Distributed Reconfigurable Hardware for Image Processing Acceleration

Julio D. Dondo, Jesús Barba, Fernando Rincón, Francisco Sánchez, David D. Fuente and Juan C. López
School of Computer Engineering, Universitty of Castilla-La Mancha
Ciudad Real, Spain
Emails: {juliodaniel.dondo, jesus.barba, fernando.rincon,francisco.smolina,david.fuente, juancarlos.lopez}@uclm.es

***Abstract*—Lately, the use of GPUs is dominant in the field of high performance computing systems for computer graphics. However, since there is ”not good for everything” solution, GPUs have also some drawbacks that make them not the best choice in certain scenarios: poor performance per watt ratio, difficulty to rewrite code to explode the parallelism and synchronization issues between computing cores, for example.**

In this work, we present the R-GRID approach based on the grid computing paradigm, with the purpose of integrating heterogenous reconfigurable devices under the umbrella of the distributed object paradigm. With R-GRID the aim is to offer an easy way to non experience hardware developers for building image processing applications using a component model. Deployment, communication, resource sharing, data access and replication of the processing cores is handled in an automatic and transparent manner, so coarse grained parallelism can be exploited effortless in R-GRID, accelerating image processing operations.

***Index Terms*—image processing; reconfigurable logic; distributed architectures; component model;**

I. Introduction

Multimedia processing imposes intensive computation capabilities and even real-time constraints in some applications (i.e. 3D content rendering for mobile devices, face recognition in surveillance systems,...). Due to the limited computation capabilities of conventional processors, many of those applications must be delegated on other kind of parallel computing environments.

Many core chips with highly efficient communication architectures emerge nowadays as a solution in this area, providing a unified programming environment for parallel high-performance applications. Most of those environments are currently based in the use of GPUs with huge computational capabilities, reduced cost but high power consumption. An example is the line of NVidia products based on the Quadro CPUs [1].

In this work a general framework for the acceleration of such kind of applications is presented, where inherent application parallelism can take benefit of custom hardware accelerators. This framework, that we call Reconfigurable Grid (R-GRID), comprises a set of spatially distributed FPGAs, and optionally general-purpose processors, that includes a set of functionalities that have been developed to provide the ability of a transparent implementation of concurrent distributed hardware applications in order to obtain maximum benefits from reconfigurability and spatial distribution of resources.

II. Reconfigurable Computing for Image Processing

Image processing has been a traditional application field for reconfigurable computing since the very beginning of this computer architecture area [1]. Reconfigurable devices provide a midway approach between pure software and custom hardware development, which make them suitable for all kind of applications where extra acceleration is needed at a reasonable cost. More specifically, image processing, and computer graphics in general, can take benefit from these architectures since they exhibit a high degree of parallelism, require high memory bandwidth and use simple fixed point operations in many cases.

Image processing is normally decomposed in a processing pipeline where a set of different operations are applied over a stream of data with the possibility of having multiple inputs and outputs. Reconfigurable devices are specially well suited for this, in special when the type and size of data is not the standard in general purpose computing. On the other side parallelism can be exploit to the last consequence, not only for fine grain repetitive operations applied to a pixel neighborhood, but also for the global concurrent processing of several images at a time.

The irruption of GPUs in these fields has raised the discussion about the use reconfigurable architectures versus the more general purpose solution they represent. However, there is not a “good for everything” solution, and each technology has its advantages and drawbacks [2]. GPUs are good at fine grain parallelism, where the same simple operations are applied to a set of pixels without much coupling between them, but they failed for those situations where multiple passes are required to complete an operation. On the other side, FPGAs run at much lower clock frequencies, with leads to lower peak speeds, although they are much more power efficient, and are also more scalable.

The solution we propose in this paper is based on the grid computing paradigm, with the purpose of integrating heterogenous processing devices (reconfigurable or not) under the umbrella of the distributed object paradigm [3]. The grid approach implies a common logic architecture of the grid,

[1]Quadro website:http://www.nvidia.com/object/quadro-fermi-home.html. Last visited July 2011.

regardless of the concrete hardware it is built upon, and a set of standard services in order to ease the design, deployment and run-time management of any application.

III. R-GRID FRAMEWORK

As stated before, R-GRID provides a distributed set of reconfigurable resources and the technology necessary for transparent deployment of multiple applications. The R-GRID main objective is to provide the clients with a safe, reliable and accessible platform to implement and deploy accelerated applications using hardware resources in a transparent way.

However, it is necessary to take into account that clients can develop their applications using an endless of different hardware structures, components, tasks or functionalities connected through very different interfaces and using different communication mechanism. In order to facilitate the deployment of such diversity of applications, R-GRID provides also an Application Development Model.

Since R-GRID is based in the distributed object paradigm, each processing element in the application is modeled as an object and data exchange flow is modeled using the consumer-producer approach.

This approach is the one typically used in multimedia software processing frameworks such as avstreams [4] or gstreamer [2]. Apart from considering processing elements as objects, another remarkable feature in R-GRID is how processing data is stored. Here no specific memory implementation is assumed, and data is accessed through a functional interface. Memory is considered a data container whose content can be read and written, but without imposing any particular kind of implementation (FIFO, shared memory, scratchpad memory, etc). We consider the result as a component.

Once the application components have been defined, the next step is to decide wether to implement each one in hardware or software, compose the application and deploy it over a certain FPGA technology. The first problem to solve is the location and connection of the core component and the rest of the application. It could be decided at compile time, and statically assigned to certain resources (reconfigurable and memory areas), but this solution is not flexible and it is against the idea of the grid. R-GRID provides the necessary infrastructure to dynamically locate the component in any of the available resources of the architecture, and make it available to the rest of the application (as described in next section). Dynamic allocation also implies the use of a dynamic addressing mechanism where the components can be accessed through a run-time resolved reference.

A. Application Description

The description of an application in R-GRID starts with a XML file. In this file all components and its corresponding binary files are listed. Binary files can shared objects for software components or a set of configuration bitstreams for reconfigurable hardware. The bitstream associated to a hardware component is related to the technology where component

[2]http://www.gstreamer.net

```
<rgrid>
  <application name="Simple">
    <component id="controller"/>
    <component id="core"/>
  </application>
  <binaries>
    <binary component="controller" type="sw"
        bin="/opt/rgrid/controller.exe"    />
    <binary component="core" type="virtex4"
        bin="/opt/rgrid/core_v4.bit"       />
    <binary component="core" type="virtex5"
        bin="/opt/rgrid/core_v5.bit"       />
  </binaries>
</rgrid>
```

Fig. 1. XML Description of the example application

will be implemented. Let us suppose an example of application formed by two cores to be placed in different resources. Figure 1 shows the XML description. In this simple case there are only two components, the controller of the application and the core processor. The task of the controller is to schedule the data distribution and execution of the other core, so it has been mapped to a software node, while the core is a pure hardware component. Both of them have at least an associated binary file, a software executable for the controller and, in this example, two hardware partial bitstreams for the core. These two partial bitstreams are necessary to deploy the core over two different kind of hardware resources.

B. Application deployment

To deploy an application it is necessary to perform a few steps. First, registered users have to register the application using the XML file and adding the corresponding binaries files indicated before. R-GRID uses this configuration file to update grid information. Once registered, it is necessary to deploy the application. For this R-GRID takes information from configuration file about bitstream or executable file locations and send it to the corresponding node to start either reconfiguration of the hardware resource or software node respectively. Finally, R-GRID updates the addresses of the new instantiated components in order to allow access to them.

IV. R-GRID ARCHITECTURE

R-GRID architecture was defined taking into account that must provide a framework for the execution of high- performance distributed applications in a multiple users environment over a scalable, distributed and heterogeneous reconfigurable platform. For this a set of services were created to facilitate user registry, application registry and deployment, and remote resources management.

A. The R-GRID Logical Architecture

Figure 2 describes the logical architecture of the platform, which has been divided into three levels: the R-Grid administration level, the platform management level and the node level.

User Administration Level and Platform Management Level are implemented in software and are part of the R-GRID Server. R-GRID Server is responsible for administration of the

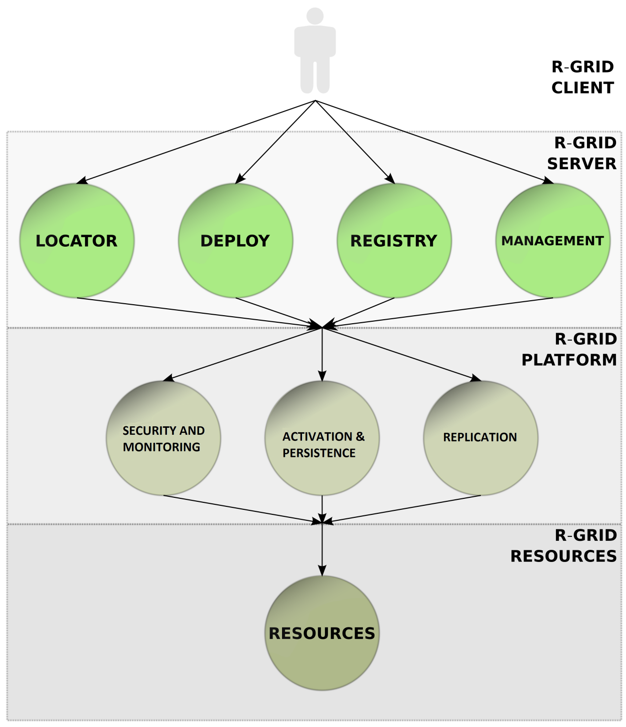

Fig. 2. R-GRID Logical Architecture

whole system, keeping information about descriptor of applications, registered users, correspondence between applications and owners, components and nodes, used resources, available resources, etc.

R-GRID Server indicates to client which kind of FPGAs are available and which one will match with user resources requirements, indicating FPGA type and model in order that client can create the corresponding bitstream for that FPGA.

1) User Administration Level: To start using R-GRID, the Server offers a user interface, fist level, which is formed by the Registry, Deploy, Root Locator and Management objects: The Registry is an object with a public interface that allows clients with the proper access rights, to register a new application in R-GRID. A repository of applications is also offered that can be used anytime and from any location. The Deploy object is accessed by users to perform the deployment of registered application. Each application is formed by one or several component and each component has an associated bitstream. Each bitstream has also associated a node.

The Root Locator object is a component that keeps information of deployed component and their location. It is intended to provide access to already deployed components.After deployment the Deploy object will update the Root Locator Object to register component address.

Management Object gives administrator access to all system functions, to ensure the proper behaviour of the R-GRID,providing to the administrator information about the availability of resources, the state of the system FPGAs nodes, the addresses of deployed objects to solve any problem that arise.

2) Platform Management Level: As second level we found R-Grid Platform management level where transversal platform management decision are taken. Issues concerning security, monitoring, activation, component replication and persistence take place at this level.

a) Security: Security module is implemented taking into account user and application points of view. In the first aspect, security module will authenticate and authorize users, checking if they have privileges to perform required actions. User identification and permission to perform actions provide security in a way that only authorized users can make use of storage, deployment, management or location services.

From application point of view, due to R-GRID runs different applications from different owners over shared resources at the same time, control application access needs to be made at each computing node level of R-GRID. At this level, security functionality will ensure that each component of each application can contact and can be contacted only by components that belong to the same application. This functionality is performed by the Object Adapter on each FPGA. In this way, the execution of each application can be performed safely.

b) Monitoring: Monitoring functionality collects all information about use and availability of grid resources, status data, load and queue status, to provide information to the administrator to facilitate grid utilization and resource brokering.

c) Advanced functionalities: Activation, Replication and Persistence are advanced functionalities that are invoked by Root Locator and Deploy objects.

Activation: is a mechanism that implement a non instantiated component when is needed by other component of the same application. This situation can occur if a component of the application was removed and is later required. In order to avoid collapse of the application, this advanced functionality instantiates the required component when that requirement occurs.

Replication: is a functionality that allows component replication, if application and resources availability permit. This functionality will create a new instance of deployed component enabling parallelism.

Persistence: In case an instantiated component is removed or stopped from the grid, Persistence allows saving its state in order to be later reused if component is reinserted. With this functionality, components can be removed from the grid and reinserted in another place when needed, without lost of data consistency.

B. The R-GRID Physical Architecture

The R-Grid resource level in figure 2 is formed by heterogeneous FPGAs nodes, so this level is the physical architecture level. R-GRID is intended to hide implementation details of computing nodes to upper levels. The relationship between the logical and physical architectures can be observed in the figure 3.

R-GRID physical architecture was implemented using dynamically reconfigurable FPGAs. Each FPGA node is divided

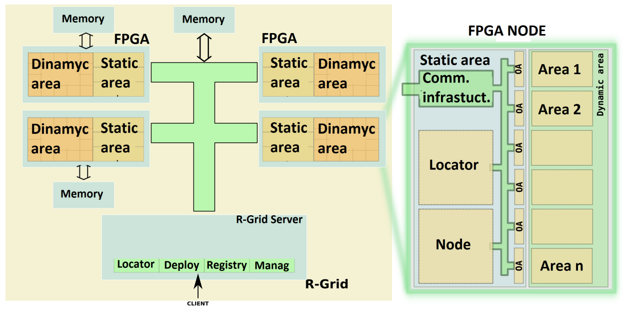

Fig. 3. R-GRID Physical Architecture

in two parts: the static part, formed by three objects: Locator, Node and Object Adapter, and a dynamic part, which is formed by several dynamically reconfigurable areas, as shown in figure 3 . Each reconfigurable area can host a component that can be formed by several objects depending on its size. The location of each instantiated object inside the partial reconfigurable area of the FPGA is solved by the local Locator object.

The partial reconfiguration process of a reconfigurable area is triggered by the *Deploy* object, placed at User Administration level. The Deploy object commands the *Node* object of the target FPGA. The Node object takes the bitstream from memory and performs a partial reconfiguration of the FPGA. The Node object isolates the partial reconfiguration mechanism inherent to each FPGA from upper levels.

The Object adapter provides the mechanism to control and to avoid the access to instantiated objects from objects belonging to different applications, providing security at application level and isolating each application from another within the system. Each object adapter has a unique global address within the system. These components are entirely hardware implemented and create the abstraction layer from technology.

C. Communication Interface

Components can be locally or remotely connected. In the former case it is equivalent to a point to point connection from the client to the server, and implies no additional logic, while in the latter it involves the use of remote adapters, in order to translate messages into the transport protocol used by the communication link.

R-GRID extends the functionality of the OOCE (Object Oriented Communication Engine) middleware [5] in order to enhance inter FPGA communication of data. The base configuration in OOCE isolates the computational aspect from communication details, mainly at chip level. In this scenario, software components can access to hardware components and vice versa. OOCE interface compilers generates the software and hardware adapters that automatically manage hardware-software interfacing. These adapter are in charge of translating bus request and module activation signals in one domain to conventional calls in the host processor. A more detailed description about the adapters and how they can be automatically generated is provided in [6].

An extra effort has been done in the abstraction of the memory interfaces and the memory hierarchy in R-GRID. For example, external memory to a component can be implemented as a shared memory in the same node or any other kind of memory in a different node. In any case, the middleware provides the component with the MemoryResource abstraction, through an adapter, so the physical access to such resource is decoupled from the component's logic. For example, it might be available through a DMA controller, a point-to-point connection or, through a Ethernet network.

In summary, data flow is performed through the requests to a MemoryResource interface. Such interface represents a generic memory and provides methods for reading and writing data blocks. The addressing of the different memories is delegated to the proxies together with the global addressing system used in R-GRID.

V. Component model in RGrid for image processing

In this section, the component model chosen in RGrid for image processing is described. The hardware processing cores deployed in the reconfigurable areas of the FPGA must follow this model in order to assure compatibility and correctness in the system. This can be guaranteed since our component model defines the communication and synchronization mechanisms in order to efficiently move data between hardware (and software) components.

The component model developed for R-Grid is inspired in the one defined by the Khronos Group for multimedia and streaming applications; the OpenMax (in brief, OMX) standard [7]. OpenMax defines a set of Application Programming Interfaces (APIs) at different layers, each layer representing a domain of compatibility: (a) application, (b) component integration and, (c) development of core functions. The main goal is to shorten the time needed to introduce new products in the market by means of reducing the effort required to port a legacy application to a new platform or architecture. However, the standard only specifies the primitives and services from an only-software position. We have extended the OpenMax vision [8], mainly working at the *Integration* and *Development* layers of the standard, to embrace hardware components to accelerated functions as proposed in RGrid.

An R-Grid application is, thus, defined as a collection of OpenMax compliant components either implemented in software or hardware. The synchronization and communication mechanisms, defined by the OMX Integration Layer (IL), have been tailored for our RGrid platform respecting the standard interfaces. This allows software version of OMX components to be codified exactly in the same way it is done for OpenMax non-RGrid applications. Software components are distributed as shared objects and the node's RGrid run-time where the component has been actually deployed is in charge of loading it and feed it with data. A more detailed description about how data is exchanged for the intra and inter node scenarios is provided next. To date, only *tunneled* means have been considered in RGrid because its non-centralized approach for data flow management.

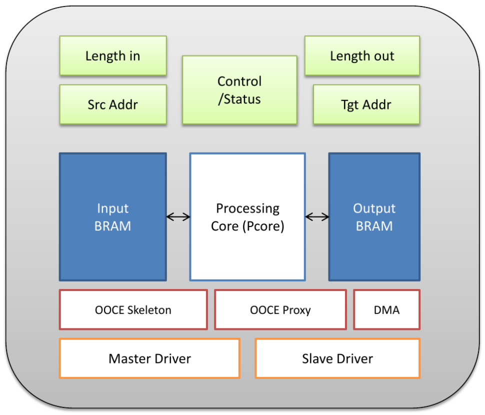

Fig. 4. OpenMax Hardware Component Architecture

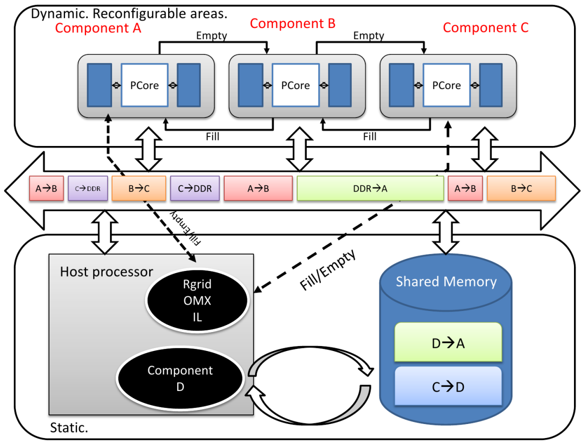

Fig. 5. HW→ (top) and HW⇔ (bottom) dataflow in RGrid FPGA node

A. Component architecture

The architecture of the *Hardware OMX Core* (HOMXC) (the hardware version for the standard OMX component) is thought as the placeholder for the PCore (*Processing Core*, the logic that implements the actual processing operator). The PCore has a fixed physical interface to the HOMXC which makes it independent of the bus technology and, therefore, the platform it is going to be deployed. Figure 4 shows the structure of a HOMXC mainly dominated by the presence of two local memories where input and output data are stored. At least one *buffer* (the minimum data unit to be processed by a HOMXC) must be in the input memory before the PCore starts to process it. It is usual to dimension such memories to hold a minimum of two buffers in order to implement *ping-pong* buffer techniques to help to reduce the number of cycles the PCore is waiting for new valid data.

The logic surrounding the PCore is not only intended for isolation purposes but also for implementation of the OMX Core functionality supported in RGrid. The *skeleton* interprets the bus requests and recognizes the operation to be triggered, typically synchronization primitives, component parameter setup, etc. The *proxy* controls the initialization of buffer transfers through the communication channel with the help of the local *DMA* control. The *drivers* are the only modules totally dependent of the physical communication protocol. They role is twofold: in one hand, they decode the bus signal activity and activates the skeleton (slave). On the other hand, they control the low level data flow through the communication channel (master).

It is worth remarking again that the implementation of the PCore is completely orthogonal to the rest of the infrastructure above described. Since the interface to the shell and the activation protocol are defined beforehand, new developments only have to focus on implementing efficient architectures for the algorithms to be accelerated. The ultimate goal is to ease as much as possible the elaboration of a HOMXC, trying to make it similar to the software case. Therefore, ongoing work is exploring the use of *Menthor Graphics Catapult C* [3] to synthesize the PCore functionality from a high level ANSI C++ description.

[3]http://www.mentor.com/esl/catapult/overview

B. Intra-node communication

This scenario refers to all data movement that takes place between RGrid components hosted in the same FPGA node. Then, two alternatives must be analyzed HW→HW and HW⇔SW. For the later, further considerations must be done depending on the role (producer/consumer) of the ends in the communication. Buffer progression in the processing chain between two HOMXCs realizes through efficient HW to HW data transmissions using a technique we call *DMA interleaving*. Each HOMXC implements a *local DMA* engine which transfers the output buffer, which it is written in the output memory, to the next-in-the-chain component's input memory (addressed by the content of the *TgtAddr* register.

DMA interleaving is an optimization that allows the transmission of parts of the buffer before the last word is placed in the output memory. This way, component's execution is parallelized with buffer transmission, but this is not the main advantage. Since the size of the individual bursts bus is reduced using this technique, the system scales better than using a classic DMA approach and, on top of that, the decentralized management of concurrent buffer transmissions between components minimizes the idle time of the bus and reduces the wait cycles due to bus congestion. Once a complete buffer is ready in the input memory of a HOMXC, the producer invokes the *Empty* operation on the consumer signaling it can start to process it. Once the buffer has been consumed (which means, completely read by the PCore) a new one is demanded to the previous component (*SrcAddr* register stores the address) in the chain (*Fill* operation). Notice that, again, the execution in the component is concurrent with input buffer transmission over the on-chip bus (see top half in figure 5).

Data exchanging involving one SW component and one HOMXC necessarily happens through shared memory. The RGrid OMX IL run-time is responsible for: (a) signaling the SW components a new buffer is ready in memory to be consumed; (b) transmitting the content of a buffer from shared memory to the local memory of the HOMXC has to process it (SW$\rightarrow$, SW component as the producer); and (c) managing the shared memory areas dedicated to intermediate storage of buffers and configuring the TgtAddr registers. When the HOMXC is the producer (HW$\rightarrow$), it does not realizes it is actually writing on the shared memory instead of another local memory's HOMXC making it its behavior homogeneous to this sceneraio.

C. Inter-node communication

Although it is desirable all the components of an application to be deployed in the same node (to reduce latency time due to communications), there are many situations where it is not possible to do this. For example, current occupation of the different FPGA nodes in the Grid may force to split the application logic up among several nodes. In this scenario, inter-node communication is a requirement for both data exchanging and synchronization purposes.

Synchronization primitives (i.e. empty, fill,...) are treat as regular invocations by the RGrid platform. Off-FPGA buffer transmissions are supervised by the RGrid OMX IL run-time. As stated in the previous section, the RGrid IL run-time is signaled any time there is a new buffer to be *delivered* to the proper consumer. Analyzing the header information of the buffer structure and the internal data the RGrid IL agent has about the deployment status of a current application, it is determined whether the destination of the buffer is local (previous scenario) or external. Thus, the buffer is transmitted to the FPGA node holding the consumer component using the global communication infrastructure, placing the buffer in the shared memory. From this point, the process continues as in the HW$\Leftrightarrow$ case. Once again, from the consumer/producer perspective, all this operational is transparent to them.

VI. Conclusion

Large data set processing, as those that can be found in image or video processing applications, need of emerging architectures that would complement the important advances made in many-core architectures. Although *one-chip* solutions such as GPUs reach incredible performance rates, writing optimized code for one specific technology is a cumbersome task. The effort, thus, is neither portable or reusable. In addition, power consumption issues are a concern nowadays and GPU do not behave well in this field.

In this paper, we have presented the RGrid approach. RGrid is a distributed infrastructure and a set of service to integrate, manage and program reconfigurable logic resources. On one hand, the use of reconfigurable logic allows accelerated hardware implementations of image processing algorithms in an efficient way. On the other hand, the application of the grid computing paradigm enables coarse grain parallelism exploitation. A RGrid user only has to think in the application functionality, modeled as a set of components. Data distribution, component communication and deployment and, in general, all the platform dependant issues of the grid are hidden behind the proposed infrastructure.

Since the level of parallelism considered in RGrid is way above the one used in other approaches, applications are easier to architect. No ad-hoc code restructuring is needed and still comparable levels of performance and throughput are achieved due to the capability of using a massive replication scheme, transparent to the developer.

A fast hardware development workflow, closer to the profile of current developers in the computer graphics area, is also being explored in RGrid. The use of high level synthesis tools will make RGrid platform more popular and accessible to the community due to the possibility to write the accelerated portions of the application in C++.

Acknowledgment

This research was supported by the Spanish Ministry of Science and Innovation under the project DAMA (TEC2008-06553/TEC), and by the Spanish Ministry of Industry and the Centre for the Development of Industrial Technology under the project ENERGOS (CEN-20091048).

References

[1] *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays.* Springer, 2010.

[2] S. Asano, T. Maruyama, Y. Yamaguchi; "Performance comparison of fpga, gpu and cpu in image processing," in *International Conference on Field Programmable Logic and Applications(FPL)*, 2009, pp. 126–131.

[3] Barba, J., Rincon F., Moya F., Lopez J.C., Dondo J.D.; "Object-based communication architecture for system-on-chip design," in *Design of Circuits and Integrated Systems (DCIS)*, november 2010.

[4] L. T. Iona Technologies and S. Nixdorf, "Control and management of audio/video streams," in *OMG Doc. Telecom 97-05-07*, 1997.

[5] J. Barba, F. Rincon, F. Moya, F. Villanueva, D. Villa, J. Dondo, and J. Lopez, "Ooce, object-oriented communication engine for soc design," in *Proc. X Euromicro Conf. on Digital System Design (DSD)*, Germany, 2007.

[6] F. Rincon, F. Moya, J. Barba, F. Villanueva, D. Villa, J. Dondo, and J. Lopez, "Transparent ip cores integration based on the distributed object paradigm," in *LNEE- Intelligent Technical Systems*, vol. 38. Springer Netherlands, February 2009, pp. 131–144.

[7] "Openmax integration layer api specification," The Khronos Group, starndard 1.0, December 2005.

[8] J. Barba, D. de la Fuente, F. Rincon, F. Moya, and J. Lopez, "Openmax hardware native support for efficient multimedia embedded systems," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 3, pp. 1722 –1729, aug. 2010.