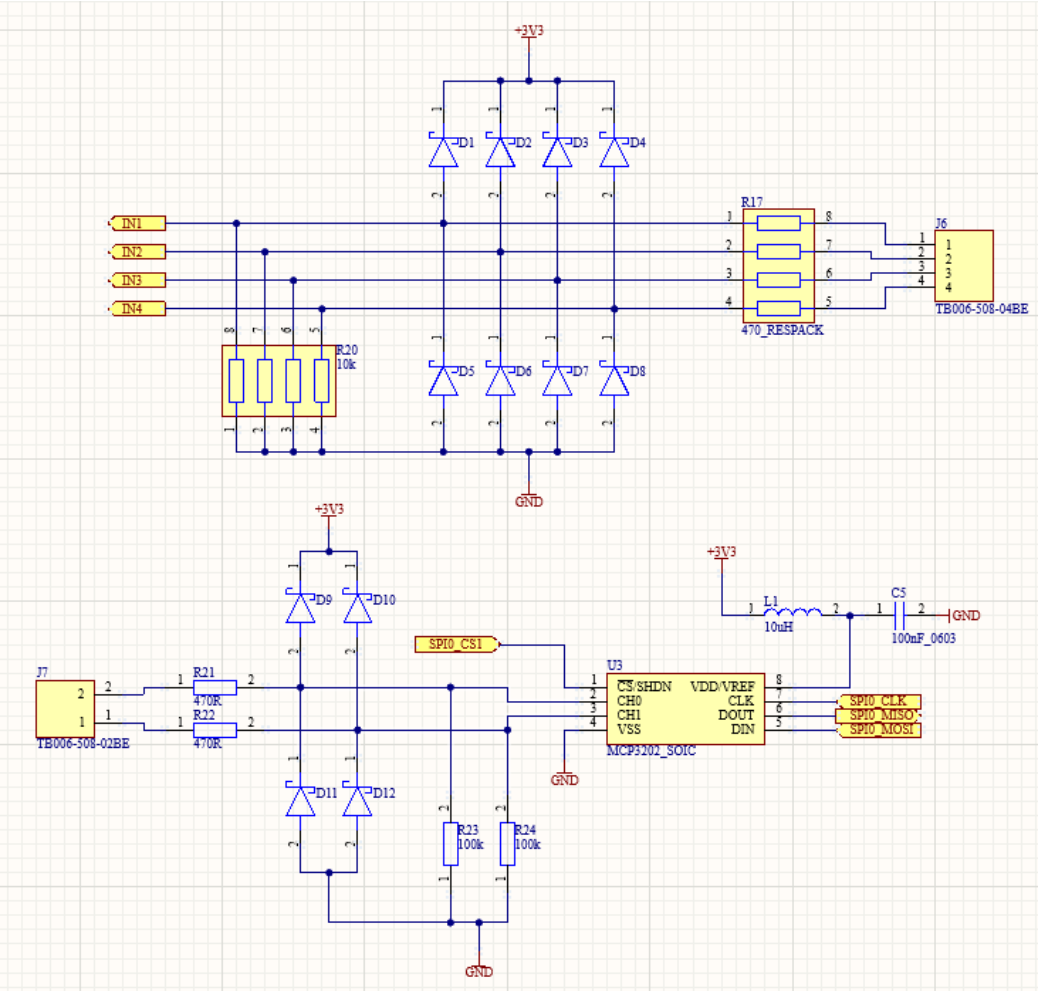


7. ábra. Raspberry Pi HAT kapcsolási rajza

3.4. Raspberry Pi HAT - Border Router tervezés

A célom volt egy olyan eszköz fejlesztése, ami átmenetet képez egy Proof-Of-Concept és egy valós késztermék között. A hardvertervezéshez Altium Designer-t használtam, mivel a világpiacon ez egyik legtöbbet használt tervező program. A 7. ábrán látható, hogy négy különböző részre lehet bontani a kapcsolási rajzot. Bal oldalon a bementek láthatóak, ahol négy darab digitális és két analóg bemenet kapott helyet. Ezek a bemenetek védettek a túlfeszültségtől (maximum 7 V), így TTL jelek esetén is használhatóak a digitális bemenetek, és nem károsodnak rossz bekötés esetén. Az analóg bemenetek 0 és 3,3 V közötti tartományon mérik a rácsatlakoztatott feszültséget, így további felhasználásokhoz nyújt lehetőséget. Mivel a Raspberry Pi által használt Broadcom chip nem rendelkezik analóg-digitális átalakítóval, így erre egy külső Microchip által gyártott SPI-on kommunikáló IC-t használtam, ami nagyságrendekkel olcsóbb lehetőséget kínált ennek a problémának a kiküszöbölésére az I2C-s társához képest. Középen a 40 pin-es hüvely található és azoknak a bekötése. Jobb felső sarokban látható az I2C-s EEPROM és egy nRF52840 SPI-on keresztül csatlakoztatva. A áthallás redukálása végett a SCK és SDO vonalokon egy-egy 68 ohmos ellenállással növeltem meg a jelváltások fel- és lefutási idejét. Jobb középen helyezkedik el a digitális kimenetek blokkja, ami 0 és 3,3 V jelszintekkel rendelkeznek és ezek is védettek a túlfeszültségtől, illetve a rossz bekötés miatt előforduló kimenetek összekötésétől is. A be- és kimenetek, illetve a 5 V és föld tápvonalak 50 mil rasztertávolságú sorkapcsokon keresztül állnak rendelkezésre a NYÁK-on. A jobb alsó sarokban pedig a PCB-n elhelyezett öt darab LED-nek és két darab

ventilátor kimenetnek a vezérlő blokkja látható. Ezek 3,3 V (logikai 1) hatására kapcsolhatók ki és be, illetve a fényerő és a forgási sebesség PWM-mel vezérelhető. A LED-ek csak esztétikai szerepet töltenek be, míg a ventilátorokkal védhető a túlmelegedéstől az eszköz. A zöld blokkok részletes bontása a 8-11. ábra között látható.

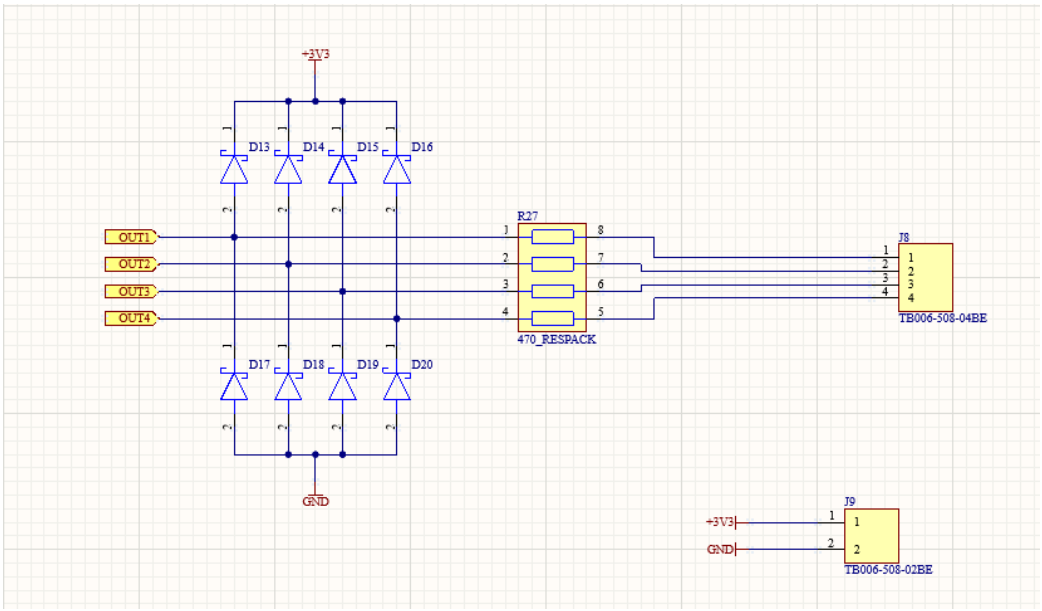


8. ábra. Raspberry Pi HAT bemenetek

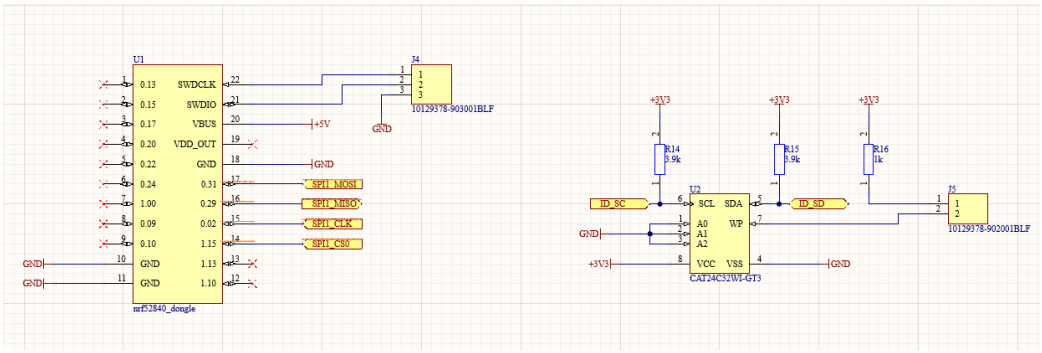
A kész NYÁK terv 3D megtekintőben a 12. ábrán látható.

3.5. Eszközdoboz

Fontosnak tartottam egy olyan ház elkészítését, amiben elhelyezhetem az átjátszót, így védve a mechanikai sérülésektől az eszközt. Ennek a modellnek

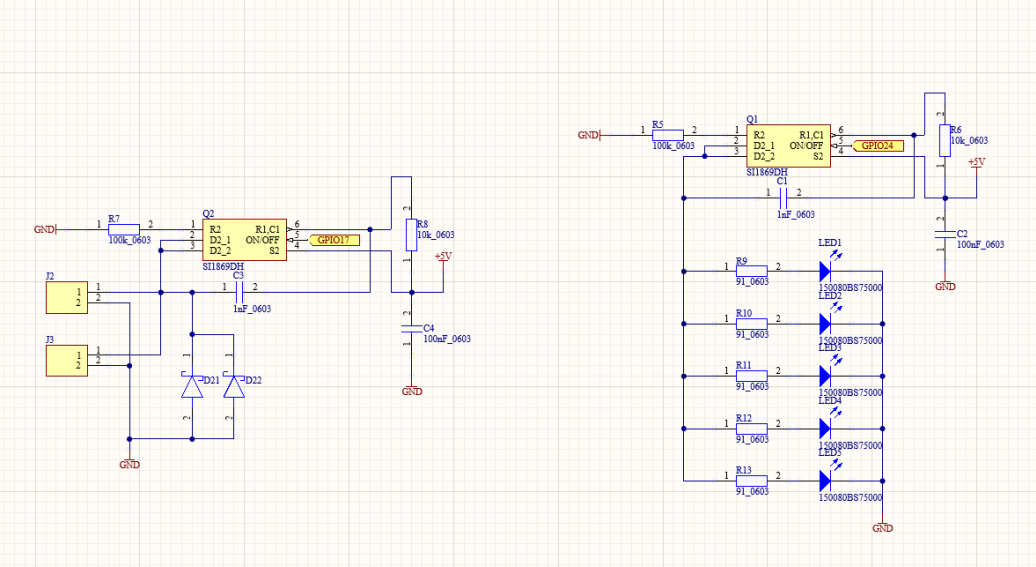


9. ábra. Raspberry Pi HAT kimenetek

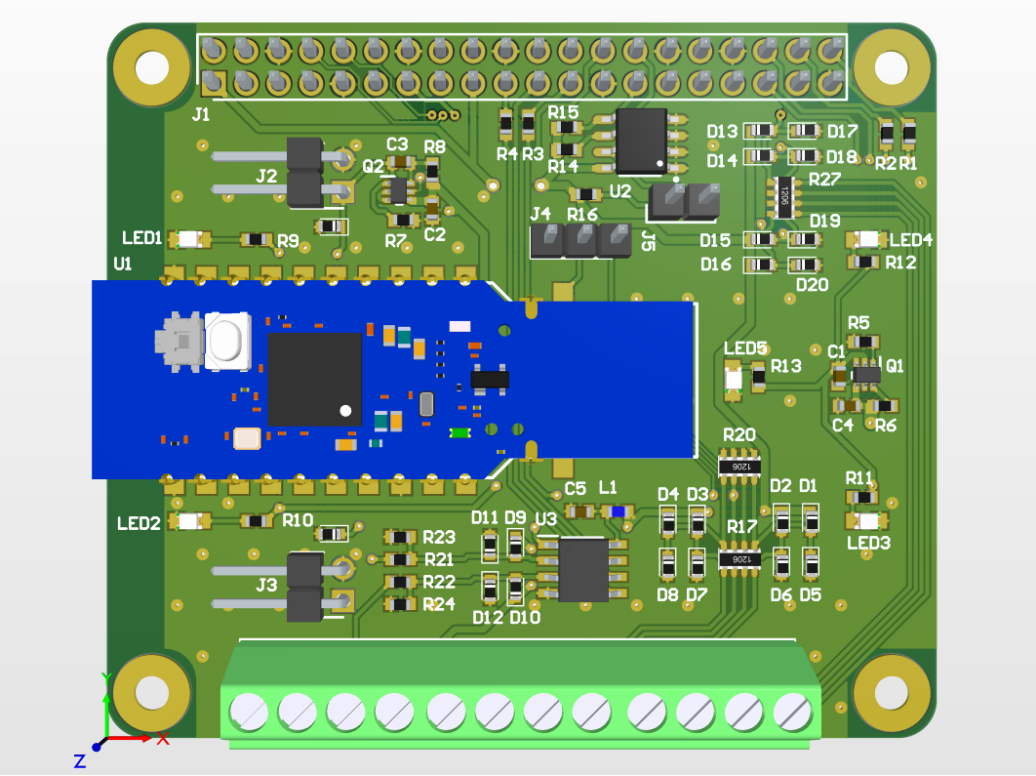


10. ábra. Raspberry Pi HAT integrált áramkörök

a létrehozásához az AutoCAD Inventor alkalmazást használtam. A modellben helyet kapott két darab 5 mm-es kefe nélküli egyenáramú (*Brushless DC* - **BLDC**) ventilátor, ami a Raspberry hűtését végzi, illetve az Ethernet és az USB-Type C portoknak kihagytam a megfelelő helyet, mivel csak USB porton keresztül kerülhet az eszköz megtáplálásra, illetve az Ethernet portról létesít kapcsolatot az eszköz az otthoni hálózattal. Helyet kapott továbbá a sorkapcsoknak az elérésére szolgáltatott rész, így a felhasználó egy lapos csavarhúzó segítségével könnyedén hozzáférhet az 5 V-os tápfeszültséghez, ki- és bemenetekhez.



11. ábra. Raspberry Pi HAT LED és ventilátor vezérlő



12. ábra. Raspberry Pi HAT 3D megjelenítőben

3.6. Szoftver

3.6.1. nRF52840 RCP szoftver

A Radio Co-Processzorhoz az OpenThread által nRF52840-hez fejlesztett csomagot használtam és fordítottam le az általam tervezett SPI lábakhoz. Ezeket az erre kijelölt header fájlban tettem meg, amit transport-config.h néven található meg. A szoftver elérhető az alábbi GitHub könyvtárban[14]. Az SPI kommunikációhoz, összesen hat darab lábra van szükség, amiből négy láb az SPI kommunikációt valósítja meg, egy lábat a mikrokontroller hardveres RESET lábára kell kötni és egy lábat pedig fent kell tartani interrupt jelzésre. Az utóbbi két láb a dokumentáció alapján opcionális és amennyiben nem használunk megszakítás jelzésére használatos lábat, abban az esetben nagyon meg lassulna a kommunikáció, mivel a Host eszköznek, folytonos lekéréseket kellene végeznie. A valóságban viszont más a helyzet, mivel a kód tartalmaz egy olyan részt, hogy amennyiben a két opcionális láb valamelyike hiányzik, abban ez esetben hibakóddal tér vissza a program. Mivel az általam használt Dongle eszköz hardveres RESET lába nincs kivezetve, hanem egy adott gombra van kötve, így az SPI kommunikáció nem valósítható meg, ezért a jóval lassabb UART kommunikációra tértem át, amihez négy lábra van szükség, két adat lábra (Rx, Tx) és két darab jelzésre (CTS, RTS). A kód feltöltése után az eszköz készen állt a kommunikációra. Érdemes megjegyezni, hogy vannak olyan lábak, amik csak kis frekvenciára (10 kHz) használhatóak, mivel chipen belül az RF modul mellett futnak, így a gyártó nem garantálja az áthallás mentességet.

3.6.2. Raspberry Pi - OpenThread hivatalos szoftver

Az OpenThread biztosít egy Raspbian alapú GNU/Linux disztribúciót a kommunikáció megvalósítására, de ez elérhető különálló alkalmazásban is, amit a fejlesztőnek kell telepítenie. A telepítés után három program épül be az operációs rendszerbe, az *otbr-agent* (OpenThread-BorderRouter rövidítésből), ami a kommunikációt végzi az RCP-vel fordítástól függően SPI, UART vagy USB-n keresztül. Szorosan kapcsolódik ehhez az avahi-daemon, ami a hálózati felületeket (*Network Interface*) deríti fel. Az *otbr-agent* egy **WPAN0** (*Wireless Personal Area*, vagyis vezeték nélküli személyi hálózat) interfészként fog megjelenni. Amennyiben használjuk az *ifconfig* programot, láthatjuk is az interfészek között. A harmadik program pedig az *otbr-web*, ami egy beállításhoz szükséges kezdetleges weboldalt biztosít a localhost címen és 80-as porton, ahol Thread hálózatot vagyunk képesek létrehozni, csatlakoztatást kezdeményezhetünk, illetve megnézhetjük egy vizualizációs felületen a Thread hálózatunkat. A csatlakoztatáshoz szükséges egy opcionális negyedik

program, amit a *ot-commissioner*[15]-nek neveznek. Ezt saját magunknak kell lefordítanunk. Mivel a weboldalon keresztül elérhetővé vált a Thread hálózat elérése, így a szó tág értelmében már beszélhetünk Border Routerről.

3.6.3. Border Router egyéb funkciót ellátó szkriptek

Boot idő után, amikor rendelkezésre áll az otbr-agent szoftver, akkor egy Python skript segítségével hoz létre az eszköz egy Thread hálózatot az *ot-cli* program hívásával. Ezzel a programmal menedzselhetjük a Co-Processzoros Thread eszközünket Linux operációs rendszeren, vagyis létrehozhatunk egy saját Thread hálózatot, csatlakozhatunk már meglévő hálózatra vagy akár feltérképezhetjük a hálózatot. A fények és a ventilátor kezelését is egy Python szkript végzi, ami folyamatosan figyeli a processzor hőmérsékletét és ha 60 °C felé emelkedik, akkor a ventilátor 50 %-os kitöltési tényezőjű vezérlő jel hatására kezd el forogni, majd ha ez még tovább emelkedik, akkor lineáris skálán módosul a kitöltési tényező egészen 100 %-ig, ami 70 °C hőmérsékletnél lép életbe.

4. Kezelőfelület - GUI (Graphical User Interface)

4.1. Weboldalhoz használt progromok

A kezelőfelületet egy weboldallal valósítottam meg, amit *Bootstrap CSS*-el, *HTML*-el készítettem el és a kliens oldalon az adatok helyességének ellenőrzését *JavaScript*-el oldottam meg. Szerver oldalon backendnek pedig *FastAPI*-t használtam, ami jelenleg az egyik leggyorsabb Python framework. Köszönhetően a Python nyelvnek

- gyorsan fejleszthető,
- a szintaktika miatt kevesebb az esélye az elírásból származó bug-oknak,
- könnyen tanulható,
- viszonylag rövid kódban fejleszthetünk, elkerülve a felesleges kód duplikációkat.

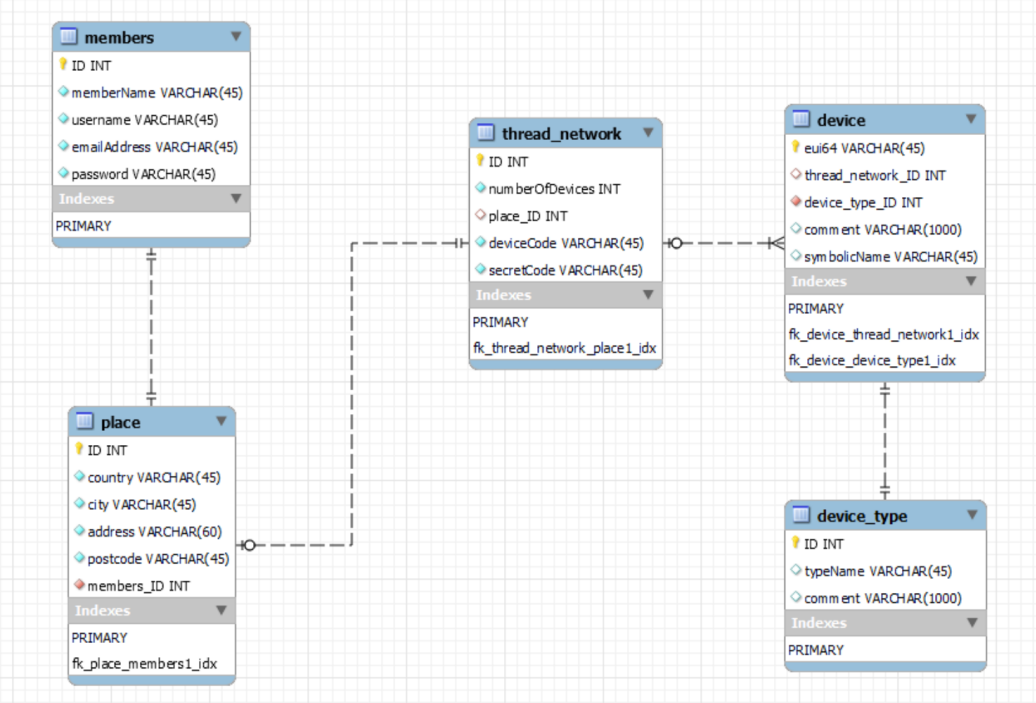
Továbbá a FastAPI-ra jellemző, hogy

- robusztus, automatikus dokumentációt készít és könnyen debuggolható,
- JSON sémára és OpenAPI-ra épül.

A weboldalt egy **ASGI** (*Asynchronous Server Gateway Interface*) webszerveren futtatam, amit *Uvicorn*-nak neveznek. Lényegében ez teremti meg a kapcsolatot a backend és a webszerver között. Vagyis abban az esetben, ha egy HTTP lekérés érkezik a kliens felől, akkor ezt a webszerver jelezni fogja a FastAPI felé. Minden kommunikáció a program és a szerver között egy Python Dictionary formátumú data struktúrában fog zajlani. A Dictionary nagy vonalakban hasonlít a JSON adat formátumra, amit szövegként tárolunk el, ezzel szemben a Dictionary az egy memória objektum. A dinamikusan változó weboldal elemeket *Jinja template*-tel valósítottam meg. Ez a program végzi a Dictionary objektumokból az adatok feldolgozását és a megfelelő helyre való beillesztését. Továbbá tartalmaz feltételes elágazásokat és foreach ciklust, amit én a későbbiekben hasznosítottam is.

4.2. Adatbázis

Mielőtt rátérnék a backend működésére, be szeretném mutatni az adatbázis modellt, amivel dolgoztam.



13. ábra. A használt adatbázis modell

4.2.1. Felhasználók

Az adatokat egy *MariaDB* adatbázisban tároltam el, ami a MySQL Open-Source verziójának tekinthető. A 13. ábrán látható, hogy a bal felső sarokból kezdődően egy külön tábla képezi a felhasználókat. Minden felhasználóhoz társítottam egy ID-t, ami automatikusan inkrementálódik és ez feleltethető meg az elsődleges kulcsnak. Egy felhasználónak van egy polgári neve és egy email címe is, ami egy-egy külön oszlopot képez. A gyorsabb bejelentkezés végett egy felhasználó nevet is társítottam az eddigiek mellé, és természetesen egy jelszót, amit a felhasználó adhat meg.

4.2.2. Elhelyezkedés

A Thread hálózatot úgy terveztem, hogy egy hálózathoz egy lakás tartozhat, mivel egy hálózatban több mint 500 eszköz tud felcsatlakozni. Így ebben a táblában tárolom el, hogy ez a lakás helyrajzilag, hol helyezkedik el. Ebben a táblában már található egy külső kulcs is, mivel egy lakás egy felhasználóhoz tartozhat és ezt a kapcsolatot egy egy-egy kapcsolat valósítja meg.

4.2.3. Thread hálózat

Minden ilyen hálózatot úgy terveztem, hogy ezeket a Border Router fogja létrehozni. Vagyis minden hálózat már előre fel kell legyen töltve az adatbázisba, hogy ha a felhasználó hozzájut egy ilyen eszközhöz, akkor azt már csak regisztrálnia kelljen a lakásához. Minden hálózatnak van egy ID-ja, ami ugyancsak automatikusan inkrementált és ez képezi az elsődleges kulcsot is. Egy oszlop tartalmazza, hogy hány eszköz került csatlakoztatásra egy hálózatban, pusztán statisztikai okokból és a későbbiekben ezzel kezelhető le az az eset, hogy ha valaki 544 eszköznél többet szeretne felcsatlakoztatni. A regisztrációs folyamat, úgy zajlik, hogy az eszköznek van egy saját neve, amit deviceCode-nak hívok a táblában és egy titkos jelszava, amit pedig secretCode-nak. Ezeket mellékelni lehet egy papíron minden eszköz mellé.

4.2.4. Eszköz típusok

Mielőtt megismerkednénk az eszközök táblával azelőtt ki kell térnem arra, hogy ez a tábla milyen adatokat tartalmaz. Minden eszköztípusnak van egy ID-ja, ami ugyancsak egy automatikusan inkrementált elsődleges kulcs és további két oszlopa. Az egyik ezek közül tárolja az eszköz típus nevét, például a későbbiekben ismertetett Thread routert és opcionális lámpakapcsolót, mint Blind Access Point, illetve a hozzá tartozó comment-et. A comment oszlopban szövegként tárolok el egy alap konfigurációját az eszköznek. Mivel

Pythonban nagyon egyszerű a JSON kezelés így az adatokat egy JSON fájlba helyezem el, majd ezt konvertálom át szöveggé.

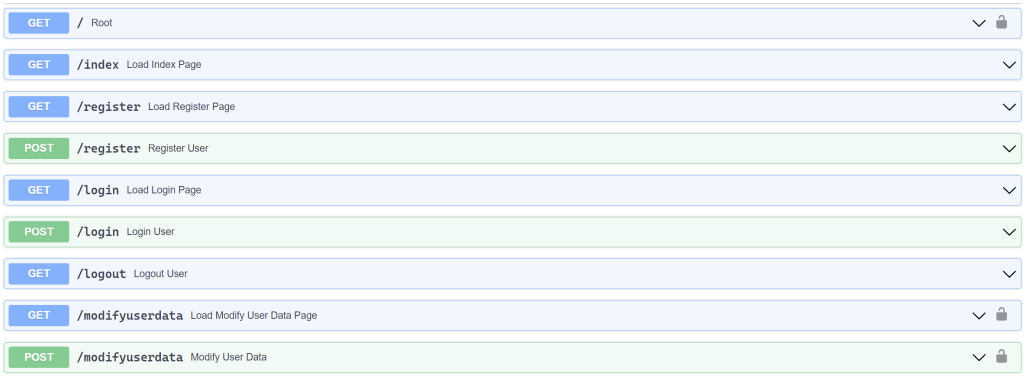
```
BLIND_ACCESS_POINT_DEVICE_TYPE_STRUCT = {
"isLampConnected": False,
"lampstate": False,
"isExtensionBoardConnected": False,
"whichExtendedActive": "00000"
}
```

4.2.5. Eszközök

Most, hogy már minden táblát ismertettem ami ezzel a táblával kapcsolatban áll, rá is térhetünk az eszközökre. Minden eszköznek van egy egyedi azonosítója, ami hasonlít az internet világából ismert MAC címre, csak azzal ellentétben ez 64 biten tárolt szám. Ez az eui64 azonosító, amit már ismertettem egy korábbi részben. Mivel minden eszközre ez egyedi, így ezt alkalmaztam elsődleges kulcsként. Mivel az eszközöket is a gyártó adja és nem tudja, hogy melyik Thread hálózathoz fog csatlakozni ezért a második oszlop ebben a táblában NULL értékkel fog inicializálódni, vagyis megengedhető, hogy a külső kulcs NULL értéket vegyen fel. Ezt jelzi a üres piros kör. Továbbá kell az eszköznek rendelkezni egy típussal, illetve a comment mezőt az alap beállításokkal át kell másolni az eszközhöz. Ez azért szükséges, mert ha egy lámpakapcsolóról van szó akkor minden lámpának egyedi állapotának kell lennie. Ez a tábla egy-több kapcsolatban áll a Thread Network táblával, mivel egy hálózathoz több eszköz is csatlakozhat.

4.3. Backend és weboldal sturktúra

A FastAPI, ahogyan a kezdeti felsorolásban írtam, automatikus dokumentációt készít, amihez a fejlesztő a /docs alatt férhet hozzá. Ennek a dokumentációnak egy részlete a 14. ábrán látható. A bal oldalon mutatja ez a dokumentáció, hogy milyen lekérések esetén hívódik meg egy adott rutin. Például a regisztráció GET lekérés esetén egy regisztrációs oldalra fogja irányítani a felhasználót, ahol HTML Form-okat talál. Abban az esetben, ha minden mezőt kitöltött és érvényes adatokat adott meg, akkor egy POST lekéréssel továbbítja ugyan erre a címre. Az adatok hitelességét és a összeomlás elleni védelmet a frontendben JavaScriptek ellenőrzik és csak is kizárólagosan akkor engedik a lekérést elvégezni, ha minden adatmező kitöltésre került és helyes. POST lekérés után az adatbázisban lementem az adatokat és átirányítom a



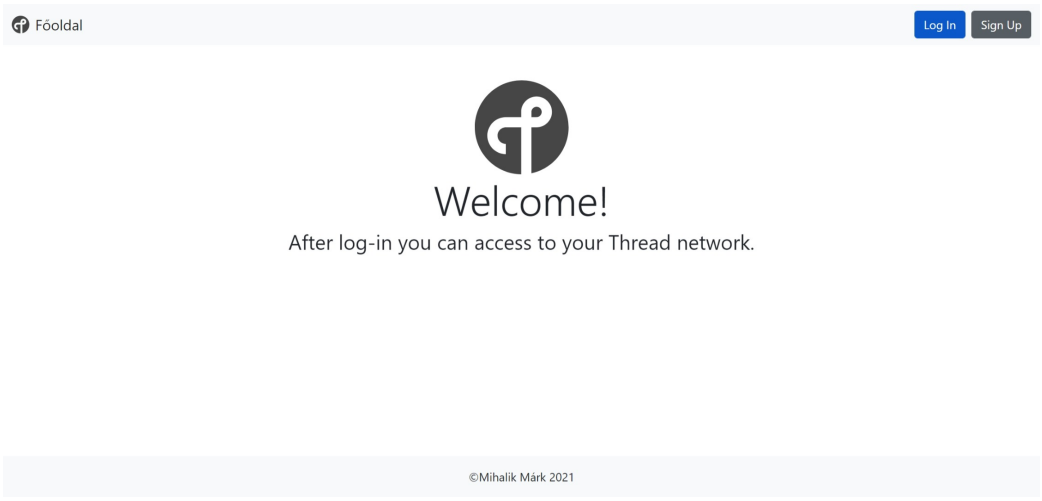
14. ábra. FastAPI automatikus dokumentációja

felhasználót egy visszajelző oldalra, ahol információt kap a regisztrációs folyamat eredményéről, ami lehet sikeres, vagy érvénytelen, mivel a jelszavak nem egyeztek meg, vagy már létezik ilyen felhasználó.
Ahol jobb oldalon lakatot látunk ott az adott weboldalhoz csak hitelesítés (*login*) után lehet hozzáférni. A bejelentkezés *OAuth2* ipari szabványon alapul. Viszont az fontos információ, hogy a hitelesítés történhet egy a HTTP protokoll fejlécében vagy *Cookie*-val. Én az utóbbi megoldást választottam és így a bejelentkezés után a böngészőben egy titkos kulccsal, SHA256 titkosítással eltárolom a bejelentkezett személy felhasználónevét, illetve a lejárati dátumot. Az enkódolás előtt, vagy dekódolás után egy JSON formátumot fogunk kapni és emiatt hívják ezt *JSON Web Token*-nek, vagyis röviden **JWT**-nek.

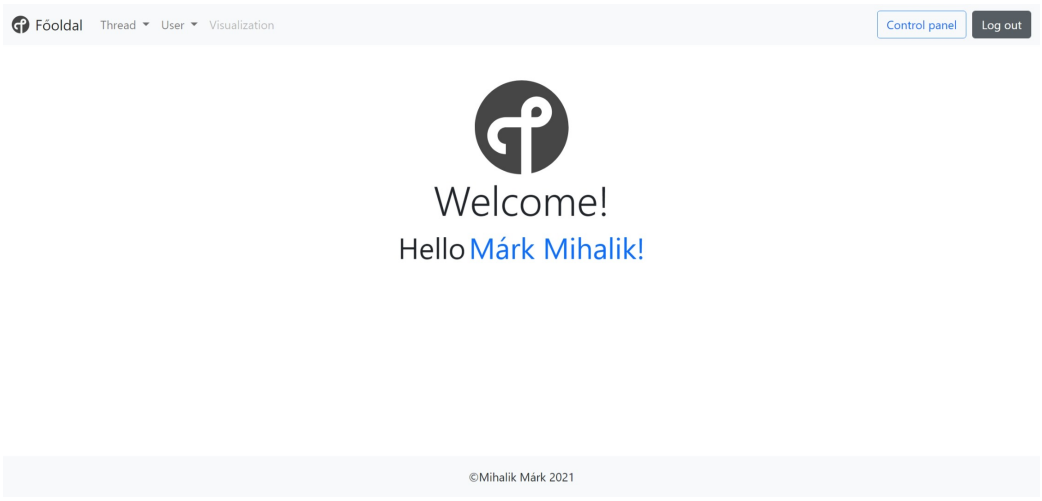
4.4. Frontend

4.4.1. Bejelentkezett felhasználó által elérhető szolgáltatások

A 15. ábrán látható két kép, hogy bejelentkezés előtt, illetve után milyen változások lesznek elérhetőek a felhasználó számára. A fejlécben módosulás történik, mivel az eddig üres helyeken legördülő (*dropdown*) menü jelenik meg. Az első ilyen menü a Thread, ahol a hálózat hozzáadását, esetleges törlését végezhetjük el. A második ilyen dropdown menüben a felhasználó lakhelyét adhatjuk hozzá, illetve ha már hozzá adtuk akkor ezt módosíthatjuk. Jobb oldalon a Login és Register gombok is megváltoztak, mivel helyettük két másik funkció kapott helyet az egyik a vezérlőpult (*Control Panel*), a másik pedig a kijelentkezés (*Log Out*).



(a) Bejelentkezés előtt

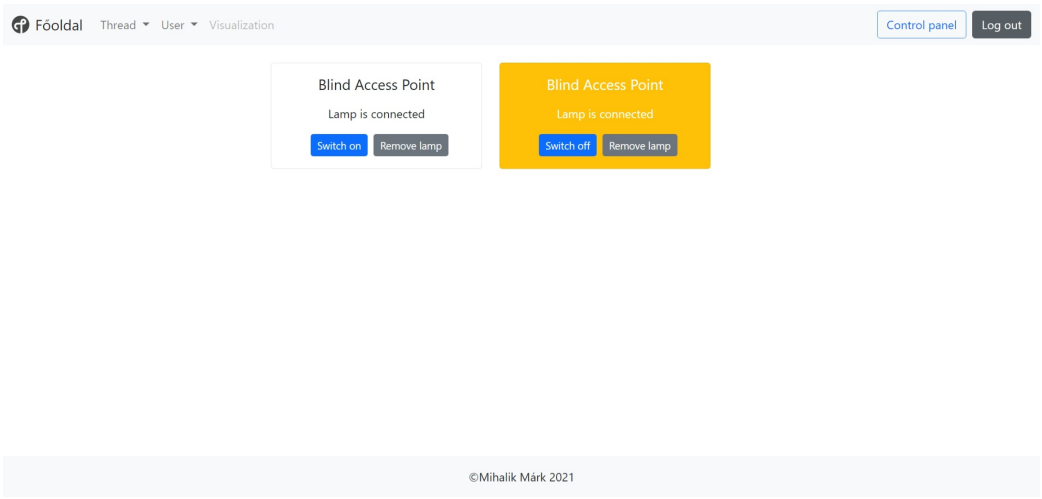


(b) Bejelentkezés után

15. ábra. A weboldal főoldalai

4.4.2. Control Panel

Erre a gombra kattintva a felhasználónak elérhetővé válnak az adatbázisban hozzá tartozó eszközök. A 16. ábrán látható, hogy az adatbázisban két darab eszköz foglal jelenleg helyet, és mind a két eszköz egy Blind Access Point. Az egyiken a lámpát kapcsoló relé aktív állapotban van, a másikon pedig nem, ezeket jelzi a sárga és fehér háttér. A lámpák szoftveresen eltávolíthatóak abban az esetben, ha ezt az eszközt csak routerként vagy REED-ként szeretnénk használni.



16. ábra. A vezérlőpult

4.5. Csatlakoztatás

Abban az esetben, ha csatlakoztatni szeretnénk az eszközünket, akkor a csatlakozáshoz szükségünk van egy Commissioner-re a hálózatban. Minden csatlakoztatáshoz szükség van az eszköz eui64 azonosítójára és az eszköz saját jelszavára (*Joiner Credential*), amit a programozó ad meg az eszközre készített szoftverben. Ez az én esetemben a Border Router, amin az OpenThread Commissionerje fut. Három fajta csatlakoztatási lehetőség biztosított így

- CLI - Command Line Interface (Linux, MacOS),
- Google Commissioner App (Android),
- Thread Group Commissioner (Android).

A CLI (Command Line Interface)-t abban az esetben érdemes használni, ha kézzel szeretnénk beírni az eui64 azonosítót és jelszót vagy valamilyen más módon képzeljük el a csatlakoztatást. Az nRF52840 rendelkezik NFC támogatással, így akár az eszközöknek az azonosítóját ezzel a módszerrel kiolvashatjuk és egy bash vagy Python szkript segítségével elindíthatjuk ezt a programot. Lehetséges megoldás lehet még a chip multiprotokol szolgáltatását kihasználni, így egyszerre futhat például egy Bluetooth LE és a Thread egy magon.
A második lehetőség a Google által készített kezdetleges Android alkalmazás, ami QR kódos csatlakoztatást tesz lehetővé. Először is rá kell csatlakozni az Androidos okostelefonunkkal ugyanarra az otthoni hálózatra, amin megtalálható az átjátszónk. Ha megjelenik, és beírtuk a Thread hálózat létrehozása-

kor beállított jelszót (Border Router maga hozza létre a hálózatot így ezt a jelszót mellékelni kell), akkor egy QR kód olvasóhoz férünk hozzá, aminek tartalmaznia kell a eui64-et és jelszót.
Ettől a megoldástól nem sokban különbözik a Thread Group által kiadott és fejlesztett applikáció, ami sokkal szebb design-nal rendelkezik. Mivel ez a hivatalos app, ezért ezt használtam a csatlakoztatásra. A QR kód generálását a következő forma szerint kell elvégezni:

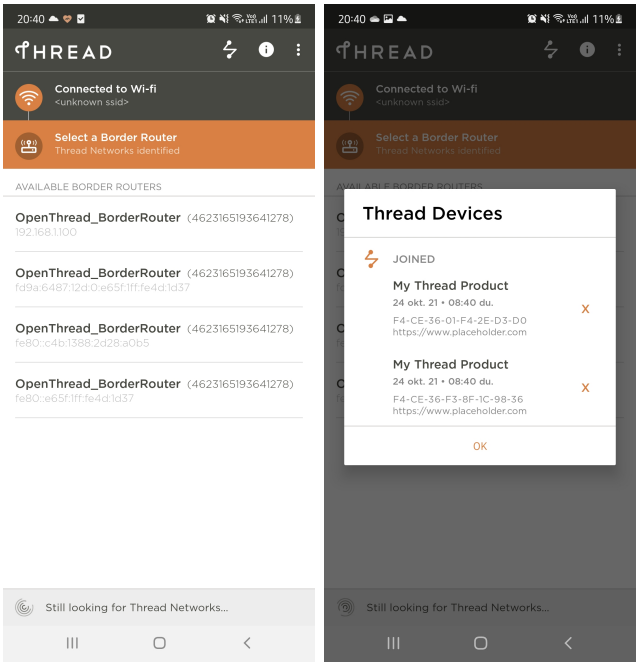
v=1&&eui=f4ce36f38f1c9836&&cc=BBBBBBBB

Ezt a szöveget három részre lehet bontani. Az első a verziószám, mivel a Thread első verziója érhető el. A második az eui64, a harmadik pedig a *Commissioner Code*. A kódgeneráláshoz, egy ingyenes Open-Source Programot használtam, aminek a neve Zint Barcode Studio[16]. A QR kódot az ISO18004-es szabvány szerint kell megalkotni, ahol a szöveg kódolásának UTF-8-at kell választani, vagy különben az applikáció nem fogja tudni az eszközt felcsatlakoztatni.

17. ábra. A csatlakozáshoz szükséges QR kód

4.6. ThreadGroup applikáció

Az applikáció használata nagyon egyszerű. Amint rácsatlakoztunk az otthoni hálózatunkra az Androidos okostelefonunkkal, akkor megkezdődik az átjáró keresése. A 18. ábra bal oldalán láthatjuk a saját, otthoni hálózatomon elérhető egy darab Border Routert, ami a 192.168.1.100-as IPv4 címen helyezkedik el. Továbbá az ez alatt található további három IPv6 cím is ugyan ehhez az eszközhöz tartozik. Amint kiválasztottuk az egyiket, akkor egy felugró ablakba be kell írni a jelszót, amivel létrehoztuk a Thread hálózatot (mellékelve egy papíron, vagy az eszköz alján megtalálható). Hitelesítés után egy QR kód



(a) Thread Átjárók (b) Csatlakoztatott eszközök

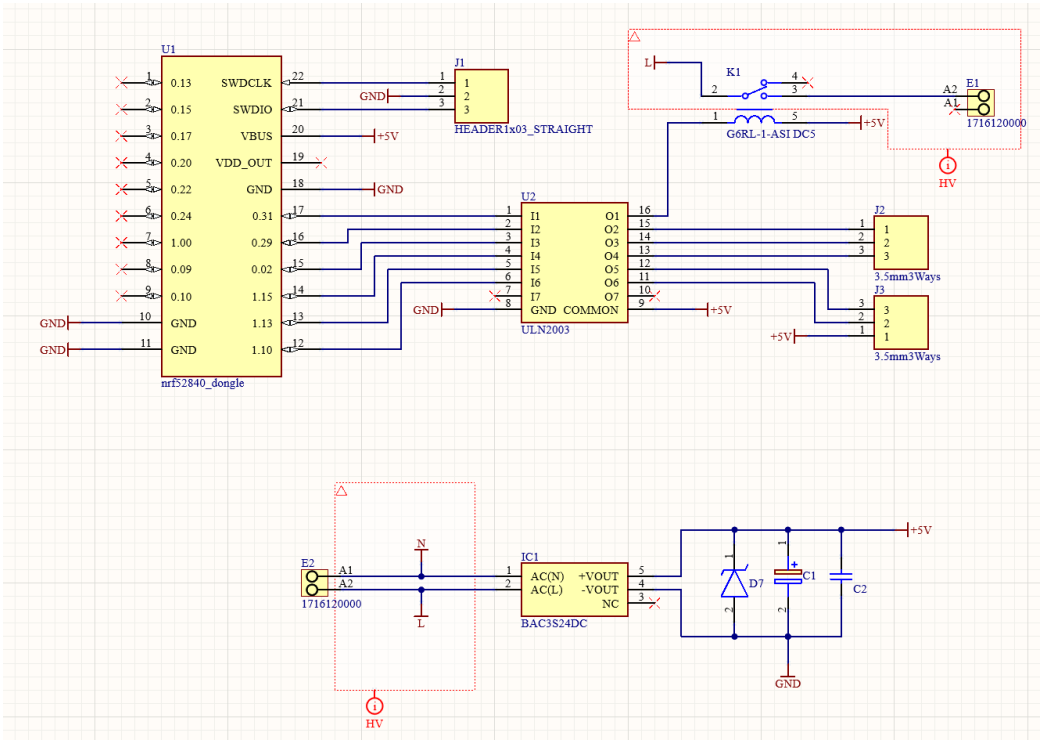
18. ábra. Thread Group applikáció

szkenner jelenik meg a kijelzőn és ha beolvassuk ezzel a csatlakoztatni kívánt eszközhöz tartozó QR kódot, akkor elindul a Commissioner, vagyis a csatlakoztatás. Pár másodperc elteltével, ha sikeresen zárult a csatlakoztatás, akkor egy pipa jelenik meg a kijelzőn. Az applikáció jobb oldali navigációs sávjának legbelsőbb elemére kattintva (két pontot összekötő zig-zag vonal) egy a 18. ábra jobb oldali képéhez hasonló felugró ablak jelenik meg az összes Thread hálózatunkra felcsatlakoztatott eszköznek a listájával. Ezek után ugyan itt távolíthatunk ell eszközöket a hálózatról. Az eszköz nevének a beállítására jelenleg semmilyen módszert nem találtam a protokollon belül.

5. Blind Access Point

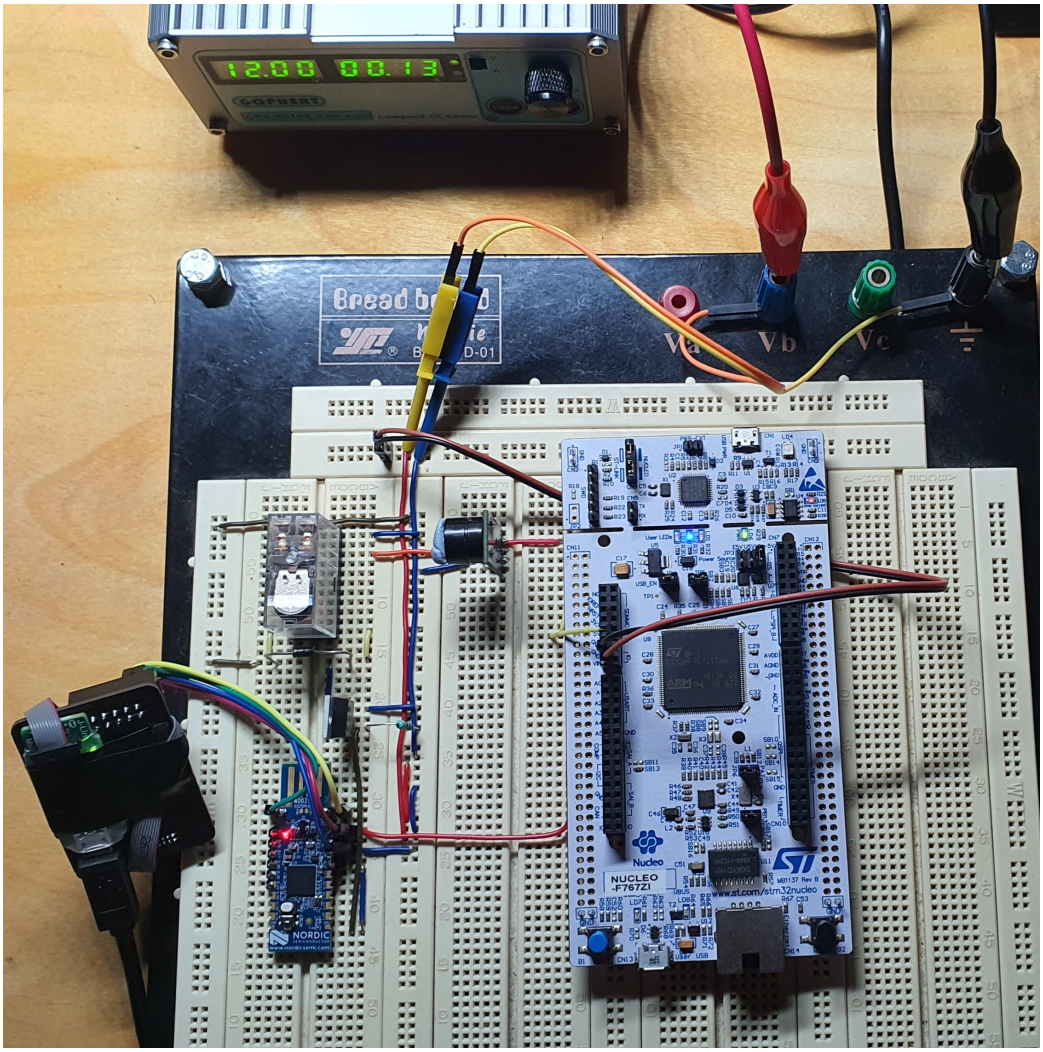
Célom volt egy olyan univerzális lámpa, illetve ajtónyitó elkészítése, ami a villanyszerelők által sokat használt 65 mm kör alakú szerelődobozba illeszthető. Ez az eszköz egy univerzális megoldást biztosít a Thread routerek számára, amivel a hálózat kiterjedése növelhető, úgy hogy a felhasználó szeme elől teljes mértékben el van rejtve.

5.1. Blind Access Point kapcsolási rajz



19. ábra. Blind Access Point kapcsolási rajza

Ezt az eszközt a piacon kapható egyik legkisebb AC/DC átalakító tápkockával (muRata BAC3S05DC) láttam el, ami 230 V hálózati feszültséget alakítja át 5 V DC-re. 1x1 colos méretével és 3 W-os maximális teljesítményével ideális kis eszközök tápellátására. Ez a modul látható a 19. ábra alsó felében. A modul 5 V-os kimenetére egy 7 V-os szupresszor diódát (tranziensvédelem), egy 220 µF-os tantál kondenzátort (simító kondenzátor) és egy 100 nF-os szűrőkondenzátort terveztem, viszont ezekkel a tápkocka is rendelkezik, így ezek csak opcionálisak. A lámpa kapcsolását egy 5 V-os relével oldottam meg, ami a modul fázis vezetékét kapcsolja rá a lámpára. Ez azért fontos, mert így helyes bekötés esetén a felhasználó életét nem veszélyezteti egy izzó csere alkalmával, ha lekapcsolt lámpa mellett véletlenül megérinti a fázis pontot. A kapcsolási rajzon látható a két HV-vel megjelölt blokk, ami jelentős, mivel így minimum 1,3 mm-re távolságnak kell lennie a fázis és nulla vezetősáv között, ezzel alkotva egy Altium szabályt.



20. ábra. Blind Access Point próbapaneles összeállítása

5.2. Próbapaneles összeállítás és szoftver készítés

A NYÁK tervezés előtt elkészítettem dugaszolható próbapanelen a koncepciót, ami 20. ábrán látható. A koncepcióhoz nem álltak rendelkezésre az adott alkatrészek, így ezeket az itthon megtalálható eszközeimmel helyettesítettem és a szoftvert ennek megfelelően készítettem el. A szoftvert a már megismert Zephyr-ben írtam meg, ahol két szál áll végzi el az eszköz feladataid. Az egyik szál kezeli az OpenThread-et, illetve az általam megírt Border Router címének lekérdezését, csatlakozást indító és kezelő függvényt, illetve a speciális eseteket, amit kezelnie kell a programozónak. Ilyen eset például az eszköz csatlakoztatása után az automatikusan meghívódó *callback* függvény,

ami információt ad az eszköz hálózatban betöltött szerepéről. A másik szál pedig a lámpa állapotának az ellenőrzését, illetve a lámpa kapcsolását végzi el. Ezt úgy valósítottam meg, hogy fél másodpercenként a Border Routerre az eszköz lekéréseket végez, ezután az ezen tárolt adatbázisból az eszköz tábla comment oszlopából kiolvassa az eszköz eui64 azonosítója alapján a lámpa állapotát, amennyiben az csatlakoztatva van. Válaszban pedig az eszköz felé továbbít egy szöveget, ami "true" vagy "false" lehet. Ezt az eszköz feldolgozza és így befolyásolja a lámpa állapotát.

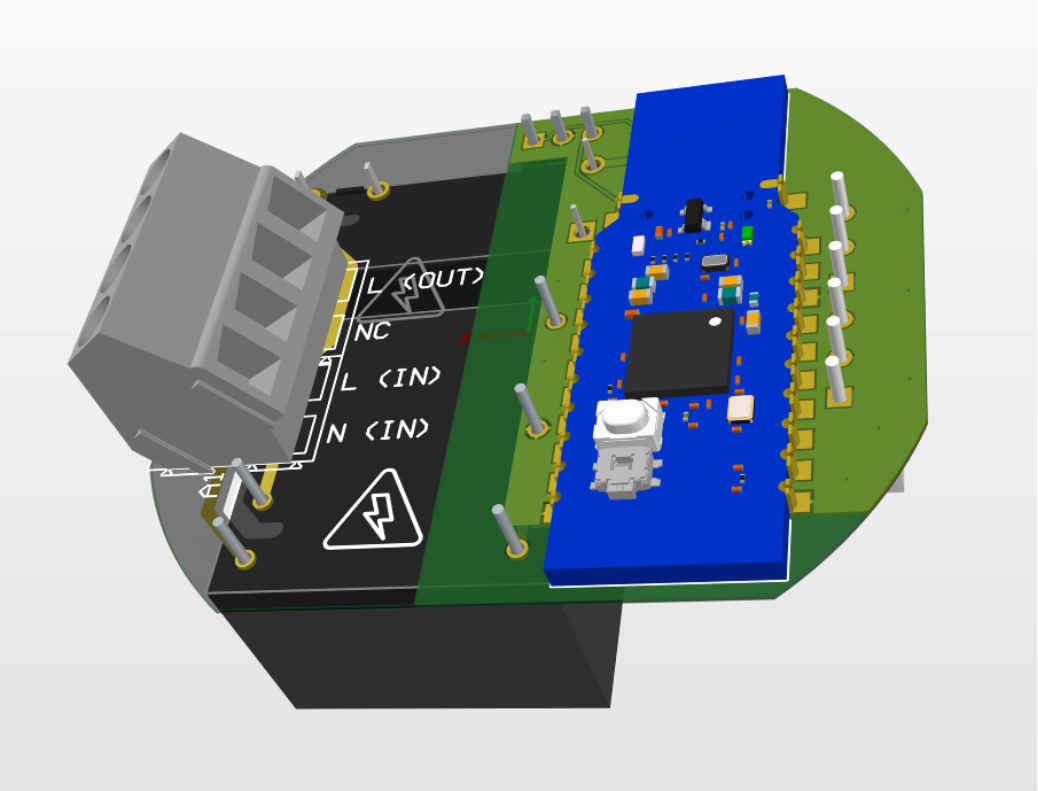
5.3. Blind Access Point NYÁK terv

A NYÁK terven (ami a 21. ábrán látható) a hálózati feszültségű részt megpróbáltam a legnagyobb távolságra helyezni az 5 V résztől. A 230 V-os részen kivágást terveztem a fázis és nulla vezetősávok közé, így megfelelő nagy távolságot alakíthattam ki, mivel száraz levegőben az átütési szilárdság 1,3 kV-nak vehető 1 mm-re vonatkoztatva. Nedves levegőben ez képes 600 V feszültségre csökkenni[17], de ez még így is majdnem kétszer nagyobb, mint a 230 V-os hálózati feszültség amplitúdójának a maximuma. A forrasztás gátló maszkot eltávolítottam a nagyfeszültségű résztől, így biztosítva, hogy a lakk ne szívhassa meg magát nedveséggel és így csökkentse le az átütési szilárdságot. Abban az esetben, ha egy lámpa nem lenne elég, akkor öt darab 5 V-os kimenet rendelkezésre áll egy relé modulnak. A relék vezérlését egy hat önálló Darlington kapcsolással rendelkező IC-vel (ULN2003) oldottam meg, ami rendelkezik flyback diódákkal minden egyes kimeneten, így biztosítva a hosszabb élettartamot.

6. Power Node

Célom volt egy olyan MTD eszköz megtervezése, ami egy nagyobb kapacitású Lithium-Polymer akkumulátor celláról üzemeltethető és egy áramkörön integrál öt különböző célterületre szánt eszközt, ami lehet

- PIR (Pyroelectric Infrared Sensor) node, vagyis mozgásérzékelő modul,
- nagyobb teljesítményű 5 V-os kimenettel rendelkező motor meghajtó (pl. kis teljesítményű öntöző szivattyú),
- környezeti tényezőket figyelő modul (hőmérséklet, páratartalom, légnyomás és fényerő),
- külső tápigényű nagyobb teljesítményű eszközök kapcsolására szolgáló modul,



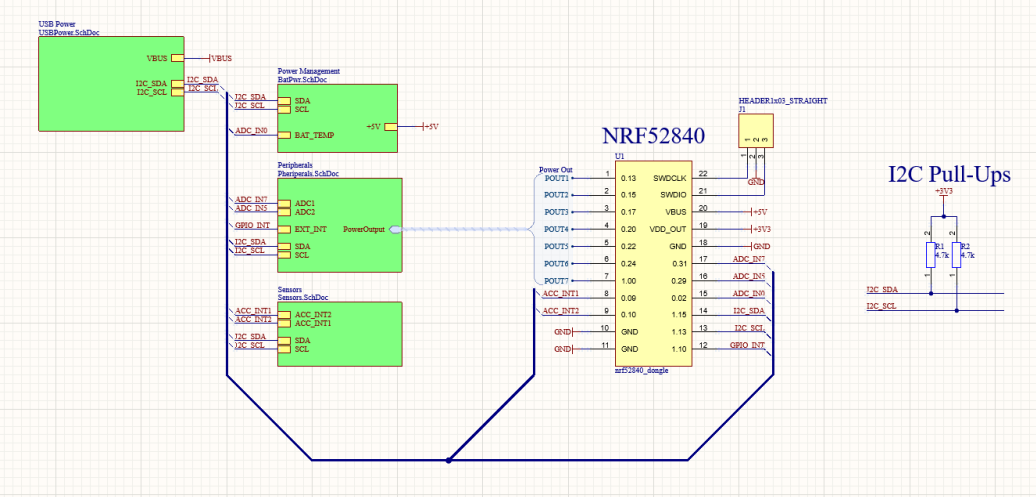
21. ábra. Blind Access Point 3D megjelenítőben

– mozgásérzékelő, ajtónyitás érzékelő modul.

Ezek mindegyike elérhető egy NYÁK-on és a végső felhasználástól függően a különböző részek beforrasztásától függően egy vagy több funkció elérhető.

6.1. Power Node kapcsolási rajz

A 22. ábrán látható ennek az eszköznek a kapcsolási rajza. Bal fentről lefelé haladva az első doboz jelöli az USB Type-C töltő csatlakozót, amivel az eszköz használható külső 5 V-os tápfeszültségről. Az I2C buszról a teljesítmény mérő IC-nek a kiolvasása végezhető. Ettől a blokktól jobbra helyezkedik el a töltő és a Li-Po akkumulátor védelme a BPS (Battery Protection System) áramkör. Az I2C busz egy másik teljesítmény mérő IC mért adatainak az elérésére szolgál, mivel így lehet információhoz juttatni a mikrokontrollert a feszültségforrás állapotáról, amiből később jelezni lehet a felhasználónak, ha az eszköz töltésre szorul. Másrészről így lehet védekezni az ellen, ha a felhasználó túl nagy áramot szeretne kivenni a rendszerből, ami az eszköz tönkremeneteléhez

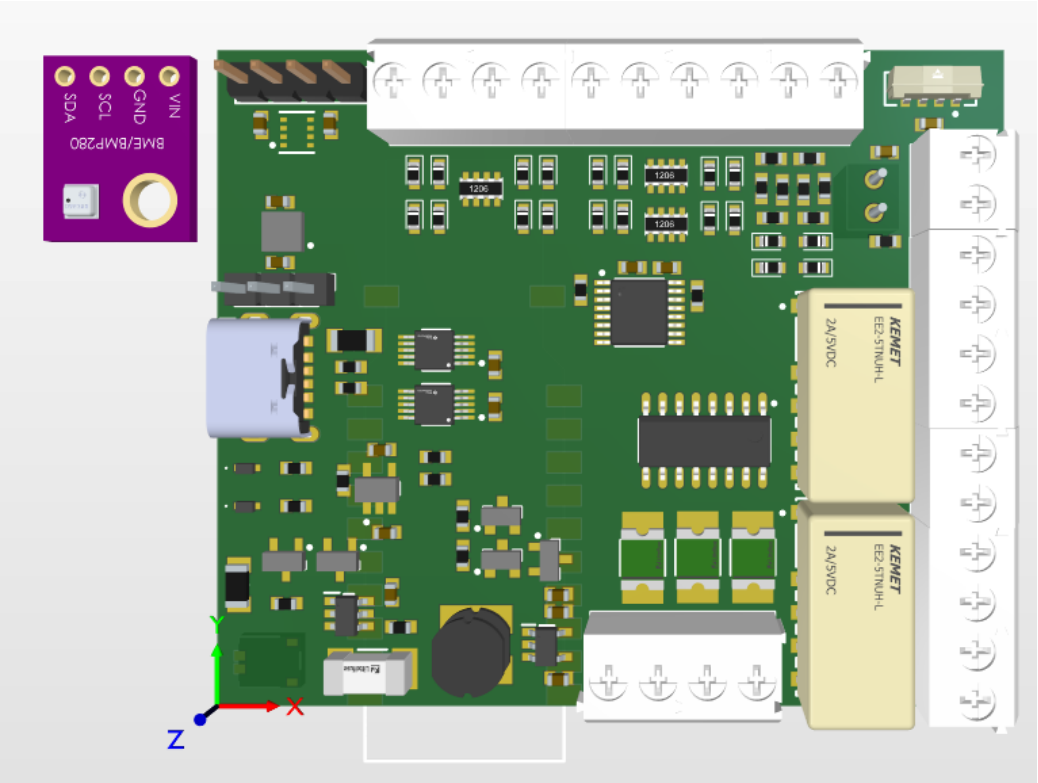


22. ábra. Power Node kapcsolási rajza

vezetne. Ebben a blokkban helyezkedik el továbbá egy 5 V boost (step-up) konverter is, ami a forrás 3 és 4,2 V-os feszültségét alakítja 80 %-os hatásfok mellett. A perifériák részben található meg három 200 mA terhelhetőségű kimenet (200 mA-es pattanó biztosítékokkal védve), illetve két impulzus relé, ami külső tápfeszültség kapcsolását teszi lehetővé maximum 48 V-ig. A digitális be- és kimeneteket egy GPIO bővítőcel (GPIO-Expander) oldottam meg, mivel a Dongle-n nem állt rendelkezésre kellő ki és bemenet. Továbbá ezek 5 V-os logikai szinttel működnek, mivel az IC (TCA6408) amivel dolgoztam külön tápforrást igényel a mikrokontroller kommunikációjához, illetve a GPIO-khoz, így szintillesztőként is funkcionál. Abban az esetben, ha a bemeneten változás történik, akkor egy megszakítás generálódik a mikrokontroller felé. A kapcsolási rajzon továbbá található két analóg bemenet is, ami túlfeszültség védelemmel az nRF52840 ADC-hez huzalozott lábaira van kötve. A szenzor résznél pedig található egy I2C-s fénymérő szenzor, egy BME280-as hőmérséklet, páratartalom és légnyomás mérő szenzor (IC és modul formában is megtalálható a panelen az LGA tokozás miatt) és egy gyorsulás mérő szenzor.

6.1.1. Power Node felépítési terv

Jelenleg a NYÁK terv félkész állapotban van, mivel csak az alkatrészek elhelyezése történt meg eddig. A panel méretének az általam választott 2500 mA h kapacitású Li-Po méretét választottam, ami 50x60 mm. A kialakított elrendezés a 22. ábrán látható.

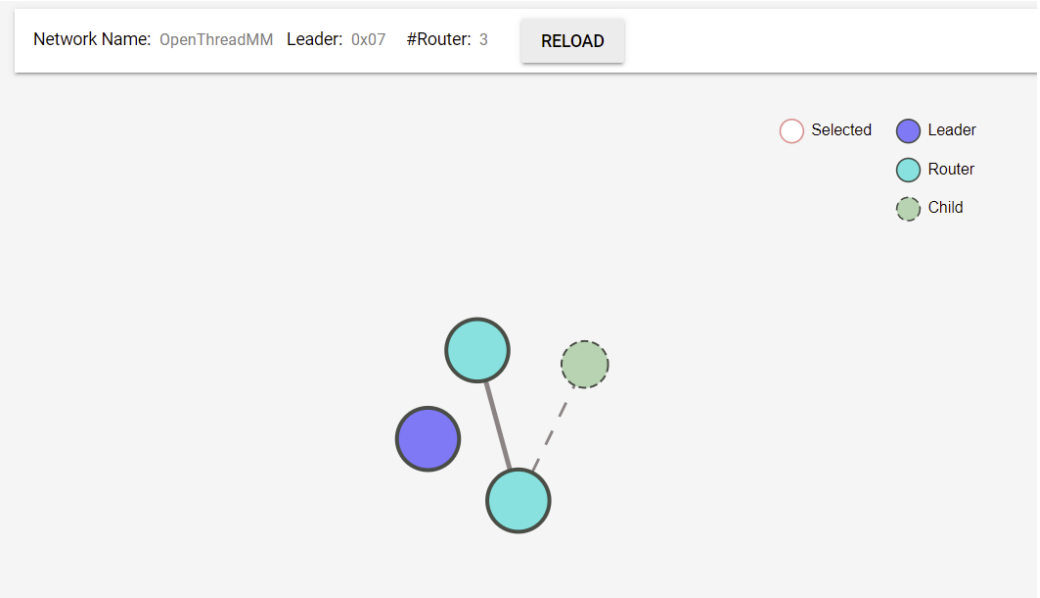


23. ábra. Power Node felépítési terv

7. Megvalósított Thread hálózat

7.1. Thread hálózat topológiája

A legyártott három Blind Access Point-tal és a Border Routerrel létrehoztam a saját otthoni Thread hálózatomat. Ezt egy 40 mm^2 alapterületű lakásban tettem meg és a négy eszközt a ház négy pontjára helyeztem el. Mindegyik eszköz sikeres csatlakoztatása után a kialakult topológia a 24. ábra alapján alakult. Jól látható, hogy a négy eszköz közül három Router szerepet egy pedig End Device (Child) szerepet töltött be. Ez azért tapasztalható, mivel az OpenThread megpróbálja az egész Thread hálózatban résztvevő routerek számát 16 és 23 eszköz között tartani. Ez a topológia, azért alakult ki, mivel a kép készítésének időpontjában az egyik FTD eszköz még nem tudta magát routernek kinevezni. A tesztelés során az eszközök stabil kapcsolatban voltak egymással és mindegyik eszköz elérhető volt a hálózatban.

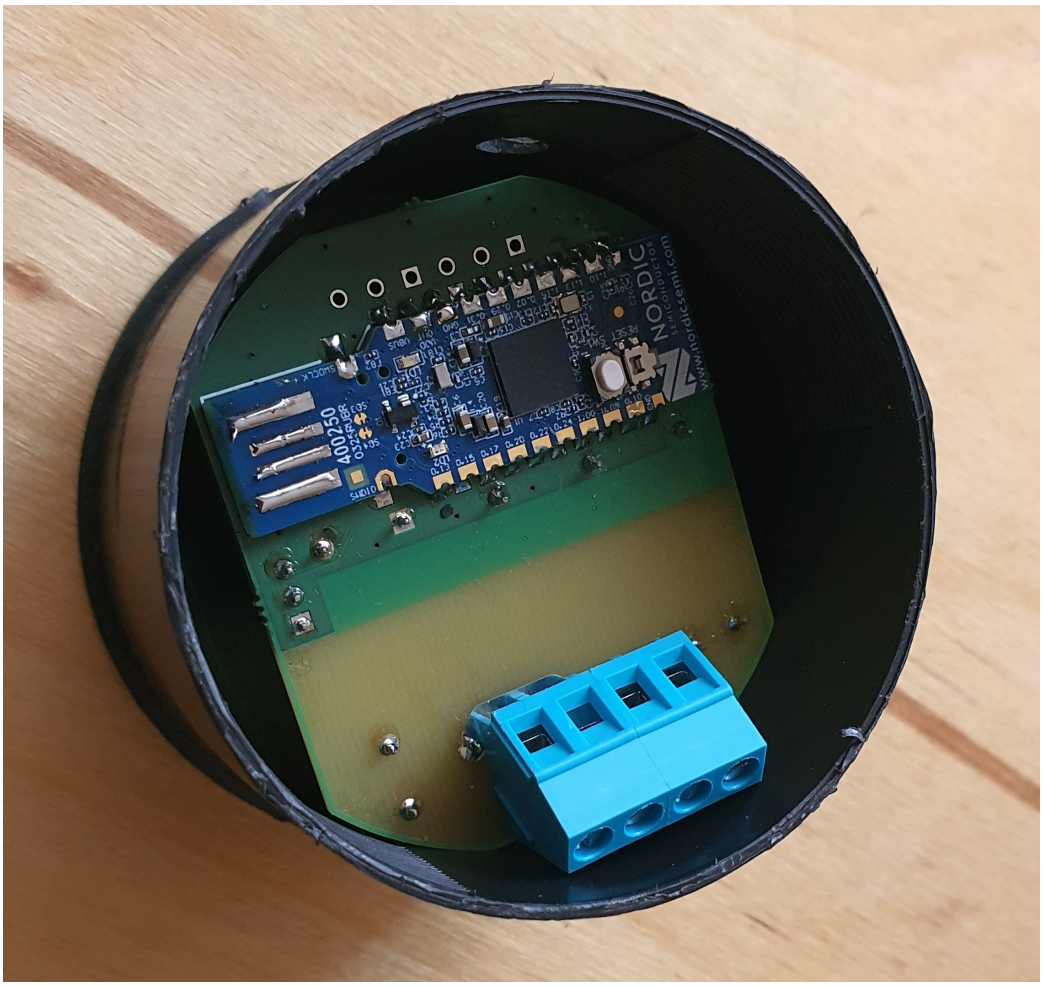


24. ábra. Saját Thread hálózatom topológiája

7.2. Border Router működés közben

25. ábra. Összeszerelt Border Router

7.3. Az elkészített Blind Access Point



26. ábra. Összeszerelt Blind Access Point a szerelődobozban

7.4. Kitekintés

Az általam bemutatott használati esetek csak egy kis részét képezi a megvalósítható eszközöknek. Nagy amerikai cégek, mint az Apple és a Google is fogalomba bocsájtott több olyan eszközt, ami támogatja a Thread protokollt. Ilyen eszköz például az Apple HomePod mini vagy a Google Nest Hub Max. Az érdeklődőknek a Thread Group[18] oldalán elérhető egy részletes lista a Thread által hitelesített eszközökről.

8. Összefoglalás

Ebben a TDK dolgozatban a kutatásom keretein belül ismertettem a Thread hálózatokban rejlő potenciális lehetőségeket okosotthonokban való alkalmazásra. Részletesen megismertem és bemutattam a Google által implementált Thread protokollt, az OpenThread-et és az előforduló eszközök szerepeit. Ismertettem az olvasót egy olyan szerver-kliens protokollal, ami beágyazott környzetben, kisebb erőforrásokkal rendelkező eszközökben is, kis overhead-el implementálhatunk. Továbbá ismertettem a Zephyr Project forradalmi megoldását különböző architektúrájú processzorokra, amivel egyszerűen, gyorsan és platformfüggetlenül lehet az elképzeléseinket a gyakorlatban szoftveres formában implementálni és amivel a kód módosítása nélkül készíthetünk különböző, ugyan olyan tudású hardverre programokat. Jelenleg 350-nél több fejlesztői kártyát támogat és megtalálhatóak benne különböző rétegek, protokollok, amikkel egyszerűen készíthetünk IoT eszközöket. Saját munkám keretein belül pedig elkészítettem egy koncepciót, amikkel a Thread által nyújtott lehetőségek elérhetőek a fogyasztók számára. Ez a koncepció tartalmaz egy adatbázis modellt és egy könnyen kezelhető GUI-t, amivel bárki, bárhonnan managelheti a saját hálózatát. A koncepcióban helyet kapott egy kész, dobozolt átjáró, amivel a felhasználó létrehozhat egy saját hálózatot, amit rácsatlakoztathat az otthoni hálózatára és az internetre. Elkészítettem egy olyan eszköz tervét, amit a felhasználás helyén látatlanul a falba lehet rejteni és biztosítja a hálózat gerincét, illetve egy adott szoba világítását lehet vele kontrollálni. Majd ezt a tervet kézzel foghatóvá tettem és egy lakásban több ilyen eszközzel kialakítottam a saját Thread hálózatomat, amivel bebízonyítottam ennek a koncepciónak a működését.

Tartalomjegyzék

Ábrák jegyzéke

Irodalomjegyzék és hivatkozások

[1] *Statista felmérése.* URL: *https://www.statista.com/outlook/dmo/smart-home/worldwide#smart-homes*.

[2] *FTD Thread halozat.* URL: *https://openthread.io/guides/images/ot-primer-leader_2x.png*.

[3] *OpenThread Node típusok.* URL: *https://openthread.io/guides/thread-primer/node-roles-and-types*.

[4] *Thread hálózat IP cimek szerinti csoportosítása.* URL: *https://openthread.io/guides/images/ot-primer-leader_2x.png*.

[5] *OpenThread IPv6 címkiosztása.* URL: *https://openthread.io/guides/thread-primer/ipv6-addressing*.

[6] *CoAP memória igénye.* URL: *https://coap.technology/*.

[7] *CoAP Cheatsheet.* URL: *https://github.com/markushx/coap-cheatsheet*.

[8] *CoAP protokol részletes specifikációja.* URL: *https://datatracker.ietf.org/doc/html/rfc7252*.

[9] *Zephyr Project felepitese.* URL: *https://se.ewi.tudelft.nl/desosa2019/chapters/zephyr/*.

[10] *Nordic nRF52840 datasheet.* URL: *https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf*.

[11] *Spinel Protokoll dokumentációja.* URL: *https://datatracker.ietf.org/doc/html/draft-rquattle-spinel-unified*.

[12] *Raspberry Hats GitHub könyvtára.* URL: *https://github.com/raspberrypi/hats*.

[13] *BCM2711 adatlap.* URL: *https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf*.

[14] *OpenThread Border Router GitHub könyvtár.* URL: *https://github.com/openthread/ot-nrf528xx*.

[15] *OpenThread Commissioner GitHub könyvtár.* URL: *https://github.com/openthread/ot-commissioner*.

[16] *Zint Barcode Generator.* URL: *https://zint.github.io/*.

[17] *DC Flashover of a Dielectric Surface in Atmospheric Conditions.* 2004. URL: *https://www.researchgate.net/publication/3165903_DC_Flashover_of_a_Dielectric_Surface_in_Atmospheric_Conditions*.

[18] *Thread hitelesített eszközök.* URL: *https://www.threadgroup.org/What-is-Thread/Thread-Benefits*.