A COMPARATIVE STUDY OF TWO SOFTWARE DEFINED RADIO PLATFORMS

Gael Abgrall (ENSIETA, BREST, FRANCE, abgralga@ensieta.fr); Frédéric Le Roy (ENSIETA, BREST, FRANCE, leroyfr@ensieta.fr); Jean-Philippe Delahaye (CELAR, BRUZ, FRANCE, jean-philippe.delahaye@dga.defense.gouv.fr); Jean-Philippe Diguet (Lab-STICC, LORIENT, FRANCE, jean-philippe.diguet@univ-ubs.fr); Guy Gogniat (Lab-STICC, LORIENT, FRANCE, guy.gogniat@univ-ubs.fr)

ABSTRACT

The aim of this paper is to compare the performances of two free SDR platforms: GNU Radio and OSSIE which are deployed on the same computer. SDR applications development or waveforms conception are pretty close in these two systems but the software architectures are totally different. In order to quantify the cost of inter-components communication, the same waveform has been characterized on both platforms. Three critical points are evaluated: latency, CPU load and memory utilization. The results show that the average CPU load in OSSIE is five times higher than GNU Radio. The issue is the same for the inter-component latency where GNU Radio is almost 25 times faster than OSSIE.

1. INTRODUCTION

The Software Radio concepts imply complex signal processing performed in software with real time requirements and constraints of embedded systems, as power consumption, memory and throughput capacity limitations. The amount of software involved has quickly increased in the new Software Defined Radio (SDR) transceiver due to this shift of paradigm from hardware to software. The reconfigurability and the flexibility of SDR systems are among the main goals of defining application by software. The technological answer is the introduction of software technologies and industrial standards into the software radio application design. Standards have been introduced to tackle the implementation issues of large software application over complex heterogeneous and distributed hardware platforms.
The software engineering for the Software Defined Radio is driven by some key software concepts such as the portability, and the reusability of waveform applications code. It also aims at using software standards to reduce the development time and cost.

The main software effort in SDR standardization belongs to the Joint Tactical Radio System (JTRS) Joint Program Executive Office (JPEO), which defines the specification of a software infrastructure: the Software Communication Architecture (SCA). The SCA [1] is based on several software technologies as Common Object Request Broker Architecture (CORBA), POSIX Operating System (OS), and software engineering techniques as model based design and object-oriented programming. The SCA specifications define the Operating Environment (OE) software. The OE specification mainly defines the Core Framework (CF), responsible for the management, the control, the configuration and deployment of the waveform applications and hardware platforms.
Today, many commercial software products are available related to SCA software. The academic research also provides free software implementation of the SCA. Two of the most popular free implementations of SCA running on a PC workstation are the SCARI OPEN [2] implementation from the CRC and the OSSIE [3] implementation provided by the Wireless research lab of Virginia Tech.

Another approach of designing waveform application comes from the free software community with the GNU Software Radio [4] project. The GNU Software Radio is a library of free software codes that define radio waveforms for a software receiver. GNU Software Radio Application (RA) modules are written in C++, while the functions that configure the RA modules into a functioning radio are written in Python [5].

This paper presents two implementations of a reference FM receiver waveform using the two different approaches: SCA with OSSIE as shown in [6] and GNU. Both implementations are based on a PC running Linux and the receiver front end is a Universal Software Radio Peripheral [7]. This paper is discussing, the pro and cons of the two approaches, in terms of waveform architecture, modularity, scalability, portability and performances. It also discusses the differences of goals and features. The two approaches have been tested in educational perspectives to evaluate their learning facilities for new engineers in field of SDR, and to evaluate these technologies for future student projects.

The remainder of this paper is organized as follows. Section 2 gives an overview of the FM waveform, signal processing

details, architecture and coding consideration in both implementations GNU Software Radio and SCA software based on OSSIE. Section 3 presents our analysis of waveform performances and impact of SCA and GNU implementations in detail, and Section 4 concludes the paper.

2. WAVEFORM AND HARWARE / SOFTWARE PLATFORM

The main objectives of this section are to define the application waveform and its different refinement to observe the influence of the application granularity onto the latency and the OSSIE host computer load.
This work is based onto the "FM broadcasting demodulation waveform" presented in [8] and [4]. The FM wave radio is received from an antenna connected to the analog input of a "BasicRx" daughter board of a USRP developed for the GNU Radio project. This board is used for coupling the antenna with one of the four 12-bit analog to digital converter of the USRP mother board.

2.1. RF signal features

The French FM broadcasting band spreading from 88 to 108 MHz is sampled without any filter at 64 MHz. This under sampling band [9] produces four aliases of the band between -32 MHz and 32MHz. The sum of theses aliases are not destructive between 20 and 24 MHz, so with this system, it is possible to demodulate the French radio station located between 88 and 92 MHz.
After conversion the signal is first decimated and down-converted to a baseband signal by a DDC (Digital Down Converter) before sending it through a USB chipset to the GPP under which OSSIE is working. Each DDC component produces a complex signal $I(t)+jQ(t)$ where $I(t)$ is the inphase signal and $Q(t)$ is the quadrature phase signal. The maximum rate to the software side is limited by the USB chipset to 32 MB/s. The *I* and *Q* samples are coded in 16-bit signed integer so the complex at the output of the DDC is then coded with one 32-bit integer. The maximum complex rate across the USB results to 8 Msps.

The bilateral stereo broadcast's spectrum received by the software spreads over a 200 kHz bandwidth. Mostly, this spectrum is composed with a bilateral sub-spectrum centered on DC with no carrier and a bandwidth of 30 kHz. This part of the spectrum carried half of the sum of the left and right sound records. The stereo spectrum is thus composed with a bilateral suppressed sub-carrier sub-spectrum centered on 38 kHz with a 30 kHz bandwidth. This sub spectrum carries half of the difference of the left and right sound records. A 19 kHz pilot tone transmits in the spectrum, which has a phase relation to the 38 kHz suppressed sub-carrier, can be use to regenerate the 38 kHz for stereo FM demodulation. In this waveform, a set of decimation filters has been used to extract from the bilateral stereo broadcast's spectrum the part of spectrum which is carried half of the sum of the left and right sound records.

2.2. Description of the FM receiver waveform

The complete FM receiver waveform, shown in Figure 1 is used to extract the mono part of the stereo signal, i.e. half of the sum of the left and right sound records described in the last sub section. This waveform has been tested over the two's platforms OSSIE and GNU Radio.

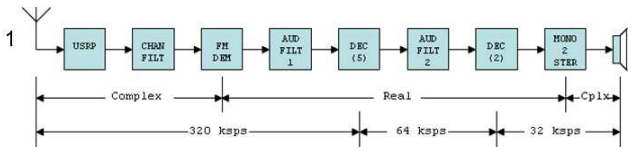

Figure 1 - FM receiver waveform

This waveform is composed with:

- an access block to the URSP,
- a set of fir decimation filters ,
- one frequency demodulator,
- one access block to the sound board of the computer

On the **USRP** block, the sampled signal is transposed to baseband. This block configures the DDC and the decimation factor of the USRP mother board filters to obtain the desired rate and carrier frequency. The first low pass filter called **CHAN FILT,** which has a 100 kHz bandwidth, preserves the integrality of the bilateral stereo broadcast's band received by the USRP. The **FM DEM** block is used to demodulate the frequency modulated signal. The C++ source code of this block calculates the phase between *I* and *Q* samples described in the last sub section. The other low pass filters called **AUD FILT 1** and **AUD FILT 2** make the rest of the filtering process. The filters are separated with decimators **DEC 1-2** to limit the filters orders. The last component of all the waveforms is the **SOUNDCARD** which is represented by a speaker on the Figure.
The access block to the computer's sound board is different on the two platforms. On the GNU Radio side, there are two inputs of type float to access to the left and right channels of the sound board whereas in OSSIE the block has just one complex input to the same two channels. It is the reason why on the OSSIE waveform it is necessary to use a block called **MONO 2 STER** to copy the demodulate signal onto the real and imaginary part of the block input. The input rate to sound board blocks is fixed to 32 ksps for both platforms.

2.3. OSSIE and GNU Radio software suite

2.3.1. OSSIE

OSSIE is an object-oriented SCA OE for which signal processing components are written in C++[1]. This OE works on a Linux OS and the software's dependencies are shown on Figure 2. This entire stack is used by OSSIE during the waveform execution. Each part of the stack impacts the memory and the host computer's load. To know the overhead produces by CORBA's communications (principally OmniORB [10]), the distribution of this software's stack on the host computer must be measured. Every OSSIE's component can be considered having two distinct parts: one part realizing the signal processing and another managing the SCA infrastructure. As in all SCA OE, the OSSIE waveforms are described in an XML file.

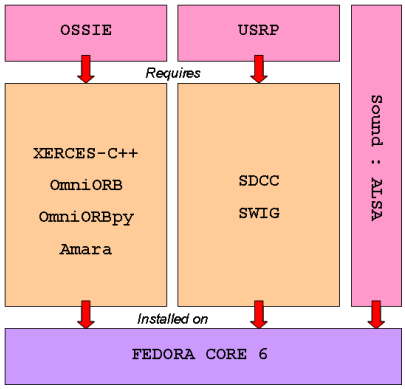

Figure 2 - OSSIE dependencies

2.3.2. GNU Radio

GNU Radio project provides a Python library of signal processing blocks and specially a class `gr.flow_graph` to tie together the signal processing blocks of waveforms. Theses blocks are executed by the computer on which GNU libraries have been installed. To minimize the critical execution time of a complete waveform, blocks are written in C++. The interconnection blocks graph (IBG) can be compared to the OSSIE waveform. The python application script uses the `gr.flow_graph` class that gives to the waveform designer the ability to describe a data flow graph of the waveform he wants to describe. To wrap C++ blocks in Python the script must import the extension modules SWIG (Simplified Wrapper and Interface Generator [11]). The next subsection describes different granularities of the FM demodulation.

2.4. Granularity variation of the OSSIE waveform

[1] Each component can be written with C++ template generated by the "WaveDev" tool of the OSSIE's suite

The granularity is related to the portability of the waveform which is one of the goal of SCA and one of the key challenges today in waveform design. The granularity of the waveform is investigated in this section to study its impact in terms of performances.
The Figure 3 illustrates the five different variations of the granularity, of the original Figure 1 waveform that we consider in this study. They are numbered from the fine-grained waveform WF1 with nine components to a coarse-grained one (WF6) with just three components.

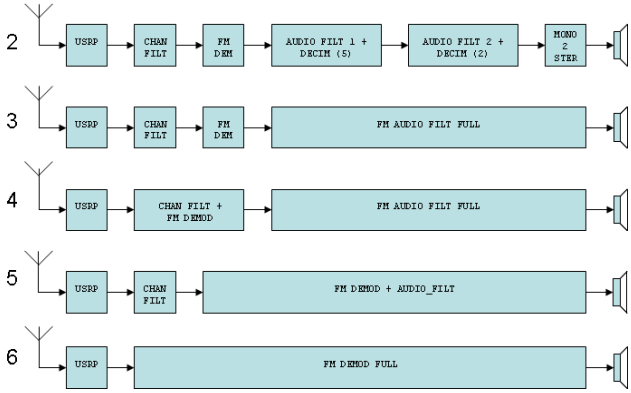

Figure 3 - Waveforms refinements

2.5. Waveforms measurements performances

In OSSIE, every component is considered like a process by the OS. With a simple "top" Linux system's command we can see a process for each component with their CPU and memory utilization. In GNU Radio it is not possible to see the different components when using this same command. All GNU blocks are gathered under one Python process. Different metrics have been defined to characterize OSSIE and GNU Radio. Indeed, it is very important to know how these two systems work and the differences between them.
The first parameters to know are the needs of the waveforms in terms of CPU and memory allocation. In a first approach, the "top" Linux system's command can show an estimation of the instantaneous CPU and memory usage of all the components of the waveform. As mentioned above, doing this test with GNU Radio does not have a strong interest because of the use of Python. On the other side, doing this measure with OSSIE can give some preliminary answers.
In order to have more accurate results, we use a profiler. This kind of software allows the user to know precisely the CPU utilization of every process which runs on the system (including the system himself if specified optionally). As in [6][12], OProfile [13] has been used to characterize the OSSIE host computer's memory and load. For GNU Radio, it's the Hotshot [14] profiler which has been used to characterize these same parameters. Indeed, due to the only

process created by the GNU Radio waveform application, the use of OProfile is not relevant to obtain the desired metrics. On the other hand, the Hotshot library of Python does not allow seeing the global system but only what happens inside the Python script.
The other crucial parameter to know in the system is the latency caused by the inter-components communications [15][16]. To realize the latency measurement, an assembly routine has been used: RDTSC [17] (ReaD Time Stamp Counter). This counter is incremented at the CPU's frequency. It allows us to know accurately the time elapsed between two calls of this counter (with a CPU frequency of 3GHz, the counter accuracy is about 333fs). In a OSSIE component, there are two functions for receiving and transmitting the data (respectively `Get()` and `Push_Packet()`). In one component the measure is done before `Push_Packet()` and in the next component of the waveform, the measure is realized after `Get()` as shown in Figure 4. The time elapsed between the two measures, is obtained by a simple difference of the two counter's values.

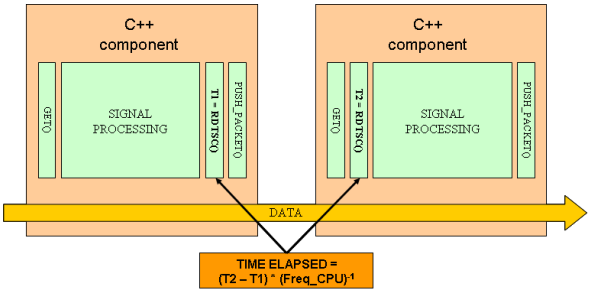

Figure 4 - Latency measure with OSSIE

In GNU Radio, The counter read is realized before the `return` command for one component and after the variables declaration for the other.
The measure of these metrics is presented on the next section. They have been obtained with a Pentium 4 with a clock speed of 3 GHz and 1 GB of memory working with the Linux distribution Fedora Core 6.

3. RESULTS ANALYSIS

The methodology for the measures have been presented in the previous section, this section will present the results. Most of them concern OSSIE, the GNU Radio's measures have been done to allow the comparison but OSSIE have a bigger research interest for us because of the use of SCA.

3.1. CPU and memory utilisation

The CORBA's level of SCA seems to bring some overhead when running a waveform. Measuring the CPU load and the memory usage is a good point to start this study. First of all, it is possible to look at these two parameters with a simple "top" command on Linux. Table 1 shows the results for all the waveforms (for a packet size of 8192 samples). These measures have been carried out on the five waveforms described earlier for OSSIE in Figure 3 and for the WF2 waveform of the Figure 3 for GNU Radio. The results displayed are the CPU load and the memory utilisation of the entire waveform. A simple calculation is realised to obtain the average value of the memory used by one component for each waveform. With these results, it is possible to see some differences between waveforms. The value of CPU usage is obviously not the same for all of these waveforms but the variation is not linear. For waveforms WF3 and WF4, the CPU usage is higher than WF1 and WF2 which have more components. With GNU Radio, the CPU uses 6% of its capacity to execute the Python process of the WF2 waveform. So, for the same waveform, the CPU requires five times more capacity with OSSIE than with GNU Radio.

Waveforms Number	%CPU	%Memory[1]	Number of Components[1]	%Mem / Comp[1]
1	36	4.4	7	0.63
2	32	3.2	5	0.64
3	47	2	3	0.67
4	45	1.3	2	0.65
5	28	1.4	2	0.7
6	22	0.7	1	0.7

Table 1 - OSSIE hardware resources utilisation
[1] without USRP and SoundCard components

With this table, it is also possible to see the memory usage. In GNU Radio, the measured value for the waveform number 2 is 1.7%. By calculating the relation between the number of OSSIE components and the memory usage, it appears that every component uses approximately 0.7% of the host memory. Increasing the components number will inevitably increase the memory usage of the waveform.
But a simple Linux command can not reveal all the parameters necessarily to know entirely the both systems. So, to understand this difference, profiling deeply OSSIE is necessarily. This operation is realised with OProfile. The Figure 5 presents the percentage of CPU load in a component which is used by the SCA infrastructure management part (non-signal processing part). with a variation of the data encapsulation's packet size. This size, defined by an attribute of the USRP component is a power of two because of a restriction of OSSIE, and the minimal size is 128 samples per packet due to USB restriction [16]. Three components are presented: one with a significant amount of signal processing (**CHAN_FILT**) and two with a tiny signal processing part (**DEC** and **MONO_2_STER**). For the filter, the measured value is under 10% when the

packet size exceeds 8192 samples but for the two others components, even when the packet size is very high (ie. more than 32768), the value of non-signal processing CPU load stays over 70%. This result shows the importance of having a certain amount of signal processing inside a component in order to limit the effect of the SCA infrastructure management part.

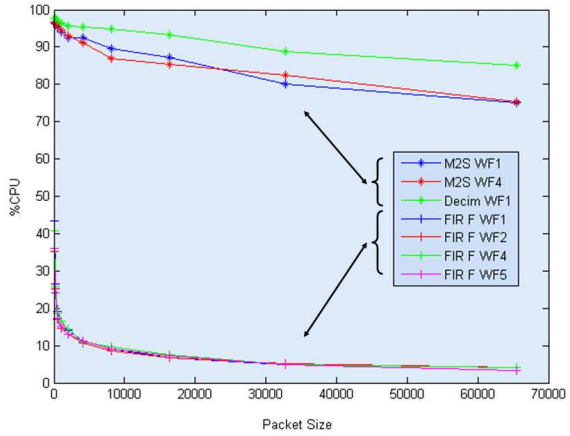

Figure 5 - Non-signal processing CPU load

With GNU Radio, profiling the application is totally different. Hotshot which is another profiler is used to see every function call made by the application and the corresponding CPU time. Because the waveform is contained in only one process, Hotshot gives just an outline of how GNU Radio works. This measure tells that the function, which used the bigger part of the CPU is the python script itself. These measures have only been realised with one packet size which is 3584 samples (default size in GNU Radio). In a first approach, it can be noticed that it is easier to study how OSSIE (multi-thread) works relatively to GNU Radio (mono thread).

There is another point which can not be explained. When profiling the **USRP** component with OSSIE, the percentage of CPU usage of the non-signal processing part falls when the packet size is 32768 samples (ie. 128 kB) as shown in Figure 6. For higher size, the measure returns to a value which is in the continuity with the other measures. This anomaly has been obtained for all waveforms of the Figure 3. It can be interpreted like the optimal packet size for a waveform using the USRP.

As shown above, for the same waveform, the ratio between OSSIE and GNU is 5 for the CPU load and 2 for the memory. It is incontestable that OSSIE is heavier than GNU Radio in terms of hardware resource demands. So, it is interesting to see if there is the same trend with the inter-components latency.

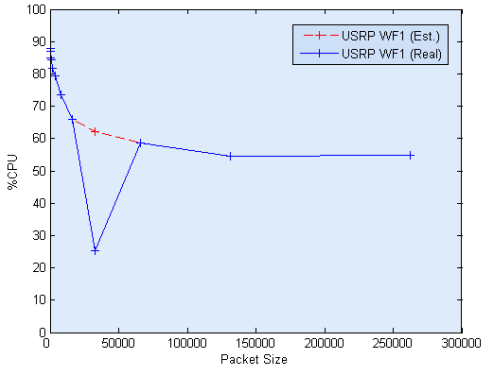

Figure 6 - USRP Component (OSSIE): non-signal processing part CPU load

3.2. Latency

This parameter has been measured between the first filter and the FM demodulator components. This channel appears on waveforms WF1, WF2 and WF3 in OSSIE and also in the GNU Radio waveform. The results are presented in two figures. The Figure 7 shows the latency for packet size from 128 samples (256 for WF1, 128 do not work due to an overload of the CPU) to 65536 samples. The Figure 8 is a zoomed part of the Figure 7 for packet size inferior to 4096 samples; it is also possible to see on these figures the measured value for GNU Radio.

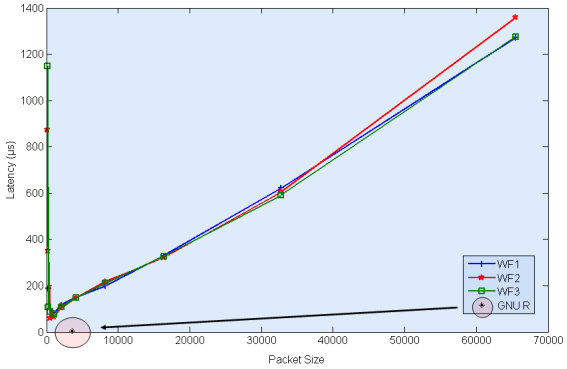

Figure 7 - Inter-components latency

While observing the curves obtained, a linear comportment can be seen for high packet size. The increase of the latency for little packet is explained by the increase of CPU load. If the packet contains a lot of samples, there are less communications in terms of `Get()` and `Push_Packet()` request, so the CPU is less solicited. It can be noticed that the three OSSIE waveforms have the same comportment. The measure done with GNU Radio gives a result totally different: for a packet size of 3584 samples, the latency between the end of the filtering and the beginning of the FM

demodulation is around 5.7µs. With OSSIE, the latency is around 146µs for 4096 samples per packet. According to these measures, the inter-components communication in GNU Radio is 25 times faster than in OSSIE. This can be explained by the use of OmniORB. Indeed, it is widely admitted that CORBA leads to an important latency in real time system communications. It is in particular due to the implementation of the GIOP/IIOP where the CORBA overhead comes in addition to the Ethernet TCP overhead [20]. In GNU Radio, the inter-components communication can be compared to a simple parameter transfer when in OSSIE, there is a real protocol to transmit and receive data.

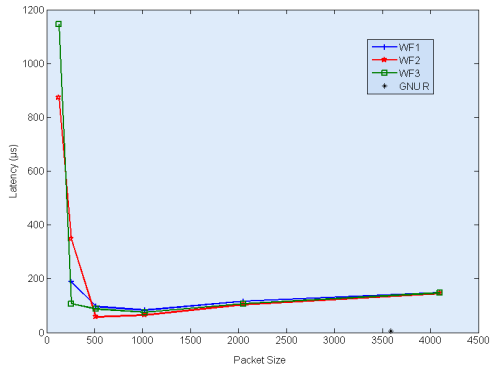

Figure 8 - Inter-component latency (small packet size)

4. CONCLUSION

We have started this work to acquire a deep knowledge of the SCA and to understand the pros and the cons of this standard. This work allows a better understanding of the CORBA impact in OSSIE (and in SCA in general) and the limits in terms of signal processing and packet size for more reliable design of our future SCA based system architecture. The latency and the computer resources usage are primordial performances in a real-time radio system. We show in this paper that the granularity study has consequences on the performances so it has to be considered as an important parameter when designing a waveform. With the results obtained up to now, a fine grain implementation of a full duplex waveform like GSM is not realizable now due to CORBA. The first thing to do is to think at a communication system which can alleviate CORBA and increase consequently the performances of a SCA based radio system. While increasing the communication architecture of the hardware platform, it will be able to take advantages of CORBA and accelerate its GIOP protocol.

5. FUTURE WORK

The good knowledge of SCA through our OSSIE experimentations will help us to investigate further in details in our future work the implementation of CORBA on a Network On Chip to take advantages of this kind of architecture to increase the CORBA performances and especially the mapping of the GIOP onto the NoC powerful hardware transport mechanisms.

6. REFERENCES

[1] "Software Communications Architecture Specification", Final/15 May 2006 V.2.2.2, JTRS Standard, Joint Program Executive Office of the Joint Tactical Radio System, http://jtrs.spawar.navy.mil/sca
[2] CRC SCARI Open,
http://www.crc.ca/en/taxonomy/term/379
[3] OSSIE, Open Source SCA Implementation :: Embedded,
http://ossie.wireless.vt.edu/
[4] GNU Software Radio project.
http://www.gnu.org/software/gnuradio/
[5] Python, http://python.org
[6] P.J. Balister, M. Robert, .J. H. Reed, "Impact of the use of CORBA for the Inter-Component Communication in SCA Based Radio", SDR Forum proceedings 2006.
[7] Ettus Research LLC, http://www.ettus.com/
[8] SDR Documentation, Dawei Shen
http://www.nd.edu/~jnl/sdr/docs/
[9] Rodney G. Vaughan, Neil L. Scott, and D. Rod White, "The Theory of Bandpass Sampling", IEEE Transactions on Signal Processing, Vol 39, NO. 9, September 1991
[10] OmniORB : Free High Performance ORB,
http://omniorb.sourceforge.net/
[11] SWIG, http://www.swig.org/
[12] P.J. Balister, C. Dietrich, .J. H. Reed, "Memory Usage of Software Communication Architecture Waveform", SDR Forum proceedings 2007.
[13] OProfile, http://oprofile.sourceforge.net/
[14] Python library reference, Part 25.8, "Hotshot -- High performance logging profiler", http://docs.python.org/lib/module-hotshot.html
[15] Thomas Tsou, Philip Balister, Jeffrey Reed, "Latency Profiling for SCA Software Radio", SDR Technical Conference 2007
[16] Thomas Schmid, Oussama Sekkat, Mani B. Srivastava, "An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radio", WiNTECH'07, September 2007
[17] P. Work, K. Nguyen, "Measure Code Sections Using the Enhanced Timer," Intel Corporation, Tech. Rep.
[18] G. Gaillard, H. Balp, M. Sarlotte, F. Verdier, "Mapping Semantics of CORBA IDL and GIOP to Open Core Protocol for Portability and Interoperability of SDR Waveform Components", Design Test and Automation 2008 Conference, DATE08.
[19] Rahul Dhar, Gesly George, Amit Malani, Peter Steenkiste, "Supporting Integrated MAC and PHY Software Development for the USRP SDR", IEEE Workshop on Networking Technologies for Software Defined Radio (SDR) Networks, September 2006
[20] Milan Zivkovic, Chunhui Liu, and Rudolf Mathar, "Implementation of OFDM Power Allocation Strategy in GNU Radio Framework", 5th Workshop on Software Radios, March 2008, Karlsruhe Germany