Πανεπιστήμιο Θεσσαλίας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων

Διπλωματική Εργασία

Θέμα:

«Σχεδιασμός και ανάπτυξη ενός συστήματος διαχείρισης μετρήσεων των χαρακτηριστικών του καναλιού σε ασύρματα δίκτυα.»

«Channel measurement methodology in wireless networks.»

Επιμελητής:
Πασσάς Βιργίλιος του Γρηγορίου

Επιβλέπων Καθηγητής:
Τασιούλας Λέανδρος
(Καθηγητής)

Συνεπιβλέπων Καθηγητής:
Κατσαρός Δημήτριος
(Λέκτορας)

2ος Συνεπιβλέπων Καθηγητής:
Κοράκης Αθανάσιος

Βόλος, Οκτώβριος 2012

Ευχαριστίες

Ύστερα από μια πορεία πέντε ετών στο Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας, ολοκληρώνω τις προπτυχιακές μου σπουδές με την εκπόνηση της παρούσας διπλωματικής εργασίας.

Θα ήθελα αρχικά να ευχαριστήσω θερμά τον κ. Κοράκη Αθανάσιο, του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, για τις χρήσιμες συμβουλές και υποδείξεις του καθώς και για την υποστήριξη που μου πρόσφερε κατά τη διάρκεια της φοίτησής μου, αλλά και κατά την εκπόνηση της διπλωματικής μου εργασίας.

Ευχαριστώ επίσης τον επιβλέποντα της εργασίας μου, Καθηγητή του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, κ. Τασιούλα Λέανδρο και τον συνεπιβλέποντα Λέκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, κ. Κατσαρό Δημήτριο για την καθοδήγησή τους.

Από καρδιάς θα ήθελα να ευχαριστήσω τον κ. Κερανίδη Ευστράτιο, υποψήφιο Διδάκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, και γενικότερα όλα τα παιδιά του NiTLab για την πολύτιμη συνδρομή τους στο να φέρω εις πέρας την διπλωματική μου εργασία.

Τέλος, ευχαριστώ θερμά την οικογένειά μου για την αμέριστη συμπαράσταση που μου παρείχε όλα αυτά τα χρόνια για την ολοκλήρωση των προπτυχιακών σπουδών μου.

Στην οικογένειά μου

Contents

ABSTRACT **7**

1. INTRODUCTION **8**

1.1 ELECTROMAGNETIC INTERFERENCE AT 2.4 GHZ 8

1.2 RECOGNISE SUCH KINDS OF INTERFERENCE 9

1.3 THE NITOS WIRELESS TESTBED 10

2. CHANNEL SENSING HARDWARE **11**

2.1 PRESENTATION OF USRP 11

2.1.1 Description of USRP *11*

2.1.2 Applications *12*

2.1.3 USRP hardware driver (UHD) *13*

2.2 USRPS 14

2.2.1 USRP1 ..... *14*

2.2.2 USRP N210 ..... *16*

2.3 DAUGHTERBOARDS 18

2.3.1 XCVR2450 2.4 GHz-2.5 GHz, 4.9 GHz-5.9 GHz Tx/Rx *18*

2.3.2 SBX 400-4400 MHz Rx/Tx *19*

3. GNU RADIO **20**

3.1 DESCRIPTION OF GNU RADIO 20

3.2 GNU RADIO COMPANION 21

4. MEASUREMENT COLLECTION FRAMEWORK **22**

4.1 DESCRIPTION OF OML 22

4.2 WAYS OF IMPLEMENTING OML 24

4.2.1 C/C++ source code *24*

4.2.2 Wrappers *25*

4.2.3 Text Protocol *25*

5. 1ST METHOD **28**

5.1 DESCRIPTION OF THE METHOD 28

5.2 DEMONSTRATION 29

5.3 USE CASES OF 1ST METHOD 31

5.3.1 1st Use Case: Wi-Fi bands *31*

5.3.2 2nd Use Case: Mobile Network bands *32*

6. 2ND METHOD 33
6.1 DESCRIPTION OF THE METHOD 33
6.2 DEMONSTRATION 35
6.3 EXAMPLE OF SQLITE 37
6.4 USE CASE OF 2ND METHOD 37

7. FUTURE WORK 38

8. REFERENCES 40

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία πραγματεύεται την σχεδίαση και ανάπτυξη ενός συστήματος διαχείρισης μετρήσεων των χαρακτηριστικών του καναλιού σε ασύρματα δίκτυα.

Το πρώτο μέρος αποτελείται από μια εισαγωγή στα ασύρματα δίκτυα, στις εξωτερικές παρεμβολές και στην ανάγκη για εργαλεία που να μπορούν να τις αναγνωρίσουν.

Στο δεύτερο μέρος κάνουμε μια παρουσίαση του hardware που χρησιμοποιήσαμε (USRP) και των λογισμικών (GNU Radio, OML) στα οποία βασιστήκαμε για να αναπτύξουμε το σύστημα μας.

Το τελευταίο μέρος της διπλωματικής διαπραγματεύεται την υλοποίηση του συστήματος, αποτελούμενου από 2 μεθόδους και μερικές περιπτώσεις όπου μπορεί να βρει εφαρμογή το προτεινόμενο σύστημα.

ABSTRACT

The present Final Project Dissertation – Thesis deals with the design and development of a framework for measurement of channel characteristics in wireless networks.

The first part of the project consists of an introduction about wireless networks, the external interference and the need of tools that can sense all kind of interference.

At the second part, we briefly present the USRPs, which is the hardware platform we use in order to take the measurements, the GNU Radio Project and the OML framework.

At the third part lies the implementation of our framework. We give a description of the two methods that we constructed and some use cases of our framework.

1. Introduction

In the last decade testbeds have been set-up to evaluate network protocols and algorithms under realistic settings. In order to draw solid conclusions about the corresponding experimental results, it is important for the experimenter to have a detailed view of the existing channel conditions. Moreover, especially in the context of non-RF-isolated wireless testbeds, where external interference severely impacts the resulting performance, the requirement of experimenters for accurate channel monitoring becomes a prerequisite.

1.1 Electromagnetic interference at 2.4 GHz

Microwave ovens operate by emitting a very high power signal in the 2.4 GHz band. Older devices have poor shielding, and often emit a very "dirty" signal over the entire 2.4 GHz band. This can cause considerable difficulties to Wi-Fi and Video senders, resulting in reduced range or complete blocking of the signal.

Many **ZigBee / IEEE 802.15.4-based** wireless data networks operate in the 2.45–2.4835 GHz band, and so are subject to interference from other devices operating in that same band.

Certain car manufacturers use the 2.4 GHz frequency for their **car alarm** internal movement sensors. These devices transmit on 2.45 GHz (between channels 8 and 9) at a strength of 500 mW. Because of channel overlap, this will cause problems for channels 6 and 11 which are commonly used default channels for Wi-Fi connections.

1.2 Recognise such kinds of interference

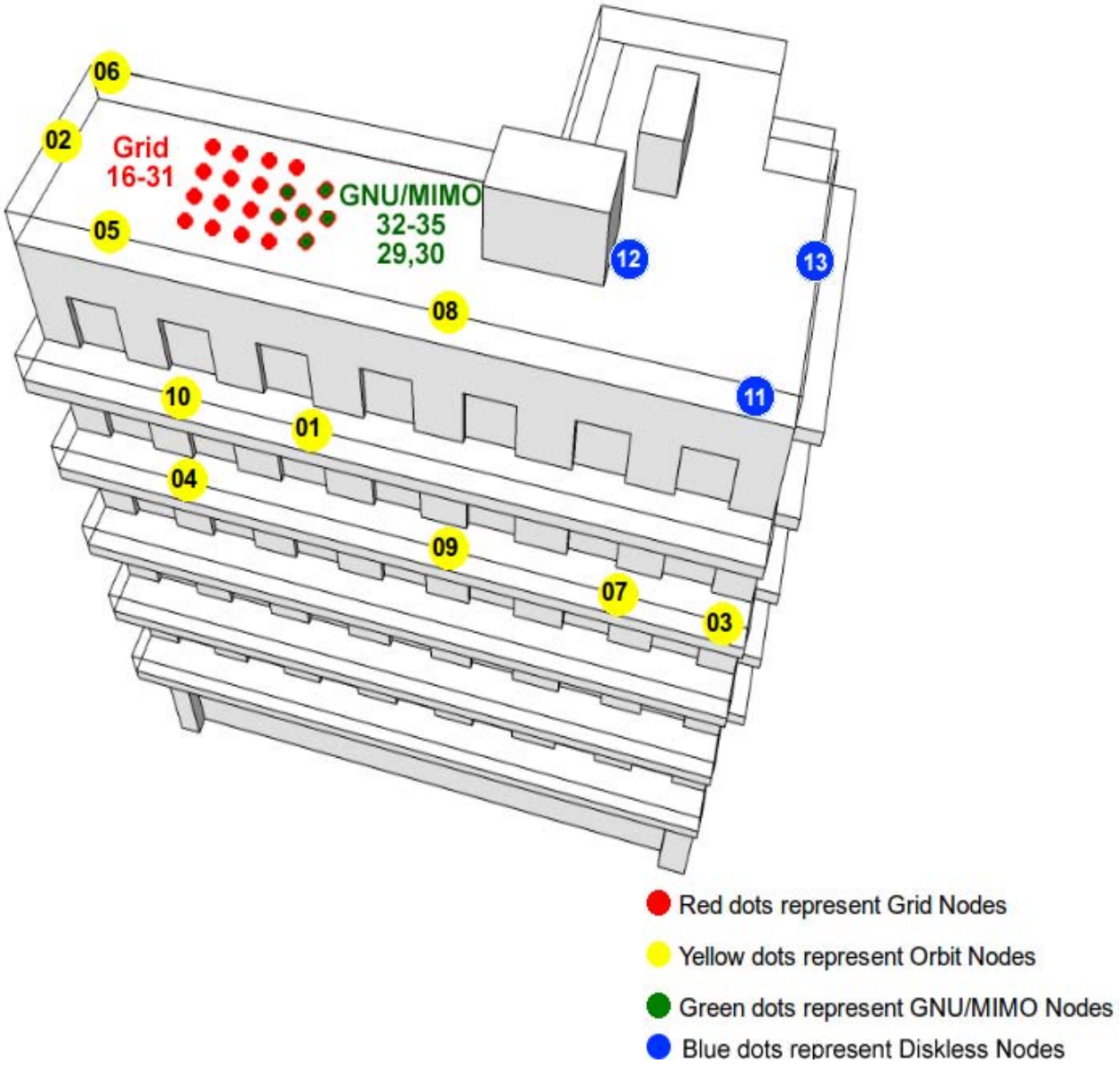
Due to the fact that commercial Wi-Fi cards cannot decode all the packets, for instance if rssi low or the packet is partially collapsed it discards the packet or if a microwave oven is in function, there is the need for sensitive and accurate tools. A solution to this is the SDR.

A software-defined radio system, or SDR, is a radio communication system where components that have been typically implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system. While the concept of SDR is not new, the rapidly evolving capabilities of digital electronics render practical many processes which used to be only theoretically possible.

Our framework is based on software-defined radio (SDR) devices that feature highly flexible wireless transceivers and are able to provide highly accurate channel sensing measurements.

1.3 The NITOS Wireless Testbed

The most important part of my work based on NITOS testbed. CERTH has developed a wireless testbed called Network Implementation Testbed using Open Source platforms (NITOS). NITOS is a testbed offered by NITLab and consists of 50 wireless nodes based on open source software. The testbed is outdoor and uses a wireless layout of nodes which can be used for measurements and experiments in a real time environment. That gives the opportunity to observe the results of an experiment out of the "secured" environment of a simulation program and to take conclusions in real dangers or problems that the final product maybe deal with.



2. Channel Sensing Hardware

2.1 Presentation of USRP

2.1.1 Description of USRP

The Universal Software Radio Peripheral (USRP) products are computer-hosted software radios. They are designed and sold by Ettus Research, LLC and its parent company, National Instruments. The USRP product family is intended to be a comparatively inexpensive hardware platform for software radio, and is commonly used by research labs, universities, and hobbyists. USRPs connect to a host computer through a high-speed USB or Gigabit Ethernet link, which the host-based software uses to control the USRP hardware and transmit/receive data. Some USRP models also integrate the general functionality of a host computer with an embedded processor that allows the USRP Embedded Series to operate in a standalone fashion.

The USRP family was designed for accessibility, and many of the products are open source. The board schematics for select USRP models are freely available for download. All USRP products are controlled with the open source UHD driver. USRPs are commonly used with the GNU Radio software suite to create complex software-defined radio systems. The USRP product family includes a variety of models that use a similar architecture. A motherboard provides the following subsystems: clock generation and synchronization, FPGA, ADCs, DACs, host processor interface, and power regulation. These are the basic components that are required for baseband processing of signals. A modular front-end, called a daughterboard, is used for analog operations such as up/down-conversion, filtering, and other signal conditioning. This modularity permits the USRP to serve applications that operate between DC and 6 GHz.

In stock configuration the FPGA performs several DSP operations, which ultimately provide translation from real signals in the analog domain to lower-rate, complex, baseband signals in the digital domain. In most use-cases, these complex samples are transferred to/from applications running on a host processor, which perform DSP operations.
The code for the FPGA is open-source and can be modified to allow high-speed, low-latency operations to occur in the FPGA.
All products in Ettus Research Bus Series use a USB 2.0 interface to transfer samples to and from the host computer. These are recommended for applications that do not require the higher bandwidth and dynamic range provided by the Network Series(USRP N200 and USRP N210).

2.1.2 Applications

This is a list of some of the applications the USRP has been used for:

- An APCO25 compatible transmitter/receiver and decoder
- RFID reader
- testing equipment
- a cellular GSM base station
- a GPS receiver
- an FM radio receiver
- an FM radio transmitter
- a digital television (ATSC) decoder
- passive radar
- synthetic aperture radar
- an amateur radio
- a teaching aid
- Digital Audio Broadcasting (DAB/DAB+/DMB) transmitter
- Mobile WiMAX receiver with USRP N2x0

2.1.3 USRP hardware driver (UHD)

The USRP hardware driver (UHD) is the device driver provided by Ettus Research for use with the USRP product family. It supports Linux, MacOS, and Windows platforms. Several frameworks including GNU Radio, LabVIEW, MATLAB and Simulink use UHD. The functionality provided by UHD can also be accessed directly with the UHD API, which provides native support for C++. Any other language that can import C++ functions can also use UHD. This is accomplished in Python through SWIG, for example. UHD provides portability across the USRP product family. Applications developed for a specific USRP model will support other USRP models if proper consideration is given to sample rates and other parameters.

It works on all major platforms (Linux, Windows, and Mac); and can be built with GCC, Clang, and MSVC compilers.

The goal of UHD is to provide a host driver and API for current and future Ettus Research products. Users will be able to use the UHD driver standalone or with third-party applications such as:

- GNU Radio
- LabVIEW
- Simulink
- OpenBTS

2.2 USRPs

2.2.1 USRP1

The USRP1 is the original USRP product and consists of:

• Four high-speed analog-to-digital converters, each capable of 64 MS/s at a resolution of 12-bit, 85 dB SFDR (AD9862).

• Four high-speed digital-to-analog converters, each capable of 128 MS/s at a resolution of 14-bit, 83 dB SFDR (AD9862).

• An Altera Cyclone EP1C12Q240C8 FPGA.

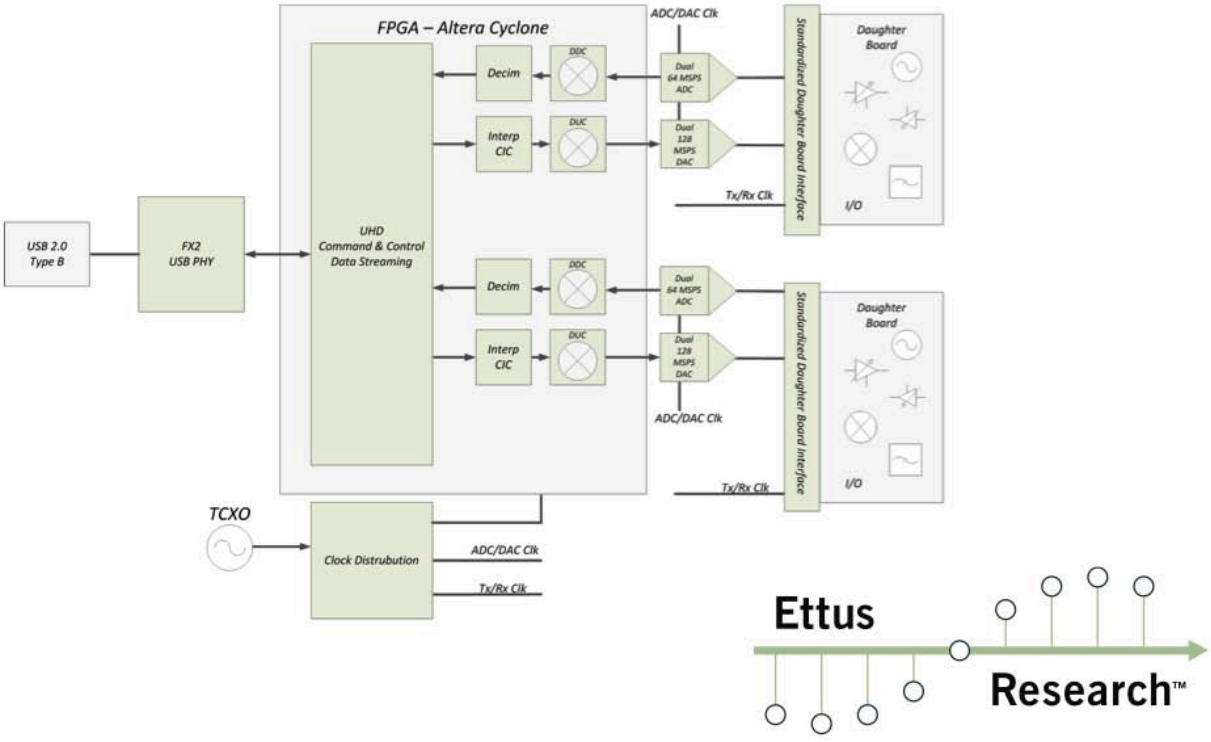
• A Cypress EZ-USB FX2 High-speed USB 2.0 controller.

• Four extension sockets (2 TX, 2 RX) in order to connect 2–4 daughterboards.

• 64 GPIO pins available through four BasicTX/BasicRX daughterboard modules (16 pins each).

• Up to 8 MHz of RF bandwidth in the receive

• Glue logic

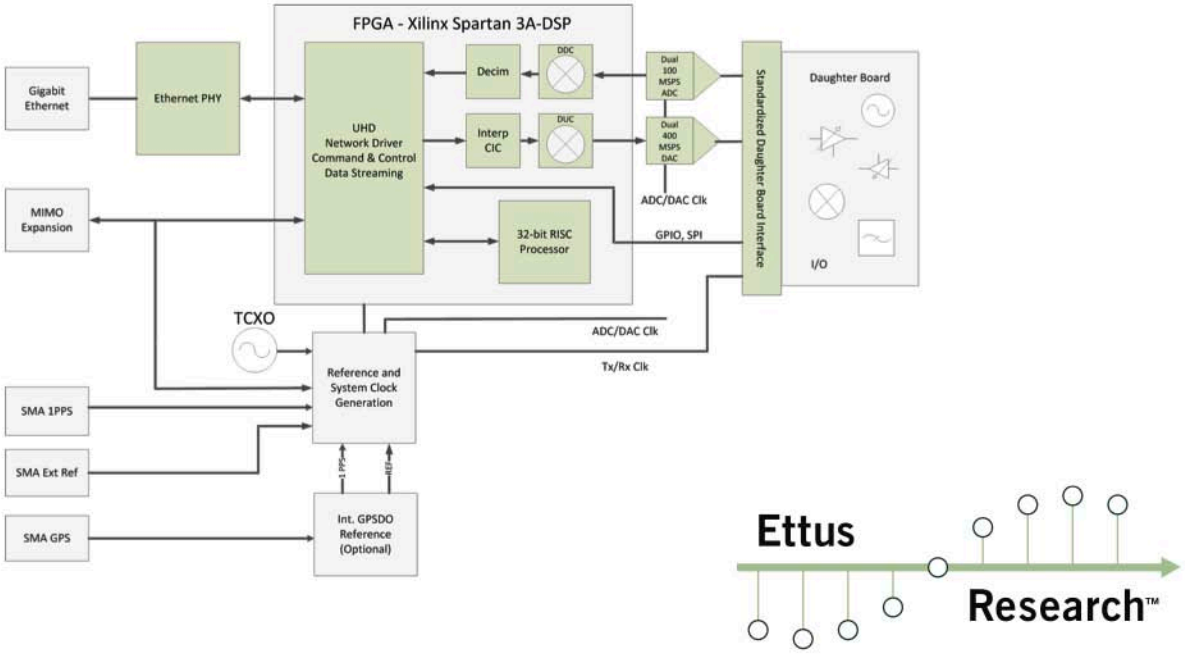


2.2.2 USRP N210

USRP N210 is part of the networked series and one of the highest performing class of hardware of Ettus Research

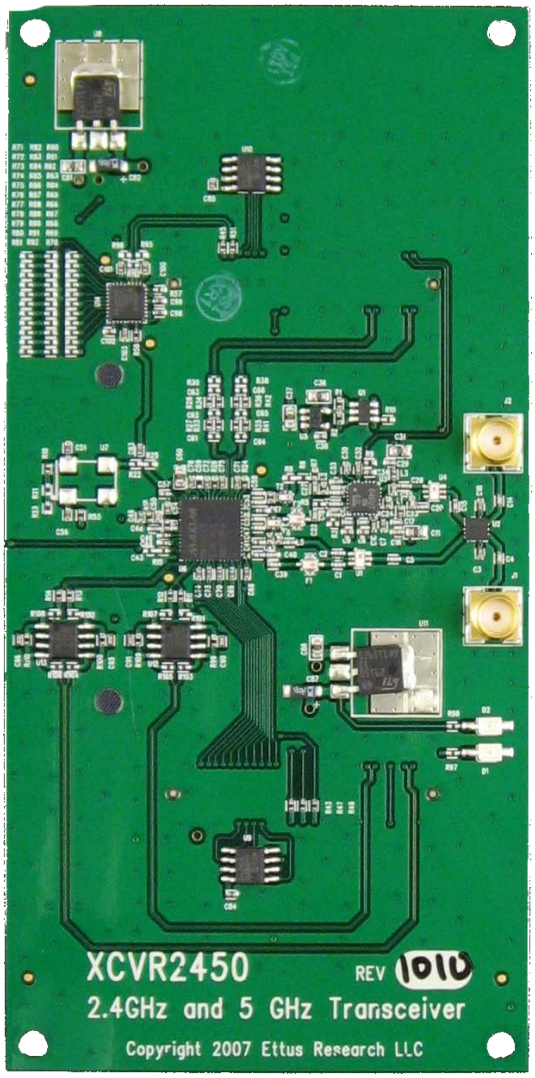
The USRP N210 consists of:

- A Xilinx Spartan-3A DSP 3400 FPGA
- 1 MB High-Speed SRAM
- Gigabit Ethernet interface
- Dual 100 MS/s, 14-bit, analog-to-digital converter
- Dual 400 MS/s, 16-bit, digital-to-analog converter
- Up to 50 MHz of RF bandwidth in the receive
- Flexible Clocking and Synchronization
- External Inputs for 10 MHz and 1 PPS signals (SMA)
- Optional GPS Disciplined Oscillator
- Ettus Research MIMO Cable that can be used to synchronize two USRP devices (sold separately)
- Support for timed commands and LO alignment with the SBX daughterboard



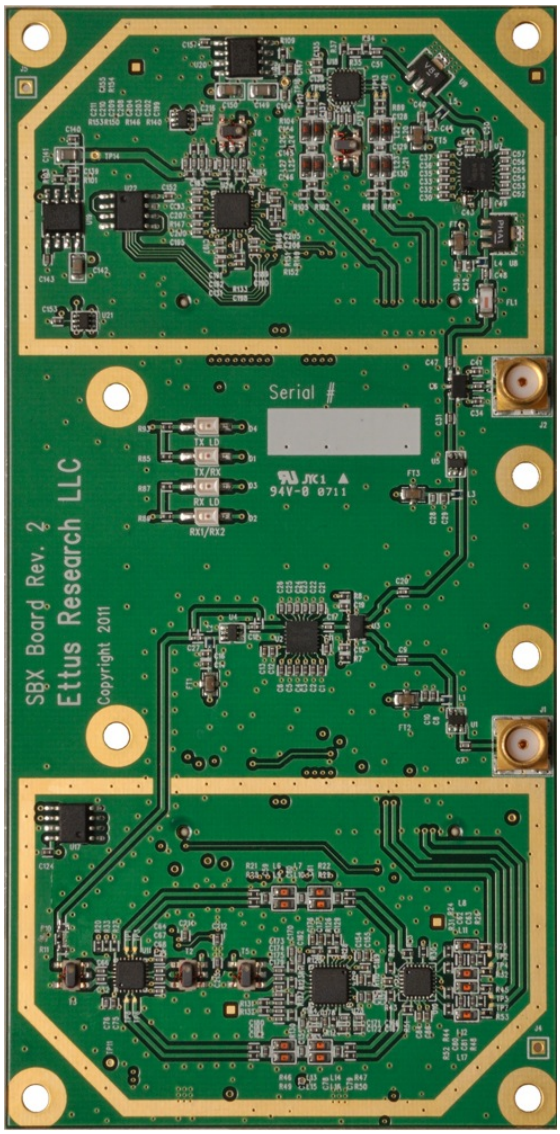
2.3 DaughterBoards

2.3.1 XCVR2450 2.4 GHz-2.5 GHz, 4.9 GHz-5.9 GHz Tx/Rx



The XCVR2450 is a high-performance transceiver intended for operation 2.4 GHz and 5.9 GHz range. Filtering on the XCVR2450 provides exceptional selectivity and dynamic range in the intended bands of operation. The typical power output of the XCVR2450 is 100 mW. Example applications include public safety, UNII, ISM, Japanese wireless and UWB development platforms. The XCVR2450 is a half-duplex transceiver.

2.3.2 SBX 400-4400 MHz Rx/Tx



The SBX is a wide bandwidth transceiver that provides up to 100 mW of output power, and a typical noise figure of 5 dB. The local oscillators for the receive and transmit chains operate independently, which allows dual-band operation. The SBX is MIMO capable, and provides 40 MHz of bandwidth. The SBX is ideal for applications requiring access to a variety of bands in the 400 MHz-4400 MHz range. Example application areas include Wi-Fi, WiMax, S-band transceivers and 2.4 GHz ISM band transceivers.

3. GNU Radio

3.1 Description of GNU Radio

GNU Radio is a free & open source software development toolkit that provides signal processing blocks to implement software-defined radio systems. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.

GNU Radio applications are primarily written using the Python programming language, while the supplied performance-critical signal processing path is implemented in C++ using processor floating-point extensions, where available. Thus, the developer is able to implement real-time, high-throughput radio systems in a simple-to-use, application-development environment.

GNU Radio supports development of signal processing algorithms using pre-recorded or generated data, avoiding the need for actual RF hardware.

GNU Radio is a signal processing package, which is distributed under the terms of the GNU General Public License. All of the code is copyright of the Free Software Foundation. The goal is to give ordinary software people the ability to 'hack' the electromagnetic spectrum, that is, to understand the radio spectrum and think of clever ways to use it.

As with all software-defined radio systems, reconfigurability is the key feature. Instead of purchasing multiple expensive radios, a single generic radio is purchased which feeds signal processing software.

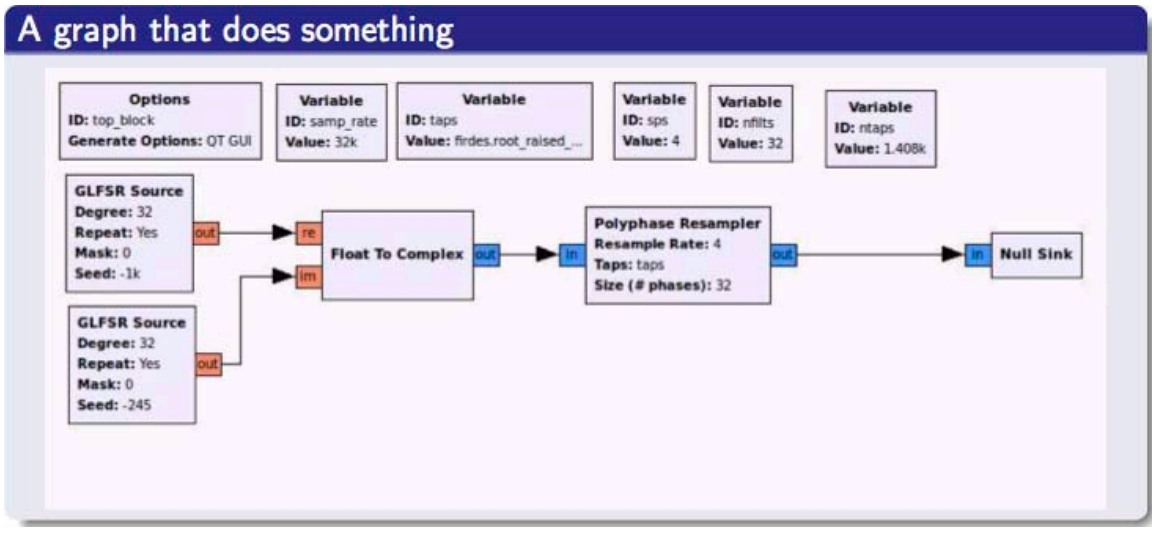
Currently only a few forms of radio can be processed in GNU Radio but if one understands the math of a radio transmission system, one can reconfigure GNU Radio to receive it.

3.2 GNU Radio Companion

GNU Radio has a graphical tool that visualize the signal processing blocks, the GNU Radio Companion (GRC)

GNU Radio Companion (GRC) is a graphical tool for creating signal flow graphs and generating flow-graph source code.

Here is an example:



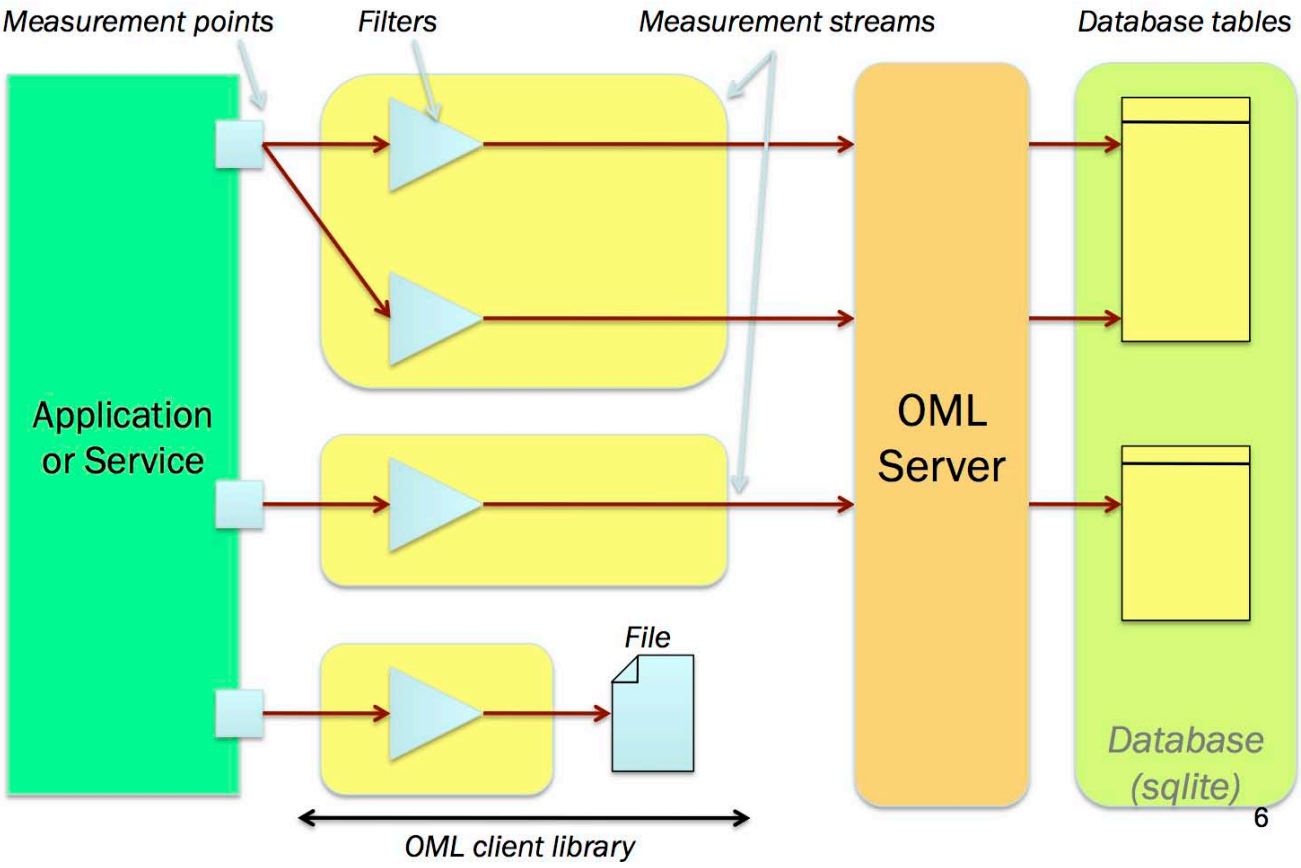
4. Measurement Collection Framework

4.1 Description of OML

OML is a software framework for measurement collection. It gives you a way to collect all the measurement data being recorded by a bunch of devices that are recording some sort of measurement data to a central location, via the network. OML is a generic framework that can be adapted to many different uses. Networking researchers who use testbed networks to run experiments would be particularly interested in OML as a way to collect data from their experiments. In fact, that's why the OML developed in the first place! However, any activity that involves measurement on many different computers or devices that are connected by a network could benefit from using OML. For instance, network monitoring, distributed simulations, or distributed sensor networks.

The OML framework is based on a client/server architecture and uses IP multicast for the client to report the collected data to the server in real-time. It defines the data structures and functions for sending/receiving, encoding/decoding and storing experiment data. With user-friendly and generic APIs, it can be easily integrated into user applications. Users can define what measurements are to be collected and stored. The clients at the experiment nodes collect measurements and send them to the collection server over a multicast channel after encoding them into XDR format.

OML supports multiple multicast channels and instances of the collection server per experiment to enhance the network scalability and provide reliability of data collection by load balancing and redundancy. An SQL database is used for persistent storage of experiment data that also allows access using standard data analysis tools like Matlab. Note that although OML is written initially with a focus on the ORBIT testbed, it can be used in various wired and wireless networking testbeds and distributed systems for data collection.



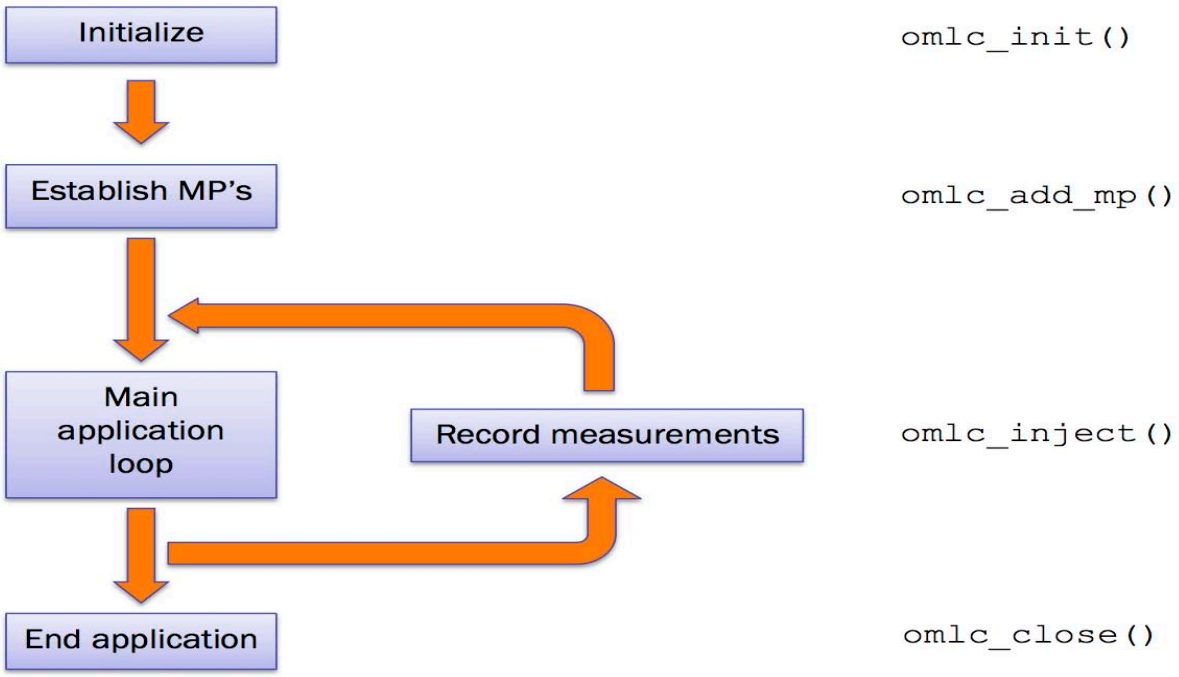
4.2 Ways of implementing OML

4.2.1 C/C++ source code

You can include OML measurement collection into your C and C++ applications by doing the following:

- Include the oml2/omlc.h header file in your source file: #include <oml2/omlc.h>
- Call omlc_init(), passing in the command line arguments from main(), before doing any option processing in your own code.
- Add measurement points using omlc_add_mp()
- Call omlc_start() to start the measurement collection process. At this point, interval-based measurement streams begin sampling.
- Call omlc_inject() whenever you want to record a measurement sample.
- Call omlc_close() when your program has finished all measurement collection activities.
- Compile and link your program against the liboml2 library.

The disadvantage of this method is that you have to modify the code when you change the application that you want to use.



4.2.2 Wrappers

If the source code is not available and the only thing that you have is an executable binary then you will have to develop a wrapper application, which will launch the executable, captures its output, formats it, and passes it on to OML. It is possible to write wrappers in C or C++ that read output from a program, parse it, and record it directly using liboml2.

It is also achievable to write a wrapper in almost any language by implementing the simple OML Text Protocol

4.2.3 Text Protocol

The text protocol was defined to simplify the sourcing of measurement streams from applications written in languages which are not supported by the OML library or where the OML library is considered too heavy. We primarily envision this protocol to be used for low-volume streams which do not require additional client side filtering. The OML client package includes a Ruby class which implements this protocol and provides convenient meta programming extensions to define measurement points easily. However, implementing the protocol from scratch in any language of choice should be very straight forward.

The protocol is loosely modeled after HTTP. The client opens a TCP connection to an OML server and then sends a header followed by a sequence of measurement tuples. The header consists of a sequence of key, value pairs terminated by a new line. The end of the header section is identified by an empty line. Each measurement tuple consists of a new-line-terminated string which contains TAB-separated text-based serialisations of each tuple element. Finally, the client ends the session by simply closing the TCP connection.

Header

The header contains the following keys. All of them have to appear exactly once, except for the 'schema' field which needs to appear once for every measurement stream carried by the connection.
protocol: Has to be "1"
experiment-id: String identifying the session and with it the database the measurements will end up in
start-time: Local UNIX time in seconds taken at the time the header is being sent
sender-id: A string identifying the source
app-name: A string identifying the application producing the measurements
content: Encoding of tuples, needs to be "text"
schema: Describes the schema of each measurement stream.

Schema Description

The description of the schema used in each measurement stream is a space-delimited concatenation of the following elements:

- local schema id
- name of the schema
- a sequence of name, type pairs, one for each element. The name and type in each pair are separated by a “:”

Each client should number its measurement streams contiguously starting from 1.

Measurement Tuple Serialization

Each tuple is serialized into a new-line terminated string with all elements separated by a TAB. In addition, three new elements are inserted before the measurements themselves. These three elements are defined as follows:

- time_stamp: A time stamp in seconds relative to the header's 'start-time'
- stream_id: This is the same number as used in the 'schema' header definition
- seq_no: A sequence number in the context of the specific measurement stream.

The sequence numbers of each measurement stream is independent of the others. That is, the first measurement in a given stream should have seq_no = 0, and all subsequent measurements in the stream should increment the sequence number by 1.

Example

protocol: 1
experiment-id: ex1
start-time: 1281591603
sender-id: sender1
app-name: generator
schema: 1 generator_sin label:string phase:double value:double
schema: 2 generator_lin label:string counter:long
content: text

0.903816 2 0 sample-1 1
0.903904 1 0 sample-1 0.000000 0.000000
1.903944 2 1 sample-2 2
1.903961 1 1 sample-2 0.628319 0.587785
2.460049 2 3 sample-3 3
2.460557 1 3 sample-3 1.256637 0.951057

5. 1st Method

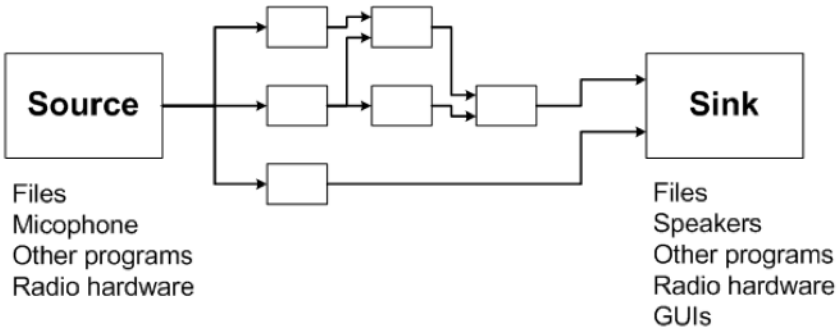
5.1 Description of the method

In this method, based on GNU Radio signal processing blocks, we wrote some scripts in python in order to gather measurement points from the USRPs into files and then open these files in order to filter the data store them and sent the results to the OML server using the module oml4r.

In detail, we developed python scripts that call signal processing blocks that take input from the USRPs and then store the input to files. When all calls end then the script is responsible to open all the files, filter the data and store them for future use from the user.

Afterwards make the computations and print the results to the standard output. In order to transfer these results to the OML server we use the Ruby module oml4r and wrote a wrapper in Ruby according to our needs. So the procedure that we just explain is actually executed by running this wrapper that it will call the python scripts and will wait to read the python script's output and then send the output to the server. This output will be stored to an SQLITE database in an efficient way.

Last step, another python script read the output and plot the results.



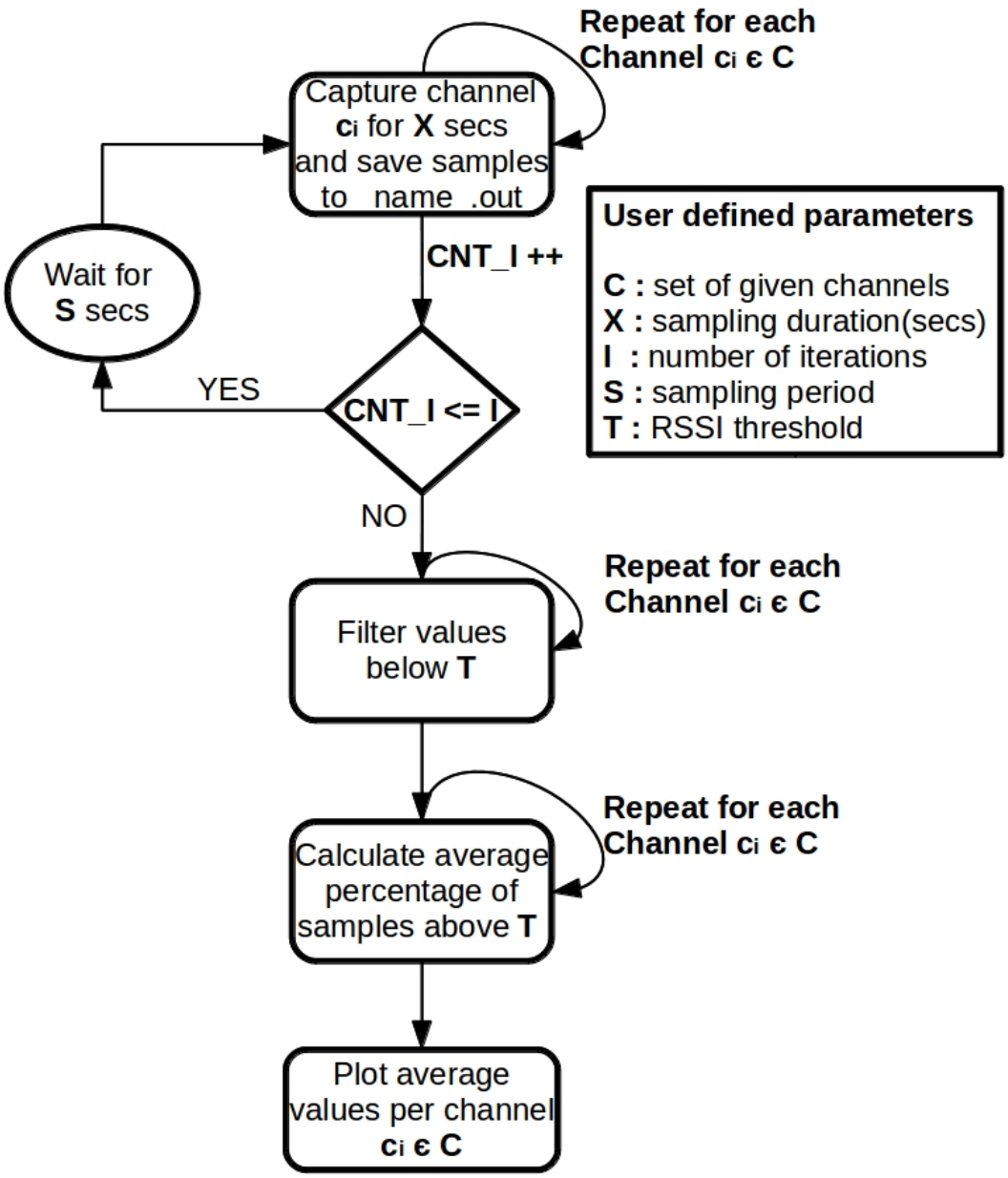
5.2 Demonstration

The main scope of the spectrum scanning procedure is to estimate the occupancy ratio per sampled frequency, regarding only Received Signal Strength (RSS) measurements that exceed a predefined RSS threshold. As for the first step, the reservation of the appropriate nodes that are equipped with USRPs is required. As for the second step, the experimenter simply executes specifically developed scripts that enable the definition of multiple sampling parameters, such as:

- the list of frequencies that will be sampled,
- the duration of sampling per individual frequency,
- the number of iterations of the repeated sensing procedure,
- the overall sampling period,
- and the RSS threshold that will be used for measurement filtering.

In the next step, the actual spectrum sensing is performed on each frequency among the list of frequencies that have been specified by the user and the gathered samples are saved locally at each node in an .out file format. Each frequency is sampled for duration equal to the provided duration and this sampling procedure is repeated for the specified number of iterations with interspace equal to the specified sampling period.

In the following step, all the locally saved files are filtered out, so that only values that are equal or above the specified threshold are taken into account. In order to accomplish this step, we set all values that are lower than the threshold equal to zero and save the filtered file into a new one. Afterwards, we calculate the average channel occupancy ratio per sampled frequency among the multiple measurements that have been gathered and store the corresponding results at an SQLITE database, with a unique identification tag. A flowchart representation of the sensing procedure follows:

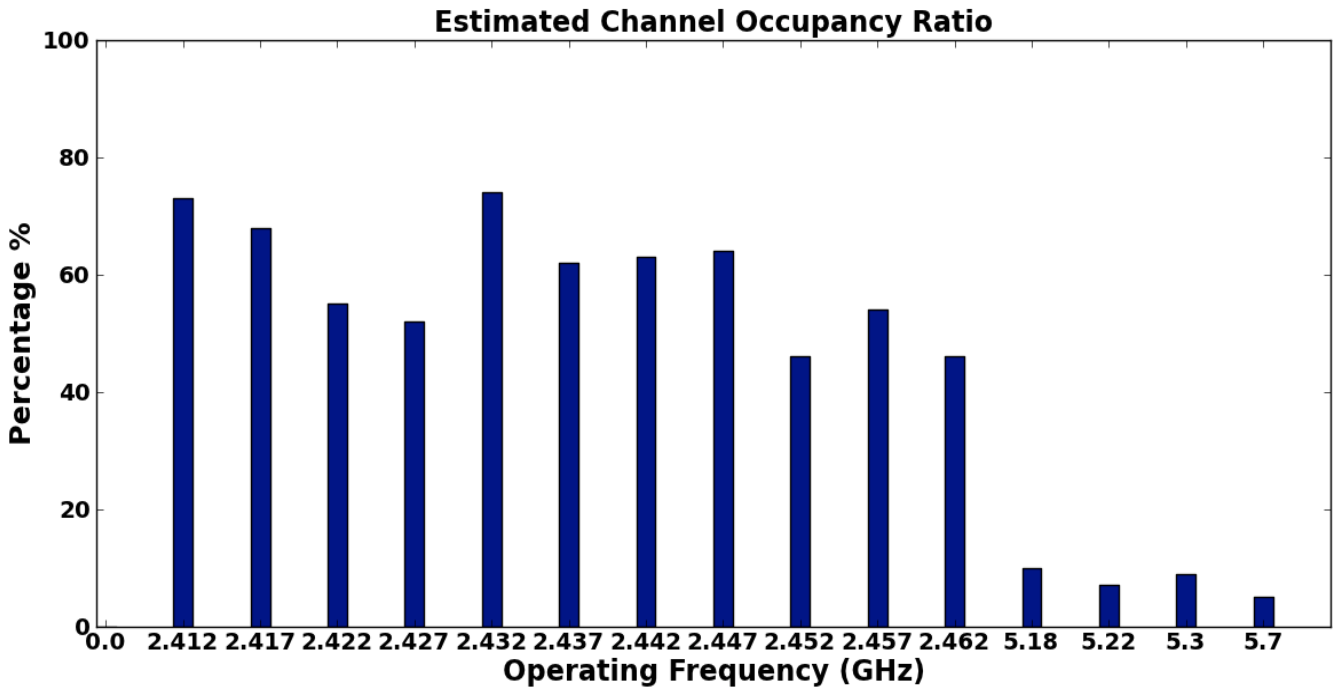


Finally, the user is able to get a graphical representation of each measurement set that has been stored in SQLITE database. Various statistical measures can be extracted from the corresponding records, such as average and deviation values per each frequency or per each individual iteration.

5.3 Use cases of 1st method

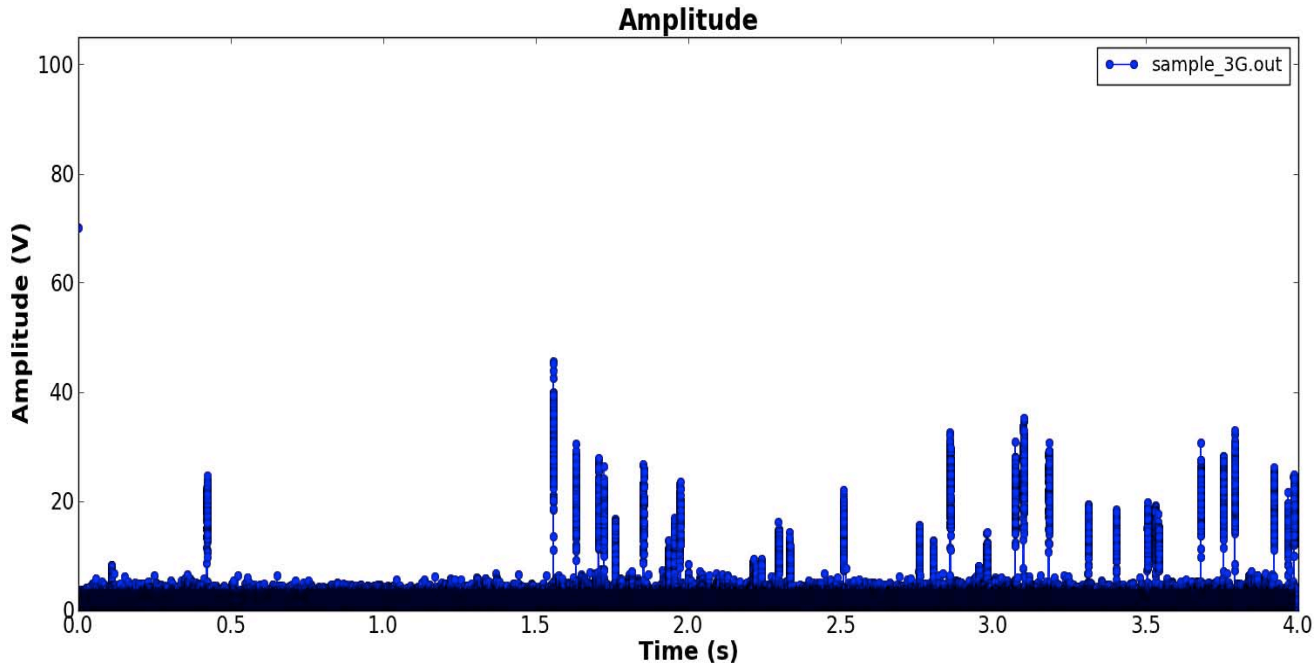
5.3.1 1st Use Case: Wi-Fi bands

Real Experimental results gathered from NITOS testbed



5.3.2 2nd Use Case: Mobile Network bands

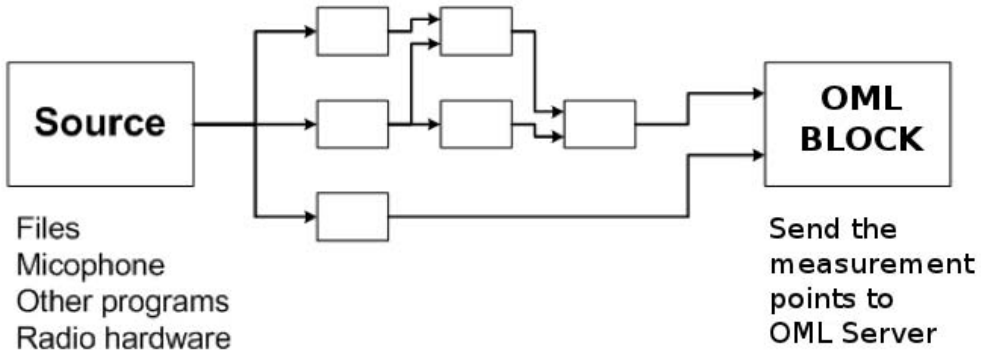
With the aid of the SBX daughterboard we are able to monitor the frequency that is used for data transfer in 3G Networks



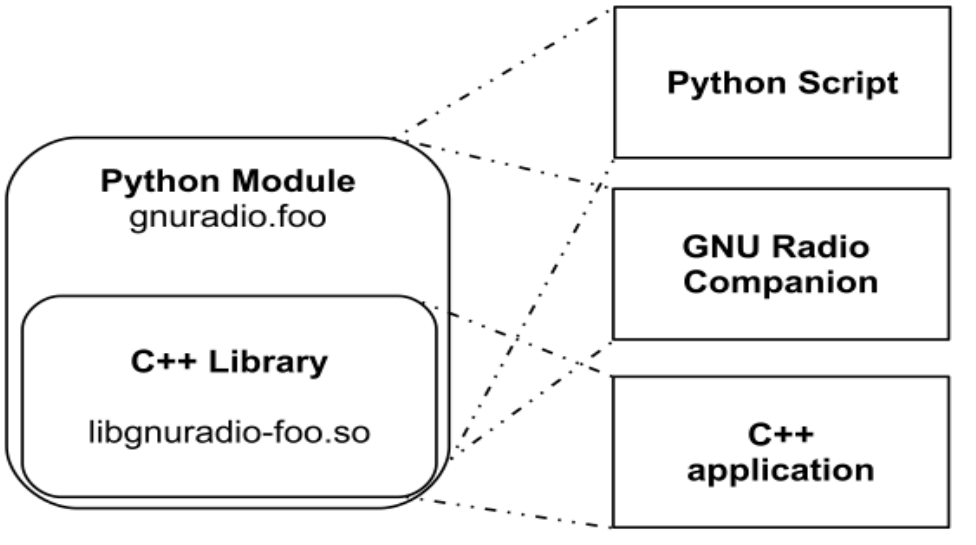
6. 2nd Method

6.1 Description of the method

In this method, we constructed a GNU Radio signal processing block in C++ that it takes a variable number of inputs of variable types and streams them to an OML server. So in this case we do not need to write the measurements into files and then access the files in order to send them to OML server.
Here is a schema of our method:



The general idea of a GNU Radio block and its structure:



The python script initializes the blocks, connect them and send the command for execution. The GNU Radio Companion part is an XML file that represent the block in the graphical tool. In the C++ application the signal processing is taking place.

In order to construct this block, we use the tool gr_modtool.py which help us to create a new module and then add a new block to this empty module. It's capable to change all the makefiles that we will use afterwards for our compiling.

A module in GNU Radio contains the following directories:

- apps: Contains test applications and examples.
- doc: Contains documentation files, including Doxygen .dox files to describe the component.
- examples: Contains example code to demonstrate usage of component blocks and algorithms.
- lib: Contains source files for the developed block.
- include: Contains header files for the developed block.
- python: Contains python scripts.
- swig: As GNU Radio combines C++ programmed blocks with Python flowgraphs, Swig tool is employed to automatically generate python interfaces for C++ blocks. Files .i required by this tool are stored in this folder.
- grc: Contains the .xml files needed by the grc tool.

After we deployed the functionality of our block filling the source and header files we compiled the module and installed it to the GNU Radio project so as to be available for our python scripts.

The task that now remain is to write a python script in order to test our block that works correctly. Afterwards we have to expand our script so as to take environment variables, namely the OML server settings:

- Experiment id
- Sender id
- OML server (in the form "tcp:serverhostname:port")

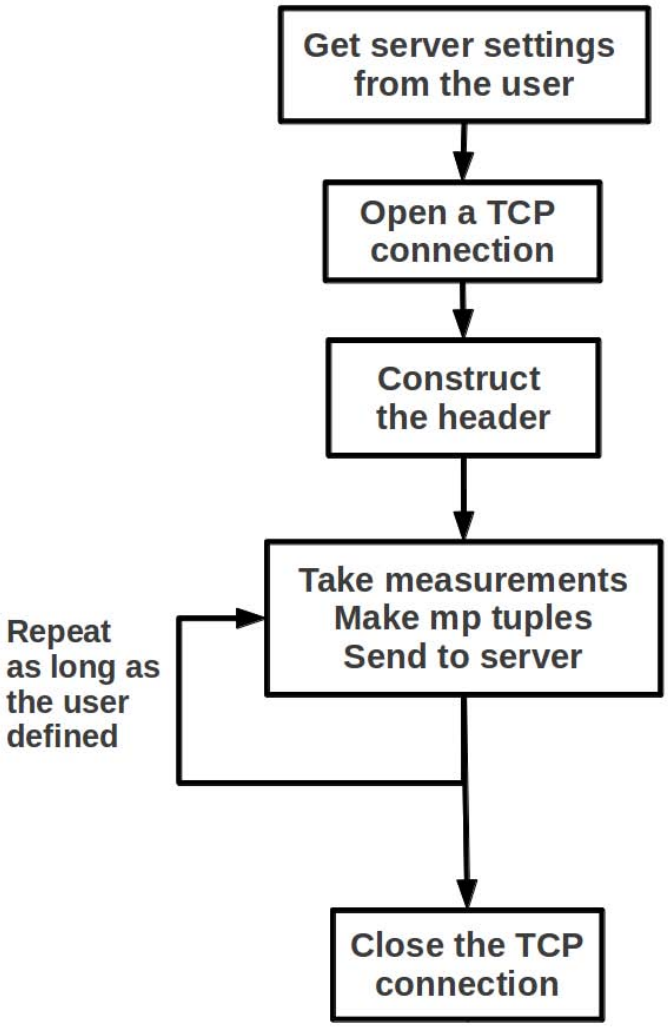
These environment variables will send to our block using it's accessors for setting values to its variables and then use them to make the TCP connection and send the measurements.

6.2 Demonstration

Order of the functions:

- Get the server settings
- Open a TCP connection with the OML server
- Construct the header of the message
- Take as input the measurements
- Construct and send the measurement tuples
- When the sensing ends, close the TCP connection

Here is the flowgraph:



This method does not include any plotting function of the gathered measurements. We just aim at storing measurements in an efficient way and so all results are available for further processing.

6.3 Example of SQLITE

1|356628|60.949927|27.669845|1|153
1|356629|60.949958|27.66985|1|156
1|356630|60.949984|27.669854|1|232
1|356631|60.950198|27.669859|1|205
1|356632|60.950236|27.673867|1|195
1|445159|67.596977|50.589441|6|190
1|445160|67.597144|50.589446|6|186
1|445161|67.597309|50.58945|6|295
1|445162|67.597476|50.589455|6|252
1|445163|67.597641|50.58946|6|179
1|445165|67.597973|50.589544|11|299
1|445166|67.598231|50.589551|11|180
1|445167|67.598329|50.589556|11|227
1|445168|67.598556|50.594609|11|185
1|445169|67.598658|50.594649|11|163

6.4 Use case of 2nd method

Use Case : CONCRETE

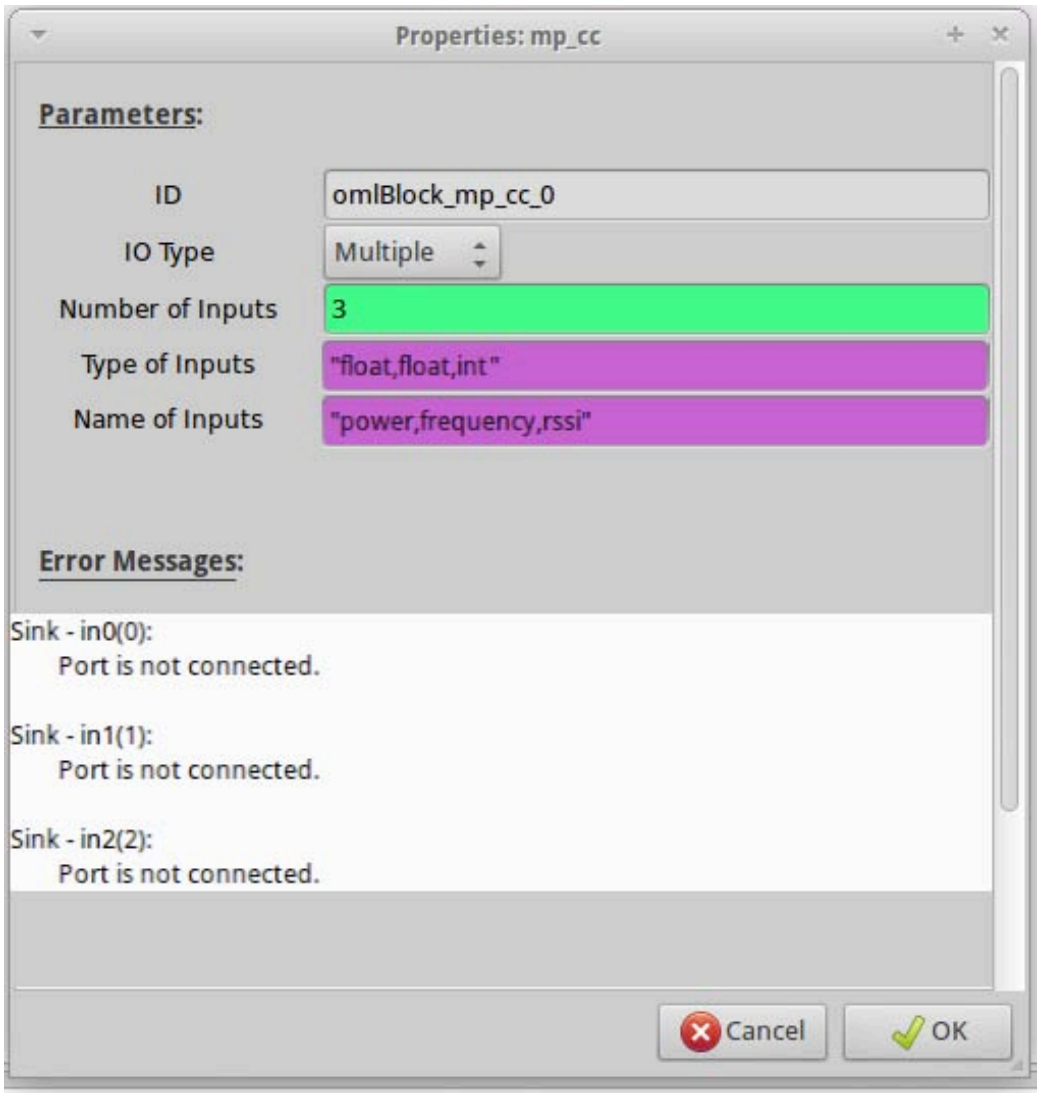
CONCRETE is a wireless environment monitoring tool and we can include our method to this tool. Because the testing is still in progress we are waiting the results and feedback.

7. Future Work

There are many things to do with this framework for measurement of channel characteristics in wireless networks.

Firstly, we expand our block in such way that what inputs it takes and streams them to the OML server then it will pass them on as outputs to the next block.

Secondly, we implement the .xml file of our block so as to be available to GNU Radio Companion and to be included to a signal processing flowgraph. Here is a first version of it:



Finally, the developed framework can be connected with the majority of existing GNU Radio Blocks as a result, we can take a step further and also consider more sophisticated techniques than plain energy detection, such as Feature detection. Feature detection enables distinction between transmissions generated by heterogeneous devices.

8. References

[1] USRP & DaughterBoards http://www.ettus.com/home

[2] USRP http://en.wikipedia.org/wiki/USRP

[3] GNU Radio project
http://gnuradio.org/redmine/projects/gnuradio/wiki

[4] modtool https://cgran.org/wiki/devtools

[5] C++ http://www.cplusplus.com/reference/

[6] Python http://docs.python.org/index.html

[7] OML http://omf.mytestbed.net/projects/oml/wiki

[8] Text Protocol
http://omf.mytestbed.net/projects/oml/wiki/Description_of_Text_protocol

[9] ORBIT Measurements Framework and Library (OML): Motivations, Design, Implementation, and Features, Manpreet Singh, Maximilian Ott, Ivan Seskar, Pandurang Kamat
WINLAB, Rutgers University

[10] Exposing GNU Radio: Developing and Debugging
http://www.trondeau.com/gr-tutorial/

[11] NiTLab http://nitlab.inf.uth.gr/NITlab/

[12] Wikipedia http://en.wikipedia.org/wiki/Main_Page

[13] OML Introduction and Tutorial, Christoph Dwertmann

[14] UHD wiki http://ettus-apps.sourcerepo.com/redmine/ettus/projects/uhd/wiki

[15] Electromagnetic interference at 2.4GHz http://en.wikipedia.org/wiki/Electromagnetic_interference_at_2.4_GHz

[16] Corgan Labs http://corganlabs.com/joomla/

[17] GNU Radio Screencasts http://www.youtube.com/user/2011HPS?feature=watch

&

http://www.youtube.com/playlist?list=PL618122BD66C8B3C4&feature=plcp