

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

ДИПЛОМНА РАБОТА

**по професия код 523050 „Техник на компютърни системи“
специалност код 5230502 „Компютърни мрежи“**

Тема: Проектиране на сензорна IoT мрежа, използваща протокола Thread.

Дипломант:
Любомир Калинов Начев

Научен ръководител:
маг. инж. В. Деведжиев

**СОФИЯ
2023**

На тази страница се слага официалното

**ЗАДАНИЕ
за дипломна работа**

Съдържание

Съдържание

Увод

Първа глава - Проучване на съществуващи сензорни IoT мрежи, протоколи и компоненти

1.1. Проучване и сравнение на съществуващи сензорни IoT мрежи

 1.1.1. Приложение на IoT мрежите

 1.1.2. Съществуващи IoT сензорни мрежови решения

 1.1.3. Сравнение и анализ на разгледаните съществуващи IoT мрежови решения

1.2. Проучване и сравнение на мрежовите протоколи, използвани в съществуващите сензорни IoT мрежи

 1.2.1. Проучване на безжични протоколи, използвани в съществуващите сензорни IoT мрежи

 1.2.2. Сравнение и анализ на разгледаните мрежови протоколи, използвани в съществуващите сензорни IoT мрежи

 1.2.3. Проучване на протоколи за предаване на данни, използвани в съществуващите сензорни IoT мрежи

 1.2.4. Сравнение и анализ на разгледаните протоколи за предаване на данни, използвани в съществуващите сензорни IoT мрежи

1.3. Проучване, сравнение и анализ на основните компоненти, използвани в съществуващите сензорни IoT мрежи

 1.3.1. Проучване, сравнение и анализ на микроконтролери и едноплаткови компютри, използвани в съществуващите сензорни IoT мрежи

 1.3.2. Проучване, сравнение и анализ на комуникационни схеми използвани в съществуващите сензорни IoT мрежи

1.3.3. Проучване на сензори и актуатори, използвани в съществуващите сензорни IoT мрежи

Втора глава - Основни изисквания и блокови схеми

2.1. Основни изисквания към проектираната сензорна мрежа

2.1.1. Архитектурни и комуникационни изисквания

2.1.2. Функционални изисквания

2.1.3. Конструктивни изисквания

2.2. Обща блокова схема и функции на блоковете

2.2.1. Функции на отделните блокове

2.3. Схеми и описание на отделните блокове

2.4 Схеми и описание на отделните подблокове

Трета глава - Проектиране на принципна електрическа схема

3.1. Избор на CAD система

3.2. Водещ възел

3.2.1. Избор на подходящи компоненти

3.2.2. Принципна електрическа схема

3.3. Основен възел

3.3.1. Избор на подходящи компоненти

3.3.2. Принципна електрическа схема

Четвърта глава - Проектиране на печатна платка

4.1. Корпуси и 3D модели на компонентите

4.2. Основни изисквания

4.2.1. Общи

4.3. Печатна платка на Водещ възел

4.3.1. Слоеве на печатната платка на Водещия възел

4.3.2. Финален вариант на печатната платка

4.3.3. 3D модел на печатната платка

4.4. Печатна платка на Основен възел

4.4.1. Слоеве на печатната платка на Основния възел

4.4.2. Финален вариант на печатната платка

4.4.3. 3D модел на печатната платка

4.5. Поръчка на печатните платки

4.5.1. Поръчка на печатната платка на Водещия възел

4.5.2. Поръчка на печатната платка на Основния възел

Пета глава - Проектиране на алгоритми и сурс код на управляващ софтуер

5.1. Използвани програми, инструменти и развойна среда

5.1.1. Софтуерни рамки (на англ. Frameworks)

5.1.2. Операционни системи в реално време (RTOS)

5.1.3. Разширения

5.2. Избор на подходящ мрежови протокол

5.2.1. OpenThread

5.2.2. Thread оптимизация на IEEE 802.15.4

5.2.3. Thread

5.3. Функционална диаграма на работата на Водещия възел на ниво ОС (FreeRTOS)

5.4. Блокова схема на алгоритъма на работа на Водещия възел

5.5. Функционална диаграма на работата на Основния възел на ниво ОС (Zephyr)

5.6. Блокова схема на алгоритъма на работа на Водещия възел

5.7. Сурс код за Водещия блок

5.8. Сурс код за Основния блок

Шеста глава - Практически резултати

6.1. Конфигуриране на работна среда и инсталация на инструменти

за разработка

6.1.1 Visual Studio Code, nRF Connect SDK, ESP-IDF, VS
разширения

6.1.2 Създаване на Github repository

6.2. Създаване на софтуер за основния възел

6.2.1. Създаване на Openthread устройство с конзола

6.2.2. Увеличаване на мощността на предаване на модулите

6.2.3. Свързване и конфигуриране на BH1750

6.2.4. Свързване и конфигуриране на BME688

6.2.5. Създаване на UDP Send функция

6.3. Създаване на софтуер за управляващия възел

6.3.1. Конфигуриране на nRF52840-Dongle като RCP

6.3.2. Създаване на работещ Border Router

6.3.3. Отваряне на UDP port за приемане на данни

6.3.4. Създаване и конфигуриране на WebSocket

6.3.5. Свързване и конфигуриране на SSD1306 Display

6.4. Процес на създаване на елементите на работещ прототип

6.4.1. Реализация на макети на прототипни платки

6.4.2. Измерване на максималното разстояние за приемане и
предаване на информация

6.4.3. Създаване на печатна платка на основния възел

6.4.4. Създаване на печатна платка на управляващия възел

6.4.5. Показване на трафик, наблюдаван с помощта на Wireshark

Заключение

Използвани съкращения

Използвана литература

Увод

Така наречената концепция Интернет на нещата (на англ. *Internet of Things*, съкр. IoT) представлява мрежа от физически обекти, които имат вградени електронни устройства, съдържащи сензори, микроконтролери, управляващ софтуер и др. технологии. Мрежата е създадена с цел свързване и обмен на данни с други устройства и системи, както и с външната околната среда през Интернет. Тази концепция за пръв път е формулирана преди около 20 г., но в последните 10 г. се изпълва с ново технологично съдържание и се появяват множество практически приложения, вследствие на което тя се счита за възходяща тенденция в съвременните Информационни технологии (на англ. *Information technology* съкр. IT) главно благодарение на широкото разпространение на съвременните безжични технологии. Доказателство за това е, че тези устройства съществуват вече навсякъде в бита и бизнеса и улесняват нашето ежедневие, като броят им до края на 2022 г. ще надхвърли 13 милиарда, а до края на 2025 г. - 75 милиарда.

Основните компоненти и технологии, които се използват в една IoT мрежа, са следните:

- Сензори - устройства, които реагират на някакъв вид въздействие от околната среда. Въздействието може да бъде светлина, топлина, движение, влага, налягане и др. Изходът на сензора обикновено е сигнал, който се преобразува и предоставя на потребителя в разбираем за него формат.

- Микроконтролери - големи интегрални схеми, които състоят от различни блокове например микропроцесор, таймери, броячи, входно/изходни (на англ. *input/output*, съкр. I/O) и други интерфейси, памет с произволен достъп (на англ. *Random Access Memory*, съкр. RAM), памет само за четене (на англ. *Read-only Memory*,

съкр. ROM) и др. Основната роля на микроконтролера е да изпълнява алгоритъм, който съдържа предварително дефинирани от потребителя задачи.

- **Мрежови протоколи** - набор от правила или процедури за предаване на данни между електронни устройства, свързани чрез някакъв вид компютърна мрежа. Протоколите се създават от международни или индустриски стандартизиращи организации.

- **Безжични технологии** - позволяват комуникация между устройствата или трансфер на данни от една точка до друга без използването на кабели. Голяма част от комуникацията бива осъществена с радиочестоти или инфрачервени вълни.

- **Сензорни мрежи** - IoT мрежи, които използват описаните по-горе компоненти и технологии, за да предават данни за условията в различни видове среди или обекти. Те използват Интернет за връзка с други компютърни мрежи, за да могат получените данни да се използват и анализират.

Целта на настоящата дипломна работа е създаването на безжична сензорна IoT мрежа, която комбинира всички гореописани компоненти и технологии. За постигането ѝ ще бъде направено проучване и анализ на съществуващи подобни IoT мрежи, използваните за тяхното реализиране основни компоненти, технологии и протоколи. На тази база ще бъдат дефинирани основните изисквания към проектираната IoT мрежа, ще се дефинира блоковата схема, ще се изберат основните компоненти, ще се проектират принципната електрическа схема и печатна платка и ще се синтезират алгоритъма и сурс кода на управляващ софтуер. В резултат на това ще бъде създаден прототип на проектираната мрежа и ще бъдат представени постигнатите практически резултати.

Първа глава - Проучване на съществуващи сензорни IoT мрежи, протоколи и компоненти

1.1. Проучване и сравнение на съществуващи сензорни IoT мрежи

1.1.1. Приложение на IoT мрежите

IoT мрежите са довели до революционна промяна в здравеопазването, производството, търговията, енергетиката и селското стопанство. Много корпорации интегрират IoT мрежи, за да имат предимство пред конкуренцията. Те се фокусират върху увеличаването на ефективността на производството чрез управление на данните в реално време и автоматизация на задачите си.

Автоматизацията и дистанционното управление, постигнато чрез IoT мрежите, предоставят на фирмите оптимизиран баланс между потреблението и пестенето на енергия. По този начин компаниите стават по-енергийно ефективни и намаляват своя въглероден отпечатък върху околната среда [1].

Няколко примера за интегрирането на IoT в ежедневието:

- Превозни средства - Самоуправляващите се автомобили и камиони използват набор от свързани устройства, за да се движат безопасно по пътищата при всякакви видове трафик и метеорологични условия. Използваните технологии включват: камери с изкуствен интелект (на англ. *Artificial Intelligence*, съкр. AI), сензори за движение и бордови компютри.

IoT мрежи се използват и в традиционните превозни средства, като се инсталират свързани устройства за наблюдение на производителността и управлението на компютъризирани системи.

- Управление на трафика - Пътната инфраструктура също стана по-свързана през последното десетилетия с камери, сензори, управление на светофари, паркомери и дори приложения за трафик, предаващи данни, които след това се използват за предотвратяване на задръствания, инциденти и осигуряване на гладко пътуване.

Свързаните устройства могат да се използват за намиране на открыти места за паркиране и предаване на тази информация на водачите.

- Умна електрическа мрежа (*Smart Grid*) - Тази мрежа позволява наблюдение в реално време на данните относно търсенето и предлагането на електроенергия. Тя включва прилагане на AI за ефективно управление на ресурсите. Компаниите за обществени услуги могат да използват тази технология за по-ефективно управление на прекъсванията или, за да идентифицират разпределението на товара и да подобрят надеждността.

- Мониторинг на околната среда - Свързаните устройства могат да събират данни, които показват качеството на въздуха, водата и почвата. Те могат също да събират метеорологични и други видове данни за околната среда.

- Умни сгради и домове - IoT системите се използват от собствениците на имоти за направата на всички сгради по-енергийно ефективни, комфортни и, вероятно, по-безопасни.

Отделно, външни потребителите могат да инсталират интелигентни технологии като ключалки на вратите, уреди, терmostати и детектори за дим, които им помагат с ежедневните им нужди.

- Управление на веригата за доставки - Чрез сензори с технологии за проследяване е възможно следенето на изпратени или получени артикули, позволявайки на клиентите и мениджърите да знаят винаги настоящото местонахождение на техните пратки и по този начин намаляват риска от тяхното загубване.

- Здравеопазване - Технологията, наричана още Интернет на медицинските неща (на англ. *Internet of Medical Things*, съкр. IoMT), може да помогне при наблюдението и поддържането на жизненоважни показатели, които да обосноват вземането на правилни клинични решения.

IoT медицинските устройства могат да помогнат при дистанционно наблюдение в реално време на пациенти, като незабавно докладват на лекарите специфични случаи като астматични пристъпи, сърдечна недостатъчност и др. Това може да доведе до потенциалното спасяване на живота на много хора.

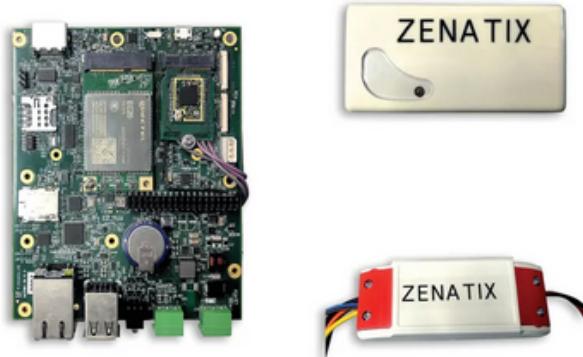
1.1.2. Съществуващи IoT сензорни мрежови решения

WattMan

WattMan представлява безжична сензорна мрежа, базирана на протокола OpenThread, създадена от индийската компания Zenatix Solutions [2]. Мрежата се използва за оптимизиране на количеството използвана енергия за "умно" осветление и "умен дом" устройства чрез автоматизирано наблюдение и контрол.

Мрежата се състои от множество устройства, които представляват комбинация от сензори, свързани към nRF52840 SoC (на англ. *System on a Chip*) модули [3]. Всяко от тях предава данни към маршрутизатори, отново базирани на nRF52840, които препращат данните към централния шлюз (базиран на Raspberry Pi Compute Module). Самият шлюз използва Ethernet, Wi-Fi или клетъчна свързаност за по-нататъшно изпращане на данните към облака. Тази комуникация става изцяло в т. нар. персонална локална мрежа (на англ. *Personal Area Network*, съкр. PAN) благодарение на протокола

OpenThread. На Фиг. 1.1. са представени нагледно устройства, използвани в мрежата WattMan.



Фиг. 1.1. Устройства, които се използват в мрежата WattMan

Табл. 1.1. Основни технически характеристики

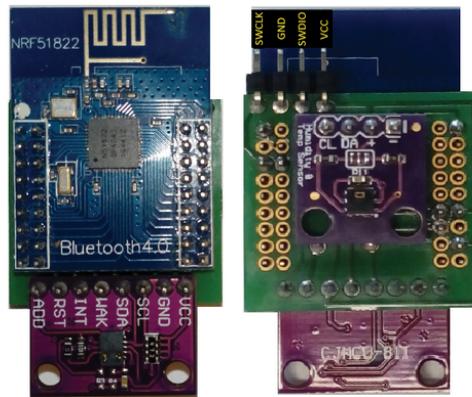
Сензори	Няма информация
Микроконтролер / SoC	nRF52840 SoC (Nordic)
Модули и компоненти за изграждане на мрежата	nRF52840 SoC, Raspberry Pi Compute Module
Протокол за комуникация	OpenThread
Захранване	2 x AAA Batteries (3V, 750mAh)
Употреба	Намаление на разхода за енергия в дома

IAQnet project

IAQnet е малка система, състояща се от едно или повече устройства за наблюдение, базирани на Bluetooth технология [4]. За комуникация между устройствата и потребителя се използва Android приложение, което показва данните за източници на замърсяване, присъстващи в нашия дом.

Основният модул на устройствата е базиран на nRF51822 SoC, който е създаден от фирмата Nordic Semiconductors [5]. Той представлява мултипротоколен SoC за ULP (на англ. *Ultra Low Power*)

безжични приложения. Освен това проектът използва два сензора - CCS811 и HTU21D [6][7]. Първият - CS811, е свръх маломощен цифров сензор с I²C(на англ. *Inter-Integrated Circuit*) интерфейс, който регистрира широка гама от летливи органични съединения (на англ. *Volatile Organic Compounds*, съкр. *VOCs*). Вторият - HTU21D, измерва прецизно температура и относителна влажност и се използва, за да се подобри точността на измерването на нивата на газовете на първия сензор. На Фиг. 1.2. е показано устройството, което комбинира основния модул и двата сензора.



Фиг. 1.2. Прототип на IAQnet устройство

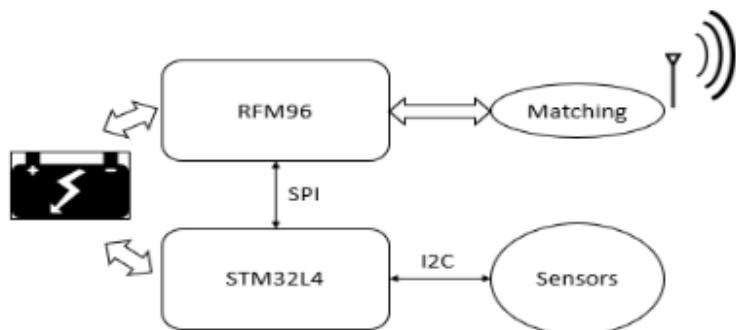
Табл. 1.2. Основни технически характеристики

Сензори	CCS811 и HTU21D
Микроконтролер / SoC	nRF52822 SoC (Nordic)
Модули и компоненти за изграждане на мрежата	nRF52822 SoC
Протокол за комуникация	Bluetooth LE
Захранване	Няма информация
Употреба	Наблюдение на качеството на въздуха в дома

LoRaWAN Wireless Sensor Network for Data Center Temperature Monitoring (DCTMS)

Тази безжична сензорна мрежа е предназначена за наблюдение на температурата в център за данни (*data center*) с цел подобряване на ефективността на охлажддането [8]. За нейната реализация се използва технологията LoRa на фирмата Semtech, която е една от най-обещаващите широкообхватни IoT комуникационни технологии и е в основата на LoraWAN протокола за комуникация.

Сензорният възел е изграден с микроконтролера - STM32L4, създаден от фирмата STMicroelectronics [9]. Той включва в себе си както нискоенергийни, така и високопроизводителни изчислителни ресурси. Отделно за управление на физическия слой на LoRa се използва приемопредавателят RFM96 SoM (на англ. *System on Module*) [10]. На фиг. 1.3. е показана схемата на едно крайно устройство.

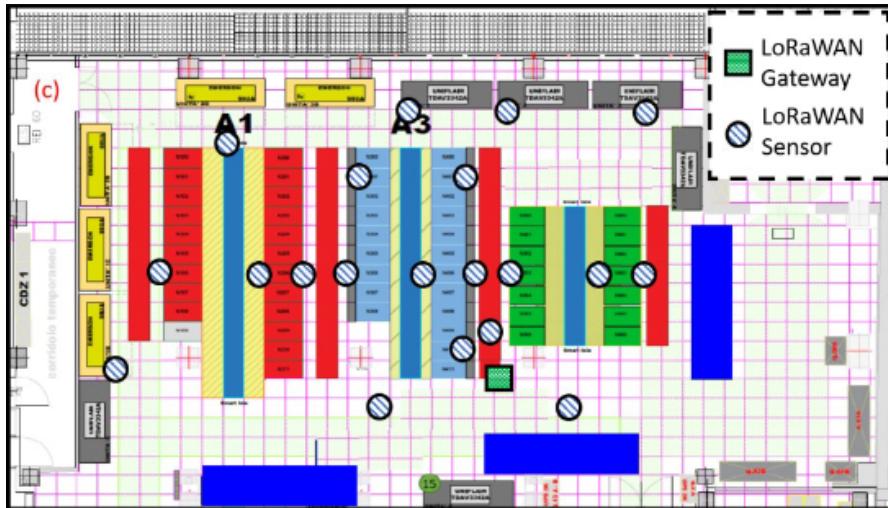


Фиг. 1.3. Схема на сензорния възел

Сензорните възли на тази мрежа са проектирани за лесно внедряване в рамките на съществуващ суперкомпютърен център. Всички експерименти на този проект са проведени в GALILEO (CINECA) - суперкомпютърния център Tier 0 за научни изследвания в Италия.

За LoRaWAN шлюз се използва MultiTech Conduit, който представлява бързо внедрим и лесно мащабируем и програмируем IoT шлюз [11]. Front-end стъпалото на LoRaWAN радиото включва Semtech интегралните схеми SX1301 и две SX1257, които демодулират

пакетите, получени едновременно на всички канали [12][13]. На фиг. 1.4. е представена карта с разположените устройства в центъра за данни.



Фиг. 1.4. Карта на GALILEO (CINECA) data център

Табл. 1.3. Основни технически характеристики

Сензори	За температура и влажност
Микроконтролер / SoC	STM32L4 (STMicro)
Модули / компоненти за изграждане на мрежата	RFM96, MultiConnect Conduit, SX1301, SX1257
Протокол за комуникация	LoRaWAN
Захранване	3.7V, 1000mAh
Употреба	Наблюдение на температурата в data център

An Intelligent and High Efficiency Street Lighting System Isle (IHESLSI)

Реализирана е система, базирана на телекомуникационна безжична мрежа ZigBee и контролна карта Raspberry-Pi, за цялостно дистанционно управление на изолирана високоефективна система за

улично осветление [14]. Системата използва устройства и сензори за управление на единичен уличен стълб и за изпращане на информация чрез ZigBee мрежа към централен стълб, оборудван с Raspberry-Pi контролна карта, която може да събира и обработва информацията.

Устройството позволява отдалечен достъп през Интернет, превръщайки се в мощна цялостна система, която може да проверява състоянието на уличните лампи и да приема подходящи действия в случай на повреда. Стълбовете се захранват с фотоволтаична енергия и представляват отличен образец за устойчиво енергийно развитие.

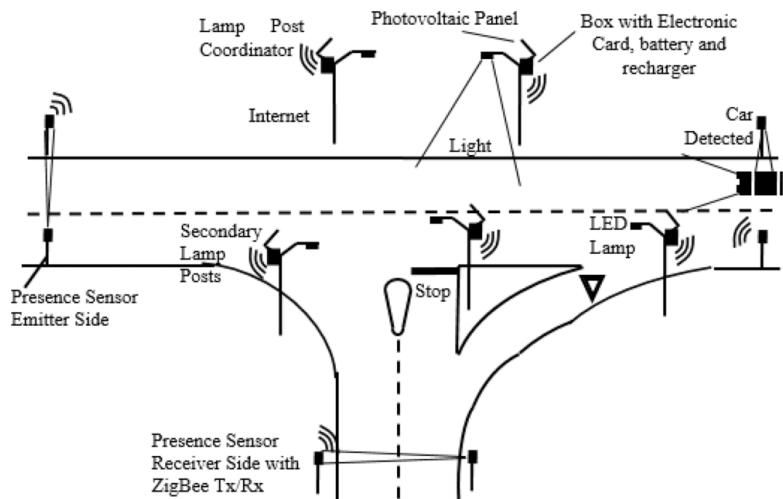
Предложената система е разделена на три основни устройства: станция за наблюдение, която се използва за безжични сензори и лампови стълбове, базова станция за управление на главния лампов стълб и захранващи устройства.

Станцията за наблюдение е базирана на микроконтролер Microchip PIC16F688 [15]. Тя позволява да се събират данни от сензори, да се управляват изпълнителни механизми и да се предава информация с помощта на модул XBee [16].

Базовата станция е екипирана с Raspberry-Pi контролна карта, GPRS(на англ. *General packet radio service*) modem и локарна станция за наблюдения.

Захранващите устройства имат соларни панели с различна големина, защото тяхното напрежението може да се променя в широк диапазон. За неговото лимитиране се използва SM3320-BATT-EV заряден контролер [17].

На фиг. 1.5. е показано типично приложение на предложената система - остров от осветителни стълбове, използвани за осветяването на кръстовище.



Фиг. 1.5. Схема на уличната система

Табл. 1.4. Основни технически характеристики

Сензори	светлинен сензор -TEPT5700, сензор за присъствие -GLV18-6/59/102/159, сензор за ток - ACS756
Микроконтролер / SoC	PIC16F688 (Microchip)
Модули / компоненти за изграждане на мрежата	модул XBee, Raspberry-Pi контролна карта, GPRS модем;
Протокол за комуникация	Zigbee
Захранване	Соларни панели
Употреба	Дистанционно управление на изолирана високоектична система за улично осветление

1.1.3. Сравнение и анализ на разгледаните съществуващи IoT мрежови решения

В таблицата по-долу са сравнени параметрите на разгледаните решения за реализиране на безжични IoT мрежи.

Табл. 1.5. Сравнение на решенията

Решение	WattMan	IAQnet	DCTMS	IHESLSI
Сензори	До 100 различни сензора	Температура, влажност и качество на въздуха	Температура и влажност	светлинен сензор -TEPT5700, сензор за присъствие -GLV18-6/59/102/159, сензор за ток - ACS756
Микроконтролер / SoC	nRF52840 SoC (Nordic)	nRF52822 SoC (Nordic)	STM32L4 (STMicro)	PIC16F688 (Microchip)
Модули / компоненти за изграждане на мрежата	nRF52840 SoC, Raspberry Pi Compute Module	nRF52822 SoC	RFM96, MultiConnect Conduit, SX1301, SX1257	XBee модул, Raspberry-Pi контролна карта, GPRS модем
Протокол за комуникация	OpenThread	Bluetooth LE	LoRaWAN	Zigbee
Захранване	2 x AAA Batteries (3V, 750mAh)	Батерия (няма конкретна информация)	LiIon battery, 3.7V, 1000mAh	Соларни панели
Употреба	Намаление на разхода за енергия в дома	Наблюдение на качеството на въздуха в дома	Наблюдение на температура та в data център	Дистанционно управление на изолирана високоефективна система за улично осветление

От направеното сравнение могат да се направят няколко основни извода:

-Две от решенията използват сензори за температура и влажност, като едно от тях използва и сензор за следене на качеството на въздуха, което допълнително разширява възможностите му за следене на параметрите на околната среда.

-Две от решенията използват микроконтролери на фирмата Nordic Semiconductor, които имат вграден универсален

приемопредавател, поддържащ алтернативно няколко безжични технологии.

-Характерно за всички проекти е, че използват готови модули, които съдържат микроконтролер и приемопредавател за съответната технология.

-Всяко едно решение използва различни популярни в областта безжични технологии.

-И четирите решения са захранвани от различни по вид и напрежение батерии, а едно от тях има и соларни панели. Съответно всяко едно от тях е предназначено за конкретно приложение, за което разработчиците му са преценили, че е най-подходящо.

1.2. Проучване и сравнение на мрежовите протоколи, използвани в съществуващите сензорни IoT мрежи

1.2.1. Проучване на безжични протоколи, използвани в съществуващите сензорни IoT мрежи

С развитието и разпространението на безжичните сензорни мрежи нуждата от нови и оптимизирани протоколи нараства. Производителите непрекъснато търсят технологии, които позволяват все по-лесна и бърза разработка и внедряване в различни сфери от бита [18][19][20].

Мрежовите протоколи се делят на следните видове:

Протоколи за локални и персонални мрежи

Bluetooth Low-Energy



Bluetooth® Low Energy

Bluetooth е една от най-разпространените технологии за комуникация на къси разстояния, която се използва

в индустрията за много персонални продукти (смарт часовници, безжични слушалки, др.).

В днешно време най-новият протокол BLE (на англ. *Bluetooth Low-Energy*) е много популярен в IoT индустрията основно заради неговата ниска консумация на енергия. Най-големият му недостатък обаче остава ограниченията комуникация на по-дълги разстояния [21].

Табл. 1.6. Основни технически характеристики

Обхват	5 - 150 метра
Честотна лента	2400 MHz
Скорост на данните	1 Mbps
Консумация	Ниска
Стандарт	IEEE 802.15.1
Лиценз за спектъра	свободен
Цена	ниска

Wi-Fi



Wi-Fi е един от най-популярните протоколи за комуникация в областта на IoT. Много разработчици избират него заради наличието му в почти всички домове. Съществуващата широка инфраструктура позволява неговото лесно вграждане и предлага бърз трансфер и възможност за обработка на големи количества данни. Най-големият му недостатък е неговата висока консумация, която е неудобна за много IoT устройства [22].

Табл. 1.7. Основни технически характеристики

Обхват	70 -250 метра
Честотна лента	2.4 / 5 GHz
Скорост на данните	100 - 600 Mbps

Консумация	Средна
Стандарт	IEEE 802.11a/b/g/n/ac/ax
Лиценз за спектъра	свободен
Цена	ниска

Протоколи за многоточкови (mesh) мрежи

ZigBee

ZigBee е един от важните протоколи в IoT сферата.



ZigBee® Той има някои значителни предимства в сложните системи, като предлага работа с ниска консумация на енергия, висока сигурност, устойчивост и висока степен на мащабируемост, както и възможност да се възползва от предимствата на безжичното управление. Едни от по-големите му недостатъци са: изискването на задълбочени знания за управлението му, ограничената оперативна съвместимост и ниската му скорост [22].

Табл. 1.8. Основни технически характеристики

Обхват	10 - 100 метра
Честотна лента	868/915/2400 MHz
Скорост на данните	20/40/250 Kbps
Консумация	Ниска
Стандарт	IEEE 802.15.4
Лиценз за спектъра	свободен
Цена	средна

6LoWPAN



6LoWPAN (на англ. *IPv6 over Low-Power Wireless Personal Area Networks*) е безжична мрежа с ниска мощност, в която всеки възел има собствен IPv6 адрес. Това позволява на възлите да се свързват директно с Интернет, като използват отворени стандарти. 6LoWPAN е mesh мрежа, която е надеждна, мащабируема и самолекуваща се. Нейните недостатъци са, че има по-малък имунитет към смущения в сравнение с други технологии и има малък обхват при липса на mesh топология [23].

Табл. 1.9. Основни технически характеристики

Обхват	10 -100 метра
Честотна лента	2.4 GHz
Скорост на данните	200 Kbps
Консумация	Ниска
Стандарт	IEEE 802.15.4
Лиценз за спектъра	свободен
Цена	ниска

Thread



Thread е технология за мрежова комуникация, базирана на IPv6 (използва 6LoWPAN в мрежовия слой за осигуряване на уникални IPv6 адреси за всяко устройство), с ниска консумация на енергия за продукти от IoT областта. Thread е надежден и сигуризиран протокол, осигуряващ бързо време за реакция, разширено покритие и дългогодишен живот на батерията.

Thread не се нуждае от централен шлюз за свързване към IP, защото мрежата му може да се самовъзстановява и автоматично да се преконфигурира. По този начин той улеснява свързването на всяко

устройство с мрежова връзка и елиминира необходимостта от собствени шлюзове или транслатори, което от своя страна намалява инвестициите и сложността на мрежовата инфраструктура. Недостатъците на Thread са ниските скорости, ограничението му до 250 свързани устройства в една мрежа и липсата на поддръжка на изключително дълъг обхват [24][25].

Табл. 1.10. Основни технически характеристики

Обхват	10 - 100 метра
Честотна лента	2.4 GHz
Скорост на данните	250 Kbps
Консумация	Ниска
Стандарт	IEEE 802.15.4
Лиценз за спектъра	свободен
Цена	ниска

Z-Wave



Z-Wave е технология за радиочестотна комуникация с ниска мощност, която е предназначена предимно за домашна автоматизация за продукти като контролери за лампи и сензори, както и за много други устройства. Z-Wave е опростен протокол, позволяващ бърза и лесна разработка. Един от големите му недостатъци е, че технологията изисква патент за употреба [22].

Табл. 1.11. Основни технически характеристики

Обхват	30 - 50 метра
Честотна лента	900 MHz
Скорост на данните	10-100 Kbps

Консумация	Ниска
Стандарт	Z-Wave Alliance
Лиценз за спектъра	свободен
Цена	средна

Протоколи за глобални мрежи с ниска мощност

LoRaWAN



LoRaWAN е протокол за широкообхватна мрежа с ниска консумация на енергия, оптимизиран за ниска енергийна консумация и поддържане на големи мрежи с милиони устройства. Насочен към приложения за широкообхватната мрежа (на англ. *Wide Area Network*, съкр. WAN), LoRaWAN е проектиран така, че да снабдява WAN мрежите с функции, необходими за поддържане на евтина, мобилна и сигурна двупосочна комуникация в областта на IoT, M2M (на англ. *Machine-to-Machine*), умните градове и промишлената индустрия. Недостатъците на технологията са нейната ниска скорост и скъпата реализация на технологията понеже, в сравнение с други безжични модули, LoRa модулите са сравнително по-скъпи [26].

Табл. 1.12. Основни технически характеристики

Обхват	2 - 20 километра
Честотна лента	865 - 925 MHz
Скорост на данните	10-50 Kbps
Консумация	Ниска
Стандарт	LoraWan
Лиценз за спектъра	свободен
Цена	средна

Sigfox



Концепцията на Sigfox е да осигури ефективно решение за свързване на маломощни M2M приложения, изискващи ниски нива на трансфер на данни, за които обхватът на Wi-Fi е твърде малък, а обхватът на клетъчните мрежи е твърде скъп и изиска твърде много енергия. Недостатъците на технологията са изключително ниската му скорост, единичната точка на повреда (single point of failure), еднопосочната му комуникация и др [26].

Табл. 1.13. Основни технически характеристики

Обхват	3 - 50 километра
Честотна лента	800 - 900 MHz
Скорост на данните	100 bps
Консумация	Ниска
Стандарт	Sigfox
Лиценз за спектъра	свободен
Цена	средна

Протоколи за клетъчни мрежи

LTE-M



LTE-Cat M1 е стандарт за свързване с ниска енергийна консумация, който свързва IoT и M2M устройства, изискващи средна скорост на предаване на данни. Той поддържа по-дълъг жизнен цикъл на батерията и предлага по-голям обхват в сградата в сравнение с клетъчни технологии като 2G, 3G и LTE-Cat 1.

Тъй като е съвместим със съществуващата LTE мрежа, CAT M1 не изиска от операторите да изграждат нова инфраструктура, за да го внедрят. В сравнение с NB-IoT (на англ. *Narrowband IoT*) LTE Cat M1 се оказва идеална в случаи на мобилна употреба, заради по-доброто си предаване на данни между клетъчните участъци и приликата си с високоскоростната LTE. Минусите на тази технология са изискването на лиценз за използването ѝ и високата ѝ цена в сравнение с други технологии [26].

Табл. 1.14. Основни технически характеристики

Обхват	1 - 11 километра
Честотна лента	1.4 MHz
Скорост на данните	1 Mbps
Консумация	Средна
Стандарт	3GPP
Лиценз за спектъра	изисква
Цена	висока

NB-IoT



Продукт на съществуващите технологии на 3GPP, Narrowband IoT е изцяло нов стандарт за радиотехнология, който осигурява изключително ниска консумация на енергия (10 години работа на батерия) и свързаност със сила на сигнала с около 23dB по-ниска, отколкото при 2G. Нещо повече - той използва съществуващата мрежова инфраструктура, което осигурява не само глобално покритие в LTE мрежите, но и гарантирано качество на сигнала. В много случаи този факт позволява прилагането на NB-IoT вместо решения, които

изискват изграждането на локални мрежи като LoRa или Sigfox. Технологията отново изисква лиценз за спектъра и е скъпа [26].

Табл. 1.15. Основни технически характеристики

Обхват	1 - 15 километра
Честотна лента	180 kHz
Скорост на данните	250 Kbps
Консумация	Средна
Стандарт	3GPP
Лиценз за спектъра	изисква
Цена	висока

1.2.2. Сравнение и анализ на разгледаните мрежови протоколи, използвани в съществуващите сензорни IoT мрежи

В таблиците по-долу са сравнени параметрите на разгледаните досега мрежови протоколи.

Табл. 1.16. Сравнение на протоколите за локален достъп

Протокол	BLE	Wi-Fi	ZigBee	6LoWPAN	Thread	Z-Wave
Обхват	5 - 150 метра	70 -250 метра	10 - 100 метра	10 -100 метра	10 - 100 метра	30 - 50 метра
Честотна лента	2400 MHz	2.4 / 5 GHz	868/915/ 2400 MHz	2.4 GHz	2.4 GHz	900 MHz
Скорост на данните	1 Mbps	100 - 600 Mbps	20/40/25 0 Kbps	200 Kbps	250 Kbps	10-100 Kbps
Консумация	Ниска	Средна	Ниска	Ниска	Ниска	Ниска
Стандарт	IEEE 802.15 .1	IEEE 802.11a/b /g/n/ac/a x	IEEE 802.15.4	IEEE 802.15.4	IEEE 802.15.4	Z-Wave Alliance
Лиценз за спектъра	свободен	свободен	свободен	свободен	свободен	свободен

Цена	ниска	ниска	средна	ниска	ниска	средна
-------------	-------	-------	--------	-------	-------	--------

Табл. 1.17. Сравнение на протоколите за отдалечен достъп

Протокол	LoRaWAN	Sigfox	LTE-M	NB-IoT
Обхват	2 - 20 километра	3 - 50 километра	1 - 11 километра	1 - 15 километра
Честотна лента	865 - 925 MHz	800 - 900 Mhz	1.4 MHz	180 KHz
Скорост на данните	10-50 Kbps	100 bps	1 Mbps	250 Kbps
Консумация	Ниска	Ниска	Средна	Средна
Стандарт	LoraWan	Sigfox	3GPP	3GPP
Лиценз за спектъра	свободен	свободен	изисква	изисква
Цена	средна	средна	висока	висока

От направеното сравнение могат да се направят следните изводи:

- Забелязва се, че повече от половината решения основно биват използвани на разстояния по-малки от 300 метра, което е достатъчно за средна по големина безжична сензорна мрежа.
- Половината протоколи работят с най-широко разпространената честота - 2.4 Ghz, което е причина за голям брой смущения.
- Скоростта на данните при повечето протоколи е килобитове в секунда, която е достатъчна за предаване на сензорни данни.
- Преобладаващата консумация на енергия е ниска и позволява използването на батерии за захранване и съответно дава по-голяма мобилност на мрежата.
- Стандартите на протоколите биват най-различни, като почти всички дават възможността за тяхното използване без необходимост от заплащане.

- Спектърът, в който работят протоколите, е нелицензиран и най-често от типа ISM (на англ. *Industrial, scientific and medical*) т.е. предназначен за индустриални, научни и медицински цели.
- Цената на лиценза за използването на конкретен протокол варира в зависимост от производителя и реализацията.

1.2.3. Проучване на протоколи за предаване на данни, използвани в съществуващите сензорни IoT мрежи

В една IoT сензорна мрежа е необходим, освен протокол за мрежова свързаност, и някакъв начин за предаване на данните. По-долу са описани някои от най-често използваните протоколи за предаване на данни в подобни мрежи.

CoAP

Протоколът CoAP (на англ. *Constrained Application Protocol*) е създаден за малки устройства с ограничени възможности, които работят в мрежи с ниска консумация на енергия и с вероятност за непостоянна (*transient*) или прекъсваща (*lossy*) комуникация. CoAP е RESTful (на англ. *Representational state transfer*) протокол и в много отношения наподобява HTTP (на англ. *Hypertext Transfer Protocol*). При него съобщенията се изпращат и получават чрез UDP (на англ. *User Datagram Protocol*) протокола. CoAP позволява интегрирането на DTLS (на англ. *Datagram Transport Layer Security*) за подобряване на сигурността [27][28].

Табл. 1.18. Основни технически характеристики

Основен вид комуникационен модел	Request - response
Режим на изпращаните съобщения	Асинхронни и синхронни
Транспортен протокол	UDP

Механизъм за надеждност	съобщения за потвърждение, за грешка и за повторно предаване
--------------------------------	--

MQTT

MQTT (на англ. *Message Queue Telemetry Transport*) е олекотен протокол за обмен на съобщения, използващ publish-subscribe модела. Той е подходящ за свързване на малки устройства с ниска консумация на енергия и ограничена памет и изчислителна мощ. MQTT е проектиран с оглед на надеждността и мащабируемостта. Сигурността му се осигурява чрез TLS (англ. *Transport Layer Security*). Неговите постоянни сесии са голямо предимство поради причината, че позволяват на протокола да се адаптира към неблагоприятни мрежови условия и да намалява времето за изграждане на връзката [28][29].

Табл. 1.19. Основни технически характеристики

Основен вид комуникационен модел	Publish - subscribe
Режим на изпращаните съобщения	Асинхронни и синхронни
Транспортен протокол	TCP
Механизъм за надеждност	3 нива на QoS (на англ. <i>Quality of Service</i>) - 0,1,2

MQTT-SN

MQTT-SN (на англ. *MQTT for Sensor networks*) е протокол, базиран на publish-subscribe модела, за изпращане на съобщения в безжични сензорни мрежи. Направен е с цел разширяване на употребата на MQTT отвъд обхвата на TCP/IP инфраструктурата за решения използвани сензори и актуатори. Той има кратка дължина на съобщението, използва като основен протокол UDP и може да работи в среди с непостоянна връзка. MQTT-SN също така разполага с

процедура за поддържане на връзка, която позволява на устройствата да заспиват, когато не са необходими, и да получават всяка информация, която ги очаква, когато се събудят [30][31].

Табл. 1.20. Основни технически характеристики

Основен вид комуникационен модел	Publish - subscribe
Режим на изпращаните съобщения	Асинхронни и синхронни
Транспортен протокол	UDP
Механизъм за надеждност	4 нива на QoS - -1,0,1,2,3

HTTP

HTTP е stateless комуникационен протокол, работещ върху TCP, който се използва за client-server комуникация. Той е еднопосочен протокол, при който клиентът изпраща заявката, а сървърът - отговора. При него всяка заявка установява нова връзка със сървъра, която се прекратява сама след получаване на отговор. HTTP използва three-way handshaking методи и препредава изгубени пакети. Той се използва предимно за уеб страници [32].

Табл. 1.21. Основни технически характеристики

Основен вид комуникационен модел	Request - response
Режим на изпращаните съобщения	Синхронни
Транспортен протокол	TCP
Механизъм за надеждност	Three-way handshake

WebSocket

WebSocket е двупосочен, full-duplex протокол, използван за client-server комуникация. Той е stateful протокол, работещ върху TCP. WebSocket е създаден с цел да осигури механизъм за браузър-базирани приложения, които имат нужда от постоянна двупосочна комуникация със сървър без необходимостта от отварянето на множество HTTP връзки. Той се явява като добра опция за приложения, чиито данни се обновяват постоянно или са в реално време[33].

Табл. 1.22. Основни технически характеристики

Основен вид комуникационен модел	Exclusive pair
Режим на изпращаните съобщения	Асинхронни
Транспортен протокол	TCP
Механизъм за надеждност	Three-way handshake

1.2.4. Сравнение и анализ на разгледаните протоколи за предаване на данни, използвани в съществуващите сензорни IoT мрежи

В таблиците по-долу са сравнени параметрите на разгледаните протоколи за предаване на данни.

Табл. 1.23. Сравнение на протоколите за предаване на данни

Протокол	CoAP	MQTT	MQTT-SN	HTTP	WebSocket
Основен вид комуникационен модел	Request - response	Publish - subscribe	Publish - subscribe	Request - response	Exclusive pair
Режим на изпращаните	Асинхронни и синхронни	Асинхронни и	Асинхронни и	Синхронни	Асинхронни

съобщения		синхронни	синхронни		
Транспортен протокол	UDP	TCP	UDP	TCP	TCP
Механизъм за надеждност	съобщения за потвърждение, за грешка и за повторно предаване	3 нива на QoS - 0,1,2	4 нива на QoS - -1,0,1,2	Three-way handshake	Three-way handshake

От направеното сравнение могат да се направят следните изводи:

- Използват се три вида комуникационни модели: request-response, publish-subscribe и exclusive pair.
- Повечето протоколи поддържат асинхронен и синхронен режим на изпращаните съобщения.
- Използват се два транспортни протокола - UDP и TCP, като TCP е по-често срещан.
- Всички имат вграден механизъм за надеждност, като MQTT-SN предлага най-много опции.

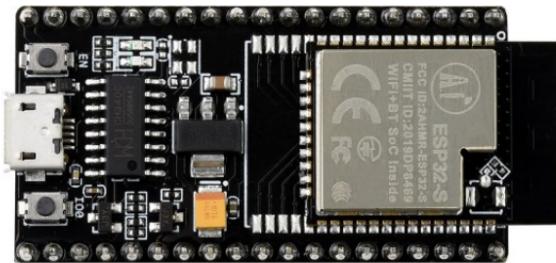
1.3. Проучване, сравнение и анализ на основните компоненти, използвани в съществуващите сензорни IoT мрежи

1.3.1. Проучване, сравнение и анализ на микроконтролери и едноплаткови компютри, използвани в съществуващите сензорни IoT мрежи

NodeMCU-32S

NodeMCU-32S е платка за разработка, базирана на микроконтролера ESP32S. Тя е разработена от компанията Espressif Systems. Платката комбинира в себе си вграден приемопредавател,

двуядрен (dual-core) процесор с висока честота и голяма flash памет [34]. Самият модул е показан на фиг. 1.6.



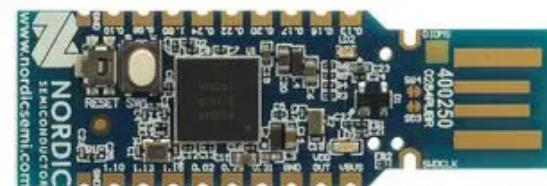
Фиг. 1.6. NodeMCU-32S

Табл. 1.24. Технически характеристики

Процесор	Dual-core Tensilica LX6 (240 MHz)
Памет	520KB RAM / 4MB Flash
Вграден приемопредавател	Има (Wi-Fi, Bluetooth)
Захранване	3.3 - 5 V
Цена	25.50lv. (13.03€)

nRF52840 Dongle

nRF52840 Dongle представлява малък, нискобюджетен USB модул с вграден приемопредавател. Той позволява лесна и бърза разработка благодарение на предоставения от Nordic nRF Connect SDK (на англ. *Software Development kit*) [35]. На фиг. 1.7. е показан външният му вид.



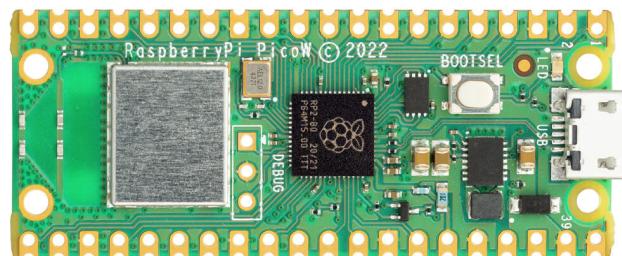
Фиг. 1.7. nRF52840 Dongle

Табл. 1.25. Технически характеристики

Процесор	Arm 32-bit Cortex-M4 (64 MHz)
Памет	256KB RAM / 1MB Flash
Вграден приемопредавател	Има (BLE, NFC, Thread, Zigbee, 802.15.4)
Захранване	1.7 - 5.5 V
Цена	36.88lv. (18.89€)

Raspberry Pi Pico W

Raspberry Pi Pico W е високопроизводителна, нискобюджетна микроконтролерна платформа, базирана на SoC RP2040, която има вграден приемопредавател, позволяващ лесна и бърза разработка на проекти с безжична комуникация [36]. Платформата е показана на фиг. 1.8.



Фиг. 1.8. Raspberry Pi Pico W

Табл. 1.26. Технически характеристики

Процесор	Dual ARM Cortex-M0+ (133MHz)
Памет	264KB RAM / 2MB Flash
Вграден приемопредавател	Има (Wi-Fi, Bluetooth)
Захранване	1.8- 5.5 V
Цена	11.91lv. (6.11€)

Сравнение

Табл. 1.27. Сравнение на прототипните платки

Прототипна платка	NodeMCU-32S	nRF52840 Dongle	Raspberry Pi Pico W
Процесор	Single or Dual-Core 32-bit LX6 Microprocessor (240 MHz)	Arm 32-bit Cortex-M4 (64 MHz)	Dual ARM Cortex-M0+ (133MHz)
Памет	520KB RAM / 4MB Flash	256KB RAM / 1MB Flash	264KB RAM / 2MB Flash
Вграден приемопредавател	Има (Wi-Fi, Bluetooth)	Има (BLE, NFC, Thread, Zigbee, 802.15.4)	Има (Wi-Fi, Bluetooth)
Захранване	3.3 - 5 V	1.7 - 5.5 V	1.8- 5.5 V
Цена	24.00лв. (12.25€)	20.38лв. (10.40€)	18.00лв. (9.18€)

От направеното сравнение се забелязва, че:

- И трите платки за разработка имат сравнително голямо място, давайки възможност за разработването на по-обемни програми без особено притеснение за запълване на мястото.
- Всички имат вграден приемопредавател, което улеснява разработката на проекти с безжична комуникация без нуждата за търсене на външен такъв.
- Всички модули работят на почти еднакво захранващо напрежение. То е ниско и позволява лесно захранване с батерии.
- И трите са сравнително малки на големина и поради тази причина могат да се интегрират без ограничение в различни среди.
- Всички са на достъпна цена.

1.3.2. Проучване, сравнение и анализ на комуникационни схеми използвани в съществуващите сензорни IoT мрежи

Комуникационни схеми, поддържащи Zigbee, Thread и Bluetooth

Nordic's SoC nRF52840

nRF52840 е 2,4 GHz безжичен SoC със свръхниска консумация на енергия, интегрираща многопротоколен 2,4 GHz приемопредавател, процесор Arm Cortex-M4F и флаш памет за програмиране [3]. Интегралната схема е показана на фиг. 1.9.



Фиг. 1.9. Корпусът на nRF52840

Табл.1.28. Технически характеристики

Честотна лента	2360 - 2500 MHz
Поддържани стандарти	BLE, NFC, Thread, Zigbee, 802.15.4
Tx Power	-20 до +8dBm
Current Tx/Rx Mode	4.8mA (0dBm)/ 4.6mA (3V)
Захранване	1.7 - 5.5V

MRF24J40MA by Microchip

MRF24J40MA е сертифициран 2,4 GHz IEEE 802.15.4 радио приемопредавателен модул. Той обединява PHY (на англ. *Physical layer*) и MAC (на англ. *Media Access Control*) функционалността в едно

чипово решение [37]. Приемопредавателят е представен нагледно на фиг. 1.10.



Фиг. 1.10. Корпусът на MRF24J40MA

Табл. 1.29. Технически характеристики

Честотна лента	2405 - 2480 MHz
Поддържани стандарти	802.15.4, ZigBee, MiWi, MiWi P2P
Tx Power	0 до +36dBm
Current Tx/Rx Mode	23mA (0dBm) / 19mA (3.3V)
Захранване	2.4 - 3.6V

Комуникационни схеми, поддържащи LoRa и Sigfox

Semtech's SX1276

Приемопредавателят SX1276 разполага с LoRa модем с голям обхват и висока устойчивост на смущения, както и минимизирана консумацията на ток [38]. Приемопредавателят е представен нагледно на фиг. 1.11.



Фиг. 1.11. Корпусът на SX1276

Табл. 1.30. Технически характеристики

Честотна лента	137 - 1020 MHz
Поддържани стандарти	LoRa
Max Tx Power	-4 до +20dBm
Current Tx/Rx Mode	29mA (+13dBm) / 12mA (3.3V)
Захранване	1.8 - 3.7V

Комуникационни схеми, поддържащи клетъчна свързаност - под NDA (на англ. *Non-disclosure agreement*)

Сравнение

Табл. 1.31. Сравнение на комуникационните схеми

Комуникационен модул	nRF52840	MRF24J40MA	SX1276
Честотна лента	2360 - 2500 MHz	2405 - 2480 MHz	137 - 1020 MHz
Поддържани стандарти	BLE, NFC, Thread, Zigbee, 802.15.4	802.15.4, ZigBee, MiWi, MiWi P2P	LoRa
Max Tx Power	-20 до +8dBm	0 до +36dBm	-4 до +20dBm
Current Tx/Rx Mode	4.8mA (0dBm)/ 4.6mA (3V)	23mA (0dBm) / 19mA (3.3V)	29mA (+13dBm) / 12mA (3.3V)
Захранване	1.7 - 5.5V	2.4 - 3.6V	1.8 - 3.7V

От направеното сравнение в горната таблица се забелязва следното:

-Приемопредавателите са предназначени за работа на определена честотна лента и поддържат повечето протоколи на нея.

-Всички имат сравнително голямо усилване на силата на предаване поради вградената антена, отстранявайки необходимостта от външна такава.

-Консумацията за предаване и приемане варира между приемопредавателите, но се забелязва, че с увеличаване на мощността за предаване, тя нараства драстично.

-Всички приемопредаватели могат да бъдат захранвани от батерии заради тяхното ниско захранващо напрежение.

1.3.3. Проучване на сензори и актуатори, използвани в съществуващите сензорни IoT мрежи

Сензор за температура - LM35DZ

LM35DZ е нискобюджетен температурен сензор, разработен от National Semiconductor - понастоящем част от фирмата Texas Instruments. Негово предимство е, че не се нуждае от външна калибрация и работи директно в градуси по Целзий [39]. Сензорът е показан нагледно на фиг. 1.12.



Фиг. 1.12. Сензор за температура - LM35DZ

Табл. 1.32. Технически характеристики

Обхват на измерване	–55°C до 150°C
Точност	±0.25°C
Захранване	4 - 30 V
Интерфейс	Аналогов

Сензор за влажност - HIH-5030/5031

Сензорите за влажност от серията HIH-5030/5031, изработени от Honeywell, са специално проектирани за OEM (на англ. *Original Equipment Manufacturer*) компании. Те позволяват директно свързване с контролер или друго устройство с помощта на почти линейното си изходно напрежение [40]. Изглед към сензора е показан на фиг. 1.13.



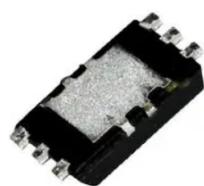
Фиг. 1.13. Сензор за влажност - HIH-5030/5031

Табл. 1.33. Технически характеристики

Диапазон на измерване	0 ~ 100% RH
Точност	±3% RH
Захранване	2.7 - 5.5 V
Интерфейс	Аналогов

Сензор за осветеност - BH1750

BH1750 е 16-битов сензор за околната светлина, направен от Rohm Semiconductors. Той е малък, мощен и нискобюджетен сензор. Използва се за откриване на промяна в интензитета на светлината и осигурява измерванията в lux [41]. Сензорът е показан нагледно на фиг. 1.14.



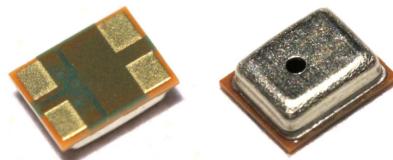
Фиг. 1.14. Сензор за осветеност - BH1750

Табл. 1.34. Технически характеристики

Диапазон на измерване	0.11-100000 lx
Точност	$\pm 20\%$
Захранване	2.4 - 3.6 V
Интерфейс	I ² C

Сензор за звук - SPW2430

SPW2430HR5H-B е миниатюрен микрофон с висока производителност и ниска консумация на енергия, който се състои от акустичен сензор, нискошумов входен буфер и изходен усилвател. Той използва доказаната високопроизводителна технология SiSonic MEMS на компанията Knowles, за да може прецизно да зачита околните звуци. Този микрофон е перфектен за всички проекти, които се нуждаят от отлична аудио производителност и RF (на англ. *Radio frequency*) устойчивост [42]. Сензорът е показан нагледно на фиг. 1.15.



Фиг. 1.15. Сензор за звук - SPW2430

Табл. 1.35. Технически характеристики

Честотен обхват	100Hz - 10KHz
------------------------	---------------

Сензитивност	от -45 до -39 dBV/Pa
Захранване	1.5 - 3.6 V
Интерфейс	Аналогов

Сензор за чистота на въздуха - CCS811

CCS811 е цифров сензор за газ с ниска консумация на енергия, който обединява сензор за газ за откриване на ниски нива на ЛОС, микроконтролер (на англ. *Microcontroller unit*, съкр. MCU) и аналогово-цифров преобразувател (на англ. *Analog-to-digital converter*, съкр. ADC) за наблюдение на местната среда и осигуряване на индикация за качеството на въздуха в помещениета чрез еквивалентен изход за CO₂ или TVOC (на англ. *Total volatile organic compounds*) през стандартен цифров интерфейс I²C [6]. Сензорът е показан нагледно на фиг. 1.16.



Фиг. 1.16. Сензор за чистота на въздуха - CCS811

Табл. 1.36. Технически характеристики

CO₂ Диапазон	400 - 8192 ppm
TVOC Диапазон	0 - 1187 ppb
Захранване	1.8 - 3.3 V
Интерфейс	I ² C

Акселерометър - LIS3DH

LIS3DH, направен от компанията StMicroelectronics, е високопроизводителен акселерометър с три оси и със свръхниска консумация на енергия. Той има вграден FIFO (на англ. *First In First Out*) буфер с 32 нива и може да засича движение, падане и ориентация в 6 или 4 посоки [43]. Сензорът е показан нагледно на фиг. 1.17.



Фиг. 1.17 Акселерометър - LIS3DH

Табл. 1.37. Технически характеристики

Диапазон на ускорението	$\pm 2g$, $4g$, $8g$, $16g$
Чувствителност	$1000 (\pm 2g) \sim 83 (\pm 16g)$
Захранване	1.7 - 3.6 V
Интерфейс	I ² C/SPI (на англ. <i>Serial peripheral interface</i>)

Комбинирани сензори - BME280/BMP280

BME280 е сензор за абсолютно барометрично налягане, температура и влажност, произвеждан от компанията Bosch. Малките му размери и ниската му консумация на енергия позволяват неготово внедряване в устройства, захранвани от батерии, като мобилни телефони, GPS (на англ. *Global Positioning System*) модули и часовници. Той е оптимизиран по отношение на консумацията на енергия и разделителната способност [44]. Сензорът BMP280 има същите параметри, като единствената му разлика е, че не може да измерва влажност [45]. Сензорът е показан нагледно на фиг. 1.18.



Фиг. 1.18. Комбинирания сензор - BME280

Табл. 1.38. Технически характеристики

Измерва	Налягане, Температура, Влажност
Диапазон	300-1100hPa; -40-85°C; 0-100% RH
Точност	±1.0hPa; ±0.5°C; ±3% RH
Захранване	1.7 - 3.6V
Интерфейс	I ² C/SPI

Комбинирани сензори - BME688/680

BME688 е първият газов сензор с AI и интегрирани сензори за налягане, влажност и температура с висока линейност и точност. Той е с малък размер и ниска консумация, позволявайки гъвкавото му интегриране в различни среди. Газовият сензор може да открива летливи органични съединения, летливи серни съединения и други газове като въглероден оксид и водород в диапазона части на милиард [46][47]. Сензорът е показан нагледно на фиг. 1.19.



Фиг. 1.19. Комбинирания сензор - BME688

Табл. 1.39. Технически характеристики

Измерва	Налягане, Температура, Влажност, Газове
Диапазон	300-1100hPa; -40-85°C; 0-100% RH
Основни сензорни изходи на газ сензора	Индекс за качество на въздуха в затворени помещения (на англ. <i>Indoor Air Quality</i> , съкр. IAQ), bVOC- и CO ₂ -еквиваленти (ppm), резултат от сканиране на газ (%) и ниво на интензивност
Скорост на измерването на газ сензора	10.8 секунди/сканиране
Точност	±0.6hPa; ±0.5°C; ±3% RH
Захранване	1.7 - 3.6V
Интерфейс	I ² C/SPI

Дисплей - SSD1306

SSD1306 представлява графичен OLED (на англ. *Organic light-emitting diode*) дисплей с I²C интерфейс. Той е малък с резолюция 128x64 пиксела и размер по диагонал 0.96 инча. Дисплеят има чисто изражение и отлична видимост при ограничено осветление. Той има 8 реда за визуализиране на информация и се явява като добра опция за компактно изобразяване на важни параметри като например час, ниво на батерия, данни от сензори, имена и др [48]. На фиг. 1.20. е показан изглед към съответния дисплей.



Фиг. 1.20. SSD1306 дисплей модул

Табл. 1.40. Технически характеристики

Размери	0.96 инча
Резолюция	128x64
Технология	OLED
Захранване	3.3 - 5V
Интерфейс	I ² C

Втора глава - Основни изисквания и блокови схеми

2.1. Основни изисквания към проектираната сензорна мрежа

По-долу ще бъдат описани изискванията към проектираната система.

2.1.1. Архитектурни и комуникационни изисквания

- Архитектурата на проектираната мрежа трябва да включва два вида възли (nodes) - основни възли, състоящи се от крайни (child) или маршрутизиращи (router) възли, и водещи/управляващи (leader) възли;
- За основен комуникационен протокол в проектираната мрежа трябва да се използва Thread или негова модификация/вариант.

2.1.2. Функционални изисквания

- Всеки от основните възли трябва да следи параметрите на сензорите, които са включени в него, и да предава информацията за стойността им към водещия възел пряко или чрез маршрутизиращ възел;
- Водещият възел трябва да приема, съхранява и обобщава получените от основните възли данни, както и да предоставя възможност за отдалечен достъп до тези данни през Интернет;

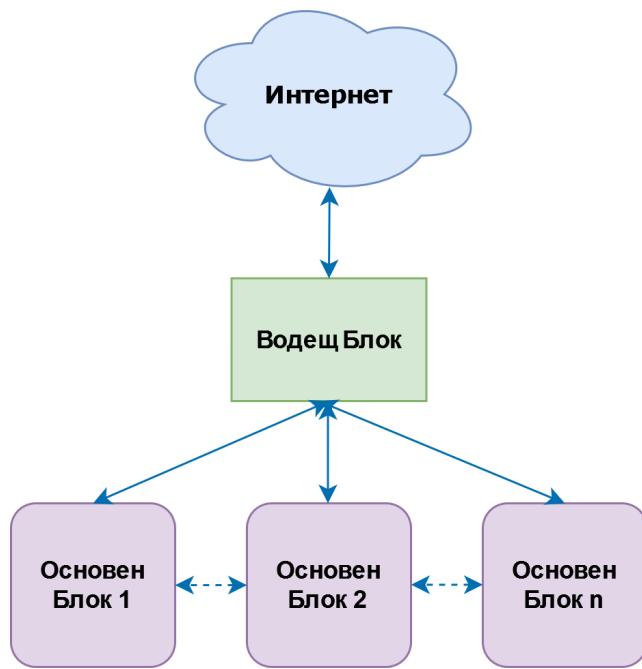
2.1.3. Конструктивни изисквания

- Основните възли на мрежата (крайни и маршрутизиращи) трябва да са оформени като самостоятелни модули/платки, включващи в себе си, в зависимост от вида на модула, съответните сензорни модули, управляващо устройство, приемопредавател и захранване;
- Водещият/управляващият възел трябва допълнително да съдържа модул, осигуряващ връзката с Интернет;

- Задължителен елемент на всеки възел е подходящ захранващ модул.

2.2. Обща блокова схема и функции на блоковете

На фиг. 2.1. е показана обща блокова схема на проектираната сензорна мрежа.



Фиг. 2.1. Обща блокова схема

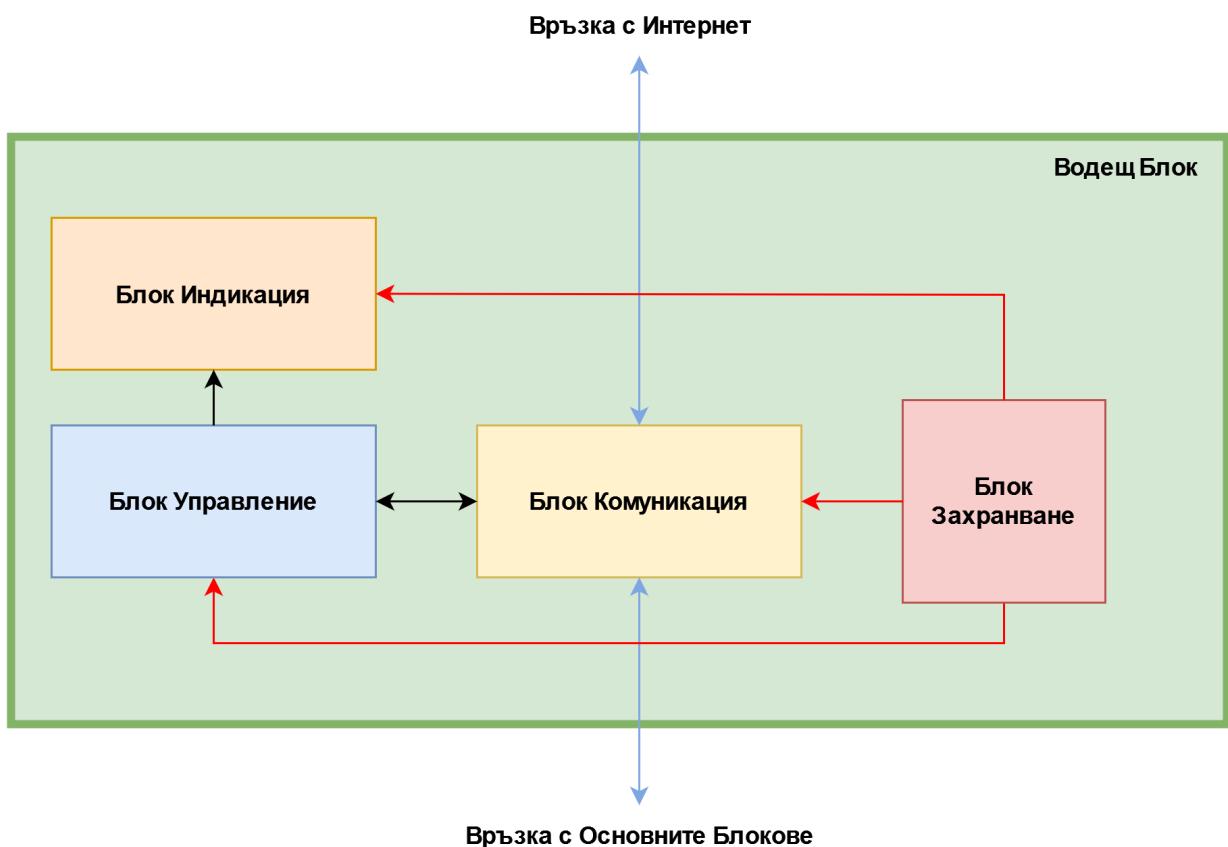
2.2.1. Функции на отделните блокове

Водещ блок - Неговото предназначение е да осигурява отдалечен достъп до устройствата в мрежата, да съхранява и обобщава данните, получени от тях, и да служи като гранично устройство (border router), позволяващо комуникацията между вътрешната мрежа и Интернет.

Основен блок - Основният блок може да функционира като краен или маршрутизиращ възел. Неговата цел е да събира информация от свързаните към него сензори и да я препраща към Водещия блок.

2.3. Схеми и описание на отделните блокове

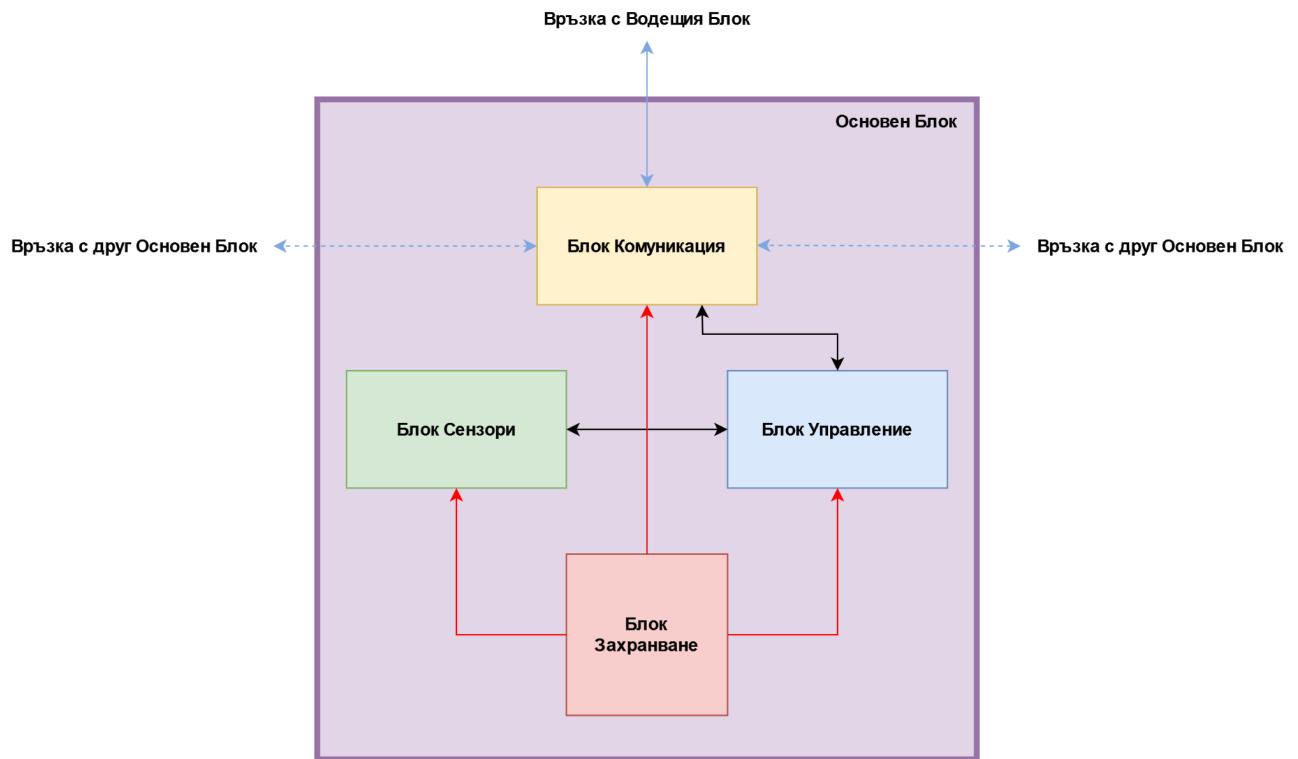
Водещ блок



Фиг. 2.2. Вътрешна блокова схема на Водещия Блок

На фиг. 2.2. е представена вътрешната блокова схема на Водещия Блок. Тя се състои от четири части - Блок Индикация, Блок Управление, Блок Комуникация и Блок Захранване, и има безжична двупосочна комуникация с Интернет и Основните Блокове.

Основен блок



Фиг. 2.3. Вътрешна блокова схема на Основния Блок

На фиг. 2.3. е представена вътрешната блокова схема на Основния Блок. Тя се състои от четири части - Блок Комуникация, Блок Сензори, Блок Управление и Блок Захранване. Освен това има постоянна двупосочна връзка с Водещия Блок и, в зависимост от топологията и функционалността на мрежата - с другите Основни Блокове.

2.4 Схеми и описание на отделните подблокове

Подблокове на Водещия Блок

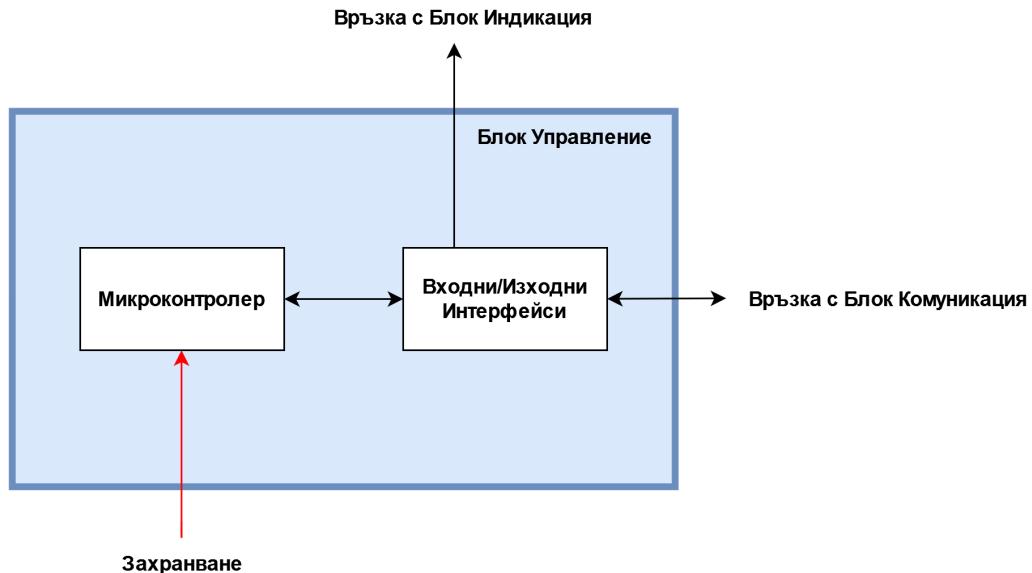
- Блок Индикация



Фиг. 2.4. Вътрешна блокова схема на Блок Индикация

На фиг. 2.4. е представена вътрешната блокова схема на Блок Индикация. Той се състои само от дисплей, който визуализира данни, получени от Блок Управление.

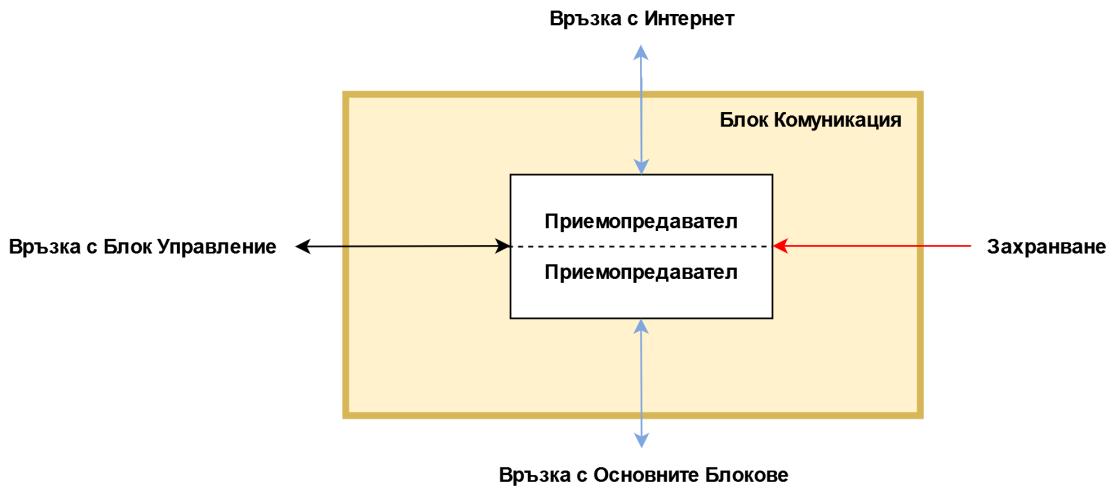
- Блок Управление



Фиг. 2.5. Вътрешна блокова схема на Блок Управление

На фиг. 2.5. е представена вътрешната блокова схема на Блок Управление. Той се състои от Микроконтролер, включващ в себе си необходимата памет, и Входни/Изходни Интерфейси. Блок Входни/Изходни Интерфейси получава информация от Блок Комуникация и я предава на микроконтролера. Той впоследствие я обработва в съответствие с алгоритъма си на работа и изпраща необходимата информация през Блок Входни/Изходни Интерфейси към Блок Комуникация и Блок Индикация.

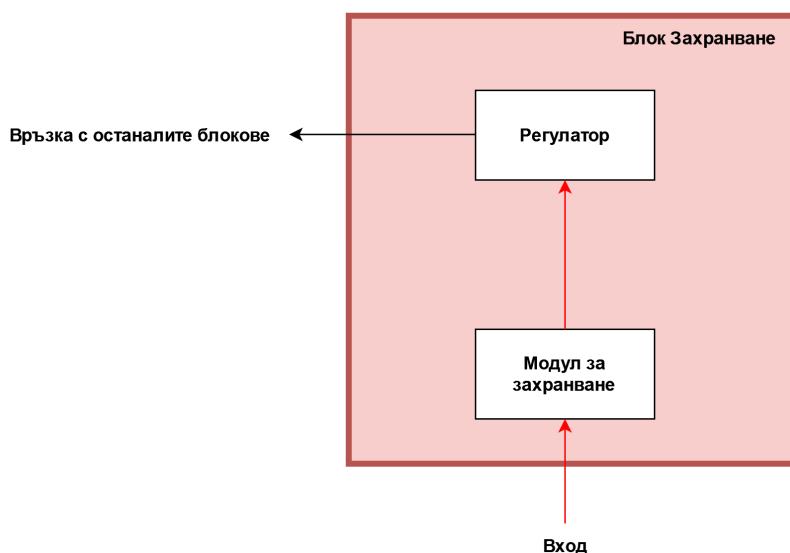
- Блок Комуникация



Фиг. 2.6. Вътрешна блокова схема на Блок Комуникация

На фиг. 2.6. е представена вътрешната блокова схема на Блок Комуникация. Той се състои от различни по вид приемопредаватели, всеки от които осъществява двупосочна връзка с Блок Управление и съответно безжична двупосочна връзка с Интернет или с Основните Блокове.

- Блок Захранване

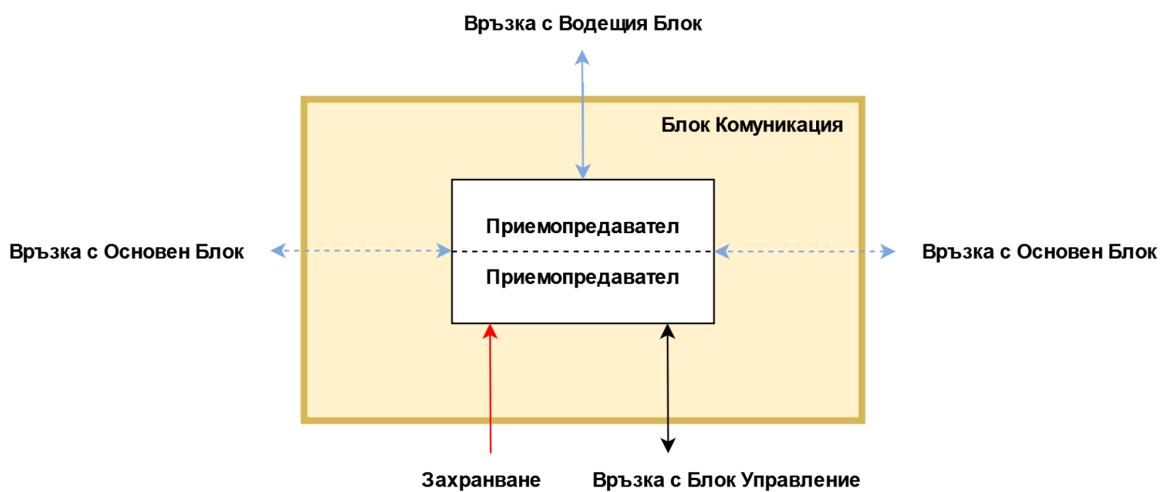


Фиг. 2.7. Вътрешна блокова схема на Блок Захранване

На фиг. 2.7. е представена вътрешната блокова схема на Блок Захранване. Той се състои от модул за захранване и регулятор, осигуряващ различните напрежения, необходими за останалите модулите.

Подблокове на Основния Блок

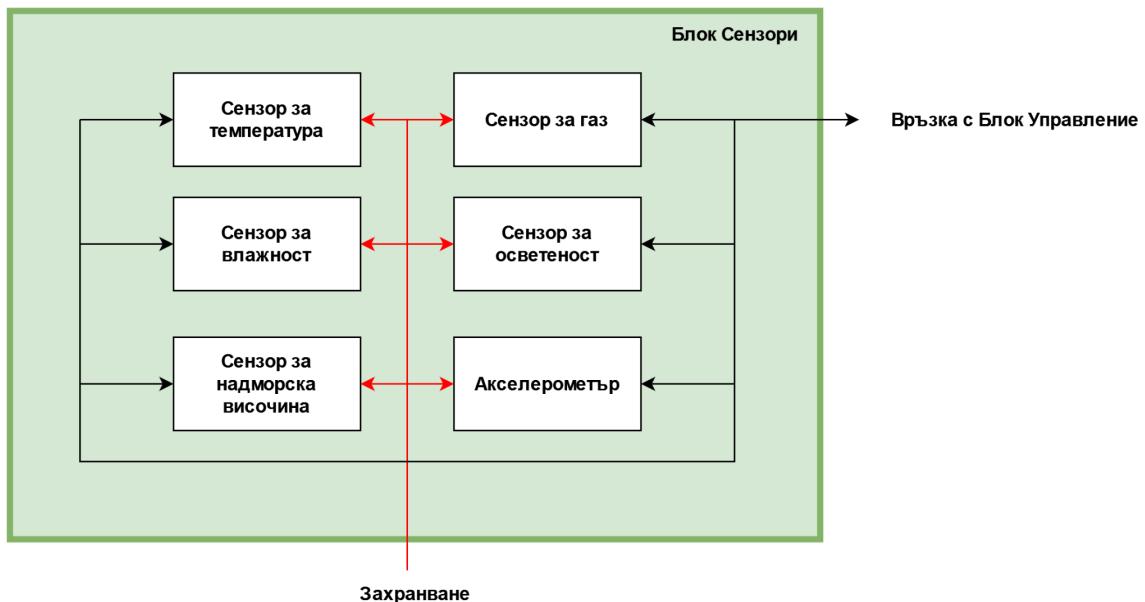
- Блок Комуникация



Фиг. 2.8. Вътрешна блокова схема на Блок Комуникация

На фиг. 2.8. е представена вътрешната блокова схема на Блок Комуникация. Неговата вътрешна структура е аналогична на тази на Блок Комуникация във Водещия Блок, но, за разлика от него, има двупосочна връзка с Водещия Блок и, в зависимост от топологията и функционалността на мрежата - връзка с друг Основен Блок.

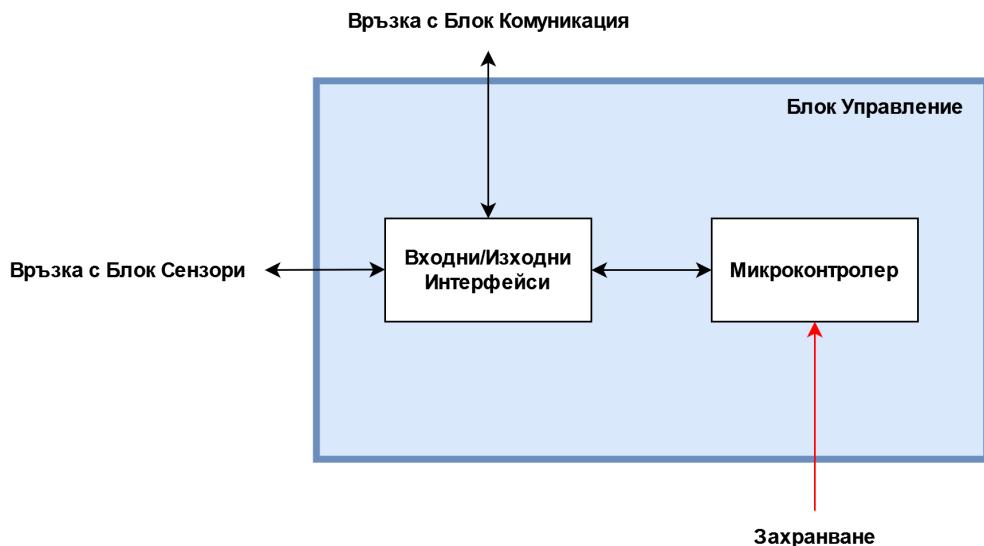
- Блок Сензори



Фиг. 2.9. Вътрешна блокова схема на Блок Сензори

На фиг. 2.9. е представена вътрешната блокова схема на Блок Сензори. Той се състои от различни по вид сензори, които приемат информация от околната среда и, в съответствие с предварително зададена конфигурация от Блок Управление, я изпращат към него в съответен формат.

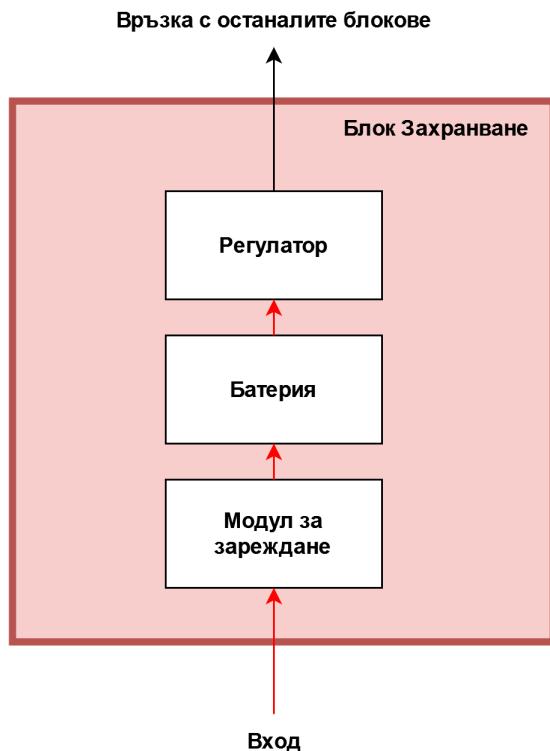
- Блок Управление



Фиг. 2.10. Вътрешна блокова схема на Блок Управление

На фиг. 2.10. е представена вътрешната блокова схема на Блок Управление. Неговата вътрешна структура е аналогична на тази на Блок Управление от Водещия Блок. Основната разлика е връзката с Блок Сензори, на който той изпраща информацията за конфигурация на всеки сензор и получава измерваните от тях параметри на околната среда. Тази информация, след съответна обработка от микроконтролера, се изпраща към Блок Комуникация.

- Блок Захранване



Фиг. 2.11. Вътрешна блокова схема на Блок Захранване

На фиг. 2.11. е представена вътрешната блокова схема на Блок Захранване. Той се състои от модул за зареждане на батерия, самата батерия, която е основния източник на енергия в устройството, и регулатор, осигуряващ различните напрежения, необходими за останалите модулите.

Трета глава - Проектиране на принципна електрическа схема

3.1. Избор на CAD система

За целите на проектирането на принципна електрическа схема и печатна платка ще бъде използвана CAD (на англ. *Computer Aided design*) системата KiCad, създадена от Jean-Pierre Charras.



KiCad е бесплатен и свободен за употреба софтуер за проектиране на електрически схеми и печатни платки. Той има интегрирани инструменти за всички процеси от проектирането - за чертаенето на самите схеми и тяхната симулация, за създаването на печатни платки, 3D визуализация на компонентите и печатните платки, и освен това позволява създаването на собствени библиотеки и компоненти. Софтуерът има много голяма общност от хора около себе си и поради тази причина има голямо количество библиотеки, компоненти и допълнителни инструменти, създадени от други разработчици [49]. KiCad, макар бесплатен и предназначен основно за любители, намира вече и интеграция в по-профессионални среди. Компании, които го използват са: CERN(на англ. *Center for European Nuclear Research*), Olimex и др. [50][51].

3.2. Водещ възел

3.2.1. Избор на подходящи компоненти

Както бе споменато в т.2.2.1, основна функция на Водещия възел е да изпълнява функцията на Border router. Това определя и избора на неговите компоненти.

Блок Управление и Блок Комуникация

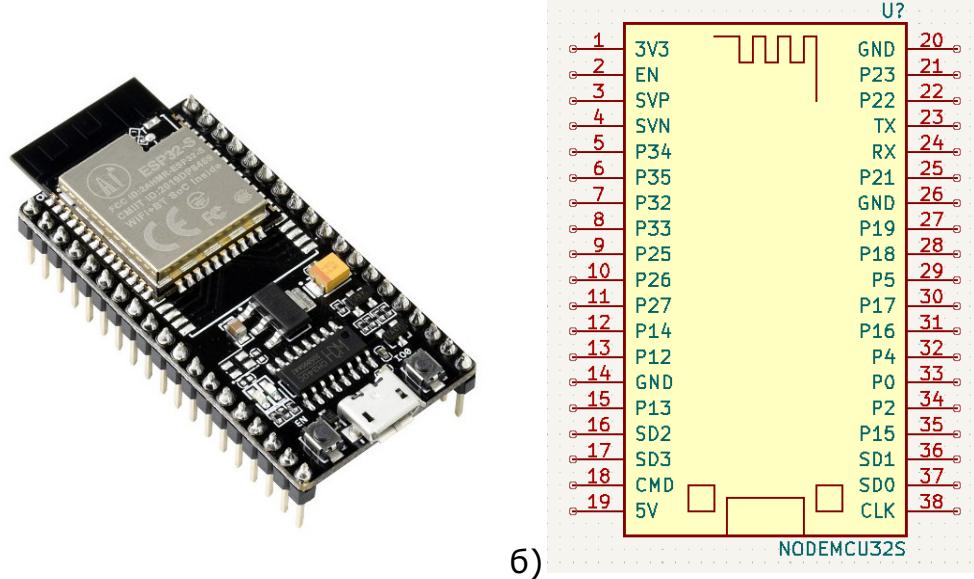
Въз основа на сравнението и анализа в т.1.3. за реализацията на Блок Управление и Блок Комуникация ще бъдат използвани следните компоненти: NodeMCU-32S и nRF52840 Dongle.

На официалния сайт на OpenThread (модификация на използваня за проекта протокол Thread, описан по-подробно в пета глава) е наличен пример за създаване на Thread Border Router с помощта на ESP32 series Wi-Fi dev kit и ESP32-H2 dev kit [52]. В случая ESP32 series Wi-Fi dev kit изпълнява ролята на host устройство, а модулът ESP32-H2 - на RCP (на нагл. *Radio Co-Processor*) (RCP е описан в т.6.3.1.), защото има вградено 15.4. Радио. Поради ниското разпространение на този модул в дипломния проект той ще бъде заменен от nRF52840 Dongle, който също има налично 15.4 радио и може да бъде конфигуриран като RCP.

Конкретна информация за отделните компоненти е описана по-долу.

NodeMCU-32S

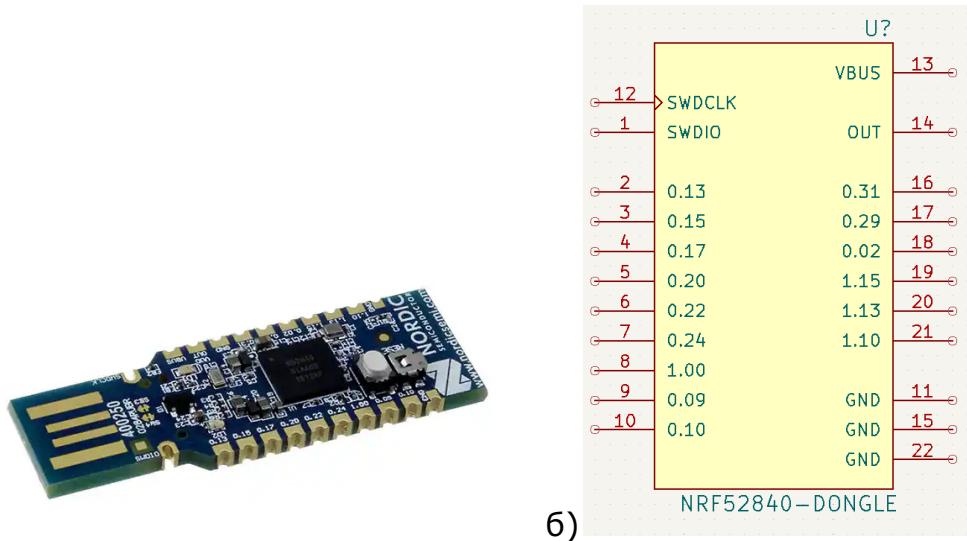
От направеното сравнение в т.1.3.1. се вижда, че модулът NodeMCU-32S има необходимата функционалност и мощност, както и вграден Wi-Fi приемопредавател, за да бъде използван като host устройство, осъществяващо връзката към Интернет [53]. Освен това той е лесно достъпен в България, както и лесен за използване поради наличието на детайлна документация и огромната общност от ентузиасти разработващи с ESP32 микроконтролери. Чрез него биват реализирани Блок Управление и Блок Комуникация (в частта си за връзка с Интернет) в Основния Блок. На фиг. 3.1.a) е показан изглед към използвания модул, а на фиг. 3.1.b) - схемното изображение на модула.



Фиг. 3.1.а) Изглед към използвания модул - NodeMCU-32S, б) Схемно изображение на NodeMCU-32S

nRF52840 Dongle

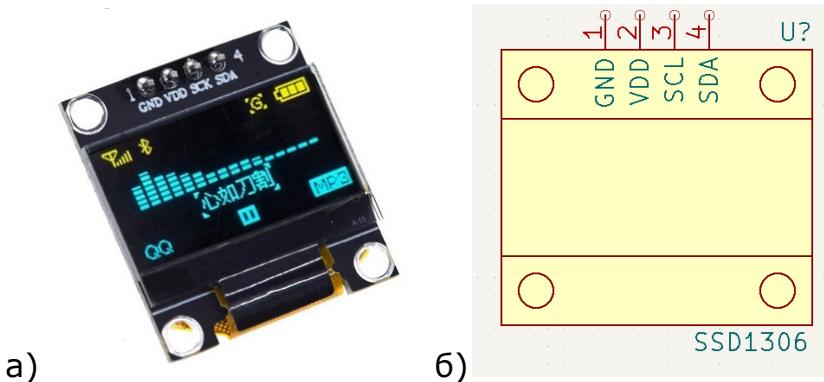
От направеното сравнение в т.1.3.1. се вижда, че nRF52840 Dongle е подходящ за използване във Водещия възел, защото съдържа в себе си nRF52840 SoC, споменат в т.1.3.2., който има вграден приемопредавател, поддържащ 802.15.4 стандарта, и може да бъде конфигуриран като RCP [54]. По този начин чрез неговото използване се реализират функциите на Блок Комуникация (в частта си за връзка с Основните възли). На фиг. 3.2.а) е показан изглед към използвания модул, а на фиг. 3.2.б) - схемното му изображение.



Фиг.3.2.а) Изглед към използвания модул - nRF52840 Dongle, б)
Схемно изображение на nRF52840 Dongle

Блок Индикация

От описанието в т.1.3.3 се вижда, че SSD1306 дисплеят предоставя достатъчно възможности, за да бъде използван в проекта. Той има малък размер, което помага за компактността на устройството, като същевременно има възможност за представяне на достатъчно информация на 8 реда. На фиг. 3.3.a). е показан изглед към използвания модул, а на фиг. 3.3.б). - схемното му изображение [55].

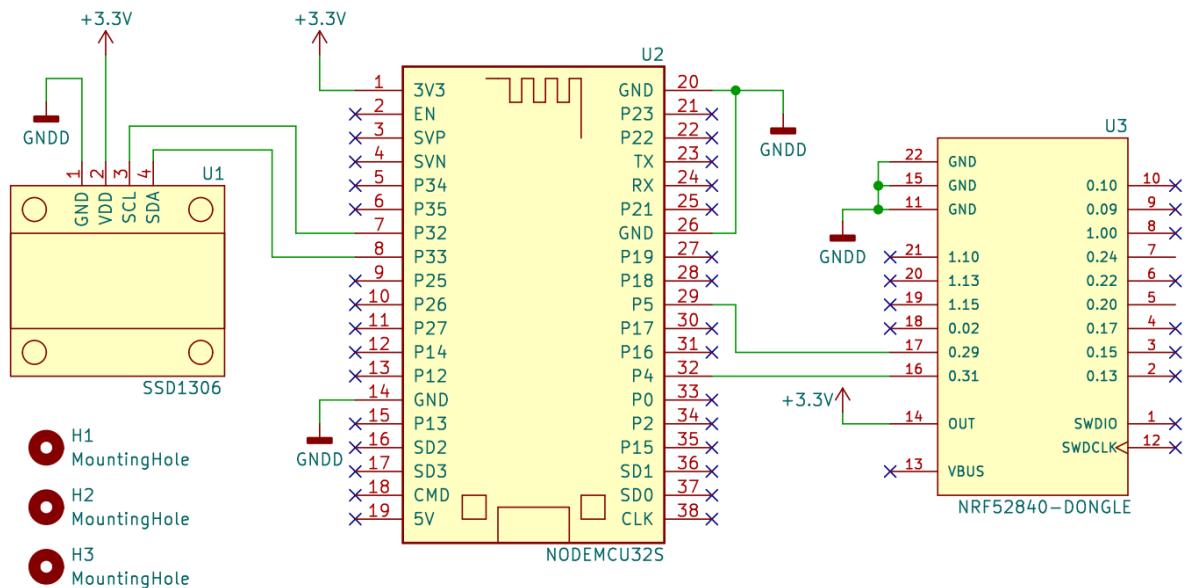


Фиг. 3.3.а) Изглед към използвания дисплей - SSD1306, б) Схемно изображение на SSD1306

Блок Захранване

Тъй като се предвижда Водещият блок да е стационарен, и е възможно да се свърже с USB кабел към компютъра, на който ще се демонстрира функционалността на системата, захранването към този блок може да се подаде по същия този кабел. Друг вариант за захранването, който също не изисква допълнителни компоненти в схемата на водещия възел, е да се захрани с външен AC/DC (на англ. *Alternating current/Direct current, бълг. Променлив ток/Постоянен ток*) адаптер с изход 5 VDC, при което в този случай възелът ще трябва да се свърже с демонстрационния компютър чрез безжична WiFi връзка. И при двата варианта на практика захранването ще се подаде на NodeMCU-32S, а от него ще се вземе захранване за останалите модули в схемата - nRF52840-Dongle и SSD10306.

3.2.2. Принципна електрическа схема



Фиг. 3.4. Принципна електрическа схема на Водещия възел

На фиг. 3.4 е показано свързването между всички компоненти на Водещия възел.

Изводите 3(SCL) и 4(SDA) на дисплея SSD1306 са свързани съответно към изводи 7(P32) и 8(P33) на микроконтролера NodeMCU32-S, осъществявайки връзка посредством I²C интерфейс. NRF52840-Dongle осъществява UART (на англ. *Universal asynchronous receiver-transmitter*) връзка чрез изводите си 7(0.29) и 16(0.31) съответно с изводи 29(P5) и 32(P4) на микроконтролера NodeMCU32-S. Захранването се подава на MicroUSB порта на управляващия модул, като през него се захранват всички останали модули чрез съответните им захранващи изводи.

3.3. Основен възел

3.3.1. Избор на подходящи компоненти

Блок Управление и Блок Комуникация

nRF52840 Dongle

Въз основа на сравнението и анализа в т. 1.3.1. за реализацията на Основния възел ще бъде използван nRF52840 Dongle. Той има достатъчна производителност, за да реализира функцията на управление и, благодарение на наличното си 802.15.4 вградено радио, ще може чрез него да се реализира, освен Блок Управление, и Блок Комуникация. Чрез избирането на nRF52840 Dongle ще се използва един и същ модул както във Водещия възел, така и в Основния. Той има добре описана документация и голяма общност от разработчици, позволяващи по-лесното му използване.

Неговият избор бива подкрепен и от Zenatriz със своето решение - WattMan, описано в т.1.1.2., които също избират nRF52840 за реализацията на своя проект поради следните причини [2]:

- nRF52840 поддържа мултипротоколна mesh мрежа, улесняваща преминаването към други популярни протоколи за безжична

свързаност без необходимостта за актуализиране на хардуера на приложението.

- Сензорите в WattMan използват две AAA батерии, които осигуряват живот на батерията от 15 до 18 месеца. Тази продължителност бива постигната благодарение на ултра ниската консумация на енергия от страна на nRF52840 SoC. Той е проектиран да минимизира консумацията на енергия чрез автоматична система за управление на мощността, която намалява консумацията с до 80% в сравнение с другата серия - nRF51.

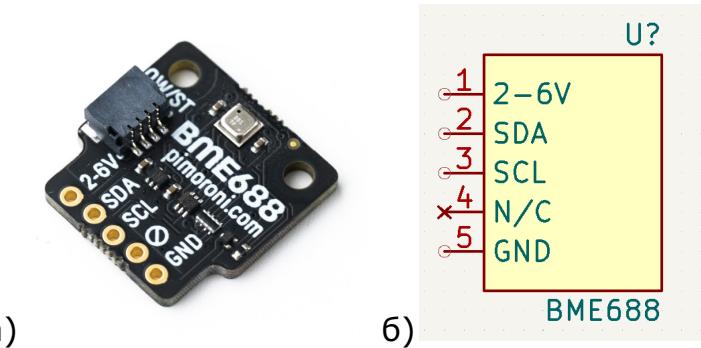
Тези фактори аргументират избора за имплементация на nRF52840 Dongle в проекта.

Блок Сензори

Въз основа на функционалните изисквания, описани в т.2.1.1, за проекта ще е необходимо наблюдението на следните величини - температура, влажност, газов състав на околната среда, надморска височина, осветеност и движение.

BME688

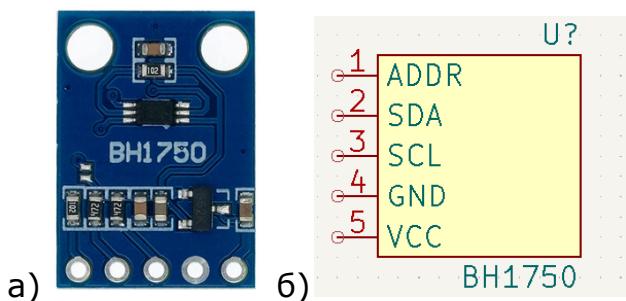
За улесняване разработката на проекта най-добре би било, ако бъдат използвани комбинирани сензори. От описаните в т.1.3.3. такива BME688 се явява добър избор, защото комбинира в себе си повечето необходими за проекта сензори - сензор за температура, влажност, надморска височина и газове. Той може да работи с I²C/SPI и е широко разпространен. Модулът, който ще бъде използван в проекта, е на компанията Pimoroni Ltd [56] и е на достъпна цена - 51lv. Изглед към модула е показан на фиг. 3.5.a), а на фиг. 3.5.6) е показано схемното му изображение.



Фиг. 3.5.a) BME688 модул на компанията Pimoroni, б) Схемно изображение на BME688

BH1750

От направения анализ в т.1.3.3. BH1750 се явява най-добър сензор за осветеност, комбинирайки в себе си добра точност и голям диапазон. Той работи чрез I²C интерфейс. Негови предимства се явяват както лесната достъпност до модулни платки, така и ниската му цена - 1,59лв. [58]. Изглед към модула е показан на фиг. 3.6.a), а на фиг. 3.6.б) е показано схемното му изображение.

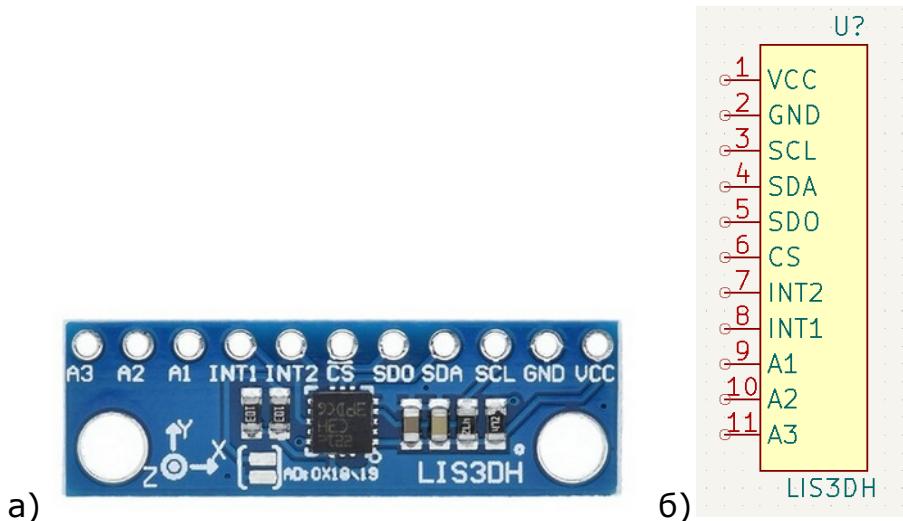


Фиг. 3.6.a) Изглед към използванния BH1750 модул, б) Схемно изображение на BH1750

LIS3DH

LIS3DH е високопроизводителен акселерометър, който ще бъде използван като сензор за движение в проекта. Негово описание е направено в т.1.3.3. LIS3DH е използван в много модулни платки, като

тази, която ще бъде използвана в проекта е на компанията TZT и струва 1.91лв [58]. Този модул е нискобюджетен и предоставя всички необходими интерфейси за работа със сензора. На фиг. 3.7.a) е показано изображение на модула, а на фиг. 3.7.б) - схемното изображение.



Фиг. 3.7.а) Изглед към използвания LIS3DH модул, б) Схемно изображение на LIS3DH

Блок Захранване

Захранването на Основния възел трябва да издържа възможно най-дълго време и да бъде максимално компактно, както и да функционира добре при различни условия.

Основният консуматор на този възел е nRF52840-Dongle, като неговата максимална консумация е около 20mA. В сравнение с нея, консумацията на останалите модули на възела е пренебрежимо малка. При това положение, за да може да работи модула достатъчно дълго (напр. поне 7 дни), ще е необходима батерия с капацитет

$$K_{bat} \geq I_{max} * t, \quad (3.1.)$$

където K_{bat} е капацитета на батерията в ампер-часа (Ah);

I_{max} е максималната консумация на възела в ампери (A);
 t е времето в часове (h).

В случая на проекта $K_{bat} \geq 0.020 * 168 = 3.360Ah$ или $3360mAh$.

За реализирането на Блок Захранване от Основния възел ще бъдат използвани следните компоненти:

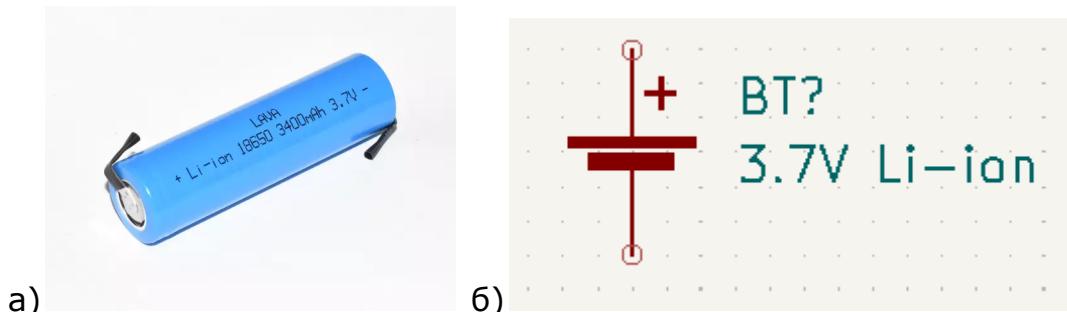
Li-ion Батерия - 3.7V/3400mAh 18650 LAVA

Литиево-йонните батерии се явяват като добър избор за портативно захранване поради тяхната крива на разряд, добрата им температурна стабилност и дълъг живот. Именно затова ще бъде използвана такава батерия за проекта.

Избраната батерия има следните параметри [59]:

- Стандартен размер - 18650 ;
- Номинален капацитет - 3400mAh;
- Напрежение на заряд - 4.2V;
- Номинално напрежение - 3.63V;
- Работна температура при разряд - -20 до 60°C;
- Цена - 13,90лв.

Изглед към батерията е показан на фиг. 3.8.a), а нейното схемно изображение - на фиг. 3.8.б).



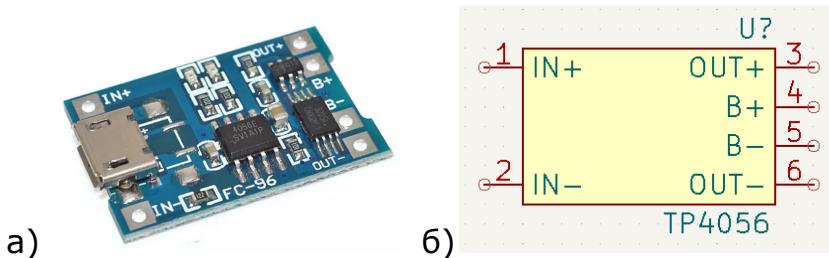
Фиг. 3.8.a) Изглед към литиево-йонната батерия, б) Схемно изображение на батерията

Модул за зареждане - TP4056

За зареждането на литиево-йонната батерия и използването ѝ като източник на енергия за Основния модул ще бъде необходим модул за зареждане. TP4056 се явява като добра опция поради добрите си параметри, ниска цена и голямото разпространение. Той има следните параметри [60]:

- Захранващо напрежение - 5V;
- Максимален заряден ток - 1A;
- Защита от претоварване по ток - 3A;
- Защита от преразреждане по напрежение - 2.5V;
- Порт за зареждане - Micro USB;
- Цена - 1,80лв.

Изглед към модула е показан на фиг. 3.9.a), а неговото схемно изображение - на фиг. 3.9.б).



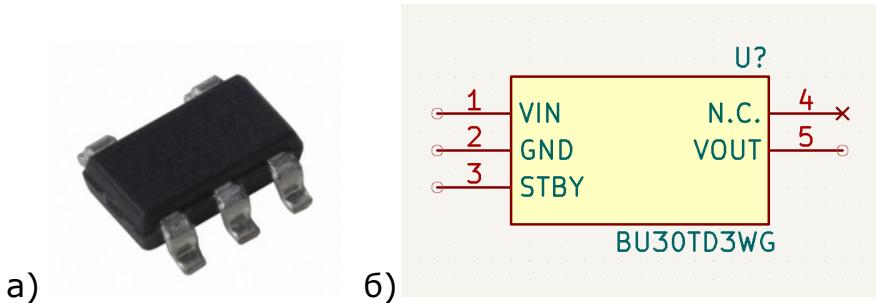
Фиг. 3.9.а) Изглед към модула TP4056, б) Схемно изображение на TP4056

Линеен стабилизатор - BU30TD3WG

За понижаване и стабилизиране на изходното напрежение от батерията е необходим регулатор. Линеен стабилизатор (на англ. Low-dropout regulator, съкр. LDO) BU30TD3WG на Rohm се явява като добър избор поради ниското му Dropout напрежение - 240mV - 460mV и необходимото за захранване на nRF52840-Dongle изходно напрежение - 3V. Той позволява максимално входно напрежение - 6V, което се явява достатъчно, при положение, че максималното

напрежение на литиево-йонната батерия е 4.2V. Цената на компонента е 0,32лв. [61].

Изглед към компонента е показан на фиг. 3.10.а), а неговото схемно изображение - на фиг. 3.10.б).



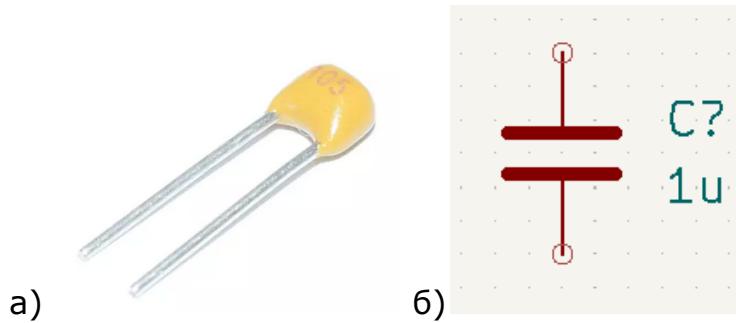
Фиг. 3.10.а) Изглед към корпуса на линейния стабилизатор, б) Схемно изображение на BU30TD3WG

Керамични кондензатори - 1u

Към линейния стабилизатор е препоръчително да бъдат свързани керамични кондензатори на входа и на изхода с минимална стойност - 47uF, определена от листа с каталожни данни на регулятора [62]. Тези кондензатори водят до филтриране на входния шум и стабилизиране на изходното напрежение, компенсирайки резки промени при изходния ток. Параметрите на керамичните кондензатори, които ще бъдат използвани в комбинация с линейния стабилизатор, са следните [63]:

- Капацитет - 1uF
- Максимално работно напрежение - 50V
- Цена - 0,30лв.

Изглед към компонента е показан на фиг. 3.11.а), а неговото схемно изображение е показано на фиг. 3.11.б).



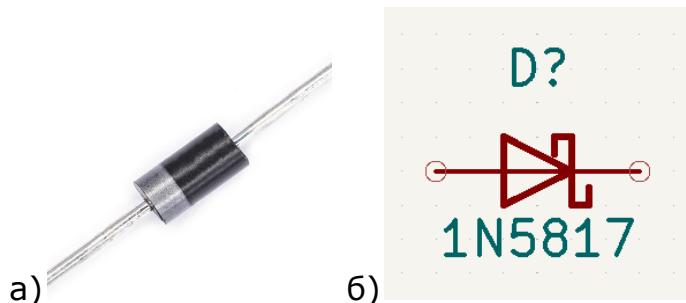
Фиг. 3.11.а) Изглед към корпуса на керамичния кондензатор, б)
Схемно изображение на керамичния кондензатор

Шотки диод - 1N5817

За защита на nRF52840-Dongle от обратно напрежение при едновременно включване на dongle-а в компютъра и кабел за зареждане в TP4056 е необходим диод. Шотки диода - 1N5817 се явява като добър избор заради ниския му пад на напрежение в права посока, висока токова способност и надеждност [64]. Неговите параметри са следните [65]:

- Максимален пад на напрежение в права посока при 1A - 0.45V
- Максимално DC блокиращо напрежение - 20V
- Максимален DC обратен ток при стайна температура - 1mA
- Цена - 0,18лв.

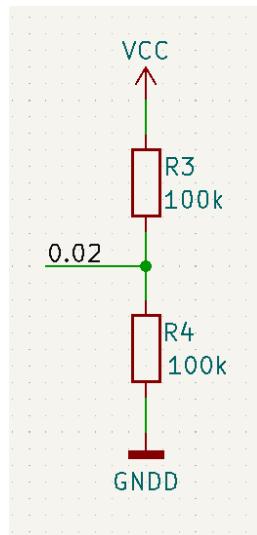
Изглед към компонента е показан на фиг. 3.12.а), а неговото схемно изображение е показано на фиг. 3.12.б).



Фиг. 3.12.а) Изглед към корпуса на 1N5817, б) Схемно изображение на 1N5817

Резисторен делител - 2x100K

За измерване на напрежението на батерията ще се използва делител на напрежение, средният извод на който се подава на аналогов вход на nRF52840-Dongle. Големината на двета резистора ще бъде 100 Ω . Входното напрежение ще идва директно от батерията и ще стига максимална стойност 4.2V. Изходното максимално напрежение при тази стойност е изчислена чрез формулата изобразена на фиг. 3.14. и ще се равнява на 2.1V. На схемата R1 е означен като R3, а R2 - като R4. Делителят на напрежение е показан на фиг. 3.13.



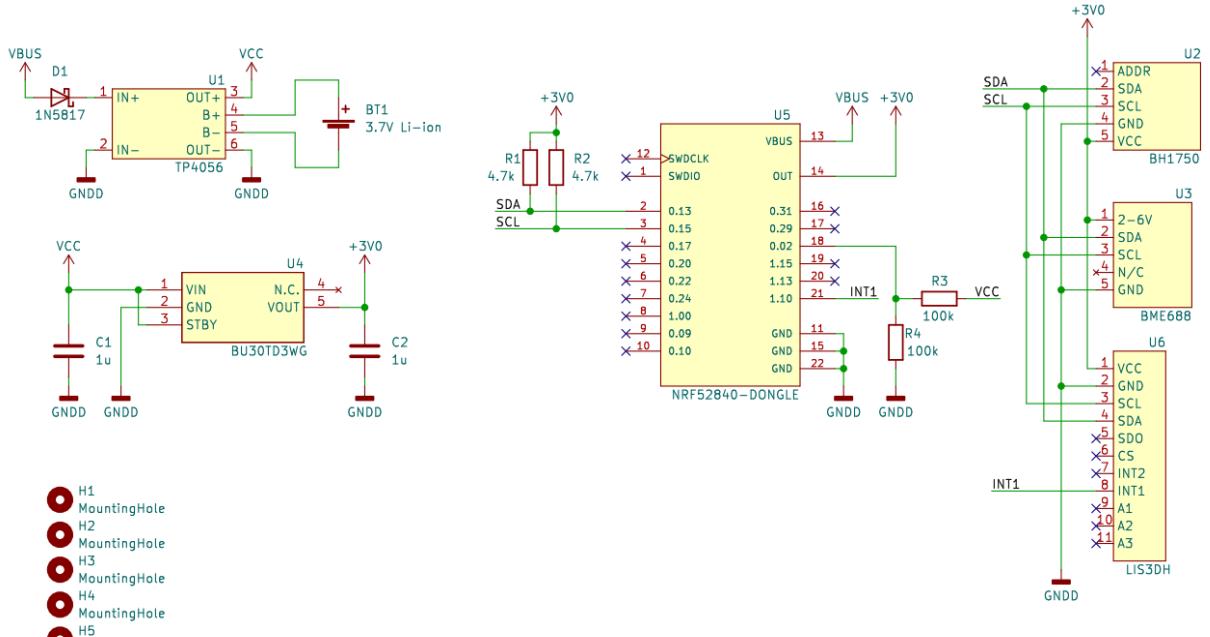
Фиг. 3.13. Схема на делителя на напрежение

$$V_{out} = V_{in} \frac{R_2}{(R_1 + R_2)} \quad (3.2.)$$

Фиг. 3.14. Формула за изчисляване на изходното напрежение

3.3.2. Принципна електрическа схема

На фиг. 3.15. е показана принципната електрическа схема на Основния възел.



Фиг. 3.15. Принципна електрическа схема на Основния възел

Елементите и модулите в горната схема се свързват, както е описано по-долу.

Батерията се свързва към изводите 4(B+) и 5(B-) на модула за захранване TP4056. От модула за зареждане се подава напрежение VCC на входа на линейния регулятор - BU30TD3WG, и входа на делителя на напрежение (R3 и R4). Към TP4056 е свързан и шотки диод, чийто катод е свързан към модула за захранване, а анодът му - към извод 13(VBUS) на nRF52840-Dongle. На вход 1(VIN) на регулатора BU30TD3WG е свързан керамичен кондензатор C1, а на изхода му 5(VOUT) - керамичен кондензатор C2. Извод 3(STBY) е свързан към захранването VCC. От изход 5(VOUT) на LDO регулатора излиза стабилизираното напрежение +3V0, което захранва микроконтролера и всички сензори. От своя страна микроконтролерът, чрез изводите си 2(0.13) и 3(0.15), осъществява I²C връзка с всички сензори. Извод 2 на микроконтролера отговаря на сигнала SDA, а извод 3 - на SCL. Освен това извод 21(1.10) на nRF52840-Dongle е

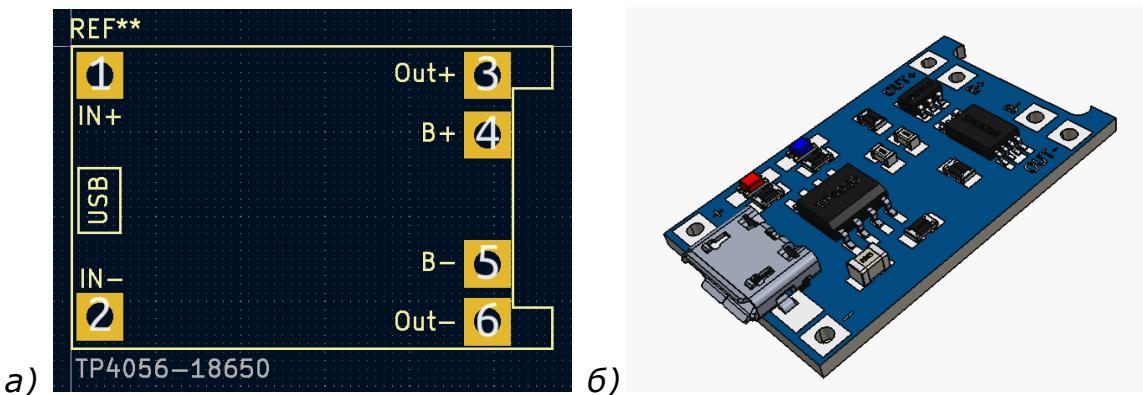
свързан с извод 8(INT1) на LIS3DH. Финално към извод 18(0.02) на микроконтролера е свързан и изхода на делителя на напрежение R3/R4. В схемата са добавени и резисторите R1 и R2, които повдигат линиите SDA и SCL към захранването 3V0, но не са задължителни, поради вече наличните pull-up резистори на всички сензорни модули.

Четвърта глава - Проектиране на печатна платка

4.1. Корпуси и 3D модели на компонентите

В тази секция ще бъдат разгледани корпусите и 3D моделите, използвани за синтезирането на печатната платка.

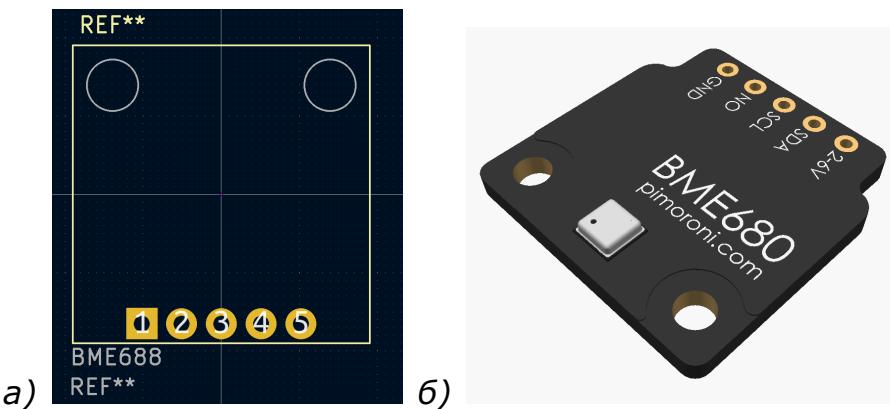
Корпус и 3D модел на TP4056 модул



Фиг. 4.1.a) Изображение на корпуса на LIS3DH модул, б) Изглед към 3D модела на LIS3DH модула

На фиг. 4.1.a) е показан корпусът на TP4056 модула, който има 6 крака и едно MicroUSB, което, макар да не е показано на корпуса, е свързано към двета входни пина - IN+ и IN-. На фиг. 4.1.б) е показан 3D моделът на съответния модул [66][67].

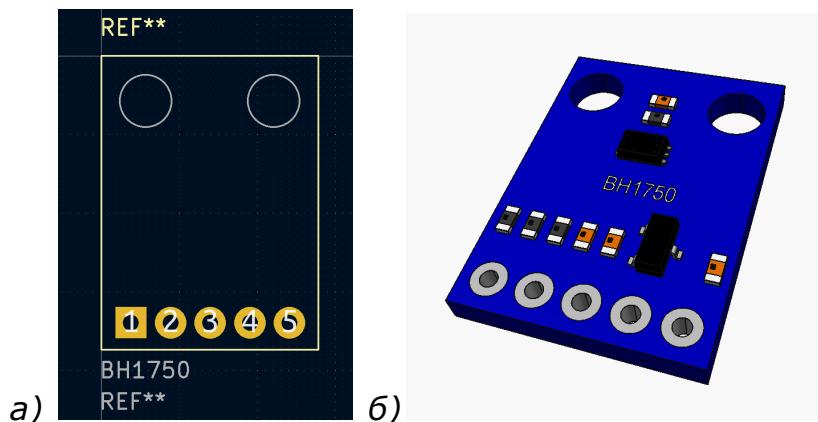
Корпус и 3D модел на BME688 модул



Фиг. 4.2.а) Изображение на корпуса на BME688 модул, б) Изглед към 3D модела на BME688 модула

На фиг. 4.2.а) е показан корпусът на BME688 модула, който има 5 крака и 2 отвора. На фиг. 4.2.б) е показан 3D моделът на съответния модул, като той е за breakout платка с BME680, но съответства напълно с размерите на BME688 [68].

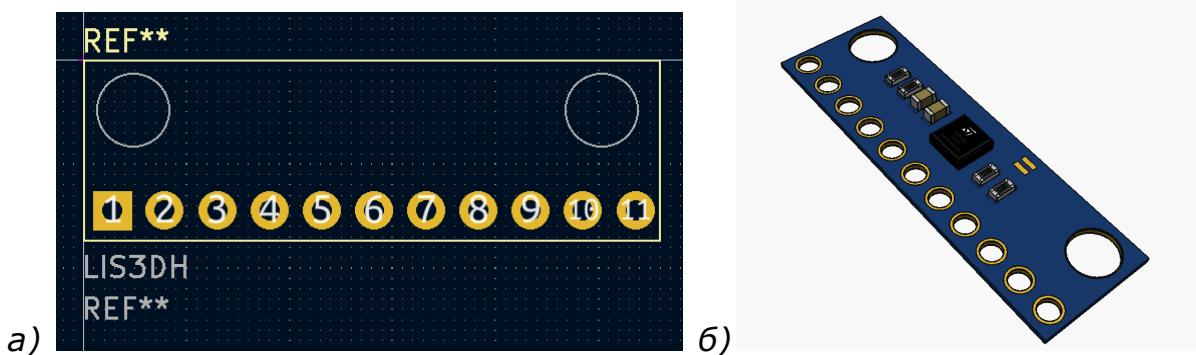
Корпус и 3D модел на BH1750 модул



Фиг. 4.3.а) Изображение на корпуса на BH1750 модул, б) Изглед към 3D модела на BH1750 модула

На фиг. 4.3.а) е показан корпусът на BH1750 модула, който има 5 крака и 2 отвора. На фиг. 4.3.б) е показан 3D моделът на съответния модул [69].

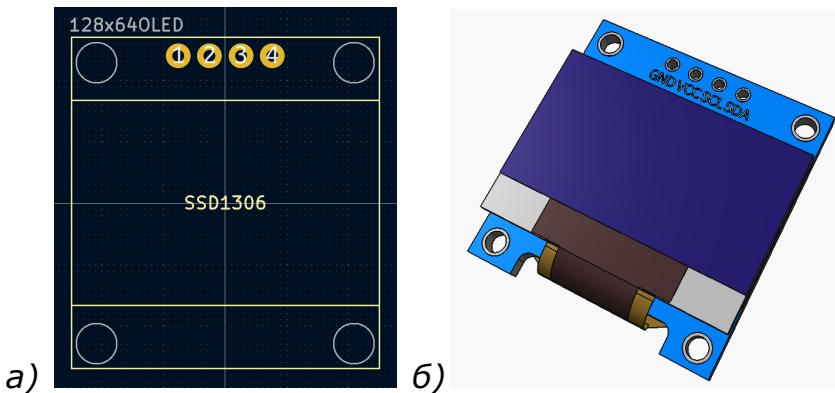
Корпус и 3D модел на LIS3DH модул



Фиг. 4.4.а) Изображение на корпуса на LIS3DH модул, б) Изглед към 3D модела на LIS3DH модула

На фиг. 4.4.а) е показан корпусът на LIS3DH модула, който има 11 крака и 2 отвора. На фиг. 4.4.б) е показан 3D моделът на съответния модул, като той е предоставен на дипломанта от консултанта на дипломната работа - инж. М. Ангелов.

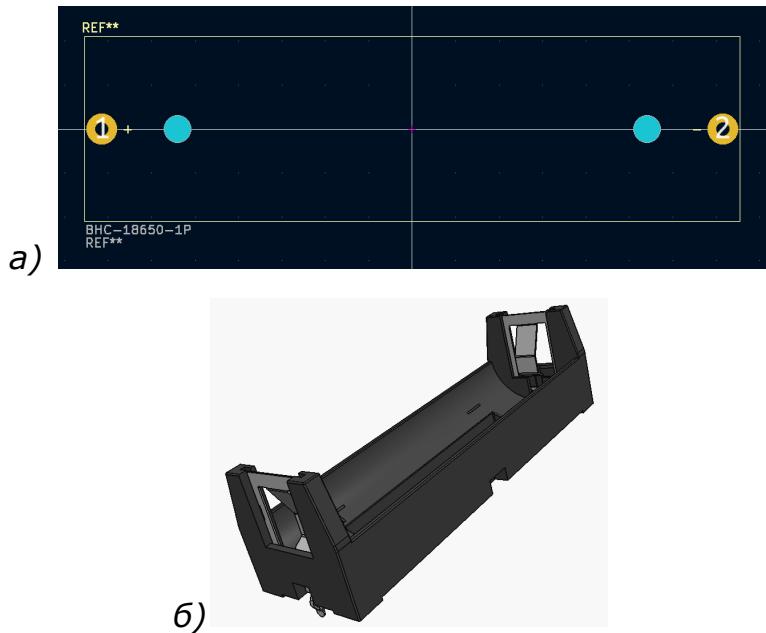
Корпус и 3D модел на SSD1306 дисплей



Фиг. 4.5.а) Изображение на корпуса на LIS3DH модул, б) Изглед към 3D модела на LIS3DH модула

На фиг. 4.5.а) е показан корпусът на SSD1306 дисплея, който има 4 крака и 4 отвора. На фиг. 4.5.б) е показан 3D моделът на съответния дисплей [70].

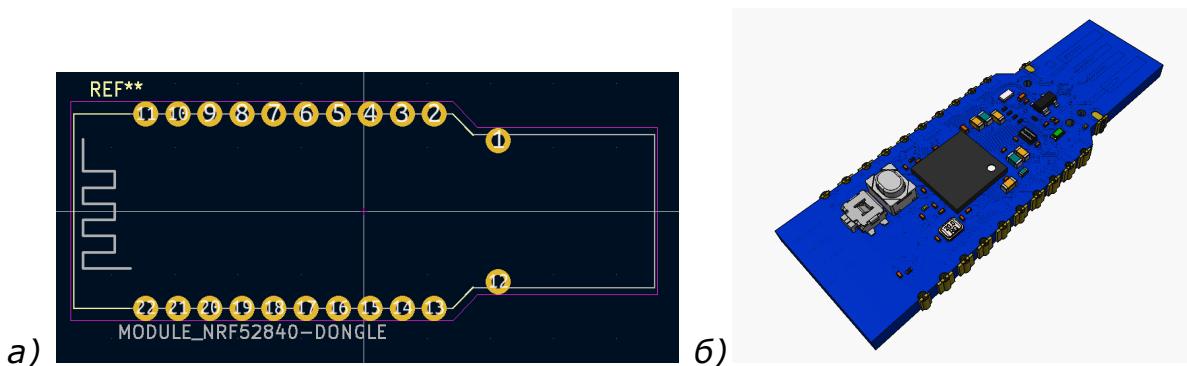
Корпус и 3D модел на ВНС-18650/1Р държач за батерия



Фиг. 4.6.а) Изображение на корпуса на ВНС-18650/1Р, б) Изглед към 3D модела на ВНС-18650/1Р

На фиг. 4.6.а) е показан корпусът на ВНС-18650/1Р - държачът за батерия. Той се състои от 2 крака за "+" и "-" и 2 дупки за винтчета. На фиг. 4.6.б) е показан 3D моделът на съответния държач на батерия [71].

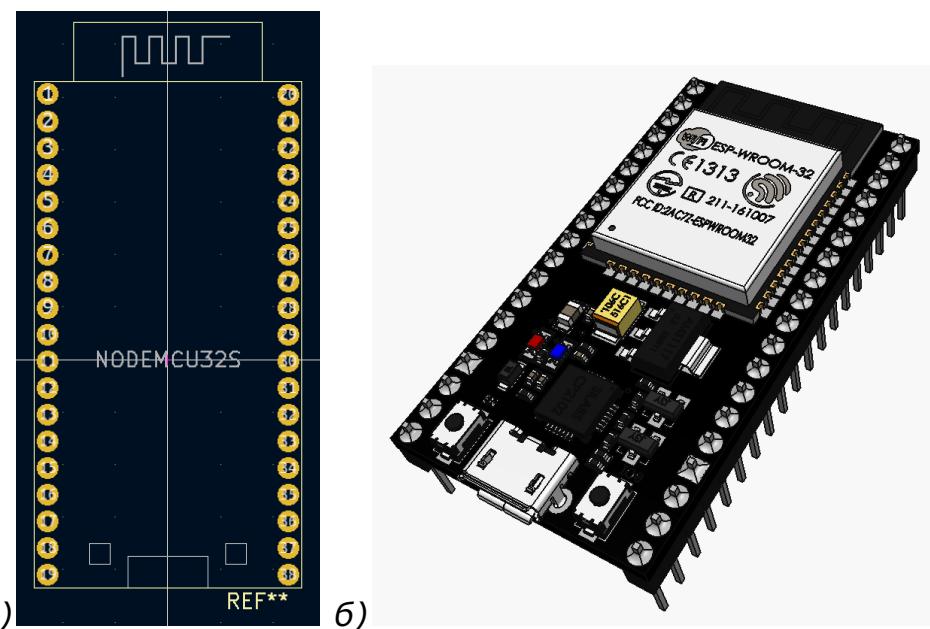
Корпус и 3D модел на nRF52840-Dongle



Фиг. 4.7.а) Изображение на корпуса на nRF52840-Dongle, б) Изглед към 3D модела на nRF52840-Dongle

На фиг. 4.7.a) е показан корпусът на nRF52840-Dongle. Той се състои от 22 крака, като върхът му влиза в USB-A женски съединител. На фиг. 4.7.б) е показан 3D моделът на съответната платка за разработка [72].

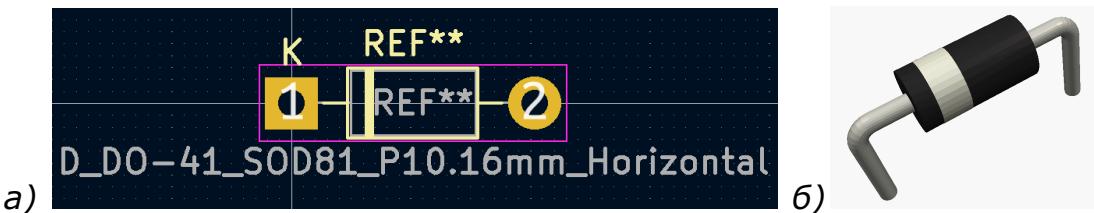
Корпус и 3D модел на NodeMCU32-S



Фиг. 4.8.а) Изображение на корпуса на NodeMCU-32S, б) Изглед към 3D модела на NodeMCU-32S

На фиг. 4.8.а) е показан корпусът на NodeMCU-32S. Той се състои от 38 крака и женско Micro-USB. На фиг. 4.8.б) е показан 3D моделът, който е използван за съответния модул. Той няма точно същия корпус, но има еднакъв брой пинове и размери, поради което ще бъде избран като заместител [73].

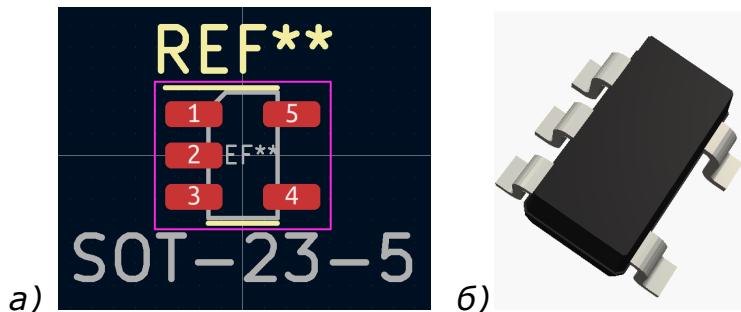
Корпус и 3D модел на Шотки диод 1N5817



Фиг. 4.9.а) Изображение на корпуса на шотки диод 1N5817, б) Изглед към 3D модела на шотки диод 1N5817

На фиг. 4.9.а) е показан корпусът на *шотки диод 1N5817*. Той има 2 крака, като първият крак е катод, показан чрез бяла черта както на корпуса, така и на 3D модела, а вторият крак - анод. На фиг. 4.9.б) е показан 3D моделът на съответния елемент.

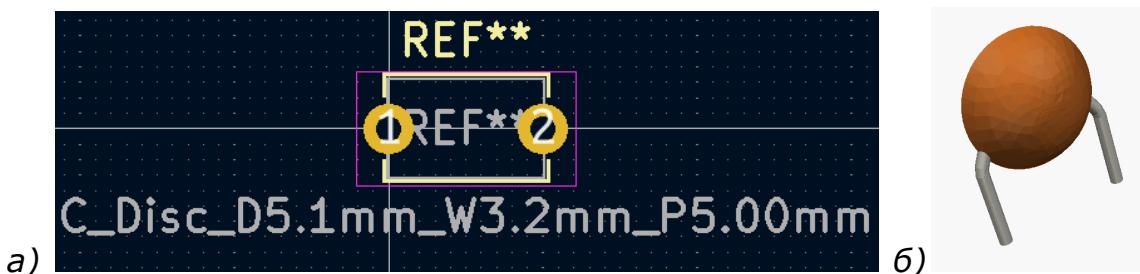
Корпус и 3D модел на LDO BU30TD3WG



Фиг. 4.10.а) Изображение на корпуса на линейния стабилизатор BU30TD3WG, б) Изглед към 3D модела на BU30TD3WG

На фиг. 4.10.а) е показан корпусът на *LDO BU30TD3WG*. Той, разлика от другите елементи, ще бъде SMD (на англ. *Surface-mounted device*). Линейният стабилизатор е от тип SOT-23 и има 5 крака. На фиг. 4.10.б) е показан 3D моделът на съответния елемент.

Корпус и 3D модел на кондензаторите



Фиг. 4.11.а) Изображение на корпуса на керамичните кондензатори,
б) Изглед към 3D модела на кондензаторите

На фиг. 4.11.а) е показан корпусът, който ще се използва за всички керамични кондензатори. Той има 2 крака. На фиг. 4.11.б) е показан 3D моделът на съответния елемент.

Корпус и 3D модел на резисторите



Фиг. 4.12.а) Изображение на корпуса на резисторите, б) Изглед към 3D модела на резисторите

На фиг. 4.12.а) е показан корпусът, който ще се използва за всички резистори. Той има 2 крака. На фиг. 4.12.б) е показан 3D моделът на съответния елемент.

4.2. Основни изисквания

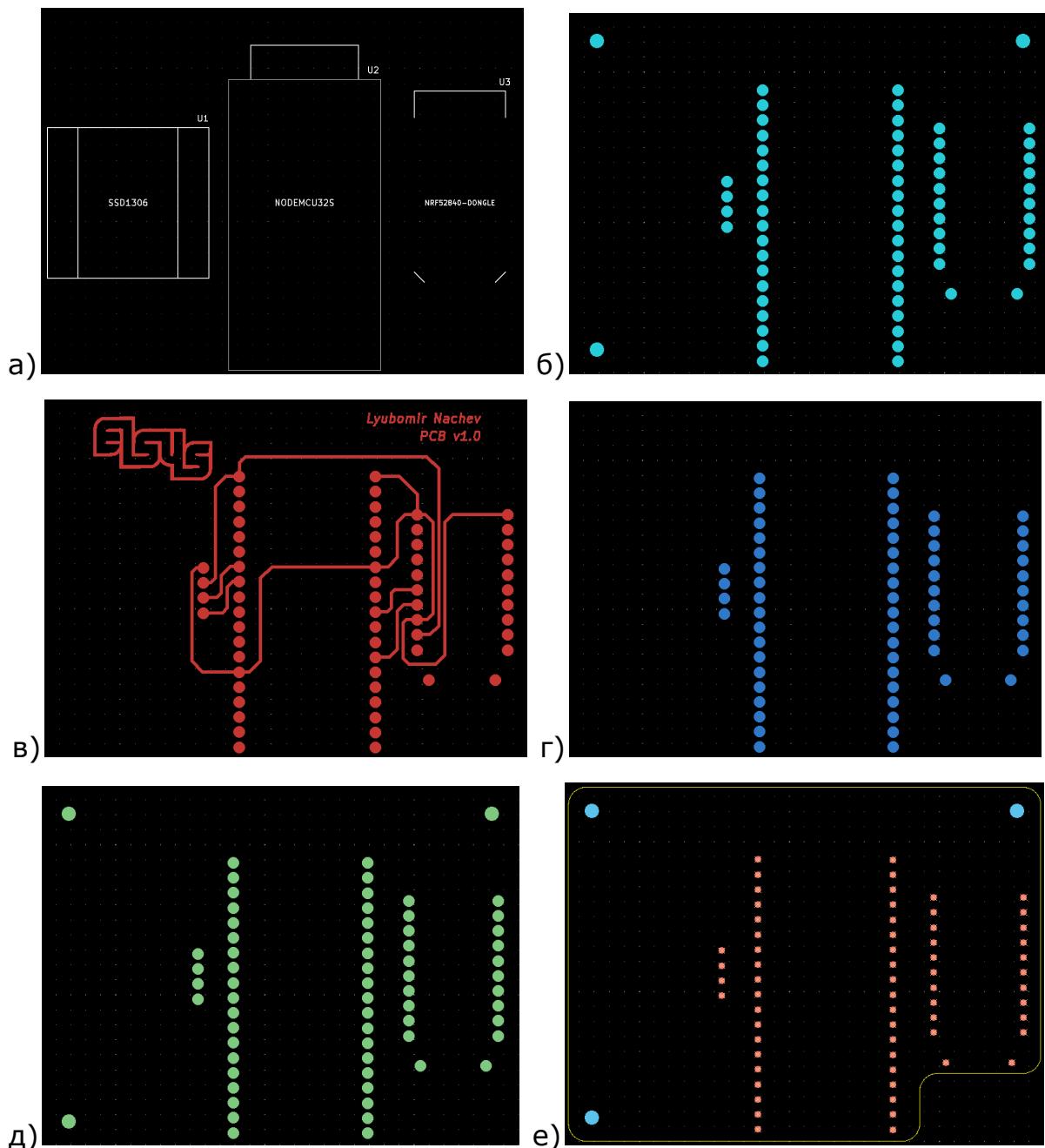
Основните изисквания ще бъдат определени от възможните параметри за генериране на печатна платка от компанията JLCPCB, които са посочени на тяхната страница [74].

4.2.1. Общи

- **Материал, от който ще бъде произведена платката** - FR-4, производство на фирмата Isola.
- **Необходими слоеве** - F.Silkscreen, F.Mask, F.Paste, F.Cu, B.Cu, B.Paste, B.Masks, B.Silkscreen.
- **Механични отвори и обработки** - Edge.Cuts, NPTH(на англ. *Non plating through hole*), PTH(на англ. *Plating through hole*)
- **Покритие на медните слоеве** - HASL (на англ. *Hot Air Solder Leveling*) с олово
- **Захранване/земя** - Захранването е опроведено на горния слой (F.Cu), а земята е copper pour на долния слой (B.Cu)
- **Дебелина на платката** - 1.6mm +/-10% и медното покритие 1 oz/35um
- **Писти (сигнални, захранващи):**
 - ширина на сигналните писти - 0.5mm;
 - ширина на захранващите писти - 0.8mm;
- **Минимално разстояния между пистите и петната:**
 - между пистите - 0.2mm
 - между петната - 0.25mm
 - между петно и писта - 0.25mm
- **Проходни петна/отвори (Vias):**
 - Големина - 0.7mm
 - Отвор - 0.3mm
- **Метализация** - да; Всички отвори са след метализация.
- **Механични отвори** - 3 размера: Ø2.5mm, Ø3mm и oval 3/7mm
- **Габарити и форма на платките:**
 - Големина на платката на водещия възел - 80mm на 60mm със заоблени ъгли
 - Големина на платката на основния възел - 90mm на 85mm със заоблени ъгли

4.3. Печатна платка на Водещ възел

4.3.1. Слоеве на печатната платка на Водещия възел



Фиг. 4.13.а) F_Silkscreen, б) F.Mask, в) F.Cu, г) B.Cu, д) B.Masks, е)
Edge.cuts + NPTH + PTH

Ситопечат на горния слой (F_Silkscreen)

На фиг. 4.13.а) е показан ситопечатът на горния слой на печатната платка на Водещия възел. Това е слоят, на който са поставени всички означения, имена и контури

Зашитна маска на горния слой (F.Mask)

На фиг. 4.13.б) е показан горният слой за защитна маска на печатната платка на Водещия възел. Това са местата, на които ще бъде поставен защитен слой, който предпазва проводниците от оксидация.

Горен слой за паста за запояване (F.Paste) - Няма

Горен слой за опроводяване (F.Cu)

На фиг. 4.13.в) е показан горният слой за опроводяване на печатната платка на Водещия възел. На него са пуснати всички проводници, необходими за свързването на компонентите.

Долен слой за опроводяване (B.Cu)

На фиг. 4.13.г) е показан долният слой за опроводяване на печатната платка на Водещия възел. Поради причината, че тя е изцяло опроводена на горния слой, на този слой има единствено медни отвори, предоставящи възможността за запояване на компонентите и от двете страни.

Долен слой за паста за запояване (B.Paste) - Няма

Зашитна маска на долнния слой (B.Masks)

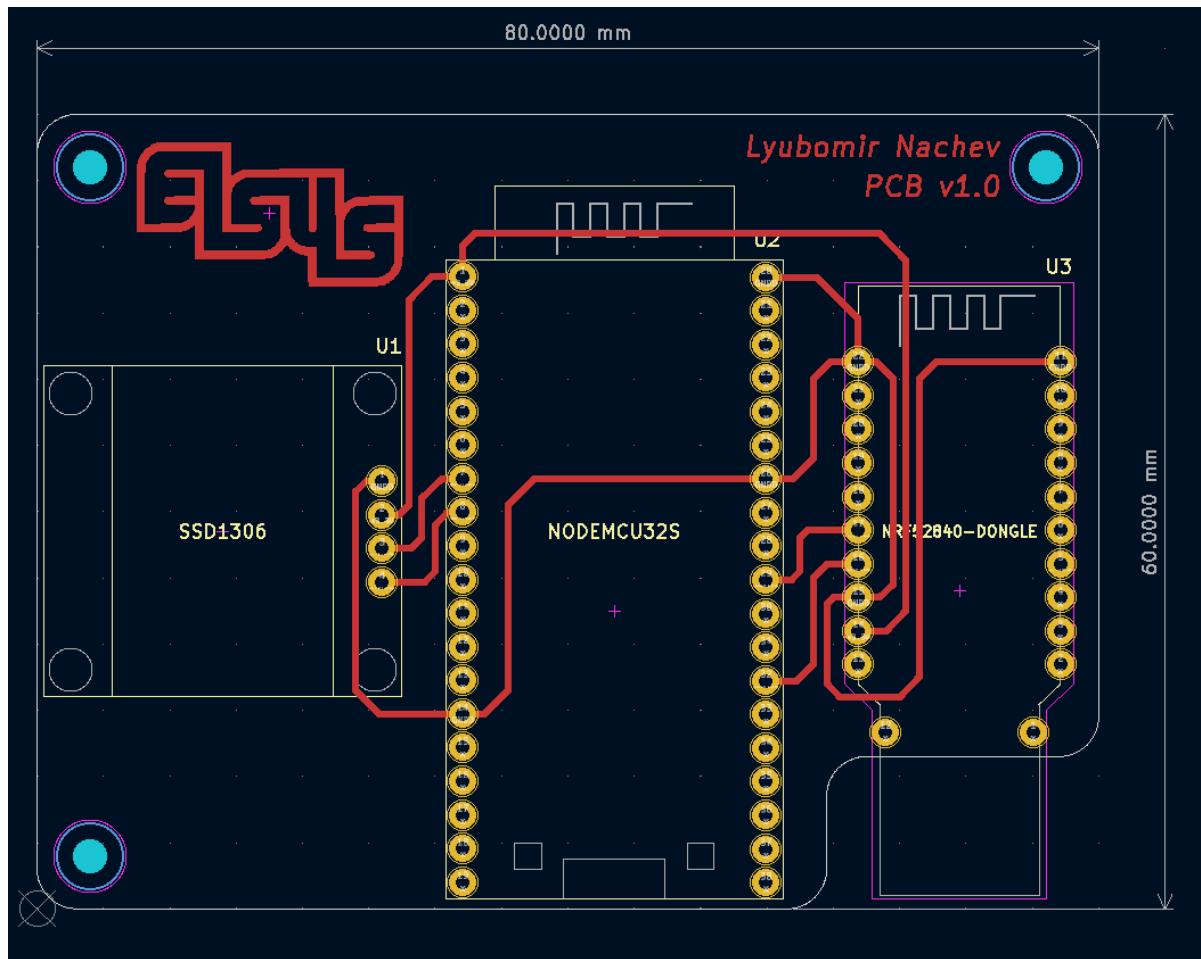
На фиг. 4.13.д) е показан долният слой за защитна маска на печатната платка на Водещия възел.

Ситопечат на долнния слой (B_Silkscreen) - Няма

Слой за изрязване комбиниран с drill файловете (Edge.cuts + NPTH + PTH)

На фиг. 4.13.e) са показани механичните слоеве, на които ще бъдат направени дупки и изрезки. С жълто е означен Edge.cuts слоят, показващ контурите за изрязване на печатната платка. С розово са показани PTH отворите, които ще бъдат пробити и ще имат медно покритие. Последно, със синьо са показани NPTH отворите, които са механично пробити дупки без медно покритие.

4.3.2. Финален вариант на печатната платка

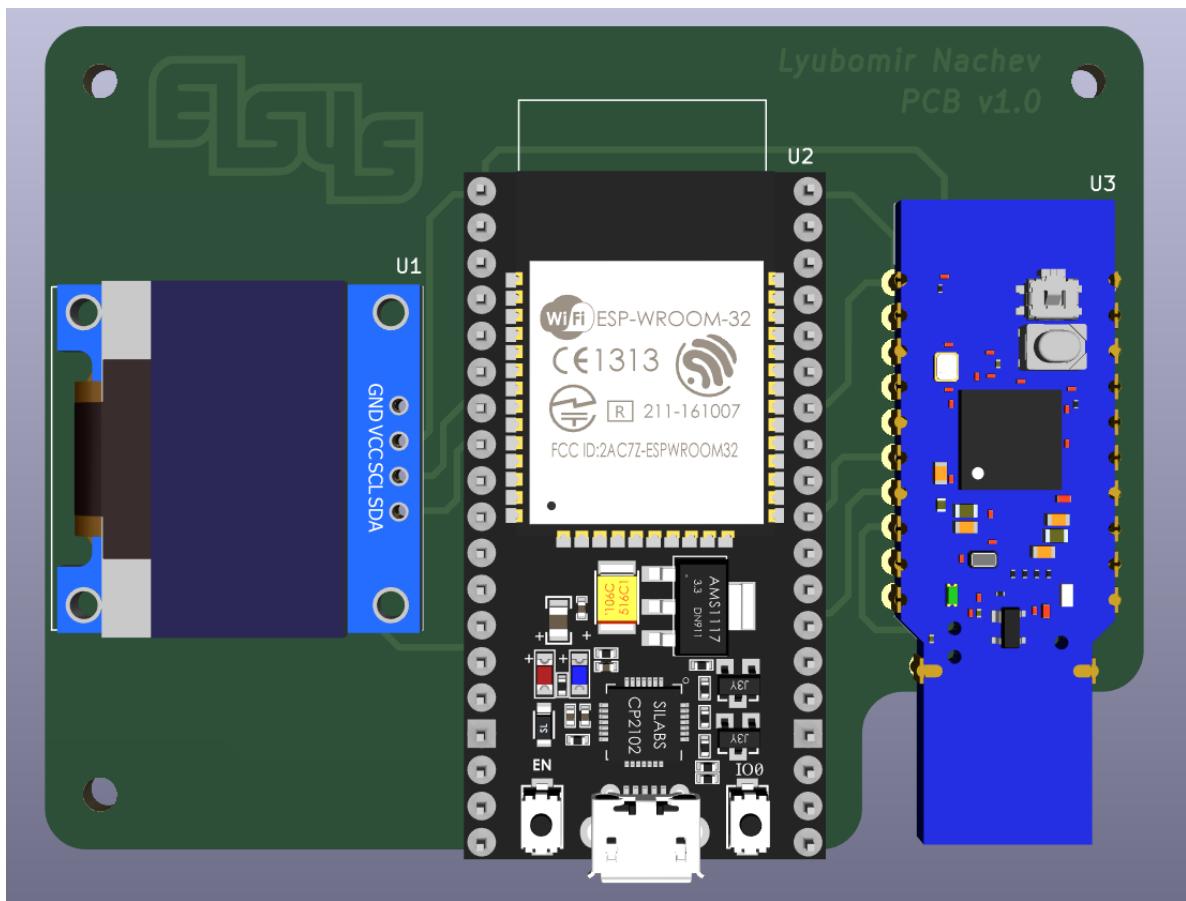


Фиг. 4.14. Цялостен изглед към печатната платка на Водещия възел

На фиг. 4.14. е показан изглед към пълната версия на печатната платка на Водещия възел. Тя е с размери - 80mm на ширина и 60mm

на дължина. Размерите могат да бъдат и по-малки, ако дисплеят бъде оставен извън платката, но е поставен върху нея за допълнителна стабилност.

4.3.3. 3D модел на печатната платка

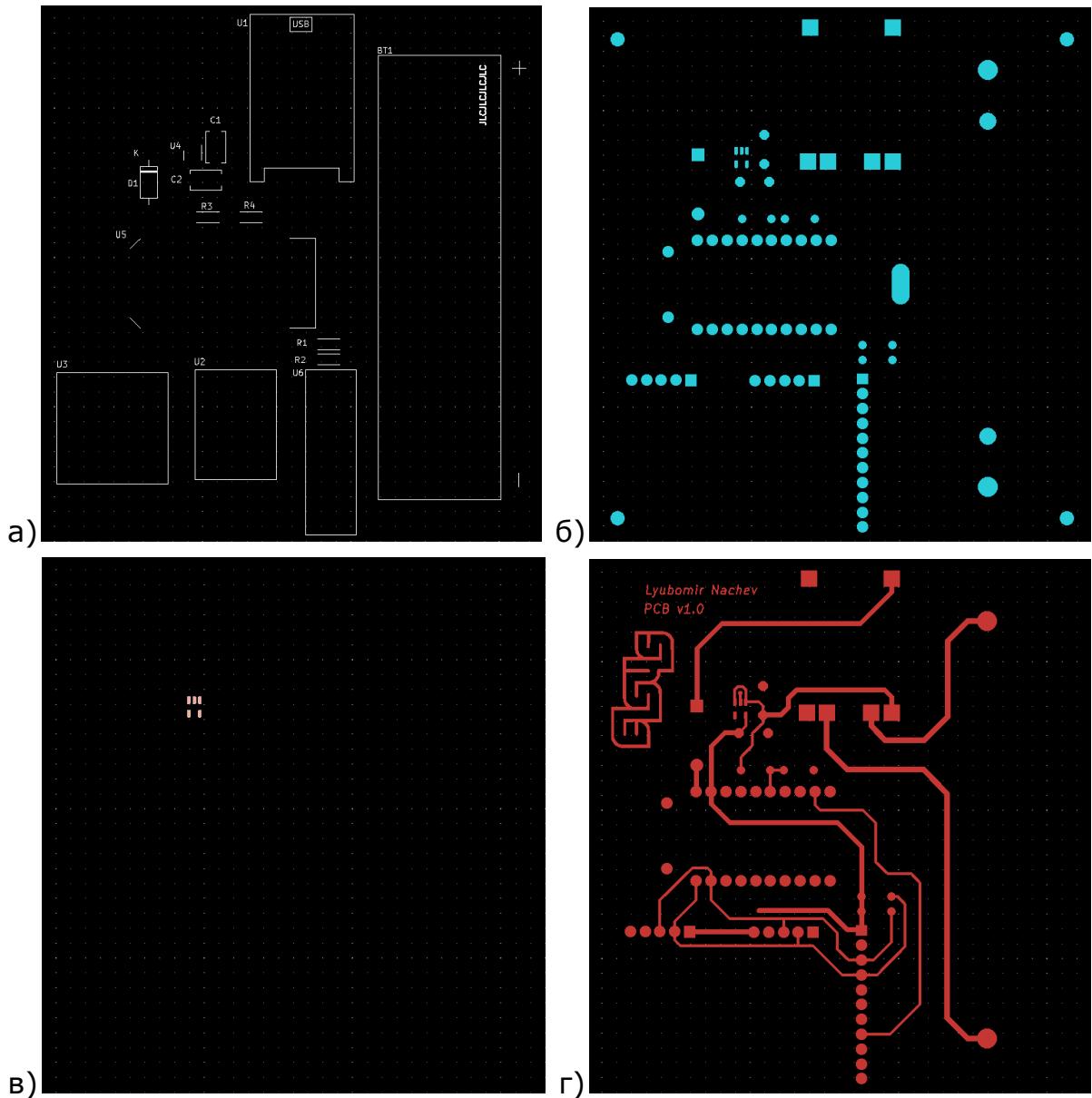


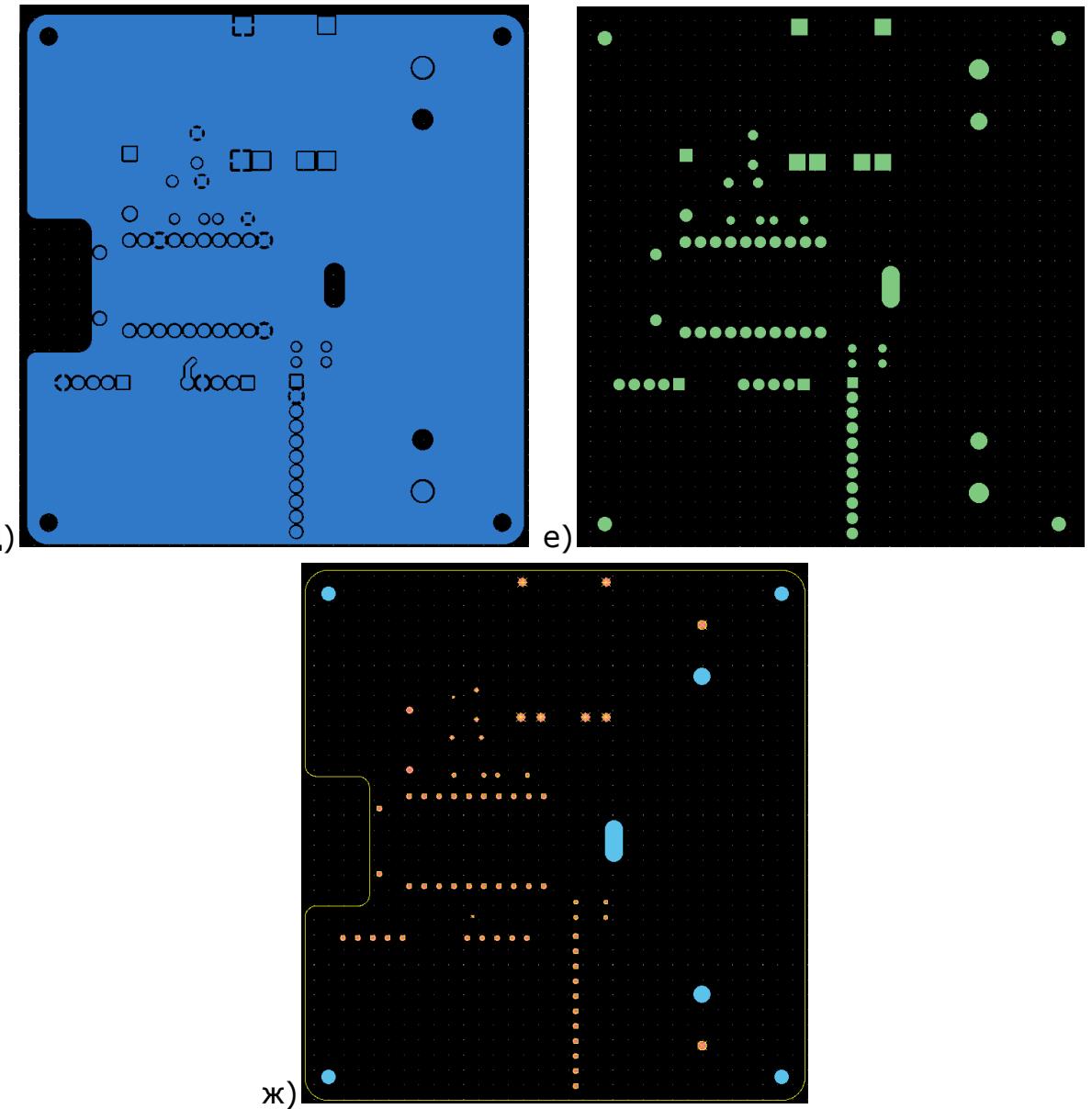
Фиг. 4.15. Изглед към симулиран 3D модел на печатната платка на Водещия възел

На фиг. 4.15. е показан изглед към 3D модел на печатната платка на Водещия възел. Единствено компонентът NODEMCU 32S няма точен 3D модел, но ситопечатът на платката показва истинските размери на модула.

4.4. Печатна платка на Основен възел

4.4.1. Слоеве на печатната платка на Основния възел





Фиг. 4.16.а) F_Silkscreen, б) F.Mask, в) F.Paste, г) F.Cu, д) B.Cu, е) B.Masks, ж) Edge.cuts + NPTH + PTH

Ситопечат на горния слой (F_Silkscreen)

Изглед към ситопечата на горния слой на Основния възел е представен на фиг. 4.16.а).

Защитна маска на горния слой (F.Mask)

Изглед към горния слой за защитна маска на Основния възел е представен на фиг. 4.16.б).

Горен слой за паста за запояване (F.Paste)

Изглед към горния слой за паста за запояване на Основния възел е показан на фиг. 4.16.в).

Горен слой за опроводяване (F.Cu)

Изглед към горния слой за опроводяване на Основния възел е показан на фиг. 4.16.г).

Долен слой за опроводяване (B.Cu)

Изглед към долнния слой за опроводяване на Основния възел е показан на фиг. 4.16.д).

Долен слой за паста за запояване (B.Paste) - Няма

Защитна маска на долнния слой (B.Masks)

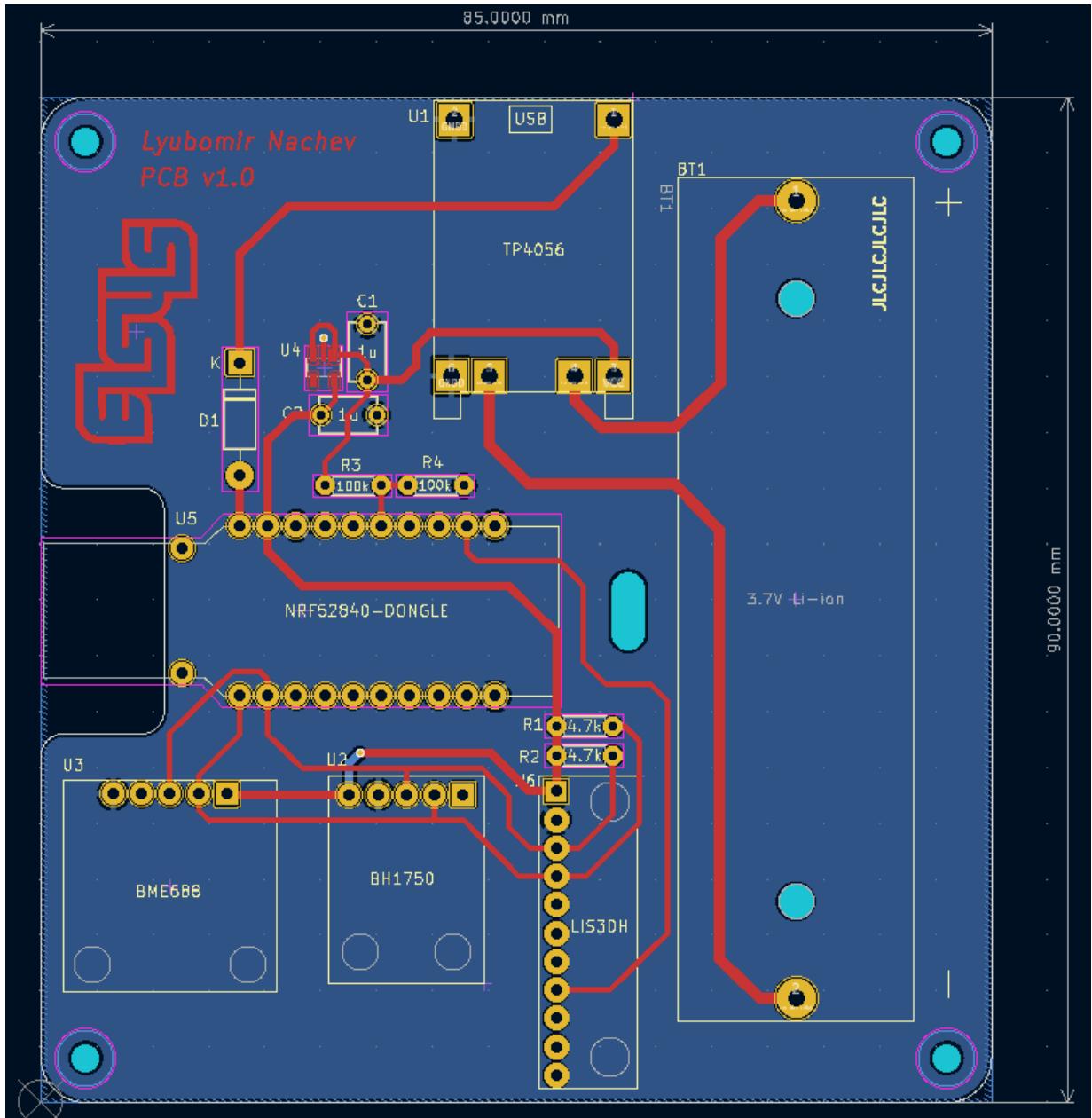
Изглед към долния слой за защитна маска на Основния възел е показан на фиг. 4.16.е).

Ситопечат на долния слой (B_Silkscreen) - Няма

Слой за изрязване комбиниран с drill файловете (Edge.cuts + NPTH + PTH)

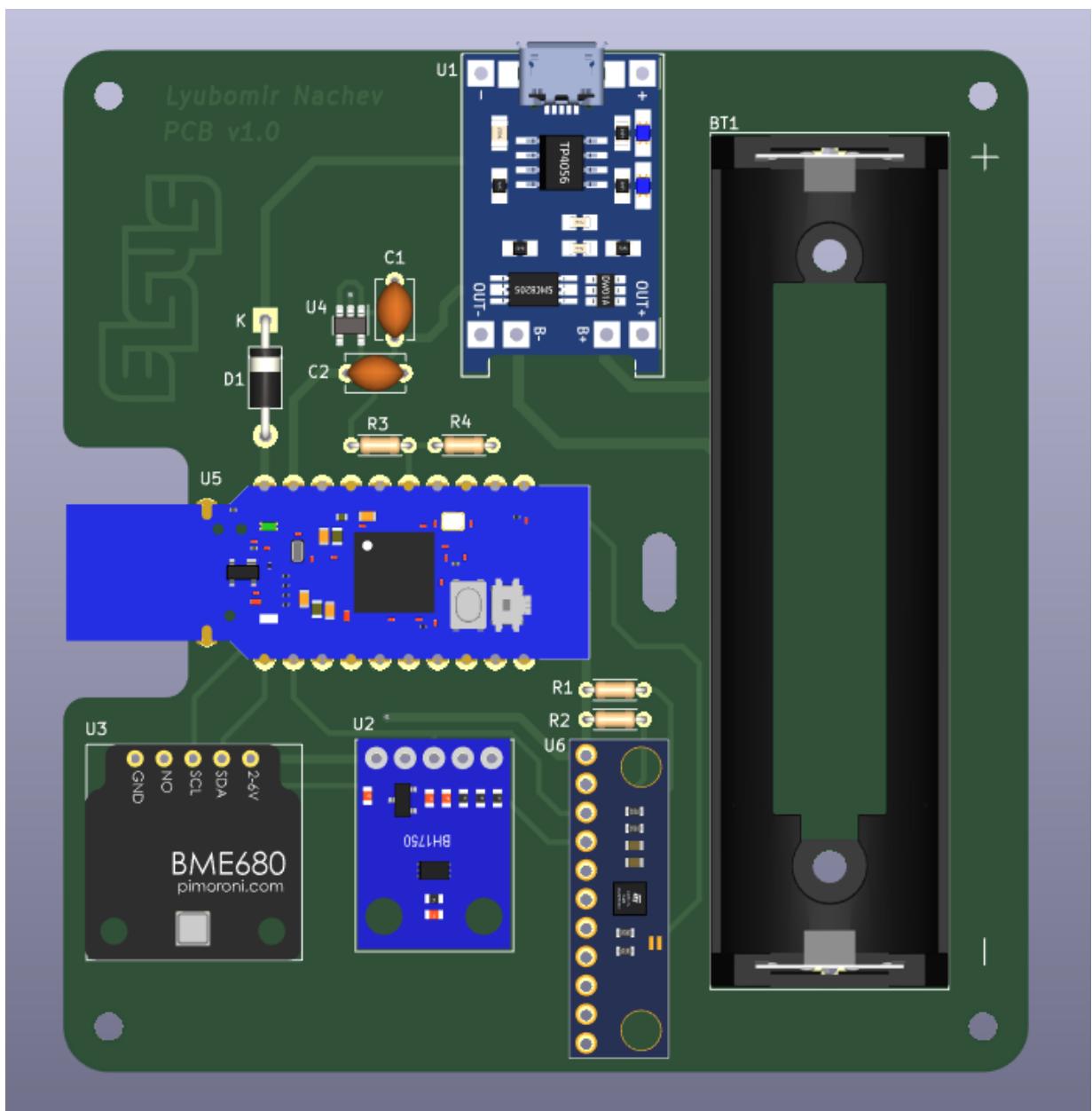
Изглед към механичните слоеве за пробиване/изрязване на печатната платка на Основния възел са показани на фиг. 4.16.ж).

4.4.2. Финален вариант на печатната платка



Фиг. 4.17. Цялостен изглед към печатната платка на Основния възел

4.4.3. 3D модел на печатната платка

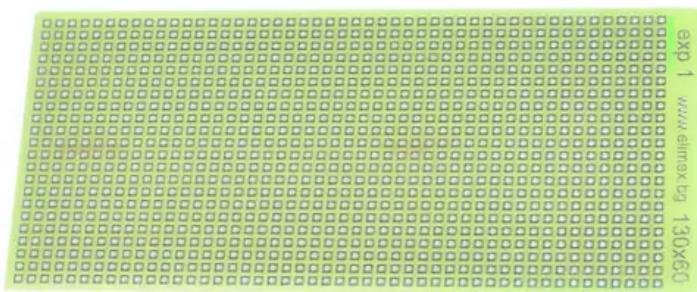


Фиг. 4.18. Изглед към симулиран 3D модел на печатната платка на Основния възел

4.5. Поръчка на печатните платки

4.5.1. Поръчка на печатната платка на Водещия възел

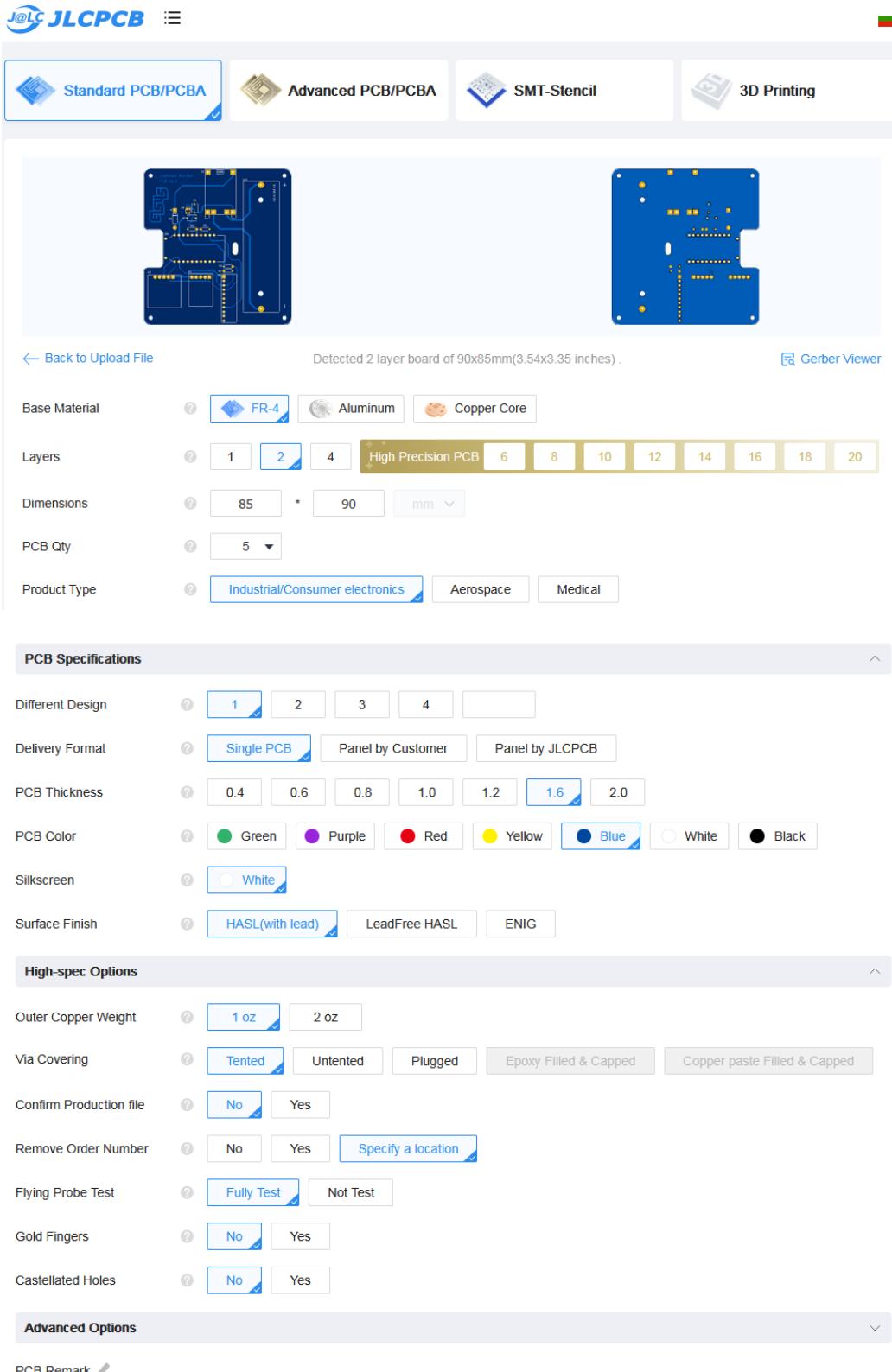
Поради наличието на само три компонента на Водещия възел бе взето решение печатната платка да бъде направена на експериментална платка. На фиг. 4.19. е показан изглед към експерименталната платка, която ще бъде използвана за Водещия възел [75].



Фиг. 4.19. Изглед към експериментална платка за запояване

4.5.2. Поръчка на печатната платка на Основния възел

Поради наличието на повече компоненти на Основния възел ще бъде поръчана печатна платка от компанията JLCPCB, която се намира в Китай. За да бъде направена поръчката трябва да се отиде на тяхната страница и да се качат в комбиниран .zip файл всички gerber и drill файлове на печатната платка [76]. На фиг. 4.20. е представен диалоговия прозорец за поръчка с нанесените изисквания [77].



Фиг. 4.20. Изглед към диалоговия прозорец за поръчка на страницата на JLCPCB

Пета глава - Проектиране на алгоритми и сурс код на управляващ софтуер

5.1. Използвани програми, инструменти и развойна среда

За разработването на сурс код на проекта ще са необходими следните софтуерни инструменти:

GitHub/Git

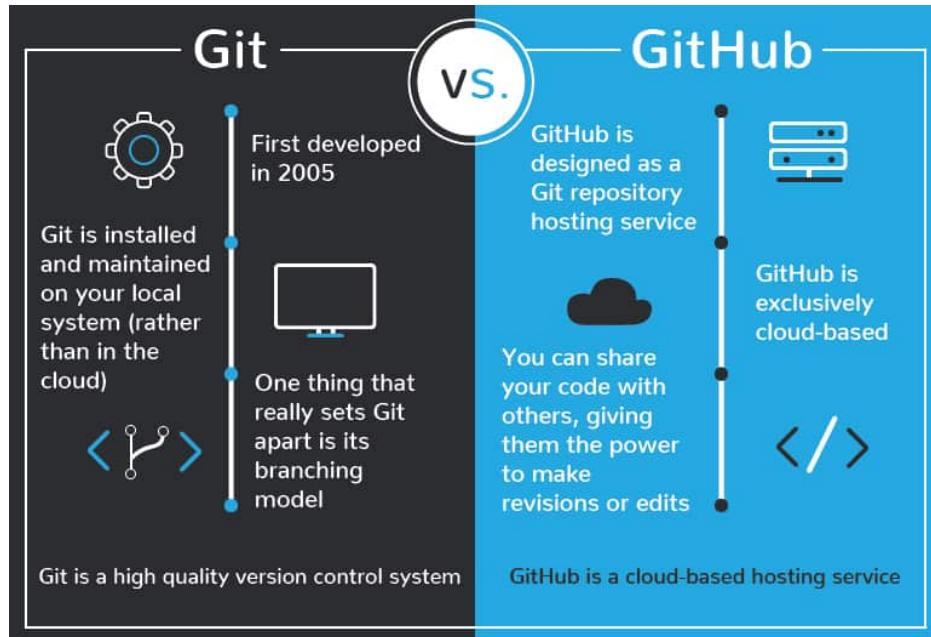
Git е популярна система за контрол на версията с отворен код, която се инсталира и управлява на локалното устройство. Отличаваща функция на git е неговият *branching* модел, който позволява създаване на независими локални клонове на кода. Това позволява да бъдат запазвани работещи версии на проекта докато паралелно се разработват нови функционалности на проекта.



Github, от друга страна, репрезентира онлайн база данни, която позволява свързването и запазването на вече съществуващите git хранилища към облака и тяхното управление през git [78]. На фиг. 5.1. са демонстрирани основните разлики между двата софтуера.

Github и Git ще бъдат използвани за създаването на backup версии на кода, които показват кога и какво е било променяно, улеснявайки откриването и премахването на грешки, както и възстановяването на определена версия.

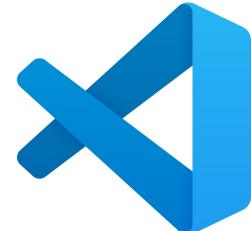
Линка към GitHub хранилището на дипломната работа може да се намери на следния линк - [79].



Фиг. 5.1. Сравнение на Git и GitHub

Visual Studio Code

За улесняването на разработката на какъвто и да е софтуер е необходима развойна среда, в която да може лесно да бъдат засечени грешки и да предоставя полезни инструменти.



Visual Studio Code (съкр. VS Code) е бесплатна, ниско ресурсна, но мощна, развойна среда на Microsoft за редактиране на код, която вгражда в себе си богата библиотека от популярни и поддържани разширения. Освен това тя има вграден контрол на версии на кода и много други възможности [80]. VS Code има голяма общност от разработчици около себе си.

Развойната среда е необходима за по-нататъшна инсталацията на разширения (extensions) като nRF Connect for VS Code и Espressif IDF, които ще бъдат използвани за разработка на сурс код за, избраните в трета глава, управляващи модули 3.2 и 3.3.

5.1.1. Софтуерни рамки (на англ. *Frameworks*)

nRF Connect SDK

nRF Connect SDK е универсален комплект за разработка на софтуер и създаване на продукти, основаващи се на всички безжични устройства, използващи интегрални схеми от сериите nRF52, nRF53 и nRF91. Този софтуерен комплект предлага на разработчиците обширна рамка за изграждане на софтуер с оптимизиран размер за устройства с ограничена памет, както и на мощен и комплексен софтуер за по-усъвършенствани устройства. nRF SDK интегрира Zephyr RTOS (на англ. *Real-time operating system*, на бълг. *Операционна система в реално време*) и широк набор от примери, приложни протоколи, протоколни стекове, библиотеки и хардуерни драйвери [81].



Той ще бъде използван за разработването на управляващ софтуер за nRF52840-Dongle.

ESP-IDF SDK

ESP-IDF (на англ. *Espressif's IoT Development Framework*) е създаден за серията SoC ESP32 на Espressif. Той представлява пълнофункционална рамка за разработка на софтуер, изградена върху ядрото на FreeRTOS.



Този SDK е open-source и има, както голямо количество документация, така и голяма общност около себе си, позволявайки улеснена работа с него [82]. Той ще бъде използван за разработването на управляващ софтуер за NodeMCU-32S.

5.1.2. Операционни системи в реално време (RTOS)

Zephyr

Zephyr RTOS е open-source RTOS, направена за устройства с ограничени ресурси. Той поддържа различни архитектури и е направен с идеята за безопасност и сигурност. Zephyr позволява лесна интеграция и конфигурация. Той има голяма общност около себе си, както и обилна документация. Има над 400 микроконтролерни платки, които поддържат Zephyr и 100+ сензора, които са вече интегрирани в операционната система [83].



Zephyr ще бъде използван в комбинация с nRF Connect SDK за улесняване на разработката на управляващ софтуер за nRF52840-Dongle.

FreeRTOS

FreeRTOS е водещата на пазара операционна система за реално време. Тя се е превърнала в стандартно решение за микроконтролери и малки микропроцесори с ограничени ресурси. Тя се явява като надеждна и поддържана операционна система, която е свободна за вграждане в комерсиални продукти. FreeRTOS поддържа повече от 40 архитектури и има огромна общност около себе си [84].

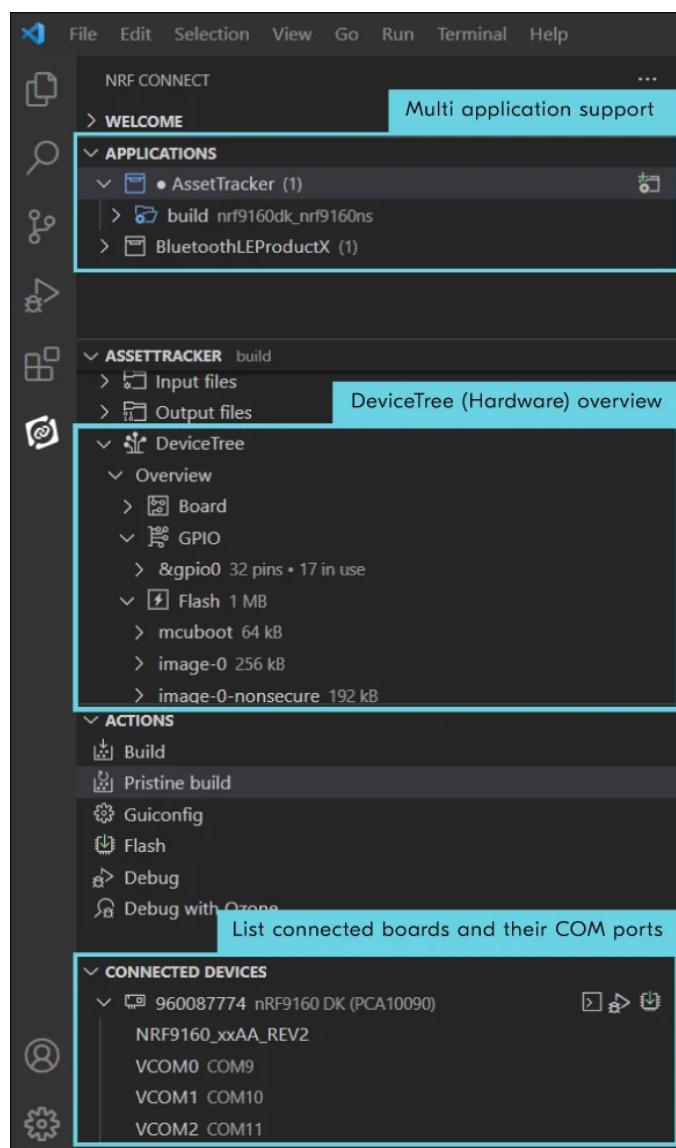


FreeRTOS ще бъде използван в комбинация с ESP-IDF за улесняване на разработката на управляващ софтуер за NodeMCU-32S.

5.1.3. Разширения

nRF Connect for VS Code

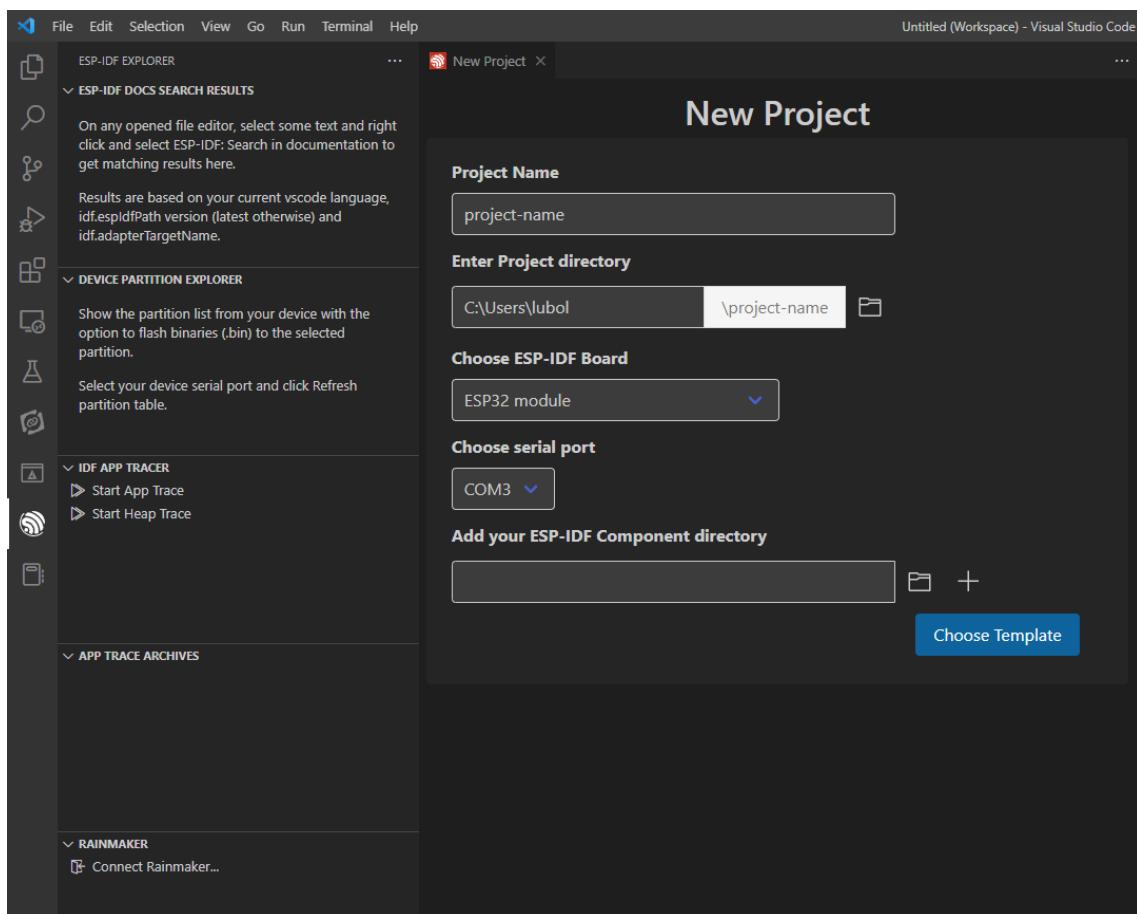
Nordic са създали разширение за VS Code, което интегрира в себе си всички необходими инструменти за удобна работа с nRF Connect SDK, позволявайки централизирана разработка и лесно управление на микроконтролера с вградена конзола, flash функция и т.н. [85]. На фиг. 5.2. е показан интерфейса на разширението.



Фиг. 5.2. Изглед към интерфейса на разширението nRF Connect

Espressif IDF

Разширението за Visual Studio Code ESP-IDF е мощен инструмент за разработчици, работещи с микроконтролера ESP32. Това е разширение с отворен код, което осигурява интуитивен и опростен интерфейс за изграждане, отстраняване на грешки и внедряване на приложения към ESP32. То включва редица полезни функции като автоматично попълване на код и поддръжка на отстраняване на синтактични грешки, които могат да помогнат за рационализиране на работните процеси при разработката [86]. На фиг. 5.3. е показан изглед към интерфейса му.



Фиг. 5.3. Изглед към интерфейса на Espressif IDF разширението

5.2. Избор на подходящ мрежови протокол

Избраният протокол за дипломната работа е Thread, и по-конкретно неговата модификация, наречена OpenThread.

Аргументация за избора на OpenThread и подробно описание на ключовите функционалности на Thread е направено в следващите подточки.

5.2.1. Openthread

Openthread е интерпретация на Thread с отворен код, разработен от компанията Google. Google публикува OpenThread, за да направи мрежова технология, използвана в техните продукти Google Nest, която да е по-широко достъпна за разработчиците, с цел ускоряване на разработката на "умни" продукти за домове и търговски сгради.

OpenThread имплементира всички мрежови слоеве на Thread (IPv6, 6LoWPAN, IEEE 802.15.4 с MAC сигурност, Mesh Link установяване, Mesh маршрутизиране) и роли на устройствата, както и поддръжка на Border Router.

OpenThread притежава цялата функционалност на Thread, допълнена с подробна документация, отворени библиотеки и готови примери. По този начин тя се явява удобна за разработката на проекта [87].

5.2.2. Thread оптимизация на IEEE 802.15.4

Спецификацията IEEE 802.15.4 е стандарт за безжична комуникация, който определя MAC и PHY слоевете, работещи със скорост 250kbps в честотна лента 2.4GHz. Проектиран с оглед на ниска консумация на енергия, 802.15.4 е подходящ за приложения, които обикновено включват голям брой възли. Thread оптимизира IEEE 802.15.4 по следния начин [88]:

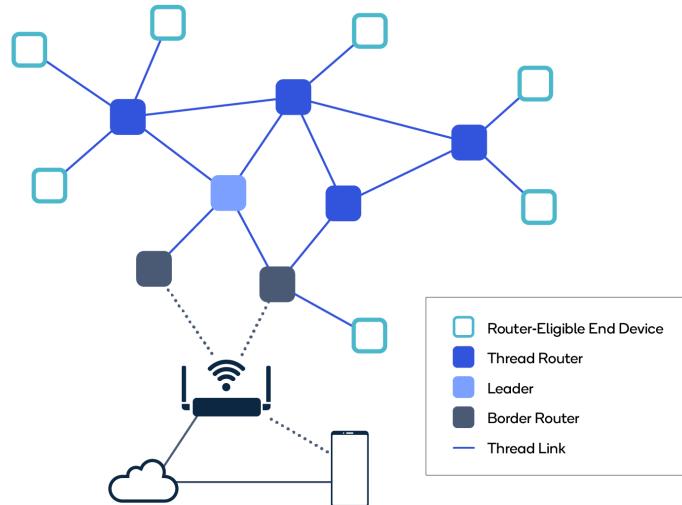
- Подобрява живота на батерията и отзивчивостта на заспало крайно устройство (англ. Sleepy end device, съкр. SED), като намалява броя на съобщения, които SED може да изпрати по въздуха.
- Намалява количеството трафик, необходимо за поддържане на връзка между SED и устройство родител, като интерпретира всички съобщения като keepalive съобщения.
- Позволява по-добра синхронизация между SED и родител чрез планиране на синхронизирани периоди на предаване/приемане без периодични заявки за данни.

5.2.3. Thread

Основните функции на Thread включват:

- Опростеност - лесна инсталация, стартиране и работа
- Сигурност - Всички устройства в мрежата Thread се удостоверяват и всички комуникации се криптират.
 - Надеждност - Самолечение на mesh мрежата без единична точка на отказ и методи за разпространение на спектъра, които осигуряват устойчивост на смущения.
 - Ефективност - Устройствата Thread с ниска консумация на енергия могат да работят в режим на спане и да намалят разхода на енергия значително.
 - Машабируемост - Thread мрежите могат да се машабират до стотици устройства

- Устройствата в Thread



Фиг. 5.4. Топология на примерна Thread мрежа

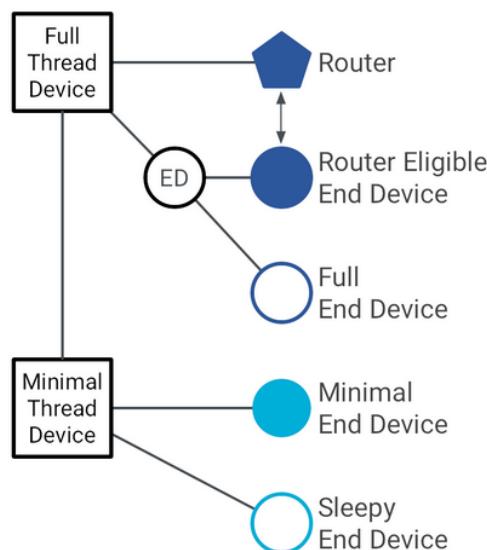
На фиг. 5.4. е показана топологията на примерна Thread мрежа. Устройствата в нея са следните [89]:

- **Border Router** - Границният маршрутизатор е устройство, което позволява препращането на информация между Thread мрежата и външната мрежа (Wi-Fi, Ethernet и т.н.).
- **Leader** - Във всяка Thread мрежа съществува едно устройство, което поема ролята на лидер. Лидерът винаги е само един и се избира автоматично. Той решава кои устройства да бъдат маршрутизатори и задава и управлява техните адреси, като използва СоАР за изпращане на маршрутизираща информация. Цялата информация, съдържаща се в лидера, съществува и в другите Thread маршрутизатори, като по този начин, ако лидерът изгуби връзка, автоматично се избира друг маршрутизатор, който да поеме неговата роля.
- **Thread Routers** - Thread маршрутизаторите предоставят услуги за маршрутизиране на мрежови устройства. Те, също така, предоставят услуги за присъединяване и сигурност на устройствата, които се опитват да се свържат в мрежата. Thread маршрутизаторите

са конфигурирани да не спят и могат, ако вече към тях не са свързани никакви крайни устройства, да се превърнат в REED (на англ. Router-Eligible End Devices).

- Router-Eligible End Devices - REEDs представляват крайни устройства, които могат да поемат ролята на Thread маршрутизатори или лидери при необходимост, в Thread мрежата. Те не препредават съобщения и не предоставят услуги за присъединяване на други устройства в мрежата. Процесът на конфигуриране на REED устройство като маршрутизатор е автоматичен и не изисква никакво взаимодействие с потребителя.

Устройствата в Thread също така се разделят на два основни типа - FTD (на англ. *Full Thread Device*) и MTD (на англ. *Minimal Thread Device*).



Фиг. 5.5. Всички възможни типове устройства в една Thread мрежа

На фиг. 5.5. са показани всички типове устройства, разделящи се главно на FTDs и MTDs.

Full Thread Device - Винаги има включен приемопредавател, абонира се за multicast адреса на всички маршрутизатори и поддържа IPv6 адресни връзки. Съществуват три вида FTDs:

- Thread маршрутизатор
- REED - може да бъде повишено на маршрутизатор
- FED (на англ. *Full End Device*) - не може да бъде повишено на маршрутизатор

Minimal Thread Device - Не се абонира за multicast адреса на всички маршрутизатори и препраща всички съобщения към своя родител (Thread маршрутизатора, към който е свързан). Работи само като крайно устройство. Съществуват два вида MTDs:

- MED (на англ. *Minimal End Device*) - приемопредавателят е винаги включен, не е необходимо да се допитва за съобщения от своя родител.
- SED - обикновено е изключено, събужда се от време на време, за да потърси съобщения от своя родител.

Мрежата има ограничение на устройствата показано в табл. 5.1.

Табл. 5.1. Максимален брой на устройствата

Роля	Лимит
Лидер	1
Маршрутизатор	32
Крайно устройство	511 на маршрутизатор

- Адресирането в Thread

В Thread мрежата има три обхвата за еднотосочно адресиране [90].

- Link-Local - всички интерфейси, които могат да бъдат достигнати с едно радиопредаване.
- Mesh-Local - всички интерфейси, достъпими в Thread мрежата.

- Global - всички интерфейси, достигими извън Thread мрежата Link-Local имат префикси от fe80::/16, докато Mesh-Local имат префикси от fd00::/8.

Unicast

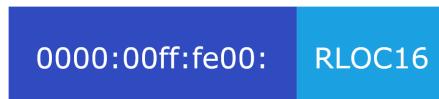
Routing Locator (съкр. RLOC) идентифицира Thread интерфейс въз основа на местоположението му в топологията на мрежата. Той е уникален за всяко устройство. Неговото образуване е разделено на няколко части:

- Генериране на RLOC16 - Това са последните 16 бита. За крайно устройство (child) RLOC16 се образува от RLOC16 на маршрутизатора (parent), който винаги завършва на нула. Процесът за калкулиране на RLOC16 на крайно устройство е показан на фиг. 5.6.



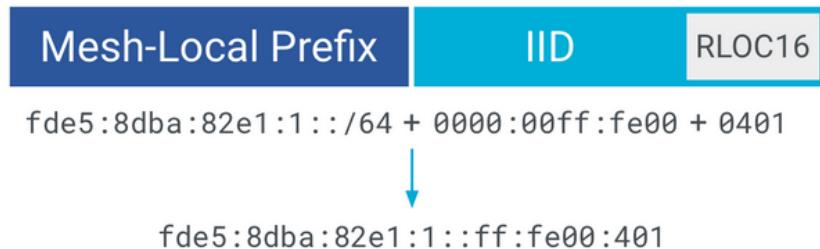
Фиг. 5.6. Калкулиране на RLOC16

- Генериране на IID (на англ. *Interface Identifier*) - Той съответства на последните 64 бита на IPv6 адреса. Процесът на неговото създаване е показан на фиг. 5.7.



Фиг. 5.7. Генериране на IID

- Създаване на RLOC - IID, комбинирано с Mesh-Local префиксът, създава крайния RLOC. Процесът е нагледно представен на фиг. 5.8.



Фиг. 5.8. Процес на генериране на RLOC

Важно е да се отбележи, че защото RLOC се базира на местоположението на възела в топологията, неговата стойност ще се промени, ако има промяна в топологията.

Друг вид IPv6 unicast адреси в Thread мрежата са Endpoint Identifiers (съкр. EIDs). Те идентифицират уникален Thread интерфейс в участък на мрежата. EIDs не зависят от топологията. Примери за такива адреси:

- Link-Local Address (съкр. LLA) - идентифицира интерфейса на нишката, достъгим с едно радиопредаване. Използва се за откриване на съседи, конфигуриране на връзки и обмен на маршрутизираща информация. Самият адрес не може да се маршрутизира. Има постоянна представка - fe80::/16
- Mesh-Local EID (съкр. ML-EID) - идентифицира интерфейс на възел независимо от топологията на мрежата. Използва се за достигане до Thread интерфейс в участък от мрежата. Нарича се също така ULA (на англ. Unique Local Address). Винаги има представка - fd00::/8.

Multicast

Multicast се използва за предаване на информация до множество устройства едновременно. В Thread мрежата определени адреси са запазени за използване като multicast и адресират различни групи устройства. Multicast адресите в Thread са показани на табл. 5.2.

Табл. 5.2. Multicast адресация

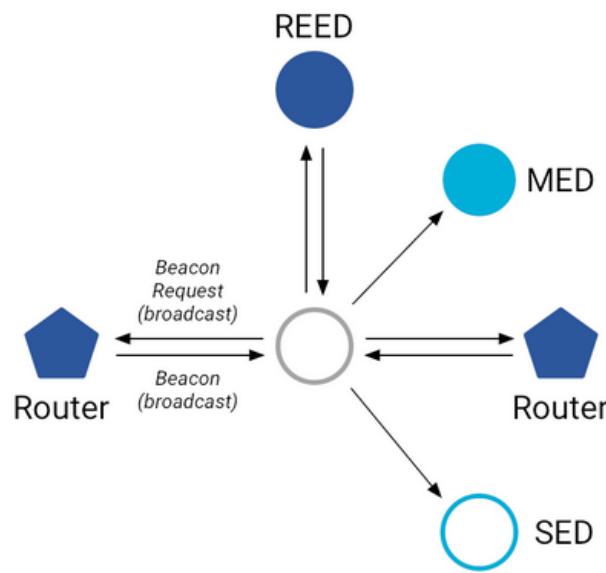
IPv6 Address	Scope	Delivered to
ff02::1	Link-Local	All FTDs and MEDs
ff02::2	Link-Local	All FTDs
ff03::1	Mesh-Local	All FTDs and MEDs
ff03::2	Mesh-Local	All FTDs

- Създаване на Thread мрежа

Thread мрежите се характеризират чрез три уникални идентификатора:

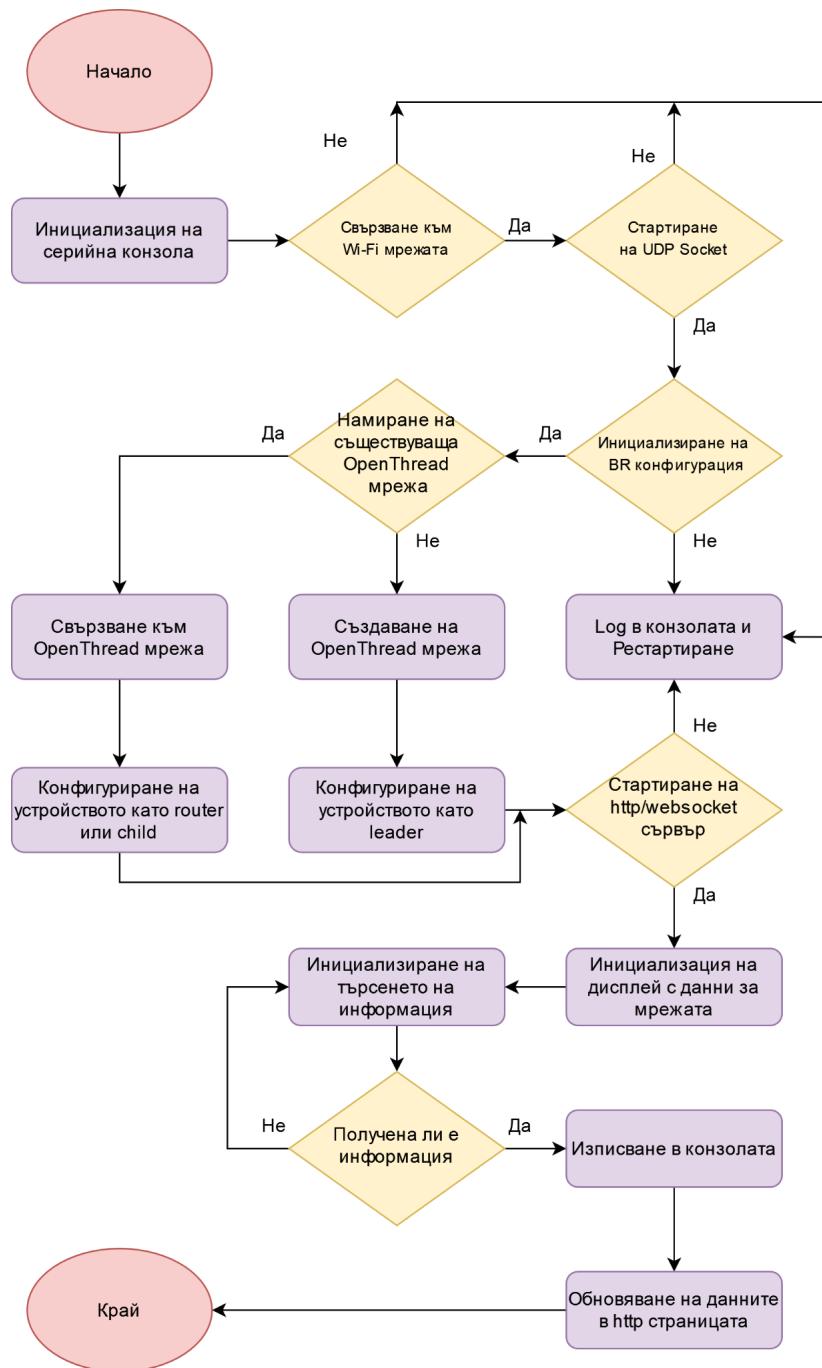
- Personal Area Network (съкр. PAN ID) - 2 байта (Примерно: 0xBEEF)
- Extended Personal Area Network ID (съкр. XPAN ID) - 8 байта (Примерно: 0xBEEF1111CAFE2222)
- Име на мрежата (Примерно: MyThreadNetwork)

При създаване на нова Thread мрежа или търсене на съществуваща такава, към която да се присъедини, Thread устройството извършва активно сканиране за 802.15.4 мрежи. Устройството изпраща 802.15.4 Beacon заявка на определен канал. Всички маршрутизатори или крайни устройства, които са в обхват, изпращат Beacon, който съдържа тяхното PAN ID, XPAN ID и Име на мрежата. Устройството повтаря процеса за всички останали канали и след като е открило всички съществуващи мрежи в обхват, то може да се прикачи към съществуваща мрежа или да създаде нова, ако не са открити такива [91]. Процесът е нагледно представен на фиг. 5.9.



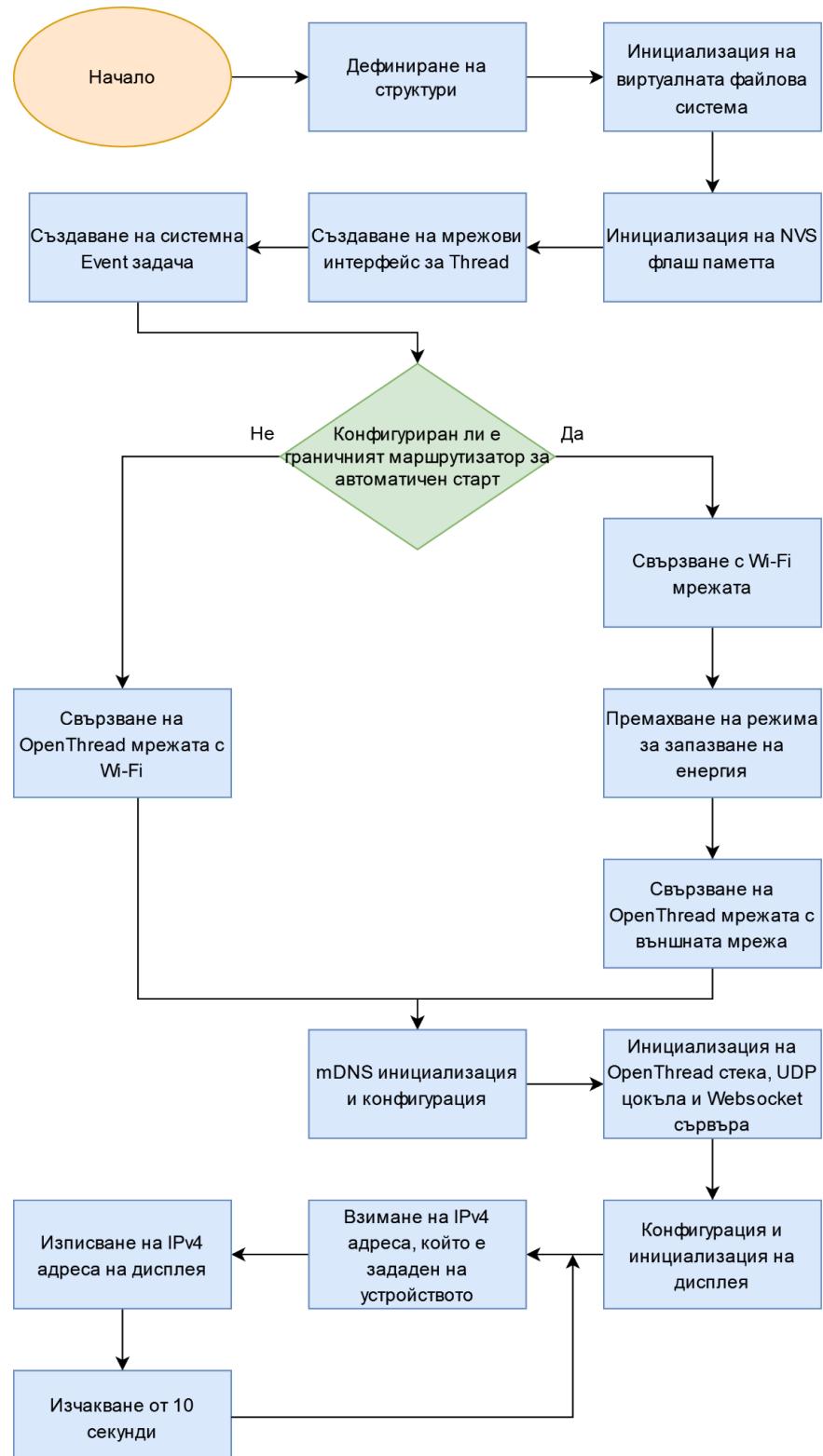
Фиг. 5.9. Процес на търсене на Thread мрежа

5.3. Функционална диаграма на работата на Водещия възел на ниво ОС (FreeRTOS)



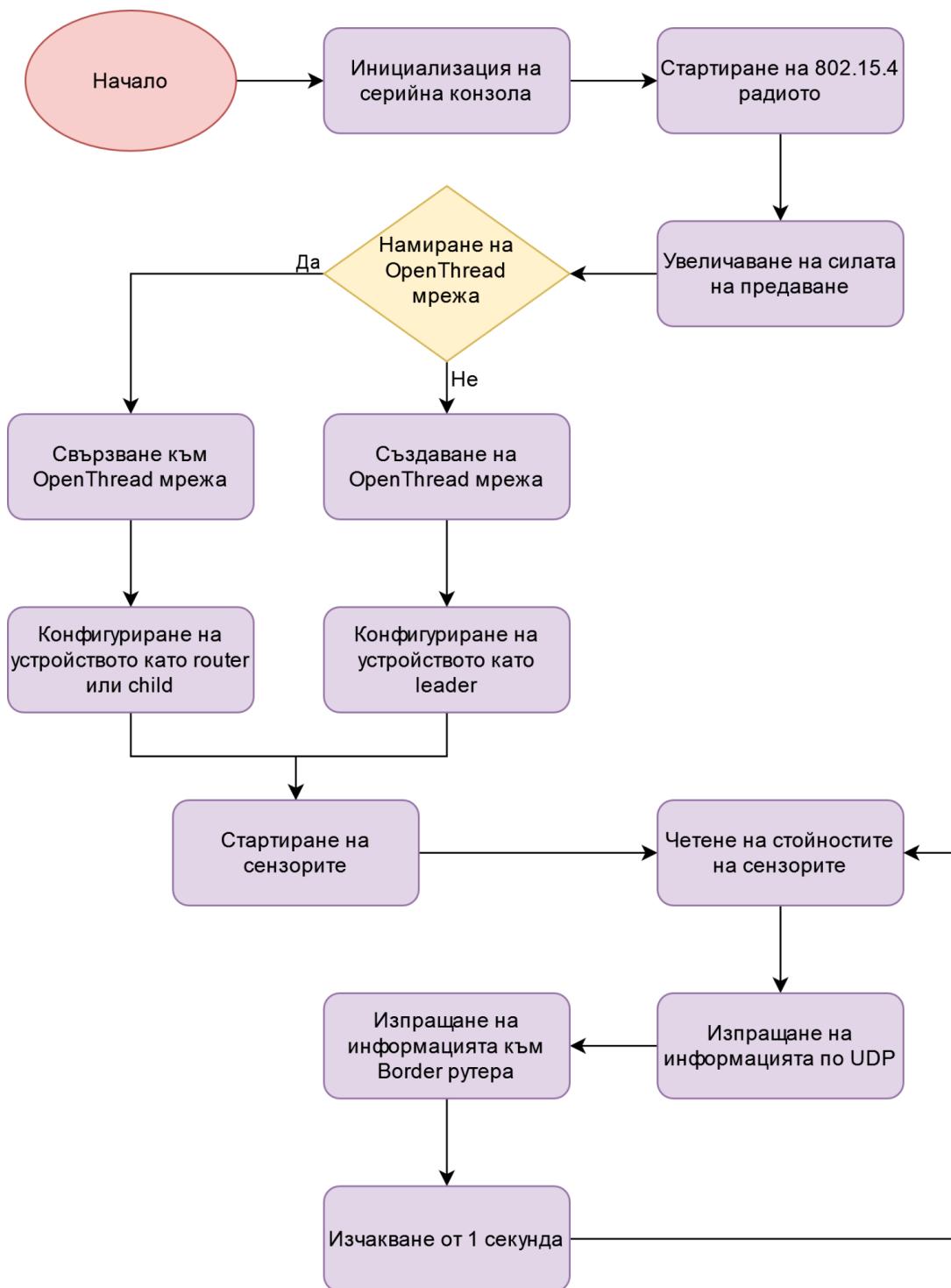
Фиг. 5.10. Функционална диаграма на работата на Водещия възел на ниво ОС

5.4. Блокова схема на алгоритъма на работа на Водещия възел



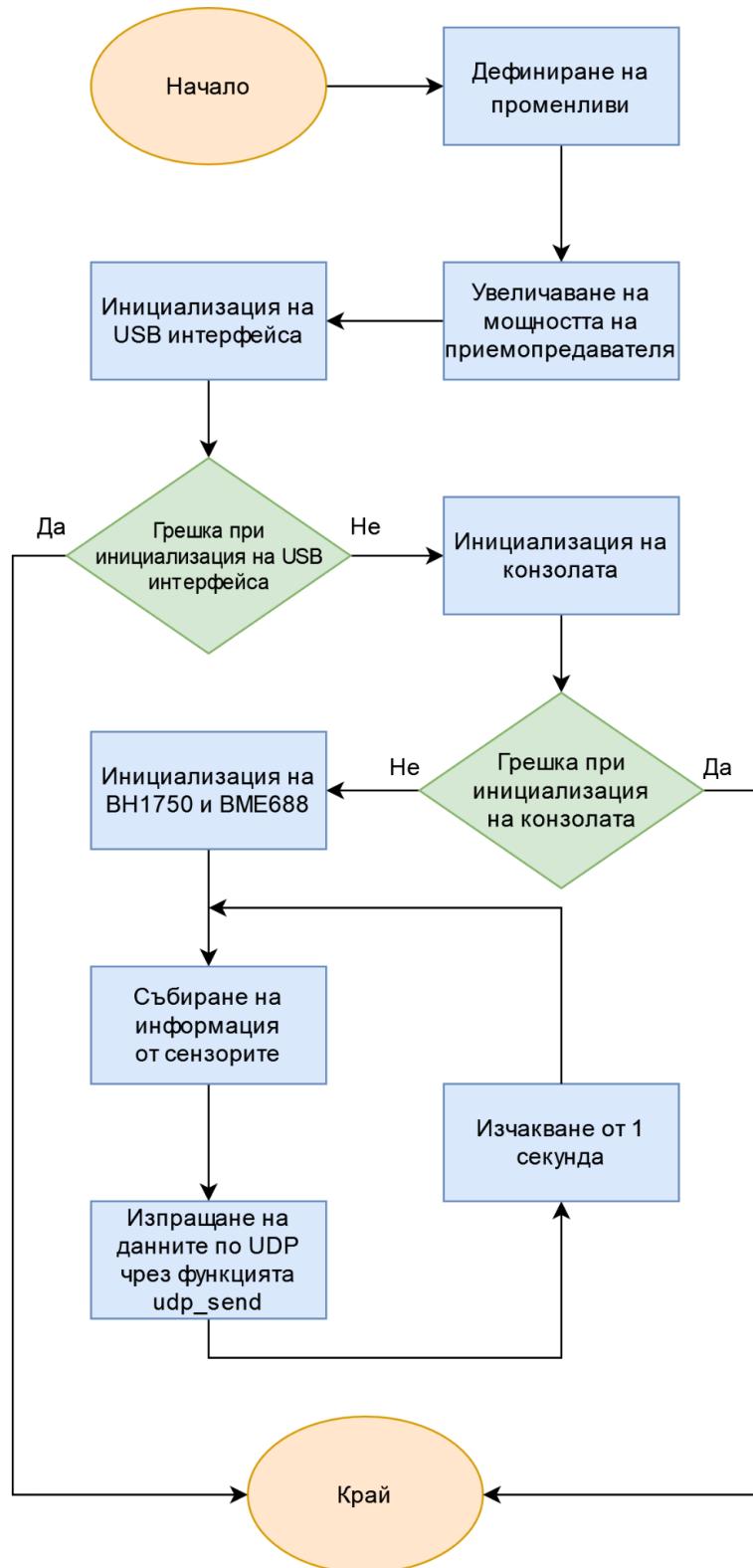
Фиг. 5.11. Блокова схема на алгоритъма на Водещия възел

5.5. Функционална диаграма на работата на Основния възел на ниво OS (Zephyr)



Фиг. 5.12. Функционална диаграма на работата на Основния възел на ниво OS

5.6. Блокова схема на алгоритъма на работа на Водещия възел



Фиг. 5.13. Блокова схема на алгоритъма на Основния възел

5.7. Сурс код за Водещия блок

Подробно обяснение на отделните функции на сурс кода е представено в шеста глава.

```
● ● ●  
1 // Standard C libraries  
2 #include <stdio.h>  
3 #include <stdlib.h>  
4 #include <string.h>  
5  
6 // Standard ESP, FreeRTOS, lwIP(Lightweight Ip) and OpenThread libraries  
7 #include "esp_check.h"  
8 #include "esp_err.h"  
9 #include "esp_event.h"  
10 #include "esp_log.h"  
11 #include "esp_netif.h"  
12 #include "esp_netif_ip_addr.h"  
13 #include "esp_netif_net_stack.h"  
14 #include "esp_openthread.h"  
15 #include "esp_openthread_border_router.h"  
16 #include "esp_openthread_cli.h"  
17 #include "esp_openthread_lock.h"  
18 #include "esp_openthread_netif_glue.h"  
19 #include "esp_openthread_types.h"  
20 #include "esp_ot_cli_extension.h"  
21 #include "esp_ot_config.h"  
22 #include "esp_ot_wifi_cmd.h"  
23 #include "esp_vfs_dev.h"  
24 #include "esp_vfs_eventfd.h"  
25 #include "esp_wifi.h"  
26 #include "mdns.h"  
27 #include "nvs_flash.h"  
28 #include "protocol_examples_common.h"  
29 #include "sdkconfig.h"  
30 #include "driver/uart.h"  
31 #include "freertos/FreeRTOS.h"  
32 #include "freertos/task.h"  
33 #include "hal/uart_types.h"  
34 #include "openthread/backbone_router_ftd.h"  
35 #include "openthread/border_router.h"  
36 #include "openthread/cli.h"  
37 #include "openthread/dataset.h"  
38 #include "openthread/dataset_ftd.h"  
39 #include "openthread/dataset_updater.h"  
40 #include "openthread/error.h"  
41 #include "openthread/instance.h"  
42 #include "openthread/ip6.h"  
43 #include "openthread/logging.h"  
44 #include "openthread/tasklet.h"  
45 #include "openthread/thread_ftd.h"  
46 #include "esp_ot_udp_socket.h"  
47 #include "lwip/err.h"  
48 #include "lwip/mld6.h"  
49 #include "lwip/sockets.h"  
50 #include "esp_eth.h"  
51 #include "esp_system.h"  
52 #include "spi_flash_mmap.h"  
53 #include <esp_http_server.h>  
54 #include "freertos/event_groups.h"  
55 #include "driver/gpio.h"  
56 #include <lwip/sockets.h>  
57 #include <lwip/sys.h>  
58 #include <lwip/api.h>  
59 #include <lwip/netdb.h>  
60 #include <sys/param.h>  
61 #include "ssd1306.h"
```

Фиг. 5.14. Добавяне на библиотеки

```
1 #define TAG "esp_ot_br"
2
3 // Enable the use of these functions if the border router is configured with auto start
4 #if CONFIG_OPENTHREAD_BR_AUTO_START
5
6 // Convert hexadecimal digit to an integer
7 static int hex_digit_to_int(char hex)
8 {
9     if ('A' <= hex && hex <= 'F') {
10         return 10 + hex - 'A';
11     }
12     if ('a' <= hex && hex <= 'f') {
13         return 10 + hex - 'a';
14     }
15     if ('0' <= hex && hex <= '9') {
16         return hex - '0';
17     }
18     return -1;
19 }
20
21 // Convert hexadecimal string to binary data
22 static size_t hex_string_to_binary(const char *hex_string, uint8_t *buf, size_t buf_size)
23 {
24     int num_char = strlen(hex_string);
25
26     if (num_char != buf_size * 2) {
27         return 0;
28     }
29     for (size_t i = 0; i < num_char; i += 2) {
30         int digit0 = hex_digit_to_int(hex_string[i]);
31         int digit1 = hex_digit_to_int(hex_string[i + 1]);
32
33         if (digit0 < 0 || digit1 < 0) {
34             return 0;
35         }
36         buf[i / 2] = (digit0 << 4) + digit1;
37     }
38     return buf_size;
39 }
```

Фиг. 5.15. Функциите `hex_digit_to_int` и `hex_string_to_binary`

```

● ● ●

1 // Configure the OpenThread network
2 static void create_config_network(otInstance *instance)
3 {
4     otOperationalDataset dataset;
5
6     if (otDatasetGetActive(instance, &dataset) == OT_ERROR_NONE) {
7         ESP_LOGI(TAG, "Already has network, skip configuring OpenThread network.");
8         return;
9     }
10
11    uint16_t network_name_len = strlen(CONFIG_OPENTHREAD_NETWORK_NAME);
12
13    assert(network_name_len <= OT_NETWORK_NAME_MAX_SIZE);
14
15    if (otDatasetCreateNewNetwork(instance, &dataset) != OT_ERROR_NONE) {
16        ESP_LOGE(TAG, "Failed to create OpenThread network dataset.");
17        abort();
18    }
19
20    dataset.mChannel = CONFIG_OPENTHREAD_NETWORK_CHANNEL;
21    dataset.mComponents.mIsChannelPresent = true;
22    dataset.mPanId = CONFIG_OPENTHREAD_NETWORK_PANID;
23    dataset.mComponents.mIsPanIdPresent = true;
24    memcpy(dataset.mNetworkName.m8, CONFIG_OPENTHREAD_NETWORK_NAME, network_name_len);
25    dataset.mComponents.mIsNetworkNamePresent = true;
26
27    if (hex_string_to_binary(CONFIG_OPENTHREAD_NETWORK_EXTPANID, dataset.mExtendedPanId.m8,
28                            sizeof(dataset.mExtendedPanId.m8)) != sizeof(dataset.mExtendedPanId.m8)) {
29        ESP_LOGE(TAG, "Cannot convert OpenThread extended pan id. Please double-check your config.");
30        abort();
31    }
32    dataset.mComponents.mIsExtendedPanIdPresent = true;
33    if (hex_string_to_binary(CONFIG_OPENTHREAD_NETWORK_MASTERKEY, dataset.mNetworkKey.m8,
34                            sizeof(dataset.mNetworkKey.m8)) != sizeof(dataset.mNetworkKey.m8)) {
35        ESP_LOGE(TAG, "Cannot convert OpenThread master key. Please double-check your config.");
36        abort();
37    }
38    dataset.mComponents.mIsNetworkKeyPresent = true;
39    if (hex_string_to_binary(CONFIG_OPENTHREAD_NETWORK_PSKC, dataset.mPskc.m8, sizeof(dataset.mPskc.m8)) !=
40        sizeof(dataset.mPskc.m8)) {
41        ESP_LOGE(TAG, "Cannot convert OpenThread pre-shared commissioner key. Please double-check your config.");
42        abort();
43    }
44    dataset.mComponents.mIsPskcPresent = true;
45    if (otDatasetSetActive(instance, &dataset) != OT_ERROR_NONE) {
46        ESP_LOGE(TAG, "Failed to set OpenThread active dataset.");
47        abort();
48    }
49    return;
50 }
51
52 // Start the OpenThread network
53 static void launch_openthread_network(otInstance *instance)
54 {
55     if (otIp6SetEnabled(instance, true) != OT_ERROR_NONE) {
56         ESP_LOGE(TAG, "Failed to enable OpenThread IPv6 link");
57         abort();
58     }
59     if (otThreadSetEnabled(instance, true) != OT_ERROR_NONE) {
60         ESP_LOGE(TAG, "Failed to enable OpenThread");
61         abort();
62     }
63     if (otBorderRouterRegister(instance) != OT_ERROR_NONE) {
64         ESP_LOGE(TAG, "Failed to register border router.");
65         abort();
66     }
67     otBackboneRouterSetEnabled(instance, true);
68 }
69 #endif // CONFIG_OPENTHREAD_BR_AUTO_START

```

Фиг. 5.16. Функциите create_config_network и launch_openthread_network

```
● ○ ● ●  
1 // Initialize and run the OpenThread stack  
2 static void ot_task_worker(void *aContext)  
3 {  
4     esp_openthread_platform_config_t config = {  
5         .radio_config = ESP_OPENTHREAD_DEFAULT_RADIO_CONFIG(),  
6         .host_config = ESP_OPENTHREAD_DEFAULT_HOST_CONFIG(),  
7         .port_config = ESP_OPENTHREAD_DEFAULT_PORT_CONFIG(),  
8     };  
9  
10    esp_netif_config_t cfg = ESP_NETIF_DEFAULT_OPENTHREAD();  
11    esp_netif_t *openthread_netif = esp_netif_new(&cfg);  
12    assert(openthread_netif != NULL);  
13  
14    // Initialize the OpenThread stack  
15    ESP_ERROR_CHECK(esp_openthread_init(&config));  
16  
17    // Initialize border routing features  
18    esp_openthread_lock_acquire(portMAX_DELAY);  
19    ESP_ERROR_CHECK(esp_netif_attach(openthread_netif, esp_openthread_netif_glue_init(&config)));  
20  
21    (void)otLoggingSetLevel(CONFIG_LOG_DEFAULT_LEVEL);  
22    esp_openthread_cli_init();  
23  
24 #if CONFIG_OPENTHREAD_BR_AUTO_START  
25     ESP_ERROR_CHECK(esp_openthread_border_router_init());  
26     create_config_network(esp_openthread_get_instance());  
27     launch_openthread_network(esp_openthread_get_instance());  
28 #endif // CONFIG_OPENTHREAD_BR_AUTO_START  
29     esp_cli_custom_command_init();  
30     esp_openthread_lock_release();  
31  
32     // Run the main loop  
33     esp_openthread_cli_create_task();  
34     esp_openthread_launch_mainloop();  
35  
36     // Clean up  
37     esp_netif_destroy(openthread_netif);  
38     esp_openthread_netif_glue_deinit();  
39     esp_vfs_eventfd_unregister();  
40     vTaskDelete(NULL);  
41 }
```

Фиг. 5.17. Функцията *ot_task_worker*

```

1 char rx_buffer[128];
2 int len;
3
4 //Initialize the UDP socket server
5 static void udp_socket_server_task(void *pvParameters)
6 {
7     int err = 0;
8     esp_err_t ret = ESP_OK;
9     char addr_str[128];
10    int listen_sock;
11    int port = CONFIG_OPENTHREAD_CLI_UDP_SERVER_PORT;
12
13    struct timeval timeout = {0};
14    struct sockaddr_storage source_addr;
15    struct sockaddr_in6 listen_addr;
16
17    inet6_aton("::", &listen_addr.sin6_addr); // Convert the unspecified address from string to binary number
18    listen_addr.sin6_family = AF_INET6;
19    listen_addr.sin6_port = htons(port);
20
21    listen_sock = socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDP); // socket(domain,type,protocol)
22    ESP_GOTO_ON_FALSE((listen_sock >= 0), ESP_OK, exit, TAG, "Unable to create socket: errno %d", errno);
23    ESP_LOGI(TAG, "Socket created");
24
25    int opt = 1;
26
27    // Allow the socket to bind to an address that is already in use
28    setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
29
30    // Disable the dual-stack setting and use only IPv6
31    setsockopt(listen_sock, IPPROTO_IPV6, IPV6_V6ONLY, &opt, sizeof(opt));
32
33    err = bind(listen_sock, (struct sockaddr *)&listen_addr, sizeof(listen_addr));
34    ESP_GOTO_ON_FALSE((err == 0), ESP_FAIL, exit, TAG, "Socket unable to bind: errno %d", errno);
35    ESP_LOGI(TAG, "Socket bound, port %d", port);
36    ESP_LOGI(TAG, "Waiting for data, no timeout");
37
38    while(1){
39        // Configure the timeout for receiving data to 0 seconds
40        timeout.tv_sec = 0;
41        setsockopt(listen_sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
42
43        socklen_t socklen = sizeof(source_addr);
44
45        // Get the length of the received data and store it in rx_buffer
46        len = recvfrom(listen_sock, rx_buffer, sizeof(rx_buffer) - 1, 0, (struct sockaddr *)&source_addr, &socklen);
47
48        // Error occurred during receiving
49        ESP_GOTO_ON_FALSE((len >= 0), ESP_FAIL, exit, TAG, "recvfrom failed: errno %d", errno);
50
51        // Get the sender's ip address as string
52        inet6_ntoa_r(((struct sockaddr_in6 *)&source_addr)->sin6_addr, addr_str, sizeof(addr_str) - 1);
53
54        rx_buffer[len] = 0; // Null-terminate whatever we received and treat as a string
55        ESP_LOGI(TAG, "Received %d bytes from %s:", len, addr_str);
56        ESP_LOGI(TAG, "%s", rx_buffer);
57    }
58    exit: // On error
59    if (ret != ESP_OK) {
60        shutdown(listen_sock, 0);
61        close(listen_sock);
62    }
63    ESP_LOGI(TAG, "Socket server is closed.");
64    vTaskDelete(NULL);
65 }

```

Фиг. 5.18. Функцията `udp_socket_server_task`

```

1  /* Websocket functions */
2
3 // Asynchronous response data structure
4 struct async_resp_arg {
5     httpd_handle_t hd; // Server instance
6     int fd;           // Session socket file descriptor
7 };
8
9 // Send an HTTP response asynchronously
10 static void ws_async_resp(void *arg)
11 {
12     char http_str[250];
13     char *data_str = rx_buffer; // Get the received UDP data from OpenThread devices
14     sprintf(http_str, "HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\n", strlen(data_str)); // HTTP Response string
15
16 //Initialize the async_resp_arg data structure
17 struct async_resp_arg *resp_arg = (struct async_resp_arg *)arg;
18 httpd_handle_t hd = resp_arg->hd;
19 int fd = resp_arg->fd;
20
21 // Send data to the client
22 ESP_LOGI(TAG, "Executing queued work fd: %d", fd);
23 httpd_socket_send(hd, fd, http_str, strlen(http_str), 0);
24 httpd_socket_send(hd, fd, data_str, strlen(data_str), 0);
25
26 free(arg);
27 }
28
29 // Handle HTTP GET requests asynchronously (enable full duplex communication)
30 static esp_err_t async_get_handler(httpd_req_t *req)
31 {
32     // Allocate memory for the async_resp_arg data structure
33     struct async_resp_arg *resp_arg = malloc(sizeof(struct async_resp_arg));
34
35     // Define the parameters
36     resp_arg->hd = req->handle;
37     resp_arg->fd = httpd_req_to_sockfd(req);
38
39     // Start the queue with the ws_async_resp handler
40     httpd_queue_work(req->handle, ws_async_resp, resp_arg);
41     return ESP_OK;
42 }
43
44 // Define the '/ws' URI for the server that is added at the end of the IP
45 static const httpd_uri_t ws = {
46     .uri      = "/ws",
47     .method   = HTTP_GET,
48     .handler  = async_get_handler, // Start the async_get_handler
49     .user_ctx = NULL,
50     .is_websocket = true
51 };
52
53 // Start the web server
54 static void start_webserver(void)
55 {
56     httpd_handle_t server = NULL;
57     httpd_config_t config = HTTPD_DEFAULT_CONFIG();
58     ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port);
59     httpd_start(&server, &config);
60     httpd_register_uri_handler(server, &ws);
61 }
62
63 /* End of Websocket functions */

```

Фиг. 5.19. Функциите за конфигуриране на Websocket

```

1 void app_main(void)
2 {
3     esp_netif_ip_info_t ip_info;
4     // Used eventfds:
5     // * netif
6     // * task queue
7     // * border router
8     esp_vfs_eventfd_config_t eventfd_config = {
9         .max_fds = 3,
10    };
11    ESP_ERROR_CHECK(esp_vfs_eventfd_register(&eventfd_config)); // Initialize the eventfd virtual filesystem
12    ESP_ERROR_CHECK(nvs_flash_init()); // Initialize the NVS flash memory
13    ESP_ERROR_CHECK(esp_netif_init()); // Create the network interface for Thread
14
15    // Create the system Event task and initialize an app event's callback function
16    ESP_ERROR_CHECK(esp_event_loop_create_default());
17
18 #if CONFIG_OPENTHREAD_BR_AUTO_START
19     ESP_ERROR_CHECK(example_connect()); // Establish a connection with the Wi-Fi network
20     ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_NONE)); // Set the power saving mode for Wi-Fi to none
21     esp_openthread_set_backbone_netif(get_example_netif()); // Connect OpenThread to the external network
22 #else
23     esp_ot_wifi_netif_init(); //Set up the communication between OpenThread and the Wi-Fi interface
24     esp_openthread_set_backbone_netif(esp_netif_get_handle_from_ifkey("WIFI_STA_DEF"));
25 #endif // CONFIG_OPENTHREAD_BR_AUTO_START
26
27    ESP_ERROR_CHECK(mdns_init()); //mDNS initialization
28    ESP_ERROR_CHECK(mdns_hostname_set("esp-ot-br"));
29
30    // Create a task for the OpenThread stack
31    xTaskCreate(ot_task_worker, "ot_br_main", 20480, xTaskGetCurrentTaskHandle(), 5, NULL);
32
33    // Create a task for the UDP socket
34    xTaskCreate(udp_socket_server_task, "ot_udp_scocket_server", 4096, xTaskGetCurrentTaskHandle(), 4, NULL);
35
36    start_webserver(); // Initialize the websocket server
37    ESP_LOGI(TAG, "Socket is running ... ...\n");
38
39    SSD1306_t dev;
40    i2c_master_init(&dev, CONFIG_SDA_GPIO, CONFIG_SCL_GPIO, CONFIG_RESET_GPIO); // I2C master initialization
41    ssd1306_init(&dev, 128, 64); // Initialize the display
42    ssd1306_clear_screen(&dev, false);
43    ssd1306_contrast(&dev, 0xff);
44
45    // Refresh every 10 seconds and update the display information
46    while(1){
47        char data[16];
48
49        // Get the IPv4 address that is assigned to the ESP32
50        ESP_ERROR_CHECK(esp_netif_get_ip_info(esp_netif_get_handle_from_ifkey("WIFI_STA_DEF"), &ip_info));
51
52        sprintf(data, "%d.%d.%d.%d", IP2STR(&ip_info.ip));
53        ssd1306_display_text(&dev, 1, "IP:      ", 16, false);
54        ssd1306_display_text(&dev, 2, data, 16, false);
55        vTaskDelay(10000 / portTICK_PERIOD_MS);
56    }
57 }

```

Фиг. 5.20. Основната функция в програмата

```
1  #
2  # Example Connection Configuration
3  #
4  CONFIG_ENV_GPIO_RANGE_MIN=0
5  CONFIG_ENV_GPIO_RANGE_MAX=39
6  CONFIG_ENV_GPIO_IN_RANGE_MAX=39
7  CONFIG_ENV_GPIO_OUT_RANGE_MAX=33
8  CONFIG_EXAMPLE_CONNECT_WIFI=y
9  # CONFIG_EXAMPLE_WIFI_SSID_PWD_FROM_STDIN is not set
10 CONFIG_EXAMPLE_PROVIDE_WIFI_CONSOLE_CMD=y
11 CONFIG_EXAMPLE_WIFI_SSID="name"
12 CONFIG_EXAMPLE_WIFI_PASSWORD="password"
13 CONFIG_EXAMPLE_WIFI_CONN_MAX_RETRY=6
14 # CONFIG_EXAMPLE_WIFI_SCAN_METHOD_FAST is not set
15 CONFIG_EXAMPLE_WIFI_SCAN_METHOD_ALL_CHANNEL=y
16
```

Фиг. 5.21. Конфигурации за автоматична връзка в *sdkconfig*

```
1  #
2  # SSD1306 Configuration
3  #
4  CONFIG_GPIO_RANGE_MAX=33
5  CONFIG_I2C_INTERFACE=y
6  # CONFIG_SPI_INTERFACE is not set
7  # CONFIG_SSD1306_128x32 is not set
8  CONFIG_SSD1306_128x64=y
9  CONFIG_OFFSETX=0
10 # CONFIG_FLIP is not set
11 CONFIG_SCL_GPIO=32
12 CONFIG_SDA_GPIO=33
13 CONFIG_RESET_GPIO=15
14 # end of SSD1306 Configuration
```

Фиг. 5.22. Конфигурации за дисплея в *sdkconfig*

```
1  #
2  # Websocket
3  #
4  CONFIG_WS_TRANSPORT=y
5  CONFIG_WS_BUFFER_SIZE=1024
6  # CONFIG_WS_DYNAMIC_BUFFER is not set
7  # end of Websocket
```

Фиг. 5.23. Конфигурации за Websocket в sdkconfig

```
1  #
2  # OpenThread Extension CLI
3  #
4  CONFIG_OPENTHREAD_CLI_ESP_EXTENSION=y
5  CONFIG_OPENTHREAD_CLI_IPERF=y
6  CONFIG_OPENTHREAD_CLI_SOCKET=y
7  CONFIG_OPENTHREAD_CLI_TCP_SERVER_PORT=12345
8  CONFIG_OPENTHREAD_CLI_UDP_SERVER_PORT=12346
9  CONFIG_OPENTHREAD_CLI_WIFI=y
10 # CONFIG_OPENTHREAD_CLI_OTA is not set
11 CONFIG_OPENTHREAD_CLI_CURL=y
12 # end of OpenThread Extension CLI
```

Фиг. 5.24. Конфигурации за OpenThread в sdkconfig

5.8. Сорс код за Основния блок

Подробно обяснение на отделните функции на сорс кода е представено в шеста глава.



```
1 #include <stdio.h>
2 #include <zephyr.h>
3 #include <device.h>
4 #include <devicetree.h>
5 #include <zephyr/logging/log.h>
6 #include <openthread/thread.h>
7 #include <net/openthread.h>
8 #include <openthread/udp.h>
9 #include <zephyr/drivers/sensor.h>
10
11 LOG_MODULE_REGISTER(cli_sample, CONFIG_OT_COMMAND_LINE_INTERFACE_LOG_LEVEL);
```

Фиг. 5.25. Добавяне на библиотеки

```

1 static void udp_send(int light, int temp1, int temp2, int hum1, int hum2,
2                     int press1, int press2, int gas1, int gas2){
3     otError error = OT_ERROR_NONE;
4     char data[300];
5     sprintf(data, "Light: %d\nTemp: %.2d°C Humidity: %.2d% Pressure: %.2dhPa Gas: %.2dΩ",
6             light, temp1, temp2, hum1, hum2, press1, press2, gas1, gas2);
7     otInstance *myInstance;
8     myInstance = openthread_get_default_instance();
9     otUdpSocket mySocket;
10
11    otMessageInfo myInfo;
12
13    //Making sure that the myInfo structure is empty
14    memset(&myInfo, 0, sizeof(myInfo));
15
16    //Set the address to which the data will be send
17    otIp6AddressFromString("ff03::1", &myInfo.mPeerAddr);
18
19    myInfo.mPeerPort = 12346;
20
21    // One cycle loop to enable the use of breaks
22    do{
23        error = otUdpOpen(myInstance, &mySocket, NULL, NULL);
24        if (error != OT_ERROR_NONE){ break; }
25        otMessage *test_Message = otUdpNewMessage(myInstance, NULL);
26        error = otMessageAppend(test_Message, data, (uint16_t)strlen(data));
27        if (error != OT_ERROR_NONE){ break; }
28        error = otUdpSend(myInstance, &mySocket, test_Message, &myInfo);
29        if (error != OT_ERROR_NONE){ break; }
30        error = otUdpClose(myInstance, &mySocket);
31    }while(false);
32
33    if(error == OT_ERROR_NONE){
34        LOG_INF("Send. %s\n", data);
35    }else{
36        LOG_INF("udpSend error: %d\n", error);
37    }
38 }

```

Фиг. 5.26. Функцията *udp_send*

```

1 void main(void)
2 {
3     int ret;
4     const struct device *dev;
5
6     otInstance *myInstance;
7     myInstance = openthread_get_default_instance();
8     otPlatRadioSetTransmitPower(myInstance, 8); //Increase the transmit power to 8dBm
9
10    ret = usb_enable(NULL);
11    if (ret != 0) {
12        LOG_ERR("Failed to enable USB");
13        return;
14    }
15
16    dev = DEVICE_DT_GET(DT_CHOSEN(zephyr_shell_uart)); //Initialize the Zephyr Shell
17    if (dev == NULL) {
18        LOG_ERR("Failed to find specific UART device");
19        return;
20    }
21    LOG_INF("Waiting for host to be ready to communicate");
22
23    // Start the BH1750 sensor
24    powerOn();
25    setMeasuringTime();
26
27    // Initialize the BME688 sensor
28    const struct device *const dev_bme = DEVICE_DT_GET(DT_NODELABEL(bme688_sensor));
29
30    struct sensor_value temp, press, humidity, gas_res;
31    while(1){
32        // Get values from the BME688 sensor
33        sensor_sample_fetch(dev_bme);
34        sensor_channel_get(dev_bme, SENSOR_CHAN_AMBIENT_TEMP, &temp);
35        sensor_channel_get(dev_bme, SENSOR_CHAN_PRESS, &press);
36        sensor_channel_get(dev_bme, SENSOR_CHAN_HUMIDITY, &humidity);
37        sensor_channel_get(dev_bme, SENSOR_CHAN_GAS_RES, &gas_res);
38
39        udp_send(getLux(), temp.val1, temp.val2, humidity.val1, humidity.val2,
40                  press.val1, press.val2, gas_res.val1, gas_res.val2);
41
42        k_sleep(K_MSEC(1000));
43    }
44 }

```

Фиг. 5.27. Основна функция на програмата

```
1 &i2c0 {
2     compatible = "nordic,nrf-twi";
3     status = "okay";
4     pinctrl-0 = <&i2c0_default>;
5     pinctrl-1 = <&i2c0_sleep>;
6     pinctrl-names = "default", "sleep";
7
8     bh1750_sensor: bh1750@23 {
9         compatible = "rohm,bh1750";
10        reg = <0x23>;
11        status = "okay";
12        label = "BH1750";
13    };
14    bme688_sensor: bme688@76 {
15        compatible = "bosch,bme688";
16        reg = <0x76>;
17        status = "okay";
18        label = "BME688";
19    };
20};
```

Фиг. 5.28. Част от devicetree за конфигуриране на i2c0 възела

```
1 / {
2     chosen {
3         zephyr,shell-uart = &cdc_acm_uart0;
4     };
5 };
6
7 &zephyr_udc0 {
8     cdc_acm_uart0: cdc_acm_uart0 {
9         compatible = "zephyr,cdc-acm-uart";
10        label = "CDC_ACM_0";
11    };
12};
```

Фиг. 5.29. Част от devicetree за конфигуриране на USB

```
1 # Network shell
2 CONFIG_SHELL=y
3 CONFIG_OPENTHREAD_SHELL=y
4 CONFIG_SHELL_ARGC_MAX=26
5 CONFIG_SHELL_CMD_BUFF_SIZE=416
6
7 # Enable OpenThread features set
8 CONFIG_OPENTHREAD_NORDIC_LIBRARY_MASTER=y
9 CONFIG_NET_L2_OPENTHREAD=y
10
11 # Generic networking options
12 CONFIG_NETWORKING=y
13 CONFIG_ASSERT=y
14 CONFIG_ASSERT_NO_COND_INFO=y
15 CONFIGMBEDTLS_SHA1_C=n
16 CONFIG_FPU=y
17 CONFIG_GPIO_SHELL=y
18
19 # Enable Zephyr logging
20 CONFIG_LOG=y
21
22 # Use printk to log messages
23 CONFIG_LOG_MODE_MINIMAL=y
24
25 # Log only errors/hard faults
26 CONFIG_LOG_MAX_LEVEL=1
27
28 CONFIG_OPENTHREAD_BORDER_ROUTER=y
29 CONFIG_OPENTHREAD_BACKBONE_ROUTER=y
30 CONFIG_OPENTHREAD_SRP_SERVER=y
31 CONFIG_NRF_802154_SOURCE_NRFXLIB=y
32 CONFIG_NRF_802154_RADIO_DRIVER=y
33
34 #Network parameter
35 CONFIG_OPENTHREAD_CHANNEL=11
36 #CONFIG_OPENTHREAD_NETWORK_NAME="OpenThread"
37 CONFIG_OPENTHREAD_PANID=7856
38 #CONFIG_OPENTHREAD_XPANID="dead00beef00cafe"
39 CONFIG_OPENTHREAD_NETWORKKEY="20118159fbfbbe2146c05ad8e91dcf77"
40 CONFIG_OPENTHREAD_NORDIC_LIBRARY_FTD=y
41 CONFIG_OPENTHREAD_MANUAL_START=n
42
43 #Configure Sensors
44 CONFIG_I2C=y
45 CONFIG_SENSOR=y
46
47 #Configure BH1750
48 CONFIG_BH1750=y
49 CONFIG_BH1750_MODE_CHM_2=y
50 CONFIG_CBPRINTF_FP_SUPPORT=y
51
52 #Configure BME688
53 CONFIG_BME688=y
```

Фиг. 5.30. Съдържанието на *prj.conf*

Шеста глава - Практически резултати

6.1. Конфигуриране на работна среда и инсталация на инструменти за разработка

За разработката на софтуерната част на дипломната работа ще бъде използван софтуерът, посочен в т. 5.1.

6.1.1 Visual Studio Code, nRF Connect SDK, ESP-IDF, VS разширения

Първата програма, която ще бъде необходима за разработката на софтуер и за двата възела, е Visual Studio Code. Тя може да бъде инсталирана от съответния сайт [79].

Втората програма ще бъде nRF Connect for Desktop, чрез която се инсталира и съответния SDK. Инсталацията на всички инструменти ще бъде малко по-комплексна, като на сайта на Nordic са предоставени всички стъпки. Там е описан и начинът за инсталация на разширението nRF Connect за VS Code [92].

Последно, ще трябва да се инсталира ESP-IDF, което ще стане по стъпките, описани на официалния сайт на Espressif [93]. Там са предоставени различни начини за инсталация, като избраният начин се изпълнява чрез VS Code разширението Espressif IDF [94].

6.1.2 Създаване на Github repository

Github хранилището ще бъде създадено през сайта, след което ще трябва да бъде инсталиран Git от официалния сайт [95].

След инсталацията ще бъде стартирана Git Bash конзола в директорията, в която ще бъде клонирано хранилището за дипломната работа, и ще бъдат изпълнени следните команди:

git clone https://github.com/LyubomirNachev/Diplomna.git - с

тази команда се създава копие на хранилището

cd Diplomna/ - с тази команда се навигира в директорията на току-що копираното хранилище

git pull - с тази команда се проверява дали хранилището е актуално и ако не е се теглят необходимите промени

6.2. Създаване на софтуер за основния възел

В тази подточка ще бъде описан процеса за създаване на управляващ софтуер за основния възел.

6.2.1. Създаване на OpenThread устройство с конзола

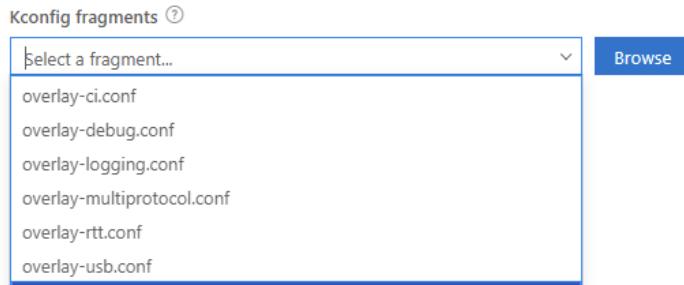
В примерите, предоставени в **nrf/** директорията на Nordic SDK, съществува CLI (на англ. *Command Line Interface*) пример, който демонстрира изпращането на команди към Thread устройство чрез използването на OpenThread Command Line Interface [96].

Примерът директно поддържа nRF52840-Dongle, премахвайки нуждата от конфигурирането на нов .overlay файл.

Всяко взаимодействие с програмата се извършва чрез серийна комуникация. CLI примерът предоставя готови конфигурационни файлове за реализирането на различни опции:

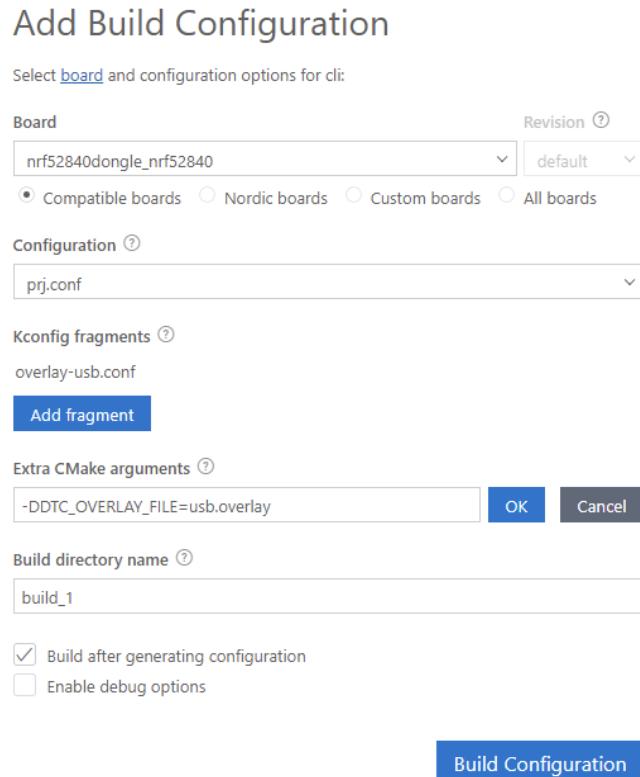
- Логване през RTT(на англ. *Real Time Transfer*)
- Отстраняване на грешки (Debugging)
- Премахване на банера при включване, както и shell prompt
- Включване на BLE поддръжка
- Поддръжка за експериментален TCP
- Поддръжка за USB

Всички тези опции се реализират с добавянето на съответните им конфигурационни файлове в полето за Kconfig фрагменти в build конфигурацията, показано на фиг. 6.1.



Фиг. 6.1. Полето за Kconfig фрагменти

Конфигурацията, която се използва за построяването на програмата за nRF52840 Dongle, е показана на фиг. 6.2.



Фиг. 6.2. Конфигурацията за построяването на програмата за nRF52840 Dongle

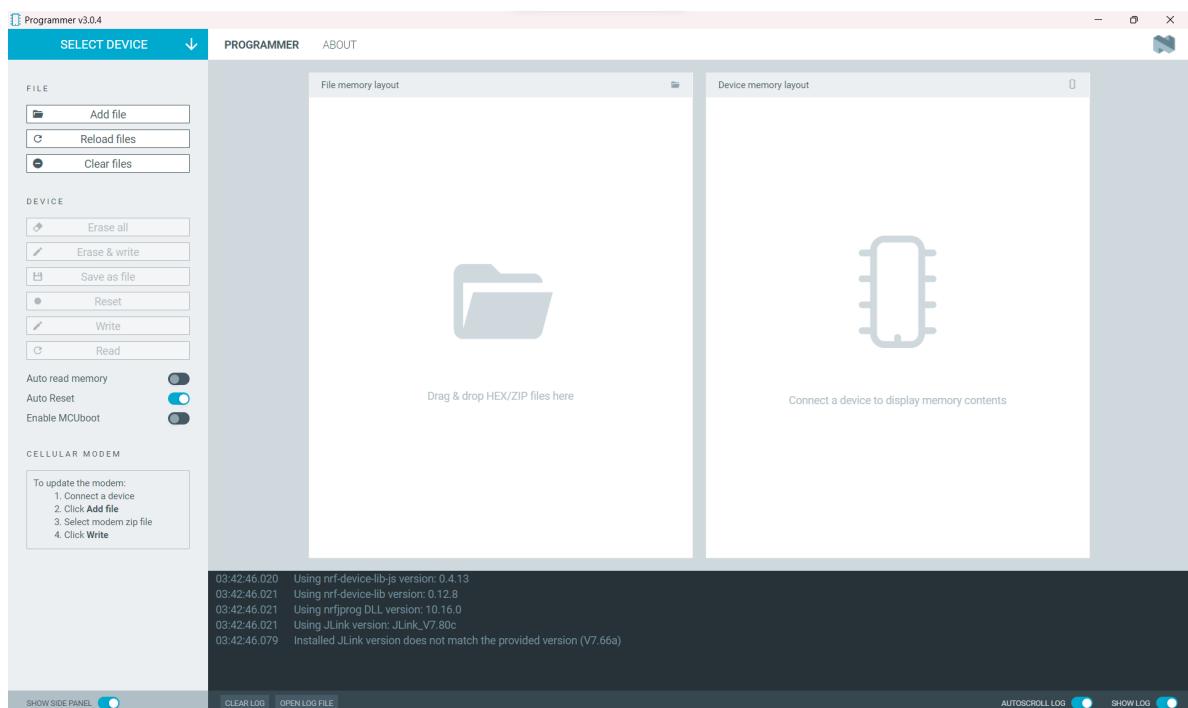
На фиг. 6.2. най-отгоре е показано как е избран съответния .overlay файл за избрания модул, а именно -

nrf52840dongle_nrf52840.overlay. След това е посочен файлът с основната конфигурация на програмата, който се избира автоматично. Следващата опция е тази, от която се избира кои настройки да бъдат добавени в програмата. В този случай е добавен само overlay-usb.conf файлът, който позволява на конзолата да се отваря през USB.

За да работи конзолата през USB е нужна допълнителна конфигурация, която ще бъде стартирана в следващата опция, а именно добавянето на допълнителен .overlay файл. В полето за допълнителни CMake аргументи ще бъде добавен редът -

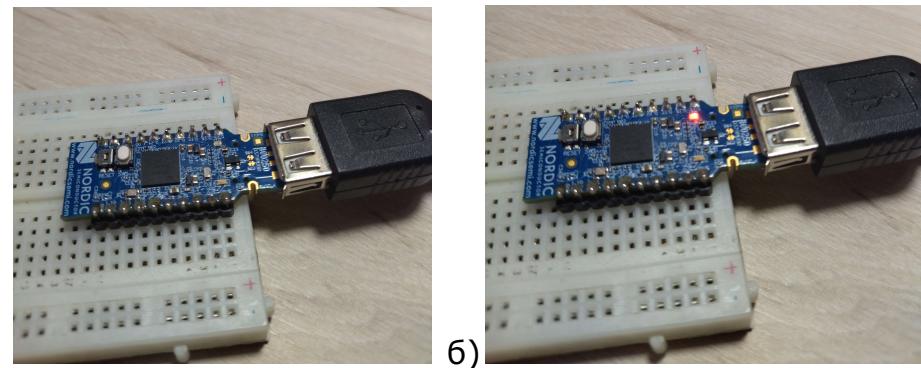
DDTC_OVERLAY_FILE=usb.overlay. Финалната опция е избирането на името на build директорията.

След като програмата се построи тя ще трябва да бъде заредена на самия nRF52840 dongle. Това се случва през апликацията nRF Connect for Desktop, която има вътрешно приложение за програмиране на платката, показано на фиг. 6.3.



Фиг. 6.3. Приложение за програмиране на платки в nRF Connect for Desktop

От приложението може да бъде избрано устройството чрез падащото меню “SELECT DEVICE”, като преди това е необходимо да бъде натиснат Reset бутона на nRF52840-Dongle. След натискане вграденият светодиод ще започне да мига в червено и платката ще се появи в менюто. Процесът е графично представен на фиг. 6.4.a) и фиг. 6.4.b).

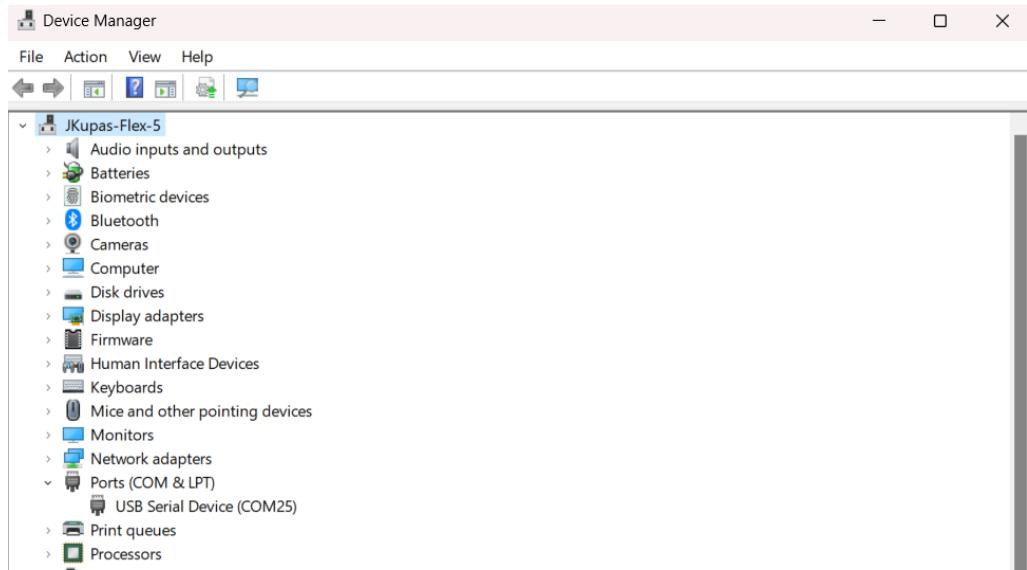


Фиг. 6.4.a) *nRF52840-Dongle* преди натискането на страничния *Reset* бутон, б) *nRF52840-Dongle* след натискането на страничния *Reset* бутон

След това се избира опцията Add file и се навигира до директорията, в която се намира построената програма. В случая - **C:\Diplomna\nrf-apps\cli_3\build_3\zephyr**. В тази директория ще има генериран zephyr.hex файл, който трябва да бъде избран и зареден на устройството.

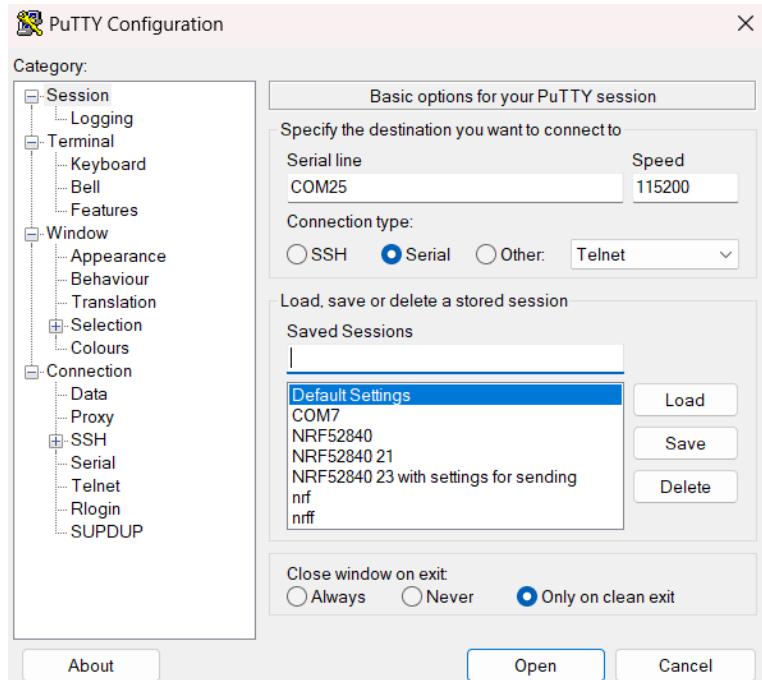
След като програмата е заредена ще бъде необходим начин за достъпване до Openthread CLI. За тази цел се използва програмата PuTTY със следната конфигурация [120]:

- Вид на връзката: Серийна (Serial)
- Серийна линия: COMx (x е номера на COM порта, на който е свързано устройството, който може да се види през Device Manager в Windows - фиг. 6.5)



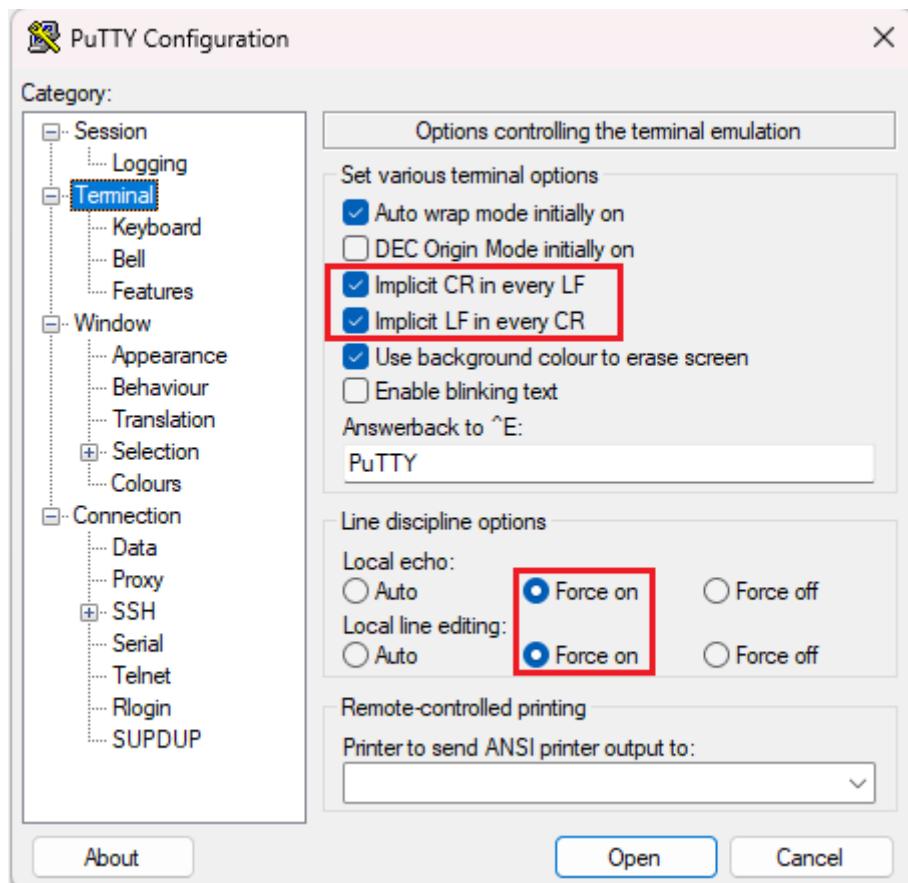
Фиг. 6.5. Програмата Device Manager в Windows

- Скорост: 115200 bit/s
- Конфигурацията е представена на фиг. 6.6.



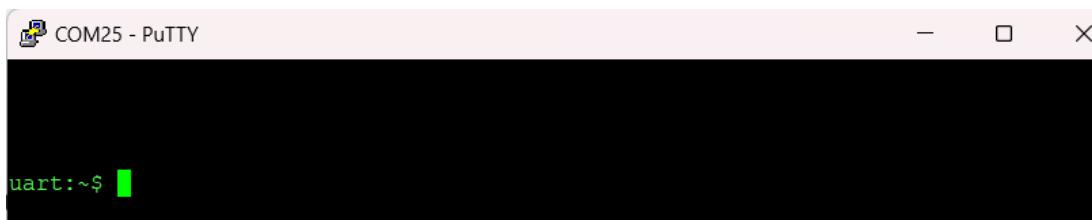
Фиг. 6.6. Конфигурация на програмата PuTTY за работа с
nRF52840-Dongle

Освен тази конфигурация ще е необходимо в категорията "Terminal", в лявото меню, да бъдат пуснати настройките показани на фиг. 6.7.



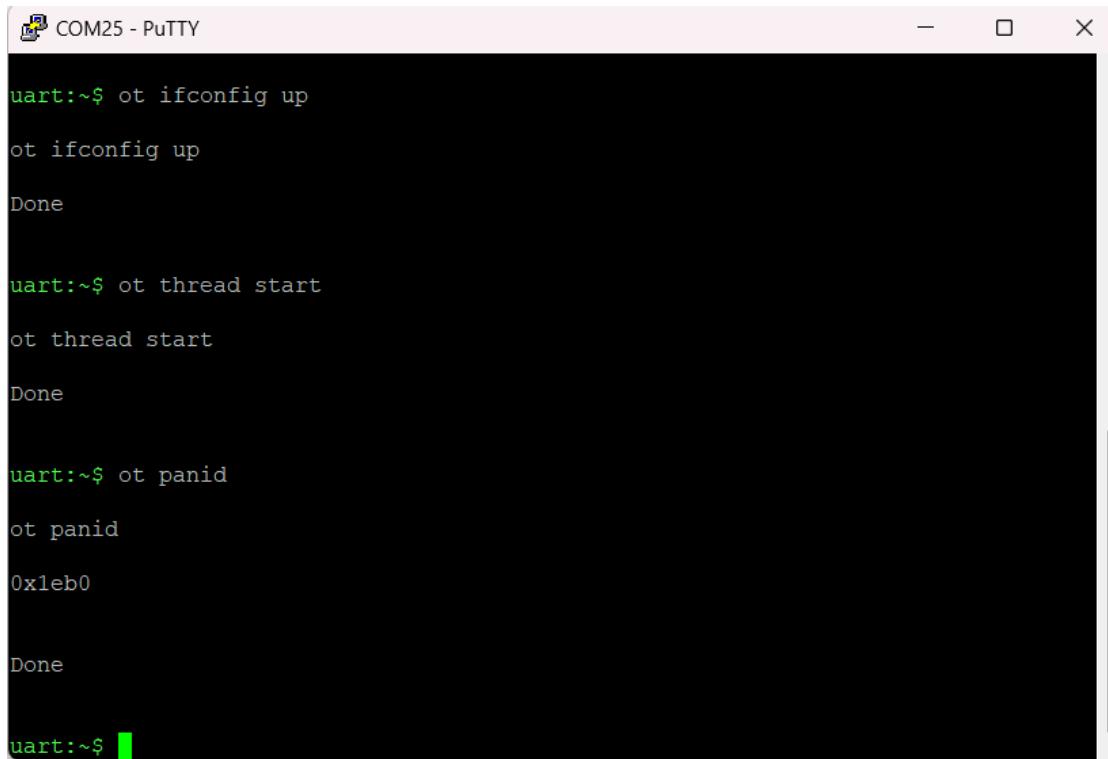
Фиг. 6.7. Настройки на конзолата

След като започне серийната комуникация ще излезе следния прозорец - фиг. 6.8.



Фиг. 6.8. Изглед към отворения OpenThread терминал

От тук нататък могат да бъдат използвани всички команди за OpenThread, като преди всяка команда тряба да бъде добавено **от**. Примерни команди са представени на фиг. 6.9.



```
uart:~$ ot ifconfig up
ot ifconfig up
Done

uart:~$ ot thread start
ot thread start
Done

uart:~$ ot panid
ot panid
0x1eb0

Done

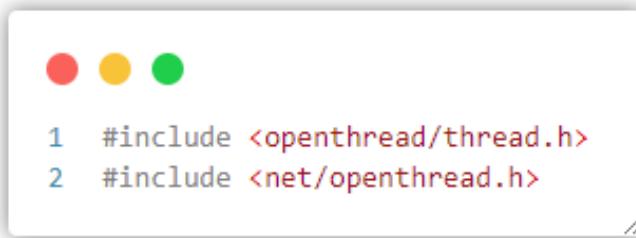
uart:~$
```

Фиг. 6.9. Примерни команди въведени в OpenThread терминала

Целият лист с команди се намира на GitHub страницата на OpenThread - [97].

6.2.2. Увеличаване на мощността на предаване на модулите

Увеличаването на мощността на предаването на модулите ще бъде направено с помощта на библиотеките, представени на фиг. 6.10.



Фиг. 6.10. Библиотеки, които ще бъдат използвани за увеличаване на мощността на предаване

В тях е предоставена готовата функция **otPlatRadioSetTransmitPower**, като нейните параметри са представени на фиг. 6.11. [98].

В проекта параметрите ще бъдат тези, представени на фиг. 6.12., като OpenThread instance структурата, която ще бъде подадена на **otPlatRadioSetTransmitPower**, е OpenThread инстанцията по подразбиране. Тя се взима чрез функцията **openthread_get_default_instance**. Параметърът **aPower**, който показва големината на силата за предаване в dBm, ще получи стойност 8.

Редовете, показани на фиг. 6.10. и фиг. 6.12., ще бъдат включени в **main** функцията на миналата програма.

otPlatRadioSetTransmitPower

The screenshot shows the documentation for the `otPlatRadioSetTransmitPower` function. It includes the function signature, a brief description, parameters, return values, and notes.

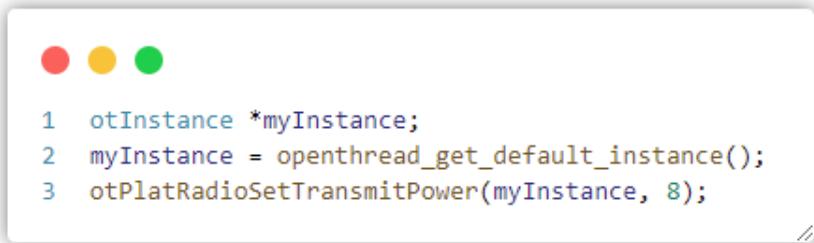
otError otPlatRadioSetTransmitPower(
 `otInstance *aInstance,`
 `int8_t aPower`
)

Set the radio's transmit power in dBm.

Note: The real transmit power will be no larger than the power specified in the max power table for the current channel.

Details		
Parameters	<code>[in] aInstance</code>	The OpenThread instance structure.
	<code>[in] aPower</code>	The transmit power in dBm.
Return Values	<code>OT_ERROR_NONE</code>	Successfully set the transmit power.
	<code>OT_ERROR_NOT_IMPLEMENTED</code>	Transmit power configuration via dBm is not implemented.

Фиг. 6.11. Параметри на функцията `otPlatRadioSetTransmitPower`



Фиг. 6.12. Параметрите на `otPlatRadioSetTransmitPower`, които са зададени в програмата

6.2.3. Свързване и конфигуриране на BH1750

За конфигурирането на BH1750 ще бъде необходимо създаването на собствен сурс код за работа със сензора поради липсата на такъв в nRF Connect SDK и Zephyr.

Първото нещо, което трябва да се направи, е да се създаде директория за новото приложение. За генерирането на начални файлове трябва да се стартира nRF Connect for VS Code и да се избере опцията “Create new application”. Изглед към прозореца за ново приложение е показан на фиг. 6.13.

New Application

Application type [?](#)

Freestanding Workspace

Freestanding applications require and use a locally installed nRF Connect SDK. [More information](#)

nRF Connect SDK [?](#)

2.1.2 (c:\Diplomna\ncs\v2.1.2) [Install...](#)

An SDK is already selected in this workspace. Open another workspace or adjust your workspace configuration to enable SDK selection.

nRF Connect Toolchain [?](#)

2.1.2 (c:\Diplomna\ncs\toolchains\v2.1.2) [Uninstall](#)

Application location [?](#)

c:\Diplomna\nrf-apps [...](#)

Application template [?](#)

zephyr/samples/sensor/isl29035 [Browse...](#)

Application name [?](#)

bh1750 [Uninstall](#)

[Create Application](#)

Фиг. 6.13. Прозорец за създаване на ново приложение

В него трябва да се променят следните параметри:

- директорията, в която ще се създаде новата програма (Application location), а именно: **c:\Diplomna\nrf-apps;**
- добавяне/копиране в директория (Application template) на съществуващ пример за друг сензор на светлина (в случая - isl29035), което е полезно за разработка на новия пример за bh1750: **zephyr/samples/sensor/isl29035;**
- името на новата програма (Application name): **bh1750.**

След като се създаде директория за новия сензор, трябва да се намери начин за неговото управление. В интернет има много библиотеки, създадени за работа с него, но на различни платформи, заради това ще е необходимо или да се работи без библиотека, или да се създаде нова такава. За по-подробно разбиране на работата на сензора е използвана информацията от следната статия [99].

Създаване на библиотека за **bh1750**

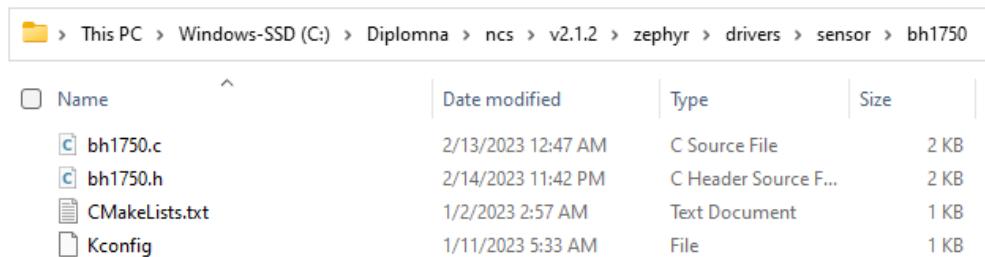
След разучаване на различни примерни библиотеки и сурс код за сензора беше решено, че най-добре ще бъде да се създаде нова библиотека, чрез която ще се улесни бъдещата работа със сензора [100][101].

За създаването на библиотека за сензора в Zephyr ще трябва да бъдат предприети следните действия [102]:

- В директорията

C:\Diplomna\ncs\v2.1.2\zephyr\drivers\sensor ще бъде създадена нова папка - **bh1750**;

- В нея ще бъдат създадени файловете, показани на фиг. 6.14.



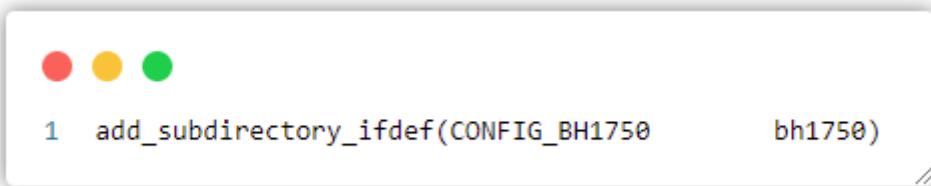
Name	Date modified	Type	Size
bh1750.c	2/13/2023 12:47 AM	C Source File	2 KB
bh1750.h	2/14/2023 11:42 PM	C Header Source F...	2 KB
CMakeLists.txt	1/2/2023 2:57 AM	Text Document	1 KB
Kconfig	1/11/2023 5:33 AM	File	1 KB

Фиг. 6.14. Файловете в папката **bh1750**

- В по-горната директория

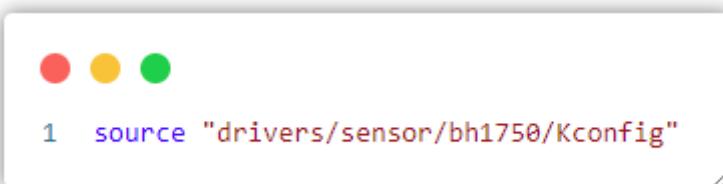
C:\Diplomna\ncs\v2.1.2\zephyr\drivers\sensor трябва да се променят CMakeLists.txt и Kconfig файла, функциите на които са описани по-задълбочено надолу, като в CMakeLists.txt трябва да се

добави редът, показан на фиг. 6.15, а в Kconfig файла - този, показан на фиг. 6.16.



```
1 add_subdirectory_ifdef(CONFIG_BH1750 bh1750)
```

Фиг. 6.15. Ред, който трябва да се добави към CMakeLists.txt



```
1 source "drivers/sensor/bh1750/Kconfig"
```

Фиг. 6.16. Ред, който трябва да се добави към Kconfig

- Финално, в директорията

C:\Diplomna\ncs\v2.1.2\zephyr\dts\bindings\sensor трябва да бъде създаден файлът - **rohm,bh1750.yaml**

След като са създадени новите файлове (**bh1750.c; bh1750.h;** **CMakeLists.txt; Kconfig; rohm,bh1750.yaml**) те трябва да се модифицират по следния начин:

rohm,bh1750.yaml

В Zephyr .yaml файловете се използват за записване на обвързваща информация, свързана с описание на хардуера на Devicetree и конфигурация на тестови случаи [103].



```
1 # A high level description of the device the binding applies to:  
2 description: |  
3     Rohm BH1750 Ambient Light Sensor. See datasheet at  
4     https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf  
5  
6 # Used to match nodes to this binding as discussed above:  
7 compatible: "rohm,bh1750"  
8  
9 # You can include definitions from other bindings using this syntax:  
10 include: i2c-device.yaml
```

Фиг. 6.17. rohm,bh1750.yaml

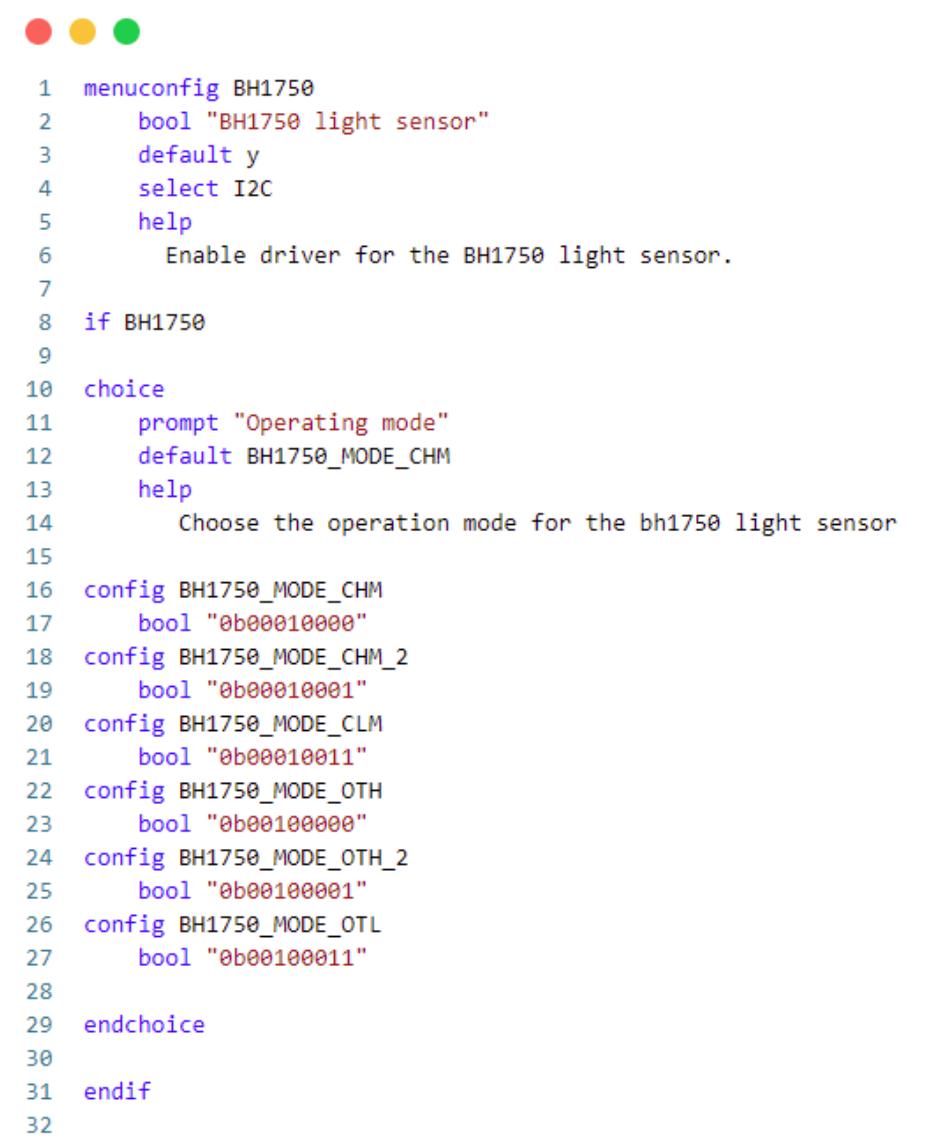
На фиг. 6.17. е показано съдържанието на rohm,bh1750.yaml файла. В него има три ключа - description, compatible и include.

В Zephyr **description** ключът предоставя основна информация за устройството, **compatible** ключът се използва за намиране на съответствие на възлите с избраната връзка, а **include** ключът се използва за включването на дефиниции от други връзки.

Kconfig

Kconfig се използва за конфигуриране на софтуерни опции, които ще бъдат имплементирани в крайното изображение (image) [102].

В случая, в Kconfig файла се дефинира опцията BH1750, която е от тип boolean. Чрез нея ще се включва и изключва опцията за активиране на сензора. Освен това има опция за конфигуриране на работния режим на сензора. Това е показано на фиг. 6.18.



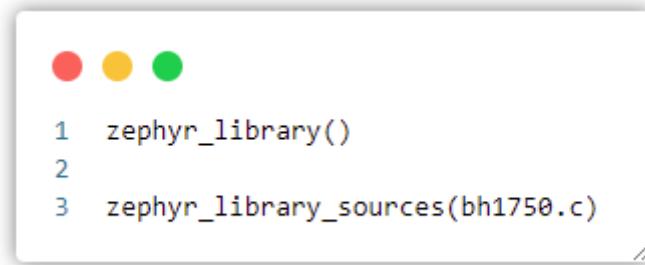
```
1 menuconfig BH1750
2     bool "BH1750 light sensor"
3     default y
4     select I2C
5     help
6         Enable driver for the BH1750 light sensor.
7
8 if BH1750
9
10 choice
11     prompt "Operating mode"
12     default BH1750_MODE_CHM
13     help
14         Choose the operation mode for the bh1750 light sensor
15
16 config BH1750_MODE_CHM
17     bool "0b00010000"
18 config BH1750_MODE_CHM_2
19     bool "0b00010001"
20 config BH1750_MODE_CLM
21     bool "0b00010011"
22 config BH1750_MODE_OTH
23     bool "0b00100000"
24 config BH1750_MODE_OTH_2
25     bool "0b00100001"
26 config BH1750_MODE_OTL
27     bool "0b00100011"
28
29 endchoice
30
31 endif
32
```

Фиг. 6.18. Kconfig файл

CMakeLists.txt

Този файл съобщава на системата за изграждане (build system), къде да намери другите файлове на приложението, и свързва директорията на приложението със системата за изграждане CMake на Zephyr [105].

Съдържанието на файла е показано на фиг. 6.19.



```
1 zephyr_library()
2
3 zephyr_library_sources(bh1750.c)
```

Фиг. 6.19. *CMakeLists.txt*

bh1750.h

Заглавният файл, който обикновено се обозначава със суфиксa .h, съдържа информация за библиотеката, която програмите, които я използват, трябва да знаят. В общия случай той съдържа константи и типове, както и прототипи за функциите, налични в библиотеката.

На фиг. 6.20. е показано как трябва да бъдат дефинирани всички регистри на BH1750, както и обръщението към функции, дефинирани в **bh1750.c** файла:

```
1 #ifndef ZEPHYR_DRIVERS_SENSOR_BH1750_BH1750_H_
2 #define ZEPHYR_DRIVERS_SENSOR_BH1750_BH1750_H_
3
4 #include <zephyr/device.h>
5 #include <zephyr/devicetree.h>
6 #include <zephyr/kernel.h>
7
8 #define BH1750_ADDRESS_1          0x23
9 #define BH1750_ADDRESS_2          0x5C
10 #define BH1750_DATA_REG_RESET    0x07
11 #define BH1750_POWER_DOWN         0x00
12 #define BH1750_POWER_ON          0x01
13 #define I2C_NODE                 DT_NODELABEL(i2c0)
14
15 #if CONFIG_BH1750_MODE_CHM
16     #define BH1750_MODE 0b00010000
17 #elif CONFIG_BH1750_MODE_CHM_2
18     #define BH1750_MODE 0b00010001
19 #elif CONFIG_BH1750_MODE_CLM
20     #define BH1750_MODE 0b00010011
21 #elif CONFIG_BH1750_MODE_OTH
22     #define BH1750_MODE 0b00100000
23 #elif CONFIG_BH1750_MODE_OTH_2
24     #define BH1750_MODE 0b00100001
25 #elif CONFIG_BH1750_MODE_OTH_L
26     #define BH1750_MODE 0b00100011
27 #endif
28
29 extern int getLux();
30 extern void powerDown();
31 extern void powerOn();
32 extern void dataRegReset();
33 extern void setMeasuringTime();
34 extern uint16_t readBH1750();
35 extern void writeBH1750(uint8_t val);
36
37 #endif /* ZEPHYR_DRIVERS_SENSOR_BH1750_BH1750_H_ */
```

Фиг. 6.20. *bh1750.h*

bh1750.c

Това е основният файл, съдържащ дефинициите на функциите, използвани в библиотеката.

```
● ● ●  
1 #define DT_DRV_COMPAT rohm_bh1750  
2  
3 #include <zephyr/drivers/i2c.h>  
4 #include "bh1750.h"  
5  
6 float measuringTimeFactor = 1;  
7 uint8_t i2c_buffer[2];  
8  
9 static const struct device *dev = DEVICE_DT_GET(I2C_NODE);  
10 int err;
```

Фиг. 6.21. Първите редове в bh1750.c файла

В първите десет реда, показани на фиг. 6.21., са включени нужните библиотеки за работа със сензора, променливата, показваща стойността на **MTReg** регистъра, буфера, използван за по-нататъшна I²C комуникация, структурата на устройството и променлива за грешка.



```
1 extern int getLux(){
2     if (!device_is_ready(dev)){
3         printk("device is not ready\n");
4         return 0;
5     }
6     uint16_t raw_lux;
7     float lux;
8     raw_lux = readBH1750();
9     if((BH1750_MODE == 0b00010001)|| (BH1750_MODE == 0b00100001)){
10        lux = (raw_lux/2.4)/measuringTimeFactor;
11    }else{
12        lux = (raw_lux/1.2)/measuringTimeFactor;
13    }
14    return (int)lux;
15 }
```

Фиг. 6.22. *getLux* функцията в *bh1750.c* файла

На фиг. 6.22. е показана **getLux** функцията, която връща стойността на измерената от сензора светлина като целочислена стойност.

```
● ● ●

1  extern void powerDown(){
2      writeBH1750(BH1750_POWER_DOWN);
3  }
4
5  void setMode(){
6      writeBH1750(BH1750_MODE);
7  }
8
9  extern void powerOn(){
10     writeBH1750(BH1750_POWER_ON);
11     setMode();
12 }
13
14 extern void dataRegReset(){
15     writeBH1750(BH1750_DATA_REG_RESET);
16 }
17
18 extern void setMeasuringTime(){
19     uint8_t mt = measuringTimeFactor*69;
20     uint8_t highByteMT = ((mt>>5) | 0b01000000);
21     uint8_t lowByteMT = (mt & 0b01111111);
22     lowByteMT |= 0b01100000;
23     writeBH1750(highByteMT);
24     writeBH1750(lowByteMT);
25 }
```

Фиг. 6.23. Функции от *bh1750.c* файла

На фиг. 6.23. Са показани функциите, които се използват за изключване, посочване на режима на работа, включване, анулиране и задаване на времето за измерване на сензора.



```
1 extern uint16_t readBH1750(){
2     err = i2c_read(dev, i2c_buffer, 2, BH1750_ADDRESS_1);
3     return ((i2c_buffer[0]<<8) + i2c_buffer[1]);
4 }
5
6
7 extern void writeBH1750(uint8_t val){
8     i2c_buffer[0] = val;
9     err = i2c_write(dev, i2c_buffer, 1, BH1750_ADDRESS_1);
10 }
```

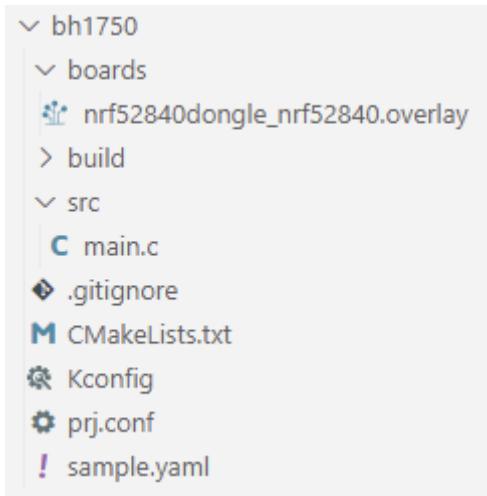
Фиг. 6.24. Функции за четене и писане в регистрите от bh1750.c файла

На фиг. 6.24. са показани двете функции за писане и четене в регистрите на сензора за светлината.

След конфигурирането на всички файлове, необходими за създаването на работеща библиотека за BH1750, ще бъде нужно да се създаде програма, която да използва новонаправената библиотека и да показва данните от сензора.

Създаване на програма за bh1750

От копирания пример за isl29035 ще бъде изтрито почти всичко, като финално в директорията с BH1750 примера ще останат файловете, показани на фиг. 6.25.



Фиг. 6.25. Файлове в папката на приложението *bh1750*

В папката *boards* ще се намира *.overlay* файла за използваната платка. Той служи като разширение на *.dts* файла с конфигурации специфично за тази програма. В него е добавена конфигурация за необходимите устройства, които ще трябва да бъдат пуснати, а именно сензорът и конзолата [106].

В *i2c0* възела е създаден дъщерен възел за сензора *BH1750*, което е необходимо, защото по-горе в библиотеката на съответния сензор беше дефинирана структура на устройството, сочеща към *i2c0* възела.

Chosen възелът не представлява реално устройство, а служи като място за предаване на данни между фърмуера и операционната система, като например *boot* аргументи. Данните в избрания възел не отразяват хардуера.

На фиг. 6.26. е показан *nrf52840dongle_nrf52840.overlay* файлът.

```
1 &i2c0 {
2     bh1750@23 {
3         compatible = "rohm,bh1750";
4         reg = <0x23>;
5         status = "okay";
6     };
7 };
8 / {
9     chosen {
10         zephyr,console = &cdc_acm_uart0;
11     };
12 };
13
14 &zephyr_udc0 {
15     cdc_acm_uart0: cdc_acm_uart0 {
16         compatible = "zephyr,cdc-acm-uart";
17     };
18 };
```

Фиг. 6.26. *nrf52840dongle_nrf52840.overlay*

Следващият файл, който е променен значително, се намира в папката **src** - файлът `main.c`, който съдържа самата програма, която ще бъде заредена на устройството. Файлът е показан на фиг. 6.27.



```
1 #include <zephyr/zephyr.h>
2 #include <zephyr/device.h>
3 #include <zephyr/drivers/sensor.h>
4 #include <stdio.h>
5 #include <zephyr/sys/util.h>
6 #include <zephyr/drivers/i2c.h>
7 #include <zephyr/sys/printk.h>
8 #include <zephyr/usb/usb_device.h>
9 #include <zephyr/drivers/uart.h>
10
11 void main(void){
12     const struct device *dev1 = DEVICE_DT_GET(DT_CHOSEN(zephyr_console));
13     uint32_t dtr = 0;
14
15     if (usb_enable(NULL)) {
16         return;
17     }
18
19     /* Poll if the DTR flag was set */
20     while (!dtr) {
21         uart_line_ctrl_get(dev1, UART_LINE_CTRL_DTR, &dtr);
22         k_sleep(K_MSEC(100));
23     }
24     powerOn();
25     setMeasuringTime();
26     while(1){
27         printk("Light: %d lx \n", getLux());
28         k_sleep(K_MSEC(10000));
29     }
30 }
```

Фиг. 6.27. *main.c*

В началото на програмата се добавят всички библиотеки, които ще бъдат използвани. След това в `main` функцията се дефинира структура на устройството, която сочи към `zephyr` конзолата.

В `Zephyr`, за да бъде получена информация за конкретен възел на `devicetree`, е необходим идентификатор на възела. Той представлява С макрос, който сочи към възела. Този възел може да

бъде взет от пътека, етикет на възела, прякор, номер на инстанцията, chosen възел или по parent/child [107].

В този случай това се постига чрез макрото **DEVICE_DT_GET**, което приема като параметър идентификатор на възел от devicetree(или overlay) файла, получен чрез друго макро - **DT_CHOSEN**. Това макро връща идентификатор на /chosen възел.

След това в програмата има проверка за успешното включване на USB последвана от проверка дали серийната конзола е отворена. Ако тя не е отворена, остатъкът от програмата няма да бъде изпълнен.

После се извиква функцията за включване на BH1750 сензора - **powerOn**, последвана от функцията за настройване на времето за измерване - **setMeasuringTime**.

Финално, програмата влиза в безкраен цикъл, който принтира данните от сензора чрез функцията **getLux** на всеки 10 секунди.

Допълнително, в root директорията на приложението ще се съдържат следните конфигурационни файлове:

- CMakeLists.txt, който служи за конфигуриране на настройките за build на програмата, показан на фиг. 6.28.

```
1 cmake_minimum_required(VERSION 3.20.0)
2 find_package(Zephyr REQUIRED HINTS ${ENV{ZEPHYR_BASE}})
3 project(bh1750)
4
5 FILE(GLOB app_sources src/*.c)
6 target_sources(app PRIVATE ${app_sources})
```

Фиг. 6.28. CMakeLists.txt

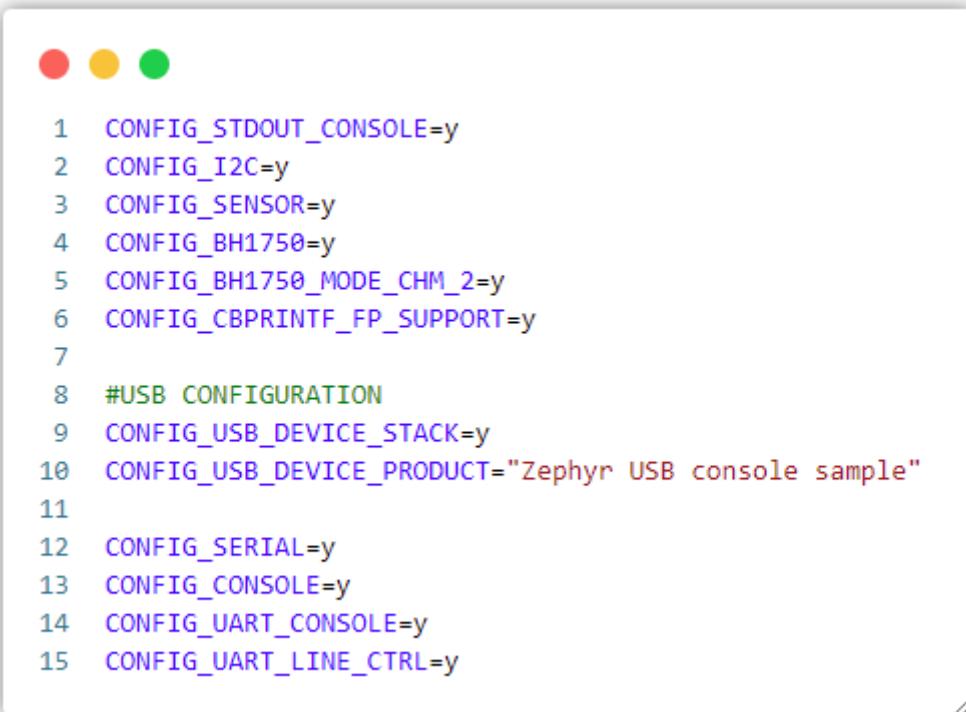
- Kconfig, служещ за конфигуриране на PID (съкр. Product ID) на USB устройството, показан на фиг.6.29.



```
1 config USB_DEVICE_PID
2     default USB_PID_CONSOLE_SAMPLE
3
4 source "Kconfig.zephyr"
```

Фиг. 6.29. Kconfig

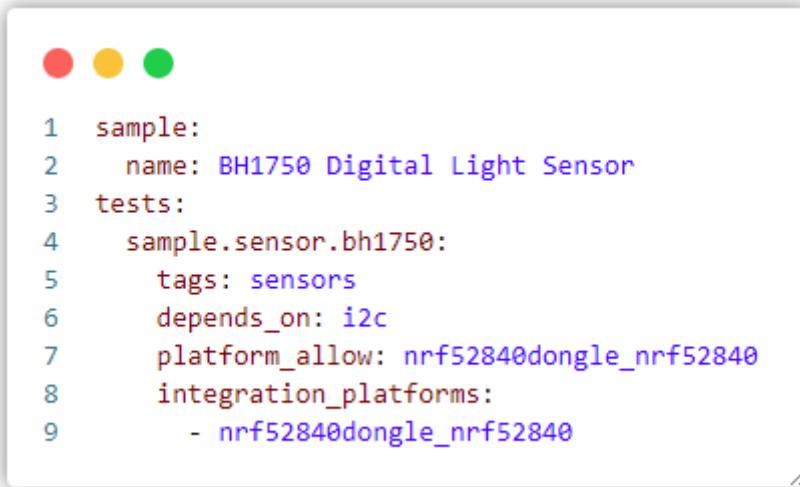
- prj.conf, показващ софтуерните опции, които ще бъдат необходими за успешно построяване на програмата, показан на фиг. 6.30.



```
1 CONFIG_STDOUT_CONSOLE=y
2 CONFIG_I2C=y
3 CONFIG_SENSOR=y
4 CONFIG_BH1750=y
5 CONFIG_BH1750_MODE_CHM_2=y
6 CONFIG_CBPRINTF_FP_SUPPORT=y
7
8 #USB_CONFIGURATION
9 CONFIG_USB_DEVICE_STACK=y
10 CONFIG_USB_DEVICE_PRODUCT="Zephyr USB console sample"
11
12 CONFIG_SERIAL=y
13 CONFIG_CONSOLE=y
14 CONFIG_UART_CONSOLE=y
15 CONFIG_UART_LINE_CTRL=y
```

Фиг. 6.30. Prj.conf

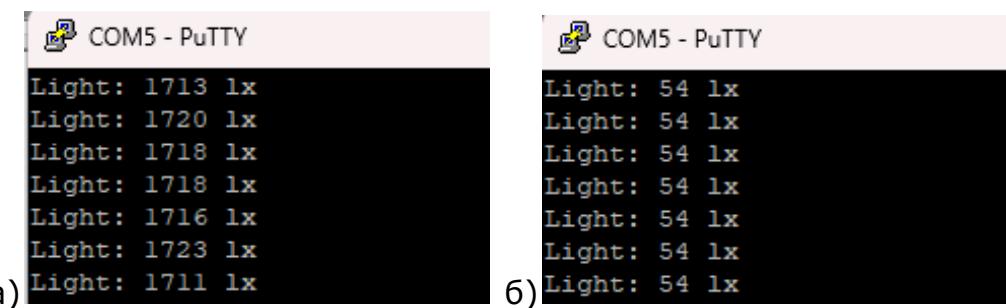
- sample.yaml, който предоставя шаблон за тестване. Показан е на фиг. 6.31.



```
1 sample:
2   name: BH1750 Digital Light Sensor
3 tests:
4   sample.sensor.bh1750:
5     tags: sensors
6     depends_on: i2c
7     platform_allow: nrf52840dongle_nrf52840
8     integration_platforms:
9       - nrf52840dongle_nrf52840
```

Фиг. 6.31. sample.yaml

На фиг. 6.32.а) и фиг. 6.32.б) е показан изходът от серийната конзола при включено устройство.



a)

Light: 1713 lx
Light: 1720 lx
Light: 1718 lx
Light: 1718 lx
Light: 1716 lx
Light: 1723 lx
Light: 1711 lx

б)

Light: 54 lx
Light: 54 lx

Фиг. 6.32.а) Изходът от серийната конзола при светната лампа в стаята, б) Изходът от серийната конзола при изгасена лампа

6.2.4. Свързване и конфигуриране на BME688

Поради наличието на вече съществуваща библиотека и пример за сензора BME680 в Zephyr разработката на управляващ софтуер за сензора BME688 ще бъде облекчена.

Тъй като BME680 и BME688 споделят едни и същи регистри, за създаването на управляващ софтуер може да се използва вече съществуващата библиотека на BME680. Въпреки това беше взето решение за съставяне на нова библиотека за този сензор. Тя ще се базира изцяло на старата, като единственото нещо, което ще бъде направено, е създаването на нова папка в **drivers/sensor** на име BME688 и копиране на файловете от библиотеката на BME680. Съдържанието на папката е показано на фиг. 6.33. След това ще бъдат заменени всички места, където се среща името BME680, на BME688 и финално ще бъде добавена към конфигурацията с включването на другите библиотеки.

This PC > Windows-SSD (C:) > Diplomna > ncs > v2.1.2 > zephyr > drivers > sensor > bme688				
Name	Date modified	Type	Size	
bme688.c	10/2/2022 7:06 PM	C Source File	13 KB	
bme688.h	10/2/2022 8:09 PM	C Header Source F...	6 KB	
CMakeLists.txt	10/2/2022 7:04 PM	Text Document	1 KB	
Kconfig	10/3/2022 12:55 AM	File	3 KB	

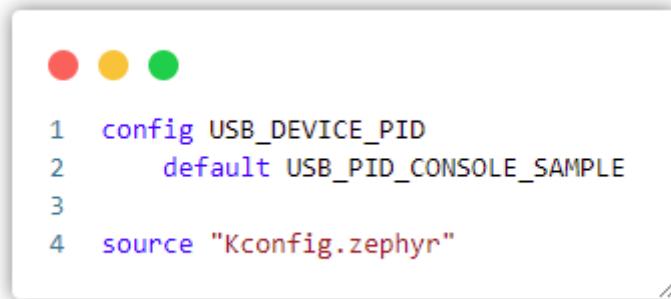
Фиг. 6.33. Файлове в папката *bme688*

Всички останали промени по Kconfig, .yaml и други файлове, които трябва да се извършат за добавянето на библиотеката, ще бъдат направени аналогично на добавянето на библиотеката за bh1750 сензора, описан в т.6.2.3.

Примерът за сензора ще се базира на вече съществуващия за BME680, но с новата BME688 библиотека и добавена USB конзола.

На по-долните фигури са описани файловете, които са в примера.

Kconfig



```
1 config USB_DEVICE_PID
2     default USB_PID_CONSOLE_SAMPLE
3
4 source "Kconfig.zephyr"
```

Фиг. 6.34. Kconfig

nrf52840dongle_nrf52840.overlay

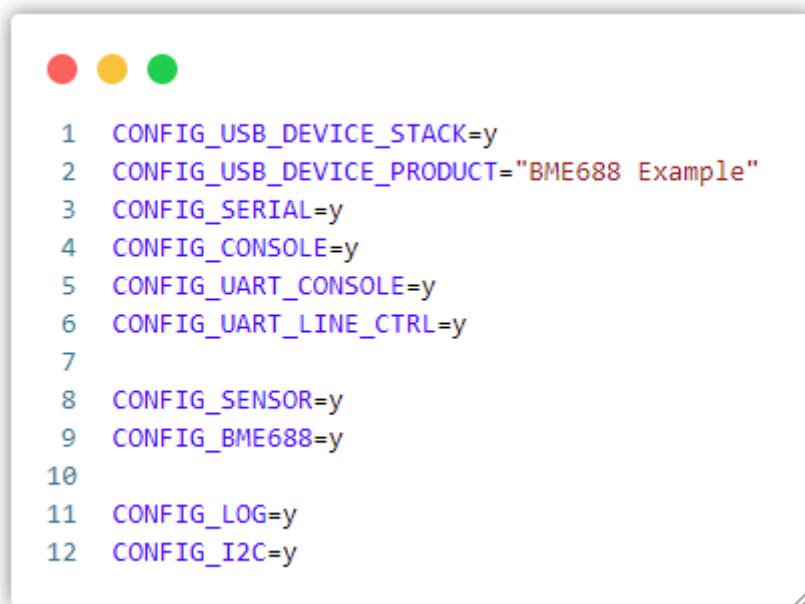


```
1 &i2c0 {
2     bme688@76 {
3         compatible = "bosch,bme688";
4         reg = <0x76>;
5         status = "okay";
6     };
7 };
8 / {
9     chosen {
10         zephyr,console = &cdc_acm_uart0;
11     };
12 };
13
14 &zephyr_udc0 {
15     cdc_acm_uart0: cdc_acm_uart0 {
16         compatible = "zephyr,cdc-acm-uart";
17     };
18 };
```

Фиг. 6.35. nrf52840dongle_nrf52840.overlay

На фиг. 6.35. е показан .overlay файлът за nRF52840-Dongle. Той отново се препокрива с този, описан в т.6.2.3, за BH1750 сензора, като единствено на i2c0 възелът, вместо bh1750 дъщерен възел, има такъв за BME688.

prj.conf



```
1 CONFIG_USB_DEVICE_STACK=y
2 CONFIG_USB_DEVICE_PRODUCT="BME688 Example"
3 CONFIG_SERIAL=y
4 CONFIG_CONSOLE=y
5 CONFIG_UART_CONSOLE=y
6 CONFIG_UART_LINE_CTRL=y
7
8 CONFIG_SENSOR=y
9 CONFIG_BME688=y
10
11 CONFIG_LOG=y
12 CONFIG_I2C=y
```

Фиг. 6.36. prj.conf

sample.yaml



```
1 sample:
2   name: BME688 Sensor sample
3 tests:
4   sample.sensor.bme680:
5     harness: sensor
6     tags: samples sensor
7     integration_platforms:
8       - nrf52840dongle_nrf52840
9     platform_allow: nrf52840dongle_nrf52840
10    sample.sensor.bme688.i2c:
11      harness: console
12      tags: sensors
13      depends_on: i2c bme688
14      extra_args: "DTC_OVERLAY_FILE=nrf52840dongle_nrf52840.overlay"
15      harness_config:
16        type: one_line
17        regex:
18          - "temp: (.*); press: (.*); humidity: (.*)"
19        fixture: fixture_i2c_bme688
20    sample.usb.console:
21      depends_on: usb_device usb_cdc
22      tags: usb
23      platform_exclude: native_posix native_posix_64
24      harness: console
25      harness_config:
26        fixture: fixture_usb_cdc
```

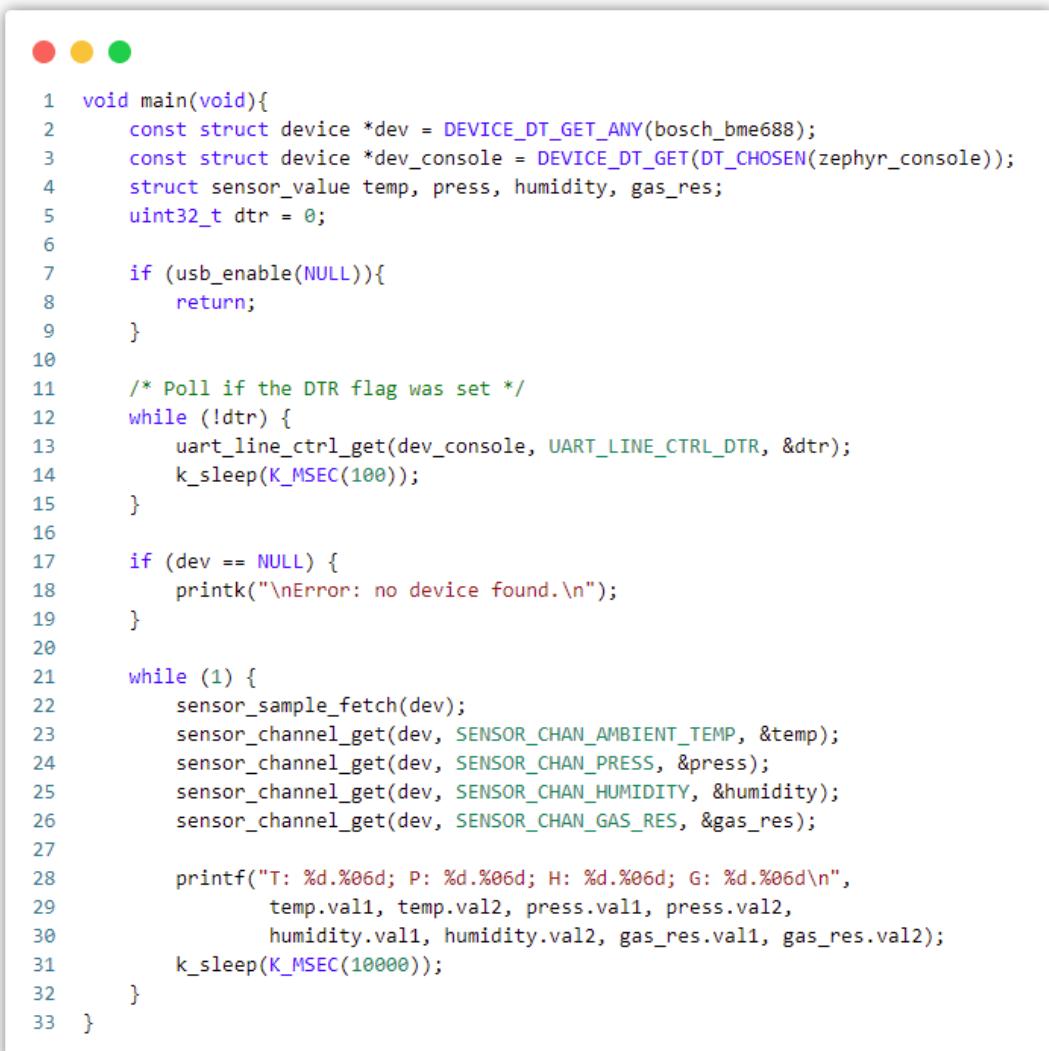
Фиг. 6.37. *sample.yaml*

main.c

```
1 #include <stdio.h>
2 #include <zephyr.h>
3 #include <device.h>
4 #include <devicetree.h>
5 #include <drivers/sensor.h>
6 #include <usb/usb_device.h>
7 #include <drivers/uart.h>
8 #include <sys/printk.h>
```

Фиг. 6.38. Включените библиотеки в main.c

На фиг. 6.38. са показани първите редове в програмата, които добавят всички необходими библиотеки.



```
1 void main(void){
2     const struct device *dev = DEVICE_DT_GET_ANY(bosch_bme688);
3     const struct device *dev_console = DEVICE_DT_GET(DT_CHOSEN(zephyr_console));
4     struct sensor_value temp, press, humidity, gas_res;
5     uint32_t dtr = 0;
6
7     if (usb_enable(NULL)){
8         return;
9     }
10
11    /* Poll if the DTR flag was set */
12    while (!dtr) {
13        uart_line_ctrl_get(dev_console, UART_LINE_CTRL_DTR, &dtr);
14        k_sleep(K_MSEC(100));
15    }
16
17    if (dev == NULL) {
18        printk("\nError: no device found.\n");
19    }
20
21    while (1) {
22        sensor_sample_fetch(dev);
23        sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temp);
24        sensor_channel_get(dev, SENSOR_CHAN_PRESS, &press);
25        sensor_channel_get(dev, SENSOR_CHAN_HUMIDITY, &humidity);
26        sensor_channel_get(dev, SENSOR_CHAN_GAS_RES, &gas_res);
27
28        printf("T: %.06d; P: %.06d; H: %.06d; G: %.06d\n",
29               temp.val1, temp.val2, press.val1, press.val2,
30               humidity.val1, humidity.val2, gas_res.val1, gas_res.val2);
31        k_sleep(K_MSEC(10000));
32    }
33 }
```

Фиг. 6.39. *main* функцията в *main.c* файла

На фиг. 6.39. е показана *main* функцията, в началото на която се дефинира *device* структура за вземане на съответния сензор от *devicetree*. Вземането на поинтер към възела на BME688 става чрез макрото **DEVICE_DT_GET_ANY**, което приема като аргумент compatibility параметър от *devicetree*, равняващ се в този случай на "bosch_bme688". Освен *device* структура за сензорът е необходима такава и за конзолата. Тя се дефинира като **dev_console** и има същите параметри, като тази, описана в BH1750 програмата. В програмата има и още една структура - **sensor_value**, която съдържа

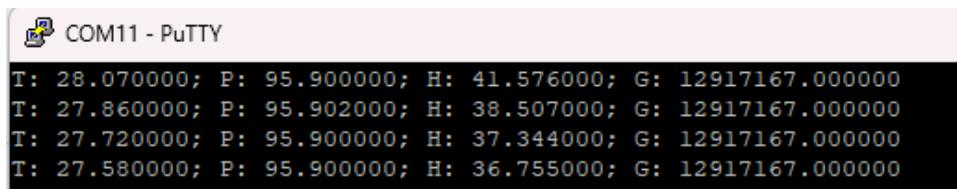
дannите от сензора, като ги разделя на две части - **val1** и **val2**. **val1** представлява стойността като цяло число, а **val2** е допълнителната дробна част, подобряваща точността на измерването. Цялостната стойност може да се вземе чрез формулатата:

$$\text{val1} + \text{val2} * 10^{(-6)} \quad (6.1)$$

След дефинирането на структурите следват проверки за включено USB и отворена серийна конзола, които са същите, като проверките в главния код на BH1750 сензора. Допълнително, има проверка за успешна инициализация на BME688 сензора.

Последно в main функцията има безкраен цикъл, който взима данните от сензора от съответния канал и ги запазва на адреса на съответната стойност в дефинираната структура по-горе. След това принтира взетите стойности като комбинирано число за всяка от измерваните величини. Цикълът се повтаря на всеки 10 секунди.

На фиг. 6.40. са показани данните от серийната конзола след качване на програмата на устройството.



```
COM11 - PuTTY
T: 28.070000; P: 95.900000; H: 41.576000; G: 12917167.000000
T: 27.860000; P: 95.902000; H: 38.507000; G: 12917167.000000
T: 27.720000; P: 95.900000; H: 37.344000; G: 12917167.000000
T: 27.580000; P: 95.900000; H: 36.755000; G: 12917167.000000
```

Фиг. 6.40. Изходните данни от програмата

6.2.5. Създаване на UDP Send функция

За изпращането на данните от основните модули към главния ще бъде използван транспортния протокол - UDP. Неговата имплементация ще бъде направена с помощта на библиотеката udp.h и ще бъде добавена към примера за Openthread конзола от т. 6.2.1. Тя ще се състои от функция за изпращане на "Hello Thread" съобщение на определен UDP port при натискане на вградения бутон на nRF52840-Dongle [108].

```
1 #include <zephyr.h>
2 #include <device.h>
3 #include <devicetree.h>
4 #include <drivers/gpio.h>
5 #include <zephyr/kernel.h>
6 #include <zephyr/drivers/uart.h>
7 #include <zephyr/usb/usb_device.h>
8 #include <openthread/thread.h>
9 #include <net/openthread.h>
10
11 #include <zephyr/logging/log.h>
12 #include <openthread/udp.h>
13
14 LOG_MODULE_REGISTER(cli_sample, CONFIG_OT_COMMAND_LINE_INTERFACE_LOG_LEVEL);
15
16 static const struct gpio_dt_spec button0_spec = GPIO_DT_SPEC_GET(DT_NODELABEL(button0), gpios);
17 static struct gpio_callback button0_cb;
```

Фиг. 6.41. Първите редове в програмата

На фиг. 6.41. са показани първите редове на програмата, които включват използваните библиотеки, макро за включването на логовете - **LOG_MODULE_REGISTER**, както и две структури. Първата структура е за инициализация на вградения бутон - **button0_spec**, а втората - за callback функционалност - **gpio_callback**.



```
1 static void udp_send(void){
2     otError error = OT_ERROR_NONE;
3     const char *buf = "Hello Thread";
4     otInstance *myInstance;
5     myInstance = openthread_get_default_instance();
6     otUdpSocket mySocket;
7
8     otMessageInfo messageInfo;
9     memset(&messageInfo, 0, sizeof(messageInfo));
10    otIp6AddressFromString("ff03::1", &messageInfo.mPeerAddr);
11    messageInfo.mPeerPort = 12346;
12
13    do{
14        error = otUdpOpen(myInstance, &mySocket, NULL, NULL);
15        if (error != OT_ERROR_NONE){ break; }
16        otMessage *test_Message = otUdpNewMessage(myInstance, NULL);
17        error = otMessageAppend(test_Message, buf, (uint16_t)strlen(buf));
18        if (error != OT_ERROR_NONE){ break; }
19        error = otUdpSend(myInstance, &mySocket, test_Message, &messageInfo);
20        if (error != OT_ERROR_NONE){ break; }
21        error = otUdpClose(myInstance, &mySocket);
22    }while(false);
23
24    if(error == OT_ERROR_NONE){
25        LOG_INF("Send.\n");
26    }else{
27        LOG_INF("udpSend error: %d\n", error);
28    }
29 }
```

Фиг. 6.42. Функцията *udp_send*

На фиг. 6.42. е показана функцията за изпращане на udp пакет. В нея има буфер ***buf**, съдържащ данните, които ще бъдат изпратени. Пойнтер към дефинираната Openthread инстанция - ***myInstance**, конфигурирана да взима инстанцията по подразбиране (аналогично на редовете, описани в т.6.2.2). **mySocket** репрезентира UDP цокъла. Структурата, репрезентираща локалния и peer IPv6 адрес на цокъла, е **messageInfo**. Функция **memset**, чрез която се подсигурява, че **messageInfo** структурата ще бъде празна. След това чрез функцията **otIp6AddressFromString** се задава IPv6 адресът, на който ще се

изпраща информацията. Избраният IPv6 адрес е ff03::1, тъй като в Thread мрежата това е multicast mesh-local адреса. Чрез него се изпраща информация на всички FTD и MED устройства в mesh мрежата (информация за multicast адресите е представена в т. 5.2.3., табл. 5.2.). После се задава UDP портът, с номер 12346, който ще се използва.

Следващата част от функцията ще бъде изпълнена в цикъл, изпълняващ се само веднъж, който е поставен, за да бъде подсигурено, че при грешка изпълнението на следващите стъпки ще бъде прекъснато. В този подсигуряващ цикъл протичат процесите: отваряне на UDP цокъла чрез функцията **otUdpOpen**; проверка дали цокълът е отворен успешно; създаване на UDP съобщение - ***test_Message**, към което след това ще бъдат добавени данните от буфера - **buf**, съдържащ информацията чрез функцията **otMessageAppend**, последван от проверка дали това е извършено успешно; изпращане на **otMessage** съобщението чрез функцията **otUdpSend**, последвана отново от проверка дали горната функция е била изпълнена успешно; затваряне на UDP цокъла, осъществено чрез функцията **otUdpClose**.

Последната част от **udp_send** функцията е проверка дали UDP съобщението е било изпратено успешно и съответно изпращане на съобщение за потвърждение или за грешка към конзолата.

```
● ● ●  
1 void button_pressed_callback(const struct device *gpiob,  
2                                 struct gpio_callback *cb,  
3                                 gpio_port_pins_t pins){  
4     udp_send();  
5 }
```

Фиг. 6.43. Функцията *button_pressed_callback*

На фиг. 6.43. е показана функцията за натискане на бутона - **button_pressed_callback**. При натискане на бутон тя извиква **udp_send** функцията и изпраща UDP съобщение.

```
● ○ ●
1 void main(void)
2 {
3     gpio_pin_configure_dt(&button0_spec, GPIO_INPUT);
4     gpio_pin_interrupt_configure_dt(&button0_spec, GPIO_INT_EDGE_TO_ACTIVE);
5     gpio_init_callback(&button0_cb, button_pressed_callback, BIT(button0_spec.pin));
6     gpio_add_callback(button0_spec.port, &button0_cb);
7
8     int err = usb_enable(NULL);
9     const struct device *dev;
10    uint32_t dtr = 0;
11
12    if (err != 0) {
13        LOG_ERR("Failed to enable USB");
14        return;
15    }
16
17    dev = DEVICE_DT_GET(DT_CHOSEN(zephyr_shell_uart));
18    if (dev == NULL) {
19        LOG_ERR("Failed to find specific UART device");
20        return;
21    }
22
23    /* Data Terminal Ready - check if host is ready to communicate */
24    while (!dtr) {
25        err = uart_line_ctrl_get(dev, UART_LINE_CTRL_DTR, &dtr);
26        if (err) {
27            LOG_ERR("Failed to get Data Terminal Ready line state: %d", err);
28            continue;
29        }
30        k_msleep(100);
31    }
32 }
```

Фиг. 6.44. *main* функцията

На фиг. 6.44. е показан сурс кодът на *main* функцията в програмата, като повечето функционалности в нея са вече описани в т. 6.2.3 и т. 6.2.4. Новата част, която я няма в миналите програми е конфигурацията на бутона, която се изпълнява с функциите:

- **gpio_pin_configure_dt** - конфигурира GPIO (на англ. *General Purpose Input/Output*) извода на бутона като вход

- **gpio_pin_interrupt_configure_dt** - включва interrupt на GPIO интерфейса и го конфигурира по такъв начин, че при промяна на логическото ниво на единица да бъде задействан.

- **gpio_init_callback** - функция, помагаща за правилното инициализиране на структурата button0_cb.

- **gpio_add_callback** - добавя обратна връзка на бутона.

На фиг. 6.45. е показан изходът от сериината конзола при натискане на бутон. На фиг. 6.46. е показано как данните са получени на вече конфигуриран BR (на англ. *Border Router*, на бълг. *Границен маршрутизатор*).

```
[00:00:19.945,556] <inf> cli_sample: Send.
```

Фиг. 6.45. Изход на сериината конзола, при натискане на бутона

```
I (29524) esp_ot_br: Received 12 bytes from FDDE:AD00:BEEF:0:4F3F:C04:22A:3EFA:  
I (29524) esp ot br: Hello Thread
```

Фиг. 6.46. Получена информация от устройството, визуализирана в конзолата на Border Router

6.3. Създаване на софтуер за управляващия възел

В тази подточка ще бъде описан процесът за създаване на софтуер за управляващия възел.

6.3.1. Конфигуриране на nRF52840-Dongle като RCP

За да може да бъде създаден Border Router с NodeMCU-32S ще бъде необходимо nRF52840-Dongle да бъде конфигуриран като RCP [109].

Накратко за Radio Co-processor дизайна:

В дизайна на RCP, ядрото на OpenThread се намира на хост процесора, а на устройството със самото Thread радио има само минимален "контролер" на MAC слоя. При този дизайн хост-процесорът обикновено не спи, за да се гарантира надеждност на Thread мрежата.

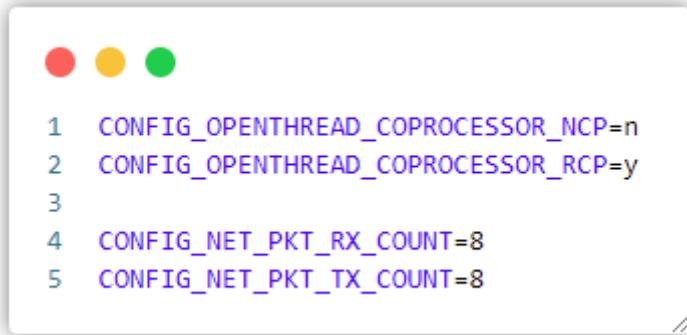
Предимството на конфигурирането като RCP е, че OpenThread може да използва ресурсите на по-мощния процесор. Този дизайн се явява полезен за устройства, които са по-малко чувствителни към ограничения на мощността, тъй като радиото е постоянно включено.

За да бъде конфигуриран nRF52840 Dongle като RCP ще бъде необходимо да се променят изброените по-долу файлове от наличния coprocessor пример в директорията -

C:\Diplomna\ncs\v2.1.2\nrf\samples\openthread\coprocessor.

Финалната версия на променените файлове ще е следната:

overlay-rcp.conf

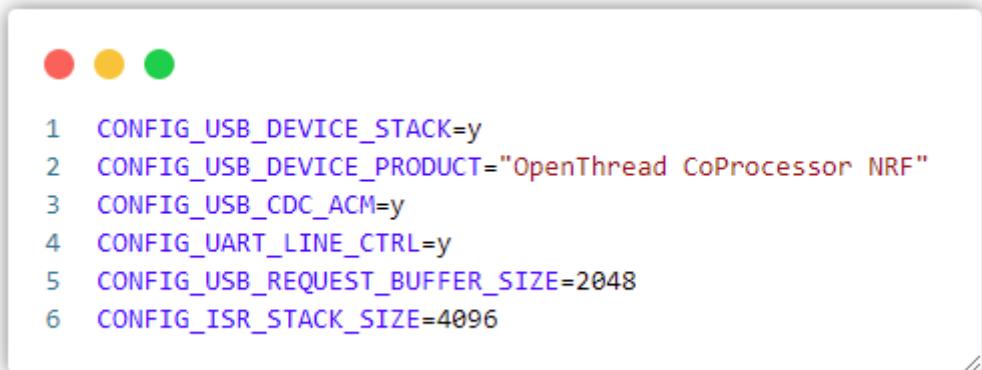


```
1 CONFIG_OPENTHREAD_COPROCESSOR_NCP=n
2 CONFIG_OPENTHREAD_COPROCESSOR_RCP=y
3
4 CONFIG_NET_PKT_RX_COUNT=8
5 CONFIG_NET_PKT_TX_COUNT=8
```

Фиг. 6.47. *overlay-rcp.conf*

На фиг. 6.47. е показан *overlay-rcp.conf* файлът, в който е важно **CONFIG_OPENTHREAD_COPROCESSOR_RCP** опцията да бъде включена.

overlay-usb-nrf-br.conf



```
1 CONFIG_USB_DEVICE_STACK=y
2 CONFIG_USB_DEVICE_PRODUCT="OpenThread CoProcessor NRF"
3 CONFIG_USB_CDC_ACM=y
4 CONFIG_UART_LINE_CTRL=y
5 CONFIG_USB_REQUEST_BUFFER_SIZE=2048
6 CONFIG_ISR_STACK_SIZE=4096
```

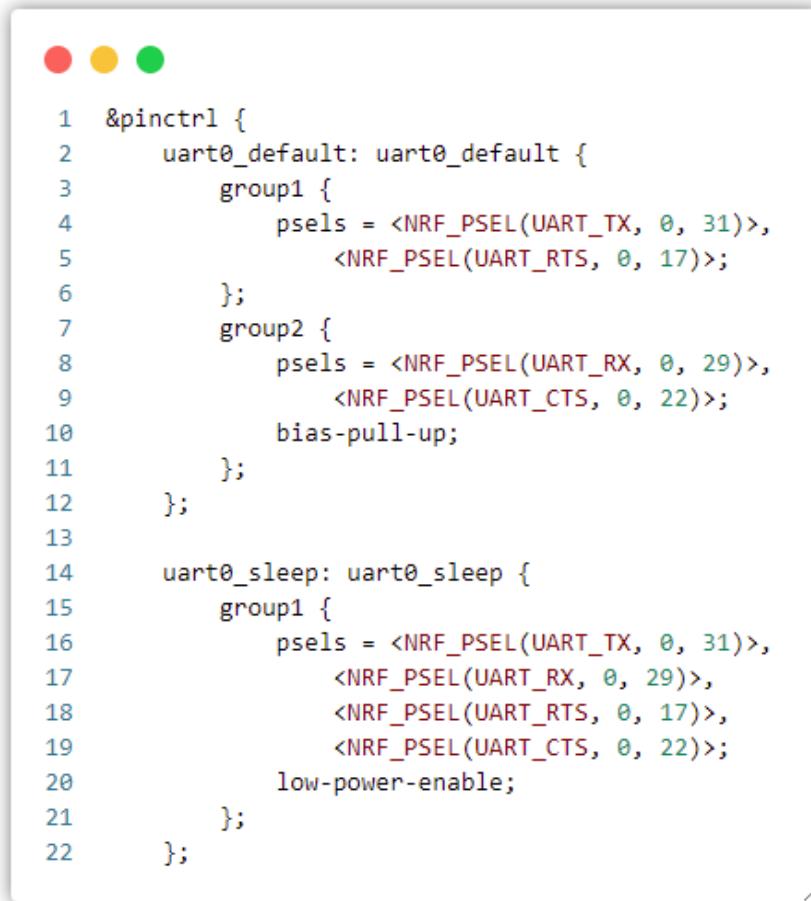
Фиг. 6.48. *overlay-usb-nrf-br.conf*

prj.conf

```
● ● ●  
1 CONFIG_NETWORKING=y  
2 CONFIG_NET_L2_OPENTHREAD=y  
3  
4 # Disable certain parts of Zephyr IPv6 stack  
5 CONFIG_NET_IPV6_NBR_CACHE=n  
6 CONFIG_NET_IPV6_MLD=n  
7  
8 # Logging  
9 CONFIG_LOG=y  
10 CONFIG_LOG_MAX_LEVEL=1  
11 CONFIG_LOG_MODE_MINIMAL=n  
12 CONFIG_LOG_BACKEND_SPINEL=y  
13 CONFIG_BOOT_BANNER=n  
14  
15 # Kernel options  
16 CONFIG_MAIN_STACK_SIZE=2560  
17 CONFIG_INIT_STACKS=y  
18  
19 # Increase logging thread stack size due to Spinel backend needs  
20 CONFIG_LOG_PROCESS_THREAD_STACK_SIZE=2048  
21  
22 CONFIG_OPENTHREAD_COPROCESSOR=y  
23 CONFIG_ENTROPY_NRF5_RNG=y
```

Фиг. 6.49. prj.conf

nrf52840dongle_nrf52840-pinctrl.dtsi



```
1 &pinctrl {
2     uart0_default: uart0_default {
3         group1 {
4             psels = <NRF_PSEL(UART_TX, 0, 31)>,
5                     <NRF_PSEL(UART_RTS, 0, 17)>;
6         };
7         group2 {
8             psels = <NRF_PSEL(UART_RX, 0, 29)>,
9                     <NRF_PSEL(UART_CTS, 0, 22)>;
10            bias-pull-up;
11        };
12    };
13
14    uart0_sleep: uart0_sleep {
15        group1 {
16            psels = <NRF_PSEL(UART_TX, 0, 31)>,
17                    <NRF_PSEL(UART_RX, 0, 29)>,
18                    <NRF_PSEL(UART_RTS, 0, 17)>,
19                    <NRF_PSEL(UART_CTS, 0, 22)>;
20            low-power-enable;
21        };
22    };
23}
```

Фиг. 6.50. Част от nrf52840dongle_nrf52840-pinctrl.dtsi файла, показваща uart0_default и uart0_sleep

На фиг. 6.50. е показана част от .dtsi файла, намиращ се в директорията - **C:/Diplomna/ncs/v2.1.2/zephyr/boards/arm/nrf52840dongle_nrf52840/**. В него са променени TX и RX номерата на пиновете на uart0 възлите.

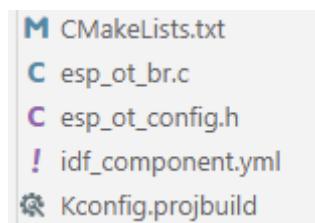
В build конфигурацията на coprocessor примера трябва да се добавят overlay-rcp.conf и overlay-usb-nrf-br.conf като Kconfig фрагменти.

6.3.2. Създаване на работещ Border Router

Това се явява първия пример, който ще бъде разработван с ESP-IDF и ще бъде предназначен за NodeMCU-32S. Той ще представлява основна конфигурация за работещ граничен маршрутизатор.

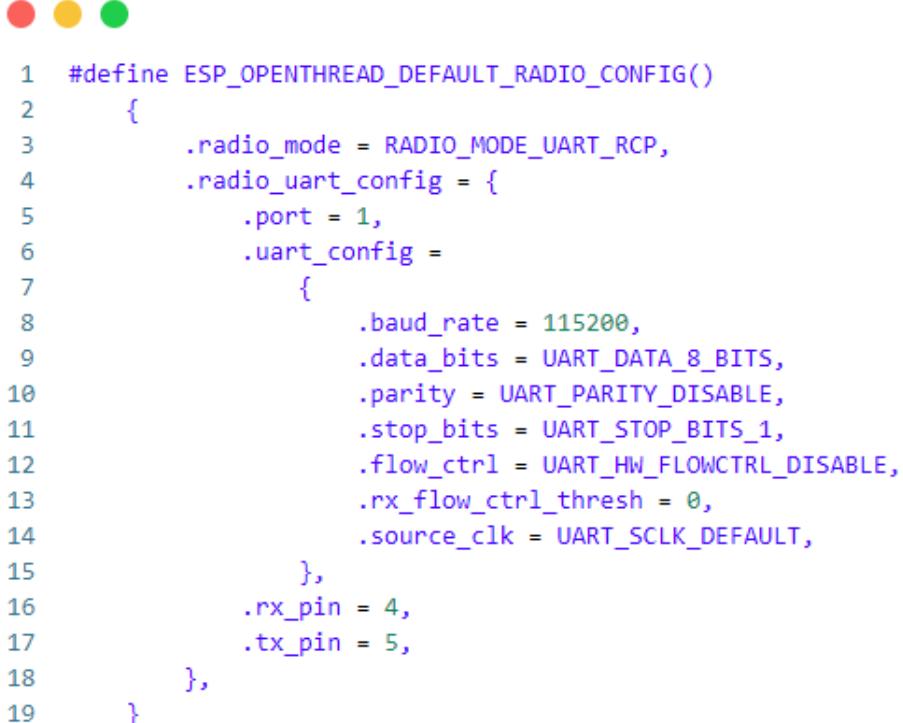
Първата стъпка за създаването на програмата е да бъде копирана примерната програма `ot_br`, намираща се в директорията - **C:\Diplomna\esp-idf\examples\openthread** в работната папка esp32-apps.

В root директорията на програмата има CMakeLists.txt, който има същата функционалност като CMakeLists.txt файловете на досегашните програми за nRF52840-Dongle-a, sdkconfig.defaults, който съдържа Kconfig настройките, и partitions.csv, който се използва в ESP-IDF програмите за определяне на partition таблица на флаш паметта. Накратко, partitions.csv файлът определя разположението на дяловете във флаш паметта. Освен всички тези файлове има и папка **main**, която съдържа управляващия код и допълнителни файлове за конфигурация. Файловете в нея са показани на фиг. 6.51.



Фиг. 6.51. Файлове в папката *main*

esp_ot_config.h



```
1 #define ESP_OPENTHREAD_DEFAULT_RADIO_CONFIG()
2 {
3     .radio_mode = RADIO_MODE_UART_RCP,
4     .radio_uart_config = {
5         .port = 1,
6         .uart_config =
7         {
8             .baud_rate = 115200,
9             .data_bits = UART_DATA_8_BITS,
10            .parity = UART_PARITY_DISABLE,
11            .stop_bits = UART_STOP_BITS_1,
12            .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
13            .rx_flow_ctrl_thresh = 0,
14            .source_clk = UART_SCLK_DEFAULT,
15        },
16        .rx_pin = 4,
17        .tx_pin = 5,
18    },
19 }
```

Фиг. 6.52. Част от esp_ot_config.h

На фиг. 6.52. е показана част от файла `esp_ot_config.h`, показваща конфигурации на OpenThread радиото. От тях единствено ще бъдат от значение `.rx_pin` и `.tx_pin`, чрез които се дефинират RX и TX изводите към които ще бъде свързан nRF52840-Dongle в конфигурация на RCP.

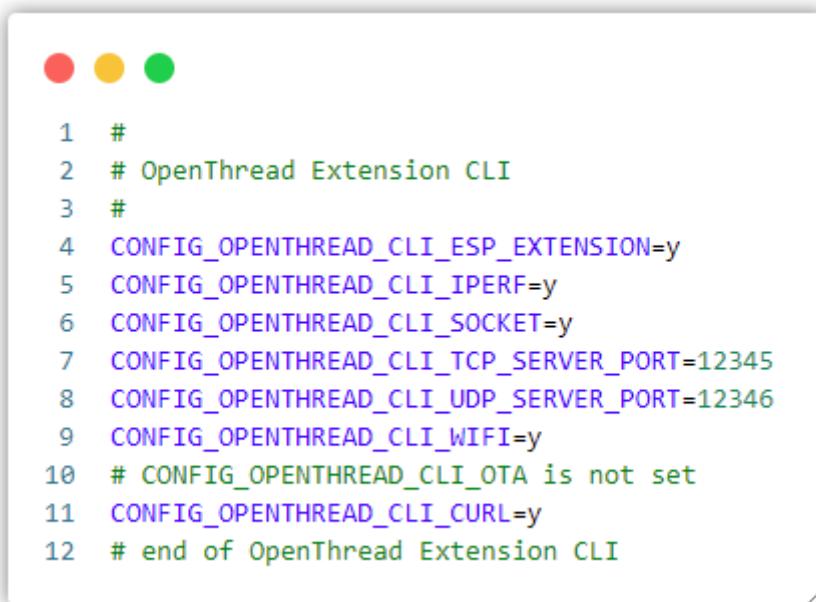
За работа на програмата не е нужно да се променя нищо по примера, а единствено трябва да се построи и качи на устройството.

6.3.3. Отваряне на UDP port за приемане на данни

Изпратените данни от основния модул достигат до всички устройства в OpenThread мрежата (подробно обяснение в т. 6.2.5.), но нито едно от тях не е конфигурирано да ги приема. Поради причината, че данните, изпращани от основните възли, трябва да стигат единствено управляващия възел, само той ще бъде конфигуриран да ги приема и визуализира на серийната си конзола. В esp-idf има готов пример за UDP сървър, от който ще бъде имплементирана основната функционалност [110].

За имплементирането на тази функционалност ще е необходимо да бъдат променени следните файлове във вече създадената в т.6.3.2 програма **ot_br** - sdkconfig и esp_ot_br.c.

sdkconfig



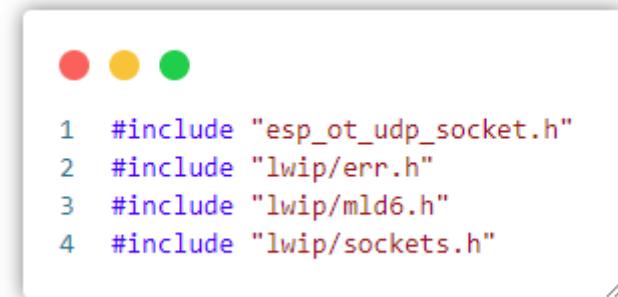
```
1 #
2 # OpenThread Extension CLI
3 #
4 CONFIG_OPENTHREAD_CLI_ESP_EXTENSION=y
5 CONFIG_OPENTHREAD_CLI_IPERF=y
6 CONFIG_OPENTHREAD_CLI_SOCKET=y
7 CONFIG_OPENTHREAD_CLI_TCP_SERVER_PORT=12345
8 CONFIG_OPENTHREAD_CLI_UDP_SERVER_PORT=12346
9 CONFIG_OPENTHREAD_CLI_WIFI=y
10 # CONFIG_OPENTHREAD_CLI_OTA is not set
11 CONFIG_OPENTHREAD_CLI_CURL=y
12 # end of OpenThread Extension CLI
```

Фиг. 6.53. Част от sdkconfig файла предназначена за конфигурация на OpenThread функционалности

В sdkconfig се съдържат Kconfig опции за OpenThread, като една от тях е **CONFIG_OPENTHREAD_CLI_UDP_SERVER_PORT**. Чрез нея

ще бъде зададен номера на UDP порта, който ще бъде използван в по-нататъшната конфигурация. На фиг. 6.53 е показана тази част от sdkconfig файла.

esp_ot_br.c



Фиг. 6.54. Библиотеки, които ще се използват за конфигуриране на UDP цокъл сървъра

На фиг. 6.54. са показани библиотеките, които ще бъдат използвани за конфигуриране на UDP цокъл сървъра.

```

1  char rx_buffer[128];
2  int len;
3
4  static void udp_socket_server_task(void *pvParameters)
5  {
6      int err = 0;
7      esp_err_t ret = ESP_OK;
8      char addr_str[128];
9      int listen_sock;
10     int port = CONFIG_OPENTHREAD_CLI_UDP_SERVER_PORT;
11
12     struct timeval timeout = {0};
13     struct sockaddr_storage source_addr;
14     struct sockaddr_in6 listen_addr;
15
16     inet6_aton("::", &listen_addr.sin6_addr);
17     listen_addr.sin6_family = AF_INET6;
18     listen_addr.sin6_port = htons(port);
19
20     listen_sock = socket(AF_INET6, SOCK_DGRAM, IPPROTO_UDP);
21     ESP_GOTO_ON_FALSE((listen_sock >= 0), ESP_OK, exit, TAG, "Unable to create socket: errno %d", errno);
22     ESP_LOGI(TAG, "Socket created");
23
24     int opt = 1;
25     setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
26     setsockopt(listen_sock, IPPROTO_IPV6, IPV6_V6ONLY, &opt, sizeof(opt));
27
28     err = bind(listen_sock, (struct sockaddr *)&listen_addr, sizeof(listen_addr));
29     ESP_GOTO_ON_FALSE((err == 0), ESP_FAIL, exit, TAG, "Socket unable to bind: errno %d", errno);
30     ESP_LOGI(TAG, "Socket bound, port %d", port);
31     ESP_LOGI(TAG, "Waiting for data, no timeout");

```

Фиг. 6.55. Част от esp_ot_br.c, в която е показана първа част на функцията udp_socket_server_task

На фиг. 6.55. е показана първата част от функцията за отваряне на UDP socket сървър - **udp_socket_server_task**. В началото ѝ има дефинирана променлива **err**, която ще съдържа бъдещи кодове за грешки, променлива **ret**, предназначена за съхраняването на връщаната стойност (return value). Дефиниран е масив - **addr_str**, който ще бъде използван за съхраняване на source IPv6 адреса на пристигащите данни. Има променлива, която ще бъде използвана за създаване на самия UDP цокъл - **listen_sock**, както и променлива **port**, съдържаща, вече дефинираната в sdkconfig файла, стойност на UDP порта. Структурите **source_addr** и **listen_addr** ще се използват

съответно за запазване на адреса, от който идва информацията, и адреса, на който сървъра ще слуша за информация.

Функцията **inet6_aton** ще бъде използвана за задаване на стойността на **listen_addr** на неуточнен адрес ('::'). След нея се дефинира семейството на адреса - **AF_INET6** и се задава стойност на порта. После се създава цокъл с функцията **socket**, съдържаща адресното семейство, типа и протокола. Ако създаването на цокъла се провали ще излезе съобщение за грешка в конзолата и функцията ще отиде на **exit** етикета - **ESP_GOTO_ON_FALSE**. При успешна инициализация, от друга страна, ще се изпрати съобщение в конзолата за успешно създаване - **ESP_LOGI**.

Променливата **opt** съдържа стойност единица и ще бъде използвана за задаване на долните две настройки. Първата има параметър **SO_REUSEADDR**, който позволява цокъла да се свърже с предварително използван адрес, а втората има параметър **IPV6_V6ONLY**, който изключва двустаковата (dual-stack) настройка, позволяваща на IPv4 и IPv6 да работят едновременно.

След това се извиква функцията **bind**, за да се свърже цокълът с определен адрес и порт. Ако операцията за свързване не е успешна кодът излиза от функцията, като прескача до етикета за изход.

```

1   while(1){
2       timeout.tv_sec = 0;
3       setsockopt(listen_sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
4       socklen_t socklen = sizeof(source_addr);
5       len = recvfrom(listen_sock, rx_buffer, sizeof(rx_buffer) - 1, 0, (struct sockaddr *)&source_addr, &socklen);
6
7       ESP_GOTO_ON_FALSE((len >= 0), ESP_FAIL, exit, TAG, "recvfrom failed: errno %d", errno);
8       // Data received
9       // Get the senders ip address as string
10      inet6_ntoa_r((struct sockaddr_in6 *)&source_addr)->sin6_addr, addr_str, sizeof(addr_str) - 1);
11
12      rx_buffer[len] = 0;
13      ESP_LOGI(TAG, "Received %d bytes from %s:", len, addr_str);
14      ESP_LOGI(TAG, "%s", rx_buffer);
15  }
16 exit:
17  if (ret != ESP_OK) {
18      shutdown(listen_sock, 0);
19      close(listen_sock);
20  }
21  ESP_LOGI(TAG, "Socket server is closed.");
22  vTaskDelete(NULL);
23 }

```

*Фиг. 6.56. Част от esp_ot_br.c, на която е показана втора част на функцията **udp_socket_server_task***

На фиг. 6.56. е показана последната част на функцията - **udp_socket_server_task**.

Тук е показано как кодът влиза в безкраен цикъл и чака получаването на входящи данни на цокъла. В него първо се задава нулева стойност на времето за прекъсване при получаване чрез функцията **setsockopt**, а след това се изчаква пристигането на данни с помощта на функцията **recvfrom**. Получените данни се съхраняват в масива **rx_buffer**, а дължината на получените данни се съхранява в променливата **len**.

Ако се появи грешка по време на получаването програмата ще прескочи на етикета за изход. Ако няма грешка програмата продължава, като следващата стъпка е научаването на IPv6 адреса на подателя, който се преобразува в четим низ от символи с помощта на функцията **inet6_ntoa_r**. След това получените данни и адресът на подателя биват изписани на конзолата.

Най-отдолу на функцията е показан етикетът за изход, който се извиква, ако по време на функцията е изникнала грешка. Той затваря цокъла и изтрива задачата от работещата програма с помощта на функцията **vTaskDelete**.



```
1 xTaskCreate(udp_socket_server_task, "ot_udp_socket_server", 4096, xTaskGetCurrentTaskHandle(), 4, NULL);
```

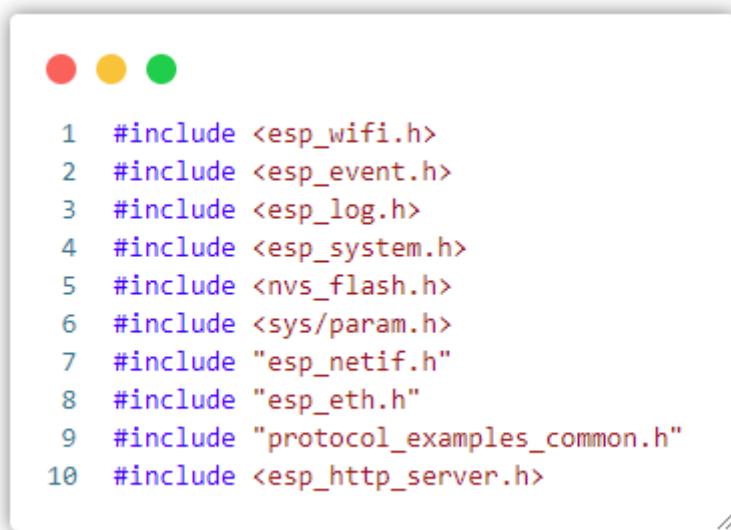
Фиг. 6.57. Редът, който се вкарва в **app_main** функцията на **esp_ot_br.c**, за инициализация на UDP цокъла

На фиг. 6.57 е показан редът, вкаран в **app_main** функцията, чрез който се стартира UDP цокъла. Това става чрез използването на функцията **xTaskCreate**, която създава нов FreeRTOS задача (task) [111]. **xTaskCreate** има следните параметри:

- Функция, която ще бъде повикана - **udp_socket_server_task**
- Име на задачата - **ot_udp_socket_server**
- Паметта, която ще бъде заделена за задачата - 4096 думи
- handle за задачата, която ще се използва като родителска задача, повикан с функцията **xTaskGetCurrentTaskHandle**
 - Приоритет - 4
 - Избирателен параметър, използван за подаване на handle на вече създадената задача - **NULL** (не се използва)

6.3.4. Създаване и конфигуриране на WebSocket

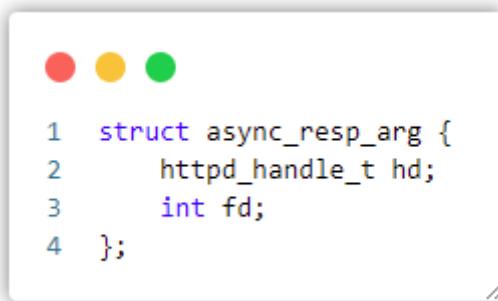
Данните, изпратени към управляващия модул, тряба да бъдат достъпни поне за устройствата, работещи в същата мрежа. WebSocket ще позволи достъп до данните чрез въвеждане на локалното IP на NodeMCU-32S в браузър, последвано от URI (на англ. *Uniform Resource Identifier*). За конфигурирането на основната функционалност на WebSocket сървър ще бъде необходимо добавянето на кода, показан от фиг. 6.57., 6.58., 6.59., 6.60., 6.61., 6.62., 6.63., към основния esp_ot_br.c файл в програмата за граничния рутер, описан в т.6.3.2. [112][113].



```
● ● ●

1 #include <esp_wifi.h>
2 #include <esp_event.h>
3 #include <esp_log.h>
4 #include <esp_system.h>
5 #include <nvs_flash.h>
6 #include <sys/param.h>
7 #include "esp_netif.h"
8 #include "esp_eth.h"
9 #include "protocol_examples_common.h"
10 #include <esp_http_server.h>
```

Фиг. 6.57. Библиотеките, които тряба да бъдат добавени, за конфигуриране на WebSocket сървър.



```
● ● ●

1 struct async_resp_arg {
2     httpd_handle_t hd;
3     int fd;
4 };
```

Фиг. 6.58. Структура, съдържаща данните за асинхронен отговор

На фиг. 6.58. е показана структурата, съдържаща данните за асинхронния отговор. В нея **hd** е сървърната инстанция, а **fd** е file descriptor на цокъл сесията.

```
1 static void ws_async_resp(void *arg)
2 {
3     char http_str[250];
4     char *data_str = &rx_buffer;
5     sprintf(http_str, "HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\n", strlen(data_str));
6     struct async_resp_arg *resp_arg = (struct async_resp_arg *)arg;
7     httpd_handle_t hd = resp_arg->hd;
8     int fd = resp_arg->fd;
9     ESP_LOGI(TAG, "Executing queued work fd: %d", fd);
10    httpd_socket_send(hd, fd, http_str, strlen(http_str), 0);
11    httpd_socket_send(hd, fd, data_str, strlen(data_str), 0);
12    free(arg);
13 }
```

Фиг. 6.59. *ws_async_resp* функцията

На фиг. 6.59. е показана функцията **ws_async_resp**, която изпраща HTTP отговор асинхронно. В нея биват форматирани данните, които ще бъдат изпратени, прави се инстанция на структурата, съдържаща данните за асинхронния отговор, изпращат се самите данни и се освобождаване на паметта, заемана от променливата **arg**.

```
1 static esp_err_t async_get_handler(httpd_req_t *req)
2 {
3     struct async_resp_arg *resp_arg = malloc(sizeof(struct async_resp_arg));
4     resp_arg->hd = req->handle;
5     resp_arg->fd = httpd_req_to_sockfd(req);
6     httpd_queue_work(req->handle, ws_async_resp, resp_arg);
7     return ESP_OK;
8 }
```

Фиг. 6.60. *async_get_handler* функцията

На фиг. 6.60. е показана функцията - **async_get_handler**, чрез която асинхронно се обработват HTTP GET заявките, позволявайки по този начин full-duplex комуникация. В нея се заделя памет за **async_resp_arg** структурата, дефинират се необходимите параметри и се стартира опашката с **ws_async_resp** handler.

```
1 static const httpd_uri_t ws = {  
2     .uri      = "/ws",  
3     .method   = HTTP_GET,  
4     .handler  = async_get_handler,  
5     .user_ctx = NULL,  
6     .is_websocket = true  
7 };
```

Фиг. 6.61. Инстанция на структурата *httpd_uri_t*

На фиг. 6.61. е показана инстанция на структурата *httpd_uri_t*, чрез която се образува "/ws" URI за сървъра, който ще бъде добавен в края на локалното IP (Примерно: 192.168.0.106/ws).

```
1 static void start_webserver(void)  
2 {  
3     httpd_handle_t server = NULL;  
4     httpd_config_t config = HTTPD_DEFAULT_CONFIG();  
5     ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port);  
6     httpd_start(&server, &config);  
7     httpd_register_uri_handler(server, &ws);  
8 }
```

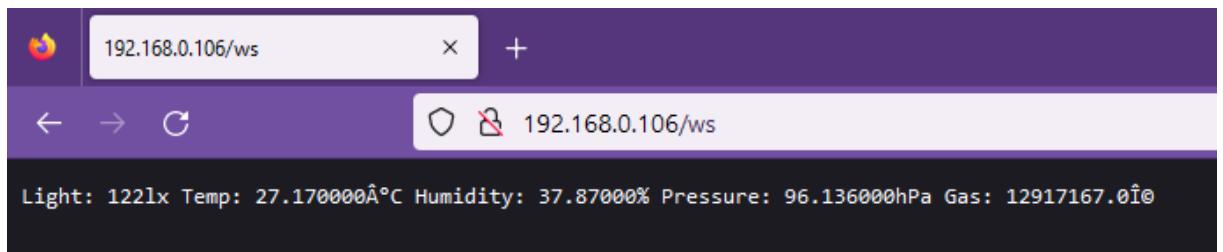
Фиг. 6.62. *start_webserver* функцията

На фиг. 6.62. е показана функцията, която стартира WebSocket сървъра - **start_webserver**.



Фиг. 6.63. Повикване на *start_webserver* функцията в *app_main*

На фиг. 6.63. е показано стартирането на WebSocket сървъра чрез повикване на функцията *start_webserver* в *app_main* функцията.



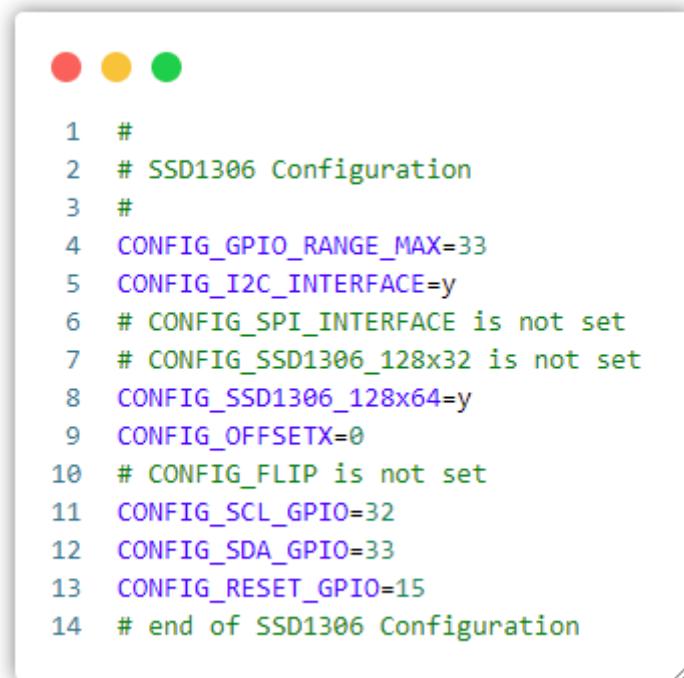
Фиг. 6.64. Изглед към *WebSocket* страницата

На фиг. 6.64. е показана страницата на WebSocket сървъра, показваща получени данни. Тя може да се достъпи чрез въвеждане на локалния IP адрес, зададен на NodeMCU-32S, и добавяне на URI - "/ws" към него в избран от потребителя браузър.

6.3.5. Свързване и конфигуриране на SSD1306 Display

За бързо виждане на важна информация относно мрежата се използва SSD1306 дисплей. Поради голямото разпространение на модела беше намерена библиотека, която улеснява работата с него - [114]. За създаването на софтуер за неговото управление ще бъде необходима промяната на следните файлове:

sdkconfig



```
1  #
2  # SSD1306 Configuration
3  #
4  CONFIG_GPIO_RANGE_MAX=33
5  CONFIG_I2C_INTERFACE=y
6  # CONFIG_SPI_INTERFACE is not set
7  # CONFIG_SSD1306_128x32 is not set
8  CONFIG_SSD1306_128x64=y
9  CONFIG_OFFSETX=0
10 # CONFIG_FLIP is not set
11 CONFIG_SCL_GPIO=32
12 CONFIG_SDA_GPIO=33
13 CONFIG_RESET_GPIO=15
14 # end of SSD1306 Configuration
```

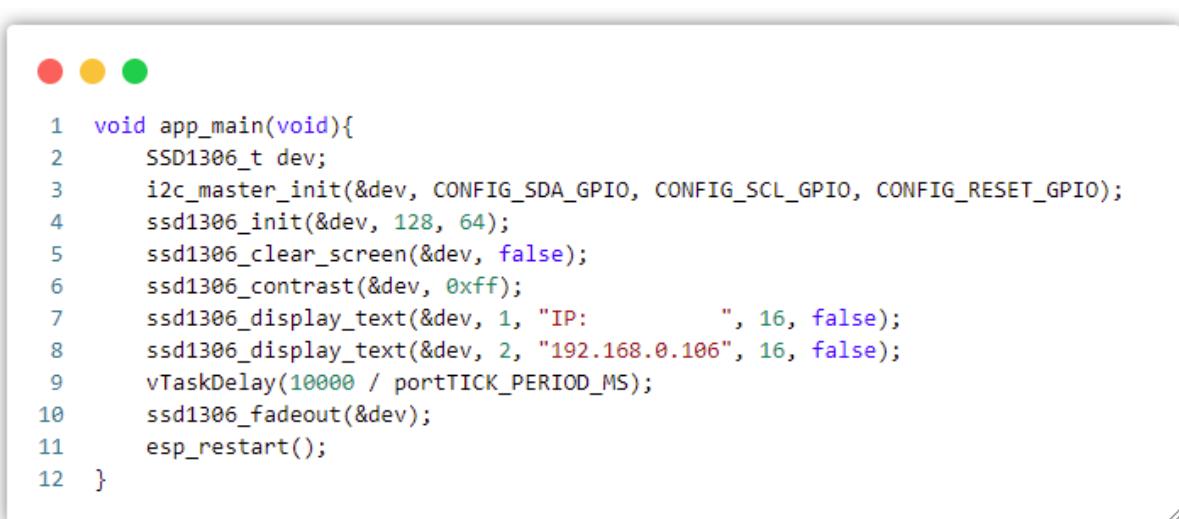
Фиг. 6.65. Част от *sdkconfig* файла, предназначена за конфигурация на съответния дисплей.

main.c



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "freertos/FreeRTOS.h"
5 #include "freertos/task.h"
6 #include "ssd1306.h"
```

Фиг. 6.66. Библиотеките, които ще бъдат необходими, за създаване на примерна програма за работа с дисплея.



```
1 void app_main(void){
2     SSD1306_t dev;
3     i2c_master_init(&dev, CONFIG_SDA_GPIO, CONFIG_SCL_GPIO, CONFIG_RESET_GPIO);
4     ssd1306_init(&dev, 128, 64);
5     ssd1306_clear_screen(&dev, false);
6     ssd1306_contrast(&dev, 0xff);
7     ssd1306_display_text(&dev, 1, "IP:      ", 16, false);
8     ssd1306_display_text(&dev, 2, "192.168.0.106", 16, false);
9     vTaskDelay(10000 / portTICK_PERIOD_MS);
10    ssd1306_fadeout(&dev);
11    esp_restart();
12 }
```

Фиг. 6.67. *app_main* функцията в програмата

На фиг. 6.67. е показана основната функция на програмата. В нея първоначално е дефинирана структура, представляща дисплея. След това е инициализирана I²C комуникация с него чрез функцията **i2c_master_init**. На следващия ред е инициализиран самият дисплей с резолюция 128 на 64 пиксела - **ssd1306_init**. После, дисплеят бива изчищен с функцията **ssd1306_clear_screen**, последвана от

функцията **ssd1306_contrast**, настройваща контраста на дисплея, в случая - 0xff, което е максималната стойност.

На следващите два реда се използва функцията **ssd1306_display_text**, чрез която се изписват два реда данни. Това е последвано от 10 секундно забавяне и избледняване на дисплея чрез функцията **ssd1306_fadeout**.

Последният ред в основната функция служи за рестартиране на микроконтролера.

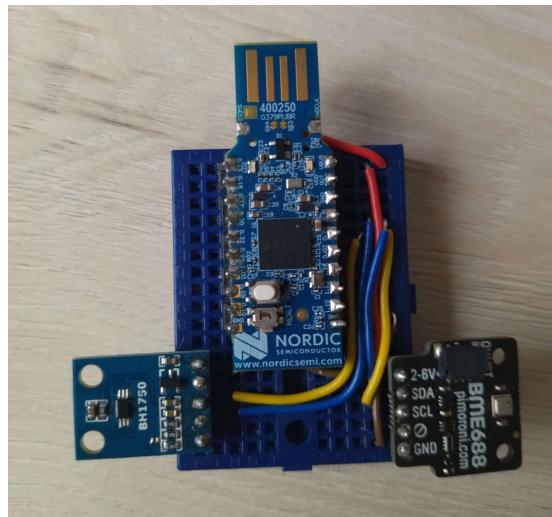
6.4. Процес на създаване на елементите на работещ прототип

6.4.1. Реализация на макети на прототипни платки

За реализирането на проекта бяха направени няколко различни макета на прототипни платки. Всяка от тях тества различна функционалност или оптимизира предишната.

- Прототипна платка 1

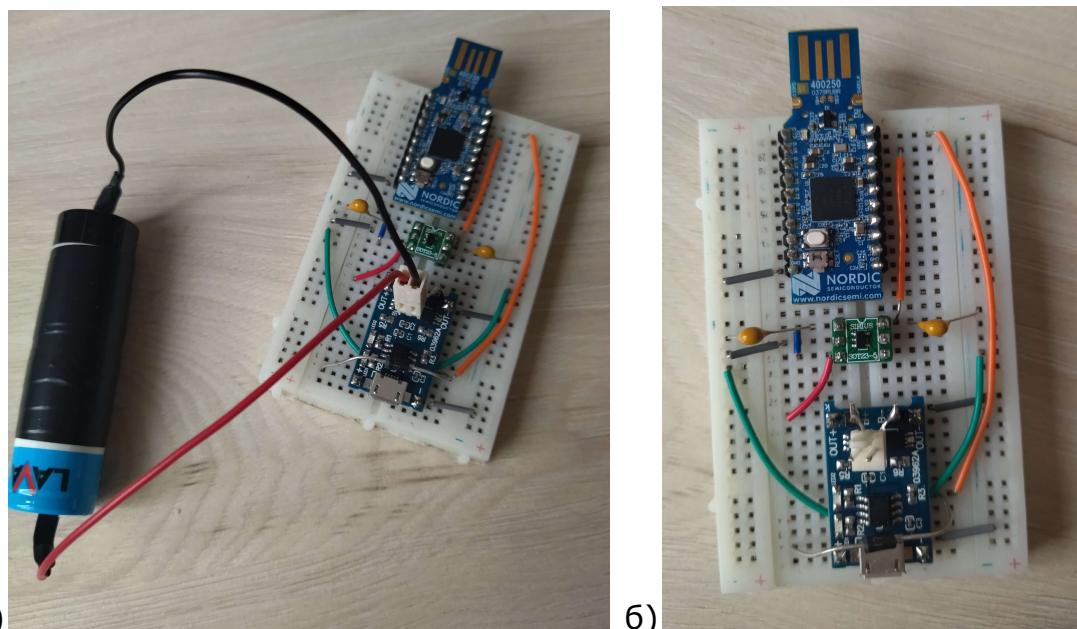
На фиг. 6.68. е показана първата прототипна платка, на която са монтирани nRF52840-Dongle и два сензора - BH1750 и BME688.



Фиг. 6.68. Прототипна платка 1

- Прототипна платка 2

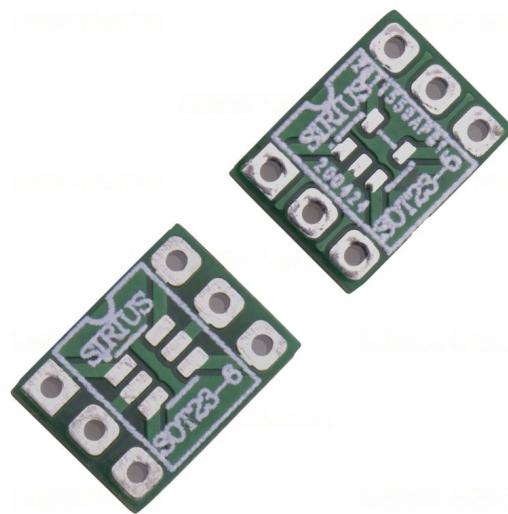
На фиг. 6.69.а) и фиг. 6.69.б) е показана следващата прототипна платка, на която е реализирано външно захранване на модула, за да може устройството да стане автономно и да не е необходимо да бъде захранвано от компютър през цялото време.



Фиг. 6.69.а) Прототипна платка 2 с батерия, б) Прототипна платка 2 без батерия

За съставянето на тази платка беше необходимо да се запои регулатора на напрежение на преходна платка поради причината, че има такъв линеен регулатор само в SMD вариант (стандарт SOT-23-5).

За тази цел беше закупена необходимата преходна платка към DIP6. На фиг. 6.70. е показана самата преходна платка, а на фиг. 6.71. е показан процесът на запояването на интегралната схема към нея [115].



Фиг. 6.70. Преходна платка от SOT-23-5 към DIP6, закупена от Sirius-PCB



Фиг. 6.71. Процес на запояване на SMD интегралната схема под лупа

Също така в документацията на nRF52840-Dongle-а е описано, че, за да се използва външно захранване, е необходимо връзката SB2 да бъде разделена и SB1 - запоена. Инструкциите са показани на фиг. 6.72. [116].

External regulated source

The nRF52840 Dongle can also be configured to be supplied from an external regulated 1.8–3.6 V source through the **VDD OUT** connection point. To enable this, **SB2** must be cut and **SB1** must be soldered.

CAUTION:



Do not have both **SB1** and **SB2** connected at the same time as this will damage the nRF52840 SoC.

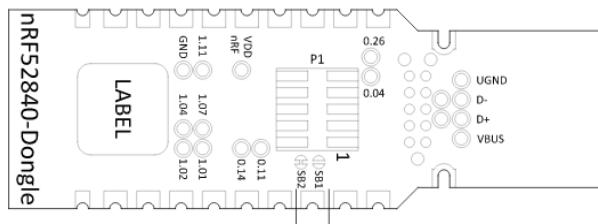
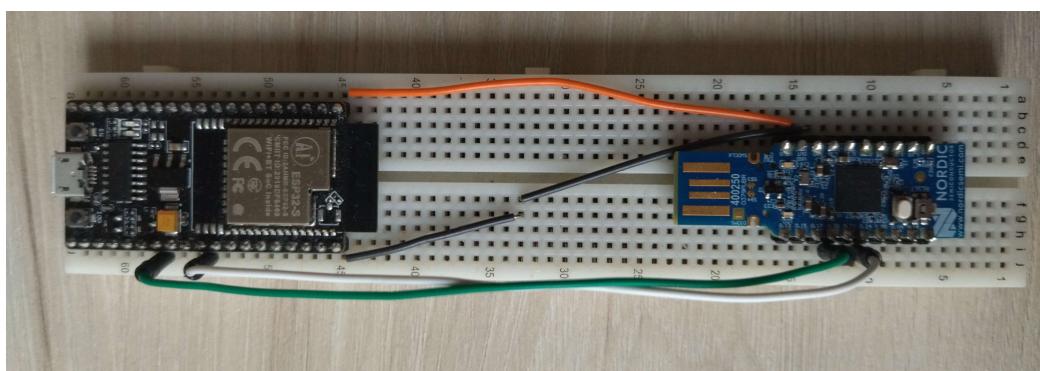


Figure 1. Configuring 1.8-3.6 V external source

Фиг. 6.72. Инструкции за захранване на nRF52840-Dongle чрез външно захранване

- Прототипна платка 3

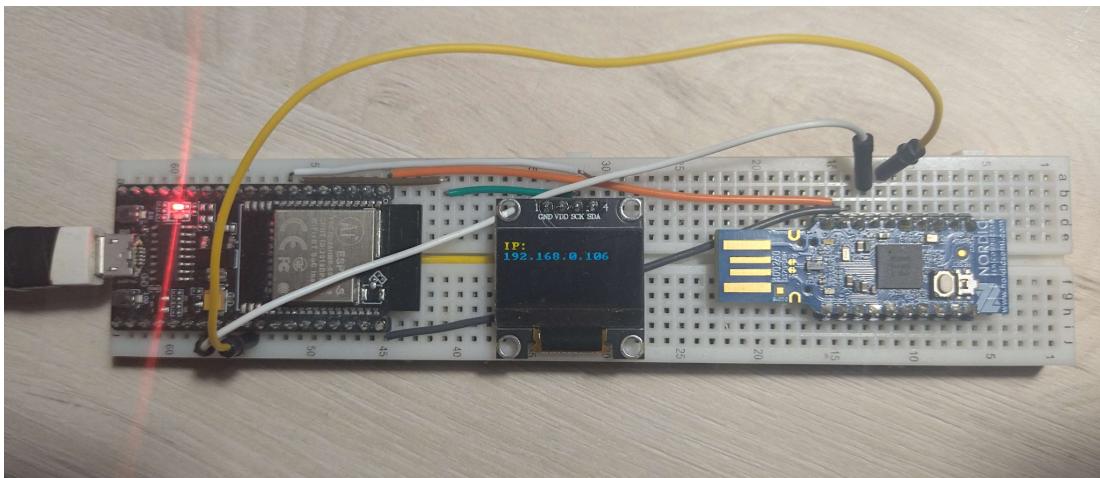
На фиг. 6.73. е показана третата прототипна платка. Тя реализира функционалността на граничния рутер и е първата прототипна платка на управляващия възел.



Фиг. 6.73. Прототипна платка 3

- Прототипна платка 4

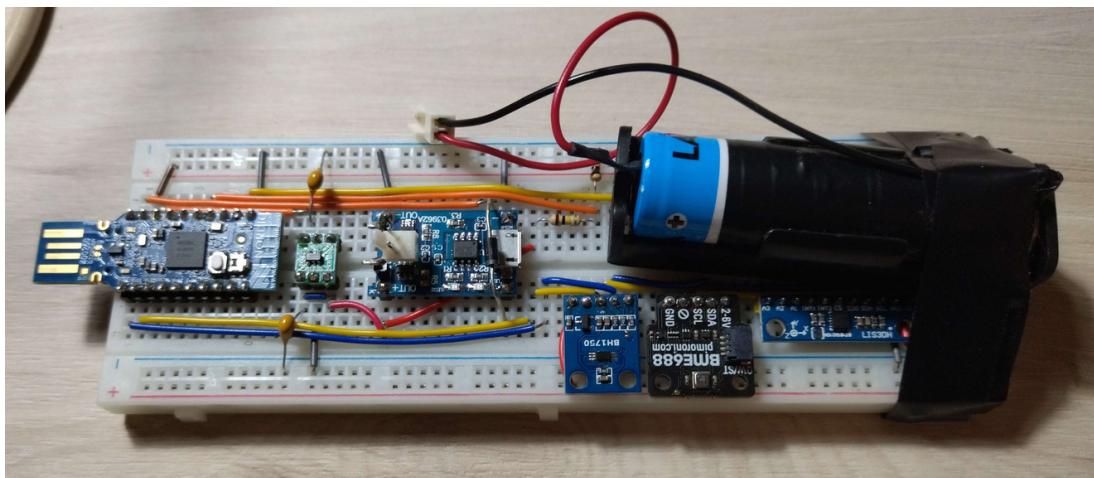
На фиг. 6.74. е показана четвъртата прототипна платка, която представлява управляващ възел с всички компоненти.



Фиг. 6.74. Прототипна платка 4

- Прототипна платка 5

На фиг. 6.75. е показана петата прототипна платка, която представлява основния възел с всички компоненти.



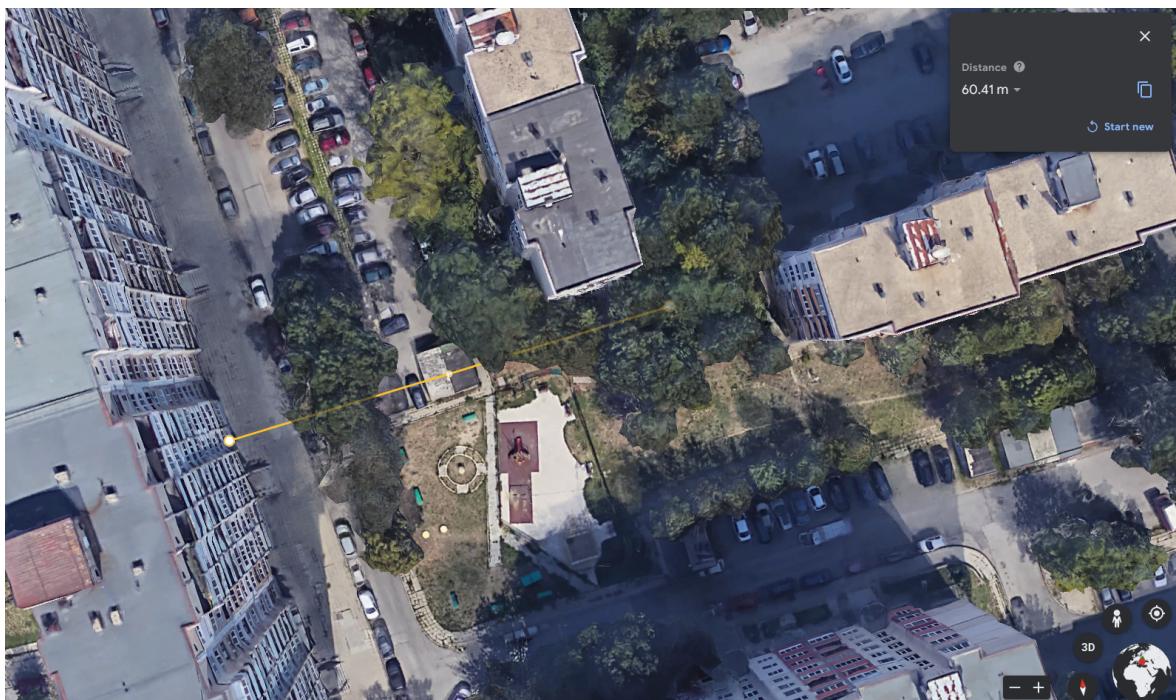
Фиг. 6.75. Прототипна платка 5

6.4.2. Измерване на максималното разстояние за приемане и предаване на информация

За да бъде проверено разстоянието, на което модулите могат да си изпращат информация, ще бъде направен реален тест. Той не е направен в перфектна среда и трябва да се има предвид, че мрежата е тествана само в едни и същи условия и резултатите могат да бъдат различни при промяна на околната среда.

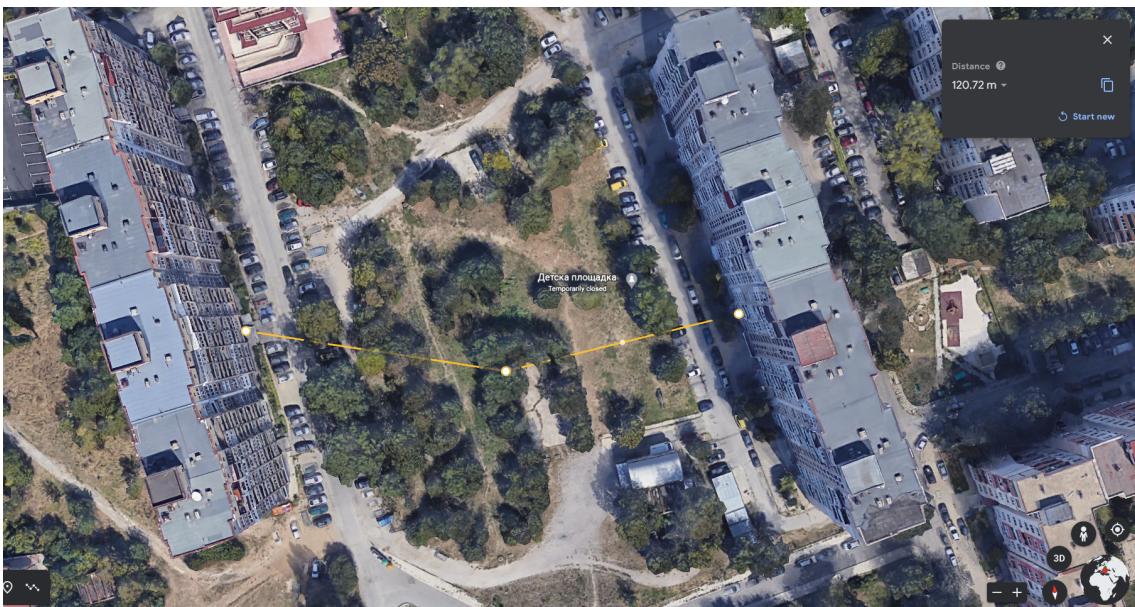
Тест на мрежата беше проведен на показаните на фиг. 6.76. и фиг. 6.77. места.

На фиг. 6.76. е показано измереното разстояние за успешно изпращане на данни между две устройства (два основни възела), което достига 60m. При тази постановка едното устройство е поставено на терасата на най-левия блок на височина 3-4m. Достигнатата най-далечна точка е на пътеката под дърветата, тоест устройството е било на височина 1m.



Фиг. 6.76. Местоположение 1

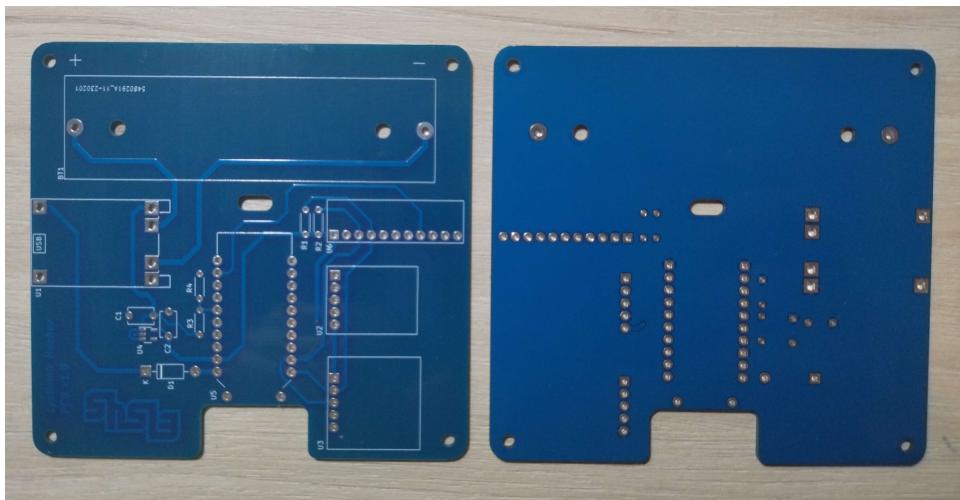
На фиг. 6.77. е показано максималното разстояние на предаване между 3 устройства, равняващо се на 120m. При тази мрежа първото устройство (управляващия възел) се намира вътре в десния блок на височина 5-6m. Следващото устройство, което е основен възел, е поставено на средната точка на височина 1.5m и на разстояние 60m от управляващия възел. Последното устройство се намира в най-ляявата точка. То отново е управляващ възел и достига 60m разстояние от другото, като устройството се намира на височина 1m.



Фиг. 6.77. Местоположение 2

6.4.3. Създаване на печатна платка на основния възел

На фиг. 6.78. е показана печатната платка на основния възел без компоненти от двете страни.



Фиг. 6.78. Печатна платка на основния възел без компоненти

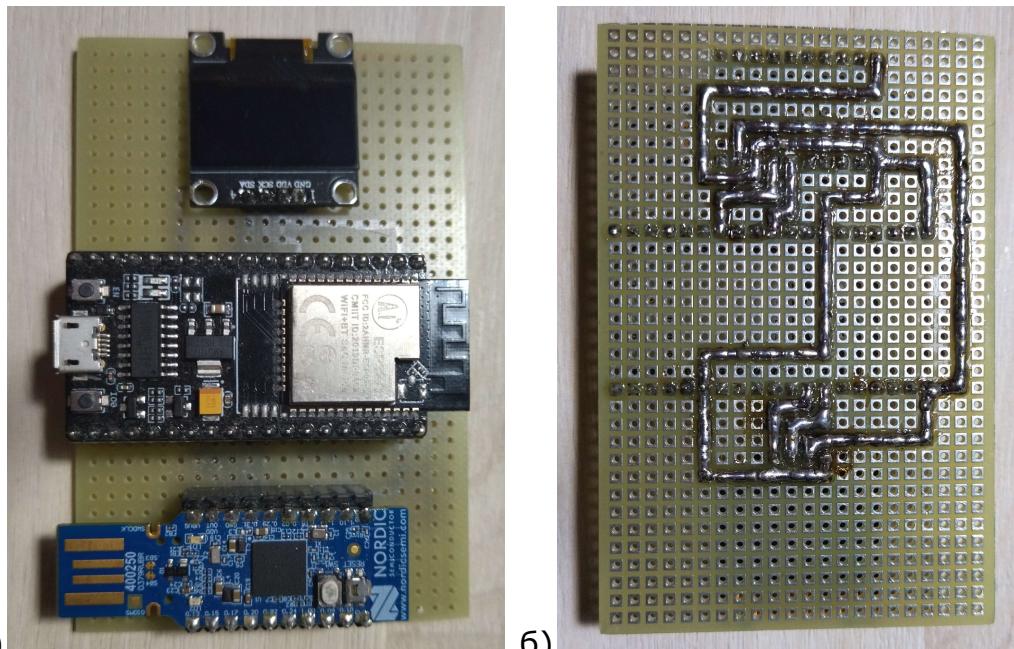
На фиг. 6.79. е показана печатната платка на основния възел след насищане с компонентите.



Фиг. 6.79. Печатна платка на основния възел с компоненти

6.4.4. Създаване на печатна платка на управляващия възел

На фиг. 6.80.а) и фиг. 6.80.б) е показана създадената печатна платка, направена за управляващия възел.



Фиг. 6.80.а) Изглед на печатната платка отпред, б) Изглед на печатната платка отзад

6.4.5. Показване на трафик, наблюдаван с помощта на Wireshark

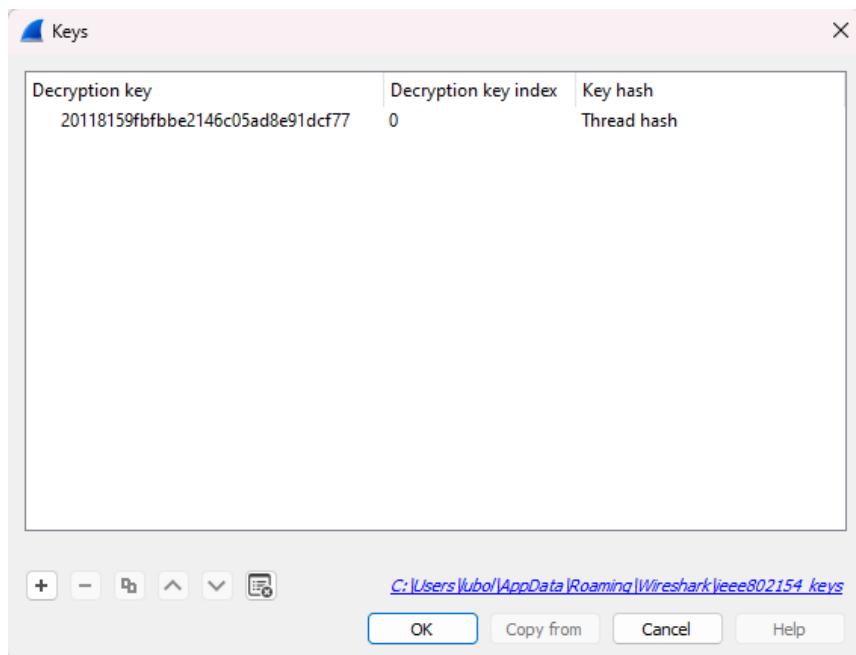
На страницата на Nordic съществува готова програма за конфигуриране на sniffer за 802.15.4. Тя позволява, с помощта на анализатора на мрежови протоколи - Wireshark, да бъдат улавяни и след това анализирани 802.15.4 пакети [117]. Програмата е поддържана от nRF52840-Dongle и следователно ще бъде качена на такъв. Инсталацията е подробно описана на сайта на Nordic - [118].

След нея, обаче, Wireshark ще трябва да бъде настроен за заснемане на Thread трафик. За целта първо ще трябва да бъдат конфигурирани ключове за декриптиране на IEEE 802.15.4 трафика,

за да може данните да бъдат показани в четим вид, което ще стане по следния начин:

След отваряне на Wireshark трябва да се изпълни следната поредица от стъпки: **Edit > Preferences > Protocols > IEEE 802.15.4 > Decryption keys (Edit)**.

На фиг. 6.81. е показан прозорецът, който потребителя ще види след изпълнение на стъпките.

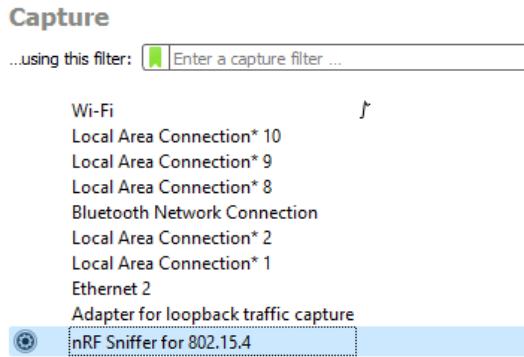


Фиг. 6.81. Прозорец за добавяне на ключове за декриптиране

В този прозорец трябва да се натисне бутона „+“ и да се добави стойност в "Decryption key" полето, съответстваща на ключа на OpenThread мрежата, която ще бъде анализирана. След това в "Decryption key index" полето ще трябва да се постави стойност 0, а в "Key hash" полето да е избрана опцията Thread hash.

След конфигурирането на ключ за декриптиране за по-нататъшна работа ще е необходима инсталацията на nRF Sniffer plugin в Wireshark. Процесът е подробно описан на страницата на Nordic - [119].

След завършване на гореописаната конфигурация в Wireshark ще излезе nRF Sniffer for 802.15.4 като интерфейс. Това е показано графично на фиг. 6.82.



Фиг. 6.82. Изглед към активните интерфейси

No.	Time	Source	Destination	Protocol	Length	Info
6911	604.773775	::a16f:943c:487:8db3	ff03::1	UDP	52	49562 → 12346 Len=3
>	Frame 6911: 52 bytes on wire (416 bits), 52 bytes captured (416 bits)					
>	IEEE 802.15.4 Data, Dst: Broadcast, Src: 0xb400					
>	6LoWPAN, Src: ::a16f:943c:487:8db3, Dest: ff03::1					
>	Internet Protocol Version 6, Src: ::a16f:943c:487:8db3, Dst: ff03::1					
>	User Datagram Protocol, Src Port: 49562, Dst Port: 12346					
> Data (3 bytes)						
	Data: 323432					
	[Length: 3]					

Фиг. 6.83. Пакет в Thread мрежата, заснет чрез Wireshark

На фиг. 6.83. е показан заснет пакет в Thread мрежата. Пакетът е с номер 6911. Направен е 604 секунди след стартирането на заснемането на пакети на nRF Sniffer интерфейса, адресът, от който идва е ::a16f:943c:487:8db3, и има дестинационен адрес - ff03::1. Транспортният протокол е UDP. Има дължина 52 и информация, показваща порт 49562, от който идва, и порт 12346, към който отива, както и дълчината на данните в него, равняващи се на 3 байта. На долните два прозореца е показана по-подробна информация за пакета, като в синьо е маркирана информацията, която се изпраща. Тя не е криптирана и е в четим вид - 242.

В Github хранилището, в папката logs, има packet tracer файлове, които съдържат заснет трафик на работеща OpenThread мрежа [79].

Заключение

В настоящата дипломна работа е разработена сензорна IoT мрежа, използваща модификация на протокола Thread - OpenThread. Реализирани са два вида устройства, изпълняващи ролята на основен и водещ възел. Добавена е функционалност за автоматично свързване на нов възел, безжично предаване на данни към водещия възел, свързване на вътрешната OpenThread мрежа към външна Wi-Fi мрежа и достъп до предадените данни от устройства, свързани към локалната Wi-Fi мрежа.

В първа глава е направено обширно проучване и сравнение на мрежови протоколи, съществуващи сензорни IoT мрежи и компоненти, използвани в тях.

Във втора глава са описани архитектурните и комуникационни изискванията към проектираната сензорна мрежа, както и функционалните и конструктивни такива. Направена е обща блокова схема, както и блокова схема на отделните блокове и подблокове.

В трета глава е аргументиран изборът на CAD система, избрани са подходящите компоненти за изпълнение на конструктивните изисквания и е добавена необходимата аргументация за техния избор. Създадени са схемни изображения и принципни електрически схеми.

В четвърта глава са показани корпусите на избраните в трета глава компоненти и са дефинирани основните изисквания за проектиране на печатните платки. Проектирани са необходимите печатни платки, като допълнително към тях е създаден и 3D модел.

В пета глава са описани използваните програми, инструменти и развойна среда. Добавено е подробно описание за избора на подходящ мрежови протокол, направени са функционални диаграми и блокови схеми на софтуера и е показана финалната версия на сурс кода.

В шеста глава са описани всички практически резултати, както и процесът за тяхното постигане.

На базата на всичко описано по-горе може да се направи извода, че заданието и основната цел на дипломната работа са изпълнени.

Като бъдещо развитие на проекта може да се добавят:

- Допълнителни сензори към основните възли с цел следене на допълнителни параметри;
- Външна антена за достигане на по-големи разстояния;
- Слънчев панел към отделните възли за повишаване на живота на батерията;
- Удобен потребителски интерфейс, достъпен от всяка към вид устройства;
- Low-Power функционалност;
- Маршрутизиране на трафика до глобалната мрежа;
- Създаване на подходяща механична конструкция за отделните възли.

Използвани съкращения

Английски

6LoWPAN - IPv6 over Low-Power Wireless Personal Area Networks

AC/DC - Alternating current/Direct current

ADC - Analog-to-digital converter

AI - Artificial Intelligence

BLE - Bluetooth Low-Energy

BR - Border Router

CAD - Computer Aided design

CERN - Center for European Nuclear Research

CLI - Command Line Interface

CoAP - Constrained Application Protocol

DTLS - Datagram Transport Layer Security

EID - Endpoint Identifier

ESP-IDF - Espressif's IoT Development Framework

FED - Full End Device

FIFO - First In First Out

FTD - Full Thread Device

GPIO - General Purpose Input/Output

GPRS - General packet radio service

GPS - Global Positioning System

HASL - Hot Air Solder Leveling

HTTP - Hypertext Transfer Protocol

I/O - Input / Output

I2C - Inter-Integrated Circuit

IAQ - Indoor Air Quality

ID - Identification

IEEE - The Institute of Electrical and Electronics Engineers

IID - Interface Identifier

IoMT – Internet of Medical Things

IoT - Internet of Things

IPv6 - Internet Protocol version 6

ISM - Industrial, scientific and medical

IT - Information technology

LDO - Low-dropout regulator

LLA - Link-Local Address

MAC - Media Access Control

MCU - Microcontroller unit

MED - Minimal End Device

ML-EID - Mesh-Local EID

MQTT - Message Queue Telemetry Transport

MQTT-SN - MQTT for Sensor networks

MTD - Minimal Thread Device

NB-IoT - Narrowband IoT

NDA - Non-disclosure agreement

NPTH - Non plating through hole

OLED - Organic light-emitting diode

PAN - Personal Area Network

PAN ID - Personal Area Network ID

PHY - Physical layer

PID - Product ID

PTH - Plating through hole

QoS - Quality of Service

RAM - Random Access Memory

RCP - Radio Co-Processor

REED - Router-Eligible End Devices

REST - Representational state transfer

RF - Radio Frequency

RLOC - Routing Locator

ROM - Read-only Memory

RTOS - Real-time operating system

RX - Receive

SCL - Serial Clock

SDA - Serial Data

SDK - Software Development kit

SED - Sleepy end device

SMD - Surface-mounted device

SoC - System on a Chip

SoM - System on Module

SPI - Serial peripheral interface

TCP - Transmission Control Protocol

TLS - Transport Layer Security

TVOC - Total volatile organic compounds

TX - Transmit

UART - Universal asynchronous receiver-transmitter

UDP - User Datagram Protocol

ULA - Unique Local Address

ULP - Ultra Low Power

URI - Uniform Resource Identifier

USB - Universal Serial Bus

VOCs - Volatile Organic Compounds

VS Code - Visual Studio Code

WAN - Wide Area Network

XPAN ID - Extended Personal Area Network ID

M2M - Machine-to-Machine

OS - Operating system

OEM - Original Equipment Manufacturer

Български

AC/DC - Променлив ток/Постоянен ток

ADC - Аналогово-цифров преобразувател

AI - Изкуствен интелект

BR - Границен маршрутизатор

I/O - Входно/изходни

IAQ - Индекс за качество на въздуха в затворени помещения

IoMT – Интернет на медицинските неща

IoT - Интернет на нещата

IT - Информационни технологии

LDO - Линеен стабилизатор

MCU - Микроконтролер

ОС - Операционна система

PAN - Персонална локална мрежа

RAM - Памет с произволен достъп

ROM - Памет само за четене

RTOS - Операционна система в реално време

SED - Заспало крайно устройство

VOCs - Летливи органични съединения

WAN - Широкообхватна мрежа

Използвана литература

- [1]<https://study.com/academy/lesson/sensor-networks-definition-operation-relationship.html>
- [2]<https://circuitcellar.com/tech-news/product-news/mesh-sensor-network-uses-nordics-multiprotocol-soc/>
- [3]<https://www.nordicsemi.com/products/nrf52840>
- [4]<https://circuitcellar.com/research-design-hub/device-measures-indoor-air-quality/>
- [5]<https://www.nordicsemi.com/products/nrf51822>
- [6]<https://www.sciosense.com/wp-content/uploads/documents/SC-00123-2-DS-3-CCS811B-Datasheet-Revision-2.pdf>
- [7][https://cdn-shop.adafruit.com/datasheets/1899 HTU21D.pdf](https://cdn-shop.adafruit.com/datasheets/1899HTU21D.pdf)
- [8]<https://arxiv.org/ftp/arxiv/papers/1902/1902.09400.pdf>
- [9]<https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series.html>
- [10]https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf
- [11]<https://www.multitech.com/brands/multiconnect-conduit>
- [12]<https://semtech.my.salesforce.com/sfc/p/#E0000000JeIG/a/44000000MDnR/Et1KWLCuNDI6MDagfSPAvqqp.Y869Flgs1LleWyfjDY>
- [13]https://semtech.my.salesforce.com/sfc/p/#E0000000JeIG/a/44000000MDmo/OfVC_rbxij4JjkT4hLzU1kq4gOXb4POLNRprWlxRIZs
- [14]http://www.ijesit.com/Volume%203/Issue%206/IJESIT201406_35.pdf
- [15]<https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/41203E.pdf>
- [16]<https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Manual.pdf>
- [17]<https://www.ti.com/lit/an/snossb76c/snossb76c.pdf?ts=1669965885202>

- [18]<https://www.iotcommunications.com/blog/types-of-iot-networks/>
- [19]https://electricenergyonline.com/EE/MagIMG/so18_byline5_img1.jpg
- [20]<https://www.militaryaerospace.com/directory/blog/14059638/iot-wireless-standards-in-a-nutshell>
- [21]<https://www.bluetooth.com/blog/wireless-connectivity-options-for-iot-applications-technology-comparison/>
- [22]<https://hashstudioz.com/blog/top-iot-communication-protocols-2020/>
- [23]<https://radiocrafts.com/technologies/6lowpan/>
- [24]<https://www.threadgroup.org/What-is-Thread/Thread-Benefits>
- [25]<https://www.intuz.com/blog/thread-vs-bluetooth-whats-a-better-communication-technology>
- [26]<https://flespi.com/blog/top-7-technologies-for-iot-connectivity-2017>
- [27]<https://dzone.com/articles/coap-protocol-step-by-step-guide>
- [28]<https://www.wallarm.com/what/coap-protocol-definition>
- [29]<https://mqtt.org/>
- [30]<https://www.u-blox.com/en/blogs/insights/mqtt-sn>
- [31]https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf
- [32]<https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>
- [33]<https://datatracker.ietf.org/doc/html/rfc6455>
- [34]https://docs.ai-thinker.com/_media/esp32/docs/nodemcu-32s_product_specification.pdf
- [35]https://infocenter.nordicsemi.com/pdf/nRF52840_Dongle_User_Guide_v1.0.pdf
- [36]<https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>
- [37]<https://www.microchip.com/en-us/product/MRF24J40MA>
- [38]<https://www.semtech.com/products/wireless-rf/lora-connect/sx1276>
- [39]<https://www.ti.com/lit/ds/symlink/lm35.pdf>

- [40]https://cdn.sparkfun.com/assets/6/3/c/7/7/HIH-5030-001_datasheet.pdf
- [41]<https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf>
- [42]<https://www.mouser.com/datasheet/2/218/SPW2430HR5H-B-1290924.pdf>
- [43]<https://www.st.com/en/mems-and-sensors/lis3dh.html>
- [44]<https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf>
- [45]<https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf>
- [46]<https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme688-ds000.pdf>
- [47]<https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf>
- [48]<https://nettigo.eu/products/oled-display-0-96-i2c-128x64-ssd1306-white>
- [49]<https://www.kicad.org/about/kicad/>
- [50]<https://home.cern/news/news/computing/kicad-software-gets-cern-treatment>
- [51]<https://olimex.wordpress.com/2022/12/09/agonlight-open-source-hardware-retro-computer-running-bbc-basic-was-captured-in-kicad-and-updated-by-olimex/>
- [52]<https://openthread.io/guides/border-router/espressif-esp32>
- [53]<https://erelement.com/wireless/nodemcu-32s>
- [54]https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/thread/tools.html#configuring-a-radio-co-processor
- [55]<https://www.aliexpress.com/item/32896971385.html>
- [56]<https://erelement.com/sensors/bme688>
- [57]<https://www.aliexpress.com/item/1005001621873442.html>
- [58]<https://www.aliexpress.com/item/4000182047375.html>

- [59]<https://elimex.bg/product/85664-akumulator-3.7v-3400mah-lc18650-lava>
- [60]<https://elimex.bg/product/74842-kit-k505-zaryadno-za-li-ion-baterii-mikro-usb>
- [61]<https://store.comet.bg/Catalogue/Product/16340/>
- [62]https://fscdn.rohm.com/en/products/databook/datasheet/ic/power/linear_regulator/buxxtd3wg-e.pdf
- [63]<https://elimex.bg/product/80846-c1uf50v-krmp-x7r-p2-54mm>
- [64]<https://store.comet.bg/CatalogueFarnell/Product/915040/>
- [65]<https://www.farnell.com/datasheets/2861411.pdf>
- [66]<https://github.com/ccadic/TP4056-18650>
- [67]<https://grabcad.com/library/03962a-hw107-lithium-battery-charging-module-1>
- [68]<https://grabcad.com/library/bme680-3>
- [69]<https://grabcad.com/library/gy-302-ambient-light-sensor-module-1>
- [70]<https://grabcad.com/library/0-96-inch-oled-lcd-module-4-pin-128x64-1>
- [71]<https://grabcad.com/library/battery-holder-18650-comf-bhc-18650-1-1>
- [72]<https://www.nordicsemi.com/Products/Development-hardware/nRF52840-Dongle/Download?lang=en>
- [73]<https://grabcad.com/library/esp32-wroom-32s-nodemcu-32s-1>
- [74]<https://jlcpcb.com/capabilities/pcb-capabilities>
- [75]<https://elimex.bg/product/15079-platka-eksperimentalna-136-ednosta-ranna-exp1>
- [76]<https://support.jlcpcb.com/article/194-how-to-generate-gerber-and-drill-files-in-kicad-6>
- [77]<https://jlcpcb.com/>
- [78]<https://devmountain.com/blog/git-vs-github-whats-the-difference/>
- [79]<https://github.com/LyubomirNachev/Diplomna>

- [80]<https://code.visualstudio.com/>
- [81]<https://www.nordicsemi.com/Products/Development-software/nrf-connect-sdk>
- [82]<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html>
- [83]<https://www.zephyrproject.org/wp-content/uploads/sites/38/2023/01/Zephyr-Overview-20230124.pdf>
- [84]<https://www.st.com/en/partner-products-and-services/freertos-kernel.html>
- [85]<https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-vs-code>
- [86]<https://marketplace.visualstudio.com/items?itemName=espressif.esp-idf-extension>
- [87]<https://openthread.io/>
- [88]<https://www.silabs.com/documents/public/user-guides/ug103-11-fundamentals-thread.pdf>
- [89]<https://openthread.io/guides/thread-primer/node-roles-and-types>
- [90]<https://openthread.io/guides/thread-primer/ipv6-addressing>
- [91]<https://openthread.io/guides/thread-primer/network-discovery>
- [92]https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/getting_started/assistant.html#gs-assistant
- [93]<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html>
- [94]<https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md>
- [95]<https://git-scm.com/downloads>
- [96]https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/samples/openthread/cli/README.html#configuration-files
- [97]<https://github.com/openthread/openthread/blob/main/src/cli/README.E.md>

- [98]<https://openthread.io/reference/group/radio-config>
- [99]<https://wolles-elektronikkiste.de/en/bh1750fvi-gy-30-302-ambient-light-sensor>
- [100]<https://github.com/claws/BH1750>
- [101]https://github.com/lamik/Light_Sensors_STM32
- [102]<https://blog.golioth.io/adding-an-out-of-tree-sensor-driver-to-zephyr/>
- [103]<https://docs.zephyrproject.org/3.2.0/build/dts/bindings.html>
- [104]<https://docs.zephyrproject.org/3.2.0/build/dts/dt-vs-kconfig.html>
- [105]<https://docs.zephyrproject.org/3.2.0/develop/application/index.html>
- [106]<https://docs.zephyrproject.org/3.2.0/build/dts/intro.html#devicetree-intro>
- [107]<https://docs.zephyrproject.org/3.2.0/build/dts/api-usage.html#dt-node-identifiers>
- [108]https://www.youtube.com/watch?v=A9c9moP_WNw
- [109]<https://openthread.io/platforms/co-processor>
- [110]https://github.com/espressif/esp-idf/blob/master/examples/protocols/sockets/udp_server/main/udp_server.c
- [111]<https://www.freertos.org/a00125.html>
- [112]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_http_server.html#websocket-server
- [113]<https://github.com/SIMS-IOT-Devices/FreeRTOS-ESP-IDF-WebSockets-Server/blob/main/Websocket-Server.c>
- [114]<https://esp32tutorials.com/oled-esp32-esp-idf-tutorial/>
- [115]<https://www.sirius-pcb.com/bg/product/i4659/%D0%9F%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D0%BD%D0%B8%D0%BA-SOT23-5-/SOT23-6-to-DIP6.html>
- [116]https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_nrf52840_dongle%2FUG%2Fnrf52840_Dongle%2Fhw_power_supply.html

[117]<https://www.nordicsemi.com/Products/Development-tools/nrf-sniffer-for-802154>

[118]https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_sniffer_802154%2FUG%2Fsniffer_802154%2Fintro_802154.html

[119]https://infocenter.nordicsemi.com/index.jsp?topic=%2Fug_sniffer_802154%2FUG%2Fsniffer_802154%2Fintro_802154.html

[120]https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf_gs_testing.html#putty