An FPGA-based Cognitive Radio Framework

Jörg Lotze[†], Suhaib A. Fahmy[†], Juanjo Noguera[*], Linda Doyle[†] and Robert Esser[*]

[†]*Centre for Telecommunications Value-chain Research*
University of Dublin, Trinity College
{jlotze,suhaib.fahmy,linda.doyle}
@tcd.ie

[*]*Xilinx Research Laboratories*
Xilinx Ireland
{juanjo.noguera,robert.esser}
@xilinx.com

Abstract— **Cognitive radios are a promising technology for future wireless networks due to their ability to change behaviour according to their operating context. These radios require a platform which offers high performance while also being highly reconfigurable. This paper proposes a novel design methodology, along with a flexible software/hardware system and associated high-level tools for the implementation of cognitive radios on modern Field Programmable Gate Arrays (FPGAs). The platform presented can be used by wireless communication engineers with no hardware design experience. We present an initial case study demonstrating the use of the platform for a real application.**

Keywords— **cognitive radio, FPGA, reconfiguration, SDR**

I INTRODUCTION

According to Mitola [1], cognitive radios are self-aware, user-aware and radio frequency (RF)-aware radios which incorporate elements of machine learning and machine vision. In other words, a cognitive radio can change its behaviour according to its context autonomously, based on experience and learning. Due to technology advances in recent years, Mitola's vision is close to becoming a reality today.

Typically machine learning and reasoning, in a cognitive radio, is implemented in a *cognitive engine*, able to interact with the reconfigurable network node. It obtains operating and environmental parameters from the radio, performs learning, reasoning and decision making, and reconfigures the radio accordingly. This paper's focus is on a radio platform that can include such a cognitive engine.

A major driver for cognitive radios are dynamic spectrum access networks [2]. In such networks the cognitive nodes continuously sense the available spectrum and use free spectrum bands for communication. This has the potential to increase spectrum utilisation, thus making efficient use of a valuable and scarce ressource. The enabling technology for dynamic spectrum access networks is the cognitive radio.

Cognitive radios are computationally intensive wireless systems that implement highly demanding digital signal processing algorithms, on different platforms. General Purpose Processors (GPPs) are the most programmable and flexible platforms, though is reflected in their relatively poor performance. On the other hand, Application Specific Integrated Circuits (ASICs) provide high performance at the cost of reduced flexibility. FPGAs are an enticing alternative since they provide some measure of the flexibility afforded by GPPs along with performance and power efficiency closer to that of ASICs.

Modern FPGAs, in addition to providing programmable logic resources, integrate embedded processors and on-chip memory resources on a single die. Conventionally, FPGAs are completely configured at start-up and the functionality implemented in the FPGA does not change during application execution (i.e., static FPGA implementation).

Recent advances in the partial reconfiguration capability of Xilinx FPGAs [3] enable the *reconfiguration* of a region of the device while the rest of the FPGA continues operating. This approach, where the application functionality is time-multiplexed on the device, has several benefits; for instance, the ability to use smaller devices for a given set of functions by alternating the specific function implemented at any one time, rather than implementing all of them together. This directly translates into cost and power consumption savings.

In this paper we propose a design methodology, a reconfigurable platform, and associated high-level tools for the implementation of cognitive radios on modern FPGAs. Furthermore, we abstract the process of radio design away from its FPGA implementation details, opening the features of modern FPGAs to engineers without hardware design experience. With the proposed system, radio designers and communication engineers will be able to build cognitive radios, benefiting from the performance and power consumption advantages of FPGAs.

We give a brief overview of related work in Section II before introducing the project goals and high level system architecture in Section III. In Section IV, we discuss an initial case study demonstrating the platform's use in an audio transmitter. Section V draws conclusions and identifies the future research direction.

II Related Work

There are a number of radio plaforms available for software defined radio and cognitive radio research, with more still in development. In the following we give a brief overview of a selection of those directly related to the one proposed in this paper and of current interest to the authors.

a) The Kansas Universtiy Agile Radio

The cognitive radio platform developed at the University of Kansas is aimed for research in the areas of cognitive radio and dynamic spectrum access [4]. It is a custom-built hybrid system of a full embedded Pentium PC running the Linux Operating System (OS), a Xilinx Virtex-II Pro FPGA, an RF front-end for the 5 GHz band, and active transmit and receive antennas. Radio applications are composed of library components which can run on the PC, in FPGA logic, or on the embedded PowerPC processor cores of the Virtex-II Pro FPGA. An intelligent radio control and management structure instantiates the components and handles reconfiguration of the radio at run-time.

b) The Wireless Open-Access Research Platform

Rice university's Wireless Open-Access Research Platform (WARP) is a scalable, extensible and programmable wireless platform with a Xilinx Virtex-II Pro FPGA as its baseband processor and up to four RF daughter boards [5]. The physical layer of a radio is implemented in FPGA logic, while MAC layer functionality can be implemented in C using the embedded PowerPC processor cores. The C programs are compiled for the bare core, without an OS. This platform allows very efficient software radio implementations either written in VHDL or generated by the MATLAB language toolbox in Xilinx SystemGenerator. However, it does not allow C-library based development of radio or cognitive functionality for the PowerPC processor cores.

c) Rutgers WINLAB

The Wireless Information Network LABoratory (WINLAB) at Rutgers University has a network centric cognitive radio project aimed at the development of a multi-band, frequency agile radio platform [6]. In its current state it has two Xilinx FPGAs, one as a baseband signal processor and the other as a network packet processor. An embedded CPU is used for management and control of the radio. The first evaluation tests use the Universal Software Radio Peripheral (USRP) as a commercially available and inexpensive RF front-end [7].

d) GNU Radio

GNURadio is an open-source software radio implementation for PCs [8]. It provides a large library of signal processing components as well as a framework, based on the Python language, to connect these components together to build a radio. Although not required, it is mostly used with the USRP RF front-end [7], which is connected to the host PC via USB 2.0. A large number of signal processing components have been developed by the open-source community making it a powerful tool for PC-based cognitive radio experiments.

e) Why another platform?

The proposed platform is a hybrid FPGA hardware and software platform, benefiting from the high speed and low power consumption of an FPGA with its embedded processor cores. In that respect it is similar to the Kansas Universtiy Agile Radio (KUAR) or WARP platforms. However, we propose a system that includes a set of design tools to abstract the hardware details from the radio designer. Additionally we utilise run-time partial reconfiguration of FPGAs, so that a cognitive engine can reconfigure hardware and software in exactly the same way.

III A New Approach to Cognitive Radio Design

a) Project Goals

The primary project goal is to allow communications engineers (*i.e.*, non-hardware experts) to quickly design and implement cognitive radios on FPGAs, thus gaining from the significant performance and power advantages they provide. FPGAs are traditionally considered to be hardware devices, which are difficult to use since they require experience in low-level hardware design.

The radio designer on the other hand prefers to focus on the radio design and not think about implementation details on the FPGAs. We will develop a set of high-level tools that will enable the radio designer to choose radio components from a library and connect them together to create a functional description of the cognitive radio without knowing low-level implementation details.

A second objective is to show that FPGAs are *programmable* computing platforms and not rigid computing devices. Hence, an important goal of the project is to demonstrate that using FPGAs we can implement cognitive radios which are *programmable* at run-time. Examples of run-time reconfiguration of the radio[1] are:

- *Parametric reconfiguration:* changing the parameters that control the functionality of the Digital Signal Processing (DSP) components used in the radio chain.
- *Structural reconfiguration:* changing the structure of the radio chain. This could mean, for example, the addition or removal of DSP components at run-time.

We will study the mechanisms required to implement these two types of radio reconfiguration on an

[1] *reconfiguration* here does not refer to FPGA partial reconfiguration

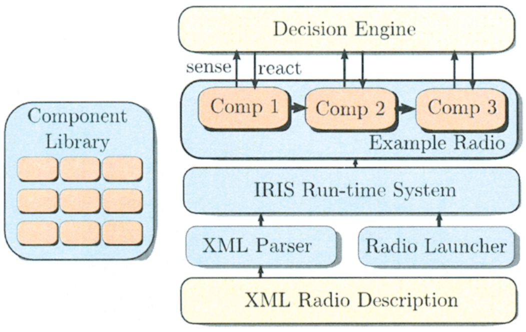


Fig. 1: The IRIS architecture.

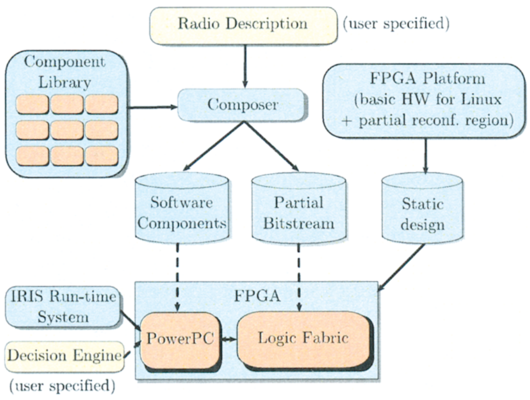


Fig. 2: The system architecture.

FPGA platform. Additionally, an important aspect of this second project goal is to investigate how the radio designer could describe and reason about this adaptation process.

As mentioned in Section I, a major driver for cognitive radio research is dynamic spectrum access networks [2]. In these networks the cognitive radio has the capability to sense available spectrum and use only free bands. Therefore, the final project goal is to investigate spectrum sensing algorithms and build a demonstrator for a dynamic spectrum access cognitive radio using the proposed FPGAs-based radio platform.

b) Implementing Radio In Software (IRIS)

The Implementing Radio In Software (IRIS) platform has been under development at Trinity College Dublin since 1999 [9]. It is a highly flexible and highly reconfigurable software radio platform for an x86 GPP running the Windows operating system.

The IRIS architecture is illustrated in Figure 1. The building blocks of an IRIS radio are DSP components, each performing a distinct task. Examples for such components are modulators, framers, or filters. Each of the components has a set of parameters and an interface to the decision engine, which allow for re-use in different radio configurations. The decision engine[2] is a software component designed for a specific radio configuration, *i.e.*, it is aware of the full radio chain while the processing components are not. This decision engine can subscribe to events triggered by radio components, and change radio parameters or reconfigure the radio's structure. A cognitive engine would therefore use this decision engine mechanism to control the radio.

The typical flow used to design a radio with IRIS is as follows. The radio designer writes an eXtensible Markup Language (XML) radio configuration specifying the radio components, their parameters and connections. If a decision engine is required, the radio designer implements it in C++ with a simple interface to the radio components and the IRIS run-time engine.

On IRIS start up, triggered by the radio launcher, the XML file is parsed and the IRIS run-time engine creates the radio by instantiating and connecting the specified components. The run-time engine then loads the decision engine, also given in the XML file, and attaches it to the components. Finally the radio is started and blocks of data generated by the source component will be processed by each of the components in the radio chain. When one of the events the decision engine has subscribed to is triggered, the decision engine can react. The reaction could be anything from diagnostic output to a full reconfiguration of the radio.

c) System Architecture

In order to execute IRIS on a Xilinx Virtex FPGA, we initially ported it to the Linux OS since IRIS requires an OS for its execution. Recently, there have been multiple research efforts to run the Linux OS on FPGAs. For the proposed framework, we use the embedded Linux OS distribution described in [10].

When executing a cognitive radio on an FPGA, the DSP components can be implemented in either the PowerPC processor or in the logic fabric. In order to support the execution of *hardware* components within IRIS, we extended the library of DSP components. Each DSP component in the library consist of two descriptions: a C++ description for a software implementation, and a VHDL description for a hardware implementation, both provided by the IRIS system. However, it is important to note that from the radio designer's perspective, both representations are functionally identical but with different non functional characteristics, *i.e.* performance, area, power, etc. The radio designer only knows about DSP components implementing the required functionalities.

To decouple the radio design as much as possible from the implementation details we propose the system architecture shown in Figure 2. All the radio designer has to provide is an XML description of the radio chain and optionally a C++ description of the decision engine. The composer takes that radio description and creates an efficient FPGA implementation consisting of some radio components implemented on the PowerPC processor and other radio components imple-

[2]In previous IRIS publications, the term *Control Logic* was used instead of *Decision Engine*.

mented in the FPGA logic fabric. Thus, the output of the composer is a partial bitstream that must be downloaded to the partially reconfigurable region defined in the FPGA (see Figure 2).

In addition, the composer will create a software wrapper for the hardware radio sub-system (*i.e.*, the partial bitstream). The main goal of this software wrapper is to implement the same interface as the rest of software components. That is, the newly created hardware sub-system is seen by the IRIS run-time system as another software component. A key feature of this software wrapper is to trigger the FPGA partial reconfiguration process (*i.e.*, actually downloading the partial bitstream), which will be triggered when the IRIS run-time system instantiates the component.

The static part of the FPGA design consists of all the infrastructure necessary to run the Linux OS. This static design is completely transparent to the radio designer. The static design used for our initial case study is described in Section IV. Given the composer output, the IRIS run-time engine instantiates the radio and the decision engine and runs the radio as described in Section III b).

Finally, it is important to note that the static design is the only building block in our proposed methodology which is FPGA and board specific. All other building blocks (*i.e.*, library of DSP components, *Composer* and IRIS run-time system) are independent of the target FPGA platform. In our framework, we plan to provide multiple static designs targeting different FPGA boards, thus making our framework portable across multiple FPGA platforms.

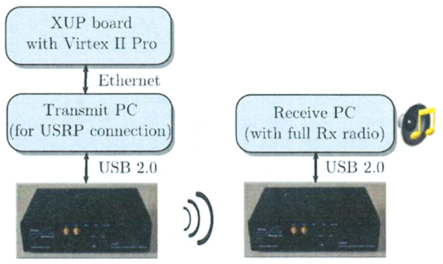


Fig. 3: The system setup in the initial case study. The only purpose of the PC at the transmitter side is to link the XUP board to the USRP due to the lack of USB 2.0 connections.

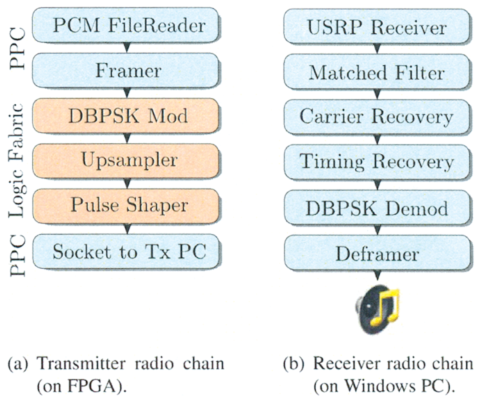


Fig. 4: Radio components for the initial case study. The PC which transmits the data using a USRP is not shown in the figure.

IV Initial Case Study

We have successfully demonstrated a live audio transmission application with the transmitter running on the Xilinx University Program (XUP) board [11] using the system architecture described above. In this section, the demonstrator setup is described.

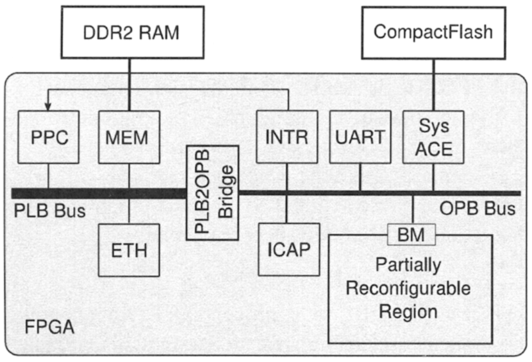
a) High Level Description

The demonstrator was implemented on a XUP board featuring a Xilinx Virtex-II Pro FPGA with an embedded PowerPC processor running the Linux OS. As an RF front-end we used the USRP [7], which can be connected to a PC using USB 2.0. The USRP is a flexible frequency agile radio front-end which can transmit and receive arbitrary waveforms at a reconfigurable frequency. It converts digital complex baseband data to RF for transmission, and vice versa for receiving. Unfortunately the XUP development board does not include a USB 2.0 master connection, so the baseband data was routed through a PC in order to access the USRP. The high level setup for the demonstrator is shown in Figure 3.

All transmitter signal processing is performed in the Virtex-II Pro FPGA while the transmit PC simply routes the processed data to the USRP. The receiver is implemented fully on a standard Windows PC using the Windows version of IRIS. Note that we only implemented physical layer functionality for this initial case study. This has the advantage that the quality of the physical channel becomes immediately apparent when listening to the received audio. The audio itself was encoded as simple 16 bit Pulse Code Modulation (PCM) samples with the sampling rate tailored to the physical layer data rate of 238 kbit/s.

b) Transmitter

The transmitter radio components are shown in Figure 4(a). The first component in the chain is the *PCM file reader*. It reads PCM encoded audio from a file in the Linux OS file system into memory and transfers this data to the *Framer*. The simple frame structure used for the initial case study starts with a frame check sequence, so that receiver can synchronise with each frame, followed by the payload length, the whitened payload and a checksum field. The payload is whitened with a pseudo random sequence known at the receiver, so that the data sent over the air appears random. This simplifies synchronisation at the receiver. The *DBPSK Modulator* performs Differential Binary Phase Shift Keying (DBPSK) modulation, encoding a binary '1' as 180° phase shift and a '0' as no phase shift. To limit the spectral footprint of the signal, it is upsampled and filtered with a root raised cosine pulse shaper in the *Upsampler* and *Pulse Shaper* components. The data



PPC	PowerPC	SysACE	Compact Flash controller
ETH	Ethernet controller	PLB	Processor Local Bus
MEM	Memory controller	OPB	On-chip Peripheral Bus
INTR	Interupt controller	BM	Bus Macros
UART	Serial port controller	ICAP	Internal Config. Access Port

Fig. 5: The static design of the FPGA.

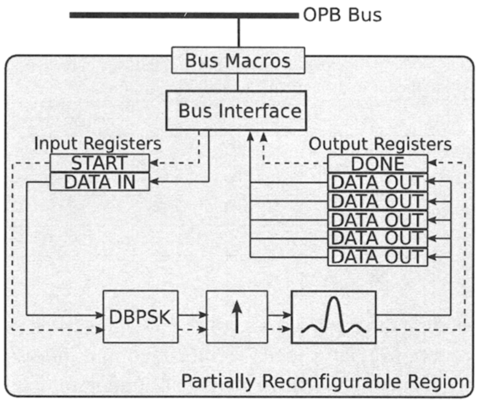


Fig. 6: Configuration of the parital reconfiguration region with input and output registers. The solid arrows represent the data flow; the dashed ones represent the control flow. Note: the number of registers is not exact.

is transferred to the transmit PC using TCP sockets, where it is then transmitted using the USRP.

In this initial case study we manually partitioned the radio chain into hardware and software components. The computationally demanding processing components are the modulator, upsampler and filter components. Therefore it is a natural choice to implement these in the logic fabric of the FPGA. Accessing the Linux OS filesystem from hardware is a complex task, but this can be performed easily in software. The same is true for sending the data over a TCP socket. Therefore we chose to implement these components in software. The framer component was implemented in software as well since its tasks are simple and there would not be much benefit if it was realised in hardware. Additionally, the data after the framer can be treated as a continuous stream of bits while the framer itself operates on larger blocks of data. Streams of bits can be easily pipelined in hardware giving the transmitter system a significant performance benefit.

c) Receiver

We only give a brief description of the receiver radio chain since it is implemented using the Windows version of IRIS which is not the focus of this paper. The receiver radio chain is shown in Figure 4(b).

After receiving signal samples from the *USRP Receiver* component, the *Matched Filter* reduces the noise in the signal. The *Carrier Recovery* and *Timing Recovery* components correct the frequency and phase offset of the receiver's local oscillator as well as the symbol timing errors. The *DBPSK Demodulator* converts the complex baseband samples into bits and the *Deframer* correlates with the frame check sequence to find the beginning of each frame. It extracts de-whitenes the payload. Finally the data is transferred to an audio player and the music is played.

d) FPGA Hardware

The static design of the Virtex-II Pro FPGA on the XUP development board is shown in Figure 5. It contains the basic hardware modules required to run the Linux OS on the FPGA and a partially reconfigurable region in which the hardware radio sub-system is implemented. The Linux OS filesystem is stored on a CompactFlash card attached to the System ACE interface controller in the FPGA. The Internal Configuration Access Port (ICAP) is used to access the FPGA configuration memory in order to load a specific radio design into the partially reconfigurable region. A Linux OS device driver for ICAP is available in the Xilinx Git repository [10].

The partially reconfigurable area has been configured for the initial case study as shown in Figure 6. As discussed in the previous section, the core components are the modulator, the upsampler and the pulse shaper. To allow the PowerPC processor to access the hardware components the bus interface functions as a slave on the OPB bus.

Valid samples are indicated using a 1-bit "valid" signal. The hardware performs DBPSK modulation, taking 1-bit symbols and producing 1-bit sample values, representing either $1 + 0\bullet$ or $-1 + 0\bullet$ in the complex baseband space. These samples are then upsampled by a factor of 4 and passed to the pulse shaper (*i.e.*, root raised cosine filter). Since the only possible sample values are ± 1, the filter is implemented without the use of multipliers. Instead, the value of each filter coefficient is either added or subtracted, depending on the input sample. This saves significant logic area, and allows the circuit to achieve the 100 MHz timing required by the OPB bus.

The utilisation of the FPGA's logic fabric shown in Table 1. It is a requirement of the static region that the PR region achieves timing of 100 MHz. Thus all hardware components must be designed to meet this constraint, as is the case for this implementation. It is clear that the partially reconfigurable region uses a small portion of the available logic. This suggests that significantly more complex hardware sub-systems can be used in future applications. This partial imple-

Table 1: FPGA utilisation in the static and partially reconfigurable (PR) regions. BlockRAMs are 18 Kb embedded memories, and slices are the basic measurement of logic utilisation.

Region	Slices	BlockRAMs
Static	5,927	14
PR	1,446	0
Available	13,696	136
Total	7,373 (53 %)	14 (10 %)

mentation did not make use of the other heterogeneous resources available such as embedded multipliers and memories, which will be of crucial importance in future designs.

e) Software Wrapper for the Radio Sub-System

A software radio component wrapper had to be written to transfer the data between software and the hardware component chain. This wrapper performs the basic steps shown in the following pseudo-code segment:

```
1 for each inputbyte do
2  FPGAWriteReg(DATA_IN, inputbyte)
3  FPGAWriteReg(START_REG, 1)
4  wait until FPGAReadReg(DONE_REG)=1
5  for each output_register
6   outputSamp[i]=FPGAReadReg(DATA_OUT[i])
7  done
8 done
```

where the functions `FPGAWriteReg` and `FPGAReadReg` write to and read from a memory-mapped register, respectively.

After the software has written a byte of data to the input register (*i.e.*, 8 symbols), it issues a "start" command by writing to the control register (*i.e.*, *START* register in Figure 6). The hardware awaits this signal and on receiving it begins to process data.

Once the hardware has finished processing the input data, this is signalled to the software by writing to the *DONE* register. The software waits for the hardware to finish its execution by polling the *DONE* register (see line 4 in the above pseudo-code). Finally, the software wrapper reads the array of sample values. (For one input byte the hardware generates output 32 samples.)

V Conclusions and Future Work

We introduced a novel cognitive radio system architecture, utilising the high performance, power efficiency and flexibility features of modern FPGAs. The proposed system can handle parts of the radio being implemented in hardware with other parts in software. We propose a composer capable of automatically deciding the hardware/software partitioning of radio components in a given chain, while hiding the implementation details from the radio designer. An initial case study successfully demonstrated the platform's use in a live audio transmission application.

One key point for future work is the implementation and test of the composer. Furthermore a more sophisticated demonstrator is underway, using the decision engine to actively reconfigure the radio running in the FPGA logic fabric. This decision engine will be a simple cognitive engine making autonomous decisions based on current signal characteristics.

We envisage that the developed platform will prove useful to the research and commercial communities in helping cognitive radios become a reality.

References

[1] J. Mitola III, "Cognitive radio: An integrated agent architecture for software defined radio," Ph.D. dissertation, Royal Institute of Technology, Stockholm, Sweden, May 2000.

[2] Q. Zhao and B. M. Sadler, "A survey of dynamic spectrum access," *IEEE Signal Process. Mag.*, vol. 24, no. 3, pp. 79–89, May 2007.

[3] P. Lysaght *et al.*, "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, Madrid, Spain, Aug. 2006.

[4] G. J. Minden *et al.*, "KUAR: A flexibile software-defined radio development platform," in *IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*, Dublin, Ireland, 17-22 Apr. 2007.

[5] K. Amiri *et al.*, "WARP, a unified wireless network testbed for education and research," in *IEEE International Conference on Microelectronic Systems Education (MSE)*, 3–4 Jun. 2007, pp. 53–54.

[6] Rutgers Winlab. (2007) Network centric cognitive radio platform. [Online]. Available: http://www.winlab.rutgers.edu/pub/docs/focus/Cognitive-Hw.html

[7] *Universal Software Radio Peripheral – The Foundation for Complete Software Radio Systems*, Ettus Research LLC, Mountain View, California, USA, Nov. 2006. [Online]. Available: http://www.ettus.com/downloads/usrp_v4.pdf

[8] Free Software Foundation, Inc. (2008) GNU radio - the GNU software radio. [Online]. Available: http://www.gnu.org/software/gnuradio/

[9] P. MacKenzie, "Software and reconfigurability for software radio systems," Ph.D. dissertation, University of Dublin, Trinity College, Ireland, 2004.

[10] Xilinx Git repository. Xilinx, Inc. [Online]. Available: git://git.xilinx.com

[11] *Xilinx University Program Virtex-II Pro Development System – Hardware Reference Manual*, Xilinx, Inc., Mar. 2005. [Online]. Available: http://www.xilinx.com/univ/XUPV2P/Documentation/XUPV2P_User_Guide.pdf