HOW TO OBTAIN MORE POWERFUL SDRs USING MULTICORE ARCHITECTURES

Raúl Dopico-López (Indra Sistemas, Aranjuez, Spain; rdopico@indra.es)
José M. Camas-Albar (Indra Sistemas, Aranjuez, Spain; jmcamas@indra.es)
Miguel A. Melchor (Tecnalia-Telecom, Zamudio, Spain; mmelchor@robotiker.es)
David Castells-Rufas (Universitat Autònoma de Barcelona, Spain; david.castells@uab.es)

ABSTRACT

Have we reached the needed performance to deal with SDR (Software Defined Radio) user needs yet? Are we limited to basic applications due to conventional SDR platforms characteristics?

During last years, end users' demand in communication characteristics has caused the need of advancements in SDR terminals in order to efficiently deal with high data-rate processing and innovative capabilities. Consequently, different SDR architectures have been proposed by providers and developers in order to palliate communication performance problems, mainly providing lightweight software implementations or multicore hardware replacements. However, those solutions are neither flexible enough to be integrated with conventional SDR platforms nor scalable enough to have their inner software adapted to forthcoming hardware advancements.

Therefore, an adaptable multicore architecture is presented in this paper as a suitable solution to cover nowadays SDR performance needs, characterized by supporting the heaviest SDR processing and being able to be fully exploited by means of advanced parallelism techniques.

1. INTRODUCTION

From Software Defined Radio emergence on, endless worries have arisen to improve its performance capabilities, and hence get more efficient communications. A wide range of alternatives have been rapidly adopted to face these issues, such as replacing conventional architecture hardware elements with more powerful ones, or even reducing software components which normally slowdown SDR performance. Nevertheless, these straight measures can be a double-edged sword; since although some timing gain can be obtained, they depend on technology advances or even forgo some of the advantages – such as portability, reconfigurability or cost and time reduction - that software provides to radio terminals.

As a consequence of above stated troubles, some multicore-based solutions have been recently adopted to counteract SDR performance problems. However, current applications are not able to fully exploit such parallel architectures and hence obtain considerable improvements in the SDR field. Even more, hardware architectures are in constant evolution and software applications are not aligned with hardware progresses, producing a scalability concern. In addition, multiple innovative SDR designs based on MPSoC architectures have arisen, but always considering whole SDR functionality inside the MPSoC, hence overlooking possible extensibility concerns from conventional architectures.

All previous things considered, herein a solution based on parallelism and High Performance Computing (HPC) is presented, preserving the benefits of software in the radios and providing alternative means to improve current SDR capabilities. Particularly, the conventional architecture model, conceived as the typical SDR combination of a GPP (General Purpose Processor), DSP (Digital Signal Processor) and FPGA (Field Programmable Gate Array), is upgraded with a new Multi-Processor System on Chip (MPSoC) co-processor, which allows to efficiently computing intensive SDR functions by using parallel programming techniques. MPSoC is used in this design as a flexible platform that, once it is defined the number of processors (referenced hereinafter as soft-cores) running in parallel on it, aids the other SDR modules/processors to execute their assigned functions.

Furthermore, an accurate MPSoC internal definition is essential to obtain a worthwhile set of results. It collects both an optimal selection for the communication mechanism between the soft-cores deployed in the MPSoC and an adequate parallel programming library to master the soft-cores concurrency. Adaptation of already known solutions, such as Network-on-Chip (NoC) to manage modules communication, or tailored solutions to deal with parallel programming in Embedded Systems are considered.

The approach followed in this article (derived from the analysis performed during the European ParMA project [1]),

introduces an SDR parallelism-based framework which aims to:

- Enhance communications time performance in order to approach to real-time transmission.
- Alleviate conventional SDR processors overload.
- Enable new SDR services thanks to these multiprocessing capabilities.

2. PARALLEL PLATFORM CHARACTERISTICS

The parallel platform to execute SDR applications consists of several parts. On the one hand, a hardware platform obviously provides the necessary computing elements and their interconnection infrastructure, but on the other hand it is also necessary a set of libraries and runtime software that coordinate the different elements to successfully run the software defined radio tasks.

2.1. MPSoC platform specification

An MPSoC is a system-on-chip with multiple computing modules (known as tiles), typically targeted for embedded applications. These computing tiles are linked to each other by an on-chip interconnect and contain most or all functionality of a complete electronic system integrating a CPU, data storage elements, specific peripherals, a bus to interconnect the different elements and inputs and outputs. An MPSoC can be homogenous if all its elements are alike or heterogeneous if they are different. A homogenous architecture has been chosen for this research as it is more suitable for MPI parallel programming technique, explained in the following section. Also, some recommendations are given in next sections regarding the use of an FPGA (provided by vendors like Xilinx or Altera) as MPSoC in order to achieve full reconfigurability. Some remarkable MPSoC characteristics are:

- The number of processors in the platform is only limited by the size of the FPGA.
- Each processor has its own memory (accessible via a fast local bus) which produces a distributed architecture.
- Other peripherals are: Timer/Counter to provide accurate time stamping of events, optional UART (Universal Asynchronous Receiver-Transmitter) for debugging purposes (log and trace statements are output to external consoles) and high speed bus transceiver to allow for high data rate communication with the external world.

2.2. Software libraries to enable Embedded Systems parallelism

The availability of a high number of processors is not a sufficient ingredient to get high parallelism level. Applications should be adapted to this multiprocessing environment by using advanced parallel programming methods and techniques. De facto standards for parallel programming in the HPC community are OpenMP and Message Passing Interface (MPI [2]), for the shared memory and message passing paradigms respectively.

The OpenMP approach puts the pressure into the multicore architecture and the compiler since a cache coherent memory hierarchy is assumed and the compiler has to create all the necessary code to make parallelism happening transparently. On the other hand, MPI puts the pressure into the programmer because he must explicitly define the communication and computation pattern.

Further differences between both paradigms reside in the scalability and design complexity. For instance, MPI is inherently more scalable and has lower architectural demands, so resulting in a simpler and cheaper design. It is true that the programmer has to burden with the cost of low level programming, but this is less crucial in the embedded domain. Both paradigms require some runtime support, but it is a more important factor in MPI due to the initialization and distribution of applications among the computing cores.

MPI is a very large API with hundreds of functions and an important memory footprint. For this reason, it is convenient to use a much reduced set of functions and a lightweight version of this API. On-Chip MPI (ocMPI [3]) is a subset of MPI that is addressing this need. This reduced subset of MPI includes the fundamental point to point and collective operation functions of MPI that are enough to create a wide range of parallel applications. Another difference between ocMPI and full MPI stack is that no user defined data types are considered in ocMPI, allowing a simpler and more compact code.

2.3. Soft-core interconnects

Several interconnection strategies can be adopted to reach targeted performance needs, a bus, a segmented bus, a mesh or a fat tree among others. The options are endless and, to make it worse, their convenience is determined by the traffic pattern exposed by the running applications.

Reconfigurable platforms running embedded applications are flexible enough to propose application specific network designs to meet energy, area or performance requirements. But to be able to create these networks a large design space must be explored. Tools such as NocMaker (Illustration 1 [4]) are needed to obtain early metric estimations of different design options in a timely fashion.

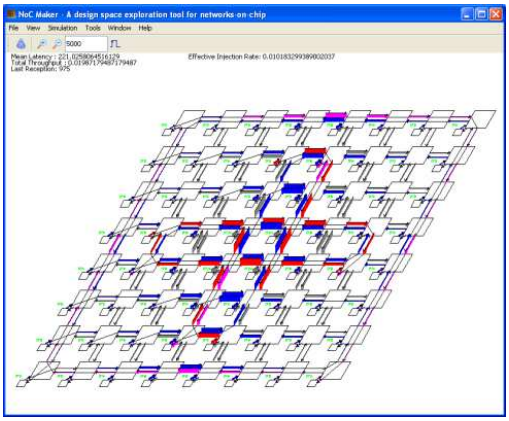

Illustration 1: NoC Maker framework for NoC design space exploration.

It is worth mentioning two well-suited examples of soft-cores interconnect models targeted to the SDR MPSoC platform:

- **P2P Network**: Point-to-point interconnection of all the processors via dedicated links (e.g. FSL protocol provided by Xilinx). This is a simple and fast solution but lacks scalability (resources grow quadratically with the number of cores). For this reason, this can be a valid choice for a small number of cores.

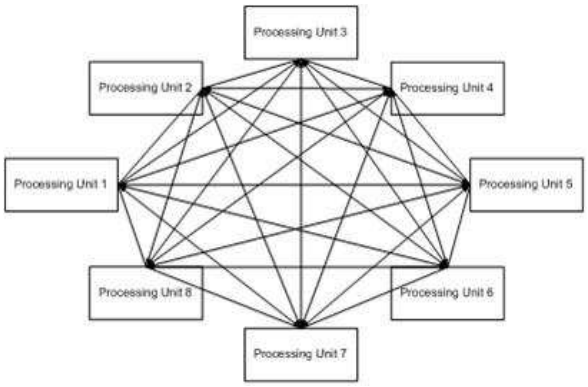

Illustration 2: P2P Network Connectivity Model.

- **Network-on-Chip (NoC):** It is the current paradigm for communications within large VLSI systems implemented on a single silicon chip. NoCs are constructed from multiple point-to-point data links interconnected by switches (also referred to as *routers*), such that messages can be relayed from any source module to any destination module over several links by making routing decisions at the switches. Two possible routing techniques on the two-dimensional mesh NoC are designed for this solution:
  - **Ephemeral circuit switching [5]** consists in a circuit switching strategy that only transfers data only during the circuit establishment signalling, tearing down the circuits immediately after connection. This approach gives a very simple hardware implementation but at the cost of adding substantial overhead and hence reducing performance. It is an interesting option if traffic is low and packets short.
  - **Wormhole routing [6]** is a packet switching variant in which packets are split into smaller units (called flits). Flits are transmitted in a packet switching fashion but circuits are reserved at the packet level. This approach introduces some buffering costs, but far less than a typical packet switching network and it exhibits much better performance compared to ephemeral circuit switching. Overall, it is a good option for large packets and sustained traffic.

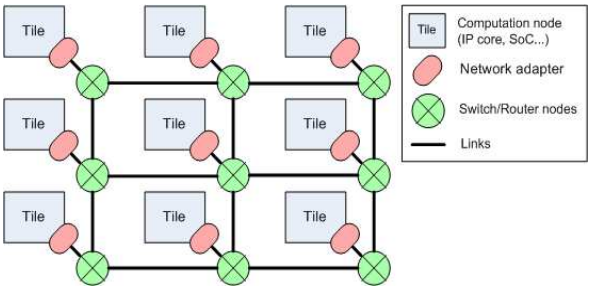

Illustration 3: Network-on-Chip Connectivity Model.

3. PARALLELIZATION OF SDR FUNCTIONS

3.1. Objective

The fact of having an MPSoC in SDR architectures provides high benefits in processing power capabilities, what is especially interesting for high intensive and processing consuming communication algorithms.

Nowadays with the emergence of new communication protocols using advanced techniques, the processing power needs are continuously increasing. In this context the possibility of including in our architecture a flexible, adaptable and scalable MPSoC allows us to adapt our architecture to the instantaneous processing requirements, what produces a high increase in terms of processing power capabilities allowing:

- The introduction of advance and emerging modulation and multi-antenna techniques.
- The improvement of the codification and error correction techniques.
- The introduction in the embedded system of emerging MAC Layer techniques, increasing the efficiency and PHY Layer usage and being able to improve the QoS perceived by the user.
- The introduction in the embedded system of emerging NET Layer advanced routing techniques, such as the deployment of MANET networks.

The consequences are quite beneficial for the applications running on the radio, and of course for the user, that will take advantage of services with higher data rate, lower Bit Error Rate (BER) and lower Latency.

3.2. Limitations

To exploit to the maximum all the possibilities of having an MPSoC coprocessor in SDR architecture, some basic concepts have to be taken into account in the Hardware and Waveform Design:

Regarding the Hardware Design, the best place in the architecture must be found for the MPSoC. Taking into account that FPGA devices are intrinsically parallel by nature, no benefits are obtained by parallelizing FPGA functions. Therefore, it seems more sensible to place the MPSoC as coprocessor of processor-type components, such as GPP and DSP, as shown in the example of Illustration 4.

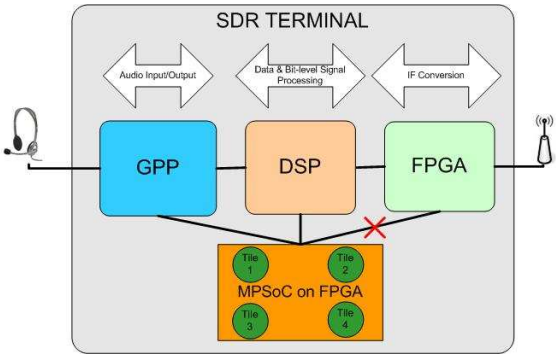

Illustration 4: MPSoC-based SDR architecture.

Regarding the Waveform design, the assignment of components to processors, among them the MPSoC, must be carefully analyzed. Those that can better exploit the parallelism must be chosen to be deployed in the MPSoC. Those functions inherently sequential (e.g. shift-register based algorithms) must be avoided in favour of those that admit parallelization (e.g, block processing based algorithms or a mapping function in a modulator). The objective is to allocate those functions that more alleviate the SDR architecture processor in the MPSoC and better take advantage of the MPSoC characteristics.

3.3. Hints on a good selection of SDR functions

As previously mentioned, the functions to be allocated in the MPSoC must be wisely selected in order to obtain the maximum benefit from the parallelism. Some good candidate examples are presented here in order to help during the selection process:

- The mapping and decider algorithms in a PSK modulator/demodulator, since the bit chain and the symbol chain can be readily divided between the different cores existing in the MPSoC.
- Block-processing bit level algorithms such us interleavers/deinterleavers or a Red-Solomon coder/decoder.
- Video or Audio CODEC and DECODEC functions.

Another important issue that must be taken into account is the time consumed by the communications between MPSoC and SDR processors. To minimize this effect it is suggested:

- All the processors in the conventional architecture running SDR functions logically connected to the next parallelized SDR functions in the MPSoC, should have a direct physical connection with MPSoC.
- When possible, the chosen parallelizable components to be deployed and executed on the MPSoC should be concatenated in the SDR Waveform logic block diagram.

4. MPSOC COPROCESSING PROBLEMS

4.1. Hardware and Packaging implications

Regarding HW issues, some considerations must be taken into account for the inclusion of the MPSoC in our architecture:

- Physical connections must exist among MPSoC coprocessor and all GPP and DSP type processors allocated in the SDR comprising “parallelization-candidate” functions in order to reduce the number of hops to the MPSoC, and hence the slowdown produced by the communications time.
- The interfaces between GPP and DSP type processors must be compatible with the ones provided by the MPSoC. In this case, an FPGA based implementation of the MPSoC eases this concern due to the inherent flexibility of these

devices that allow the implementation of different interfaces.

- The inclusion of the MPSoC increases the size of our HW architecture and may require resizing up the enclosure of our equipment.
- One of the main drawbacks of current multi-core architectures is the high power consumption. This must be taken into account when dimensioning power supply and cooling needs.

4.2. MPSoC further needs: operating environment

The bare minimum software required by the MPSoC platform and ocMPI is a Board Support Package (with drivers for all peripherals) and standard C libraries. Even in this limited environment useful parallel applications can be written and executed in the platform. However, as the complexity of the applications increase, the application developer could need to benefit from additional support, such as a small multitasking kernel or operating system with a memory or flash-based file system (e.g. Xilkernel, VxWorks, etc). As an example, some SDR functions to be parallelized could take advantage of common open source programming libraries which may require some OS services (e.g. threads, semaphores, timers, file systems and so on, which could be used by some SDR functions like Audio CODEC).

5. FUTURE STEPS IN SDR PARALLELISM

An MPSoC solution to alleviate communication performance problems in SDR has been outlined inside this article, proposing a new multi-core architecture within the SDR terminal. However, this emerging solution opens the door to further considerations within the SDR world that are currently pivotal issues, such as SCA and security features.

5.1. SCA coverage to MPSoC platform

Software Communications Architecture (SCA [7]) is an SDR standardized framework, which provides a common architecture for all SDR developments. This world-wide known *de-facto* standard brings forth advantageous properties such as extensibility, code reusability and cost reduction.

The multicore solution herein presented, could be easily represented in the SCA by considering the MPSoC as a new *SCA ExecutableDevice* characterised by a set of properties. Specifically, this new *MPSoCDevice* would be able to load and execute the subset of SDR application functions that are going to be parallelized in the MPSoC (corresponding with the SDR heavy functions previously allocated in the GPP/DSP). Besides, a few properties could be defined for this *MPSoCDevice*, such as:

- Computing tiles: This property defines a list of computing tiles, each of them specified with particular characteristics (a suitable solution for heterogeneous MPSoCs).
- Connectivity Model: The user could define a specific interconnect between soft-cores, like P2P Network, NoC Wormhole or Ephemeral NoC (see section 2.3).

In order to carry out a full interaction between the SCA component model and the MPSoC, the following aspects are recommended to be taken into account:

- The FPGA which represents the MPSoC should offer a dynamic load of its binary code. It would enable the possibility of transparently reprogramming the synthesized platform by changing the *MPSoCDevice* parameters provided.
- The *load* and *execute* functions provided by this SCA *ExecutableDevice* would permit to deploy the SDR selected functions on the current parallel platform, by incremental synthesis from the platform binary code.

5.2. Security functions parallelization

Recent developments are considering security features as key factors in SDR architecture. In fact, SCA defined some overall security guidelines [8] that are currently under consideration to face new security implementations in SDR architectures. Generally speaking, SCA introduces a security zones distribution which splits the conventional GPP into three differentiated processors (RED, Crypto and BLACK), each of them performing a different subset of functions. Inside this security architecture, the functions executed within the Cryptographic Sub-System (CSS) deal with high-data processing algorithms that imply a great amount of time.

Therefore, it is suggested to think over MPSoC platforms within the CSS that can be used as aid-computing systems in order to minimize the impact of security implementation on our SDR performance. Due to security reasons, an independent MPSoC platform should be used only by the CSS, since the INFOSEC boundary composed of RED and BLACK processors have less restrictive security requirements and thus should not be allowed to access data allocated within the cryptographic boundary.

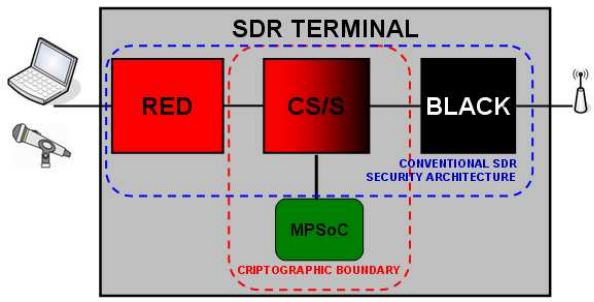

Illustration 5: New parallel architecture considering security boundaries.

5. CONCLUSIONS

As part of the parallel-based solution for SDR performance enhancement, important aspects have been tackled in this article:

- An architecture design for the parallel platform has been presented, including hardware and software recommendations.
- Typical problems encountered during the SDR functions selection and possible helpful advices have been given.
- A set of needs, drawbacks and implications attached to MPSoC usage have been illustrated, as well as possible directions to counteract them.
- Prospective movements to finally include parallelism and multicore processing in the SDR world have been introduced, looking for a perfectly suited integration with the latest SDR progresses.

All in all, this article has presented the possibilities that multi-core architectures and parallelism techniques could offer to SDR conventional systems to maximize their benefits. Thus, it has been sketched an innovative design targeted to pave the way to future initiatives considering parallelism as an emerging enabler to maximize SDR capabilities and performance.

6. REFERENCES

[1] ParMA Project, http://www.parma-itea2.org/
[2] V. Kumar, A. Grama, A. Gupta, and G. Karypis. "Introduction to Parallel Computing". Benjamin Cummings, 1994.
[3] Jaume Joven, David Castells, Jordi Carrabina, “An Efficient MPI Microtask Communication Stack for NoC-based MPSoC Architectures”, ACACES, 2008.
[4] D. Castells-Rufas, J. Joven, S. Risueño, E. Fernandez, J. Carrabina, "NocMaker: A CrossPlatform Open-Source Design Space Exploration Tool for Networks on Chip". INA-OCMC Workshop, Paphos, Cyprus, January, 2009.
[5] D. Castells-Rufas, J. Joven, J. Carrabina, "A Validation and Performance Evaluation Tool for ProtoNoC". International Symposium on System-on-Chip 2006 (SOC2006). Tampere, Finland, November 13-16, 2006.
[6] W. Dally, B. Towles, “Principles and Practices of Interconnection Networks”. Morgan Kaufmann, 2004.
[7] JPEO JTRS, "Software Communications Architecture Specification", Version 2.2.2, 15th May 2006.
[8] JPEO JTRS, "Security Supplement to the Software Communications Architecture Specification", MSRC-5000 SEC V1.1, 17th November 2001.