Using Multiple Channels to Improve SDR Flexibility and Performance

Roberto de Matos*†, Antônio Augusto Fröhlich†, and Leandro Buss Becker*

* Department of Automation and Control Systems (DAS), Federal University of Santa Catarina (UFSC)
P.O.Box 476, 88040-900 - Florianópolis - SC - Brazil
† Laboratory for Software and Hardware Integration (LISHA), UFSC, Florianópolis - SC - Brazil

Emails: rmatos@das.ufsc.br, guto@lisha.ufsc.br, lbecker@das.ufsc.br

***Abstract*—Software Defined Radios (SDR) emerged as an alternative approach to provide adaptable wireless communication capacity. Instead of providing specific hardware interfaces for any communication standard, it provides a generic physical layer that together with a software part is able to reproduce several communication standards. A typical drawback of SDRs is the high coupling between the software and the hardware layers, which makes hard to migrate the software part to different hardware without modifications. To overcome this problem, this paper presents a flexible architecture for SDRs that is based on the well-know channel concept. By using multiple channels the proposed architecture is able to abstract the interface of any SDR hardware, so that it becomes easy to make hardware modifications. The paper also presents a hardware implementation for our proposed multiple channel system, which shows a considerable performance improvement compared to a typical solution. To prove our concept we implemented the proposed architecture using the GNU Radio framework and performed experiments running the same application on different hardware (USRP, USRP2, and a modified USRP2). The results from such experiments are presented and discussed along the paper.**

I. Introduction

Software Defined Radio (SDR) [1] is a technology designed to allow flexible wireless communication systems. Its goal is to get the software as close as possible to the antenna and then use it to filter, modulate, demodulate, and perform other stages of transmission and reception paths. Its main advantage is the flexibility of the physical layer, which allows communication with other radios in different frequencies or modulation only by modifying the software. The ideal SDR would eliminate almost all hardware, maintaining only an ADC to take samples from the antenna to the software. However, there aren't ADCs fast enough to sample all the required bandwidth. Thus, more hardware is needed to down-convert the chosen band, so that the ADCs can sample it.

The software technology employed in SDRs offer a much quicker development cycle and in-field functions upgrades. Software components are easily modifiable and can be re-used in different hardware platforms [2]. The price to pay is an increased energy consumption compared to fixed function ASICs. Thus, it is only justifiable using SDR if the physical layer flexibility is really important.

Now a days there are several SDR architectural proposals (see [2], [3], [4], [5], [6], [7]). They normally adopt processing distribution by grouping asymmetric processing units like FPGAs, DSPs, and GPPs. This scenario leads to a vast number of SDR architectures, each one with its own complex programming libraries (APIs). As the organization of the components and interfaces is completely different on each architecture, the software components are normally tightly coupled to the platform, making re-use more difficult.

To cope with this problem this paper presents a new flexible architecture for SDRs, which main characteristic is the uncoupling of channels from the different physical layers. Our proposal consists in abstracting the interface of any SDR hardware to as many channels as possible. By using this concept we simplify the creation of different physical layers, as the only action needed is to configure the channel characteristics in software. Given such information, all specific configurations from each hardware can be extracted. This allows the creation of a self-contained components repository (loosely coupled from hardware), which can be configured to run on different hardware platforms without requiring any modification of the software. Another contribution of this paper concerns an optimized hardware implementation for our proposed multi-channel system, which presents a considerable performance improvement in comparison to standard solutions without losing any flexibility.

To demonstrate our concept we implemented a practical experiment, using the proposed architecture to create an IEEE 802.15.4 receiver. This receiver was implemented with the GNU Radio framework [2]. Experiments consisted of running the receiver in three different hardware architectures: USRP, USRP2, and our modified USRP2. Obtained results show a relevant performance improvement, as presented and discussed along the paper.

The remaining parts of this paper are organized as follows. The next section presents an overview of the related technologies (SDR, GNU Radio, and USRP). Section III presents the details of the proposed channel architecture. Section IV describes our prototype implementation and the conducted performance evaluation. Finally, the conclusions and future work directions are presented in Section V.

978-1-4673-0009-4/12/$26.00 ©2012 IEEE

II. Software Defined Radio Architectures

Currently, there are many architectures and frameworks for SDR development. One of the most used is the GNU Radio [2], which was chosen to develop this work. The main reason is because it is public-available and because it has a very active community. GNU Radio works with many hardware platforms, including the USRP (Universal Software Radio Peripheral) and its extensions. The USRP is a low-cost and flexible platform for software defined radios that allows a rapid development and experimentation cycle.

GNU Radio started with the funding of John Gilmore and it has been developing since 1998, with a complete reformulation in 2004. The simple use and high performance for Digital Signal Processing (DSP) are achieved by the hybrid nature of the framework, which uses C++ to write the core of the signal processing components (Processing Blocks) and Python to connect these components in a data flow graph to make filters, modulators, demodulators and other structures that compose the radios.

The Universal Software Radio Peripheral (USRP) is a modifiable, low-cost, and flexible platform developed by Matt Ettus [3] for GNU Radio project. It is composed of ADCs (*Analog-to-Digital Converters*), DACs (*Digital-to-Analog Converters*), an FPGA (*Field Programmable Gate Array*), slots for daughterboards (RF front ends), and a communication interface, which connect the GNU Radio framework with the RF world. The daughterboards have the function of down-converting from received carrier frequency to intermediate frequency and the inverse for the transmission chain. Currently there are a couple versions of the USRP board (USRP, USRP2, N210, E100, and the newly announced B100). In this work we used the second version (USRP2).

Figure 1 shows a typical architecture for the physical layer (PHY) implementation using GNU Radio framework and the USRP family. We observed that it is difficult to change or add more physical layers in such architecture. This is due to the fact that the configuration of each PHY becomes dependent on the others and also that the hardware resources (e.g. spectrum utilization) must be verified and managed by the user. In case of addition/modification of a PHY, the whole system should be reviewed to ensure the absence of conflicts. This is a significant drawback for applications that need dynamic change or upgrade of PHYs like Cognitive Radios or Multiple Networks Systems.

III. Proposed Channel Architecture

Most existing hardware that implements the functions related to the early stages of SDR processing rely on very complex APIs, which require programmers to have a deep knowledge of the hardware. As a consequence, the implementations are tightly coupled to the hardware and the final system is very sensitive to platform changes. For example, consider the need of migrating the PHY to a new hardware, like for instance exchanging USRP for USRP2. Although both hardware play the same role in the system, the organization of the components and interfaces are completely different on each other. The respective APIs do not make this transparent to the programmer.

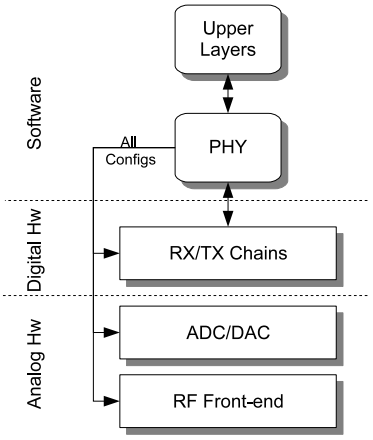

Fig. 1. Traditional Implementations of Physical Layer.

The architecture proposed in this work overcomes these problems by creating an interface responsible for channel allocation regardless of the hardware that implements the functions. Thereby channels can be reconfigured at any time in the communication process, while the hardware-specific settings and resources management is the responsibility of the abstraction layer. Additionally, this strategy allows channel separation to be implemented in any layer of the system (e.g. DSP, FPGA). The architecture also makes transparent to the PHY the amount of boards that are processing the channels. In other words, the capture process and the initial spectrum processing do not matter for the final result, only what matters is the set of channels.

An overview of the proposed architecture is shown in Figure 2. The main blocks are the *Channel Manager* and the *Abstraction Layer*. Using the *Channel Manager* the PHYs can allocate any number of channels by passing the central frequency and bandwidth of each channel, and can also read information about the channels allocated. The *Channel Manager* uses *dynamic factors* and *static factors* to determine whether a channel can be allocated or not. *Dynamic factors* are related with the competition by the resources, for instance, different channels cannot overlap each other. On the other hand, *static factors* are the resource limits of each hardware, which can be easily loaded by *Channel Manager* through a configuration file. A domain analysis of SDR hardware leads to three classes of resources regarding channel allocations, as follows:

RF Front-End Capacity: The function of the RF Front-End is to convert the window of interest to an Intermediate Frequency (IF), where the signal is sampled by ADCs for reception and inverted for transmission. Each RF Front-End possesses a range of possible frequencies with maximum and minimum limits.

Sampling Window: After converting to the IF, a window of the spectrum is sampled by the ADC, considering the limitations imposed the Nyquist-Shannon theorem,

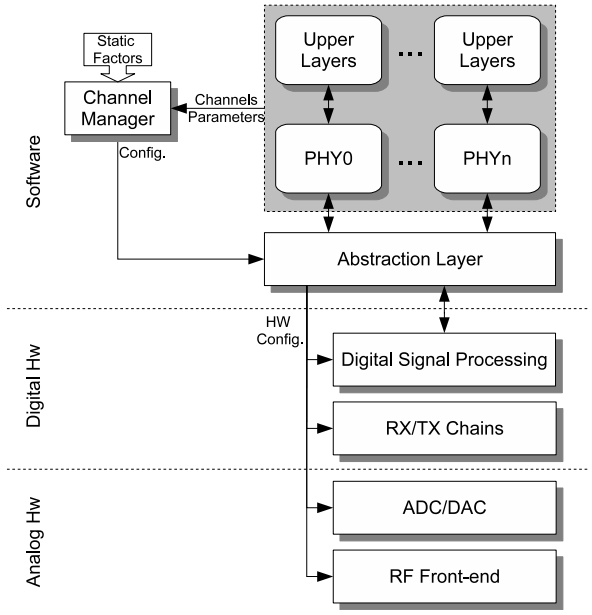

Fig. 2. Proposed Multichannel Architecture.

i.e., the sampling frequency must be twice the frequency of interest. In other words, for a window of 100MHz, the ADC should sample the spectrum at 200MHz or at 100MHz, but in complex-sampling mode.

Communication Interface: The hardware transmits the samples to a processing unit where software can be executed to define the behavior of the physical layer. If the ADC is connected to the processor via high speed interfaces there is no problem. However, it is common to use slower interfaces, creating a bottleneck for the sampling window. For instance, in USRP and USRP2 architectures, USB and Gigabit Ethernet interfaces are respectively used, limiting the size of the sampling window processed by the host up to 8MHz using USRP and up to 25MHz using USRP2.

The *Abstraction Layer* block was designed to keep the interface simple and immutable while connecting the PHYs. It is composed by the hardware adapters that convert the configurations received from the *Channel Allocator* to the actual hardware in use, like USRP, USPR2, or any other SDR hardware. If the number of channels implemented in hardware is not enough for the number of channels requested by the PHYs and it is possible to achieve the requested configuration adding software blocks (e.g. Frequency Translator, Decimators), those blocks will be activated in the *Abstraction Layer* by the *Channel Manager*.

IV. Implementation and Evaluation

The proposed architecture was implemented as *sink* and *source* blocks of the GNU Radio framework. It works with the original USRP and USRP2, and also with a modified USRP2, whose hardware was modified to increase the number of channels processed in hardware (four in this case). New

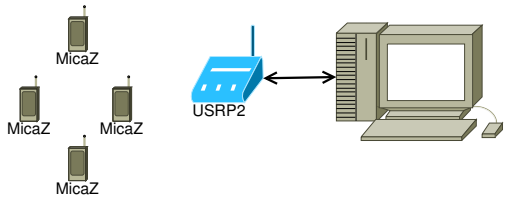

Fig. 3. Experimentation Environment.

SDR boards could also be added by simply implementing the hardware support in the *Abstraction Layer* and updating the *static factors* in the *Channel Manager*.

To test our prototype implementation, the IEEE 802.15.4 multi-channel demodulator from UCLA [8] was adopted. More than simply testing the prototype, we aimed at verifying the impact of migrating the channel separation from software to hardware. Therefore, two different test-scenarios were created. In the first scenario a USRP2 was used to run the four channel separations in software. The second scenario used the modified USRP2 to execute the same tasks (remembering that here all channel separations were implemented in hardware). Both experiments were executed on an Intel QuadCore processor (Q9559) PC running at 2.83GHz, with 4GB of RAM, and running Ubuntu 9.10 (2.6.31-19 kernel). The interface with USRP2 used a Broadcom BCM5755 GigE chipset.

Our experimentation environment was composed of four MicaZ[1] motes, one USRP2, and one host, as depicted in Figure 3. The MicaZ motes used a CC2420 radio with a standard implementation of 802.15.4. Each mote was in charge of transmitting messages periodically using a different channel. The channel occupation did not influence the experiments. Each experiment was executed for 300 seconds, logging performance information every 2 seconds to generate three different curves: *IDLE*, *USR*, and *SYS*. The *USR* curve indicates the CPU utilization required by the developed test applications, which are executed in the user space. The *SYS* curve represents the CPU utilization of applications/services in system space. Finally the *IDLE* curve shows the percentage of free/unused CPU processing capacity.

Instead of using the architecture presented in Figure 2, whose channel separators were in hardware, the first test used the structure shown in Figure 4. In this structure the *Channel Manager* configured the USRP2 to send the maximum sample window (25MHz) and selected the channels using GNU Radio software blocks. The results from such experiments are presented in Figure 5. It shows a high CPU overhead, which is caused by the predominance of the blocks in software. This is expected given the parallel nature of the functions, such as frequency translation and decimator at high levels of samples per second.

In the second test the reconfigurable hardware of the USRP2 was modified to include the four channel selectors, as depicted in Figure 6. It should be highlighted that there was no modification in the multi-channel demodulator, once the

[1]Sensor node developed by Crossbow

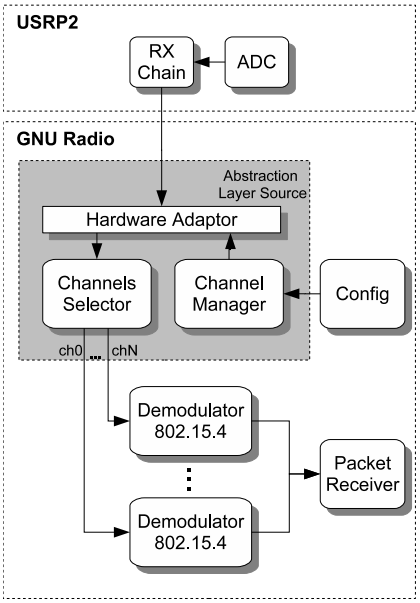

Fig. 4. First experiment: channels separation implemented in software.

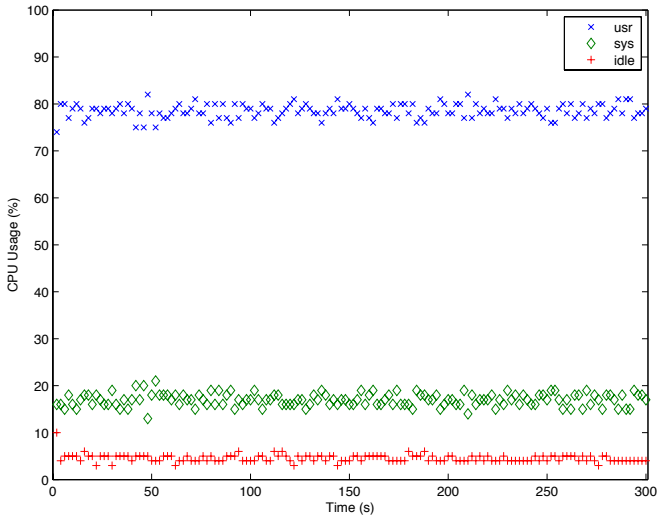

Fig. 5. CPU usage in the first experiment.

Channel Manager and the *Abstraction Layer* blocks allowed this modification in a transparent way. Figure 7 presents the obtained results, which shows a significant performance increase (observe the IDLE curve). The migration of the channel separation to the reconfigurable hardware allowed the frequency spectrum window captured by the RX chain to be "sliced" into channels with central frequencies and distinct widths, so that only the relevant data was sent to the host.

A comparison of the average CPU usage in both tests is shown in Figure 8. The decrease of 68.6% of CPU usage in user space (*USR*) for the second test shows a significant performance improvement to execute the same experimentation scenario. Moreover, the decrease of data flow between the modified USRP2 and the host caused other benefits, for example the reduction on the number of interruptions – this

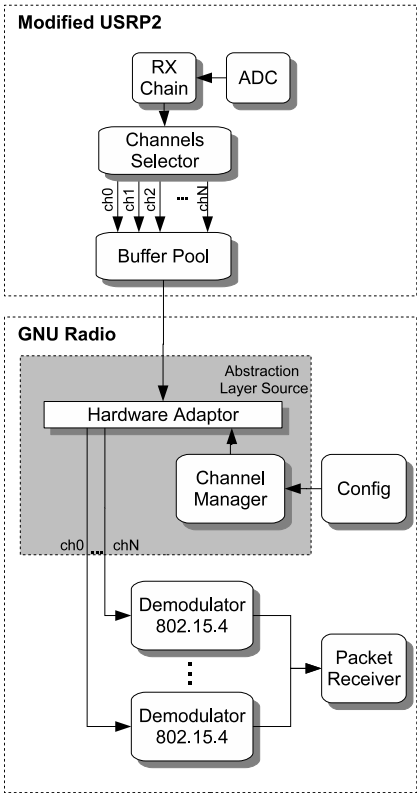

Fig. 6. Second experiment with hardware channels separation.

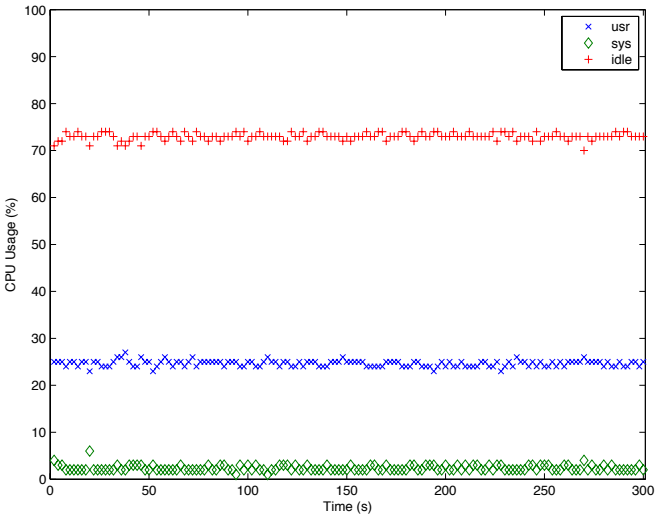

Fig. 7. CPU usage of hardware channels separation scenario.

can be observed by the reduction of the system load (*SYS*), used to handle the interruptions. The overall reduction in CPU usage (considering both *SYS* and *USR* load) was 86.1%. This comparison shows that it is very appropriate to make the migration of all channel separation stages to hardware in simultaneous multichannel systems. The system keeps the same flexibility achieved by the software blocks and spare CPU usage. Again, this is due to the blocks that can add real

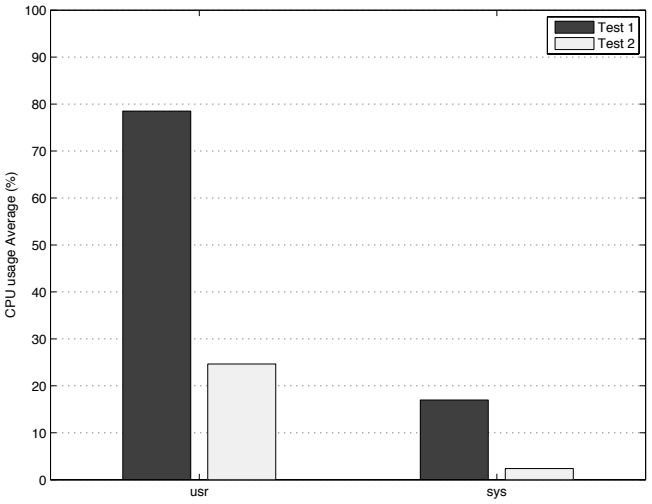

Fig. 8. Average CPU usage.

flexibility for SDRs, such as the physical layers ones.

Beyond CPU performance, another advantage of channel management in hardware is a better use of the bandwidth from the USRP2 communication interface. For example, the standard 802.15.4 predicts the use of 16 channels in a frequency window from 2400MHz to 2483.5MHz. Each channel requires a bandwidth of 2 MHz and a channel separation of 3 MHz. Therefore, in the first test set we used the largest possible raw sampling window (there was no previous channel processing). In other words, channel separation samples (void data) were sent to the host as well. So, indeed, each channel occupies 5 MHz of the communication interface bandwidth, which is limited at 25 MHz in USRP2. Consequently, the maximum possible number of channels is five, considering consecutive channels. In fact, instead of using 25MHz windows, only 10MHz (40%) of useful information could be used to achieve similar results. On the other hand, the complete migration of the channel separation to hardware allowed configuring each channel independently, sending only 2MHz of useful data per channel to the host. That allows up to 12 non-consecutive channels and increases the communication interface bandwidth usage to 96%.

To double check the hardware adapter implementations in the *Abstraction Layer* and the static factors in the *Channel Manager*, an additional test was executed using USRP. However, the number of PHYs was decreased to two, because the original USRP has only two channel selectors in hardware and the USB bandwidth is only 8 MHz, which allows only a raw sampling window with two channels (2 x 2MHz + 3MHz of channel separation = 7MHz). Like in the other tests, all the specific hardware configurations were extracted based on requested channels. Actually, both channel decoders were able to run in any of the three supported boards without any modification.

V. Conclusions and Future Work

This paper presented a flexible architecture for SDRs that is based on the well-know channel concept. It is intended to overcome the high coupling between the software and the hardware layers observed in other SDR implementations. The proposal is sustained by the fact that the interface of any SDR hardware can be abstracted to a multiple channel interface, where each channel has an independent central frequency and bandwidth. This is then further used to determine all hardware specific configurations.

The proposed architecture was implemented using the GNU Radio framework and was tested with the USRP, USRP2, and our modified USRP2. The proposal is, however, easily extendable to other SDR hardware. Details about our modified version were also addressed. It consisted mainly of a hardware implementation for the channel separation stage, which presented a considerable performance improvement compared to the software solution. It is important to mention that this implementation was perfectly supported in the proposed architecture, as it did not affect the proposed structure while keeping the flexibility.

As future work, we intend to improve the optimization algorithm of *Channel Manager* for a better deployment of the requested channels using the *static factors* information. Moreover, we intend to implement more *hardware adaptors* to extend the architecture to other SDR hardware. Finally, we should address a better integration between the physical layer projects and the MAC.

VI. Acknowledgments

Thanks are given to the Brazilian funding agencies FINEP, CNPq, and CAPES (under grant 0116-11-7), which contributed substantially for the development of this work.

References

[1] J. Mitola, "The software radio architecture," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 26–38, May 1995.
[2] E. Blossom, "Gnu radio," http://www.gnu.org/software/gnuradio, 2009.
[3] M. Ettus, "Universal software radio peripheral," http://www.ettus.com/, 2009.
[4] P. Balister, T. Tsou, and J. H. Reed, "Software defined radio on small form factor systems," Mar 2007.
[5] WARP, "Rice university wireless open-access research platform (warp)," http://warp.rice.edu, 2009.
[6] KUAR, "Kansas university agile radio," https://agileradio.ittc.ku.edu/, 2009.
[7] B. Ackland, D. Raychaudhuri, M. Bushnell, C. Rose, and I. Seskar, "High performance cognitive radio platform with integrated physical and network layer capabilities," http://www.winlab.rutgers.edu/pub/docs/NeTS-ProWiN1.pdf, Tech. Rep., Jul 2005.
[8] L. Choong, "Multi-channel ieee 802.15.4 packet capture using software defined radio," Tech. Rep., Mar 2009.