ftw. Technical Report

FTW-TR-2008-002
Printed August 2008

Open Source Software-Defined Radio: A survey on GNUradio and its applications

Danilo Valerio

ftw. Forschungszentrum Telekommunikation Wien
Donaucitystrasse 1, 1220 Vienna, AUSTRIA

FTW-TR-2008-002
Printed August 2008

Open Source Software-Defined Radio: A survey on GNUradio and its applications

Danilo Valerio
ftw. Forschungszentrum Telekommunikation Wien,
Donau-City-Strasse 1, A-1220 Vienna, Austria
valerio@ftw.at

Abstract

This report analyzes the feasibility of using a Software Defined Radio solution for research purposes. We focused on the open source GNUradio project and studied its suitability for reproducing and analyzing some widespread wireless protocols, such as IEEE 802.11, Bluetooth, IEEE 802.15.4, and GSM. We found that the use of GNUradio with the Universal Software Radio Peripheral can help researchers in avoiding the closed source firmwares/drivers of commercial chipsets by providing a full customizability at physical and datalink layers. On the other hand, software radios are not always capable of correctly reproducing operations previously done in the hardware domain. This leads to several limitations with widespread standards. In this report we try to provide a picture of such limitations and the current status of the GNUradio framework.

This work has been supported by the Telecommunications Research Center Vienna (ftw.) project N0. Ftw is supported by the Austrian Government and by the City of Vienna within the competence center program COMET. This work would not have been possible without the precious information contained in the official GNUradio mailing list archive. A special thanks goes to all the active participants.

Contents

Chapter 1

Introduction

The exponential increase in chip computing power encouraged radio engineers to continuously rethink the design of radio transceivers. The past decades saw a continuous proliferation of radio hardware. The traditional analogue hardware radio architecture consisted of a superheterodyne transceiver, where signal from the antenna is converted down to an Intermediate Frequency (IF), filtered, converted down to the baseband, and finally demodulated. This simple design has been the key success factor for the spread of televisions, FM radios, and first generation mobile phones. Later in the 80s the diffusion of fast and cheap Digital Signal Processors (DSP) led up to the development of digital transceivers. A digital radio transceiver is divided into two parts: a *radio Front-End* (FE), whose purpose is limited to a narrowband frequency downconversion followed by an Analogue-to-Digital Conversion (ADC), and a *radio Back-End* (BE), which is responsible for the remaining signal processing steps, such as (de)modulation, filtering, and channel (de)coding in the digital domain. This architecture succeeded mainly because of the low-cost availability of Application Specific Integrated Circuit (ASIC) chipsets but suffers from strict limitations in terms of flexibility. In fact ASICs are customized for a particular use, rather than intended for general-purpose use. Although both FE and BE can be expanded (e.g. by adding new narrowband or baseband cards), rapid evolving protocols do not let the investment be amortized.

The need to upgrade a radio transceiver via software updates with a marginal investment led in the past few years to the diffusion of the so-called Software-Defined Radio (SDR) architectures. In these systems the signal processing is performed (or at least managed) via software by using Field-Programmable Gate Arrays (FPGA), General Purpose Processors (GPP), or any other programmable device. By using SDR, engineers try to move the software domain as close as possible to the antenna in order to guarantee a higher flexibility. The decision of what should be implemented in software and what in hardware depends on the performance requirements of each particular implementation.

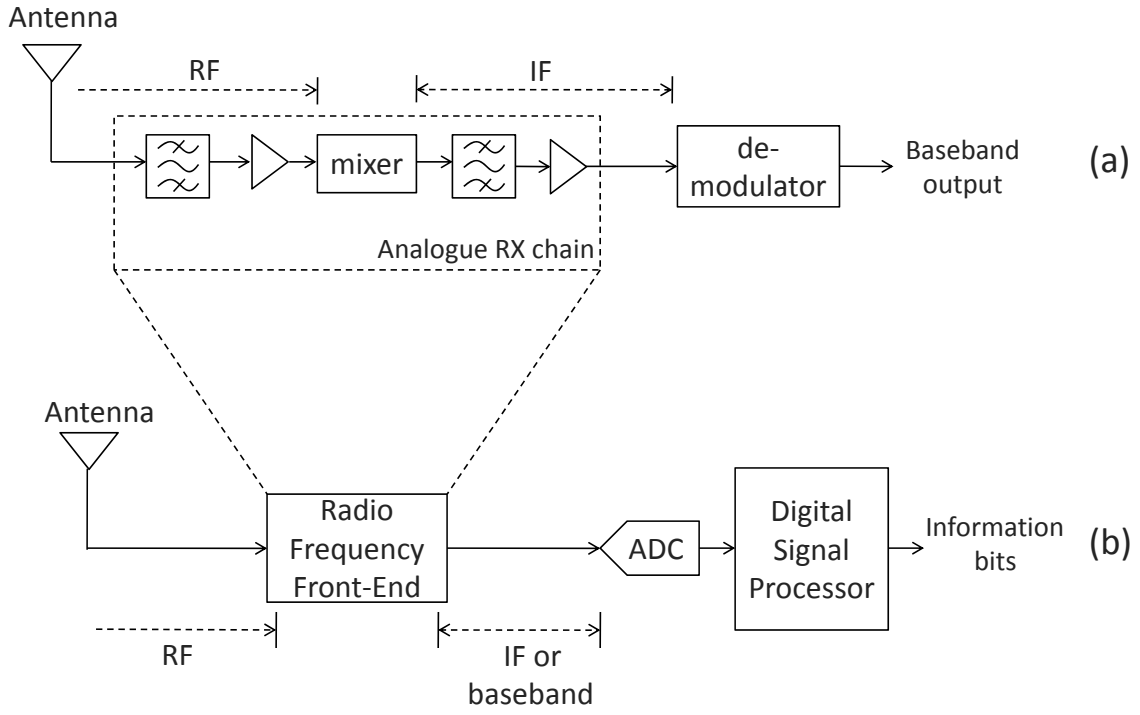

Figure 1.1. Analogue (a) and digital (b) hardware receivers

Software Defined Radio

We have defined an SDR as a radio platform that uses software techniques on digitized radio signals. Therefore, at a certain stage in the receiving chain the signal is digitized and passed to the software domain. Besides flexibility, the main aim of SDR is "to turn hardware problems into software problems" [1], enabling radio engineers to work in a more accessible domain. More in general, an SDR is divided into two subsystems: the *hardware-defined subsystem* and the *software-defined subsystem*. The separation between hardware and software subsystems is not fixed. Ideally, an SDR could consist of an antenna, an ADC, and a software-defined subsystem (see Fig. 1.2). Realizing such a device would require that each of the three following conditions are met:

- The antenna should be capable to operate at the frequency of all radio signals of interest;
- The ADC and the DAC should have a sampling rate greater than two times the frequency of the signals of interest;

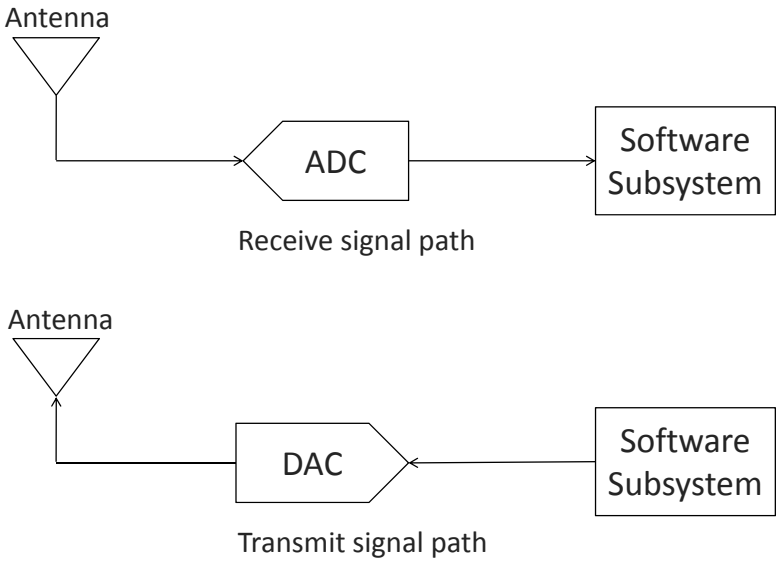

Figure 1.2. Block diagram of an ideal SDR

- The software subsystem should have enough processing power to handle the signal processing of all radio signals of interest.

In practice ADCs and DACs are not fast enough to process a large portion of the spectrum and antennas are designed for specific frequency bands. As a result, ideal SDR are realized only for particularly simple technologies (e.g. AM radio). In a typical SDR the hardware-defined subsystem consists of a wideband Radio FE that takes a portion of spectrum and shifts it to the IF prior to digitization. The software-defined subsystem receives the digital wideband signal, sends it to a set of digital downconverters to isolate the required carrier, and proceeds with the demodulation. It is important to note the fundamental difference between the radio FE as defined for common digital transceivers and the one used in SDR. The former is narrowband. It takes a narrow portion of the spectrum and shifts it to IF or baseband. The latter, in contrast, is usually wideband. It performs the same operation for a wider portion of spectrum, in order to be suitable for different technologies. Further filtering is then done in the software domain. The designing goal of SDR remains to make the FE as general as possible and act on the software subsystem for the implementation of new functionalities. The software-defined subsystem can also be composed by a combination of programmable devices, provided that the connection between them is fast enough to transfer DSP data. There are several SDR solutions available on the market. In this report we focus on the GNUradio platform, due to its flexible and entirely open source design.

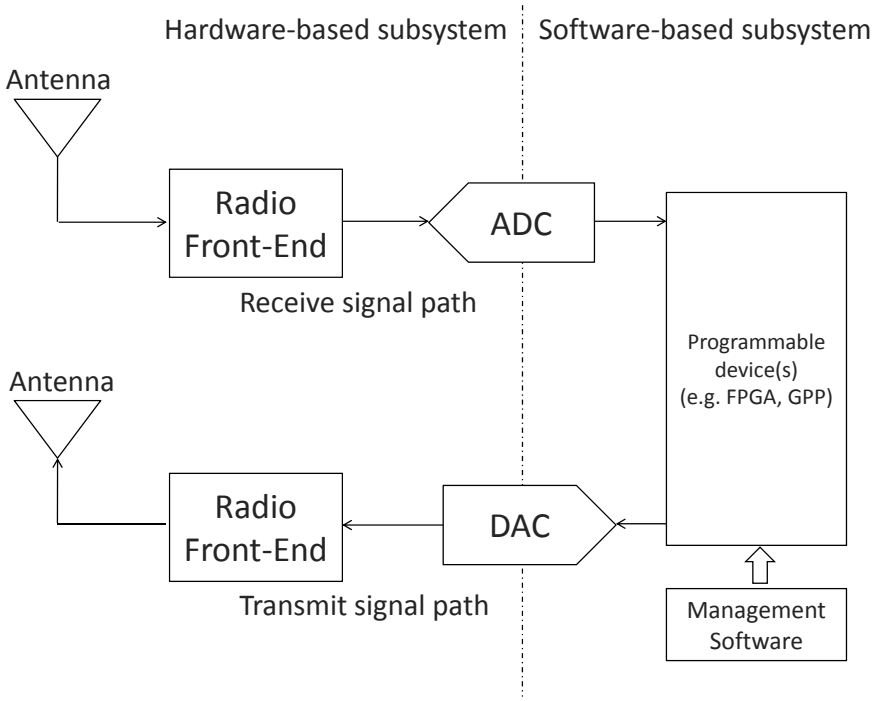

Figure 1.3. Block diagram of a typical SDR

The remainder of this report is organized as follows: In chapter 2 we analyze an SDR solution from Ettas Research, namely Universal Software Radio Peripheral (USRP), pointing out its main limitations. In chapter 3 we provide a short overview of the GNUradio framework. We further analyze the suitability of the GNUradio platform for some widespread wireless technologies. Finally, in 5 we draw the main conclusions.

Chapter 2

The Universal Sofware Radio Peripheral

The Universal Software Radio Peripheral (USRP) is a device developed by Ettus Research LLC [2], which turns general purpose computers into flexible SDR platforms. The core of the USRP is a motherboard with four high-speed ADCs and DACs and an Altera Cyclone EP1C12 FPGA. The ADCs/DACs are connected to the radio FEs (called daughterboards), while the FPGA is connected to a USB2 interface chip toward a general purpose computer. Figure 2.1 depicts the block diagram of the USRP revision 4. The main principle behind the USRP is that the digital radio tasks are divided between the internal FPGA and the external host CPU. The high speed general purpose processing, like down and up conversion, decimation, and interpolation are performed in the FPGA, while waveform-specific processing, such as modulation and demodulation, are performed at the host cpu. Besides datasheets and CAD files, the most complete source of information regarding the USRP is probably the document "The USRP under 1.5X Magnifying Lens!" [3], where a short presentation of the USRP is given, followed by a collection of frequently asked questions gathered from various Internet forums. In order to understand the limitations of the USRP, in the following subsections we will look at each component separately.

2.1 Motherboard

Figure 2.2 depicts the layout of the USRP mainboard. The USRP is completely designed under an open specification project using free and open source CAD software, i.e. schematics, cad files, and all other specification details are available for download on [2]. gEDA and PCB have been used for schematics and board layout. Also the FPGA design is open, therefore the FPGA firmware can be easily modi-

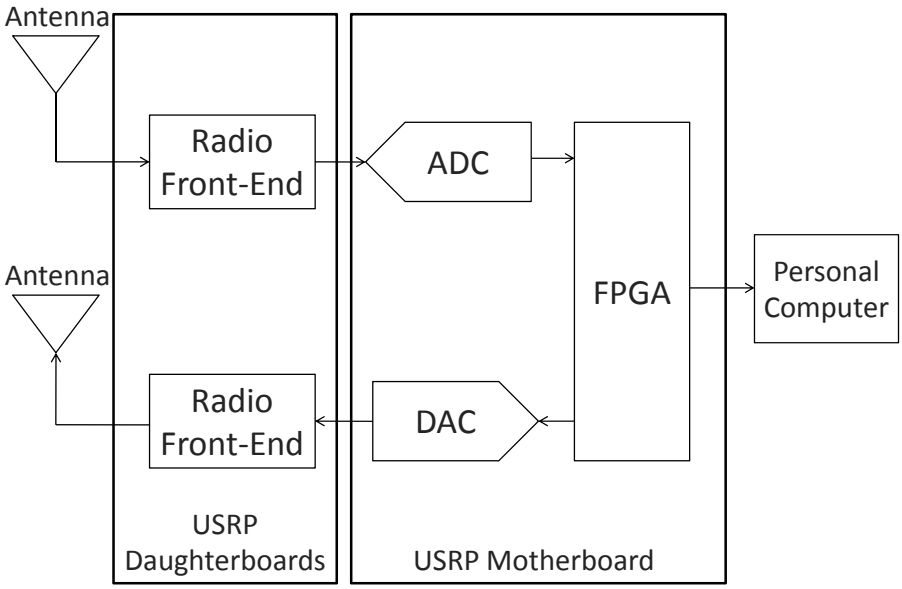

Figure 2.1. USRP simplified block diagram

fied. It is worth to take a closer look at the picture. The four white slots are used for connecting the daughterboards. Two slots are used for transmission (TXA and TXB) and two for reception (RXA and RXB). Signals from/to the daughterboards pass through the ADCs/DACs (the black chips between the white slots). At the center of the motherboard is the Altera Cylone FPGA, where part of the DSP is performed. Some centimeters under the FPGA one can see the USB interface chip, used for the communication from/to the host cpu. More details on each chip are given in the following subsections.

2.2 ADCs/DACs

The USRP contains four 12 bits ADCs. The sampling rate is 64 Msamples per second, therefore a signal with bandwidth up to 32 MHz can be digitized (see nyquist theorem). In other words, in order to digitize signals of frequency higher than 32 MHz without introducing aliasing the radio FE needs to downconvert it prior to the ADC. The board is also provided with a Programmable Gain Amplifier (PGA) before the ADCs, in order to use the whole available input range in case of weak signals. On the transmitting side there are four 14 bits DACs with a sampling

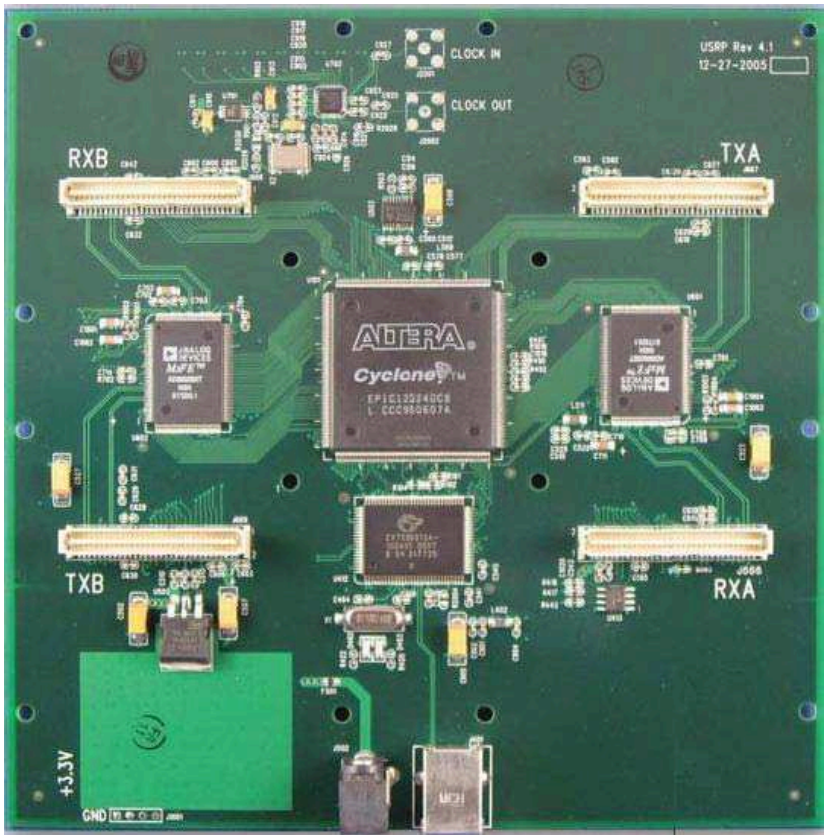

Figure 2.2. USRP mainboard (from [2])

rate of 128 Msamples per second. In this case the nyquist frequency is 64 MHz, although oversampling should be used for better filter performance. DACs are also followed by PGA, providing up to 20dB gain.

2.3 FPGA

The FPGA plays a central role in the USRP design. The USRP FPGA uses Verilog hardware description language, compiled by using Quartus II web edition from Altera. This compiler is available for free, therefore customized Verilog code can be compiled and uploaded to the FPGA firmware. The standard configuration is already suitable for a variety of applications. In alternative, Verilog code for particular applications can be found in the GNUradio community website [4]. The FPGA is connected to the ADCs, the DACs, and the USB controller. In the receiving path, the analog signal is converted by the ADC in 12 bit samples and then

passed to the FPGA for further processing. Here, a multiplexer routes the signal to the appropriate Direct Down Converter, which converts the signal to the baseband and decimates it by a factor specified by the user. Data is then interleaved (if there are multiple channels) and passed in 16 bit samples to the USB2 controller for transmission to the host cpu. The USB connection has a nominal bandwidth of 32MB/s half-duplex, i.e. bandwidth needs to be partitioned between down and up-link. The USB bandwidth is one of the most stringent limitations when designing a software radio with USRP. 16 bit samples (2 bytes) result in 32 MBytes/2 Bytes=16 Msamples per second, which provides a maximum signal bandwidth of 8 MHz, according to the nyquist theorem. This requires certain DSP steps to be done in the FPGA prior to the data transfer to the host cpu.

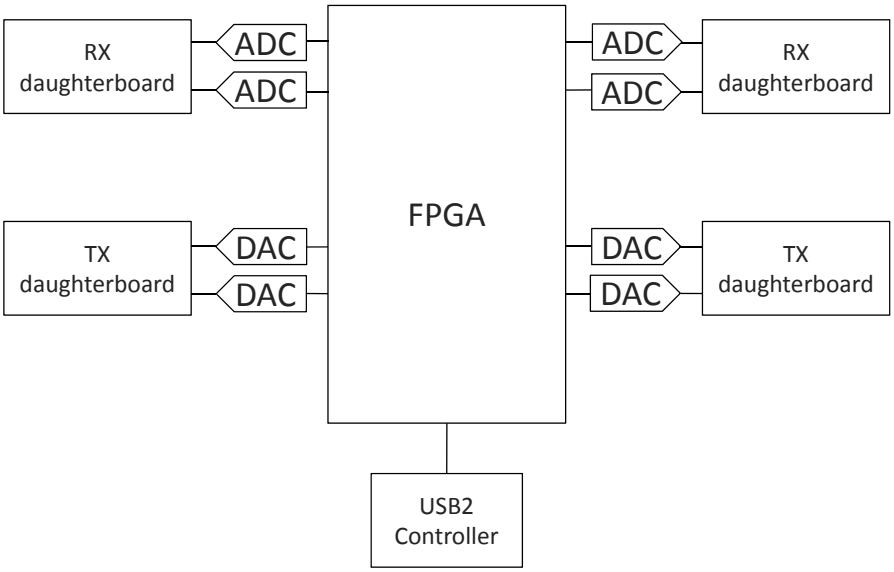

Figure 2.3. FPGA connections

2.4 Daughterboards

The USRP motherboard can be connected to four radio FE, namely daughterboards. Each of them has access to 2 high-speed ADCs/DACs and contains two SMA connectors for input and output signals. Ettus Research LLC produces ten daughterboards for different applications:

- BasicRX, 0.1-300 MHz receive
- BasicTX, 0.1-200 MHz transmit
- LFRX, DC-30 MHz receive
- LFTX, DC-30 MHz transmit
- TVRX, 50-860 MHz receive
- DBSRX, 800-2400 MHz receive
- RFX900, 800-1000 MHz Transceiver
- RFX1200, 1150-1400 MHz Transceiver
- RFX1800, 1500-2100 MHz Transceiver
- RFX2400, 2250-2900 MHz Transceiver

The BasicRX/TX daughterboards serve as simple entry/exit points for a signal. They do not have any mixer, filter, or amplifier, so they are just interfaces between the motherboard and the ether. The RFX series consists of complete RF transceiver systems with independent local oscillators, transmission/reception switches, and interestingly they are MIMO capable. In this technical report we focus on the RFX2400 since it enables reception and transmission in the 2.4 GHz ISM band. RFX2400 uses direct conversion, i.e. signals are translated to the baseband without any intermediate stage. It is able to transmit/receive signals with 20 MHz bandwidth and it contains a filter around the ISM band (i.e. 2400-2483 MHz), which can be easily bypassed. The maximum transmit power is 50mW (17dBm). Note that the functions of the daughterboards are also controllable via software/FPGA.

Chapter 3

The GNUradio Project

3.1 Introduction

After the signal has been processed by the USRP FPGA, the stream of bits finally flows through the USB connection to the host cpu. It is here that the GNUradio framework comes into play. GNUradio is a free software toolkit licensed under the GPL for implementing software-defined radios. Initially, it was mainly used by radio amateur enthusiasts, but it gained exponential interest from the research world, in an attempt to stay away from closed source firmwares/drivers, and low level of customizability of commercial chips. The GNUradio project was founded by Eric Blossom. It supports natively Linux, and packages are precompiled for the major Linux distributions. A port to Windows has been also developed, but it provides limited functionalities. The GNUradio website is a vast source of information [4] and there are various active forums on the web.

In GNUradio, a transceiver is represented by a graph, where the vertices are signal processing blocks and the edges represent the data flow between them. In this way, the process of developing an SDR is similar to what is done in the hardware domain, where physical RF building blocks are connected to each other to create a hardware radio. The package itself includes by default several building blocks for signal and information processing. In the following sections we will describe some of them.

3.2 Architecture

So far, we have seen that a GNUradio application consists of a flow graph. The vertices of the graph can be signal sources, sinks or processing blocks. Each block is

characterized by attributes that indicate the number of input and output ports, and the type of data that it can process. Clearly, a source block has only outgoing ports, a sink block has only incoming ports, and a processing block can have multiple incoming and outgoing ports. Signal processing blocks can be either synchronous or asynchronous. As far as regards the data format, blocks process usually complex floats, but other data formats are also possible (float, short, and chars). Unfortunately, GNUradio is very badly documented regarding the blocks that are provided. In order to understand the hundreds of implemented blocks, the only practicable way is to look at the source code and the documents generated by doxygen.

An incomplete list of implemented blocks is divided in [10] in the following functional groups:

- Mathematical operations (add, multiply, log, etc.)
- Interleaving, delay blocks, etc.
- Filters (FIR, IIR, Hilbert, etc.)
- FFT blocks
- Automatic Gain Control (AGC) blocks
- Modulation and demodulation (FM, AM, PSK, QAM, GMSK, OFDM, etc.)
- Interpolation and decimation
- Trellis and Viterbi support

While, the provided source and sink blocks include:

- Signal generators
- Noise generators
- Pseudo random number generators
- USRP source and sink (to transmit/receive signals via USRP)
- Graphical sinks (Oscilloscope, FFT, etc.)
- Audio source and sink
- File source and sink (reads/writes samples from/to a file)
- User Datagram Protocol (UDP) source and sink (to transport samples over a network)

3.3 Implementation

In GNUradio the blocks are created in C++ and the graphs are built and run in Python. SWIG2 is used as interface between them. Basically, C++ is used for low-level programming, while Python is used on a higher level, to create graphs or higher level blocks. GNUradio class hierarchy is depicted in [5]. It is possible to

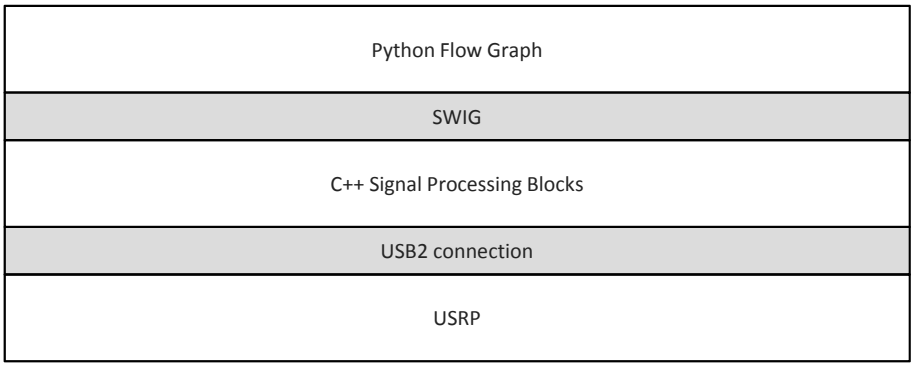

Figure 3.1. GNU radio framework structure

see that most of the signal processing blocks are derived from the class *gr_block*. In order to create new blocks (in C++), a new class needs to be derived from the gr_block class (or any of its subclasses). We do not go further into the creation of new signal processing blocks. Interested reader can find useful information in the tutorial "How to write a Block" by Eric Blossom [6].

Creating a flow graph with Python is straightforward. By using SWIG all the blocks are accessible from the Python source code. It is important to note that GNUradio provides classes to interface with the USRP (and the developers strongly suggest to use this device). However GNUradio is a more generic framework. There are no hindrances in using other hardware. A simple AM receiver for example can be implemented without USRP, since the frequency is not that high and the antenna can be connected directly to the ADC.

Programming a GNUradio application in Python does not require full knowledge of the C++ code in each blocks. The concept is similar to the OSI reference model. Signal processing blocks use the services offered by the hardware, and provide services to the python code through the SWIG interface.

3.4 Existing implementations

GNUradio enables engineers to implement several technologies in the software domain. Typically, three steps are involved in the process:

- Deeply understand the physical layer of the technology of interest;
- Create the required signal processing blocks or use the ones provided within the GNUradio package;
- Create the flow graph and tune the blocks.

This can be a tedious process and several limitations can arise during the implementation due to impossibility of the software to always correctly reproduce the hardware tx/rx chain. In the GNUradio package some python applications are provided, such as realtime capture and display of analog TV stations, transmission and reception of FM signals, transmission of dial tones to the sound card, etc. However, several groups around the world implemented new modules, reproducing a variety of widely used technologies. Unfortunately, to the best of our knowledge there is no online listing directory that groups all the available implementations. Extensive research is required in order to identify what has been already implemented and with which limitations.

We give here some examples of third-party modules that make use of GNUradio and an underlying USRP board. A complete GPS receiver has been implemented in [7]. It uses the DBSRX daughterboard for receiving GPS signals, and contains interfaces to the Google Earth data. Eric Blossom presented in [8] a passive radar by using FM frequencies (the code is included in the CVS GNU radio repository). A DVB-T module has been presented in [9], which is capable of receiving (and playing) DVB-T signals. Part of a DAB receiver has also been implemented in [10] (code is available at [11]), including a working physical layer, part of Fast Information Channel (FIC) to see the names of the radio stations, but without the Main Service Channel (required to actually hear something). Wiser SRL [12] claims to have implemented a GSM-R Signal Integrity Detection System (code is not available at the time of writing). Implementations for IEEE 802.11, Bluetooth, IEEE 802.15.4, and GSM are also available with certain limitations. Because of the importance of such applications, we treat them separately in the following subsections.

3.4.1 IEEE 802.11

There have been several attempts to reproduce the IEEE 802.11 signals with GNUradio. As we have seen in chapter 2, the USB bandwidth is limited to 32 MB/s.

The IEEE 802.11 signal spectrum is 11 MHz, therefore the minimum sampling rate would be 22 Msamples/s. By using 16 bits samples (8 bits for each I and Q sample) we would require a bandwidth of 2*22 Ms = 44 MB/s, which is above the USB2 limit. At the time of writing we can count two successful implementations that try to avoid the USB data rate bottleneck.

A first broadly used implementation has been developed within the ADROIT project funded by the DARPA's ACERT program and is available in [13]. In this implementation the signal bandwidth is reduced to 4 MHz before being passed to the USB controller. In this way, the signal is basically subsampled, therefore the Signal to Noise Ratio (SNR) degrades. Several posts in [14] report a successful reception of IEEE 802.11 data at 1 Mbps (DBPSK). More information can be gathered by looking at the available source code.

Hamed Firooz [15] proposed another implementation. In its architecture, despreading operation is performed in the FPGA prior to the USB, rather than in the host CPU. In this way, the data to be transfered through the USB connection is considerably reduced. Recall that IEEE 802.11 uses Direct-Sequence Spread Spectrum (DS-SS), with a code sequence of 11 chips per symbol. By despreading the signal before the USB connection, the symbol rate is reduced to 1 Msymbol per second, which is easily supported by the USRP. When the signal arrives at the PC, the IEEE 802.11 demodulator and Physical Layer Convergence Protocol (PLCP) from the ADROIT project is used for decoding the frame. Details on the implementation and the Verilog code for the Altera FPGA are freely available in [15].

3.4.2 Bluetooth

At a glance it could seem that the low power and datarate characteristics of Bluetooth make it easy to be implemented in the USRP/GNUradio. A more accurate analysis reveals that this is not always true.

Bluetooth communications use frequencies from 2402 MHz to 2480 MHz. One bluetooth channel is 1 MHz wide resulting in a total of 79 channels. The data is modulated using Gaussian Frequency Shift Keying (GFSK) with a symbol rate of 1 MSamples/s. GFSK is not explicitly implemented in GNUradio, but since it is a variant of Gaussian Minimum Shift Keying (GMSK) it can be easily implemented by using the included modules. Also the symbol rate does not pose any problem in the USRP. However Bluetooth uses a technique known as Frequency Hopping Spread Spectrum (FH-SS). A communication does not take place on a single channel (as in IEEE 802.11) but hops between different channels. In bluetooth the channel changes every 625 μs, and the hopping sequence is calculated from the clock and the MAC address of the master device (exchanged during the pairing procedure). Therefore, tuning the USRP on one channel would allow us to retrieve only sporadic packets from a communication. For a USRP to decode an entire blue-

tooth communication one needs to monitor the entire bluetooth spectrum, which is clearly unfeasible, or try to follow the hops sequence. The RFX-2400 daughterboard takes around 200 μs to settle to a frequency when a tune command is issued. This means that roughly the first third of a transmission slot will be missed as the radio tunes to the frequency. Some tests on a bluetooth implementation has been performed by Dominic Spill [17], who proposes the use of two bluetooth daughterboards in order to avoid the frequency settlement issue (while one daughterboard listens to one channel, the other one settles to the next channel). In [18] the problem is avoided by forcing the communication to a single channel. The most explicative words on the issue are probably given in [19], a thread on the GNUradio mailing list.

3.4.3 802.15.4

IEEE 802.15.4 is a relatively simple protocol at the basis of the Zigbee stack designed for sensor networks. It is meant to work in different frequency bands, although the 2.4 GHz ISM band is the most widely used. The latter is divided in 16 channels spaced 5 MHz apart from each other. The radios use direct-sequence spread spectrum coding and are modulated by using Orthogonal Quadrature Phase Shift Keying (O-QPSK). The raw, over-the-air data rate is 250 kbit/s per channel in the 2.4 GHz band. One bit at the source is encoded in 4 source symbols, which are in turn coded into 32 chips in the spreading process. This results in a chiprate of 2 Mchips/s. Since O-QPSK transports two bits on each modulation symbol, we have finally a channel symbol rate of 1 Msymbols/s. Neither the symbol rate nor the signal spectrum pose limitations to the GNUradio platform. A complete implementation of the IEEE 802.15.4 PHY/MAC layer has been presented in [20]. The code is open and freely available at [21]. However the authors measured Round Trip Times (RTT) three to six times larger than conventional radiochips. They found that this is due to the impossibility of the current implementation to reach a RX-TX turnaround time of 12 symbol periods (192 μs) as mandated by the standard[1].

Another implementation has been presented in [22]. The source code however is not publicly available, therefore we will not consider it further.

3.4.4 GSM

The GSM Software Project [16] aims at "bring together all the folks that are interested in building a gsm receiver/analyzer for less than 1000$". The project is

[1]The RX-to-TX turnaround time is defined as the shortest time possible at the air interface from the trailing edge of the last chip (of the last symbol) of a received PPDU to the leading edge of the first chip (of the first symbol) of the next transmitted PPDU.

divided into nine sub-projects, each with a specific goal. Two of them involve transmission, reception, and over-the-air monitoring of the GSM network:

- **The GSM Receiver Project** - aims at receiving GSM signals using the USRP.
- **The GSM Sending and Channel Hopping Project** - aims at sending GSM signals using the USRP and implementing channel hopping for receiving.
- **The OpenTsm Project** - aims at modifying the firmware of the Vitel TSM30 mobile phone for the collection of traces.
- **The A5 Cracking Project** - aims at cracking (decrypting) the A5/1 encryption algorithm used in GSM.
- **The GSM Decoding Project** - aims at decoding and converting data from the Traffic CHannel (TCH). The project's goal is to convert the speech channel data into PCM/WAV/MP3 and the SMS data into text files.
- **The Debug Trace Project** - aims at turning a Nokia 3310 into a trace mobile.
- **The SimCom Trace Project** - aims at receiving debug information from the digital baseband by using a SIM5210 module.
- **The UMTS/3G Project** - The project just started. The goal is to receive/send on UMTS and assess the security of UMTS.
- **The SIM Tookit Research Project** - aims at understanding how the SIM works and what is possible with it. It's related to security and applets that are installed remotely (via OTA) by the operator.

The GSM monitoring and decoding module for GNUradio/USRP has been presented in [23] and is available for free in [24]. It uses the USRP to capture live data, GNU Radio and custom modules to demodulate and decode the GSM packets (GMSK modulation is included in the default GNUradio package), and Wireshark to display the data. Currently, it is possible to monitor most of the GSM base station control channels.

- FCCH - frequency correction channel.
- SCH - synchronization channel.
- BCCH - broadcast control channel.
- PCH - paging channel.
- AGCH - access grant channel.
- SACCH - slow associated control channel.

- SDCCH - stand-alone dedicated control channel.

The module is reported to be in a “alpha” state, “not usable”, and with several limitations. According to the project website:

1. The Mueller and Muller clock recovery method does not always handle the quarter-bits present in a GSM burst. A more reliable method must be implemented. Until then, this software will suffer from a large number of receive errors even with a high signal-to-noise ratio.
2. Wireshark dissects most GSM packets except those specific to the Um interface, the wireless interface between the mobile and the BTS. A patch is available, which implements a portion of the Um interface, but it is incomplete and limited to the Bbis frame type.
3. The GSM tower needs to be found by hand and inserted into the python script.
4. The code is designed to support all frequency bands but only U.S. is implemented.
5. The code is receive-only and currently can only monitor tower to mobile transmissions.

Due to its complexity, the GSM software Project will be object of future studies.

Chapter 4

GNUradio and Research

In the past few years several research groups have chosen GNUradio and the USRP as a testbed platform. The main reason is that it provides an increased level of customizability overcoming some of the limitations of the commercial off-the-shelf hardware. In fact, while the use of the USRP could be worthless when experimenting with pure network layer (and above) algorithms, it becomes essential in case of physical layer research and cross-layer protocol design.

In [25] GNUradio is used to implement an Analog Network Coding scheme. The proposed algorithm makes use of network-level information to cancel interference at the receivers at signal level. Testing such a scheme in usual commercial platforms would have been expensive and would have required to re-program the FPGAs. On the other hand, a simulation approach would have been too inaccurate. The authors used GNUradio in its default configuration and implemented their mechanism with relatively simple python code.

A wireless cross-layer testbed called "Hydra" has been developed at the University of Texas at Austin [26] by using GNUradio and USRP boards. Hydra lets physical, datalink, and network layers interact with each other facilitating cross-layer experimentations. Each node consists of a general purpose computer connected to a USRP. The physical layer is based on GNUradio. More specifically, an OFDM implementation based on IEEE 802.11a is used. The PHY has an interface to the MAC layer, implemented over UDP through a local IP connection. The MAC and the network layer are implemented with Click [27], an open source framework developed at MIT for building flexible, high performance routers. Click is very versatile. It allows to easily compose MAC protocols and scheduling algorithms. Moreover, it interfaces to the linux TCP/IP stack via a simple tunnelling mechanism. This allows to test the developed protocols with tools running at the application layer. In [26] Hydra is used to test a SNR-based rate-adaptive MAC protocol, showing some of the potentialities of an SDR-based testbed.

GNUradio has gained interest also in other fields. In [18] flaws in the bluetooth privacy mechanisms are showed by sniffing with a USRP device. In [28] GNU-radio is used for implementing physical layer techniques for securing wireless sytems. Finally, some of the most important emulation platforms, such as Emu-lab [29] and Orbit-lab [30], already allow physical layer experiments by including USRP-equipped nodes.

Chapter 5

Conclusions

Driven by the increasing interest from the research community, we have reported about the current status of the main open source software-defined radio solution. We showed that GNUradio provides, together with the USRP, a comfortable environment for developing and testing new ideas, while it presents some limitations in reproducing current widespread technologies. More specifically, the limited bandwidth of the USB2 technology and some hardware-specific limitations pose a strict bottleneck in comparison to common digital transceivers. However, Ettus LLC already announced a new USRP v2 board. A new design and a gigabit ethernet connection (replacing the USB2) should overcome most of the encountered problems. According to the official GNUradio forum, the new USRP board will be presented officially at the end of the summer. We hope this will place a milestone for the researchers to definitively abdicate closed and costly commercial products, often badly documented and poorly customizable. The idea that software programmers and electric engineers finally joined each other and shared their competences leads to strong expectations. We strongly believe that in future, GNUradio and open source software-defined radios will be the de-facto standard for investigating new paradigms, such as cross-layer communication and cognitive radio.

References

[1] E. Blossom, "GNU Radio: Tools for Exploring the Radio Frequency Spectrum" Linux Journal, 2004, http://www.linuxjournal.com/article/7319

[2] Ettus Research LLC., http://www.ettus.com

[3] Firas Abbas Hamza, "The USRP under 1.5X Magnifying Lens", http://gnuradio.org/trac/attachment/wiki/UsrpFAQ/USRP_Documentation.pdf

[4] GNUradio Official website, http://gnuradio.org/trac

[5] GNUradio Doxygen documentation, Class Hierarchy, http://gnuradio.org/doc/doxygen/inherits.html

[6] E. Blossomm, "How to write a Signal Processing Block", http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html

[7] Gregory W Heckler, "Exploring GPS with Software Defined Radio", http://www.gps-sdr.com

[8] Eric Blossom, "I see Airplanes! How to build your own radar system", 22nd Chaos Communication Congress, December 2005, Berlin

[9] V. Pellegrini, G. Bacci, M. Luise, "Soft-DVB, a Fully Software, GNURadio Based ETSI DVB-T Modulator", 5th Karlsruhe Workshop on Software Radios, March 2008, Karlsruhe, Germany

[10] Andreas Mueller, "DAB software receiver implementation", Swiss Federal Institute of Technology Zurich, June 2008

[11] GR-DAB, http://people.ee.ethz.ch/ andrmuel/files/gnuradio/gr-dab.tgz

[12] WISER S.r.l. Wireless System Engineering Research, "GRIDES: GSM-R Signal Integrity Detection System", Livorno, Italy

[13] BBN technology - Acert Savane Server, http://acert.ir.bbn.com/

[14] GNUradio mailing list, http://lists.gnu.org/archive/html/discuss-gnuradio/

[15] Hamed Firooz, Implementation of Full-Bandwidth 802.11b Receiver, http://span.ece.utah.edu/pmwiki/pmwiki.php?n=Main.80211bReceiver

[16] The GSM Software Project, http://wiki.thc.org/gsm

[17] Dominic Spill, "Implementation of the Bluetooth stack for software defined radio, with a view to sniffing and injecting packets" University College London, May 2007.

[18] Dominic Spill, Andrea Bittau, "Bluesniff, Eve meets Alice and Bluetooth", The first USENIX workshop on Offensive Technologies, Boston, MA, August 2007

[19] GNU-Radio mailing list. Thread "802.11 and Bluetooth", http://lists.gnu.org/archive/html/discuss-gnuradio/2006-12/msg00089.html

[20] Thomas Schmid, Tad Dreier, Mani B Srivastava, "Software Radio Implementation of Short-range Wireless Standards for Sensor Networking", SenSys 2006 , November 2006.

[21] Networked and Embedded Systems Laboratory, UCLA, Zigbee sourcecode, http://acert.ir.bbn.com/projects/gr-ucla/,

[22] Stefan Knauth, "Implementation of an IEEE 802.15.4 Transceiver with a Software-defined Radio setup", Technical Report, Lucerne University of Applied Sciences

[23] Steve, Joshua Lackey, David Hulton, "The A5 Cracking Project: Practical attacks on GSM using GNU Radio and FPGAs", Chaos Communication Camp 2007, August 2007, Berlin

[24] Groupe Special (Software) Mobile (GSSM), http://thre.at/gsm

[25] S. Katti, S. Gollakota, D. Katabi "Embracing Wireless Interference: Analog Networ Coding", ACM SIGCOMM Computer Communication Review, Vol. 37, Issue 4, Pages: 397-408, October 2007

[26] K. Mandke, S.H. Choi, G. Kim, R. Grant "Early Results on Hydra: A Flexible MAC/PHY Multihop Testbed", IEEE Vehicular Technology Conference Spring, Dublin, Ireland, April 23 - 25, 2007

[27] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek "The Click modular router", ACM Trans. Comput. Syst., vol. 18, no. 3, Pages: 263-297, 2000

[28] Z. Li, W. Xu, R. Miller, W. Trappe, "Securing Wireless Systems via Lower Layer Enforcements", 5th ACM workshop on Wireless Security, Pages: 33-42, Los Angeles, 2006

[29] Emulab, Total Network Testbed, http://www.emulab.org/

[30] Orbit-lab, http://www.orbit-lab.org/