BIBLIOTEKA
POLSKIEGO KRÓTKOFALOWCA 31

Krzysztof Dąbrowski
OE1KDA

Radiostacje i odbiorniki z cyfrową obróbką sygnałów

Tom 3

Wiedeń 2016

© Krzysztof Dąbrowski OE1KDA
Wiedeń 2016

Opracowanie niniejsze może być rozpowszechniane i kopiowane na zasadach niekomercyjnych w dowolnej postaci (elektronicznej, drukowanej itp.) i na dowolnych nośnikach lub w sieciach komputerowych pod warunkiem nie dokonywania w nim żadnych zmian i nie usuwania nazwiska autora. Na tych samych warunkach dozwolone jest tłumaczenie na języki obce i rozpowszechnianie tych tłumaczeń.
Na rozpowszechnianie na innych zasadach konieczne jest uzyskanie pisemnej zgody autora.

Radiostacje i odbiorniki
z cyfrową obróbką sygnałów
(SDR)

Tom 3

Krzysztof Dąbrowski OE1KDA

Wydanie 1

Wiedeń, maj 2016

Spis treści

Sommaire
Émetteurs et récepteurs définie par logiciel (RDL)
Volume 3

Wstęp

W poprzednich tomach zostały przedstawione różne modele odbiorników i radiostacji opartych na cyfrowej obróbce sygnałów, czyli programowalnych. Technika ta rozwija się obecnie dynamicznie i ciągle pojawiają sie niewe modele i opracowania zarówno fabryczne jak i amatorskie. Część opracowań klasy popularnej opiera się o paluszkowe odbiorniki DVB-T z procesorem RLT2832U lub podobnym firmy Realtek i głowicą w.cz. R820T lub R820T2. Standardowo dolna granica pokrywanego zakresu jest równa 24 MHz, a więc nie pokrywają one zakresów fal długich, średnich i krótkich. Po dodaniu nawet prostego konwertera częstotliwości zakresy te stają się dostępne i stąd część modeli pracuje na tej właśnie zasadzie (np. „DX Patrol"). W starszych modelach występowała głowica E4000 o dolnej granicy odbioru 64 MHz ale już od dłuższego czasu zaprzestano jej produkcji. Obie głowice pokrywają wprawdzie zakres odpowiednio do 1,7 lub 1,85 GHz ale każda z nich posiada w pewnym miejscu lukę na skali dostępnych częstotliwości.
Innym często stosowanym odbiornikiem jest Fun Cube Dongle Pro Plus (FCD2). Niezależnie jednak od wykonania sprzęt programowalny wynaga użycia odpowiedniego oprogramowania. Do modeli fabrycznych dodawane są często programy opracowane przez producenta, a niektóre z nich j.np. FDM-Duo albo IC-7300 mogą pracować autonomicznie bez połączenia z PC ale dla rozwiązań amatorskich albo półamatorskich stosowane są uniwersalne programy odbiorcze, j.np. SDR#, HDSDR, SDR-Radio, Linrad, CubicSDR, GQRX i podobne. Linrad, CubicSDR i GQRX są dostępne również w wersjach dla Linuksa i MacOS, a „SDR Touch" pracuje pod Androidem 4.0 lub nowszym.
W zależności od wersji Windows i od programu konieczne może być dodatkowe zainstalowanie bibliotek „NET Framework" 3.5 lub 4.6 dostępnych w witrynie Microsoftu.
Poszczególne typy odbiorników mogą wymagać (również często zależnie od programu odbiorczego) zainstalowania dodatkowych sterowników, j.np. Zadig dla odbiorników z RTL2832, albo dodatkowych bibliotek sterujących np. *ExtIO_xxxx.dll*. Szczegóły podano w instrukcjach wybranych programów.

Rys. 1. Odbiornik Fun Cube Dongle Pro+

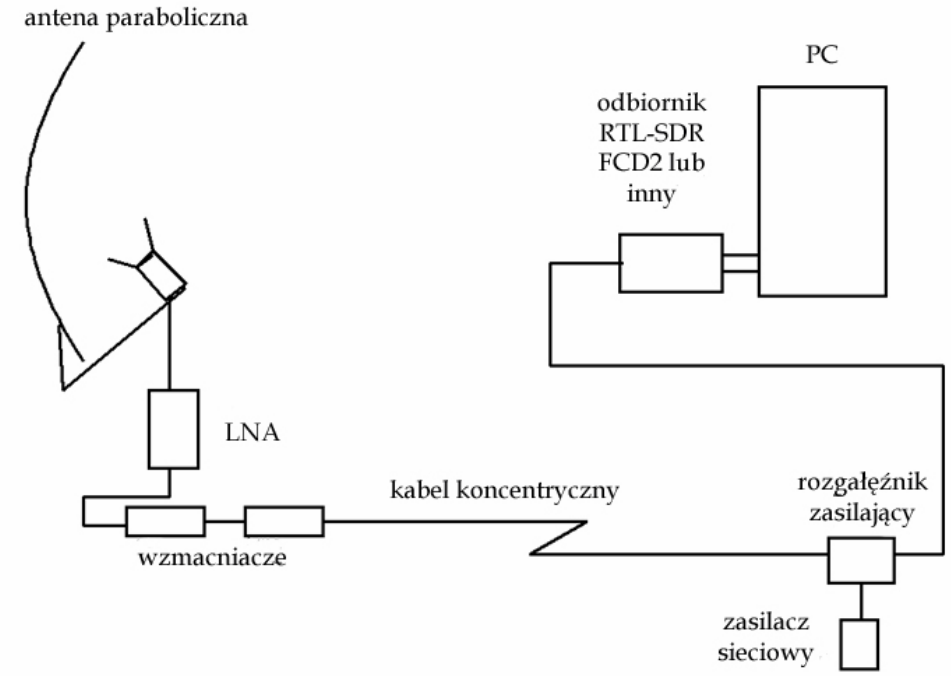
Tabela 1. Sprzęt nadawczo-odbiorczy i pasujące do niego najważniejsze programy SDR

Sprzęt	Programy	Uwagi
FDM-DUO-R	SDR#, HDSDR, SDR-Radio	RX
FDM-DUO	FDM-SW2	TRX
FDM-S1	FDM-SW1	RX
FDM-S2	FDM-SW2	RX
Cloud IQ SDR (RFSpace)	SpectraVue, SDR#, GNU-Radio, SDR-Radio	RX
Fun Cube Dongle Pro+	SDR#, HDSDR	RX
RTL2832U DVB-T	SDR#, HDSDR	RX, sterownik Zadig
DX Patrol	SDR#, HDSDR, SDR Touch dla Androida	RX, sterownik Zadig
Colibri SDR	ExpertSDR2	RX
Perseus	HFSpan	RX,

SunSDR2-PRO	ExpertSDR2	TRX
MB-1 SDR	ExpertSDR2	TRX
ANANxxx („Apache”)	Power SDR OpenHPSDR, GNU Radio Open HPSDR, cuSDR, GHPSDR3, G0ORX/N6LYT dla Androida	rodzina TRX, 10 – 200 W

Rys. 2. Oparty na paluszku z RTL2832U i R820T szerokozakresowy odbiornik „DX Patrol”

Główną dziedziną zastosowań odbiorników programowalnych jest wprawdzie odbiór emisji stacji naziemnych ale odbiorniki FCD i FCD Pro+ zostały opracowane dla odbioru satelitów amatorskich. Niezależnie od rodzaju odbiornika można je także wykorzystać w amatorskiej radioastronomii (rys. 3) a w przyszłości także do odbioru stacjonarnego satelity amatorskiego, który ma pracować w paśmie 10 GHz.



Rys. 3. Schemat blokowy radioteleskopu amatorskiego z odbiornikiem programowalnym

Krzysztof Dąbrowski OE1KDA
Wiedeń
20 maja 2016

SDR#

Instrukcja obsługi programu

Spis treści

Wstęp

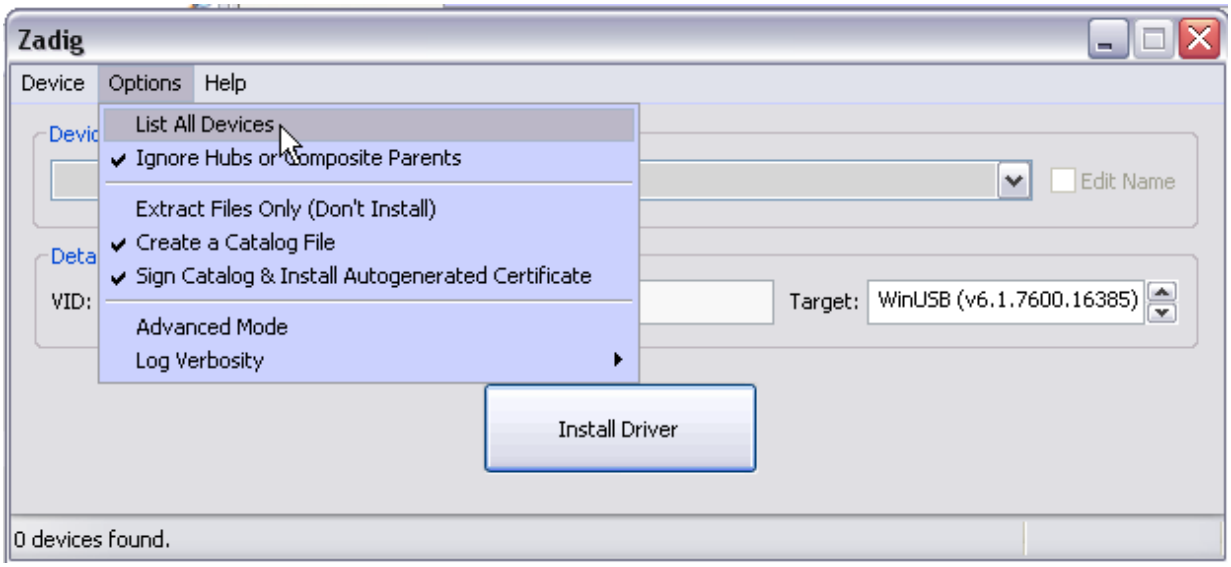
SDR# jest programem odbiorczym współpracującym z wieloma modelami odbiorników programowalnych (ang. SDR), w tym z popularnymi odbiornikami DVB-T z RTL2832U przy użyciu sterownika zadig, z Fun Cube Dongle Pro Plus (FCD2), FiFiSDR, AIRSpy itp. W ostatnim czasie pojawiło się też wiele modeli odbiorników opartych o paluszki DVB-T (DX Patrol Mk3 itp.), które dzięki dodatkowemu konwerterowi częstotliwości pokrywają również pasma fal długich, średnich i krótkich. W wersji dla systemu Windows program wymaga zainstalowania biblioteki „NET Framework 4.6" lub innej jej wersji zależnej od systemu operacyjnego. Pod Windows 10 jest ona już standardowo zainstalowana. Aktualne wersje SDR# pracują pod Windows od 7 wzwyż, a wersje starsze pracujące także pod Windows XP nie są już oficjalnie dostępne, ale można je z pewnością jeszcze gdzieś znaleźć. Wymagają one instalacji biblioteki „NET Framework 3.5".

Instalacja

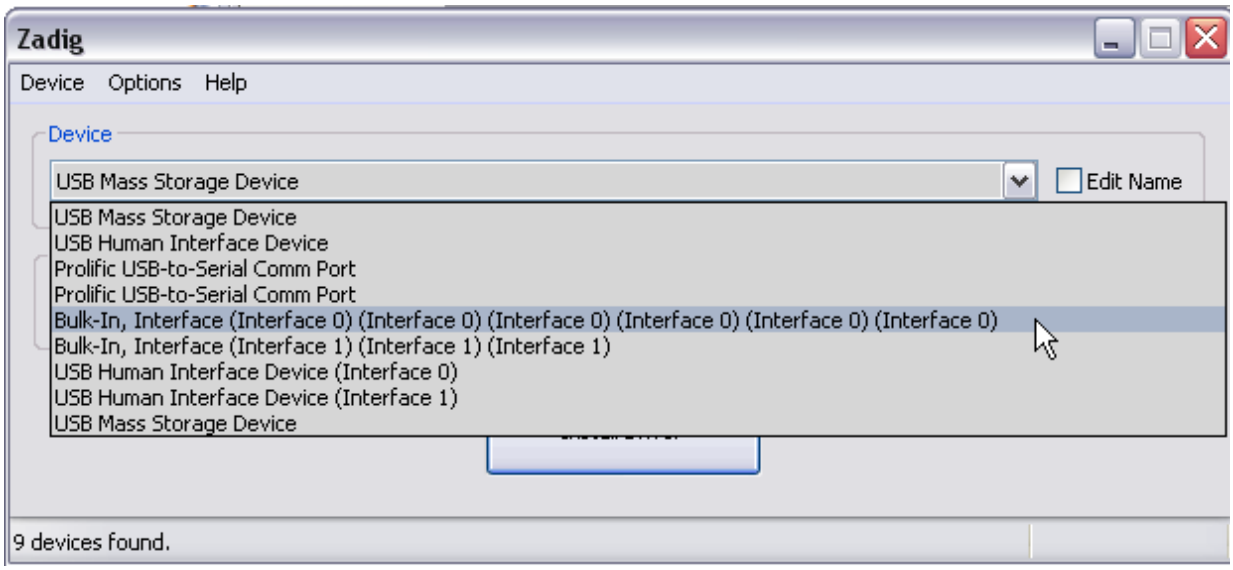
Archiwum programu jest dostępne w Internecie pod adresem: *http://airspy.com/download* i nosi nazwę *sdr-install.zip* lub podobną. Po jego rozpakowaniu do dowolnego katalogu np. C:\SDRSharp należy wywołać program instalacyjny.
W obecnej wersji archiwum zawiera trzy pliki: *httpget.exe*, *install.bat* lub *install-rtlsdr.bat* i *unzip.exe*. Do zapoczątkowania instalacji służy plik wsadowy *install.bat*. W trakcie instalacji nie są dokonywane żadne wpisy w rejestrze Windows i nie są też instalowane żadne sterowniki ani w katalogu Windows\system32, ani w innych katalogach systemowych. W czasie instalacji program httpget pobiera z internetu aktualną wersję programu i sterownika zadig, który musi być zainstalowany oddzielnie.
Po zakończeniu instalacji katalog docelowy zawiera m.in. kilkanaście plików bibliotek .dll i plik wywoławczy programu *SDRSharp.exe*. Wśród bibliotek *.dll* znajdują się m.in. biblioteki przeznaczone do współpracy z poszczególnymi rodzajami odbiorników: *rtlsdr.dll, sdriq.sdr, SDRSharp.AIRSPY.dll, SDRSharp.FUNcube.dll, SDRSharp.FUNcubeProPlus.dll, SDRSharp.HackRF.dll, SDRSharp.RTLSDR.dll, SDRSharp.RTLTCP.dll, SDRSharp.SDRIP.dll, SDRSharp.SDRIQ.dll, SDRSharp.SoftRock.dll* itd.
W żadnym wypadku nie należy instalować oprogramowania dołączonego do odbiornika DVB-T i służącego do odbioru telewizji za jego pomocą.

Instalacja sterownika Zadig

Sterownik Zadig umożliwia wczytanie nieprzetworzonych danych z odbiornika w celu ich wykorzystania przez programy odbiorcze. Najnowsze wersje sterownika pracuja prawidłowo w środowiskach Windoes Vista, 7, 8 i 10. Dla Windows XP konieczna jest jego specjalna wersja – *zadig_xp*.
W celu zainstalowania sterownika Zadig należy wywołać lik *zadig.exe* znajdujący się w katalogu zawierającym SDRSharp. Przed zainstalowaniem sterownika należy włożyć odbiornik DVB-T do dowolnego gniazda USB i nie zgodzić się na instalację sterownika proponowanego standardowo przez Windows. Przełożenie odbiornika do innego gniazda powoduje konieczność ponownego zainstalowania sterownika Zadig. Następnie należy w menu „Options” („Opcje”) zaznaczyć punkt „List All Devices” („Uwzględnij wszystkie urządzenia”).

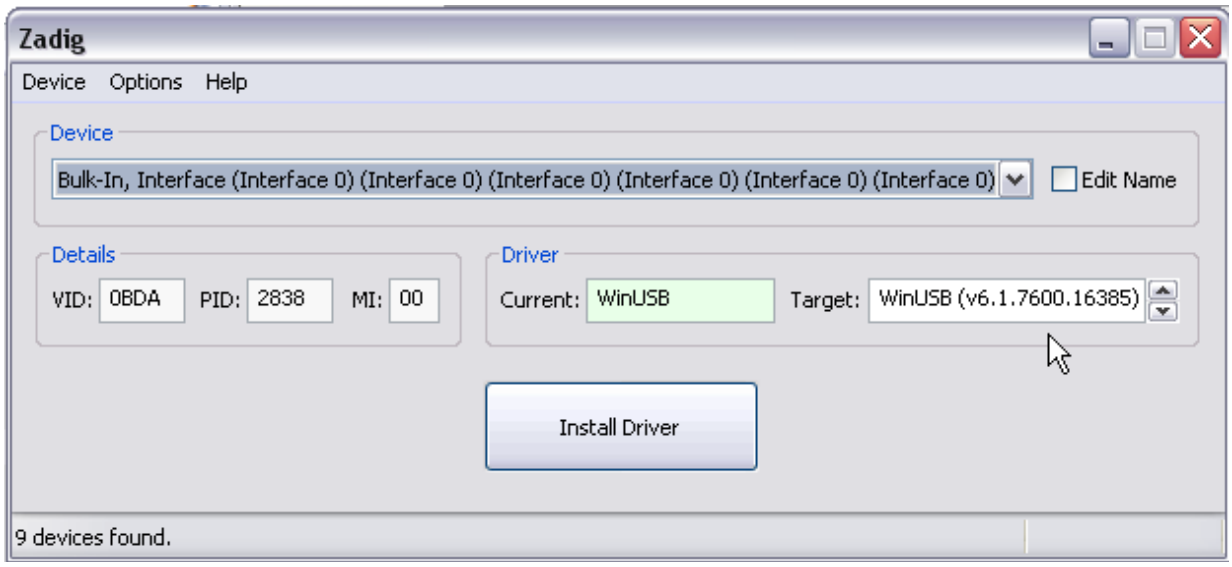


Rys. 1.2.1



Rys. 1.2.2

Z rozwijanej listy w ramce „Device” („Urządzenie”) wybierane jest urządzenie, dla którego ma być instalowany sterownik. Dla odbiornika DVB-T – „Bulk-In Interface (Interface 0)” – wybierany jest automatycznie sterownik WINUSB – numer wersji może oczywiście różnić się od podanego na ilustracji.
W niektórych przypadkach zamiast nazwy „Bulk-In Interface (Interface 0)” może w spisie występować nazwa RTL2832UHIDIR lub RTL2832U. Nie należy wybierać natomiast pozycji „USB Receiver (interface 0) jeżeli pojawi się w spisie.
Jeżeli wybór został dokonany prawidłowo można rozpocząć instalację przez naciśnięcie przycisku „Install Driver” („Instaluj sterownik”). W niektórych sytuacjach przycisk może być podpisany jako „Replace Driver” („Zastąp sterownik”).



Rys. 1.2.3

Sterownik Zadig jako obsługujący sprzęt jest niezbędny nie tylko dla programu SDRSharp ale i dla pozostałych takich jak HDSDR, SDR-RadioCubicSDR itp.

Konfiguracja

Po zakończeniu instalacji można wywołać „SDRSharp” i rozpocząć jego konfigurację.

Wybór i konfiguracja odbiornika

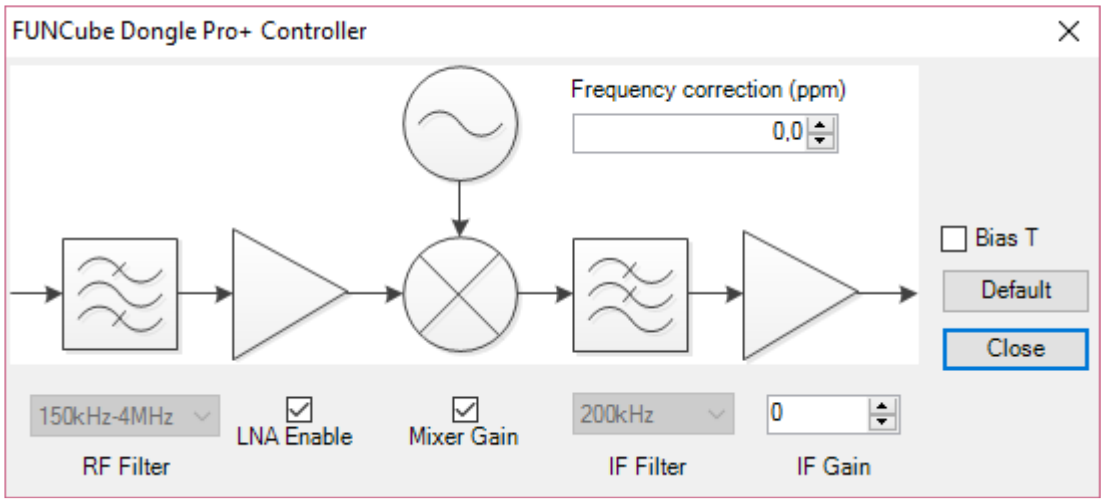
W lewej górnej części okna głównego znajduje się rozwijana lista obsługiwanych odbiorników Są wśród nich m.in. odbiorniki kompatybilne z modelem „SoftRock” z syntezerem Si570, FiFiSDR, AIRSpy, dwa modele Fun Cube Dongle (Pro i Pro+), telewizyjne odbiorniki z mikrokontrolerem RTL2832U podłączone bezpośrednio lub przez lokalną sieć (TCP), odbiorniki dostarczające kwadraturowych strumieni IQ i inne. Ich spis jest zależny od wersji programu. Niektóre z pozycji j.np. „SoftRock (Si570)” dotyczą całych grup kompatybilnych rozwiązań odbiorników. W dalszym ciągu rozdziału przedstawiono kilka przykadów konfiguracji odbiorników wraz z omówieniem ich najważniejszych parametrów. Ze względu na różnorodność modeli trudno jest omówić wszelkie kombinacje i parametry ale autor ma nadzieję, że czytelnicy na tej podstawie potrafią zidentyfikować parametry i ich znaczenie dla innych rozwiązań i rozpocząć eksperymenty od w przybliżeniu dobrych ustawień.

FCD2

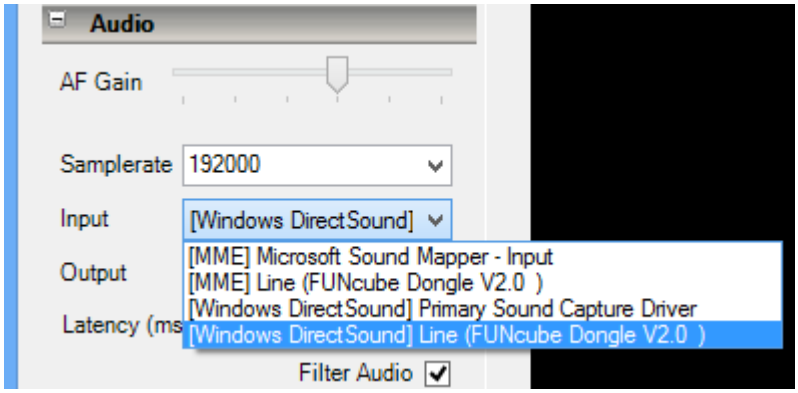
Rys. 1.3.1

W menu obsługiwanych urządzeń wybierany jest odbiornik „Fun Cube Dongle Pro+” lub jego starsza wersja „Fun Cube Dongle Pro” w zależności od posiadanego wyposażenia. Po naciśnięciu przycisku konfiguracji sprzętu oznaczonego symbolem trybika otwierane jest okno zawierające przybliżony schemat blokowy odbiornika. Pierwszy stopień układu – filtr pasmowy – jest przełączany automatycznie w zależności od częstotliwości odbioru. Zależnie od warunków lokalnych i używanej anteny można natomiast włączyć przedwzmacniacz LNA i wzmocnienie mieszacza. Możliwości te trzeba jednak wykorzystywać ostrożnie aby nie doprowadzić do przesterowania dalszych stopni odbiornika i do powstania w ywniku tego składowych intermodulacyjnych. Szerokość pasma filtru pośredniej częstotliwości („IF Filter”) jest przełączana automatycznie w zależności od wybranego w SDR# rodzaju emisji – dokładnie od jej szerokości pasma ustawianej w oknie głównym SDR#. Główną część wzmocnienia zapewnia wzmacniacz p.cz. i dobierając je warto trochę poeksperymentować dla uzyskania optymalnych wyników. Podobnie warto również poeksperymentować ze wzmocnieniem poprzednich stopni.

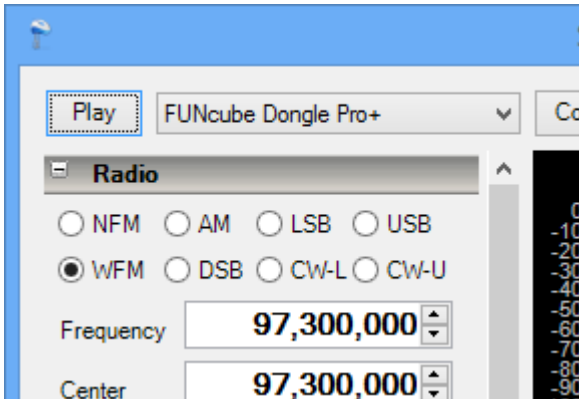
Układ poprzedniego modelu był w oknie konfiguracyjnym przedstawiony znacznie dokładniej z rozróżnieniem m.in. poszczególnych stopni p.cz. ale być może utrudniało to orientację w ustawieniach mniej doświadczonym użytkownikom. Szerokość wyświetlanego pasma ograniczona maksymalną częstotliwością próbkowania wynosi 192 kHz, a oficjalny zakres odbioru rozciąga się od 150 kHz do 1,9 GHz z przerwą pomiędzy 240 i 420 MHz. W rzeczywistości zakres ten jest nieco szerszy a przerwa – węższa.



Rys. 1.3.2. Konfiguracja odbiornika FCD2



Rys. 1.3.3 Wybór źródła dźwięku

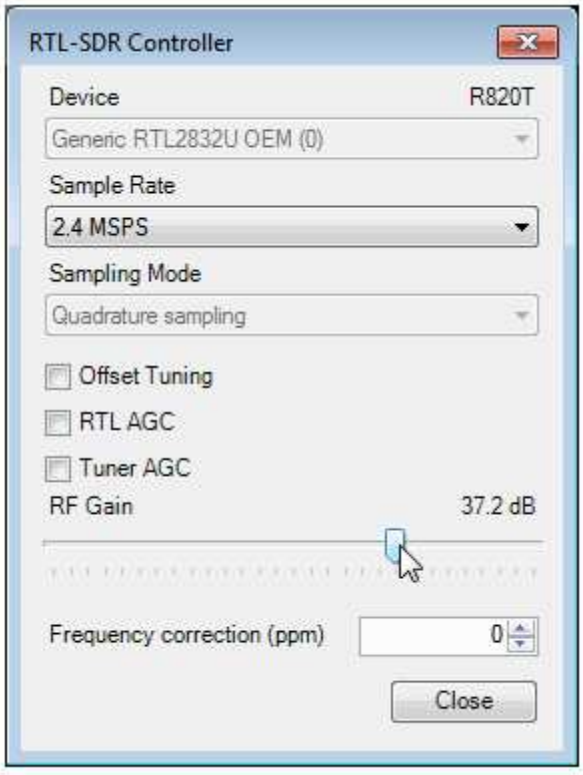


Rys. 1.3.4. Wybór emisji i częstotliwości odbioru. Wygląd sekcji zależy od wersji programu

Po dokonaniu podstawowej konfiguracji i dostrojenia można włączyć odbiór za pomocą przycisku oznacznonego trókątem jak w magnetofonach lub podpisanego „Play” w zależnosci od wersji programu.

Odbiornik DVB-T z procesorem RTL2832

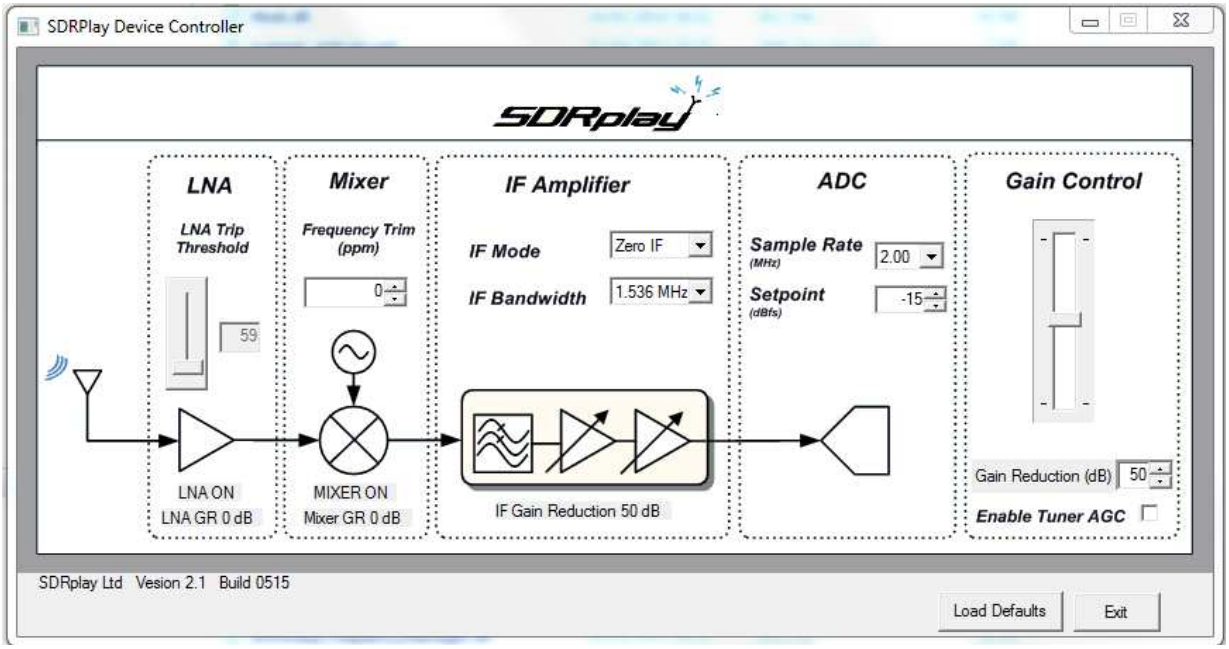
W menu wyboru odbiornika należy wybrać pozycję „RTL-SDR / USB” dla odbiornika podłączonego bezpośrednio do złącza USB komputera albo pozycję „RTL-SDR / TCP” dla odbiornika dostępnego przez sieć lokalną. Odbiornik może być podłączony do innego dowolnego dostępnego w sieci komputera PC albo mikrokomputera w rodzaju „Maliny” (patrz tom 24).
Przycisk konfiguracji (z symbolem trybika) otwiera okno zawierające między innymi poziomy suwak przeznaczony do regulacji całkowitego wzmocnienia toru odbiorczego. Pasującą do danej sytuacji i używanej anteny wartość trzeba dobrać eksperymentalnie stopniowo zwiększając je tak, aby uzyskać możliwie dobry stosunek sygnału użytecznego do szumów. Oznacza to zwiększanie wzmocnienia do czasu kiedy poziom szumów obserwowanych na ekranie także zacznie wzrastać.
Ogólnie rzecz biorąc jest ono również w mniejszym lub większym stopniu zależne od zakresu częstotliwości. W odbiornikach z konwerterem pokrywającym zakres fal krótkich jest ona oczywiście różna dla zakresów wyższych nie wymagających przemiany i dla zakresów niższych. Domyślnie wzmocnienie jest ustawione na zero. Standardowo odbiorniki DVB-T pokrywają zakres 24 – 1766 MHz. Pokrycie zakresów poniżej 24 MHz wymaga więc zastosowania konwertera częstotliwości.
Ze względu na to, że są one wpierwszym rzędzie przeznaczone do odbioru szerokopasmowej cyfrowej telewizji dobrze jest je wygrzać przed użyciem przez co najmniej 20 minut aby poprawić stabilność oscylatora na czas pracy.



Rys. 1.3.5. Parametry odbiornika RTL-SDR

W oknie tym możliwy jest również wybór rodzaju ARW („RTL AGC”, „Tuner AGC” – dla R820T), wybór częstotliwości próbkowania w polu „Sample Rate” i wprowadzenie poprawki częstotliwości w polu „Frequency correction”.

Odbiornik SDRPlay



Rys. .3.6. Okno konfiguracyjne odbiornika SDRPlay

Podobnie jak dla FCD2 okno zawiera uproszczony schemat blokowy odbiornika i daje możliwość wprowadzenia poprawki częstotliwości dla stopnia przemiany, szerokości pasma p.cz., parametrów ARW i całkowitego wzmocnienia w torze odbiorczym.
Dla wzmacniacza wstępnego ustawiany jest tutaj próg, przy którym ARW dokonuje jego wyłączenia. Przy wyższszym progu wzmacniacz pozostaje wprawdzie dłużej włączony przy wzroście poziomu sygnału ale może to nieść za sobą niebezpieczeństwo przesterowania odbiornika i wystąpienia modulacji skrośnej. W odbiorniku FCD wzmacniacz LNA pracuje przez cały czas ze stałym wybranym wzmocnieniem lub też jest stale wyłączony.
Poprawka częstotliwości heterodyny w stopniu przemiany pozwala na skorygowanie skali w stosunku do prawdziwych częstotliwości odbieranych stacji.
W sekcji wzmacniacza p.cz. możliwy jest wybór zerowej częstotliwości pośredniej lub niskiej odbiegającej od zera. Do wyboru są częstotliwości 450 kHz, 1,62 MHz i 2,048 MHz. Spis możliwych szerokości pasma poniżej jest dostosowywany automatycznie do wybranego filtru p.cz. Pełna gama możliwości jest dostępna tylko dla zerowej p.cz.
W sekcji ARW możliwy jest wybór częstotliwości próbkowania i progu reakcji ARW. Wyższy próg oznacza pełniejsze wysterowanie przetwornika analogowo-cyfrowego.
Przy górnym położeniu suwaka regulacji wzmocnienia toru odbiorczego jest ono maksymalne, a przy dolnym minimalne. Pole „Gain Reduction” ustala różnicę poziomu wzmocnienia w stosunku do maksimum.
Po włączeniu ARW dla głowicy („Enable tuner AGC”) wyłączona jest jej ręczna regulacja wzmocnienia.
Przycisk „Load Defaluts” powoduje ustawienie domyślnych wartości parametrów, a przycisk „Exit” – zamknięcie okna.

Meldunki błędów

W trakcie wywoływania, konfiguracji lub pracy programu mogą wyświetlać się meldunki informujące o wystąpieniu różnych nieprawidłowych sytuacji lub trudności.
Niektóre z tych przypadków wymieniamy poniżej.

- Meldunek „No Device selected” w trakcie uruchamiania programu mimo podłączonego odbiornika – trzeba upewnić się, że instalacja przebiegła prawidłowo do końca i że w katalogu programu znajduje się biblioteka lub biblioteki dla danego typu odbiornika. Jednym z powodów nieprawidlowego przebiegu instalacji może być brak odpowiednich uprawnień do wybranego katalogu. Windows od 7 wzwyż wymaga uprawnień administratora dla instalacji programów w katalogu „Programm Files” lub w jednym z podporządkowanych, a standardowo przydziela jedynie prawa odczytu. Lepiej jest instalować programy w założonych dla nich katalogach leżących bezpośrednio na C:\, np. C:\SDR#.
- Meldunek „No compatible device found” w trakcie uruchamiania programu mimo podłączonego odbiornika. Może on być spowodowany przez podłączenie odbiornika za pomocą długiego kabla USB niskiej jakości. Inną przyczyną może być niekompatybilność niektórych złączy USB 3.0 z odbiornikiem – sprawdzić to można podłączając odbiornik do innego złącza. Dalszą przyczyną może być nieprawidłowo pracujący odbiornik. W tym przypadku można sprawdzić go podłączając do innego kmoputera.
- Meldunek „Compatible devices have been found but are all busy” – konieczna może być powtórna instalacja sterownika Zadig i ewentualnie wypróbowanie innych złączy USB komputera. Można także na próbę odłączyć wszystkie inne urządzenia zewnętrzne podłączone do złączy USB, albo przejście z częściej sprawiającego kłopoty złącza USB 3.0 na złącze USB 2.0. W tym i w poprzednim przpadku pomóc może także instalacja sterownika w trybie chronionym. Warto także upewnić się, że w trakcie instalacji sterownika Zadig nie została wybrana pozycja „USB Receiver (Interface 0)”. Jeśli tak należy to skorygować wybierając właściwą: „Bulk in interface ....” albo RTL2832... Jeżeli nie występują w spisie trzeba sprawdzić czy zaznaczona została pozycja „List all devices” („Uwzględnij wszystkie urządzenia”) w menu „Options” („Opcje”). Pole „Ignore Hubs or Composite Parents” („Ignoruj rozgałęźniki i urządzenia złożone”) powinno pozostać nie zaznaczone. Czasem jednak wystarczy tylko wyjąć odbiornik z gniazda i włączyć go z powrotem.
- Meldunek “Unable to load DLL ‘rtlsdr': the specified module could not be found. (Exception from HRESULT: 0x8007007E)” może być spowodowany brakiem biblioteki “Visual C++ Runtime” lub jej uszkodzoną instalacją. Pomocą może być ponowna instalacja biblioteki. Jest ona dostępna w witrynie „Microsoftu” pod adresem *https://www.microsoft.com/en-us/download/details.aspx?id=8328*.
- W środku widma wyświetlanego w polu wodospadu widoczny jest stały sygnał, który nie daje się usunąć. Jest to efekt spowodowany właściwościami wielu konstrukcji odbiorników Można usunąć go matematycznie zaznaczając punkt “Correct IQ” (“Skoryguj symetrię kanałów IQ”).
- Instalacja sterownika Zadig trwa bardzo długo i kończy się niepowodzeniem. Najczęściej jest to spowodowane próbą instalacji bez posiadania uprawnień administratora. Należy nacisnąć plik instalacyjny sterownika prawym klawiszem myszy i zaznaczyć pozycję “Wykonuj jako administrator”.
- W spisie urządzeń w oknie instalatora brakuje pozycji “Bulk-in Interface (Interface 0)” – konieczne jest sprawdzenie czy została zaznaczona pozycja pozycja „List all devices” („Uwzględnij wszystkie urządzenia”) w menu „Options” („Opcje”) i zaznaczyć ją w razie potrzeby. Zamiast tego urządzenia w spisie może także występować urządzenie o nazwie zaczynającej się od RTL i wówczas ono jest tym właściwym. Brak jakiejkolwiek z tych pozycji może być spowodowany niedziałaniem odbiornika, który należy wówczas wymienić.
- W menu wyboru urządzenia w oknie głównym SDRSharp brakuje pozycji RTL-SDR/USB – może to wskazywać na omyłkowe pobranie niewłaściwej lub przestarzałej wersji. Rozwiązaniem jest pobranie wersji aktualnej.
- Odbiornik nie funkcjonuje w gnieździe USB 3.0. Sprzętowe sterowniki złączy USB 3.0 bywają bardziej kapryśnie i nie zawsze obsługują wszystkie podłączone urządzenia. Przeważnie współ-

pracują one poprawnie z odbiornikami RTL-SDR ale bywają wyjątki. Pomocne okazuje się przeniesienie odbiornika do złącza USB 2.0.

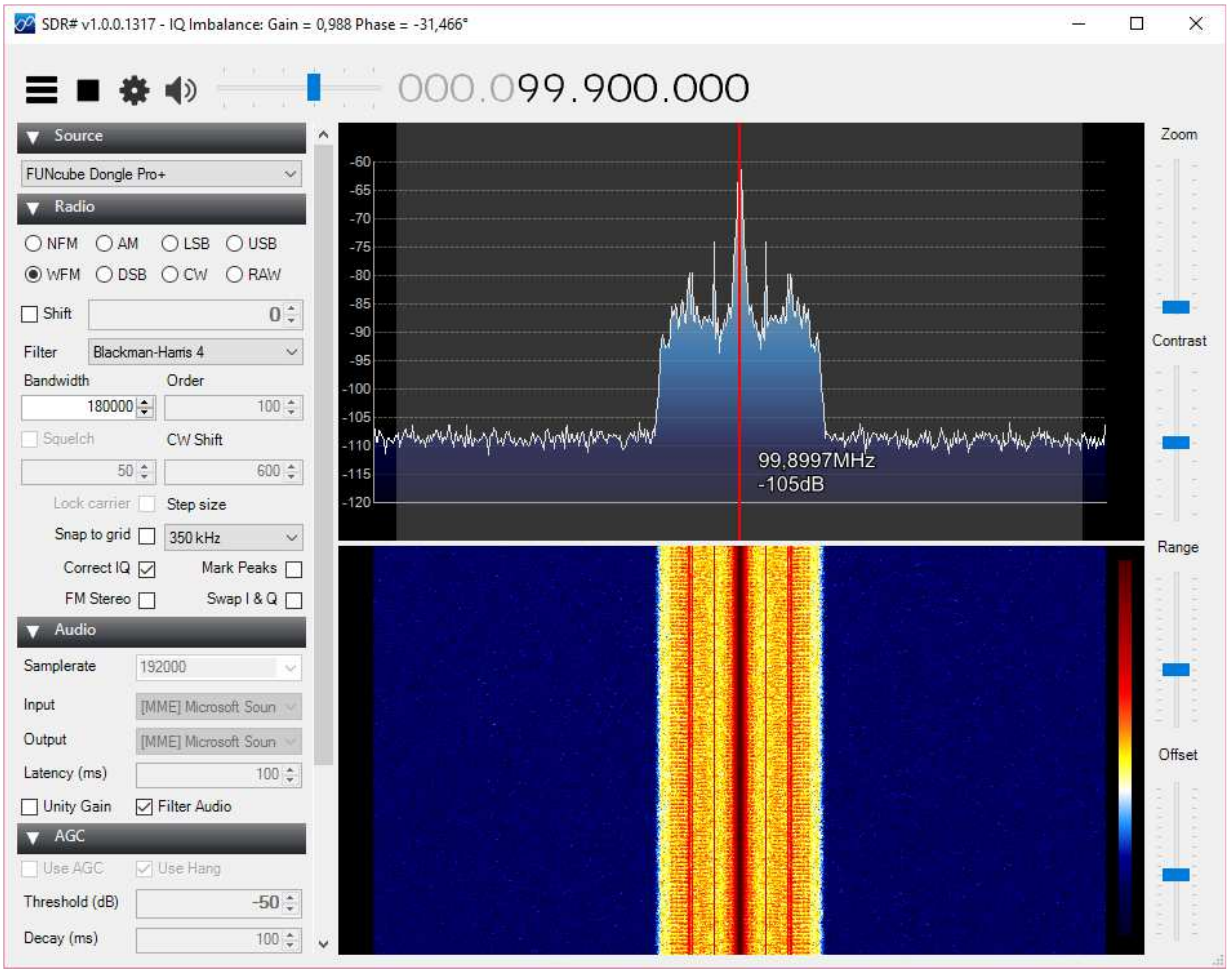
- W trakcie instalacji za pomocą pliku *install-rtlsdr.bat* wyświetlane są meldunki o niemożliwości znalezienia różnych plików lub katalogów. Jest to spowodowane wywołaniem pliku bezpośrednio ze spakowanego archiwum. Przed rozpoczęciem instalacji programu należy rozpakować archiwum ZIP.
- W trakcie instalacji kilkakrotnie miga na ekranie okno wiersza poleceń i na koniec nic nie zostaje zainstalowane. Sytuacja taka może wystąpić w niektórych wersjach Windows albo być spowodowana działaniem niektórych programów antywirusowych i zaradzić temu można przez ręczną instalację sterowników RTL-SDR. Sposób ręcznej instalacji podano w dokumencie *http://www.rtl-sdr.com/manual-installation-of-sdr/*.
- Zadig zgłasza meldunek "*System policy has been modified to reject unsigned drivers*" pod Windows 8. Należy wówczas pobrać najnowszą wersję sterownika (2.1) spod adresu *http://zadig.akeo.ie/*. Pliki instalacyjne nowych wersji programu pobierają automatycznie sterownik w wersji 2.1.
- Zły odbiór, odbiornik wydaje się być mało czuły. Powodem może być zbyt niskie wzmocnienie ustawione w konfiguracji odbiornika. Dla odbiorników RTL-SDR wzmocnienie jest domyslnie ustawione na zero. Przyczyną może być także niedostateczna antena, usterki w doprowadzeniu anteny (kablu, wtykach itp.). Jeżeli wszystko wydaje się być w porządku powodem może być też uszkodzenie odbiornika. Standardowa antenka dodawana do odbiorników DVB-T jest większości przypadków niewystarczająca.
- Meldunek SDR# "Application failed to initialize properly (0xc0000135). Click OK to terminate." Może być spowodowany brakiem biblioteki ".NET Framework" lub jej niewłaściwą wersją.
- Meldunek SDR# „Object reference not not to an instance of an object" oznacza, że albo sterownik systemu dźwiękowego nie jest wogóle zainstalowany albo system dźwiękowy jest wyłączony w systemie Windows. Należy włączyć go we właściwościach urządzeń dźwiękowych.
- Odbiornik ciągle traci połączenie ze złączem USB – w przypadku gdy jest on połaczony za pomocą kabla należy podłączyć go bezpośrednio aby stwierdzić czy kabel nie jest uszkodzony. Utrata połączenia także przy bezpośrednim podłączeniu odbiornika oznacza jego defekt.
- Stały brak połączenia i w odbiornikach wyposażonych w diody świecące brak ich świecenia oznacza defekt odbiornika.
- Obciążenie jednostki centralnej (CPU; procesora) dochodzi do 100%. Sytuacja taka może wystąpić na słabszych komputerach. Największe obciążenie komputera powoduje graficzna powierzchnia obsługi. Zasadniczo wymaga ona użycia w komputerze co najmniej procesora dwurdzeniowego. Na słabszych komputerach pomocne może być ograniczenie częstotliwości próbkowania do 1 MHz lub poniżej, zmniejszenie rozdzielczości wyświetlania widma, wyłączenie korekcji symetrii IQ („Correct IQ") i zmniejszenie rzędu filtrów cyfrowych.
- Instalacja sterownika Zadig spowodowała zakłócenia w pracy klawiatury, myszy lub innych urządzeń podłączonych do gniazd USB – sygnalizuje to wybór niewłaściwego urządzenia w oknie instalacyjnym sterownika. Należy ponownie zainstalować sterownik tego urządzenia z poziomu Windows, a następnie ponownie zainstalować Zadig.
- Odbiornik wyposażony w głowicę w.cz. R820T2 jest wyświetlany jako R820T w wynikach programu *rtl_test*. Elektrycznie oba odbiorniki są identyczne poza niewielkimi różnicami w możliwych szerokościach pasma filtrów p.cz. W części cyfrowej oba odbiorniki nie różnią się między sobą. Najbardziej widoczną różnicą jest oznaczenie na obwodzie scalowym.
- Program antywirusowy traktuje SDR# jako wirus. SDR# jest często aktualizowany automatycznie a więc niektóre programy antywirusowe gorszej jakości mają często okazję do takich fałszywych meldunków. Po pewnym czasie i większej liczbie instalacji program zyska uznanie w oczach producentów programów antywirusowych i sytuacja ulegnie powolnej poprawie.
- Pola wyboru emisji przesuwają się w sposób niekontrolowany na ekranie. Przyczyną może być wyskalowanie monitora, które można w razie potrzeby zmienić na poziomie właściwości monitora w systemie Windows.

- Meldunek “An error occurred loading a configuration file: Access to the path ‘C:\Program Files\SDR\s14i12qq.tmp’ is denied. (C:\Program Files\SDR\SDRSharp.exe.Config) —> System.UnauthorizedAccessException: Access to the path ‘C:\Program Files\SDR\s14i12qq.tmp’ is denied.” W trakcie zamykania programu jest on spowodowany ograniczeniem prawa zapisu w katalogu chronionym przez Windows (C:\Program Files). Rozwiązaniem może być zainstalowanie program w katalogu C:\SDR lub podobnym.
- Pilot zdalnego sterowania dodawany do niektórych modeli odbiorników funkcjonuje tylko z oryginalnym oprogramowaniem czyli w trakcie odbioru telewizji. Nie jest on obsługiwany przy pracy jako odbiornik programowalny (SDR).

Obsługa programu

Okno główne

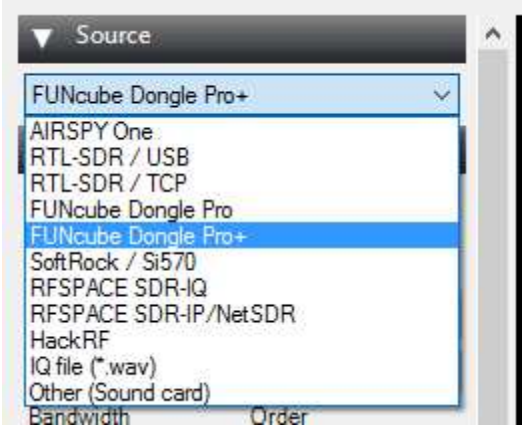
Wymiary okna można zmieniać przez przeciąganie myszą jego krawędzi. U góry okna głównego po lewej stronie znajduje się przycisk wywoływania lub ukrywania menu oznaczony symbolem trzech pasków.
Do rozpoczęcia odbioru po dokonaniu wszystkich niezbędnych ustawień służy, drugi od lewej, przycisk odtwarzania (trójkąt). W trakcie odbioru jego miejsce zajmuje przycisk „Stop” służący do przerwania odbioru (prostokąt). Dostrojenia do pożądanej częstotliwości można dokonywać naciskając górną lub dolną część cyfr, lub wybierając je i obracając kółkiem myszy.
Przycisk oznaczony symbolem trybika otwiera okno konfiguracyjne odbiornika a przycisk oznaczony symbolem głośnika służy do jego wyciszania lub ponownego włączania. Na prawo od niego znajduje się suwak regulacji siły głosu oraz cyfrowa skala częstotliwości. Naciskanie myszą na górną lub dolną połowę cyfr powoduje przestrajanie danej pozycji odpowiednio w górę lub w dół. Do strojenia służy również środkowe kółko myszy.
Najeżdżanie myszą na większość elementów obsługi powoduje wyświetlenie w chmurkach krótkiej informacji o nich.



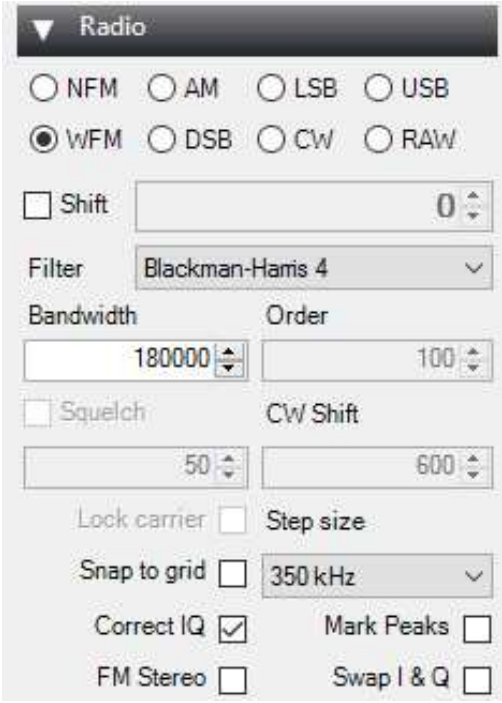
Rys. 1.4.1. Okno główne programu

Przeważającą część okna po prawej stronie zajmują wskaźniki widma (górny) i wodospadowy (dolny). Linię dzielącą je można przeciągać myszą w górę lub w dół zmieniając w ten sposób ich wymiary.
Po prawej stronie wskaźników umieszczone są suwaki służące kolejno (licząc z góry w dół) do regulacji szerokości wyświetlanego widma, kontrastu wskaźników, pionowej skali czułości i poziomu wyj-

ściowego (początku tej skali). W niektórych wersjach programu występuje tutaj także regulator szybkości odświeżania obrazu („Speed”), a w innych znajduje się on w menu bocznym.
Dla zapoznania się z ich funkcjami i znalezienia ustawień najbardziej odpowiadających użytkownikowi w danych warunkach warto po prostu poeksperymentować z nimi.
Po prawej stronie okna głównego znajduje się obszar menu pozwalający na ustawienie wielu istonych w czasie pracy parametrów.
Jest on podzielony na sekcje „Source” („Źródłó” = „Odbiornik”), „Radio” „Audio”, „AGC” (automatyczna regulacja wzmocnienia – ARW), „FFT Display” (analiza danych przeznaczonych do wyświetlania za pomocą Szybkiej Transformaty Fouriera – FFT), „Zoom FFT” (wybór dodatkowych okienek wskaźnika), „Noise Blanker” (tłumienie zakłóceń), „Digital noise reduction” (cyfrowego eliminatora szumów, „Recording” (nagrywanie) i „Frequency Manager” (baza danych częstotliwości). Całkowity zbiór dostępnych tutaj elementów zależy od zainstalowanych dodatków i również od wersji programu. Sekcje programów dodatkowych są zaznaczone gwiazdką w tytule.



Rys. 1.4.2. Wybór odbiornika. Ich spis zależy od wersji programu. Odbiornik RTL-SDR może być podłączony bezpośrednio („RTL-SDR / USB”) lub poprzez sieć lokalną („RTL-SDR / TCP”). Niektóre pozycje dotyczą całych grup kompatybilnych urządzeń



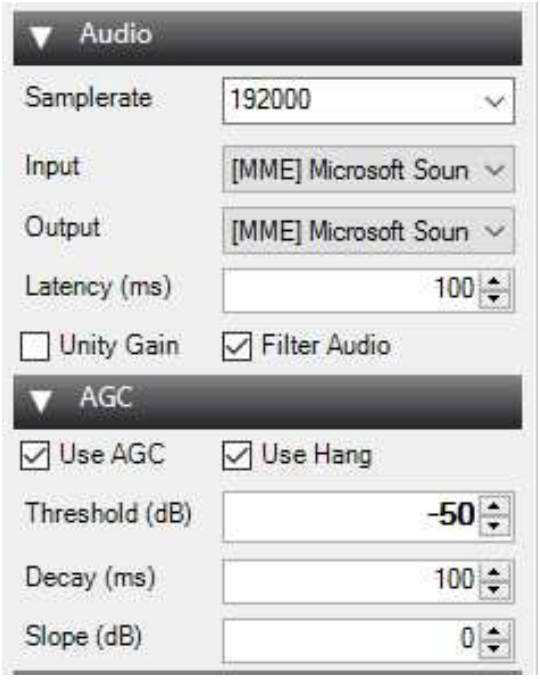
Rys. 1.4.3. Sekcja „Radio”

Sekcja „Radio” zawiera parametry związane z konfiguracją odbiornika.

- Górne pola służą do wyboru rodzaju odbieranej emisji. Do wyboru są: wąskopasmowa modulacja częstotliwości (NFM) stosowana w łącznościach amatorskich i profesjonalnych, szeroko-

pasmowa modulacja częstotliwości stosowana w radiofonii (WFM), modulacja amplitudy (AM), modulacja jednowstęgowa z dolną wstęgą (LSB), modulacja jednowstęgowa z górną wstęgą (USB), dwuwstęgowa modulacja ze stłumioną falą nośną (DSB), telegrafia (CW) i surowe nieprzetworzone dane („RAW").

- Poniżej znajduje sie pole wyboru względnego odstrojenia „Shift", przy czym jego zmiana możliwa jest dopiero po włączeniu (zaznaczeniu) funkcji.
- Pola „Filter" („Rodzaj filtru"), „Filter Bandwidth" („Szerokość pasma filtru") i „Filter Order" („Rząd filtru") definiują parametry filtru cyfrowego używanego przez SDR#. Dla każdego z modeli odbiorników ustawiane są standardowo parametry domyślne ale operator może je dowolnie zmieniać. Wybierana szerokość pasma jest widoczna na wskaźnikach widma. Po ponownym uruchomieniu programu parametry wracają do wartości standardowych.
- Pole „Squelch" („Blokada szumów") powoduje włączenie blokady szumów wyciszającej odbiór do momentu kiedy odbierany sygnał przekroczy ustawiony próg ustawiony w polu poniżej. Blokada szumów jest czynna tylko dla emisji AM i NFM.
- Pole wysokości tonu dudnieniowego dla odbioru telegrafii „CW shift" określa odstęp między częstotliwościami odbioru i nadawania istotny tylko dla urządzeń nadawczych.
- Pola „Snap to grid" („Wyrównaj do siatki") i „Step size" („Krok") ułatwiają dostrajanie się do ustalonych kanałów tak gdzie są one używane. Krok, zależny od pasma i odbieranej służby jest wybierany z rozwijanego spisu. Oparcie się na siatce powoduje, że nawet przybliżone dostrojenie jest korygowane przez program i sprowadzane do najbliższej wartości określonej przez siatkę częstotliwości.
- Pole „Correct IQ" powoduje włączenie algorytmu równoważącego oba kanały i kompensującego dzięki temu składowe niepożądane z drugiej wstęgi. Funkcja ta powinna być zasadniczo zawsze włączona o ile nie ma ważnego powodu do jej wyłączenia.
- Pole zamiany kanałów I i Q „Swap I and Q" powoduje zamianę strumieni wyjściowych kanałów synfazowego (I) i kwadraturowego (Q) bez przełączeń układowych. Może to być potrzebne w niektórych modelach odbiorników.
- Pole „FM stereo" powoduje włączenie odbioru stereofonicznego przy odbiorze radiofonii (WFM).
- Pole „Mark peaks" powoduje włączenie znaczników maksimów sygnału na wskaźniku widma jako ułatwienia przy strojeniu. Wskaźniki te mają kształt kółek.
- Pole „Lock carrier" powoduje włączenie synchronizacji demodulatora z nośną odbieranego sygnału dla emisji AM i dwuwstęgowej z wytłumioną nośną (DSB).

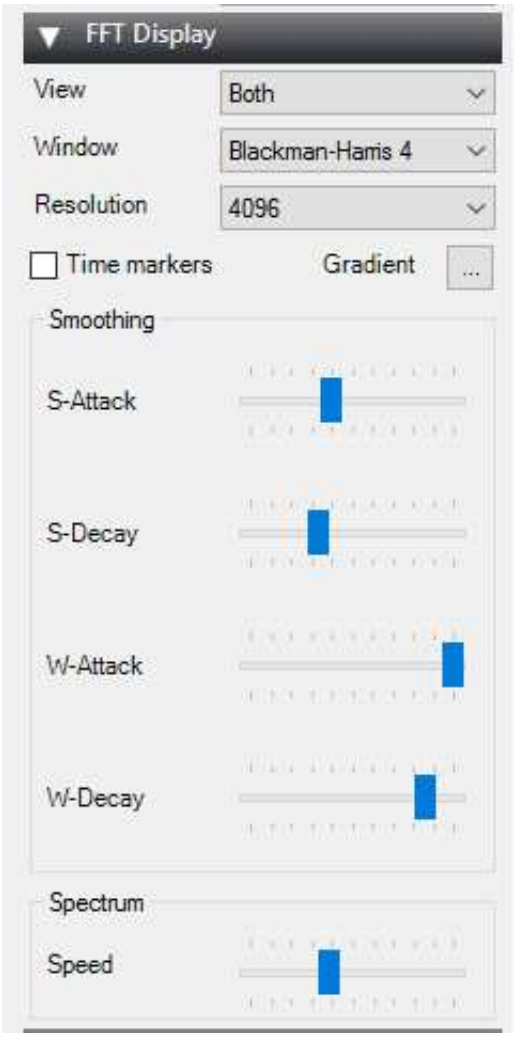


Rys. 1.4.4. Sekcje „Audio" i ARW"

W większości przypadków w sekcji „Audio” można pozostawić wybrane tam domyślnie źródło sygnału i włączoną filtrację (w polu „Filter Audio”). Wyłączenie filtracji może być potrzebne tylko w niektórych szczególnych przypadkach. Kanał wyjściowy dla sygnału m.cz. pozostaje domyślny w trakcie zwykłego odbioru, ale w przypadku dekodowania emisji cyfrowych przez inny program należy zmienić go na „Virtual Audio Cable” lub inny jego odpowiednik – oczywiście po uprzednim zainstalowaniu tego programu. Częstotliwość próbkowania („Samplerate’) jest zależna od wybranego – w polu „Input” – źródła dźwięku. Pole „Wyjście” („Output”) jest istotne w przypadku wyposażenia komputera w większą liczbę podsystemów dźwiękowych. Zaznaczenie pola „Unity Gain”)wzmocnienie równe jedności”) powoduje obniżenie wzmocnienia toru do jedności.

Sekcja „AGC” („ARW”) służy do sterowania pracą automatycznej regulacji wzmocnienia i do jej włączenia. W większości sytuacji można pozostawić ustawienia domyślne. Należą do nich próg („Threshold”), szybkość opadania (”Decay”) i nachylenie zbocza („Slope”).

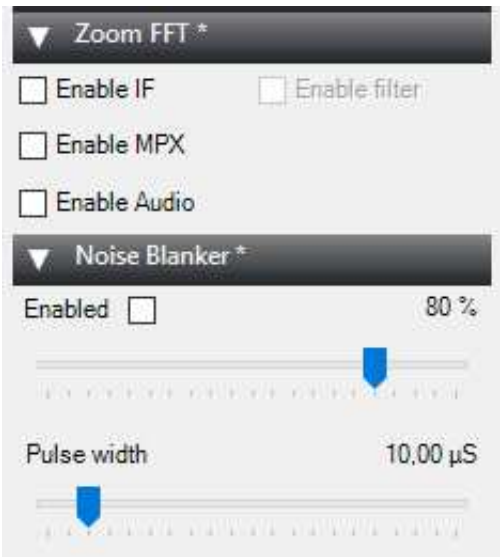
Sekcja „FFT Display” („Analiza FFT dla wyświetlacza widma”) może zachować domyślne ustawienia parametrów. Dla użytkownika istotne lub najważniejsze może być dostosowanie rozdzielczości do własnych potrzeb.



Rys. 1.4.5. Sekcja parametrów analizy za pomocą Szybkiej Transformaty Fouriera

W polu „Sposób wyświetlania” („View”) wybiera się wyświetlanie widma, wskaźnik wodospadowy lub oba albo też ich całkowite wyłączenie.
Pole rodzaju filtracji analizy wpływające na pracę algorytmu FFT („Window”) zawiera domyślnie ustawienie okna rodzaju „Blackman-Harrisa”.

Rozdzielczość pozioma w polu „Resolution” może leżeć w granicach 512 – 4194304 punktów. Wartością domyślną jest 4096 punktów. Zwiększenie ich liczby oznacza dokładniejsze i precyzyjniejsze wyświetlanie danych ale jednocześnie powoduje znaczny wzrost obciążenia komputera. Zalecane jest ograniczenie rozdzielczości do rzeczywiście potrzebnej. Dobrą wartością kompromisową są 16384 punkty.
Zaznaczenie pola „Use time marker” powoduje wyświetlanie daty i czasu na wskaźniku wodospadowym.
Przycisk „Gradient” powoduje otwarcie okna modyfikacji skali kolorów na wskaźniku wodospadowym.
Suwaki „S-Attack”, „S-Decay”, „W-Attack” i „W-Decay” regulują odpowiednio dla wskaźnika widma i wodospadowego reakcję na narastanie i opadanie odebranych sygnałów.
Ostatrni regulator suwakowy – u dołu – służy do zmiany szybkości odświeżania wskaźników w tych wersjach programu, w których nie jest on umieszczony po prawej stronie z boku okna.
Sekcja „Zoom FFT” zawiera pola służące do włączenia dodatkowych wskaźników widma p.cz. („IF”), kompletnego sygnału stereofonicznego („MPX”) i widma m.cz. („Audio”).



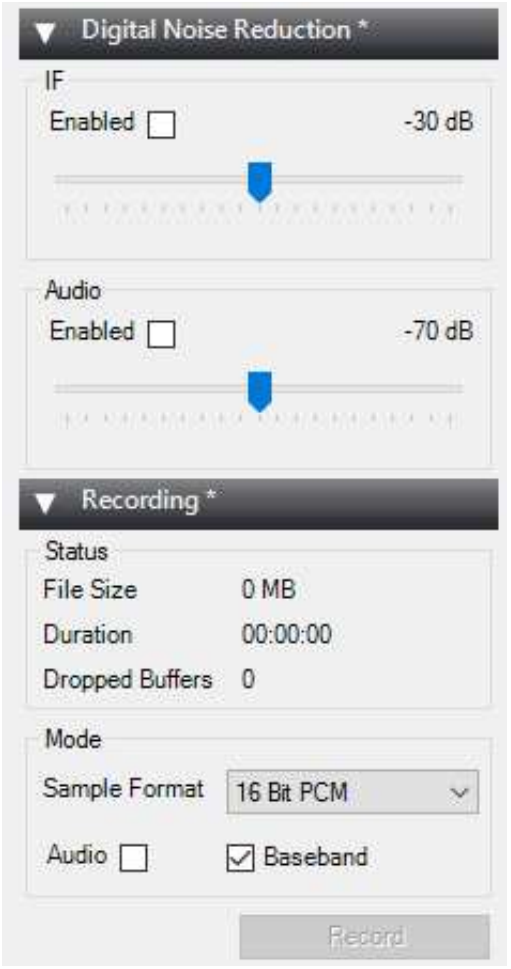
Rys. 1.4.6. Sekcje „Zoom FFT” i eliminatora zakłóceń

W sekcji eliminatora zakłóceń impulsowych znajduje się jego wyłącznik („Enable”), suwak regulacji progu i suwak szerokości impulsu wyciszającego.

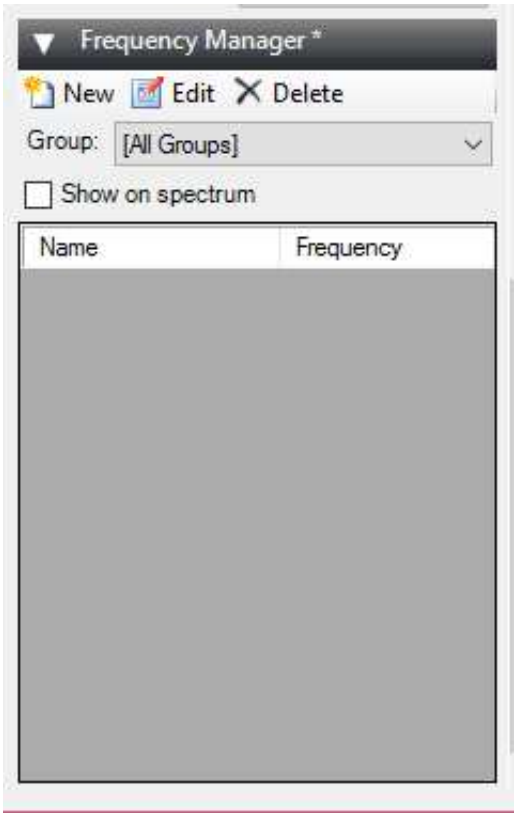
Sekcja cyfrowego eliminatora szumów zawiera włączniki i regulatory odpowiednio dla eliminatorów w torze częstotliwości posredniej („IF”) i małej („Audio”).

W dolnej ramce sekcji nagrywania („Mode”) wybierany jest z rozwijanego spisu format nagrywanych danych a w polach poniżej ich rodzaj – dane m.cz. („Audio”) lub całe pasmo podstawowe („Baseband”). Przycisk „Record” służy do włączenia nagrywania, a w jego trakcie – do zaprzestania.
Górna ramka („Status”) informuje o przebiegu nagrania: wielkości pliku, czasie nagrania i buforach.

Sekcja bazy danych częstotliwości stacji (rys. 1.4.8) zawiera w górnym menu przyciski do wpisywania nowych danych „New”), modyfikowania istniejących wpisów („Edit”) i do ich kasowania („Delete”).
W rozwijanej liście poniżej wybierane są grupy wyświetlanych wpisów: wszystkie wpisy lub też tylko ulubione. Po zaznaczeniu pola „Show on spectrum” dane ze spisu mogą być zaznaczone na wskaźniku widma.



Rys. 1.4.7. Sekcje eliminatora szumów i nagrywania



Rys. 1.4.8. Sekcja bazy danych (programu dodatkowego)

Wskaźniki widma i wodospadowy

Na wskaźniku widma wyświetlane jest ciągłe widmo sygnału w.cz. w wybranym podzakresie. Podzakres ten zależy od:

1. Wybranej dla odbiornika częstotliwości próbkowania. Określa ona jaki podzakres jest przetwarzany na postać cyfrową przesyłaną do komputera do dalszego przetwarzania w SDRSharp.
2. Ustawień wskaźnika wybranych w sekcji „FFT Display".
3. Ustawienia powiększenia za pomocą suwaka „Zoom" („Powiększenie") znajdującego się po prawej stronie okna. W dolnej pozycji suwaka wyświetlany jest maksymalny podzakres przesyłany z odbiornika do komputera. W miarę przesuwania go do góry następuje zawężenie wyświetlanego z coraz większą dokładnością podzakresu.

Poprzez ocenę wyglądu sygnału i względnych wartości jego składowych w stosunku do maksimum można po nabyciu pewnej wprawy prawidłowo rozpoznawać sygnały różnych emisji.
Cenną właściwością SDR# jest możliwość dopasowywania pracy wyświetlaczy do własnych aktualnych potrzeb.
Ustawienia parametrów szybkiej transformaty Fouriera decydują o podstawowej rozdzielczości wyświetlacza widma. Dla większości zastosowań wystarczy umiarkowana liczba wykorzystywanych punktów – 4096. Przeważnie nie obciąża ona też nadmiernie procesora. Zbyt duża liczba punktów może spowodować nadmierne obciążenie dające urywany sygnał z głośnika. Zaleca się korzystanie tylko z takiej ilości punktów, która jest niezbędna w danej sytuacji ale nie z większej.
Powiększenie pozwala na dokładniejsze obejrzenie wycinka pasma w sytuacji kiedy jest to potrzebne, np. dla ułatwienia precyzyjnego dostrojenia do stacji lub dla dokładniejszej analizy sygnału.
Dostrojenie do stacji następuje też przez naciśnięcie w odpowiednim miejscu na wskaźnik, ale możliwe jest też przeciąganie myszą po naciśnięciu go lewym klawiszem i przesuwanie bez puszczenia go – czyli w sposób typowy dla funkcji przeciągania pod Windows.

Dekodowanie emisji cyfrowych

Dekodowanie amatorskich i niektórych profesjonalnych emisji cyfrowych odbywa się identycznie jak w przypadku wszystkich innych rodzajów odbiorników za pomocą programów terminalowych: MultiPSK, Fldigi, MixW itp. Do dekodowania lotniczych komunikatów ACARS („Aircraft Communications Addressing and Reporting System") służą programy ACARSD, PDW lub MultiPSK. Komunikaty te są nadawane w pierwszym rzędzie na częstotliwościach 131,525, 131,550, 131,725 i 136,900 MHz. Do ich odbioru konieczna jest pionowa antena zewnętrzna. W podobny sposób można również odbierać obrazy z satelitów meteorologicznych nadających w paśmie 137 MHz.
Do odbioru radiofonii DAB+ służy natomiast pakiet oprogramowania SDR-J.
O ile jednak w przypadku odbiorników klasycznych wystarczyło doprowadzenie kablem sygnału z wyjścia głośnikowego lub słuchawkowego odbiornika do wejścia mikrofonowego lub linii komputera. W sytuacji gdy funkcje odbiorcze realizowane są programowo konieczna jest wymiana danych pomiędzy programem odbiorczym a terminalowym. Najwygodniejszym rozwiązaniem jest w takiej sytuacji skorzystanie z wirtualnego programowego kabla połączeniowego. Najpopularniejszymi z programów tego rodzaju są „Virtual Audio Cable" (VAC) i „free VB-Cable".

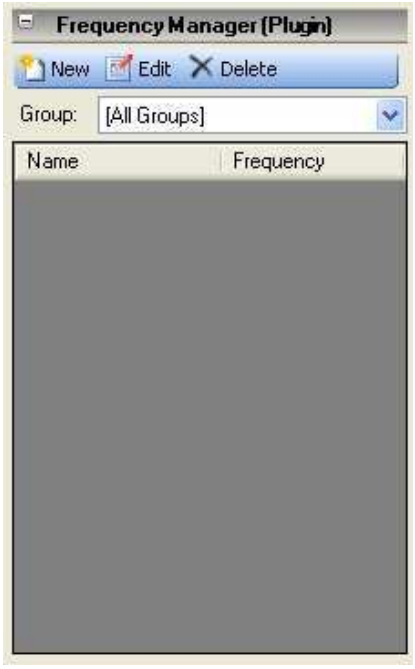
Programy dodatkowe

Dzięki dodatkowym programom takim jak program nagrywający („Audio Recorder") albo miernik poziomu sygnału („Pegelmesser") możliwe jest rozszerzenie funkcjonalności programu SDRSharp. Tabela 1.5.1. zawiera informacje o programach dostępnych w czasie powstawania skryptu. Z biegiem czasu warto jednak będzie poinformować się w Internecie o nowościach. Część z nich wyświetla swoje okna konfiguracyjno-sterujące w postaci oddzielnych sekcji w bocznym menu SDRSharp.

Tabela 1.5.1. Programy rozszerzające funkcjonalność SDRSharp

Nazwa	Funkcje programu
Zoom FFT	Powiększenie obrazu na wyświetlaczu widma. Ułatwia precyzyjne strojenie. Dodatkowy filtr i płynne przesuwanie pasma p.cz.
Audio FFT	Wyświetlanie widma sygnałów m.cz.
Noise Blanker	Eliminator zakłóceń impulsowych o regulowanym progu i czasie wyciszania.
Noise Reduction	Eliminacja szumów p.cz. i m.cz. o regulowanych progach.
Recorder	Nagrywanie sygnałów strumieni IQ i m.cz.
Frequency Manager	Zarządzanie bazą danych częstotliwości
Jeff Knapp Scanner	Rozbudowana baza danych częstotliwości i funkcje przeszukiwania pasm.
Vasili's Scanner	Rozbudowana baza danych częstotliwości i funkcje przeszukiwania pasm. *http://www.rtl-sdr.ru/*
Vasili's CTCSS Decoder	Dekoder tonów CTCSS i blokada szumów CTCSS. http://www.rtl-sdr.ru/
Vasili's Recorder	Nagrywanie sygnału m.cz. lub sygnału w pasmie podstawowym. Konwerter częstotliwości próbkowania. *http://www.rtl-sdr.ru/*
Vasili's Audio Processor	Filtr pasmowy, inwerter pasma m.cz., deemfaza, blokada szumów sterowana mową (VOX). *http://www.rtl-sdr.ru/*
Satellite Tracker	Za pośrednictwem programu „My DDE Client 1.05" współpracuje z „Orbitronem". Klient DDE i Orbitron są dostępne pod adresem *http://www.stoff.pl/downloads.php*.
ACARS Decoder	Dekoder lotniczych komunikatów ACARS

„Frequency Manager"



Rys. 1.5.1. Okno główne

Okno główne „Frequency Managera” zawiera następujące elementy:
Pole „New” – dodanie aktualnego ustawienia częstotliwości, środka i emisji do pliku *frequency.xml*.
Pole „Edit” – umożliwia modyfikację wpisów na liście.
Pole „Delete” – służy do skasowania wybranego wpisu.
Spis grup („Group”) pozwala na wybór plików założonych grup, wszystkich grup lub ulubionych.

„Recorder”

Rys. 1.5.2. Okno główne

Okno programu zawiera informacje o wielkości nagranego pliku, czasie nagrania, formacie danych i ich rodzaju (sygnał m.cz. lub całe pasmo podstawowe)

Współpraca z „Orbitronem”

Program śledzący tory satelitów i innych pojazdów kosmicznych „Orbitron” i klienta DDE pn. „My DDE Client 1.05“ można pobrać z witryny autora *http://www.stoff.pl/downloads.php*. Po pobraniu i zainstalowaniu Orbitronu należy zaktualizować dane satelitów za pomocą funkcji aktualizacyjnej programu.
W pliku setup.config w katalogu „Config” „Orbitrona” należy na końcu dodać linie:
[Drivers]
MyDDE=C:\Program Files\orbitron\DDE\MyDDE.exe
SDRSharp=SDRSharp.exe
Do programu SDRSharp należy pobrać dodatek „Satellite Tracker”.
W pliku *SDRSharp.exe*.config znajdującym się w katalogu programu należy na końcu dodać pustą linię i następujące linie

```
<sharpPlugins>
    <add key="SatelliteTracker"
value="SDRSharp.SatelliteTracker.SatelliteTrackerPlugin,SDRSharp.SatelliteTr
acker" />
```

HDSDR

Instrukcja obsługi programu w wersji 2.70

Spis treści

Wstęp

HDSDR (obecnie w wersji 2.70) jest następcą popularnego dawniej programu „Winrad HD" autorstwa Alberta di Bene, I2PHD. Program pracuje pod wersjami Windows XP, 7, 8.1 i 10.
Podobnie jak SDR# również HDSDR może współpracować z wieloma typami odbiorników programowalnych, do których należą m.in. telewizyjne odbiorniki DVB-T z procesorem RTL2832U i oparte na nich bardziej rozbudowane rozwiązania (przykładowo DXpatrol), Fun Cube Dongle Pro+, FiFi SDR, Elad FDM-S1, Elad FDM-S2, Elad FDM-DUO, Bonito RadioJet 1102S, PM-SDR, Softrock, FA-SDR, WINRADIO, SDR-IQ, SDR-14, HPSDR, PERSEUS, Elecraft KX3 i wieloma innymi.
HDSDR nie komunikuje się ze sprzętem bezpośrednio, a za pośrednictwem dodatkowych bibliotek ExtIO_xxx.dll specyficznych dla każdego urządzenia, dzięki czemu możliwe staje się korzystanie z nowych konstrukcji w miarę ich pojawiania się. Biblioteki ExtIO są dostępne w witrynach producentów lub konstruktorów sprzętu, często pod zbiorczą nazwą „Winrad ExtIO". Odbiorniki DVB-T oparte na RTL2832U wymagają także instalacji sterownika Zadig natomiast Fun Cube Dongle Pro+ nie wymaga żadnego. Szczegółowe informacje o niezbędnych bibliotekach i sterownikach podane są w instrukcjach sprzętu i na witrynach internetowych producentów. Sterowniki i dodatkowe biblioteki powinny być zainstalowane w tym samym katalogu, w którym znajduje się HDSDR.
Program jest wyposażony w oddzielne wskaźniki wodospadowe dla sygnałów w.cz. i m.cz. demodulatory emisji AM (w tym demodulator synchroniczny), FM, SSB i CW, umożliwia nadawanie tymi emisjami, umożliwia też zapis odbieranych sygnałów w.cz. i m.cz. oraz odtwarzanie nagranych plików, zdalne sterowanie sprzętem, współpracę z programami terminalowymi dla emisji cyfrowych oraz współpracę poprzez kanały DDE z takimi programami jak Orbitron, Ham Radio Deluxe, WXTrack i innymi.
Ustawienia konfiguracyjne mogą być zapisywane w oddzielnych profilach co ułatwia korzystanie z różnego wyposażenia lub w różnych sytuacjach albo też przez różnych użytkowników.
Przedstawione w dalszej części instrukcji ujęcia ekranowe i zestawy funkcji mogą się nieco różnić dla poszczególnych wersji programu.

Instalacja

Aktualną wersję HDSDR można pobrać z Internetu z witryny *www.hdsdr.de*. Po pobraniu i zapisaniu archiwum na dysku należy wywołać program instalacyjny *HDSDR_install.exe*.

HDSDR może być wprawdzie instalowany w katalogu położonym w standardowej gałęzi *C:\Program Files\HDSDR* ale począwszy od Windows 7 zwykli użytkownicy mają ograniczone prawa dostępu do katalogów systemowych i wszelkie instalacje i zmiany w nich dokonywane wymagają uprawnień administratora. Znacznie lepszym rozwiązaniem jest zainstalowanie programu w niezależnym katalogu np. C:\HDSDR.
Po zakończonej instalacji katalog programu zawiera pliki:
HDSDR.exe,
delete_settings.cmd,
hdsdr_keyboard_shortcuts.htm,
HDSDR_release_notes.txt,
unins000.dat,
unins000.exe,
WinRadPX.dll.
Dodatkowo do nich konieczne jest pobranie z Internetu bibliotek *ExtIO_xxxx.dll* przeznaczonych do obsługi posiadanych odbiorników. Program instalacyjny *HDSDR_install.exe* można skasować ponieważ nie jest on już potrzebny, ale można go także pozostawić. Na pulpicie można także założyć skrót do wywołania *HDSDR.exe*.
Do obsługi różnych typów odbiorników – przykładowo regulacji wzmocnienia i innych parametrów, strojenia – konieczne jest zainstalowanie dodatkowych bibliotek *ExtIO_xxxx.dll*, j.np. *ExtIO_RTL2832.dll*, *Ext_IO_FCD_g0mjw.dll*, *ExtIO_Si570.dll*, *ExtIO_FiFi.dll*. W sytuacjach gdy obsługiwany ma być tylko jeden typ odbiornika i w związku z tym zainstalowana jest tylko jedna odpowiadająca mu biblioteka program startuje od razu. W przypadku zainstalowania większej liczby bibliotek otwierane jest okno pozwalające na wybór pożądanej biblioteki. Instalacja bibliotek polega na ewentualnym rozpakowaniu ich i skopiowaniu do katalogu zawierającego HDSDR.
Telewizyjne odbiorniki DVB-T oparte na procesorze RLT2832U wymagają też zainstalowania sterownika Zadig. Sposób jego instalacji został opisany w instrukcji do programu SDR#.
Począwszy od wersji 2.13 HDSDR współpracuje także z radiostacjami programowalnymi ale pod warunkiem, że odpowiednia biblioteka *ExtIO_xxxx* udostępnia funkcję nadawania.

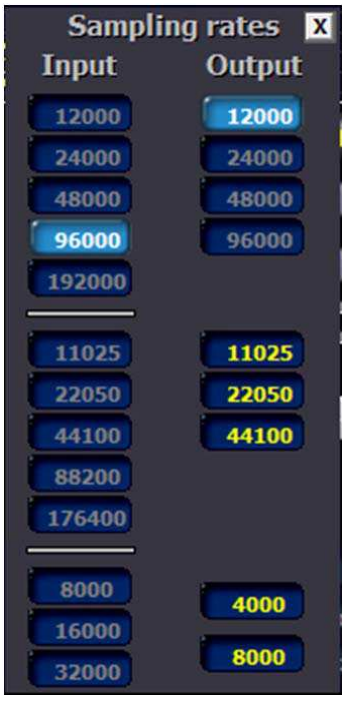
Rys. 2.2.1. Okno wyboru biblioteki ExtIO_xxx

Konfiguracja

Po wywołaniu HDSDR w dolnej lewej części jego okna głównego widoczne są przyciski funkcyjne, z których część służy do skonfigurowania programu. Przycisk „Bandwidth [F6]" służy do ustawienia częstotliwości próbkowania w kanałach wejściowym i wyjsciowym, a „Options [F7]" – zapewnia dostęp do różnych ustawień. Źródło dźwięku, czyli odbiornik, jest wybierane w punkcie „Soundcard [F5]". Przycisk „Start [F2]" powoduje rozpoczęcie odbioru – oczywiście po dokonaniu niezbędnych ustawień.

Rys.2.2.2. Przyciski funkcyjne w oknie głównym HDSDR

Po naciśnięciu przycisku „Samplerate" („Częstotliwość próbkowania") lub klawisza funkcyjnego F6 otwierane jest okno konfiguracyjne dające możliwość wyboru częstotliwości próbkowania oddzielnie dla kanału wejściowego (strumieni IQ) i wyjściowego (zdemodulowanego sygnału).



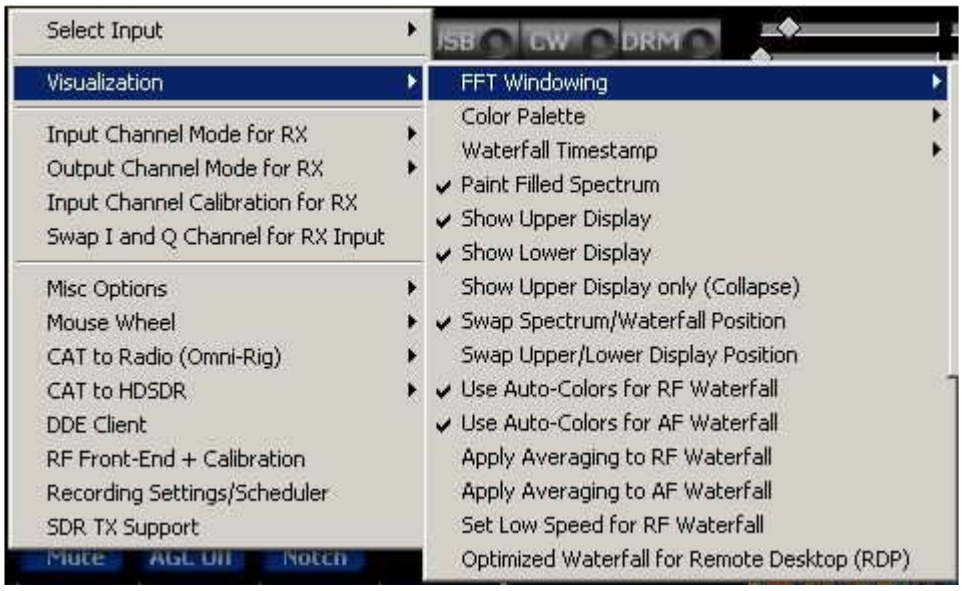
Rys. 2.2.3 Wybór częstotliwości próbkowania (F6)

Częstotliwość próbkowania w kanale wejściowym zależna jest od możliwości technicznych podłączonego odbiornika i przykładowo dla FCD wynosi standardowo 96000 Hz. Decyduje ona także o szerokości wyświetlanego na ekranie podzakresu. Częstotliwość próbkowania wybrana dla kanału wyjściowego decyduje natomiast o maksymalnej odtwarzanej częstotliwości akustycznej. Teoretycznie wynosi ona połowę częstotliwości próbkowania, a w praktyce nieco mniej. Dla wybranej na ilustracji częstotliwości 12000 Hz wynosi więc ona około 6000 Hz czyli 6 kHz i jest wystarczająca do odbioru radiofonii AM albo amatorskich emisji SSB, CW, RTTY itd.

Do odbioru radiofonii FM albo transmisji danych z większymi szybkościami korzystniejsza będzie większa wartość. Po zakończeniu konfiguracji próbkowania okno jest zamykane za pomocą krzyżyka w jego prawym górnym rogu.
Przycisk „Options [F7]" zapewnia dostęp do różnych ustawień dotyczących m.in. źródeł dźwięku i sposobów wyświetlania informacji:

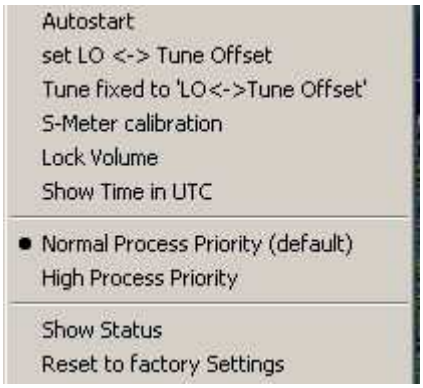
- „Select Input" – służy do wyboru źródła dźwięku (podsystemu dźwiękowego, plików WAV, dostępu do miksera wejściowego Windows itp.),
- „Visualisation" – do wyboru sposobu wyświetlania informacji, pożądanych wskaźników itp.,
- „Input Channel Mode" – dla odbiorników programowalnych (SDR) ustawiany jest tryb IQ, a pozostałe dla odbiorników dostarczających sygnałów mono lub do celów specjalnych,
- „Output Channel Mode for RX" – należy wybrać dwa kanały (I i Q), inne ustawienia do celów specjalnych,
- „Mouse wheel" – wybór parametrów strojenia przy użyciu kółka myszy,
- „Help/Update" – sprawdzanie możliwości aktualizacji i wywołanie pomocy.

Rys. 2.2.4. Menu ustawień różnych (F7) – wybór źródła sygnału



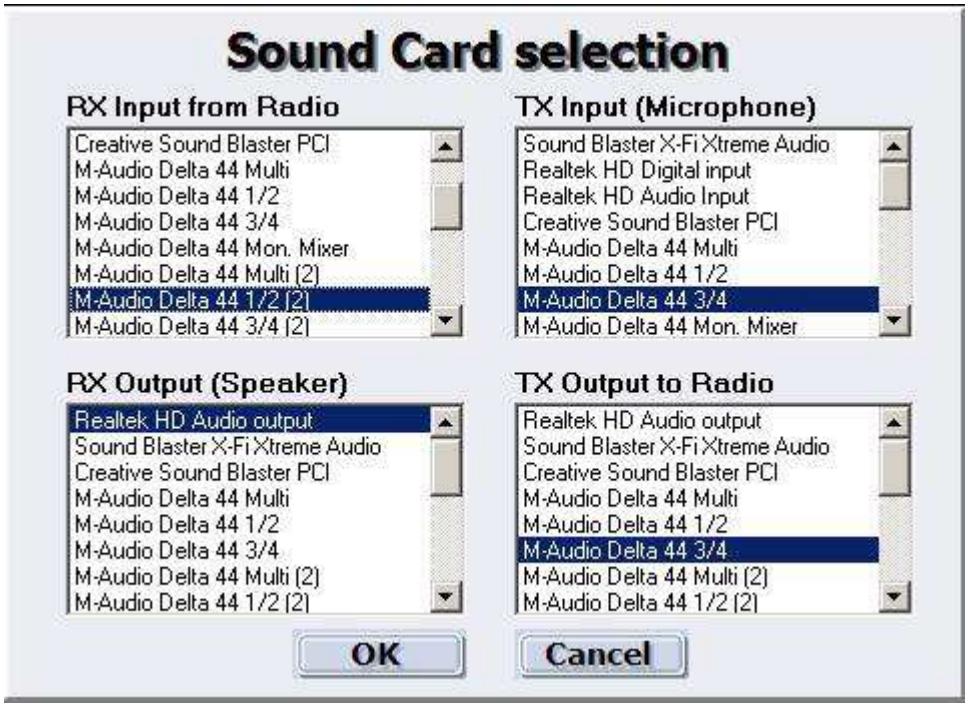
Rys. 2.2.5. Menu ustawień różnych (F7) – wyświetlanie

Pozycja „FFT Windowing” decyduje o sposobie selekcji próbek dla algorytmu szybkiej transformaty Fouriera. Najlepszym sposobem zapoznania się z wpływem ustawień jest ich wypróbowanie.
Pozycja „Color Palette” służy do wyboru zestawu kolorów. Również i tu warto wypróbować możliwości i wybrać najbardziej przypadającą do gustu.
Następna pozycja służy do włączenia wyświetlania daty i czasu na wskaźniku wodospadowym, a dalsze – do wyboru wyświetlanych pól i ich kolorów.
W menu ustawień różnych możliwy jest też wybór kanałów lewego lub prawego albo obu oddzielnie przy odbiorze i nadawaniu, zamiany kanałów I i Q itd.



Rys. 2.2.6. Okno ustawień mieszanych

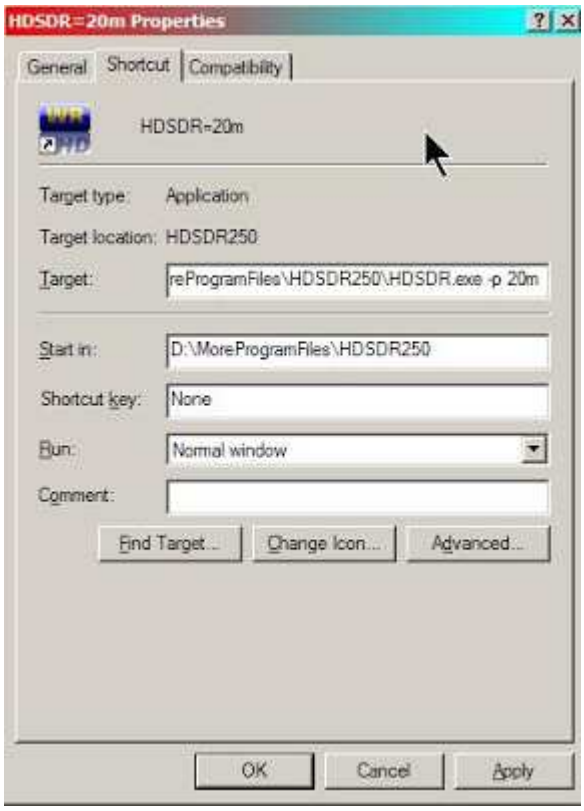
Pozycja „Autostart” po zaznaczeniu powoduje, że HDSDR po uruchomieniu znajduje się od razu w trybie odbiorczym bez konieczności naciskania przycisku „Start”.
Pozycja trzecia powoduje ustalenie stałego odstępu częstotliwości w polach „LO” i „Tune”.
Dla użytkowników pragnących dokonywać dokładniejszych pomiarów poziomu sygnału przewidziano też możliwość kalibracji miernika „S”. Dokładność skalibrowanego miernika wystarcza jednak zasadniczo i tak tylko do pomiarów porównawczych.
Pozostałe punkty mają mniejsze znaczenie lub mogą zachować ustawienia domyślne.



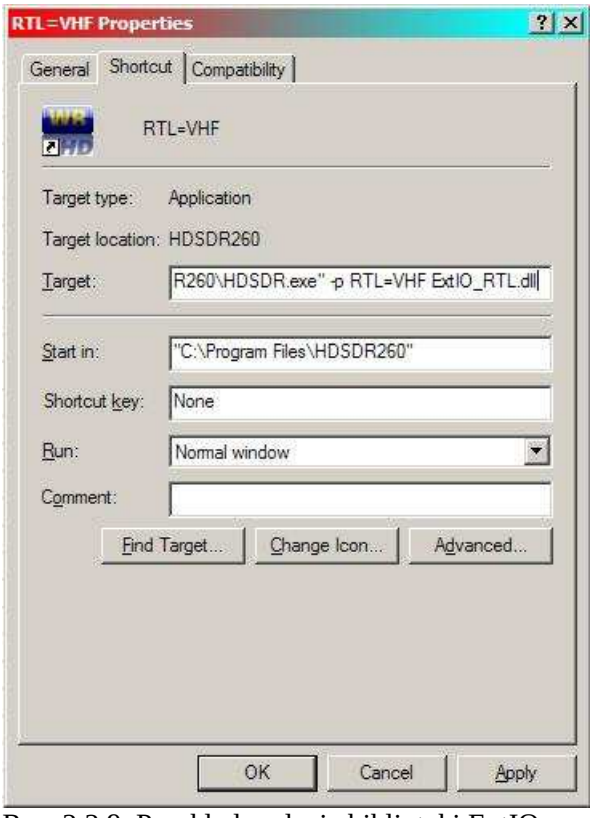
Rys. 2.2.7. Wybór systemu dźwiękowego (F5). W oknie powinny występować nazwy podłączonego sprzętu

Użytkownicy mogą zapisać większą liczbę różnych profili konfiguracyjnych i w zależności od potrzeb wywoływać je podając w wywołaniu po parametrze –p nazwę profilu, np. *HDSDR.exe –p FCD* dla od-

biornika „Fun Cube Dongle Pro+” lub 20m – dla odbioru w paśmie 20 m. Dla każdego z takich wywołań wygodnie jest założyć odpowiedni skrót (symbol) na pulpicie. Wygodnie jest też podać od razu potrzebną bibliotekę *ExtIO_xxx.dll*.



Rys. 2.2.8. Sposób podania profilu we właściwościach symbolu na pulpicie



Rys. 2.2.9. Przykład podania biblioteki ExtIO w wywołaniu (tutaj po profilu RTL=VHF)

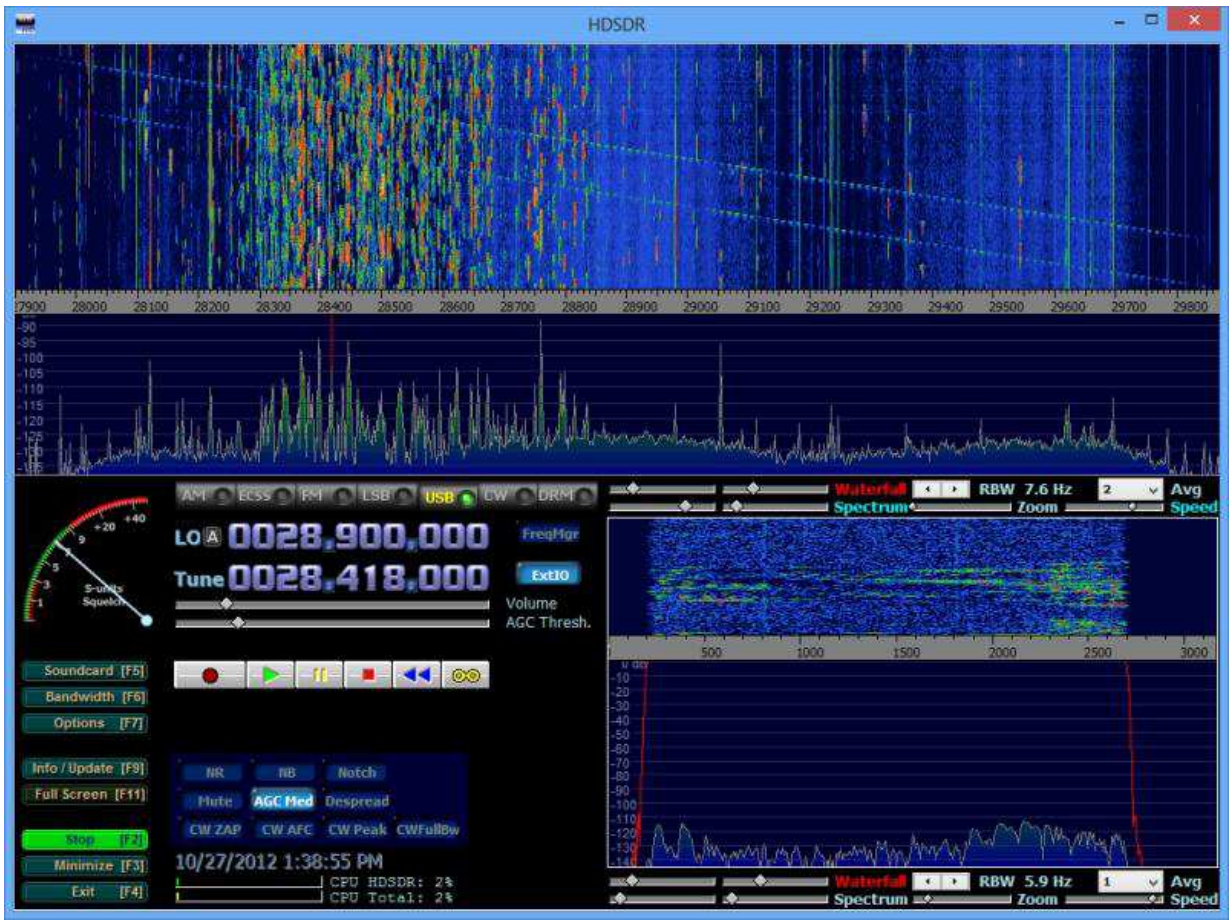
Odbiornik FCD

Korzystanie z odbiornika „Fun Cube Dongpe Pro+" lub z jego poprzedniego modelu wymaga aby w katalogu HDSDR znajdowała się biblioteka *ExtIO_FCD_G0MJW.dll*. Biblioteka nie wymaga żadnej skomplikowanej instalacji wystarczy po pobraniu jej z internetu skopiować do katalogu docelowego. Odbiornik pokrywa zakres 0,15 MHz – 1,9 GHz z przerwą pomiędzy 240 i 420 MHz. Maksymalna szerokość wyświetlanego zakresu wynosi 192 kHz.

Odbiornik SDRplay

Przed podłączeniem odbiornika do PC należy pobrać z witryny *sdrplay.com* program instalujący sterownik i Windows API. Program nosi nazwę *MiricsSDRAPIInstaller_1.0.1.exe* lub podobną różniącą się numerem wersji. Instalacja oprogramowania przebiega w sposób typowy dla środowiska Windows. Dopiero po jej pozytywnym zakończeniu można podłączyć odbiornik do dowolnego złącza USB. Konieczne jest też zainstalowanie biblioteki ExtIO dla SDRplay. Najwygodniej zrobić to przy użyciu programu instalacyjnego dostępnego pod odnośnikiem „SDRplay EXTIO installer" – *SDRPlay_SDRSharp_Plugin_v1.3.exe* lub o podobnej nazwie różniącej się numerem wersji. Biblioteka ExtIO jest identyczna dla HDSDR, SDR# i innych podobnych programów.
SDRplay pokrywa zakres 0,1 – 2000 MHz, jesty wyposażony w 12-bitowy przetwornik a/c, a szerokość wyświetlanego zakresu dochodzi do 8 MHz.

Obsługa



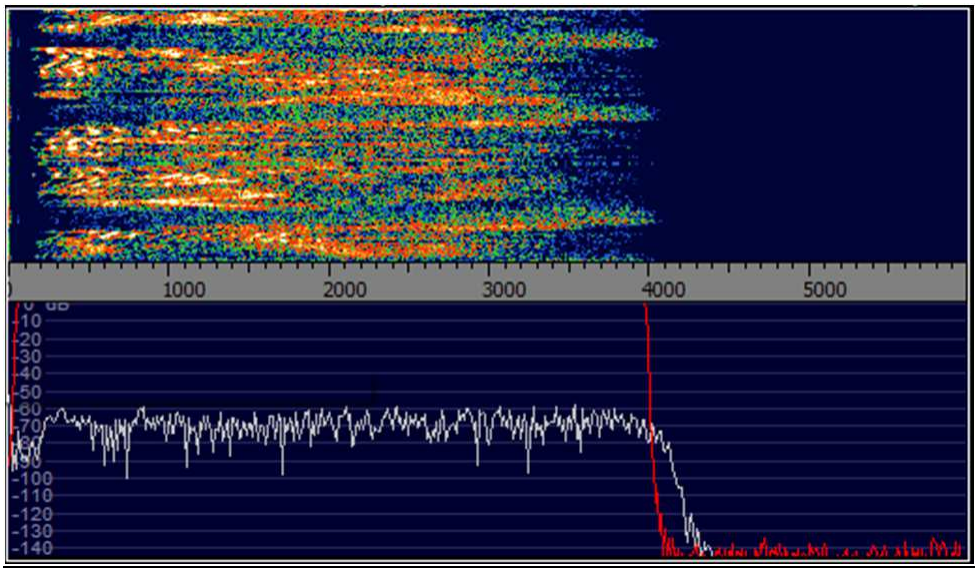
Rys. 2.3.1. Główne okno HDSDR

Po wywołaniu programu – pliku *hdsdr.exe* – otwierane jest jego główne okno zawierające przyciski służące do wywołania najważniejszych funkcji, elementy służące do sterowania odbiornikami i ich strojenia, widma i wskaźniki wodospadowe oraz informacje o pracy sprzętu i programu.
Krótkie informacje o funkcjach poszczególnych elementów wyświetlane są w chmurkach przy przesuwaniu nad nimi znacznika myszy.

Elementy okna głównego

Górną połowę okna zajmują wskaźniki widma i wodospadowy. Wskaźnik widma posiada wprawdzie po lewej stronie skalę w dB ale różnorodność sprzętu, jego ustawień oraz ewentualnie włączona automatyczna regulacja wzmocnienia powodują, że skali tej nie można traktować zbyt dokładnie.
Po najechaniu myszą na widoczne na wskaźnikach sygnały powoduje wyświetlenie dokładniejszych informacji o częstotliwości i sile sygnału.
Zakres skali wskaźnika widma jest ustawiany regulatorem suwakowym znajdującym się po lewej stronie w linii podpisanej „Spectrum” („Widmo”). Suwak drugi od lewej strony służy do ustawienia mimimum skali. Trzeci regulator suwakowy umożliwia rozciągnięcie wskaźnika, a czwarty czyli znajdujący się po prawej stronie – reguluje szybkość jego odświeżania.
Znajdująca się nad nią linia regulatorów suwakowych jest podpisana „Waterfall” i dotyczy wskaźnika wodospadowego. Licząc kolejno od lewej zawiera ona regulatory jasności i kontrastu, rozdzielczości wskaźnika (pasma w Hz – „RBW”) i liczby uśrednianych pomiarów („Avg”).
Naciśnięcie myszą na wybrany sygnał na wskaźnikach powoduje dostrojenie odbiornika do tej stacji.
Pomiędzy obydwoma wskaźnikami znajduje się skala częstotliwości wyskalowana w kHz. Skalę tą można przeciągać w lewo lub w prawo za pomocą myszy. Przeciąganie ich w górę lub w dół zmienia

proporcje obydwu wskaźników. Na życzenie na wskaźniku wodospadowym może być wyświetlana data i czas.
Miernik siły sygnału służy również do ustawiania progu działania blokady szumów poprzez naciśnięcie lewym klawiszem myszy na wybrane miejsce na skali. Nastawiony próg jest zaznaczony czerwoną linią. Naciśnięcie prawym klawiszem myszy uruchamia automatyczny wybór progu.
W prawej dolnej części okna głównego znajdują się wskaźnik widma zdemodulowanego sygnału i znany z wielu innych programów wskaźnik wodospadowy. Szerokość przenoszonego pasma można zmieniać przez przeciąganie zboczy filtru na wskaźniku widma.
W dolnej lewej części znajdują się elementy służące do obsługi odbiornika, strojenia, wyboru emisji, charakterystyki ARW, eliminatorów szumów i zakłóceń, filrów zaporowych, wskaźnik siły odbioru itp.

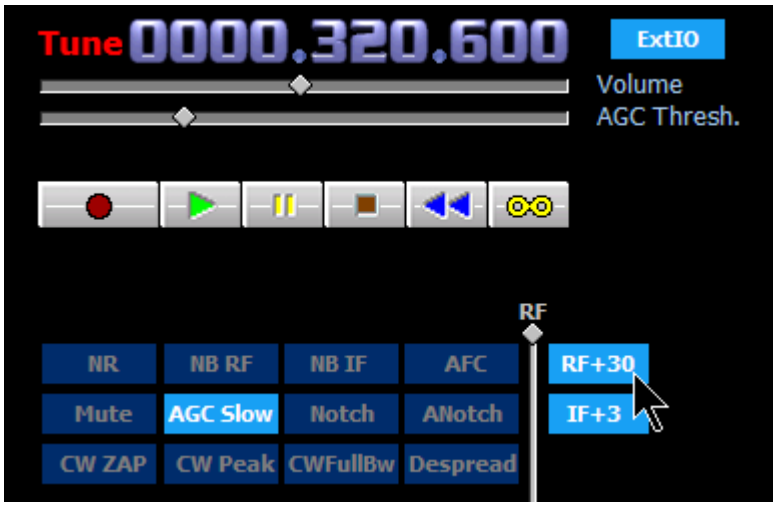


Rys. 2.3.2. Wskaźniki widma i wodospadowy sygnału zdemodulowanego w oknie głównym. Czerwoną linią zaznaczona jest charakterystyka przenoszenia filtru. Można ją przeciągać myszą

Rys. 2.3.3. Pole obsługi odbiornika i nagrywania

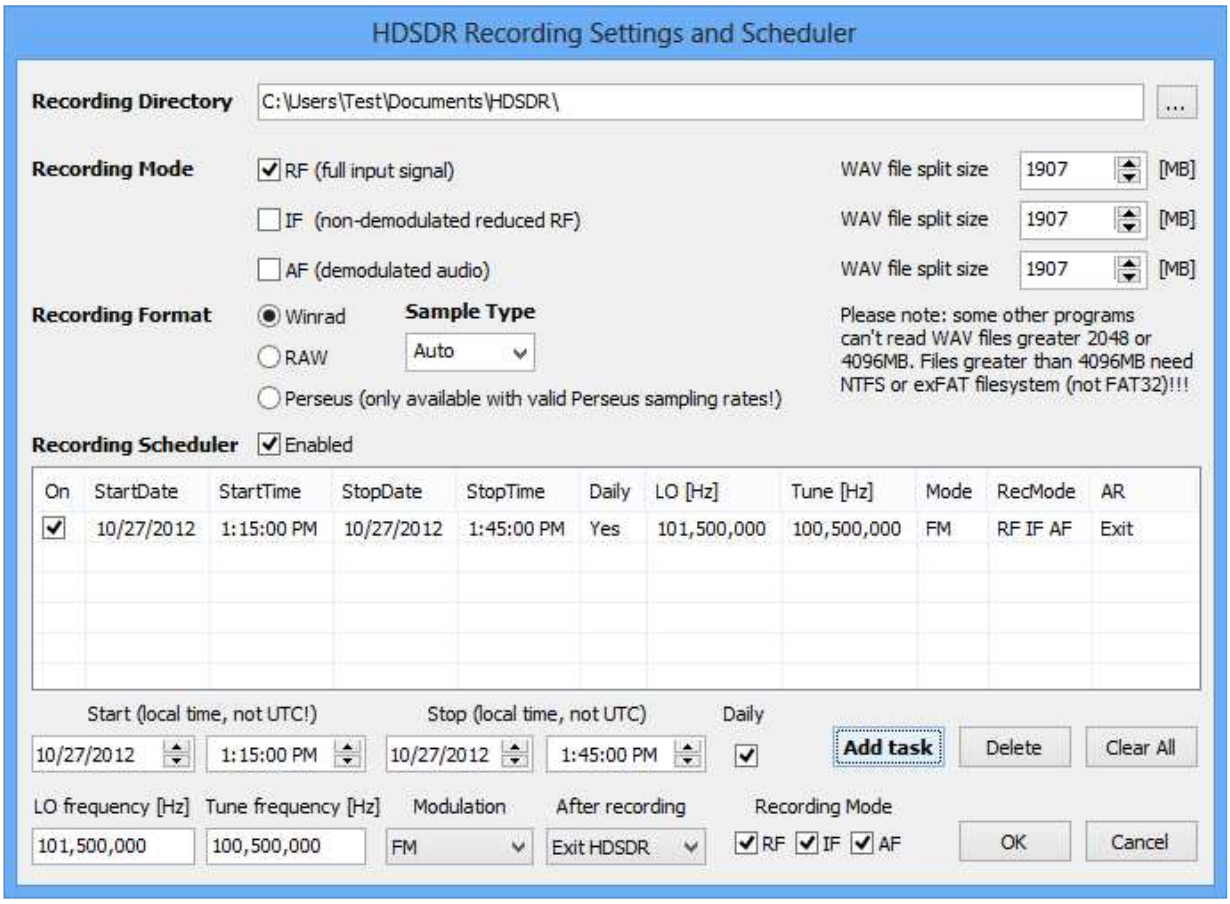
U góry pola znajduje się rząd przycisków służących do wyboru demodulowanej emisji. Przycisk AM powoduje włączenie demodulatora amplitudy np. do odbioru radiofonii na falach długich, średnich lub krótkich. Jego naciśnięcie, podobnie jak naciśnięcie każdego innego poza modulacją FM powoduje wyświetlenie poniżej suwakowego regulatora progu działania ARW („AGC Thresh.”). Przycisk wybranej emisji zmienia kolor napisu i wyświetla się na nim zielony sygnalizator. W trakcie odbioru modulacji FM po prawej stronie obok omówionych dalej przycisków wyświetlany jest pionowy suwakowy regulator szerokości pasma.
HDSDR pozwala na nagrywanie na dysku zarówno strumieni I/Q jak i zdemodulowanych sygnałów m.cz. i oczywiście odtwarzania tych nagrań, ich przewijania itd. Możliwe jest także automatyczne nagrywanie według ustalonego rozkładu. Ostatni, prawy, przycisk powoduje odtwarzanie nagrania w pętli.
Elementy służące do nagrywania i odtwarzania przypominają elementy znane z magnetofonów i znajdują się w środkowej części pola widocznego na ilustracji 2.3.4.
Poniżej znajduje się grupa przycisków służących do wywołania różnych funkcji przydatnych w trakcie odbioru:

- „NR” – służy do włączenia eliminatora szumów, przy włączonym eliminatorze po prawej stronie obok przycisków wyświetlany jest pionowy regulator progu działania,
- „NB” – do włączenia eliminatora zakłóceń impulsowych („NB RF” – w torze w.cz., „NB IF” – w torze m.cz.), przy włączonym eliminatorze po prawej stronie obok przycisków wyświetlany jest pionowy regulator progu działania,
- „Notch” – do włączenia filtru zaporowego wycinającego sygnały zakłócające,
- „Anotch” – do włączenia automatycznego filtru zaporowego,
- „Mute” – do wyciszenia głośnika
- „AGC” – do wyboru stałej czasu ARW – reakcji wolnej, średniej lub szybkiej,
- „AFC” – do włączenia automatycznego dostrojenia, ARCz, niedostępne dla SSB i DRM,
- „Despread” – do włączenia funkcji skupiającej widmo sygnałów telegraficznych rozproszonych w wyniku warunków propagacji, np. na trasie połączenia przez odbicie od księżyca,
- „CW ZAP” – poszukiwanie i wychwytywanie maksimum sygnału CW w pobliżu częstotliwości dostrojenia,
- „CW AFC” – do włączenia automatycznego dostrojenia do sygnału telegraficznego,
- „CW Full BW” – wyświetlanie sygnału m.cz. z maksymalną szerokością (tylko dla CW),
- „CW Peak” – filtracja sygnału CW w okolicy maksimum, po prawej stronie wyświetla się regulator szerokości zakresu.



Rys. 2.3.4. Przyciski wywołania różnych funkcji odbiorczych

Suwak z podpisem „Volume” służy do regulacji siły głosu a naciśnięcie jego podpisu powoduje wyciszenie głośnika. Jego włączenie następuje po ponownym naciśnięciu głośnika myszą.



Rys. 2.3.4. Okno rozkładu nagrywania

W pierwszej linii wybierany jest katalog, którym są zapisywane pliki z nagraniami. U dołu okna ustawiane są częstotliwości heterodyny i dostrojenia, rodzaj emisji i akcja podejmowana przez program po zakończeniu nagrania. Może to być wyjście z HDSDR („Exit HDSDR") lub pozostanie w programie „None"). W oknie należy także wybrać rodzaj nagrywanego sygnału: pełny wygnał w.cz. („RF"), wyselekcjonowany sygnał p.cz. („IF") lub zdemodulowany sygnał m.cz. wybranej stacji („AF").
Po ustawieniu wszystkich pożądanych parametrów termin nagrania jest wpisywany do tabeli za pomocą przycisku „Add task" („Dodaj termin"). Przycisk „Delete" służy do skasowania wybranej linii z tabeli, a przycisk „Clear all" – do usunięcia całej jej zawartości.

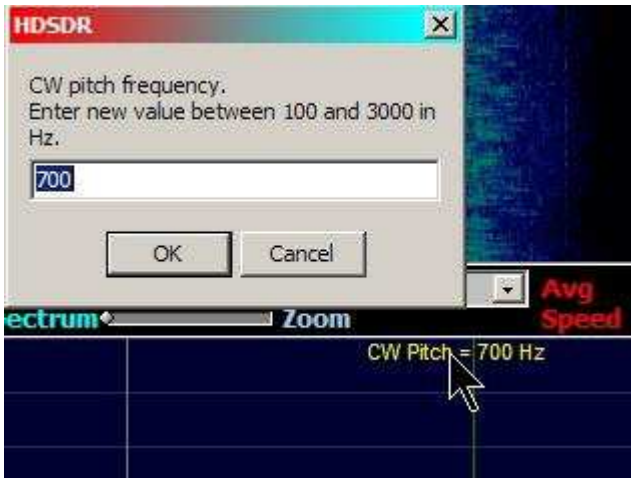
Strojenie odbiornika

Częstotliwości pracy (dostrojenia) są podawane w Hz. Linia z podpisem „LO" podaje częstotliwość środkową wyświetlanego podzakresu, a linia podpisana „Tune" – częstotliwość dostrojenia odbiornika w tym podzakresie. Wyjście poza wyświetlany podzakres wymaga zmiany częstotliwości w linii „LO". Dla odbiorników sterowanych kwarcowo – dostrojonych do stałej częstotliwości – należy tu podać częstotliwość kwarcu. Oznaczenie „A" lub „B" pomiędzy podpisem „LO" i wartością częstotliwości heterodyny informuje o wybranym VFO A lub B. Wyboru dokonuje się przez naciśnięcie litery.
W celu przestrojenia heterodyny można nacisnąć wybraną pozycję liczby i za pomocą kółka myszy albo klawiszy strzałek w prawo lub w lewo zmienić jej wartość.
Naciśnięcie prawym klawiszem myszy podpisu linii „LO" lub „Tune" powoduje zablokowanie strojenia tej częstotliwości, a naciśnięcie lewym klawiszem myszy podpisu „Tune" powoduje otwarcie okna bezpośredniego cyfrowego wprowadzania częstotliwości.
Częstotliwość pracy można też wprowadzić na klawiaturze. Bez dodatkowej litery na końcu jest ona interpretowana przez program jako wartość w kHz, a z literą „M" na końcu – jako wartość w MHz, przykładowo 14.2 M oznacza 14,2 MHz. Na zakończenie konieczne jest naciśnięcie klawisza „Enter". Dodatkowe wygodne możliwości strojenia podano w rozdziale poświęconym znaczeniu klawiszy i ich kombinacji.

Naciśnięcie prawym klawiszem myszy na linię „LO” lub „Tune” pozwala na zablokowanie zmian ich wartości. Napis na przycisku automatycznego dostrajania („AFC”) w kolorze zielonym informuje o włączeniu funkcji.

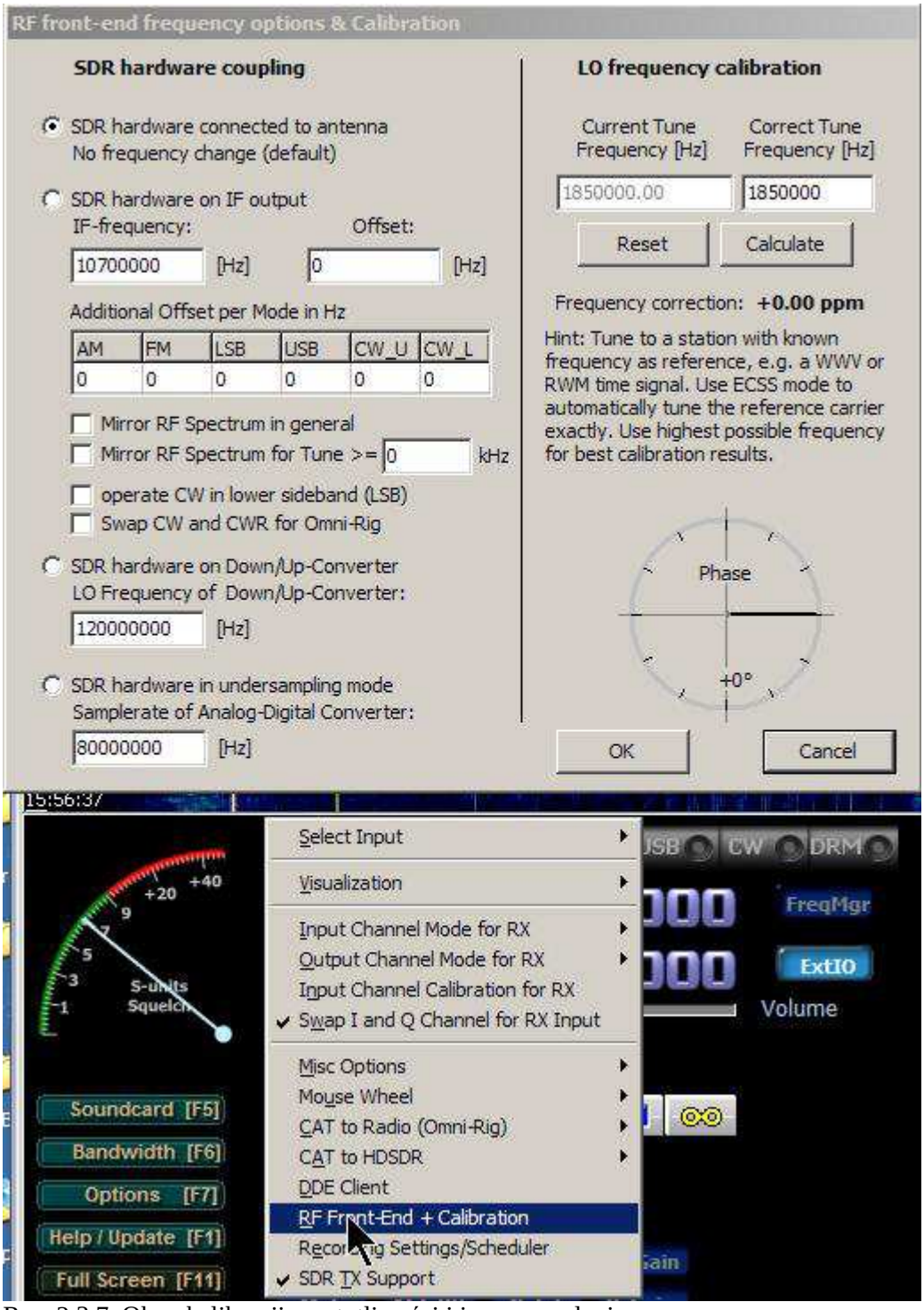
Rys. 2.3.5. Wybór kroku strojenia dla kółka myszy. Pozycja „Direction” w środkowym menu pozwala na zmianę kierunku przestrajania za pomocą kółka

W celu zmiany częstotliwości dudnienia przy odbiorze telegrafii należy nacisnąć klawisz CTRL i naciskając lewym klawiszem myszy linię oznaczająca środkową częstotliwość filtru przeciągnąć ją w pożądaną stronę. Drugim sposobem jest naciśnięcie myszą na pożądaną częstotliwość trzymając naciśnięty klawisz CTRL. Trzecią z możliwości jest naciśnięcie myszą na żółty napis „CW pitch” („Ton dudnieniowy CW”) co powoduje otwarcie okna dialogowego pokazanego na ilustracji 2.3.6.



Rys. 2.3.6. Bezpośrednie wprowadzanie częstotliwości tonu CW

Kalibracja częstotliwości



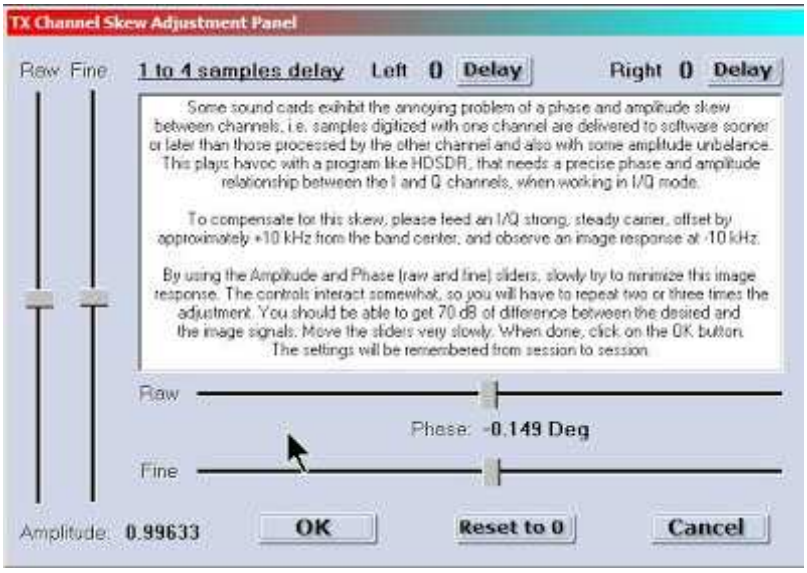
Rys. 2.3.7. Okno kalibracji częstotliwości i jego wywołanie

Użycie funkcji kalibracji HDSDR jest zalecane tylko dla odbiorników wyposażonych w syntezer inny niż Si570 i zasadniczo tylko gdy odbiornik nie dysponuje własnymi możliwościami kalibracji.
Okno kalibracji jest otwierane poprzez menu programu lub po naciśnięciu prawym klawiszem myszy przycisku „ExtIO” na ekranie.
W lewejczęści okna kalibracji należy podać czy odbiornik jest połączony bezpośrednio z anteną czy też do wyjścia p.cz. radiostacji (albo innego odbiornika). W tym drugim przypadku konieczne jest wpisanie czestotliwości pośredniej tego urządzenia i ewentualnych odstrojeń od niej (poprawek) dla różnych rodzajów emisji. Dwie dalsze możliwości dotyczą podłączenia odbiornika do wyjścia konwertera (należy

wówczas podać jego częstotliwość wyjściową – pośrednią) lub odbioru z wykorzystaniem harmonicznej częstotliwości próbkowania (próbkowaniem podharmonicznym).
Odbiornik należy dostroić do stacji o znanej dokładnie częstotliwości, najlepiej do wzorca częstotliwości, wpisać poprawną częstotliwość do pola po prawej stronie („Current Tune Frequency”) a następnie nacisnąć przycisk „Calculate” („Oblicz”). Dokładność kalibracji jest większa dla stacji pracujących na wyższych częstotliwościach.
Dokładne dostrojenie do odbieranej częstotliwości najłatwiej osiągnąć korzystając z detektora synchronicznego (ECSS) i dostrajając odbiornik tak, aby wskaźnik fazy nie poruszał się.

Tłumienie sygnałów lustrzanych

Słabą stroną odbiorników z bezpośrednią przemianą częstotliwości, w tym także odbiorników programowalnych (SDR) jest występowanie symetrycznie w stosunku do środka pasma i w niewielkiej odległości od sygnałów odbieranych ich odbić zwierciadlanych. Dzięki odbiorowi kwadraturowemu tzn. występowaniu zarówno sygnału synfazowego I jak i kwadraturowego Q możliwa jest ich eliminacja (kompensacja), co wymaga zapewnienia odpowiednio dobrej symetrii amplitudy i fazy obu kanałów. Symetria ta zależna jest wprawdzie od konstrukcji odbiornika albo użytego podsystemu dźwiękowego ale w pewnym stopniu daje się także skorygować programowo.
Dla niektórych odbiorników j.np. FCD efekty te praktycznie nie występują w normalnej pracy a jedynie mogą się zdarzyć w wyjątkowych sytuacjach, przykładowo po powtórnym wystartowaniu Windows bez wyłączania komputera, albo po zamianie w programie strumieni I i Q. Najprostszym rozwiązaniem może być wyjęcie odbiornika z gniazda USB i ponowne jego włączenie co spowoduje jego powtórną inicjalizację.
W innych sytuacjach lub dla innych modeli odbiorników można skorzystać z funkcji korekcyjnej programu. W celu dokonania programowej korekcji symetrii należy dostroić odbiornik do stabilnej nośnej tak aby znajdowała się ona ok. +10 kHz od srodka kanału. W pobliżu częstotliwości -10 kHz daje się wówczas zaobserwować sygnał zwierciadlany. Za pomocą suwaków amplitudy i fazy (najpierw kompensacji zgrubnej a potem dokladnej) należy starać się uzyskać minimum sygnału o częstotliwości zwierciadlanej. Regulacje te wpływają do pewnego stopnia wzajemnie na siebie dlatego konieczne jest ich kilkukrotne powtarzanie na przemian. Możliwe jest w ten sposób uzyskanie wytłumienia sygnału lustrzanego nawet o 70 dB w stosunku do pożądanego. Suwaki powinno się przesuwać bardzo powoli i ostrożnie. Na zakończenie procesu należy nacisnąć przycisk OK.
Ustawione w ten sposób parametry są zapamiętywane przez program i używane po jego następnych wywołaniach.



Rys. 2.3.8.

Funkcja kompensacji jest dostępna po naciśnięciu przycisku „Options” („Opcje”) na ektranie lub po naciśnięciu klawisza funkcyjnego F7 i nosi nazwę „Channel skew compensation...” („Kompensacja niesymetrii kanałów...”).

Poszczególne modele odbiorników różnią się między sobą liczbą i jakością zainstalowanych filtrów wejściowych. W niektórych sytuacjach mogą one okazać się niewystarczające i konieczne może być dodanie filtrów zewnętrznych, fabrycznych lub własnej konstrukcji.
Dla zmniejszenia wpływu zakłóceń powodowanych przez komputer korzystnie jest podłączyć odbiornik nie bezpośrednio do jego gniazda USB, a za pomocą kabla.

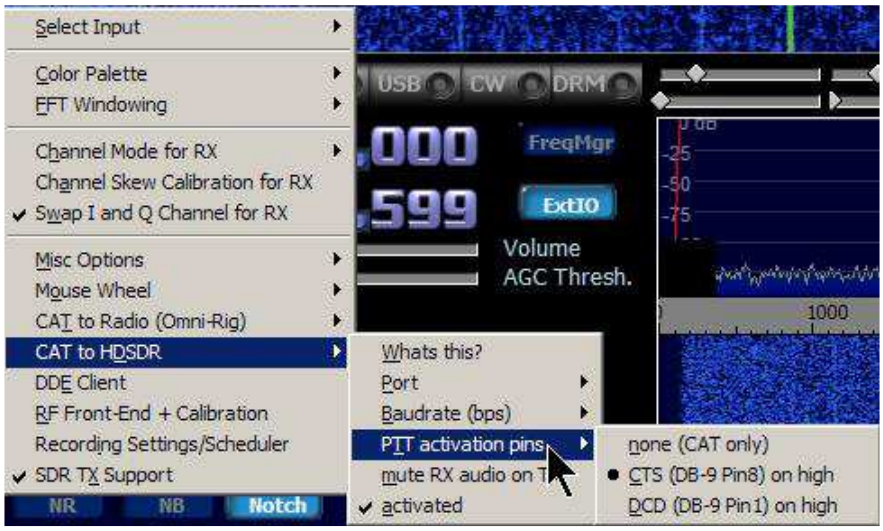
Odbiór satelitów

Do kompensacji efektu Dopplera przy odbiorze staelitów za pomocą FCD został opracowany program SATCONTROL_FCD.exe. Przestraja on odpowiednio do potrzeb heterodynę odbiornika za pomocą funkcji biblioteki *ExtIO_FCD_G0MJW.dll*. Program jest dostępny w Internecie w postaci skomprymowanego archiwum w formacie ZIP. Należy go po rozpakowaniu skopiować do katalogu, w którym znajduje sięHDSDR. W tym samym katalogu znajdują się też pliki TLE zawierające parametry orbit satelitów.
SATCONTROL_FCD.exe powinien być wywołany po uruchomieniu HDSDR.

Sterowanie i współpraca radiostacjami

HDSDR posiada wbudowane funkcje zgodne z normą OmniRig co pozwala na sterowanie za pośrednictwem złącza CAT kompatybilnych z tym radiostacji takich jak np. FT-847, IC-9100. Funkcja sterowania „OmniRig (CAT)” znajduje się w menu „Options [F7]”. Pozwala to na wykorzystanie odbiornika DVB-T, FCD i podobnych jako drugiego odbiornika radiostacji amtorskich.
Odbiornik programowalny (SDR) można także podłączyć do wyjścia p.cz. radiostacji – o ile posiada takowe – uzyskując w ten sposób przystawkę panoramiczną o szerokości zakresu zależnej od pasma przepuszczania filtrów w radiostacji.

Rys. 2.3.9. Funkcje zdalnego sterowania Omni-Rig

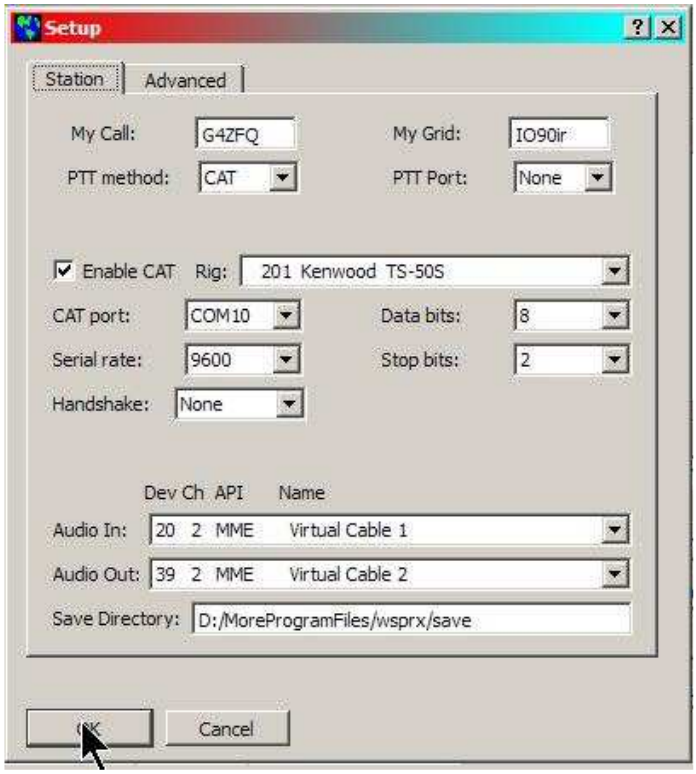


Rys. 2.3.10. Funkcje zdalnego sterowania CAT

W widocznym na ilustracji 2.3.9 menu możliwy jest wybór złącza szeregowego COM dla zdalnego sterowania, ustawienie parametrów komunikacji i wybór innych sposobów kluczowania nadajnika.

Odbiór emisji cyfrowych

Odbiór emisji cyfrowych takich jak PSK31, Olivia, RTTY, APRS, SSTV, faksymile itd. wymaga skorzystania z odpowiedniego programu terminalowego: MultiPSK, Fldigi, MixW itd. Zdemodulowane przez HDSDR sygnały są doprowadzane do programów terminalowych poprzez „Virtual Audio Cable” („VAC”) lub podobne programy. W celu połączenia HDSDR z programem typu „VAC” należy w menu systemu dźwiękowego [F5] wybrać jako kanał wyjściowy zamiast głośnika właśnie ten program.



Rys. 2.3.11. Przykład konfiguracji WSPRX do współpracy z HDSDR przez „VAC” i zdalnego sterowania sprzętem

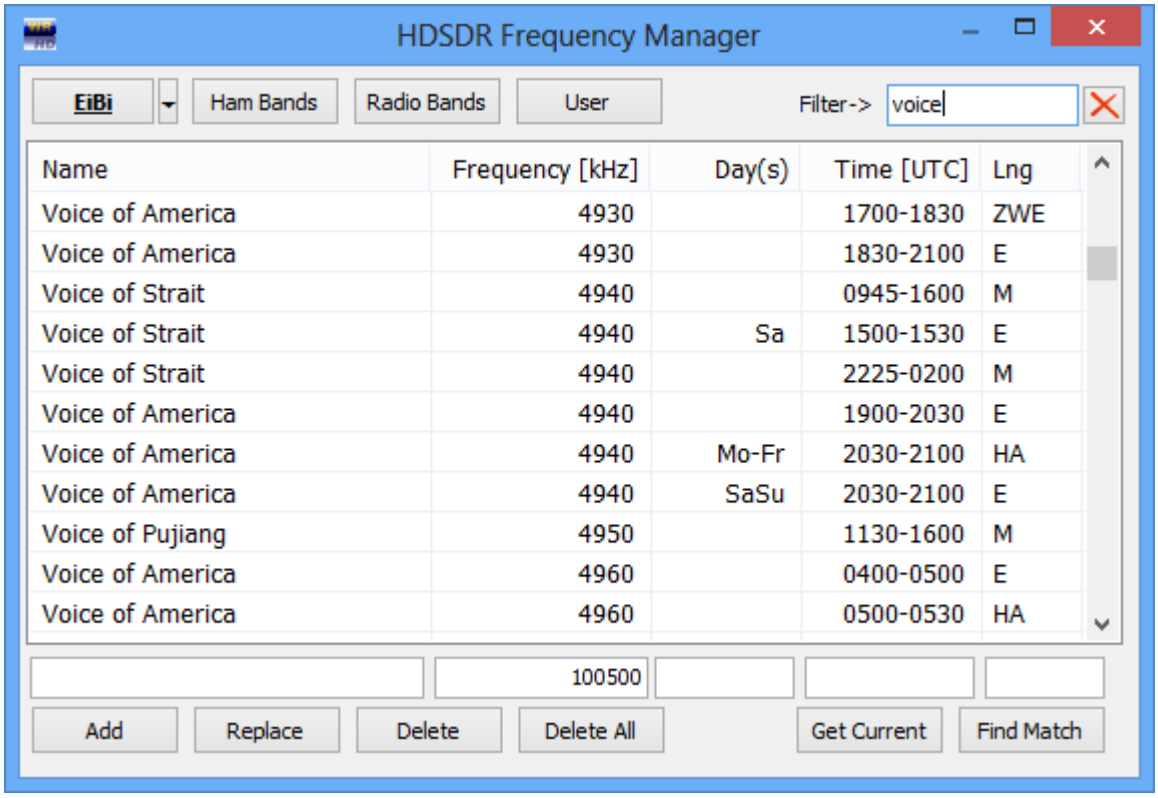
Wielokanałowy dekoder telegrafii „CW Skimmer” autorstwa VE3NEA pozwala natomiast na monitorowanie wycinków pasm telegraficznych i śledzenie odbieranych wywołań i łączności. Program wymaga doprowadzenia przez wirtualny kabel strumieni danych I i Q z HDSDR. Podanie sygnału m.cz. zamiast obydwu strumieni jest również możliwe ale ogranicza obserwowany zakres do 3 kHz. „CW Skimmer” może też dostrajać HDSDR do pożądanej częstotliwości za pomocą poleceń standardu „OmniRig”.

Transmisja

Transmisja jest możliwa pod warunkiem, że używana biblioteka ExtIO posiada funkcje nadawcze. Również dla nadawania możliwa jest zamiana kanałów IQ i optymalizacja tłumienia niepożądanej wstęgi.
Praca emisjami cyfrowymi wymaga połączenia HDSDR z programem terminalowym za pomocą wirtualnego kabla m.cz. („VAC”). Konieczny jest również drugi podsystem dźwiękowy.
W trybie zdalnego sterowania CAT do kluczowania nadajnika mogą być wykorzystane sygnały CTS (kontakt 8), RTS (4) lub DCD (1).

Rys. 2.3.12. Wybór kanałów używanych do nadawania

Baza danych częstotliwości



Rys. 2.3.13. Okno bazy danych stacji

Główne okno bazy danych stacji, otwierane za pomocą przycisku „FreqMgr”, zawiera ich spis wraz z częstotliwościami, dniami tygodnia i godzinami nadawania interesujacych użytkownika programów. W ostatniej kolumnie podawany jest język, w którym nadawany jest program. Przyciski i pola umieszczone powyżej tabeli służą do selekcji pożądanych rodzajów danych np. pasm radiofonicznych, amatorskich, transmisji głosowych itp. Przycisk „EiBi” służy do pobrania i zapisu gotowych rozkładów transmisji dostępnych w Internecie pod adresem *http://www.eibi.de.vu/*. Przyciski „Ham Bands” i „Radio Bands” służą do wyboru odpowiednio pasma amatorskich i radiofonicznych, a przycisk „User” do wyboru jednego z pięciu profili użytkowników.
Przycisk „Add” („Dodaj”) służy do dodania nowego wpisu, przy czym dane wpisywane są w linii pól znajdującej się powyżej przycisków.
Przycisk „Replace” („Zastąp”) umożliwia zastąpienie wpisu jego zmodyfikowaną wersją.
Przycisk „Delete” („Skasuj”) służy do skasowania wybranej linii w tabeli (wybranego wpisu) a przycisk „Delete All” („Skasuj wszystko”) – do całkowitego skasowania zawartości tabeli.

Kombinacje klawiszy

Aktualny dla danej wersji spis kombinacji klawiszy ułatwiających obsługę HDSDR zawiera plik *hdsdr_keyboard_shortcuts.htm* znajdującym się w katalogu programu.
Dostępne obecnie kombinacje zawiera tabela 2.4.1.

Tabela 2.4.1

Klawisz lub kombinacja	Znaczenie
C	Dostrojenie heterodyny odbiornika do środkowej częstotliwości wyświetlanego podzakresu
CTRL + O	Wpisanie nowej częstotliwości dostrojenia heterodyny
CTRL + T	Wpisanaie nowej częstotliwości środkowej
CTRL + Q	Wpisanie nowej podręcznej częstotliwości srodkowej (automatyczne strojenie heterodyny)
CTRL + V	Przełączanie heterodyny
CTRL + B	Zarządzanie pasmami
CTRL + „Strona w górę”	Przestrojenie w górę o szerokość wyświetlanego podzakresu
CTRL + „Strona wdół”	Przestrojenie w dół o szerokość wyświetlanego podzakresu
CTRL + „Home”	Wybór kroku strojenia za pomocą kółka myszy
CTRL + „Strzałka w górę”	Przestrojenie w górę częstotliwości środkowej o krok kółka
CTRL + „Strzałka w dół”	Przestrojenie częstotliwości środkowej w dół o krok kółka
CTRL + „Strzałka w prawo”	Przestrojenie górę odbiornika (heterodyny) o krok kółka
CTRL + „Strzałka w dół”	Przestrojenie w dół odbiornika o krok kółka
CTRL + A	Emisja AM
CTRL + C	Telegrafia (CW)
CTRL + D	Cyfrowa radiofonia DRM
CTRL + E	Włączenie detektora synchronicznego AM (ECSS)
CTRL + F	Emisja FM
CTRL + L	Emisja SSB z dolną wstęgą
CTRL + U	Emisja SSB z górną wstęgą
B	Włączenie eliminatora zakłóceń impulsowych („NB”)
R	Włączenie redukcji szumów („NR”)
G	Przełączanie stałej czasu ARW – szybkiej, średniej, wolnej lub wyłączenie ARW
M	Wyciszanie odbioru
+/-	Regulacja siły głosu
Z	Funkcja poszukiwania maksimów – ZAP
F	Włączenie lub wyłączenie automatycznego dostrojenia – ARCz (AFC)
P	Filtr podbijający dla telegrafii
D	Kompensacja rozpraszania sygnałów telegraficznych przy odbiorze EME
„Duże litery” + W	Przewijanie wstecz pliku dźwiękowego WAV
„Duże litery” + L	Odtwarzanie w pętli pliku dźwiękowego WAV
„Duże litery” + R	Nagrywanie w pliku dźwiękowym WAV
„Duże litery” + P	Odtwarzanie pliku dźwiękowego WAV
„Duże litery” + S	Zatrzymanie nagrywania lub odtwarzania
„Duże litery” + B	Pauza
H	Wyświetlenie powierzchni obsługi biblioteki *ExtIO_xxx.dll* – jeżeli to możliwe
F2	Włączenie lub wyłączenie odbioru przez HDSDR
F3	Zminimalizowanie okna HDSDR
F4	Zakończenie pracy HDSDR
F5	Otwarcie okna dialogowego wyboru podsystemów dźwiękowych

F6	Otwarcie okna wyboru częstotliwości próbkowania
F7	Otwarcie okna konfiguracji „Options”
F8	Start Win
F9	Informacje o programie i aktualizacja
Klawisz „Pauza”	Ukrycie programu przed niepożądanymi oczami

Dodatek A
Przykładowa biblioteka ExtIO.DLL

Zawarte w dodatku przykłady są przeznaczone dla programistów mających doświadczenie w językach C i C++. Mają one ułatwić zaprogramowanie biblioteki sterującej dla nowych konstrukcji sprzętu. Pliki przytoczono z witryn podanych w zawartych w nich komentarzach bez dokonywania żadnych zmian.

Plik nagłówkowy LC_ExtIO_Types.h

```
//------------------------------------------------------------------------

#ifndef LC_ExtIO_TypesH
#define LC_ExtIO_TypesH

// specification from http://www.sdradio.eu/weaksignals/bin/Winrad_Extio.pdf
// linked referenced from http://www.weaksignals.com/

// for C99 compiler just #include <stdint.h>
// MS VC++ 2008 Express does not have stdint.h Try
http://msinttypes.googlecode.com/svn/trunk/stdint.h
#include "stdint.h"
// for other compilers you may try http://www.azillionmonkeys.com/qed/pstdint.h
// or try boost: http://www.boost.org/doc/libs/1_36_0/boost/cstdint.hpp

/*
 * I. INITIALIZATION + OPEN SEQUENCE
 * ================================
 * ExtIoSetSetting()
 *   optional: previously saved settings are delivered to ExtIO
 *   call once with idx -1 to sign ExtIO that this functionality is supported,
 *   so that ExtIO may inhibit loading/storing any .ini
 * InitHW()
 *   mandatory: initialize ExtIO. May do nothing!
 * VersionInfo()
 *   optional: delivers SDR program name and version to ExtIO. Allows to check if some necessary
 *     functional extension is (not) supported by SDR program
 * GetAttenuators()
 *   optional: show ExtIO, that SDR program supports controlling RF Gain/Attenuator(s)
 *     you should prefer this over determining SDR program's capabilities over VersionInfo()
 * ExtIoGetMGCs()
 *   optional: show ExtIO, that SDR program supports controlling IF Gain/Attenuator(s)
 *     you should prefer this over determining SDR program's capabilities over VersionInfo()
 * [ ActivateTx() ]
 *   optional: try's to activate Tx functionality
 * SetCallback()
 *   mandatory: callback function pointer is given to ExtIO. ExtIO may inform SDR program of events
 *     using this callback interface and the extHWstatusT enums
 * OpenHW()
 *   mandatory: prepare ExtIO for start .. or fail for any reason!
 *
 *
 * II. START SEQUENCE
 * ==================
 * StartHW()
 *   mandatory: start processing
```

```
*
*
* III. WORK
* =========
* SetHWLO() and many other functions ...
*
*
* IV. STOP SEQUENCE (== 'undo' of Start sequence)
* ================
* StopHW()
*   mandatory: start processing
*
*
* V. CLOSE SEQUENCE (== 'undo' of init + stop sequence)
* ================
* ExtIoGetSetting()
*   optional: get and save settings for next time.
*    call ExtIoGetSetting() before CloseHW() to get correct settings.
*    do not call without súccessful OpenHW()
* CloseHW()
*   mandatory: close hardware. Processing is not started again (with StartHW),
*   unless OpenHW() is called again
*   this function is called only when prior OpenHW() was successful
*   take care not to free already freed or never allocated resources
*
*/

// function implemented by Winrad / HDSDR; see enum extHWstatusT below
typedef int             (* pfnExtIOCallback)  (int cnt, int status, float IQoffs, void *IQdata);

// mandatory functions, which have to be implemented by ExtIO DLL
#define EXTIO_MAX_NAME_LEN    16    /* name is displayed in Winrad/HDSDR's menu */
#define EXTIO_MAX_MODEL_LEN   16    /* model is not used */
typedef bool    (__stdcall * pfnInitHW)        (char *name, char *model, int& hwtype); // for hwtype see
enum extHWtypeT
typedef bool    (__stdcall * pfnOpenHW)        (void);
typedef void    (__stdcall * pfnCloseHW)       (void);
typedef int     (__stdcall * pfnStartHW)       (long extLOfreq);
typedef void    (__stdcall * pfnStopHW)        (void);
typedef void    (__stdcall * pfnSetCallback)   (pfnExtIOCallback funcptr);
typedef int     (__stdcall * pfnSetHWLO)       (long extLOfreq);   // see also SetHWLO64
typedef int     (__stdcall * pfnGetStatus)     (void);

// optional functions, which can be implemented by ExtIO DLL
// for performance reasons prefer not implementing rather then implementing empty functions
//   especially for RawDataReady
typedef long    (__stdcall * pfnGetHWLO)       (void);            // see also GetHWLO64
typedef long    (__stdcall * pfnGetHWSR)       (void);
typedef void    (__stdcall * pfnRawDataReady)   (long samprate, void *Ldata, void *Rdata, int
numsamples);
typedef void    (__stdcall * pfnShowGUI)       (void);
typedef void    (__stdcall * pfnHideGUI)       (void);
typedef void    (__stdcall * pfnSwitchGUI)     (void);            // new: switch visibility of GUI
typedef void    (__stdcall * pfnTuneChanged)   (long tunefreq);   // see also TuneChanged64
typedef long    (__stdcall * pfnGetTune)       (void);            // see also GetTune64
```

```
typedef void   (__stdcall * pfnModeChanged)   (char mode);
typedef char   (__stdcall * pfnGetMode)       (void);
typedef void   (__stdcall * pfnIFLimitsChanged)(long lowfreq, long highfreq);  // see also
IFLimitsChanged64
typedef void   (__stdcall * pfnFiltersChanged) (int loCut, int hiCut, int pitch);  // lo/hiCut relative to
tuneFreq
typedef void   (__stdcall * pfnMuteChanged)   (bool muted);
typedef void   (__stdcall * pfnGetFilters)    (int& loCut, int& hiCut, int& pitch);

// optional functions - extended for receivers with frequency range over 2147 MHz - used from HDSDR
// these functions 64 bit functions are prefered rather than using the 32 bit ones
// for other Winrad derivations you should additionally implement the above "usual" 32 bit functions
typedef int    (__stdcall * pfnStartHW64)     (int64_t extLOfreq);   // "StartHW64" with HDSDR >=
2.14
typedef int64_t (__stdcall * pfnSetHWLO64)     (int64_t extLOfreq);
typedef int64_t (__stdcall * pfnGetHWLO64)     (void);
typedef void   (__stdcall * pfnTuneChanged64)  (int64_t tunefreq);
typedef int64_t (__stdcall * pfnGetTune64)     (void);
typedef void   (__stdcall * pfnIFLimitsChanged64)  (int64_t lowfreq, int64_t highfreq);

// optional functions, which can be implemented by ExtIO DLL
// following functions may get called from HDSDR 2.13 and above

// "VersionInfo" is called - when existing - after successful InitHW()
// with this information an ExtIO may check which extHWstatusT enums are properly processed from
application
typedef void   (__stdcall * pfnVersionInfo)   (const char * progname, int ver_major, int ver_minor);

// "GetAttenuators" allows HDSDR to display a knob or slider for Attenuation / Amplification
// see & use extHw_Changed_ATT enum if ATT can get changed by ExtIO dialog window or from
hardware
#define EXTIO_MAX_ATT_GAIN_VALUES 128
typedef int    (__stdcall * pfnGetAttenuators) (int idx, float * attenuation);  // fill in attenuation
                    // use positive attenuation levels if signal is amplified (LNA)
                    // use negative attenuation levels if signal is attenuated
                    // sort by attenuation: use idx 0 for highest attenuation / most damping
                    // this functions is called with incrementing idx
                    //   - until this functions returns != 0, which means that all attens are already
delivered
typedef int    (__stdcall * pfnGetActualAttIdx)(void);                  // returns -1 on error
typedef int    (__stdcall * pfnSetAttenuator)  (int idx);               // returns != 0 on error

// "SetModeRxTx" will only get called after a successful activation with ActivateTx()
// see extHw_TX_Request/extHw_RX_Request enums below if modehange can get triggered from user
/ hardware
typedef int    (__stdcall * pfnSetModeRxTx)   (int modeRxTx);       // see enum extHw_ModeRxTxT

// generation and transmission of I/Q samples for TX mode requires activation of the ExtIO DLL
// contact LC at hdsdr.de  to ask for activation keys and algorithm
typedef int    (__stdcall * pfnActivateTx)    (int magicA, int magicB);

// (de)activate all bandpass filter to allow "bandpass undersampling" (with external analog bandpass
filter)
// intended for future use: it may get set automatically depending on LO frequency and the "ExtIO
Frequency Options"
```

```
//   deactivation of bp/lp-filters when real LO (in HDSDR) is > ADC_Samplerate/2 in undersampling
mode
typedef int     (__stdcall * pfnDeactivateBP)   (int deactivate);
                    // deactivate == 1 to deactivate all bandpass and lowpass filters of hardware
                    // deactivate == 0 to reactivate automatic bandpass selection depending on frequency

// optional "ExtIoGetSrates" is for replacing the Soundcard Samplerate values in the Samplerate
selection dialog
//   by these values supported from the SDR hardware.
// see & use extHw_Changed_SampleRate enum ... and "GetHWSR". Enumeration API as with
"GetAttenuators"
// intended for future use - actually not implemented/called
#define EXTIO_MAX_SRATE_VALUES  32
typedef int     (__stdcall * pfnExtIoGetSrates) (int idx, double * samplerate);  // fill in possible
samplerates
                    // this functions is called with incrementing idx
                    //    - until this functions returns != 0, which means that all srates are already delivered
typedef int     (__stdcall * pfnExtIoGetActualSrateIdx) (void);             // returns -1 on error
typedef int     (__stdcall * pfnExtIoSetSrate)  (int idx);                  // returns != 0 on error

// optional function to get 3dB bandwidth from samplerate
typedef long    (__stdcall * pfnExtIoGetBandwidth)  (int srate_idx);       // returns <= 0 on error

// optional function to get center (= IF frequency) of 3dB band in Hz - for non I/Q receivers with 0
center
typedef long    (__stdcall * pfnExtIoGetBwCenter)   (int srate_idx);       // returns 0 on error, which is
default

// optional function to get AGC Mode: AGC_OFF (always agc_index = 0), AGC_SLOW,
AGC_MEDIUM, AGC_FAST, ...
// this functions is called with incrementing idx
//    - until this functions returns != 0, which means that all agc modes are already delivered
#define EXTIO_MAX_AGC_VALUES  16
typedef int     (__stdcall * pfnExtIoGetAGCs) (int agc_idx, char * text);  // text limited to max 16 char
typedef int     (__stdcall * pfnExtIoGetActualAGCidx)(void);             // returns -1 on error
typedef int     (__stdcall * pfnExtIoSetAGC) (int agc_idx);              // returns != 0 on error
// optional: HDSDR >= 2.62
typedef int     (__stdcall * pfnExtIoShowMGC)(int agc_idx);              // return 1, to continue showing
MGC slider on AGC
                                                // return 0, is default for not showing MGC slider

// for AGC in AGC_OFF (agc_idx == 0), which is (M)anual (G)ain (C)ontrol
// sometimes referred as "IFgain" - as in SDR-14/IP
#define EXTIO_MAX_MGC_VALUES  128
typedef int     (__stdcall * pfnExtIoGetMGCs)(int mgc_idx, float * gain);  // fill in gain
                    // sort by ascending gain: use idx 0 for lowest gain
                    // this functions is called with incrementing idx
                    //    - until this functions returns != 0, which means that all gains are already delivered
typedef int     (__stdcall * pfnExtIoGetActualMgcIdx) (void);             // returns -1 on error
typedef int     (__stdcall * pfnExtIoSetMGC) (int mgc_idx);              // returns != 0 on error

// not used in HDSDR - for now
// optional function to get 3dB band of Preselectors
// this functions is called with incrementing idx
//    - until this functions returns != 0, which means that all preselectors are already delivered
```

```
// ExtIoSetPresel() with idx = -1 to activate automatic preselector selection
// ExtIoSetPresel() with valid idx (>=0) deactivates automatic preselection
typedef int     (__stdcall * pfnExtIoGetPresels)        ( int idx, int64_t * freq_low, int64_t * freq_high );
typedef int     (__stdcall * pfnExtIoGetActualPreselIdx) ( void );      // returns -1 on error
typedef int     (__stdcall * pfnExtIoSetPresel)         ( int idx );   // returns != 0 on error

// not used in HDSDR - for now
// optional function to get frequency ranges usable with SetHWLO(),
//   f.e. the FUNcube Dongle Pro+ should deliver idx 0: low=0.15 high=250 MHz and idx 1: low=420
high=1900 MHz
//        with a gap from 250MHz to 420 MHz. see http://www.funcubedongle.com/?page_id=1073
//   if extIO is told to set a not-supported frequency with SetHWLO(), then the extIO should callback
with extHw_Changed_LO
//     and set a new frequency, which is supported
// this functions is called with incrementing idx
//    - until this functions returns != 0, which means that all frequency ranges are already delivered
typedef int     (__stdcall * pfnExtIoGetFreqRanges)        ( int idx, int64_t * freq_low, int64_t *
freq_high );

// not used in HDSDR - for now
// optional function to get full samplerate of A/D Converter
//   useful to know with direct samplers in bandpass undersampling mode
//   example: Perseus = 80 000 000 ; SDR-14 = 66 666 667
//   return <= 0 if undersampling not supported (when preselectors not deactivatable)
typedef double  (__stdcall * pfnExtIoGetAdcSrate) ( void );

// HDSDR >= 2.51
// optional functions to receive and set all special receiver settings (for save/restore in application)
//   allows application and profile specific settings.
//   easy to handle without problems with newer Windows versions saving a .ini file below programs as
non-admin-user
// Settings shall be zero-terminated C-Strings.
// example settings: USB-Identifier(for opening specific device), IP/Port, AGC, Srate, ..
// idx in 0 .. 999 => NOT more than 1000 values storable!
// description max 1024 char
// value max 1024 char
// these functions are called with incrementing idx: 0, 1, ...
// until ExtIoGetSetting() returns != 0, which means that all settings are already delivered
typedef int     (__stdcall * pfnExtIoGetSetting) ( int idx, char * description, char * value ); // will be
called (at least) before exiting application
typedef void    (__stdcall * pfnExtIoSetSetting) ( int idx, const char * value ); // before calling InitHW()
!!!
  // there will be an extra call with idx = -1, if theses functions are supported by the SDR app
  // suggestion: use index 0 as ExtIO identifier (save/check ExtIO name) to allow fast skipping of all
following SetSetting calls
  //   when this identifier does not match

// not used in HDSDR - for now
// handling of VFOs - see also extHw_Changed_VFO
// VFOindex is in 0 .. numVFO-1
typedef void    (__stdcall * pfnExtIoVFOchanged) ( int VFOindex, int numVFO, int64_t extLOfreq,
int64_t tunefreq, char mode );
typedef int     (__stdcall * pfnExtIoGetVFOindex)( void );   // returns new VFOindex

// hwtype codes to be set with pfnInitHW
```

```
// Please ask Alberto di Bene (i2phd@weaksignals.com) for the assignment of an index code
// for cases different from the above.
// note: "exthwUSBdataNN" don't need to be from USB. The keyword "USB" is just for historical
reasons,
//   which may get removed later ..
typedef enum
{
   exthwNone       = 0
  , exthwSDR14      = 1
  , exthwSDRX       = 2
  , exthwUSBdata16  = 3 // the hardware does its own digitization and the audio data are returned to
Winrad
                  // via the callback device. Data must be in 16-bit  (short) format, little endian.
  , exthwSCdata     = 4 // The audio data are returned via the (S)ound (C)ard managed by Winrad. The
external
                  // hardware just controls the LO, and possibly a preselector, under DLL control.
  , exthwUSBdata24  = 5 // the hardware does its own digitization and the audio data are returned to
Winrad
                  // via the callback device. Data are in 24-bit  integer format, little endian.
                  //   each sample is 32 bit with values from -2^-23 to +2^23 -1
  , exthwUSBdata32  = 6 // the hardware does its own digitization and the audio data are returned to
Winrad
                  // via the callback device. Data are in 32-bit  integer format, little endian.
  , exthwUSBfloat32 = 7 // the hardware does its own digitization and the audio data are returned to
Winrad
                  // via the callback device. Data are in 32-bit  float format, little endian.
  , exthwHPSDR      = 8 // for HPSDR only!

#if 1
  // for future use - actually not supported! it's intended for RTL2832U based DVB-T USB sticks
  , exthwUSBdata8   = 9 // the hardware does its own digitization and the audio data are returned to
Winrad
                  // via the callback device. Data must be in 8-bit  (unsigned) format, little endian.
#endif
} extHWtypeT;

// status codes for pfnExtIOCallback; used when cnt < 0
typedef enum
{
  // only processed/understood for SDR14
    extHw_Disconnected        = 0     // SDR-14/IQ not connected or powered off
  , extHw_READY               = 1     // IDLE / Ready
  , extHw_RUNNING             = 2     // RUNNING  => not disconnected
  , extHw_ERROR               = 3     // ??
  , extHw_OVERLOAD            = 4     // OVERLOAD => not disconnected

  // for all extIO's
  , extHw_Changed_SampleRate  = 100   // sampling speed has changed in the external HW
  , extHw_Changed_LO          = 101   // LO frequency has changed in the external HW
  , extHw_Lock_LO             = 102
  , extHw_Unlock_LO           = 103
  , extHw_Changed_LO_Not_TUNE = 104   // CURRENTLY NOT YET IMPLEMENTED
                              // LO freq. has changed, Winrad must keep the Tune freq. unchanged
                              // (must immediately call GetHWLO() )
  , extHw_Changed_TUNE        = 105   // a change of the Tune freq. is being requested.
```

```
                        // Winrad must call GetTune() to know which value is wanted
  , extHw_Changed_MODE      = 106   // a change of demod. mode is being requested.
                        // Winrad must call GetMode() to know the new mode
  , extHw_Start             = 107   // The DLL wants Winrad to Start
  , extHw_Stop              = 108   // The DLL wants Winrad to Stop
  , extHw_Changed_FILTER    = 109   // a change in the band limits is being requested
                        // Winrad must call GetFilters()

  // Above status codes are processed with Winrad 1.32.
  //   All Winrad derivation like WRplus, WinradF, WinradHD and HDSDR should understand them,
  //   but these do not provide version info with VersionInfo(progname, ver_major, ver_minor).

  , extHw_Mercury_DAC_ON    = 110   // enable audio output on the Mercury DAC when using the
HPSDR
  , extHw_Mercury_DAC_OFF   = 111   // disable audio output on the Mercury DAC when using the
HPSDR
  , extHw_PC_Audio_ON       = 112   // enable audio output on the PC sound card when using the
HPSDR
  , extHw_PC_Audio_OFF      = 113   // disable audio output on the PC sound card when using the
HPSDR

  , extHw_Audio_MUTE_ON     = 114   // the DLL is asking Winrad to mute the audio output
  , extHw_Audio_MUTE_OFF    = 115   // the DLL is asking Winrad to unmute the audio output

  // Above status codes are processed with Winrad 1.33 and HDSDR
  //   Winrad 1.33 and HDSDR still do not provide their version with VersionInfo()

  // Following status codes are processed when VersionInfo delivers
  //  0 == strcmp(progname, "HDSDR") && ( ver_major > 2 || ( ver_major == 2 && ver_minor >= 13 )
)

  // all extHw_XX_SwapIQ_YYY callbacks shall be reported after each OpenHW() call
  , extHw_RX_SwapIQ_ON      = 116   // additionaly swap IQ - this does not modify the menu point /
user selection
  , extHw_RX_SwapIQ_OFF     = 117   //   the user selected swapIQ is additionally applied
  , extHw_TX_SwapIQ_ON      = 118   // additionaly swap IQ - this does not modify the menu point /
user selection
  , extHw_TX_SwapIQ_OFF     = 119   //   the user selected swapIQ is additionally applied

  // Following status codes (for I/Q transceivers) are processed when VersionInfo delivers
  //  0 == strcmp(progname, "HDSDR") && ( ver_major > 2 || ( ver_major == 2 && ver_minor >= 13 )
)

  , extHw_TX_Request        = 120   // DLL requests TX mode / User pressed PTT
                        //   exciter/transmitter must wait until SetModeRxTx() is called!
  , extHw_RX_Request        = 121   // DLL wants to leave TX mode / User released PTT
                        //   exciter/transmitter must wait until SetModeRxTx() is called!
  , extHw_CW_Pressed        = 122   // User pressed  CW key
  , extHw_CW_Released       = 123   // User released CW key
  , extHw_PTT_as_CWkey      = 124   // handle extHw_TX_Request as extHw_CW_Pressed in CW
mode
                        //  and   extHw_RX_Request as extHw_CW_Released
  , extHw_Changed_ATT       = 125   // Attenuator changed => call GetActualAttIdx()
```

```
  // Following status codes are processed when VersionInfo delivers
  //  0 == strcmp(progname, "HDSDR") && ( ver_major > 2 || ( ver_major == 2 && ver_minor >= 14 )
)

#if 0
  // Following status codes are for future use - actually not implemented !
  //  0 == strcmp(progname, "HDSDR") && ( ver_major > 2 || ( ver_major == 2 && ver_minor >>> 14
) )

  // following status codes to change sampleformat at runtime
  , extHw_SampleFmt_IQ_UINT8  = 126   // change sample format to unsigned 8 bit INT (Realtek
RTL2832U)
  , extHw_SampleFmt_IQ_INT16  = 127   //          -"-          signed 16 bit INT
  , extHw_SampleFmt_IQ_INT24  = 128   //          -"-          signed 24 bit INT
  , extHw_SampleFmt_IQ_INT32  = 129   //          -"-          signed 32 bit INT
  , extHw_SampleFmt_IQ_FLT32  = 130   //          -"-          signed 16 bit FLOAT
#endif
  // following status codes to change channel mode at runtime
  , extHw_RX_ChanMode_LEFT    = 131   // left channel only
  , extHw_RX_ChanMode_RIGHT   = 132   // right channel only
  , extHw_RX_ChanMode_SUM_LR  = 133   // sum of left + right channel
  , extHw_RX_ChanMode_I_Q     = 134   // I/Q with left  channel = Inphase and right channel =
Quadrature
                              // last option set I/Q and clear internal swap as with extHw_RX_SwapIQ_OFF
  , extHw_RX_ChanMode_Q_I     = 135   // I/Q with right channel = Inphase and left  channel =
Quadrature
                              // last option set I/Q and internal swap as with extHw_RX_SwapIQ_ON

  , extHw_Changed_RF_IF       = 136   // refresh selectable attenuators and Gains
                               // => starts calling GetAttenuators(), GetAGCs() & GetMGCs()
  , extHw_Changed_SRATES      = 137   // refresh selectable samplerates => starts calling
GetSamplerates()

  // Following status codes are for 3rd Party Software, currently not implemented in HDSDR
  , extHw_Changed_PRESEL      = 138  // Preselector changed => call ExtIoGetActualPreselIdx()
  , extHw_Changed_PRESELS     = 139  // refresh selectable preselectors => start calling
ExtIoGetPresels()
  , extHw_Changed_AGC         = 140  // AGC changed => call ExtIoGetActualAGCidx()
  , extHw_Changed_AGCS        = 141  // refresh selectable AGCs => start calling ExtIoGetAGCs()
  , extHw_Changed_SETTINGS    = 142  // settings changed, call ExtIoGetSetting()
  , extHw_Changed_FREQRANGES  = 143  // refresh selectable frequency ranges, call
ExtIoGetFreqRanges()

  , extHw_Changed_VFO         = 144  // refresh selectable VFO => starts calling ExtIoGetVFOindex()

  // Following status codes are processed when VersionInfo delivers
  //  0 == strcmp(progname, "HDSDR") && ( ver_major > 2 || ( ver_major == 2 && ver_minor >= 60 )
)
  , extHw_Changed_MGC         = 145  // MGC changed => call ExtIoGetMGC()

} extHWstatusT;

// codes for pfnSetModeRxTx:
typedef enum
{
```

```
   extHw_modeRX  = 0
  , extHw_modeTX  = 1
} extHw_ModeRxTxT;

// macro to call callback function with just status extHWstatusT
#define EXTIO_STATUS_CHANGE( CB, STATUS )   CB( -1, STATUS, 0, NULL )

#endif /* LC_ExtIO_TypesH */
```

Plik ExtIO_Demo.ccp

```
#define EXTIO_EXPORTS       1
#define HWNAME              "Demo-1.00"
#define HWMODEL             "Demo ExtIO"
#define SETTINGS_IDENTIFIER "Demo-1.x"
#define NUM_CARRIER         2
#define LO_MIN              100000LL
#define LO_MAX              7500000000LL
#define LO_PRECISION    5000L
#define EXT_BLOCKLEN    512       /* only multiples of 512 */
#define BORLAND             0

#include "ExtIO_Demo.h"

//---------------------------------------------------------------------
// #define WIN32_LEAN_AND_MEAN             // Selten verwendete Teile
der Windows-Header nicht einbinden.
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

//---------------------------------------------------------------------

#pragma warning(disable : 4996)

#define snprintf _snprintf

static bool SDR_supports_settings = false;  // assume not supported
static bool SDR_settings_valid = false;      // assume settings are
for some other ExtIO

static char SDR_progname[32+1] = "\0";
static int  SDR_ver_major = -1;
static int  SDR_ver_minor = -1;

static unsigned  gCustomSamplerate = 48000;
static int    gHwType = exthwUSBfloat32;

static int    giExtSrateIdx = 0;
static unsigned gExtSampleRate = 48000;
static double gaCarrierFreq[NUM_CARRIER] = {5000.0, 10000.0}; // Hz
static double gaCarrierLevel[NUM_CARRIER] = {-40.0, -20.0};   // dB
volatile double  gaCarrierAmp[NUM_CARRIER];
volatile double  gaCarrierPhaseInc[NUM_CARRIER];
```

```
volatile int64_t glLOfreq = 0L;
bool   gbInitHW = false;
int    giAttIdx = 0;
int    giDefaultAttIdx = 4; // 0 dB
int    giMgcIdx = 0;
int    giDefaultMgcIdx = 0; // 0 dB
int    giAgcIdx = 0;
int    giDefaultAgcIdx = 1; // Auto
int    giThrIdx = 0;
int    giDefaultThrIdx = 2; // Threshold: 20 dB
int    giWhatIdx = 0;

pfnExtIOCallback pfnCallback = 0;
volatile bool gbExitThread = false;
volatile bool gbThreadRunning = false;
volatile int  giParameterUsed = 0;
volatile int  giParameterSetNo = 0;

//--------------------------------------------------------------------

static void setAmpVal()
{
   float attVal;
   float gainVal;
   GetAttenuators( giAttIdx, &attVal );

   //ExtIoGetMGCs(giMgcIdx, &gainVal);
   // do not use ExtIoGetMGCs() - in case agc or thr is on!
   switch (giMgcIdx)
   {
   case 0:       gainVal = 0.0F;  break;
   case 1:       gainVal = 3.0F;  break;
   case 2:       gainVal = 6.0F;  break;
   }

   for ( int k = 0; k < NUM_CARRIER; ++k )
   {
      if (   gaCarrierLevel[k] + attVal + gainVal < -400.0
          || gaCarrierFreq[k] <= -0.5 * gExtSampleRate
          || gaCarrierFreq[k] >=  0.5 * gExtSampleRate
          )
      {
          gaCarrierAmp[k] = 0.0;
          gaCarrierPhaseInc[k] = 0.0;
      }
      else
      {
          gaCarrierAmp[k] = pow( 10.0, ( gaCarrierLevel[k] + attVal +
gainVal ) / 20.0 );
          // f = dphi / 2*PI * fs  <=>  dphi = 2*PI*(f / fs)
          gaCarrierPhaseInc[k] = 2.0 *
3.14159265358979323846264338327 95 * gaCarrierFreq[k] /
(double)gExtSampleRate;
      }
   }
}
```

```
//---------------------------------------------------------------------

DWORD WINAPI GeneratorThreadProc( __in  LPVOID lpParameter )
{
    double aLocalCarrierAmp[NUM_CARRIER];
    double aLocalCarrierPhaseInc[NUM_CARRIER];
    double aLocalCarrierPhase[NUM_CARRIER];
    unsigned LocalSampleRate;

    DWORD lTicks = GetTickCount();
    DWORD lTicksNew = 0;
    DWORD lTicksPPM = lTicks;
    unsigned long generatorCount = 0;

    {
        for ( int k = 0; k < NUM_CARRIER; ++k )
            aLocalCarrierPhase[k] = 0.0;

        LocalSampleRate = gExtSampleRate;
    }

    while ( !gbExitThread )
    {
        SleepEx( 1, FALSE );
        if ( gbExitThread )
            break;

        // new parametrization?
        if ( giParameterUsed != giParameterSetNo )
        {
            ++giParameterUsed;
            setAmpVal();
            for ( int k = 0; k < NUM_CARRIER; ++k )
            {
                aLocalCarrierAmp[k] = gaCarrierAmp[k];
                aLocalCarrierPhaseInc[k] = gaCarrierPhaseInc[k];
            }
            LocalSampleRate = gExtSampleRate;
        }

        lTicksNew = GetTickCount();
        unsigned long elapsedMs = (unsigned long)(lTicksNew - lTicks);
        generatorCount += (LocalSampleRate * elapsedMs + 500L) /
1000L;

        while ( generatorCount >= EXT_BLOCKLEN && !gbExitThread )
        {
            float samplesFlt[ EXT_BLOCKLEN * 2 ];
            memset(&samplesFlt[0], 0, EXT_BLOCKLEN * 2 *
sizeof(float));

            for ( int k = 0; k < NUM_CARRIER; ++k )
            {
                if ( aLocalCarrierAmp[k] <= 0.0 )
                    continue;
```

```
            const double ampFactor = aLocalCarrierAmp[k];
            const double phaseInc  = aLocalCarrierPhaseInc[k];
            double phase = aLocalCarrierPhase[k];

            unsigned i, j = 0;
            for ( i = j = 0; i < EXT_BLOCKLEN; ++i )
            {
                samplesFlt[ j++ ] += (float)(ampFactor *
cos(phase));
                samplesFlt[ j++ ] += (float)(ampFactor *
sin(phase));

                phase += phaseInc;
                if      ( phase >  3.14159265358979323846264338327795
)   phase -= 2.0 * 3.14159265358979323846264338327795;
                else if ( phase < -3.14159265358979323846264338327795
)   phase += 2.0 * 3.14159265358979323846264338327795;
            }

            aLocalCarrierPhase[k] = phase;
        }

        generatorCount -= EXT_BLOCKLEN;

        if (pfnCallback && !gbExitThread)
        {
            if (gHwType == exthwUSBfloat32)
                pfnCallback(EXT_BLOCKLEN, 0, 0.0F, &samplesFlt[0]);
            else
            {
                int   samplePCM[EXT_BLOCKLEN * 2];
                if (gHwType == exthwFullPCM32)
                {
                    for (int k = 0; k < 2 * EXT_BLOCKLEN; ++k)
                        samplePCM[k] = (int)(samplesFlt[k] *
2147483648.0F);
                }
                else if (gHwType == exthwUSBdata32)
                {
                    for (int k = 0; k < 2 * EXT_BLOCKLEN; ++k)
                        samplePCM[k] = (int)( samplesFlt[k] *
8388608.0F );
                }
                else if (gHwType == exthwUSBdata24)
                {
                    // error!
                    for (int k = 0; k < 2 * EXT_BLOCKLEN; ++k)
                        samplePCM[k] = (int)( samplesFlt[k] *
8388608.0F );
                }
                else if (gHwType == exthwUSBdataS8)
                {
                    signed char * samplesS8 = (signed char
*)((void*)(&samplePCM[0]));
                    for (int k = 0; k < 2 * EXT_BLOCKLEN; ++k)
```

```
                    {
                        int pcm = (int)(samplesFlt[k] * 128.0F);
                        samplesS8[k] = (pcm < -128) ? -128 : ( (pcm >
127) ? 127 : pcm );
                    }
                }
                else if (gHwType == exthwUSBdataU8)
                {
                    unsigned char * samplesU8 = (unsigned char
*)((void*)(&samplePCM[0]));
                    for (int k = 0; k < 2 * EXT_BLOCKLEN; ++k)
                    {
                        int pcm = (int)(samplesFlt[k] * 128.0F) +
128;
                        samplesU8[k] = (pcm < 0) ? 0 : ((pcm > 255) ?
255 : pcm);
                    }
                }
                pfnCallback(EXT_BLOCKLEN, 0, 0.0F, &samplePCM[0]);
            }
        }
    }

    lTicks = lTicksNew;
  }
  gbExitThread = false;
  gbThreadRunning = false;
  return 0;
}

static void stopThread()
{
  if ( gbThreadRunning )
  {
    gbExitThread = true;
    while ( gbThreadRunning )
    {
      SleepEx( 10, FALSE );
    }
  }
}

static void startThread()
{
  gbExitThread = false;
  gbThreadRunning = true;
  ++giParameterSetNo;

  CreateThread( NULL   // LPSECURITY_ATTRIBUTES lpThreadAttributes
    , (SIZE_T)(64 * 1024)   // SIZE_T dwStackSize
    , GeneratorThreadProc   // LPTHREAD_START_ROUTINE
lpStartAddress
    , NULL              // LPVOID lpParameter
    , 0                 // DWORD dwCreationFlags
    , NULL              // LPDWORD lpThreadId
    );
```

```
}

//---------------------------------------------------------------------

#if BORLAND
//---------------------------------------------------------------------
// Important note about DLL memory management when your DLL uses the
// static version of the RunTime Library:
//
// If your DLL exports any functions that pass String objects (or
structs/
// classes containing nested Strings) as parameter or function
results,
// you will need to add the library MEMMGR.LIB to both the DLL
project and
// any other projects that use the DLL. You will also need to use
MEMMGR.LIB
// if any other projects which use the DLL will be performing new or
delete
// operations on any non-TObject-derived classes which are exported
from the
// DLL. Adding MEMMGR.LIB to your project will change the DLL and its
calling
// EXE's to use the BORLNDMM.DLL as their memory manager. In these
cases,
// the file BORLNDMM.DLL should be deployed along with your DLL.
//
// To avoid using BORLNDMM.DLL, pass string information using "char
*" or
// ShortString parameters.
//
// If your DLL uses the dynamic version of the RTL, you do not need
to
// explicitly add MEMMGR.LIB as this will be done implicitly for you
//---------------------------------------------------------------------

#pragma argsused
BOOL WINAPI DllMain(HINSTANCE hModule, DWORD ul_reason_for_call,
LPVOID lpReserved)

#else
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
#endif
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
```

```
}

//---------------------------------------------------------------------
extern "C" bool __declspec(dllexport) __stdcall InitHW(char *name,
char *model, int& type)
{
    type = gHwType;
    strcpy(name,  HWNAME);
    strcpy(model, HWMODEL);

    if ( !gbInitHW )
    {
        // do initialization

        glLOfreq = 6075000L; // just a default value
        // .......... init here the hardware controlled by the DLL
        // ......... init here the DLL graphical interface, if any

        giAttIdx = giDefaultAttIdx;
        giMgcIdx = giDefaultMgcIdx;
        giAgcIdx = giDefaultAgcIdx;
        giThrIdx = giDefaultThrIdx;

        setAmpVal();

        gbInitHW = true;
    }
    return gbInitHW;
}

//---------------------------------------------------------------------
extern "C" bool EXTIO_API OpenHW(void)
{
    // .... display here the DLL panel ,if any....
    // .....if no graphical interface, delete the following statement
    //::SetWindowPos(F->handle, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE |
SWP_NOSIZE);

    if ( pfnCallback )
        pfnCallback( -1, extHw_Changed_ATT, 0.0F, 0 );

    // in the above statement, F->handle is the window handle of the
panel displayed
    // by the DLL, if such a panel exists
    return gbInitHW;
}

//---------------------------------------------------------------------
extern "C" int  EXTIO_API StartHW(long LOfreq)
{
    int64_t ret = StartHW64( (int64_t)LOfreq );
    return (int)ret;
}

//---------------------------------------------------------------------
extern "C" int64_t EXTIO_API StartHW64(int64_t LOfreq)
```

```
{
    if (!gbInitHW)
        return 0;

    stopThread();

    SetHWLO64(LOfreq);

    startThread();

    // number of complex elements returned each
    // invocation of the callback routine
    return EXT_BLOCKLEN;
}

//---------------------------------------------------------------------
extern "C" void EXTIO_API StopHW(void)
{
    stopThread();
    return;  // nothing to do with this specific HW
}

//---------------------------------------------------------------------
extern "C" void EXTIO_API CloseHW(void)
{
    // ..... here you can shutdown your graphical interface, if any.
    if (gbInitHW )
    {
        /* close port */
    }
    gbInitHW = false;
}

//---------------------------------------------------------------------
extern "C" int  EXTIO_API SetHWLO(long LOfreq)
{
    int64_t ret = SetHWLO64( (int64_t)LOfreq );
    return (ret & 0xFFFFFFFF);
}

extern "C" int64_t EXTIO_API SetHWLO64(int64_t LOfreq)
{
    // ..... set here the LO frequency in the controlled hardware
    // Set here the frequency of the controlled hardware to LOfreq
    const int64_t wishedLO = LOfreq;
    int64_t ret = 0;

    // calculate nearest possible frequency
    // - emulate receiver which don't have 1 Hz resolution
    LOfreq += LO_PRECISION / 2;
    LOfreq /= LO_PRECISION;
    LOfreq *= LO_PRECISION;

    // same LO - but user wanted change?
    if ( LOfreq == glLOfreq )
    {
```

```
        if ( wishedLO < glLOfreq )
            LOfreq -= LO_PRECISION;
        else if ( wishedLO > glLOfreq )
            LOfreq += LO_PRECISION;
    }

    // check limits
    if ( LOfreq < LO_MIN )
    {
        LOfreq = LO_MIN;
        ret = -LO_MIN;
    }
    else if ( LOfreq > LO_MAX )
    {
        LOfreq = LO_MAX;
        ret = LO_MAX;
    }

    // take frequency
    glLOfreq = LOfreq;

    if ( gbInitHW )
    {
        // tune to that frequency
        // @TODO: recalc / modify carrier frequencies???
        // int64_t err = wishedLO - glLOfreq;
    }

    if ( wishedLO != LOfreq  &&  pfnCallback )
        pfnCallback( -1, extHw_Changed_LO, 0.0F, 0 );

    // 0 The function did complete without errors.
    // < 0 (a negative number N)
    //     The specified frequency is lower than the minimum that the
hardware is capable to generate.
    //     The absolute value of N indicates what is the minimum
supported by the HW.
    // > 0 (a positive number N) The specified frequency is greater
than the maximum that the hardware
    //     is capable to generate.
    //     The value of N indicates what is the maximum supported by
the HW.
    return ret;
}

//---------------------------------------------------------------------
extern "C" int  EXTIO_API GetStatus(void)
{
    return 0;  // status not supported by this specific HW,
}

//---------------------------------------------------------------------
extern "C" void EXTIO_API SetCallback( pfnExtIOCallback funcptr )
{
    pfnCallback = funcptr;
    return;
```

```
}

//---------------------------------------------------------------------
extern "C" long EXTIO_API GetHWLO(void)
{
   return (long)( glLOfreq & 0xFFFFFFFF );
}

extern "C" int64_t EXTIO_API GetHWLO64(void)
{
   return glLOfreq;
}

//---------------------------------------------------------------------
extern "C" long EXTIO_API GetHWSR(void)
{
   // This DLL controls just an oscillator, not a digitizer
   return gExtSampleRate;
}

//---------------------------------------------------------------------

// extern "C" long EXTIO_API GetTune(void);
// extern "C" void EXTIO_API GetFilters(int& loCut, int& hiCut, int&
pitch);
// extern "C" char EXTIO_API GetMode(void);
// extern "C" void EXTIO_API ModeChanged(char mode);
// extern "C" void EXTIO_API IFLimitsChanged(long low, long high);
// extern "C" void EXTIO_API TuneChanged(long freq);

// extern "C" void    EXTIO_API TuneChanged64(int64_t freq);
// extern "C" int64_t EXTIO_API GetTune64(void);
// extern "C" void    EXTIO_API IFLimitsChanged64(int64_t low,
int64_t high);

//---------------------------------------------------------------------

// extern "C" void EXTIO_API RawDataReady(long samprate, int *Ldata,
int *Rdata, int numsamples)

//---------------------------------------------------------------------
extern "C" void EXTIO_API VersionInfo(const char * progname, int
ver_major, int ver_minor)
{
  SDR_progname[0] = 0;
  SDR_ver_major = -1;
  SDR_ver_minor = -1;

  if ( progname )
  {
    strncpy( SDR_progname, progname, sizeof(SDR_progname) -1 );
    SDR_ver_major = ver_major;
    SDR_ver_minor = ver_minor;

   // possibility to check program's capabilities
   // depending on SDR program name and version,
```

```
    // f.e. if specific extHWstatusT enums are supported
  }
}

//-----------------------------------------------------------------------

// following "Attenuator"s visible on "RF" button

extern "C" int EXTIO_API GetAttenuators( int atten_idx, float *
attenuation )
{
    // fill in attenuation
    // use positive attenuation levels if signal is amplified (LNA)
    // use negative attenuation levels if signal is attenuated
    // sort by attenuation: use idx 0 for highest attenuation / most
damping
    // this functions is called with incrementing idx
    //    - until this functions return != 0 for no more attenuator
setting

    switch ( atten_idx )
    {
    case 0:       *attenuation = -30.0F;  return 0;
    case 1:       *attenuation = -20.0F;  return 0;
    case 2:       *attenuation = -10.0F;  return 0;
    case 3:       *attenuation = -6.0F;   return 0;
    case 4:       *attenuation =  0.0F;   return 0;
    case 5:       *attenuation =  9.0F;   return 0;
    default:  return 1;
    }
    return 1;
}

extern "C" int EXTIO_API GetActualAttIdx(void)
{
    return giAttIdx; // returns -1 on error
}

extern "C" int EXTIO_API SetAttenuator( int atten_idx )
{
    int iPrevAttIdx = giAttIdx;

    switch ( atten_idx )
    {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        giAttIdx = atten_idx;
        if ( iPrevAttIdx != giAttIdx )
            ++giParameterSetNo;
        return 0;
    default:
        return 1; // ERROR
```

```
    }
    return 1;  // ERROR
}

//------------------------------------------------------------------

// optional function to get AGC Mode: AGC_OFF (always agc_index = 0),
AGC_SLOW, AGC_MEDIUM, AGC_FAST, ...
// this functions is called with incrementing idx
//    - until this functions returns != 0, which means that all agc
modes are already delivered

extern "C" int EXTIO_API ExtIoGetAGCs(int agc_idx, char * text)  //
text limited to max 16 char
{
    switch (agc_idx)
    {
    case 0:       strcpy(text, "MGC"); return 0;
    case 1:       strcpy(text, "AGC"); return 0;
    case 2:       strcpy(text, "Thr"); return 0;
    //case 3:     strcpy(text, "?");      return 0;
    default:  return 1;
    }
    return 1;
}

extern "C" int EXTIO_API ExtIoGetActualAGCidx(void)
{
    return giAgcIdx; // returns -1 on error
}

extern "C" int EXTIO_API ExtIoSetAGC(int agc_idx)
{
    // returns != 0 on error
    int iPrevAgcIdx = giAgcIdx;

    switch (agc_idx)
    {
    case 0:
    case 1:
    case 2:
    //case 3:
        giAgcIdx = agc_idx;
        if (iPrevAgcIdx != giAgcIdx)
        {
            ++giParameterSetNo;
            if (pfnCallback )
                EXTIO_STATUS_CHANGE(pfnCallback, extHw_Changed_RF_IF);
        }
        return 0;
    default:
        return 1;  // ERROR
    }
    return 1;  // ERROR
}
```

```
// optional: HDSDR >= 2.62
extern "C" int EXTIO_API ExtIoShowMGC(int agc_idx)     // return 1,
to continue showing MGC slider on AGC
// return 0, is default for not showing MGC slider
{
    switch (agc_idx)
    {
    case 0:   return 1; // MGC
    case 1:   return 1; // AGC
    case 2:   return 1; // Thr
    //case 3: return 1; // ?
    default:
        return 0; // ERROR
    }
    return 0; // ERROR
}

//------------------------------------------------------------------

// following "MGC"s visible on "IF" button

extern "C" int EXTIO_API ExtIoGetMGCs(int mgc_idx, float * gain)
{
    // fill in gain
    // sort by ascending gain: use idx 0 for lowest gain
    // this functions is called with incrementing idx
    //    - until this functions returns != 0, which means that all
gains are already delivered

    switch (giAgcIdx)
    {
    case 0:   // MGC
        switch (mgc_idx)
        {
        case 0:       *gain = 0.0F; return 0;
        case 1:       *gain = 3.0F; return 0;
        case 2:       *gain = 6.0F; return 0;
        default:  return 1;
        }
        break;
    case 1:   // AGC
        //return 1;
        switch (mgc_idx) // set threshold!
        {
        case 0:       *gain = 0.0F; return 0;
        case 1:       *gain = 10.0F;   return 0;
        case 2:       *gain = 20.0F;   return 0;
        case 3:       *gain = 30.0F;   return 0;
        case 4:       *gain = 40.0F;   return 0;
        default:  return 1;
        }
        break;
    case 2:   // Thr
        switch (mgc_idx)
        {
        case 0:       *gain = 10.0F;   return 0;
```

```
        case 1:       *gain = 20.0F;   return 0;
        case 2:       *gain = 30.0F;   return 0;
        case 3:       *gain = 40.0F;   return 0;
        case 4:       *gain = 50.0F;   return 0;
        default:  return 1;
        }
        break;
    case 3:   // ?
        switch (mgc_idx)
        {
        case 0:       *gain = 50.0F;   return 0;
        case 1:       *gain = 60.0F;   return 0;
        default:  return 1;
        }
        break;
    }
    return 1;
}

extern "C" int EXTIO_API ExtIoGetActualMgcIdx(void)
{
    switch (giAgcIdx)
    {
    case 0:   // MGC
        return giMgcIdx; // returns -1 on error
    case 1:   // AGC
        //return -1;
        return giThrIdx;
    case 2:   // Thr
        return giThrIdx;
    case 3:
        return giWhatIdx;
    }
    return -1;
}

extern "C" int EXTIO_API ExtIoSetMGC(int mgc_idx)
{
    int iPrevMgcIdx = giMgcIdx;
    int iPrevThrIdx = giThrIdx;

    switch (giAgcIdx)
    {
    case 0:   // MGC
        switch (mgc_idx)
        {
        case 0:
        case 1:
        case 2:
            giMgcIdx = mgc_idx;
            if (iPrevMgcIdx != giMgcIdx)
                ++giParameterSetNo;
            return 0;
        default:
            return 1; // ERROR
        }
```

```
        break;

    case 1:    // AGC
        //break;

    case 2:    // Thr
        switch (mgc_idx)
        {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
            giThrIdx = mgc_idx;
            if (iPrevMgcIdx != giThrIdx)
                ++giParameterSetNo;
            return 0;
        default:
            return 1; // ERROR
        }
        break;
    case 3:    // ?
        switch (mgc_idx)
        {
        case 0:
        case 1:
            giWhatIdx = mgc_idx;
            return 0;
        default:
            return 1; // ERROR
        }
        break;
    }
    return 1; // ERROR
}

//------------------------------------------------------------------

extern "C" int EXTIO_API ExtIoGetSrates( int srate_idx, double *
samplerate )
{
    switch ( srate_idx )
    {
    case 0:       *samplerate =   48000.0;return 0;
    case 1:       *samplerate =   96000.0;return 0;
    case 2:       *samplerate =  192000.0;return 0;
    case 3:       *samplerate =  384000.0;return 0;
    case 4:       *samplerate =  768000.0;return 0;
    case 5:       *samplerate = 1536000.0;return 0;
    case 6:       *samplerate = 2400000.0;return 0;
    case 7:       *samplerate = 3072000.0;return 0;
    case 8:       *samplerate = 6144000.0;return 0;
    case 9:       *samplerate = gCustomSamplerate;  return 0;
    default:  return 1; // ERROR
    }
    return 1; // ERROR
```

```
}

extern "C" int  EXTIO_API ExtIoGetActualSrateIdx(void)
{
    return giExtSrateIdx;
}

extern "C" int  EXTIO_API ExtIoSetSrate( int srate_idx )
{
    double newSrate = 0.0;
    if ( 0 == ExtIoGetSrates( srate_idx, &newSrate ) )
    {
        giExtSrateIdx = srate_idx;
        gExtSampleRate = (unsigned)( newSrate + 0.5 );
        ++giParameterSetNo;
        return 0;
    }
    return 1; // ERROR
}

extern "C" long EXTIO_API ExtIoGetBandwidth( int srate_idx )
{
    double newSrate = 0.0;
    long ret = -1L;
    if ( 0 == ExtIoGetSrates( srate_idx, &newSrate ) )
    {
        switch ( srate_idx )
        {
            case 0:       ret = 40000L; break;
            case 1:       ret = 80000L; break;
            case 2:       ret = 160000L;   break;
            case 3:       ret = 320000L;   break;
            case 4:       ret = 640000L;   break;
            case 5:       ret = 1280000L;  break;
            case 6:       ret = 2000000L;  break;
            case 7:       ret = 2560000L;  break;
            case 8:       ret = 5120000L;  break;
            case 9:       ret = (long)(gCustomSamplerate * 0.8);
    break;
            default:  ret = -1L;    break;
        }
        return ( ret >= newSrate || ret <= 0L ) ? -1L : ret;
    }
    return -1L;   // ERROR
}

//---------------------------------------------------------------------

extern "C" int EXTIO_API ExtIoGetSetting( int idx, char *
description, char * value )
{
    const char * hwTypeStr = 0;
    switch (gHwType)
    {
    default:
    case exthwUSBfloat32:   hwTypeStr = "FLOAT"; break;
```

```
    case exthwUSBdata24: hwTypeStr = "PCM24"; break;
    case exthwUSBdata32: hwTypeStr = "PCM2432";  break;
    case exthwFullPCM32: hwTypeStr = "PCM32"; break;
    case exthwUSBdataU8: hwTypeStr = "PCMU8"; break;
    case exthwUSBdataS8: hwTypeStr = "PCMS8"; break;
    }

    switch ( idx )
    {
    case 0: snprintf( description, 1024, "%s", "Identifier" );
    snprintf( value, 1024, "%s", SETTINGS_IDENTIFIER ); return 0;
    case 1:   snprintf( description, 1024, "%s", "SampleRateIdx" );
    snprintf( value, 1024, "%d", giExtSrateIdx );       return 0;
    case 2:   snprintf( description, 1024, "%s", "AttenuationIdx" );
    snprintf( value, 1024, "%d", giAttIdx );         return 0;
    case 3:   snprintf( description, 1024, "%s", "0_Freq_Hz" );
    snprintf( value, 1024, "%.3f", gaCarrierFreq[0] );  return 0;
    case 4:   snprintf( description, 1024, "%s", "0_Level_dB" );
    snprintf( value, 1024, "%.3f", gaCarrierLevel[0] ); return 0;
    case 5:   snprintf( description, 1024, "%s", "1_Freq_Hz" );
    snprintf( value, 1024, "%.3f", gaCarrierFreq[1] );  return 0;
    case 6:   snprintf( description, 1024, "%s", "1_Level_dB" );
    snprintf( value, 1024, "%.3f", gaCarrierLevel[1] ); return 0;
    case 7:   snprintf( description, 1024, "%s", "SampleType
FLOAT/PCM24/PCM2432/PCM32/PCMU8/PCMS8" );    snprintf( value, 1024,
"%s", hwTypeStr );   return 0;
    case 8:   snprintf( description, 1024, "%s", "SampleRate Hz" );
    snprintf( value, 1024, "%u", gCustomSamplerate );   return 0;
    default:  return -1;// ERROR
    }
    return -1;// ERROR
}


extern "C" void EXTIO_API ExtIoSetSetting( int idx, const char *
value )
{
    double newSrate;
    float  newAtten = 0.0F;
    int tempInt;
    // now we know that there's no need to save our settings into some
(.ini) file,
    // what won't be possible without admin rights!!!,
    // if the program (and ExtIO) is installed in C:\Program files\..
    SDR_supports_settings = true;
    if ( idx != 0 && !SDR_settings_valid )
        return;   // ignore settings for some other ExtIO

    switch ( idx )
    {
    case 0:       SDR_settings_valid = ( value && !strcmp( value,
SETTINGS_IDENTIFIER ) );
                  // make identifier version specific??? - or not ==>
never change order of idx!
                  break;
    case 1:       tempInt = atoi( value );
```

```
            if ( 0 == ExtIoGetSrates( tempInt, &newSrate ) )
            {
                giExtSrateIdx = tempInt;
                gExtSampleRate = (unsigned)( newSrate + 0.5 );
            }
            break;
   case 2:      tempInt = atoi( value );
            if ( 0 == GetAttenuators( tempInt,&newAtten ) )
                giDefaultAttIdx = giAttIdx = tempInt;
            break;
   case 3:      gaCarrierFreq[0]  = atof(value);  break;
   case 4:      gaCarrierLevel[0] = atof(value);  break;
   case 5:      gaCarrierFreq[1]  = atof(value);  break;
   case 6:      gaCarrierLevel[1] = atof(value);  break;
   case 7:      if (!strcmp(value, "FLOAT") || !strcmp(value,
"float") || !strcmp(value, "FLT") || !strcmp(value, "flt") ||
!strcmp(value, "FLOAT32") || !strcmp(value, "float32"))
                gHwType = exthwUSBfloat32;
            else if (!strcmp(value, "PCM24") || !strcmp(value,
"pcm24"))
                gHwType = exthwUSBdata24;
            else if (!strcmp(value, "PCM2432") || !strcmp(value,
"pcm2432"))
                gHwType = exthwUSBdata32;
            else if (!strcmp(value, "PCM32") || !strcmp(value,
"pcm32"))
                gHwType = exthwFullPCM32;
            else if (!strcmp(value, "PCMU8") || !strcmp(value,
"pcmu8") || !strcmp(value, "PCM8") || !strcmp(value, "pcm8"))
                gHwType = exthwUSBdataU8;
            else if (!strcmp(value, "PCMS8") || !strcmp(value,
"pcms8"))
                gHwType = exthwUSBdataS8;
            else
                gHwType = exthwUSBfloat32;
            break;
   case 8:      if ( atoi(value) > 0 )
                gCustomSamplerate = atoi(value);
            break;
   }

}

//----------------------------------------------------------------------
```

Plik ExtIO_Demo.h

```
#ifdef EXTIO_EXPORTS
#define EXTIO_API __declspec(dllexport) __stdcall
#else
#define EXTIO_API __declspec(dllimport)
#endif

#include "LC_ExtIO_Types.h"

extern "C" bool EXTIO_API InitHW(char *name, char *model, int& type);
```

```
extern "C" int64_t EXTIO_API StartHW64(int64_t freq);
extern "C" bool EXTIO_API OpenHW(void);
extern "C" int  EXTIO_API StartHW(long freq);
extern "C" void EXTIO_API StopHW(void);
extern "C" void EXTIO_API CloseHW(void);
extern "C" int  EXTIO_API SetHWLO(long LOfreq);
extern "C" int64_t EXTIO_API SetHWLO64(int64_t LOfreq);
extern "C" int  EXTIO_API GetStatus(void);
extern "C" void EXTIO_API SetCallback(pfnExtIOCallback funcptr);// void extIOCallback(int cnt, int
status, float IQoffs, short IQdata[]);
extern "C" long EXTIO_API GetHWLO(void);
extern "C" int64_t EXTIO_API GetHWLO64(void);
extern "C" long EXTIO_API GetHWSR(void);

// extern "C" long EXTIO_API GetTune(void);
// extern "C" void EXTIO_API GetFilters(int& loCut, int& hiCut, int& pitch);
// extern "C" char EXTIO_API GetMode(void);
// extern "C" void EXTIO_API ModeChanged(char mode);
// extern "C" void EXTIO_API IFLimitsChanged(long low, long high);
// extern "C" void EXTIO_API TuneChanged(long freq);
// extern "C" void    EXTIO_API TuneChanged64(int64_t freq);
// extern "C" int64_t EXTIO_API GetTune64(void);
// extern "C" void    EXTIO_API IFLimitsChanged64(int64_t low, int64_t high);
// extern "C" void EXTIO_API RawDataReady(long samprate, int *Ldata, int *Rdata, int numsamples);

extern "C" void EXTIO_API VersionInfo(const char * progname, int ver_major, int ver_minor);
extern "C" int EXTIO_API GetAttenuators(int idx, float * attenuation);  // fill in attenuation
                        // use positive attenuation levels if signal is amplified (LNA)
                        // use negative attenuation levels if signal is attenuated
                        // sort by attenuation: use idx 0 for highest attenuation / most damping
                        // this functions is called with incrementing idx
                        //    - until this functions return != 0 for no more attenuator setting
extern "C" int EXTIO_API GetActualAttIdx(void);                     // returns -1 on error
extern "C" int EXTIO_API SetAttenuator(int idx);                    // returns != 0 on error
extern "C" int EXTIO_API ExtIoGetAGCs(int agc_idx, char * text);
extern "C" int EXTIO_API ExtIoGetActualAGCidx(void);
extern "C" int EXTIO_API ExtIoSetAGC(int agc_idx);
extern "C" int EXTIO_API ExtIoShowMGC(int agc_idx);
extern "C" int EXTIO_API ExtIoGetMGCs(int mgc_idx, float * gain);
extern "C" int EXTIO_API ExtIoGetActualMgcIdx(void);
extern "C" int EXTIO_API ExtIoSetMGC(int mgc_idx);
extern "C" int  EXTIO_API ExtIoGetSrates(int idx, double * samplerate);  // fill in possible samplerates
extern "C" int  EXTIO_API ExtIoGetActualSrateIdx(void);             // returns -1 on error
extern "C" int  EXTIO_API ExtIoSetSrate(int idx);                   // returns != 0 on error
extern "C" long EXTIO_API ExtIoGetBandwidth(int srate_idx);       // returns != 0 on error
extern "C" int  EXTIO_API ExtIoGetSetting( int idx, char * description, char * value ); // will be called
(at least) before exiting application
extern "C" void EXTIO_API ExtIoSetSetting( int idx, const char * value ); // before calling InitHW()
!!!
```

Plik ExtIO_Demo.def

```
LIBRARY      ExtIO_Test.DLL

EXPORTS
```

```
    InitHW                      = _InitHW@12
    OpenHW                      = _OpenHW@0
    StartHW                     = _StartHW@4
;   StartHW64                   = _StartHW64@
    StopHW                      = _StopHW@0
    CloseHW                     = _CloseHW@0
    SetHWLO                     = _SetHWLO@4
    GetStatus                   = _GetStatus@0
    SetCallback                 = _SetCallback@4

    GetHWLO                     = _GetHWLO@0
    GetHWSR                     = _GetHWSR@0

;    GetTune                     = _GetTune@0
;    GetFilters                  = _GetFilters@12
;    GetMode                     = _GetMode@0
;    ModeChanged                 = _ModeChanged@4
;    IFLimitsChanged             = _IFLimitsChanged@8
;    TuneChanged                 = _TuneChanged@4

;    ShowGUI                     = _ShowGUI@0
;    HideGUI                     = _HideGUI@0

;    RawDataReady                = _RawDataReady@16

    VersionInfo                 = _VersionInfo@12

    GetAttenuators              = _GetAttenuators@8
    GetActualAttIdx             = _GetActualAttIdx@0
    SetAttenuator               = _SetAttenuator@4

    ExtIoGetAGCs                = _ExtIoGetAGCs@8
    ExtIoGetActualAGCidx        = _ExtIoGetActualAGCidx@0
    ExtIoSetAGC                 = _ExtIoSetAGC@4
    ExtIoShowMGC                = _ExtIoShowMGC@4

    ExtIoGetMGCs                = _ExtIoGetMGCs@8
    ExtIoGetActualMgcIdx        = _ExtIoGetActualMgcIdx@0
    ExtIoSetMGC                 = _ExtIoSetMGC@4

; 64 Bit:

    SetHWLO64                   = _SetHWLO64@8
    GetHWLO64                   = _GetHWLO64@0
;    GetTune                     = _GetTune64@
;    IFLimitsChanged64           = _IFLimitsChanged64@
;    TuneChanged64               = _TuneChanged64@

; Samplerate / Bandwidth:
    ExtIoGetSrates              = _ExtIoGetSrates@8
    ExtIoGetActualSrateIdx      = _ExtIoGetActualSrateIdx@0
    ExtIoSetSrate               = _ExtIoSetSrate@4
    ExtIoGetBandwidth           = _ExtIoGetBandwidth@4

; Settings
    ExtIoGetSetting             = _ExtIoGetSetting@12
```

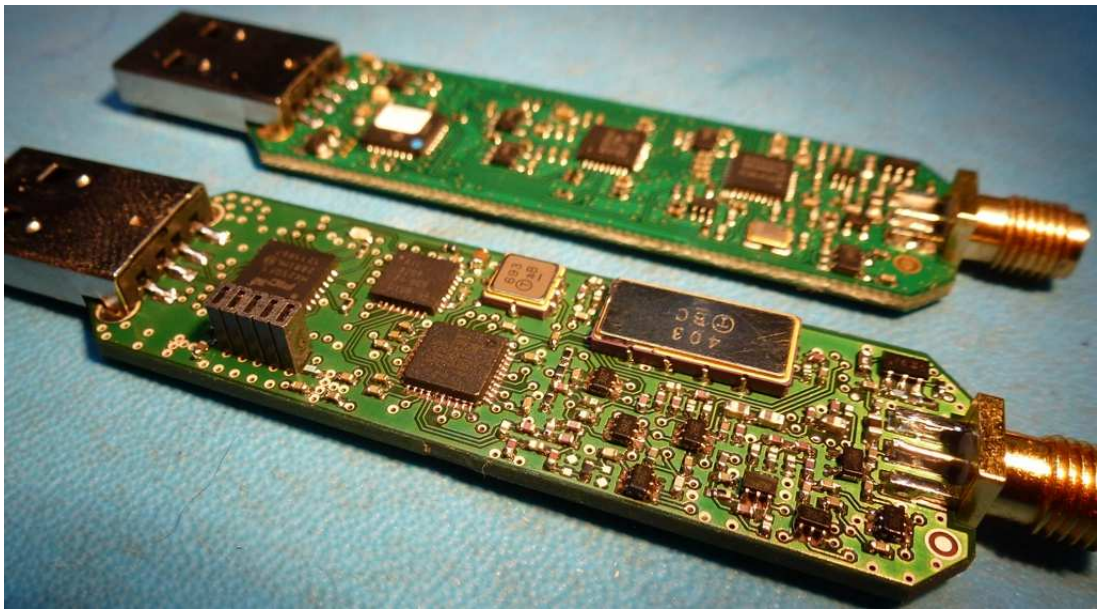
```
    ExtIoSetSetting                 = _ExtIoSetSetting@8
```

Dodatek A

Porównanie odbiorników „Fun Cube Dongle” z telewizyjnymi paluszkami DVB-T

Zainteresowani miniaturowymi odbiornikami programowalnymi USB mają do wyboru specjalnie w tym celu skonstruowany model „Fun Cube Dongle” (FCD) lub telewizyjny odbiornik DVB-T, dający się wykorzystać w nietypowy sposób dzięki specjalnie opracowanemu oprogramowaniu. Oba rozwiązania mają swoje wady i zalety.

Do konkurencji stanęły aktualny model odbiornika „Fun Cube Dongle Pro+” (FCD2) i jeden z licznych bezimiennych komputerowych odbiorników DVB-T wyposażonych w procesor sygnału RTL-2823U oraz głowicę odbiorczą R820T. Odbiornik DVB-T jest wyposażony w gniazdo antenowe MCX, natomiast FCD – w standardowe gniazdo SMA, co ułatwia zastosowanie 50-omowych kabli antenowych różnej maści: RG-316, RG-58, H-155 itd.



Fot. A.1. Konstrukcja wewnętrzna FCD i FCD2

Kandydat pierwszy „Fun Cube Dongle Pro+”

Instalacja FCD2 przebiega bezproblemowo. Po wetknięciu go do gniazda system operacyjny instaluje automatycznie standardowy sterownik dla systemu dźwiękowego USB. Pozwala to wielu programom na bezpośredni dostęp do odbiornika bez udziału pomocników w rodzaju „Virtual Audio Cable” i przy mniejszym obciążeniu CPU. Gwarantowany zakres odbioru rozciąga się od 150 kHz do 1,9 GHz z przerwą pomiędzy 240 a 420 MHz. W badanym egzemplarzu leżała ona między 260 i 417 MHz. Czułość i poziom szumów w pasmach amatorskich 2 m i 70 cm nie sprawiają zawodu. Odbiornik osiąga podane 0,15 µV przy odstępie 12 dB SINAD także dla zakresów powyżej 440 MHz, przy czym górna granica pomiaru wynosiła 1 GHz. Współczynnik szumów w obu pasmach wynosi 3,5 dB – przyp. tłum.
Ponieważ FCD2 jest wyposażony w 11 przełączanych filtrów pasmowych występuje w nim – szczególnie przy silnych sygnałach odbieranych – wyraźnie mniej składowych intermodulacyjnych aniżeli w odbiorniku DVB-T.
Czułość w zakresie krótkofalowym jest odrobinę niższa, co pozwala na ustawienie w konfiguracji większego wzmocnienia p.cz.

Sygnał odbierany jest w FCD2 przetwarzany w przetworniku-16 bitowym, co zapewnia większą rozdzielczość wskaźnika wodospadowego i ułatwia precyzyjniejsze dostrojenie. Oprócz tego odbiornik posiada szerszy zakres dynamiki i dzięki temu większą odporność na zniekształcenia intermodulacyjne. W odbiornikach DVB-T sygnały są natomiast przetwarzane z rozdzielczością 8 bitów (co daje teoretyczny zakres dynamiki tylko ok. 57 dB – przyp. tłum.). Szerokość pasma wyświetlanego przez FCD2 jest niestety ograniczona do 192 kHz.

Fot. A.2. Konstrukcja odbiornka telewizyjnego DVB-T

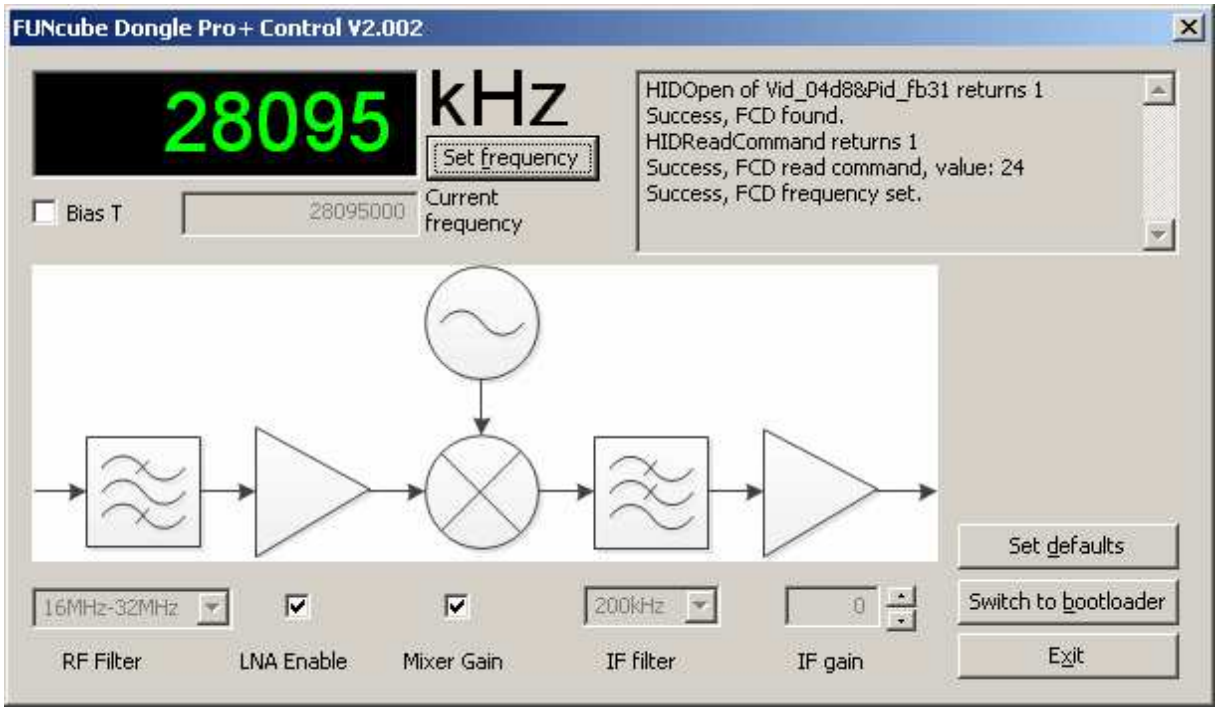
Kandytat drugi: odbiornik DVB-T

Instalacja odbiornika wymaga trochę więcej wysiłku, ale nie jest w rzeczywistości skomplikowana. Konieczne jest uprzednie ręczne zainstalowanie sterowników Zadig (dostępnych najczęściej w postaci pliku *zadig.exe* – przyp. tłum.) i dopiero potem uruchomienie programu odbiorczego (np. SDR# [2]). Zakres pracy głowicy odbiorczej R820 rozciąga się od około 24 MHz do 1,7 GHz bez przerw, a impedancja wejściowa odbiornika wynosi nominalnie 75 Ω (pomiary opisane w poz. [1] wykazują jednak, że dopasowanie na wejściu nie jest zbyt dobre). Z przeprowadzonych pomiarów wynika, że czułość 0,15 µV była osiągana w całym zakresie pomiarowym do 1 GHz. Stosunek sygnału do szumu wypada gorzej wskutek wyższego poziomu szumów własnych. Różnica ta jest jednak praktycznie niezauważalna na słuch. Przy odbiorze silniejszych sygnałów występuje jednak wyraźnie więcej składowych intermodulacyjnych i sygnałów zakłócających m.cz.

Automatyczna regulacja wzmocnienia

O ile w FCD2 przy odbiorze słabych sygnałów na falach krótkich możliwe jest zwiększenie wzmocnienia p.cz., o tyle w odbiorniku DVB-T konieczne jest wyłączenie ARW i regulowanie czułości za

pomocą wzmocnienia stopnia w.cz., co wyraźnie podnosi jakość odbioru. W przeciwnym przypadku bardzo szybko dochodzi do przesterowania odbiornika i związanych z nim zniekształceń.
W trakcie pomiarów innego egzemplarza, opisanych w poz. [1], uzyskano dla wyłączonej ARW i ustawionego maksymalnego wzmocnienia odstęp sygnału od szumu dochodzący do 40 dB. Również przy włączonej ARW (wzmocnienie ustawiane jest wówczas automatycznie na maksimum) uzyskano taki sam odstęp. Przy wyłączonej ARW maleje on w miarę obniżania wzmocnienia. W badanym odbiorniku zakres regulacji wzmocnienia wynosił ok. 50 dB co łącznie z zakresem dynamki leżącym również w pobliżu 50 dB pozwoliło autorowi artykułu [1] na wykorzystanie paluszka DVB-T jako odbiornika pomiarowego o dynamice ok. 100 dB (przyp. tłum.).
Szerokość wyświetlanego zakresu przekracza 2 MHz co umożliwia przykładowo obserwację całego pasma 2 m. Przy odbiorze emisji wąskopasmowych np. PSK31 daje się zauważyć gorsza stabilność częstotliwości w porównaniu z FCD2. Odbiór zakresu poniżej 24 MHz jest możliwy przy użyciu dodatkowego konwertera – np. konwertera opracowanego dla poprzedniego modelu Fun Cube.



For. A.3. Schemat blokowy FCD2 na panelu sterowania i konfiguracji

Porównanie i podsumowanie

Fun Cube zapewnia wyraźnie lepszą zrozumiałość transmisji amatorskich zarówno dzięki mniejszej ilości składowych intermodulacyjnych jak i niższemu poziomowi szumów własnych. Różnica ta jest praktycznie niezauważalna przy odbiorze stacji radiofonicznych. Obciążenie CPU w komputerze autora testu wynosiło dla FCD2 około 30 %, podczas gdy dla odbiornika DVB-T było ono w przybliżeniu dwa razy wyższe – czyli około 60 %. Oznacza to, że dla Fun Cube możliwe jest zwiększenie rozdzielczości FFT w programie, co owocuje mniejszym prawdopodobieństwem występowania przerw dźwięku w trakcie odbioru. Tłumaczy to też powody większego opóźnienia sygnału przy odbiorze na paluszku telewizyjnym.
Użytkownicy, którym zależy na optymalnej zrozumiałości dźwięku i stabliności częstotliwości, a przykładający mniej wagi do ciągłości zakresu odbioru, względnie korzystający ze starszych i słabszych komputerów i akceptujący wyższą cenę wyjdą lepiej na swoje wybierając Fun Cube. Zdaniem autora jest to rozwiązanie lepsze technicznie.
Wszyscy, którym nie przeszkadzają niedociągnięcia techniczne odbiorników DVB-T mogą cieszyć się z niskiej ceny urządzeń i wynikającego z niej bardzo korzystnego stosunku przydatności do ceny. Oprócz tego odbiornik DVB-T można wykorzystywać także zgodnie z jego przeznaczeniem – tzn. do

odbioru telewizji. Program odbiorczy SDR# dekoduje sygnały RDS w obu typach odbiorników. Odbiornik FCD2 kosztuje w Niemczech ok. 185 euro.

Porady końcowe

Praktyka autora wykazała, że dobrze jest korzystać z przedłużacza USB, dzięki czemu odbiornik można umieścić w pewnej odległości od komputera i zmniejszyć w ten sposób wpływ powodowanych przez niego zakłóceń. Powinno się także unikać stosowania rozgałęźników USB ponieważ niektóre z nich są źródłem dodatkowych silnych zakłóceń. W szczególnych przypadkach o jakości odbioru może decydować także wybór gniazda USB – ale dla FCD2 nie dało się zauważyć związanego z tym pogorszenia odbioru.
Uwagę zwraca także znaczny rozrzut parametrów odbiorników telewizyjnych. Dotyczyło to przykładowo poziomu składowych intermodulacyjnych i zakłóceń dźwięku.

Tabela A.1
Porównanie parametrów FCD i odbiornika DVB-T

Parametr	FCD	Odb. DVB-T
Szerokość pasma	192 kHz	2 MHz
Zakres odbioru	150 kHz – 1,9 GHz (przerwa między 240 – 420 MHz)	24 MHz – 1,7 GHz
Rozdzielczość przetw. a/c	16 bitów	8 bitów
Impedancja wejściowa	50 Ω	75 Ω
Obciążenie CPU (przykł., zal. od programu)	30 %	60 %
Gniazdo antenowe	SMA	MCX

Michael Siebert, DF2EQ
z CQDL 3/2014 tłum. i opr. Krzysztof Dąbrowski, OE1KDA

[1] „Die unendliche SDR-Geschichte geht weiter: Untersuchung eines DVB-T Sticks mit R820T-Tuner und RTL2832U-Decoder als Messempfänger”, Gunthard Kraus, DG8GB, „UKW Berichte” 1/2014, str. 3

Adresy internetowe

[1] http://airspy.com/download – witryna sdrsharp
[2] http://sdrsharp.com
[3] www.hdsdr.de – witryna HDSDR
[4] www.weaksignals.com – witryna Alberta di Bene, I2PHD
[5] www.rtl-sdr.com – różne pomysły wykorzystania odbiorników RTL
[6] www.stoff.pl – witryna „Orbitronu”

W serii „Biblioteka polskiego krótkofalowca” dotychczas ukazały się:

Nr 1 – „Poradnik D-STAR”, wydanie 1 i 2
Nr 2 – „Instrukcja do programu D-RATS”
Nr 3 – „Technika słabych sygnałów” Tom 1
Nr 4 – „Technika słabych sygnałów” Tom 2
Nr 5 – „Łączności cyfrowe na falach krótkich” Tom 1
Nr 6 – „Łączności cyfrowe na falach krótkich” Tom 2
Nr 7 – „Packet radio”
Nr 8 – „APRS i D-PRS”
Nr 9 – „Poczta elektroniczna na falach krótkich” Tom 1
Nr 10 – „Poczta elektroniczna na falach krótkich” Tom 2
Nr 11 – „Słownik niemiecko-polski i angielsko-polski” Tom 1
Nr 12 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 1
Nr 13 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 2
Nr 14 – „Amatorska radioastronomia”
Nr 15 – „Transmisja danych w systemie D-STAR”
Nr 16 – „Amatorska radiometeorologia”
Nr 17 – „Radiolatarnie małej mocy”
Nr 18 – „Łączności na falach długich”
Nr 19 – „Poradnik Echolinku”
Nr 20 – „Arduino w krótkofalarstwie” Tom 1
Nr 21 – „Arduino w krótkofalarstwie” Tom 2
Nr 22 – „Protokół BGP w Hamnecie”
Nr 23 – „Technika słabych sygnałów” Tom 3, wydanie 1 i 2
Nr 24 – „Raspberry Pi w krótkofalarstwie”
Nr 25 – „Najpopularniejsze pasma mikrofalowe”
Nr 26 – „Poradnik DMR”
Nr 27 – „Poradnik Hamnetu”
Nr 28 – „Budujemy Ilera” Tom 1
Nr 29 – „Budujemy Ilera” Tom 2
Nr 30 – „Konstrukcje D-Starowe”
Nr 31 – „Radiostacje i odbiorniki z cyfrową obróbką sygnałów” Tom 3