OpenBTS

Installation and Configuration Guide

Alexsander Loula
alex.loula@gmail.com
v0.1 (2009-05-25)

1. Introduction

This guide will give you a brief overview of the OpenBTS. The main goal of the OpenBTS project is present de GSM air interface to standard GSM handsets and uses the Asterisk PBX software to connect calls.

At the end of this guide you will be able to make call between GSM handsets and any kind of device compatible with Asterisk in your own network, in other words, without pay by the calls.

2. The GSM network

A GSM network is a complex system composed by several components. The last mile of this system is the BTS (Base Transceiver Station). The BTS is responsible to transmit and receive the RF (Radio Frequency) signals to the user terminal (cell phone, PDA, modem, etc). The BTS's are controlled by a BSC (Base Station Controller) that is connected to the MSC/VLR (Mobile Switching Center/Visitor Location Register). Basically, the MSC/VLR is responsible to authenticate the user against the database (HLR - Home Location Register, AuC - Authentication Center).

Below is a picture of the key elements of a GSM network:

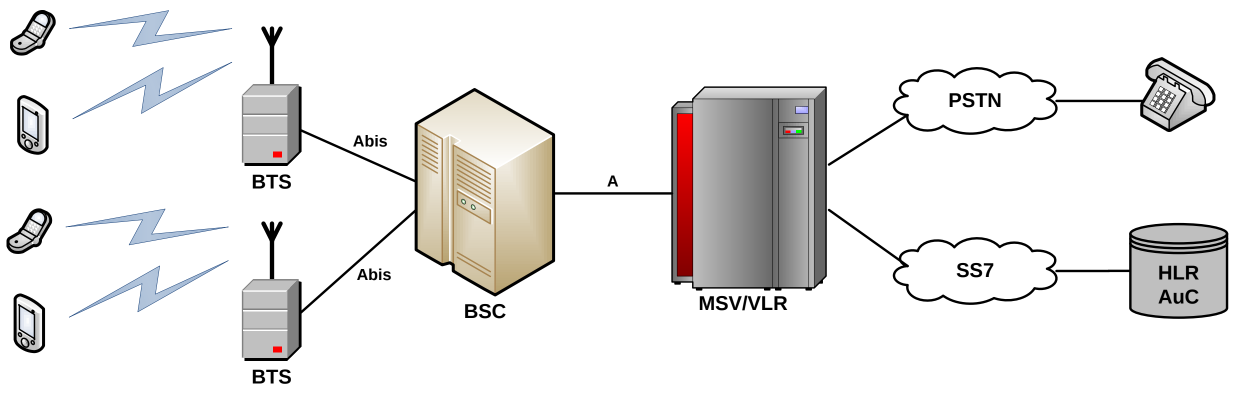


Figure 1 - Key elements of a GSM network

3. The OpenBTS project

The OpenBTS (Base Transceiver Station) project is an effort to construct an open-source Unix application that uses the Universal Software Radio Peripheral (USRP) to present a GSM air interface ("Um") to standard GSM handsets and uses the Asterisk software PBX to connect calls.

The OpenBTS uses the USRP hardware to receive and transmit the GSM signaling. This is done by using the GNU Radio framework. The Asterisk is used to interface the GSM calls between the cellular phones under the OpenBTS network. Any other device that can be connected to the Asterisk can be also used.

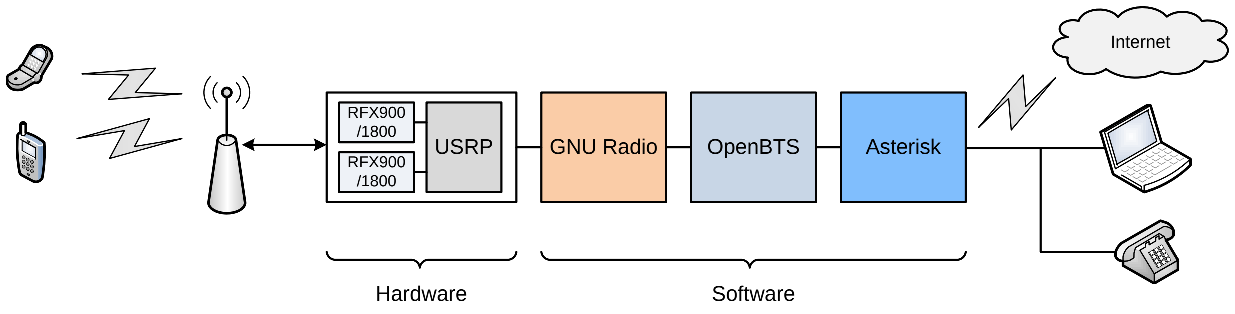


Figure 2 - System overview

The GNU Radio is a free software development toolkit that provides the signal processing runtime and processing blocks to implement software radios using readily-available, low-cost external RF hardware (in this case the USRP).

The USRP (Universal Software Radio Peripheral) is a hardware designed by Ettus Research to allow general purpose computers to function as high bandwidth software radios. In essence, it serves as a digital baseband and IF section of a radio communication system. There are several daughterboard's that can be used with the USRP covering from DC to 5.9 GHz. In our case we can use the RFX900, to cover the GSM 850 and 900 bands, or the RFX1800, to cover the GSM 1800 and 1900 bands.

Daughterboard	RFX900	RFX 1800
Frequency Range	750 to 1050 MHz	1.5 to 2.1 GHz
Transmit Power	200mW (23dBm)	100mW (20dBm)

4. Requirements

This guide will cover the installation on a GNU/Linux machine. It's highly recommended to follow the software and hardware requirements covered on this guide.

Hardware:

- 01 - Computer (Core 2 Duo 2.0 GHz, 2GB RAM, USB port);
- 01 - USRP-PKG (USRP Package, includes Motherboard, Enclosure, 2 RF Cables, USB Cable, Power Supply, and Hardware Package – USD 700);
- 02 - RFX900 for GSM 850/900 (800-1000MHz Transceiver, 200 mW output – USD 275 each);
- 02 - RFX1800 for GSM 1800/1900 (1.5-2.1 GHz Transceiver, 100 mW output – USD 275 each);
- 02 - VERT900 (824-960 MHz, 1710-1990 MHz Quad-band Cellular/PCS and ISM Band Vertical Antenna, 3dBi Gain, 9 Inches, Ideal for RFX900 and RFX1800).
- 01 - Unlocked cellular phone;
- 01 - SIM Card (preferred for those with possibility to edit network list).

All hardware items except the computer, cell phone and SIM Card can be found directly from Ettus Research.

Software:

- GNU/Linux - Ubuntu 8.04 - 32 bits;
- OpenBTS 2.3;
- GNURadio 3.1.3;
- C++ Boost 1.37.

5. GNU Radio installation

a. Installing the dependencies:

```
sudo apt-get update
sudo apt-get -y install swig g++ automake1.9 libtool python-dev \
libcppunit-dev sdcc libusb-dev libasound2-dev libsdl1.2-dev \
python-wxgtk2.8 subversion guile-1.8-dev libqt4-dev \
ccache python-opengl libgsl0-dev python-cheetah python-lxml \
libqwt5-qt4-dev libqwtplot3d-qt4-dev qt4-dev-tools \
fftw3-dev doxygen python-numpy-ext
```

b. Getting and installing boost libraries:

```
wget http://kent.dl.sourceforge.net/sourceforge/boost/boost_1_37_0.tar.gz
tar xvzf boost_1_37_0.tar.gz
cd boost_1_37_0
BOOST_PREFIX=/opt/boost_1_37_0
./configure --prefix=$BOOST_PREFIX --with-
libraries=thread,date_time,program_options
make
sudo make install
```

c. Getting and installing GNURadio:

```
cd
wget ftp://ftp.gnu.org/gnu/gnuradio/gnuradio-3.1.3.tar.gz
tar xvzf gnuradio-3.1.3.tar.gz
cd gnuradio-3.1.3
./configure --with-boost-include-dir=$BOOST_PREFIX/include/boost-1_37/
make
sudo make install
sudo ldconfig
```

d. Adding user permissions to work with the USRP:

```
sudo addgroup usrp
sudo addgroup <YOUR_USER> usrp
echo 'ACTION=="add", BUS=="usb", SYSFS{idVendor}=="fffe",
SYSFS{idProduct}=="0002", GROUP:="usrp", MODE:="0660"' > tmpfile
sudo chown root.root tmpfile
sudo mv tmpfile /etc/udev/rules.d/10-usrp.rules
```

e. Testing the USRP:

- Restart the computer (it should work without it, but even restarting the udev service, the USRP worked with user privileges only by restarting the machine):

```
sudo reboot
```

- Connect the USRP to the USB port

```
cd /usr/local/share/gnuradio/examples/usrp/
./usrp_benchmark_usb.py
Testing 2MB/sec... usb_throughput = 2M
ntotal    = 1000000
nright    = 998435
runlength = 998435
```

```
delta     = 1565
OK
Testing 4MB/sec... usb_throughput = 4M
ntotal    = 2000000
nright    = 1998041
runlength = 1998041
delta     = 1959
OK
Testing 8MB/sec... usb_throughput = 8M
ntotal    = 4000000
nright    = 3999272
runlength = 3999272
delta     = 728
OK
Testing 16MB/sec... usb_throughput = 16M
ntotal    = 8000000
nright    = 7992153
runlength = 7992153
delta     = 7847
OK
Testing 32MB/sec... usb_throughput = 32M
ntotal    = 16000000
nright    = 15986239
runlength = 15986239
delta     = 13761
OK
Max USB/USRP throughput = 32MB/sec
```

- Check if the maximum throughput between USB and USRP is 32MB/sec.

6. OpenBTS installation and settings

a. Installing the dependencies:

```
cd
sudo apt-get install asterisk libosip2-dev libortp7-*
```

b. Getting the source code:

```
sftp openbts@kestrelsp.com
password: "wd9xcv!"
sftp> get openbts-2.3JeanLafitteOE.tar.gz
Fetching /Users/openbts/openbts-2.3JeanLafitteOE.tar.gz to openbts-
2.3JeanLafitteOE.tar.gz
exit
```

c. Installing:

```
tar xvzf openbts-2.3JeanLafitteOE.tar.gz
mv openbts-2.3JeanLafitte openbts-2.3
cd openbts-2.3
export LIBS=-lpthread
./configure
make
sudo make install
```

d. Configuring the settings:

This is a very important step. After getting everything compiled, it's time to configure the OpenBTS. Open the *apps/OpenBTS.config* with you preferred editor:

- The GSM.MCC (Mobile Country Code) can be set according to your country. In my case is 724 (Brazil). A complete table with these codes can be found here:

  http://en.wikipedia.org/wiki/Mobile_country_code

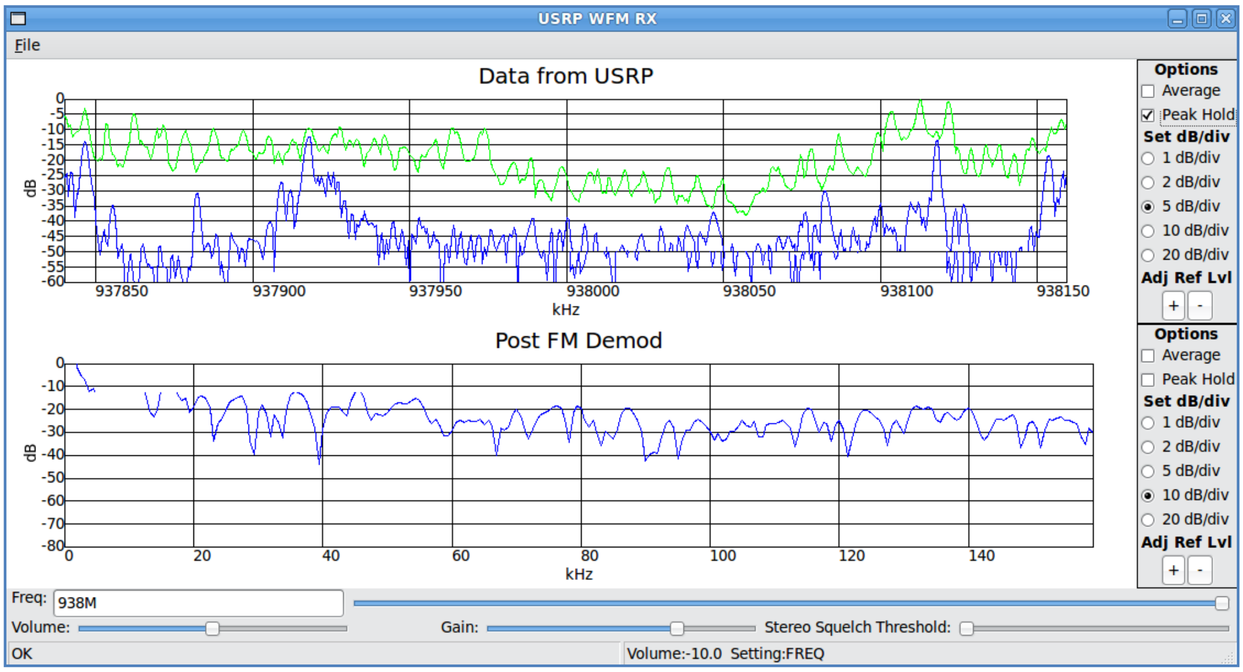
- The GSM.MNC (Mobile Network Code) must be any code between 0 and 99 since it's not used by a local operator. A good way to check it is by scanning the network with the phone and checks the operator's code. Normally it'll be showed in the MCC-MNC format (e.g. 724-05). This means that the country is Brazil and network code is 05.

- The GSM.Band defines the frequency band that the OpenBTS will operate. The best is to use a band not allocated in your region, but sometimes this is not possible. If it's your case, you'll need to check using a Spectrum Analyzer, what is the band has a free space. This link shows the frequency and channel allocation by the GSM bands (NOTE: Downlink is the frequency that the BTS transmits, so that's the one's we need to care about):

  http://en.wikipedia.org/wiki/GSM_frequency_ranges

- Since this is a low cost project, it's very probable that you won't have a Spectrum Analyzer that is an expensive test instrument. The good news is that GNURadio has a simple one's, which can be used to check the band and channel allocations. To use it, go to GNURadio examples folder and execute the *usrp_wfm_rcv_pll.py*:

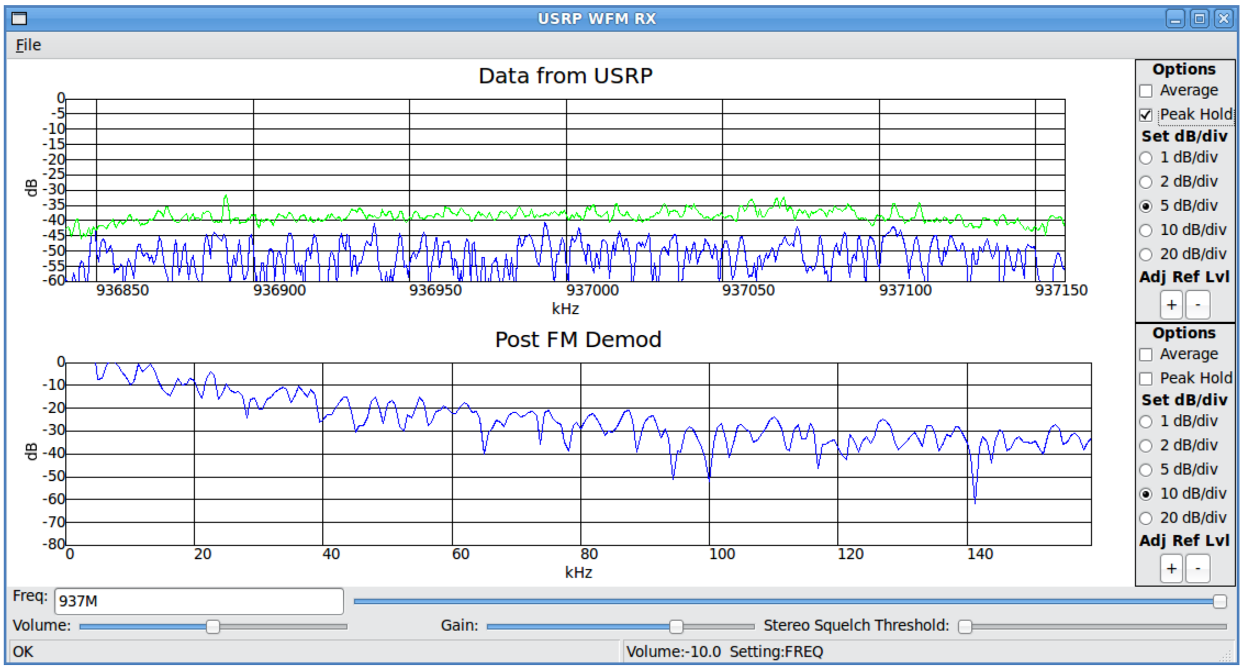
```
cd /usr/local/share/gnuradio/examples/usrp
./usrp_wfm_rcv_pll.py
```



Freq: 938M
Volume:
Gain:
Stereo Squelch Threshold:
OK
Volume:-10.0 Setting:FREQ

Figure 3 - USRP WFM RX analyzing between 937.85 and 938.15 MHz

- By changing the frequency ("Freq:"), we can scan the GSM bands to check what is the best place to operate with the OpenBTS. The figure above shows that the frequencies near of 938 MHz are being used. The "Peak Hold" option can be useful here.
- On the screen bellow we can see that the frequencies near of 937 MHz are not used.



Upper options: Average [ ], [✓ Peak Hold]; Set dB/div: [✓ 5 dB/div]

Lower options: Average [ ], Peak Hold [ ]; Set dB/div: [✓ 10 dB/div]

Freq: 937M

Volume:

Gain:

Stereo Squelch Threshold:

OK

Volume:-10.0 Setting:FREQ

Figure 4 - USRP WFM RX analyzing between 936.85 and 937.15 MHz

- Using Spectrum Analyzer test equipment, we can see the full range of GSM 900 band, from 925 to 960 MHz downlink spectrum. The spikes are the frequencies (carriers) used by the operator:

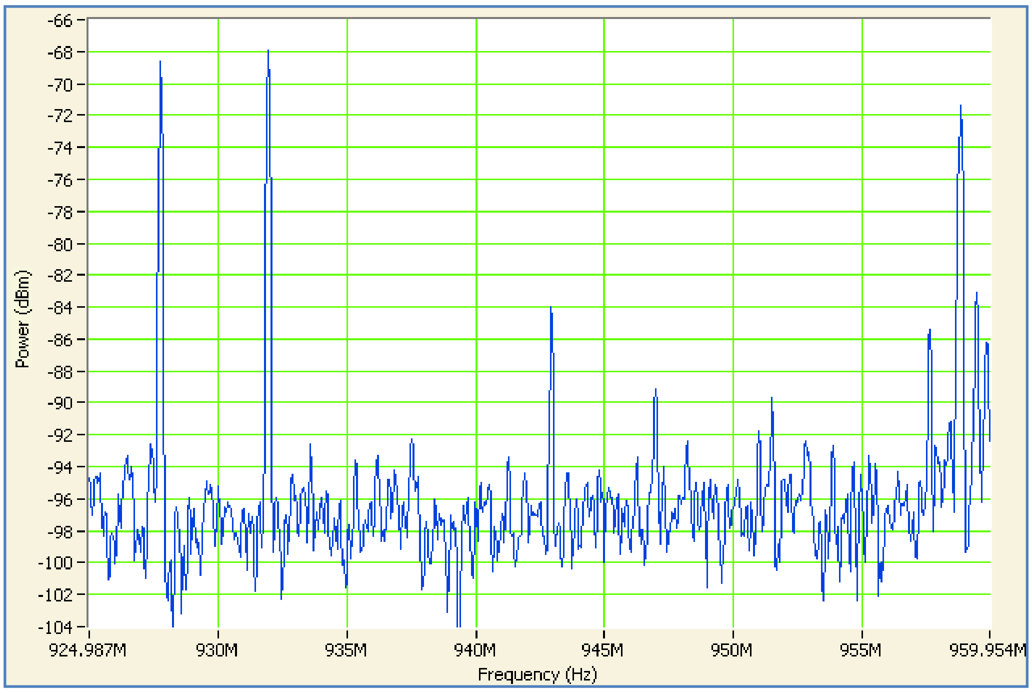


Figure 5 - GSM 900 band downlink spectrum

- Now we can set the GSM.ARFCN (Absolute RF Channel). This WEB application generates the ARFCN tables for the 4 GSM bands:

  http://www.aubraux.com/design/arfcn-calculator.php

- For the GSM 900, that is the chosen band, the table is the following:

ARFCN	Frequency (MHz)	ARFCN	Frequency (MHz)
1	935.2	2	935.4
3	935.6	4	935.8
5	936	6	936.2
7	936.4	8	936.6
9	936.8	10	937
11	937.2	12	937.4
13	937.6	14	937.8

- For the 937 MHz downlink frequency (BTS to the cellular phone), the ARFCN is 10.
- Below is my *OpenBTS.config* file (my modifications are in grey):
-

```
# Sample OpenBTS configuration file.
# Format of each line is. <key><space><value>
# The key name can contain no spaces.
# Everything between the first space and the end of the line becomes the
value.
# Comments must start with "#" at the beginning of the line.
# Blank lines are OK.

# As a gerenal rule, non-valid configuration values will crash OpenBTS.


#
# Logging parameters
#

# The initial global logging level: ERROR, WARNING, NOTICE, INFO, DEBUG,
DEEPDEBUG
LogLevel INFO

# The log file path.  If not set, logging goes to stdout.
# LogFileName test.out


#
# Transceiver parameters
#

# Transceiver interface
# This TRX.IP is not really adjustable.  Just leave it as 127.0.0.1.
TRX.IP 127.0.0.1
# This value is hard-coded in the transcevier.  Just leave it alone.
TRX.Port 5700

# Path to transceiver binary
```

```
TRX.Path ../Transceiver/transceiver

# TRX logging.
# Logging level.
TRX.LogLevel ERROR
# Logging file.  If not defined, logs to stdout.
# TRX.LogFileName test.out




#
# SIP, RTP, servers
#

# Asterisk PBX
Asterisk.IP 127.0.0.1
Asterisk.Port 5060

# Messaging server
Messenger.IP 127.0.0.1
Messenger.Port 5063

# Local SIP/RTP ports
SIP.Port 5062
RTP.Start 16484
RTP.Range 98

# Local SMS port for short code delivery.
SMSLoopback.Port 5064


#
# Special extensions.
#

# Routing extension for emergency calls.
# PBX.Emergency 2101


#
# SIP parameters
#

# SIP registration period in seconds.
# Ideally, this should be slightly longer than GSM.T3212.
SIP.RegistrationPeriod 3600

#
# SIP Internal Timers.  All timer values are given in millseconds.
# These are from RFC-3261 Table A.
#

# SIP Timer A, the INVITE retry period, RFC-3261 Section 17.1.1.2
SIP.Timer.A 1000



#
# SMS parameters
#
```

```
# ISDN address of source SMSC when we fake out a source SMSC.
SMS.FakeSrcSMSC 0000
# ISDN address of destination SMSC when a fake value is needed.
SMS.DefaultDestSMSC 0000


# The SMS HTTP gateway.
# Comment out if you don't have one.
# SMS.HTTP.Gateway api.clickatell.com

# IF SMS.HTTP.Gateway IS DEFINED, SMS.HTTP.AccessString MUST ALSO BE
DEFINED.
# SMS.HTTP.AccessString sendmsg?user=xxxx&password=xxxx&api_id=xxxx


# The "Welcome Message" is sent to uprovisioned handsets that try to
register.
# Comment out if you don't want this feature.
# WELCOME MESSAGE MUST BE LESS THAN 161 CHARACTERS.
# SMS.WelcomeMessage Welcome to OpenBTS
# SMS.WelcomeMessage Your handset attempted to register with OpenBTS.

# IF SMS.WelcomeMessage IS DEFINED, SMS.WelcomeShortCode MUST ALSO BE
DEFINED.
SMS.WelcomeShortCode 0000


#
# GSM
#

# Network and cell identity.

# Network Color Code, 0-7
GSM.NCC 0
# Basesation Color Code, 0-7
GSM.BCC 0
# Mobile Country Code, 3 digits.
# MCC MUST BE 3 DIGITS.  Prefix with 0s if needed.
GSM.MCC 724
# Mobile Network Code, 2 or 3 digits.
GSM.MNC 66
# Location Area Code, 0-65535
GSM.LAC 667
# Cell ID, 0-65535
GSM.CI 0
# Network "short name" to display on the handset.
# SHORT NAME MUST BE LESS THAN 8 CHARACTERS.
GSM.ShortName OpenBTS

# Assignment type for call setup.
# This is defined in an enum AssignmentType in GSMCommon.h.
# 0=Early, 1=VeryEarly.
GSM.AssignmentType 1

# Band and Frequency

# Valid band values are 850, 900, 1800, 1900.
GSM.Band 900
# Valid ARFCN range depends on the band.
#GSM.ARFCN 29
```

```
GSM.ARFCN 10

# Downlink tx power level, dB wrt full power
GSM.PowerAttenDB 0

# Beacon parameters.

# L1 radio link timeout advertised on BCCH.
# This is the RAW parameter sent on the BCCH.
# See GSM 10.5.2.3 for encoding.
# Value of 15 gives 64-frame timeout, about 30 seconds on the TCH.
# This should be coordinated with T3109.
GSM.RADIO_LINK_TIMEOUT 15

# Attach/detach flag.
# Set to 1 to use attach/detach procedure, 0 otherwise.
# This will make initial registration more prompt.
# It will also cause an un-regstration if the handset powers off.
GSM.ATT 1

# CCCH_CONF
# See GSM 10.5.2.11 for encoding.
# Value of 1 means we are using a C-V beacon.
GSM.CCCH_CONF 1

# Maximum RACH retransmission attempts
# This is the RAW parameter sent on the BCCH.
# See GSM 04.08 10.5.2.29 for encoding.
GSM.RACH.MaxRetrans 3

# Parameter to spread RACH busts over time.
# This is the RAW parameter sent on the BCCH.
# See GSM 04.08 10.5.2.29 for encoding.
GSM.RACH.TxInteger 14

# Access class flags.
# This is the RAW parameter sent on the BCCH.
# See GSM 04.08 10.5.2.29 for encoding.
# Set to 0 to allow full access.
GSM.RACH.AC 0




#
# GSM Timers.  All timer values are given in milliseconds unless stated
otherwise.
# These come from GSM 04.08 11.2.
#

# T3212, registration timer.
# Unlike most timers, this is given in MINUTES.
# Actual period will be rounded down to a multiple of 6 minutes.
# Any value below 6 minutes disables periodic registration.
# Ideally, this should be slightly less than the SIP.RegistrationPeriod.
GSM.T3212 6
```

7. Asterisk settings

a. Getting SIM Card IMSI (International Mobile Subscriber Identity):

- The phone registration is based on the IMSI number stored in the SIM Card. If you don't have this number, it´s possible to use this Python script to do it. Create a new file in your preferred text editor and paste the script on it. Please take care of indentation, this is important for Python.

```
#!/usr/bin/env python
# Coded by Alexsander Loula
# Email: alex.loula@gmail.com

import serial,string

def readuntilok(s):
    ol=[]
    while 1:
        c=s.read()
        if not c:
            break
        ol.append(c)
        ostring="".join(ol)
        if len(ol)>3 and ostring[-4:]=="OK\r\n":
            break
    return ostring

def cmd(s,cmd):
    s.write(cmd+"\r")
    r=readuntilok(s)
    r=r.split("\n")
    for i in range(len(r)):
        r[i]=r[i][:-1]
    return r

def cota(s,cmd):
    s.write(cmd+"\r")
    r=readuntilok(s)
    r=r.replace('"','')
    r=r.split("\n")
    for i in range(len(r)):
        r[i]=r[i][:-1]
    return r

### INIT Serial Port
ser=serial.Serial('/dev/ttyACM0',115200,timeout=3)
ser.write('ATZ\r')
line=ser.read(10)

### Read IMSI
imsi = cmd(ser,'AT+CIMI')[1]
imsi = imsi.split()[-1]
imsi = 'IMSI: ' + imsi[1:16]
print imsi

### Close Serial Port
ser.close()
```

- Save the file as *getimsi.py*;
- This script depends of the Python serial module to control the phone over serial (RS-232 or USB) through AT commands. To install it type:

```
sudo apt-get install python-serial
```

- Make the file executable:

```
sudo chmod +x getimsi.py
```

- Connect a phone with AT commands through serial port capabilities and run the script:

```
./getimsi.py
```

- It must output something like:

```
IMSI: 724311320422052
```

- This number will be used to configure the Asterisk

b. Backup the */etc/extensions.conf* and */etc/sip.conf*:

```
cd /etc/asterisk
sudo cp extensions.conf extensions.conf_ori
sudo cp sip.conf sip.conf_ori
```

c. Copy *~/openbts-2.3/AsteriskConfig/extensions.conf* and *sip.conf* to the */etc/asterisk*:

```
sudo cp ~/openbts-2.3/AsteriskConfig/sip.conf .
sudo cp ~/openbts-2.3/AsteriskConfig/extensions.conf .
```

d. Edit the */etc/asterisk/extension.conf*:

```
...

[sip-local]
; local extensions
exten => 2100,1,Macro(dialSIP,wiredPhone)
exten => 2101,1,Macro(dialSIP,softPhone)
; This is a simple mapping between extensions and IMSIs.
exten => 2102,1,Macro(dialSIP,724311320422052)

...
```

- The extension 2101 will be used by the soft phone and the 2102 by the cell phone (724311320422052 is the IMSI).

e. Edit the */etc/asterisk/sip.conf*:

```
...

[softPhone]
callerid=2101
canreinvite=no
type=friend
context=sip-external
allow=ulaw
allow=gsm
```

```
host=dynamic

; This is a GSM handset entry.
; You need one for each SIM.
; The IMSI is a 15-digit code in the SIM.
; You can see it in the Control log whenever a phone tries to register.
[724311320422052] ; <- The IMSI is used as a SIP user ID.
canreinvite=no
type=friend
context=sip-external
allow=gsm
host=dynamic

...
```

f. Restart the Asterisk:

```
sudo /etc/init.d/asterisk restart
```

8. Testing the OpenBTS

a. Setting the phone:

- The phone settings are crucial for test OpenBTS. Firstly you need to make sure the GSM network setting is accord to the band you have selected on *OpenBTS.config* file.
- An operator SIM Card comes with a preferred network list. This is also called PLMN list (Public Land Mobile Network) that's composed by the Mobile Country Code and the Mobile Network Code (MCC-MNC).
- To guarantee that the phone will start scanning our network, you have to put your PLMN as the first preferred network.
- When the phone is registered on a network it gets a TMSI (Temporary Mobile Subscriber Identity). TMSI is randomly assigned by the VLR to every mobile in the area. It's good to clear the TMSI to get the first registration on the OpenBTS. A way to do it is to turn off the phone and take off the battery.

b. Execute OpenBTS

```
cd ~/openbts-2.3/apps
./OpenBTS

OpenBTS, Copyright 2008, 2009 Free Software Foundation, Inc.
Contributors:
      Kestrel Signal Processing, Inc.:
            David Burgess, Harvind Samra, Raffi Sevlian, Roshan Baliga
      GNU Radio:
            Johnathan Corgan
      Incorporated GPL libraries and components:
            libosip2, liportp2
This program comes with ABSOLUTELY NO WARRANTY.
This is free software;
you are welcome to redistribute it under the terms of GPLv3.
```

```
Use of this software may be subject to other legal restrictions,
including patent licsensing and radio spectrum licensing.
All users of this software are expected to comply with
applicable regulations.


1242936099.801724 3082733248:
Starting the system...
1242936100.8228 WARNING 3082733248 TRXManager.cpp:269: retrying
transceiver command after response timeout
1242936101.0380 INFO 3052067728 RadioResource.cpp:366: Pager::pageAll
paging 0 mobile(s)
1242936101.0382 INFO 3082733248 OpenBTS.cpp:199: system ready
1242936101.038262 3082733248:
Welcome to OpenBTS.  Type "help" to see available commands.

OpenBTS>
```

- If you have access to a Spectrum Analyzer, you should see a waveform (carrier) like this:

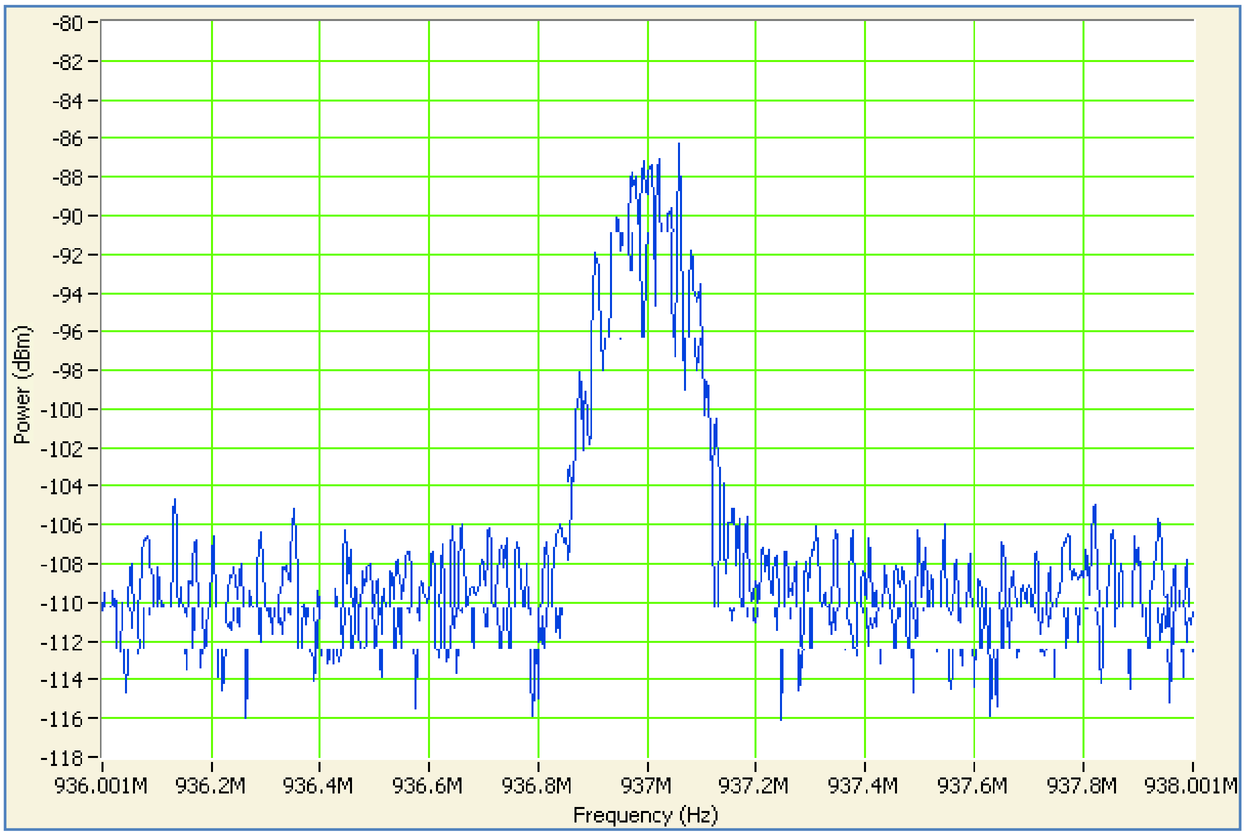


Figure 6 - Waveform of USRP generating the GSM signal

c. Turn on your phone

- You should see something like this on the OpenBTS CLI (Command Line Interface) at the phone registration:

```
OpenBTS> 1242936116.4806 INFO 3073633168 RadioResource.cpp:150:
AccessGrantResponder RA=0x18 when=0:1778916 age=24
```

```
1242936116.4809 INFO 3073633168 RadioResource.cpp:191:
AccessGrantResponder sending PageMode=(0) DedicatedModeOrTBF=(TMA=0
Downlink=0 DMOrTBF=0) ChannelDescription=(typeAndOffset=SDCCH/4-1 TN=0
TSC=0 ARFCN=10) RequestReference=(RA=24 T1'=29 T2=22 T3=36)
TimingAdvance=0
1242936116.9887 INFO 3069905808 MobilityManagement.cpp:117:
LocationUpdatingController MM Location Updating Request LAI=(MCC=724
MNC=31 LAC=0x3eee) MobileIdentity=(IMSI=724311320422052)
1242936116.9911 INFO 3069905808 MobilityManagement.cpp:170:
LocationUpdatingController registration SUCCESS: IMSI=724311320422052
1242936133.4187 INFO 3073633168 RadioResource.cpp:150:
AccessGrantResponder RA=0xe7 when=0:1782585 age=24
1242936133.4189 INFO 3073633168 RadioResource.cpp:191:
AccessGrantResponder sending PageMode=(0) DedicatedModeOrTBF=(TMA=0
Downlink=0 DMOrTBF=0) ChannelDescription=(typeAndOffset=SDCCH/4-3 TN=0
TSC=0 ARFCN=10) RequestReference=(RA=231 T1'=0 T2=25 T3=33)
TimingAdvance=0
1242936133.7486 INFO 3066710928 MobilityManagement.cpp:59:
CMServiceResponder MM CM Service Request serviceType=MOC
mobileIdentity=(TMSI=0x4a15b323)
```

d. Testing the MT (Mobile Terminate) call

- You can use a SIP softphone to do this test. For GNU/Linux, I recommend the Twinkle. You can install by typing:

```
sudo apt-get install twinkle
```

- Configure the Twinkle to register on Asterisk and call the cellular phone (number 2102 - defined in Asterisk settings).
- The phone should start ringing and the OpenBTS CLI will give you a output like this:

```
1242936133.7487 INFO 3066710928 CallControl.cpp:556: MOC: MM CM Service
Request serviceType=MOC mobileIdentity=(TMSI=0x4a15b323)
1242936134.2190 INFO 3066710928 CallControl.cpp:615: MOC: CC Setup
TI=(0,0) CalledPartyBCDNumber=(type=unknown plan=E.164/ISDN digits=2101)
1242936134.2193 INFO 3066710928 CallControl.cpp:179: assignTCHF sending
AssignmentCommand for 0xbff5e798 on 0xbff5e9b4
1242936134.9155 INFO 3057658768 RadioResource.cpp:276:
AssignmentCompleteHandler service=MOC
1242936134.9156 INFO 3057658768 CallControl.cpp:697: MOC: transaction:
ID=1804289383 TI=(0,0) IMSI=724311320422052 to=2101 Q.931State=MOC
initiated SIPState=Starting
1242936134.9498 INFO 3057658768 CallControl.cpp:715: MOC A: wait for
Ringing or OK
1242936135.1865 INFO 3057658768 CallControl.cpp:715: MOC A: wait for
Ringing or OK
1242936135.1866 INFO 3057658768 CallControl.cpp:726: MOC A: SIP:Ringing,
send Alerting and move on
1242936135.4046 INFO 3057658768 CallControl.cpp:756: MOC: wait for SIP
OKAY
1242936136.7268 INFO 3057658768 CallControl.cpp:793: MOC: sending Connect
to handset
1242936136.9539 INFO 3057658768 CallControl.cpp:538: MOC MTC connected,
entering callManagementLoop
```

- Answer the call on the cell phone and start the conversation.

e. Testing the MO (Mobile Originate) call

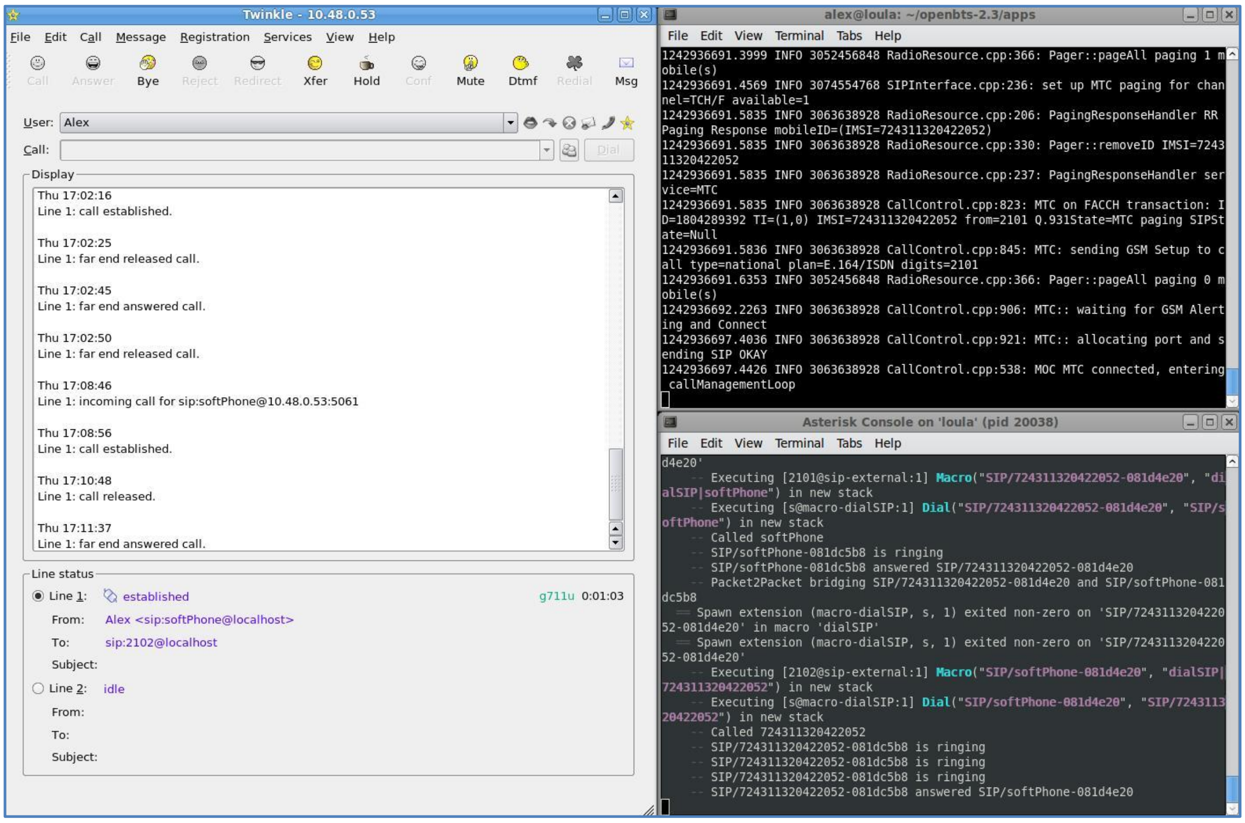
- On the cell phone call the softphone (number 2101 - defined in the Asterisk settings).
- The phone softphone should start ringing and the OpenBTS CLI will give you a output like this:

```
1242936158.0970 INFO 3063249808 RadioResource.cpp:330: Pager::removeID
IMSI=724311320422052
1242936158.0970 INFO 3063249808 RadioResource.cpp:237:
PagingResponseHandler service=MTC
1242936158.0970 INFO 3063249808 CallControl.cpp:823: MTC on FACCH
transaction: ID=1804289386 TI=(1,0) IMSI=724311320422052 from=2101
Q.931State=MTC paging SIPState=Null
1242936158.0970 INFO 3063249808 CallControl.cpp:845: MTC: sending GSM
Setup to call type=national plan=E.164/ISDN digits=2101
1242936158.2269 INFO 3052067728 RadioResource.cpp:366: Pager::pageAll
paging 0 mobile(s)
1242936158.7152 INFO 3063249808 CallControl.cpp:906: MTC:: waiting for
GSM Alerting and Connect
1242936165.5375 INFO 3063249808 CallControl.cpp:921: MTC:: allocating
port and sending SIP OKAY
1242936165.5728 INFO 3063249808 CallControl.cpp:538: MOC MTC connected,
entering callManagementLoop
```

- Answer the call on the cell phone and start the conversation.
- You can see the OpenBTS command options by typing ‘help’ on the CLI:

```
OpenBTS> help
assignment [type] -- get/set assignment type (early, veryearly)
calls -- print the transaction table
exit -- exit the application.
help -- list available commands or gets help on a specific command.
lai [MCC] [MNC] [hex-LAC] -- get/set location area identity (MCC, MNC,
LAC)
load -- print the current activity loads.
loglevel [level] -- get/set the logging level, one of {ERROR, ALARM,
WARN, NOICE, INFO, DEBUG, DEEPDEBUG}.
sendsms <IMSI> <src> -- send SMS to <IMSI>, addressed from <src>, after
prompting.
setlogfile <path> -- set the logging file to <path>.
tmsis ["clear"] -- print/clear the TMSI table.
uptime -- show BTS uptime and BTS frame number.
```

This is a screenshot from the OpenBTS, Asterisk and Twinkle on o MO call:


Twinkle - 10.48.0.53
File Edit Call Message Registration Services View Help
Call Answer Bye Reject Redirect Xfer Hold Conf Mute Dtmf Redial Msg
Dial
alex@loula: ~/openbts-2.3/apps
File Edit View Terminal Tabs Help
Asterisk Console on 'loula' (pid 20038)
File Edit View Terminal Tabs Help

User: Alex

Call:

Display

```
Thu 17:02:16
Line 1: call established.

Thu 17:02:25
Line 1: far end released call.

Thu 17:02:45
Line 1: far end answered call.

Thu 17:02:50
Line 1: far end released call.

Thu 17:08:46
Line 1: incoming call for sip:softPhone@10.48.0.53:5061

Thu 17:08:56
Line 1: call established.

Thu 17:10:48
Line 1: call released.

Thu 17:11:37
Line 1: far end answered call.
```

Line status

```
● Line 1:   established                    g711u 0:01:03
  From:     Alex <sip:softPhone@localhost>
  To:       sip:2102@localhost
  Subject:
○ Line 2:   idle
  From:
  To:
  Subject:
```

```
1242936691.3999 INFO 3052456848 RadioResource.cpp:366: Pager::pageAll paging 1 m
obile(s)
1242936691.4569 INFO 3074554768 SIPInterface.cpp:236: set up MTC paging for chan
nel=TCH/F available=1
1242936691.5835 INFO 3063638928 RadioResource.cpp:206: PagingResponseHandler RR
Paging Response mobileID=(IMSI=724311320422052)
1242936691.5835 INFO 3063638928 RadioResource.cpp:330: Pager::removeID IMSI=7243
11320422052
1242936691.5835 INFO 3063638928 RadioResource.cpp:237: PagingResponseHandler ser
vice=MTC
1242936691.5835 INFO 3063638928 CallControl.cpp:823: MTC on FACCH transaction: I
D=1804289392 TI=(1,0) IMSI=724311320422052 from=2101 Q.931State=MTC paging SIPSt
ate=Null
1242936691.5836 INFO 3063638928 CallControl.cpp:845: MTC: sending GSM Setup to c
all type=national plan=E.164/ISDN digits=2101
1242936691.6353 INFO 3052456848 RadioResource.cpp:366: Pager::pageAll paging 0 m
obile(s)
1242936692.2263 INFO 3063638928 CallControl.cpp:906: MTC:: waiting for GSM Alert
ing and Connect
1242936697.4036 INFO 3063638928 CallControl.cpp:921: MTC:: allocating port and s
ending SIP OKAY
1242936697.4426 INFO 3063638928 CallControl.cpp:538: MOC MTC connected, entering
 callManagementLoop
```

```
d4e20'
    -- Executing [2101@sip-external:1] Macro("SIP/724311320422052-081d4e20", "di
alSIP|softPhone") in new stack
    -- Executing [s@macro-dialSIP:1] Dial("SIP/724311320422052-081d4e20", "SIP/s
oftPhone") in new stack
    -- Called softPhone
    -- SIP/softPhone-081dc5b8 is ringing
    -- SIP/softPhone-081dc5b8 answered SIP/724311320422052-081d4e20
    -- Packet2Packet bridging SIP/724311320422052-081d4e20 and SIP/softPhone-081
dc5b8
  == Spawn extension (macro-dialSIP, s, 1) exited non-zero on 'SIP/7243113204220
52-081d4e20' in macro 'dialSIP'
  == Spawn extension (macro-dialSIP, s, 1) exited non-zero on 'SIP/7243113204220
52-081d4e20'
    -- Executing [2102@sip-external:1] Macro("SIP/softPhone-081d4e20", "dialSIP|
724311320422052") in new stack
    -- Executing [s@macro-dialSIP:1] Dial("SIP/softPhone-081d4e20", "SIP/7243113
20422052") in new stack
    -- Called 724311320422052
    -- SIP/724311320422052-081dc5b8 is ringing
    -- SIP/724311320422052-081dc5b8 is ringing
    -- SIP/724311320422052-081dc5b8 is ringing
    -- SIP/724311320422052-081dc5b8 answered SIP/softPhone-081d4e20
```

Figure 7 - Screenshot of OpenBTS, Asterisk and Twinkle

9. Conclusion

To bring a GSM stack to a low cost hardware is not an easy task. The OpenBTS is giving this power for us, and as open source project, it can be used as excellent start point to learn how the GSM system works.

References

http://gnuradio.org/trac/wiki/OpenBTS

http://gnuradio.org/trac

http://en.wikipedia.org/wiki/GSM

http://www.ettus.com/