EXPERIENCES IN THE CO-DESIGN OF SOFTWARE AND HARDWARE ELEMENTS IN A SDR PLATFORM

Magdalena Sánchez Mora (Institute of Microelectronics and Wireless Systems (IMWS), National University of Ireland Maynooth (NUIM), Ireland; msanchez@eeng.nuim.ie); Jörg Lotze (Centre for Telecommunications Value-Chain Research (CTVR), Trinity College Dublin, Ireland; jlotze@tcd.ie); Gerry Corley (IMWS, NUIM, Ireland; gcorley@eeng.nuim.ie); Ronan Farrell (IMWS, NUIM, Ireland; rfarrell@eeng.nuim.ie)

ABSTRACT

The Center for Telecommunications Value-Chain Research (CTVR) has developed an integrated software radio platform, which consists of a RF transceiver, called the Maynooth Adaptive Radio System (MARS), and a software radio framework, called Implementing Radio in Software (IRiS). Developing a complete SDR system involves a wide range of disciplines. For example, the construction of our integrated software radio platform requires the interaction between the two subsystems: MARS and IRIS, which result in design challenges at hardware, firmware and software level. The experiences and challenges faced in achieving this integration will be described as well as a demonstration showing the working condition of the integrated software radio platform. The demonstration consists of an image reception using DQPSK modulation with IRIS and MARS at 2.41GHz. This demonstrates the capability to implement more complex experiments in future, such as audio or video transmission, and cognitive radios for dynamic spectrum access techniques. The MARS platform's software interface also opens new research opportunities for integration with other third party software radio frameworks.

1. INTRODUCTION

With the proliferation of reconfigurable radio software, it has become viable to develop low cost, high performance software defined radio (SDR) systems. The improving maturity of the technology has reduced the cost and increased the reliability of these systems, to a level where they are no longer concept demonstrators but can be considered for mainstream applications. Though the technology is improving, construction of a complete SDR system requires the integration of hardware, firmware and software components with an overarching system perspective on partitioning resources and tasks within the system. The level of complexity and range of disciplines required presents severe challenges to most research or design teams.

There are several frameworks available for software defined radio and cognitive radios research. The three we examined are IRIS, GNU Radio and OSSIE. IRIS is a mature and modular software radio framework that has been developed over the past number of years by CTVR [1] as part of their effort in cognitive radios. IRIS allows the dynamic construction of waveforms using a library of signal processing components. GNU Radio is a software application for building and deploying software defined radio systems under a GNU General Public License [2], originally developed by the Massachusetts Institute of Technology (MIT). The GNU Radio is most commonly used with the USRP RF front-end [3] with a customized API, although it can be integrated with other hardware platforms. The fact that it is open source, easy to use and has efficiently implemented numerous signal-processing modules has made it very popular amongst the SDR research communities. The Open Source SCA Implementation-Embedded (OSSIE) is another open source software defined radio developed by Virginia Tech [4]. As with IRIS and GNU Radio, OSSIE is a tool for rapid development of SDR components and waveforms applications but is more complex and limited regarding the amount of signal processing blocks available.

TABLE I
SOFTWARE RADIO PLATFORMS

	Strenghts	**Main Goal**
GNU Radio	Open-Source Software & Easy to use	Software Radio and efficient signal processing modules
IRIS	Rapid prototyping capabilities	Cognitive Radio
OSSIE	Great Flexibility & High Complexity	Training & Education

OSSIE is primarily intended for use in educational and training applications, IRIS focuses more on cognitive radio and GNU Radio's main goal is to provide a free and quite complete signal processing package that gives researchers the opportunity to explore radio systems (see Table I). A performance evaluation of OSSIE and GNU Radio is also presented at this conference [5]. Therefore, a wide range of research possibilities arises from the integration of MARS with these three radio frameworks. Elements of the reconfigurable radio platform were presented at SDR'07 Technical Conference [6][7].

This paper describes the experiences in achieving the integration of MARS with IRIS and shows a demonstration of the two systems working together: an image file transmission using DQPSK modulation. This demonstration proves that the integration was successfully achieved and allows more complex experiments to be carried out in future, such as audio or video transmission. It also allows developing cognitive radio systems, using a cognitive engine to sense the environment, decide which waveform to use, and reconfigure the running radio to accommodate these decisions. One example of such a system is the cyclostationary feature detector presented in [8].

The structure of this paper is as follows: Section 2 includes a detailed description of each of the components of the integrated software radio platform, Section 3 describes the radio configuration of the demonstrated case study, and Section 4 shows some test results including the constellation diagram and the recovered image. Section 5 presents the conclusions and ends by outlining the future research lines to be followed.

2. INTEGRATED SOFTWARE RADIO PLATFORM

Figure 1 shows the elements of the integrated software radio platform: MARS and a standard laptop computer. In order to enable the communication between IRIS and MARS, several software modules were developed such as a USB device driver, an API library, the firmware running on the USB microcontroller and two additional IRIS DSP components.

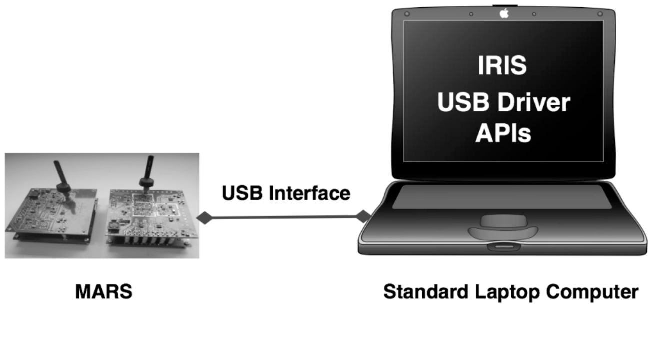


Figure 1. Integrated Software Radio Platform

The following subsections describe each of these components and the challenges encountered in the integration of the two systems focusing on the software aspects.

2.1. Maynooth Adaptive Radio System (MARS)

MARS was designed to be flexible, easy to use and inexpensive. To meet these requirements it was decided to use direct conversion receiver and transmitter architectures as this approach allows large baseband bandwidth, avoids the use of fixed IF filters and has a lower component count than comparable superheterodyne or low IF configurations. Data and control communication is performed via a USB interface between the transceiver and a laptop PC.

A block diagram of MARS is shown in Figure 2. The transceiver connects to a PC via a USB 2.0 interface that has a maximum data transfer rate of 480Mbps. In the baseband section there are two 16 bit DAC's with a sample frequency of 200Msps and two 16 bit ADC's with a maximum sampling frequency of 100Msps. These provide the I and Q signals to the RF transmitter and receiver boards. Although the ADC's and DAC's are capable of very high conversion rates, currently the maximum operating speed of the system is limited by the USB interface and the radio software running on the PC.

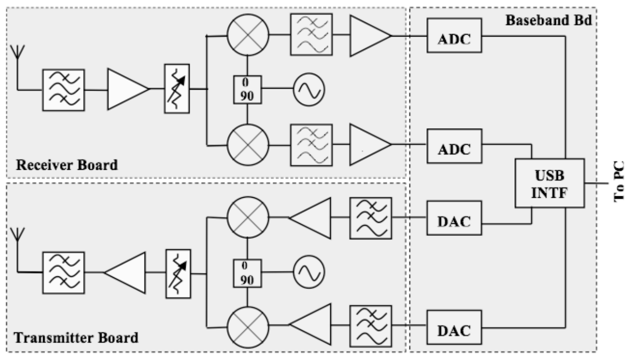


Figure 2. Block diagram of MARS

The RF transmitter operates between 2.1 and 2.45GHz, with a maximum output power of 24dBm and 256 levels of power control all under full software control. The RF receiver operates over the same frequency range, has a noise figure of 5dB, bandwidth of 40MHz and 48dB of gain control. A wider frequency range is possible with adjustment of the power amplifier matching network.

Figure 3 shows a picture of MARS, which consists of several hardware elements including a radio transmitter, a radio receiver and a baseband interface. Both the RF transmitter and receiver boards plug into the baseband boards in ´piggyback´ fashion through a series of SMB connectors, this makes for a low loss robust connection.

In our design, a key criterion was to explore the concept of pushing the software engine as close to the antenna as possible, therefore the responsibility for processing the raw IQ data was deemed to be a software engine task.

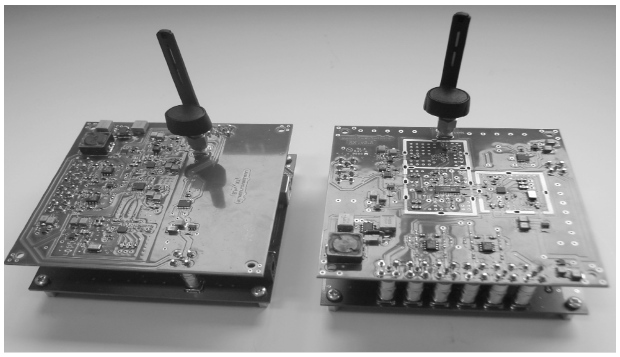


Figure 3. MARS boars

2.2. Firmware, USB driver and API library

As we are dealing with a new SDR hardware platform that uses a USB microcontroller, a customized USB driver and firmware were developed. An efficient API library was also implemented in order to provide an interface with third party waveform applications. Figure 4 shows a high-level vision of the interconnection between the elements of the integrated radio platform.

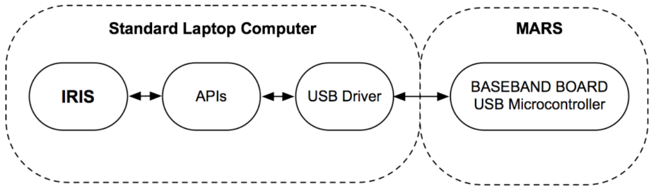


Figure 4. IRIS to MARS interface

As can be seen in Figure 4, the main requirements in terms of software are embedded code running on the USB microcontroller, an optimized USB driver and an API library providing an interface with third-party waveform development tools, such as IRIS. The three modules were written in the C language due to its low-level capabilities, speed and portability between a wide variety of PCs and operating systems.

The principal challenges were first to provide high-speed and continuous data transfer without data loss, and second to enable the re-configurability of the hardware devices. The debugging stage also required an additional effort due to the different location of the software modules: firmware at hardware space, device driver in kernel space, and API library in user space.

High-speed data transfer without data loss was achieved by using optimized techniques in both the USB driver and the embedded code. The USB driver utilizes USB request blocks (URBs) [9][10] as the data structure for transmitting/receiving information due to their ability to be queued. This queue of URBs guarantees that there will be always information waiting to be processed in the communication channel, which causes maximum usage of bandwidth and a continuous stream of information. The data loss problem was solved by using bulk transfer type guaranteed delivery of data [11]. Regarding the embedded code running on the USB microcontroller, the General Programmable Interface (GPIF) [12][13] was programmed directly instead of through the 8051 CPU as this programmable state machine provides the highest bandwidth achievable over the physical layer. The GPIF is powerful and versatile but added great complexity to our firmware development stage.

Regarding the hardware re-configuration capabilities, the API library includes C functions for configuring the sampling rate, the local oscillator frequency, the receive chain gain control, and other parameters. Changing any of these parameters is as easy as calling the appropriate function in software specifying the new parameter.

The API library, the USB driver and the embedded code provide the required interface between IRIS and MARS. It has been proven to work efficiently since the received image in the QPSK experiment was perfectly recovered with no data loss or visible degradation. The experiment configuration and the obtained results are presented in Sections 3 and 4.

2.3. Implementing Radio In Software (IRIS)

The software radio framework utilized in our system is the IRIS software radio framework. IRIS has been under development at Trinity College Dublin since 1999 [1]. It is a highly flexible and highly reconfigurable software radio platform for a GPP running either Windows or Linux.

The IRIS architecture is illustrated in Figure 5. 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 control logic, which allow for re-use in different radio configurations. The control logic 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 control logic 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 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. Optionally the radio designer can implement a control logic for radio reconfiguration. On IRIS start up 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 control logic 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. The control logic can react to events triggered by components, with anything from diagnostic output to a full reconfiguration of the radio.

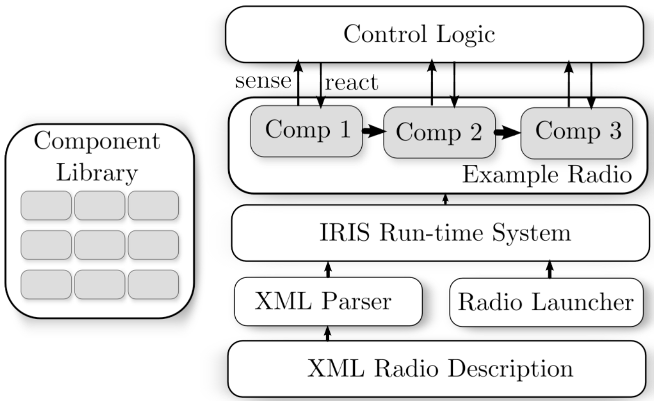


Figure 5. IRIS architecture

In order to integrate our RF transceiver with IRIS, two new IRIS DSP components needed to be developed to provide an interface through an optimized low-level API to interact with MARS.

These new components (one for transmit operation and one for receive operation) interact with the API library to set up the RF transceiver hardware. During radio operation, the IRIS components transmit or receive data to or from the hardware.

To avoid data loss in the receiver IRIS component, it was required to read continuously from MARS. We accomplished this by implementing the data fetch in a thread running independently of the DSP component. This thread continuously reads data from the hardware and inserts it into a buffer shared with the IRIS component. When this component is executed, it reads the data from this buffer and outputs it to the next component in the radio chain. Care had to be taken to avoid performance problems due to unnecessary movements of data in memory. No data can get lost in the transmitter component, thus its implementation proved to be simpler.

These new components enable the transmission and reception of complex waveforms utilizing other DSP components available in IRIS.

3. INITIAL CASE STUDY

To test MARS together with IRIS we successfully transmitted an image. To isolate transmitter from receiver problems we decided to use the USRP front-end as transmitter, which is known to work, and focus on the receiver, which is more challenging to implement. The high level experimental setup is depicted in Figure 6.

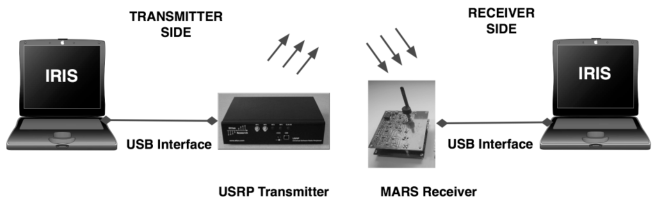


Figure 6. Experiment Setup

The transmitter radio components are depicted in Figure 7(a). The first component in the chain is the *Image Reader*, which reads a bitmap image and provides the data to the *Framer* component. The simple frame structure used for the initial case study starts with a frame check sequence, so that receiver can synchronize 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 synchronization at the receiver. The *DQPSK Modulator* modulates every pair of bits to one of four phase shifts in the I-Q-space using differential encoding. 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, respectively. This I and Q data is then transferred to the USRP by the *USRP Transmitter* software component, upconverted, and transmitted.

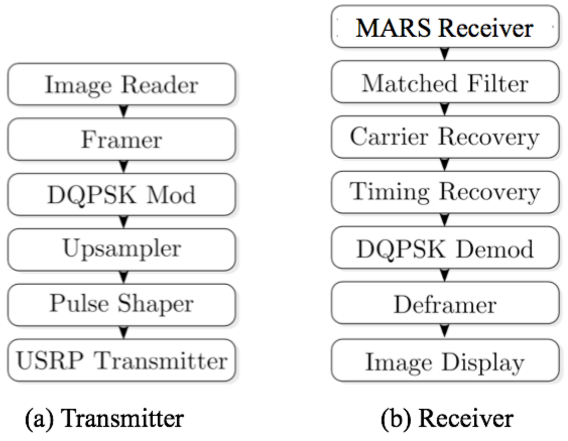
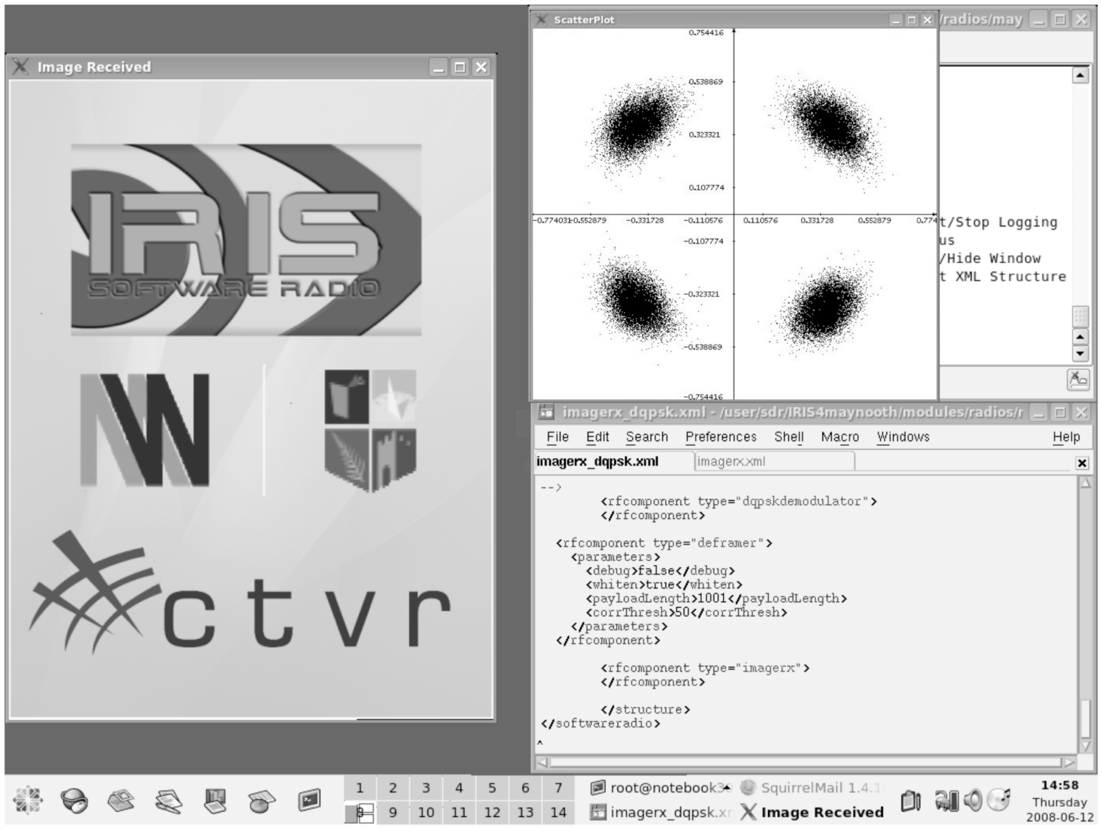


Figure 7. IRIS transmitter and receiver chains

The first IRIS component in the receiver chain, shown in Figure 7(b), is the *MARS Receiver* software component as explained in Section 2.3. It receives I and Q data from the



```
-->
        <rfcomponent type="dqpskdemodulator">
        </rfcomponent>

  <rfcomponent type="deframer">
    <parameters>
      <debug>false</debug>
      <whiten>true</whiten>
      <payloadLength>1001</payloadLength>
      <corrThresh>50</corrThresh>
    </parameters>
  </rfcomponent>

        <rfcomponent type="imagerx">
        </rfcomponent>

        </structure>
</softwareradio>
^
```

Figure 8. DQPSK receiver laptop screenshot

RF hardware and makes it available to IRIS. The *Matched Filter* is another root raised cosine filter matched to the one used at the transmitter. It increases the SNR of the signal and removes inter-symbol interference introduced by the pulse shaper at the transmitter. It is likely that the local oscillator of the transmitter and receiver are not coherent. The *Carrier Recovery* corrects the frequency and phase offsets using a Costas loop [14]. After that the *Clock Recovery* component interpolates the data to the optimum sampling points for each symbol using a modified Mueller and Mueller algorithm [15]. The *DQPSK Demodulator* converts these symbols back to bits by assessing the phase shifts between samples. In the *Deframer* that data is correlated with the known frame check sequence to find the beginning of a frame, the payload is extracted, converted to bytes, and dewhitened. The *Image Receiver* component can then display this information in an image.

In the experiment we used a 1Msps transfer rate, with each symbol represented by four IQ samples. We are able to increase this rate, limited by the speed achievable on the USB 2.0 interface and the CPU performance of the laptop PC used. The results are presented in the following section.

4. RESULTS

The image selected for the experiment is a logo montage of the different institutions supporting this research as well as the IRIS logo. Figure 8 shows a screenshot of the receiver laptop with the recovered image on the left and a live constellation diagram (before the demodulator) on the top. The screenshot also shows IRIS running on a Linux terminal and the xml file containing the receiver chain configuration (bottom right).

The quality of the recovered image was consistently high with no visible errors. The DQPSK constellation diagram clearly shows the DQPSK symbol positions. Even though there is some scattering of the symbols, it is not sufficient to cause loss of data.

5. CONCLUSIONS AND FUTURE WORK

Achieving a sustained high performance between these different systems has proven to be a difficult challenge and in this paper we have showed the key issues encountered, and our corresponding solutions. High-speed and continuous data transfer was accomplished by using URBs in the USB device driver and by programming the GPIF in the USB microcontroller, both URBs and the GPIF provide the highest bandwidth achievable over the USB physical layer. Data loss was avoided by using bulk transfer type in the USB driver, which guarantees data delivery, and by implementing a multithread mechanism plus a circular buffer on IRIS.

The integrated software radio platform described in this paper means a big step forward in our SDR research activities and opens new opportunities for integration with other frameworks for software defined radio such us GNU radio. Also more sophisticated multimedia applications can be carried out, such as audio or video streaming, with a small additional effort since high-speed data communication was achieved as shown in this paper. And lastly the cognitive radio capabilities of the IRIS framework will be explored with MARS and compared against the USRP hardware platform.

Further over-the-air RF performance measurements will be taken for a better indication of the MARS receiver quality, such as bit error rate, interference and blocking. Also the MARS transmitter will replace the USRP transmitter so the whole system functionality will be verified in subsequent experiments. Finally our future work includes using extending this platform to support large arrays of individually controllable antennas and extending the frequency range and capabilities to explore next generation TETRA applications [16].

7. ACKNOWLEDGEMENTS

Research presented in this paper was funded by a Centre for telecommunications Value-Chain Research (SFI 03/CE03/I405) by Science Foundation Ireland under the National Development Plan. The authors gratefully acknowledge this support.

8. REFERENCES

[1] P. MacKenzie, “Software and reconfigurability for software radio systems”, Ph.D. dissertation, Trinity College Dublin, Ireland, 2004.

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

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

[4] Open Source SCA Implementation – Embedded (OSSIE), http://ossie.wireless.vt.edu

[5] A. Palomo, R. Villing, R. Farrell, “Software Defined Radio Architectures Evaluation”, SDR Technical Forum, Washington DC, October 2008.

[6] M. Sánchez Mora, G. Baldwin, R. Farrell, “Software Engine Development for SDR”, SDR Technical Forum 2007, 5-9 November 2007, Denver, Colorado.

[7] G. Baldwin, L. Ruíz, R. Farrell, “Low-Cost Experimental Software Defined Radio System”, SDR Technical Forum 2007, 5-9 November 2007, Denver, Colorado.

[8] P.D. Sutton, K.E. Nolan, L.E. Doyle, "Cyclostationary Signatures for Rendezvous in OFDM-Based Dynamic Spectrum Access Networks", 2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, DySPAN, vol., no., pp.220-231, 17-20 April 2007.

[9] J. Corbet, A. Rubini, G. Kroah-Hartman, *Linux Device Drivers*, Third Edition, O'Reilly, 2005.

[10] G. Kroah-Hartman, *Linux kernel in a nutshell*, O'Reilly, 2006.

[11] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC and Philips, *Universal Serial Bus Specification*, Revision 2.0, April 27, 2000.

[12] Cypress Semiconductor Corporation, *EZ-USB Technical Reference Manual*, version 1.4, 2000-2006.

[13] Cypress Semiconductor Corporation, *EZ-USB® FX2™ GPIF Primer*, 2003.

[14] M. Xiao, T. Cheng, "Improved Implementation of Costas Loop for DQPSK Receivers Using FPGA", International Conference on Communication Technology (ICCT), vol., no., pp.1-4, Nov. 2006.

[15] G.R. Danesfahani. T.G. Jeans, "Optimisation of modified Mueller and Muller algorithm", Electronics Letters, vol.31, no.13, pp.1032-1033, 22 Jun 1995.

[16] L. Gao, R. Farrell, “Using SDR to embed WiMAX channels within the TETRA framework”, SDR Technical Forum, Washington DC, October 2008.