RUN-TIME AND DEVELOPMENT FRAMEWORK FOR HETEROGENEOUS HARDWARE RADIOS

Xavier Revés, Vuk Marojevic, Antoni Gelonch

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

Abstract

Software Radio concept provides increased features to radio access networks. The reconfiguration of radio equipment requires the existence of an architecture, a common framework, that allows a flexible management, in any sense, of software running on radio processors. Such framework must take into account the heterogeneity of hardware devices and platforms where the radio applications will run. To avoid excessive overhead, a conceptually simple but at the same time useful structure has been developed to deploy radio applications, from physical layer to application layer, on heterogeneous software and hardware environments.

1. INTRODUCTION

Some years ago the Software Radio (SR) concept came to scene promising a revolutionary change in radio infrastructure. The concept, initially focused on physical radio layer, has spanned to any layer within the protocol stack of radio terminals and to network management. But the advantages associated to SR will not be achieved if it does not exist a common framework and a formal design methodology to fully expand them.

Even though the coverage area of SR has dramatically increased, the core concept still remains not completely solved. The core software is basically that dealing with harder real-time tasks which is not easy to separate from underlying hardware (interrupts, memory addresses, processor instructions, etc.), thus creating a dependency of software on hardware. This dependency may limit the actual flexibility of radio terminals to accommodate different configurations as requested by the access network. Removing such dependency is generally achieved through abstraction layers (e.g. HAL, *Hardware Abstraction Layer*), a well-known concept in computing world. Then, following this idea, radio capabilities empowered through use of software should be separated from digital hardware processors available on a given platform, including general purpose processors (GPP), digital signal processors (DSP), field programmable gate arrays (FPGA) and others. Whichever is the abstraction layer finally used, it must provide a framework that, at least, includes mechanisms to uniquely specify which software must run on the terminal and to ensure a safe behaviour of such software, keeping radio terminal within standard restrictions and respecting the user-operator contract bounds.

In this document one possible abstraction layer architecture being currently developed, P-HAL (*Platform-Hardware Abstraction Layer*), is described. It has been found useful to implement radio algorithms and covers raw devices as well as more complex platforms. It goes beyond the simple abstraction mechanism offering also an execution environment and run-time control of the radio application.

2. THE ABSTRACTION LAYER

2.1 General Overview

The basic concept under P-HAL is providing a location and time context to create software pieces completely independent of the platform where they have to be executed. Here the platform term includes a high-level system based on GPP and an operating system (OS) as well as raw hardware (boards with DSP or FPGA devices). Because of currently available tools, algorithms within the presented environment are programmed using standard programming languages, like C/C++ and VHDL. Note that even there exist paradigmatic programming differences between GPP or DSP and FPGA processors, the aim is treating them the same way, not necessarily making the software/hardware division found in heterogeneous systems. It is expected that future development tools allow creating a binary executable file for any of the previous processors from a single source code.

A key aspect of P-HAL is that it provides the means to add as much processing resources (hardware) as required for a given application, not having to modify the software of the application at all. The addition (or removal) of plug-and-play hardware on a system is closely related to the possibility of building-up the radio terminal with different hardware from different providers. The different hardware topologies, configurations and, above all, the way as tasks are assigned impose restrictions to the way that different hardware is integrated to construct SR platforms. Figure 1 shows a representation of context where P-HAL works providing the following set of features:

- Real-time seamless exchange of information from one P-HAL compliant platform to another (BRIDGE).
- Isochronism of data and processes running on different platforms (SYNC).

- Platform-wide coordinated process control, scheduling, logging and error control (KERNEL).
- Real-time system monitoring, data and statistics retrieval and adaptation of processes set-up parameters (STATS).

All the previous features (and other not included for simplicity) are accessible to the application through P-HAL Application Programming Interface (API).

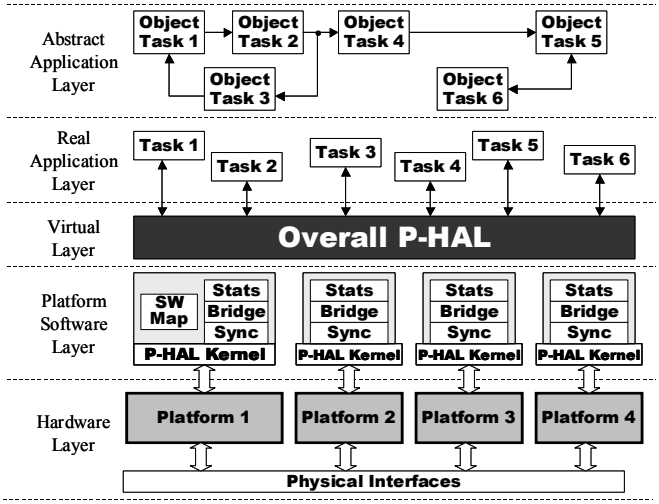

Fig. 1. P-HAL schematic representation

The three first features in the list produce the effect, from the application view side, of having a single, unified, platform. Then, the same application description works on a single machine or on multiple distributed machines, provided that a P-HAL layer exists in between the application and the hardware. It does not matter what hardware runs an specific part of the application, this part will just observe other parts as being aside. In the figure four processing machines (platforms) have their respective P-HAL running and only an executive virtual P-HAL entity is shown to application distributed blocks. In any case the application is not aware of the underlying hardware.

2.2 Other Considerations

In principle platforms only have some minor information about other platforms to work together. One or more (direct or indirect) shared interfaces are the only liaison between each pair of platforms (e.g. network, bus, etc. interface). Through such interfaces P-HAL compliant software on each platform builds-up a virtual larger platform. Of course the performance of this interface may limit the aggregate computational capacity of the overall platform but in general not more than internal platform interfaces may do.

Since some parts of the application have to deal with real-time issues two important timing considerations have been added to P-HAL. First one, isochronisms of platforms, requires that the existent interfaces between them are used to create a local time reference adjusted with the local reference of one platform. A synchronism procedure is required overcoming those delays present in the aforementioned interfaces. To limit the effect of some synchronism misalignments and to create a temporal execution framework for the application, time is divided into slots. Such time slots are large enough to allow each P-HAL to schedule all the application tasks. At the same time, slots are short enough to limit the amount of buffering in the interfaces.

By other hand, the application should not be aware of the actual time slot length. The SR application, which is organised as a set of software blocks or objects (see figure 1), is mapped to the different processors available on the platform that are hidden under P-HAL. Each software block or task is programmed independently of the processor that is going to execute it. Then, the only things that blocks may see are P-HAL functionalities accessed through its API. Strictly speaking, application software blocks are simple data processors. Data at their outputs are provided to the blocks where they are connected to. The receiver is not even known at programming time. When developing the application different software objects are plugged creating a processing stream. Every time slot a block receives data and has to process it within such time slot. The correct real-time behaviour is achieved if each software block is able to process before the end of the slot all data available at the beginning of the slot. Moreover, P-HAL has to guarantee that each block receives at each time slot any data sent to it in the previous one.

2.3 Current Implementation

P-HAL is currently under final steps of development for four different platforms of a SR heterogeneous platform. One Linux PC (completed), one VME bus diskless Sparc workstation (completed), one VME bus quad DSP board (to complete kernel) and one VME bus octal FPGA board. Each individual platform requires different skills to offer a P-HAL compliant interface. Just as an example of the different approaches followed in the latest two platforms, figure 2 shows the organisation of P-HAL on them. For each DSP, tasks are distributed along time (*dispatching kernel*) while, for each FPGA, tasks are distributed along space.

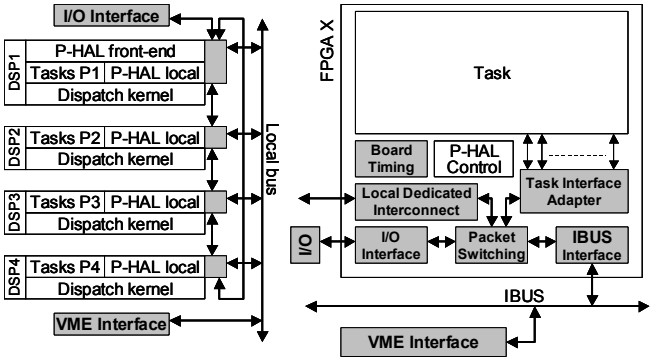

Fig. 2. P-HAL implementation case on DSP and FPGA

AKNOWLEDGEMENT

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