P-HAL: A MIDDELWARE FOR SDR APPLICATIONS

Antoni Gelonch, Xavier Revés, Vuk Marojevik, Ramon Ferrús (Dept. of Signal Theory and Communications, Universitat Politècnica de Catalunya, Barcelona, Spain; {antoni,xavier.reves,marojevic,ferrus}@tsc.upc.edu);

ABSTRACT

Software Defined Radio (SDR) is an emerging technology with the objective of implementing the processing tasks required in a radio transceiver in software rather than in dedicated hardware. As a result, programmable devices such as Instruction Set Processors (ISPs) and reconfigurable logic devices, e.g. Field Programmable Gate Arrays (FPGAs), may be mixed for building a Reconfigurable Heterogeneous Hardware Platform. The peculiarities of communication systems and the fact that they must be implemented under SDR environments, where the main objective is to deal with the reconfiguration process, introduce a specific problematic in the development and management of such applications. In that context the presented work addresses the development of a suitable Middleware, defined as P-HAL (Platform & Hardware Abstraction Layer) that tries to advance in the process of defining a common framework to develop and deploy software radio applications by eliminating platform (hardware and support software) dependencies.

1. INTRODUCTION

Even though the evolution followed by the Software Radio [1] concept from its beginning it can not be considered a mature technology. There is still some interesting research areas to be explored and, what is more important, it is still opening new facets and views on its possible evolution mixing technologies from different areas. In addition, the increase in flexibility requirements is claimed for all the agents involved in its development due to the fact that the concept, initially focused on physical radio layer, has been span from the radio terminals (base stations or mobile equipment) to network management, including resource management for optimum service provisioning.

As its initial basis is the implementation in software of the processing tasks required in a radio receiver, its development relays in concepts related with the computer technology. We should consider, therefore, the need to reasonably split the radio transceiver in different tasks or processes and identify the most suitable processor where to execute them. Among the currently available we can identify General-Purpose Processors (GPP), Digital Signal Processors (DSP), FPGA, processor's arrays, among others. In addition, no one processor is capable, working standalone, of providing the required computational power of a SDR terminal and even more it becomes unfeasible for a Base Station. Therefore, this introduces the heterogeneous computing platforms into the SDR arena where an array of different processor should be capable to provide the required computational demand. Moreover, this scenario becomes reinforced by the high quantity of different processors and the fact that each hardware manufacture will select the most appropriate set of them according to its hardware topology, its development tools and, of course, its business model.

We should not forget that the radio applications impose some important constraints to the execution process where the available hardware must deal with harder real-time tasks. These constraints create important difficulties to the separation from underlying hardware (interrupts, memory addresses, processors instructions, etc), creating a strong dependency of the software from hardware. In addition such dependency limits the capacity of the radio equipment to accommodate different configurations as requested by the access networks.

Under such scenario it is interesting to consider one of the most important benefits of the SDR concepts, which is the capacity to assure the software portability and reusability as the basis of the reconfiguration concept. Therefore, some of the strongest effort must be done in the development of a common framework (HAL, middleware, execution environment, development tools) [2] capable of hiding the hardware characteristics to the radio application and removing the dependencies between software and hardware. Moreover, it is necessary that such framework can provide the necessary management functionalities to assure the demanded flexibility and assume the constraints that the radio applications execution imposes. In addition, it is mandatory to assure that this common framework do not introduces extra unacceptable computational overhead to the processing platform. Although the computational capacity of a processor is increasing every year, the computational requirements grow more quickly pushed by the new communication standards requirements and the forecasted applications.

The next sections make an overview of the context where this work is centered, going later to describe some relevant aspects of a suitable middleware for SDR applications focusing on the concepts developed under the P-HAL (Platform and Hardware Abstraction Layer) framework .

2. COMPUTATIONAL PLATFORM

As mentioned above, the processing requirements of a SDR application require the utilization of multiple processors. Every processor class results more efficient for a given subset of the tasks required for the communications algorithm. Then, the assumption of having an heterogeneous set of hardware processing platforms and above it a software layer separating both, the application and the platforms, is reasonable.

This array of heterogeneous processors can be seen as a set of processors distributed in a network with communication interfaces established among them, what clearly establishes a heterogeneous computing working framework. Inside such heterogeneous computing context there are several ideas that can be explored. The first focuses in the data exchange process among different processors and evaluates the possibility to develop a packets based network. It makes sense if optimum utilization of the communication resources among processors is an objective. Also, attending to the reconfiguration process, the routing is an advantage if the process of mapping of the application on hardware results in indirect transfers among processors. Under such approach some kind of IP-like network (identification of every one of the processors belonging to the network) is mandatory.

The second idea focuses in the suitable topologies for accommodating the radio application under the SDR concept. Notice that some topologies, which consider distributed computing and communication resources, could better address the reconfiguration process and provide higher flexibility to achieve improvements in, for example, power consumption, reconfiguration speed, dynamic management, etc. In addition, the mapping process can experience additional difficulties due to the hardware topology in especial in the base station case where a higher set of running processes (several users) is assumed and a more dynamic reconfiguration framework must be considered.

Finally, a third idea is oriented towards the side of the analog world. It is obvious that any radio application requires analog interfaces. Then, aside the digital processors it is necessary to consider the presence of the analog processors side as part of the computing platform. In case that the available hardware platforms do not include analog components that can be treated as processors, some mechanisms to provide access and management of the analog part are mandatory. If only radio applications are focused, such action can be highly simplified, as it will be seen later on.

3. SOFTWARE RADIO APPLICATIONS

The peculiarities of communication systems and the fact that they must be implemented in SDR environments, where the reconfiguration process is fairly problematic, can be summarized as:

- Most RATs access the transmission medium by means of a time-slot based division.
- Periodic execution of the same set of functions while receiving continuous or burst data streams.
- Real-time processing requirements on limited resources, where speedup is not the prime objective.
- Partial and total dynamic reconfiguration of the different layers in the protocol stack.
- An increasing heterogeneity of processing platforms.
- Highly variable computing loads and different real-time restrictions as a function of the radio standard.
- A higher efficiency in spectrum occupation generally requires more computing power.
- Higher bandwidth demands due to new user services.

Maybe the most important feature around radio applications is the strong temporal relationship of the algorithms involved in a communication system. Indeed, most parameters are defined in terms of frequencies or periods while the information flows at a given number of symbols per second. In the previous list, the first bullet point states that the computing resource management explores some time granularity. Computing resources periodically execute the same processing chain, where the execution within a period or processing time slot must finish on time. That is, instead of speeding-up the execution, the objective is to properly deal with real-time issues. Note that the last implies two different timing approaches that not always coincide in the same execution sequence or criterion. Appropriate timing mechanism should be provided by the SDR middleware in order to fit the real-time objectives.

By other hand, a radio application is often represented as a set of black boxes that process data in a streamline. Any one of these boxes can perform its tasks independently of the other boxes except for the need of interfaces to exchange information from the ones to the others. Simply stating, a radio application can be decomposed in a set of independent execution threads that use standard interfaces to allow their interconnection. Through universal interfaces, it becomes possible to plug and unplug application components as well as reorganize the application, that is, alter the data stream path and processing steps. Because that the provision of suitable interface mechanism must be one of the relevant services provided by the SDR middleware.

To summarize, a radio application is defined as a set of independent objects [3] that exchange data through well-defined interfaces and process them following a given temporal framework. Note that the independence of objects empowers the possibility to develop them by separate, thus facilitating the distributed or team-based application development.

4. MIDDLEWARE SERVICES AND FUNCTIONS

From the previous section it can be concluded that the temporal management of the applications together with the possibility of moving data are the cornerstones of any platform willing to provide a means to execute a SDR application. In general this is not new but the scope of the implications of the previous sentence has to be focused to issue a useful implementation (see next section).

However, some additional services to the application are foreseen. In particular, monitoring the application or allowing it to show internal status values is crucial. Through such parameters decisions in upper control layers can be performed. Then, a method to publish such internal parameters has to be added. In the reverse sense, any control entity may wish to modify any of the parameters that a given object can change dynamically, at run-time. For this reason the publishing method must include both, read-only and read-write parameters. Finally, it is possible that the application requires to setup some basic parameters at the beginning of its execution. In this case it should be able to request and obtain them.

Five fundamental services have been reported: time control, data exchange, parameter monitoring, external parameter modification and parameter request, which can be enclosed in three different service types: time, data and parameters. About functions, they include all the tasks that do not directly interact with the application objects but that are necessary for the correct application execution. The first function is obvious and must be the capacity to launch the different execution threads (what was called objects) on the several heterogeneous processing platforms. After this it is necessary to have the capacity to pause and stop such threads. A second function is related to the timing control and consists would consist in a mechanism to ensure that all the temporal references of all the platforms and processors an them are synchronized. This may seem obvious but when multiple platforms are joined together it is likely that every one has its own timer and clock source, which are not necessarily running synchronized.

The common denominator as well in services as functions is the simplicity one just to save as much computational resources as possible for the algorithms in a radio application.

5. P-HAL MIDDLEWARE

The developed Middleware, defined as P-HAL (Platform & Hardware Abstraction Layer) tries to advance in the process of defining a common framework to develop and deploy software radio applications by eliminating platform (hardware and support software) dependencies. The functionalities identified and developed under a Hardware Platform mixing DSPs and FPGAs [4], includes real-time seamless exchange of information from one P-HAL compliant platform to another (BRIDGE), isochronisms of data and processes running on different platforms (SYNC), coordinated process control and scheduling on any platform (KERNEL), real-time system monitoring and data and statistics retrieval (STATS), real-time adaptation of processes set-up parameters (STATS), event logging and error control (KERNEL) and computing and platform resources management [5].

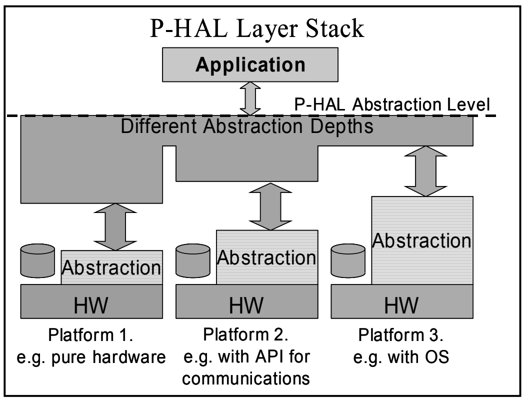

Figure 1. Different levels of abstraction

As stated in Figure 1, in some cases the abstraction is deeper than in other cases but shows that it is possible to implement the features of P-HAL in devices/platforms as different as Linux-based hosts on embedded processors, DSPs and FPGAs. Clearly, the abstraction complexity is, to some extent, the measure of the cost of the middleware (overhead) that can be defined as the use of resources required to translate the virtual services offered to the application to the optimized hardware-dependent functions. This overhead is always a dependent on the platform. However, for some particular study cases it has been observed that overhead can be well below 1% of available resources. In particular, the functionalities related with the coordination and synchronization requires and important part of the development efforts. Some of them are described in next subsections.

5.1. Attaining temporal control

As stated, the execution of the radio application in real-time is necessary. Starting from the idea that objects building the application are implemented independently and not necessarily focused to a given application, it is clear that timing control has to be completely external to the application objects. Moreover, as mentioned above, the hardware layer is considered to be build up on the basis of multiple platforms from different manufacturers that can interact through standardized communications interfaces. Then, there are no hardware mechanisms to synchronize them in a precise way (e.g. common clock).

With the previous two features in mind and taking into account the other features of radio applications, the timing control in P-HAL has been designed to be obtained through a slotted division of the time. This division has both advantages and also some penalties, but the former compensate the latter. The first advantage is that allows a simple and clean control of the execution time of the application objects, having the possibility to periodically control that every task is finished within a given deadline. The second advantage is related with the first one in the sense that every object has a limited work to do in a given time slot, which is proportional to the amount of data to be processed. This amount is also proportional to the time slot length. Then, if into every time slot any object processes the data generated by the other objects in the previous time slots, a data pipeline is achieved and there is no need to schedule the execution of the objects, thus simplifying the operating system scheduling algorithms. Additionally, since the time slot length can be relatively long compared to the operation frequencies of hardware, the data pipeline among different platforms reduces the complexity of the management of the transfers when trying to guarantee real-time.

The previous features are illustrated in the Figure 2. Three objects exchange data, which flows from A to B and then from B to C. After A has generated new data the middleware sends them to B, taking into account the slot end deadline. In the next slot, object B takes the newly available data, process them and, finally, sends them to C. In the third time slot, object C will start processing data. After the initial slots, every new one there is data to process, achieving a continuous data flow. The penalty of this approach is that some latency is introduced from the data input to output, which can be computed straightforward taking into account the slot length and the number of stages in the application. It is simple to see that to avoid large latencies short time slots are required and then, as faster is the hardware shorter can be the time slot. Another issue to take into account is the effect of this pipeline in the presence of data loops in the applications. Consider the case where the output of C would enter again into A as an additional feedback input. It can be seen that the data pipeline becomes a delay on the signal samples. Two possible solutions to this problem are envisaged. First, grouping objects involving loops in a single object. Second, including with the application an indication of the maximum slot length supported.

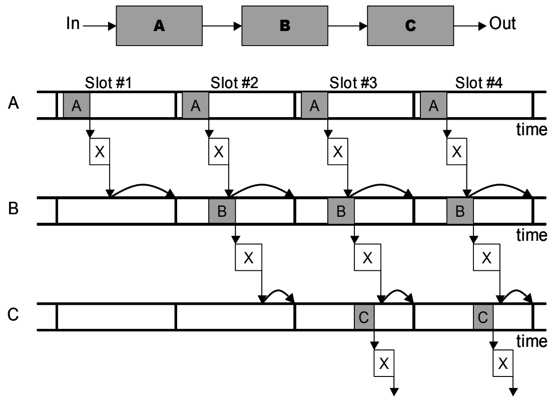

Figure 2. Slotted time and data pipeline

The figure also shows the effect of running the three objects on a single processor, which executes them sequentially. On the contrary, they would run simultaneously on different processors or in case of being separate objects on the same FPGA area. Note that the amount of time that any object takes to complete its processing is governed by the amount of data in a time slot at its input. This imposes some structure to the programming of objects since the amount of operations they are performing has to be always related to the amount of data available in the current time slot and having as limit the maximum number of operations that can be done in the decided length for the time-slot in that system. Then, the structure of program must be data-oriented and not time-oriented. With this approach the real-time is guaranteed if the sum of all the objects required processing power and bandwidth is not higher than the available ones.

5.2. Synchronization of multiple platforms

Since the execution time framework is based on a large time interval, the slot, it is only necessary to achieve a synchronization precision that is small compared to the time slot length. On any given platform with its own implementation of the P-HAL services, the way as the time is controlled is not relevant if enough resolution is provided there.. When two platforms like this are joined both internal timers have to periodically synchronize to compensate the tolerance of the oscillator frequency. For instance, if a time slot of 1ms is considered and a difference in slot references of up to 0.5% is accepted, a synchronization procedure has to be carried out every 50ms or less if the relative stability is of 100ppm. Taking into account the procedure described following, this value may give an idea of the order of magnitude of the speed of the hardware.

The synchronization procedure between two platforms uses the available communications interfaces. Within the overall platform controlled by P-HAL there is a hierarchical structure of timers. Every P-HAL incorporates a time server (MASTER) and a time client (SLAVE). After the top level P-HAL control mechanisms have assigned one role to every entity of every platform, the time from the primary reference is propagated to other platforms. All those that have direct connection with the primary reference request the time to it and receive an answer with the current time. Similarly, those platforms not having direct connection may request the time to any other platform already connected with the reference, as it is shown in the .

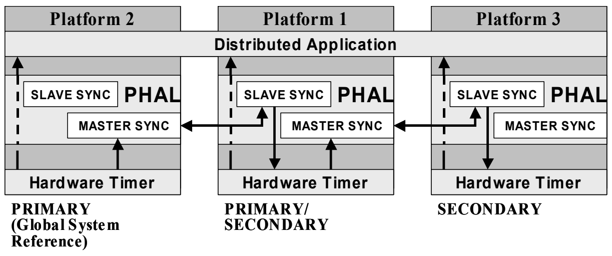

Figure 3. Synchronization hierarchy and propagation

The possible propagation of the time across multiple master servers is not recommendable since it introduces additional uncertainty to the references that the distributed application can see (every part of the application receives the reference from the local hardware through the P-HAL functions). However, since the P-HAL maintenance procedures will use some slot time and the data transfers always happen after some processing, it is unlikely that a platform receives data tagged with a future time slot. Anyway, if this where the case the data wait to be processed in the corresponding slot, according to the pipeline. By the other hand, if data arrive from the previous slot, it is considered as delayed data. This can finally result in a real-time fault if the object that had to process the data has already completed its work in the current time slot. But, as mentioned above, this is highly unlikely given the synchronization precision that can be achieved.

One of the most important decisions to take in the P-HAL context is to determine which platform is going to be the primary system reference. But according to the need to generate and receive signal from the real world (analog), it is reasonable to take the reference from the platform that incorporates the A/D and D/A devices. If, moreover, it includes the local oscillators for mixers and RF subsystem in a realizable Software Radio (the frequencies generated from a single source), all the system time references can be coherent.

The synchronization procedure starts with a request from the slave to the master of its local time. Upon receiving the request the master answers returning such parameter. In total, including header a payload, the P-HAL packet is 24 bytes long in both cases. The total time to complete this handshake determines the precision of the synchronization which can be almost zero in case that both transfers take the same time. In the case of two SOTA PCs with a Linux OS, the network handshake when running in high priority real-time mode is less than 100us, leading to a synchronization error much lower than 50us (worst case error, half the handshake time). However, in a embedded system with platforms including DSPs and FPGAs, when interacting through a VME bus, the error after synchronization can be as little as shown in the Figure 4 (1ms time slot). Since the transfers lasts few nanoseconds, the handshake length is of very few microseconds, and then, the error is very small. The chronogram in Figure 4 has been taken from impulses generated by the timers belonging to relatively obsolete DSP and FPGA devices (Texas Instruments TMS3206701 and Xilinx XC4013) running on a VME bus.

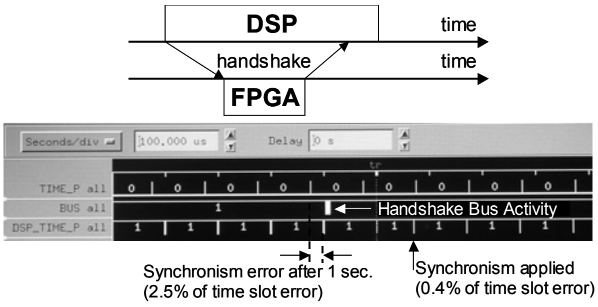

Figure 4. Time reference error before and after handshake

5.3. Data routing

Together with the time control, the capacity to move data from one object to another in a seamless manner is basic. Within P-HAL every object has a logical identifier within a given processor (the one running it), a processor identifier and, finally, a platform identifier. The composition of the three identifiers is the logical address of the object. However, to actually move data in and out the object, every logical interconnection has to be assigned to a physical route. Then, every P-HAL processor includes a routing table that indicates, for every object logical connection, the physical interface to use and the identifier of the target device (e.g. an address in a bus interface) where another P-HAL entity will continue the movement of data.

The specification of the routing table is done during the application mapping procedure, where the location of every object is determined and the best route selected. Any change in the application requires an update in the routing tables. It is clear that using a packet-oriented approach in the previous scenario has many advantages. To the possibility to tag every packet with a time stamp it is added the simplicity in managing simultaneously on a given physical interface the different logical data flows.

5.4. Interfacing the real world

The applications remain almost completely enclosed within the virtual context offered by the middleware. They exchange and process data on the basis of some externally set parameters and supply other externally accessible values. But it is necessary to specify in the application context the input and output of the processed data, either in the channel side or in the user side. Then, it might be necessary to reach the hardware from the application. Of course not the actual hardware but a given one that does not change as a function of the underlying platform. This abstracted hardware is accessible through the hardware abstraction layer part of P-HAL. In this case the HAL is defined as a set of application objects that may be available with the P-HAL implementation of a given platform. If the set of platforms include at least one with the possibility of implementing such objects, the application will be successfully mapped on them, otherwise it will not. Then, in the process of definition of the application, HAL objects are instantiated in the same way as radio function objects are (e.g. FIR, NCO, etc.). Six HAL objects are identified so far:

1. TEMPO: Objects without input data require time information. This object provides the "t" axis to objects requiring it.
2. B_LINK: to enter and leave data to/from the user space application. If the radio application includes all the communications stack layers, the upper one, the user application, requires a means to send and receive data.
3. RX_CH: object whose input selects a given channel (central frequency, bandwidth and sampling rate). Its outputs are the samples of the input signal.
4. RX_PWR: object whose input selects the amplification that must suffer (in dB) the signal that is currently being received through RX_CH.
5. TX_CH: transmission side equivalent to RX_CH.
6. TX_PWR: transmission side equivalent to RX_PWR.

All these elements require a specific implementation since they control the available hardware, which in part is analog. It is possible that some parts of the object are implemented by using digital technology and others by using the analog one. However, the presence of such objects is only a temporal feature of P-HAL to achieve implementations in current technology. The idea here is to span the description of the application to all the set of functions, not worrying about its actual implementation, analog or digital. The final decision of the objects that are mapped on digital processors and those that are mapped on analog ones is left to a mapping algorithm that takes into account the available resources. The implications of a mapping algorithm are out of the scope of this paper but its presence is very relevant to achieve a good utilization of the platforms resources and take advantage of the flexibility offered by the middleware.

6. CONCLUSIONS

In this paper we have presented a middleware suitable to run on heterogeneous processing devices to offer the required services to software radio applications running in real-time. The well-known features of such applications allow creating a thin abstraction layer from the platform, whichever it is, to spend as few resources as possible in the path from the virtual application to the real hardware.

The temporal control for real-time execution of distributed applications on several platforms is one of the most interesting problems to solve. In this case the division of time in slots together with data pipelining has been considered an adequate solution because of its simple management, the reduced implementation cost and the limited drawbacks. This feature together with the application organization in independent objects exchanging data that is conveyed in packets, make of P-HAL a very flexible middleware for the deployment of software radios on SOTA digital processing devices.

7. REFERENCES

[1] J. Mitola III, "The Software Radio Architecture", IEEE Communications Magazine, Vol. 33, No. 5, pp. 26-37, May 1995.

[2] A. Munro, "Mobile Middleware for the Reconfigurable Software Radio", IEEE Communications Magazine, Vol. 38, No. 8, pp. 152-161, August 2000.

[3] A. Gray, C. Lee, P. Arabshahi, J. Srinivasan, "Object-Oriented Reconfigurable Processing for Wireless Networks", IEEE International Conference on Communications, 2002.

[4] X. Revés, V. Marojevic, R. Ferrús, A. Gelonch, "FPGA's Middleware for Software Radio Applications", 2005 International Conference on Field Programmable Logic and Applications (FPL'05). Tampere, Finland. August 24-26, 2005.

[5] Xavier Revés, Antoni Gelonch, Vuk Marojevic, Ramón Ferrús. "Software Radio: unifiying the reconfiguration process over heterogeneous platforms". EURASIP Journal on Applied Signal Processing. September 2005.

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