Make your LTE network with LimeSDR and srsLTE for fun and ... whatever

Edit article

18-22 minutes

Important Disclaimer

You can't simply build your phone service creating interference in the cellphone network: ***it's illegal***. So don't be stupid, take precautions. These test was performed using equipment with very low emissions (less than 5 mW), with low gain antennas or directly using coaxial cables, all in a controlled environment. So the working range was less than 3 meters of radius using the antennas, in a shielded room as further precaution.

Introduction

One of my previous job involved a billing system for 3G cellphone network, IMHO the particularly interesting stuff was the part involving GSM networking. At that time I purchased some books to study the details of that world, but only few information was applicable to that job, so my brain called the garbage collector on that topic. Lucky me, recently I had the chance to recall some concepts materializing my old dream: be able to play with a toy cellphone network.

Software Defined Radio (SDR) and Open Source software

What made possible the project is the availability of inexpensive SDR (Software Defined Radio) devices, a special radio transceiver implementing in software its subsystems, instead of using discrete components (hardware). The key word, there, is software: its extreme programmable nature permits SDRs to be real universal radio and give them the ability to emulate a wide range of radio equipment, included the RRU (Remote Radio Unit) subsystem present in modern eNode-B stations (evolved Node-B) of a modern cellphone network. The other important component is, obviously the availability of an open source LTE stack like srsLTE (see here for further details: https://www.srslte.com/).

LimeSDR Mini

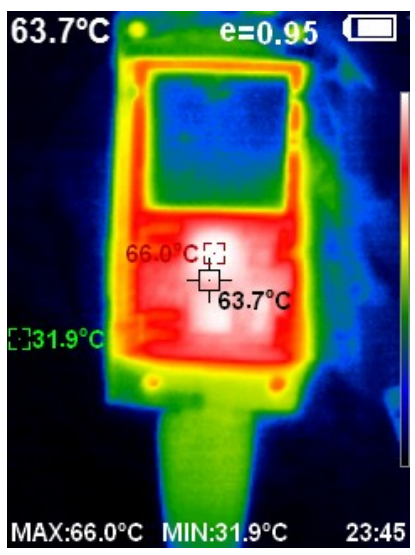
I used two LimeSDR Mini devices (why two ? I'll explain that in the following paragraphs).

See here for details:

This hardware is designed as laboratory test device, the radio emissions are very weak (5 mW) but it's sufficient for our experiments. The device is sold without antennas, you have to buy two antennas for apparatus of the type compatible with the frequencies you will set. I purchased them on Ebay.

Heat Damage Prevention

LimeSDR Mini with this application become hot quickly as shown in this image, taken using a thermal camera:



as you can see, after few minutes of operation, we have two hot spot, **66.0** and **63.7 °C.**

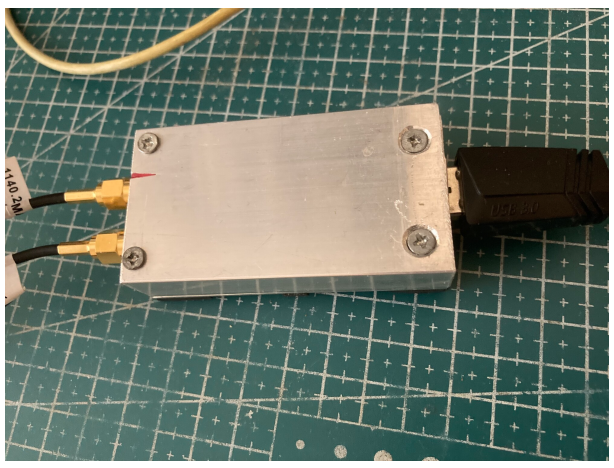
Moreover, reading the specifications, can see that there is no thermal sensor on the PCB, by default:

Temperature sensor (unpopulated)

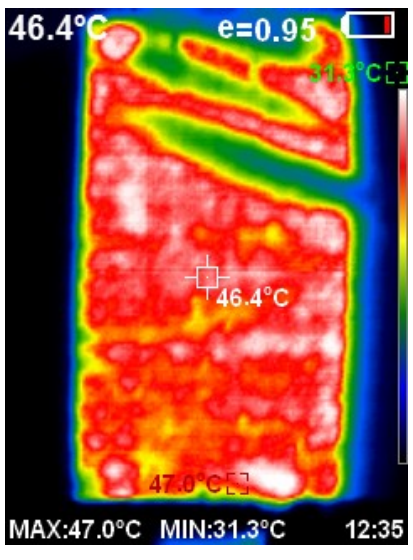
(see here: https://wiki.myriadrf.org/LimeSDR-Mini_v1.1_hardware_description)

The two spots have correspondence with the two square chips near the USB connector, as you can see in this ordinary photo of the device with the same orientation :

To prevent heat damage, I've built custom heat sinks and aluminum boxes, the internal heat sinks and the aluminum cases almost touch each other, between them I added CPU thermal paste basically transforming the whole box in a heat sink:



The thermal camera confirms that this improvised heat sink is working, this photo was taken on the second device after similar work:



You can also install a fan (a fan (5V or 3.3V) connector is present on the PCB , see here: https://wiki.myriadrf.org/LimeSDR-Mini_v1.1_hardware_description).

I have to check the temperature of the components under the shield (the blue square in the thermal image) to check if heat sinking is needed. I purchased some thermal pad for further improvement.

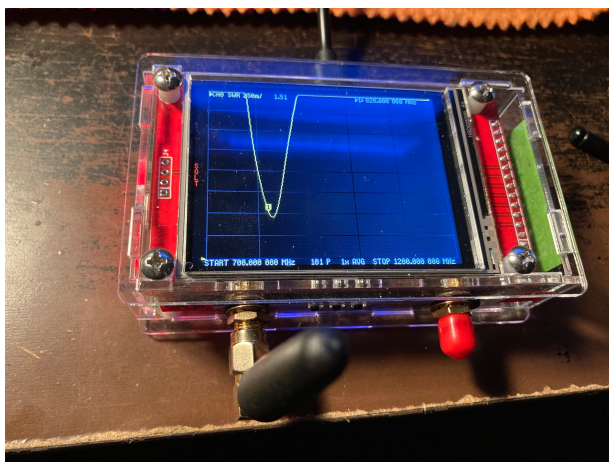
Antennas

Every HAM operator can tell you: better a cheap radio with a good antenna then an expensive radio with a unreliable antenna.

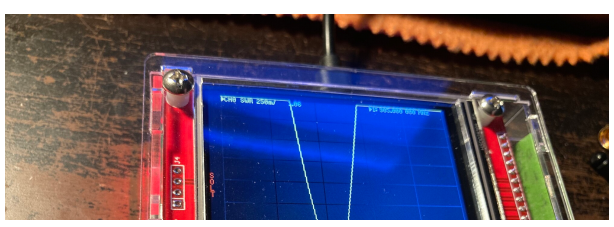
I have a inexpensive VNA that was an invaluable help to select the proper antenna for this project. The problems with the cheap rubber-ducky fall in two categories:

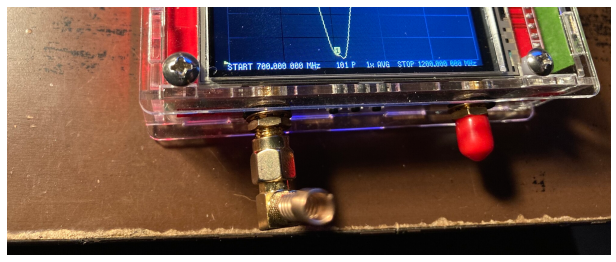
1. the antenna hasn't the declared resonance frequency because they wasn't well-designed ;
2. the antenna hasn't the declared resonance frequency because there are flaws in the construction technique.

To explain this point two, let see the following photos:



this cheap antenna, tested with the VNA , has optimal working frequencies non in the range declared by the seller, but the problem isn't in the design, it is more subtle, The real antenna is inside the rubber cap and has a form similar to a spring. The cap touche the antenna, compressing it and changing its electrical characteristics. So the problem is in that cap, as I demonstrate removing it:





Now, as you can see in this picture, without the cap the antenna is compliant with the declared specifications.

The following photo show a better antenna: the specifications are exactly as specified by the producer:

No alt text provided for this image

Because the complexity of this king of projects, the availability of some troubleshooting instruments , like a VNA and a spectrum analyzer are recommended .

Other Hardware and Software

- **Computers**: I used two old i5 computer with 8GB of ram each and let's say 60GB of free space on disk; the RAN emulator and the UE emulator were running directly on the host OS on both machines. No other heavy application was running on those machines at test time;
- **Cellphone:** an old LTE protocol compatible cellphone;
- **SIM Cards:** two Sysmocom test sim cards (one used directly in the cellphone, the keys of the second one used in the emulated UE, see here: http://shop.sysmocom.de/products/sysmoISIM-SJA2);
- **OS:** Ubuntu 18 LTS

Dependencies installation

- **Build Tools and Distribution Libraries**:

```
sudo apt install build-essential -y
sudo apt install git -y
sudo apt install cmake  -y
sudo apt install clang  -y
sudo apt install libpython-dev python-numpy swig
-y
sudo apt install libi2c-dev libusb-1.0-0-dev git
g++ cmake libsqlite3-dev libwxgtk3.0-dev
freeglut3-dev  -y
sudo apt install libfftw3-dev libmbedtls-dev
libboost-program-options-dev libconfig++-dev
libsctp-dev  -y
```

- **Photos**:

```
git clone https://github.com/pothosware/lms-rtl-
sdr.git
cd lms-rtl-sdr/
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Troubleshooting: when I cloned Photos from GitHub, a bug was present. I write this note in case the problem is still present:

```
librtlsdr.so.0.5git: undefined reference to
`LMS_EnableCalibCache'
collect2: error: ld returned 1 exit status
src/CMakeFiles/rtl_power.dir/build.make:97: recipe
for target 'src/rtl_power' failed
make[2]: *** [src/rtl_power] Error 1
```

To solve the problem I changed that symbol in:

```
librtlsdr.c
```

with this one:

```
LMS_EnableCache
```

- **SoapySDR:**

```
git clone https://github.com/pothosware
/SoapySDR.git
cd SoapySDR/
mkdir build
cd build
cmake ../
make
sudo make install
sudo ldconfig
```

- **LimeSuite:**

```
git clone https://github.com/myriadrf/LimeSuite.g
cd LimeSuite/
git checkout stable
mkdir builddir
cd builddir
cmake ../
make -j 2
sudo make install
```

```
sudo ldconfig
cd ../udev-rules
sudo bash install.sh
```

Drivers test

Let's test the driver functionality :

```
SoapySDRUtil --info
######################################################
##     Soapy SDR -- the SDR abstraction library
##
######################################################

Lib Version: v0.8.1-g6ff969ef
API Version: v0.8.0
ABI Version: v0.8
Install root: /usr/local
Search path:  /usr/local/lib/SoapySDR/modules0.8
(missing)
No modules found!
Available factories... No factories found!
Available converters...
 -  CF32 -> [CF32, CS16, CS8, CU16, CU8]
 -  CS16 -> [CF32, CS16, CS8, CU16, CU8]
 -  CS32 -> [CS32]
 -   CS8 -> [CF32, CS16, CS8, CU16, CU8]
 -  CU16 -> [CF32, CS16, CS8]
 -   CU8 -> [CF32, CS16, CS8]
 -   F32 -> [F32, S16, S8, U16, U8]
 -   S16 -> [F32, S16, S8, U16, U8]
 -   S32 -> [S32]
 -    S8 -> [F32, S16, S8, U16, U8]
 -   U16 -> [F32, S16, S8]
 -    U8 -> [F32, S16, S8]
```

LimeSDR Mini and USB troubles

Time to test the LimeSDR Mini, let's insert the device in a USB 3.0 port, these commands do the job:

```
lsus
... omissis ...
Bus 004 Device 002: ID 0403:601f Future Technology
Devices International, Ltd

lsusb -D /dev/usb/bus/004/002
... omissis ...
bcdUSB               3.10
... omissis ...
b
```

the "omissis" string indicate useless information I omitted.

LimeSDR mini requires a USB 3.0 (or more recent) controller. It seems to work with USB 2.0 but the data stream processing will fail, so is useless to try. Another source of trouble is the cable: during my first test the connection continuously downgraded to 2.0 protocol and the connections between RAN (Radio Access Network) and UE (User Equipment) was impossible.

So better test the negotiated protocol with the *SoapySDRUtil* utility:

```
LimeUtil --find
```

In the output you should have the following line, that specifies "USB 3.0" as connection protocol:

```
 * [LimeSDR Mini, media=USB 3.0, module=FT601,
addr=24607:1027, serial=1D53891D60912C]
```

With this procedure you will avoid lot of pain.

Launch this command to check if there are other problems:

```
SoapySDRUtil --probe
```

that also display full information about the device in use.

LimeSDR Firmware Updating

This command will update the LimeSDR firmware automatically downloading the last version available:

```
LimeUtil --update
```

srsLTE: Software installation

Finally, the core element, the cellphone network software:

```
git clone https://github.com/srsLTE/srsLTE.git
cd srsLTE
mkdir build
cd build
cmake ../
make -j 2
sudo make install
sudo ldconfig
srsran_install_configs.sh  user
```

- **Troubleshooting**: The version 21.04 I cloned from Github had a problem in this file:

  ```
  lib/src/phy/rf/rf_soapy_imp.c
  ```

  a missing prefix in a header file path produced missing symbols, I solved the problem with this change:

```
$ git diff lib/src/phy/rf/rf_soapy_imp.
diff --git a/lib/src/phy/rf/rf_soapy_imp.c
b/lib/src/phy/rf/rf_soapy_imp.c
index 4af516a8c..672ca83e7 100644
--- a/lib/src/phy/rf/rf_soapy_imp.c
+++ b/lib/src/phy/rf/rf_soapy_imp.c
@@ -33,7 +33,7 @@
 #include <SoapySDR/Logger.h>
 #include <SoapySDR/Time.h>
 #include <SoapySDR/Version.h>
-#include <Types.h>
+#include <SoapySDR/Types.h>c
```

So, if the problem persists, it's necessary to change this line:

```
#include <Types.h>
```

with this one:

```
#include <SoapySDR/Types.h>
```

srsLTE Software: eNode-B Configuration

On the eNode-B machine, these configuration steps are necessary:

- **enb.conf :** you can find this file in ~/.config/srsran/, I change the following parameters in **[rf]** section, I did no change to the other parameters present in that file :

```
[rf]

dl_earfcn = 6200
tx_gain = 80
rx_gain = 40

device_name = soapy
device_args = auto
time_adv_nsamples = 70
```

- Further explanation about dl_earfcn parameter is available in EARFCN paragraph. The parameter time_adv_nsamples is directly related to the hardware characteristics and could be necessary to change its value. The values assigned to tx_gain and rx_gain are specific of LimeSDR mini with a specific antenna type, so you could have necessity to change those values;

- **epc.conf:** this file is present in ~/.config/srsran/, too. I only changed the *user db file name, db_file* (section *[hss]*), then *APN* name and *DNS address* (parameter *apn* and *nds_addr* , section *[mme]*) from the default configuration, as reported in the following example:

```
[hss]

 db_file = user_db.csv

[mme]

 apn = bgapn
 dns_addr = 8.8.8.8
```

- **user_db.csv :** this is the same user db files specified in epc.conf (see previous point), it contains information to identify an user from his SIM card, for example the IMEI. This file also is present in ~/.config/srsran/:

```
ue3,mil,9017xxxxxxxxxx,41cac66efb4xxxxxxxxxxxxxxxxxxxx,opc,e7aaxxxxxxxxxxxxxxxxxxxxxxxxxxxx,9000,000000006
ue3,mil,901700000040919,f613aa817d0b6e490c6489c294fbc4cb,opc,4593xxxxxxxxxxxxxxxxxxxxxxxxxx,9000,000000006
```

I replaced part of the values in my file with some "x", this CSV file follows this format:

```
Name,Auth,IMSI,Key,OP_Type,OP/OPc,AMF,SQN,QCI,IP_alloc
```

This is the full explanation, taken from the sample file available in the software distribution:

```
# Name:     Human readable name to help
distinguish UE's. Ignored by the HSS
# Auth:     Authentication algorithm used by the
UE. Valid algorithms are XOR
#           (xor) and MILENAGE
(mil)

# IMSI:     UE's IMSI
value

# Key:      UE's key, where other keys are derived
from. Stored in hexadecimal
# OP_Type:  Operator's code type, either OP or
OPc
# OP/OPc:   Operator Code/Cyphered Operator Code,
stored in hexadecimal
# AMF:      Authentication management field,
stored in hexadecimal
# SQN:      UE's Sequence number for freshness of
the authentication
# QCI:      QoS Class Identifier for the UE's
default bearer.
# IP_alloc: IP allocation stratagy for the
SPGW.
#           With 'dynamic' the SPGW will
automatically allocate IPs

#           With a valid IPv4 (e.g. '172.16.0.2')
the UE will have a statically assigned IP.
#
```

EARFCN

EARFCN (E-UTRA Absolute Radio Frequency Channel Number) is a method of identifying the carrier frequencies to use (both uplink and downlink) with a single code represented by a integer in the range 0 – 65535. You can find on Internet online converter that, providing the EARFCN code, return the corresponding uplink and downlink frequencies. In the example here, 6200 is specified as EARFCN code, this is the translation of that code in frequencies:

No alt text provided for this image

OS Configuration and eNode-B Services Starting

Before starting the services, the CPU governor should be set to "performance" to enable maximum compute power and throughput for this application:

```
echo "performance" | sudo tee /sys/devices/system
/cpu/cpu*/cpufreq/scaling_governor
```

On the PC designed as eNode-B, must be started the srsLTE services and must be activated the ip forwarding, so first, using two different terminals, start epc and enb services, in the first terminal launch the command:

```
sudo srsepc
```

If all works fine, a new network device like the following should be present:

```
17: srs_spgw_sgi:
<POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500
qdisc fq_codel state UNKNOWN group default qlen 50
    link/none
    inet 172.16.0.1/24 scope global srs_spgw_sgi
       valid_lft forever preferred_lft forever
    inet6 fe80::ceeb:ed04:9297:f5e2/64 scope link
stable-privacy
       valid_lft forever preferred_lft forever0
```

then, in the second terminal launch the command:

```
sudo srsenb --expert.lte_sample_rates=true
```

Finally, let's start with ip forwarding (change wlp3s0 with the proper interface connected to the local router):

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
1>/dev/null
sudo iptables -t nat -A POSTROUTING -o wlp3s0 -j
MASQUERADE
```

Cellphone configuration

I suggest to perform this test before the one involving the emulated UE because, having an appliance working for sure, could be handing to troubleshoot errors.

- disable WIFI, so that the phone must use LTE to connect to the Internet;
- enable cellular data usage, if disabled. I also disabled data roaming and I choose LTE as preferred network type:

  No alt text provided for this image

- set as APN the one specified in **epc.conf:**

  No alt text provided for this image

  If all is configured properly, you should see "Software Radio System RAN" as provider and an LTE connection on cellphone display, as shown in the following image:

  No alt text provided for this image

  and in the srsepc log, you should find trace of the IP assigned to that phone:

  No alt text provided for this image

srsLTE Software: UE Configuration

On the UE PC, the following configuration steps are required:

- ue.conf: this file is present in ~/.config/srsran/, changes in the sections [rf], [rat.eutra], [usim], [nas]

```
[rf]
freq_offset = 0
tx_gain = 80
rx_gain = 40
time_adv_nsamples = 70
device_name = soapy
device_args = auto

[rat.eutra]
dl_earfcn = 6200

[usim]
mode = soft
algo = mil
opc  = 4593xxxxxxxxxxxxxxxxxxxxxxxxxxxx
k    = f613xxxxxxxxxxxxxxxxxxxxxxxxxxxx
imsi = 9017xxxxxxxxxx
imei = 3534xxxxxxxxxx

[nas]
apn = bgapn
```

```
apn_protocol = ipv4
```

Notes:

- **[rf]** has the same values for the parameters with the same name present in the [rf] section of **enb.conf** file on eNode-B machine;
- **[rat.eutra]**: the **dl_earfcn** parameter has the same value of the parameter with the same name present in [rf] section of **enb.conf** file on eNode-B machine;
- **[usim]:** contains the attributes of a valid SIM card;
- **[nas]:** in this section is configured the **APN**: must be the same present in **epc.conf** on eNode-B machine, **[mme]** section. The other parameter, apn_protocol , specifies that IPV4 will be used.
- As usual, part of the SIM attributes was censored , with some "x";

OS settings and UE Services Starting

As seen in eNode-B case, the CPU governor should be set to "performance":

```
echo "performance" | sudo tee /sys/devices/system
/cpu/cpu*/cpufreq/scaling_governor
```

This start the software emulated cellphone, so, using a terminal on the PC designed as UE, launch the following command:

```
sudo srsue --expert.lte_sample_rates=true
```

If all was configured without errors, a TUN interface will be created:

```
8: tun_srsue:
<POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500
qdisc fq_codel state UNKNOWN group default qlen 50
    link/none
    inet 172.16.0.3/24 scope global tun_srsue
       valid_lft forever preferred_lft forever0
```

Using the ping command from UE machine, it's possible to test the connectivity with the eNode-B:

```
$ ping 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
64 bytes from 172.16.0.1: icmp_seq=1 ttl=64
time=33.3 ms
64 bytes from 172.16.0.1: icmp_seq=2 ttl=64
time=24.0 ms
64 bytes from 172.16.0.1: icmp_seq=3 ttl=64
time=23.0 ms
64 bytes from 172.16.0.1: icmp_seq=5 ttl=64
time=36.6 ms
```

To route the traffic through the LTE interface, *use route* and *ip route* Linux command specifying as gateway the eNode-B address, example:

```
sudo ip route add default via 172.16.0.1
```

Now the ping should pass through the TUN interface, the radio link to the eNode-B, then routed to its default GW, etc.

This new, longer, route implies a longer average in round trips :

```
$ ping google.i
PING google.it (142.250.180.131) 56(84) bytes of
data.
64 bytes from mil04s43-in-f3.1e100.net
(142.250.180.131): icmp_seq=1 ttl=114 time=49.7 ms
64 bytes from mil04s43-in-f3.1e100.net
(142.250.180.131): icmp_seq=2 ttl=114 time=48.5 ms
64 bytes from mil04s43-in-f3.1e100.net
(142.250.180.131): icmp_seq=3 ttl=114 time=470 ms
64 bytes from mil04s43-in-f3.1e100.net
(142.250.180.131): icmp_seq=4 ttl=114 time=89.3 ms
64 bytes from mil04s43-in-f3.1e100.net
(142.250.180.131): icmp_seq=5
```

Conclusions

This is a toy cellphone infrastructure: with the configuration I shown no commercial service can be provided but, IMHO, it is a good laboratory environment to learn / do experiments with cellphone network. That is the reason I started this project, hoping it give me the opportunity to improve my knowledge, about both software and hardware side.Having time I would like to publish other articles, next one could be related to the troubleshooting. Other thing I would like to do is some software application. If you have suggestions, advises, applications ideas, materials or links to materials about this topic you think it is interesting or useful, let me know.