1. Зміст

[1 Зміст 1](#_Toc406595090)

[2 Вступ 3](#_Toc406595091)

[3 Теорія 4](#_Toc406595092)

[3.1 Використані бібліотеки 4](#_Toc406595093)

[3.1.1 javahidapi 4](#_Toc406595094)

[3.1.2 Google Gson 5](#_Toc406595095)

[3.1.3 Apache Commons Lang 6](#_Toc406595096)

[3.1.4 Apache Commons Collections 7](#_Toc406595097)

[3.1.5 ControlsFX 7](#_Toc406595098)

[3.1.6 jSSC 8](#_Toc406595099)

[3.1.7 Reflections 8](#_Toc406595100)

[3.1.8 usb4java 10](#_Toc406595101)

[4 Практика 11](#_Toc406595102)

[4.1 Пошук пристроїв 11](#_Toc406595103)

[4.2 Абстрактний клас Device 11](#_Toc406595104)

[4.3 Взаємодія з пристроєм 11](#_Toc406595105)

[4.3.1 Прийняття даних з пристрою та генерування пакету 11](#_Toc406595106)

[4.4 Логування 12](#_Toc406595107)

[4.5 Часова шкала, програвання збережених записів 12](#_Toc406595108)

[4.6 Використання вбудованого Wi-Fi адаптеру для сканування каналів 12](#_Toc406595109)

[4.7 Графічний інтерфейс 12](#_Toc406595110)

[4.7.1 Легенда 12](#_Toc406595111)

[4.7.2 Меню налаштувань 12](#_Toc406595112)

[4.8 Допоміжні класи 12](#_Toc406595113)

[4.9 MetaGeek Wi-Spy Gen 1 12](#_Toc406595114)

[4.9.1 Використання високорівневих функцій бібліотеки usb4java 13](#_Toc406595115)

[4.9.2 Використання низькорівневих функцій бібліотеки usb4java 14](#_Toc406595116)

[4.9.3 Використання JNI 14](#_Toc406595117)

[4.9.4 Розбір даних з пристрою 15](#_Toc406595118)

[4.10 MetaGeek Wi-Spy 2.4x2 16](#_Toc406595119)

[4.10.1 Ініціалізація 16](#_Toc406595120)

[4.10.2 Розбір даних з пристрою 18](#_Toc406595121)

[4.11 Texas Instruments ez430-RF2500 18](#_Toc406595122)

[4.11.1 Пошук потрібних регістрів та встановлення коректних параметрів 21](#_Toc406595123)

[4.11.2 Підключення до MDRV 25](#_Toc406595124)

[4.12 Air View 2 25](#_Toc406595125)

[4.12.1 Ініціалізація 25](#_Toc406595126)

[4.12.2 Розбір даних з пристрою 26](#_Toc406595127)

[4.13 Unigen ISM Sniffer 26](#_Toc406595128)

[4.14 Wixel Pololu 26](#_Toc406595129)

[5 Список літератури 28](#_Toc406595130)

1. Вступ
2. Теорія
   1. Використані бібліотеки
      1. javahidapi

Java HID API є JNI, що дозволяє використовувати бібліотеку HIDAPI з Java коду. HIDAPI є мультиплатформенною бібліотекою, яка дозволяє додатку взаємодіяти з USB та Bluetooth пристроями HID-класу під Windows, Linux і Mac OS X.

Після створення екземпляра HIDManager ви можете використовувати деякі з його методів. Метод listDevices повертає список HID пристроїв, які активні в даний момент часу. Кожен пристрій представлено екземпляром класу HIDDeviceInfo, який містить інформацію про пристрої. Щоб відкрити пристрій потрібно викликати метод Open.

HIDManager також надає кілька зручних методів для швидкого пошуку і відкриття пристрою або шляху (openByPath) або через vendor id/product id/serial number (openById).

Кожний відкритий пристрій представлено класом HIDDevice. Якщо пристрій відкрито кілька разів - HIDDevice буде однаковий, але безпеки потоків не гарантується.

HIDAPI може бути використаний в рамках однієї з трьох ліцензій:

* GNU Public License, версія 3.0
* BSD-стиль ліцензії
* Оригінал ліцензії HIDAPI

Ліцензія обирається на розсуд користувача JavaHIDAPI. Наприклад:

* Автор програми, який використовує GPL зможе використати JavaHIDAPI під умовами GPL.
* Автор комерційного програмного забезпечення з закритим вихідним кодом, ймовірно, використовуватиме JavaHIDAPI відповідно до умов ліцензії BSD-стилю або оригінал ліцензії JavaHIDAPI.
  + 1. Google Gson

Gson це бібліотека для перетворення об'єктів Java у формат JSON. Вона також може бути використана для перетворення рядка JSON до еквівалентного об'єкта Java. Gson може працювати з довільними Java об'єктами, включаючи вже існуючі об'єкти, на які немає сирцевого коду.

Переваги Gson:

* Забезпечує прості методи toJson і fromJson для перетворення Java об'єктів в JSON і навпаки
* Вже існуючі об'єкти можуть бути перетворені в та з JSON
* Розширена підтримка Java Generics
* Користувацькі подання для об'єктів

В даному проекті використовується для підтримки функціонування функцій налаштувань програми та зберігання та програвання збережених треків.

* + 1. Apache Commons Lang

Стандартні Java бібліотеки не в змозі забезпечити достатньо методів для маніпулювання основними класами. Apache Commons Lang надає ці додаткові методи.

Lang надає безліч допоміжних утиліт для java.lang API, зокрема методів маніпуляції String, основні чисельні методи, reflection, concurrency, створення і серіалізация властивостей системи. Крім того, він містить основні вдосконалення java.util.Date.

В даному проекті ця бібліотека використовується для біль зручного визначення ОС на якій запущена програма (клас org.apache.commons.lang3. SystemUtils).

* + 1. Apache Commons Collections

Java Collections Framework був важливим доповненням в JDK 1.2. Він додав, багато потужних структур даних, які прискорюють розробку найбільш значущих Java-додатків. Commons-Collections розвинули класи JDK шляхом надання нових інтерфейсів, реалізацій і утиліт. Надається під Apache License.

В даному проекті ця бібліотека використовується клас пакету BidiMap – представляє собою мапу, ключ якої, може буди використаний як значення, а значення як ключ.

* + 1. ControlsFX

ControlsFX є проект з відкритим кодом для JavaFX, яка покликана забезпечити дійсно високоякісний UI та інші інструменти, які доповнюють JavaFX. Бібліотека має JavaDoc документацію високої якості. Надається під BSD 3-Clause License.

В даному проекті ця бібліотека використовується як зручний засіб відображення діалогових вікон.

* + 1. jSSC

jSSC (Java Simple Serial Connector) - бібліотека для роботи з COM портами з Java. jSSC підтримує Win32 (Win98-Win8), Win64, Linux (x86, x86-64, ARM), Solaris (x86, x86-64), Mac OS X 10.5 і вище (x86, x86-64, PPC, PPC64). Надається під GNU Lesser GPL.

В даному проекті ця бібліотека використовується як основний засіб для взаємодії з COM пристроями (Wixel Pololu, Texas Instruments ez430RF2500, Air View 2).

* + 1. Reflections

В інформатиці відбиття або означає процес, під час якого програма може відстежувати і модифікувати власну структуру і поведінку під час виконання. Парадигма програмування, покладена в основу відображення, називається рефлексивним програмуванням. Це один з видів метапрограмування.

У більшості сучасних комп'ютерних архітектур програмні інструкції (код) зберігаються як дані. Різниця між кодом і даними в тому, що виконуючи код, комп'ютери обробляють дані. Тобто інструкції виконуються, а дані обробляються так, як написано цими інструкціями. Однак програми, написані за допомогою деяких мов, здатні обробляти власні інструкції як дані і виконувати, таким чином, рефлексивні модифікації. Такі самомодифікуючі програми в основному створюються за допомогою високорівневих мов програмування, що використовують віртуальні машини (наприклад, Smalltalk, скриптові мови).

Reflections сканує директорію класів, індексує метадані та дозволяє получати доступ до них під час виконання програми.

Використовуючи Reflections ви можете запросити такі метадані:

* підтипи певного типу
* типи / constructos / методи / поля з анотацією
* отримати всі погодження ресурсів відповідні регулярному виразу
* отримати всі методи з конкретною сигнатурою, параметрами та типом повернення

В даному проекті ця бібліотека використовується для знаходження всіх підкласів класу Device, тим самим генеруючи список всіх підтримуваних профілів пристроїв.

* + 1. usb4java

usb4java це бібліотека Java для доступу до USB-пристроїв. Вона заснована на native libusb 1.0 і використовує Java NIO буфери для обміну даними між libusb і Java. usb4java також підтримує стандарт javax-USB (JSR-80) через розширення usb4java-javax.

Підтримує платформи Linux (x86 32/64 біт, ARM 32 біт), OS X (x86 32/64 біт) і Windows (тільки x86 32/64 біт). Але інші платформи можуть працювати так само добре (якщо є Java 6 і підтримуються libusb) шляхом компіляції бібліотеки JNI вручну.

В даному проекті бібліотека використовується у якості допоміжної.

1. Практика
   1. Пошук пристроїв

DeviceConnectionListener – клас, задача якого сканувати систему на предмет підключень пристроїв через задані проміжки часу (за замовчуванням 1 секунда).

Клас реалізує шаблон програмування Singleton тому, що немає сенсу запускати в одній програмі декілька екземплярів цього класу. Також реалізує шаблон програмування Observer, за допомогою якого відбувається нотифікація підписчиків на подію підключення пристрою. Клас DeviceConnectionHandler підписується на подію підключення пристрою, викликає статичний метод для пошуку конкретного пристрою, яке підтримує програма, та, у випадку успіху, запускає потік для роботи з пристроєм.

* 1. Абстрактний клас Device та його реалізації

Device – клас, який абстрагує загальну поведінку для всіх пристроїв.

* + 1. Статичний фабричний метод

Звичайний спосіб отримання екземпляру класу - відкритий конструктор. Існує ще один метод - клас може забезпечити статичний фабричний метод, який повертає екземпляр класу.

Однією з переваг статичних фабричних методів є те, що на відміну від конструкторів, вони мають імена. Друга перевага статичних фабричних методів - вони не зобов'язані створювати новий об'єкт при виклику. Третя перевага статичних фабричних методів є те, що на відміну від конструкторів, вони можуть повернути об'єкт будь-якого підтипу.

У класі Device використовується статичний фабричний метод за наступною сигнатурою:

public static Device getConcreteDevice(DeviceInfo deviceInfo)

Клас DeviceConnectionHandler викликає фабричний метод getConcreteDevice з параметром DeviceInfo в якому знаходиться інформація о пристрої.

Задача методу getConcreteDevice – на основі даних з DeviceInfo повернути ініціалізований конкретний екземпляр класу пристрою. Це досягається за допомогою спеціального механізму – Reflection. Використовуючи Reflection реалізовано пошук конкретного класу по всім можливим нащадкам абстрактного класу Device. Такий підхід дає можливість додавати підтримку реалізацій нових пристроїв не чіпаючи при цьому інші класи.

* + 1. Конкретні реалізації

Для реалізації конкретного пристрою треба заповнити шаблон DeviceTemplate:

public class DeviceTemplate extends Device

{

public final static String FRIENDLY\_NAME = "";

public final static String VENDOR\_ID = "";

public final static String PRODUCT\_ID = "";

public final static float INITIAL\_FREQUENCY = 2400f;

public final static float CHANNEL\_SPACING = 0f;

public final static byte[] END\_PACKET\_SEQUENCE = new byte[]{};

public final static boolean MANUAL\_DEVICE\_CONTROL = false;

@Override

public void initializeDevice()

{ }

@Override

public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)

{

ArrayList<Byte> finalArray = new ArrayList<>(dataToParse);

return finalArray;

}

@Override

public byte[] customReadMethod()

{

return new byte[0];

}

}

Поле FRIENDLY\_NAME використовується для ідентифікації пристрою у графічному інтерфейсі.

Поле VENDOR\_ID використовується для зберігання ідентифікатору виробника пристрою. Повинен бути у шістнадцятковій системі числення. На приклад "1FFB".

Поле PRODUCT\_ID використовується для зберігання ідентифікатору пристрою. Повинен бути у шістнадцятковій системі числення.

Поле INITIAL\_FREQUENCY використовується для зберігання мінімальної частоти, яку пристрій може бачити (Base Frequency). Значення береться з документації к пристрою. На приклад 2400f.

Поле CHANNEL\_SPACING використовується для визначення між каналами. Значення береться з документації к пристрою. На приклад 327.450980f.

Поле END\_PACKET\_SEQUENCE використовується для зберігання символів кінця пакету. Символи кінця пакету використовуються класом RxRawDataReceiver для генерування пакетів зі значеннями RSSI.

Поле MANUAL\_DEVICE\_CONTROL встановлюється, коли треба отримати прямий контроль на пристроєм USBHID. Це потрібно у ситуаціях, коли бібліотека за замовчуванням не спрацьовує за якихось причин. У цьому режимі програма не відкриватиме та не робитиме спроб зчитати з пристрою. Метод customReadMethod активується. Приклад – клас MetaGeekWiSpyGen1.

Метод initializeDevice з наступною сигнатурою:

public void initializeDevice()

використовується для ініціалізації пристрою, якщо це потрібно. Інакше тіло методу можна залишити пустим.

Метод parse з наступною сигнатурою:

public ArrayList<Byte> parse(ArrayList<Byte> dataToParse)

використовується для розбору даних, які були сформовані пристроєм. Формат повернення – масив байтів, кожен елемент якого – значення RSSI.

Метод customReadMethod з наступною сигнатурою:

public byte[] customReadMethod()

{

return new byte[0];

}

використовується для пере визначення стандартної поведінки USBHID пристроїв. Для використання цього методу треба встановити прапорець MANUAL\_DEVICE\_CONTROL.

У шаблоні DeviceTemplate реалізована JavaDoc документація за допомогою якої кожен може додати в програму підтримку свого пристрою.

* 1. Взаємодія з пристроєм
     1. Прийняття даних з пристрою та генерування пакету
  2. Реєстрація повідомлень

Для реєстрації повідомлень використовується вбудований клас java.util.logging.Logger пакету java.util.logging. Об'єкт Logger використовується для запису повідомлень для конкретної системи або компонента програми. Logger, як правило, використовує ієрархічну структуру імен. Імена можуть бути довільними, як правило, встановлюються на основі імені пакета чи імені класу. Крім того, можна створювати "анонімний" Logger, що не зберігаються в просторі імен реєстратора.

Об'єкти Logger може бути отримана шляхом виклику getLogger factory methods, який створить новий Logger або поверне підходящий існуючуй. Важливо відзначити, що Logger повернений одним з методів getLogger може бути знищений при збірці сміття в будь-який момент, якщо на нього немає посилання.

Повідомлення пересилаються на зареєстровані оброблювачі, які можуть пересилати повідомлення в різні напрямки, в тому числі консолі, файли, журнали ОС і т.д.

Реєстратор повідомлень підтримує декілька рівнів повідомлень: OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST.

В даному проекті за реєстрацію повідомлень відповідає клас ApplicationLogger. Приклад використання:

ApplicationLogger.setup();

ApplicationLogger.LOGGER.info("MetaGeek\_WiSpy24x2 has been initialized");

Метод ApplicationLogger.setup викликається один раз при старті програми для ініціалізації об’єкту реєстратора повідомлень та створення файлу повідомлень. У класі MyLogFormatter, який розширює клас Formatter, вказано формат виводу за допомогою пере визначення наступної функції, яка викликається для кожної реєстрації повідомлення:

@Override

public String format(LogRecord record)

Можливий варіант зареєстрованих повідомлень виглядає так:

16.12.14 12:55:48.523 [INFO] [logging.ApplicationLogger.setup] Logger has initialized

16.12.14 12:55:48.572 [INFO] [updater.UpdateChecker.initCurrentVersion] Current version: Version{major=0, minor=3}

16.12.14 12:55:49.385 [INFO] [connectionlistener.DeviceConnectionListener.runSchedule] Listening schedule has started. Waiting for devices...

16.12.14 12:55:49.397 [WARNING] [wirelessadapter.WirelessAdapterCommunication.searchWirelessAdapters] phy#0

16.12.14 12:55:49.418 [INFO] [connectionlistener.DeviceConnectionListener.deviceConnectionEvent] USB Device CONNECTED

16.12.14 12:55:49.448 [INFO] [wirelessadapter.WirelessAdapter.setUpChannelToFrequencyMap] channelToFrequencyMap: {1=2412, 2=2417, 3=2422, 4=2427, 5=2432, 6=2437, 7=2442, 8=2447, 9=2452, 10=2457, 11=2462, 12=2467, 13=2472, 14=2484}

Формат виводу наступний:

[Date] [Time] [Logging Level] [The Class from which logger has been called] [Message]

* 1. Часова шкала, програвання збережених записів
  2. Використання вбудованого Wi-Fi адаптеру для сканування каналів
  3. Графічний інтерфейс
     1. Легенда
     2. Меню налаштувань
  4. Допоміжні класи

В програмі використовуються спеціальні класи, в яких сформовані статичні методи. Ці методи використовуються у програмі у різних місцях. Класи можна знайти у пакеті com.rasalhague.mdrv.Utility. На приклад наступна функція повертає псевдовипадкове число типу int з заданого діапазону:

public static int randInt(int min, int max)

Для приведення Pid чи Vid пристрою до чотирьох символів використовується функція

public static String normalizePidVidToLength(String str)

Щоб виклику Runnable в JFX Thread і чекати доки він не завершиться можна використати функцію

public static void runAndWait(final Runnable run) throws InterruptedException, ExecutionException

Для завантаження Native бібліотек використано функцію, яка розпаковує обрану бібліотеку до тимчасової директорії.

public static void loadLibraryFromJar(String path) throws IOException

* 1. MetaGeek Wi-Spy Gen 1

MetaGeek Wi-Spy Gen 1 як і вся лінійка пристроїв MetaGeek Wi-Spy підключається до операційної системи як HID пристрій. Підключенні даного пристрою до програмного комплексу MDRV за допомогою стандартної бібліотеки JavaHIDAPI не вдалося ні під linux ні під Windows. Ось так виглядає виключення:

java.io.IOException: The supplied user buffer is not valid for the requested operation.

at com.codeminders.hidapi.HIDDevice.read(Native Method)

З коду помилки видно, що функція читання, яка являє собою Native Method, вважає переданий їй буфер, в який треба зчитати данні, не дійснім.

Далі наведено шляхи вирішення проблеми.

* + 1. Використання високорівневих функцій бібліотеки usb4java

У різних бібліотеках реалізації можуть відрізнятися, тому було вирішено спробувати бібліотеку usb4java. Але при спробі виконати читання даних:

int received = pipe.syncSubmit(data);

програма зависає.

При спробі відіслати на пристрій IRP пакет:

irp = device.createUsbControlIrp((byte) (UsbConst.REQUESTTYPE\_DIRECTION\_IN |

UsbConst.REQUESTTYPE\_TYPE\_STANDARD |

UsbConst.REQUESTTYPE\_RECIPIENT\_DEVICE),

UsbConst.REQUEST\_CLEAR\_FEATURE, (short) 8, (short) 8);

irp.setData(data);

device.syncSubmit(irp);

виникає виключення:

javax.usb.UsbPlatformException: USB error 2: Unable to submit control message: Invalid parameter

Ситуація схожа з кореневою проблемою з буфером обміну.

* + 1. Використання низькорівневих функцій бібліотеки usb4java

При використанні низькорівневих функцій бібліотеки usb4java ситуація не змінилась – проблема з буфером обміну.

* + 1. Використання JNI

Основуючись на програмі Kismet Spectools, працювала з даним пристроєм, для вирішення проблеми було вирішено зробити JNI до С++. Ключовим фактором для роботи пристрою став виклик функції бібліотеки libusb

int libusb\_detach\_kernel\_driver (libusb\_device\_handle \*dev, int interface\_number)

яка працює тільки для ОС linux. Вона виконує від’єднання драйверу ядра внаслідок чого можна зробити захват інтерфейсу:

libusb\_detach\_kernel\_driver(dev\_handle, 0)

libusb\_claim\_interface(dev\_handle, 0);

Після цого можна зчитати данні з пристрою:

libusb\_control\_transfer(dev\_handle, USB\_ENDPOINT\_IN + USB\_TYPE\_CLASS + USB\_RECIP\_INTERFACE, HID\_GET\_REPORT, (HID\_RT\_FEATURE << 8), 0, buf, buf\_size, TIMEOUT);

* + 1. Розбір даних з пристрою

Пристрій повертає пакети довжиною в 8 символів:

[0, 4, 2, 2, 1, 2, 1, 1]

[7, 2, 3, 7, 0, 0, 2, 0]

[14, 1, 2, 0, 1, 2, 2, 3]

[21, 1, 2, 1, 3, 2, 2, 1]

[28, 1, 3, 3, 2, 0, 0, 2]

[35, 1, 0, 1, 1, 1, 3, 1]

[42, 2, 1, 2, 2, 1, 1, 3]

[49, 2, 2, 1, 16, 31, 31, 16]

[56, 2, 1, 1, 3, 1, 2, 1]

[63, 3, 1, 3, 2, 1, 2, 3]

[70, 3, 1, 1, 2, 2, 1, 2]

[77, 1, 2, 0, 2, 2, 1, 0]

Перший елемент масиву можна інтерпретувати як кількість вже переданих значень RSSI, чи як порядковий номер наступного значення RSSI. Від другого до восьмого включно – значення RSSI.

Після того, як пристрій пробігає по всьому спектру, данні приводяться до чисельного типу та заносяться у масив, до якого, у кінець, підмішується символ кінця пакету, заданий константою END\_PACKET\_SEQUENCE:

END\_PACKET\_SEQUENCE = new byte[]{-1};

[4, 2, 2, 1, 2, 1, 1, 2, 3, 7, 0, 0, 2, 0, 1, 2, 0, 1, 2, 2, 3, 1, 2, 1, 3, 2, 2, 1, 1, 3, 3, 2, 0, 0, 2, 1, 0, 1, 1, 1, 3, 1, 2, 1, 2, 2, 1, 1, 3, 2, 2, 1, 16, 31, 31, 16, 2, 1, 1, 3, 1, 2, 1, 3, 1, 3, 2, 1, 2, 3, 3, 1, 1, 2, 2, 1, 2, 1, 2, 0, 2, 2, 1, 0, -1]

* 1. MetaGeek Wi-Spy 2.4x2
     1. Ініціалізація

Для того, щоб ініціалізувати пристрій потрібно знати спеціальну послідовність ініціалізації, яку потрібно передати пристрою. Послідовність ініціалізації представляє собою набір команд, які пристрій здатний сприймати. У даному випадку офіційної документації немає, тому використаємо сніффер USB трафіку USBlyzer у поєднанні з програмою MetaGeek Chanalyzer, у якій вже реалізовано ініціалізацію пристрою (Рисунок 1 - Сніффінг USB трафіку для визначення послідовності ініціалізації).



1. Сніффінг USB трафіку для визначення послідовності ініціалізації

При спробі відправити команду ініціалізації до пристрою під ОС Windows виникає наступне виключення:

java.io.IOException: The parameter is incorrect.

at com.codeminders.hidapi.HIDDevice.write(Native Method)

яке говорить про те, що в Native Method був переданий не коректний параметр. Треба відмітити, що той самий програмний код коректно працює під linux та не працює під Windows. У подальшому можлива спроба виправити це за допомогою JNI, а поки що цей пристрій підтримується тільки на linux.

* + 1. Розбір даних з пристрою

За одну передачу пристрій видає наступну послідовність:

[74, 0, 0, 0, 0, 33, 67, 67, 69, 66, 68, 65, 67, 66, 66, 64, 67, 67, 67, 67, 67, 67, 70, 62, 68, 68, 64, 66, 68, 68, 67, 68, 67, 68, 64, 67, 68, 68, 65, 67, 66, 68, 69, 65, 66, 66, 68, 68, 65, 69, 67, 66, 67, 66, 67, 68, 66, 66, 66, 68, 67, 67, 68, 65, 74, 59]

Структура пакету схожа на MetaGeek Wi-Spy Gen 1. Перший елемент масиву можна інтерпретувати як кількість вже переданих значень RSSI, чи як порядковий номер наступного пакету значень RSSI. Від другого до п’ятого включно – сервісна інформація. Всі інші – значення RSSI зі здвигом на 170 одиниць.

* 1. Texas Instruments ez430-RF2500

Перед застосуванням, пристрій треба було попередньо налаштувати. Проблема була в тому, що пристрій не бачив частину спектру. Далі наводяться графіки, що підтверджують проблему.



1. Перший канал під загрузкою



1. Шостий канал під загрузкою



1. 11-й канал під загрузкою

З графіків вище видно, що діапазон частот до шостого каналу, тобто частоти 2401-2426, а також після 11-го каналу, тобто частоти 2473-2483 випадають із зони видимості пристрою.

Для вирішення даної проблеми треба:

1. Визначити, які регістри відповідають за потрібні налаштування та встановити коректні параметри.
2. Прошити пристрій виправленим кодом.
3. Провести тестування.

У ході роботи над даним пристроєм було використано програмне забезпечення SmartRF Studio, та програмний код із статті Creating a Spectrum Analyzer to Measure Noise. Для компіляції програмного коду та відправки його до пам’яті пристрою було використано IDE IAR Embedded Workbench.

* + 1. Пошук потрібних регістрів та встановлення коректних параметрів

У заготовочному файлі source\_code\drivers\mrfi\smartrf\CC2500\smartrf\_CC2500.h було знайдено об’явлення регістрів. Нижче приведені регістри, які відповідають налаштуванню Base Frequency:

#define SMARTRF\_SETTING\_FREQ2 0x5D

#define SMARTRF\_SETTING\_FREQ1 0x93

#define SMARTRF\_SETTING\_FREQ0 0xB1

Для перевірки введемо ці значення до SmartRF Studio.





1. Перевірка значень регістрів у SmartRF Studio

Рисунок 5 вказує, що при таких значеннях регістрів Base Frequency = 2433, що підтверджує проблему. Підставимо коректні значення Base Frequency:





1. Коректні значення регістрів для Base Frequency

#define SMARTRF\_SETTING\_FREQ2 0x5С //old val 0x5D

#define SMARTRF\_SETTING\_FREQ1 0x58 //old val 0x93

#define SMARTRF\_SETTING\_FREQ0 0x9В //old val 0xB1

Тестування змін:



1. Тестування скорегованого Base Frequency

Рисунок 7 засвідчує коректність нових значень Base Frequency.

Далі скорегуємо значення регістрів Channel Spacing, який відповідає за відстань між дискетами, так, щоб Carrier frequency був близьким до максимальний для даного пристрою (2484 kHz). Для цього використаємо поле Channel number та Carrier frequency (Рисунок 8 ).





1. Корегування значень регістру Channel Spacing

Рисунок 9 засвідчує коректність змін Channel Spacing.



1. Тестування змін Channel Spacing
   * 1. Підключення до MDRV

Пристрій визначається системою як COM. Формат пакету дуже простий – спочатку йдуть значення RSSI, а у кінці – символ кінця строки (‘\n’). Пристрій ініціалізується самостійно при підключенні до комп’ютера.

* 1. Air View 2
     1. Ініціалізація

Визначається як COM. Пристрій ініціалізується за допомогою передачі на нього спеціальних послідовностей, які приведені нижче:

byte[] intByte = new byte[]{0x69, 0x6E, 0x74}; //int

byte[] bsByte = new byte[]{0x0A, 0x62, 0x73, 0x0A}; //.bs.

Відповідно до ASCII перша послідовність означає “int”, тобто “initialize”. Друга послідовність – “.bs.” – “begin scan”.

* + 1. Розбір даних з пристрою

Пристрій визначається системою як COM. Формат пакету дуже простий – спочатку йдуть значення RSSI, а у кінці – символ кінця строки (‘\n’).

* 1. Unigen ISM Sniffer

Визначається як USBHID. Пристрій ініціалізується та починає передавати значення RSSI самостійно, при підключенні до комп’ютера. Значення RSSI потрібно корегувати за наступною формулою:

(((aByte - 135) + 100) \* 1.428) – 100

Де aByte – передане пристроєм значення.

* 1. Wixel Pololu

Визначається як COM. Пристрій ініціалізується та починає передавати значення RSSI самостійно, при підключенні до комп’ютера. Пакет, як і у Air View 2 та Texas Instruments ez430-RF2500 - спочатку значення RSSI, а у кінці – символ кінця строки (‘\n’).

1. Висновки
2. Список літератури
3. Стаття Creating a Spectrum Analyzer to Measure Noise - http://cnx.org/contents/35441aa1-e93c-4c5b-82fe-bc1aa16bf6fd@10.6:19/eZWSN:\_Experimenting\_with\_Wire