THE COST OF AN ABSTRACTION LAYER ON FPGA DEVICES FOR SOFTWARE RADIO APPLICATIONS

Xavier Revés, Vuk Marojevic, Antoni Gelonch, Ramon Ferrús

Universitat Politècnica de Catalunya, C/Jordi Girona 1-3 08034 Barcelona, Spain,
{xavier.reves,marojevic,antoni,ferrus}@tsc.upc.es

Abstract – Software Radio applications require a framework to develop and deploy applications, especially those related to radio infrastructure. It is interesting that a given application may be executed on any software radio. But, since the hardware platforms used in this context will have multiple architectures and devices, a software layer to make applications independent from hardware is mandatory. Ad-hoc software for a given hardware platform may produce the best software performance. Conversely, when software is not targeted to any concrete platform the lost of performance may be excessive and the overhead introduced by any platform-dependent library could become intolerable. In this paper the resource utilization of a software radio application using a simple hardware abstraction layer is studied and compared to an ad-hoc implementation to make an assessment of the introduced overhead. The particularity of the hardware abstraction layer is that it runs on a platform which only processors are FPGA devices.

Keywords – Software Radio, Abstraction Layer, CDMA Implementation, FPGA.

I. INTRODUCTION

There exist an interest to provide increased flexibility to the radio equipment to be used in future wireless access networks. The additional flexibility may be obtained through radios that implement almost every radio function by means of software running on digital processors. When in this situation, it is possible to talk about a Software Radio, that is, a radio terminal where radio tasks are "simple" programs [1]. This concept is widely extended at present and the possible advantages of having software radio terminals and how to achieve such advantages to improve service provisioning (from the point of view of users, operators, service providers, etc.) are object of studies [2].

Although some software radio key features are easily assumed, like reconfigurability through software download (from any source, local or remote) making possible, for instance, the adaptation of terminal or network infrastructure to the equipment at the end other of the radio link, or providing ubiquitous connectivity throughout the wide scope of standards, it is not yet clear which has to be the common framework to deploy such features and any other additional to come in the future.

There are actually a wide bunch of issues to finally reach the envisaged software radio terminal. In this paper the software portability and deployment and their costs are focused. Concretely, section II makes an overview to what motivated the presented work. Following, section III, presents the key concepts of a platform abstraction mechanism implemented to deploy software radio applications on heterogeneous hardware/software platforms. Sections IV and V treat the development of such mechanism for FPGA devices and analyze the different overhead sources experienced when comparing the same system implementation for both cases, ad-hoc and abstraction-based. Finally, section VI draws some conclusions on the presented work.

II. MOTIVATION AND OBJECTIVE

A. General discussion

Most of software radio implementations found in the literature implement their low-level software (understood as functions dealing with intensive physical layer tasks) taking into account the hardware resources and features of the platform that is going to execute such application. The software constructed this way is hardly portable across different platforms and, although the core of software is programmed to make it easily adaptable to other applications (e.g. object-oriented approaches to simplify software reuse), there exists a period during which this core may be profiled to adequately work (e.g. correct real-time behavior) on a different hardware platform. This is because it does not exist a software context dealing with the arduous task of providing all the support that a software radio application requires. This support must include real-time, monitoring, inter-process communication, among others, at a reduced cost, that is, with low overhead. Even though there are multiple real-time operating systems running on different hardware machines with an important set of libraries to make software portable (e.g. POSIX), their features are not the most adequate to support the development of intensive processing tasks like those found in software radios. Consider, for instance, an intermediate frequency (IF) digital filter of 256 coefficients at a sampling rate of 50MHz. The processing required for such implementation reaches the astonishing (maybe at present not as much astonishing as some years ago) value of 12800 MMAC (Millions of Multiply-Accumulate operations). This processing demand must be covered, at present, by ASIC or FPGA [3][4] devices. These processors, strictly speaking, do not include the possibility to execute any OS in the market

like those found for general-purpose processors (GPP) or even digital signal processors (DSP).

The previous example put on the table the fact that not all the digital processors are equally treated and used. Although this is obvious, since the context where GPPs are generally found includes an OS and many support libraries to develop applications, why not considering a similar approach for DSP and FPGA? Here similar approach does not mean "similar philosophy", that is, programming by using libraries, but using the same libraries with the same functions that are compatible with GPP, DSP and FPGA environments. After all, the present state-of-the-art may require the combined use of FPGA, DSP and GPP for software radios. Unfortunately, it is not likely, although it is possible after some effort, to see an FPGA with RT-Linux or a DSP with Vx-Works. It would be simply a waste of resources because such processors must use their resources for other purposes. However, a mechanism to make programs portable across platforms, and that may use the platform resources independently of actual hardware configuration, is very interesting. Such approach, together with the concept of network distributed computing applied to embedded systems, would help the development of software radio equipment and applications where multiple embedded boards of different brands would collaborate to achieve the application objective.

B. Objective

In the previous paragraph, the need for a software development and deployment framework for Software Radios has been implicitly introduced. To make software applications portable across platforms, some kind of hardware abstraction layer is required since not all the hardware for software radios uses the same processors and has the same architecture. But beyond the hardware compatibility there is the software one. It may also be necessary to construct a software abstraction layer to make a given application to run on certain platforms with full software support (e.g. OS).

Then, given the exposed context, the objective is to provide a uniform library and/or software abstraction layer that works indistinctly on GPP (and associated OS), DSP (without any additional support but the mentioned software) and FPGA. Additionally, the abstraction layer must provide a seamless mechanism to aggregate to the overall hardware platform as much processing devices as required, of any brand, to distribute the software radio application across such devices in the optimum way. This feature offers the possibility to interoperate different platforms from different manufacturers within single radio equipment (base stations and even terminals), thus facilitating the implementation of open architecture radio systems where innovative software and/or hardware may be added as soon as they come to market, as it happens in the PC sector. Returning to heterogeneous processor environment, note that while the two first processor types (GPP and DSP) offer a similar programming paradigm, the last one (FPGA) offers a completely different paradigm. The design of the abstraction layer has to remind the processor differences and provide an adequate, single solution, for all of them. This single adaptation may produce an excessive cost in FPGA devices if their features are not taken into account.

III. THE PLATFORM ABSTRACTION LAYER

A. The application structure

Before entering into detail about the abstraction layer structure and features, some words about the structure of a software radio application must be given. This is important because application structure and abstraction layer are highly correlated.

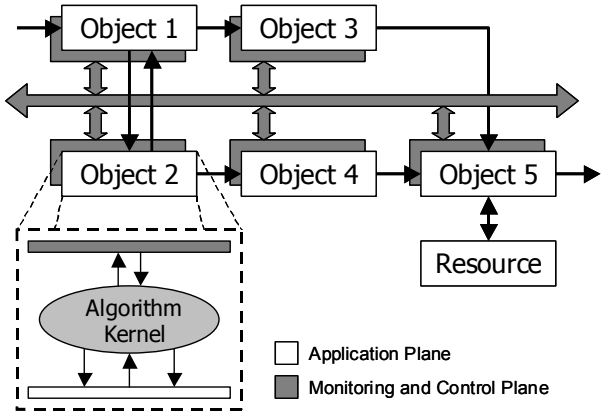

Fig. 1. Application structure and detail of an object.

A software radio application can be described as a set of objects concurrently running (see Fig. 1) on one or more processors (depending on their capabilities). Each one of the objects is programmed independently from the context (other objects surrounding) and has a set of interfaces to exchange data with other objects or with system controlling entity. To allow an almost seamless integration of objects, interfaces have a very simple model: data traveling through interfaces are like a sequence of samples of a signal. This model is valid to transmit any radio transceiver internal signal, parameters, etc. As shown in Fig. 1, the algorithm kernel (e.g. FIR filter, Trellis decoder, Wave generator, etc.) is isolated from the environment through a set of functions/blocks provided within the abstraction layer API (*Application Programming Interface*). This means that the abstraction layer, the only one that knows the hardware where the algorithm is being executed, deals with anything but the algorithm.

Bandwidth of interfaces and computing capabilities of processors on a given set of hardware are the most important aspects to take into account when translating the application to the hidden platforms.

B. P-HAL

Since the abstraction layer constructed deals with highly developed platforms (e.g. workstation) and also with almost pure hardware ones (e.g. multi-DSP/FPGA boards), it has been named P-HAL (*Platform-Hardware Abstraction Layer*).

The identified support that this abstraction layer has to provide, according to the application structure, can be summarized in the next few aspects (see also Fig. 2).

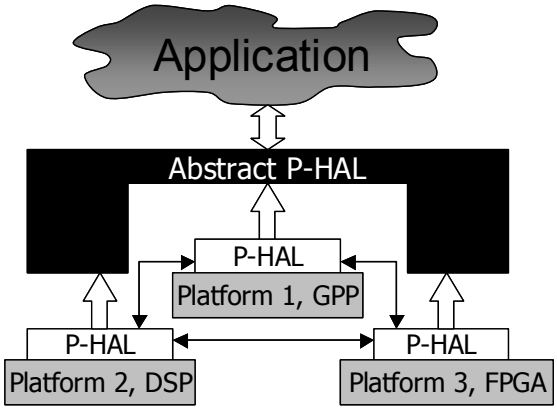

Fig. 2. Schematic representation of P-HAL

1) Seamless communication

P-HAL provides transparent communication mechanisms among on-board and off-board application objects. Each object is assigned an identifier, similarly as IP addresses are assigned to machines, and application data packets flow through routing mechanisms. This means that it is not strictly required a direct connection between the platform running an object and the platform running another object linked by a given interface.

Different P-HALs interface through any standardized connection, like an Ethernet, PCI backplane, etc., where only the connection address (and port) is known of other P-HALs, following the IP concept.

2) Monitoring and control

Through monitoring and control interfaces, each object offers the possibility to periodically observe or modify the value of internal parameters or gathered information. It is crucial in software radio environments to obtain information about the radio interface and modify its behavior if required. Such interfaces allow doing it.

3) Real-time

A software radio application requires tight timing control to work properly. Since the application runs distributed on different platforms, the timing control has to be extended to all of them. Although each individual platform controls the local timing of processes, there is a synchronization procedure among platforms that allows them keeping the same timing. To make the synchronization precision and real-time monitoring less critical, time is divided into slots that determine the periodicity of execution of tasks, distribution of data and CPU assignment.

4) Object launch and mapping

Since the application is described independently from the hardware, given a particular hardware organization, the software objects have to be distributed on the different processors according to computing demand and interfacing bandwidth. After this initial procedure is done, interfaces have to be configured (virtual circuits) to carry data accordingly. Each platform P-HAL is in charge of its internal resources. The last step is making the objects to run.

Note that the objects must be available as binary executable for the targeted processor. Another possibility would be providing the source code and compiling it to obtain the executable file. If only P-HAL API is used, there should be no problem to obtain such executable. But the used language is another step to cover since there can be different languages involved when several objects independently programmed are joined in a single application. For instance, C/C++ is a good candidate for GPP or DSP but VHDL or Verilog are adequate for FPGA. Anyway, this is a problem likely solved in the future by new generations of CAD tools.

IV. P-HAL FOR AN FPGA BOARD

A. Constituent blocks

The selected language to describe the P-HAL functions for FPGA is VHDL. The resulting blocks are grouped under a black box that has an interface (what we could call the API) that allows connecting the algorithms to it. The blocks included within this black box are in charge of carrying out all the P-HAL tasks. In Fig. 3 a detail of an example of these blocks is shown.

For simplicity in Fig. 3 only one task is assigned to one FPGA device, although there is no limit on it. Actually this may represent a part of a partially reconfigurable FPGA. What remains unaltered is the fact that each task has a set of associated blocks that run concurrently on the same FPGA (or maybe on another FPGA with direct connection) to provide the adequate support for the local hardware.

Note that the required blocs from one platform to another may differ depending on the actual hardware to interface. The example shown in Fig. 3 corresponds to an FPGA that provides access to a local high-speed bus with capabilities to access other boards, two ad-hoc interfaces with other FPGAs (or other devices on the same or on another board) and access to SRAM memory (some objects may require this resource, like Direct Digital Synthesizers). To provide

maximum flexibility to the process of mapping algorithms on the hardware set, all the interfaces are made available. Moreover of resources dedicated to interfaces and switching/routing packets among them, there is a packet routing table to adequately configure virtual connections and the timing control block to guarantee the correct timing behavior.

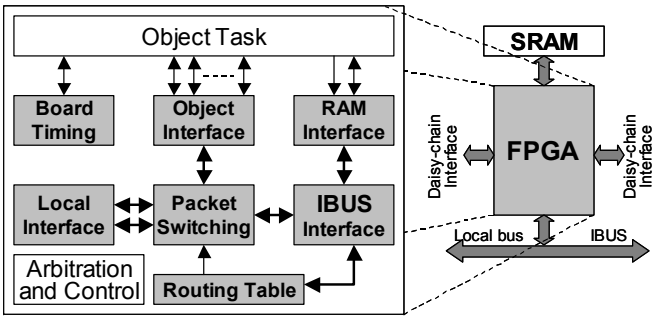

Fig. 3. Example of blocks on a FPGA to implement P-HAL

B. Resource utilization

Table 1 summarizes the amount of resources consumed by this set of functions when measured on Xilinx Virtex family of FPGAs [5]. The measurement is made using logic elements (LE), each one considered to have one flip-flop and one 4-input look-up-table (LUT). Those implementations that do not use LUTs or flip-flops are translated to approximately equivalent LEs. Additional resources for proper internal FPGA routing are not accounted. In the table the transfer speed achieved by the 32-bit interfaces is also shown. That some overhead is added because of the abstraction layer cannot be discussed, like happens in DSP and GPP. The extra amount of required computing capabilities (used basically for interfacing functions) has to be assumed when implementing the true application with platform abstraction. Note that figures in Table 1 are relative to the implementation on a given environment. A different context would lead to different resource demand. Additionally, for comparison purposes, the occupancy results are approximately unified under LEs to avoid mixing other available FPGA resources, like dedicated memory blocks, which are actually used to implement P-HAL.

There are two major issues to consider in the overhead. The first one is the problem of increased clock frequency (compared to data sampling rate) to consume the inactivity periods. The second one is the problem of buffering data to accept interface latencies. As shown in Table 1, buffering may be the most consuming resource in terms of chip area. By the other hand, increased clock frequency, despite the inactivity periods, increases the power consumption.

Buffering seems to be the worst element to combat in this implementation of P-HAL. In DSP and GPP there exist inherent buffering because everything has to be stored in memory, but FPGA devices can be seen like stream processors that do not store data but only process them. When the concept of packets is introduced it is necessary to have the capability to store, modify, route, compound, etc. packets. Unavoidably, this adds storage requirements to FPGAs. If external memories are used, like in other families of processors, the latency when accessing memory transforms and FPGA into something similar to such other processors, loosing part of the performance. So it seems that FPGA suffer the consequence of having taken the software model from the more extended GPP paradigm (easily extended to DSP).

Table 1
Rounded resource utilization example of P-HAL.

	Speed*	LEs
IBUS Master/Slave, Bi-dir., 64bytes buffer	4 (~1.5)	280
Local Interface, Bi-dir., 64bytes buffer	4 (~2)	160
Object Interface, Bi-dir., 64bytes buffer	4 (~2)	220
RAM Interface, Bi-dir., 64bytes buffer	4 (~2)	150
Packet switching, four interfaces Bi-dir. switch	4 (~2)	350
Arbitration and control	N/A	40
Routing Table, 16 entries	N/A	50
Timing	N/A	80
Buffering (one FIFO byte)	1	1

* Peak transfer unidirectional (approximated maximum average bidirectional) in bytes per second relative to clock.

To reduce buffering, the packets may be shorter but this implies increased traffic on interfaces, thus reducing their actual bandwidth and also reducing the maximum allowed latency. Then, a trade-off between interface bandwidth and buffering occupancy is unavoidable. Just as an example, a FIR filter generating at its output a stream of 16-bit samples at 1MHz requires a sustained bandwidth of 2Mbytes/second. If, for instance, the interface used to carry this stream has a peak transfer rate of 100Mbytes/second but uses to have a latency of about 10us before transmitting (e.g. round-robin bus arbitration with many bus users), a buffering of at least 20 bytes is required.

V. A REAL IMPLEMENTATION ON AN ABSTRACTED FPGA PLATFORM

In [6], the implementation of an indoor CDMA system implemented on FPGAs was presented. There the implementation was ad-hoc, adapting each block to the well-known available hardware. With this in mind, a reasonably small FPGA board could be used but when the abstract functions are used much more resources are required. As a summary, the number of LEs used for that implementation is shown (see [6] for details):

- Mobile terminal: required 6380 LEs, fitted on one board with 9216 LEs (8 FPGA).
- Base station for 16 users: required 99350 LEs, fitted on 17 boards with 9216 LEs (156672 LEs).

To compare the previous figures with real devices, just to say that a Virtex XCV400 could store the aforementioned LEs for the mobile terminal. More modern devices of this family include on-chip dedicated arithmetic that would help the improvement of the ad-hoc solution. But it does not exist the possibility to specify these special functions when the application is programmed without any knowledge about the hardware that is going to execute it. There only exist the possibility that the compiler is able to map the program to the actually available resources, which are variable from family to family.

There are different sources of overhead when using the abstraction layer:

- The additional LEs for P-HAL added to each application object.
- Inefficient mapping of application objects on FPGAs. That is, some FPGAs may be partially empty because the available room is not enough for another object.

Considering a board (or a set of them) including several FPGAs (XCV150: medium-small size to allow reconfiguration of individual blocks) with the previously mentioned interfaces and dividing the terminal and base station functions into conceptually separated objects (11 objects for one user mobile terminal, 4 for 16-users base station transmitter and 5 double objects, because of diversity, for each base station receiver), the occupancy would reach the following rounded figures:

- Mobile terminal: 16510 LEs, fitted on six FPGAs with 3456 LEs each (20736 LEs).
- Base station for 16 users: 250510 LEs, fitted on 98 FPGAs (338688 LEs).

To obtain these results a manual mapping of objects has been done, which in general is better than an automatic mapping. On each FPGA all the P-HAL resources are allocated plus one additional object interface per each additional object running on the same FPGA and an additional packet switch if more than one object and per every three additional ones.

The first approach results in poor resource utilization that represents an increment exceeding 150% at both ends. A second approach would try to reduce the amount of resources for interfaces, but to provide full support (control, monitoring, timing, etc.) almost nothing can be reduced. Instead of dividing the design into multiple FPGA, if all the terminal or base station could fit within a large FPGA, the figures would relax:

- Mobile terminal: with P-HAL support for 11 objects, 10960 LEs (fits into one XCV600).
- Base station for each user: with P-HAL support for 14 objects, 155440 LEs (each user considered individually could fit into one XCV600).

Now the increase in resource utilization is limited to about 72% in the terminal and only about 56% in the base station. From one approach to the other the basic difference is that P-HAL resources are shared among more objects (thus increasing required P-HAL performance). Then, the conclusion is that abstraction makes more sense in large FPGAs where overhead has lower impact. The same is found in GPP or DSP since software abstraction layers gain validity in powerful processors and not in small ones.

VI. CONCLUSIONS

In this paper the cost in terms of resource utilization when implementing Software Radio applications on abstracted FPGA platforms has been evaluated. When adapting FPGAs to work in distributed processing environments there is a very large increment in resource demand. A possible structure of such adaptation, P-HAL, has been presented and figures derived from its implementation together with figures obtained from an ad-hoc radio implementation have been compared. The result shows how abstraction in FPGAs has a great impact on system available resources.

AKNOWLEDGEMENTS

This work has been supported by CYCIT (Spanish National Science Council) under grant TIC2003-08609.

REFERENCES

[1] J. Mitola III, “The Software Radio Architecture”, *IEEE Communications Magazine*, vol. 33, pp. 26-37, May 1995.

[2] http://www.sdrforum.org

[3] A. Gathener, T. Stetzler, M. McMahan and E. Auslander, “DSP-Based Architectures for Mobile Communications: Past, Present and Future”, *IEEE Communications Magazine*, vol. 38, pp. 84-90, January 2000.

[4] M. Cummings and S. Haruyama, “FPGA in the Software Radio”, *IEEE Communications Magazine*, vol. 37, pp. 108-112, February 1999.

[5] http://www.xilinx.com

[6] X. Revés, A. Gelonch and F. Casadevall, “Software Radio implementation of a DS-CDMA indoor subsystem based on FPGA devices”, in *Proceedings of the 12th IEEE International Symposium on PIMRC*, vol: 1, San Diego (USA), 30 Sept.-3 Oct. 2001