


Program Współpraca Polska-RPA	RAPORT CZĄSTKOWY z realizacji projektu w ramach programu międzynarodowego Polska-RPA			 Narodowe Centrum Badań i Rozwoju
Nr raportu	IR-RATfor5G-09			
Data aktualizacji raportu:	2021.09.21	Wersja	2	
Numer umowy	PL-RPA2/02/RATfor5G+/2019	Akronim	RATfor5G+	
Okres realizacji projektu	od	2019.01.01	do	2022.06.30
Tytuł projektu	Technologie dostępu radiowego dla standardu 5G i przyszłych generacji sieci bezprzewodowych			
Tytuł raportu	Radia programowalne			

Początkowe etapy pracy polegały na wyborze radiów programowalnych. Udało się wyróżnić kilka modeli USRP, które potencjalnie brano pod uwagę.

Tabela 1 Modele USRP

Nazwa	B210	2921	B200	2944	2932	N210
Częstotliwość	70 MHz – 6 GHz	2.4 GHz - 2.5 GHz i od 4.9 GHz do 5.9 GHz	70MHz - 6GHz	10MHz - 6GHz	400MHz - 4.4GHz	do 6 GHz
Łączność	SuperSpeed USB 3.0	Ethernet	SuperSpeed USB 3.0	MXIe, Ethernet	Ethernet	Ethernet
Przepustowość	56MHz	20MHz	56MHz	160MHz	20 MHz	50 MHz 8 bit samples, 25 MHz 16 bit samples

Z racji tego, że na uczelni posiadano modele 2932 oraz N210, te dwa urządzenia zostały wstępnie wybrane jako podstawy do realizacji środowiska testowego. Z czasem uczelnia zakupiła również kilka innych modeli, w tym: Nuand BladeRF oraz USRP B200, które najbardziej odpowiadają założeniu.

Nuand BladeRF 2.0 micro xA9



Rys. 1 Nuand BladeRF micro xA9 w dodatkowej obudowie

Jest to radio programowalne oferujące szereg funkcjonalności, które może posłużyć jako podstawa do budowy odbiornika lub nadajnika, ponieważ jest dwukanałowe. Co ważne, posiada po dwa porty do wysyłania i odbierania. Urządzenie działa na wszystkich najpopularniejszych systemach

operacyjnych, tj. Windows, Linux oraz Mac OSX. Pierwsze podejścia były robione na Windowsie, aczkolwiek radio nie chciało się poprawnie skonfigurować.

Za drugim razem został wybrany system Ubuntu 16.04 LTS (Xenial Xerus) i tutaj osiągnięto pobeżny sukces pod względem konfiguracji. Oficjalna dokumentacja [18] dostępna na GitHubie twórców niestety nie pozwala wyeliminować wszystkich błędów. Udało się jednak znaleźć skrypt stworzony przez jednego z użytkowników GitHuba, który do pewnego etapu pomaga zainstalować urządzenie poprawnie.

```
14 mkdir -p $HOME/src
15
16 export BLADE="$HOME/blade"
17 mkdir -p $BLADE
18
19 # BladeRF Dependencies
20 sudo apt install -y libusb-1.0-0-dev libusb-1.0-0 build-essential cmake libncurses5-dev libtecla1 libtecla-dev pkg-config gi
21
22 # BladeRF Build and Installation
23 cd ~/src
24 git clone https://github.com/Nuand/bladeRF.git ./bladeRF
25 cd bladeRF
26 cd host/
27 mkdir build && cd build
28 cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local -DINSTALL_UDEV_RULES=ON ../
29 make && sudo make install && sudo ldconfig
30
31 # BladeRF Firmware and FPGA
32 cd $BLADE
33 wget https://www.nuand.com/fx3/bladeRF_fw_latest.img
34 wget https://www.nuand.com/fpga/hostedx4-latest.rbf
```

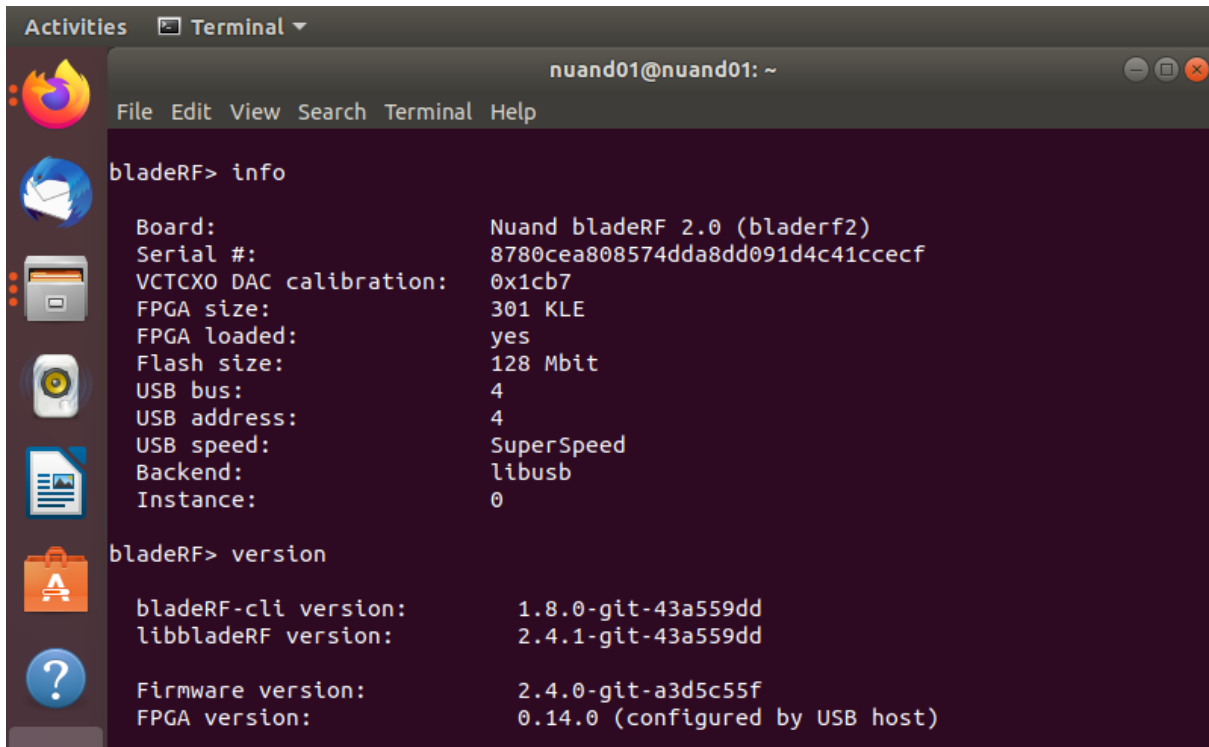
Rys. 2 Zrzut ekranu skryptu instalacyjnego

Warto nadmienić, iż dla sprawdzanego urządzenia linijki 33 i 34 trzeba było zmienić na inne ze względu na sterowniki dla innego modelu, tj. xA9. Po tych zmianach konfiguracja samego bladeRF była udana.



Rys. 3 Tłumik 50 OHM

Aby uruchomić urządzenie zalecane jest podłączenie wszystkich anten, a na wejściach TX założenie dodatkowo tłumika 50 OHM. Dopiero po ściągnięciu sterowników i instalacji, trzeba załadować FPGA komendą `bladeRF-cli -l <ścieżka/do/fpga>`. Niefortunnie jest to zrobione tak, że przy każdym podłączeniu urządzenia, czy wyłączeniu komputera wymagane jest ponowne załadowanie FPGA, chociaż w internecie można się spotkać ze skryptami to automatyzującymi, aczkolwiek nie są to oficjalne kody. Następnie trzeba załadować firmware komendą `bladeRF-cli -f <ścieżka/do/firmware>`. To wykonuje się jednorazowo, a nie jak w przypadku FPGA. Po załadowaniu nowego firmware'u konieczny jest reset urządzenia. Po tym, ponownie ładuje się na urządzenie FPGA i diody na radiu zaczynają migać. Następnie, aby sprawdzić działanie wprowadzono podstawowe komendy.



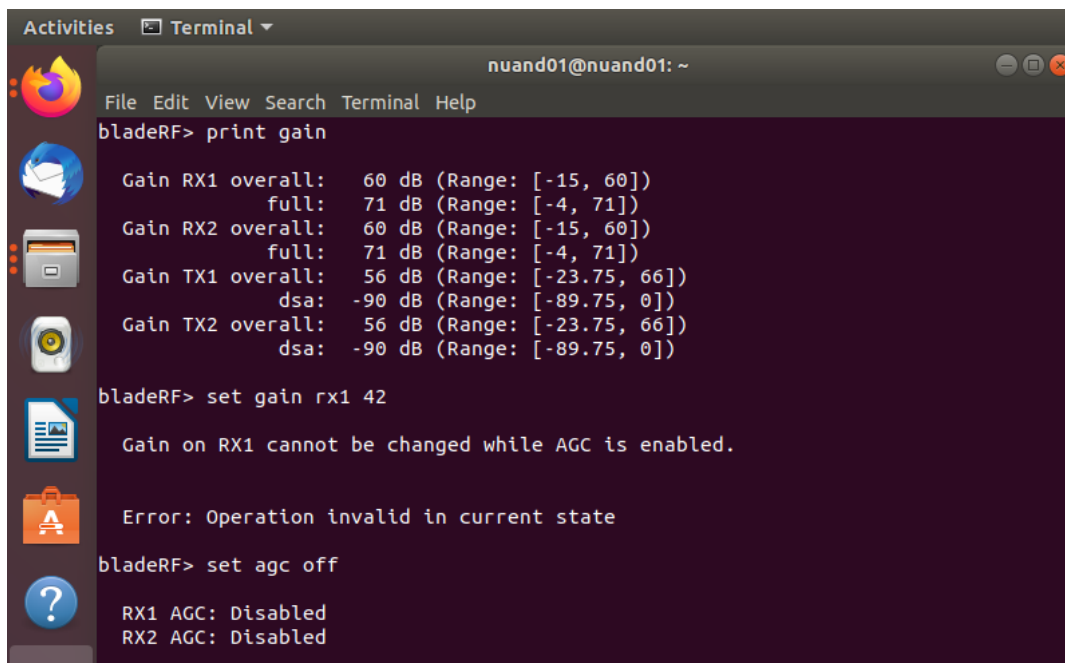
```
Activities Terminal
nuand01@nuand01: ~
File Edit View Search Terminal Help
bladeRF> info
Board: Nuand bladeRF 2.0 (bladerf2)
Serial #: 8780cea808574dda8dd091d4c41ccec
VCTCXO DAC calibration: 0x1cb7
FPGA size: 301 KLE
FPGA loaded: yes
Flash size: 128 Mbit
USB bus: 4
USB address: 4
USB speed: SuperSpeed
Backend: libusb
Instance: 0

bladeRF> version
bladeRF-cli version: 1.8.0-git-43a559dd
libbladeRF version: 2.4.1-git-43a559dd

Firmware version: 2.4.0-git-a3d5c55f
FPGA version: 0.14.0 (configured by USB host)
```

Rys. 4 Terminal Linux pokazujący podstawowe parametry bladeRF po wywołaniu komendy info

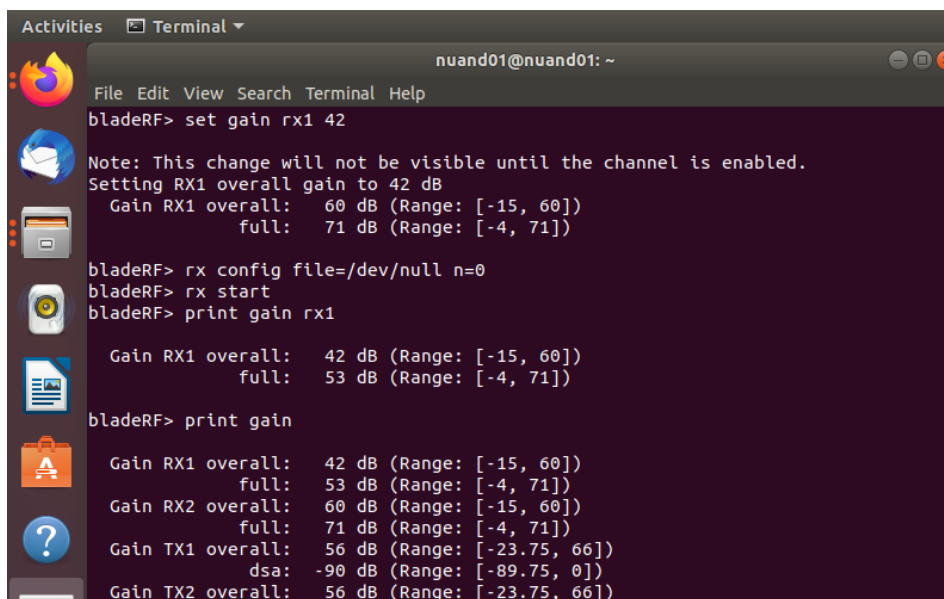
Sprawdzono zakres pasma urządzenia.



```
nuand01@nuand01: ~  
bladeRF> print gain  
Gain RX1 overall: 60 dB (Range: [-15, 60])  
full: 71 dB (Range: [-4, 71])  
Gain RX2 overall: 60 dB (Range: [-15, 60])  
full: 71 dB (Range: [-4, 71])  
Gain TX1 overall: 56 dB (Range: [-23.75, 66])  
dsa: -90 dB (Range: [-89.75, 0])  
Gain TX2 overall: 56 dB (Range: [-23.75, 66])  
dsa: -90 dB (Range: [-89.75, 0])  
  
bladeRF> set gain rx1 42  
Gain on RX1 cannot be changed while AGC is enabled.  
  
Error: Operation invalid in current state  
  
bladeRF> set agc off  
RX1 AGC: Disabled  
RX2 AGC: Disabled
```

Rys. 5 Terminal Linux pokazujący zakres pasma urządzenia.

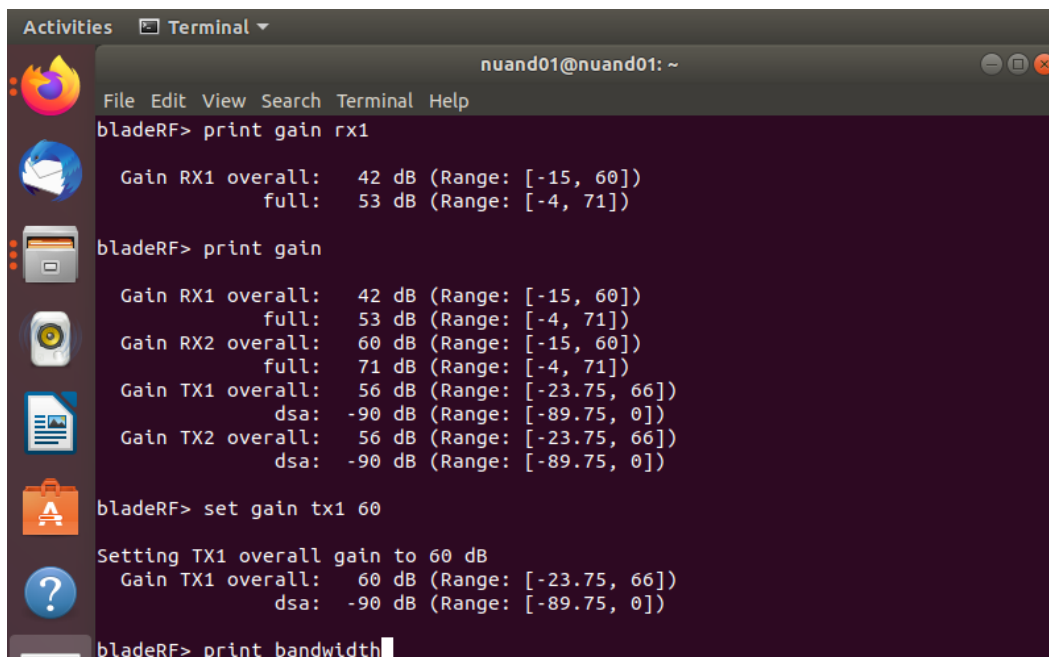
Aby ustawić jakąś wartość konieczne jest wyłączenie agc, co uczyniono.
Zmieniono wartość rx1 na 42 dB.



```
nuand01@nuand01: ~  
bladeRF> set gain rx1 42  
Note: This change will not be visible until the channel is enabled.  
Setting RX1 overall gain to 42 dB  
Gain RX1 overall: 60 dB (Range: [-15, 60])  
full: 71 dB (Range: [-4, 71])  
  
bladeRF> rx config file=/dev/null n=0  
bladeRF> rx start  
bladeRF> print gain rx1  
Gain RX1 overall: 42 dB (Range: [-15, 60])  
full: 53 dB (Range: [-4, 71])  
  
bladeRF> print gain  
Gain RX1 overall: 42 dB (Range: [-15, 60])  
full: 53 dB (Range: [-4, 71])  
Gain RX2 overall: 60 dB (Range: [-15, 60])  
full: 71 dB (Range: [-4, 71])  
Gain TX1 overall: 56 dB (Range: [-23.75, 66])  
dsa: -90 dB (Range: [-89.75, 0])  
Gain TX2 overall: 56 dB (Range: [-23.75, 66])
```

Rys. 6 Zmiana wartości rx1 na 42 dB

Ustawiono wartość tx1 na 60 dB.



```
nuand01@nuand01: ~
bladeRF> print gain rx1
Gain RX1 overall: 42 dB (Range: [-15, 60])
                  full: 53 dB (Range: [-4, 71])

bladeRF> print gain
Gain RX1 overall: 42 dB (Range: [-15, 60])
                  full: 53 dB (Range: [-4, 71])
Gain RX2 overall: 60 dB (Range: [-15, 60])
                  full: 71 dB (Range: [-4, 71])
Gain TX1 overall: 56 dB (Range: [-23.75, 66])
                  dsa: -90 dB (Range: [-89.75, 0])
Gain TX2 overall: 56 dB (Range: [-23.75, 66])
                  dsa: -90 dB (Range: [-89.75, 0])

bladeRF> set gain tx1 60
Setting TX1 overall gain to 60 dB
Gain TX1 overall: 60 dB (Range: [-23.75, 66])
                  dsa: -90 dB (Range: [-89.75, 0])

bladeRF> print bandwidth
```

Rys. 7 Zmiana wartości tx1 na 60 dB

Ustawiono pasmo ogólne na 1.5Mhz, a później zmieniono konkretnie dla RX na 4MHz.



```
nuand01@nuand01: ~
Setting TX1 overall gain to 60 dB
Gain TX1 overall: 60 dB (Range: [-23.75, 66])
                  dsa: -90 dB (Range: [-89.75, 0])

bladeRF> print bandwidth
RX1 Bandwidth: 18000000 Hz (Range: [200000, 56000000])
RX2 Bandwidth: 18000000 Hz (Range: [200000, 56000000])
TX1 Bandwidth: 18000000 Hz (Range: [200000, 56000000])
TX2 Bandwidth: 18000000 Hz (Range: [200000, 56000000])

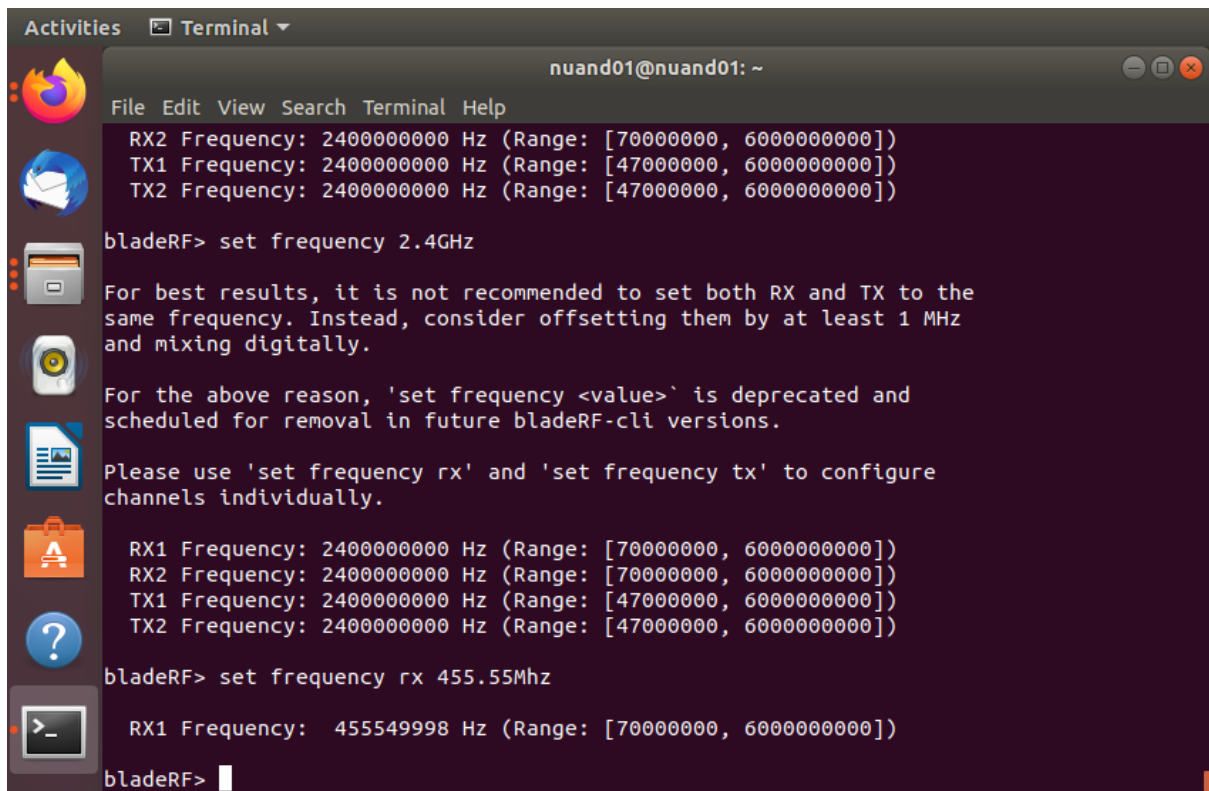
bladeRF> set bandwidth 1.5Mhz
RX1 Bandwidth: 1500000 Hz (Range: [200000, 56000000])
RX2 Bandwidth: 1500000 Hz (Range: [200000, 56000000])
TX1 Bandwidth: 1500000 Hz (Range: [200000, 56000000])
TX2 Bandwidth: 1500000 Hz (Range: [200000, 56000000])

bladeRF> set bandwidth RX 4MHz
RX1 Bandwidth: 4000000 Hz (Range: [200000, 56000000])

bladeRF>
```

Rys. 8 Zmiana pasma ogólnego na 1.5MHz i pasma RX na 4MHz

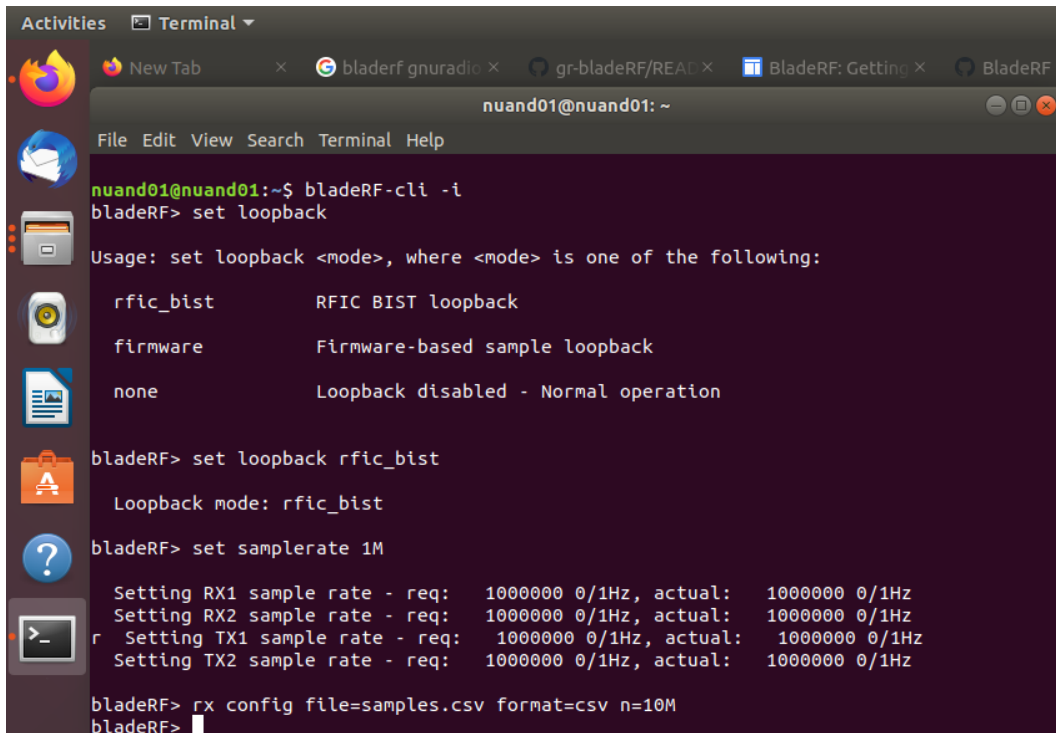
Następnie sprawdzono częstotliwość, którą zmieniono na 2.4Ghz, a później tylko dla rx na 455.55Mhz



```
nuand01@nuand01: ~  
File Edit View Search Terminal Help  
RX2 Frequency: 2400000000 Hz (Range: [70000000, 6000000000])  
TX1 Frequency: 2400000000 Hz (Range: [47000000, 6000000000])  
TX2 Frequency: 2400000000 Hz (Range: [47000000, 6000000000])  
  
bladeRF> set frequency 2.4GHz  
  
For best results, it is not recommended to set both RX and TX to the  
same frequency. Instead, consider offsetting them by at least 1 MHz  
and mixing digitally.  
  
For the above reason, 'set frequency <value>' is deprecated and  
scheduled for removal in future bladeRF-cli versions.  
  
Please use 'set frequency rx' and 'set frequency tx' to configure  
channels individually.  
  
RX1 Frequency: 2400000000 Hz (Range: [70000000, 6000000000])  
RX2 Frequency: 2400000000 Hz (Range: [70000000, 6000000000])  
TX1 Frequency: 2400000000 Hz (Range: [47000000, 6000000000])  
TX2 Frequency: 2400000000 Hz (Range: [47000000, 6000000000])  
  
bladeRF> set frequency rx 455.55Mhz  
  
RX1 Frequency: 455549998 Hz (Range: [70000000, 6000000000])  
  
bladeRF>
```

Rys. 9 Zmiana częstotliwości

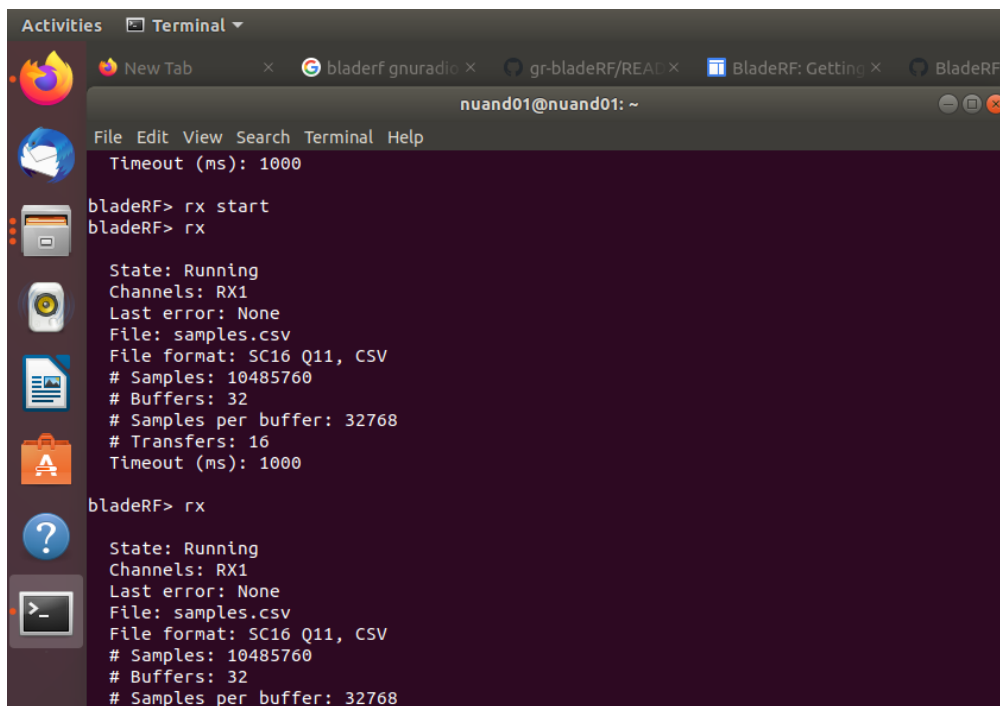
Wszystkie ustawienia na tym etapie są już gotowe i aby sprawdzić działanie radia, trzeba skonfigurować próbkę. Odbywa się to po pierwsze poprzez ustawienie loopback, ponieważ domyślnie jest wartość none, a po drugie ustawienie wartości próbki, w tym przypadku 1M. Następnie podano ścieżkę, gdzie ma być zapisany plik z danymi, jednak nawet po spakowaniu jest on zbyt duży, aby go załączyć.



```
nuand01@nuand01: ~  
bladeRF> set loopback  
Usage: set loopback <mode>, where <mode> is one of the following:  
  
rfic_bist      RFIC BIST loopback  
firmware      Firmware-based sample loopback  
none          Loopback disabled - Normal operation  
  
bladeRF> set loopback rfic_bist  
Loopback mode: rfic_bist  
  
bladeRF> set samplerate 1M  
Setting RX1 sample rate - req: 1000000 0/1Hz, actual: 1000000 0/1Hz  
Setting RX2 sample rate - req: 1000000 0/1Hz, actual: 1000000 0/1Hz  
Setting TX1 sample rate - req: 1000000 0/1Hz, actual: 1000000 0/1Hz  
Setting TX2 sample rate - req: 1000000 0/1Hz, actual: 1000000 0/1Hz  
  
bladeRF> rx config file=samples.csv format=csv n=10M  
bladeRF>
```

Rys. 10 Ustawienie próbki

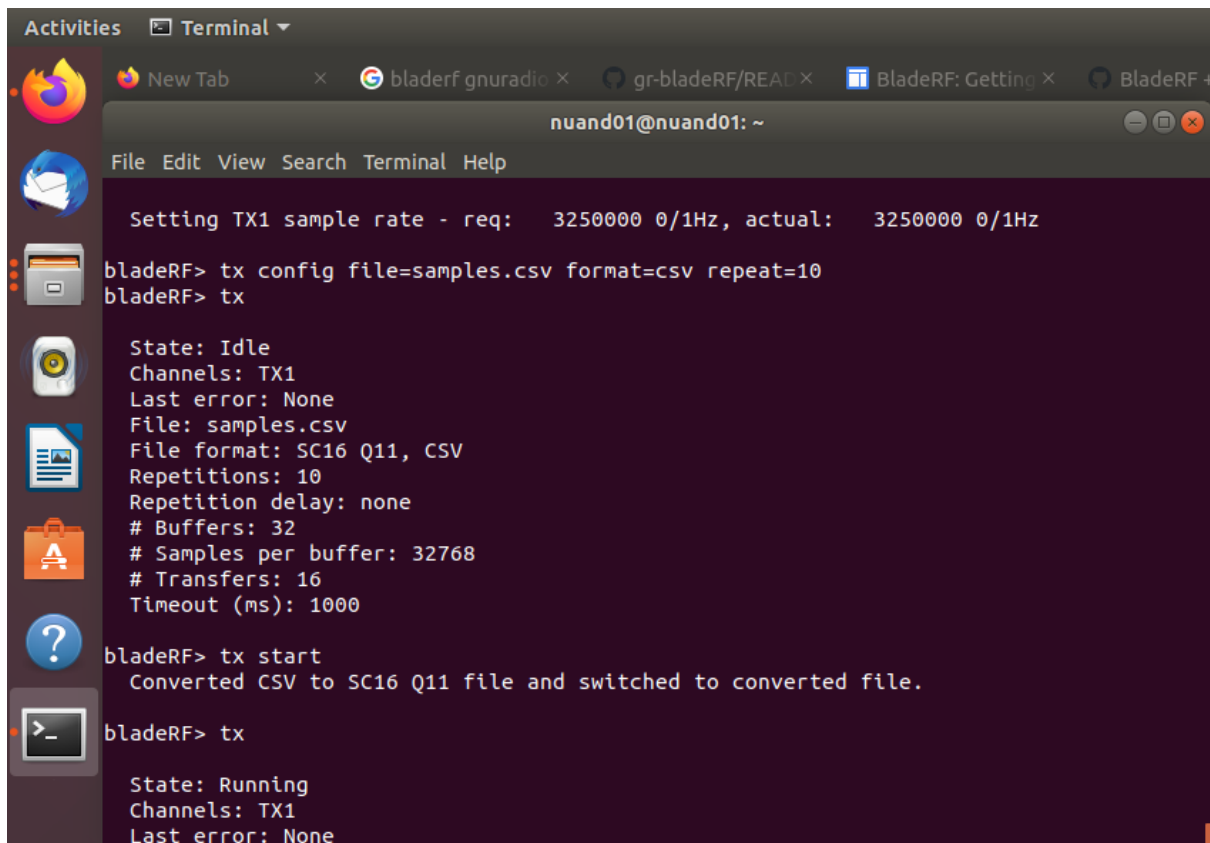
Po tej konfiguracji, wystartowano rx.



```
Timeout (ms): 1000  
bladeRF> rx start  
bladeRF> rx  
  
State: Running  
Channels: RX1  
Last error: None  
File: samples.csv  
File format: SC16 Q11, CSV  
# Samples: 10485760  
# Buffers: 32  
# Samples per buffer: 32768  
# Transfers: 16  
Timeout (ms): 1000  
  
bladeRF> rx  
  
State: Running  
Channels: RX1  
Last error: None  
File: samples.csv  
File format: SC16 Q11, CSV  
# Samples: 10485760  
# Buffers: 32  
# Samples per buffer: 32768
```

Rys. 11 Start rx

Finalnie po wygenerowanym pliku, ustawiano go dla tx i również włączono tx.



```
Setting TX1 sample rate - req: 3250000 0/1Hz, actual: 3250000 0/1Hz
bladeRF> tx config file=samples.csv format=csv repeat=10
bladeRF> tx
State: Idle
Channels: TX1
Last error: None
File: samples.csv
File format: SC16 Q11, CSV
Repetitions: 10
Repetition delay: none
# Buffers: 32
# Samples per buffer: 32768
# Transfers: 16
Timeout (ms): 1000
bladeRF> tx start
Converted CSV to SC16 Q11 file and switched to converted file.
bladeRF> tx
State: Running
Channels: TX1
Last error: None
```

Rys. 12 Ustawienie próbki tx oraz start

Po podłączeniu urządzenia do analizatora widma, zakończono sukcesem wysłanie i odebranie próbki.

USRP-2932

Kolejnym urządzeniem, które sprawdzano było USRP 2932. Urządzenie jest produkowane przez firmę National Instruments, która jest również właścicielem programu LabView. Domyślna konfiguracja jest właśnie na tę platformę. Tym razem wybrano system Windows w wersji 10. Po zarejestrowaniu programu i podłączeniu urządzenia, program wykrywał urządzenie, ale z powodu późniejszych błędów nie udało się sprawdzić jego działania. Próby używania LabView zaniechano.



Rys. 3 Dwie sztuki USRP-2932

USRP B200

Ostatnim brany pod uwagę urządzeniem było USRP B200 produkowane przez firmę Ettus Research. Sama płytko nie ma żadnej obudowy, co może być problematyczne przy ewentualnym zastosowaniu komercyjnym, ale nie jest przeszkodą w używaniu. Dwie sztuki, które były sprawdzane posiada uczelnia, w związku z czym można było je bez problemów przetestować. Radio posiada najwięcej plusów ze wszystkich testowanych, a dzięki wsparciu działania w Matlabie finalnie posłużyło za budowę testbedu.



Rys. 14 USRP B200