



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Факултет Компютърни системи и технологии

Катедра “Компютърни системи”

ДИПЛОМНА РАБОТА

Тема:

Централизирана система за многоканално поточно аудио в домашна WiFi мрежа от източник “потребителско мобилно устройство”

Дипломант:

Иво Карапетев

Научен ръководител:

маг. инж. Валери Иванов

Консултант:

доц. д-р инж. Валентин Молов

СОФИЯ

2024

Съдържание

Използвани термини.....	3
Увод.....	5
1. Обзор на съществуващи решения и технологии.....	7
1.1. Решения на пазара за многоканално разпространение на аудио.....	7
1.2. Протоколи за многоканално разпространение и управление на аудио потоци.....	12
1.3. Програмни средства за обработка и управление на аудио процесите в рамките на Linux-базирана хардуерна система.....	20
1.4. Обзор на аппаратните възли заложени в изходната постановка.....	26
2. Проектиране на системна архитектура.....	29
2.1. Системна диаграма.....	29
2.2. Дефиниране на функционалните изисквания към системата.....	31
2.3. Хардуерна блок схема на централния възел.....	32
2.4. Дефиниране на функционални изисквания към централния възел.....	33
2.5 Подбор на основни инфраструктурни компоненти за централния възел.....	34
2.6. Хардуерна блок схема на периферните възли.....	45
2.7. Дефиниране на функционални изисквания към периферните възли.....	47
2.8. Подбор на основни инфраструктурни компоненти за периферните възли.....	48
3. Проектиране на “централен възел” и “периферни възли”.....	54
3.1 Функционално описание на централния възел.....	54
3.2 Реализация на централния възел.....	56
3.3. Функционално описание на периферните възли.....	75
3.4. Реализация на периферните възли.....	77
Практически резултати и приложение.....	90
Заключение.....	104
Литературни източници.....	106

Използвани термини

A2DP - Advanced Audio Distribution Profile
ALSA - Advanced Linux Sound Architecture
ARM - Advanced RISC Machine
Codec - COder + DECoder
ESP-ADF - Espressif Audio Development Framework
ESP-IDF - Espressif IoT Development Framework
I2C - Inter-Integrated Circuit
I2S - Inter-IC Sound
LPDDR SDRAM - Low-power Double Data Rate SDRAM
SDRAM - Synchronous Dynamic RAM
NVS - Non-Volatile Storage
PSRAM - PseudoStatic RAM
RAM - Random-Access Memory
RISC - Reduced Instruction Set Computer
RPi - Raspberry Pi
RTCP - RTP Control Protocol
RTP - Real-time Transport Protocol
RTSP - Real-Time Streaming Protocol
SPI - Serial Peripheral Interface
TCP - Transmission Control Protocol
UART - Universal Asynchronous Receiver-Transmitter
UDP - User Datagram Protocol
АЦП - Аналогово-Цифров Преобразувател (ADC - Analog to Digital Converter)
ЦАП - Цифрово-Аналогов Преобразувател (DAC - Digital to Analog Converter)

УВОД

През последните години, благодарение на интензивното развитие на технологиите и възникването на концепцията за “Интернет на Нещата” (“Internet of Things”), се разкриват нови възможности, стремящи се към подобряването качествата и автоматизацията на дома, чрез внедряването на тези технологии в него. Възникват термините “домашна автоматизация” (“Home Automation”) и “умни домове” (“Smart Homes”), които имат своите различия помежду си, но се преплитат, за да изпълнят една обща цел - приспособяването на крайния потребител към модерният прогресиращ свят. Съвременните домове включват в себе си контрола и автоматизирането на лампите, отоплението (термостати), вентилацията, климатичната система, охранителната система също така и домашните електроуреди като хладилници, съдомиялни, сушилни, фурни и др. Всичко това благодарение на “умните” устройства и системи предлагани на пазара. Най-често наблюдението и управлението на тези устройства се извършва през Интернет посредством безжичните технологии Wi-Fi или Bluetooth. По-комплексните системи от взаимно свързани устройства, съгласно идеята за “Internet of Things”, се състоят от множество ключове и/или сензори, които са свързани към един централен възел, служещ за контрол и наблюдение на цялата система. Централният възел притежава потребителски интерфейс, който може да представлява стенен панел, мобилно приложение, система с гласово управление или други

Но възможностите на “умните домове” не се изчерпват до тук – предоставят се устройства и методи, които улесняват озвучаването на множеството стаи в дома или имплементирането на аудио система за “домашно кино” (“home theatre”), като премахват зависимостта от окабеляване и сложни контролиращи системи. Терминът “home theatre” най-често бива употребяван във връзка с оборудването у дома, което се стреми да пресъздаде характерния звук и картина (audio and video experience), което ни предоставя киното. В зависимост от типа, броя и позиционирането на озвучителната техника и шумоизолираността на стаята са дефинирани различни стандарти, показващи степента, до която оборудването се доближава до това на киното. С напредването на времето всяка

технология и решение се стремят да бъдат до колкото е възможно по-лесни и удобни за използване, с цел подобряването на потребителското изживяване.

Целта на тази дипломна работа е да се реализира работоспособен модел на централизирана домашна озвучителна система, чрез подбор на съществуващи средства и начини, която да използва безжичната локална мрежа като среда за разпространение на аудио потоци. Потребителят чрез своето (мобилно) устройство да има възможността безжично да се свързва към централна точка от системата и по този начин да разпространява желаната от него музика в различни стаи / помещения от своя дом, където да бъдат възпроизвеждани акустично. Този модел ще има възможност да бъде в основата на разработването на краен потребителски продукт. Моделът ще послужи и като проверка за практическото реализиране на една домашно озвучителна система (Proof of Concept).

Първа глава

Обзор на съществуващи решения и технологии

В тази глава се разглеждат мрежови решения и протоколи, програмни средства за обработка и управление на звука в Linux-базирана система, както и подходящ хардуер, които са целесъобразни с реализирането на една безжична домашна озвучителна система. Но първо ще бъде направен оглед на съществуващи продукти на пазара, като основната цел е да се обърне внимание на общите характеристики, които да бъдат приложени в текущата дипломна работа.

1.1. Решения на пазара за многоканално разпространение на аудио

1.1.1. Audio Pro multi-room

Audio Pro [1] многоканална система за разпространение на аудио в множество стаи:

- Позволява интернет свързаност чрез Wi-Fi и Ethernet
- Възможност за безжична връзка чрез Bluetooth
- Предлагат единични тонколони (*Фиг. 1.1.*) както и цели озвучителни системи, състоящи се от няколко различни тонколони и сабуфери, които могат да бъдат подбрани според изискванията и целите на потребителя
- Притежават вътрешна батерия
- Поддържа интернет музикални услуги от AirPlay, Spotify Connect, Tidal, Deezer, TuneIn, Qobuz и др.
- Комбинира в себе си добро качество на звука и ниска цена

- Не са имплементирани методи за гласов контрол
- Собствено мобилно приложение за контрол и музикални услуги



*Фиг. 1.1.
Audio Pro Addon C5 speaker[1]*

1.1.2. SONOS multi-room system

SONOS [1] многоканална система за разпространение на аудио в множество стаи:

- Огромен набор от звукопроизвеждащи и звукообработващи устройства поради дългогодишното позициониране на пазара
- Изработени са различни видове водоустойчиви колонки
- Предоставят продукти за висококачествено домашно кино (“home theatre system”) (Фиг. 1.2.)
- Използва Wi-Fi и Ethernet като методи за разпространение на звук, но не използва Bluetooth като безжична технология
- Интегрира в себе си напълно развитото гласово командване от Alexa voice, подобно на тон колонката Amazon Echo

- Гласов контрол на съвместими домашни уреди, лампи, ключове и тн.
- Бързо увеличаващ се брой на поддържани услуги, работи се по интеграция и с Google Assistant
- Поддържа интернет музикални услуги от Amazon Music, Google Play Music, Spotify и много др.
- Контрол на ниските и високите честоти.
- Поддръжка на Class-D цифрови усилвателя.



Фиг. 1.2. SONOS 5.1 Surround Set[1]

1.1.3. Bluesound multi-room system

Bluesound [1] многоканална система за разпространение на аудио в множество стаи:

- Високоскоростно предаване на музика чрез безжичните технологии Wi-Fi и Bluetooth
- използва Ethernet, достигащ Gigabit скорости (Gigabit Ethernet)
- Изключително висока прецизност и качество при възпроизвеждането на високи и ниски тонове - Hi-Fi (“High-Fidelity”) аудио, използващо 24bit/192kHz

- Възможност за управление чрез мобилно/настолно приложение или чрез интегриран Amazon Alexa гласов контрол
- Съвместима с AirPlay технологията, разработена от Apple, за синхронизация на възпроизвежданото аудио между отделните говорители



Фиг. 1.3. Bluesound Wireless multi-room speakers and components[1]

1.1.4. Denon HEOS multi-room system

Denon HEOS [1] многоканална система за разпространение на аудио в множество стаи:

- Предоставят не само единични говорители и буфери, но също така и усилватели, комплекти за озвучаване на домашно кино (Фиг. 1.4.) дори и електронни грамофони
- Използват Ethernet, Wi-Fi и Bluetooth като технологии за разпространение на звук
- Поддържа всички популярни аудио формати за висококачествен звук - FLAC, WAV, ALAC, MP3, AAC, WMA и др.
- Имплементиран гласов контрол на базата на Amazon Alexa
- Поддържа интернет музикални услуги от Amazon Music, Google Play Music, Spotify и много др.



Фиг. 1.4. Denon HEOS Home Theatre - DHT-S516H[1]

1.2. Протоколи за многоканално разпространение и управление на аудио потоци

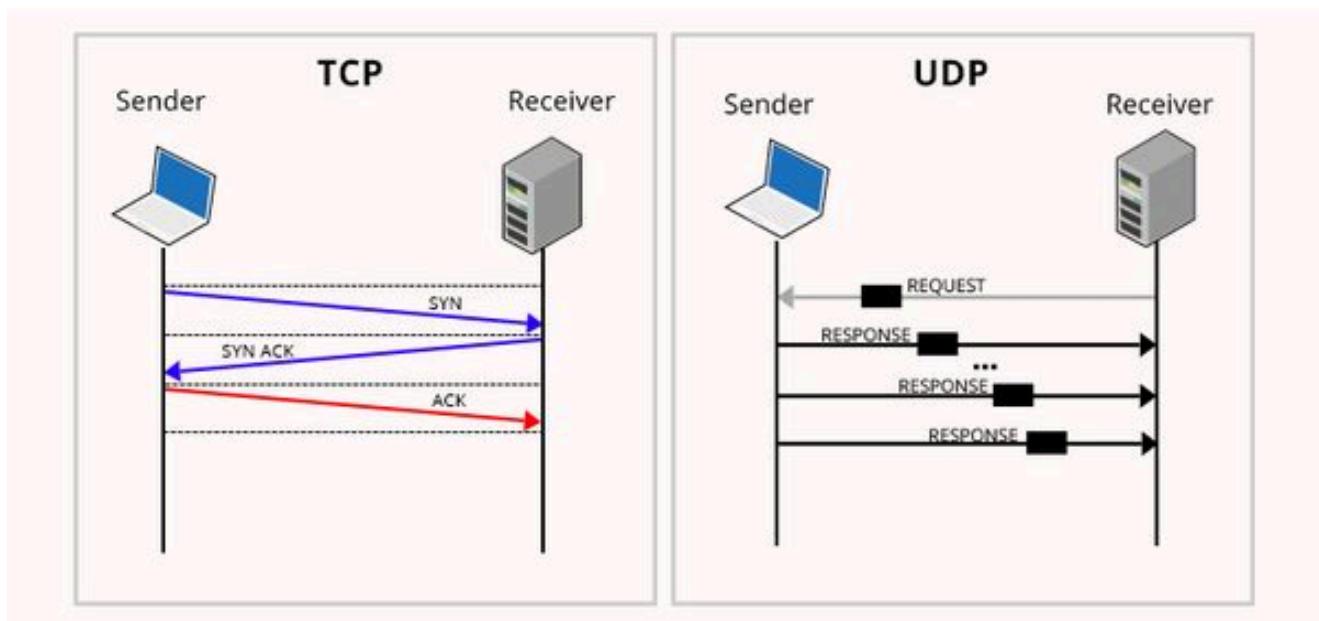
1.2.1. Сравнение на транспортните протоколи UDP и TCP в контекста на разпространение на аудио

UDP (User Datagram Protocol), заедно с TCP (Transmission Control Protocol) са мрежови протоколи, намиращи се на транспортния слой от OSI модела, който описва принципния начин на комуникация и строежа на телекомуникационните и компютърните мрежи. И двата протокола са отговорни за преноса на данни между две или няколко крайни точки в интернет мрежата, но всеки от тях има своите специфики и следователно служат за различни цели.

TCP протоколът изисква изграждането на връзка ("connection-oriented"), която ще служи за преноса на данните по мрежата до отдалеченото устройство, като използва конвенцията за "3-way-handshake", показано на *Фиг. 1.6. - лява картичка*, за осъществяването на тази връзка. TCP протокола се класифицира като "надежден" ("reliable"), но бавен поради допълнителните изчисления и дейности, които гарантират изпращането и получаването на **всички** IP пакети в правилния ред ("segment sequencing").

Затова TCP се използва за електронни съобщения на всички уеб базирани приложения и страници.

UDP от своя страна не изисква изграждането на връзка (“connectionless”) и не изпълнява допълнителни изчисления, които да осигурят последователността и достигането на цялата информация до получаващата страна, демонстрирано на Фиг. 1.5. - дясна картичка. Затова UDP е бърз, но “ненадежден” (“unreliable”) и е подходящ за пренасянето на аудио и видео информация, тъй като преносът на данни е в реално време (“real-time”), т.е. с минимално времезакъснение, и се допуска известна загуба на пакети.



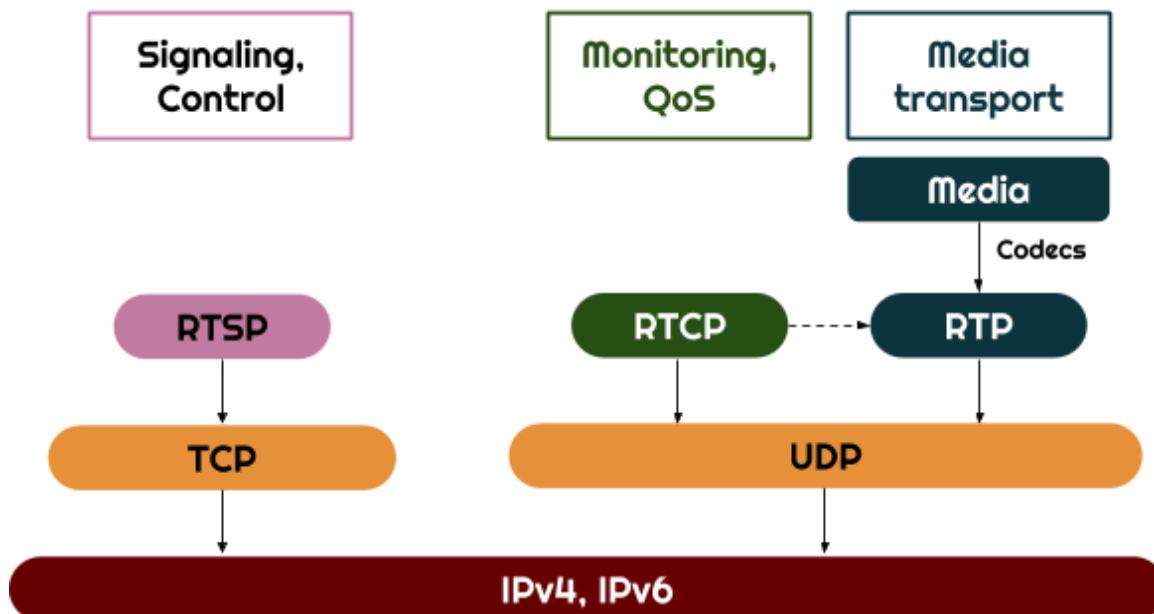
Фиг. 1.5. Разлика при предаване с TCP и UDP[2]

1.2.2. Обзор на протоколите RTP и RTSP

RTP (Real-time Transport Protocol) протокола се използва за предаване на потоци от информация в реално време (“real-time streaming”) върху изградена IP мрежа, базирайки се на UDP транспортния механизъм (Фиг. 1.7.). RTP намира приложение в множеството интернет музикални услуги (“music streaming”), както и във VoIP телефонията или видеоконференции.

В действителност RTP се състои от два протокола: самият RTP протокол, чиято отговорност е единствено преноса на конкретните потоци от информация, кодирани от определен аудио или видео кодек; и RTCP (RTP Control Protocol), който служи за събиране на статистически данни, с чиято помощ RTCP управлява RTP потоците, тяхното предаване през интернет мрежата и прилагането на “Quality of Service” (QoS) услуги. Също така RTCP спомага за пристигането на информацията в правилната последователност с минимални загуби.

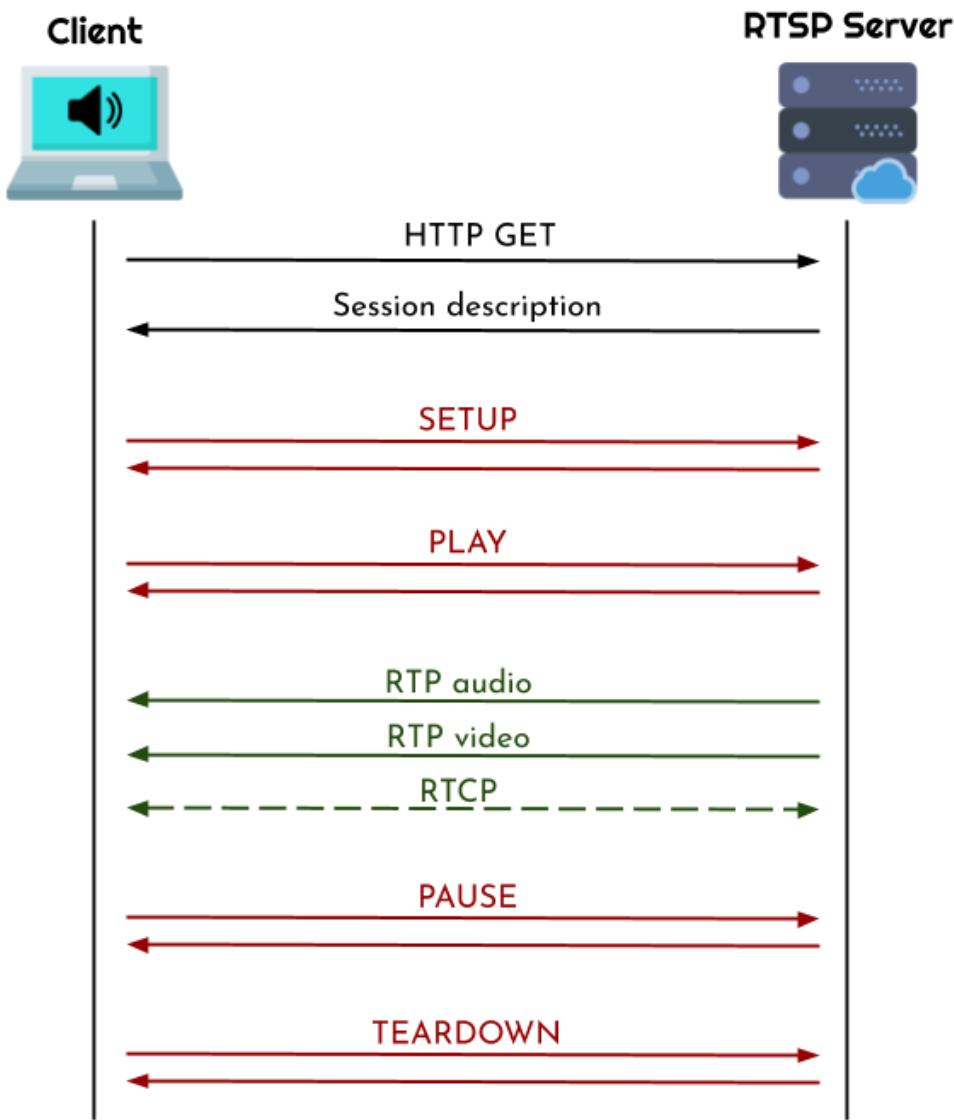
RTSP (Real-Time Streaming Protocol) е протокол за управление на аудио и видео потоци, който за разлика от RTP, се базира на транспортния протокол TCP и изгражда медийна сесия между RTSP сървър и клиентско устройство (Фиг. 1.7.). Подобно на HTTP протокола, RTSP използва методологията “заявка/отговор” (“request/response”) за поддържане на сесията и комуникацията между крайните точки. Тази сесия е своеобразният път, по който RTP потоците биват насочени към клиентското устройство. RTSP имплементира основни функционални команди като *възпроизвеждане* (*play*), *спиране на пауза* (*pause*), *запис* (*record*) и т.н. на медийната информация, които да бъдат изпълнени в реално време от клиентското устройство.



Фиг. 1.6. Структура на протоколите за пренос на данни в реално време

Основните RTSP заявки са (*изобразени на Фиг. 1.7.*):

- “SETUP” - обуславя начина и методите за транспорта на даден поток от информация. Тази заявка трябва да бъде изпратена преди заявката за възпроизвеждане “PLAY”. Заявката съдържа универсален указател на ресурс (“Uniform Resource Locator” или “URL”) на медийния поток, започващ по следния начин: “*rtsp://...*”, и поле, което включва локален порт за приемане на RTP данни (аудио или видео) и друг за RTCP данни (мета информация). Отговорът на сървъра обикновено потвърждава избраните параметри и попълва липсващите части, като например избраните от сървъра портове.
- “PLAY” - заявка за възпроизвеждане. Възможно е и последователното изпращане на няколко “PLAY” заявки. Заявката съдържа един или обобщен (сбор от няколко) URL адрес на медийните потоци и информация за тяхната дължина.
- “RECORD” - заявка за запис, като заявката съдържа в себе си времевата рамка, в която ще бъде изпълнен записа. Хронологичното време на времевата рамка се дефинира според местоположението на устройствата в конкретната часова зона (UTC). Сървърът самостоятелно решава дали да съхрани записаните данни под URL адреса на “RECORD” заявката или друг URL.
- “PAUSE” - временно спира един или всички медийни потоци, така че по-късно могат да бъдат възстановени с “PLAY” заявка. “PAUSE” заявката съдържа обобщен (сбор от няколко) или един URL на медийните потоци и параметър за диапазон, обозначаващ кога да бъде направена паузата. Когато параметърът за диапазон е пропуснат, паузата настъпва незабавно за неопределено време.
- “TEARDOWN” - използва се за прекратяване на сесията. Заявката спира всички медийни потоци и освобождава всички данни, свързани със сесията на сървъра.



Фиг. 1.7. Изграждане на RTSP сесия

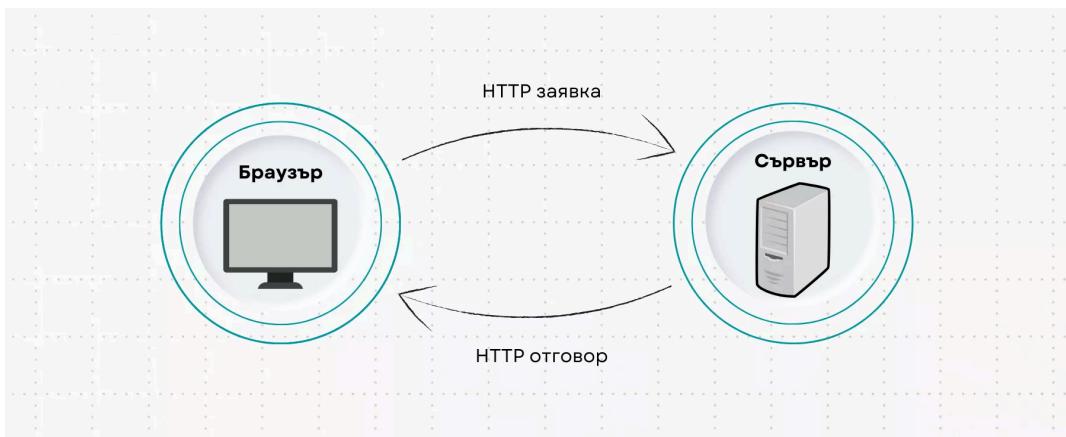
1.2.3. Обзор на HTTP и свойствата му за разпространение на съдържание в реално време (HTTP live-streaming)

HTTP (Hypertext Transfer Protocol) [18] като протокол седи в основата на съвременната глобална интернет мрежа. Протоколът се използван от уеб браузъри и сървъри за заявяване и доставяне на съдържание, като уеб страници, изображения, видеа и

други файлове. HTTP работи в модел клиент-сървър, където клиентът (обикновено уеб браузър) изпраща заявка до сървъра, а сървърът отговаря с искания ресурс.

HTTP е “без състояние” (stateless), т.е. всяка заявка клиента към сървъра е независима и не запазва информация от предишни такива. HTTP поддържа т.нар “методи” за заявка на информация, основните от които са:

- “GET”: Използва се за извлечане на представление на ресурс от сървъра, без да се правят промени в съществуващите данни на сървъра.
- “POST”: Чрез него се изпращат данни към сървъра, за да създаде нов ресурс или да актуализира съществуващ.
- “PUT”: Използва се за пълно обновяване или създаване на ресурс на сървъра, като предоставените данни заместват целия ресурс на посоченото място.
- “PATCH”: Частично обновява съществуващ ресурс, като прави само определени промени в него, вместо да го замени изцяло.
- “DELETE”: Премахва ресурс от сървъра, като обикновено няма възстановяване след успешното изтриване.



Фиг. 1.8. Начин на процедиране на HTTP протокола[18]

Що се отнася за доставянето на ресурси в реално време (live-streaming) като като аудио или видео, HTTP поддържа тази функционалност като медийните файлове (т.е. тези, които съдържат аудио и/или видео) се разделят на малки сегменти които се изпращат до

клиента (например, уеб браузър или медиен плейър) в реално време. От своя страна той може да възпроизвежда сегментите в реално време, докато те все още пристигат при него, което позволява на потребителите да започнат да се наслаждават на съдържанието без да има необходимост целият файл да бъде изтеглен предварително.

HTTP доставяне на ресурси в реално време се използва широко, защото работи ефикасно с наличната уеб инфраструктура (като мрежи за доставка на съдържание (Content Delivery Networks - CDN) и HTTP сървъри), поддържа адаптивно предаване на потоци с различна скорост, което регулира качеството на потока от данни в зависимост от мрежовите условия на потребителя. Това осигурява добро потребителско преживяване и се използва в услуги като YouTube, Netflix и радио предавания на живо.

1.2.4. Обзор на Bluetooth и A2DP профил за връзка с мобилно устройство

Bluetooth е стандартизирана технология за безжична комуникация между устройства, намиращи се на къси разстояния едно от друго. Едно от съвременните приложения на Bluetooth технологията е пренос на качествен стерео звук между две или няколко точки, например от мобилен телефон, настолен компютър към стерео Bluetooth слушалки, колонки и т.н. Но преди да започне какъвто и да е обмен на информация, Bluetooth устройствата трябва да определят помежду си вида на предаваната информация и начина по който тя ще бъде представена и изпратена. За тази цел се използват Bluetooth профили (Bluetooth profiles), които се основават на основния Bluetooth стандарт за комуникация.

Така A2DP (Advanced Audio Distribution Profile)[19] профила, който освен пренос на стерео музикални данни в реално време, е способен да пренася и глас, например разговор по телефона през Bluetooth слушалки или “handsfree”. Така в днешно време по-голямата част от Bluetooth устройствата, отредени за безжично предаване на аудио информация, напр. Bluetooth слушалки, “handsfree”, Bluetooth колонки, дори и външни хардуерни приспособления, като този показан на *Фиг. 1.9*, поддържат и разбират от A2DP Bluetooth профила.



Фиг. 1.9. Bluetooth A2DP стерео аудио предавател

1.3. Програмни средства за обработка и управление на аудио процесите в рамките на Linux-базирана хардуерна система

1.3.1. ALSA софтуерна архитектурна платформа

ALSA (Advanced Linux Sound Architecture)[12] е основен компонент от Linux ядрото (“Linux kernel”), който предоставя хардуерни драйвери (hardware drivers) за звуковите карти в едно устройство. ALSA бива възприеман като най-ниският слой от Linux ядрото, който взаимодейства пряко с хардуерната част на устройството и се грижи за звуковата функционалност на цялата система. Но също така ALSA е компонентът отговорен за миксирането и разделянето на различни аудио потоци, получаването на “stereo” или “mono” звуково възприятие, съвместимост с по-горни слоеве на звуковата архитектура на устройството и много други които я класифицират като бърза и стабилна звукова платформа, конкурентноспособна с нейните алтернативи - “CoreAudio” (за продукти на Apple) и “ASIO” (за Windows машини).

По-голямата част от днешните Linux базирани системи поддържат и работят върху звуковата платформа на ALSA, която избягва същественият проблем на по-старите платформи като OSS (“Open Sound System”) - само едно софтуерно приложение може да достъпи аудио хардуерната част за единица време.

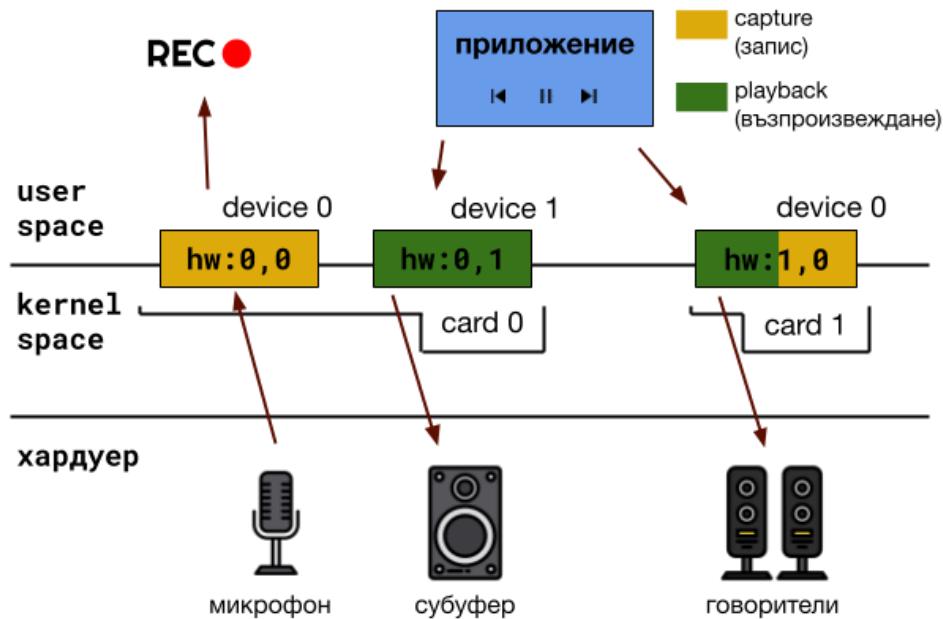
Освен тези дейности, изолирани от потребителя, ALSA групира набор библиотеки, вместени в потребителското пространство (“user space”) на всяко Linux устройство, за да могат разработчиците на приложения да използват функционалността на ALSA, където нивото на взаимодействие с драйверите е на по-високо ниво, т.е. конфигурацията е по-лесна и удобна, отколкото прякото взаимодействие с драйверите на ядрото, *визуално обяснени на Fig. 1.10. и Fig. 1.11.*

За да може да управлява и пренасочва всички аудио процеси и потоци, дали било влизачи в Linux устройството, или излизачи от него, ALSA изгражда йерархична структура, състояща се от *3 основни компонента: звукови карти (cards), устройства (devices) и под-устройства (subdevices)*, като всеки от тях изпълнява своята важна роля при процеса на възпроизвеждане (*playback*) или процеса на запис (*capture*) на един или

няколко аудио потока. На *Фиг. 1.10.* е показано визуално обяснението за йерархичната структура на ALSA.

Card компонентът обозначава звуковата карта, било тя физическа или виртуална, като разлика между отделните звукови карти се прави на базата на идентификационен номер, започващ от 0, и със всяка следваща карта се увеличава с единица. Напр.: *card 0*, *card 1*, *card 2*...

Device компонентът е точката, където се извършва достъпа до хардуерните приспособления за възпроизвеждане и запис. Те биват *capture device*, *playback device*, комбинация от двете, миксиращ *device (mix)* и много други. Подобно на *card*, *device* компонентите се обозначават с номер, започващ от нула: *device 0*, *device 1* и т.н. За ALSA е достатъчно да бъдат посочени *card* и *device* компоненти, за да може да се ориентира през кой вход или изход на устройството да бъде насочен аудио сигнала.



Фиг. 1.10. Опростен модел - работа на ALSA йерархична структура

Subdevice компонентите се използват от *device* компонентите при нужда от предаване при предаване на “многоканален” (“multi-channel”) аудио сигнал, като един *subdevice* обозначава един аудио канал. И тук се използва същият метод при номерирането.

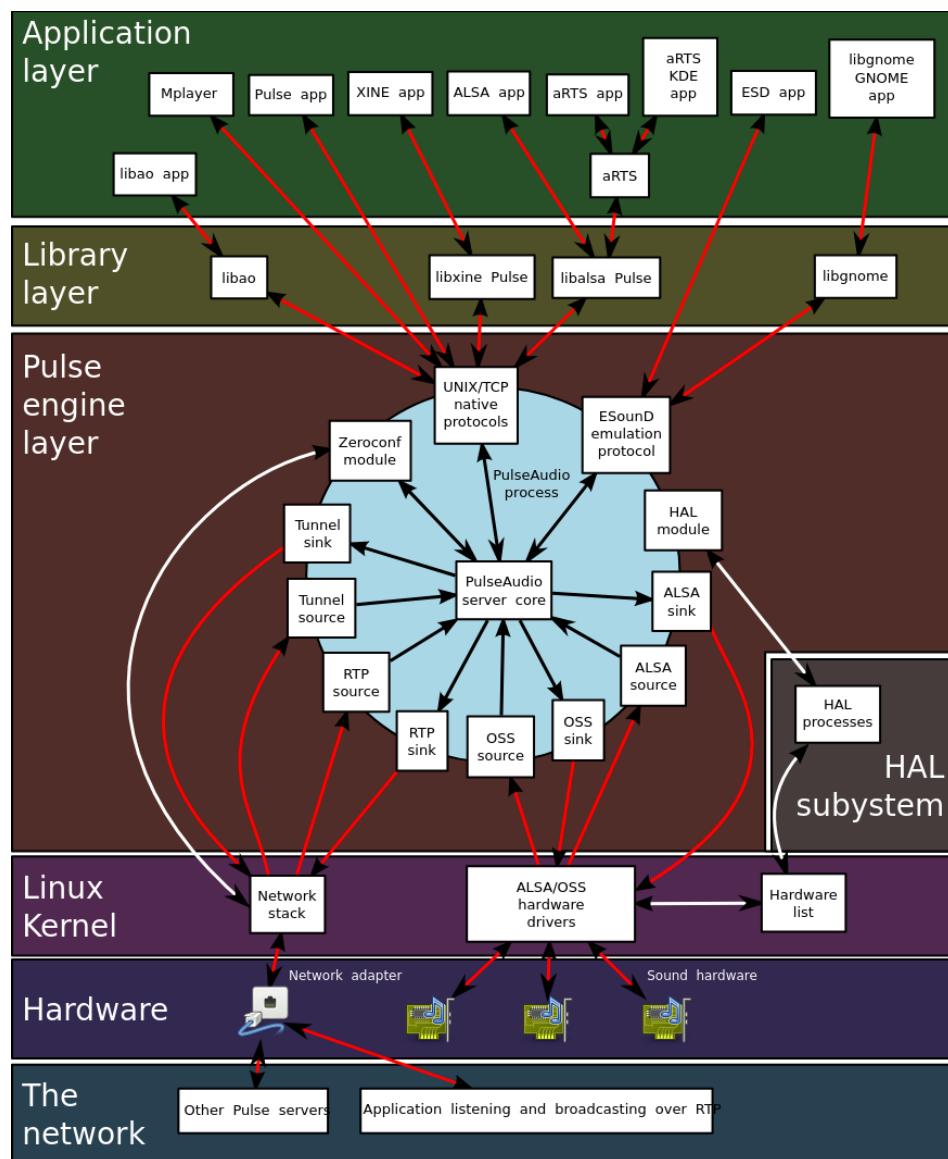
В своите конфигурационни файлове ALSA запазва обозначава тези компоненти под формата на символен низ (“string”) използвайки следната конвенция:

hw:<card number>,<device number>,<subdevice number>

Ако някой от параметрите е изпуснат, се подразбира, че неговата стойност е 0.

Например **hw:0,1,0** (или по-подробно **hw:CARD=0,DEV=1,SUBDEV=0**) е еднозначно с **hw:0,1** (или по-подробно **hw:CARD=0,DEV=1**)

1.3.2. PulseAudio звукова сървърна програма



Фиг. 1.11. Поглед върху различните процеси и начина на работата на PulseAudio[20]

PulseAudio[20] е звукова сървърна програма, което означава че се вмества между софтуерните програми и хардуерните драйвери, и точно това е нейното предназначение. PulseAudio също така предлага лесно мрежово поточно предаване (“network streaming”) през локални устройства. По-важната отличителна четна на PulseAudio е, че способен да се справи с големия набор от аудио приложения, без значение от техните специфики и възможности, *както е показано на Фиг. 1.11.*



PulseAudio

Фиг 1.12. Лого на PulseAudio

За разлика от ALSA, PulseAudio може да работи на операционни системи различни от Linux, но тяхната взаимовръзка се утвърждава с времето и се е превръща в своеобразен стандарт за новите Linux дистрибуции. PulseAudio създава виртуален слой, който най-често се намира над звуковата платформа ALSA и потоците от данни пряко или непряко преминават през сървърната програма.

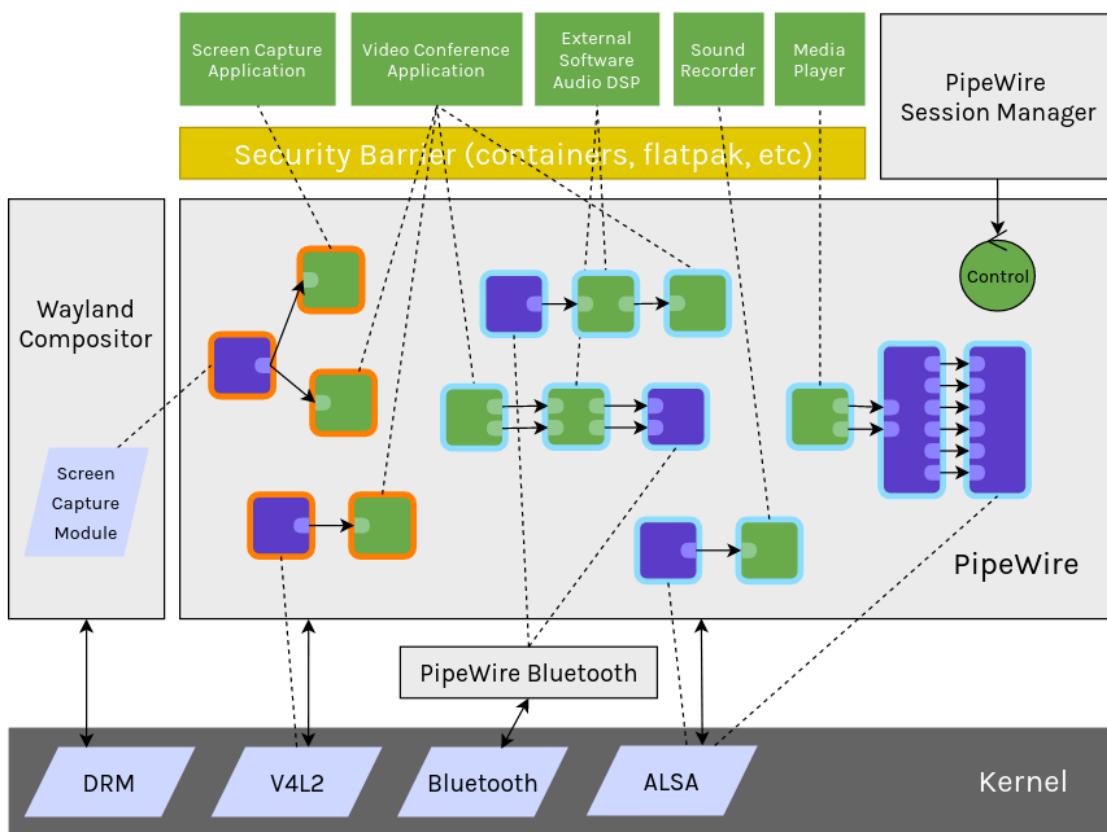
PulseAudio също така дефинира свои термини като “sink” - получаващата страна, която може да бъде някое устройство в мрежата или аудио изходите на звуковата карта, и “source” - изпращащата страна, която в повечето случаи представлява приложение за музикални услуги (“audio streaming app”), входни аудио портове или отдалечно устройство в мрежата.

1.3.3. PipeWire мултимедийна система

PipeWire[21] е мултимедийна система от следващо поколение предназначен за Linux, която обработва както аудио, така и видео потоци. Той е проектиран да замени или да работи заедно със съществуващи технологии като PulseAudio (за потребителско аудио) и JACK (за професионално аудио), предоставяйки решение за управление на различни видове медийни потоци. PipeWire подобрява обработката с ниска латентност, което го

прави подходящ за всичко - от просто аудио възпроизвеждане до сложни професионални аудио настройки като звукозаписни студия.

Едно от основните предимства на PipeWire е неговата гъвкавост. Той позволява на множество приложения да имат достъп до едни и същи аудио и видео устройства едновременно (Фиг 1.13), което е съществувало като ограничение при по-старите системи. Той също така поддържа разширени функции като обработка в реално време, което го прави идеален за професионална аудио работа, като същевременно е достатъчно прост за ежедневна употреба.



Фиг 1.13 PipeWire мултимедийна система[21]

PipeWire се използва широко в съвременните Linux дистрибуции, защото предлага по-добра производителност, по-ниска латентност и по-лесна конфигурация. Също така подобрява сигурността на управлението на медийни потоци, тъй като позволява по-фин контрол върху достъпа до аудио и видео устройства, което е важно за поверителността. Накратко, PipeWire е мощно решение за управление на мултимедия на Linux системи,

предлагашо по-добра интеграция, производителност и функционалност от своите предшественици.

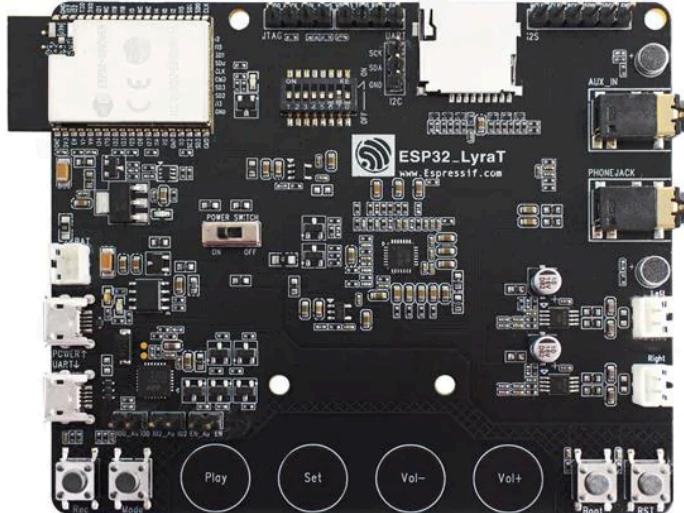
1.4. Обзор на апаратните възли заложени в изходната постановка

За реализацията на централния възел е необходима висока изчислителна мощност, тъй като обработката на звук, буфериране и извършване на разнообразни конфигурации не е в силата на Arduino или какъвто и да е микроконтролер с RISC компютърна архитектура. Затова тук преднина заема едноплатковия микрокомпютър Raspberry Pi, който от своя страна притежава ARM компютърна архитектура, даващ му по-широк набор от възможности като напр. инсталирането на операционна система (особено ако е Linux базирана), което пък от своя страна предоставя удобна и лесна среда за работа със самият микрокомпютър. Вниманието е насочено конкретно към модела на Raspberry Pi - 4B, който предоставя 3 варианта: 1, 2 или 4GB оперативна DDR4 RAM памет. И в трите случая паметта е достатъчна. Предоставя също и мощен процесор (Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz [11]) и поддържа последните стандарти за безжична комуникация - 2.4 GHz и 5.0 GHz IEEE 802.11ac (Wi-Fi 5) и Bluetooth 5.0.



Фиг. 1.14. Едноплатков микрокомпютър Raspberry Pi 4 модел B 4GB RAM

За реализацията на периферните възли, т.е. страните, в които ще се приема аудиото и ще бъде възпроизвеждано акустично (*Вижс т. 2.1.*), е необходимо да съдържа аудио кодек чип, т.е. филтри, аналогово-цифрови преобразуватели (АЦП), цифрово-аналогови преобразуватели (ЦАП) и др. като направата им от началото ще коства изключително много време, особено ако трябва да се интегрира заедно с Arduino. Затова подходящо решение в случая се явява един от микроконтролерите на Espressif Systems - ESP-LyraT V4.3[8], който е идеалната хардуерна платформа за разработка на приложения и проекти включващи аудио и видео разпространение, обработка и възпроизвеждане. ESP-LyraT V4.3 съдържа в себе си хардуерен модул ESP-WROOWER-E, който предоставя Wi-Fi и Bluetooth възможност за свързаност, което го прави подходящ за безжични аудио приложения, например, Wi-Fi или Bluetooth аудио високоговорители, дистанционни контролери, базирани на реч, свързани умни устройства с една или повече аудио функционалност и др, .



Фиг. 1.15. Микроконтролер Espressif ESP32 модел LyraT V4.3

Втора глава

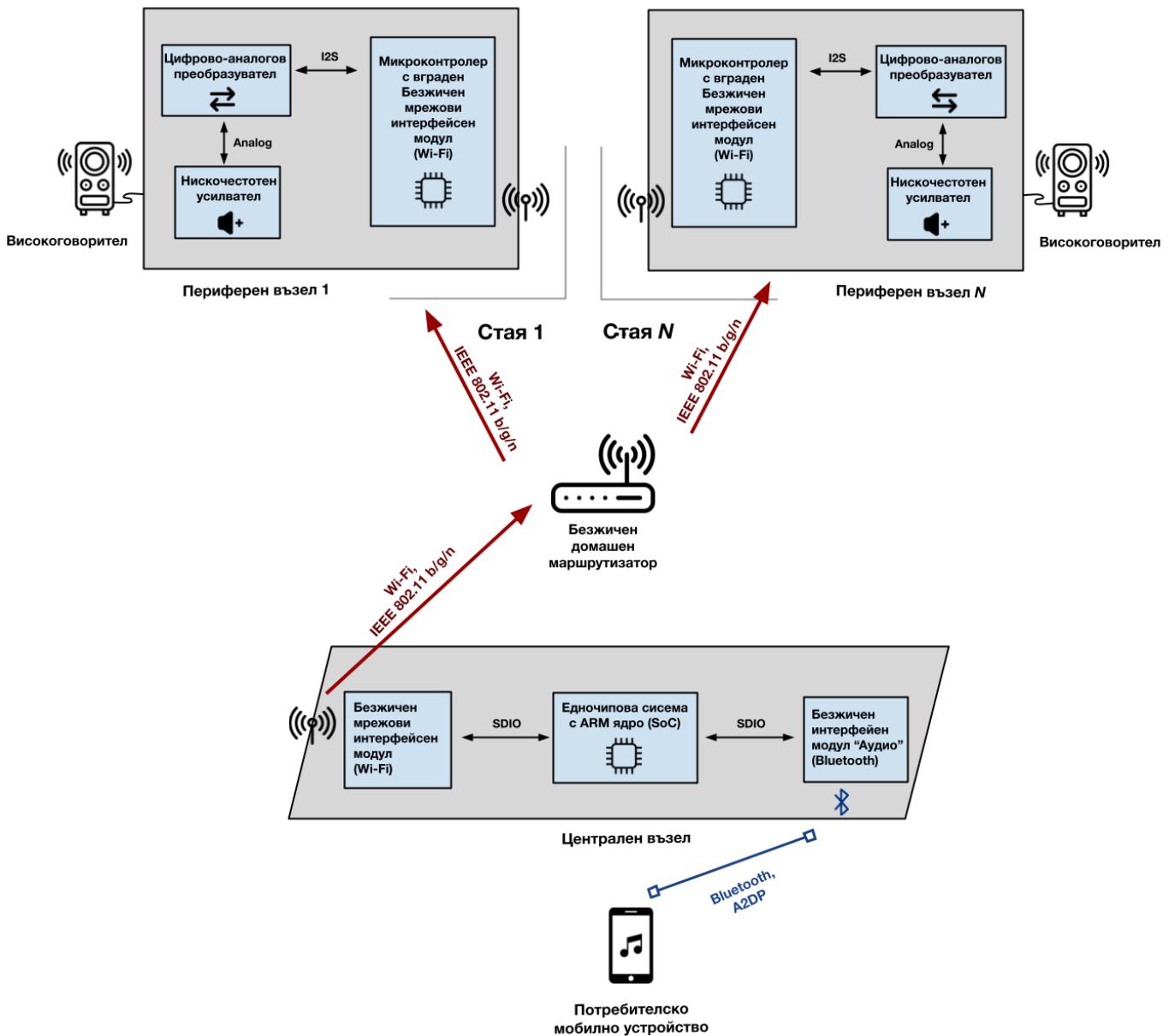
Проектиране на системна архитектура

В тази глава са приложени блок схеми и диаграми за визуализиране на самата архитектура както като общо, така и в частност. Описани са и основните функционални изисквания към работоспособния модел на дипломната работа и отделно за централния и периферните възли. Тази глава включва и подбор на основните инфраструктурни компоненти за отделните възли, които ще способстват за реализацията на текущата дипломна работа.

2.1. Системна диаграма

Диаграмата на *Fig 2.1* показва различните основни блокови елементи от които се състои:

- Потребителското мобилно устройство, което ще служи като източник на аудио поточните данни. То ще бъде свързано посредством Bluetooth към централния възел.
- Централният възел ще приема изпратените по Bluetooth аудио поточни данни през неговия безжичен интерфейсен “аудио” модул
- Данните ще бъдат пренасочени посредством SDIO комуникация към процесорното ARM ядро където ще претърпят необходимите промени
- Данните ще бъдат изпратени към безжичния мрежов интерфейс на централния възел, от където ще бъдат разпространени през домашната Wi-Fi мрежа
- Домашният безжичен маршрутизатор ще се погрижи аудио поточните данни да бъдат получени от периферните възли
- Периферните възли чрез своите вътрешни модули приемат данните ще извършват необходимите обработки. След което цифровите аудио данни се преобразуват в аналогов сигнал за възпроизвеждане чрез интегрирания ЦАП.
- Сигналът минава през нискочастотния усилвател към говорителя, където се възпроизвежда акустично.



Фиг. 2.1. Опростен модел на структурата на текущата дипломна работа

2.2. Дефиниране на функционалните изисквания към системата

Относно цялостният работоспособен модел на домашната озвучителна система са дефинирани следните изисквания:

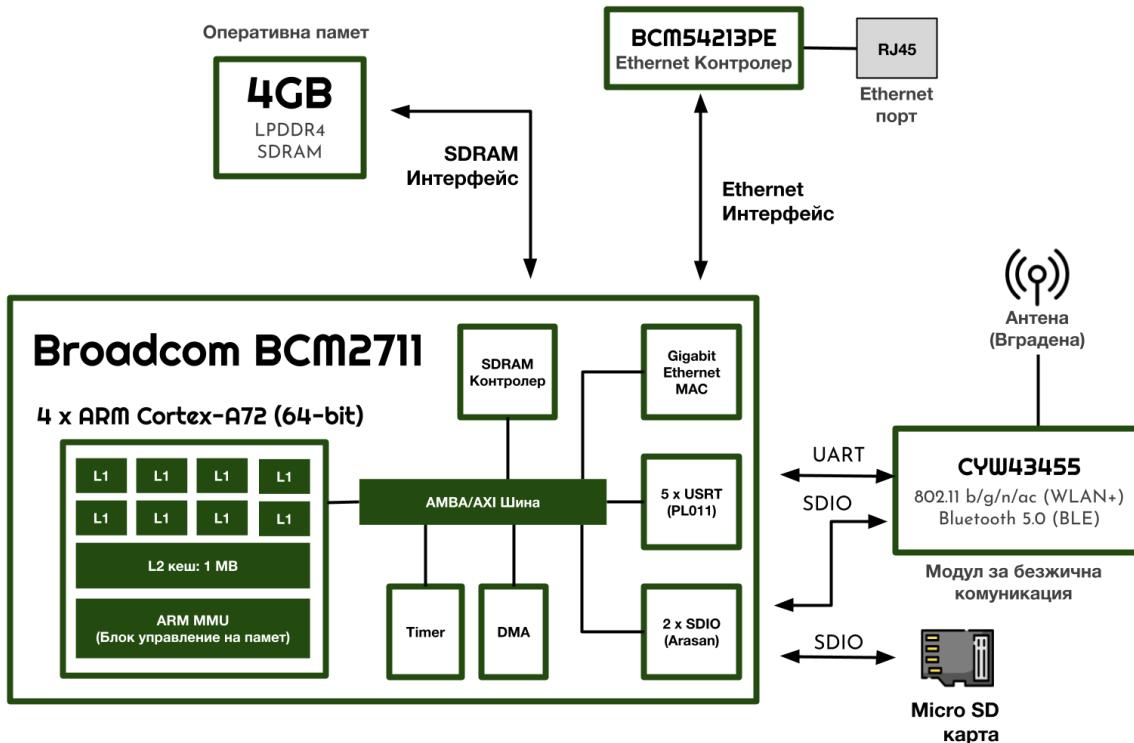
- Домашната система да представлява архитектура, състояща се от един централен възел и множество периферни възли (представляващи множеството стаи у дома), които обменят помежду си информация в реално време
- Централният възел да приема аудио потоци от Bluetooth устройство (мобилен телефон, лаптоп)
- Централният възел да бъде свързан към безжичната локална мрежа, т.е. да има собствен IP адрес, и да изпраща, насочва и управлява конкретните аудио потоци към съответните периферни възли
- Централният възел да има възможност да изпраща контролни съобщения за изключване/вклучване на звука (*mute/unmute*) на даден един или няколко периферни възли
- Всеки периферен възел да бъде свързан с безжичната локална мрежа, т.е. всеки от тях да има свой собствен IP адрес, да приема адресираният към него съответен аудио поток и да го възпроизвежда акустично
- Всеки периферен възел да изпълнява изпратените от централния възел команди

2.3. Хардуерна блок схема на централния възел

Централният възел се базира на едноплатковия компютър “Raspberry Pi 4B”, модел с 4GB RAM памет, който притежава следните хардуерни компоненти, необходими за реализирането на централния възел като част от дипломната работа:

- Broadcom BCM2711 чип[11] - по-мощен ARM (“Advanced RISC Machine”) процесор в сравнение с предходните микрокомпютри Raspberry Pi. В приложение за цялостно поемане на изчислителната мощност, изисквана от софтуерната част на дипломната работа. Спецификации:
 - Архитектура ARMv8
 - Процесор с четири ядра (Quad core) Cortex-A72
 - 64-битов

- Едночипова система (System on a Chip - SoC)
- Работи на честота от 1.8 GHz
- 4GB LPDDR4 SDRAM - динамична RAM памет с ниска консумация от четвърто поколение (Low-power Double Data Rate Synchronous Dynamic RAM)
- Micro SD - за съхранение на цялостната операционна система и нейната файлова структура.
- *BCM54213PE* контролер[11] + Gigabit Ethernet порт при предпочтение за директна жична връзка към домашния маршрутизатор
- *CYW43455*[11] - Wi-Fi + Bluetooth модул, което позволява обмен на голямо количество информация без огромни изисквания към консумираната енергия. Поддържа Wi-Fi стандартите 802.11 b/g/n/ac, което означава че работи на честоти 2.4 GHz и 5 GHz.



Фиг 2.2. Хардуерна блок схема на използвани компоненти в Raspberry Pi 4B

2.4. Дефиниране на функционални изисквания към централния възел

- Да бъде базиран на едноплатковия компютър Raspberry Pi 4
- Да използва Linux-базирана операционна система
- Да бъде с 64-битова ARM-базирана компютърна архитектура
- Да бъде разпознаваем като Bluetooth устройство, което :
 - е видимо за всички активни Bluetooth устройства
 - Позволява установяването на връзка (pair)
 - е приемник на безжична аудио информация
 - Разбира от A2DP профил
- Да обработва приетия по Bluetooth аудио поток, да го изпраща към множество периферни възли по Wi-Fi (IEEE 802.11 b/g/n)
- Да използва HTTP като протокол за разпространение на аудио потоци

2.5 Подбор на основни инфраструктурни компоненти за централния възел

Тук ще бъдат изброени подбранныте компоненти (технологии, решения, софтуер), които ще бъдат необходими и са използвани за реализирането на Raspberry Pi 4 микрокомпютър като централен възел в системата.

2.5.1 Linux операционната система

За осъществяването на аудио функционалността на централният възел се нуждае от стабилна звукова архитектура, достатъчно лесна и разбираема за конфигуриране и работа с нея. Тъй като централният възел ще представлява едноплатков микрокомпютър Raspberry Pi 4 и ще използва ресурсите на Linux операционната система, в конкретния случай ще се използва вече обяснената и дефинирана в преходната *Първа глава ALSA архитектура*[12], която удовлетворява целите на настоящата дипломната работа (*Вижс. 1.3.1. ALSA софтуерна архитектурна платформа*). И отново, базирайки се на факта че ще бъде използвана Linux операционна система, ще бъде използван **BlueZ протоколния стек**[17] за реализацията на Bluetooth свързаността, тъй като **BlueZ** е толкова внедрен в Linux операционните системи колкото и **ALSA**. **BlueZ** е осигурява поддръжка за основните Bluetooth слоеве и протоколи. Той е гъвкав, ефективен и използва модулно изпълнение.

2.5.2 BlueZ протоколен стек

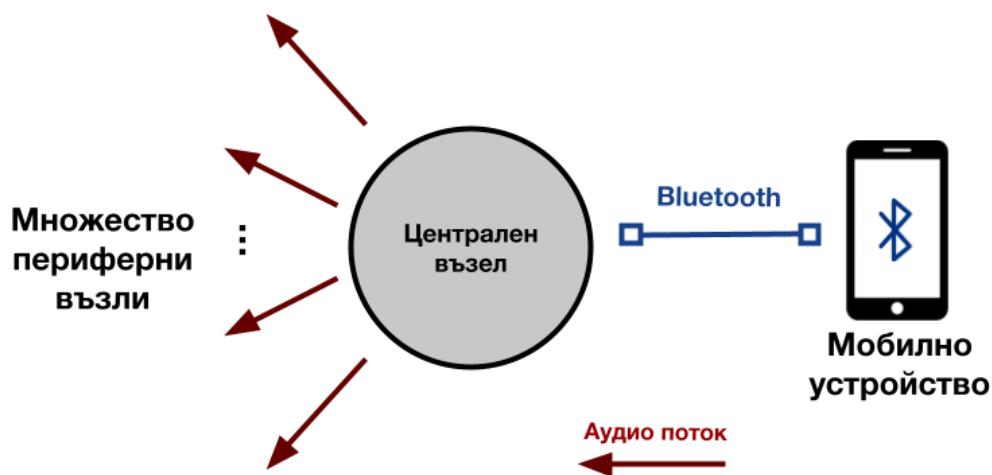
За целите на дипломната работа централният възел позволява безжична комуникация с мобилно устройство. Мобилното устройство се разглежда като източник, откъдето изхожда аудио информацията, т.е. мястото където потребителят избира музиката, която да пусне, без значение от платформата за музикални услуги - *Spotify, YouTube, Deezer* и т.н. Централният възел се разглежда като приемник, който следва да дублира подадената му информация и да я разпространи към съответните периферни възли. Комуникацията се изгражда на базата Bluetooth и A2DP профила за “audio streaming”, както е показано на *Фиг. 3.13.*

BlueZ[17] предоставя и много спосobi за конфигуриране на Bluetooth системната услуга, и те ще бъдат разгледани по-подробно в следващата глава.

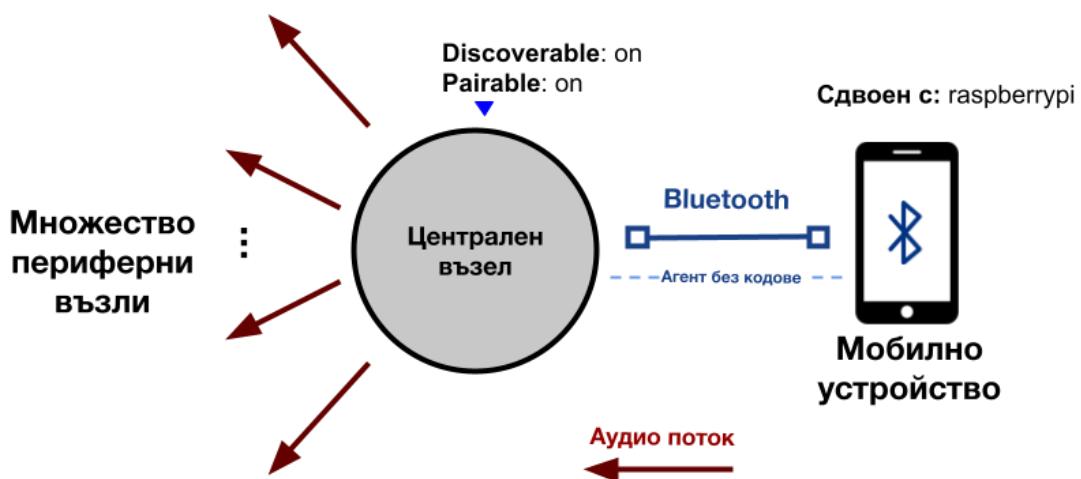
Централният възел е конфигуриран така, че да бъде лесно разпознаваем за други Bluetooth устройства:

- Автоматично своят Bluetooth модул при зареждане на операционната система.
- Настроен е така, че да бъде постоянно откриваем (“discoverable”) и готов за осъществяване на връзка между отделните устройства (“pairable”)

- Използване на агент, който ще елиминира необходимостта от обмен на код за потвърждение на връзката (“Bluetooth pairing code”), т.е. централният възел ще бъде “приет” от потребителското устройство като Bluetooth аудио устройство (слушалки, колонка). Известен като **NoInputNoOutput**.
- Запомня “paired” устройствата, за да може повторното осъществяване на връзка да бъде за по-малко време



Фиг. 2.3. Модел на безжична Bluetooth връзка



Фиг. 2.4. Изграждане на безжична Bluetooth връзка

2.5.3 PipeWire мултимедийна система

PipeWire[21] е много подходящ избор що се касае за съвременни решения, които да способстват реализирането на една такава озвучителна система върху Linux-базирана операционна система. Тя работи паралелно и заедно с **ALSA звуковата архитектурата** като те постоянно си обменят аудио и видео информация. Заедно с това има собствена имплементация на широко разпространения из множеството Linux устройства **PulseAudio звукова сървърна програма**[20], за да може да има поддръжка за приложенията в операционната система, които поради една или друга причина са зависими от **PulseAudio** (както например следващият компонент, който изисква да бъде използван **PulseAudio**). Тази имплементация е известна като *pipewire-pulse*[22] и е използвана в настоящата дипломна работа.

В случая Pipewire е и много подходящ тъй като има безпроблемна интеграция с **BlueZ** като лесно може да обработи приетите по Bluetooth аудио поточни данни.

Също така **PipeWire** може да бъде конфигуриран така, че всякакви аудио поточни данни, приети по Bluetooth да ги пренасочва към звуковата карта за възпроизвеждане. **Raspberry Pi OS** версия 12 Bookworm[23] идва с предварително зададена конфигурация за **PipeWire** която да изпълнява точно това действие без необходима намеса от програмиста / инженера.

2.5.4 Liquidsoap скриптов източник на мултимедия

Не толкова нов (проекта е започнат през 2004), но продължаващ активно да се поддържа и разработва, **Liquidsoap**[26] скриптовият източник на мултимедия е способ, който заедно с **Icecast**[27] (обяснен подробно в следващата точка) са в основата на разпространението на онлайн аудио и видео съдържание в реално време, или т.нар. интернет радиостанции (internet radios) and уеб телевизия (webtvs). Изцяло с отворен код, **Liquidsoap** всъщност от своите разработчици е категоризиран като “*скриптов език за разпространение на аудио и видео*” (“*Audio & Video Streaming Language*”) [25], защото се състои от специално дефиниран скриптов език, чрез който се обуславя създаването на

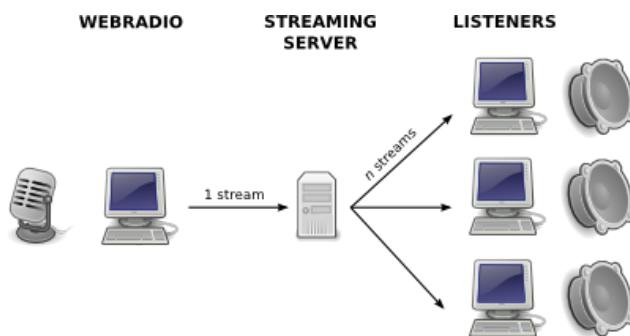
аудио и видео потоци, които могат да имат всевъзможни източници и приемници, като например: звуков файл или папка / плейлист, които да бъдат подадени към радио сървър като Icecast (най-типичният случай), аудио запис произхождащ от микрофон на устройството да бъде пренасочен към файл, да бъде запазен или пренасочен към възпроизвеждащо устройство и множество други случаи и примери. (Фиг. 2.6)



Фиг. 2.5. Лого на Liquidsoap[26]

На фона на своите алтернативи, Liquidsoap се откроява с това, че:

- **Изключително гъвкав и програмируем:** Прави го идеален за създаване на способи за динамично поточно предаване, като кръстосано затихване (crossfade), миксиране на множество входове, планиране и създаване на персонализирани радио потоци, което може да е по-трудно за прилагане с други инструменти.
- **Поддръжка на множество формати и протоколи:** Liquidsoap поддържа широка гама от аудио формати (MP3, AAC, Ogg Vorbis и др.) и протоколи за разпространение в реално време (Icecast, Shoutcast, HLS).
- **Модулен дизайн:** позволява на потребителите да добавят или премахват функции според нуждите, което го прави лек и ефективен за различни случаи на употреба.



Фиг. 2.6. Типична употреба на Liquidsoap[25]

В настоящата дипломна работа **Liquidsoap** ще бъде интегриран за пренасочване и кодиране в MP3 формат на прииждащите Bluetooth аудио поточни данни, които реално биват предоставяни на **Liquidsoap** през **PipeWire** (и по конкретно модулът *pipewire-pulse*) към следващият компонент от системата **Icecast**.

2.5.5 Icecast мултимедиен сървър

Icecast[27] е мултимедиен сървър с отворен код, предназначен за излъчване и разпространение на аудио съдържание през глобалния интернет. **Icecast** осигурява необходимата инфраструктура за създаване на интернет радиостанции или поточно предаване на аудио в реално време, като целта поради, която е избран в текущата дипломна работа е точно тази - способността му лесно да предоставя начин, по който да разпространява аудио данни в реално време на множество устройства в локалната мрежа. **Icecast** поддържа различни аудио формати като MP3, Ogg Vorbis и Opus, което го прави универсален за различни видове аудио съдържание.



Фиг. 2.7. Лого на *Icecast*[27]

Icecast2 също е много конфигурируем, позволяйки на потребителите да персонализират настройките на потока, да управляват слушателите и да се интегрират със софтуер за стрийминг като **Liquidsoap**, обяснен в предходната точка, за неограничени възможности за разпространение на аудио. Освен това, **Icecast** е добре поддържан от общност от разработчици и потребители, което гарантира непрекъснати актуализации и подобрения.

Icecast може да бъде използван чрез една от неговите основни версии - **icecast** - която е по-старата и вече неподдържана, или **icecast2** - която е по-нова и съвременна, използвана в текущата дипломна работа.

2.5.6 Docker платформа за предоставяне на софтуерни контейнери



Фиг. 2.8. Лого на Docker[28]

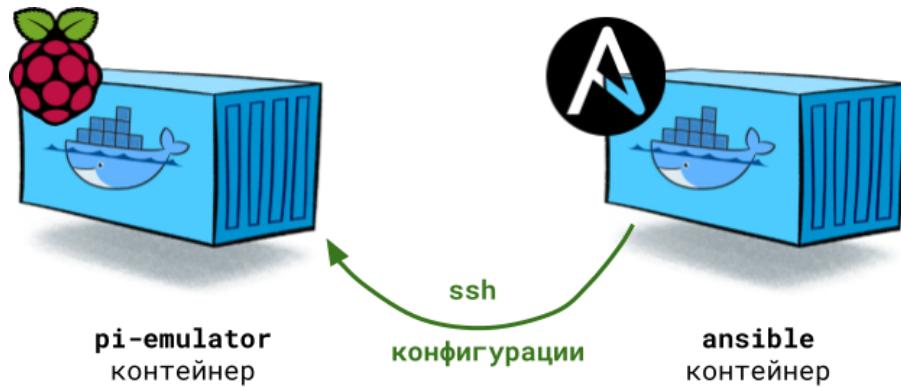
Docker[28] е платформа с отворен код, предназначена да улесни разработването, внедряването и управлението на приложения с помощта на контейнеризация. “Контейнерите” (containers) са леки, самостоятелни “пакети”, които обединяват приложение и всичките решения и инструменти, библиотеки и конфигурации, от които зависи (dependencies), необходими за изпълнение. За разлика от виртуалните машини, контейнерите споделят ядрото (kernel) на операционната системата, което ги прави много по-ефективни и по-бързи за стартиране, като същевременно предоставя изолирана среда за всяко приложение.

Основната цел на **Docker** е да рационализира процеса на разработка и внедряване на софтуер, като гарантира, че приложенията работят последователно в различни среди. Това елиминира често срещания проблем, където се забелязва, че приложението работи както се очаква само на определена машин, но не и на другите.

В текущата дипломна работа Docker ще бъде използван в две направления. Първото е за емулиране на Raspberry Pi OS Lite, 12 Bookworm (ревизия **2024-07-04**)[24] в изолирана среда с цел тя да бъде конфигурирана спрямо изискванията на централния възел (и използването на до тук описаните архитектурни компоненти за постигане целите на дипломната работа) и заедно с направените промени тя да бъде запазена (export) във “.img” формат, който директно да може да бъде зареден / флашнат (flash) върху SD карта чрез например **balenaEtcher**[29], **Raspberry Pi Imager**[23] или друг софтуер за флашване

на операционни системи върху SD карти. Най-просто казано чрез тази стъпка се изгражда изцяло персонализирана операционна система, въз основа на Raspberry Pi OS Lite, 12 Bookworm (ревизия 2024-07-04)[24], със необходимите конфигурации, изпълващи изискванията за централен възел, готова да бъде заредена върху SD карта и стартирана на самия Raspberry Pi . По този начин цялата конфигурация е лесно да бъде копирана и приложена върху множество Raspberry Pi 4 устройства, спестявайки време и усилия.

Второто направление е за дефинирането и прилагането на конкретните конфигурации и промени върху самата операционна система. Те отново са в изолирана среда и на практика чрез “отдалечен” достъп прилагат зададените промени. Но тази стъпка зависи от звеното описано в следващата точка, тъй като чрез **Ansible**[30] се реализира задаването и прилагането на съответните промени по операционната система, а не чрез **Docker**. **Docker** единствено способства на **Ansible** като софтуерен инструмент да бъде изолиран и конфигурациите да бъдат компактни.



Фиг. 2.8. Взаимодействие между отделните контейнери

2.5.7 Ansible софтуерен инструмент за автоматизация

Ansible[30] е инструмент за автоматизация с отворен код, използван за управление на конфигурацията, внедряване на приложения и автоматизация на задачи на множество системи и машини. Той позволява улесненото дефиниране и управление на инфраструктури с помощта на прости, четими “за обикновения човек” файлове написани на конфигурационния език YAML. **Ansible** работи без агенти, което означава, че не

изисква инсталиране на специален софтуер. По този начин **Ansible** си осигурява лекота и улеснение при различни настройки.



Фиг. 2.9. Лого на Ansible[30]

Ansible, както бе споменато, ще бъде използван с цел автоматизиране на конфигурацията, която ще бъде заредена върху операционната система. Това води със себе си и допълнителни предимства като:

- наблюдение последователността на направените конфигурации
- тъй като конфигурациите се пазят в YAML файлове, това спомага следене на версите на самите файлове, т.е. интеграция с Git като система за следене на версии
- лесна модификация на конфигурациите и т.н.
- избягване на грешки при въвеждането на конфигурациите

2.6. Хардуерна блок схема на периферните възли

Всички периферни възли използват едноплатковият микроконтролер ESP32 модел “LyraT V4.3”, който притежава следните хардуерни компоненти, необходими за реализирането на периферния възел като част от дипломната работа:

- ESP32-WROVER-E[8][31] - модул, чието предназначение е да предоставя Wi-Fi и Bluetooth свързаност за обмен на данни, като включва в себе си външна 4MB SPI flash памет и 8MB PSRAM памет за “тъкаво съхранение на данни”[8]. ESP32-WROVER позволява интегритет с различна периферия като интерфейс за SD карта, Ethernet, UART, I2S и Аудио кодек чип.
 - Поддържа Wi-Fi стандартите 802.11 b/g/n (802.11n до 150 Mbps), което означава че работи на честоти 2.4 GHz и 5 GHz

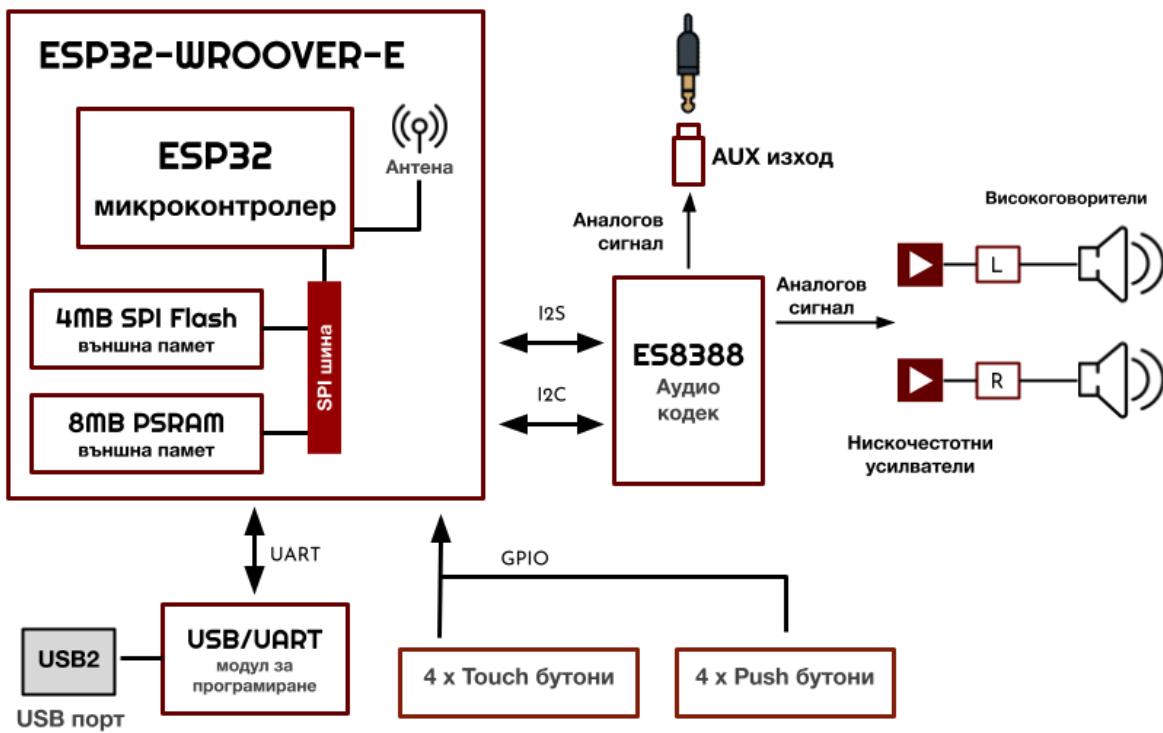
- Антената на този модел е вътрешна



Фиг. 2.10. ESP32-WROVER-E модул

- Аудио кодек чип ES8388[32] - висока производителност с ниска мощност и цена. Състои се от двуканален АЦП, двуканален ЦАП, микрофон усилвател, усилвател за слушалки, цифрови звукови ефекти и аналогови функции за смесване и усилване.
 - 24-bit, 8 kHz до 96 kHz честота на дискретизация
 - 95-96 dB SNR (signal to noise ratio)
 - Работно напрежение между 1.8V и 3.3V
 - Консумация:
 - 7 mW при възпроизвеждане;
 - 16 mW при възпроизвеждане и звукозапис;
- Micro SD - използва се за запис или възпроизвеждане на аудио файлове под различни формати MP3, WAV и т.н.
- Аудио изходи
 - 1 ляв и 1 десен изход за свързване на говорител със съпротивление от 4Ω и 3 W
 - 1 аудио жак за свързване на слушали или други високоговорители

Приемането на адресиран аудио поток и конфигурационни данни протича по начина, показан на хардуерната блокова схема на *Фиг. 3.18.*:



Фиг. 2.11. Хардуерна блок схема на използваниите компоненти в платка *ESP32 LyraT V4.3*

2.7. Дефиниране на функционални изисквания към периферните възли

- Да се базира на фамилия микроконтролера ESP32
- Да притежава Wi-Fi модул поддържащ минимум IEEE 802.11 b
- Да притежава Wi-Fi свързаност, т.е. да притежава IP адрес
- Да прочита идващата аудио информация и да я преобразува в съответния звуков сигнал
- Да притежава аудио кодек (АЦП + ЦАП) за трансформацията на дигиталните аудио поточни данни в аналогов сигнал, които да бъде възпроизведен акустично

- Да има възможност да декодира аудио информация предоставена в различни аудио формати (AAC, MP3, т.н)
- Да притежава интерфейс за пренос на данни съгласно I2S (Inter Integrated-Circuit Sound) протокола
- Да притежава възможност за свързване на говорител, който ще възпроизвежда звуковия сигнал акустично

2.8. Подбор на основни инфраструктурни компоненти за периферните възли

Тук ще бъдат изброени подбраните компоненти (технологии, решения, софтуер), които ще бъдат необходими и са използвани за изграждането на цялостната работеща инфраструктура на избраните вече ESP32-LyraT V4.3 микроконтролери като периферни възли в системата.

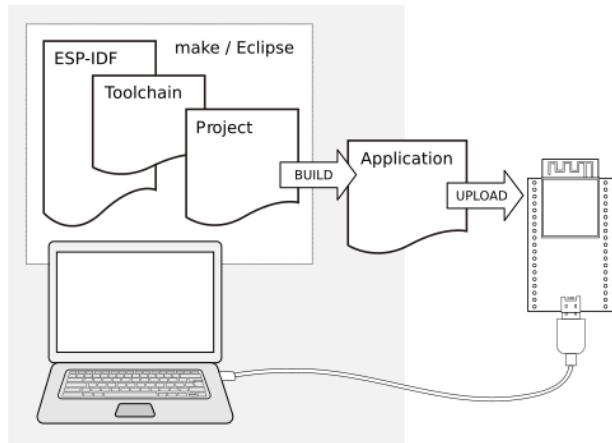
2.8.1 Софтуерните платформи за разработка ESP-IDF и ESP-ADF

Тези две платформи **ESP-IDF[8]** (Espressif IoT Development Framework) и **ESP-ADF[9]** (Espressif Audio Development Framework) са платформи изцяло с отворен код и официално поддържани от **Espressif Systems** за разработка върху произвежданите от тях микроконтролери. Те сами предоставят основните хардуерни и софтуерни ресурси, които помагат на разработчиците на приложения да изграждат своите идеи. За програмирането и цялостното реализиране на микроконтролерите ESP32-LyraT V4.3 като периферните възли спрямо изискванията, се използват точно тези софтуерни платформи.

ESP-IDF е предназначена за бърза разработка и развитие на приложения за пряко свързани или обвързани с “Интернет на нещата” (“Internet of Things” -IoT), с Wi-Fi и/или Bluetooth свързаност, управление на захранването и др. системни функции (*Фиг. 2.2.*).

ESP-IDF поддържа функции като Wi-Fi и/или Bluetooth (класически и с ниска консумация на мощност, т.е. **Bluetooth Classic** и **Bluetooth Low Energy - BLE**), което го

прави идеален за създаване на свързани, интелигентни устройства. Той предоставя широк набор от инструменти, периферни драйвери и система за изграждане, позволяваща на разработчиците да изграждат стабилни приложения с висока производителност. **ESP-IDF** се използва широко в IoT общността за проекти, вариращи от домашна автоматизация до индустритални системи за контрол.



Фиг. 2.12. Стъпките за разработка приложения за ESP32

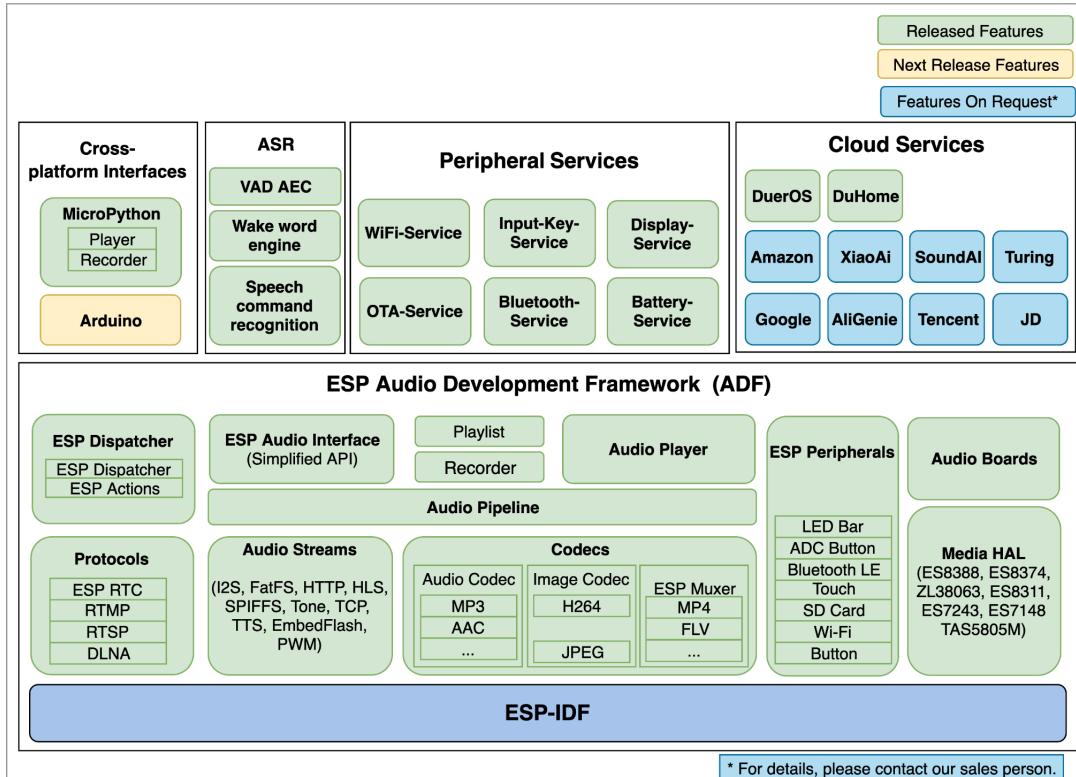
ESP-ADF е предназначена за изграждане на IoT аудио приложения и е изградена въз основа на вече пояснената **ESP-IDF** (Фиг. 2.13). Тя предоставя богат набор от библиотеки и инструменти, специално пригодени за аудио обработка, което я прави идеален за разработване на проекти като интелигентни високоговорители, системи за гласово разпознаване или приложения за аудио разпространение в реално време (Фиг. 2.13), което е и целта в настоящата дипломна работа.

ESP-ADF поддържа различни аудио кодеци, входно/изходни устройства, аудио ефекти и интеграция с облачни услуги за гласови асистенти. Той опростява разработката на аудио-базирани приложения, като предлага предварително изградени компоненти за аудио запис, възпроизвеждане и комуникация.

ESP-ADF има определен набор от микроконтролери [45], които поддържа, един от които е вече избраният за целите на дипломната работа ESP32-LyraT V4.3. Той е най-основният микроконтролер с най-широк обхват от възможни приложения.

За дипломната работа ще бъдат използвани следните версии на платформите:

- **ESP-IDF** - v4.4.4
- **ESP-ADF** - v2.6



Фиг. 2.13. ESP-ADF компонентни разширения (версия v2.6)[8]

2.8.2 FreeRTOS операционна система в реално време за микроконтролери



Фиг. 2.14 Лого на FreeRTOS[33]

FreeRTOS[33] е “олекотена” (lightweight) операционна система в реално време (Real-Time Operating System - RTOS) с отворен код, предназначена за вградени системи и микроконтролери. Тя предоставя възможности за многозадачност (multitasking), т.e. различни задачи или процеси да се изпълняват едновременно, да управляват ресурси и да

реагират ефективно на събития в реално време. FreeRTOS се използва широко във вградени системи, където точното време (timing) и управление на задачите (task management) са критични.

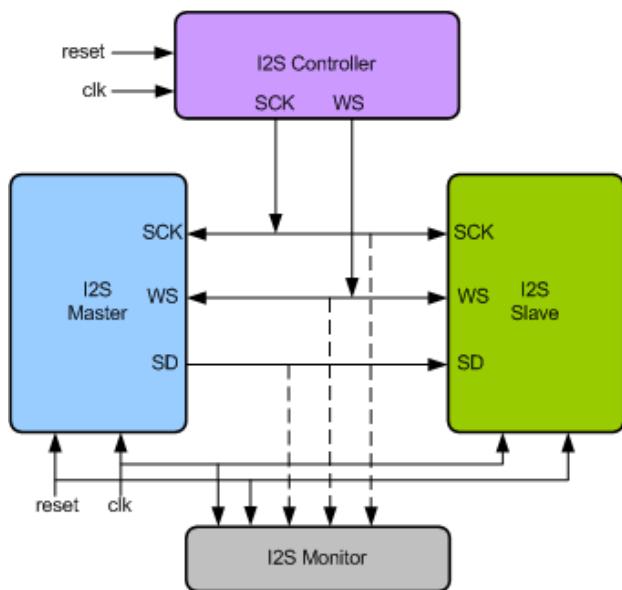
Във връзка с **ESP-IDF** (Espressif IoT Development Framework) и **ESP-ADF** (Espressif Audio Development Framework), **FreeRTOS** е основната операционна система, която управлява операционните задачи и системните ресурси за приложения, работещи въз основа на ESP32 микроконтролери. Както **ESP-IDF**, така и **ESP-ADF**, са изградени върху FreeRTOS, като използват неговите функционалности.

2.8.3 I2S сериен интерфейсен протокол за предаване на цифрово аудио

I2S[34] или Inter-Integrated Circuit Sound (по-правилно да се изписва като **I²S**) е комуникационен протокол, предназначен за предаване на цифрови аудио данни между устройства, като микроконтролери, цифрови сигнални процесори и аудио компоненти като кодеци, ЦАП (цифрово-аналогови преобразуватели) или АЦП (аналогово-към -цифрови преобразуватели). I2S се използва за изпращане на висококачествени, синхронизирани аудио данни в стерео формати. Той е оптимизиран за обработка на аудио потоци, като осигурява предаване с ниска латентност и висока точност.

Приликата му с **I2S** и **I2C**[35] (Inter-Integrated Circuit) не е случаена тъй като **I2S** е изграден на базата на **I2C** комуникационния протокол, който се използва за свързване и управлението на множество нискоскоростни устройства в паралел като сензори, дисплеи или EEPROM към микроконтролер, като се използват само два проводника: SDA (линия за данни) и SCL (линия на тактов сигнал). **I2C** позволява комуникация между множество устройства на една и съща шина и се използва широко във вградени системи за комуникация с малък обхват и ниска честотна лента между чипове.

I2S е неразделна част от **ESP-ADF** (Espressif Audio Development Framework), защото е основният интерфейс, използван за обработка на аудио данните. В **ESP-ADF**, **I2S** за *ESP32-LyraT V4.3*, който ще бъде използван за целите на дипломната работа свързва **ESP-WROOVER-E** модулът с интегрирания **ES8388** аудио кодек чип, който ще преобразува получените аудио данни във сигнал готов за възпроизвеждане.



Фиг. 2.15. Примерна схемна диаграма на работата на I2S[34]

Трета глава

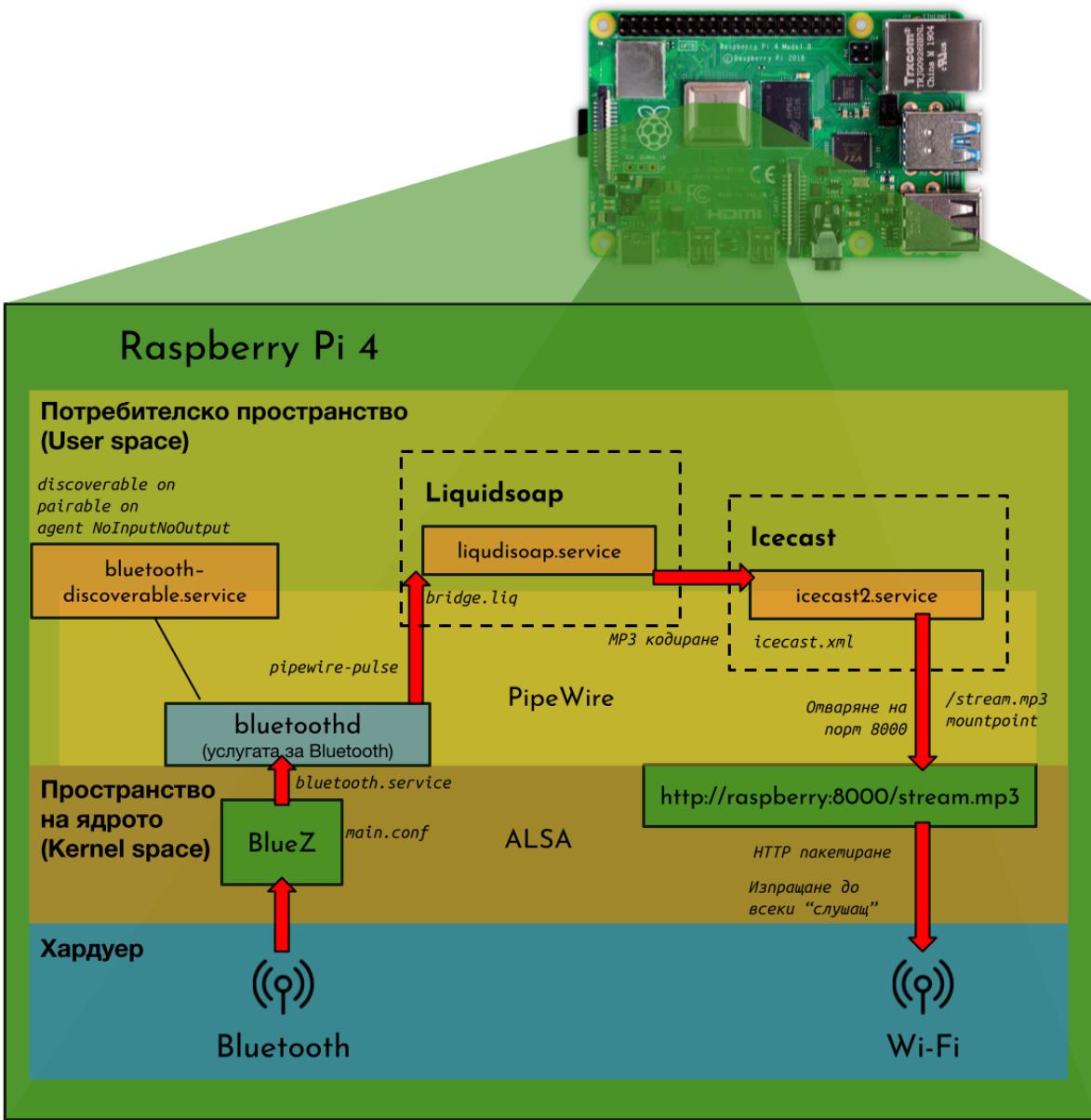
Проектиране на “централен възел” и “периферни възли”

Тази глава цели да изясни способите и начините, които са използвани за изграждането на работоспособния модел на дипломната работа с вече дефинирани изисквания към нея. Приложено е и детайлно функционално описание на хардуерните и софтуерните компоненти в дипломната работа.

3.1 Функционално описание на централния възел

Централният възел е много по-комплексен спрямо взаимовръзките между отделните компоненти, които го изграждат като такъв, отколкото периферните възли, но с предоставената диаграма на *Фиг 3.1.* може да се разбере в каква последователност са подредени те и как аудио данните преминават от един компонент в друг.

- Началната точка на цялата система е потребителското устройство. То установява Bluetooth безжична комуникация с централния възел, за което е отговорен **BlueZ протоколния стек**.
- Централният възел остава винаги откриваем (**discoverable**) и сдвояем (**pairable**) като регистрира агент (**NoInputNoOutput**), който не изисква обмяната на кодове за сдвояване
- При подаването на звукови данни **BlueZ** ги пренасочва през **ALSA звуковата архитектура** (защото всичко що се отнася до звук минава през нея) и се подават директно на **PipeWire мултимедийната система**



Фиг. 3.1. Компонентите и техните взаимосвързаности, използвани от централния възел

- **PipeWire** от своя страна пренасочва потока данни към потребителската скриптов програма, написана на **Liquidsoap**, която “изисква” от него да прочете тези данни. Всъщност тя изисква да прочете данните от **PulseAudio**, затова тук интеграцията чрез *pipewire-pulse* намира приложение
- **Liquidsoap скрипта** прочита аудио данните, които му биват подадени и ги кодира в MP3 формат (частотата на дискретизация и броят канали автоматично се задават, тъй като **Liquidsoap** е способен да ги “отгатне” от подадения аудио поток), като

подава вече кодираните данни към **Icecast медиийния сървър**, като също така дефинира точката (mountpoint), която ще бъде използвана за “слушане” на тези аудио данни

- **Icecast** като финално звено регистрира заявената точка за “слушане” и разпространява кодираните аудио данни през нея посредством **HTTP** протокола към всеки периферен възел, който “слуша” на тази точка
- Тъй като **HTTP** е интернет протокол данните се подават към Wi-Fi мрежовата карта и оттам вече се разпространяват безжично в локалната мрежа

3.2 Реализация на централния възел

Тъй като стремежът (описание в т.2.5.6 и 2.5.7) е цялата конфигурация и настройки, които ще бъдат приложени върху едно Raspberry Pi 4 устройство, е първо да бъдат автоматизирани, второ да могат да се прилагат на много други евентуални Raspberry Pi 4 устройства и трето - за улеснение при модификации, ще бъдат използвани Docker и Ansible.

3.2.1 Конфигурация на Docker контейнери

Както е описано в т.2.5.6, Docker ще бъде използван за 2 неща:

- 1) За използване на контейнер, който ще се погрижи да емулира (да създаде виртуална среда за) Raspberry Pi OS Lite, 12 Bookworm (ревизия **2024-07-04**)[24] с цел създаването на потребителски конфигурираната операционна система, която да бъде използвана заредена върху Raspberry Pi 4, изпълнявайки изискванията за централен възел. Името на този контейнер ще бъде **pi-emulator**.
- 2) За създаване на контейнер, който изолирано ще се свързва отдалечно чрез **ssh сигурна връзка** за да осъществи промените които трябва да бъдат приложени, за да превърнат операционната система на Raspberry Pi в операционната система на централния възел. Както бе описано в т. 2.5.7 тези промени се пазят във **YAML** файлове, които **Ansible** използва за да ги приложи на практика. Тези **YAML** файлове съдържат т.н. задачи (tasks), които обуславят самите промени и Ansible ги прилага последователно една по една. За

по-голямо улеснение задачите могат да са разпределени по различни YAML файлове и/или да бъдат групирани в роли (roles). Групите от роли се наричат “представление” (play), а множеството от “представления” - “книга за представленията” или playbook. В текущия случай playbook има само един play. За целите на дипломнаата работа са подбрани 3 роли:

- **setup** - която ще се погрижи да изчисти всякакви грешки и да подготви ОС за работа (създаване на потребител, смяна на парола, инсталлиране на пакети)
- **bluetooth** - конфигурации свързани с Bluetooth
- **wi-fi** - конфигурации свързани с безжичното разпространение на аудио потоците по Wi-Fi

Името на този контейнер ще бъде **ansible**.

Накрая след като Ansible приложи всички дефинирани промени без проблем емулираната вече персонализирана операционна система се спира и се експортира във “*.img” формат, който позволява лесно да бъде зареден върху SD карта.

Dockerfile (Всичко необходимо за **ansible** контейнера):

Инсталиране на **Ansible** инструмента вътре в контейнера заедно със всички необходимо за него пакети и библиотеки.

```
FROM debian:bookworm-slim
```

```
RUN apt-get update && apt-get install -y \
    iutils-ping \
    openssh-client \
    sshpass \
    ca-certificates \
    python3 \
    pipx \
    vim

# Clear cache, reduce image size
RUN rm -rf /var/lib/apt/lists/*
```

```

ENV PATH=/root/.local/bin:$PATH

RUN pipx install --include-deps ansible-core

RUN mkdir -p /etc/ansible/ && \
echo "[local]\nlocalhost" > /etc/ansible/hosts

ENV ANSIBLE_GATHERING=smart \
ANSIBLE_SSH_PIPELINING=True \
EDITOR=vim

WORKDIR /etc/ansible/playbooks

```

compose.yml

За улеснение на управлението на двата контейнера ще бъде използван **docker compose**, който ще използва следната конфигурация:

```

services:
  pi-emulator:
    image: ptrsrpi/pi-ci
    entrypoint: []
    command:
      - bash
      - -c
      - |
        /app/run.py resize 8G -y
        /app/run.py start
    ports:
      - 2222:2222
    volumes:
      - ./dist:/dist

  ansible:
    image: justivo/ansible

```

```

command: ansible-playbook playbook.yml
depends_on:
  - pi-emulator
volumes:
  - ./config:/etc/ansible/playbooks/
  - ./dist:/distro

networks:
  default:
    driver: bridge
    ipam:
      config:
        - subnet: 172.16.1.0/24
          ip_range: 172.16.1.0/30
          gateway: 172.16.1.1

```

config/playbook.yml

Playbook конфигурацията на Ansible за изпълнението на всички задачи (и тези които са в роли)

```

- name: Creating custom Raspberry Pi OS image
  hosts: emulated
  gather_facts: false

  vars_files:
    - defaults/credentials.yml
    - defaults/wifi.yml
    - vars/credentials.yml
    - vars/wifi.yml

  pre_tasks:
    - name: Waiting the emulator to boot up the virtual pi
      ansible.builtin.import_tasks:
        file: tasks/wait.yml

```

```

roles:
  - setup
  - bluetooth
  - wi-fi

post_tasks:
  - name: Export the emulated image to *.img format
    ansible.builtin.include_tasks:
      file: tasks/export.yml

```

3.2.2 Подготовка на операционната система (“setup” роля)

config/roles/setup/tasks/main.yml

Основната setup “задача”, която извиква в себе си няколко “подзадачи”

```

---
- name: Package installation
  ansible.builtin.import_tasks:
    file: packages.yml

- name: Fixing boot mount mismatch
  ansible.builtin.import_tasks:
    file: boot.yml

- name: Setting up preferences
  ansible.builtin.import_tasks:
    file: prefs.yml

- name: User related settings
  ansible.builtin.import_tasks:
    file: users.yml

```

config/roles/setup/tasks/packages.yml

Първият набор от подзадачи конфигурира системата да използва за да инсталира **PipeWire**, както всички негови необходими пакети, по правилния начин. Кешът на пакетите се актуализира, за да се гарантира, че се използва най-новата информация за пакетите. Освен това се копира конфигурационен файл, за да се приоритизират пакети, свързани с **Liquidsoap** (защото ако бъдат инсталирани от хранилищата на Raspberry Pi, **Liquidsoap** ще има проблеми, докато ако се инсталират тези от хранилищата на Debian[36] - всичко е наред), и се инсталират Icecast, Liquidsoap и други основни пакети.

```
- name: Add bullseye-backports repository to sources list in order to install
pipewire
  ansible.builtin.apt_repository:
    repo: "deb http://deb.debian.org/debian bullseye-backports main contrib
non-free"
    filename: "bullseye-backports"
    state: present

- name: Install PipeWire and all its dependancies from bullseye-backports
  ansible.builtin.apt:
    name:
      - pipewire
      - wireplumber
      - pipewire-pulse
      - libspa-0.2-bluetooth
    default_release: "bullseye-backports"
    state: present

- name: Update apt
  ansible.builtin.apt:
    update_cache: true

- name: Prioritize liquidsoap-related packages from Debian repos
```

```

ansible.builtin.copy:
  src: files/ffmpeg.pref
  dest: "{{ apt_preferences_dir }}/ffmpeg.pref"
  mode: "0600"

- name: Install required packages
  ansible.builtin.apt:
    name:
      - bluez-tools # bt-agent
      - icecast2
      - liquidsoap
      - vim
    state: present

```

Алтернатива като команди:

```

echo "deb http://deb.debian.org/debian bullseye-backports main contrib non-free"
| sudo tee /etc/apt/sources.list.d/bullseye-backports.list
sudo apt-get update
sudo apt-get install -t bullseye-backports pipewire wireplumber pipewire-pulse
libspa-0.2-bluetooth
sudo apt-get update
sudo cp files/ffmpeg.pref /etc/apt/preferences.d/ffmpeg.pref
sudo chmod 0600 /etc/apt/preferences.d/ffmpeg.pref
sudo apt-get install bluez-tools icecast2 liquidsoap vim

```

config/roles/setup/files/ffmpeg.pref

Файла с предпочтенията за Debian хранилищата[36]

```

Package: ffmpeg libavcodec-dev libavcodec59 libavdevice59 libavfilter8
libavformat-dev libavformat59 libavutil-dev libavutil57 libpostproc56
libswresample-dev libswresample4 libswscale-dev libswscale6
Pin: origin deb.debian.org
Pin-Priority: 1001

```

config/roles/setup/tasks/boot.yml

Смяна на “точка за монтиране” (boot mountpoint) от **/boot** на **/boot/firmware** тъй като съществуват проблеми при зареждането на Raspberry Pi 4 OS Lite, 12 Bookworm.

```
- name: Change default boot mount point
  ansible.builtin.shell: sed -iE 's|\s/boot\s| /boot/firmware |g' /etc/fstab
```

Алтернатива като команда:

```
sed -iE 's|\s/boot\s| /boot/firmware |g' /etc/fstab
```

config/roles/setup/tasks/prefs.yml

Конфигуриране на клавиатурата да използва US американски шаблон по подразбиране и задаване на SSID и парола, които да използва, за да се свързва автоматично към локалната мрежа

```
- name: Change default keyboard layout to US
  ansible.builtin.lineinfile:
    path: /etc/default/keyboard
    regexp: "^\$XKBLAYOUT="
    line: '$XKBLAYOUT="us"'

- name: Automatically connect to SSID on boot
  ansible.builtin.blockinfile:
    path: /etc/wpa_supplicant/wpa_supplicant.conf
    append_newline: true
    prepend_newline: true
    block: |
      country=BG
      network={
        scan_ssid=1
        ssid="{{ wifi_ssid }}"
      }
```

```

    psk="{{ wifi_password }}"
    key_mgmt=WPA-PSK
}
```

Алтернатива като команди:

```
sudo sed -i 's/^XKBsetLayout=.*/XKBLayout="us"/' /etc/default/keyboard
sudo tee -a /etc/wpa_supplicant/wpa_supplicant.conf > /dev/null <<EOL
```

```

country=BG
network={
    scan_ssid=1
    ssid="{{ wifi_ssid }}"
    psk="{{ wifi_password }}"
    key_mgmt=WPA-PSK
}
EOL
```

config/roles/setup/tasks/users.yml

Премахване на логването без парола, създаването на нов потребител по подразбиране, задаване на администраторска парола, както и настройки за влизане чрез терминала и засилване сигурността при отдалечен достъп.

```
---
- name: Remove autologin
  ansible.builtin.shell: sed -i 's/--autologin root //'
  /etc/systemd/system/serial-getty@ttyAMA0.service.d/override.conf

- name: Remove ssh auto login
  ansible.builtin.shell: sed -i 's/permitEmptyPasswords yes/#PermitEmptyPasswords
no/' /etc/ssh/sshd_config

- name: Enable getty service for /dev/tty1
  ansible.builtin.systemd:
    name: getty@tty1.service
```

```

enabled: true

- name: Enable default /dev/tty1 login prompt
  ansible.builtin.copy:
    src: /etc/systemd/system/serial-getty@ttyAMA0.service.d/override.conf
    dest: /etc/systemd/system/getty@tty1.service.d/override.conf
    remote_src: true

- name: "Create default user `{{ user_name }}`"
  ansible.builtin.user:
    name: "{{ user_name }}"
    password: "{{ user_password | password_hash }}"
    groups:
      - sudo
    append: true

- name: Set root password
  ansible.builtin.user:
    name: root
    password: "{{ root_password | password_hash }}"

```

Алтернатива като команди:

```

sudo sed -i 's/--autologin root //'
/etc/systemd/system/serial-getty@ttyAMA0.service.d/override.conf
sudo sed -i 's/permitEmptyPasswords yes/#PermitEmptyPasswords no/'
/etc/ssh/sshd_config
sudo systemctl enable getty@tty1.service
sudo cp /etc/systemd/system/serial-getty@ttyAMA0.service.d/override.conf
/etc/systemd/system/getty@tty1.service.d/override.conf
sudo useradd -m -G sudo -p "$(openssl passwd -1 {{ user_password }})" {{ user_name }}
sudo usermod --password "$(openssl passwd -1 {{ root_password }})" root

```

3.2.3 Bluetooth конфигурации (“bluetooth” роля)

config/roles/bluetooth/tasks/main.yml

Задаването на централния възел винаги да бъде откриваем (**discoverable on**), сдвояем(**pairable on**) и да регистрира агента **NoInputNoOutput**, който да премахне зависимостта от кодове за сдвояване. За това е регистрира системна услуга *bluetooth-discoverable.service*, която да изпълнява всеки път при зареждане на ОС.

```
- name: Set default Bluetooth settings to act as speaker
  ansible.builtin.shell: >
    sed -i \
      -e 's/#AutoEnable.*/AutoEnable = true/' \
      -e 's/#JustWorksRepairing.*/JustWorksRepairing = always/' \
      -e 's/#FastConnectable.*/FastConnectable = true/' \
      -e 's/#AlwaysPairable.*/AlwaysPairable = true/' \
      -e 's/#DiscoverableTimeout.*/DiscoverableTimeout = 0/' \
      -e 's/#PairableTimeout.*/PairableTimeout = 0/' \
      {{ bluetooth_config }}

- name: Register Bluetooth discovery service
  ansible.builtin.copy:
    src: files/bluetooth-discoverable.service
    dest: /etc/systemd/system/bluetooth-discoverable.service
    mode: "644"

- name: Enable Bluetooth discovery service on boot
  ansible.builtin.systemd_service:
    name: bluetooth-discoverable
    enabled: true
```

config/roles/bluetooth/files/bluetooth-discoverable.service

Системната услуга за Bluetooth конфигурация при зареждане на ОС.

[Unit]

Description=Make Bluetooth discoverable on boot

After=bluetooth.target

[Service]

ExecStart=/bin/bash -c "/usr/bin/bluetoothctl discoverable on &&

/usr/bin/bt-agent -c NoInputNoOutput"

KillSignal=SIGINT

[Install]

WantedBy=multi-user.target

Алтернатива като команди:

```
sudo sed -i \
-e 's/#AutoEnable.*/AutoEnable = true/' \
-e 's/#JustWorksRepairing.*/JustWorksRepairing = always/' \
-e 's/#FastConnectable.*/FastConnectable = true/' \
-e 's/#AlwaysPairable.*/AlwaysPairable = true/' \
-e 's/#DiscoverableTimeout.*/DiscoverableTimeout = 0/' \
-e 's/#PairableTimeout.*/PairableTimeout = 0/' \
/etc/bluetooth/main.conf
```

```
sudo cp files/bluetooth-discoverable.service
/etc/systemd/system/bluetooth-discoverable.service
sudo chmod 644 /etc/systemd/system/bluetooth-discoverable.service
sudo systemctl enable bluetooth-discoverable.service
```

3.2.4 Wi-Fi и streaming конфигурации (“wi-fi” роля)

config/roles/wi-fi/tasks/main.yml

Основната wi-fi “задача”, която извиква в себе си 2 “подзадачи”

- **name:** Icecast server related configuration
ansible.builtin.import_tasks:
 file: icecast.yml

- **name:** Liquidsoap related configuration
ansible.builtin.import_tasks:
 file: liquidsoap.yml

config/roles/wi-fi/tasks/icecast.yml

Конфигуриране и управление на **Icecast мултимедийния сървър**. Актуализиране на конфигурационния файл на **Icecast**, за да промени паролите за достъп, с цел подобряване на сигурността. Системната услуга *icecast2.service* на **Icecast** се стартира (тъй като **Icecast** я има предефинирана) и е активирана да работи при зареждане на ОС.

- **name:** Configure authentication for Icecast
ansible.builtin.replace:
 path: "{{ icecast_config_file }}"
 regexp: "{{ item-regexp }}"
 replace: "{{ item.replace }}"
loop_control:
 label: "{{ item.name }}"
loop:
 - **name:** Change source password
 regexp: <source-password>.*</source-password>
 replace: <source-password>{{ user_password }}</source-password>

```

- name: Change relay password
  regexp: "<relay-password>.*</relay-password>"
  replace: "<relay-password>{{ user_password }}</relay-password>"

- name: Change admin password
  regexp: "<admin-password>.*</admin-password>"
  replace: "<admin-password>{{ root_password }}</admin-password>"

- name: Start the Icecast service
  ansible.builtin.systemd_service:
    name: icecast2
    enabled: true

```

Алтернатива като команди:

```

sudo sed -i 's#<source-password>.*</source-password>#<source-password>{{ user_password }}</source-password>#' {{ icecast_config_file }}
sudo sed -i 's#<relay-password>.*</relay-password>#<relay-password>{{ user_password }}</relay-password>#' {{ icecast_config_file }}
sudo sed -i 's#<admin-password>.*</admin-password>#<admin-password>{{ root_password }}</admin-password>#' {{ icecast_config_file }}
sudo systemctl enable icecast2.service

```

config/roles/wi-fi/tasks/liquidsoap.yml

Създава се директория за потребителя по подразбиране за съхранение на **Liquidsoap** скриптовите програми. Необходимите конфигурации на **Liquidsoap**, включително източници на поток и идентификационни данни, се добавят към тази директория. И накрая се регистрира и активира потребителски създадена системна услуга, която да изпълнява основната **bridge.liq** скриптова програма за пренасочването на Bluetooth аудио данните идващи от **PipeWire** към **Icecast** мултимедийния сървър чрез **Liquidsoap**. Името на услугата е *liquidsoap.service* и се стартира при зареждане на ОС.

```

---  

- name: Create Liquidsoap configuration folder for default user
  ansible.builtin.file:
    path: "/home/{{ user_name }}/liquidsoap"
    state: directory
    owner: "{{ user_name }}"

- name: Add the Liquidsoap stream source file
  ansible.builtin.template:
    src: templates/bridge.liq.j2
    dest: "/home/{{ user_name }}/liquidsoap/bridge.liq"
    owner: "{{ user_name }}"

- name: Add credentials to be used by Liquidsoap
  ansible.builtin.template:
    src: templates/password.liq.j2
    dest: "/home/{{ user_name }}/liquidsoap/password.liq"
    owner: "{{ user_name }}"

- name: Register Liquidsoap bridge service
  ansible.builtin.template:
    src: templates/liquidsoap.service.j2
    dest: /etc/systemd/system/liquidsoap.service

- name: Enable Liquidsoap bridge service at boot
  ansible.builtin.systemd_service:
    name: liquidsoap
    enabled: true

```

Алтернатива като команди:

```

sudo mkdir -p /home/{{ user_name }}/liquidsoap
sudo chown {{ user_name }} /home/{{ user_name }}/liquidsoap
sudo cp templates/bridge.liq.j2 /home/{{ user_name }}/liquidsoap/bridge.liq

```

```
sudo chown {{ user_name }} /home/{{ user_name }}/liquidsoap/bridge.liq
sudo cp templates/password.liq.j2 /home/{{ user_name }}/liquidsoap/password.liq
sudo chown {{ user_name }} /home/{{ user_name }}/liquidsoap/password.liq
sudo cp templates/liquidsoap.service.j2 /etc/systemd/system/liquidsoap.service
sudo systemctl enable liquidsoap.service
```

config/roles/wi-fi/templates/bridge.liq.j2

Въпросната скриптова програма отговорна за четенето на Bluetooth аудио поточните данни, идващи от **PipeWire** (по-конкретно **pipewire-pulse**, тъй като Liquidsoap изисква **PulseAudio**) към **Icecast** на съответния порт и съответната точка за “слушане” (mountpoint):

<IP на центр. взл.>:8000/stream.mp3

```
%include "password.liq"

set("log.file",true)
set("log.file.path", "/home/{{ user_name }}/liquidsoap/bridge.log")

bluetooth_audio = input.pulseaudio()

output.icecast(
    %mp3,
    host="localhost",
    port=8000,
    password=source_password,
    mount="stream.mp3",
    bluetooth_audio
)
```

config/roles/wi-fi/templates/liquidsoap.service.j2

Услугата за *liquidsoap.service*. Тя зависи от *bluetooth-discoverable.service* и PipeWire като потребителска услуга, тоест не може да стартира действето си преди *bluetooth-discoverable.service* и PipeWire да са били вече заредени и вървящи като услуги.

```

[Unit]
Description=Liquidsoap Streamer Service
After=network.target bluetooth-discoverable.service

[Service]
Environment="PULSE_SERVER=unix:/run/user/1000/pulse/native"
Environment="PIPEWIRE_RUNTIME_DIR=/run/user/1000/pipewire"
Type=simple
ExecStart=/usr/bin/liquidsoap --debug /home/{{ user_name }}/liquidsoap/bridge.liq
Restart=always
RestartOnFailure=yes
RestartSec=5
User={{ user_name }}

[Install]
WantedBy=multi-user.target

```

3.2.5 Експротиране до .img файл

След успешното завършване на задачите за създаване на персонализирана ОС за централния възел, се изпълняват още едни допълнителни задачи (post_tasks), които да трансформират тази ОС изходен “*.img” файл.

config/tasks/export.yml

```

---
- name: Shutdown the Raspberry Pi Emulator
  ignore_unreachable: true
  ansible.builtin.command: shutdown -h now
  async: 1
  poll: 0

- name: Wait until the Emulator stops
  ignore_unreachable: true

```

```

ansible.builtin.pause:
  seconds: 10

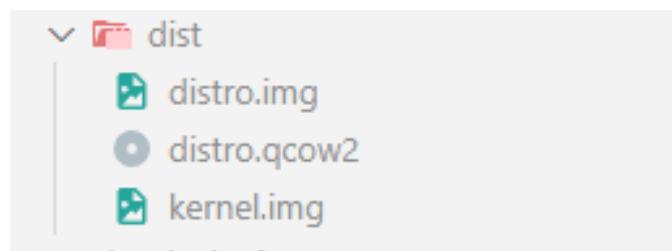
- name: Install `qemu-img` utility inside ansible container
  delegate_to: localhost

ansible.builtin.apt:
  update_cache: true
  name: qemu-utils
  state: present

- name: Use `qemu-img` to convert to a bootable disk image
  delegate_to: localhost
  ansible.builtin.command: qemu-img convert -f qcow2 -O raw /distro/distro.qcow2
/distro/distro.img
  register: convert

  retries: 10
  delay: 1
  until: convert.rc == 0

```



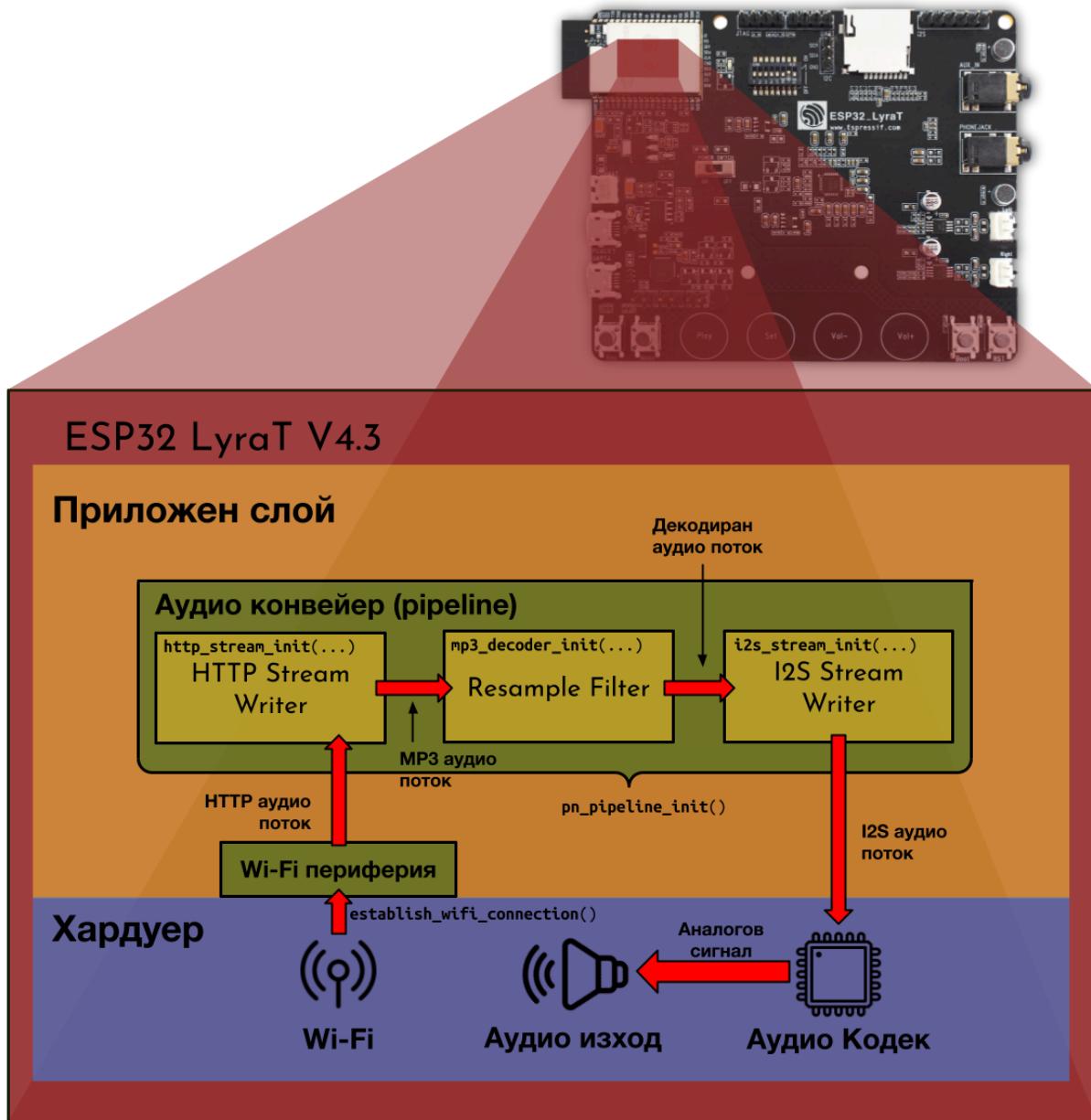
Фиг. 3.2 “dist” папка с изходния “distro.img” файл

Резултатът се съхранява в **dist** папката в основната папка на проекта и изходния “*.img” файл е запазен под името **distro.img** (Фиг. 3.2). Това е файлът, който е готов да бъде зареден върху SD карта.

3.3. Функционално описание на периферните възли

Периферните възли имат по-проста структура на изграждащите блокове спрямо на централния възел, но и тук има особености, тъй като блоковете на периферните възли се разглеждат на по-ниско (граничещо с хардуера) ниво спрямо софтуерните блокове на централния възел. Във ESP-LyraT някои от процесите се извършват паралелно.

- Инициализирането на Wi-Fi свързаността не е първата и най-важна стъпка, тя може да се извърши и на по късен етап, тъй като тя е асинхронна операция за ESP32. На практика няма как да има приемане на аудио данни без периферният възел да няма Wi-Fi свързаност в локалната мрежа.
- Осъществяването на Wi-Fi свързаност става чрез инициализация на *периферията* на ESP-WROVER-E[31] модула, която всъщност е вътре в него, отговорна за Wi-Fi
- Преди да започне каквато и да е била обработка, **ESP-ADF** задължава да бъде инициализирана т.н. аудио конвейер (audio pipeline), който може да се състои от различни по вид и характер аудио елементи. За целите на дипломната работа ще бъдат инициализирани 3 аудио елемента.
- Първият и най-важен аудио елемент е **HTTP Stream**[37] от тип **reader**, т.е. такъв който ще приема и чете прииждащите аудио поточни данни по **HTTP** на подаден **URI** адрес през Wi-Fi, който ще отговаря на точката за “слушане” създадена на централния възел от **Icecast**
- След това **HTTP Stream** предава аудио данните към вторият аудио елемент от аудио конвейера (pipeline) - **MP3 Codec**[38], от тип **decoder** чиято роля ще е да декодира данните, които преди това са били кодирани в MP3 формат от **Liquidsoap** преди да бъдат разпространени из безжичната локална мрежа
- Третото последно звено е **I2S Stream**[39] от тип **writer**, който ще “запише” декодираните вече аудио цифрови данни в ES8388[32] кодек чипа
- Кодек чипа ще ги преобразува в аналогов сигнал за възпроизвеждане подава сигнала към аудио AUX жак или изход за говорители



Фиг. 3.3 Компонентите и техните взаимосвързаности, използвани от периферните възли

3.4. Реализация на периферните възли

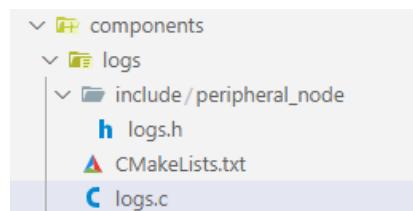
За реализацията на описаната функционалност на периферните възли върху микроконтролер ESP32-LyraT V4.3 е необходимо използването на кода, съхраняващ се в GitHub хранилището на дипломната работа[41]. Това е стандартна структура на **ESP-ADF**

проект, който се състои от множество сложи способи за “изграждане” (build), рефериране на изходния код и други градивни блокове, които поради своята обемност няма да бъдат разгледани подробно в текущата дипломна работа - само компонентите и кода, които изграждат периферния възел като такъв.

Цялото внимание ще бъде насочено към основния файл **main.c**, намиращ се в папка **main**. Но преди това е необходимо да се дадат пояснения за дефинираните компоненти в папка **components**, които ще бъдат използвани.

3.4.1 Logs потребителски компонент

В подпака **components/logs** (Фиг 3.4) се намират функциите изработени за улеснение при работа с **log** съобщения. Всички функции (Фиг 3.5.) са просто обвивка около стандартните методи на ESP (**ESP_LOGI**, **ESP_LOGE**, **ESP_LOGW**) като премахват необходимостта всеки път да се подава **TAG** на всяко съобщение, което дори в конкретния случай е предефинирано.



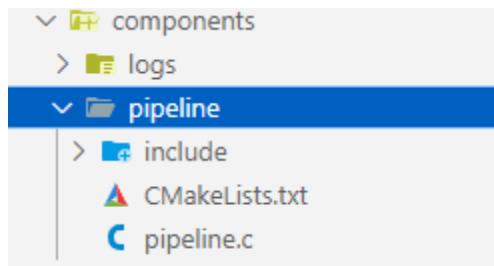
Фиг.3.4. Структура на папките спрямо “logs” компонента

```
6 static const char *TAG = "PERIPHERAL_NODE";
7
8 void logi(const char *msg, ... )
9 {
10     char buff[256];
11
12     va_list args;
13     va_start(args, msg);
14
15     vsprintf(buff, msg, args);
16
17     ESP_LOGI(TAG, "%s", buff);
18
19     va_end(args);
20 }
```

Фиг. 3.5. Примерна функция от “logs” компонента

3.4.2 Pipeline потребителски компоненти

Поради същата причина и по същия начин **pipeline** компонента е създаден по подобие на **logs** за улеснение. Те са просто обвивка около **audio_pipeline_*** функциите за работа с аудио конвейер.



Фиг.3.6. Структура на папките спрямо “pipeline” компонента

```
32 esp_err_t pn_pipeline_run()
33 {
34     return audio_pipeline_run(pipeline);
35 }
36
37 esp_err_t pn_pipeline_stop()
38 {
39     return audio_pipeline_stop(pipeline);
40 }
41
42 esp_err_t pn_pipeline_wait_for_stop()
43 {
44     return audio_pipeline_wait_for_stop(pipeline);
45 }
```

Фиг. 3.7. Функции от “pipeline” компонента

3.4.3 Необходими библиотеки

Библиотечните файлове, които ще се необходими.

```
#include <stdio.h>

// основни функции на FreeRTOS
#include "freertos/FreeRTOS.h"
```

```

#include "freertos/task.h"

// Функции за енергозависимата памет
#include "nvs_flash.h"
// Част от build процеса
#include "sdkconfig.h"

// Функции за работа с аудио елементи
#include "audio_event_iface.h"
#include "audio_element.h"
#include "audio_common.h"
#include "board.h"

// Функции за работа с периферни (Wi-Fi) модули
#include "esp_peripherals.h"
#include "periph_wifi.h"
#include "esp_wifi.h"

// Библиотеки за съответните аудио елементи,
// които ще бъдат използвани
#include "http_stream.h"
#include "i2s_stream.h"
#include "mp3_decoder.h"

// Създадените потребителски компоненти
#include "peripheral_node/logs.h"
#include "peripheral_node/pipeline.h"

```

3.4.4 Инициализиране на Wi-Fi периферия

С цел по-добра четимост, е създадена функцията `establish_wifi_connection()` чиято цел е инициализацията на Wi-Fi периферията и установяване на връзка с локалната Wi-Fi мрежа. Конфигурацията се съдържа в променливата `wifi_config`. Променливите

CONFIG_WIFI_SSID и **CONFIG_WIFI_PASSWORD** се задават от конфигурационното меню чрез **idf.py menuconfig** в секция “*Wi-Fi Connectivity Configuration*” (Фиг. 4.5).

След това функцията инициализира Wi-Fi периферното устройство с помощта на **periph_wifi_init**, като полученият манипулятор (handle) се съхранява в **wifi_handle**. След известяване, че Wi-Fi периферното устройство е вече стартирано, то използва **esp_periph_start** за активиране на Wi-Fi. Функцията изчаква за неопределено време Wi-Fi връзката с помощта на **periph_wifi_wait_for_connected**.

```
58 esp_err_t establish_wifi_connection()
59 {
60     logi("[-*] Initializing Wi-Fi peripherals");
61     esp_periph_config_t periph_config = DEFAULT_ESP_PERIPH_SET_CONFIG();
62     periph_set = esp_periph_set_init(&periph_config);
63     periph_wifi_cfg_t wifi_config = {
64         .wifi_config.sta.ssid = CONFIG_WIFI_SSID,
65         .wifi_config.sta.password = CONFIG_WIFI_PASSWORD,
66     };
67     esp_periph_handle_t wifi_handle = periph_wifi_init(&wifi_config);
68
69     logi("[-*] Starting the Wi-Fi peripheral for connection");
70     esp_periph_start(periph_set, wifi_handle);
71     periph_wifi_wait_for_connected(wifi_handle, portMAX_DELAY); You, 1
72
73     return ESP_OK;
74 }
```

Фиг. 3.8. Функцията “establish_wifi_connection” за установяване на Wi-Fi връзка

3.4.5 Инициализация на аудио конвейера (pipeline)

За тази стъпка е необходимо първо да се инициализира самият аудио конвейер (pipeline), което се извършва изключително просто тъй като се използва потребителски дефинираната **pn_pipeline_init()** от компонента **pipeline**.

```

logi("[ 2 ] Creating audio pipeline for playback");
pn_pipeline_init();

```

Фиг. 3.9 Инициализиране на аудио конвейера

След което е ред за инициализация за първият от аудио елементите - **HTTP Stream** от тип **reader**. **http_stream_cfg_t http_config** се инициализира със стойности по подразбиране с помощта на **HTTP_STREAM_CFG_DEFAULT()**. Функцията **event_handle_for_http_stream** е присвоена като манипулятор (handle) на събития произлизащи от **HTTP Stream** элемента. Функцията **event_handle_for_http_stream** (*Фиг. 3.11*) обработва различни събития за HTTP звуковия поток. Той проверява **event_id** и извършва действия въз основа на типа събитие. Ако **event_id** е **HTTP_STREAM_RESOLVE_ALL_TRACKS**, функцията връща **ESP_OK**, което означава, че не са необходими допълнителни действия. Ако събитието е тип **HTTP_STREAM_FINISH_TRACK**, то извиква **http_stream_next_track**, за да премине към следващата песен. За **HTTP_STREAM_FINISH_PLAYLIST** той извиква **http_stream_fetch_again**, за да презареди плейлиста. Ако нито едно от тези събития не се случи, функцията връща **ESP_OK**.

Потокът е настроен да действа като “четец” (**AUDIO_STREAM_READER**). Накрая, **HTTP Steam** елемента се инициализира с помощта на **http_stream_init(&http_config)**

```

113     logi("[ 3 ] Creating required audio elements for pipeline");
114     logi("[--] Initializing an HTTP stream reader to read the incoming HTTP audio");
115     http_stream_cfg_t http_config = HTTP_STREAM_CFG_DEFAULT();
116     http_config.event_handle = event_handle_for_http_stream;
117     http_config.type = AUDIO_STREAM_READER;
118     http_config.enable_playlist_parser = true;
119     http_stream_reader = http_stream_init(&http_config);

```

Фиг. 3.10 Инициализиране на HTTP Stream om mun reader

```

83     int event_handle_for_http_stream(http_stream_event_msg_t *message)
84     {
85         if (message->event_id == HTTP_STREAM_RESOLVE_ALL_TRACKS)
86         {
87             return ESP_OK;
88         }
89
90         if (message->event_id == HTTP_STREAM_FINISH_TRACK)
91         {
92             return http_stream_next_track(message->el);
93         }
94         if (message->event_id == HTTP_STREAM_FINISH_PLAYLIST)
95         {
96             return http_stream_fetch_again(message->el);
97         }
98         return ESP_OK;
99     }

```

Фиг. 3.11 Използваната функция като манипулатор (handle) на събития за `http_stream_reader`

Следващият аудио елемент, който бива инициализиран е вторият подред **MP3 Decoder**. Чрез **DEFAULT_MP3_DECODER_CONFIG()** се задава начална конфигурация на елемента, която е по подразбиране и **mp3_decoder_init(&mp3_config)** инициализира самия **MP3 Decoder** елемент.

```

121     logi("[*] Initializing an MP3 decoder to decode the incoming
● 122     mp3_decoder_cfg_t mp3_config = DEFAULT_MP3_DECODER_CONFIG();
123     mp3_decoder = mp3_decoder_init(&mp3_config);

```

Фиг. 3.12 Инициализация на MP3 Decoder

По много подобен начин се инициализира и последният от всички аудио елементи **I2S Stream** от тип **writer**. Чрез **I2S_STREAM_CFG_DEFAULT()** се задава начална конфигурация на елемента, която е по подразбиране и **i2s_stream_init(&i2s_stream_config)** инициализира самия **I2S Stream** елемент, като преди това е зададен да бъде конкретно от тип **writer** чрез **AUDIO_STREAM_WRITER** стойността.

```

125     logi("[--] Initializing an I2S stream to write data to codec chip");
126     i2s_stream_cfg_t i2s_stream_config = I2S_STREAM_CFG_DEFAULT();
127     i2s_stream_config.type = AUDIO_STREAM_WRITER;
128     i2s_stream_writer = i2s_stream_init(&i2s_stream_config);
...

```

Фиг. 3.13 Инициализация на IS2 Stream om mun writer

Следващите редове (*Фиг 3.14*) регистрират аудио компоненти на в аудио конвейера (pipeline). **pn_pipeline_register** е обвивка на функцията **audio_pipeline_register**, която извършва регистрацията. Регистрират се трите компонента с техните етикети (tags). Това позволява на аудио конвейера (pipeline) да управлява и обработва тези компоненти като част от потока. Заедно с това чрез **pn_pipeline_link(&link_tag[0], 3);** ги “свързва” една спрямо друга за да може всеки от компонентите да знае къде е позициониран в цялата аудио последователност.

```

130     logi("[ 4 ] Registering all these elements to the dedicated audio pipeline");
131     pn_pipeline_register(http_stream_reader, http_stream_tag);
132     pn_pipeline_register(mp3_decoder, mp3_decoder_tag);
133     pn_pipeline_register(i2s_stream_writer, i2s_stream_tag);
134
135     logi("[ 5 ] Linking the elements in proper order: http_stream→mp3_decoder→i2s_sti
136     const char *link_tag[3] = {http_stream_tag, mp3_decoder_tag, i2s_stream_tag};
137     pn_pipeline_link(&link_tag[0], 3);

```

Фиг. 3.14 Регистриране и свързване на аудио елементите в рамките на аудио конвейера

Функция **audio_element_set_uri** конфигурира елемента **HTTP Stream** от тип **reader** да получава аудио данни от конкретния **URI**, зададен чрез **CONFIG_STREAM_URI** конфигурационната променлива. С други думи, **audio_element_set_uri** настройва адреса на потока, който **http_stream_reader** трябва да чете.

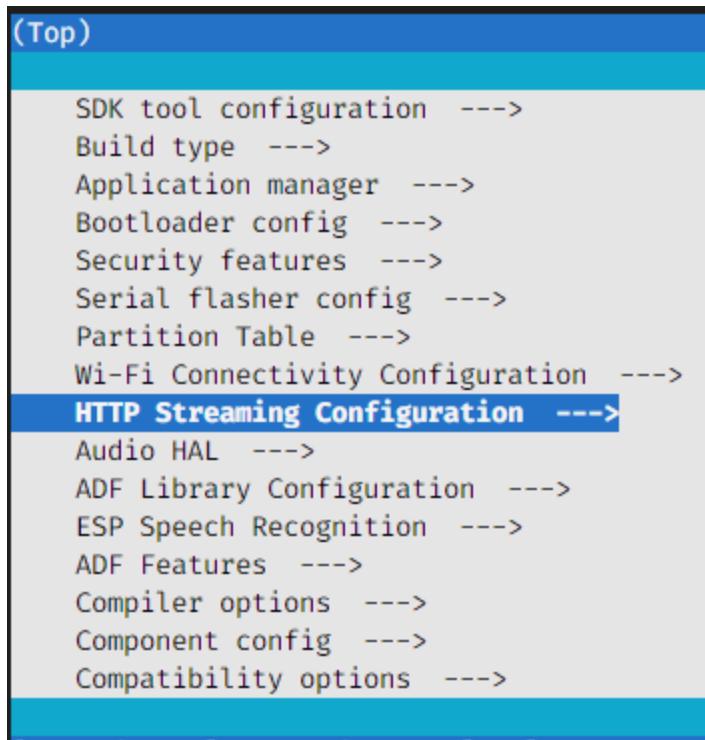
```

139     logi("[ 6 ] Setting up streaming URI for the HTTP stream reader");
140     logi("[--] HTTP stream URI - %s", CONFIG_STREAM_URI);
141     audio_element_set_uri(http_stream_reader, CONFIG_STREAM_URI);

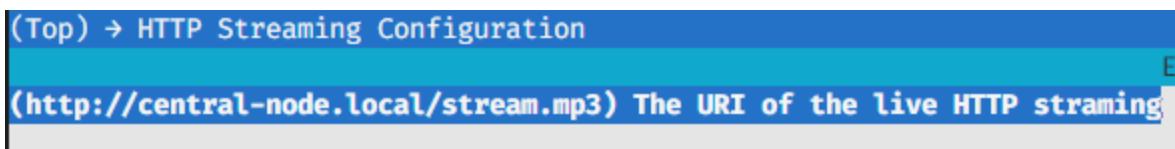
```

Фиг. 3.15 Задаване на URI за http_stream_reader

Самата конфигурационна променлива **CONFIG_STREAM_URI** може да бъде зададена потребителски чрез изпълнението на **idf.py menuconfig**, което отваря менюто за конфигурация, и в секция “*HTTP Streaming Configuration*” тя може да бъде променена (Фиг. 3.16 и Фиг. 3.17).



Фиг. 3.16 Конфигурационно меню на ESP-ADF
(*idf.py menuconfig*)



Фиг. 3.17 Конфигурационна опция за *URI*

3.4.6 Задаване на слушател на събития (event listener)

Функцията **initialize_event_listener** създава и инициализира слушател за събития (event listener). Създава се конфигурация по подразбиране за слушателя на събития чрез **AUDIO_EVENT_IFACE_DEFAULT_CFG()**, след което се инициализира чрез

audio_event_iface_init(&event_config). Резултатът се съхранява в променливата **event**. Слушателят се “закача” за аудио аудио конвейера, за да “слуша” за всякакви промени в аудио елементите. (Фиг 3.19)

```

35 void initialize_event_listener()
36 {
37     audio_event_iface_cfg_t event_config = AUDIO_EVENT_IFACE_DEFAULT_CFG();
38     event = audio_event_iface_init(&event_config);
39 }
40

```

Фиг. 3.18 Инициализиране на event listener

```

logi("[ 7 ] Initialize event listener");
initialize_event_listener();
logi(" * Catching all events from the elements inside the pipeline");
pn_pipeline_set_listener(event);

```

Фиг. 3.19 Задаване на event listener на аудио конвейера

3.4.7 Стартране на аудио конвейера

Най-просто чрез **pn_pipeline_run()**; потребителска обвивка.

```

logi("[ 9 ] Starting the audio pipeline");
pn_pipeline_run();

```

Фиг. 3.20 Стартране на аудио конвейера

След което се влиза в безкраен цикъл, който ще служи за отреагиране на всяка промяна, която регистрирания слушател на събития засече в някои от аудио елементите. В него има две условия:

1) Ако източникът на съобщението е **mp3_decoder** и типът на източника е **AUDIO_ELEMENT_TYPE_ELEMENT**, а командата е **AEL_MSG_CMD_REPORT_MUSIC_INFO**, то:

- Получава се информация за аудио елемента чрез **audio_element_getinfo**, където **sound_information** съдържа информация за аудио потока (напр. честота на дискретизация, брой битове и канали).

- Въз основа на получената информация, се настройва вътрешния I2S тактов сигнал с **i2s_stream_set_clk**, който актуализира честотата на дискретизация, броя битове и канали на **i2s_stream_writer**, за да съответства на MP3 потока.

```

164     /* If the incoming message was originated from the MP3 decoder */
165     if (message.source == (void *)mp3_decoder &&
166         message.source_type = AUDIO_ELEMENT_TYPE_ELEMENT &&
167         message.cmd = AEL_MSG_CMD_REPORT_MUSIC_INFO)
168     {
169         audio_element_info_t sound_information = {0};
170         audio_element_getinfo(mp3_decoder, &sound_information);
171
172         logi("[--] Receiving information from `mp3_decoder`:");
173         logi("[--] - sample_rates = %d", sound_information.sample_rates);
174         logi("[--] - bits = %d", sound_information.bits);
175         logi("[--] - channels = %d", sound_information.channels);
176
177         /* Setup the internal I2S clock according to decoded MP3 stream */
178         i2s_stream_set_clk(
179             i2s_stream_writer,
180             sound_information.sample_rates,
181             sound_information.bits,
182             sound_information.channels);
183
184         continue;
185     }

```

Фиг. 3.21 Разчитане на информацията за аудио потока от MP3 Decoder конфигуриране на I2S спрямо нея

- 2) Ако източникът на съобщението е **http_stream_reader**, типът на източника е **AUDIO_ELEMENT_TYPE_ELEMENT**, командата е **AEL_MSG_CMD_REPORT_STATUS**, и данните в съобщението показват статус **AEL_STATUS_ERROR_OPEN** (което означава, че е възникла грешка при отваряне):

- Спира се всички компоненти на конвейера с **pn_pipeline_stop_all()**.
- Нулират се състоянието на **mp3_decoder** и **i2s_stream_writer** с **audio_element_reset_state**.
- Рестартира се целият аудио конвейер (pipeline) с **pn_pipeline_reset_all()**, след което се стартира отново с **pn_pipeline_run()**.

```
187     /* Restart the stream if the http_stream_reader receives stop event (cau
188     if (message.source == (void *)http_stream_reader &&
189         message.source_type == AUDIO_ELEMENT_TYPE_ELEMENT &&
190         message.cmd == AEL_MSG_CMD_REPORT_STATUS &&
191         (int)message.data == AEL_STATUS_ERROR_OPEN)
192     {
193         logw("[-!] Restarting stream because of errors in http_stream");
194         pn_pipeline_stop_all();
195
196         audio_element_reset_state(mp3_decoder);
197         audio_element_reset_state(i2s_stream_writer);
198
199         pn_pipeline_reset_all();
200         pn_pipeline_run();
201         continue;
202     }
203 }
```

Фиг. 3.22 Рестартиране на аудио конвейера при грешка в HTTP Stream

Четвърта глава

Практически резултати и приложение

След като са обяснени начина и способите, чрез които е изграден работоспособния модел на текущата дипломна работа - конкретно за централния възел, конкретно за периферните възли, и взаимовръзките между тях, е ред да се прибегне до това как практически да бъде използвано и приложено.

Кодът и конфигурациите са разпределени в две отделни GitHub хранилища, едно за централния възел **central-node[40]** (съдържащ всичко необходимо за превръщането на всеки Raspberry Pi 4 в централен възел), и друго - за периферните възли **peripheral-node[41]** (съдържащ целият код, който да превърне всяко ESP-LyraT V4.3 в периферен възел). Съществува и още едно хранилище **thesis[42]**, което съдържа връзки към предходните 2 хранилища.

По лесен начин всеки може да се сдобие с тях тъй като са публично достъпни и с отворен код, дори някой може да даде собствените си подобрения върху този проект.

Единствените изисквания към използването на кода и конфигурациите от текущата дипломна работа към системата, на която ще бъдат свалени са:

- да бъдат Linux-базирана или MacOS операционна система. Ако е Windows за момента могат да бъдат използвани WSL (Windows Subsystem for Linux)[43] в комбинация с Docker Desktop (тествани).
- Да има инсталиран и конфигуриран Docker Engine[28]
- Да има в наличност инсталирани и работещи ESP-IDF и ESP-ADF среди. Ако не са налични, в документацията на ESP-ADF е описан подробно със стъпки целият инсталационен процес.[8]

Започвайки от централния възел, цялото съдържание на хранилището **central-node[40]** може бъде изтеглено чрез команда:

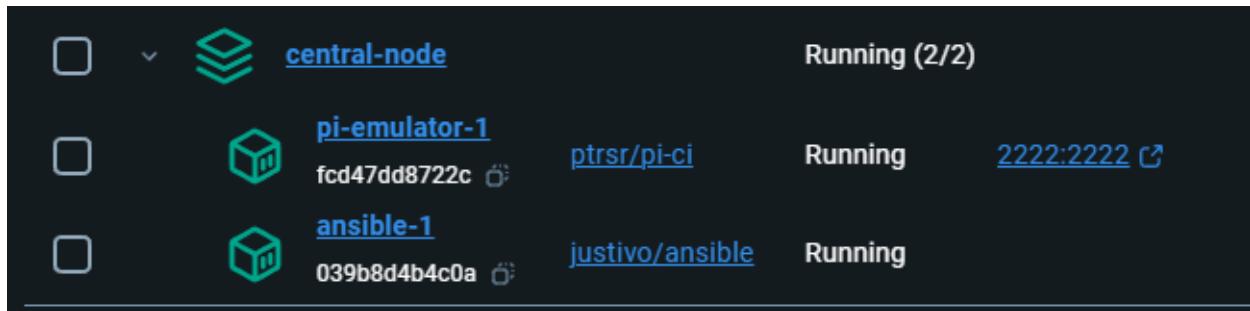
```
git pull https://github.com/Heaven-Waves/central-node.git
```

Или чрез изтегляне на архивиран **zip** файл директно от уеб интерфейса на GitHub.

След клониране / разархивиране се навигира в създадената **central-node** папка. Оттам се разкрива цялото съдържание на конфигурациите, които да бъдат направени върху Raspberry Pi OS. Единственото необходимо действие е да се стартира **setup.sh** скрипта. Това ще инициира процеса за създаването на персонализираната ОС, с необходимите конфигурации за централния възел, и ще експортира ОС до ***.img** файл.

```
$ ./setup.sh
[+] Building 0.2s (10/10) FINISHED
  ⇒ [internal] load build definition from Dockerfile
  ⇒ → transferring dockerfile: 560B
  ⇒ [internal] load metadata for docker.io/library/debian:bookworm-slim
  ⇒ [internal] load .dockignore
  ⇒ → transferring context: 78B
  ⇒ [1/6] FROM docker.io/library/debian:bookworm-slim
  ⇒ → CACHED [2/6] RUN apt-get update && apt-get install -y    iputils-ping    openssh-client    sshpass    ca-
  ⇒ → CACHED [3/6] RUN rm -rf /var/lib/apt/lists/*
  ⇒ → CACHED [4/6] RUN pipx install --include-deps ansible-core
  ⇒ → CACHED [5/6] RUN mkdir -p /etc/ansible/ && echo "[local]\nlocalhost" > /etc/ansible/hosts
  ⇒ → CACHED [6/6] WORKDIR /etc/ansible/playbooks
  ⇒ → exporting to image
  ⇒ → exporting layers
  ⇒ → writing image sha256:af1c0081777d742e429c7e207b5aacfcd53a68753c729d1d06cc4a39d83c94a
  ⇒ → naming to docker.io/justivo/ansible
[+] Running 2/2
✓ Container central-node-pi-emulator-1 Started
✓ Container central-node-ansible-1 Started
```

Фиг. 4.1 Съобщения на стандартния изход при стартирането на **setup.sh** скрипта



Фиг. 4.2 Визуализация на стартирани контейнери през Docker Desktop

По-конкретно:

- 1) Създаване Docker изображение от показания т. 3.2.1 **Dockerfile**, който ще послужи за съхранението и изпълнението на **Ansible** конфигурациите. По подразбиране името на изображението е **justivo/ansible** то ще бъде изградено до дефинираният контейнер в т.3.2.1 **ansible**.
- 2) Изграждане (build) публично достъпното Docker изображение **ptrsr/pi-ci**[44] до Docker контейнер (наречен по-рано в т.3.2.1 **pi-emulator**), което е емулатора, който ще бъде използван за създаването на персонализирана операционна система + да

изгради вече създаденото изображение **justivo/ansible** (т.е в тази стъпка се изпълнява Docker конфигурацията в **compose.yml** файла от т. 3.2.1)

- 3) След като контейнерите биват изградени, те се стартират автоматично **pi-emulator** стартира емулатора и зарежда операционната система, а **ansible** изчаква емулираната операционна система да зареди напълно за да се свърже отдалечно към нея с **ssh** за да приложи различните конфигурации обяснени в т.3.2.
- 4) След като **ansible** контейнера завърши своето изпълнение и приложи всички промени се изпълняват допълнителните задачи, пояснени в т. 3.2.5, които спират емулатора и експортират персонализираната операционна система в *.img файл

След всички тези стъпки **distro.img** файла в папка **dist** (т. 3.2.5) може да бъде зареден директно върху SD карта.

Лог съобщенията от **ansible** контейнера трябва да имат следния вид:

```
2024-09-10 22:31:53 PLAY [Creating custom Raspberry Pi OS image] ****
2024-09-10 22:31:53
2024-09-10 22:31:53 TASK [Wait for emulator startup] ****
2024-09-10 22:34:02 ok: [raspberrypi]
2024-09-10 22:34:02
2024-09-10 22:34:02 TASK [setup : Add bullseye-backports repository to sources list in order to
install pipewire] ***
2024-09-10 22:34:24 ok: [raspberrypi]
2024-09-10 22:34:24
2024-09-10 22:34:24 TASK [setup : Install PipeWire and all its dependancies from bullseye-backports]
***
2024-09-10 22:35:26 ok: [raspberrypi]
2024-09-10 22:35:26
2024-09-10 22:35:26 TASK [setup : Update apt] ****
2024-09-10 22:36:55 ok: [raspberrypi]
2024-09-10 22:36:55
2024-09-10 22:36:55 TASK [setup : Prioritize liquidsoap-related packages from Debian repos] ****
2024-09-10 22:37:07 ok: [raspberrypi]
2024-09-10 22:37:07
2024-09-10 22:37:07 TASK [setup : Install required packaages] ****
2024-09-10 22:37:29 ok: [raspberrypi]
2024-09-10 22:37:29
2024-09-10 22:37:29 TASK [setup : Change default boot mount point] ****
```

```
2024-09-10 22:37:35 changed: [raspberrypi]
2024-09-10 22:37:35
2024-09-10 22:37:35 TASK [setup : Change default keyboard layout to US] ****
2024-09-10 22:37:40 ok: [raspberrypi]
2024-09-10 22:37:40
2024-09-10 22:37:40 TASK [setup : Automatically connect to SSID on boot] ****
2024-09-10 22:37:44 ok: [raspberrypi]
2024-09-10 22:37:44
2024-09-10 22:37:44 TASK [setup : Remove autologin] ****
2024-09-10 22:37:49 changed: [raspberrypi]
2024-09-10 22:37:49
2024-09-10 22:34:24 [WARNING]: Platform linux on host raspberrypi is using the discovered Python
2024-09-10 22:34:24 interpreter at /usr/bin/python3.11, but future installation of another Python
2024-09-10 22:34:24 interpreter could change the meaning of that path. See
2024-09-10 22:34:24 https://docs.ansible.com/ansible-
2024-09-10 22:34:24 core/2.17/reference_appendices/interpreter_discovery.html for more information.
2024-09-10 22:37:49 TASK [setup : Remove ssh auto login] ****
2024-09-10 22:37:54 changed: [raspberrypi]
2024-09-10 22:37:54
2024-09-10 22:37:54 TASK [setup : Enable getty service for /dev/tty1] ****
2024-09-10 22:38:04 ok: [raspberrypi]
2024-09-10 22:38:04
2024-09-10 22:38:04 TASK [setup : Enable default /dev/tty1 login prompt] ****
2024-09-10 22:38:10 ok: [raspberrypi]
2024-09-10 22:38:10
2024-09-10 22:38:10 TASK [setup : Create default user `thesis`] ****
2024-09-10 22:38:18 changed: [raspberrypi]
2024-09-10 22:38:18
2024-09-10 22:38:18 TASK [setup : Set root password] ****
2024-09-10 22:38:25 changed: [raspberrypi]
2024-09-10 22:38:25
2024-09-10 22:38:25 TASK [bluetooth : Set default Bluetooth settings to act as speaker] ****
2024-09-10 22:38:30 changed: [raspberrypi]
2024-09-10 22:38:30
2024-09-10 22:38:30 TASK [bluetooth : Register Bluetooth discovery service] ****
2024-09-10 22:38:42 ok: [raspberrypi]
2024-09-10 22:38:42
2024-09-10 22:38:42 TASK [bluetooth : Enable Bluetooth discovery service on boot] ****
2024-09-10 22:38:53 ok: [raspberrypi]
2024-09-10 22:38:53
2024-09-10 22:38:53 TASK [wi-fi : Configure authentication for Icecast] ****
2024-09-10 22:39:05 ok: [raspberrypi] => (item=Change source password)
```

```

2024-09-10 22:39:05 ok: [raspberrypi] => (item=Change relay password)
2024-09-10 22:39:05 ok: [raspberrypi] => (item=Change admin password)
2024-09-10 22:39:05
2024-09-10 22:39:05 TASK [wi-fi : Start the Icecast service] ****
2024-09-10 22:39:14 ok: [raspberrypi]
2024-09-10 22:39:14
2024-09-10 22:39:14 TASK [wi-fi : Create Liquidsoap configuration folder for default user] ****
2024-09-10 22:39:19 ok: [raspberrypi]
2024-09-10 22:39:19
2024-09-10 22:39:19 TASK [wi-fi : Add the Liquidsoap stream source file] ****
2024-09-10 22:39:31 ok: [raspberrypi]
2024-09-10 22:39:31
2024-09-10 22:39:31 TASK [wi-fi : Add credentials to be used by Liquidsoap] ****
2024-09-10 22:39:41 ok: [raspberrypi]
2024-09-10 22:39:41
2024-09-10 22:39:41 TASK [wi-fi : Register Liquidsoap bridge service] ****
2024-09-10 22:39:51 ok: [raspberrypi]
2024-09-10 22:39:51
2024-09-10 22:39:51 TASK [wi-fi : Enable Liquidsoap bridge service at boot] ****
2024-09-10 22:39:59 ok: [raspberrypi]
2024-09-10 22:39:59
2024-09-10 22:39:59 TASK [Export the emulated image to *.img format] ****
2024-09-10 22:40:00 included: /etc/ansible/playbooks/tasks/export.yml for raspberrypi
2024-09-10 22:40:00
2024-09-10 22:40:00 TASK [Shutdown the Raspberry Pi Emulator] ****
2024-09-10 22:40:08 changed: [raspberrypi]
2024-09-10 22:40:08
2024-09-10 22:40:08 TASK [Wait until the Emulator stops] ****
2024-09-10 22:40:18 Pausing for 10 seconds
2024-09-10 22:40:18 ok: [raspberrypi]
2024-09-10 22:40:18
2024-09-10 22:40:18 TASK [Install `qemu-img` utility inside ansible container] ****
2024-09-10 22:40:22 ok: [raspberrypi -> localhost]
2024-09-10 22:40:22
2024-09-10 22:40:22 TASK [Use `qemu-img` to convert to a bootable disk image] ****
2024-09-10 22:40:49 changed: [raspberrypi -> localhost]
2024-09-10 22:40:49
2024-09-10 22:40:49 PLAY RECAP ****
2024-09-10 22:40:49 raspberrypi : ok=30    changed=8      unreachable=0      failed=0
skipped=0    rescued=0    ignored=0
2024-09-10 22:40:49

```

Приложена снимка на Raspberry Pi 4 като централен възел, който е използван в текущата дипломна работа се намира на *Фиг 4.3:*



Фиг. 4.3. Raspberry Pi 4 централният възел, използван в дипломната работа

Кодът, който трябва да бъде изпълнен на периферните възли, подобно на централния възел, може да бъде изтеглен директно от **peripheral-node[41]** GitHub хранилището чрез команда:

```
git pull https://github.com/Heaven-Waves/peripheral-node.git
```

Или чрез изтегляне на архивиран **zip** файл директно от уеб интерфейса на GitHub.

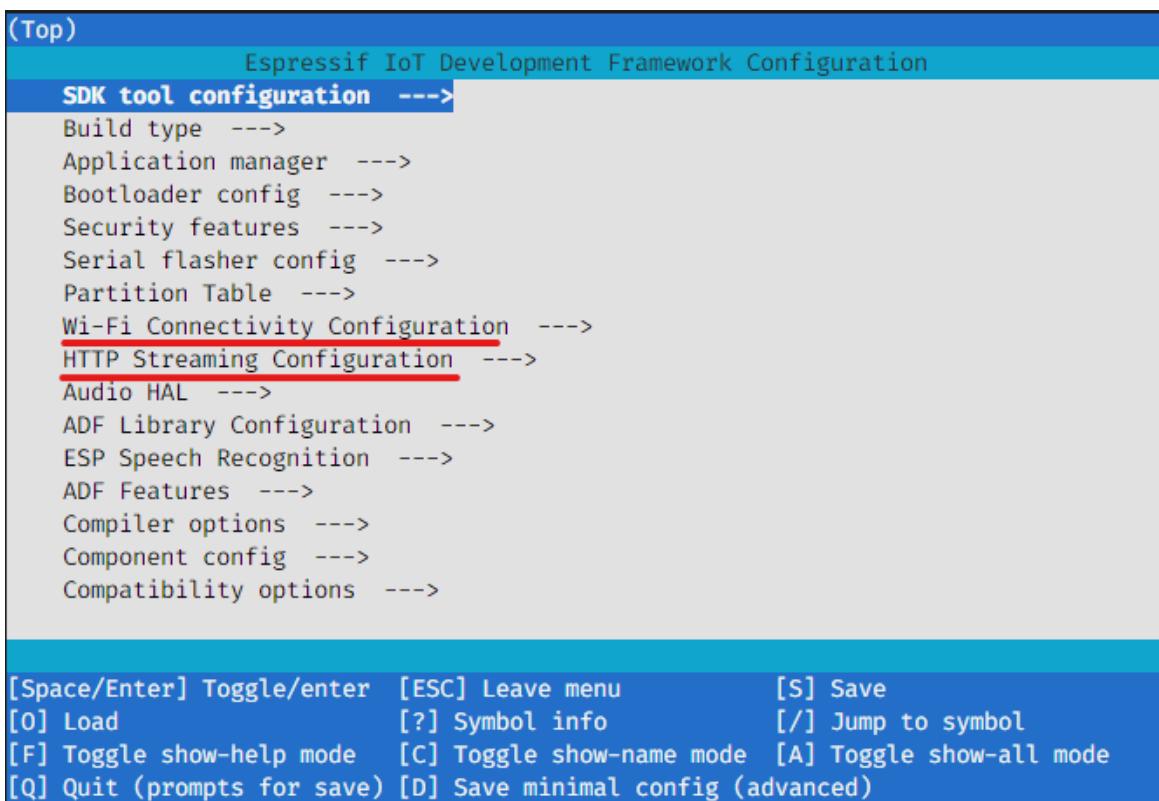
За него ще бъдат необходими ESP-ADF и ESP-IDF да бъдат налични на системата, където ще бъде изтеглен кода (за в бъдеще и това може да бъде поместено в Docker контейнери). Начина по който трябва да бъдат изтеглени може да се намери в тяхната документация[8].

Местоположението се сменя да бъде в папка **peripheral-node**, където в нея се изпълнява команда, която да извади конфигурационното меню за текущия проект:

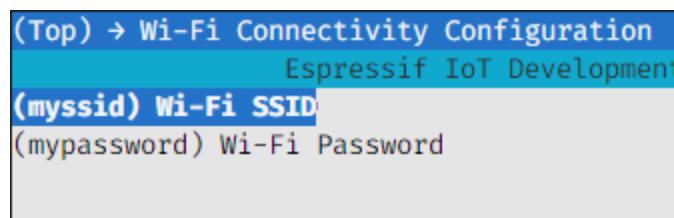
`idf.py menuconfig`

Необходимите секции, на които трябва да се обърне внимание са (*Фиг 4.4*):

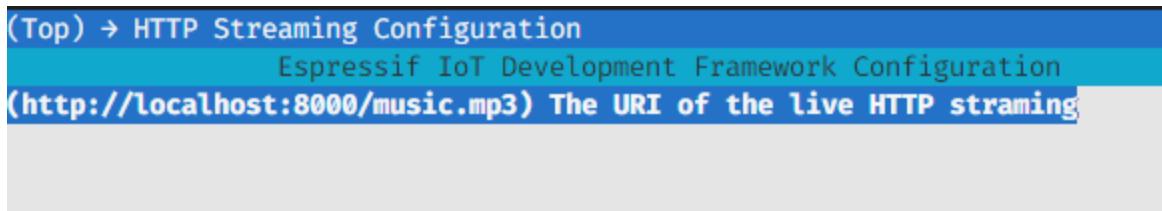
- “**Wi-Fi Connectivity Configuration**”, (*Фиг 4.5*) където се конфигурират:
 - *WIFI_SSID* - името (SSID) на локалната Wi-Fi мрежа
 - *WIFI_PASSWORD* - паролта за нея
- “**HTTP Streaming Configuration**”, (*Фиг 4.6*) където се конфигурира:
 - *STREAM_URI* - URI (mountpoint на Icecast) към който



Фиг. 4.4 Необходимите секции за ръчна конфигурация от menuconfig



Фиг. 4.5 Секция “Wi-Fi Connectivity Configuration”



Фиг. 4.6 Секция “HTTP Streaming Configuration”

След попълването на правилните данни в тези секции и изпълнението на следните команди:

```
idf.py build  
idf.py -p PORT flash monitor
```

където **PORT** е името на серийния port, на който е свързан за UART извода на микроконтролера, трябва да има знак на конзолата, че кодът е готов да бъде зареден върху ESP32-LyraT V4.3.

След успешно завършена фаза на зареждане (build step), лог съобщенията от серийния port към стандартния изход трябва да имат вид подобен на този, показан на *Фиг. 4.7* и *Фиг. 4.8*.

Приложена снимка на един от периферните възли съставен от ESP32-LyraT 4.3 и свързана към него колонка чрез аудио AUX жак комуникация, който е използван в текущата дипломна работа се намира на *Фиг. 4.9*.

Цялостната картина на всички възли, работещи според изискванията дефинирани в т. 2.4 за централния възел и т. 2.7 за периферните възли, и работещи безжично посредством локалната мрежа е показва на *Фиг 4.10*.

След като вече системата е изправна и е в готовност за използване, остава единствено да бъде добавено мобилното потребителско устройство за да може домашната озвучителна система да придобие цялостен характер (*Фиг 4.11* и *Фиг 4.12*).

```

I (1147) AUDIO_HAL: Codec mode is 2, Ctrl:1
I (1157) PERIPHERAL_NODE: [ 2 ] Creating audio pipeline for playback
I (1157) PERIPHERAL_NODE: [ 3 ] Creating required audio elements for pipeline
I (1167) PERIPHERAL_NODE: [-*-] Initializing an HTTP stream reader to read the incoming
I (1177) PERIPHERAL_NODE: [-*-] Initializing an MP3 decoder to decode the incoming HTT
I (1187) MP3_DECODER: MP3 initon = {0};
I (1187) PERIPHERAL_NODE: [-*-] Initializing an I2S stream to write data to codec chip
I (1197) I2S: APPLL expected frequency is 22579200 Hz, real frequency is 22579193 Hz
I (1207) I2S: DMA Malloc info, datalen=blocksize=1200, dma_buf_count=3
I (1217) I2S: DMA Malloc info, datalen=blocksize=1200, dma_buf_count=3 .
I (1217) I2S: I2S0, MCLK output by GPIO0
I (1227) PERIPHERAL_NODE: [ 4 ] Registering all these elements to the dedicated audio
I (1237) PERIPHERAL_NODE: [ 5 ] Linking the elements in proper order: http_stream→mp
ip] - channels = %d", sound_information.channels);
I (1247) AUDIO_PIPELINE: link el→rb, el:0x3f800c20, tag:http, rb:0x3f8010b4
I (1257) AUDIO_PIPELINE: link el→rb, el:0x3f800da4, tag:mp3, rb:0x3f8060fc
I (1267) PERIPHERAL_NODE: [ 6 ] Setting up streaming URI for the HTTP stream reader
I (1267) PERIPHERAL_NODE: [-*-] HTTP stream URI - http://192.168.100.171:8000/stream.m
I (1277) PERIPHERAL_NODE: [ 7 ] Initialize event listener
I (1287) PERIPHERAL_NODE: * Catching all events from the elements inside the pipeline
I (1297) PERIPHERAL_NODE: [ 8 ] Establishing for Wi-Fi connection (Initializing periph
I (1307) PERIPHERAL_NODE: [-*-] Initializing Wi-Fi peripherals
E (1307) gpio: gpio_install_isr_service(449): GPIO isr service already installed
I (1317) PERIPHERAL_NODE: [-*-] Starting the Wi-Fi peripheral for connection
I (1327) AUDIO_THREAD: The esp_periph task allocate stack on internal memory
I (1337) wifi:wifi driver task: 3ffcac88, prio:23, stack:6656, core=0
I (1337) system_api: Base MAC address is not set

```

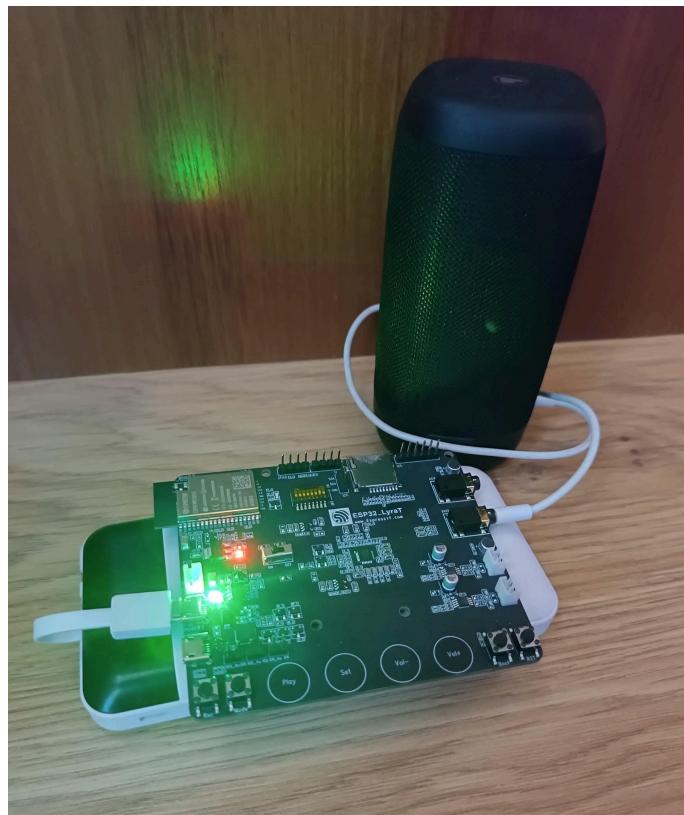
Фиг. 4.7 Лог съобщения от от периферния възел 1/2

```

I (8777) AUDIO_ELEMENT: [http] AEL_MSG_CMD_RESUME,state:1
I (8787) AUDIO_ELEMENT: [mp3] AEL_MSG_CMD_RESUME,state:1
I (8787) MP3_DECODER: MP3 opened
I (8797) AUDIO_ELEMENT: [i2s] AEL_MSG_CMD_RESUME,state:1
I (8797) I2S_STREAM: AUDIO_STREAM_WRITER
I (8807) wifi:<ba-add>idx:1 (ifx:0, 9e:e9:31:73:3e:2f), tid:0, ssn:0, winSize:64
I (8817) AUDIO_PIPELINE: Pipeline started
I (8997) HTTP_STREAM: total_bytes=0
I (9167) CODEC_ELEMENT_HELPER: The element is 0x3f800da4. The reserve data 2 is 0x0.
I (9187) PERIPHERAL_NODE: [-*-] Receiving information from `mp3_decoder`:
I (9187) PERIPHERAL_NODE: [-*-] - sample_rates = 44100
I (9197) PERIPHERAL_NODE: [-*-] - bits = 16
I (9197) PERIPHERAL_NODE: [-*-] - channels = 2
I (9227) AUDIO_ELEMENT: [i2s] AEL_MSG_CMD_PAUSE
I (9227) I2S: APPLL expected frequency is 22579200 Hz, real frequency is 22579193 Hz
I (9227) AUDIO_ELEMENT: [i2s] AEL_MSG_CMD_RESUME,state:4
I (9237) I2S_STREAM: AUDIO_STREAM_WRITER

```

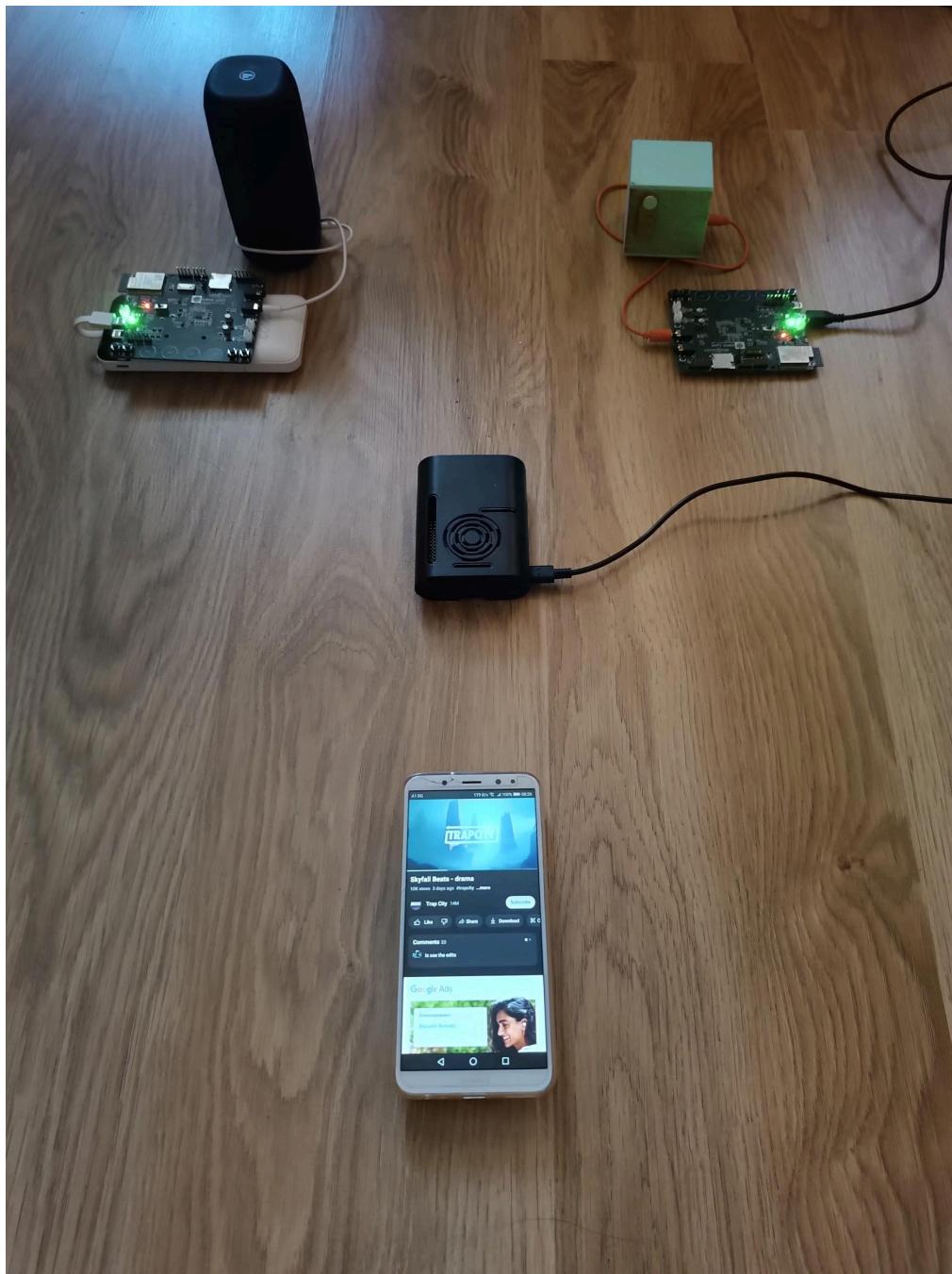
Фиг. 4.8 Лог съобщения от от периферния възел 2/2



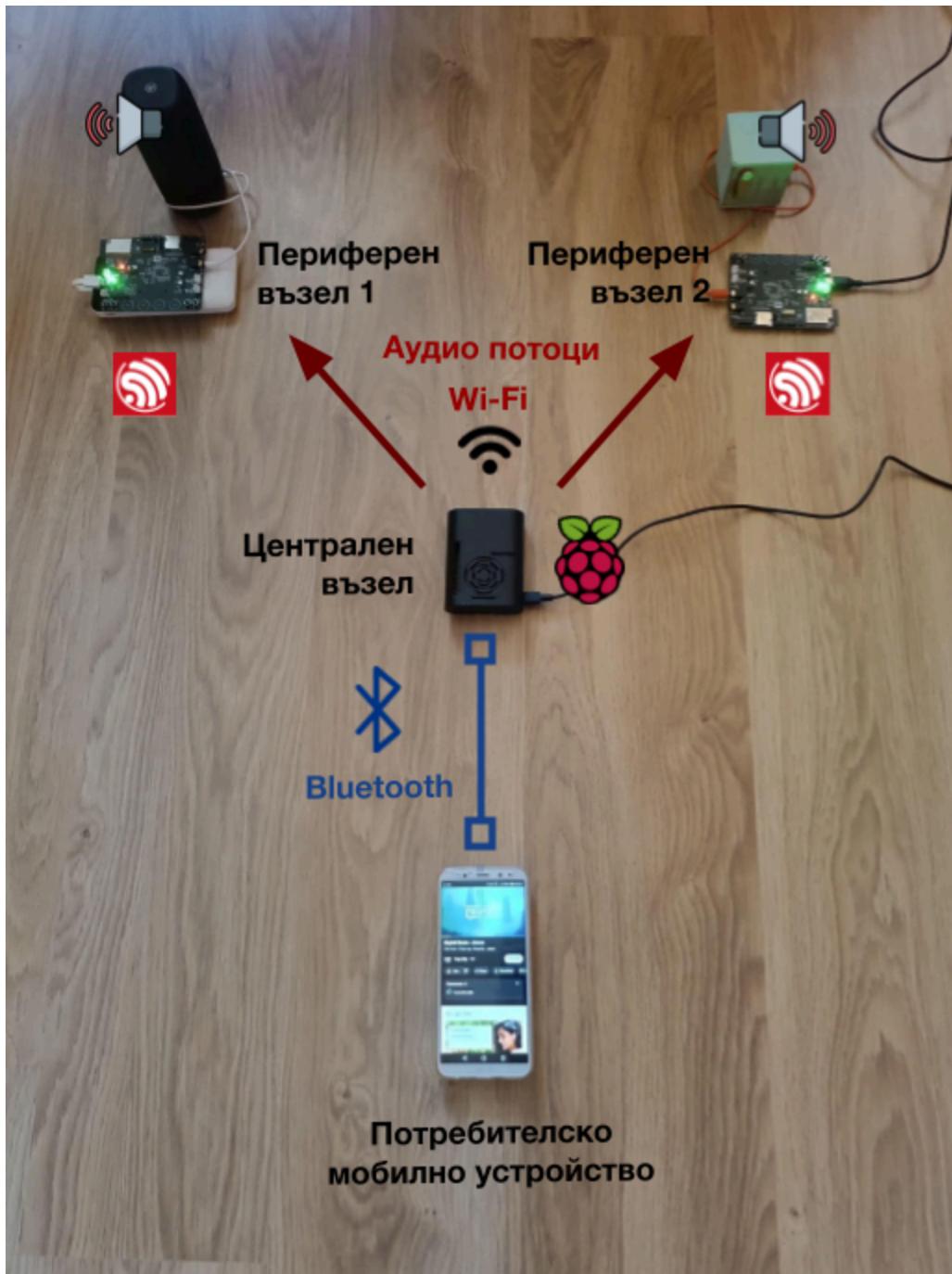
Фиг. 4.9 Един от ESP32-LyraT V4.3 периферните възли използвани в дипломната работа



Фиг. 4.10 Централният възел и 2 периферни възли



Фиг. 4.11 Реализиран модел да домашнашната озвучителна система, спрямо изискванията на дипломната работа



Фиг. 4.12 Реален модел на дипломната работа с разграфени пътищата на аудио поточните данни и обозначения за съответните възли

В домашни условия настоящата дипломна работа бива изпробвана при различни условия, едно от които е раздалечаването на възлите един от друг в рамките на жилищните

помещения. Примерно местоположение на всеки от възлите в апартамент е дадено на *Фиг.*

4.13



Фиг. 4.13 Примерно разпределение на възлите из помещенията в даден апартамент

Наблюдава се:

- Увеличаването на разстоянието между възлите води до увеличаване на закъснението отчетено от потребителското мобилно устройство до възпроизведения звук при периферните възли (расте с повече от 5 секунди)
- Увеличаването на шума в безжичната среда увеличава и възможността за загуба на пакети

Заключение

В настоящата дипломна работа се разглежда един от начините, чрез които е възможно да бъде реализирана една централизирана домашна озвучителна система, която да използва безжичната локална Wi-Fi мрежа като среда за разпространение на аудио поточните данни. Този начин се състои от подбрани както софтуерни (**Linux**-базирана операционна система с вградена **ALSA** звукова архитектура и **BlueZ** протоколен стек за поддръжка на *Bluetooth*, **PipeWire** мултимедийна система за управление и пренасочване на аудио потоци, **Liquidsoap** и **Icecast** за разпространението на и излъчването на аудио потоците в онлайн среда), протоколни (**A2DP** профил, отговорен за без проблемния обмен на аудио данни между *Bluetooth* устройства, **HTTP** протокола, чрез който се пренасят аудио данните и се осъществява разпространението на аудио потоци в реално време (*streaming*), **I2S** протокола за комуникация между съответните хардуерни аудио модули и аудио кодек (АЦП + ЦАП) чип) така и хардуерни компоненти (**Raspberry Pi 4** - със своята ARMv8 архитектура, мощен двуядрен 1.8GHz процесор, подходящ за сложните изчисления на централния възел, **ESP32-LyraT V4.3** микроконтролери като периферни възли с вградените Wi-Fi модул за приемане на аудио поточните данни и аудио кодек чип, който да преобразува получаваните аудио поточни данни), които могат да бъдат използвани като шаблон за бъдещ краен продукт или проект за автоматизация в домашна среда.

Като за бъдещото развитие на този проект могат да се разгледат начините за намаляване на закъснението на звука между потребителското устройство (източника) и периферните възли (крайната точка на възпроизвеждане) като се използват различни начини за буфериране на данните, увеличаване честота на дискретизация, намаляване шума в безжичната мрежа или др. Също така е необходимо намирането на по-сигурен способ за синхронизацията на звука между отделните периферни възли за да се намалят до максимална степен разликите при възпроизвеждането. Могат да бъдат изработени и персонализирани платки или интегрални схеми със специално предназначение (ASIC) за съответните възли с цел повишаване на ефективността, производителността и надеждността.

В тази дипломна работа бе проведено задълбочено проучване във връзка с аудиото в Linux-базираните системи (разбиране на **ALSA** звуковата архитектура, **BlueZ**, как работят **системните услуги**, особености на Linux върху **ARM** архитектура, решения за разпространение на аудио в реално време като **Liquidsoap** и **Icecast**), автоматизирани инструменти (до голяма степен **Ansible** и **Docker**), средите за разработка на фамилията микроконтролери ESP32 (**ESP-IDF** и **ESP-ADF**), което доведе до получените резултати.

Литературни източници

1. **Svetlik, Joe.** Последно актуализиран 30 Юли 2024. "Best multi-room systems 2024: all tested by expert reviewers.
<<https://www.whathifi.com/best-buys/streaming/best-multi-room-systems>>
2. **Gorman, Ben.** 23 Февруари 2023. "TCP vs UDP: Differences between the protocols".
<<https://www.avast.com/c-tcp-vs-udp-difference>>
3. **Vocal.** "Real-Time Transport Protocol (RTP)".
<<https://www.vocal.com/voip/rtp/>>
4. **Vocal.** "Real-Time Streaming Protocol (RTSP)".
<<https://www.vocal.com/v2oip/rtsp/>>
5. **Wikipedia.** Последно актуализиран 4 Април 2024. "Real Time Streaming Protocol".
<https://en.wikipedia.org/wiki/Real-Time_Streaming_Protocol>
6. **Wikipedia.** Последно актуализиран 16 Юли 2024. "Real Time Streaming Protocol".
<https://en.wikipedia.org/wiki/Network_socket>
7. **ESP-ADF Документация.** Последно актуализиран 16 Януари 2024. "ESP32-LyraT V4.3 Getting Started Guide".
<<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/design-guide/dev-boards/get-started-esp32-lyrat.html>>
8. **ESP-ADF Документация.** Последно актуализиран 26 Януари 2024. "ESP32-LyraT V4.3 Hardware Reference"
<<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/design-guide/dev-boards/board-esp32-lyrat-v4.3.html>>

9. **ESP-ADF Документация.** Начална страница.
[<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/index.html>](https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/index.html)
10. **ESP-IDF Документация.** Начална страница.
[<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html>](https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html)
11. **Raspberry Pi (Trading) Ltd.** Юни 2019. “Raspberry Pi 4 Model B” каталожна информация.
[<https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf>](https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf)
12. **Kysela, Bagnara, Iwai, van de Pol.** “ALSA project - the C library reference”.
[<https://www.alsa-project.org/alsa-doc/alsa-lib/index.html>](https://www.alsa-project.org/alsa-doc/alsa-lib/index.html)
13. **Holtmann, Matthias Hahn, Noring.** “bluetoothd(8) - Linux man page”.
[<https://linux.die.net/man/8/bluetoothd>](https://linux.die.net/man/8/bluetoothd)
14. **ArchLinux Wiki.** Последно актуализиран 9 Юни 2024. “Bluetooth”.
[<https://wiki.archlinux.org/index.php/bluetooth>](https://wiki.archlinux.org/index.php/bluetooth)
15. **Morrison, Graham.** 28 Април 2010. “Linux audio explained”.
[<https://www.techradar.com/news/audio/linux-audio-explained-685419>](https://www.techradar.com/news/audio/linux-audio-explained-685419)
16. **Morrison, Graham.** 28 Април 2010. “Linux audio explained: Part 2”.
[<https://www.techradar.com/news/audio/linux-audio-explained-685419/2>](https://www.techradar.com/news/audio/linux-audio-explained-685419/2)
17. **Уебсайт на BlueZ.** Начална страница.

<<https://www.bluez.org/>>

18. **PubNub публикация.** “What is HTTP Streaming?”

<<https://www.pubnub.com/guides/http-streaming/>>

19. **Wikipedia.** “Advanced Audio Distribution Profile (A2DP)”

<[https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles#Advanced_Audio_Distribution_Profile_\(A2DP\)](https://en.wikipedia.org/wiki/List_of_Bluetooth_profiles#Advanced_Audio_Distribution_Profile_(A2DP))>

20. **Freedesktop PulseAudio.** Начална страница

<<https://www.freedesktop.org/wiki/Software/PulseAudio/#:~:text=PulseAudio%20is%20a%20sound%20server,mobile%20devices%2C%20by%20multiple%20vendors.>>

21. **ArchWiki.** “PipeWire”

<<https://wiki.archlinux.org/title/PipeWire>>

22. **Документация на PipeWire.** “pipewire-pulse”

<https://docs.pipewire.org/page_man_pipewire-pulse_1.html>

23. **Raspberry Pi Organisation.** Страница за сваляне на Raspberry Pi OS.

<<https://www.raspberrypi.com/software/operating-systems/>>

24. **Raspberry Pi Organisation.** Raspberry Pi OS Lite Bookworm 2024-07-04.

<https://downloads.raspberrypi.com/raspios_lite_arm64/images/raspios_lite_arm64-2024-07-04/>

25. **Savonet Liquidsoap.** Начална страница на документация на Liqiodsoap.

<<https://www.liquidsoap.info/doc-2.1.3/>>

26. **Savonet Liquidsoap**. Начална страница на Liqiodsoap.

<<https://www.liquidsoap.info/>>

27. **Xiph.Org Icecast**. Начална страница на Icecast.

<<https://www.icecast.org/>>

28. **Docker Curriculum - Srivastav, Prakhar**. “What is Docker?”

<<https://docker-curriculum.com/>>

29. **Balena Etcher**. Начална страница.

<<https://etcher.balena.io/>>

30. **Red Hat Ansible**. “Red Hat Ansible Automation Platform”

<<https://www.redhat.com/en/technologies/management/ansible>>

31. **Espressif Systems**. Каталожна информация за ESP32-WROVER-E и
ESP32-WROVER-IE

<https://www.espressif.com/sites/default/files/documentation/esp32-wrover-e_esp32-wrover-ie_datasheet_en.pdf>

32. **Everest Semiconductors**. Каталожна информация за ES8388.

<<http://www.everest-semi.com/pdf/ES8388%20DS.pdf>>

33. **FreeRTOS**. Начална страница.

<<https://www.freertos.org/>>

34. **Wikipedia**. I²S

<<https://en.wikipedia.org/wiki/I%C2%B2S>>

35. **Wikipedia.** I²C

<<https://en.wikipedia.org/wiki/I%C2%B2C>>

36. **Debian.** Debian хранилищата

<<https://ftp.debian.org/debian/dists/bookworm/>>

37. **ESP-ADF Документация.** HTTP Stream

<<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/api-reference/streams/index.html#http-stream>>

38. **ESP-ADF Документация.** MP3 Decoder.

<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/api-reference/decoders/mp3_decoder.html>

39. **ESP-ADF Документация.** I2S Stream

<<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/api-reference/streams/index.html#i2s-stream>>

40. GitHub хранилище на код и конфигурации за централен възел.

<<https://github.com/Heaven-Waves/central-node>>

41. GitHub хранилище на код и конфигурации за периферните възли.

<<https://github.com/Heaven-Waves/peripheral-node>>

42. GitHub хранилище на предходните две.

<<https://github.com/Heaven-Waves/thesis>>

43. **Microsoft.** “How to install Linux on Windows with WSL”

<<https://learn.microsoft.com/en-us/windows/wsl/install>>

44. **Peters, Ruud.**Github хранилище на проекта **pi-ci**.

<<https://github.com/ptrsr/pi-ci/>>

45. **ESP-ADF Документация.** “Development Boards¶”

<<https://espressif-docs.readthedocs-hosted.com/projects/esp-adf/en/latest/design-guide/dev-boards/index.html>>