SDR IMPLEMENTATION ISSUES: RF FRONT END NONLINEARITY AND DYNAMIC COMPUTING RESOURCE ALLOCATION

Feng Ge and Charles W. Bostian
Virginia Polytechnic Institute and State University, Wireless @ Virginia Tech, Center for Wireless Telecommunications, Virginia Tech, Blacksburg, VA 24061, USA;
{gef, bostian}@vt.edu

ABSTRACT

This paper addresses two practical issues governing Software Defined Radio (SDR) performance that are often overlooked: RF front end nonlinearity and dynamic computing resource allocation. While the effects that we discuss are certainly well known in some parts of the SDR community, they frequently come as a surprise to inexperienced radio designers. These individuals typically begin with the Universal Software Radio Peripheral (USRP) and GNU Radio, so in this paper we use these platforms as the basis for our analyses and experiments.

Current SDR performance still depends heavily on analog radio frequency (RF) technologies. Intermodulation and other nonlinear effects in these devices make it very challenging to create an RF front-end that is applicable to a variety of signals with widely differing center frequencies, modulation bandwidths, and power levels. Unanticipated intermodulation products can seriously degrade receiver performance. Digital signal processing in wireless communication is fundamentally a real-time task. Achieving real-time performance in SDRs puts stringent requirements on dynamic computing resource allocation; these requirements may be much higher than those in conventional digital radios. We explain why and investigate the impact of computing resource allocation on SDR performance.

1. INTRODUCTION

The software radio concept is built upon the use of reconfigurable (programmable) hardware whose operation can be changed through software modifications [1]. SDR's one inherent goal is to *move digital signal processing functions progressively closer to the radio antenna* [1]. Doing so paves the way to achieving SDR's promise of *multi-band multi-mode reconfigurability* by using software methods to replace analog functions. However, currently it is very challenging to create an RF front-end and associated AD/DA converters that are applicable to a variety of signals with widely differing center frequencies, modulation bandwidth, and power levels [2]. Therefore, SDR's current performance still depends heavily on analog RF technologies. For example, RF front-end performance metrics, particularly non-linearity and dynamic range significantly limit SDR's.

Digital communication systems [3] are fundamentally real-time because data are transmitted as segments at the PHY layer; to receive all the data, the time to process each data segment is limited since the radio must accept and process each incoming data segment before the next one arrives. Such requirements are stringent in the digital domain when signal processing functions are complicated and there is limited battery life to support sufficient computing resources. It is possible to achieve both real-time performance and low power consumption for a single digital waveform because (1) the RF radio signals can be down-converted to the baseband with a necessary minimum data rate, thus minimizing required computing resources; (2) the required digital signal processing functions can be fully optimized and executed on Application-Specific Integrated Circuits (ASICs). Most commercial radios are produced, (cell-phones for example) in this way.

However, it is quite a different story for SDR. First, moving digital signal processing functions closer to the radio antenna requires a high data rate and more functional components in the software domain; this implies additional computing resources such as computing cycles in signal processing and memory space to hold more signal samples. Second, SDR usually needs to reconfigure itself to support more than one application; this makes it very challenging to highly optimize computing resource allocation for dynamically configured radio functions. Third, developing and executing SDR systems requires a running computing system. Such a system consumes computing resources, maybe at a significant level, without even running any SDR code.

Among the three most popular computing systems for SDR development – field-programmable gate arrays (FPGAs), general purpose processors (GPPs), and digital signal processors (DSPs) – FPGAs are difficult to reconfigure and consume significant power, but they can support real-time performance [4]. GPPs are capable of reconfiguration but are power demanding and also suffer from execution latency [5]. DSPs consume less power and can reconfigure for

simple real-time tasks but are not able to support computationally intensive tasks [1].

The emerging cognitive radios (CRs) [6] require not just radio platform reconfiguration, but also the abilities to learn and to adapt. This further complicates the above issues if SDR is used as a supporting platform. For example, nonlinearity not only impacts radio performance [7], but also fundamentally limits CR's application in dynamic spectrum access (DSA) [8] and other fields [9]. CR also may be aware of its computing resources [10]. However, how to dynamically allocate and manage computing resources for different functions with real-time performance while satisfying some power constraint is very challenging [11].

2. SDR RF FRONT END NONLINEARITY

RF front end nonlinearity severely impacts radio performance. Fundamentally, nonlinearity comes from nonlinear elements like diodes and transistors that are used in RF amplifiers and mixers. If an input signal, for example, a sinusoidal signal $A = \cos(2\pi f_0 t)$ has a power level that lies beyond the linear region of such devices, the output signal will include frequency components at f_0 as well as at $2f_0$, $3f_0$, etc. Further, if there exists more than one input signal with one or more having power levels beyond the linear regions of such devices, they will interact with each other and degrade the overall performance on each signal. There are mainly three nonlinearity problems: inter-modulation distortion, desensitization, and cross-modulation [1]. In this section, we present some analysis on the impact of RF nonlinearity on SDR performance through USRP and GNU Radio [12].

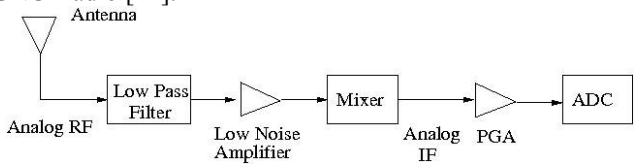

Figure 1. The RF front end in the USRP.

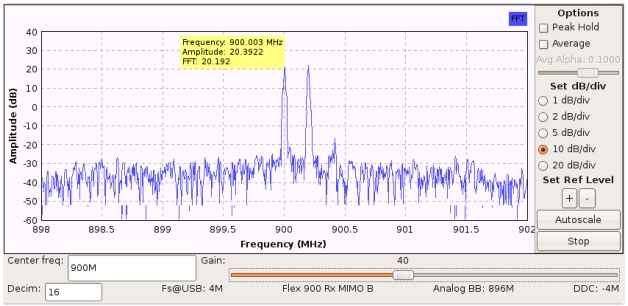

Figure 2. Two signals at a low receiving gain.

2.1. Types of Non-linearity Distortion

2.1.1. Inter-modulation

The RF front end is shown in Figure 1, where GNU Radio controls the receiver gain through the programmable gain amplifier (PGA) before the ADC. To illustrate the inter-modulation, we set up two transmitters with the same power levels and at close center frequencies (900MHz and 900.2MHz) as in Figure 2. Both transmitted GMSK modulated data with a baseband data rate of 50vB/s. A receiver captured 4MHz bandwidth of signal centered at 900MHz. When we set the receiving gain in the PGA at 40% of the maximum value, only two signals were shown in the frequency domain. However, as we increased the receiving gain to 45%, multiple signals appeared, as shown in Figure 3. The additional signals are third and fifth-order intermodulation products that appear in the IF passband and cause downstream interference. At the receiver output they are indistinguishable from unwanted signals that exist at the input.

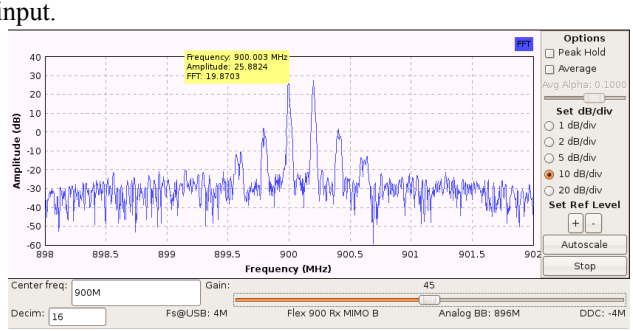

Figure 3. Apparent input signals under high gain conditions. Only the largest two actually exist at the antenna

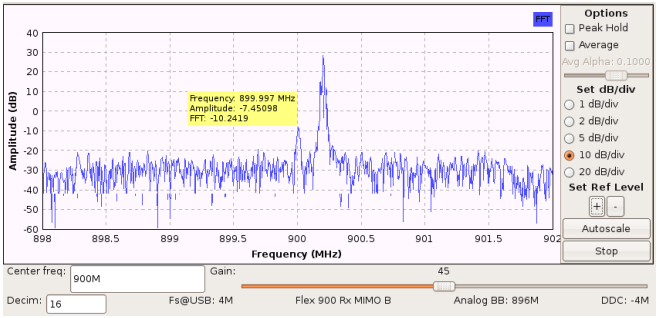

Figure 4. Two signals with different transmitting powers.

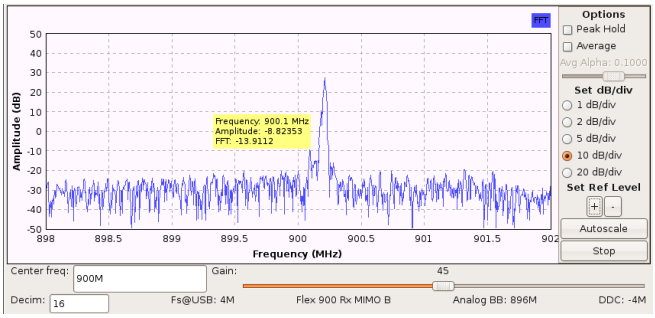

Figure 5. Adjacent channel interference.

2.1.2. Adjacent Channel Interference

We also studied the problem of adjacent channel interference (ACI). This is particularly noticeable when the receiver must simultaneously process multiple independent signals or when the weak signal is adjacent to a strong signal from a nearby transmitter. To demonstrate ACI, we decreased one signal's power level 32 dB lower than the other signal, as shown in Figure 4. We then moved the center frequencies of the two signals closer at a separation of 100 KHz as shown in Figure 5. We can see that the frequency selectivity of the receiver is degraded as the energy from the strong signal leaks into the bandwidth of the weak signal.

2.2. Two Application Examples

2.2.1. Nearby Channel Data Communication

An SDR can't set its receiver gain too high because of non-linearity as shown in Figure 3. However, if the receiver gain is too low, the received signal's SINR might be too low, therefore resulting in high BER and packet error rate (PER). Here we used GNU Radio and USRP 1 (with a FLEX 900 daughterboard) to characterize PER variation against SINR. Instead of varying the receiver gain, we change the transmitter power over a large range because it is a more appropriate way to vary SINR received at the receiver. (The noise level remains constant.) The receiver gain was set at one third of the maximum value at the PGA in Figure 1. In calculating PER, the receiver received 5000 packets and each one had an error checking header. The SINR is the effective input SINR as measured in GNU Radio and thus including all of the effects of the receiver operations and components. Figure 6 shows the results and it also indicates the SINR where inter-modulation begins when the receiver is processing equally strong signals on adjacent channels.

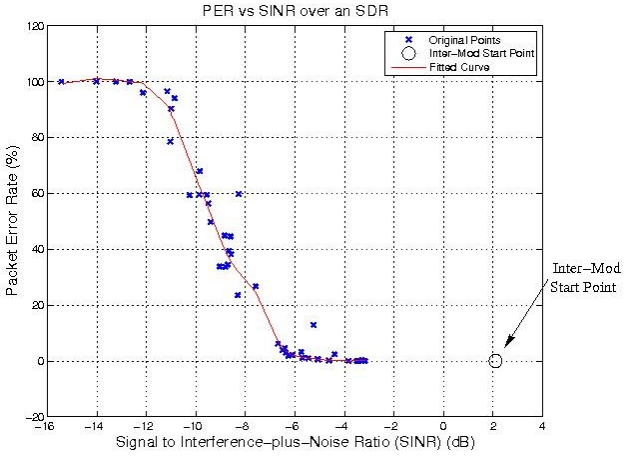

Figure 6. PER vs. SINR distribution.

The calculated SINR was determined by USRP's architecture because the signals used in calculation went through function components such as filtering, amplification, and gain control. However, the same signal samples were also used in demodulation; therefore, Figure 6 does demonstrate that non-linearity and SINR significantly limit the proper operating range for received signals at different power levels. In this example the effective dynamic range of the receiver is only about 7 dB.

2.2.2. RF Signal Detection

Inter-modulation also impacts signal detection over a wide frequency range, particularly for the detection of signals at a low power level. This is especially important for cognitive radio and DSA applications [7]. To guarantee non interference to primary users, a sensor is needed to detect even weak signals. To do so, the sensor has to set its receiving gain very high. However, a nearby strong signal might cause inter-modulated signals. Without further differentiation, such signals will be treated as primary users, artificially limiting available frequency bands.

3. DYNAMIC COMPUTING RESOURCE ALLOCAT ION IN SDR

Fundamentally, any radio system must guarantee a real-time performance. From the perspective of network data communication, data are transmitted as segments at the PHY layer; to receive all the data, processing time for each data segment is limited — determined by the transmitting data rate — since the radio must accept and process this data segment before the next segment arrives. Therefore, the overall signal processing in a radio system must have this deadline-driven constraint [13] — they must be completed within a certain time or radios may not function appropriately. Following the receiver path chain, the antenna and other analog parts can process signals of any bandwidth at an almost instant speed. Therefore, the real-time performance constraint mainly lies in the digital part.

After the ADC, a minimum amount computing resource, for example, computing cycles and memory space, is needed to process the digital samples in achieving a needed real-time speed that matches the incoming digital samples' speed. Different waveforms require different amounts of computing resources; for example, 16 QAM requires more computing resource than BPSK in achieving the same symbol rate. In conventional digital radios, the analog-to-digital conversion occurs either at the baseband or at a low IF band, which dramatically reduces the speed of the digital signal samples. *Therefore, the baseband data rate is usually at the same order of the sample rate after ADC*. Thus, *the real-time performance requirement of conventional digital radios can be equally determined by the baseband data rate*. In commercial digital radios like a DSP based P25 radio, the computing resource requirement for all P25 radio tasks is

thoroughly quantified, based on the selected waveforms and the baseband data rates to achieve real-time performance. A DSP with required computing resources (in terms of CPU and memory), usually with some leeway is dedicated to the corresponding signal processing functions that correspond to such waveforms and data rates.

3.1. Support A Real-time Performance in SDR

SDR is quite different from conventional digital radios in that it usually captures signals samples at a high IF frequency and sends them to the software domain. The result is that the digital domain has to deal with a much higher sample rate after the ADC than the baseband data rate – sometimes several orders of magnitude higher. For example, a P25 radio may only require a data rate of 9.6 KB/s [14], but an SDR requires a sample rate of at least 1 MS/s if it captures signals with a bandwidth of 0.5MHz; about a 100 times higher data rate. Depending on the type of digital signal processing functions after ADC and before baseband signal processing, processing such a much higher sample rate in the digital domain may require more computing resources than processing a specific waveform at the baseband. This is certainly true for required memory space and its associated memory operations. Therefore, *the real-time performance requirement of SDR is determined by both the IF band sample rate and the baseband data rate for a specific waveform.* For simple waveforms, processing the IF band sample rate requires far more computing resources than the baseband data rate in achieving the required real-time performance and dramatically increases the associated power consumption, which fundamentally limits SDR's applications.

Further, unlike conventional digital radios which often dedicate different optimized computing devices for different parts of a system, SDR accommodates many digital signal processing functions in a single computing domain and targets many radio applications. Not only does this limit the overall system optimization, but also it complicates the system performance analysis because all running functions compete with each other for computing resource. Such competition is particularly complicated in GPPs and DSPs because CPU and memory consumption for multiple functions can be non-linear in most operating systems. This creates a big challenge in analyzing CPU and memory requirements for allocating resource dynamically in achieving real-time performance. While FPGAs can, in theory, allocate computing resources – the programmable gates -- in a linear manner for different functions, the resource usage can't be optimized, resulting in a waste of power consumption.

Further, GPP-based SDR has execution latency issues because of the memory hierarchy in GPPs, the peripherals connecting the RF front end to GPPs, and multiple data buffers along the radio transceiver signal processing chain [5]. The execution latency in GPP based SDR further limits its ability to support real-time performance.

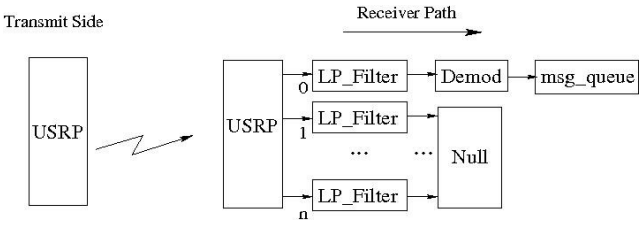

Figure 7. Experiment set up.

3.2. Experimental Investigation of Dynamic Computing Resource Allocation in SDR

3.2.1. Experiment Settings

In this section, we use a concrete example to illustrate the impact of computing resource allocation on the real-time performance requirement in GNU Radio (version 3.1 as the present paper was written) – a GPP based SDR system. As shown in Figure 7, we set up a direct radio link between two nodes by continuously transmitting a fixed amount of data from the transmitter to the receiver using GMSK. The data is transmitted in fixed-length packets at a fixed rate. In the receiver path, the USRP sends digital samples at an IF band to its connected GPP. In our experiment, the sampling rate fed to the GPP is fixed at 4 MS/s and the baseband data rate is 50 KB/s with 2 samples per symbol. The samples are then low-pass filtered before demodulation. The final baseband data is saved in a message queue. All the function blocks in Figure 7 are implemented in terms of signal processing blocks which require CPU cycles and memory space to do their work. More details about how GNU Radio works is available in [5]. To analyze the relationship between computing resource allocation and real-time performance, we also added a variable number of copies of the same low-pass filter after the USRP, but feeding their output samples to a null block which simply dumps the data. The low-pass filter is a finite impulse response filter using Hamming window with 1241 taps. The GPP we used was an Intel Duo CPU (model T9300) with a frequency of 2.50GHz. It has 3.5 GB of memory.

3.2.2. Experiment Results and Analysis

In this experiment, the receiver had a basic signal path including the signal processing blocks of USRP, demodulation, and only one low pass filter. It needed to receive 5000 packets; the overall computing resource consumption in receiving those packets and the PER were calculated. We then added a variable number of extra low-pass filters as shown in Figure 7 and measured both the

computing resource consumption and PER accordingly. Specifically, we use four methods in estimating the computing resource consumption. We use the package SYSSTAT [15] and Oprofile in measuring the overall user domain and individual GNU Radio function CPU utilization, the Python Profiler [16] in measuring the overall CPU time to process all received packets, and the Linux system monitor in measuring the memory consumption. The Python Profiler uses deterministic profiling by precisely timing the intervals between events such as function call, function return, and exception events [16] while SYSSTAT and Oprofile use statistical profiling by randomly sampling the effective instruction pointer and deducing where time is being spent. To minimize the OS impact, we restarted the computer in a clean environment for each running case.

Figure 8 shows the variation of overall CPU consumption in the user domain, including both GNU Radio and other running system programs, as we increase the number of extra low pass filters. By total CPU, we mean the total available CPU cycles of a computing device. The CPU consumption is around 2% of all available CPU cycles without running GNU Radio. Though not shown in the figure, we also found from the Linux system monitor that the memory utilization remains almost constant around 397MB (11% overall utilization) when running GNU Radio and 373 MB (10.5% overall utilization) when not running GNU Radio. Therefore, memory consumption has a negligible impact in our experiment.

As we can see from Figure 8, no trend seems to exist purely for the overall user domain CPU utilization. There may be multiple reasons to explain this. Fundamentally, GPP's CPU has limited hardware resources, for example, registers and ALUs. They are shared among different tasks as the OS assigns resource and CPU clocks for each running task. There is variation for the overall system CPU utilization and such variation may obscure the CPU increment introduced by running extra filters solely in GNU Radio.

To further investigate the variation of computing resource consumption, we then specifically analyze the CPU consumption of the low pass filters by using Oprofile because the filter operation is the main CPU consumption increment among all running cases. As shown in Figure 9, the CPU consumption of all running low pass filters increases linearly to a total of 7 filters, and then drops at 8, followed by a continuous increase afterward. Two lines are shown in Figure 9; one is the CPU percentage of all running filters against the overall running programs in the user domain, the other is the CPU percentage of all running filters against all the available system CPU. We can see that each extra filter only consumes about 0.3% of the overall CPU; that result confirms our estimation in the previous paragraph. Because now we can confirm that the required CPU increases linearly at least for the first 7 running cases, next we study the impact of CPU consumption increase on SDR performance -- here the PER. As shown in Figure 10, the PER for the first 7 running cases is, in almost all cases 0, then it increases abruptly to almost 100%.

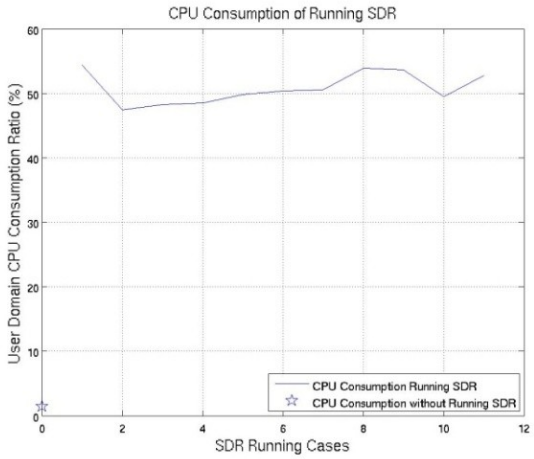

Figure 8. User domain overall CPU consumption variation.

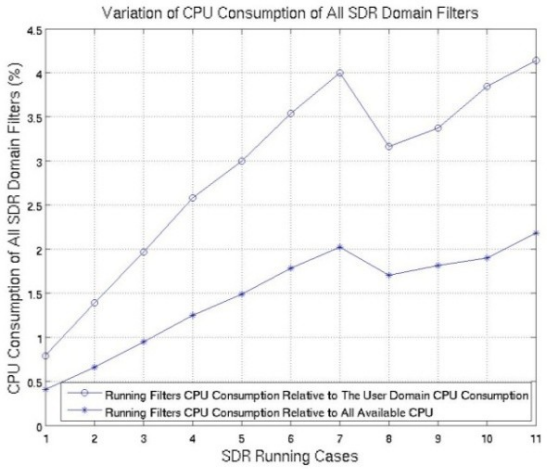

Figure 9. CPU consumption variation of different number of low pass filters.

To further investigate the CPU consumption, we used Python Profiler to measure the overall timing requirement in receiving 5000 packets. We found that the required time is around 96.5 seconds for all the first 7 running cases, and then it increases sharply for following cases, as does the PER correspondingly. We also found that the receiver path, starting from running case 8, experienced a significant number of USRP overruns (By this we mean that USRP samples are dropped because they were not read in time by the following signal processing block.). In such situations, the transmitter actually needed to send more data packets so that the receiver could receive up to 5000 packets. Therefore, the sharp increase in CPU time increase indicates the increase in the number of dropped signal samples.

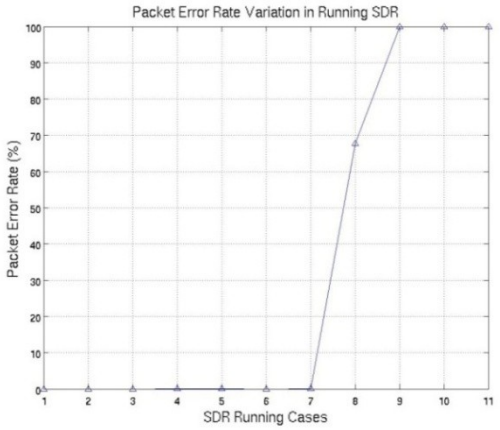

Figure 10. PER distribution as the number of low pass filters increase. This illustrates how the PER increases once the CPU runs out of time to do the number of computations associated with a single packet.

Given the above data results, we now can understand why the CPU percentage of all running filters in Figure 9 doesn't increase linearly after the number of 7. GNU Radio execution slows down if signal samples are dropped once in a while and the CPU consumption dynamics thus change. But most importantly, *at a threshold value, a small amount of required CPU increment may result that an SDR receiver performance deteriorates abruptly because it can't allocate enough computing resources to processing the incoming digitized samples at the required real-time speed*.

4. CONCLUSION

We studied the non-linearity of SDR's RF front end and its impacts on the SDR's performance. Through the examples of GNU Radio and USRP, our experiment showed that inter-modulation significantly limits SDR's receiver gain operation range. Further, we concluded that SDR may require much more CPU resource and battery power than conventional digital radios. Very importantly, SDR performance may deteriorate abruptly if not enough computing resources are available because of the real-time constraint. This has a significant implication for SDR development: *the computing resource requirement of all running applications must be thoroughly quantified under all possible execution situations and the SDR must provide sufficient resources for executing at least the maximum case*.

5. ACKNOWLEDGEMENT

This project is supported by the National Science Foundation (NSF) under Grant No. CNS-0519959 and by the National Institute of Justice, Office of Justice Programs, U.S. Department of Justice under Award No. 2005-IJ-CX-K017 . The opinions, findings, and conclusions expressed are those of the authors and do not necessarily reflect the views of NSF and the Department of Justice.

6. REFERENCES

1.Reed, J.H., *Software Radio: A Modern Approach to Radio Engineering,* . 2004, Englewood Cliffs, NJ: Prentice-Hall.
2. Le, B., et al., *Analog-to-digital converters*, in *IEEE Signal Processing Magazine*. 2005. p. 69-77.
3. Proakis, J.G., *Digital Communications*. 4 ed. 2000, New York: McGraw Hill.
4. Fong, R.J., S.J. Harper, and P.M. Athanas. *A versatile framework for FPGA field updates: an application of partial self-reconfiguration*. in *Rapid Systems Prototyping, 2003. Proceedings. 14th IEEE International Workshop on*. 2003.
5. Ge, F., et al., *Software Defined Radio Execution Latency*, in *Software Defined Radio Technical Conference*. October, 2008: Washington D.C.
6. Mitola, J., *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. 2000, Royal Institute of Technology (KTH).
7. Marshall, P.F., *Cognitive Radio as a Mechanism to Manage Front-End Linearity and Dynamic Range.* IEEE Communications Magazine, 2009. **47**(3): p. 81-87.
8. Daniel DePardo, J.B.E., James A. Roberts, Victor R. Petty, Alexander M. Wyglinski, Paul J. Kolodzy, and Michael J. Marcus, *Observations of Potential Secondary User Device Effects on Digital Television Receivers*, in *Technical Report ITTC-FY2008-TR-41420-06*. 2007.
9. ElNainay, M.Y., et al. *Channel allocation for dynamic spectrum access cognitive networks using Localized island Genetic Algorithm*. in *5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops (TridentCom 2009)*. 2009. Washington D.C.
10. Ge, F., et al., *Cognitive Radio: From Spectrum Sharing to Adaptive Learning and Reconfiguration*, in *2008 IEEE Aerospace Conference,*. 2008: Big Sky Montana, MT.
11. Chunlin, L. and L. Layuan, *Dynamic resource allocation for joint grid user and provider optimisation in computational grid.* Int. J. Comput. Appl. Technol., 2006. **26**(4): p. 242-250.
12. Blossom, E., *Exploring GNU Radio.* http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html, November 2004.
13. Liu, J.W.S., *Real-Time Systems* 2000: Prentice Hall.
14.http://www.apcointl.org/frequency/project25/information.html.
15. http://pagesperso-orange.fr/sebastien.godard/.
16.http://www.vislab.uq.edu.au/users/manuals/python.2.4/lib/profile-instant.html.