МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.

«___» _____2022 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО–КВАЛІФІКАЦІЙНОГО РІВНЯ
"БАКАЛАВР"

Тема: Програмний модуль зчитування інформації з SDR-приймача

Виконавець: Ковальов Микита Анатолійович

Керівник: Кучеров Дмитро Павлович

Нормоконтролер:

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітньо-кваліфікаційний рівень бакалавр

Напрям (спеціальність) 123 "Комп'ютерна інженерія

(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

Литвиненко О.Є.

«___» _____2022 р.

ЗАВДАННЯ

на виконання дипломної роботи (проекту)

Ковальова Микити Анатолійовича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема проекту (роботи): Програмний модуль зчитування інформації з SDR-приймача

затверджена наказом ректора від " 15 " лютого 2022 року №251/ст

2. Термін виконання проекту (роботи): з 16.05.22 до 19.06.22

3. Вихідні дані до проекту (роботи): середовище розроблення Visual Studio Code, GNU Radio, MS Word інформація про програмне-кероване радіо.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) Аналіз доцільності;

2) Проектування;

3) розроблення;

5. Перелік обов’язкового графічного матеріалу:

1) Діаграми Use Case;

2) Екранна форма інтерефейсу;

3) Діаграма послідовності;

4) Тест-кейси.

6. Календарний план

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі дипломного проектування	13.05.22	
2	Вивчити спеціальну літературу і технічну документацію	14.05.22	
3	Проаналізувати технологію SDR-прийому основні поняття і визначення	16.05.22	
4	Написати розділ 1.	18.05.22	
5	Написати розділ 2.	21.05.22	
6	Виконати розробку програми	25.05.22	
7	Написати розділ 3.	27.05.22	
8	Оформити пояснювальну записку	29.05.22	
9	Підготовити графічний демонстраційний матеріал	29.05.22	

7. Дата видачі завдання _____

Керівник дипломного проекту _____ Кучеров Дмитро Павлович

(підпис)

Завдання прийняв до виконання _____ Ковальов Микита Анатолійович

(підпис випускника) (П.І.Б.)

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Програмний модуль зчитування інформації з SDR-приймача»:

44 с., 13 рис., 20 літературних джерел, 1 додаток.

SDR-ПРИЙМАЧ, РАДІОСТАНЦІЯ, СИГНАЛ, РАДІОЧАСТОТНЕ ОПІЗНАВАННЯ, ВІЙСЬКОВИЙ ЗВ’ЯЗОК.

Об’єкт проектування – зчитування інформації з SDR-приймача.

Предмет проектування – розробка програмного модуля зчитування інформації з SDR-приймача.

Мета проектування – спроектувати та розробити програмного модуля зчитування інформації з SDR-приймача.

Результати дипломного проектування рекомендується використовувати при створенні програмних додатків, що націлені на зчитування інформації з SDR-приймача.

Дипломний проект присвячений актуальній тематиці розробки програмних модулів може використовуватись в учбовому процесі НАУ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

АЦП – аналогово-цифровий перетворювач.

ІЧ – інфрачервоний.

ЦАП – цифро-аналоговий перетворювач

ЦОС – цифрова обробка сигналів

SDR (ПВР) – програмно визначена радіосистема

ВСТУП

Програмно визначена радіосистема (англ. Software-defined radio, SDR, укр. ПВР) — радіопередавач та/або радіоприймач, що використовує технологію, що дозволяє за допомогою програмного забезпечення встановлювати або змінювати робочі радіочастотні параметри, включаючи, зокрема, діапазон частот, тип модуляції або вихідну потужність, за винятком зміни робочих параметрів, що використовуються під час звичайної попередньо визначеної роботи з попередніми установками радіопристрою, відповідно до тієї чи іншої специфікації або системи.

ПВР виконує значну частину цифрової обробки сигналів на звичайному персональному комп'ютері або на ПЛІС.

Метою такої схеми є радіоприймач або радіопередавач довільних радіосистем, що змінюється шляхом програмної переконфігурації (звідси походить альтернативне найменування таких систем — програмно-конфігуровані).

Подібні радіосистеми широко застосовуються для військових додатків [1] та бездротових послуг зв'язку, оскільки дозволяють обслуговувати велику кількість радіопротоколів.

Устаткування для ПВР зазвичай складається з супергетеродинного приймача, який перетворює сигнал із радіочастоти на проміжну, аналого-цифрового та цифро-аналогового перетворювачів (АЦП та ЦАП).

Одна з перших систем ПВР розроблялася американськими військовими під назвою SpeakEasy. Метою проекту було використання програмної обробки для емуляції понад 10 існуючих військових радіосистем, що функціонують у діапазоні від 2 до 20 *МГц*. Іншою метою була можливість підтримки будь-яких нових схем кодування та модуляції, щоб військові могли використовувати досконаліші модуляції та кодування.

Дана технологія дозволяє замінити величезну різноманітність існуючих і розроблюваних конструкцій радіоприймачів і трансіверів, як серійних, так і,

перш за все, аматорських, побудованих за складною супергетеродинною схемою, на обмежену кількість доступних апаратних блоків

Об’єкт проектування – зчитування інформації з SDR-приймача.

Предмет проектування – розробка програмного модуля зчитування інформації з SDR-приймача.

Мета роботи – спроектувати та розробити програмного модуля зчитування інформації з SDR-приймача.

РОЗДІЛ 1. ДОЦІЛЬНІСТЬ ВИКОРИСТАННЯ SDR-ПРИЙМАЧІВ

1.1. Технології реалізації програмно-керованого радіо

Основний принцип SDR технології полягає у реалізації функцій радіосистеми (раніше виконуваних у аналоговому вигляді за допомогою електронних пристроїв) – у цифровому вигляді за допомогою обробки оцифрованого сигналу. І навіть не тільки у перенесенні обробки в цифру, а й у можливості застосовувати більше складних інструментів для обробки сигналів. Але це аж ніяк не означає, що фізичне втілення будь-якого типового сучасного SDR трансівера – це АЦП/ЦАП + модуль ЦОС, а все інше – не потрібне (а якщо воно є, то воно другорядне). А, відповідно, не потрібно знань, пов'язаних з розумінням аналогової схемотехніки радіомодуля. Як достатньо отримати АЦП, ЦАП з крутими параметрами, підключити їх до антени з одного кінця, до ПК з іншого - і ось готовий SDR на всі випадки і на всі покоління). Може такого підходу і достатньо для демонстрації роботи принципів SDR, але цього недостатньо для реальної зв'язкової апаратури [1].

Поява нових технологій не скасовує фізику, природу речей та аналогову схемотехніку [2].

Як відомо, за допомогою АЦП має сенс оцифровувати сигнал із крайньою частотою сигналу у два або більше разів меншої частоти семплювання. Тільки в цьому випадку інформацію із сигналу можна відновити без втрати. Дуже часто виникають ситуації, коли крайня частота сигналу перевищує половину частоти семплювання існуючих у світі АЦП, або коли варіант використання таких АЦП невиправдано дорогий для конкретного проекту. В цьому випадку нікуди не подітися від необхідності перенесення сигналу в нижню частину спектра в аналоговому вигляді. І це повсюдно використається. Не кажучи вже про необхідність посилювати та фільтрувати аналоговий сигнал для досягнення необхідних характеристик радіосистеми [3].

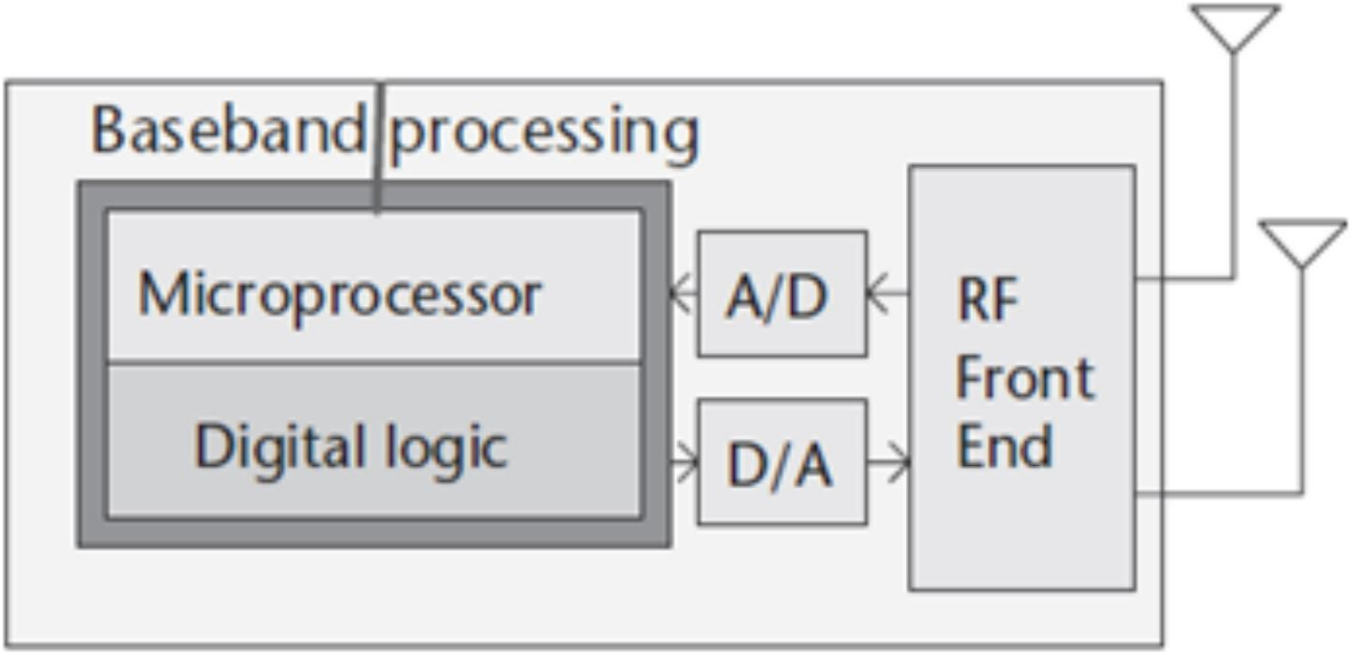

Рис.1.1. Типова архітектура SDR-радіо

Припустимо, розробник, який мало мав справи безпосередньо з радіоелектронікою, дізнався, що таке диво — SDR, його можна програмувати, обробляти/синтезувати сигнали програмними засобами, займатися радіохакінгом тощо. Тобто за допомогою програмного коду можна формувати реальні фізичні радіосигнали, обробляти прийняті [4]. І людині тема цікава, вона повністю в неї занурюється і ... і вона найчастіше не починає вивчати теорію радіозв'язку, радіотехніку, вона робить у цій темі те, що вже вміє і що їй подобається більше - освоює інструменти програмного проектування, починає згадувати/вивчати ЦГЗ , - в результаті вважає ці області найголовнішим і важливим у трансівері, все інше йде на далекий задній план і начебто воно не так важливе і в принципі не так потрібне. Однак такий підхід працює, як правило, тільки в межах аматорського використання подібних пристроїв. Насправді, якщо самостійно необхідно спроектувати систему зв'язку, IoT пристрій, якусь РЛС і т.п., крім алгоритмів роботи у цифровому вигляді не менш важливу роль відіграють радіочастотні характеристики трансівера, які при розробці обов'язково розраховуються, моделюються, оптимізуються. Ці характеристики багато в чому визначають допустимий діапазон застосування конкретного радіомодуля [5].

HackRF One є відкритим проектом, вихідний код його доступний.

Апаратні характеристики практично у всіх джерелах наведені досить мізерним переліком:

• Робоча частота від 1 *МГц* до 6 *ГГц*

• напівдуплексний трансивер

• до 20 мільйонів зразків в секунду

• 8-бітові квадратурні вибірки (8-бітовий I і 8-бітовий Q)

• сумісний з GNU Radio, SDR# тощо

• програмне налаштування посилення RX і TX та фільтра основної смуги

• програмно керована потужність порту антени (50 *мА* при 3,3 *В*)

• Роз'єм для антени SMA

• синхронізації SMA по годиннику

• зручні кнопки для програмування

• внутрішні штифтові роз'єми для розширення

• високошвидкісний USB 2.0

• живлення від USB

• апаратне забезпечення з відкритим кодом

Через USB-кабель модуль підключається до ПК. Через цей інтерфейс HackRF One отримує живлення від хоста, через нього відбувається передача всіх даних у цифровому вигляді. На стороні HackRF One маємо мікроконтролер LPC4320FBD144 з ARM Cortex-M4 ядром, стоїть CPLD XC2C64A-7VQG100C, з'єднана з мікросхемою MAX5864, яка являє собою два АЦП і два ЦАП у єдиному корпусі. Розрядність АЦП = 8 біт, розрядність ЦАП = 10 біт. У цілому нині характеристики в цих АЦП і ЦАП досить скромні. Максимальна частота їх тактування становить 22 *МГц* (що і є обмежуючим фактором максимально досяжної частотної смуги сигналу, теорему Котельникова не перехитрити [6]).

1.2. Сфери застосування SDR-приймачів

Дана технологія дозволяє замінити величезну різноманітність існуючих і розроблюваних конструкцій радіоприймачів і трансіверів, як серійних, так і, перш за все, аматорських, побудованих за складною супергетеродинною схемою,

на обмежену кількість доступних апаратних блоків, що працюють під управлінням ПО, що розробляється спільнотою [7]. Це призведе до спрощення та здешевлення конструкцій, суттєвого поліпшення характеристик, підтримки будь-яких видів модуляції, появи великої кількості сервісних функцій, а також прискорить розробку, оскільки програмне забезпечення може вдосконалюватися одночасно всім співтовариством [8]. Таке стало можливим з появою доступних швидких ЦАП та АЦП (іноді достатньо звукової плати ЕОМ) та здешевленням ПЕОМ та DSP-процесорів.

Найперше дана технологія була використана в проекті SpeakEasy, який розроблявся армією США[9].

Об’єднана тактична радіосистема (JTRS) була програмою американських військових для виробництва радіоприймачів, які забезпечують гнучкий і сумісний зв’язок. Приклади радіотерміналів, які потребують підтримки, включають ручні, автомобільні, бортові та демонтовані радіостанції, а також базові станції (фіксовані та морські). Ця мета досягається за рахунок використання систем SDR, заснованих на міжнародно схваленій відкритій архітектурі програмного зв'язку (SCA). Цей стандарт використовує CORBA в операційних системах POSIX для координації різних програмних модулів. Програма забезпечує новий гнучкий підхід для задоволення різноманітних потреб солдатів у зв’язку за допомогою програмно-програмованих радіотехнологій. Вся функціональність і можливості розширення побудовані на основі SCA. Гнучкість SDR призводить до дорогої складності, неможливості оптимізації, ускладнення застосування новітніх технологій. SCA, незважаючи на своє військове походження, перебуває на стадії оцінки комерційними постачальниками радіо на предмет застосування в їхніх сферах. Однак прийняття систем SDR загального призначення за межами військового, розвідувального, експериментального та аматорського використання за своєю природою перешкоджає тому факту, що цивільні користувачі можуть використовувати інші, більш гнучкі та економічні архітектури. Тим не менш, притаманна програмному забезпеченню гнучкість радіо може дати значні переваги в довгостроковій перспективі, як тільки фіксовані витрати на його впровадження

знизиться настільки, щоб перевищити вартість повторного перепроектування спеціально створених систем. Це пояснює зростаючий комерційний інтерес до технології. Інфраструктурне програмне забезпечення на основі SCA та інструменти швидкого розвитку для навчання та досліджень SDR надаються проектом Open Source SCA Implementation – Embedded [10,11].

Ось лише короткий перелік напрямків, де технологія SDR успішно запроваджується вже зараз: Mesh, Smart-Grid, Smart Antenna, MI MO, RFID, WRAN, DVB, DRM, LTE [12].

Використання пристроїв на базі SDR дозволяє успішно вирішувати основні питання при модернізації обладнання для бездротового зв'язку, а це терміни, вартість та ризики. Якщо раніше створення та виробництво телекомунікаційних платформ з нуля займало від 2 до 3 років, зараз проектування нового пристрою на базі SDR-платформи може тривати від 6 місяців до 1 року. Сюди входить оптимізація апаратної частини та ПЗ, а також написання конструкторської документації. Використання готової програмно-апаратної платформи дозволяє закрити і виробничі питання: досвідчені зразки пристрою доступні вже на старті проекту, налагоджений процес постачання компонентів, обрані оптимальні виробничі майданчики [13].

Українські та закордонні розробники, системні інтегратори та оператори бездротового зв'язку зможуть на власному досвіді переконатися в тому, що програмно-реалізоване радіо – це скорочення термінів, зниження витрат та ефективне управління інвестиційними ризиками.

Громадянські рації, які вимагають реєстрації, працюють на частотах 433 і 446 *МГц*.

Щоб слухати рації на частоті 27 *МГц*, потрібен тюнер із мікросхемою R820T або зовнішній конвертер у випадку E4000. Оптимальна антена для 27 *МГц* вже потрібна серйозніша, довжиною ~2,59 або ~1,23 *м*.

З історичних причин радіозв'язку в авіації використовується амплітудна модуляція. Зазвичай, передачі з літаків краще чути, ніж від диспетчерів або погодних інформаторів на землі. Діапазон частот - 117-130 *МГц*.

До приходу інтернету КВ-радіостанції були одним із способів дізнаватися новини з іншого кінця земної кулі — короткі хвилі, відбиваючись від іоносфери, можуть братися далеко за горизонтом. Велика кількість КВ-радіостанцій існує й досі, їх можна шукати в діапазоні ~8–15 *МГц*.

Подальший розвиток – цифрові DRM-радіостанції: на коротких хвилях передається стислий звук із корекцією помилок + додаткова інформація. Слухати їх можна за допомогою декодера Dream. Діапазон частот для пошуку – від 0 до 15 *МГц*. Потрібно пам'ятати, що для таких низьких частот може знадобитися велика антена.

Крім цього, можна почути передачі радіоаматорів - на частотах 1810-2000 *кГц*, 3500-3800 *кГц*, 7000-7200 *кГц*, 144-146 *МГц*, 430-440 *МГц* та інших.

Одна з незвичайних можливостей - прийом навігаційних сигналів із супутників GPS на TV-тюнер. Для цього знадобиться активна GPS-антена (з підсилювачем). Підключати антену до тюнера потрібно через конденсатор, а до конденсатора (з боку активної антени) - батарейка на 3 для живлення підсилювача в антені.

Далі можна або обробляти злитий дамп ефіру – це може бути цікаво для вивчення принципів роботи GPS, – або використовувати GNSS-SDR, який реалізує декодування сигналів GPS в реальному часі.

Прийняти аналогічним способом сигнал із ГЛОНАСС-супутників було б важко - там різні супутники передають на різних частотах, і всі частоти в смугу RTL2832 не поміщаються.

RTL2832 можна використовувати для налагодження радіопередавачів, підслуховування за радіонянями та аналоговими радіотелефонами, для розбору протоколів зв'язку в іграшках на радіокеруванні, радіодзвінках, пультів від машин, погодних станцій, систем віддаленого збору інформації з датчиків, електролічильників. З конвертором можна зчитувати код із найпростіших 125 *кГц* RFID міток. Сигнали можна записувати днями, аналізувати і потім повторити в ефір на передаючому обладнанні. При необхідності тюнер можна підключити до Android-пристрою, Raspberry Pi або іншого компактного комп'ютера для організації автономного збору даних з радіоефіру.

Можна приймати фотографії з погодних супутників і слухати передачі з МКС — але вже знадобляться спеціальні антени, підсилювачі. Фотографії декодуються програмою WXtoImg [14].

Є можливість захоплювати зашифровані дані, що передаються GSM-телефонами (проект airprobe), якщо в мережі вимкнено frequency-hopping.

Можливості SDR на основі RTL2832 все-таки не безмежні: до Wi-Fi і Bluetooth він не дістає по частоті, і, навіть якщо зробити конвертер, через те, що смуга захоплених частот не може бути ширшою за ~2,8 *МГц*, неможливо прийматиме навіть один канал Wi-Fi. Bluetooth 1600 разів на секунду змінює робочу частоту в діапазоні 2400-2483 *МГц*, і за цим не встигнути. З цієї причини неможливий повноцінний прийом аналогового телебачення (там потрібна смуга 8 МГц, що приймається, з 2,8 *МГц* можна отримати тільки чорно-білу картинку без звуку). Для таких застосувань потрібні серйозніші SDR-приймачі: HackRF, bladeRF, USRP1 та інші.

Типове аматорське програмне радіо використовує приймач прямого перетворення. На відміну від приймачів прямого перетворення більш далекого минулого, використовувані технології змішування базуються на детекторі квадратурної вибірки та збуджувачі квадратурної вибірки.

Продуктивність приймача цієї лінійки SDR безпосередньо пов'язана з динамічним діапазоном використовуваних аналого-цифрових перетворювачів. Радіочастотні сигнали перетворюються в діапазон частот звуку, який дискретується високопродуктивним АЦП аудіочастот. SDR першого покоління використовували звукову карту ПК 44 *кГц* для забезпечення функціональності АЦП. Новіші програмно-визначені радіостанції використовують вбудовані високопродуктивні АЦП, які забезпечують більший динамічний діапазон і більш стійкі до шумів і радіочастотних перешкод.

Швидкий ПК виконує операції цифрової обробки сигналів (DSP) за допомогою програмного забезпечення, специфічного для апаратного забезпечення радіо. Кілька програмних реалізацій радіо використовують бібліотеку SDR з відкритим кодом DttSP.

Програмне забезпечення SDR виконує всю демодуляцію, фільтрацію (як радіочастоту, так і аудіочастоту) та покращення сигналу (вирівнювання та бінауральне представлення). Використання включає в себе будь-яку поширену аматорську модуляцію: азбуку Морзе, односмугову модуляцію, частотну модуляцію, амплітудну модуляцію та різноманітні цифрові режими, такі як радіотелетайп, телебачення з повільним скануванням і пакетне радіо. Аматори також експериментують з новими методами модуляції: наприклад, проект з відкритим кодом DREAM декодує техніку COFDM, яку використовує Digital Radio Mondiale.

Існує широкий спектр апаратних рішень для радіоаматорів і домашнього використання. Існують рішення для приймачів професійного рівня, наприклад Zeus ZS-1 або Flex Radio, рішення для домашнього пивоваріння, напр. Приймач PicAStar, комплект SoftRock SDR і рішення для стартера чи професійного приймача, напр. FiFi SDR для коротких хвиль або когерентний багатоканальний SDR-приймач Quadrus для коротких хвиль або VHF/UHF у прямому цифровому режимі роботи.

1.3. Основні показники SDR-прийому

Основними показниками прийому є показники ЦАП та АЦП, оскільки швидкодія не має вирішального впливу. Одним із найвпливовішим показником є тактова частота. Також варто виділити параметри чутливості, лінійності та вибірковості. А також тип модуляції.

На практиці ж важливим показником є енергоефективність, оскільки дані системи в основному призначені для використання там, де відсутнє джерело безперебійного енергопостачання як таке.

Функціональні властивості апаратури: діапазон приймання частот, чутливість, вибірковість, якість відтворювання радіосигналу, гучність звучання, потужність відтворюваного електричного сигналу, можливість утворення стереомовлення, можливість використання додаткових пристроїв (підсилю-вачів низької частоти, магнітофона, годинників, акустичних систем, головних телефонів тощо). Діапазон приймання частот (радіохвиль) - ділянка частот, у

межах якої здійснюється радіоприйом для конкретного виду і моделі радіоприймальної апаратури [15].

Чутливість - це властивість радіоприймача приймати радіо-сигнали при їх мінімальній напрузі на вході (антенні) та забезпечувати при цьому нормовану вихідну потужність. При визначенні чутливості (та інших показників якості) приймача на його вхід подається стандартний сигнал: F = 1000 *Гц*; m = 30% (для АМ тракту). Стандартна вихідна потужність приймача P_{CT} = 50 *мВт* і P_{CT} = 5 *мВт* (остання для приймачів з номінальною вихідною потужністю менше 150 *мВт*). У наведеному визначенні чутливості не враховуються перешкоди (власні шуми приймача і зовнішні перешкоди), тому така чутливість називається ідеальною. Вона визначається тільки коефіцієнтом посилення приймача. Здатність приймача приймати слабкі сигнали на тлі перешкод оцінюється реальної чутливістю [16].

Кількісно вона характеризується мінімальним напругою вхідного радіосигналу, при якому на виході приймача відношення напруги корисного сигналу до напруги перешкод – не менше заданого значення при стандартній потужності.

Вибірковість (селективність) — це здатність приймача виділяти корисний радіосигнал із всієї кількості сигналів, які впливають на антену. Розглядають два види вибірковості: вибірковість по сусідньому каналу і вибірковість по додаткових каналах прийому. Частотна характеристика радіоприймачів чи діапазон відтворюваних частот характеризує ширину діапазону звукових частот, відтворюваних радіоприймачем без спотворень. Якість відтворювання сигналу характеризується ступенем спотворень, які вносилися радіоприймальним пристроєм до спектра відтворюваних частот. Зайві частоти погіршують якість передач, сприяють появі небажаних ефектів. Гучність відтворюваних звуків визначається перш за все потужністю відтворюваного електричного сигналу низької частоти. Вона визначається можливістю апаратури озвучувати приміщення, різного за площею.

Висновки за розділом

В даному розділі було розглянуто основний принцип SDR технології, основні поняття радіотехніки та будову SDR ресівера. Виділено проект HackRF One, його основні характеристики та апаратні складові.

Основні характеристики HackRF One:

• Робоча частота від 1 *МГц* до 6 *ГГц*

• Напівдуплексний трансивер

• До 20 мільйонів зразків в секунду

• 8-бітові квадратурні вибірки (8-бітовий I і 8-бітовий Q)

• Сумісний з GNU Radio, SDR# тощо

• Програмне налаштування посилення RX і TX та фільтра основної смуги

• Програмно керована потужність порту антени (50 *мА* при 3,3 *В*)

• Роз'єм для антени SMA

• Синхронізації SMA по годиннику

• Зручні кнопки для програмування

• Внутрішні штифтові роз'єми для розширення

• Високошвидкісний USB 2.0

• Живлення від USB

• Апаратне забезпечення з відкритим кодом

Описано сфери застосування SDR та саму історію появи технології.

Виділено основні параметри SDR пристроїв такі як діапазон приймання частот, чутливість, вибірковість, якість відтворювання радіосигналу, гучність звучання, потужність відтворюваного електричного сигналу, можливість утворення стереомовлення, можливість використання додаткових пристроїв.

РОЗДІЛ 2.ОСОБЛИВОСТІ ОБРОБЛЕННЯ ІНФОРМАЦІЇ SDR-ПРИЙМАЧЕМ

2.1. Методи виміру частоти сигналів

Вимірювання частоти цифровим методом (методом дискретного рахунку) реалізовано в цифрових (електронно-рахункових) частотомірах (ЕРЧ). Принцип дії ЕРЧ заснований на вимірюванні частоти відповідно до її визначенням, т. Е. На рахунку числа імпульсів за певний інтервал часу.

ЕРЧ зручні в експлуатації, мають широкий діапазон вимірюваних частот (від декількох герц до сотень мегагерц) і дозволяють отримати результат вимірювання з високою точністю (відносна похибка вимірювання частоти 10^{-6} ... 10^{-9}).

ЕРЧ є багатофункціональними приладами. Залежно від режиму їх роботи можна проводити вимірювання не тільки частоти і відносини двох частот, але і інтервалів часу (періоду проходження періодичних сигналів і інтервалу, заданого тимчасовим положенням двох імпульсів) [17].

Досліджуваний гармонійний сигнал u (t) подається на вхідний пристрій, що забезпечує посилення або ослаблення його до значення, необхідного для роботи наступних пристроїв частотоміра, і фільтрацію досліджуваного сигналу.

Формуючий пристрій формує послідовність коротких імпульсів ($U_{фп}$ (рахункових імпульсів), які прямують з періодом T_x і пов'язаних з досліджуваним сигналом і (1) моментом появи так, що передні фронти цих імпульсів практично збігаються з моментами переходу досліджуваного сигналу $u(t)$ через нульове значення на осі часу з позитивною похідною [18].

Пристрій формування і управління формує прямокутний імпульс $U_{фп}$ каліброваної тривалості, отриманий шляхом ділення частоти опорного генератора і визначає час вимірювання. В якості опорного генератора, як правило, використовується високостабільний термостатований кварцовий генератор частотою 1 або 5 *МГц*.

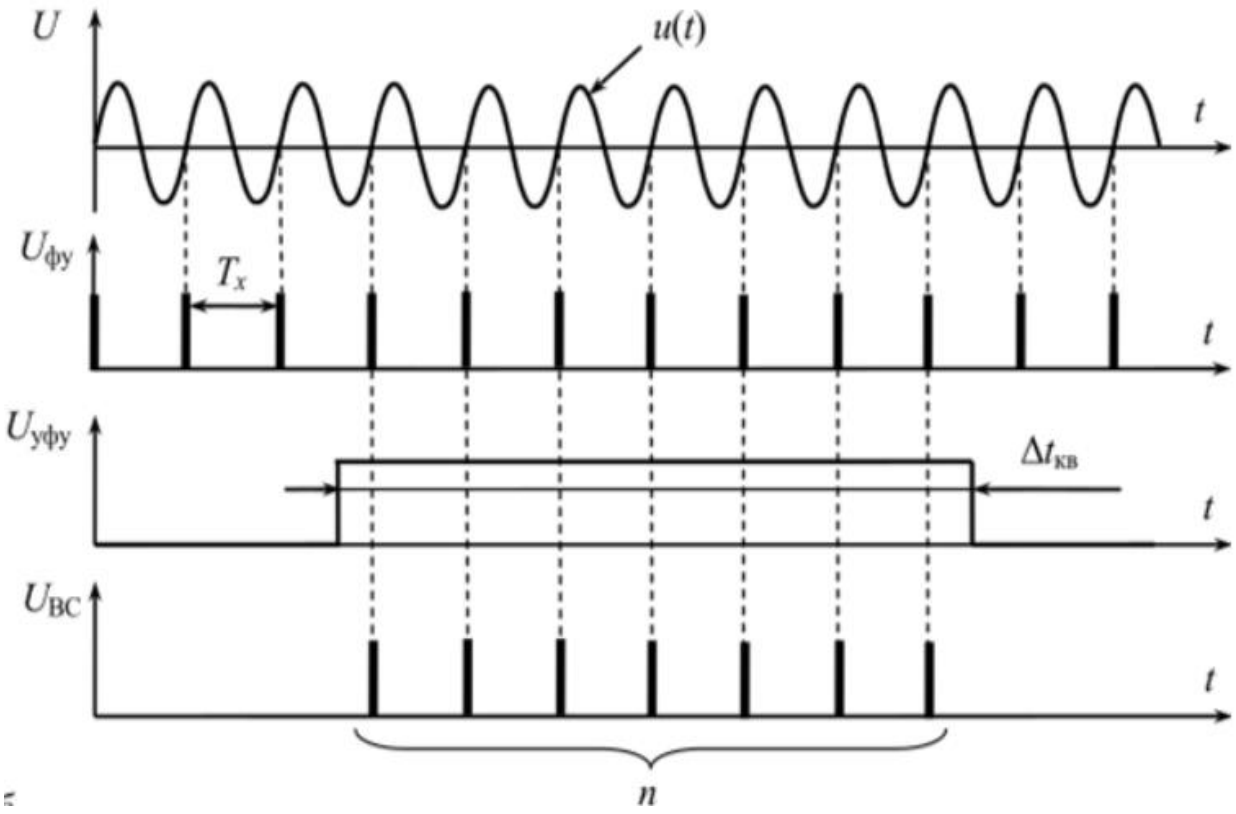

Рис.2.1. Діаграми роботи ЕРЧ

Лічильник забезпечує підрахунок і запам'ятовування числа імпульсів, що пройшли через часовий селектор за час стрибає-імпульсу. Інформація з лічильника надходить на цифровий пристрій, на якому з'являється інформація в одиницях частоти [20] Перед початком нового циклу вимірювань необхідно підготувати лічильник, скинувши свідчення минулого циклу. Це робиться через ланцюг скидання від пристрою формування і управління.

Взаємозв'язок між тривалістю імпульсу і числом рахункових імпульсів (з точністю до одного періоду T_x) визначається виразом 2.1.

$$\Delta t_{кв} = nT_x = n/f_x \qquad (2.1)$$

Звідки

$$f_x = n/\Delta t_{кв} = n \times 10^a \qquad (2.2)$$

При a = 0 число n відповідає вимірюваній частоті f_x в герцах, при a = 3 число n відповідає вимірюваній частоті f_x в кілогерцах, при a = 6 число n відповідає вимірюваній частоті f_x в мегагерцах і т.д. Таким чином, вимірювана частота f_x дорівнює числу імпульсів n, пропущених тимчасовим селектором, помножених на коефіцієнт 10^a, що визначає одиницю виміру частоти і кількість значущих цифр при відліку.

Похибка вимірювання частот при використанні методу дискретного рахунку має дві основні складові. Перша складова - це похибка формування зразкового інтервалу часу $ДГ_{кв}$, протягом якого часовий селектор пропускає імпульси. Ця похибка в основному визначається відносною нестабільністю частоти опорного кварцового генератора рівною $\delta_д$ в межах 10^{-7} ... 10^{-4} .

Другою складовою похибки вимірювання частоти є похибка, обумовлена взаємним розташуванням інтервального і рахункових імпульсів, звана погрішністю дискретизації, що дорівнює одному періоду $\delta_д$

$$\delta_д = 1/n = 1 / (f_x \times \Delta t_{кв}) \qquad (2.3)$$

Таким чином, загальний вираз для визначення відносної похибки запишеться в наступному вигляді:

$$\delta_f = \delta_{кв} + 1 / (f_x \times \Delta t_{кв}) \qquad (2.4)$$

Як видно з наведених формул, похибка дискретизації зменшується зі збільшенням вимірюваної частоти і інтервалу $\delta_д$. Але зі збільшенням $\delta_д$ зменшується швидкодія ЕРЧ. У реальних приладах максимальний час вимірювання обмежується значенням 10 c, тому при вимірюванні досить низьких частот застосування розглянутого методу неефективно.

2.2. GNU Radio

GNU Radio – це вільна платформа для цифрової обробки сигналів.

GNU Radio являє собою набір програм і бібліотек, які дозволяють створювати довільні радіосистеми, схеми модуляції, форми прийнятих і переданих сигналів в яких всі параметри задаються програмно, а для захоплення і генерації сигналів застосовуються найпростіші апаратні пристрої. Проект поширюється під ліцензією GPLv3. Код більшості компонентів GNU Radio написаний на мові Python, а частини, критичні до швидкодії і часу затримки, написані на мові C++, що дозволяє використовувати пакет при вирішенні завдань у режимі реального часу.

У комбінації з універсальними програмованими прийомопередавачами USRP2, що не прив'язані до смуги частот і типів модуляції сигналу, платформа може бути використана для створення таких пристроїв, як базові станції для GSM мереж, пристрої для дистанційного читання RFID-міток (електронні посвідчення і пропуски, смарт-карти), GPS-ресивери, Wi-Fi, приймачі та передавачі FM-радіо, TV-декодери, пасивні радари, спектральні аналізатори тощо. Крім USRP, пакет може використовувати й інші апаратні компоненти для вводу і виводу сигналів, наприклад, доступні драйвери для звукових карт, TV-тюнерів, пристроїв Softrock, Comedi, Funcube і S-Mini.

До складу також входить колекція фільтрів, канальних кодеків, модулів синхронізації, демодуляторів, еквалайзерів, голосових кодеків, декодерів і інших елементів, необхідних для створення радіосистем. Зазначені елементи можуть бути використані як цеглинки для компонування готової системи, що у поєднанні з можливостями за визначенням потоків даних між блоками, дозволяє проектувати радіосистеми навіть без навичок програмування.

Вперше опублікований у 2001 році, GNU Radio є офіційним пакетом GNU. Філантроп Джон Гілмор ініціював GNU Radio з фінансуванням у розмірі 320 000 доларів США Еріку Блоссому для створення коду та виконання обов’язків з управління проектами.

GNU Radio починалося як розгалуження коду Pspectra, який був розроблений проектом SpectrumWare в Массачусетському технологічному інституті (MIT). У 2004 році було завершено повне переписування GNU Radio, тому сьогодні GNU Radio більше не має оригінального коду Pspectra.

Метт Еттус приєднався до проекту як один із перших розробників і створив Universal Software Radio Peripheral (USRP), щоб забезпечити апаратну платформу для використання з програмним забезпеченням GNU Radio. У 2004 році Метт заснував Ettus Research LLC і почав продавати USRP, які працювали з GNU Radio.

У вересні 2010 року Ерік Блоссом пішов з посади керівника проекту, і його замінив Том Рондо.

На початку проекту основні розробники почали проводити піврічні Hackfests . У 2011 році проект GNU Radio розпочав щорічну конференцію під назвою «GRCon», яка зазвичай проводить Hackfest в останній день конференції.

У березні 2016 року Том Рондо пішов у відставку і був замінений Беном Хілберном на посаді керівника проекту та Джонатаном Корганом, давнім супроводжувачем, як головним архітектором.

У січні 2018 року Джонатан Корган пішов у відставку з посади головного архітектора і був замінений Маркусом Мюллером.

У вересні 2020 року GNU Radio стало частиною Інституту SETI (некомерційна багатодисциплінарна дослідницька та освітня організація) для всіх фінансових та договірних цілей.

У жовтні 2020 року Бен Хілберн і тодішні керівники проекту проголосували за реорганізацію керівництва GNU Radio Project, сформувавши генеральну асамблею з набором підзаконних актів, які регулюють деталі роботи організації. Рада з трьох членів, що складається з обраних членів генеральної асамблеї, взяла на себе функції, які раніше виконував керівник проекту.

GNU Radio Companion — це графічний інтерфейс користувача, який використовується для розробки програм GNU Radio. Це інтерфейс бібліотек GNU Radio для обробки сигналів . GRC був розроблений Джошем Блумом під час його навчання в Університеті Джона Хопкінса (2006-2007), а потім

розповсюджувався як безкоштовне програмне забезпечення для Hackfest у жовтні 2009 року . Починаючи з випуску 3.2.0, GRC офіційно входив у комплект із дистрибутивом програмного забезпечення GNU Radio.

GRC фактично є інструментом генерації коду Python. Коли блок-графік «компілюється» в GRC, він генерує код Python, який створює потрібні вікна та віджети графічного інтерфейсу, а також створює та з'єднує блоки в блок-схеми.

GRC наразі підтримує створення графічного інтерфейсу за допомогою набору інструментів Qt .

2.3. Захват та обробка сигналу

Для захоплення сигналу використовується протокол NEC. Це доволі відомий протокол розроблений компанією NEC. Наразі існує безліч варіацій протоколу.

Особливості формату:

- 8 біт даних та 8 біт команда
- Адреса та команда передаються двічі для виключення помилки
- Метод кодування біт – інтервалами
- Несуча частота 38 *кГц*
- Час передачі біта 1.125 *мс* або 2.25 *мс*.

NEC формат використовує передачі інформації опорну частоту 38 *кГц* зі шпаруватістю 3 чи 4. Усі пакети крім преамбули мають тривалість 560 *мкс* (21 імпульс опорної частоти). "Одиниця" передається інтервалом 2.25 *мс*, "нуль" – інтервалом 1.125 *мс*.

Типовий формат посилки NEC складається з преамбули - пакета несучої частоти тривалістю 9 *мс*, за якою слідує проміжок 4.5 *мс*. Преамбула допомагає приймачеві встановити необхідний рівень посилення і нуля, хоча сучасні приймачі цього не потребують. Далі слідує реперний пакет стандартної тривалості і за нею 32 інтервали, що відповідають 4 байтам інформації, причому кожен байт передається молодшим бітом уперед. Адреса та команда

передаються двічі, вдруге вони передаються у додатковому коді. Загальна тривалість передачі пакета стала, оскільки кожен біт передається спочатку в прямому, потім в інверсному вигляді. На малюнку показано передачу з адресою 59 та командою 16.

Рис.2.3. Передача з використанням протоколу NEC

Одна з найбільш поширених модифікацій протоколу NEC відрізняється від стандартного тим, що адреса має не 8, а 16 *біт*. При цьому адреса не передається в додатковому коді і загальна кількість біт зберігається. Однак тривалість пакету стає залежною від коду адреси.

Рис.2.4. Передача з використанням протоколу NEC на 16 бітний адрес

Структурна схема роботи наступна.

ІЧ пульт -> ІЧ приймач -> радіо передавач -> sdr приймач(апаратна частина) -> sdr приймач(програмна частина)

Дана схема зображена на рис 2.5.

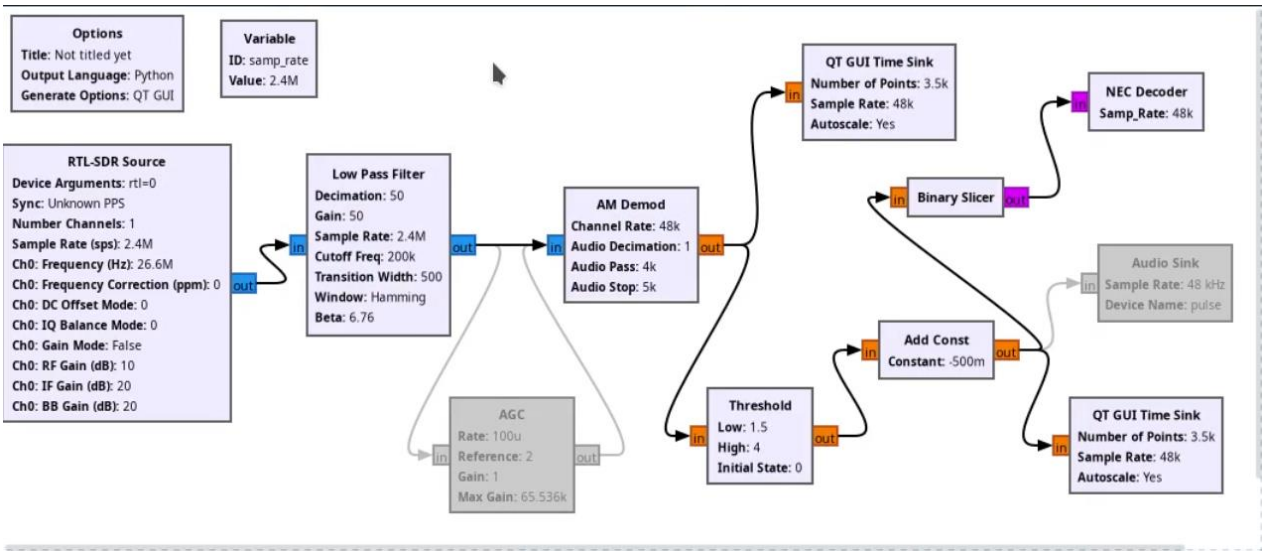

Рис.2.5. Структурна схема

Перший блок RTL-SDR Source це перший блок. Основні параметри це частота 26,4 *Mhz* та рейт 2,4 мегасемпли. Останні три параметри це підсилення сигналу.

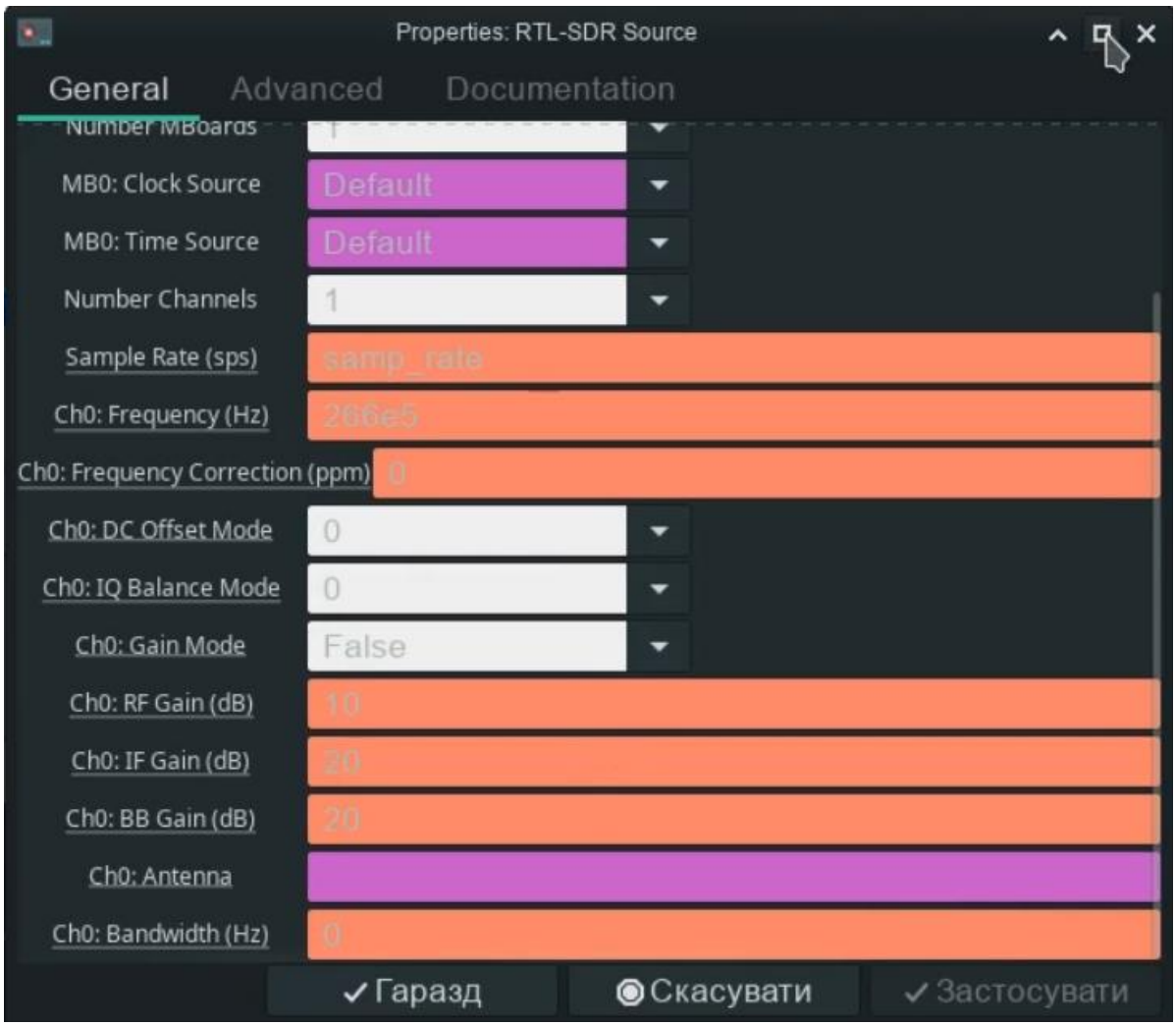

Рис.2.6. Параметри блоку RTL-SDR Source

Далі йде фільтр нижніх частот. Він відфільтровує все що більше 20 *кГц* а також виконує децимацію. Тобто відправляє далі лише кожен 50-ий семпл. Це знижує навантаження, оскільки 2,4 мегасемпли це дуже багато і приймач не справиться з таким обсягом даних.

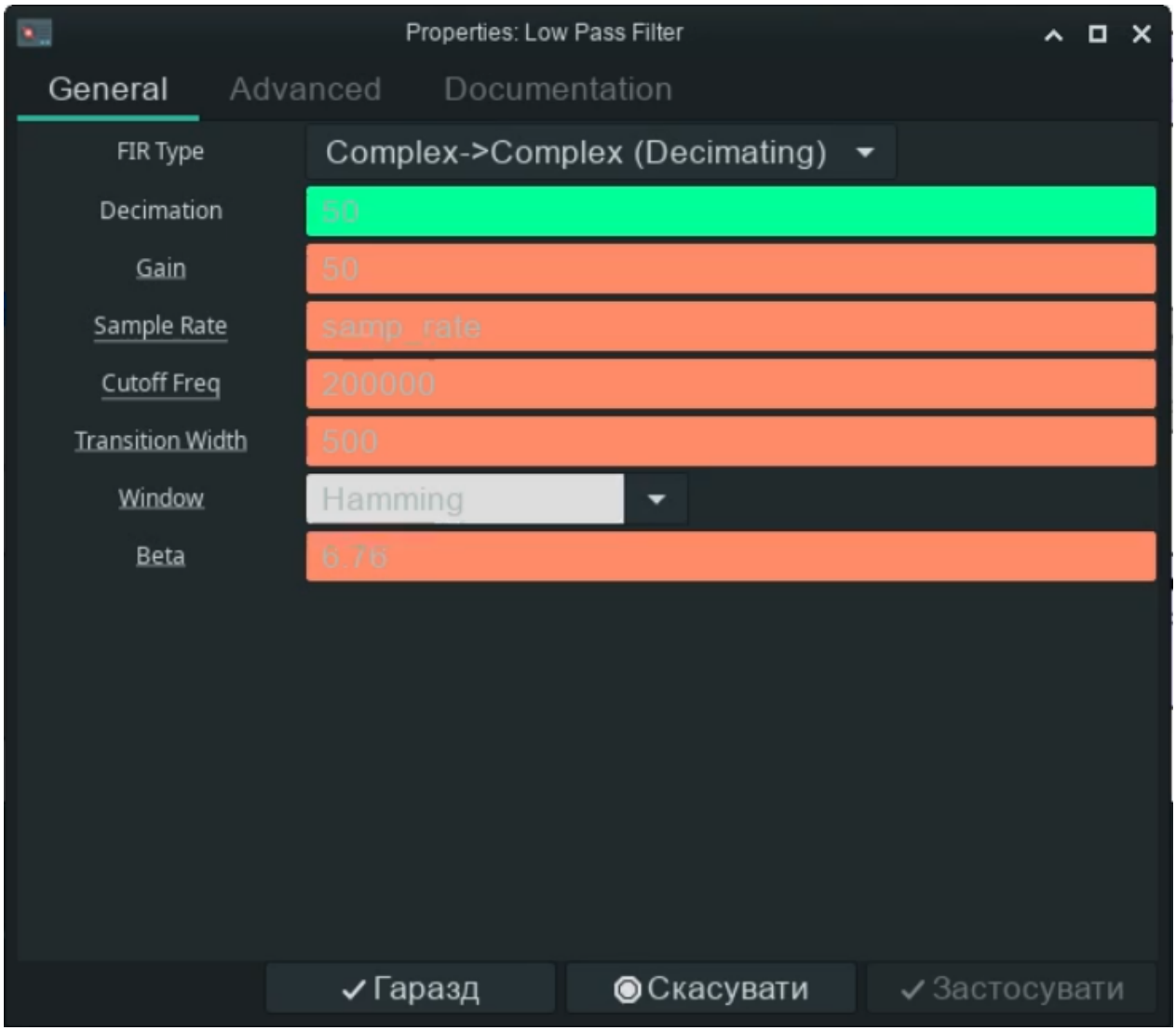

Рис.2.7. Параметри фільтру нижніх частот

Наступним блоком йде амплітудний демодулятор який дає на виході 48 *кГц*. Він відфільтровує все, що вище 4 *кГц*.

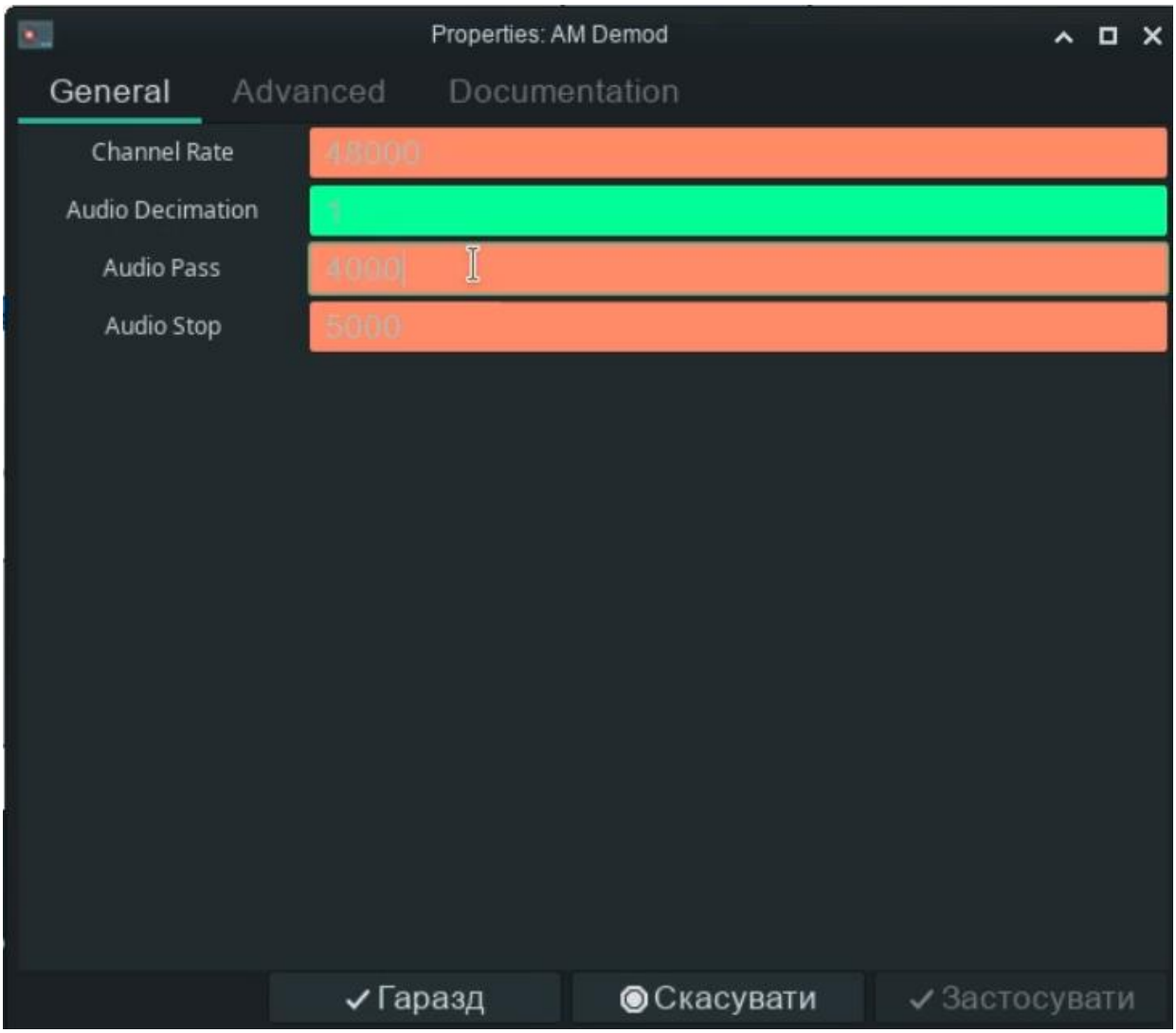

Рис.2.8. Амплітудний демодулятор

Далі сигнал передається на Threshold. Це потрібно аби вирівняти сигнал. Все що вище 4 буде 1.0, а все що нижче 1,5 буде 0.0. Наступним блоком є додавання константи -0.5. тобто на виході з цього блоку буде 0.5 або -0.5. далі сигнал йде на binary slicer. Цей блок все що нижче 0 ставить в 0, а все що вище - в 1. На виході буде потік байт, який передається вже в саму програму. Яку описано нижче.

На рисунку 2.9 зображено оригінальний сигнал та відфільтрований тестові сигнали. (блок QT GUI time sink)

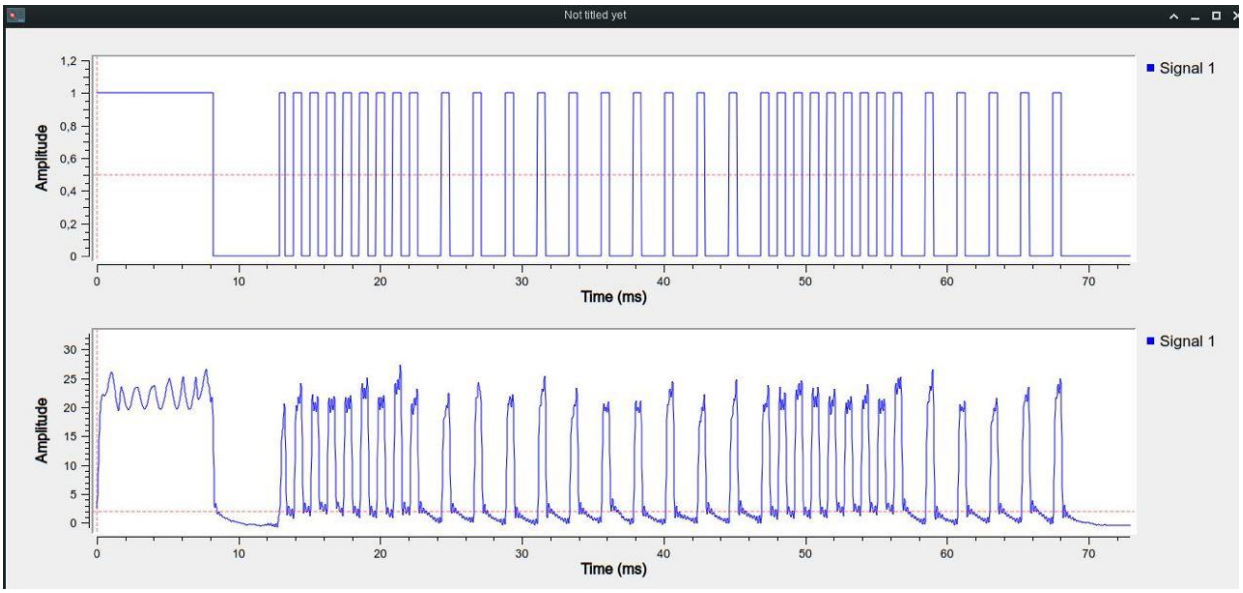

Рис.2.9. Оригінальний та відфільтрований сигнал

Висновки за розділом

В даному розділі було розглянуто методи виміру сигналів. Встановлено що похибка дискретизації зменшується зі збільшенням вимірюваної частоти і інтервалу $\delta_д$. Але зі збільшенням $\delta_д$ зменшується швидкодія ЕРЧ. У реальних приладах максимальний час вимірювання обмежується значенням 10 с, тому при вимірюванні досить низьких частот застосування розглянутого методу неефективно.

Також було описано коротку історію виникнення GNU Radio та саму суть даного проекту. Описано протокол NEC, який використовується для захоплення сигналу, а також структурну схему роботи з сигналом. Показано оригінальний та результуючий сигнали.

Особливості формату (японський варіант):

- 8 біт даних та 8 біт команда
- Адреса та команда передаються двічі для виключення помилки
- Метод кодування біт – інтервалами
- Несуча частота 38 *кГц*
- Час передачі біта 1.125 *мс* або 2.25 *мс*.

Зображена та описана структурна схема процесу.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ

3.1. Прийом сигналу по протоколу NEC

Вихід з Binary Slicer-а передається в нашу програму написану на python.

Байти ловить функція def work(self, input_items, output_items).

Перший елемент параметру input_items це і є список семплів, які обробляються. Параметр self – це глобальна змінна.

Існує 7 станів машини та стан error. Пройдемося по коду.

Лістинг 3.1. Стан 0

```
if self.state == 0:
        if i == 1:
            self.state = 1
            #print("State 1")
            self.count += 1
            continue
```

Якщо стан рівний 0 та імпульс рівний 1, то записуємо стан рівним 1, збільшуємо кількість семплів на 1 і обриваємо цикл.

Лістинг 3.2. Стан 1

```
    if self.state == 1:
      if i == 0:
        #print(self.period*self.count)
        if self.period*self.count > 0.007 and self.period*self.count < 0.009:
            self.state = 3
            #print("State 3")
            self.count = 0
            continue
```

```
                else:
                    print("Error1")
                    print(self.period*self.count)
                    self.state = "Error"
            self.count += 1
```

Якщо стан рівний 1, та імпульс рівний 0 та довжина сигналу від 7 до 9 мс, то встановлюємо стан 3, обнуляємо лічильник та починаємо цикл з наступної ітерації.

Якщо ж тривалість була до 7 чи від 9 *мс*, то стан змінюється на error. У такому випадку машина пропускає наступні 80 *мс* запису, аби пропустити проблемний сигнал. Цього повинно бути достатньо для більшості ситуацій. Також обнуляємо стан, лічильники та очищуємо масив з даними і збільшуємо кількість семплів на 1.

Лістинг 3.3. Стан 3

```
    if self.state == 3:
            if i == 1:
                #print(self.period*self.count)
                if self.period*self.count > 0.0035 and self.period*self.count < 0.0055:
                    self.state = 4
                    #print("State 4")
                    self.count = 0
                    continue
                if self.period*self.count > 0.0017 and self.period*self.count < 0.0028:
                    self.state = 7
                    #print("State 7")
                    self.count = 0
                    continue
```

```
                else:
                    print("Error3")
                    print(self.period*self.count)
                    self.state = "Error"
            self.count += 1
```

Якщо стан рівний 3, та імпульс рівний 1. Якщо сигнал від 3,5 *мс* до 5,5 *мс*, то переходимо в стан 4. Якщо ж сигнал від 1,7 до 2,8 *мс*, то переходимо в стан 7. Це повтор попередньої команди. Скидаємо лічильник. Якщо ж сигнал не входить в дані часові межі, то знову ж таки переходимо в стан помилки.

Лістинг 3.4. Стан 4

```
    if self.state == 4:
            if i == 0:
                #print(self.period*self.count)
                if self.period*self.count > 0.00035 and self.period*self.count < 0.00073:
                    self.state = 5
                    #print("State 5")
                    #self.count = 0
                    continue
                else:
                    print("Error4")
                    print(self.period*self.count)
                    self.state = "Error"
            self.count += 1
```

Якщо стан рівний 4 та імпульс рівний 0. Якщо сигнал від 350 мкс до 730 мкс, то переходимо в стан 5 та НЕ скидаємо лічильник. В іншому випадку обробляємо як помилку.

Лістинг 3.5. Стан 5

```
        if self.state == 5:
                    if i == 1:
                        #print(self.period*self.count)
                        if self.period*self.count > 0.002 and self.period*self.count < 0.003:
                            self.data.append(1)
                            self.bits += 1
                            self.count = 0
                            if self.bits == 32:
                                #print("State 6")
                                self.state = 6
                                continue
                            self.state = 4
                            #print("State 4")
                            continue
                        elif self.period*self.count > 0.0008 and self.period*self.count < 0.0016:
                            self.data.append(0)
                            self.bits += 1
                            self.count = 0
                            if self.bits == 32:
                                #print("State 6")
                                self.state = 6
                                continue
                            self.state = 4
                            #print("State 4")
                            continue
                        else:
                            print("Error5")
```

```
                print(self.period*self.count)
                self.state = "Error"
            self.count += 1
```

В стані 5 ми визначаємо який біт нам прийшов. Якщо сигнал від 2 *мс* до 3 *мс*, то нам прийшла 1, яку ми додаємо в масив даних та збільшуємо масив бітів. Якщо ж сигнал від 0,8 *мс* до 1,6 *мс*, то прийшов 0, виконуємо з ним аналогічні операції.

Якщо прийшло 32 *біт*, то це кінець пакету і переводимо машину в стан 6.

Лістинг 3.6. Стан 6

```
        if self.state == 6:
            if i == 0:
                #print(self.period*self.count)
                if self.period*self.count > 0.00035 and self.period*self.count < 0.00073:
                    self.state = 0
                    #print("State 0")
                    self.count = 0
                    self.bits = 0
                    self.convert_bits()
                    self.data.clear()
                    print(self.comm, "\n")
                    command(self.comm)
                    continue
                else:
                    print("Error6")
                    print(self.period*self.count)
                    self.state = "Error"
            self.count += 1
```

Стан 6. Це обробка завершального імпульсу. Скидаєм стан в 0, обнуляєм лічильник семплів, конвертуємо дані в біти та очищаємо дані. Записуємо код команди та викликаємо обробник команд.

Лістинг 3.7. Стан 7

```
        if self.state == 7:
            if i == 0:
                #print(self.period*self.count)
                if self.period*self.count > 0.00035 and self.period*self.count < 0.00073:
                    self.state = 0
                    print("Repeat: ", self.comm, "\n")
                    command(self.comm)
                    continue
                else:
                    print("Error7")
                    print(self.period*self.count)
                    self.state = "Error"
                self.count += 1
```

Лістинг 3.8. Стан Error

```
    if self.state == "Error":
            if self.period*self.count > 0.080:
                self.state = 0
                self.count = 0
                self.bits = 0
                self.data.clear()
            self.count += 1
```

3.2. Допоміжні функції програми

Функція convert_bits.

Дана функція розбиває 32 цілі значення (int) на 4 байти. По два байти на адрес та команду. Змінні adr та adr1 це пряма та інвертована адреса. Вони повинні співпадати, після повторної інверсії інвертованої адреси. Якщо вони не співпадають, отже сигнал спотворений і це помилка, проте на практиці таке трапляється вкрай рідко.

Змінні comm та comm1 – це пряма та інвертована команда. Вони також повинні співпадати після повторної інверсії.

Далі в функції є 4 цикли для кожної змінної. В кожному циклі змінна типу int з масиву заміняється на біт у числі. Див деталі на лістингу 3.9.

Лістинг 3.8. Конвертація з цілих в біти

```
j = 0
for i in range(0, 8):
    if lst[i] == 0:
        adr = clear_bit(adr, j)
    if lst[i] == 1:
        adr = set_bit(adr, j)
    j += 1
```

Після цього йде перевірка чи адреси та команди співпадають. Якщо так, то записуємо їх в глобальні змінні, якщо ж ні – виводимо повідомлення про помилку.

Не менш важливою є функція command. Дана функція перевіряє прочитану команду та емулює відповідну операцію.

Лістинг 3.8. Функція command

```
def command(comm):
    if comm == 7:
        keyboard.tap(Key.media_volume_down)
```

```
    elif comm == 21:
        keyboard.tap(Key.media_volume_up)
    elif comm == 68:
        keyboard.tap(Key.media_previous)
    elif comm == 64:
        keyboard.tap(Key.media_next)
    elif comm == 67:
        keyboard.tap(Key.media_play_pause)
```

Як бачимо з лістингу 3.10

команда 7 зменшує гучність

команда 21 збільшує гучність

команда 68 перемикає на попередній трек

команда 64 перемикає на наступний трек

команда 67 ставить на паузу.

Загальна блок схема зображена на рис 3.1.

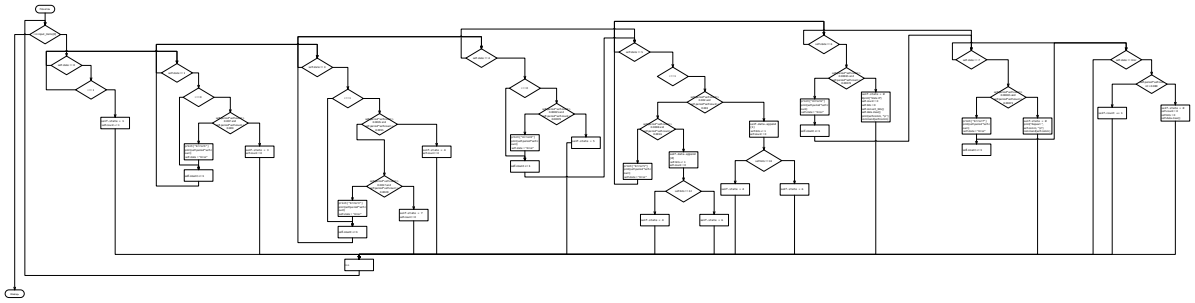

Рис.3.1. блок схема алгоритму

3.3. Використані бібліотеки

Було підключено дві основні бібліотеки gnuradio, pynput, PyQt5 та argparse.

Gnuradio – це бібліотека для роботи з платформою gnuradio. Вона включає в себе весь потрібний інструментарій для отримання даних від платформи та його подальшої обробки.

Pynput – це бібліотека для керування мишею та клавіатурою в програмах python. В нашому випадку використовується для емуляції натискання клавіш клавіатури для відтворення. Емулюються наступні команди:

- стоп,
- наступний трек,
- попередній трек,
- збільшити гучність,
- зменшити гучність

Таким чином діаграма варіантів використання буде такою:

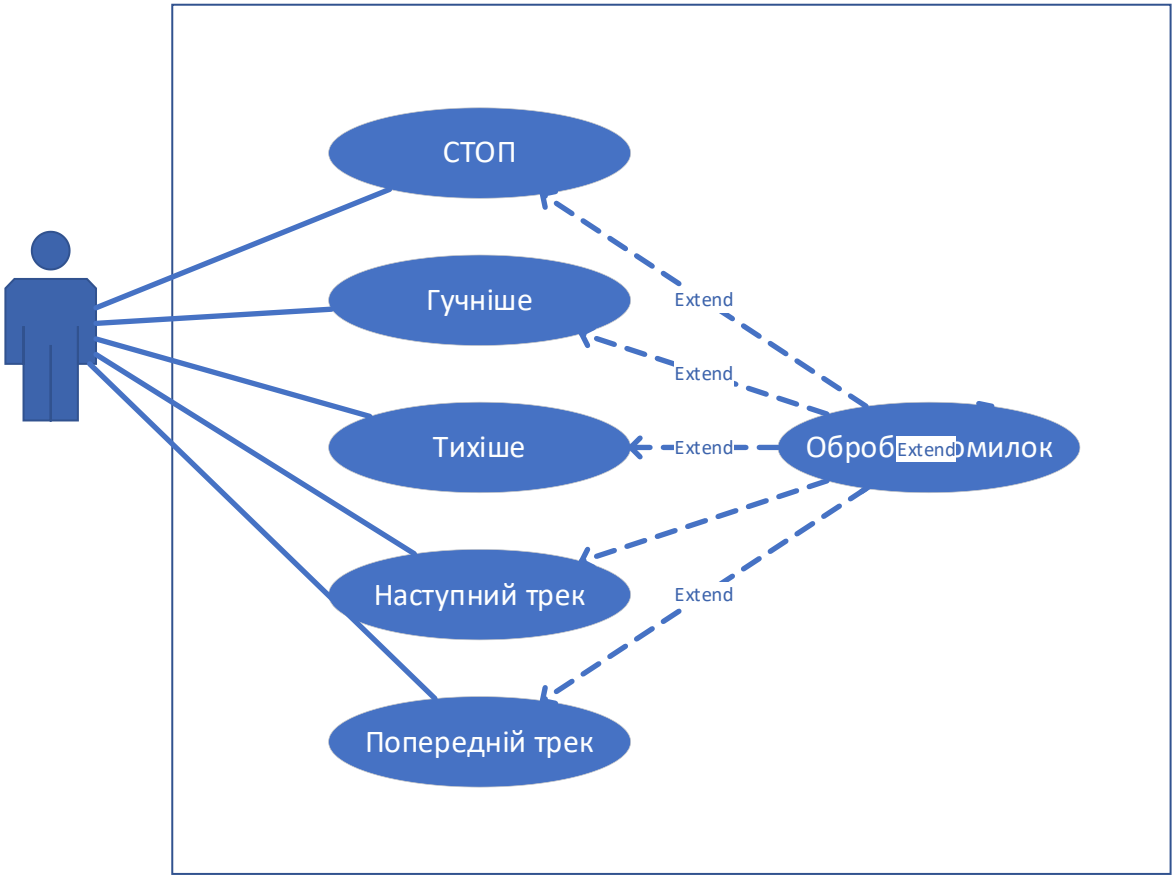

Рис 3.2. Діаграма варіантів використання.

PyQt5 – це бібліотека графічного інтерфейсу на python. PyQt розроблено британською компанією Riverbank Computing. PyQt працює на всіх платформах, що підтримуються Qt: Linux та інші UNIX-подібні ОС, Mac OS X та Windows. Існує 3 версії: PyQt6, PyQt5 та PyQt4, що підтримують відповідні версії Qt. PyQt поширюється під ліцензіями GPL (2 та 3 версії) та комерційною.

PyQt практично повністю реалізує можливості Qt. Це понад 600 класів, понад 6000 функцій та методів.

PyQt також включає Qt Designer – дизайнер графічного інтерфейсу користувача. Програма pyuic генерує Python код із файлів, створених у Qt Designer. Це робить PyQt дуже корисним інструментом швидкого прототипування. Крім того, можна додавати нові графічні елементи управління, написані на Python, Qt Designer.

Раніше PyQt поставлявся разом із середовищем розробки Eric, написаним на PyQt. Eric має вбудований наладчик і може бути використаний для створення консольних програм. Тепер вона доступна як окремий проект.

Argparse – це парсер командного рядка. Використовувався для відладки програми.

3.4. Використаний приймач

Для приймання сигналу було використано RTL2832U.

Нижче наведена схема пристрою.

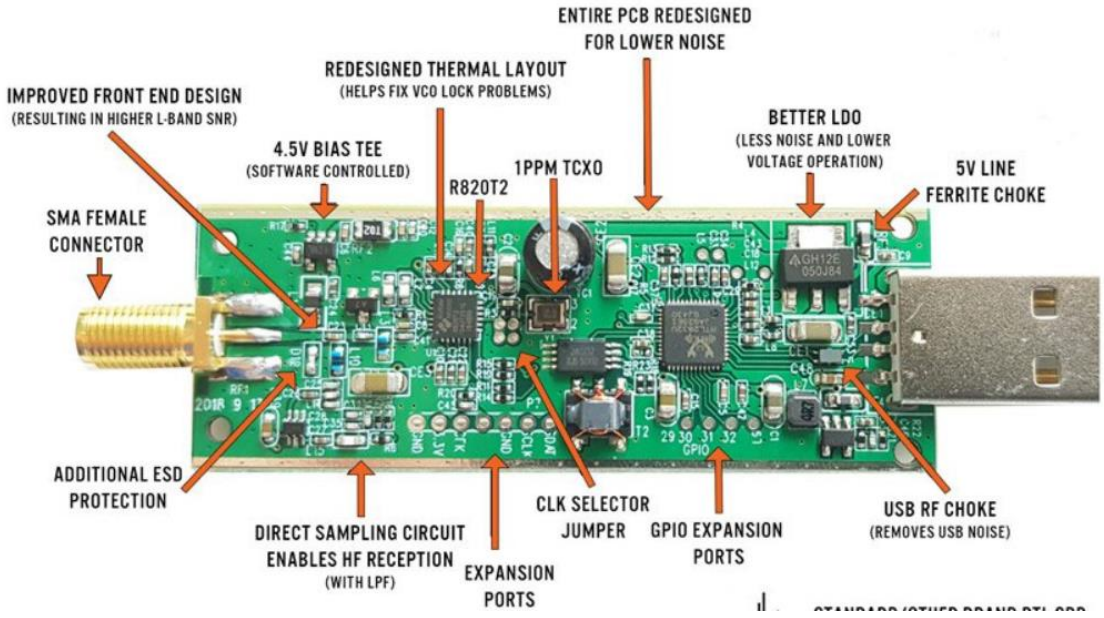

Рис.3.3. Схема RTL2832U

Основні параметри:

- Осцилятор з температурною компенсацією <1 *PPL* – точне налаштування та майже нульовий температурний дрейф (макс. початкове зсув 2 *PPM*, температурний дрейф 0,5-1 *PPM*)
- Роз'єм для антени SMA – SMA є поширеним і для нього доступно багато адаптерів і антен. Він також більш міцний.
- Живлення 4,5 *В* від USB – це дозволяє RTL-SDR живити малошумні підсилювачі (наприклад, LNA4ALL, HABAMP, RTL-SDR Blog ADS-B LNA) та активні антени через коаксіальний кабель. Можна ввімкнути програмно..
- Режим прямого дискретизації ВЧ - від 500 *кГц* до 24 *МГц* з прямою дискретизацією. Просто підключіть КВ-антену до порту SMA і виберіть режим Q-гілки. Існує вбудований фільтр нижніх частот 25 *МГц*, але для оптимальної продуктивності можуть знадобитися додаткові ВЧ фільтри (наприклад, для ослаблення сильного АММ).
- Алюмінієвий корпус і пасивне охолодження. Ці пристрої мають алюмінієвий корпус і пасивне охолодження за допомогою кремнієвої термопрокладки. Це зупиняє збій прийому через нагрівання при використанні вище ~1,2 *ГГц*.
- Покращені антени. Приймає наземні та супутникові сигнали.
- Різні додаткові вдосконалення в порівнянні з іншими RTL-SDR - тюнер R820T2, пасивні компоненти вищої якості, дросель на лінії USB для зменшення шуму USB, значно покращена конструкція друкованої плати для значно менше внутрішніх шпорів і шумів, різні контактні майданчики, покращений ESD захист, додаткові байпасні конденсатори та феритові дроселі лінії електропередачі, покращена схема узгодження переднього кінця, модифікована конструкція живлення для покращення довгострокової надійності та кращий LDO.

Висновки за розділом

В даному розділі описано програмну реалізацію алгоритму прийому сигналу по протоколу NEC. Описано 7 станів програми та їх обробники, а також

стан помилки та операції при виявленні помилок чи невірного сигналу. Описані додаткові функції програми, які дозволяють скоригувати дані.

Розглянуті основні бібліотеки, які використані в проекті серед яких: gnuradio, pynput, PyQt5 та argparse.

Описано будову та переваги пристрою RTL2832U, який приймає сигнал. Описано його технічні характеристики.

- Осцилятор з температурною компенсацією <1 *PPM* (*мг/л*)
- Роз’єм для антени SMA
- Режим прямої дискретизації ВЧ - від 500 *кГц* до 24 *МГц* з прямою дискретизацією
- Алюмінієвий корпус і пасивне охолодження.
- Покращені антени.
- Різні додаткові вдосконалення в порівнянні з іншими RTL-SDR

ВИСНОВКИ ЗА РОБОТОЮ

В даній дипломній роботі було реалізовано програмну систему для керування медіа-відтворенням на ПК з допомогою пульту.

Обґрунтовано доцільність використання SDR приймачів. Технології реалізації та сфери застосування даного обладнання. Основний принцип SDR технології, основні поняття радіотехніки та будову SDR ресівера. Виділено проект HackRF One, його основні характеристики та апаратні складові.

Основні характеристики HackRF One:

• Робоча частота від 1 *МГц* до 6 *ГГц*

• Напівдуплексний трансивер

• До 20 мільйонів зразків в секунду

• 8-бітові квадратурні вибірки (8-бітовий I і 8-бітовий Q)

• Сумісний з GNU Radio, SDR# тощо

• Програмне налаштування посилення RX і TX та фільтра основної смуги

• Програмно керована потужність порту антени (50 *мА* при 3,3 *В*)

• Роз'єм для антени SMA

• Синхронізації SMA по годиннику

• Зручні кнопки для програмування

• Внутрішні штифтові роз'єми для розширення

• Високошвидкісний USB 2.0

• Живлення від USB

• Апаратне забезпечення з відкритим кодом

Описано сфери застосування SDR та саму історію появи технології.

Виділено основні параметри SDR пристроїв.

Розглянуто методи виміру сигналів. Також було описано коротку історію виникнення GNU Radio та саму суть даного проекту. Описано протокол NEC, який використовується для захоплення сигналу, а також структурну схему роботи з сигналом. Показано оригінальний та результуючий сигнали.

Особливості формату NEC (японський варіант):

- 8 біт даних та 8 біт команда
- Адреса та команда передаються двічі для виключення помилки
- Метод кодування біт – інтервалами
- Несуча частота 38 *кГц*
- Час передачі біта 1.125 *мс* або 2.25 *мс*.

Зображена та описана структурна схема процесу.

Описано програмну реалізацію алгоритму прийому сигналу по протоколу NEC. Описано 7 станів програми та їх обробники, а також стан помилки та операції при виявленні помилок чи невірного сигналу. Описані додаткові функції програми, які дозволяють скоригувати дані.

Розглянуті основні бібліотеки, які використані в проекті.

Використано пристрій RTL2832U, який приймає сигнал. Описано його технічні характеристики, переваги та будову.

Основні технічні характеристики RTL2832U.

- Осцилятор з температурною компенсацією <1 PPM
- Роз’єм для антени SMA
- Режим прямої дискретизації ВЧ - від 500 *кГц* до 24 *МГц* з прямою дискретизацією
- Алюмінієвий корпус і пасивне охолодження.
- Покращені антени.

Різні додаткові вдосконалення в порівнянні з іншими RTL-SDR

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. S. Norsworthy, R. Schreier, G. Temes. Delta-Sigma Data Converters. ISBN 0-7803-1045-4.
2. Mingliang Liu. Demystifying Switched-Capacitor Circuits'. ISBN 0-7506-7907-7.
3. Behzad Razavi. Principles of Data Conversion System Design. ISBN 0-7803-1093-4.
4. David Johns, Ken Martin. Analog Integrated Circuit Design. ISBN 0-471-14448-7.
5. Phillip E. Allen, Douglas R. Holberg. CMOS Analog Circuit Design. ISBN 0-19-511644-5.
6. Котельников В. А. О пропускной способности эфира и проволоки в электросвязи — Всесоюзный энергетический комитет. // Материалы к I Всесоюзному съезду по вопросам технической реконструкции дела связи и развития слаботочной промышленности, 1933. Репринт статьи в журнале УФН, 176:7 (2006), 762—770.
7. Software defined radio : architectures, systems, and functions. Dillinger, Madani, Alonistioti. Wiley, 2003. 454 pages. ISBN 0-470-85164-3 ISBN 9780470851647
8. Cognitive Radio Technology. Bruce Fette. Elsevier Science & Technology Books, 2006. 656 pags. ISBN 0-7506-7952-2 ISBN 9780750679527
9. Software Defined Radio for 3G, Burns. Artech House, 2002. ISBN 1-58053-347-7
10. Software Radio: A Modern Approach to Radio Engineering, Jeffrey H. Reed. Prentice Hall PTR, 2002. ISBN 0-13-081158-0

11. Signal Processing Techniques for Software Radio, Behrouz Farhang-Beroujeny. LuLu Press.

12. RF and Baseband Techniques for Software Defined Radio, Peter B. Kenington. Artech House, 2005, ISBN 1-58053-793-6

13. The ABC's of Software Defined Radio, Martin Ewing, AA6E. The American Radio Relay League, Inc., 2012, ISBN 978-0-87259-632-0

14. Software defined radio : architectures, systems, and functions. Dillinger, Madani, Alonistioti. Wiley, 2003. 454 pages. ISBN 0-470-85164-3 ISBN 9780470851647

15. Cognitive Radio Technology. Bruce Fette. Elsevier Science & Technology Books, 2006. 656 pags. ISBN 0-7506-7952-2 ISBN 9780750679527

16. Software Defined Radio for 3G, Burns. Artech House, 2002. ISBN 1-58053-347-7

17. Software Radio: A Modern Approach to Radio Engineering, Jeffrey H. Reed. Prentice Hall PTR, 2002. ISBN 0-13-081158-0

18. Signal Processing Techniques for Software Radio, Behrouz Farhang-Beroujeny. LuLu Press.

19. RF and Baseband Techniques for Software Defined Radio, Peter B. Kenington. Artech House, 2005, ISBN 1-58053-793-6

20. The ABC's of Software Defined Radio, Martin Ewing, AA6E. The American Radio Relay League, Inc., 2012, ISBN 978-0-87259-632-0

ДОДАТОК А. КОД ПРОГРАМИ

```
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first class it finds
to get ports and parameters of your block. The arguments to __init__  will
be the parameters. All of them are required to have default values!
"""

import numpy as np
from gnuradio import gr

from pynput.keyboard import Key,Controller
keyboard = Controller()



def set_bit(value, bit):
    return value | (1<<bit)
def clear_bit(value, bit):
    return value & ~(1<<bit)

def command(comm):
    if comm == 7:
        keyboard.tap(Key.media_volume_down)
    elif comm == 21:
        keyboard.tap(Key.media_volume_up)
    elif comm == 68:
```

```
            keyboard.tap(Key.media_previous)
        elif comm == 64:
            keyboard.tap(Key.media_next)
        elif comm == 67:
            keyboard.tap(Key.media_play_pause)

    class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
        """Embedded Python Block example - a simple multiply const"""

        def __init__(self, samp_rate=48000):  # only default arguments here
            """arguments to this function show up as parameters in GRC"""
            gr.sync_block.__init__(
                self,
                name='NEC Decoder',   # will show up in GRC
                in_sig=[np.int8],
                out_sig=[]
            )
            # if an attribute with the same name as a parameter is found,
            # a callback is registered (properties work, too).
            self.samp_rate = samp_rate
            self.period = 1/samp_rate
            self.state = 0
            self.bits = 0
            self.count = 0
            self.data = []
            self.adr = 0
            self.comm = 0
```

```
def convert_bits(self):
    lst = self.data
    adr = 0
    adr1 = 0
    comm = 0
    comm1 = 0
    j = 0
    for i in range(0, 8):
        if lst[i] == 0:
            adr = clear_bit(adr, j)
        if lst[i] == 1:
            adr = set_bit(adr, j)
        j += 1
    j = 0
    for i in range(8, 16):
        if lst[i] == 1:
            adr1 = clear_bit(adr1, j)
        if lst[i] == 0:
            adr1 = set_bit(adr1, j)
        j += 1
    j = 0
    for i in range(16, 24):
        if lst[i] == 0:
            comm = clear_bit(comm, j)
        if lst[i] == 1:
            comm = set_bit(comm, j)
        j += 1
    j = 0
    for i in range(24, 32):
        if lst[i] == 1:
```

```
            comm1 = clear_bit(comm1, j)
        if lst[i] == 0:
            comm1 = set_bit(comm1, j)
        j += 1

    if adr == adr1 and comm == comm1:
        self.adr = adr
        self.comm = comm
    else:
        print("\n\n", adr, "     ", comm, "\n\n")
        print("\n\n", adr1, "     ", comm1, "\n\n")
        print("Bit error")

def work(self, input_items, output_items):
    """example: multiply with constant"""
    #output_items[0][:] = input_items[0] * self.example_param

    for i in input_items[0]:
        if self.state == 0:
            if i == 1:
                self.state = 1
                #print("State 1")
                self.count += 1
                continue

        if self.state == 1:
            if i == 0:
                #print(self.period*self.count)
                if self.period*self.count > 0.007 and self.period*self.count < 0.009:
                    self.state = 3
                    #print("State 3")
```

```
                    self.count = 0
                    continue
                else:
                    print("Error1")
                    print(self.period*self.count)
                    self.state = "Error"

            self.count += 1



        if self.state == 3:
            if i == 1:
                #print(self.period*self.count)
                if self.period*self.count > 0.0035 and self.period*self.count <
0.0055:
                    self.state = 4
                    #print("State 4")
                    self.count = 0
                    continue
                if self.period*self.count > 0.0017 and self.period*self.count <
0.0028:
                    self.state = 7
                    #print("State 7")
                    self.count = 0
                    continue
                else:
                    print("Error3")
                    print(self.period*self.count)
                    self.state = "Error"

            self.count += 1
```

```
            if self.state == 4:
                if i == 0:
                    #print(self.period*self.count)
                    if self.period*self.count > 0.00035 and self.period*self.count < 0.00073:
                        self.state = 5
                        #print("State 5")
                        #self.count = 0
                        continue
                    else:
                        print("Error4")
                        print(self.period*self.count)
                        self.state = "Error"

                self.count += 1

            if self.state == 5:
                if i == 1:
                    #print(self.period*self.count)
                    if self.period*self.count > 0.002 and self.period*self.count < 0.003:
                        self.data.append(1)
                        self.bits += 1

                        self.count = 0

                        if self.bits == 32:
                            #print("State 6")
                            self.state = 6
                            continue
```

```
                    self.state = 4
                    #print("State 4")
                    continue
                elif self.period*self.count > 0.0008 and self.period*self.count <
0.0016:
                    self.data.append(0)
                    self.bits += 1

                    self.count = 0

                    if self.bits == 32:
                        #print("State 6")
                        self.state = 6
                        continue

                    self.state = 4
                    #print("State 4")
                    continue
                else:
                    print("Error5")
                    print(self.period*self.count)
                    self.state = "Error"

            self.count += 1

        if self.state == 6:
            if i == 0:
                #print(self.period*self.count)
                if self.period*self.count > 0.00035 and self.period*self.count <
0.00073:
                    self.state = 0
```

```
                    #print("State 0")
                    self.count = 0
                    self.bits = 0
                    self.convert_bits()
                    self.data.clear()
                    print(self.comm, "\n")
                    command(self.comm)
                    continue
                else:
                    print("Error6")
                    print(self.period*self.count)
                    self.state = "Error"
            self.count += 1

        if self.state == 7:
            if i == 0:
                #print(self.period*self.count)
                if self.period*self.count > 0.00035 and self.period*self.count < 0.00073:
                    self.state = 0
                    print("Repeat: ", self.comm, "\n")
                    command(self.comm)
                    continue
                else:
                    print("Error7")
                    print(self.period*self.count)
                    self.state = "Error"
            self.count += 1

        if self.state == "Error":
            if self.period*self.count > 0.080:
```

```
                    self.state = 0
                    self.count = 0
                    self.bits = 0
                    self.data.clear()
                self.count += 1



        return len(input_items[0])
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#
# SPDX-License-Identifier: GPL-3.0
#
# GNU Radio Python Flow Graph
# Title: Not titled yet
# GNU Radio version: 3.9.4.0

from distutils.version import StrictVersion

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print("Warning: failed to XInitThreads()")
```

```
from PyQt5 import Qt
from gnuradio import qtgui
from gnuradio.filter import firdes
import sip
from gnuradio import analog
from gnuradio import blocks
from gnuradio import digital
from gnuradio import filter
from gnuradio import gr
from gnuradio.fft import window
import sys
import signal
from argparse import ArgumentParser
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
import NEC_epy_block_1 as epy_block_1  # embedded python block
import osmosdr
import time



from gnuradio import qtgui

class NEC(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Not titled yet", catch_exceptions=True)
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Not titled yet")
        qtgui.util.check_set_qss()
        try:
```

```
    self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
except:
    pass
self.top_scroll_layout = Qt.QVBoxLayout()
self.setLayout(self.top_scroll_layout)
self.top_scroll = Qt.QScrollArea()
self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
self.top_scroll_layout.addWidget(self.top_scroll)
self.top_scroll.setWidgetResizable(True)
self.top_widget = Qt.QWidget()
self.top_scroll.setWidget(self.top_widget)
self.top_layout = Qt.QVBoxLayout(self.top_widget)
self.top_grid_layout = Qt.QGridLayout()
self.top_layout.addLayout(self.top_grid_layout)

self.settings = Qt.QSettings("GNU Radio", "NEC")

try:
    if StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
        self.restoreGeometry(self.settings.value("geometry").toByteArray())
    else:
        self.restoreGeometry(self.settings.value("geometry"))
except:
    pass

##################################################
# Variables
##################################################
self.samp_rate = samp_rate = 2400000

##################################################
```

```
# Blocks
##################################################
self.rtlsdr_source_0 = osmosdr.source(
    args="numchan=" + str(1) + " " + 'rtl=0'
)
self.rtlsdr_source_0.set_time_unknown_pps(osmosdr.time_spec_t())
self.rtlsdr_source_0.set_sample_rate(samp_rate)
self.rtlsdr_source_0.set_center_freq(266e5, 0)
self.rtlsdr_source_0.set_freq_corr(0, 0)
self.rtlsdr_source_0.set_dc_offset_mode(0, 0)
self.rtlsdr_source_0.set_iq_balance_mode(0, 0)
self.rtlsdr_source_0.set_gain_mode(False, 0)
self.rtlsdr_source_0.set_gain(10, 0)
self.rtlsdr_source_0.set_if_gain(20, 0)
self.rtlsdr_source_0.set_bb_gain(20, 0)
self.rtlsdr_source_0.set_antenna('', 0)
self.rtlsdr_source_0.set_bandwidth(0, 0)
self.qtgui_time_sink_x_1 = qtgui.time_sink_f(
    3500, #size
    48000, #samp_rate
    "", #name
    1, #number of inputs
    None # parent
)
self.qtgui_time_sink_x_1.set_update_time(0.10)
self.qtgui_time_sink_x_1.set_y_axis(-1, 1)

self.qtgui_time_sink_x_1.set_y_label('Amplitude', "")

self.qtgui_time_sink_x_1.enable_tags(True)
```

```
        self.qtgui_time_sink_x_1.set_trigger_mode(qtgui.TRIG_MODE_NORM,
qtgui.TRIG_SLOPE_POS, 0, 0, 0, "")
        self.qtgui_time_sink_x_1.enable_autoscale(True)
        self.qtgui_time_sink_x_1.enable_grid(False)
        self.qtgui_time_sink_x_1.enable_axis_labels(True)
        self.qtgui_time_sink_x_1.enable_control_panel(False)
        self.qtgui_time_sink_x_1.enable_stem_plot(False)


        labels = ['Signal 1', 'Signal 2', 'Signal 3', 'Signal 4', 'Signal 5',
            'Signal 6', 'Signal 7', 'Signal 8', 'Signal 9', 'Signal 10']
        widths = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        colors = ['blue', 'red', 'green', 'black', 'cyan',
            'magenta', 'yellow', 'dark red', 'dark green', 'dark blue']
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
            1.0, 1.0, 1.0, 1.0, 1.0]
        styles = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        markers = [-1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1]


        for i in range(1):
            if len(labels[i]) == 0:
                self.qtgui_time_sink_x_1.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_time_sink_x_1.set_line_label(i, labels[i])
            self.qtgui_time_sink_x_1.set_line_width(i, widths[i])
            self.qtgui_time_sink_x_1.set_line_color(i, colors[i])
            self.qtgui_time_sink_x_1.set_line_style(i, styles[i])
```

```
            self.qtgui_time_sink_x_1.set_line_marker(i, markers[i])
            self.qtgui_time_sink_x_1.set_line_alpha(i, alphas[i])

        self._qtgui_time_sink_x_1_win =
sip.wrapinstance(self.qtgui_time_sink_x_1.qwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_time_sink_x_1_win)
        self.qtgui_time_sink_x_0 = qtgui.time_sink_f(
            3500, #size
            48000, #samp_rate
            "", #name
            1, #number of inputs
            None # parent
        )
        self.qtgui_time_sink_x_0.set_update_time(0.10)
        self.qtgui_time_sink_x_0.set_y_axis(-1, 1)

        self.qtgui_time_sink_x_0.set_y_label('Amplitude', "")

        self.qtgui_time_sink_x_0.enable_tags(True)
        self.qtgui_time_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_NORM,
qtgui.TRIG_SLOPE_POS, 2.0, 0, 0, "")
        self.qtgui_time_sink_x_0.enable_autoscale(True)
        self.qtgui_time_sink_x_0.enable_grid(False)
        self.qtgui_time_sink_x_0.enable_axis_labels(True)
        self.qtgui_time_sink_x_0.enable_control_panel(False)
        self.qtgui_time_sink_x_0.enable_stem_plot(False)


        labels = ['Signal 1', 'Signal 2', 'Signal 3', 'Signal 4', 'Signal 5',
            'Signal 6', 'Signal 7', 'Signal 8', 'Signal 9', 'Signal 10']
        widths = [1, 1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1]
        colors = ['blue', 'red', 'green', 'black', 'cyan',
            'magenta', 'yellow', 'dark red', 'dark green', 'dark blue']
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
            1.0, 1.0, 1.0, 1.0, 1.0]
        styles = [1, 1, 1, 1, 1,
            1, 1, 1, 1, 1]
        markers = [-1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1]



        for i in range(1):
            if len(labels[i]) == 0:
                self.qtgui_time_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_time_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_time_sink_x_0.set_line_width(i, widths[i])
            self.qtgui_time_sink_x_0.set_line_color(i, colors[i])
            self.qtgui_time_sink_x_0.set_line_style(i, styles[i])
            self.qtgui_time_sink_x_0.set_line_marker(i, markers[i])
            self.qtgui_time_sink_x_0.set_line_alpha(i, alphas[i])

        self._qtgui_time_sink_x_0_win =
sip.wrapinstance(self.qtgui_time_sink_x_0.qwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_time_sink_x_0_win)
        self.low_pass_filter_0 = filter.fir_filter_ccf(
            50,
            firdes.low_pass(
                50,
                samp_rate,
                200000,
```

```
            500,
            window.WIN_HAMMING,
            6.76))
        self.epy_block_1 = epy_block_1.blk(samp_rate=48000)
        self.digital_binary_slicer_fb_0 = digital.binary_slicer_fb()
        self.blocks_threshold_ff_1 = blocks.threshold_ff(1.5, 4, 0)
        self.blocks_add_const_vxx_0 = blocks.add_const_ff(-0.5)
        self.analog_am_demod_cf_0 = analog.am_demod_cf(
            channel_rate=48000,
            audio_decim=1,
            audio_pass=4000,
            audio_stop=5000,
        )



        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_am_demod_cf_0, 0), (self.blocks_threshold_ff_1,
0))
        self.connect((self.analog_am_demod_cf_0, 0), (self.qtgui_time_sink_x_0,
0))
        self.connect((self.blocks_add_const_vxx_0,                          0),
(self.digital_binary_slicer_fb_0, 0))
        self.connect((self.blocks_add_const_vxx_0, 0), (self.qtgui_time_sink_x_1,
0))
        self.connect((self.blocks_threshold_ff_1, 0), (self.blocks_add_const_vxx_0,
0))
        self.connect((self.digital_binary_slicer_fb_0, 0), (self.epy_block_1, 0))
        self.connect((self.low_pass_filter_0, 0), (self.analog_am_demod_cf_0, 0))
```

```
            self.connect((self.rtlsdr_source_0, 0), (self.low_pass_filter_0, 0))


        def closeEvent(self, event):
            self.settings = Qt.QSettings("GNU Radio", "NEC")
            self.settings.setValue("geometry", self.saveGeometry())
            self.stop()
            self.wait()

            event.accept()

        def get_samp_rate(self):
            return self.samp_rate

        def set_samp_rate(self, samp_rate):
            self.samp_rate = samp_rate
            self.low_pass_filter_0.set_taps(firdes.low_pass(50, self.samp_rate, 200000,
500, window.WIN_HAMMING, 6.76))
            self.rtlsdr_source_0.set_sample_rate(self.samp_rate)

    def main(top_block_cls=NEC, options=None):

        if StrictVersion("4.5.0") <= StrictVersion(Qt.qVersion()) <
StrictVersion("5.0.0"):
            style = gr.prefs().get_string('qtgui', 'style', 'raster')
            Qt.QApplication.setGraphicsSystem(style)
        qapp = Qt.QApplication(sys.argv)

        tb = top_block_cls()

        tb.start()
```

```
    tb.show()

    def sig_handler(sig=None, frame=None):
        tb.stop()
        tb.wait()

        Qt.QApplication.quit()

    signal.signal(signal.SIGINT, sig_handler)
    signal.signal(signal.SIGTERM, sig_handler)

    timer = Qt.QTimer()
    timer.start(500)
    timer.timeout.connect(lambda: None)

    qapp.exec_()

if __name__ == '__main__':
    main()
```