EURASIP Journal on Applied Signal Processing 2005:16, 2626–2640

Software Radios: Unifying the Reconfiguration Process over Heterogeneous Platforms

Xavier Revés

Departament de Teoria del Senyal i Communicacions, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain

Email: xavier.reves@tsc.upc.es

Antoni Gelonch

Departament de Teoria del Senyal i Communicacions, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
Email: antoni@tsc.upc.es

Vuk Marojevic

Departament de Teoria del Senyal i Communicacions, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
Email: marojevic@tsc.upc.es

Ramon Ferrús

Departament de Teoria del Senyal i Communicacions, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
Email: ferrus@tsc.upc.es

Received 31 January 2004; Revised 21 February 2005

Future radio transceivers supporting the software radio concept will provide increased features for radio access networks. However, the reconfiguration of radio equipment requires the existence of an architecture, a common framework, which allows the flexible management of software running on radio processors. Such a framework must take into account the heterogeneity of hardware devices and platforms for radio applications. Since the flexibility has a cost in terms of added overhead, a conceptually simple but efficient structure that allows powerful mechanisms to develop and deploy software radio applications is required. This paper describes our approach, the reasons that motivated it, and some implementation issues. The proposed framework is essentially based on four items: an abstraction layer which hides any platform-dependent issue, a simple time-driven software structure, a delimited interface format for software blocks which does not actually constrain communication, and a global time-reference mechanism to guarantee real-time behaviour.

Keywords and phrases: software (-defined) radio, reconfiguration, flexible radio, software abstraction layer (middleware), hardware abstraction layer, radio execution environment.

1. INTRODUCTION

Software radio [1, 2] is an emerging technology that promises great advantages to radio system engineers and to the wide segment of potential or actual users of these systems. The concept spans from the radio terminals (base stations or mobile equipment) to network management, including resource management for optimum service provision. The key aspect that unleashes the potential behind software radio is the unbinding of radio capabilities from hardware. Software running on digital hardware processors, including general-purpose processors (GPP), digital signal processors (DSP), field-programmable gate arrays (FPGA), and so forth, may accomplish the hard processing tasks from intermediate frequency (IF) to bit stream at the physical layer and, of course, the tasks at higher layers. The increase in flexibility allows considering the radio interface as a dynamic and adjustable part of the system rather than a fixed one. Unfortunately, the state-of-the-art has not yet reached the technological point to construct an end-to-end completely flexible radio system. The reason is that some parts still remain working in the analogue domain.

In software radio equipment, wide-bandwidth high-resolution analogue-to-digital and digital-to-analogue converters change the information from the real world to the numerical world. Here the information is managed by soft-programmable digital processors. At present, there is a plethora of processors whose brute force may deal with harder processing tasks, like IF filtering, Viterbi decoding, and so forth. A processing demand of some GOPS (billions of operations per second) is not scaring any radio designer any more. In addition, other tradeoffs emerge, like power consumption, especially in battery-powered terminals, the number of processors, their architecture and interconnection. Anyway, after having found a suitable combination and organisation of processors, that is, a heterogeneous processing platform, the full potential of software radio comes to the scene.

Software radio embraces many possibilities to offer improved and personalised radio services; for instance, quick standard evolution through open standards that are updated whenever new needs or new solutions emerge, adaptation of the access network configuration according to instantaneous population distribution, global roaming, and so forth. However, achieving these advantages is not possible without a common framework and a formal design methodology (e.g., [3, 4]) that fully expands software radios. This framework must at least include a mechanism to specify what software must run on the terminal (fixed or mobile), either directly downloading the executable files or through a more reasonable way, and another to ensure the safe behaviour of the software, keeping radio terminals within standard restrictions and respecting the user-operator contract bounds. A third, complementary and very important mechanism can be also envisaged: a network *intelligent* entity, centralised or distributed, determining the *when*, *why*, *what*, and *how* in a reconfigurable software radio environment (W3HSR). For instance, when the terminal needs a reconfiguration, why must it be reconfigured, what parts of the terminal are to be reconfigured, and how to do it.

The next sections present a step further towards a possible answer to the problem of the development of software radios. It is a simple execution environment, a run-time management and abstraction layer that due to the simplicity of its concept provides a powerful mechanism to develop and deploy software radio applications. Because of the high-level plane to where this tool moves the application, not only radio applications may take benefit from it. However, its internal architecture and specific features are oriented to provide good support to the particularities of radio applications.

2. THE RADIO RECONFIGURATION PROBLEM

The first two questions (the "when" and the "why") receive an answer from the status of jointly managed, different wireless access networks. Independently of the complexity of the algorithms used to decide whether a terminal should be reconfigured or not, the only issue to consider in the context of this paper is the possibility that the management elements may have access to contextual information provided by each terminal.

The remaining two questions (the "what" and the "how") may be more difficult to answer. Every terminal has its particularities that make it difficult to homogenise the reconfiguration procedure. As a general approach, software may come from different sources, not being the same for different terminals. A possible reason is that software optimisation makes optimised products that are potentially better than the products of competitors. Then it is likely that the optimised algorithms are not public to avoid competitors reaching the same performance level. This single reason already advocates for different software solutions on terminals, although others are possible. There are other issues related to software that are not discussed in this paper, like the guarantee of software security, download mechanisms, user-operator contractual bindings' agreement, and so forth [5].

At the hardware level, there exists a multitude of different mobile terminals. Such a variety of terminals would be, by itself, a source of heterogeneity for the execution context of a software radio. In base stations, an "unlimited" number of hardware combinations are already possible employing embedded processing cards from different manufacturers. Going beyond, why we do not consider the possibility to have portable terminals with "variable" plug-and-play hardware. Imagine a user selecting the RF minicard from manufacturer A because of its high sensibility, dual-channel operation, and maximum transmitted power among others. The IF minicard may be from manufacturer B because of the number of simultaneous channels that it may capture, the computational performance, the power consumption, and so forth. Finally, baseband (BB) minicard is selected from manufacturer C because of the power consumption versus bit rate, weight, and price. All these cards can be inserted within a chassis including a general-purpose processor running an operating system and interfacing the keyboard, screen, camera, and any other terminal peripheral. Over such a heterogeneous platform, the software radio application must be executed, not only involving the general-purpose processor but above all the processors within RF, IF, and BB sections. Hence, in any radio interface endpoint, there is a large amount of heterogeneity.

The previous paragraphs introduce the problem of providing a safe, unified, regulated environment for the reconfiguration of software radio terminals with heterogeneous hardware and probably nonpublic heterogeneous software solutions. Such environment would allow taking the full advantage of the software radio concept to the network management entity, the users, the hardware manufacturers, and the software programmers.

3. EXECUTION ENVIRONMENT

From the software radio network control centre point of view, maybe the solution could just be as simple as having software radio terminals with homologated software on them, coming from any provider that the user may buy depending on its terminal hardware. The terminal would then accept commands from the control centre to move from one mode to another (e.g., SDR forum approach [6]) within a given list of possible agreed (standardised) modes and then exchanges the software binaries loaded on the processors. For a given mode and purchased software, the terminal could simply have one or more files with information about the software blocks to launch, which are all locally or remotely accessible. In general, local access to software (e.g., stored in a memory card) should be preferable over downloading from a remote database because downloading software is relatively slow. However, this option is useful for special situations. Also other options may be considered like, for instance, compacting the amount to download not through file compression but through efficient methods of describing software. The availability of some kind of "virtual radio machine" (VRM) inside the terminal would provide yet another possibility. Thus, it is quick to describe most of typical operations in radio systems through instantiation of radio entities together with some additional parameters, while avoiding the download of detailed information.

A first approach may be as simple as the mode switching stated before. Approaches that are more flexible could increase the possibility of improvement of other relevant radio issues, like resource management (RRM), quality of service (QoS), battery life, and so forth. As flexibility adds value to radio interfaces, the conceived software-hardware structure should be the departing point from which to explore the problems associated with the terminal reconfiguration process. Before entering into the detailed description, it is necessary to consider different software contexts. If in software radio applications it is desirable to use any hardware platform with any kind of processor (GPP, DSP, FPGA, etc.) and use software from any homologated and licensed provider, at least the constraints within one of the next three envisaged contexts (or a combination of them) have to be granted.

(i) Software for the platforms comes in form of binary executable code.
   (a) A compiled version for each possible processor kind (or family of compatible processors) must exist.
   (b) The code must not include any action that may depend on surrounding processor hardware unknown at compile time. Then, a set of standard dynamically linkable libraries needs to be available on the platform.

(ii) Software for the platforms comes in form of high-level programming language.
   (a) Each platform has to have a compiler that is able to translate the high-level algorithm description into a binary executable code.
   (b) Libraries, dynamic or static, have to exist in the target platform to overcome hardware dependencies.

(iii) Software for the platforms comes in form of interpretable language.
   (a) A virtual machine (VM) has to run on each platform. In the radio application context, a low-resource utilisation and low-overhead VM are mandatory.
   (b) Specialised radio algorithms must be available to VM to achieve a minimum performance. Even different libraries for different standards could be required to solve the peculiarities that each one may offer. These libraries must come with the platform or similarly as stated in the first situation in the list.

In any case, the solution to terminal heterogeneity is finally libraries and/or virtual machines. They hide hardware particularities (abstraction), and algorithm-related software is at the end translated to processor-specific language/executable. As a result, the same restrictions, methodologies, concepts, procedures, and so forth. found in general-purpose computing platforms are found in software radio platforms. Certainly, if only the software radio applications' niche is considered, it is possible to make some simplifications over the more general approach. Nevertheless, one may wonder if this is the only possibility. Is there another efficient way to program the radio algorithms not requiring the availability of processor-specific executables or specialised radio libraries for each platform? With the current state-of-the-art, the answer is "probably not."

4. THE ABSTRACTION LAYER

The first and basic step to advance in the process of defining a common framework to develop and deploy software radio applications is eliminating platform (hardware and support software) dependencies. The next step is making it simpler to specify radio applications, worrying only about algorithm description. Based on available tools, the languages used to program algorithms within the presented environment are standard high-level languages like C/C++ and VHDL.

In this context, the possibility to add as much processing resources as required for a given application, not having to modify it at all, is of high importance. The possibility of configuring the radio terminal with different hardware from different providers requires the capability of adding (and removing) plug-and-play hardware to (from) the system. The different hardware topologies, configurations, and above all, assigned tasks impose restrictions on the integration of different hardware to construct software radio platforms. Because of these and the previously stated objectives, a multiplatform software abstraction layer, *platform-hardware abstraction layer* P-HAL has been defined and constructed to provide the following set of features (schematically represented in Figure 1):

(i) real-time seamless exchange of information from one P-HAL compliant platform to another (bridge);
(ii) isochronisms of data and processes running on different platforms (sync);
(iii) coordinated process control and scheduling on any platform (kernel);
(iv) real-time system monitoring and data and statistics retrieval (stats);
(v) real-time adaptation of processes' set-up parameters (stats);
(vi) event logging and error control (kernel).

From the application's point of view, the three first features in the list produce the effect of having a single platform. Thus, the same application description works on a single machine or on multiple distributed machines, if the P-HAL layer exists in between the application and the hardware. It does not

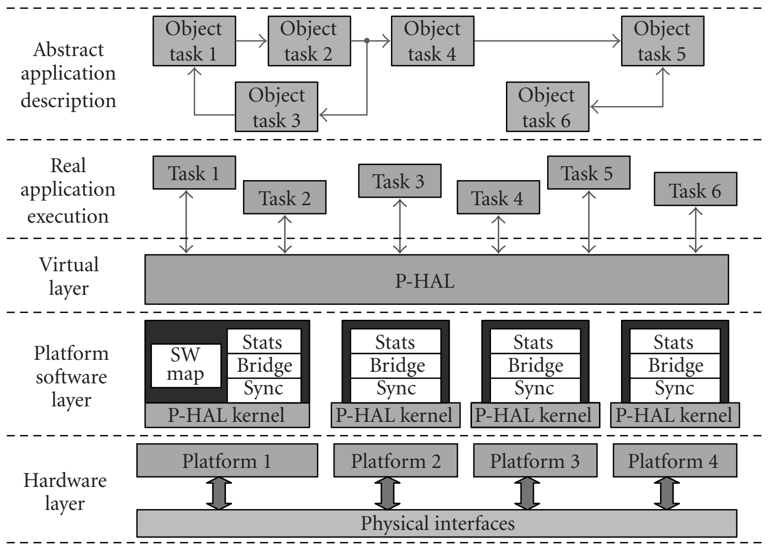

Figure 1: Representation of the platform abstraction layer.

matter which hardware runs a specific part of the application, this part will just observe other parts as being aside. The previous situation and other P-HAL details appear in Figure 1, where four processing machines (platforms) have their respective P-HAL. From the point of view of the application blocks, only a virtual P-HAL entity is visible. In any case, the application is not aware of the underlying hardware. The following sections describe the required mechanisms to achieve the pursued objectives.

4.1. Merging multiple platforms into one: interplatform data bridging

In the proposed model, one platform almost does not need to know anything about other platforms with which it cooperates. Because of this, the low-level communication process has to be simplified and not tied to any specific configuration. For such purpose, considering the IP network model is interesting. Different platforms sharing a network connection may use an IP address and a port number as the single union point to exchange all the necessary information. Since software radio applications require tough timing control and fast interfaces (with reduced latencies and very high bandwidth), it would be appropriate having the different platforms connected through high-speed backplanes or even dedicated connections. In such context, the IP model is still valid. In an embedded processing environment, using the backplane bus with a 32- or 64-bit address may have the same meaning as an IPv4 or IPv6 address in a network domain. A given platform has only to know, for instance, one backplane address for each additional accessible platform. This address serves as interface to send information to processes (parts of the application) running there or to the corresponding P-HAL counterpart. The address actually does not represent a typical memory address but rather an access point to the platform that accepts streams of nonoverlapping data. Before P-HAL is completely set up, only one platform, which receives the name of P-HAL "master," knows these addresses. Then, it distributes them to other platforms to construct the whole P-HAL environment.

Packets of data convey the application interface streams. A packet goes either to another task within the same platform or to another platform. In the latter case, P-HAL bridge handles the transfer, as it knows the addresses (IP, backplane, etc.) of all the access points of the remaining platforms. If there is no physical connectivity with a particular platform and data has to be moved to it, it is possible to use intermediate platforms, like in an IP network with its routers and gateways. This, however, is not considered an interesting solution for real-time applications because of the increased delay. The bridging process is possible as each platform is assigned a range of virtual addresses (or labels). The processes on a platform are assigned a label within the platform range and a routing table is created.

4.2. Synchronising platforms and timing control

With the objective of observing a single and virtual time throughout the abstract platform, it is mandatory that the internal timers within each particular platform be synchronised with enough precision. The features of a communications interface, the single interplatform interaction point, determine the amount of available precision. Since there are unknown latencies and delays in the communication process, it is obvious that full synchronisation of all the timers is difficult (actually impossible) to achieve. Therefore, the expected amount of synchronisation has to be enough for the timing control provided by the virtual platform. For instance, if actions in the different real platforms are periodical with a period of one second, a misalignment of one millisecond should not be critical. Sync is the component within P-HAL that deals with timing issues.

With the objective of reducing synchronisation requirements, the time on each platform divides into time slots that

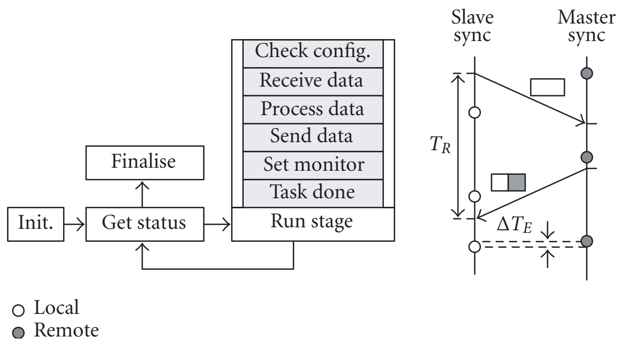

FIGURE 2: Object task structure and timing synchronisation procedure.

become the minimum references on each individual platform. Then, each individual platform runs, governed by its P-HAL layer, at the best-suited timing interval, no matter what it is: 1 second, 60 milliseconds, 50 microseconds, and so forth. Merely two restrictions are imposed in the presented model. First, synchronisation error with other platforms must be much lower (in the order of 1%) than the time interval selected. Second, there must be an integer relation between any two combinations of time-interval lengths of two different platforms. Figure 2 depicts the synchronisation procedure based on the measurement of the round-trip time (T_R) of a synchronisation packet. Strictly speaking, the maximum error associated to this synchronisation procedure is half the round-trip time, although in case of an almost symmetric delay, the error is much lower (ΔT_E). Since the misalignment is always lower than the time to move any data packet from the origin process running on one board to the destination on another, there is no need to take care about it. One master sync and one slave sync appear in the figure. This is the relative role played by two different platforms. In general, there will be a single master sync if there exists full connectivity of one platform with any one else, but the system is not restricted in this sense.

Time slots control the movement of data and the process scheduling. The scheduling mechanisms do not introduce significant overhead in the focused application type. This is because all the tasks, or at least those that require more computational power, always have work to do. More precisely, every process periodically gets some CPU time during each time slot to perform the operations necessary within such time interval. For instance, a digital filter processing data at 100 ksamples per second with a time slot of 1 millissecond performs 100 multiply-accumulate (MAC) operations per time slot. Each process receives CPU time once per time slot to perform its assigned operations, at any position and in any order with respect to other processes within a time slot. Upon completion of operations within a time slot, the process remains stopped until the next time slot.

4.3. Describing an application

For the purpose of application development, we assume that the above-proposed virtual platform with communication and timing mechanisms is provided. The traditional application programming interface (API) scheme is the way to access services provided by P-HAL. The description of an application in the context of P-HAL uses an object-oriented-like approach [7], although the term "block diagram" is preferred because "object-oriented" is rather associated to programming environments using C++ classes, for instance. From Figure 1 (abstract application layer part), an application is made of several blocks (objects) and interfaces indicating the data flow. Objects are programmed independently without worrying about the context of execution. Some (few in general) interfaces-per-object provide the possibility to exchange information with other blocks. A simple definition of the data format of object interfaces is important. The definition, moreover of providing a means for data format agreement, takes into account the requirements of any interface between different objects of a software radio application. Although any standard interface description language (like IDL, used in CORBA applications [8]) could be applied, a more simplistic approach is useful to avoid unnecessary waste of resources without losing generality. Interfaces are then considered to carry out the data samples of time-variant signals (e.g., $x(n)$, $z(m)$). Any standardised or "easy-to-interpret" binary format (e.g., 16-bit two's complement, 32-bit IEEE floating point, etc.) represents the amplitude. These "signals" will travel through the overall P-HAL implemented mechanisms. The software block does not know their destination because of the mentioned independence between application description and object programming. In the previous filter example, it processes its input samples as they arrive, according to the time-slot divisions, and generates output samples at the same speed. No work is done while data is not available.

Like in any block-diagram-oriented description, a hierarchy is used to encapsulate a set of objects within a more complex function. At the lowest object level, an algorithm is described using any of the aforementioned approaches: binary file, high-level code, or function instantiation (VM-like approach). For any source, however, the original program structure, as shown in Figure 2, must follow some rules. These consist of an initialisation stage, a time-slot-based execution body, and a finalisation stage. Within the execution stage, any incoming data is processed and outgoing data is generated. From the point of view of the software object, and due to the message-based process structure, the execution proceeds as long as data is available. Upon reaching the end of data processing, P-HAL timing control entity assesses the correct application behaviour in terms of real time. The shown structure stems from the generally employed sequential programming but it may receive a different interpretation when applied to processors that inherently use different programming paradigms (fully concurrent processors may do everything at the same time).

During the initialisation stage, the object may draw a set of parameters (through P-HAL functions) to create a profile of its actual tasks. The set of parameters depends on the object programmer and its purpose at the time of programming. However, it might be interesting having a subset of

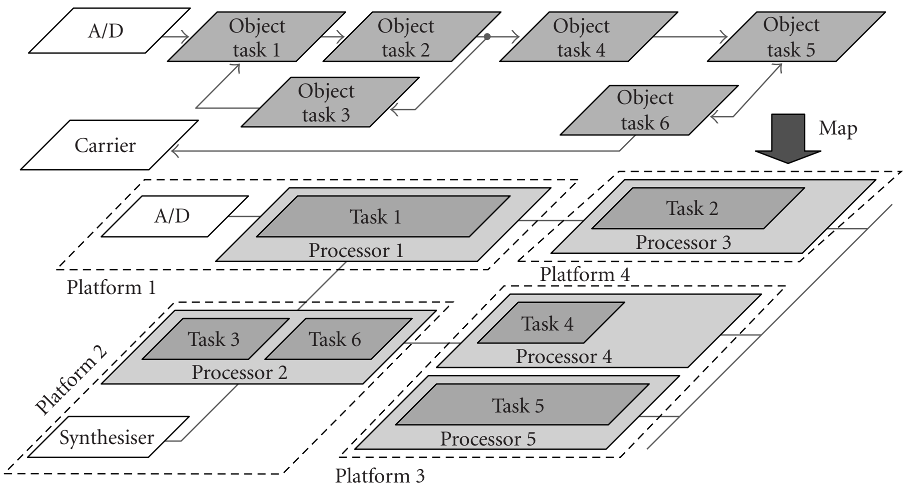

Figure 3: Sample mapping process.

predetermined parameters for some particular blocks to allow a general management of software through automated tools. Following the filter example, the cutoff frequency, passband, attenuation, and so forth. could be specified. The object would then compute the required coefficients corresponding to this purpose during the initialisation stage.

Additionally, there are some special blocks (dark grey in Figure 3). These elements must be available to all software radio applications to make them portable across platforms. "A/D" and "carrier" are two possible examples in the stage of waveform processing.

4.4. *Mapping and launching software*

After defining the application, it may be loaded into the computing machine. Only P-HAL knows how to manage the hardware for the purpose of software loading. The loading process depends on the P-HAL frontend. Loading is possible by using a set of commands. Simply stated, before having the application running, a set of steps is necessary.

(i) P-HAL receives a plain description of the application objects to launch and their interfaces, including parameters about the number and type of operations per second, data precision, memory required, interface bandwidth, and so forth.

(ii) P-HAL receives a plain model of the hardware, including processors and its computing capacity, interfaces and their transfer speed and modes, type and amount of resources, and so forth. Interface speed is especially important for interfaces between different platforms (used by bridge).

(iii) Using the two previous information sets, the mapping component (one per overall platform, see Figure 1) within P-HAL selects the most appropriate processor to run an object.

(iv) After mapping, the binaries are loaded into each processor. P-HAL requests them through its frontend in case they are not already available on the corresponding platform.

(v) P-HAL configures internal interfaces to provide the required connectivity to objects.

(vi) With the binaries loaded, the initialisation process starts and then the application runs.

(vii) P-HAL monitors the real-time behaviour based on the defined time slots. If a real-time fault happens on any process, it is possible to resume the mapping step to remap the objects that incurred into the fault.

In Figure 3, a very simplified mapping process is shown for some blocks of a radio receiver. Mapping starts at the block closest to A/D because of its processing demand, which is in general the highest among the receiver blocks. It is mapped onto adjacent and most powerful processors. That is, the mapping procedure is priority driven, where priorities are assigned to blocks by assuming as relevant parameters the processing demand and the required interface bandwidth. Complex mapping algorithms can be applied (e.g., [9]), especially when, for example, P-HAL must provide support to a base station implementation.

4.5. *Monitoring and modifying software behaviour*

Due to possible software malfunctioning, monitoring is a necessary part of the system. Clearly, in addition to this software behaviour, some relevant information from the application side has to be provided to the radio access network [1]. For such a purpose, P-HAL includes two additional control mechanisms. Firstly, in addition to the static parameters assigned to each process according to application needs and retrieved at the initialisation stage, the object may include dynamic parameters to update the software behaviour whenever system management requires it. The second control mechanism provides data in the reverse sense. The application sets some parameters that external entities may use to monitor the radio interface. In both cases, the application designer decides what parameters to use from the list available to the external management entity. P-HAL API supports all the previous tasks. The application has solely to worry about getting and setting parameters. Similarly, application

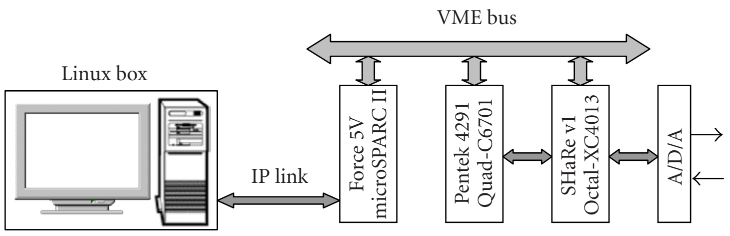

Figure 4: Integrated hardware.

managers use P-HAL frontend to arrive at each parameter to be set or retrieved.

The modification of the software behaviour is not only associated with modifying dynamic parameters but also with modifying objects (exchanging ones for others) and interfaces. To some extent, this modification represents submitting a new application description following the previously described steps. The purpose, however, is slightly different. The objective is to provide differential-like modification of objects running on the platform. This means that if one or more blocks have to be modified under some given circumstances, it will be quickly done at run time, pausing application during rearrangement of blocks and interfaces. The aim is to provide a means to toggle from one mode to another in a seamless manner without disconnecting the radio interface. Support for such a mechanism is mandatory within P-HAL as well as within the application description. Moreover, it provides a mechanism to optimise the computing resources on a platform as a function of the environment (e.g., C/I, ACP, transmitted power, etc.).

5. APPLICATION ON A REAL HETEROGENEOUS HARDWARE PLATFORM

A real heterogeneous hardware platform is the test bed to profile the stated concepts and framework and assess its real-world applicability. Without the intention of limiting the scope of P-HAL, the integration of heterogeneous hardware under the same abstraction layer is implemented on five different platforms. Each one only knows where other platforms are located in its interface space, following the single address approach introduced before. Considering the next five platforms (see Figure 4), the heterogeneity becomes obvious.

(i) PC with a Pentium-III at 500 MHz processor and Linux Red Hat 7.1 distribution OS.

(ii) Force Computers [10] Force-5V diskless workstation with microSPARC^{TM} II (32-bit RISC microprocessor) offering 34 MFLOPS peak performance at 110 MHz and Solaris 7 OS.

(iii) Pentek [11] board 4291 with four Texas Instruments TMS320C6701 DSP processors at a maximum of 167 MHz. Total computing performance beyond 5 billion operations per second.

(iv) SHaRe [12] FPGA board furnished with eight Xilinx XC4013 devices and a total performance of about 4 billion MAC operations per second.

(v) Data acquisition board with a single Xilinx XCV150 FPGA, high-speed A/D (12-bit), and D/A (14-bit), 500 MHz bandwidth and sampling rate beyond 100 Msamples/second.

Except in the case of DSP processors and Virtex FPGA, the remaining hardware is relatively old. The use of this hardware is advantageous in terms of forcing the implementation of an abstraction layer that imposes low system overhead. With low overhead, most of the processing capabilities of processors remain for the application. Processors able to cope with processing demand of software radio applications that run at low frequency and/or with low logic utilisation are interesting for battery-powered terminals, where power and size are major concerns. Although in the computing domain the tendency seems to move to highly featured and huge applications using plenty of resources of hardware, features like reduced size and power efficiency are dominant in wide market sectors.

5.1. Interfaces considerations

The platforms deal with four different interfaces, as observed in Figure 4. The PC and workstation interface through a TCP/IP over a 10BaseT link. This relatively slow interface guarantees a synchronisation error (workstation being the system's time server) lower than 0.25 millisecond and provides a net transfer speed of less than 0.5 MBps (in-system measurement). The workstation, DSP board, and FPGA board share a common VME bus backplane whose transfer speed, depending on the boards involved in the transmission and the data format and sense, ranges from about 2 MBps to approximately 80 MBps. Taking the best combination of these transfer speeds, the synchronisation error between the DSP and FPGA boards is less than 1 microsecond. DSP and FPGA boards share another interface (partially ad hoc) limited to 66 MBps. Finally, the acquisition board is connected to the FPGA board with an ad hoc interface at up to 128 MBps (peak), which is used to transfer the system time to the last board with an error also lower than 1 microsecond. According to the previous timings, the PC and the workstation would use time slots of 25 millliseconds and the remaining boards time slots of 100 microseconds. Consider the following example for the implications of such timings. Buffering requirements for the time intervals of 1 microsecond could be as large as 1600 bytes (e.g., for a UTRAN CDMA signal at four 8-bit samples per chip). This recommends the division of data into multiple packets successively transferred when

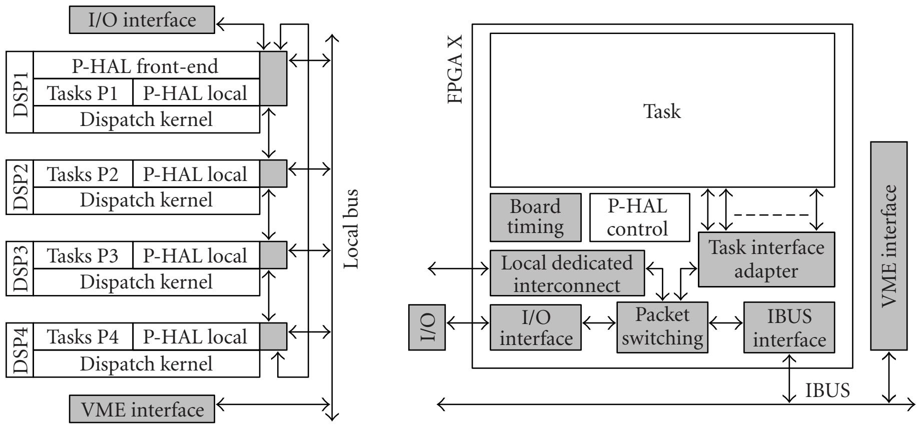

Figure 5: DSP board P-HAL software and single FPGA P-HAL detail.

the interface runs on an FPGA, especially if packets have to be stored into internal FPGA memory. On the other hand, the workstation, with objects processing a bit stream of 2 Mbps, requires a buffering of about 6250 bytes, only a factor of four for a platform where memory is easily obtained without suffering an important performance penalisation.

Actually, the initial function of the PC machine was to serve the disk to the workstation. The inclusion of network support to P-HAL extends its usability to distribute computing across networks of general-purpose machines. Applications in the general-purpose computing sector have many other well-known software mechanisms available to distribute computing, like RPC or CORBA. However, the simplicity of concepts and functions behind P-HAL, fruit of its orientation to tough tasks, makes of it a candidate for safe and quick developments [13].

During the development of P-HAL, several factors arouse that have not been initially considered. These include, for instance, data endianness. These observations are proving that the abstraction of a platform has many steps to follow.

5.2. General implementation aspects

Implementing all the tasks associated with P-HAL abstraction layer over Linux or Solaris machines is relatively easy because of the large amount of available software libraries and the OS itself. Moreover, there is no problem to control the interfaces at hardware level, as all necessary drivers exist. Certainly, the simplicity that allows the quick development of software has the price of reducing the computing efficiency. For the DSP and FPGA boards, such performance loss is unacceptable.

In the case of the DSP board, an approach with a standard OS (Linux, as open source, could be adapted) can be perfectly assumed. Despite the particularities of the processor that make it adequate for signal processing tasks, there are no large conceptual differences between DSPs and GPPs. Therefore, a simple kernel (able to switch between different processes) runs on each of the four DSPs. In addition, a part of P-HAL runs on each processor, whilst the board P-HAL frontend (input/output gate, synchronism, software loading) only runs on one DSP, the one that is interfacing external platforms. The tasks are based on a continuous packet generate and/or dispatch loop that manages control data as well as processing data. Figure 5 illustrates a diagram of the structure of P-HAL software within the DSP board and the hardware interfaces it deals with.

When implementing P-HAL on FPGAs, transferring all the previous concepts becomes very costly in terms of area utilisation (now the main resource considered is silicon area or, equivalently, logic gates, instead of CPU time). The conception of applications on FPGA is quite different from the conception in sequential processors [14]. P-HAL must employ complex state machines able to process packets in and out in the same way as, for instance, the DSP board does. Another relevant issue here is the possibility to have multiple processes running on an FPGA and swap them in and out, as it is done on common computing platforms. In this case, to simplify the control mechanisms, just one application process is allowed per FPGA, apart from the P-HAL process. Since there are eight available FPGAs independently configurable, each board may run up to eight processes. In Figure 5, a schematic representation of the generic interface offered by P-HAL to the application process appears. The usage of logic within FPGAs depends mostly on the buffer size for each interface. While controlling state machines for common part of P-HAL use less than 40 CLB [15] (twice as much if frontend is included), each additional stored byte for buffer FIFOs requires half a CLB (in Virtex devices, RAM may be implemented without consuming CLBs). To save CLBs in the current implementation, short packets are transmitted over FPGA interfaces (64 bytes maximum) that have an acceptable impact on interface efficiency. All these issues (and many more) are important when providing the figures of the hardware model.

The blocks represented in Figure 5 have different meanings for the DSP and FPGA cases. In regards to a DSP platform, blocks represent processes stored in memory that alternatively get some CPU time. Conversely, regarding FPGA platforms, blocks represent concurrent processes that occupy different areas within the FPGA logic.

5.3. P-HAL performance and overhead

While the application moves towards an abstract context being unbound from the underlying hardware, P-HAL is tied to each particular hardware platform. On the one hand, the performance depends on the adaptation facility of the P-HAL rules to the underlying hardware. A hardware that is oriented to support mechanisms that are required by P-HAL will provide a lower overhead than a hardware that is completely orthogonal to the P-HAL requirements. On the other hand, performance also depends on the solution that the programmer has considered for the P-HAL software. As a result, the figures appearing in the next sections are valid for the above-stated hardware and the particular way of implementing P-HAL on it. However, they give a good idea of what one can expect when using an abstraction layer like P-HAL for the focused applications.

In Linux or Solaris platforms, overhead is not the major concern since higher communications layers are expected to be running there (from the network, OSI model layer 3, up to the application layer, OSI layer 7). Rather, the major concern is maintaining the time granularity that the application under P-HAL requires. This also depends on system load because P-HAL uses the underlying kernel services for its purposes. Also because scheduling mechanisms in Linux or Solaris kernel are basically oriented to provide a fair CPU assignment, P-HAL only guarantees that objects are executed within time slot bounds if no other processes consume a lot of CPU time (even momentarily). In order to keep a good time resolution, the kernel has suffered modifications to reduce scheduling latency (10 milliseconds by default) to the order of 0.5–1 millisecond (depending on the machine performance). Lower scheduling latency introduces overhead in normal operation because of the increased amount of possible scheduling actions per time unit. However, when P-HAL controls most CPU consuming processes, the amount of process swaps reduces to one per time slot and per process, which reduces the overhead to be almost imperceptible.

The performance of using an approach like P-HAL can be measured in terms of the additional resources that are required to run the abstraction layer against the benefits that it provides. The benefits of the scheme have been described in the previous sections while in the following ones, its cost is being quantified. The measurement is made by computing either the percentage of time or the percentage of area that the executive components of P-HAL use all along the activity of the application. Such percentages are assigned the qualifier of overhead.

5.3.1. Overhead in the DSP platform

The produced overhead running an application on P-HAL or running an ad hoc application is represented in Figure 6, where it appears as a temporal distribution of the parts of the software that run on one DSP. Objects A and B share the processor and P-HAL is in charge of providing access to resources (through a kernel service) and scheduling the execution of objects. In the figure, tasks located at the *process swap* and *kernel plane* levels are actually tasks that are assigned

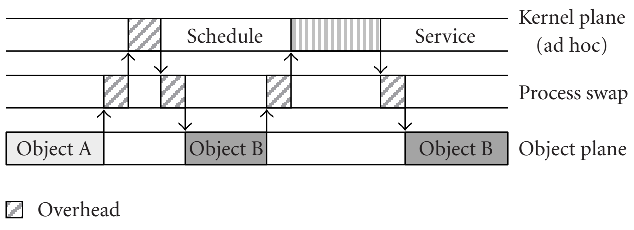

Figure 6: Overhead sources in DSP.

to P-HAL, while tasks located at the *object plane* are the actual execution of objects. Then, overhead only appears when the processor runs tasks located at the two highest levels in the figure. To measure the actual overhead, it is necessary to make a more in-depth analysis of those tasks located at the highest levels.

The first form of overhead is the time required to access the P-HAL kernel services where specialised ad hoc functions perform the requested actions (e.g., DMA transfer from one memory location to another) by using the underlying hardware. On the used DSP board, accessing a kernel service takes about 100 CPU cycles (save the process context and restore the kernel to provide the service). Since the real clock frequency is of 100 MHz, this represents 1 microsecond. In a system with a time slot of 1 millisecond (100000 cycles), accessing a single service uses 0.1% of the available time. This percentage is relative to the performance of the processor since in a 200 MHz processor, the overhead would be 0.05%. To some extent, it is also relative to the solution taken by P-HAL programmer (better or worse utilisation of hardware).

Following the P-HAL approach, each object running on P-HAL has to request at least four services per time slot (according to Figure 2). The minimum overhead per object is around 0.4% of the available time. The algorithm itself and ad hoc service functions are necessary in any case, either using P-HAL or not, then they cannot be considered overhead.

A second source of overhead is scheduling different objects that run simultaneously on the same processor. An ad hoc implementation would simply run one function after the other (considering one function per object) but here more elaborated procedures are required. Since the timing structure is simple enough to be just useful for a typical radio application, scheduling consists of a sequential assignment of CPU to the different object processes. In Figure 6, overhead during a scheduling (from object A to object B) consists of an initial process swap to reach the kernel process. The kernel process performs the schedule, which requires some time, and finally another process swap is performed to move the execution point to object B. On the available platform, scheduling takes around 2 microseconds (measured from signals captured with a logic analysis system, Figure 7) including also the context swap. Taking into account the software architecture of objects running on P-HAL and the minimum amount of services that an object requires, 1 microsecond of additional time overhead per object and time slot has to be considered.

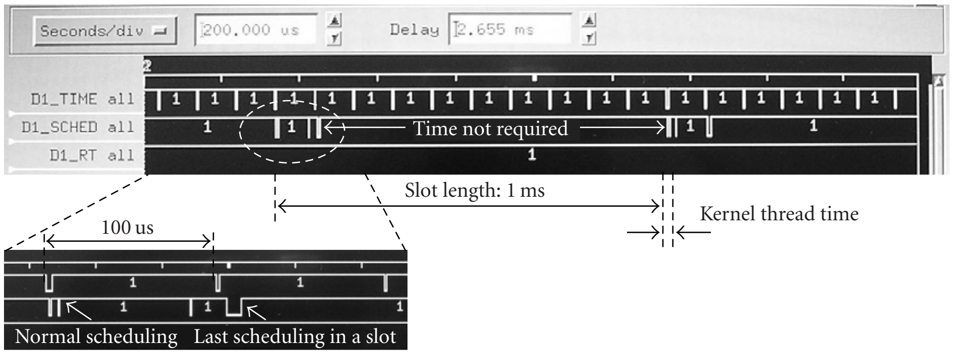

Figure 7: Timing diagram for DSP scheduling.

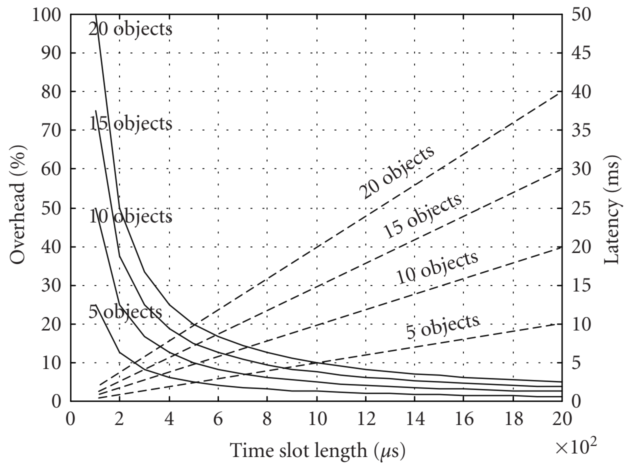

Figure 8: Overhead and latency variation as a function of objects and time slots.

Hence, the minimum accumulated overhead rises to 0.5%, 0.4% for services (4 microseconds), and 0.1% for scheduling (1 microsecond), of available time per object under the previous slot conditions (1 millisecond). If more services are requested, overhead will increase. Figure 7 gives an idea of the little amount of time consumed by P-HAL processes within a time slot, leaving a very high percentage of resources for the radio algorithm (spot area). The label "time not required" in the figure just shows that after all the objects have finished their processing, the processor may remain stopped until the next time slot. When the slot length changes or the number of objects increases, the total overhead follows the curves in Figure 8. As it is expected, increasing the slot length leads to longer execution time by application objects and lower execution time by P-HAL. Conversely, longer time slots increase latency because of execution pipeline, which may have an important impact on the application.

Another kind of overhead to consider is the overhead which is produced by the control mechanisms within P-HAL. However, these control mechanisms are present in any application, either ad hoc or platform-independent. For this reason, although in the figures they may appear under the overhead label, they are not actually considered as overhead. Real-time statistics and time control can be the sources of major resources consuming.

In the first case, since any management entity could request many parameters from objects (but only those provided by them), the overhead has been limited to 10% of available time. In addition, in Figure 7, the time consumed by a P-HAL kernel thread dealing with control mechanisms is shown (at the beginning of the second time slot). In a 1-millisecond time slot, at most, 100 microseconds are for that execution thread. This, of course, can change from implementation to implementation.

In the second case, time control employs CPU resources periodically. The real-time clock (RTC) control mechanism also appears in Figure 7 (D1_TIME label), which is executed once every 100 microseconds partly because this is the minimum resolution time assigned to the kernel to swap processes (shorter periods are achieved if processes give up CPU when they are done or waiting for a service). Thus, an additional 0.1% of a time slot is reserved for maintaining the RTC. A mechanism not shown in Figure 7 is the synchronism procedure between platforms. However, this procedure just performs a couple of VME bus accesses in intervals of one second, which actually requires a very short time.

Overhead in terms of increased memory utilisation is not included since it is negligible in terms of amount and cost.

5.3.2. *Overhead in FPGA platform*

The overhead measurement on the FPGA platform has the same limitations than on the DSP platform: they are for an individual implementation on a particular platform. Moreover, there are two different approaches when measuring the overhead. One is considering any buffering included within an FPGA. This would be the case where interfaces do not have any FIFO memory to store the emitted packets. When all the implemented memory is within the FPGA, overhead rises tremendously (over 100% of logic gates [16]) compared with the ad hoc implementation. This bad initial figure appears when FPGAs are relatively small (with 50 k–150 k gates, especially if compared with current multimillion gate FPGAs) and when plenty of interfaces are available per FPGA: two daisy-chain interfaces, one bus interface, and one memory interface (see Figure 5). A less generic implementation

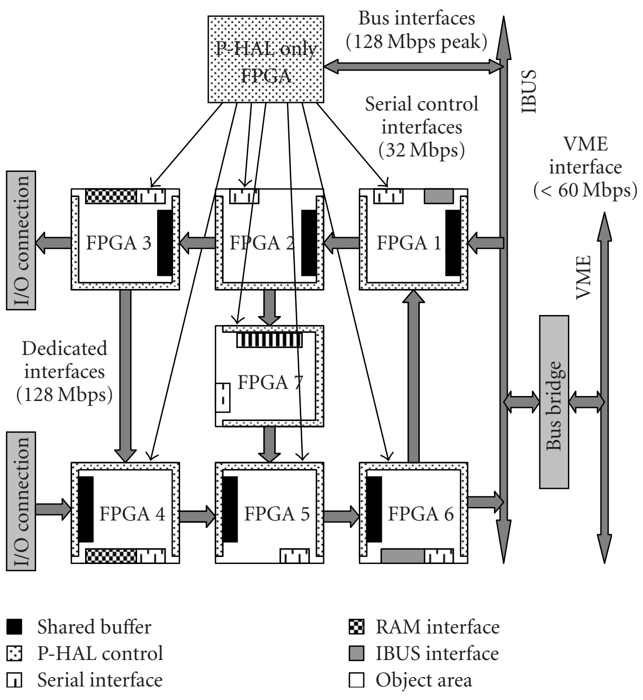

Figure 9: Adjusted P-HAL for SHaRe platform.

of P-HAL is necessary to reduce the amount of capabilities available per FPGA. As a result, overhead figures are more acceptable. Effectively, this does not limit P-HAL attributes to FPGA implementations but rather reduces the amount of hardware resources taken into account.

The board used for initial trials and tests of P-HAL contains XC4013 FPGAs (13 k gates only). Saving resources is therefore crucial. With eight FPGAs available, the total gate count is of about 104 k gates. In Figure 9, a detailed view of the minimum P-HAL architecture on SHaRe board appears. The situation depicted in Figure 5 is preferable from the flexibility point of view and the ease of the mapping procedure. The limited amount of resources on this board does not allow loading communication modules of complete real systems. This is not a concern of P-HAL; it is possible to find similar problems when constructing an ad hoc application.

The overhead measurement accounts for required logic resources for two unidirectional dedicated interfaces, timing control, and configuration registers. Some cases also include bus (IBUS, a packet-oriented interface designed to provide a good support to P-HAL mechanisms, as appears in Figure 5 or Figure 9) and RAM access logic. Interfaces only include a (shared) buffer at the receiver side. Configuration registers are mainly devoted to routing purposes. Under such conditions, the percentage of resources occupied by P-HAL is of about 25% (140 CLB, about 3.5 k to 7.5 k gates per FPGA). The FPGA that is only devoted to P-HAL is not taken into account in this figure. However, note that the same approach with, for instance, Virtex 1000 FPGAs (1 M gates) would result in a utilisation of less than 0.5% with the same offered P-HAL functionalities.

5.3.3. *Performance summary*

As mentioned above, performance can be measured as a ratio between extra resources utilisation and benefits. Table 1 summarises both, the approximated cost of using a structure like P-HAL on a state-of-the art device (based on the aforementioned implementation), and the advantages it carries. The conclusion is that processing at lower radio layers can be organised based on an abstraction layer that enforces software portability and interoperability at the same time that isolates the application from the underlying hardware. Even though, it is true that the approach requires additional resources (cost of nontuned software/hardware). It is also true that the complexity of modern communications standards requires an increasing independence of algorithms from hardware to speed up development and reduce time-to-market.

Following the previous line, obtaining good results (e.g., power efficiency) depends on the capacity of the development tools to adapt a hardware-independent program (algorithm) to a specific processor. The human resources that many times at present tackle the intensive task of optimising chip utilisation (either programmable or ASIC) will tend to reduce increasingly when tools achieve enough efficiency levels. Finally, when the application target is to use programmable devices, the increase in power consumption of the presented approach can be roughly estimated as the percentage shown in the table.

5.4. *Radio applications on P-HAL*

To validate P-HAL as a candidate framework for software radios on heterogeneous platforms, a set of objects has been adapted to P-HAL rules. However, because of the limited amount of computational resources, simplified schemes have been adopted. Despite this limitation, the essence of the applications remains unaltered. Therefore, trial applications are good enough to detect limitations and make improvements to the presented architecture. Note that the objective here is not to optimise transceiver algorithms, but provide a good framework for both, development and execution of radio applications.

Application objects are loaded from the Solaris workstation to any other platform. The role of this workstation is to interface with the external management entity, whatever it may be. Objects are loaded through P-HAL configuration mechanisms. The time it takes to download the program of an object to a processor depends on hardware and may range from the about 60 milliseconds to configure an XC4013 FPGA through an 8 Mbps interface (its configuration bus) to about 2 milliseconds to load an executable file to one DSP.

Note that P-HAL not only performs configuration tasks, but it runs in parallel to the application providing the set of functionalities described before. Then, the radio application actually runs thanks to the presence of P-HAL, otherwise it would be useless because the objects are programmed independently from hardware. As shown in Figures 10 and 11, the arrows interfacing modules are actually pieces of software that run on the different processors involved in the data exchange procedure.

Table 1: Performance comparison and tools requirements.

Design type	Resources (DSP)	Resources (FPGA)	Benefits	Human effort	Tools effort
On abstraction layer	∼ 0.5%–1%	∼ 0.5%–1%	Flexibility (+), acceptable power efficiency	Relatively low	High
Custom	0%	0%	Power efficiency, low flexibility	High	Medium

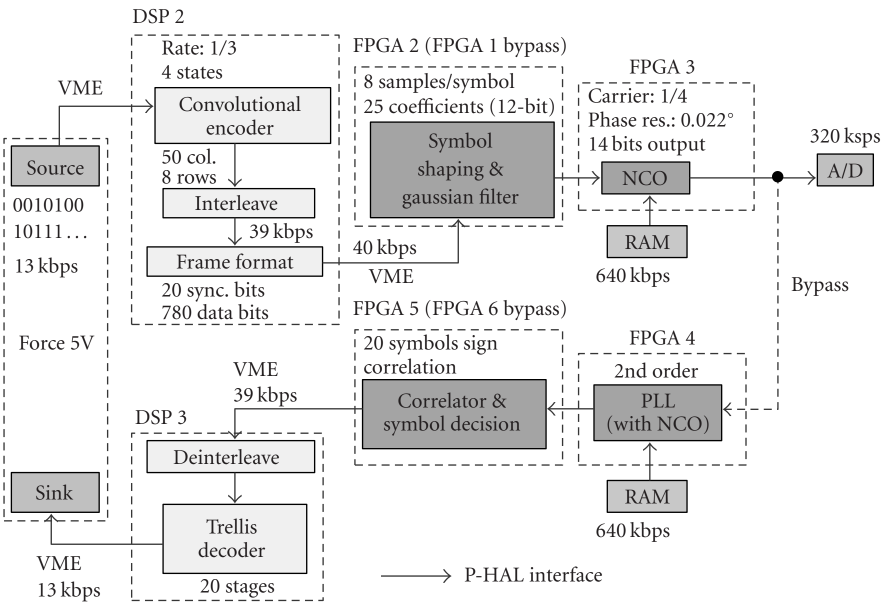

Figure 10: GMSK transceiver.

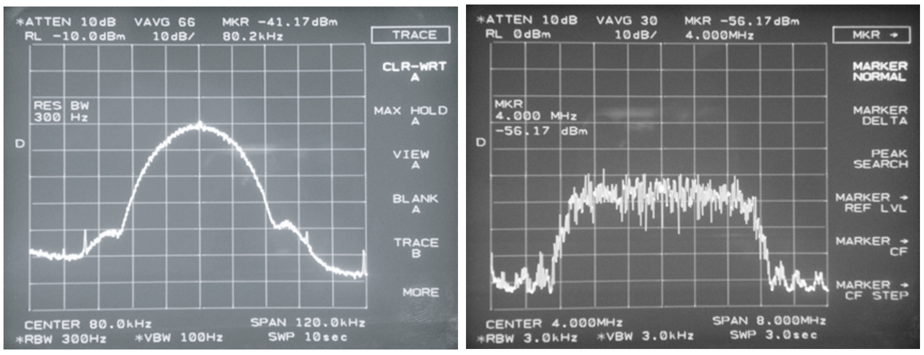

Figure 11: GMSK and QPSK CDMA generated spectrum.

5.4.1. *Development flow*

In a first stage, the mapping of objects is done manually for best results. The testing of mapping algorithms is outside the scope of this document, although it is a cornerstone in the process from the abstract application down to hardware.

Objects are programmed in either C or VHDL. Ideally, programming the object once would be enough but development tools for GPP/DSP and FPGA are quite different. For each object, a binary executable file is obtained. Such an executable file is processor-specific. In general, an executable file would be necessary for each possible processor. However, in this case, the target processor is known, and then only one binary file is necessary. Figure 12 represents the development flow diagram for the two kinds of platforms.

5.4.2. *Trial no. 1: GMSK transceiver*

First testing application is a GMSK transceiver as shown in Figure 10. The transmitter consists of a convolutional encoder, an interleaver, a frame formatter (for synchronisation purposes), a sampling rate adjustment module, a Gaussian

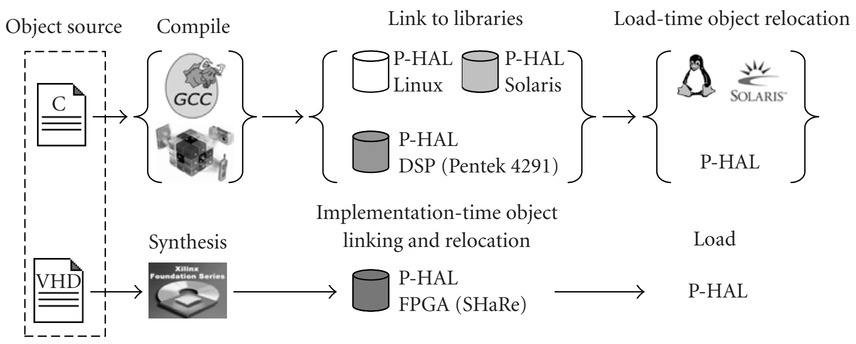

Figure 12: Development flow.

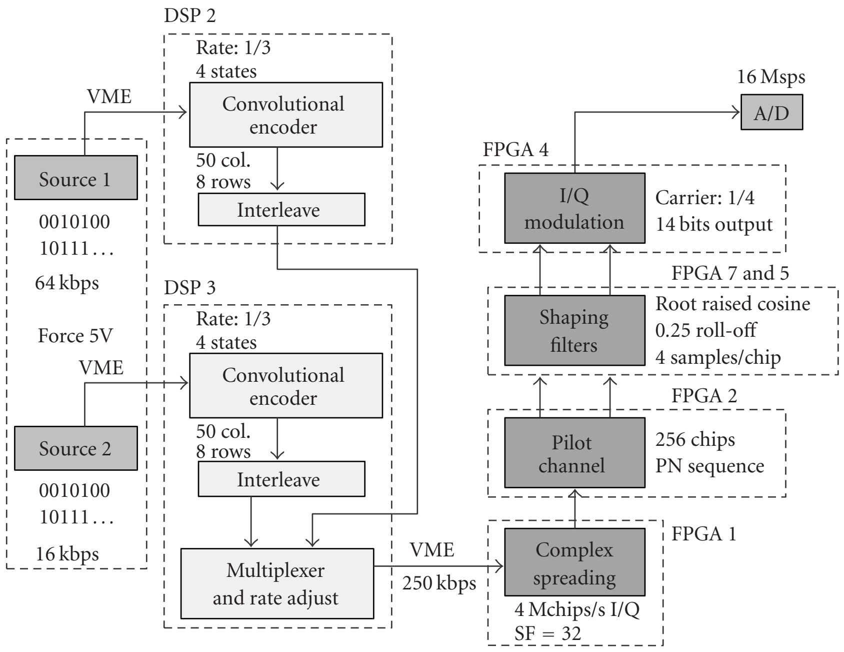

Figure 13: QPSK CDMA transmitter.

filter, and a numerically controlled oscillator (NCO). The receiver consists of a phase-locked loop (PLL), a symbol synchroniser, a symbol decision module, a trellis decoder, and a deinterleaver. A random number generator on the workstation provides random bits to be transmitted. A dummy object gathers all received bits. In total, there are eleven different objects running on seven different processors. The spectrum of the generated modulation is shown in Figure 11.

The application is divided into objects that are mapped onto the processing devices as illustrated in Figure 10. Data speed is rather low so that it is possible to transmit everything on the VME bus. The required aggregate net transfer speed on VME bus is approximately 13 kBps only. To simplify the timing management, the time slot is 10 milliseconds for all platforms. Time reference of P-HAL is originated at the A/D platform.

This modest application serves as validation of the correct P-HAL behaviour on all the aforementioned platforms.

5.4.3. *Trial no. 2: QPSK CDMA transceiver*

The CDMA transmitter comprises another modulation (Figure 13). Bit processing is similar as in the former trial at the same time as data transfer speed and sampling frequency have increased. On the FPGA side, there are not enough resources to construct transmitter and receiver. This problem is simply solved by adding an additional board and including P-HAL on it.

5.4.4. *Comments about implementations*

The applications do not have loops involving more than one object. This is to avoid mixing the signal time with the execution context time. Signal samples suffer from the time slot division delay, and then in case of a loop, this would affect the processing algorithm. This effect imposes different design rules to the radio application because fast loops in a virtual execution context are not realistic. For instance, consider a frequency error detection algorithm working in the baseband. It may modify the local oscillator frequency between RF and IF only if the delay of one time slot, as imposed by P-HAL, is not critical.

At the management level, P-HAL performs the swapping of the FPGA configurations—from GMSK to QPSK CDMA—upon a request from any external management tool. The procedure starts loading a different mapping file. From this file, P-HAL loads new executable binaries to the FPGAs. A different linkage is necessary for every FPGA model, due to the nonunified connectivity architectures (at physical level, see Figure 9). Equivalent is to say that each FPGA processor is different. Nevertheless, in GPP or DSP, there is a similar situation. At load time, the OS makes a relocation of executable code to adapt it to the context. In FPGA, this procedure is more costly and it is preferable to perform it at compile/link time.

6. CONCLUSIONS AND FUTURE WORK

The current work has been focused on the design and development of a platform-hardware abstraction layer (P-HAL). It provides a platform-independent support to the application processes, especially those related to software radio concept but not limited to them. The most relevant features include transparent data interfacing, real-time control, process scheduling, and task monitoring. The key aspect of such approach is to deal with a set of different hardware/software systems, DSP devices, FPGA arrays, and GPP running standard OS. Despite the heterogeneity, it has the capacity to offer a single virtual platform.

The environment defines a common structure for the different application processes, a global timing framework, and a uniform, plain, interface format. The timing mechanism uses a division of time into slots of variable length depending on the capabilities of processors and the associated support software. On the other hand, the proposed interface definition allows detaching the different pieces of an application, thus supplying a seamless integration of independently developed software. All the previously described features are provided at a minimum overhead to avoid the unnecessary waste of computational resources, a proof that it is possible to employ such approach in realistic contexts from now on.

The tests of P-HAL continue with other more complex applications and platforms. They will allow us to completely profile the presented architecture, maybe adding new features or even changing some rules. Further enhancements are expected for the P-HAL in order to become a mature tool to easily develop software radio applications.

ACKNOWLEDGMENT

This work has been supported by CYCIT Spanish National Science Council (CYCIT) under Grant TIC2003-08609, which is partially financed from the European Community through the FEDER Program.

REFERENCES

[1] J. Mitola, "The software radio architecture," *IEEE Commun. Mag.*, vol. 33, no. 5, pp. 26–38, 1995.
[2] W. H. W. Tuttlebee, "Software-defined radio: facets of a developing technology," *IEEE Pers. Commun.*, vol. 6, no. 2, pp. 38–44, 1999.
[3] S. Srikanteswara, J. H. Reed, P. Athanas, and R. Boyle, "A soft radio architecture for reconfigurable platforms," *IEEE Commun. Mag.*, vol. 38, no. 2, pp. 140–147, 2000.
[4] A. Munro, "Mobile middleware for the reconfigurable software radio," *IEEE Commun. Mag.*, vol. 38, no. 8, pp. 152–161, 2000.
[5] S. Gultchev, K. Moessner, and R. Tafazolli, "Controlling reconfiguration," in *Proc. 3rd IEE International Conference on 3G Mobile Communication Technologies*, pp. 474–478, London, UK, May 2002.
[6] M. Cummings and S. Heath, "Mode switching and software download for software defined radio: the SDR forum approach," *IEEE Commun. Mag.*, vol. 37, no. 8, pp. 104–106, 1999.
[7] A. A. Gray, C. Lee, P. Arabshahi, and J. Srinivasan, "Object-oriented reconfigurable processing for wireless networks," in *Proc. IEEE International Conference on Communications (ICC '02)*, vol. 1, pp. 497–501, New York, NY, USA, 2002.
[8] J. Bertrand, J. W. Cruz, B. Majkrzak, and T. Rossano, "CORBA delays in a software-defined radio," *IEEE Commun. Mag.*, vol. 40, no. 2, pp. 152–155, 2002.
[9] V. Chaudhary and J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 3, pp. 328–346, 1993.
[10] http://www.forcecomputers.com.
[11] http://www.pentek.com.
[12] X. Revés, A. Gelonch, F. Casadevall, and J. L. García, "Software radio reconfigurable hardware system (SHaRe)," in *Proc. 10th International Workshop Field-Programmable Logic and Applications (FPL '00)*, vol. 1896 of *Lecture Notes in Computer Science*, pp. 332–341, Villach, Austria, August 2000.
[13] R. Ferrús, X. Revés, A. Umbert, and F. Casadevall, "Real-time emulation of RRM strategies for UMTS bearer services," in *Proc. 56th IEEE Vehicular Technology Conference (VTC '02)*, vol. 2, pp. 955–959, Vancouver, Canada, September 2002.
[14] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proc. IEEE*, vol. 86, no. 4, pp. 615–638, 1998.
[15] Xilinx, Inc., *The Programmable Logic Data Book*, 2004.
[16] X. Revés, V. Marojevic, A. Gelonch, and R. Ferrús, "The cost of an abstraction layer on FPGA devices for software radio applications," in *Proc. 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '04)*, vol. 3, pp. 1942–1946, Barcelona, Spain, September 2004.

Xavier Revés was born in Sarroca de Lleida (Spain) in 1974. He received the Engineer of Telecommunication degree from the Universitat Politècnica de Catalunya (UPC), Spain, in 1998. He is currently applying for the Ph.D. degree in the Department of Signal Theory and Communications at the UPC. In 2001, he joined UPC as an Assistant Professor, which is his current status. His research is focused on radio communications, with special emphasis on digital processing platforms including hardware and software architectures, software radio technologies, and analogue/digital transceiver front-end architectures. Since 1999, he has been participating in several projects of technological transfer to industry, including the European FUSE Program. He has also been and is currently involved in European Framework Program projects like ARROWS and EVEREST.

Antoni Gelonch received the Engineer of Telecommunication and Dr. Eng. degrees from the Universitat Politècnica de Catalunya (UPC), Spain, in 1991 and 1997, respectively. In 1992, he joined UPC, where he became an Associate Professor in 1997. His main activities after graduation were focused on the field of digital communications with particular emphasis on digital radio. From 1992, he has been involved mainly in the analysis and development of digital mobile radio systems. His research interests include cellular and personal communication system, multipath transceiver design, and software radio techniques including their relation with the reconfigurable computing, mobility, and radio resources management. In the last ten years, he participated in several research projects founded by both public and private organizations. In particular, the most important projects in which he has participated have been the CODIT Project of the RACEII Program and the RAINBOW Project in ACTS Program. In the context of the 5th European Framework Program, he has participated in the WINEGLAS IST Project. At present, under the 6th European Program, he is involved in the EVEREST, which is related to the end-to-end QoS in wireless heterogeneous networks and in the E2R, where the reconfiguration concept (software radio) extends to the complete wireless network.

Vuk Marojevic received the Dipl.-Ing. degree in electrical engineering from the University of Hannover, Germany, in 2003. During his graduate study, he spent three terms at the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, where he is currently working towards a Ph.D. degree in communications. Since 2004, he has been supported by a grant from the Catalan Government. His research interests are in the field of software-defined radio architectures and algorithms as well as operating system support for reconfigurable computing devices.

Ramon Ferrús was born in Batea (Tarragona), Spain, on March 16, 1971. He received the Engineer of Telecommunications degree from the Universitat Politècnica de Catalunya, Spain, in 1996, and the Ph.D. degree from the same university in 2000. In 1996, he joined UPC, where he was an Assistant Professor from 1996 to 2002. He is currently an Associate Professor in the Signal Theory and Communications Department. He currently lectures on mobile communications, data transmission systems, and communication theory. Since 1996, his research field has been mobile radio communications. His doctoral thesis contributed to the study of the radio access and handover mechanisms in cellular systems. Currently, his research interest is focused on QoS, mobility, and radio resource management in the context of heterogeneous IP-based mobile communications systems. He has actively participated in several research projects in the frame of European Programs ACTS (RAINBOW Project) and IST (WINE GLASS, ARROWS, and EVEREST Projects), where his main contributions have been focused on the development of real-time hardware/software platforms for the emulation of different wireless access networks.