



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA
STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

Praca dyplomowa
inżynierska

Implementacja programowego dekodera DAB+ na komputerze jednokładowym

Software implementation of DAB+ decoder on single-board computer

Imię i nazwisko
Kierunek studiów
Opiekun pracy

Kacper Patro, Paweł Szulc
Teleinformatyka
dr inż. Jarosław Bułat

Kraków, rok 2015

Spis treści

| | |
|--|----|
| 1. Wstęp | 1 |
| 2. Opis teoretyczny zagadnień | 3 |
| 3. Biblioteka <code>sdrdab</code> na komputerze jednokładowym | 5 |
| 3.1. Wykorzystane komponenty sprzętowe | 5 |
| 3.2. Dostosowywanie systemu GNU/Linux Ubuntu | 8 |
| 3.3. Implementacja biblioteki <code>sdrdab</code> na procesorze ARM | 16 |
| 3.4. Interfejs graficzny | 22 |
| 3.5. Aplikacja pilot | 40 |
| 4. Testy wydajności biblioteki <code>sdrdab</code> | 50 |
| 5. Podsumowanie | 56 |
| Bibliografia | 57 |

OŚWIADCZENIE AUTORÓW

Oświadczamy, świadomi odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

.....
(Podpisy autorów)

1. Wstęp

DAB (ang. *Digital Audio Broadcasting*) jest standardem cyfrowej transmisji dźwięku. Pozwala na nadawanie wielu stacji w jednym przedziale częstotliwości (multipleksie). Strumień audio kodowany jest w standardzie MP2 (ang. *MPEG-1 Audio Layer-2*). Poza dźwiękiem, oferowane mogą być inne usługi takie jak tekst, obrazy czy dane pakietowe [1].

Powstały w 2007r. standard DAB+ jest jego rozszerzeniem. Użycie kodeka HE-AAC v2 (ang. *High-Efficiency Advanced Audio Coding version 2 profile*), dzięki wydajniejszym algorytmom kompresji, umożliwia ograniczenie przepływności bitowej strumienia audio, nie wpływając na jakość dźwięku. Dzięki temu możliwe jest zwiększenie liczby stacji w jednym paśmie [2], [3]. Radio cyfrowe jest naturalnym następcą analogowego radia z modulacją częstotliwości (FM). Głównymi zaletami są:

- lepsza, cyfrowa jakość dźwięku
- wydajniejsze wykorzystanie pasma radiowego
- możliwość elastycznego modelowania usług w obrębie jednego multipleksu

Wady, w stosunku do radia FM, to:

- większa złożoność nadajnika i odbiornika
- wysoka cena odbiorników

Chcąc zaproponować alternatywę wobec dedykowanych urządzeń, zdecydowano się zaprojektować oraz wykonać odbiornik radiowy w standardzie DAB+ wykorzystując technikę radia programowalnego. W takim rozwiązaniu funkcje elektroniki przejmują program komputerowy. Wystarczy posiadać komputer z odpowiednią aplikacją i moduł radia z anteną (kosztujący około pięćdziesiąt złotych), by stworzyć odbiornik posiadający funkcjonalności rozwiązania sprzętowego. Na zajęciach projektowych, *Radio programowalne w praktyce*, będących przedmiotem obieralnym na studiach pierwszego stopnia Teleinformatyki na Akademii Górniczo-Hutniczej opracowano bibliotekę `sdrdab` [4]. W niniejszej pracy, współautorzy projektu wykorzystują go do zaimplementowania dekodera DAB+ na komputerze jednokładowym.

Celem pracy było skonstruowanie urządzenia opartego o komputer jednoukładowy, dekodującego sygnał radiowy w standardzie DAB+. Miałby on stanowić tanią alternatywę dla komercyjnych rozwiązań. Założono, że urządzenie będzie pozwalać na odgrywanie strumienia audio z możliwością przełączania stacji, za pomocą panelu dotykowego lub aplikacji mobilnej. Dodatkową funkcjonalnością będzie wyświetlanie spektrogramu widma radiowego i szacunkowego parametru SNR.

Niniejsza praca składa się z pięciu części, wliczając ten wstęp. Drugi rozdział związany jest z zagadnieniami teoretycznymi. W rozdziale trzecim, *Biblioteka sdrdab na komputerze jednoukładowym*, opisano:

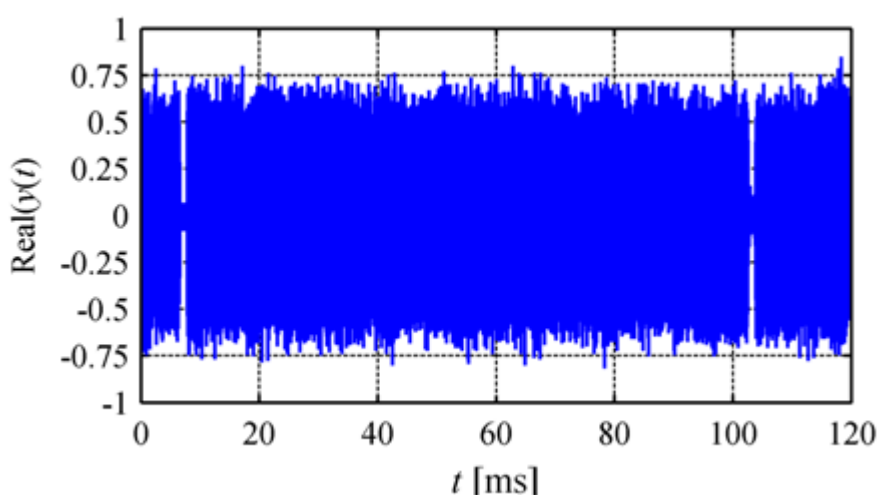
- sprzęt komputerowy i system operacyjny
- prace nad zwiększeniem wydajności zastosowanej biblioteki
- rozszerzenia biblioteki sdrdab
- implementowanie interfejsu graficznego i aplikacji-pilota

W rozdziale czwartym, poświęconym testom, opisano metody weryfikacji zaimplementowanych rozwiązań. Praca zakończona jest podsumowaniem otrzymanych wyników.

Punkty *Wykorzystane komponenty sprzętowe*, *Dostosowywanie systemu GNU/Linux Ubuntu*, *Interfejs graficzny*, *Aplikacja pilot* wykonane zostały przez Kacpra Patro. Punkty *Opis teoretyczny zagadnień*, *Implementacja biblioteki sdrdab na procesorze ARM*, *Testy wydajności biblioteki sdrdab* wykonane zostały przez Pawła Szulca. Pozostałe punkty wykonane zostały wspólnie.

2. Opis teoretyczny zagadnień

System DAB+ wykorzystuje technikę transmisji na wielu nośnych – OFDM (ang. *Orthogonal Frequency-Division Multiplexing*). Na każdej z nośnych dane modulowane są metodą DQPSK (ang. *Differential Quaternary Phase Shift Keying*). Standard wyróżnia cztery rodzaje transmisji (I-IV), które różnią się liczbą podnośnych OFDM. Podstawowymi jednostkami transmisji są ramki, których długość i struktura także zależą od trybu transmisji, oddzielone tzw. symbolami NULL. Przykładowy sygnał w dziedzinie czasu przedstawiony został na rysunku 1.



Rys. 1. Fragment odebranego sygnału w dziedzinie czasu
Źródło: [2]

Najważniejsze elementy ramki to FIC (ang. *Fast Information Channel*) z informacjami o konfiguracji multipleksu (bitrate, lista stacji, sposób kompresji), oraz MSC (ang. *Main Service Channel*) przenoszący dane konkretnego serwisu, np. dźwięk. Standard DAB+ używa kodeka dźwięku HE-AACv2. Z sekwencji kilku ramek (liczba zależy od trybu transmisji) tworzona jest super ramka, która rozumiana jest jako samodzielne dane AAC. Poszczególne stacje w multipleksie mogą być kodowane za pomocą MP2. W Polsce radio cyfrowe dostępne jest w multipleksie Polskiego Radia [5].

Technika radia programowalnego (ang. *SDR – Software Defined Radio*) rozwinęła się dzięki rosnącym możliwościom współczesnych procesorów. Takie rozwiązanie pozwala na zastąpienie modułów odbiornika radiowego napisanym programem. Dzięki temu, poza modułem radia, niepotrzebne jest specjalne urządzenie, a napisany program można później udostępnić.

Główne ograniczenia radia programowalnego to:

- parametry modułu radia
- wydajność procesora

Dostępnych jest wiele modułów radia, różniących się ceną oraz parametrami takimi jak zakres częstotliwości czy rozdzielczość przetwornika analogowo-cyfrowego. Urządzenia USRP (ang. *Universal Software Radio Peripheral*), poza szerokim pasmem i 14-bitowym przetwornikiem A/C, mogą pracować również w trybie nadajnika [6]. Znacznie niższą ceną charakteryzują się moduły DVB-T (ang. *Digital Video Broadcasting-Terrestrial*), posiadające przetwornik o rozdzielczości ośmiu bitów. Biblioteka `rtl-sdr` wykorzystuje możliwości tych urządzeń do odbieru danych radiowych z zakresu nawet 24 – 2200 MHz i częstotliwości próbkowania dochodzącej do 3,2 MS/s [7]. Więcej informacji na temat SDR można znaleźć w [8]. Technika SDR ma też zastosowanie w dydaktyce. Wymaga od adeptów programowania umiejętności pisania wydajnego kodu, który musi być wykonywany w czasie rzeczywistym. Istotna jest też znajomość zagadnień telekomunikacyjnych i radiowych.

W niniejszej pracy skorzystano z biblioteki `sdrdab`, która powstała w ramach przedmiotu poświęconemu radiu programowalnemu. Odbywał się on na zajęciach studiów pierwszego stopnia Teleinformatyki na Akademii Górniczo-Hutniczej w Krakowie. Celem tego przedmiotu było zaimplementowanie biblioteki w języku C++ pozwalającej na dekodowanie sygnału DAB+ w czasie rzeczywistym. Biblioteka `sdrdab` dostępna jest na licencji LGPL, pod adresem [4]. Z biblioteki skorzystano podczas realizacji niniejszej pracy inżynierskiej. Stawiając sobie za cel wypracowanie taniego rozwiązania, użyto modułu DVB-T REALTEK 2832U.

3. Biblioteka sdrdab na komputerze jednokładowym

Dobór odpowiedniego sprzętu i oprogramowania do realizacji założonych celów jest wymagającym zadaniem. Zaakceptowanie pewnych wad rozwiązania niesie za sobą konsekwencje w kolejnych fazach projektu. W przypadku tej pracy, zdecydowano o użyciu sprawdzonych rozwiązań, szerzej opisanych w dalszej części rozdziału.

3.1. Wykorzystane komponenty sprzętowe

Przy wyborze komputera jednokładowego kierowano się stosunkiem wydajności do ceny i dostępnością urządzenia na rynku detalicznym. Ostatecznie, wybór zawężono do dwóch kandydatów wycenionych na około dwieście złotych (patrz tabela 1).

Tabela 1. Zestawienie porównywanych komputerów jednokładowych

| --- | Odroid C1+ | Raspberry Pi 2 |
|------------|--|--|
| CPU | Amlogic ARM Cortex-A5 4x1.5Ghz | Broadcom ARM Cortex-A7 4x900MHz |
| GPU | Mali-450 MP2 GPU | Broadcom Videocore IV |
| Pamięć RAM | 1GB | 1GB |
| Inne | slot kart SD, 4 x USB, HDMI, Gigabit Ethernet | slot kart SD, 4 x USB, HDMI, Ethernet |

Źródło: opracowano na podstawie [9] i [10]

Biorąc pod uwagę sposób działania biblioteki sdrdab, kluczowym parametrem była wydajność procesora. Opierając się na dostępnych testach wydajnościowych, zdecydowano o użyciu platformy Odroid C1+ firmy Hardkernel (patrz rysunek 2).

Z urządzeniem Odroid C1+ zintegrowano ekran dotykowy. Ze względu na stosunek jakości do ceny, wybrano siedmio-calowy wyświetlacz AT070TN90 o rozdzielczości WVGA współpracujący z kontrolerem VS-TY2662-V1. Ekran LCD oraz akcesoria przedstawiono na rysunku 3. Kontroler VS-TY2662-V1 jest modułem zapewniającym zasilanie i będącym mostem pomiędzy ekranem i źródłem sygnału wideo. Od strony kontrolera, do komunikacji z wyświetlaczem AT070TN90 użyto szerokiej taśmy FFC, od strony urządzenia Odroid C1+

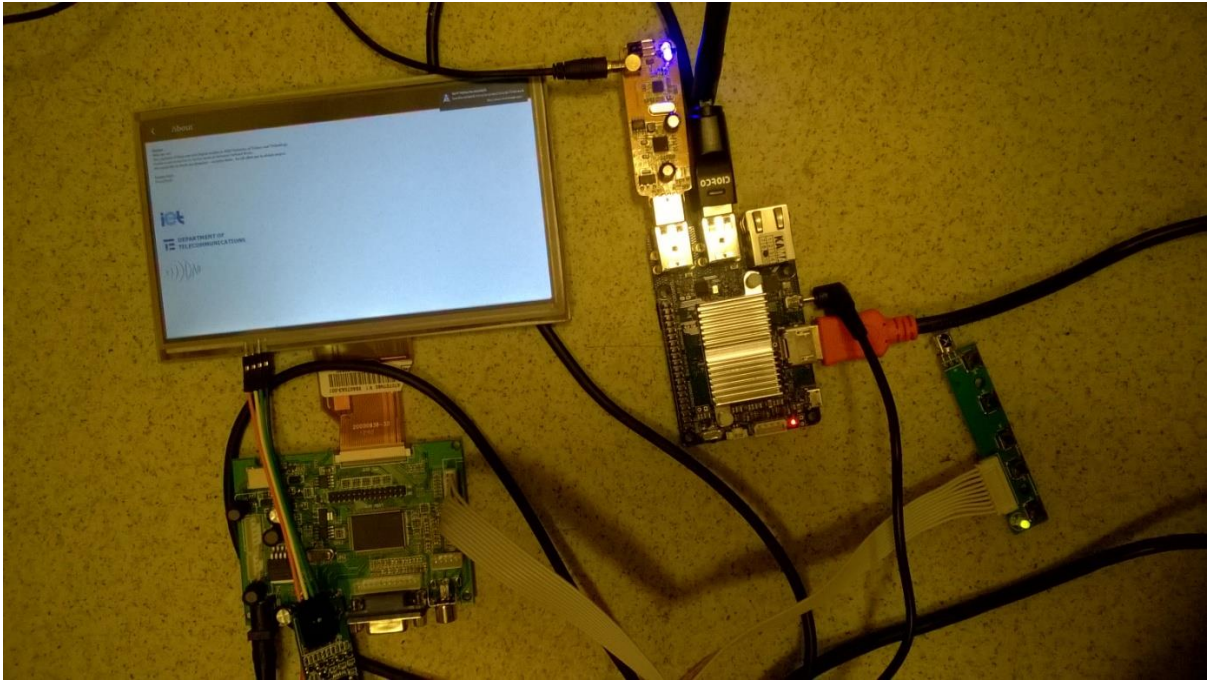
skorzystano z wbudowanego portu HDMI. Wyświetlacz AT070TN90 wyposażony został w rezystancyjną nakładkę dotykową. Nakładka składa się z dwóch pokrywających się powierzchni płaskich, tworzących wspólnie wyjście czteropinowe. Wyjście połączone jest taśmą FFC z modułem konwertującym sygnał, którego wyjściem jest port USB – ostatecznie podłączony do urządzenia Odroid C1+. Na rysunku 4 przedstawiono fizyczne, a na rysunku 5 logiczne połączenie sprzętu.



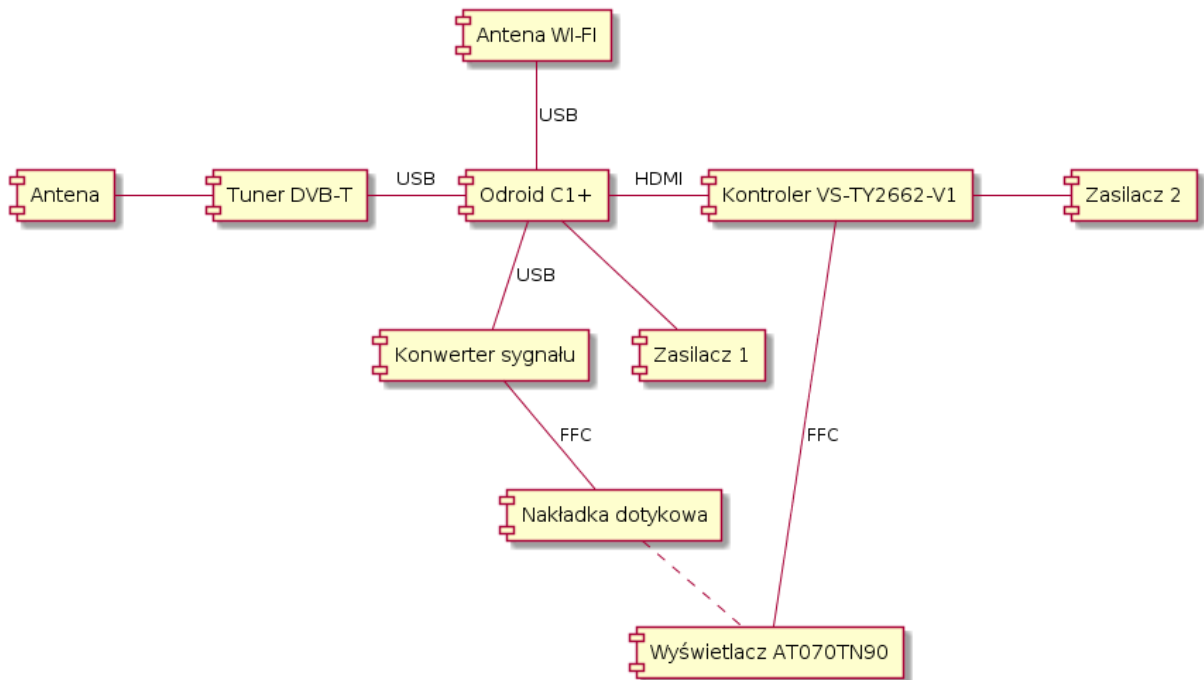
Rys. 2. Odroid C1+
Źródło: [11]



Rys. 3. Wyświetlacz AT070TN90 wraz z kontrolerem VS-TY2662-V1 i rezystancyjną nakładką dotykową
Źródło: [12]



Rys. 4. Fizyczne połączenie sprzętu



Rys. 5. Logiczny schemat połączeń sprzętu¹

¹ Zarówno urządzenie Odroid C1+ jak i kontroler VS-TY2662-V1 wymagają własnego źródła zasilania

3.2. Dostosowywanie systemu GNU/Linux Ubuntu

Urządzenie Odroid C1+ wspierane jest przez wiele systemów operacyjnych, począwszy od Androida na OpenELEC kończąc. W implementacji tego projektu użyto systemu Ubuntu 14.04.3 LTS (v1.6). Bezpośrednią przyczyną takiego, a nie innego, wyboru jest zgodność systemu zarządzania pakietami APT, używanego w fazie implementacyjnej biblioteki sdrdab oraz popularność systemu operacyjnego, co przekłada się na pokaźnie wsparcie techniczne ze strony społeczności.

Ze względu na ograniczoną moc obliczeniową urządzenia Odroid C1+, w systemie operacyjnym wyłączone zostały niekonieczne do działania sdrdab usługi. Domyślnie, wraz z startem systemu operacyjnego uruchamiany jest interfejs graficzny biblioteki sdrdab zrealizowany w oparciu na platformie programistycznej Qt, nie serwer graficzny X11. Procesor Amlogic S805, wykorzystywany przez urządzenie Odroid C1+, domyślnie pracuje z częstotliwością taktowania 1500 MHz. W celu podniesienia wydajności, częstotliwość ta została podniesiona do 1728 MHz (patrz tabela 2) – maksymalnej wartości obsługiwanej przez jądro. Nie zmniejszyło to stabilności platformy.

Tabela 2. Podnoszenie częstotliwości taktowania procesora Amlogic S805

```
echo 1728000 >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

Domyślnie, jądro systemu Ubuntu automatycznie ładuje moduł sterownika obsługi DVB. Moduł ten konfliktuje z biblioteką sdrdab, dlatego też zdecydowano o usunięciu go poprzez dodanie odpowiedniej instrukcji (patrz tabela 3) do pliku `/etc/modprobe.d/blacklist.conf`. Instrukcja zapobiega automatycznemu ładowaniu sterownika DVB.

Tabela 3. Zatrzymywanie automatycznego ładowania modułu sterownika obsługi DVB

```
blacklist dvb_usb_rtl28xxu
```

Pierwotnie, biblioteka `sdrdab` pisana była w środowisku IDE Eclipse. W celu przeportowania jej na platformę o ograniczonych zasobach zdecydowano się na użycie narzędzia CMake [13] do budowy systemu i tym samym uwolnienia biblioteki `sdrdab` od zależności związanych z środowiskiem Eclipse. Dodatkowo, zunifikowano to proces kompilacji na platformach ARM i x86.

Przygotowanie pliku `CMakeLists.txt` – głównego pliku konfiguracyjnego, na którym bazuje narzędzie CMake – wymagało trzech kroków: zidentyfikowania zależności biblioteki `sdrdab`, napisania modułów CMake odpowiadających za przeszukiwanie środowiska pod kątem obecności wymaganych bibliotek i konfiguracji samego procesu kompilacji. Biblioteka `sdrdab` posiada następujące zależności biblioteczne: `rtl-sdr`, `ecc`, `samplerate`, `pthread`, `fftw3f`, `gststreamer`, `gststreamer-app`. Wymagane oprogramowanie zostało zainstalowane przy pomocy systemu APT, lub w przypadku `rtl-sdr` i `ecc` – skompilowane bezpośrednio z źródeł (`sdrdab` korzysta z indywidualnie dostosowanej wersji biblioteki `rtl-sdr`, kod źródłowy `rtl-sdr` i `ecc` dystrybuowany jest wraz z `sdrdab`). Dla bibliotek `samplerate`, `fftw3f`, `gststreamer`, `gststreamer-app` koniecznym było dostarczenie dodatkowych modułów CMake do poprawnego wykrycia obecności bibliotek. Każdy z modułów, przy pomocy funkcji języka skryptowego CMake `find_path` i `find_library` sygnalizuje wyższym warstwom narzędzia CMake o obecności poszukiwanych zależności w systemie operacyjnym i ich lokalizacji. Dodatkowo, do wskazania na potencjalne lokalizacje bibliotek, skorzystano z narzędzia `pkg-config` i odpowiadającej mu funkcji CMake `pkg_check_modules`. Konfiguracja procesu kompilacji wymagała zebrania lokalnie dystrybuowanych oraz odnalezionych przez dodatkowe moduły lokalizacji bibliotek, wyznaczenie odpowiedniej kolejności i celów kompilacji oraz określenia odpowiednich flag kompilatora co przedstawiono w tabeli 4.

Tabela 4. Flagi kompilacji, fragment pliku CMakeLists.txt

```
...
ADD_DEFINITIONS (-Wall)
ADD_DEFINITIONS (-Wextra)
ADD_DEFINITIONS (-Wno-unused-parameter)
ADD_DEFINITIONS (-Wno-unused)
ADD_DEFINITIONS (-Wsign-compare)
ADD_DEFINITIONS (-fPIC)
ADD_DEFINITIONS (-O2)
ADD_DEFINITIONS (-dH)
ADD_DEFINITIONS (-fno-strict-aliasing)
ADD_DEFINITIONS (-ffunction-sections)
ADD_DEFINITIONS (-fdata-sections)
ADD_DEFINITIONS (-finline)
ADD_DEFINITIONS (-finline-functions-called-once)
ADD_DEFINITIONS (-fpermissive)
ADD_DEFINITIONS (-pg)
ADD_DEFINITIONS (-march=armv7-a)
ADD_DEFINITIONS (-mfloat-abi=hard)
ADD_DEFINITIONS (-mtune=cortex-a5)
ADD_DEFINITIONS (-mvectorize-with-neon-quad)
ADD_DEFINITIONS (-ftree-vectorize)
ADD_DEFINITIONS (-mfpu=neon)
ADD_DEFINITIONS (-fno-exceptions)
ADD_DEFINITIONS (-pipe)
...
```

Źródło: opracowano na podstawie [14]

Istotną kwestią było dobranie flag kompilatora w taki sposób, aby wynikowy kod binarny był zoptymalizowany do działania na wybranej platformie sprzętowej. Najważniejsze flagi to:

- *-dH*: sprawia, że podczas naruszenia pamięci (ang. *segmentation fault*) wykonywany jest zrzut pamięci do pliku, dzięki czemu możliwe jest analizowanie programu za pomocą np. narzędzia gdb
- *-mfpu=neon*, *-ftree-vectorize*, *-mvectorize-with-neon-quad*: pozwala na używanie rozszerzenia Neon oraz automatyczną optymalizację fragmentów kodu przez wykonywanie nawet czterech działań w jednej instrukcji

Ze względu na małą moc obliczeniową urządzenia Odroid C1+ i złożoność platformy programistycznej Qt, wybranej do stworzenia graficznego interfejsu użytkownika, zdecydowano się na skonfigurowanie środowiska do kompilacji skrośnej. Do działania modułu QtQuick [15], na którym oparty został graficzny interfejs użytkownika, wymagana jest obecność OpenGL ES 2, co z kolei wymagało dobrania odpowiednich wersji i nagłówek biblioteki Mesa oraz sterowników procesora graficznego Mali. W efekcie powstał plik konfiguracyjny, zrozumiały dla narzędzia Qt, specyfikujący wymagane zależności (patrz tabela 5). Kompilacja skrośna wymagała odpowiedniego zamontowania obrazu systemu operacyjnego Ubuntu² (z uwzględnieniem przesunięć partycji), naprawy uszkodzonych dowiązań symbolicznych obrazu (ze względu na punkt montowania) i skrośnej kompilacji Qt przy użyciu zestawu narzędzi Linaro (patrz tabela 6).

² Zamontowanie obrazu pozwoliło na jednoczesną instalację skompilowanych plików bezpośrednio w obrazie systemu Ubuntu, jak również dostęp do docelowego systemu plików, wymaganego podczas kompilacji Qt

Tabela 5. Fragment pliku konfiguracyjnego kompilacji skróśnej Qt

```
...
QMAKE_INCDIR_EGL = $$[QT_SYSROOT]/usr/include/mali/EGL
QMAKE_LIBDIR_EGL = $$[QT_SYSROOT]/usr/lib/arm-linux-gnueabi/mali-egl
QMAKE_INCDIR_OPENGL_ES2 = $$[QT_SYSROOT]/usr/include/mali/GLES2
QMAKE_LIBDIR_OPENGL_ES2 = $$[QT_SYSROOT]/usr/lib/arm-linux-gnueabi/mali-egl
...
QMAKE_LIBS_EGL += -lMali
QMAKE_LIBS_OPENGL_ES2 += -lMali
...
```

Źródło: opracowano na podstawie [16]

Tabela 6. Kompilacja skróśna platformy Qt

```
./configure -release -confirm-license -opensource -device
linux-odroid-c1-g++ -opengl es2 -no-pch -system-xcb -nomake
tests -nomake examples -no-kms -skip qtwebengine -no-gtkstyle
-no-warnings-are-errors -device-option CROSS_COMPILE=~/gcc-
linaro-arm-linux-gnueabi-4.9-2014.09_linux/bin/arm-linux-
gnueabi- -sysroot /mnt/odroid-c1-rootfs -prefix
/usr/local/qt5odroid -hostprefix /usr/local/qt5odroid
```

Źródło: opracowano na podstawie [17]

Ostatnim etapem konfiguracji projektu qmake [18] było właściwe zdefiniowanie ścieżek środowiska uruchomieniowego w taki sposób, aby wskazywały na poprawne lokalizacje systemu docelowego oraz dołączenie wymaganych bibliotek (patrz tabela 7).

Tabela 7. Fragment konfiguracji ścieżek środowiska uruchomieniowego (biblioteka sdrdab umieszczona w folderze /opt/sdr), plik projektu qmake

```
...
QT += qml quick network websockets
...
unix:!macx: LIBS += -L$$[QT_SYSROOT]/opt/sdr -lsdrdab
...
unix:!macx: LIBS += -L$$[QT_SYSROOT]/opt/sdr/rtlsdr-bin/src/ -lrtlsdr
...
INCLUDEPATH += $$[QT_SYSROOT]/opt/sdr/sdrdab/src
DEPENDPATH += $$[QT_SYSROOT]/opt/sdr/sdrdab/src
...
unix:!macx:: LIBS += -pthread -lgstreamer-1.0 -lgobject-2.0 -lglib-2.0 -lpcre
...
```

Użyto parametru `-rpath` linkera (patrz tabela 8), w celu wskazania na moduły bibliotek współdzielonych podczas uruchamiania aplikacji. Zapobiegło to wykorzystywaniu globalnych, niewłaściwych wersji bibliotek.

Tabela 8. Użycie parametru `-rpath`, plik projektu qmake

```
...
LIBS += -Wl,-rpath=/opt/sdr
LIBS += -Wl,-rpath=/opt/sdr/rtlsdr-bin/src
...
```

Uruchomienie obsługi kontrolera dotyku wymagało rekompilacji jądra systemu Ubuntu [19]. Do rekompilacji jądra użyto plików źródłowych udostępnianych przez firmę Hardkernel [20]. Sterowniki kontrolera (USB Touchscreen Driver) skompilowano w formie modułów jądra. Poprawną podmianę jądra wykonano przy pomocy narzędzi `update-initramfs` (utworzenie obrazu `initramfs`) oraz `mkimage` (utworzenie obrazu U-Boot) (patrz tabela 9).

Tabela 9. Podmiana jądra systemu operacyjnego

```
update-initramfs -c -k `cat include/config/kernel.release`
mkimage -A arm -O linux -T ramdisk -C none -a 0 -e 0 -n uInitrd -d
/boot/initrd.img-`cat include/config/kernel.release` /boot/uInitrd-`cat
include/config/kernel.release`
cp /boot/uInitrd-`cat include/config/kernel.release` /media/boot/uInitrd
```

Źródło: opracowano na podstawie [21]

W ramach projektu uwzględniono możliwość strumieniowania audio na urządzenia obsługujące standard DLNA (ang. *Digital Living Network Alliance*). Ze względu na sposób zarządzania strumieniem audio przez bibliotekę sdrdab, konkretniej system GStreamer, uruchomienie wsparcia dla serwera DLNA było zadaniem relatywnie prostym. Wyjściem sdrdab jest serwer dźwięku PulseAudio, który oferuje szerokie spektrum konfiguracyjne w kwestii przekierowywania strumieni audio, a serwer DLNA Rygel [22] jest jedną z dostępnych implementacji DLNA, którego niewątpliwą zaletą jest integracja z biblioteką GStreamer. Punktem wspólnym dla obu aplikacji jest wtyczka `org.gnome.UPnP.MediaServer2.PulseAudio`. Cała konfiguracja serwera zawiera się w pliku `rygel.conf` (patrz tabela 10).

Tabela 10. Zawartość pliku konfiguracyjnego serwera Rygel

```
[External]
enabled=true
```

Do poprawnego skonfigurowania serwera DLNA Rygel wymagane są pakiety `pavucontrol` i `paprefs`:

- *pavucontrol*: pozwala na kontrolę poziomu głośności strumieni
- *paprefs*: umożliwia uruchomienie dostępu do sieciowego serwera DLNA

Rygel do właściwego działania wymaga od PulseAudio załadowania modułu `module-http-protocol-tcp`. Aby moduł ładowany był automatycznie wraz z startem systemu, dodano instrukcję do pliku konfiguracyjnego PulseAudio `/etc/pulse/default.pa` (patrz tabela 11).

Tabela 11. Fragment pliku konfiguracyjnego PulseAudio

```
...  
load-module module-http-protocol-tcp  
...
```

Po uruchomieniu serwera Rygel, strumień audio dostępny jest dla urządzeń wspierających DLNA pod ścieżką *Audio on odroid->ODROID-HDMI Analog Stereo->ODROID-HDMI*. Strumień zakodowany jest w formacie MP3.

Zgodnie z pierwotnymi założeniami, zrezygnowano z uruchamiania serwera graficznego X11 na rzecz aplikacji uruchamianej za pomocą interfejsu EGL. Takie podejście ogranicza zakres interakcji użytkownika końcowego z powłoką systemu operacyjnego i oszczędza moc obliczeniową urządzenia. W celu automatycznego uruchamiania serwera Rygel i graficznego interfejsu użytkownika zmodyfikowano procedury startowe systemu operacyjnego, dodając instrukcje w pliku `.bashrc` (patrz tabela 12).

Tabela 12. Fragment pliku konfiguracyjnego `.bashrc`

```
...  
rygel > /tmp/rygel.log 2>&1 &  
~/dabgui  
...
```

3.3. Implementacja biblioteki `sdrdab` na procesorze ARM

Wykorzystana biblioteka w wersji standardowej nie pozwalała na płynne odtwarzanie dźwięku w czasie rzeczywistym. By zwiększyć wydajność, konieczne okazały się jej modyfikacje. Do profilowania `sdrdab` użyte zostało narzędzie Callgrind programu Valgrind wraz z nakładką Kcachegrind [23]. Analiza wykazała, że zastosowany w bibliotece algorytm Viterbiego zajmuje większość czasu procesora – 83% [24], [25]. Kod algorytmu, napisany w języku C++, okazał się zbyt wolny na wykonanie go w czasie rzeczywistym na wybranej platformie sprzętowej. Zdecydowano się na przepisanie fragmentu kodu odpowiadającego za mnożenie skalarnie wektorów bezpośrednio w języku assemblera dla procesora ARM Cortex-A5. Dodatkowo użyto instrukcji wektorowych Neon w celu przyspieszenia operacji arytmetycznych (rozszerzenie Neon jest elementem większości procesorów serii ARM Cortex-A). Zastosowanie techniki SIMD (ang. *Single Instruction, Multiple Data*) pozwala na równoległe wykonanie jednego działania na maksymalnie czterech 32-bitowych rejestrach [26]. Fragment kodu opatrzony został dyrektywą preprocesora (patrz tabela 13).

Tabela 13. Dyrektywa preprocesora

```
#ifdef ARM_NEON_FLAG
```

Źródło: kod biblioteki `sdrdab`

Flaga `ARM_NEON_FLAG` jest ustawiana podczas kompilacji, w momencie wykrycia architektury ARM. W przeciwnym wypadku, używany jest kod napisany w języku C++. Dzięki temu, zachowana jest kompatybilność z procesorami innych architektur.

Tabela 14 przedstawia kod w języku asembler, który wykonuje działania iloczynu skalarnego oraz dodawania. Są one równoznaczne z działaniami (1), (2).

$$\text{Value1} = *refOut1tmp \circ yIntmp + accMetricBuff_ [*(inStat1_index+mRead)] \quad (1)$$

$$\text{Value2} = *refOut2tmp \circ yIntmp + accMetricBuff_ [*(inStat2_index+mRead)] \quad (2)$$

Zmienne Value1 i Value2 są typu float, refOut1tmp i refOut2tmp to wskaźniki na bufory czterech zmiennych typu int o wartościach 1 lub -1. yIntmp to wskaźnik na bufor czterech zmiennych float, accMetricBuff_ jest tablicą float, a zmienne typu int: inStat1_index, inStat2_index, mRead to indeksy zmieniające się podczas kolejnych iteracji. Przy tych założeniach, wynik działania algorytmu jest identyczny z pierwotnym (napisanym w języku C++) . Neon używa tych samych rejestrów, co koprocesor (ang. *FPU – Floating-Point Unit*). Zależności między rejestrami koprocesora, a wektorami rozumianymi przez moduł Neon (3), (4).

$$\{q0\} \Leftrightarrow \{d0, d1\} \Leftrightarrow \{s0, s1, s2, s3\} \quad (3)$$

$$\{q1\} \Leftrightarrow \{d2, d3\} \Leftrightarrow \{s4, s5, s6, s7\} \quad (4)$$

s0 – pojedynczy, 32-bitowy rejestr koprocesora

d0 – wektor zawierający dwa pojedyncze rejestry

q0 – wektor zawierający cztery pojedyncze rejestry

Algorytm w języku asembler wykonuje następujące operacje:

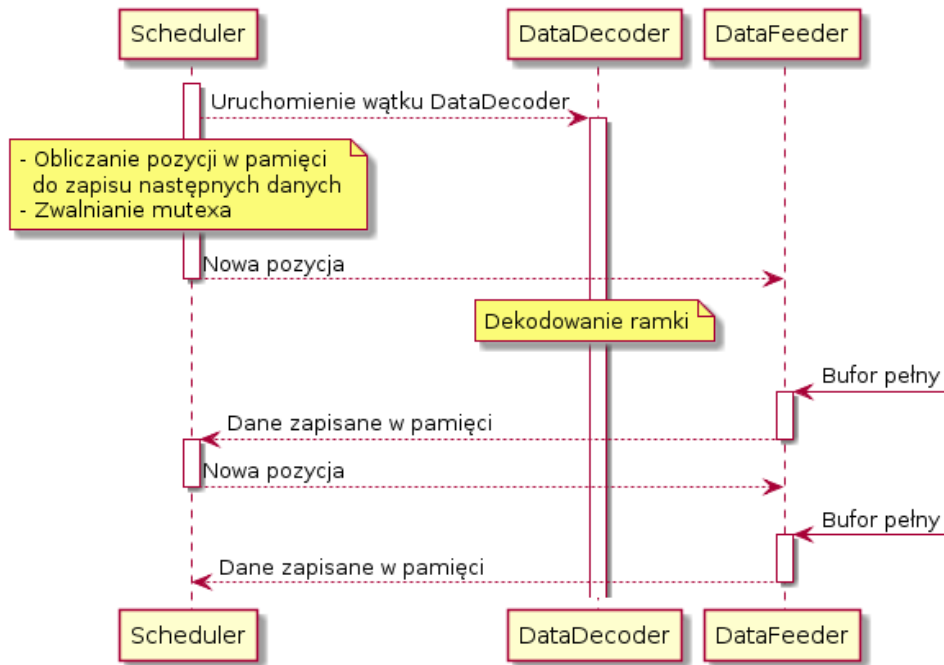
- wczytuje wektorowo bufory do rejestrów koprocesora
- dokonuje mapowania zmiennych int
 - 1 => 0x0
 - -1 => 0x80000000
- wykonuje operację XOR na wektorach zmiennych float ze zmapowanymi wcześniej liczbami int, dzięki temu w miejscu liczby 1 znak zostaje zachowany, a -1 – odwrócony
- sumuje cztery zmienne float ze sobą i dodaje do wyniku odpowiednią liczbę z tablicy accMetricBuff_

Taki kod unika obciążających procesor operacji mnożenia. W miarę możliwości, instrukcje na tych samych rejestrach są przeplatane – używanie rejestrów wynikowych, jako argument następnego działania także spowalnia czas wykonania. Osiągnięto znaczną, ponad trzykrotną, redukcję czasu wykonania tego fragmentu kodu w stosunku do języka C++.

Tabela 14. Kod w języku asembler mnożący skalarnie dwa wektory

```
__asm(  
    "vld1.i32 {q0}, [%2]\n\t"  
    "vld1.i32 {q1}, [%3]\n\t"  
    "vld1.f32 {q2}, [%4]\n\t"  
    "vmov.i32 q4, #0x80000000\n\t"  
    "vmov.i32 q5, #0x00000000\n\t"  
    "vmov s24, %5\n\t"  
    "vmov s25, %6\n\t"  
  
    "vbsl.i32 q0, q4, q5\n\t"  
    "vbsl.i32 q1, q4, q5\n\t"  
  
    "veor.f32 q0, q2\n\t"  
    "veor.f32 q1, q2\n\t"  
  
    "vpadd.f32 d0, d0, d1 \n\t"  
    "vpadd.f32 d2, d2, d3 \n\t"  
  
    "vadd.f32 s0, s0, s1\n\t"  
    "vadd.f32 s4, s4, s5 \n\t"  
  
    "vadd.f32 s0, s0, s24\n\t"  
    "vadd.f32 s4, s4, s25 \n\t"  
  
    "vmov %0, s0\n\t"  
    "vmov %1, s4\n\t"  
  
    : "=r" (Value1), "=r" (Value2)  
    : "r"(refOut1tmp), "r" (refOut2tmp), "r" (yIntmp), "r"  
(accMetricBuff_[ *inStat1_index+mRead ]), "r"  
(accMetricBuff_[ *inStat2_index+mRead ])  
    :  
    );
```

Źródło: kod biblioteki sdrdab



Rys. 7. Schemat działania wypracowanej wersji biblioteki

Biblioteka `sdrdab` została rozszerzona o generowanie punktów widmowej gęstości mocy do prezentacji na wykresie. W tym celu generowanych jest 1024 punktów widmowej gęstości mocy z częstotliwością odświeżania około 1 Hz. Do wyznaczenia transformaty Fouriera użyto, zastosowaną wcześniej w programie, bibliotekę `fftw3` [27]. Następnie dla każdego punktu wyliczona zostaje widmowa gęstość mocy (5).

$$PSD_n = 10 \log_{10} \left(\frac{|X_n|^2}{2 f_s N G} \right) \quad (5)$$

X_n – wartość n-tego prążka transformaty Fouriera

f_s – częstotliwość próbkowania (2,048 MHz)

N – całkowita liczba prążków (1024)

G – programowo ustawione wzmocnienie tunera w skali liniowej

Otrzymujemy 1024 prążków widmowej gęstości mocy w skali decybelowej, z przedziału (6).

$$\left\langle f_c - \frac{f_s}{2} ; f_c + \frac{f_s}{2} \right\rangle \quad (6)$$

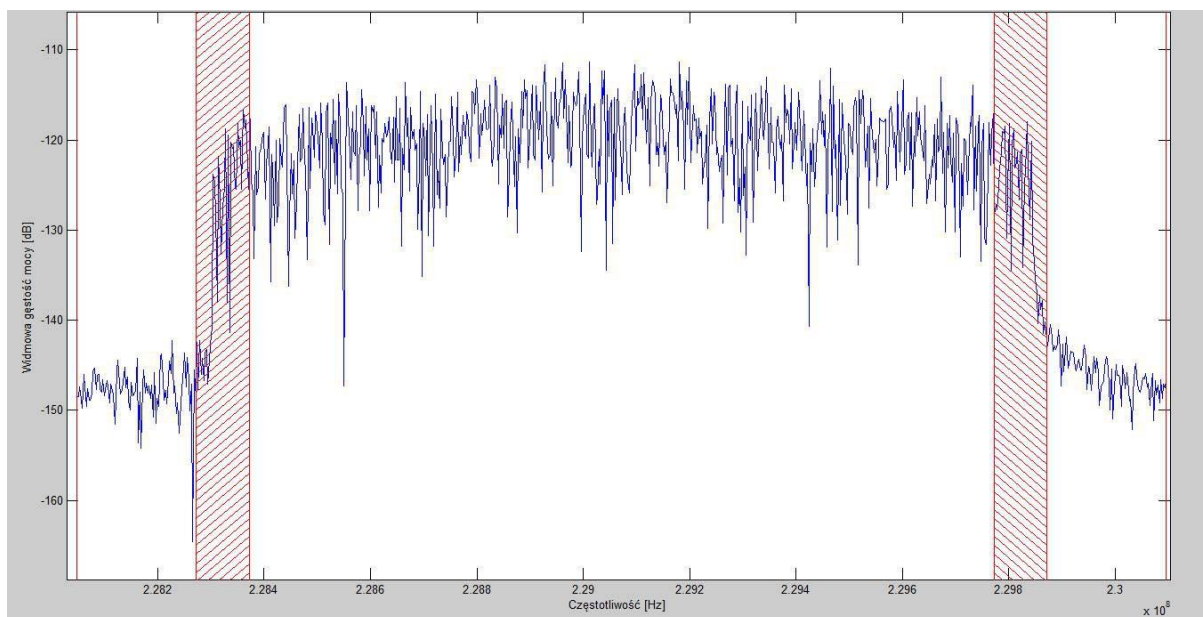
f_c – częstotliwość nośnej

Znając charakterystykę widmową sygnału można z łatwością oszacować SNR (ang. *Signal to Noise Ratio*), przy założeniu że mamy do czynienia z kanałem typu AWGN (ang. *Additive White Gaussian Noise*). W tym celu obliczono średnią wartość widmowej gęstości mocy w paśmie nadawanego sygnału oraz poza tym pasmem. Różnica tych wartości wyrażonych w skali logarytmicznej jest szukanym parametrem SNR (wzory od 7 do 9). Na rysunku 8 przedstawiono przedziały, dla których moc jest uznawana za szum i sygnał. Na wykres naniesiony został też przykładowy wynik funkcji wyznaczającej widmo.

$$SNR = mean (PSD(x \in S)) - mean (PSD(x \in N)) \quad (7)$$

$$S = < -0,7MHz ; 0,7MHz > \quad (8)$$

$$N = < -1,024MHz ; -0,8MHz > \cup < 0,8MHz ; 1,024MHz > \quad (9)$$



Rys. 8. Wykres widmowej gęstości mocy z zarysowanymi przedziałami sygnału i szumu

3.4. Interfejs graficzny

Ze względu na język kodu źródłowego biblioteki `sdrdab`, do wykonania graficznego interfejsu użytkownika wybrane zostało środowisko Qt [27] w wersji Open Source. Platforma programistyczna Qt umożliwiała wybór pomiędzy bardziej dojrzałym, ale przeznaczonym dla aplikacji desktopowych `QtWidget`, a nowoczesnym, stworzonym z myślą o interfejsach dotykowych `QtQuick`. Z racji wspomnianych wyżej założeń projektowych, oczywistym wyborem była druga wersja biblioteki.

Moduł `QtQuick` to standardowa biblioteka dla aplikacji pisanych z użyciem języka QML, zapewniająca dostęp do podstawowych typów tworzących interfejs użytkownika. Sam QML jest deklaratywnym językiem opartym na JavaScript. Zachowania i własności, w odróżnieniu od podejścia imperatywnego, wpisane są bezpośrednio w definicje obiektów, które widziane są z zewnątrz jako autonomiczna całość. Dokumenty QML są rozszerzalne przez inne dokumenty QML lub JavaScript, gdzie kolejną kluczową kwestią jest tzw. *dynamic scoping* – nazwy zmiennych mogą być rozwiązywane w różny sposób w zależności od kontekstu. Obiekty w dokumencie QML tworzą hierarchię drzewa o wspólnym korzeniu. Każdy obiekt w dokumencie QML, oprócz potomków, zawierać może unikalny atrybut `id` (unikalny identyfikator), własności, sygnały, procedury obsługi, metody i dołączone własności oraz dołączone procedury obsługi. Każdy element drzewa dokumentu QML posiada niejawną stan podstawowy. Mechanizm stanów pozwala na zwięzły opis własności obiektu w danym momencie, zawarty w pojedynczej logicznej jednostce. Unikalnym mechanizmem do komunikacji pomiędzy obiektami są sygnały i sloty. W uproszczeniu, mechanizm slotów i sygnałów jest udoskonaleniem³ wywołania zwrotnego (ang. *callback*); sygnał jest emitowany w następstwie konkretnego zdarzenia, gdy slot jest funkcją wywoływaną w odpowiedzi na konkretny sygnał. W języku QML, gdy własności obiektu powiązane są zależnościami, odpowiedzią na zmianę wartości jest ponowne wykonanie wyrażenia wiążącego i przypisanie nowej wartości. Dołączone własności oraz dołączone procedury obsługi są mechanizmem umożliwiającym dekorowanie obiektów dodatkowymi własnościami i procedurami obsługi.

Od strony kodu C++, użyty został system Property. Zadeklarowane w odpowiedni sposób własności obiektu, stają się automatycznie dostępne z poziomu dokumentu QML, zachowując wszystkie właściwości obiektów języka QML (patrz tabela 15).

³ Są mechanizmem *type-safe*, rozluźniają zależności pomiędzy wykonywanymi funkcjami

Tabela 15. System Property platformy programistycznej Qt

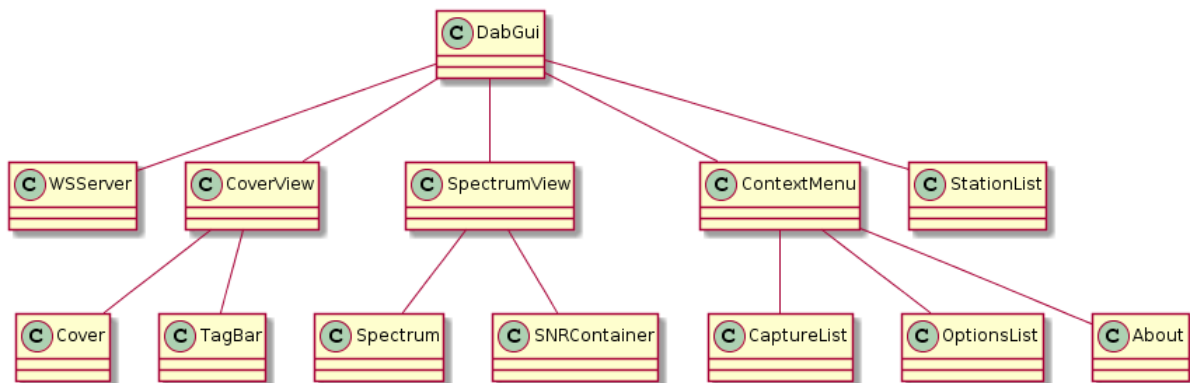
```
Q_PROPERTY(type name
            (READ getFunction [WRITE setFunction] |
            MEMBER memberName [(READ getFunction | WRITE setFunction)])
            [RESET resetFunction]
            [NOTIFY notifySignal]
            [REVISION int]
            [DESIGNABLE bool]
            [SCRIPTABLE bool]
            [STORED bool]
            [USER bool]
            [CONSTANT]
            [FINAL])
```

Źródło: opracowano na podstawie [29]

Uproszczona struktura dokumentu QML przedstawiona została na rysunku 9. Widok QML zachowuje się jak stos, to jest odpowiednie wyświetlane interfejsy, będące wynikiem interakcji z użytkownikiem, są dokładane lub zdejmowane ze stosu. Zerowym elementem stosu jest widok, na który składa się domyślny lub aktualnie przesyłany przez stację obraz, odebrane wiadomości tekstowe, boczne menu kontekstowe, boczne menu wyboru stacji i przycisk przejścia do widoku widma. Widok `SpectrumView` jest widokiem dzielącym z `CoverView` zakres widoczności tj. w jednym momencie widoczny może być `CoverView` lub `SpectrumView`. Menu boczne mogą zostać otwarte w dowolnym momencie, gdy głębokość stosu wynosi jeden - gdy nie jest otwarte żadne podmenu. Po wybraniu podmenu z menu kontekstowego otwarty zostaje widok `CaptureList`, `OptionsList` lub `About`. Otwarcie każdego z wymienionych powoduje zwiększenie głębokości stosu i wyświetlenie menu nawigacyjnego.

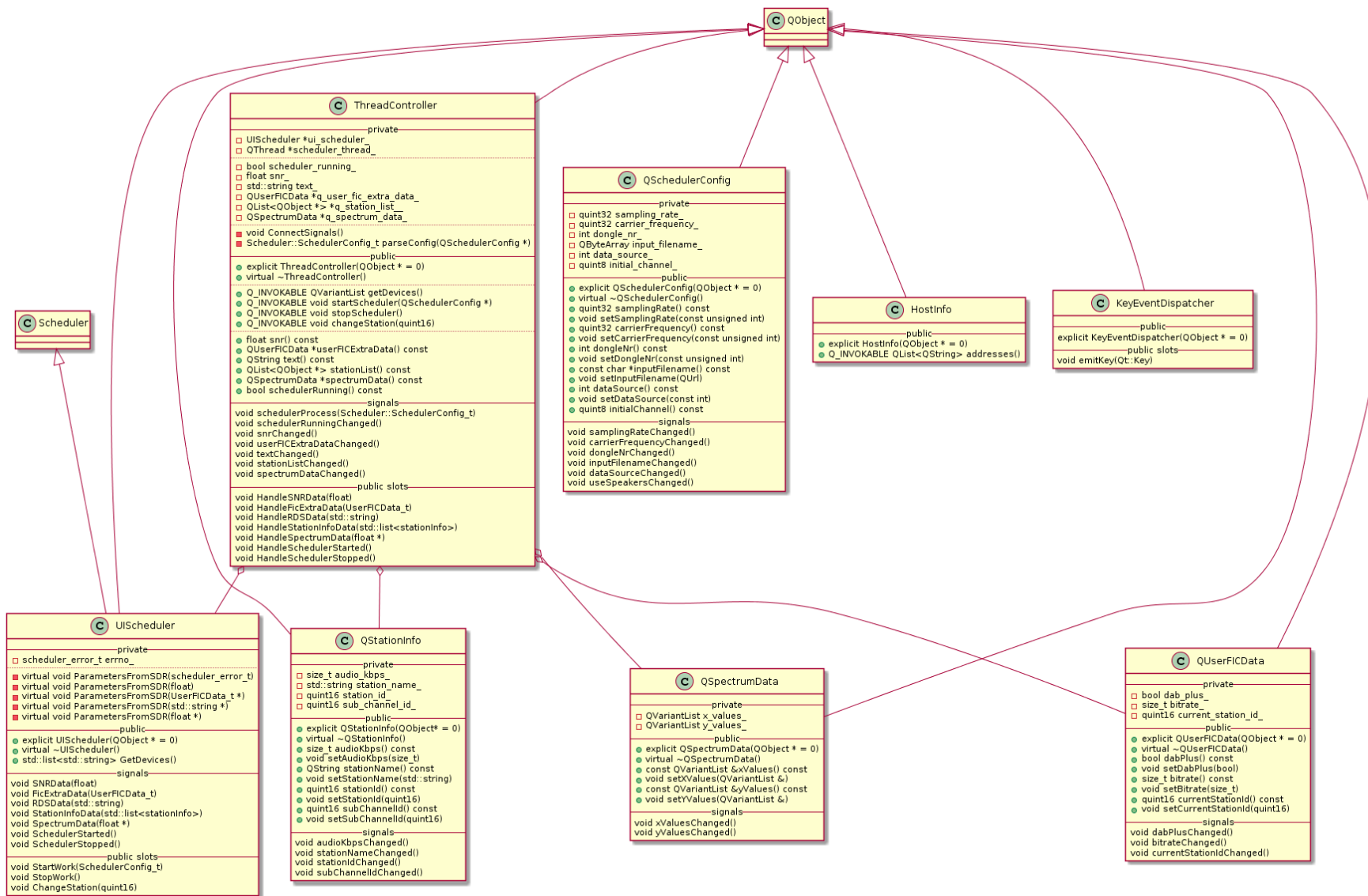
Podmenu `CaptureList` składa się z dwóch sekcji: wyboru odtwarzania z pliku, lub z „powietrza” przy pomocy wykrytych przez `sdrdab` urządzeń. Wybranie którejkolwiek z opcji spowoduje zatrzymanie odtwarzania z bieżącego i przeładowanie aplikacji na nowe źródło. Podmenu `OptionsList` pozwala ustawić parametry częstotliwościowe `sdrdab` (częstotliwość próbkowania i nośną) oraz odczytać adresy IP urządzenia. Z podmenu `OptionsList` zintegrowana jest wirtualna klawiatura dotykowa. Podmenu `About` zawiera informacje o autorach aplikacji.

WSServer hermetyzuje serwer WebSocket, który szerzej opisany zostanie w kolejnych rozdziałach tej pracy.



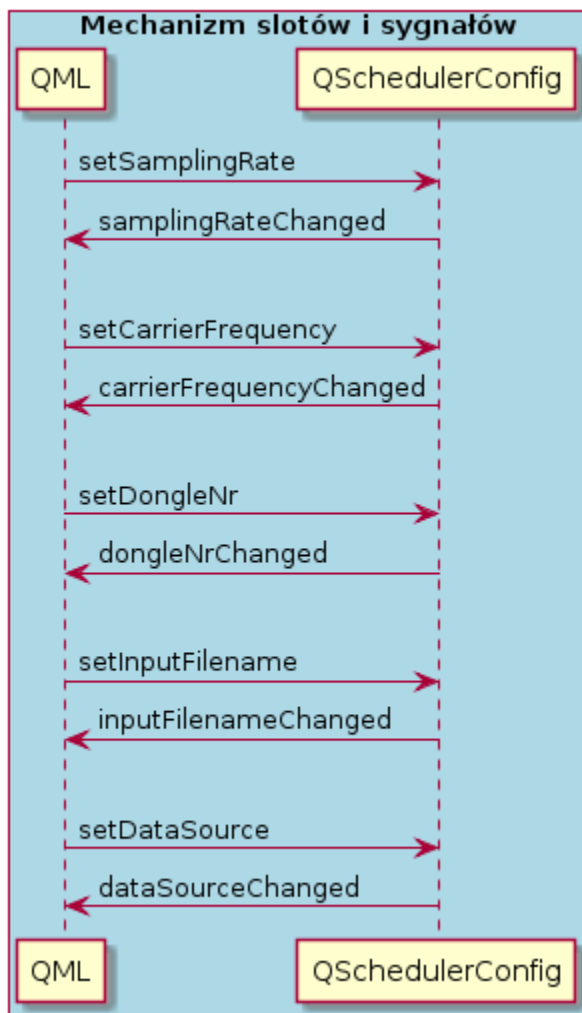
Rys. 9. Uproszczona struktura dokumentu QML

Pełna struktura klas C++ przedstawiona została na rysunku 10. Każda z klas dziedziczy po klasie `QObject`. Dziedziczenie po klasie `QObject` uruchamia wsparcie dla wszystkich dodatkowych mechanizmów unikalnych dla platformy programistycznej Qt, w tym mechanizmu slotów i sygnałów, centralnego punktu platformy programistycznej Qt, który spłaszcza i separuje model klas C++.



Rys. 10. Pełna struktura klas C++

Klasa `QSchedulerConfig` reprezentuje konfigurację wejściową biblioteki `sdrdab`, pozwalając na modyfikację parametrów z poziomu dokumentu QML. Modyfikowalnymi parametrami są: częstotliwość próbkowania i nośna oraz rodzaj źródła: plik lub „powietrze” (patrz rysunek 11).

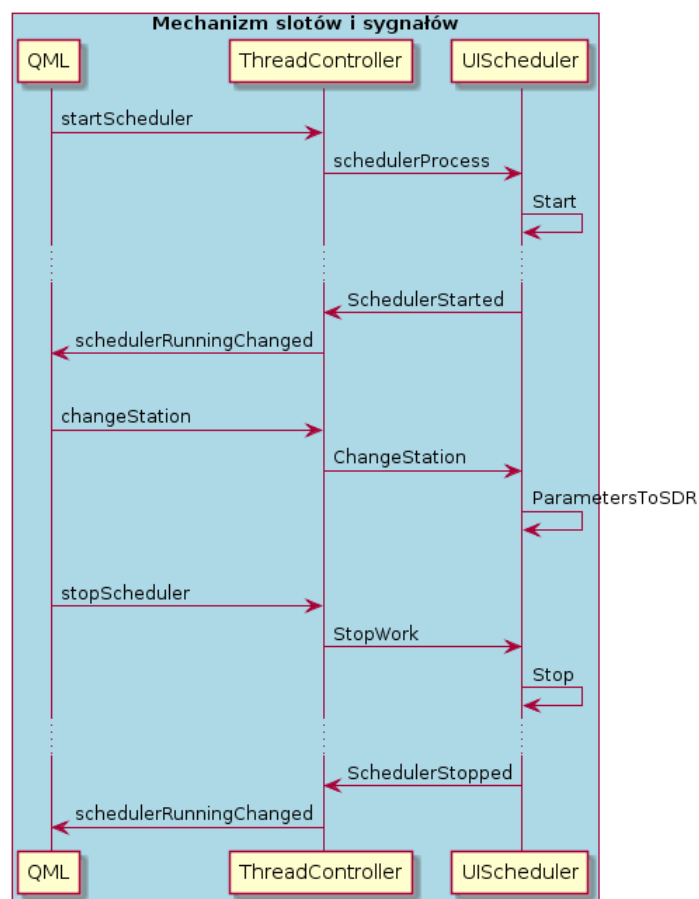


Rys. 11. Diagram sekwencji klasy `QSchedulerConfig`

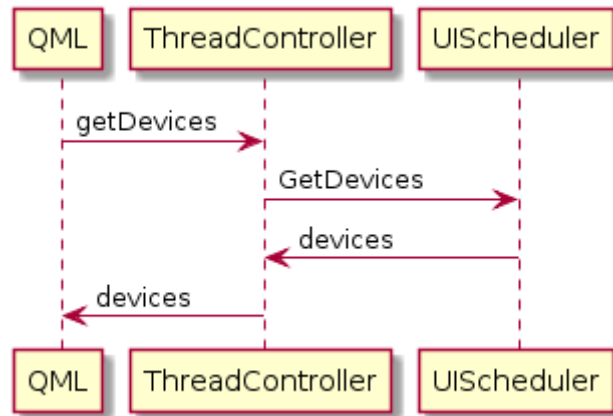
Klasa `ThreadController` steruje zachowaniem biblioteki `sdrdab`: uruchamia i przerywa działanie biblioteki, kontroluje i zapisuje odebrane parametry sygnału DAB+ (patrz rysunek 12). Listę dostępnych dla biblioteki `sdrdab` urządzeń, obiekt klasy `ThreadController` pobrać może przy użyciu metody `GetDevices` obiektu klasy `UIScheduler` (patrz rysunek 13).

W klasie UIScheduler zastosowano mechanizm wielodziedziczenia. Oprócz dziedziczenia po klasie QObject, dziedziczenie po klasie Scheduler pozwoliło na dogodną implementację funkcji wirtualnych, będących interfejsem biblioteki sdrdab. Główną rolą klasy UIScheduler jest zawiadamianie o zdarzeniach związanych z parametrami sygnału DAB+ i stanie wewnętrznym obiektu (uruchomieniu lub zatrzymaniu działania). Funkcja uruchamiająca pracę biblioteki sdrdab, Start, jest blokująca, co wymusiło przeniesienie obiektu klasy UIScheduler do osobnego wątku. Klasa ThreadController uruchamia UIScheduler w osobnym wątku i łączy odpowiednie sloty i sygnały tych klas.

Po rozpoczęciu pracy, obiekt UIScheduler emituje sygnał SchedulerStarted(QSchedulerConfig). W czasie pracy UISchedulera, możliwa jest zmiana stacji przy użyciu slotu ChangeStation. Zakończenie pracy sygnalizowane jest sygnałem SchedulerStopped.

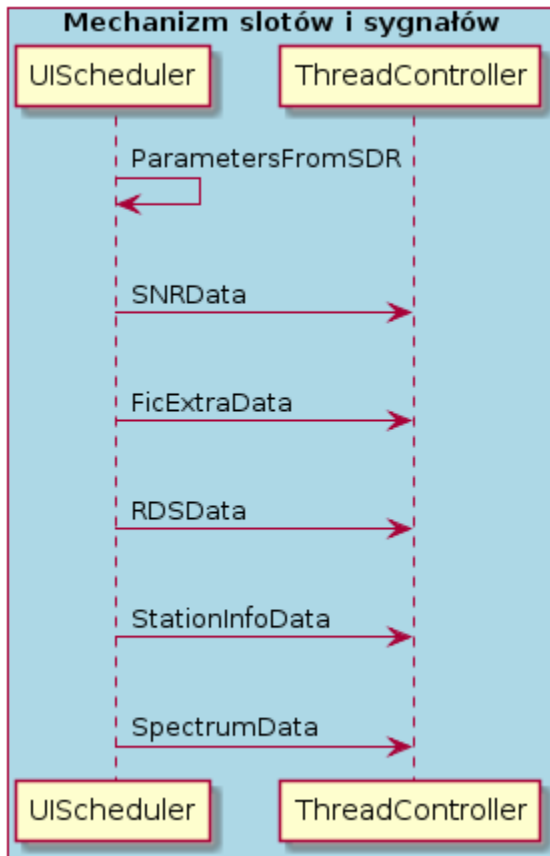


Rys. 12. Diagram sekwencji klasy ThreadController i UIScheduler

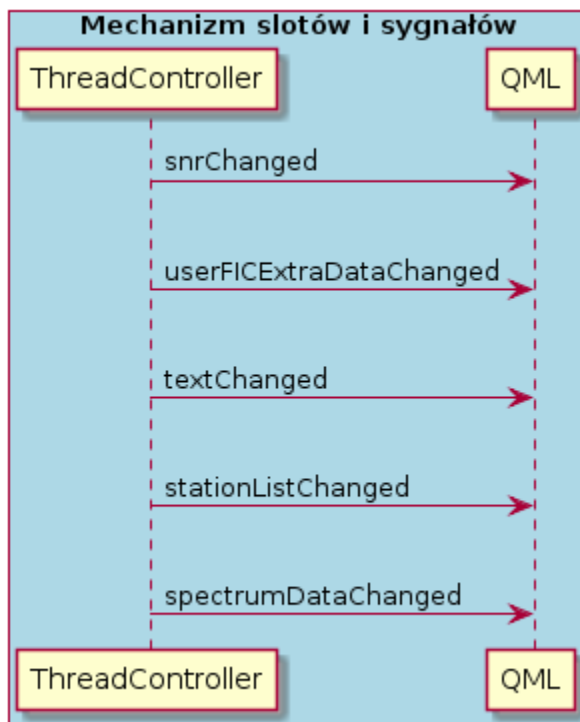


Rys. 13. Diagram sekwencji klasy ThreadController i UIScheduler

Po odebraniu danych z biblioteki sdrdab, obiekt klasy UIScheduler sygnalizuje zmiany emitując odpowiednie sygnały: SNRData, FicExtraData, RDSData, StationInfoData lub SpectrumData (patrz rysunki 14 do 15). Dane są badane pod względem poprawności i umieszczane w odpowiednich strukturach. Na najwyższym poziomie, obiekt klasy ThreadController sygnalizuje zmiany w kierunku dokumentu QML.

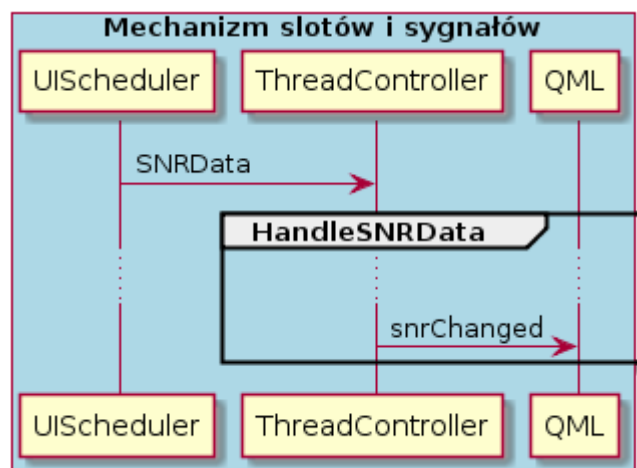


Rys. 14. Odbieranie danych z biblioteki sdrdab, komunikacja klas ThreadController i UIScheduler

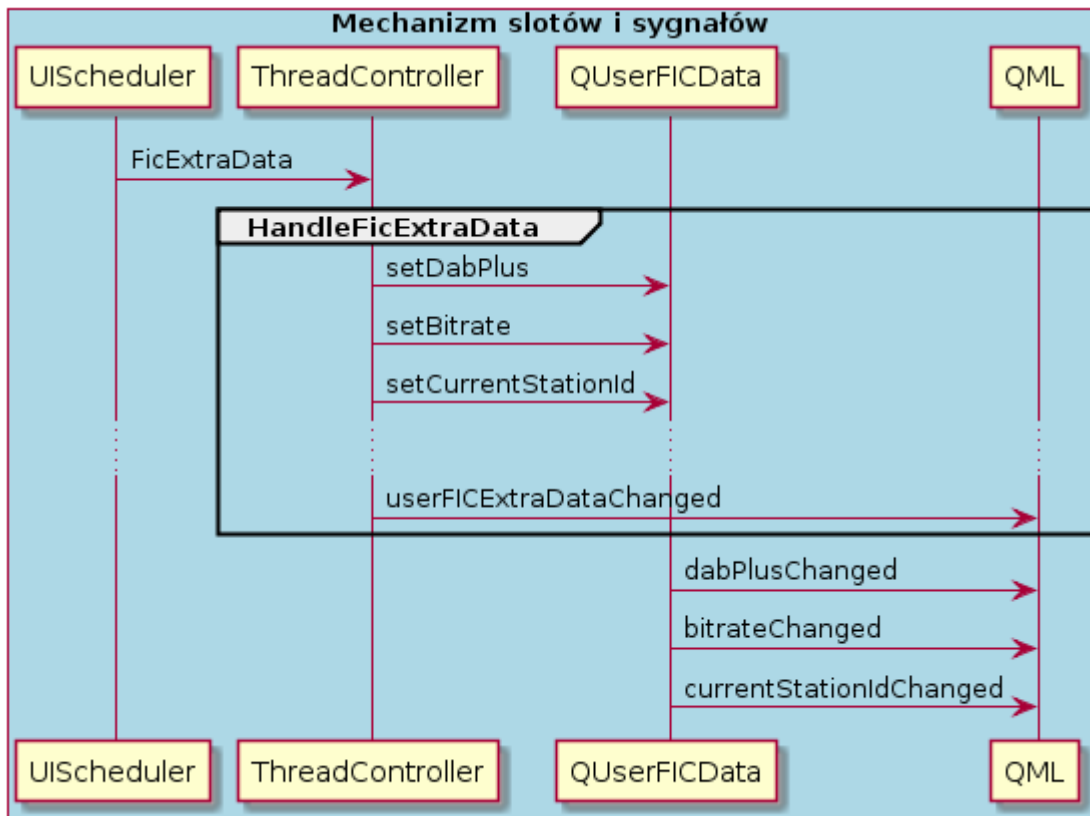


Rys. 15. Diagram sekwencji klasy ThreadController

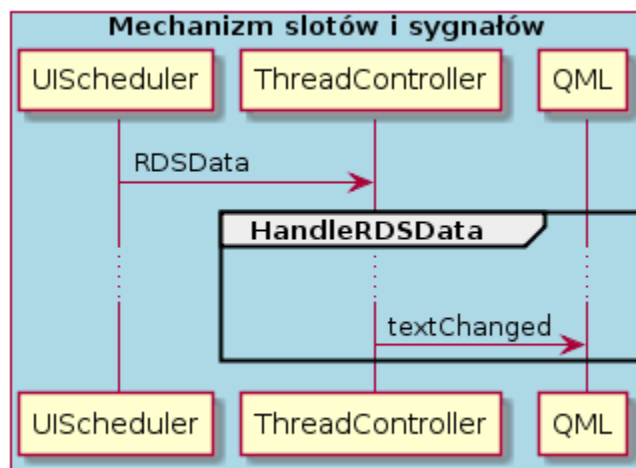
Klasy QUserFICData, QStationInfo, QSpectrumData, atrybuty klasy ThreadController, przechowują odebrane, odpowiednio sformatowane parametry sygnału DAB+. Ze względu na strukturalną budowę, klasy sygnalizują również zmiany pojedynczych pól (patrz rysunki 16 do 20).



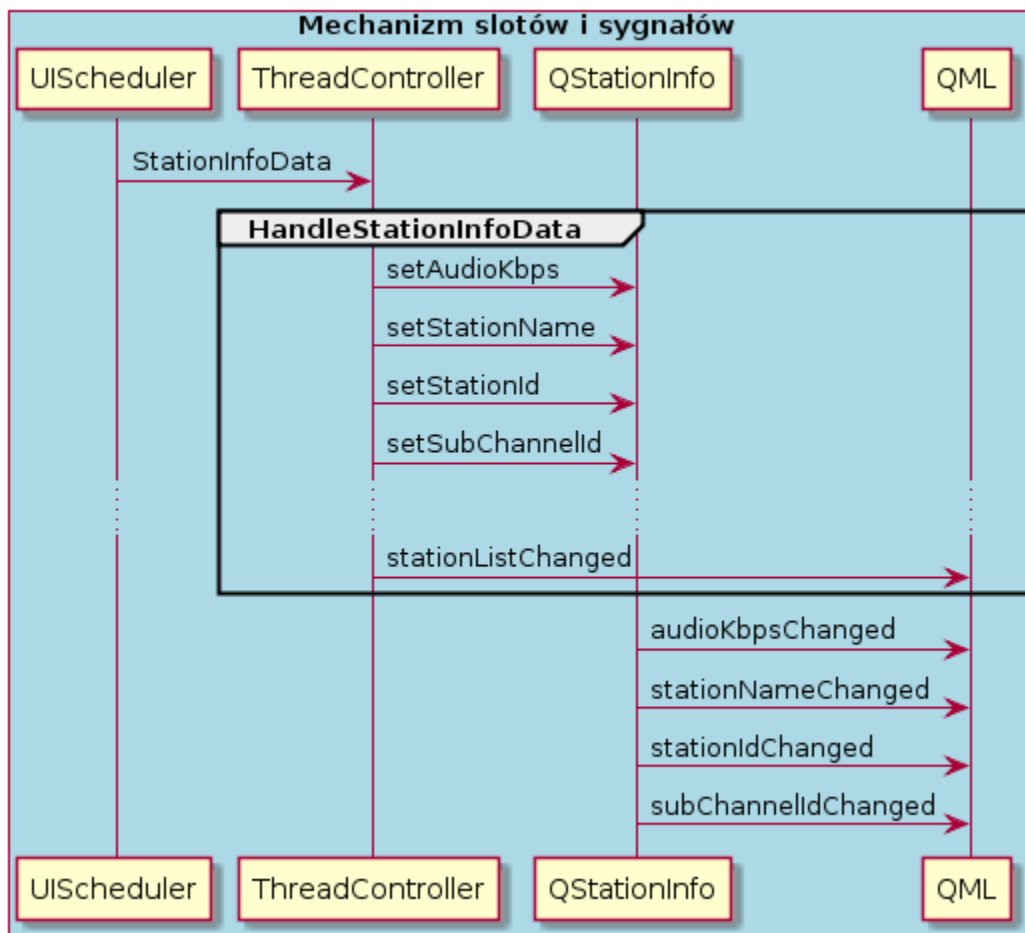
Rys. 16. Diagram sekwencji dla otrzymanych SNRData



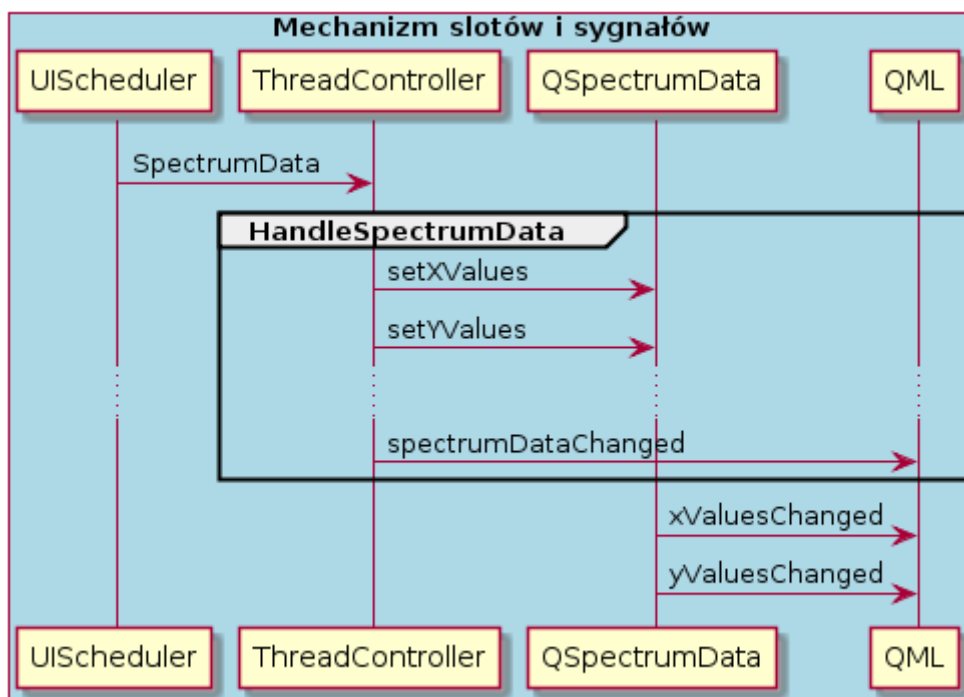
Rys. 17. Diagram sekwencji dla otrzymanych FICExtraData



Rys. 18. Diagram sekwencji dla otrzymanych RDSData

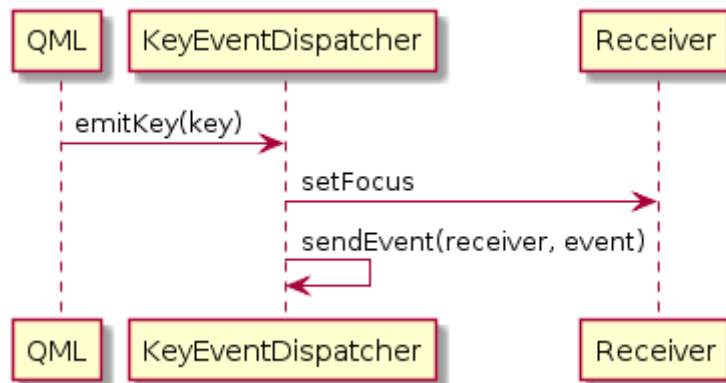


Rys. 19. Diagram sekwencji dla otrzymanych StationInfoData



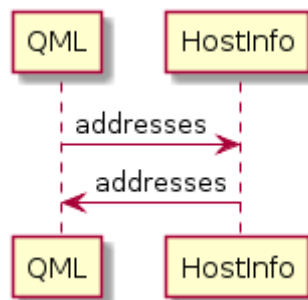
Rys. 20. Diagram sekwencji dla otrzymanych SpectrumData

Dodatkowo, by spełnić pierwotne założenia projektu, koniecznym było wykonanie klasy obsługującej zdarzenia związane z wirtualną klawiaturą dotykową – dyspozytor `KeyEventDispatcher`. W momencie naciśnięcia przycisku wirtualnej klawiatury dotykowej, wyemitowany zostaje sygnał zawierający informacje identyfikujące wybrany znak. Dyspozytor `KeyEventDispatcher` reaguje na wydarzenie, podejmując akcje mające na celu poinformowanie odbiorcy (patrz rysunek 21).



Rys. 21. Diagram sekwencji klasy `KeyEventDispatcher`

Do celów pobierania i wyświetlania adresów IP urządzenia, na którym uruchomiony został interfejs graficzny, zaprojektowana została klasa `HostInfo` (patrz rysunek 22) z publiczną metodą `addresses`.



Rys. 22. Diagram sekwencji klasy `HostInfo`

Aby móc korzystać z mechanizmu slotów i sygnałów platformy programistycznej Qt, koniecznym jest zarejestrowanie każdej klasy w meta systemie Qt. Dla klas będących elementami biblioteki Qt, założenie jest spełnione niejawnie. W przypadku nowych, zdefiniowanych przez użytkownika typów, wymagane jest użycie funkcji `qRegisterMetaType` (patrz tabela 16).

Tabela 16. Sygnatura funkcji `qRegisterMetaType`

```
int qRegisterMetaType(const char * typeName)
```

Źródło: na podstawie [30]

Obiekty klas `QSchedulerConfig`, `ThreadController`, `KeyEventDispatcher` i `HostInfo` znajdują się na najwyższej warstwie abstrakcji. Są one dostępne bezpośrednio w każdym miejscu dokumentu QML zgodnie z wzorcem projektowym singleton [31] (patrz tabela 17).

Tabela 17. Rejestrowanie obiektów klas `QschedulerConfig`, `ThreadController`, `KeyEventDispatcher` i `HostInfo` w dokumencie QML

```
engine.rootContext()->setContextProperty("schedulerConfig", &scheduler_config);  
engine.rootContext()->setContextProperty("threadController", &thread_controller);  
engine.rootContext()->setContextProperty("keyEventDispatcher",  
&key_event_dispatcher);  
engine.rootContext()->setContextProperty("hostInfo", &host_info);
```

Główny dokument QML, będący opisem interfejsu graficznego programu, ładowany jest do silnika aplikacji za pomocą instrukcji `load` (patrz tabela 18).

Tabela 18. Sygnatura funkcji `load`

```
void QQmlApplicationEngine::load(const QUrl & url)
```

Źródło: [32]

Biblioteka Qt wykorzystuje własne typy danych, często o rozszerzonej funkcjonalności względem typów podstawowych. Konwersje pomiędzy podstawowymi typami danych, które zachodzą automatycznie przedstawiono w tabeli 19.

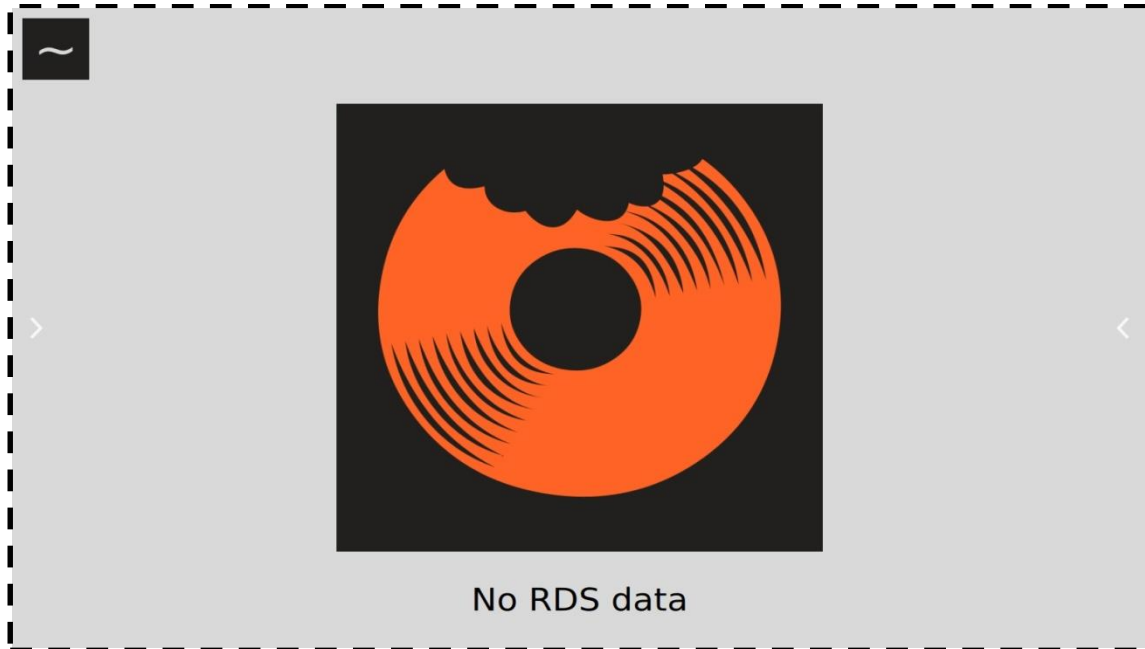
Tabela 19. Skrócona tabela przedstawiająca konwersje pomiędzy podstawowymi typami

| Typ | Typ QML |
|-------------------|---------|
| bool | bool |
| unsigned int, int | int |
| double | double |
| float, qreal | real |
| QString | string |
| QUrl | url |

Źródło: opracowano na podstawie [33]

Dla złożonych typów koniecznym było zaprojektowanie funkcji pośredniczących pomiędzy kodem C++ a QML. Korzystając z mechanizmu konwersji niejawnych i zasad konwersji QML uzyskano zakładane cele tj. całkowitą synergię na styku C++ i QML. Przechowywane dane są odczytywane i zapisywane zarówno z kodu C++ jak i QML za pomocą tych samych funkcji. Do minimum ograniczono operacje wymagające kopiowania dużej ilości pamięci. W przypadku typów złożonych, dane odczytywane i zapisywane są z funkcji operujących na wskaźnikach do wcześniej zarezerwowanych obszarów. W przypadku listy stacji i próbek widmowej gęstości mocy, użyta została właściwość automatycznej konwersji do tablicy JavaScript: dla listy stacji listy obiektów `QList<QObject *>`, dla próbek widma listy obiektów `QList<QVariant>`.

Główny ekran aplikacji został zaprezentowany na rysunku 23. Na środku znajduje się miejsce na domyślny lub aktualnie przesyłany przez stację obraz, pod obrazem – miejsce na odebrane wiadomości tekstowe. W lewym górnym rogu umieszczony jest przycisk uruchamiający widok wykresu widmowej gęstości mocy. Strzałka (przycisk) po lewej stronie rozwija menu kontekstowe aplikacji, strzałka (przycisk) po prawej – listę dostępnych stacji.



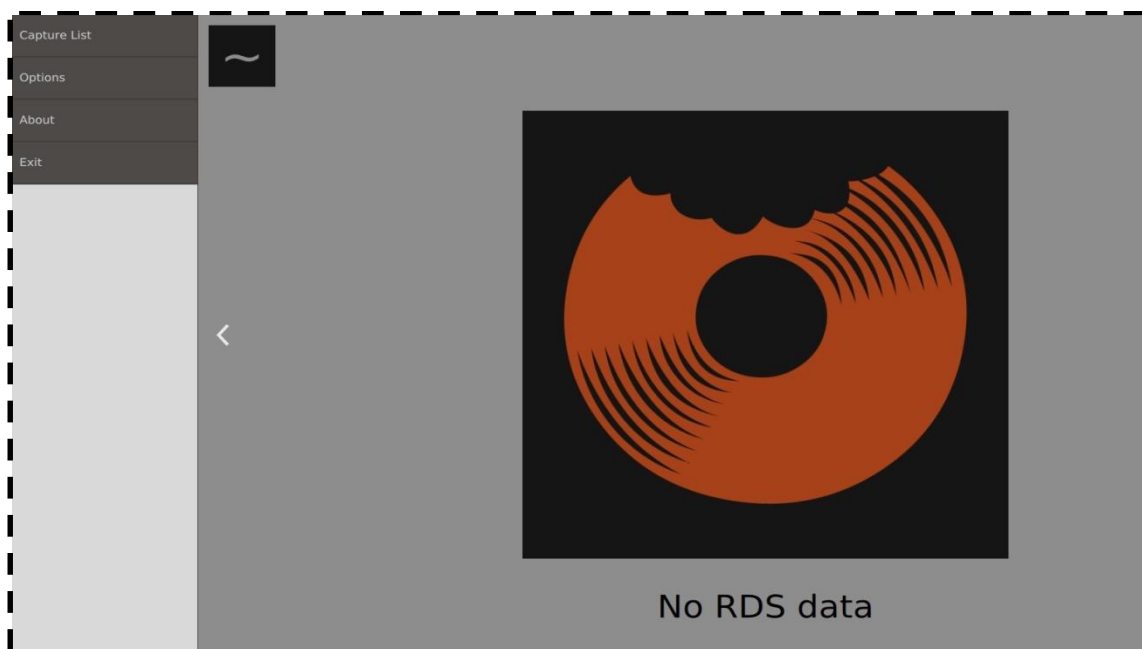
Rys. 23. Główny ekran aplikacji

Na rysunku 24 pokazano widok widmowej gęstości mocy sygnału. Na środku ekranu wyświetlone jest widmo sygnału. W lewym górnym rogu umieszczony jest przycisk powrotu do głównego ekranu aplikacji (obraz głównego ekranu), w prawym górnym rogu wypisywana jest aktualna wartość parametru SNR.



Rys. 24. Ekran widoku spectrum

Menu kontekstowe aplikacji z dostępnymi opcjami przedstawiono na rysunku 25.



Rys. 25. Boczne menu kontekstowe

Podmenu wyboru źródła sygnału pokazano na rysunku 26. W lewym górnym rogu umieszczony jest przycisk powrotu do ekranu głównego aplikacji (lub widoku widma, jeśli był uruchomiony). Na tym ekranie możliwe jest wybranie źródła danych: odtwarzania z pliku lub tunera USB DVB-T.



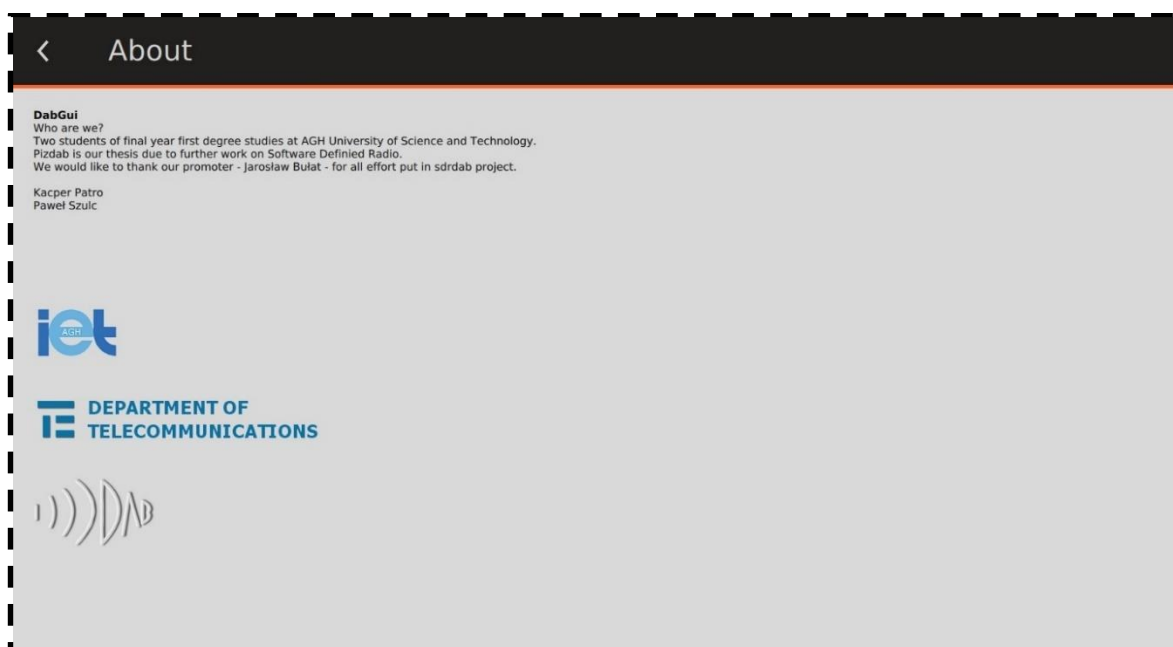
Rys. 26. Podmenu wyboru źródła sygnału

Podmenu opcji przedstawione na rysunku 27 uożliwia ustawienie częstotliwości próbkowania i częstotliwość nośnej za pomocą widocznej na dole klawiatury wirtualnej. Pod nagłówkiem *IP addresses* wyświetlone są adresy IP urządzenia.



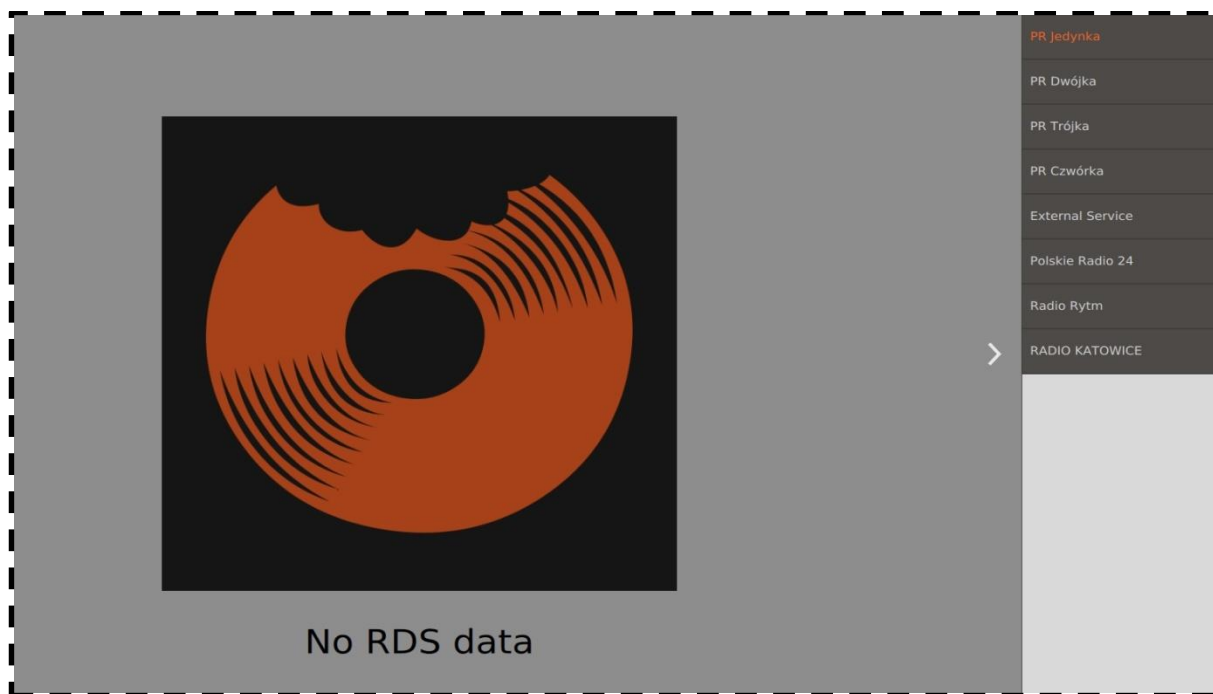
Rys. 27. Podmenu opcji

Na rysunku 28 zaprezentowano podmenu *About*.



Rys. 28. Podmenu *About*

List dostępnych stacji z aktualnie odtwarzaną stacją zaprezentowano na rysunku 29.



Rys. 29. Lista dostępnych stacji z aktualnie odtwarzaną stacją

3.5. Aplikacja pilot

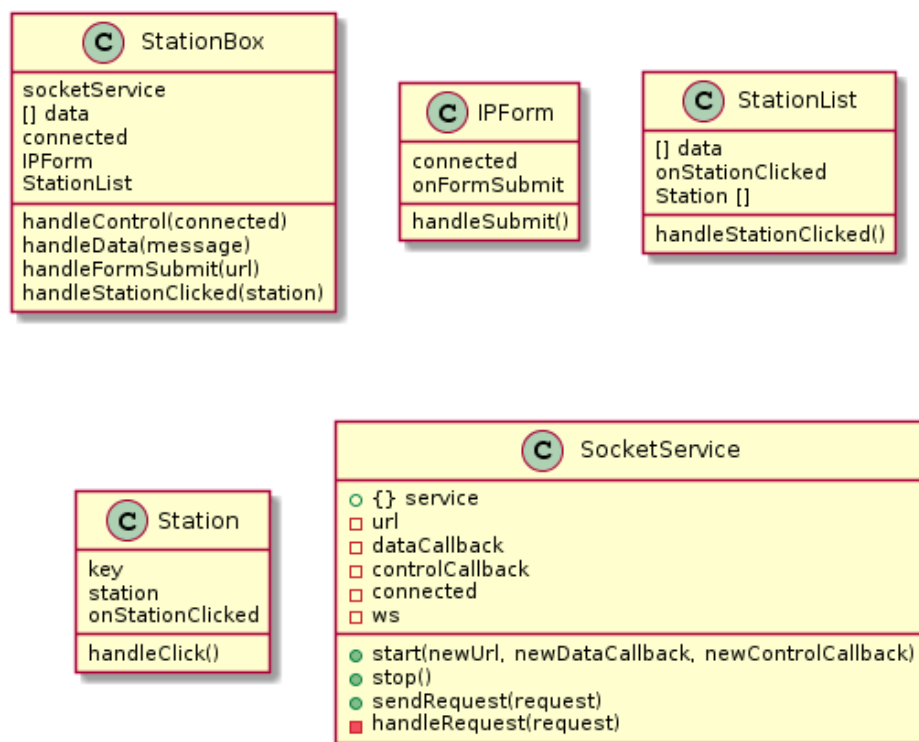
Do zaprojektowania zdalnej aplikacji realizującej funkcję pilota, pozwalającego na zmianę aktualnie odtwarzanej stacji, wybrano bibliotekę ReactJS [34]. Jest ona przeznaczona do tworzenia interfejsów użytkownika w oparciu o języki JavaScript i HTML. Cechuje ją deklaratywność, szybkość i elastyczność. Tak jak QML, ReactJS automatycznie aktualizuje widok elementu w następstwie zmiany jego właściwości. ReactJS korzysta ze składni nazywanej JSX.

ReactJS wybrany został ze względu na szybkość działania i przejrzystość kodu. Wynikowy kod aplikacji napisanej w ReactJS jest zrozumiały dla każdej osoby mającej doświadczenie z językiem JavaScript. Deklaratywność ReactJS, w porównaniu na przykład do biblioteki jQuery, systematyzuje poszczególne elementy aplikacji. Kolejne komponenty są od siebie wyraźnie odgraniczone, co jest zadaniem o wiele trudniejszym do osiągnięcia w jQuery. W dokumencie JSX zdefiniowane są wyłącznie funkcje widoku w odniesieniu do modelu MVC (ang. *Model-View-Controller*). Domyślnie, React implementuje wiązania jednokierunkowe, to znaczy od rodzica do potomka. Przekazane własności dostępne są w obiekcie potomka w polu `this.props`. Własności `this.props` są niezmiennie – należą do obiektu rodzica i są tylko przez niego modyfikowalne. Z drugiej strony, pole `this.state` jest prywatne dla każdego komponentu z osobna i zmieniane może być przez funkcję `this.setState`. Kiedy zmienia się stan komponentu, widok elementu i elementów zależnych jest automatycznie aktualizowany. Taki sposób budowania aplikacji usprawnia proces utrzymania kodu, aczkolwiek może być w niektórych przypadkach niewystarczający. Nowe elementy w bibliotece ReactJS tworzone są za pomocą funkcji `React.createClass`. Jedynym wymaganym elementem jest posiadanie przez nowo utworzony element metody `render`, używanej do generowania struktury HTML.

Ze względu na to, że aplikacja napisana została w językach JavaScript i HTML, może zostać uruchomiona w każdej przeglądarce internetowej. Bezdyskusyjną zaletą takiego rozwiązania jest fakt, że każde urządzenie wyposażone w przeglądarkę ma możliwość sterowania biblioteką `sdrdab`. Biorąc pod uwagę popularność oprogramowania Web, na rynku pojawiły się platformy programistyczne pozwalające stworzyć tzw. *natywną* aplikację mobilną z wcześniej przygotowanego kodu JavaScript, CSS, HTML. Jednym z popularniejszych narzędzi tego typu jest PhoneGap. Niewątpliwym atutem narzędzia PhoneGap jest liczba wspieranych platform, co jest ważne ze względu na rozmiar rynku

mobilnego. Dzięki Adobe® PhoneGap™ Build [35], rola programisty w przygotowywaniu środowiska uruchomieniowego jest ograniczona do minimum. Publicznie dostępne repozytorium z poprawnie skonfigurowaną platformą PhoneGap jest wystarczające, by narzędzie PhoneGap™ Build przygotowało pliki binarne na trzy najpopularniejsze platformy: Android, iOS i WP. Tak przygotowane binaria gotowe są do wgrania na urządzenia mobilne.

Głównym komponentem w strukturze pilota jest element StationBox (patrz rysunek 30), który hermetyzuje elementy IPForm i StationList. Komponent StationBox przechowuje listę stacji wraz z aktualnie odtwarzaną stacją. Każda stacja opisana jest przez pięć pól: station_tile, kbps, station_id, sub_channel_id i current_station (patrz tabela 20).

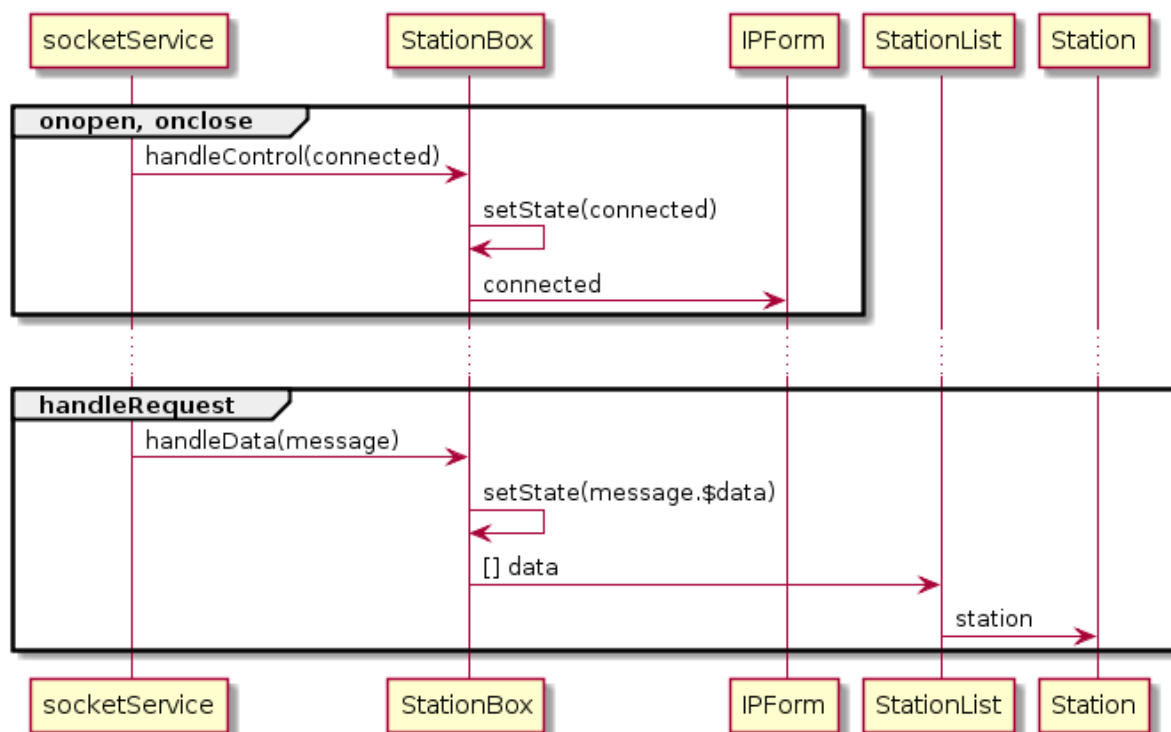


Rys. 30. Uproszczona struktura klas JavaScript

Tabela 20. Struktura opisująca stację

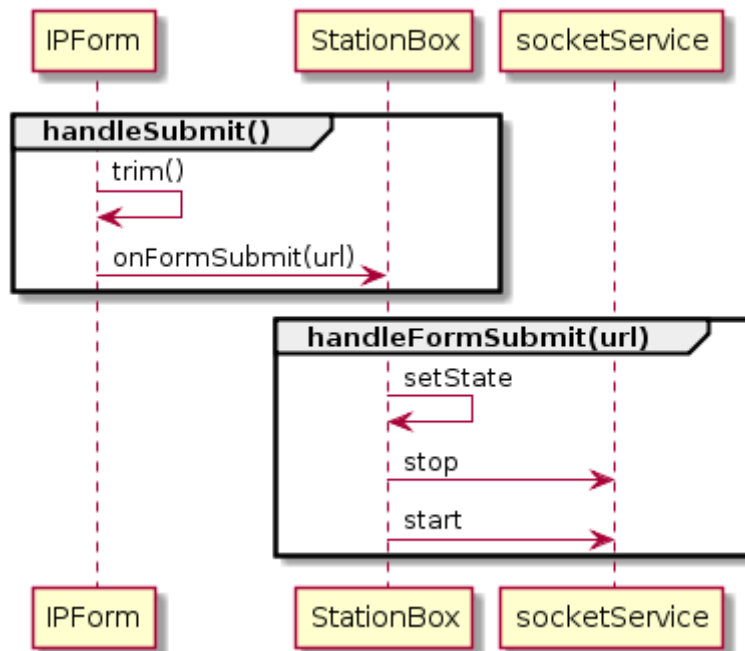
| Pole | Opis |
|-----------------|--|
| station_title | tytuł stacji |
| kbps | szybkość transmisji |
| station_id | liczbowy identyfikator stacji |
| sub_channel_id | liczbowy identyfikator podkanału |
| current_station | pole logiczne indukujące, czy stacja jest aktualnie odtwarzaną |

Element `StationBox` dba o aktualność wyświetlanych danych: wysyła i odbiera komunikaty serwera, zarządza lokalnym gniazdem `WebSocket` (patrz rysunek 31).



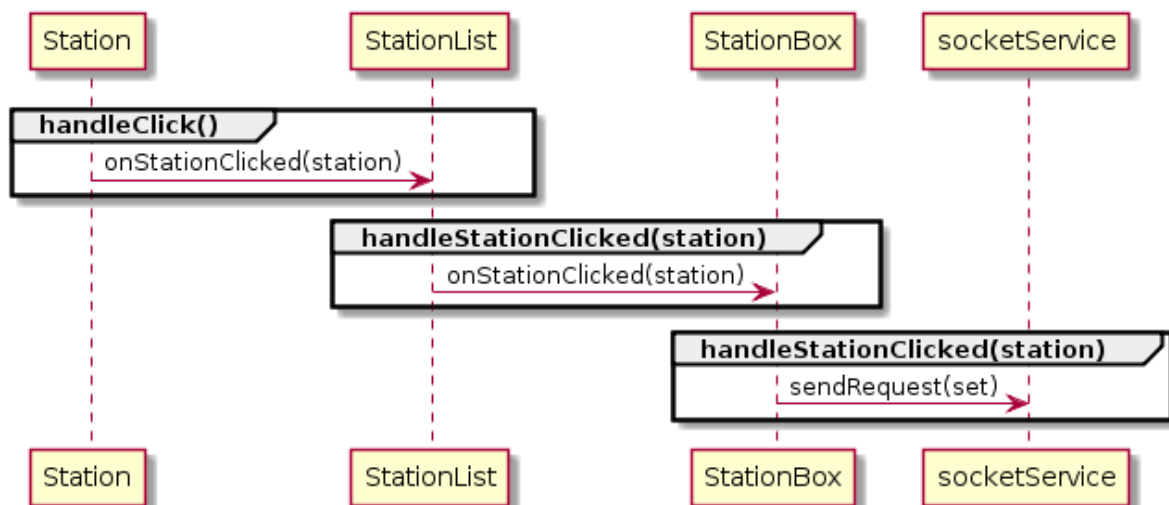
Rys. 31. Diagram sekwencji dla zdarzeń typu `onopen` (`onopen`), `onclose` (`onclose`), `onmessage` (`handleRequest`)

Klasa `IPForm` uwidacznia formularz adresu IP. Obiekt klasy `IPForm` waliduje wprowadzone przez użytkownika dane, które następnie przekazuje, za pomocą otrzymanej od rodzica funkcji wywołania zwrotnego, do obiektu `StationBox` (patrz rysunek 32).



Rys. 32. Diagram sekwencji dla formularza adresu IP

Obiekty klasy Station jest widoczną reprezentacją jednej dostępnej stacji. Station sygnalizuje wydarzenia związane z kliknięciem na polu reprezentującym stację. StationList reprezentuje listę stacji. Dla każdej stacji w liście stacji otrzymanej od rodzica, StationList tworzy widoczny element Station. Za pomocą otrzymanych funkcji wywołania zwrotnego, sygnały są asynchronicznie propagowane od Station aż do StationBox (patrz rysunek 33).



Rys. 33. Diagram sekwencji dla propagacji sygnału stationClicked

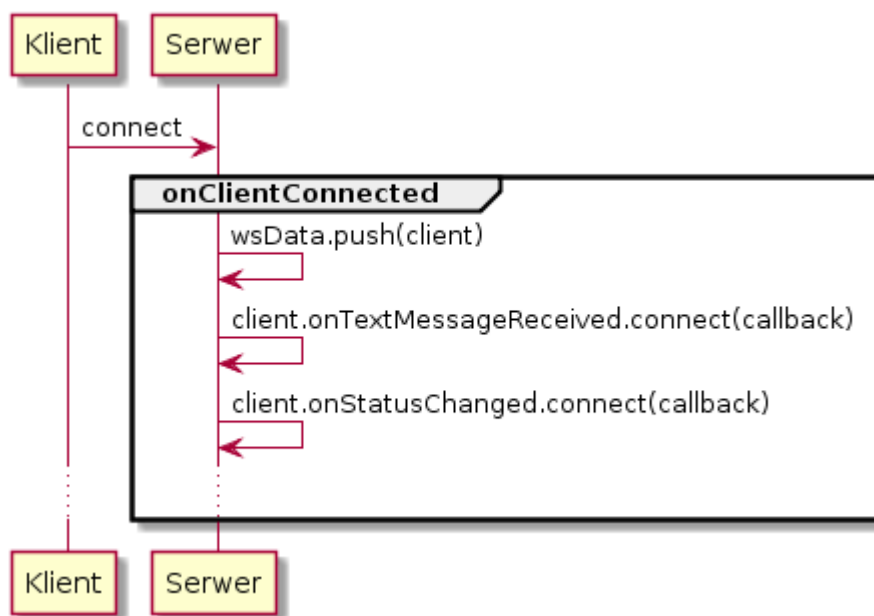
Komunikacja z serwerem odbywa się z pomocą klasy `SocketService`. `SocketService` składa się z dwóch warstw – sterowania i danych. Obiekty korzystające z klasy `SocketService` dostarczają dwie funkcje wywołania zwrotnego: jedną dla warstwy sterowania i jedną dla warstwy danych. Funkcja odpowiadająca za sterowanie informuje, za pomocą pola logicznego, obiekty korzystające z klasy `SocketService` o stanie połączenia (połączony lub niepołączony). Funkcja odpowiadająca za płaszczyznę danych koduje dane w formacie JSON. Za nadawanie wiadomości natomiast odpowiada funkcja `sendRequest`.

Zdefiniowano trzy typy wiadomości: `set`, `init`, `up`. Nadanie wiadomości `set` jest bezpośrednim następstwem kliknięcia na polu reprezentującym stację. Od strony serwera, wiadomość `set` prowadzi do zmiany aktualnie odtwarzanej stacji. Wiadomość `init` jest generowana dla każdego nowego klienta dołączającego do serwera i zawiera pełny opis dostępnych stacji. Wiadomość `up` generowana jest w wyniku jakiegokolwiek zmiany w polach opisujących listę stacji i wysyłana do każdego podłączonego klienta `WebSocket`. Wiadomość JSON składa się z dwóch pól, `$type` i `$data`, których zawartość przedstawiona jest w tabeli 21.

Tabela 21. Struktura przesyłanych wiadomości

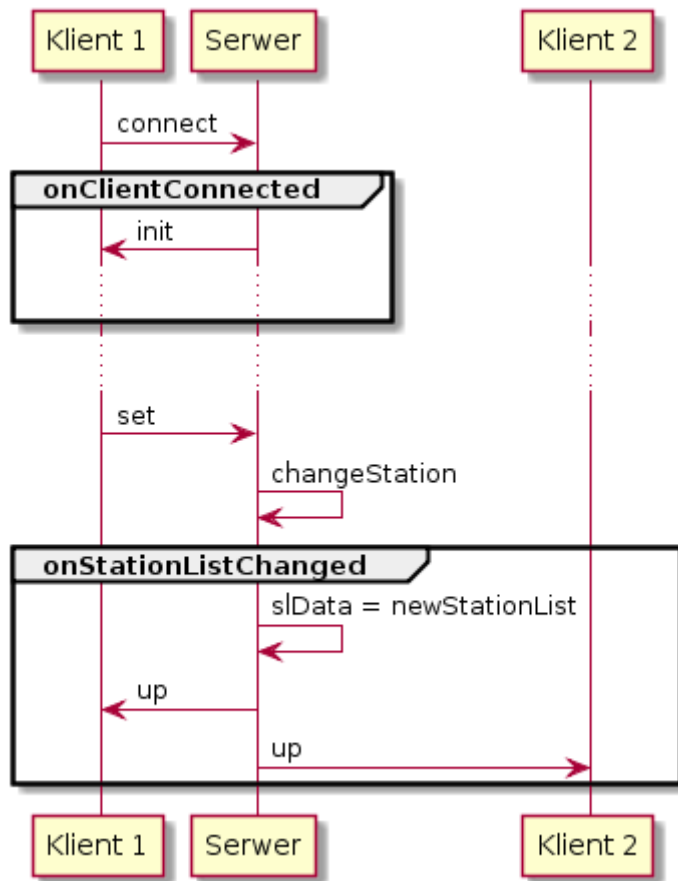
| \$type | \$data |
|---------------|--|
| set | sub_channel_id |
| init | [[{station_title, kbps, station_id, sub_channel_id, current_station}, ...] |
| up | [[{station_title, kbps, station_id, sub_channel_id, current_station}, ...] |

Serwer uruchamiany jest jednocześnie z graficznym interfejsem użytkownika. Do implementacji serwera użyty został moduł `QtWebSockets` platformy programistycznej Qt. Serwer posługuje się dwoma strukturami: `slData` (ang. *station list*) przechowującą aktualną listę dostępnych stacji z aktualnie odtwarzaną stacją oraz `wsData` (ang. *web socket*), zawierającą informacje o wszystkich podłączonych klientach. W trakcie dołączania klienta (patrz rysunek 34), serwer aktualizuje strukturę `wsData` i łączy odpowiednie funkcje wywołania zwrotnego. Funkcja wywołania zwrotnego dla wydarzenia `onTextMessageReceived` rozpoznaje otrzymane od klienta żądanie zmiany stacji, propagując przy użyciu metody `changeStation` obiektu klasy `ThreadController` komunikat do biblioteki `sdrdab`. Funkcja wywołania zwrotnego dla wydarzenia `onStatusChanged` reaguje na zmiany statusu (zamknięcia) połączenia i aktualizuje strukturę `wsData`. Serwer nasłuchuje na porcie 8080.



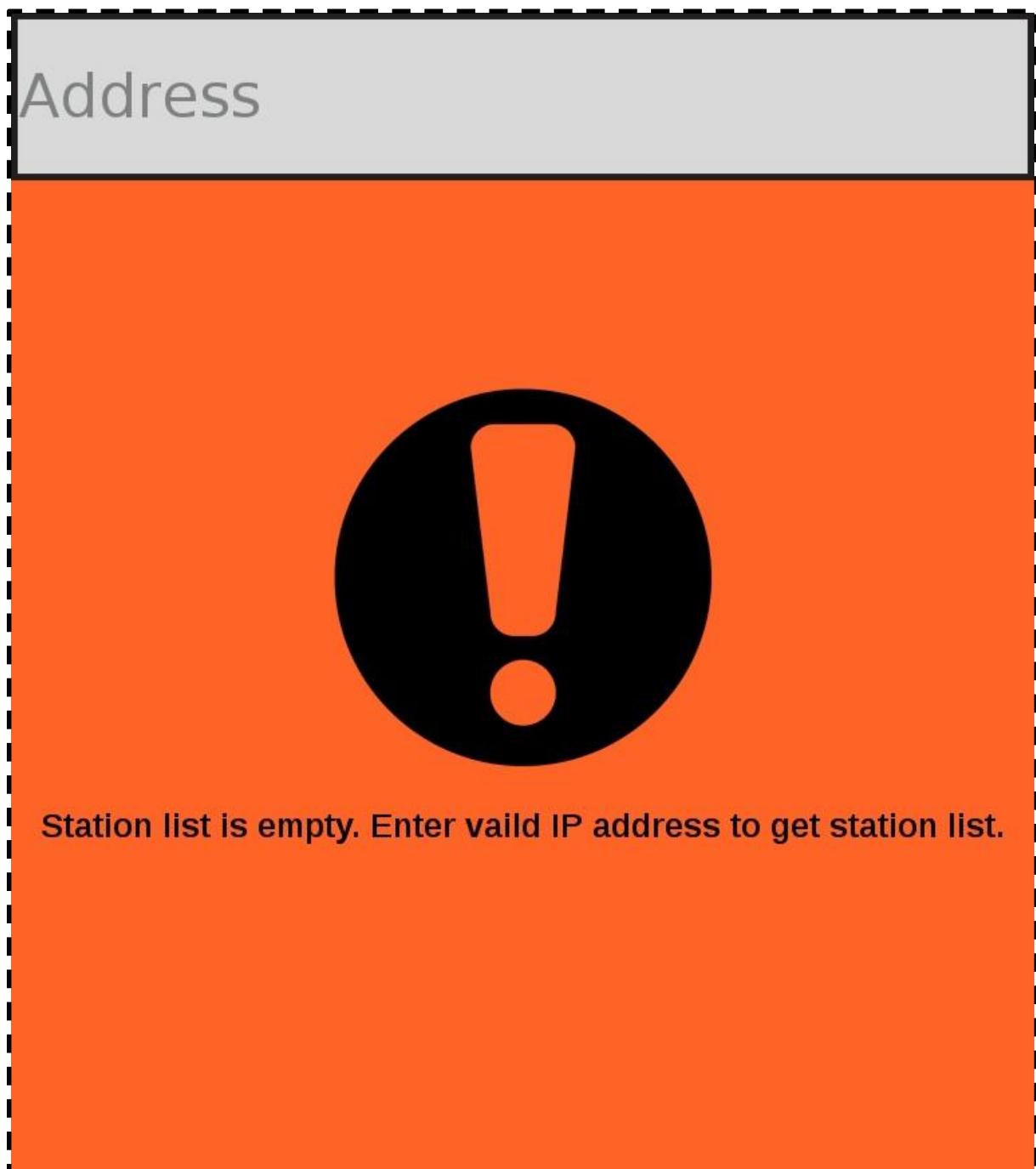
Rys. 34. Obsługa dołączającego klienta

Serwer nasłuchuje na sygnał `onStationListChanged` od obiektu klasy `ThreadController`. W przypadku zmiany któregoś z pól opisującego listę stacji, uaktualniana jest struktura `slData` a aktualizacja wysyłana jest do wszystkich klientów (patrz rysunek 35).



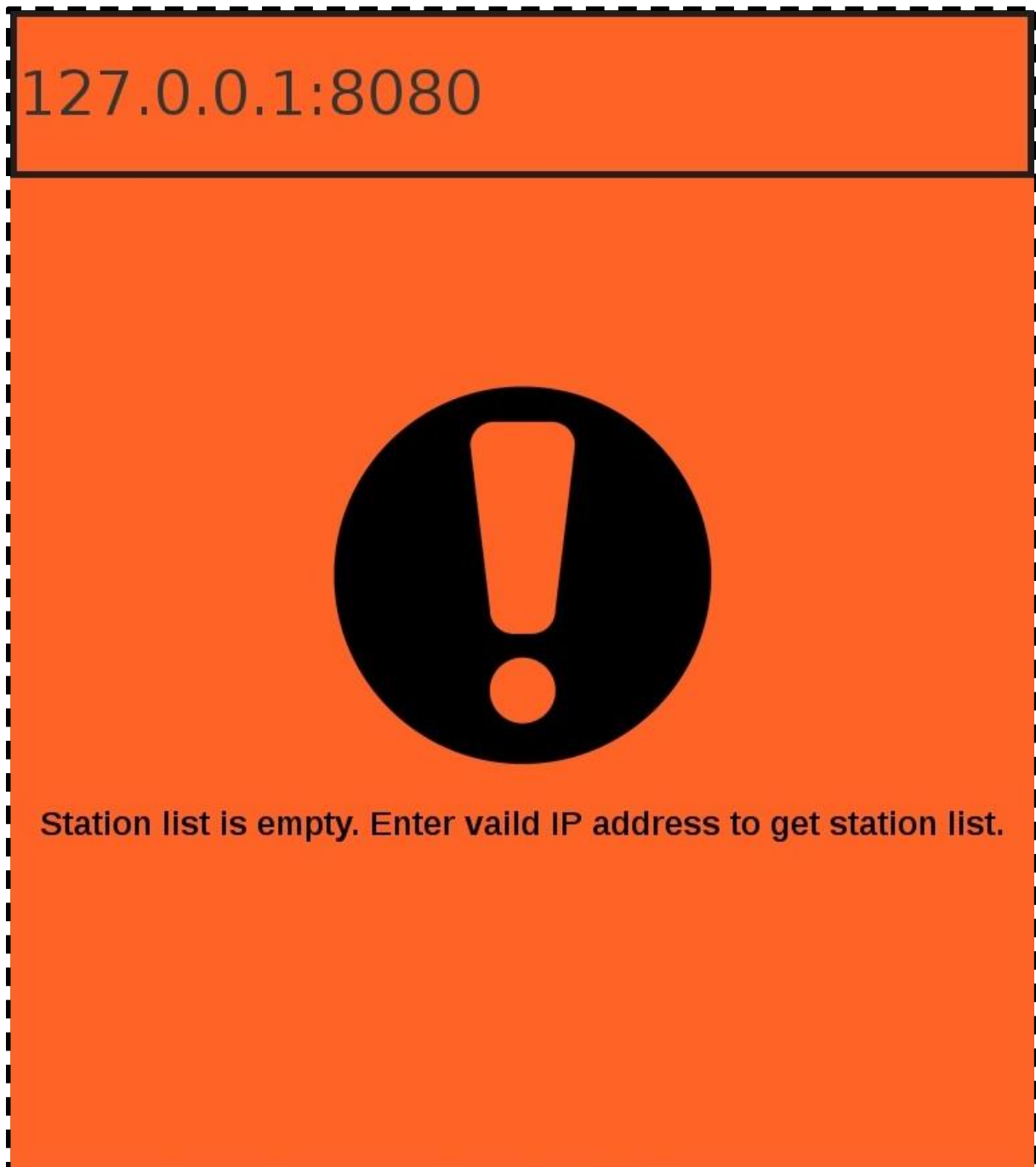
Rys. 35. Propagacja aktualizacji do wszystkich podłączonych klientów

Główny ekran aplikacji przedstawiono na rysunku 36. Szare tło paska *Address* sygnalizuje brak aktywnego połączenia z serwerem.



Rys. 36. Główny ekran aplikacji, szary pasek adresu sygnalizuje brak połączenia z serwerem

Pomarańczowe tło paska *Address* wskazuje na aktywne połączenie z serwerem (patrz rysunek 37).



Rys. 37. Główny ekran aplikacji, pomarańczowy pasek adresu sygnalizuje aktywne połączenia z serwerem

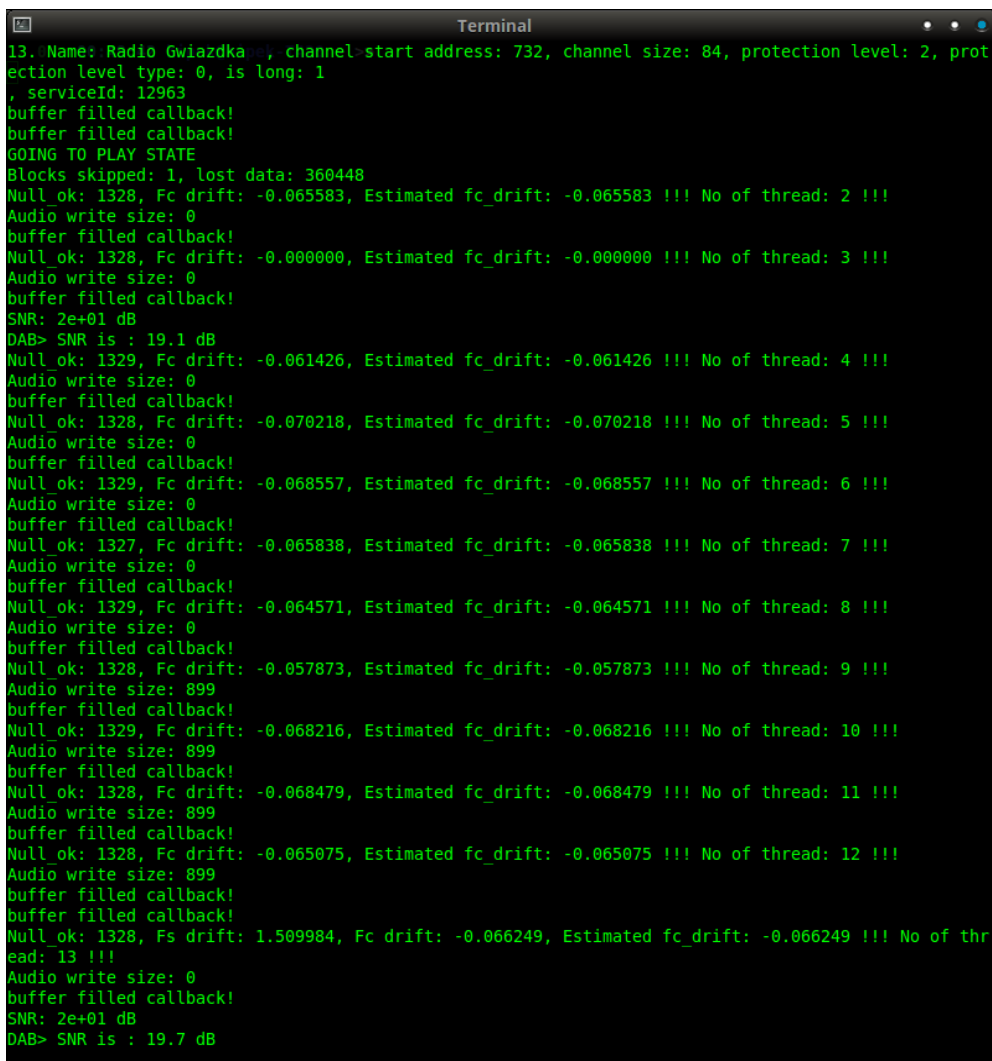
Widok listy stacji z aktualnie odtwarzaną stacją przedstawiono na rysunku 38.



Rys. 38. Lista dostępnych stacji i aktualnie odtwarzana stacja (PR Jedyńka)

4. Testy wydajności biblioteki sdrdab

Pierwszą próbę uruchomienia podjęto na urządzeniu Raspberry Pi 1. Po udanej kompilacji i pierwszym uruchomieniu aplikacji, platforma została jednak odrzucona. Przeszarżała architektura procesora, niska częstotliwość taktowania i brak rozszerzeń Neon do przetwarzania wektorowego uniemożliwiały płynne odtwarzanie dźwięku. Podczas testów związanych z samą biblioteką sdrdab używana została załączona pomocnicza aplikacja sdrdab-cli. Dzięki temu możliwe było równoległe rozwijanie interfejsu graficznego oraz samej biblioteki. W połączeniu ze standardowymi elementami środowiska Linux (m. in. grep, tail, tee) okazała się przydatnym narzędziem do obserwacji interesujących parametrów w czasie rzeczywistym. Zrzut ekranu pokazano na rysunku 39.



```
Terminal
13. Name: Radio Gwiazdka, channel start address: 732, channel size: 84, protection level: 2, protection level type: 0, is long: 1, serviceId: 12963
buffer filled callback!
buffer filled callback!
GOING TO PLAY STATE
Blocks skipped: 1, lost data: 360448
Null ok: 1328, Fc drift: -0.065583, Estimated fc_drift: -0.065583 !!! No of thread: 2 !!!
Audio write size: 0
buffer filled callback!
Null ok: 1328, Fc drift: -0.000000, Estimated fc_drift: -0.000000 !!! No of thread: 3 !!!
Audio write size: 0
buffer filled callback!
SNR: 2e+01 dB
DAB> SNR is : 19.1 dB
Null ok: 1329, Fc drift: -0.061426, Estimated fc_drift: -0.061426 !!! No of thread: 4 !!!
Audio write size: 0
buffer filled callback!
Null ok: 1328, Fc drift: -0.070218, Estimated fc_drift: -0.070218 !!! No of thread: 5 !!!
Audio write size: 0
buffer filled callback!
Null ok: 1329, Fc drift: -0.068557, Estimated fc_drift: -0.068557 !!! No of thread: 6 !!!
Audio write size: 0
buffer filled callback!
Null ok: 1327, Fc drift: -0.065838, Estimated fc_drift: -0.065838 !!! No of thread: 7 !!!
Audio write size: 0
buffer filled callback!
Null ok: 1329, Fc drift: -0.064571, Estimated fc_drift: -0.064571 !!! No of thread: 8 !!!
Audio write size: 0
buffer filled callback!
Null ok: 1328, Fc drift: -0.057873, Estimated fc_drift: -0.057873 !!! No of thread: 9 !!!
Audio write size: 899
buffer filled callback!
Null ok: 1329, Fc drift: -0.068216, Estimated fc_drift: -0.068216 !!! No of thread: 10 !!!
Audio write size: 899
buffer filled callback!
Null ok: 1328, Fc drift: -0.068479, Estimated fc_drift: -0.068479 !!! No of thread: 11 !!!
Audio write size: 899
buffer filled callback!
Null ok: 1328, Fc drift: -0.065075, Estimated fc_drift: -0.065075 !!! No of thread: 12 !!!
Audio write size: 899
buffer filled callback!
Null ok: 1328, Fs drift: 1.509984, Fc drift: -0.066249, Estimated fc_drift: -0.066249 !!! No of thread: 13 !!!
Audio write size: 0
buffer filled callback!
SNR: 2e+01 dB
DAB> SNR is : 19.7 dB
```

Rys. 39. Zrzut ekranu konsoli programu sdrdab-cli-bin

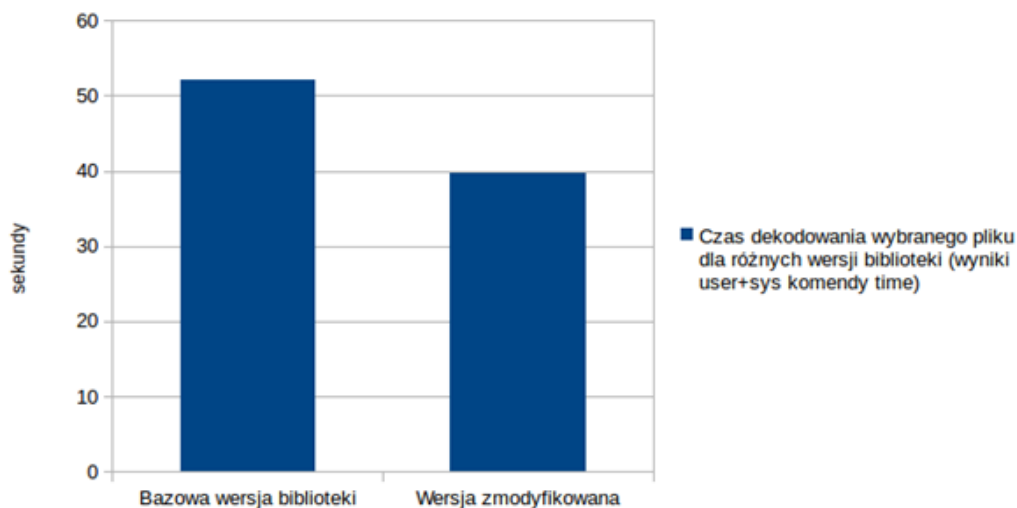
Podczas porównywania różnych wersji skompilowanej biblioteki, istotnymi parametrami była liczba odrzuconych buforów (informacja o ewentualnych stratach pojawiała się na ekranie konsoli) oraz płynność odtwarzanego dźwięku. Intuicyjnie, płynniejszy dźwięk świadczył o lepszej wydajności biblioteki.

Wydajność fragmentu algorytmu Viterbiego, odpowiadającego za mnożenie wektorów, sprawdzana była za pomocą funkcji pomocniczej `time`. Wyodrębniono fragmenty kodu i skompilowano bez flag optymalizacyjnych. W tabeli 22 przedstawiono czasy wykonania 100 milionów powtórzeń fragmentu kodu.

Tabela 22. Porównanie czasu wykonania fragmentów kodu

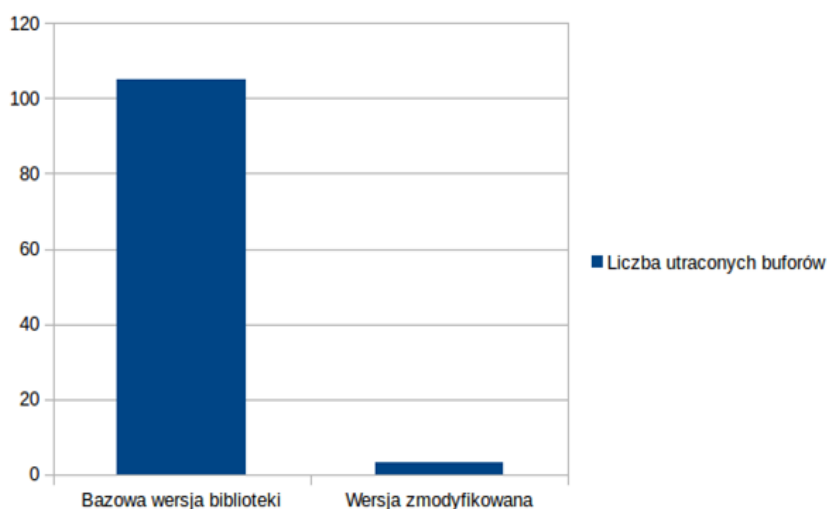
| Wynik funkcji <code>time</code> | Kod w języku C++ (sekundy) | Kod w języku assembler (sekundy) |
|--|---------------------------------------|---|
| real | 11,685 | 3,795 |
| user | 11,670 | 3,770 |
| sys | 0,000 | 0,020 |

Z tabeli 22 wynika, że wydajność fragmentu kodu, będącego wąskim gardłem całej biblioteki, wzrosła trzykrotnie. Podobnie, za pomocą funkcji `time`, zmierzono całkowity czas dekodowania identycznego pliku danych surowych dla obu wersji biblioteki (patrz rysunek 40). Czas trwania dźwięku zakodowanego w tym pliku wynosił trzydzieści sekund. Odtwarzanie było płynne w obu przypadkach, gdyż podane wartości są przeliczane na jeden rdzeń procesora (w przypadku wykorzystania wielu rdzeni, wskazana wartość może być większa niż czas rzeczywisty).

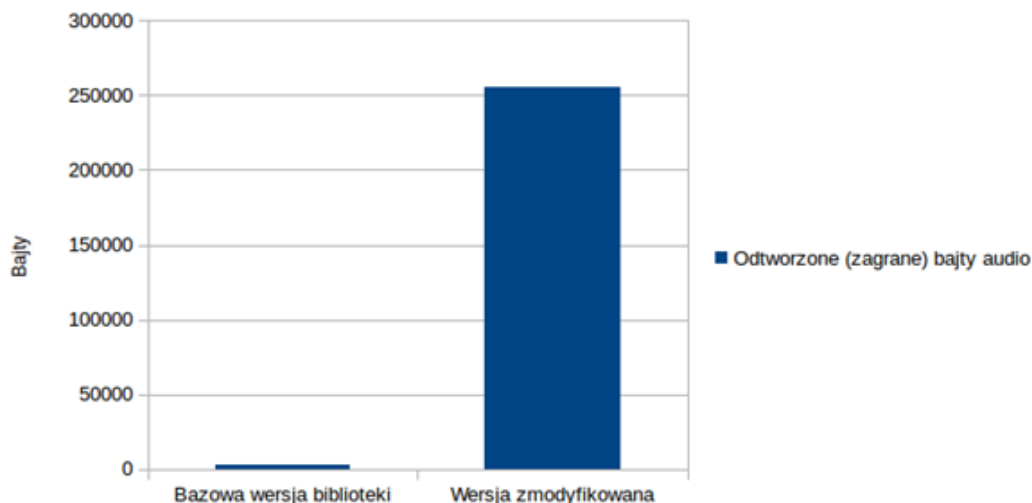


Rys. 40. Porównanie czasu dekodowania pliku dla obu wersji biblioteki

Następne testy dotyczyły ilości utraconych danych podczas odtwarzania dźwięku z powietrza. Nasłuchiwano na multipleksie Polskiego Radia w Krakowie. Napisane zostały pomocnicze skrypty w języku Perl, które na podstawie logów z aplikacji wyliczały interesujące parametry. Podczas tego testu zebrano 300 buforów surowych próbek. Rysunek 41 przedstawia różnice w liczbie utraconych buforów wejściowych, a rysunek 42 liczbę poprawnie zdekodowanych bajtów audio. Dla podstawowej wersji biblioteki odrzucony jest co trzeci bufor co skutkuje stratą niemal całego audio. Zaobserwowany wzrost wydajności jest na tyle znaczny, że zrezygnowano z uwzględniania bazowej wersji biblioteki w kolejnych testach.



Rys. 41. Porównanie sumy straconych buforów obu wersji bibliotek



Rys. 42. Porównanie zdekodowanych bajtów dźwięku

Postanowiono dokładniej zmierzyć płynność dźwięku dla różnych stacji. Dane zostały zebrane z powietrza. Obserwacje rozpoczęto w 10 sekundzie dekodowania multipleksu, a zakończono w 142. W tym czasie zapełnionych zostało 1500 buforów wejściowych. Na podstawie przepływności bitowej stacji, oszacowano liczbę bajtów dźwięku, które powinny zostać zdekodowane przez 132 sekundy, aby odtwarzanie było płynne i porównano ze zmierzonymi wartościami. Na tej podstawie obliczono czas ciszy i zaproponowano go jako miarę płynności dźwięku (im dłuższy czas ciszy, tym dźwięk mniej płynny). Średni SNR sygnału wejściowego dla tych pomiarów wyniósł 22dB. Wyniki przedstawia tabela 23.

Tabela 23. Wynik testów podczas dekodowania wybranych stacji

| Stacja | Bajty dźwięku w ramce | Liczba buforów odrzuconych na wejściu | Odrzucone super ramki z powodu błędnego CRC | Szacowany czas ciszy (sekundy) |
|----------------|------------------------------|--|--|---------------------------------------|
| PR Jedyńka | 1559 | 0 | 0 | 0,91 |
| PR Dwójka | 1779 | 2 | 1 | 3,78 |
| Radio Poland | 889 | 8 | 0 | 42,67 |
| Radio Kraków | 1219 | 0 | 0 | 0,21 |
| Radio Dzieciom | 999 | 0 | 0 | -0,02 |

Tylko jedna ramka została odrzucona z powodu błędów wykrytych podczas kontroli parzystości. Wnioskować więc można, że przy oszacowanym parametrze SNR, jakość jest wystarczająca do poprawnego dekodowania sygnału. Zebrane dane nie pozwalają stwierdzić zależności między przepływnością bitową strumienia audio kanału, a liczbą odrzuconych buforów czy płynnością odtwarzania. Dla pomiaru dotyczącego stacji Radio Kraków można zaobserwować nieproporcjonalny wpływ odrzuconych buforów wejściowych na stracony dźwięk. Po dokładniejszej analizie zauważono, że ciąg strat buforów sprawiał, że nie dekodowano audio w kolejnych ramkach, nawet gdy dane nie są tracone. Nie udało się zidentyfikować przyczyny tego problemu. Przy pomiarze dotyczącego stacji Radio Dzieciom widać ujemny szacowany czas ciszy. Biblioteka nie pozwala na obliczenie dokładnego czasu przez jaki dźwięk jest odtwarzany – związane jest to z obranym okresem obserwacji. Pomiar oparte są o liczbę bajtów wpisanych do bufora dźwięku – nie znaczy to, że identyczna liczba bajtów jest odegrana na głośnikach w tym samym czasie. Przy stacji PR Jedyńka czas ciszy wynosi niemal sekundę mimo braku odrzuconych ramek z powodu błędów kontroli parzystości. Podczas analizy logów zaobserwowano brak zapisanego audio do bufora przy kilku pojedynczych ramkach. Biblioteka nie informuje o przyczynie braku danych audio.

Aplikację debugowano za pomocą narzędzi Valgrind, gdb i QtCreator. W czasie testów wykryto i poprawiono szereg błędów związanych z niepoprawną inicjalizacją obiektów typu mutex, conditional variable, co przedstawiono w tabeli 24. Niepoprawny sposób użycia tych obiektów prowadził do powstawania zakleszczeń i błędów naruszeń pamięci, które skutecznie wyeliminowano.

Tabela 24. Fragment wyniku programu Valgrind

```
==23670== Thread 2:  
==23670== Conditional jump or move depends on uninitialised value(s)  
==23670==   at 0x52BDC30: pthread_cond_signal@@GLIBC_2.3.2  
(pthread_cond_signal.S:54)  
==23670==   by 0x4E58881: Scheduler::CalculateFsDrift()  
(scheduler.cc:957)  
==23670==   by 0x4E5A566: Scheduler::CalculateAndSetDrifts()  
(scheduler.cc:928)  
==23670==   by 0x4E5A64D: Scheduler::Sync() (scheduler.cc:373)  
==23670==   by 0x4E5BA37: Scheduler::Process(Scheduler::data_source_t)  
(scheduler.cc:1660)  
==23670==   by 0x4E5BCB9: Scheduler::Start(Scheduler::SchedulerConfig_t)  
(scheduler.cc:1752)  
==23670==   by 0x403FBC: CLIScheduler::Start(void*)  
(cli_scheduler.cc:279)  
==23670==   by 0x52B9181: start_thread (pthread_create.c:312)  
==23670==   by 0x5BD347C: clone (clone.S:111)
```

5. Podsumowanie

Wraz z ze zwiększaniem pokrycia sygnałem DAB+ terenu Polski, zapotrzebowanie na odpowiednie odbiorniki będzie wzrastało. Obecnie jedynym dostawcą tej usługi w kraju jest Polskie Radio. Lepsza jakość dźwięku, dodatkowe funkcje i wydajniejsze wykorzystanie pasma w porównaniu do techniki FM powinny jednak przekonać do tego rozwiązania zarówno odbiorców, jak i właścicieli stacji radiowych. Mamy nadzieję, że nasz projekt przyczyni się do przyspieszenia procesu przejścia z radiofonii analogowej do cyfrowej. By tak się stało, uważamy, że potrzebny jest powszechny dostęp do odbiorników. Nasze rozwiązanie jest alternatywą dla dedykowanych urządzeń. Zostało oparte o powszechnie dostępne i uniwersalne komponenty w skład których wchodzi:

- komputery jednokładowe, takie jak Raspberry Pi, Odroid, BeagleBoard, które cieszą się coraz większą popularnością; między innymi jako routery bezprzewodowe, serwery HTTP/FTP, elementy automatyzacji domów itd.
- uniwersalny tuner może być stosowany do odbioru radia FM, telewizji DVB-T, czy skanowania częstotliwości

Założone cele zostały zrealizowane, jednak wciąż istnieją elementy, o które można wzbogacić tę aplikację. W szczególności są to:

- umożliwienie dekodowania innych usług, m.in. obrazów i trybu pakietowego
- obsługę innych platform sprzętowych, np. Raspberry Pi, urządzeń z systemem Android
- dalszą poprawę wydajności całej biblioteki
- ułatwienie instalacji, np. przygotowanie gotowych plików binarnych na odpowiednie platformy

Bibliografia

- [1] ETSI EN 300 401: *Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*, 1 2006, Adres: http://www.etsi.org/deliver/etsi_en/300400_300499/300401/01.04.01_40/en_300401v010401o.pdf [Data uzyskania dostępu: 21.12.2015r.]
- [2] ETSI TS 102 563: *Digital Audio Broadcasting (DAB); Transport of Advanced Audio Coding (AAC) audio*, 5 2010, Adres: http://www.etsi.org/deliver/etsi_ts/102500_102599/102563/01.02.01_60/ts_102563v010201p.pdf [Data uzyskania dostępu: 21.12.2015r.]
- [3] W. Hoeg i T. Lauterbach: *Digital Audio Broadcasting: Principles and Applications of DAB, DAB + and DMB, Third Edition*, Wiley, 2009
- [4] Dokumentacja biblioteki sdrdab, Adres: <https://sdr.kt.agh.edu.pl/sdrdab> [Data uzyskania dostępu: 21.12.2015r.]
- [5] Polskie Radio: *Cyfrowe Radio DAB+*, Adres: <http://www.polskieradio.pl/240,Cyfrowe-radio-DAB> [Data uzyskania dostępu: 21.12.2015r.]
- [6] Ettus Research: *Universal Software Radio Peripheral*, Adres: <http://www.ettus.com> [Data uzyskania dostępu: 21.12.2015r.]
- [7] Dokumentacja biblioteki rtl-sdr, Adres: <http://git.osmocom.org/rtl-sdr> [Data uzyskania dostępu: 21.12.2015r.]
- [8] H. Bogucka: *Technologie radia kognitywnego*, Warszawa, PWN, 2013
- [9] Specyfikacja sprzętu Odroid C1+, Adres: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141578608433 [Data uzyskania dostępu: 21.12.2015r.]
- [10] Porównanie wybranych komputerów jednokładowych, Adres: <http://www.androidauthority.com/raspberry-pi-2-vs-odroid-c1-vs-hummingboard-vs-mips-creator-ci20-599418> [Data uzyskania dostępu: 21.12.2015r.]
- [11] Zdjęcie urządzenia Odroid C1+, Adres: <http://dn.odroid.com/homebackup/201412051744489785.jpg> [Data uzyskania dostępu: 21.12.2015r.]
- [12] Zdjęcie ekranu AT070TN90, Adres: http://g01.a.alicdn.com/kf/HTB1naSOJFXXXatXXXXq6xXFXXXK/Tontec-7-Raspberry-Pi-LCD-Touch-Screen-Display-TFT-Monitor-AT070TN90-with-Touchscreen-Kit-HDMI-VGA.jpg_640x640.jpg [Data uzyskania dostępu: 21.12.2015r.]
- [13] Dokumentacja narzędzia CMake, Adres: <https://cmake.org/documentation> [Data uzyskania dostępu: 21.12.2015r.]
- [14] Optymalizacje kompilatora gcc, Adres: <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html> [Data uzyskania dostępu: 21.12.2015r.]
- [15] Dokumentacja modułu QtQuick, Adres: <http://doc.qt.io/qt-5/qtquick-index.html> [Data uzyskania dostępu: 21.12.2015r.]

- [16] Przykładowy plik qmake.conf, Adres: <http://code.qt.io/cgit/qt/qtbase.git/tree/mkspecs/devices/linux-rasp-pi2-g++/qmake.conf> [Data uzyskania dostępu: 21.12.2015r.]
- [17] Kod źródłowy Qt, Adres: <http://doc.qt.io/qt-5/build-sources.html> [Data uzyskania dostępu: 21.12.2015r.]
- [18] Dokumentacja systemu qmake, Adres: <http://doc.qt.io/qt-5/qmake-manual.html> [Data uzyskania dostępu: 21.12.2015r.]
- [19] K. Yaghmour: *Building Embedded Linux Systems*, O'Reilly Media, 2008
- [20] Pliki źródłowe jądra Odroid C1+, Adres: <https://github.com/hardkernel/linux/tree/odroidc-3.10.y> [Data uzyskania dostępu: 21.12.2015r.]
- [21] Wybrane strony man podręcznika systemu Linux
- [22] Dokumentacja projektu Rygel, Adres: <https://wiki.gnome.org/Projects/Rygel> [Data uzyskania dostępu: 21.12.2015r.]
- [23] Dokumentacja narzędzia Valgrind, Adres: <http://valgrind.org/docs/manual/index.html> [Data uzyskania dostępu: 21.12.2015r.]
- [24] Plik wynikowy narzędzia Valgrind, Adres: <https://drive.google.com/file/d/0B26rJlyOcot6eEM2bTcxbFd1VXc/view?usp=sharin> [Data uzyskania dostępu: 21.12.2015r.]
- [25] J. Bułat, T. Zieliński i inni: *Zrób to sam: komputerowy odbiornik RTL-SDR radia cyfrowego DAB+*, 2015
- [26] ARM: *ARMv7-AR Reference Manual*, Adres: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.architecture.reference/index.html> [Data uzyskania dostępu: 21.12.2015r.]
- [27] Dokumentacja biblioteki fftw, Adres: <http://www.fftw.org> [Data uzyskania dostępu: 21.12.2015r.]
- [28] Dokumentacja biblioteki Qt, Adres: <http://doc.qt.io/qt-5> [Data uzyskania dostępu: 21.12.2015r.]
- [29] Dokumentacja systemu Property, Adres: <http://doc.qt.io/qt-5/properties.html> [Data uzyskania dostępu: 21.12.2015r.]
- [30] Dokumentacja klasy QMetaType, Adres: <http://doc.qt.io/qt-5/qmetatype.html> [Data uzyskania dostępu: 21.12.2015r.]
- [31] E. Gamma i inni: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1998
- [32] Dokumentacja klasy QQmlApplicationEngine, Adres: <http://doc.qt.io/qt-5/qqmlapplicationengine.html> [Data uzyskania dostępu: 21.12.2015r.]
- [33] Konwersje typów pomiędzy QML a C++, Adres: <http://doc.qt.io/qt-5/qtqml-cppintegration-data.html> [Data uzyskania dostępu: 21.12.2015r.]
- [34] Dokumentacja biblioteki ReactJS, Adres: <https://facebook.github.io/react> [Data uzyskania dostępu: 21.12.2015r.]
- [35] Narzędzie Adobe® PhoneGap™ Build, Adres: <https://build.phonegap.com> [Data uzyskania dostępu: 21.12.2015r.]