A Dynamically Adaptable Bus Architecture for Trading-Off Among Performance, Consumption and Dependability in Cyber-Physical Systems

J.Valverde, A.Rodriguez, J.Camarero, A.Otero, J. Portilla, E.de la Torre, T.Riesgo

I. INTRODUCTION

As technology evolves, physical and digital worlds are getting closer. New sensor and communication technologies together with the increasing capability of embedded systems to process big amounts of data are allowing Cyber-Physical Systems (CPSs) to grow and become more complex and suitable for a larger number of applications. As CPSs evolve, some problems inherent to technology are becoming important limiting factors. Power consumption, for instance, is becoming a key limitation [1]. At the same time, this fact is also affecting in terms of performance, since, even though the resources available per chip are increasing, the frequency of operation has stalled. Besides, as the level of integration is increased, it is more difficult to keep defect density under control, so new fault tolerant techniques need to be included. Taking into account the features of CPSs and the technology problems they are facing, it is clear that a more intelligent and dynamic use of resources is required.

In this work, a reconfigurable FPGA-based architecture embedded in an FPGA-based high performance wireless sensor node (called HiReCookie) is proposed. The HiReCookie node was already presented by the authors in [2] and [3], where its architecture, capabilities and very competitive energy consumption were probed. The node capabilities are combined with a Virtual Architecture, so called ARTICo^3 (Reconfigurable Architecture for an Intelligent Management of Consumption, Computation and Confidentiality, fault tolerance and security). This architecture permits the dynamic use of resources while using an execution model where tasks are accelerated, taking advantage of a multiple thread parallelization scheme similar to CUDA.

The paper is organized as follows. In section II, similar strategies studied in the state of the art are reviewed. Section III gives an overview of the problems addressed in ARTICo^3. In section IV, the ARTICo^3 architecture is detailed. In section V, the execution model is explained. In section VI, a real example to see some possibilities offered by the architecture is shown. Finally, in section VII, some conclusions are drawn.

II. STATE OF KNOWLEDGE

Many examples of CPSs can be found in the state of the art. In [2], some patterns and requirements are detailed. The authors mention applications related to distribute sensing and surveillance for crisis response, networked satellites, etc. The authors also highlight important features of these systems such as dynamic network adaptation according to the number of nodes, heterogeneous networks, location and time awareness, etc.

Dependability and computing capabilities are crucial when designing CPSs. Dependability can be understood as a composition of many features to provide the service the system was designed for, in a proper way. Reliability, confidentiality, safety, fault prevention, fault tolerance, etc. have been widely studied in the state of the art while the dynamic combination of them is not often addressed. When using SRAM-based FPGAs, DPR is a very suitable solution to mitigate Single Event Effects (SEEs). By means of using hardware redundancy, fault tolerant capabilities can be achieved since DPR can be used to replicate modules in a dynamic way [5] and to substitute them in case of damage [6]. In [7] different Triple Module Redundancy (TMR) implementations are studied injecting faults into the FPGA bitstream. Confidentiality and integrity of communications are also key aspects to guarantee a proper deliver of service. Different techniques have been already discussed in the literature such as hash or encryption algorithms like the ECC in [8]. Another factor of special importance in CPSs is side channel attack (SCA) protection. In this case, careful cyphering IP block designs must be done to mitigate this problem. Dual-rail approaches are the most widely used SCA mitigation techniques. It consists on replicating logic modules and making one of them to work in exactly reverse way to the first one, thus balancing and masking data dependent leakages.

Power consumption is another critical aspect in CPSs, especially in wireless systems. Traditionally, Wireless Sensor Network (WSN) platforms have included low performance microcontrollers that provided enough processing capabilities while they ensured ultra-low power consumption. However, when facing complex applications, these processors are not powerful enough, while power restrictions are still present. It is possible to find a myriad of different techniques in the state of the art ranging from energy harvesting proposals to others that take advantage of fast processing techniques together with low power modes to save as much energy as possible [3].

CPSs can be also quite demanding in a third factor, processing. Parallel processing speeds up data-intensive computations and has been widely analyzed throughout the literature over the last decades [9]. Communication-centric systems, especially those with multiprocessors, often use NoCs (Networks on Chip) in order to reduce data-transfer latency in streaming applications [9], e.g. image processing. However, bus-based architectures are still a feasible alternative in those systems that do not present restrictive scalability requirements. For instance, FLEXBUS [11] is presented as a dynamic topology that is capable of adapting its logic resources in order to optimize on-chip communications.

In this work, an architecture and methodology to include several of these strategies together, being capable to handle them in a dynamic way is proposed. The bus-based architecture proposed in this work dynamically adapts itself to a changing number of connected elements by means of a special unit that gathers all the requests and manages data transfers transparently.

III. Problem Overview

Embedded and distributed systems have big resource limitations in terms of power, fault tolerance and processing capabilities. This is a very restrictive factor since they are carrying out, in some cases, very intensive and critical tasks. Besides, in CPSs, working conditions are very likely to change over time but devices must continue delivering service with the available resources at all times. The application designer may not be aware of these changeable conditions, and hence the system must be capable of adjusting its resources transparently, and in real time. In the proposed solution, the biggest advantage is how the resources are organized and how data delivery is adapted to accelerate data transactions.

Power limitations can be seen from different perspectives. On the one hand, restrictions inherent to technology limit the frequency of operation and thus performance. Not everything can be working at the same time using the maximum number of resources with the available power budget. On the other hand, in some cases, these systems need to be wireless and so powered by batteries. In order to ease the effect of this limitation, the following techniques are included in the system design: divide the node into power islands, use a partial-initial configuration of the FPGA to reduce power consumption during wake-up time, use parallel processing to accelerate tasks and delegate tasks to other members of the network.

Fault tolerance and protection against Side-Channel Attacks are also critical aspects to take into consideration. CPSs are often working in critical environments where errors and faults are not tolerable. Thus, they must include certain techniques to be able to perform self-diagnosis, self-healing, fault isolation and protection against SCA. The proposed techniques included in the architecture are Double or Triple HW Module Redundancy (DMR or TMR), Dual-rail data delivery and processing to provide Side Channel Attack Protection (SCA), island isolation in case of failure and Dynamic and Partial Reconfiguration (DPR) to overwrite or move modules in damaged areas.

Applications with real time requirements and/or intensive data processing can take advantage of hardware acceleration. The strategies provided by the ARTICo^3 architecture are based on a CUDA-like execution model where parallel computations are divided in hardware kernels to accelerate execution. Besides, such nodes may be networked in order to define a model of computation based on a hybrid distributed processing model, where nodes may also be FPGA, CPU or GPU based.

IV. Virtual Architecture: ARTICo^3

The ARTICo^3 architecture is a general bus-based architecture that can be used not only for the HiReCookie platform (see details in [2]) but also be ported into any other platform where a dynamic trade-off among consumption, dependability and computation could be beneficial. The general idea is having a system able to adjust its resources in real time taking advantage of DPR features. The architecture adaptation is context-aware but application independent. ARTICo^3 is divided into two different regions: static and dynamic. The static region includes those modules that are not reconfigured in real time while the dynamic one hosts different hardware accelerators that can be replicated or multiplexed in time depending on the working needs.

A. Execution Model

ARTICo^3 follows an execution model similar to the CUDA approach, while the underlying hardware is completely different. In the CUDA model, many software processors, called Streaming Multiprocessors (SM), run potentially parallelizable tasks called kernels while these kernels are divided into segments called threads. These threads are grouped in blocks. Every different thread block is independent since it does not share data with other blocks. On the contrary, in ARTICo^3, a task can be executed with the help of a variable set of hardware accelerators, where each thread block runs in different copies of the same hardware block. The number of block accelerators can be changed dynamically, depending on resource availability and desired operation point. In the CUDA execution model, all data must be transferred from the host memory to the device memory prior to the kernel invocation. In this architecture, however, each processing element can start whenever each thread block has available its input data. Data is provided by a block called Data Shuffler (DS), coordinated with the Resource Manager RM, which makes coalesced access to memory in order to maximize the efficiency of burst accesses to memory.

Establishing an analogy between ARTICo^3 and CUDA execution models, potential parallelizable computations are tagged as kernels, whereas each concurrent execution of the same kernel will be known as a thread. The architecture concept is to adapt the CUDA execution model to ARTICo^3, keeping as many advantages as possible such as transparent parallelism mapping and resource allocation or even thread-level scalability, while changing completely the underlying hardware.

B. General Architecture

The architecture of the system is shown in Figure 1. As it can be seen, the architecture is composed of the following modules:

- Host (Microblaze): This is the embedded processor running the application code.
- Resource Manager (RM): It is the module in charge of finding the working point depending on both internal and external conditions. It will schedule the use of available resources to work within different operation modes.
- Data Shuffler (DS): This is the data dispatcher in charge of delivering data to the reconfigurable kernels.
- Kernel Wrappers: Every kernel is implemented within a common wrapper in order to be used with the ARTICo^3 architecture. The wrapper includes local memory, I/O registers, and an interface with the DS module.
- Interfaces with external components, HW_ICAP module and memory controllers.

C. Resource Manager

The RM is the brain of the architecture. It can be understood at two different levels. In the first level, the working point is calculated according to three coordinates: consumption, computation and dependability needs. The level that corresponds to each one of these coordinates depends on both external and internal conditions. In the case of the HiReCookie platform, these conditions could be the level of battery, current consumption per Power Island, radio messages, fault detections, etc. With these three coordinates, the solution scope resembles a surface within a cube where axes represent pure redundant solutions for fault tolerance, security and acceleration. According to this dynamic point, different operation modes are configured in the DS module.

In the second level, the RM works as a task scheduler organizing the tasks required by the host code. Which kernels are to be invoked, the number of blocks per kernel working in parallel, the number of threads per block or the amount of information delivered are decided by the RM in order to make an intelligent use of resources.

D. Data Shuffler

The shuffler unit is the interface between the reconfigurable blocks and the static region. The way data is sent and gathered depends on the operation mode configured by the RM. Due to the dynamic position of the blocks; the addressing is not done using their location but a specific identifier of each kernel. In this way, the location of every block in the reconfigurable area is arbitrary, so if their identifier is the same, they are considered part of the same kernel. In order to save time when addressing kernels, both the identifier and the internal address are coded in the bus address. In this way, it is possible to include any data transaction in only one bus command from the RM. The DS unit can be configured to work in six different operation modes depending on the needs of the system:

Mode 0: ***Replica Single***, in this mode several replicas of the same block are addressed in parallel (blocks having the same ID). The voter included in the DS module votes the results and gives only the correct one. An error code is stored in the status register in case discrepancies are found.

Mode 1: ***Replica Single for SCA Protection***, when SCA protection is required, a kernel and its negative copy are loaded with the same ID so data are sent exactly at the same time and noise effects can be masked.

Mode 2: ***Replica Burst***, in this mode, different blocks are addressed in parallel. Unlike the previous two modes, burst data are sent to all of them. Data transactions are now performed through the data bus.

Mode 3: ***Replica Burst for SCA Protection***; this mode is a combination of modes 1 and 2.

Mode 4: ***Block Interleaved Burst***, burst data is sent to different blocks. Depending on the selected block size, a different number of words are sent to each block. This mode permits parallel-processing capabilities. Even though data are not delivered in parallel but in a cascaded way, the overlapping of several processing elements can be achieved to accelerate execution.

Mode 5: ***Reduction mode***, data are sent to the processing elements in the same way as in mode 4. Whenever parallel processing is carried out, blocks may provide only partial results e.g. MAC operations. Therefore, when a read-reduction command is performed, these partial results are stored in a global memory attached to the DS module where different operations can be performed with them.

E. Kernel Wrapper

Kernel logic must be placed inside a special wrapper in order to be used within the architecture. The wrapper consists of a communication interface with the DS module, some input/output registers and internal BRAM blocks. The general idea is that a thread block of a single kernel is placed in a reconfigurable slot so they can be changed or replicated whenever is required. In case a required kernel is not present in the local library, the system can ask for remote transfer from any other member of the network. As it was mentioned before, every block has an ID register. This ID is the name the RM uses to invoke the kernel. If several blocks share the same ID, the architecture understands that data must be sent to all of them. The content of this register can be edited from the RM so kernels can have a dynamic ID.

V. Results and Examples

In order to demonstrate how ARTICo^3 can improve the performance of many applications while providing a better use of their resources, this section shows an example on the implementation of an application case with the HiReCookie Platform. The use case is the Eigenfaces-face recognition algorithm has been implemented by the authors in [12].

The Eigenfaces face recognition algorithm is divided in two main stages. In the first stage, a training image set is used both to build an orthogonal basis of the face subspace and to define an average face. This process is usually carried out offline and hence, it will not be discussed here. The second stage takes an input image, removes the average face and projects it onto the face

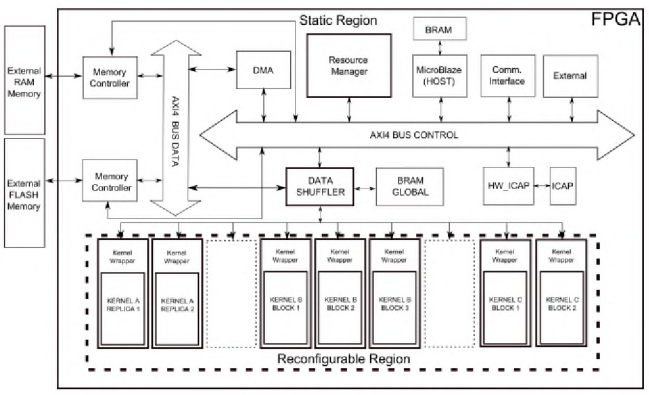

Figure 1: ARTICo^3 Architecture

subspace using the basis obtained in the previous stage. Then, the Euclidean distance between the projection and the references is calculated. This process is performed online and therefore, hardware acceleration capabilities must be used in order to reduce computing time. An important limiting factor when loading different blocks of a kernel dynamically is reconfiguration speed. In this example, the recognition algorithm is included in a Spartan 6 FPGA, so it is better to include the previous tasks on a single kernel and replicate its blocks so it does not need to be reconfigured unless the operation mode needs to be changed. Therefore, in the proposed solution, the kernel that includes the recognition algorithm is divided into 8 blocks (each block is configured in a clock region of the Spartan 6).

To make a comparison with other implementation targets, the algorithm has been implemented in three different platforms with different configurations and processors. Some results can be found in Table 1. The objective of this analysis is to compare performance, price and energy consumption among different approaches. The conclusions are summarized as follows:

- Computing time: as it was expected, GPUs provide the fastest solution followed by the HiReCookie with ARTICo^3. The improvement by using the architecture can be seen comparing the solution with the single hardware accelerator, where time is reduced over 20 times.
- Energy consumption: although computation time is 6 times slower than in the case of the Tesla GPU, energy consumption is 20 times lower in the case of ARTICo^3.
- Price: the price of the Spartan 6 is comparable to the rest of the platforms except in for the Tesla GPU, which is ten times higher.

VI. Conclusions and Future Work

The concepts of urgency, confidentiality, fault tolerance or priority for a task are widely known and have been applied to computing, as well as being addressed many times for research, but most of them as isolate objectives. However, this work has shown that the replication of modules, together with the adequate and efficient provision of data between the accelerators and memories, can serve all these purposes in a very similar manner. Acceleration with parallel execution of threads is a well-studied model of computation, and there are tools and models, like CUDA, which may be used as a common specification entry for GPU platforms as well as for this type of HW accelerated architectures. The ARTICo^3 architecture permits to implement a parallel kernel invocation method, which would launch parallel thread blocks in a variable number of accelerators as they are progressively being reconfigured. Dynamic security and dependability levels are also possible. In a varying context scenario, tasks that are not considered critical might become (or stop being) critical under some unforeseen conditions, so the use of dynamic DMR or TMR, is also an advantage. A face recognition algorithm has been implemented in different technologies in order to compare performance, price and energy consumption, with satisfactory results. Although GPUs are faster, ARTICo^3 offers a comparable solution in terms of computing time, while consumption is lower.

Table 1: Execution Implementation Comparison

Device	Model	Energy (J)	Power (W)	Price (€)	Execution Time (ms)
Laptop	CPU 2.20 GHz i7-2670QM	0.855	45	120	19
	GPU 1.33 GHz GeForce GT 540M	0.147	35	92	4.2
Desktop PC	CPU: 2.80 GHz i7-870	2.550	170	140	15
	GPU 1.15 GHz Tesla C2075	0.500	238	1470	2.1
HiReCookie (Spartan6 FPGA)	CPU 100 MHz MicroBlaze	10.480	1.31	116	8000
	Single HW Accelerator	0.047	1.35	116	350
	ARTICO^3	**0.021**	**1.75**	**116**	**12.2**

In conclusion, the proposed architecture opens up a scenario where trade-off decisions at system level can be easily ported into HW, at run-time. The application specification requires explicit parallelism declaration, as for CUDA, and an API for memory transfers, kernel invocation and synchronization (as for CUDA, also), and data reduction, permits easy porting of multi-thread models. This API, as well as an enhanced RM, are future lines of work.

VII. Acknowledgment

This work was supported by the Spanish Ministry of Economy and Competitiveness under the project DREAMS (Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems) with number TEC2011-28666-C04-02.