Execution Modeling in Self-Aware FPGA-Based Architectures for Efficient Resource Management

Alfonso Rodríguez, Juan Valverde, César Castañares, Jorge Portilla, Eduardo de la Torre and Teresa Riesgo

Abstract—**SRAM-based FPGAs have significantly improved their performance and size with the use of newer and ultra-deep-submicron technologies, even though power consumption, together with a time-consuming initial configuration process, are still major concerns when targeting energy-efficient solutions. System self-awareness enables the use of strategies to enhance system performance and power optimization taking into account run-time metrics. This is of particular importance when dealing with reconfigurable systems that may make use of such information for efficient resource management, such as in the case of the ARTICo^3 architecture, which fosters dynamic execution of kernels formed by multiple blocks of threads allocated in a variable number of hardware accelerators, combined with module redundancy for fault tolerance and other dependability enhancements, e.g. side-channel-attack protection.**

In this paper, a model for efficient dynamic resource management focused on both power consumption and execution times in the ARTICo^3 architecture is proposed. The approach enables the characterization of kernel execution by using the model, providing additional decision criteria based on energy efficiency, so that resource allocation and scheduling policies may adapt to changing conditions. Two different platforms have been used to validate the proposal and show the generalization of the model: a high-performance wireless sensor node based on a Spartan-6 and a standard off-the-shelf development board based on a Kintex-7.

Index Terms—**Self-awareness, dynamic and partial reconfiguration, dynamic resource management, FPGAs.**

I. Introduction

Hardware accelerators are used to speed-up data-intensive tasks, taking advantage of data-level parallelism and dedicated hardware design. However, one of their main disadvantages is that the number of available resources is usually limited. This leads to systems that, in general, lack the flexibility that software-approaches provide. Dynamic and Partial Reconfiguration (DPR) of FPGAs may alleviate this problem by resource multiplexing in time, so that the amount of such resources is virtually unlimited, but restricted in every moment by the total amount available in the device [1]. In a context where applications increase their complexity continuously, FPGA-based systems need to adapt to more demanding requirements. As a result, the complexity of almost any strategy used to achieve optimal resource allocation surges, specially if not only external but also internal conditions must be taken into account.

Efficient resource management policies require both execution modeling and real-time measurements to provide accurate estimations and predict the future behavior of the system. If a specific set of system metrics is monitored in real-time by the system itself, i.e. making it self-aware during execution, the acquired knowledge can be used to guide both resource allocation and task scheduling within the FPGA towards optimal solutions [2]. Moreover, modeling the behavior of the system during execution provides a framework in which decisions to move tasks from one operating point to another in the solution space can be made.

In this paper, execution modeling has been implemented in ARTICo^3 [3], a multi-platform virtual architecture that allows the execution of multikernel and multithread applications by using DPR to change hardware accelerators according to these model-defined requirements. The dynamic use of resources and operation modes applied to move the working point around the solution space is highly dependent on the targeted platform. Therefore, the task scheduler of the architecture must be aware of the possibilities offered by the platform in order to achieve a proper resource distribution. This flexibility, within a low-power and high-performance context, permits the application of scheduling strategies similar to the methodologies used by operating systems. In any case, both the way resources are scheduled and the operation mode used are completely transparent to the application code running in the host processor.

The rest of this paper is organized as follows: section II shows an overview of the state of the art, section III presents the features of the architecture on which the model is based. The proposed model is thoroughly covered and analyzed in Section IV; Section V shows the experimental results that confirm the validity of the model as well as its generalization, and Section VI provides the conclusions and the future work.

II. Related Work

As it was already introduced, reconfigurable hardware architectures require a dynamic use of resources, typically enabled by DPR. Some bus-based examples can be found in the literature: in [4], the authors prosed a bus-based architecture, so-called FLEXBUS, to adapt the connectivity and mapping of different components by dynamically adapting the communication topology using a bridge by-pass. Thus, they can create

a network of bus segments connected by these bridges. In a context where different hardware accelerators are changed at run time to work as co-processors, being able to perform a proper task scheduling is a must. In [5], the authors present a dynamically adaptable architecture that allows the inclusion of an OS kernel for management tasks. Crucial aspects such as reconfiguration speed, critical when implementing strategies for task scheduling in dynamically reconfigurable architectures, are addressed. Dynamic changes do not only affect logic blocks but also memory resources. Moreover, the necessity of dynamic task scheduling on these platforms has also been covered [6]. In [7], an adaptive architecture for dynamic management and allocation of on-chip FPGA Block RAM memory (BRAM) resources is presented to deal with changing memory footprints of different reconfigurable modules. One of the main goals of these architectures is providing a framework to ease the dynamic reconfiguration process. In [8], the authors propose a technique called I/O bars to provide dynamically-changing point-to-point connections from the static area of the FPGA to some reconfigurable modules apart from the ReCoBus-builder to assist partial run-time reconfiguration. In the ARTICo^3 framework, the dynamic use of the available resources is improved by the combination of two methodologies: execution modeling (power consumption) and task profiling (performance and power consumption).

Since power consumption is considered one of the main pitfalls of SRAM-based FPGAs, due to the fact that the static contribution is significantly larger than in other devices, power modeling appears as a mandatory component in almost any solution in which energy efficiency is a requirement. Therefore, power models focusing on different levels have been proposed. In [9], for instance, a power model at device level is presented. The model is then used to predict the impact of algorithm implementations on FPGA devices. It is widely known that DPR can be used to reduce the overall power consumption in the device, since it allows time-multiplexing of resources and therefore, designs can be implemented in smaller FPGAs (with less static power consumption) and dark-silicon areas are minimized. Hence, some works present models with different granularity levels to predict the power consumption during the reconfiguration process and use that information to feed an energy estimator, as in [10].

Moreover, models to estimate the power consumption of hardware modules in FPGAs have been widely studied, mostly to predict and to try to reduce it at design stages. In [11], the authors present a high-level technique to estimate power consumption for arithmetic hardware blocks such as polynomial evaluations with very close results to the ones provided by Xilinx XPower estimator. Besides, not only arithmetic blocks but also major event signatures such as buses or memory transactions need to be studied. In [12], these two elements are included together with a study of the variability of the power consumption according to LUT numbers and a comparison with real measurements in Virtex 5. However, very few works address on-line power management and profiling in order to dynamically adapt hardware resources, which is one of the main contributions of the ARTICo^3 framework.

Application profiling has also been widely covered in the literature. However, most profilers focus on software implementations, i.e. performance or power consumption is profiled either by analyzing the program counter or the instruction bus of the processor. An example of this type of approach is SnoopP [13], a profiling framework that monitors the program counter of the CPU for improved HW/SW codesign. A similar approach is used in LEAP [14]. Software profiling has been successfully applied in systems with dynamic adaptation capabilities, as it has been shown in [15] and [16].

Although software profilers are more common, some examples that explore performance profiling in hardware cores can be also found in the literature. For instance, in [17], a framework to help designers debug their applications is presented. The proposed approach inserts monitoring cores and provides software drivers to ease application profiling so that focus can be put on the application itself instead on the profiling infrastructure.

Dynamic resource managers are also common elements in reconfigurable computing, as it can be seen in [18], where the authors propose a smart reconfiguration methodology targeting application performance. The ARTICo^3 approach, on the other hand, focuses its optimization capabilities not only on execution performance but also on energy efficiency by taking into account the aforementioned elements: modeling and profiling.

According to [19], there are three main cornerstones in multiprocessor systems: hardware architectures, design tools and management of runtime adaptations. The ARTICo^3 framework already provides solutions at architectural level by defining the processing elements, the memory structure and the communication infrastructure. The work presented here establishes the foundation for handling other of the main cornerstones: adaptation in real time by means of self-awareness through execution modeling.

III. Architectural Support

ARTICo^3 is a bus-based virtual architecture which features a set of reconfigurable slots suitable for SRAM-based FPGAs (in order to take advantage of DPR). Any combination, in number and position of reconfigurable slots can host hardware accelerators for a specific task, and a dispatcher unit is in charge of delivering data to and collecting results from them, accordingly to the configuration but independently from the application. A reconfiguration engine is used to swap different hardware accelerators, and thus achieve the aforementioned time multiplexing of resources. This architecture has already been presented in [3], and its Model of Computation (MoC), together with the basics for a Dynamic Resource Manager, were presented in [20].

Although ARTICo^3 has several internal modules and features, each one with its own functionality, only those needed to fully understand the model foundations are discussed in this section.

A. Data Transactions

In ARTICo^3, the data dispatcher unit is called data Shuffler. It acts as a bridge between the reconfigurable slots and the rest of the system, i.e. the static region. The data Shuffler interfaces the rest of the system using a standard connection to the shared bus where all peripherals are. The architecture also features a dedicated bus for large data transactions, which are done in bursts using a Direct Memory Access (DMA) engine for two main reasons: on the one hand, they are less time-consuming and the bus can benefit from having more data transfers using the same amount of time; on the other hand, DMA accesses are much more energy-efficient.

The way of interfacing with the different reconfigurable slots changes depending on the operating mode in which the architecture is working, and on the direction data are being transfered (write to or read from the slots). In modes that use hardware redundancy, data are forwarded to the reconfigurable slots in multicast fashion, and retrieved through a voter unit that merges different paths and enables fault tolerance through module redundancy. In high performance modes, each reconfigurable slot is enabled during part of the whole data transfer. Hence, data are read from or written to the slots only when they are enabled, being the data Shuffler in charge of the sequencing process. Furthermore, the built-in data reduction engine can be enabled, taking advantage of data serialization in the bus, i.e. using less hardware resources, to perform a specific operation (e.g. add, maximum, minimum, or comparison with fixed value) on the data that are being read from the accelerators.

B. Model of Computation

The architecture provides a CUDA-like MoC, in which program sections with explicit data-level parallelism (called kernels) are split into different execution threads. These threads are grouped into thread blocks. Blocks cannot have data dependencies, but threads within the same block can. This is an important feature that allows transparent scalability in kernel execution: depending on the number of available resources, blocks can be executed in parallel or sequentially, since the obtained result is the same.

In ARTICo^3, kernels define a functionality, which is then implemented as data-independent thread blocks in the shape of hardware accelerators. The number of available slots, i.e. slots that are idle or yet to be loaded by the reconfiguration engine, determines the amount of thread blocks that can be executed at a given instant. The execution depends on the mode on which the architecture is working: when there are fault tolerance or security requirements and data are delivered strictly in parallel, the execution is also purely parallel in those blocks with hardware redundancy (for voting purposes or even for side channel attack protection), whereas in those situations following a SIMD-like approach, where thread blocks operate with different data, the execution is overlapped between accelerators (due to data serialization through the bus that links the memories and the data Shuffler). The larger the overlapping is, the higher the acceleration.

Kernel invocation involves data transferences to all its thread blocks, which are addressed sequentially (high performance modes), in parallel (hardware redundant modes), or using a hybrid approach (by combining both strategies) in the same DMA transfer. Each hardware accelerator starts its execution as soon as it has received all its input data. Hence, the effects of data serialization are mitigated and the aforementioned features, i.e. parallelism or overlapping, are achieved.

IV. System Modeling

In order to provide the architecture with self-aware capabilities, it is necessary to first model the different stages of the operation cycle. This model will be used by the Dynamic Resource Manager of the architecture to generate efficient allocation and scheduling policies in different kernel invocations, taking into account real-time execution metrics such as power consumption, elapsed times, battery level in portable devices, or requirements in terms of fault tolerance to allow a maximum number of faults per transaction for instance. The normal operation cycle of a thread block consists of three stages: reconfiguration of the hardware accelerator (whenever it is required), data transfers and kernel execution. The first stage will be referred to as the reconfiguration model, whereas the second and the third ones will be assembled in what will be called the transaction model.

A. Reconfiguration Model

Since the reconfiguration engine in ARTICo^3 is software-based, reconfiguration performance shows several limitations. In particular, bitstream composition to perform module relocation, together with the fact that configuration frames are read word by word from the memory, generate a large overhead in reconfiguration times. Experimental results in different platforms confirm these problems: in Zynq devices, only bitstream composition affects, since the transference of configuration data is done using a DMA transfer through the Processor Configuration Access Port (PCAP); in FPGA-only devices, on the other hand, the two problems are present when using the standard reconfiguration engine to access the Internal Configuration Access Port (ICAP).

Moreover, as reported in [10], power consumption during reconfiguration processes depends on several factors, such as the previous configured logic and the module that is being loaded at that moment.

All these factors favor the usage of a simplified model, in which the power consumption in the FPGA core equals the peak value during reconfiguration. In order to keep this value at a minimum, clock gating techniques are applied to each slot that is being loaded with a hardware accelerator, so that the clock is inactive during reconfiguration. The power consumption in the external memory, on the other hand, is almost equal to its static value, due to the aforementioned inefficient data readings.

B. Transaction Model

As opposed to reconfiguration processes, where the behavior cannot be easily modeled, power consumption during kernel

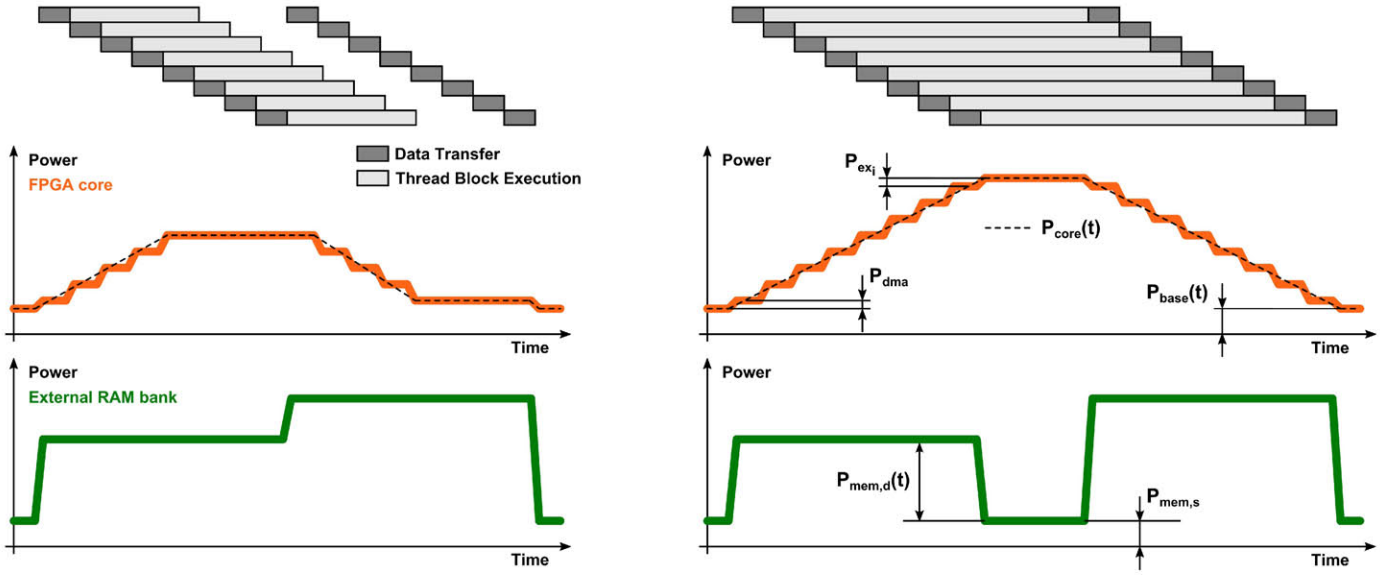

Fig. 1. Transaction model in high performance modes. Thread block execution can be memory-bounded (left), or computing-bounded (right). The dashed line represents the simplified power consumption model in the FPGA core.

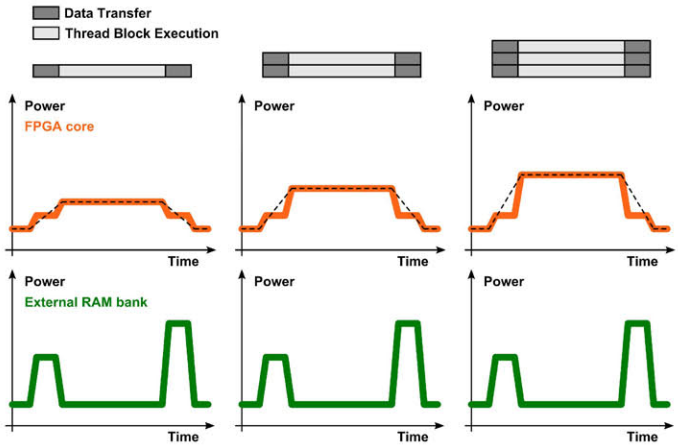

Fig. 2. Transaction model when adding hardware redundancy. The contribution due to data transfers is the same in all three cases, whereas dynamic power consumption due to thread block execution increases over the reference value (one thread block, left) when adding DMR (two thread blocks, center) and TMR (three thread blocks, right).

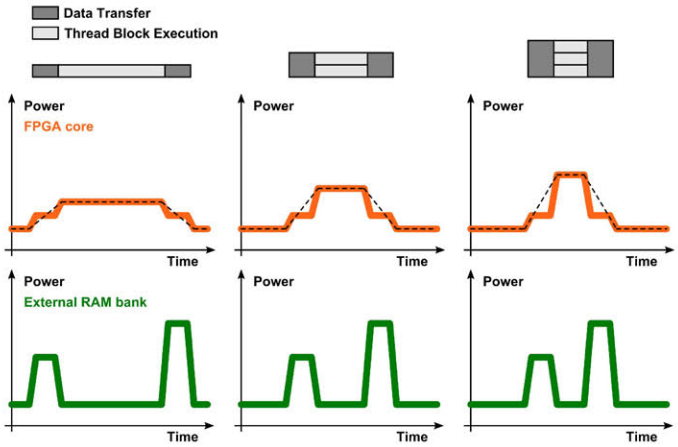

Fig. 3. Transaction model when using multithreading within a thread block. The contribution due to data transfers is the same, whereas dynamic power consumption due to thread block execution increases when adding more threads. However, since threads within a thread block execute in parallel, the execution time is decreased, thus reducing the impact of static current on the overall energy consumption.

execution in ARTICo^3 can be modeled using the FPGA core and the external RAM memory contributions. This can be done because power consumption in the other power rails has been proved to be constant during the whole process.

The contribution of the FPGA core to the power consumption model is expressed as follows:

$$P_{core}(t) = P_{base}(t) + P_{dma}(t) + \sum_{i=1}^{n_k} n_{ex_i}(t) P_{ex_i}(t) \quad (1)$$

$$P_{base}(t) = P_{static}(t) + \sum_{i=1}^{n_k} n_i(t) P_i \quad (2)$$

P_{core} represents the power consumption of the FPGA core at a given instant; P_{base} is the power consumption of the system when no kernel is being executed, which might change since it depends on the number of slots that are loaded with hardware accelerators and the activity in the static region; P_{dma} is the dynamic contribution to the power consumption due to DMA data transactions inside de FPGA; n_k is the number of kernels loaded in the reconfigurable slots; n_{ex_i} is the number of thread blocks of kernel i being executed at a given instant; and P_{ex_i} is the dynamic contribution to the power consumption due to one thread block of kernel i being executed.

Equation (2) shows how P_{base} is computed when the system

has n_k kernels, each of them with n_i thread blocks loaded in the reconfigurable slots. P_{static} is the contribution of the static region, i.e. the power consumption of the whole system without any hardware accelerator, whereas P_i is the contribution of one thread block of kernel i loaded in one reconfigurable region by the mere fact of being there. Note that P_{base} does not represent the static power consumption of the FPGA and that, in some cases, n_{exi} might not be equal to n_i.

The contribution of the external RAM memory to the power consumption model is expressed as follows:

$$P_{mem}(t) = P_{mem,s} + P_{mem,d}(t) \tag{3}$$

P_{mem} represents the power consumption of the external RAM memory; and $P_{mem,s}$, $P_{mem,d}$ are the values of static and dynamic power consumption in the external memory power rail respectively.

Therefore, the overall power consumption in the ARTICo^3 system can be approximated taking into account these two contributions with the following expression:

$$P(t) = P_{core}(t) + P_{mem}(t) \tag{4}$$

In order to make an efficient implementation in the Dynamic Resource Manager, the model can be simplified by making the following assumptions:

1) Equations (1), (2) and (3) are discretized using sampling time T_s, to account for the self-measuring Analog to Digital Converter (ADC) circuitry.
2) Thread blocks have uniform power consumption during their execution.
3) DMA transfers and memory accesses have uniform power consumption.

Normal system operation in high performance modes, i.e. with execution overlapping, can find bottlenecks in two different stages: memory accesses (and the associated DMA data transfer through the bus), and thread block execution. Depending on the limiting factor, the execution can be classified as memory-bounded or computing-bounded.

- Memory-bounded execution: shows high bus occupancy, since the DMA engine is always busy transferring data. One or more thread blocks have finished their processing and are idle waiting to be read.
- Computing-bounded execution: generates reduced bus occupancy and memory usage due to idle times between data transferences. Thread blocks continue processing data after the transfer has finished, and therefore show no idle times.

These two possibilities are covered by the proposed model, as it can be seen in Fig. 1, where an example invocation with overlapped execution of eight thread blocks of the same kernel is shown. Notice the idle times in the FPGA core for memory-bounded execution, and in the external memory for computing-bounded execution. Also notice the opposite behavior in the other power rail: continuous power consumption in the external memory for memory-bounded execution and no idle times in the FPGA core consumption for computing-bounded execution.

The proposed model is also applicable in scenarios where hardware redundancy exists. For instance, Fig. 2 shows the impact of adding more copies of the same thread block to perform Double Module Redundancy (DMR) and Triple Module Redundancy (TMR). The data transfer component of the power consumption (P_{dma}) remains the same, whereas the number of thread blocks of the given kernel (n_{ex_i}) being executed is increased, leading to a proportional increase in terms of overall power consumption.

To model all the features ARTICo^3-enabled execution provides, the model can also be extended to multithread execution of a single, i.e. only one, thread block, as shown in Fig. 3. In this case, adding more threads generates not only an increase in the power consumption of the thread block, but also a decrease in its execution time due to parallelization, which in turn implies a more energy-efficient execution, as it has been proved by experimental results.

Therefore, all capabilities of the architecture fit in the proposed model, since any kernel execution can be characterized by a combination of three different features: execution overlapping of thread blocks, hardware redundancy of a thread block, and multithreading in a single thread block.

V. Experimental Results

The experimental setup uses two different platforms to validate the model: the HiReCookie high-performance wireless sensor node (Spartan-6, XC6SLX150-2FGG484) [21], and the KC705 evaluation board from Xilinx (Kintex-7, XC7K325T-2FFG900). Both platforms have built-in circuitry to measure power consumption in different power rails. In addition, the ARTICo^3 implementations on these platforms include a timer to measure elapsed times during reconfiguration and kernel execution.

A. Reconfiguration Model

Results obtained during the reconfiguration stage in both platforms are summarized in Table I, where f_{clk} is the system clock frequency (in both static and dynamic regions) and f_{icap} is the ICAP clock frequency. Technology limitations impose significantly smaller values in the latter for Spartan-6 FPGAs. This, together with the 16-bit ICAP datapath in those devices (as opposed to the 32-bit ICAP datapath in 7-Series FPGAs), decreases reconfiguration throughput. Moreover, the software-based approach leads to low performance ratios between measured and theoretical results in terms of throughput, as it was previously stated in Section IV. The reported values of power and energy consumption are higher in the KC705 development board since ARTICo^3 slots occupy much more logic resources in that platform than in the HiReCookie node (this values are also reflected in Table I).

B. Transaction Model

A reduced set of kernels has been implemented in order to test the architecture and the accuracy of the proposed model.

TABLE I
RECONFIGURATION OVERHEADS PER SLOT

Parameter	Platform	
	KC705	HiReCookie
f_{clk} (MHz)	100	100
f_{icap} (MHz)	100	20
Slot Size (kB)*	505	124.67
Theoretical throughput (MB/s)	400	40
Reconfiguration throughput (MB/s)	5.15	3.64
Performance ratio (%)	1.28	9.1
Power consumption (mW)	596.87	290.34
Reconfiguration time (ms)	95.7	33.46
Energy (mJ)	57.12	9.71

* Configuration memory footprint of an ARTICo^3 slot.

TABLE II
MODEL PARAMETERS: AES256 CTR KERNEL

Parameter	Value (mW)	
	KC705	HiReCookie
P_{dma}	6.93	5
P_i	38.66	44.55
P_{ex_i}	31.57	22.21
$P_{mem,s}$	792	91.6
$P_{mem,d}$ (read)	768	133.4
$P_{mem,d}$ (write)	1368	101.25

This library has kernels with both memory-bounded execution (Sobel and median filters) and computing-bounded execution (AES and SHA-2). Actual measurements in the power rails using an oscilloscope confirm the validity of the proposed approach, as well as the differences between these two types of kernel executions. Fig. 4 shows the behavior of the median filter kernel, which is memory-bounded, in the HiReCookie platform. Fig. 5, on the other hand, shows the behavior of the AES256 CTR kernel, which is computing-bounded, in the KC705 development board. Notice that the obtained results resemble what was predicted by the model in Fig. 1.

The values of the parameters that feed the model vary from one kernel to another. Table II shows the characterization of an AES256 CTR kernel in both platforms, which has been obtained from different runs. Each of these runs differ in only one parameter, e.g. number of thread blocks loaded (to compute the power consumption of one hardware accelerator that is idle), or number of thread blocks used (to compute the dynamic power consumption of one hardware accelerator during execution).

Once the kernel has been characterized, the model has been verified with real measurements from the two platforms. Fig. 6 shows the comparison between the real measurements and the estimations provided by the model in the KC705 board. Note that, in order to obtain similar execution times, input data sizes increase at a rate of 32kB per additional thread block. Therefore, each block has to process the same amount of data in all runs. Otherwise, the execution times would have decreased when adding more thread blocks due to execution overlapping, as expected in the model. Fig. 7 gives a better view of the power consumption in the FPGA core, and shows that the execution becomes more energy-efficient when increasing the number of thread blocks and exploiting overlapping. Moreover, note that the static power consumption represents roughly 60% of the overall value.

The accuracy of the model can be analyzed taking into account the correlation between the measurements and the estimated values. Table III sums up the obtained values in both test platforms. Notice that the model provides significantly worst results when modeling executions of only one thread block. This is mainly due to the minimum increase in the power consumption in those situations, a behavior that has to be taken into account by the Dynamic Resource Manager when profiling new kernels. The HiReCookie can only use up to 4 thread blocks of the AES256 CTR kernel, and thus no correlation values are provided for 5 and 6 thread blocks. In addition, and since the circuitry of the power rails has larger capacitors, the correlation values are slightly lower in the HiReCookie node than in the KC705 board.

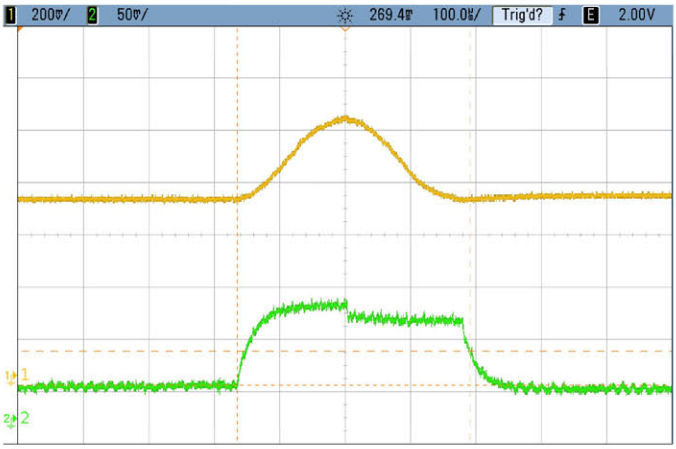

Fig. 4. Memory-bounded kernel execution in ARTICo^3 on the HiReCookie node. Notice that the external RAM memory has larger power consumption in read than in write operations.

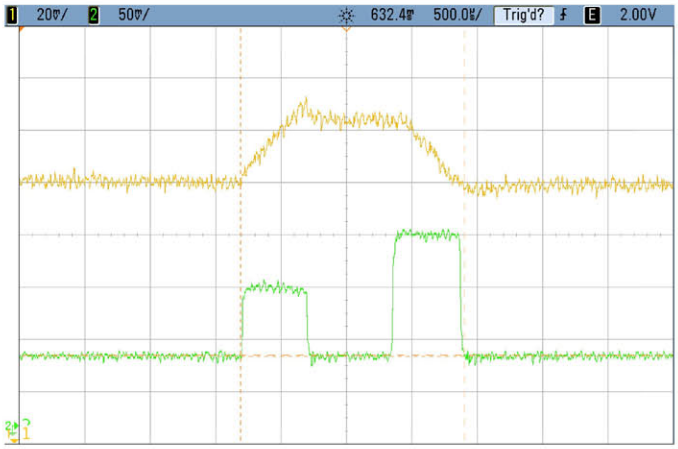

Fig. 5. Computing-bounded kernel execution in ARTICo^3 on the KC705 board. Notice that the external RAM memory has larger power consumption in write than in read operations.

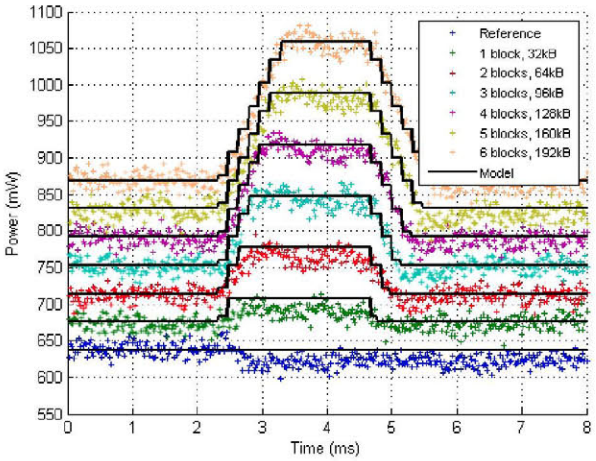

Fig. 6. Close-up of the power consumption model in the KC705 development board executing an AES256 CTR kernel with different number of thread blocks and input data sizes.

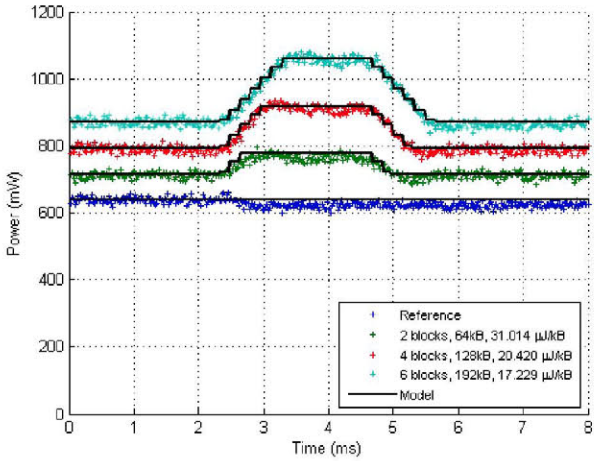

Fig. 7. Overall power consumption of an AES256 CTR kernel in the KC705 development board. Note that the energy efficiency is increased when the number of thread blocks is increased.

TABLE III
MODEL ANALYSIS: AES256 CTR KERNEL

Thread Blocks	Correlation Model-Measurements	
	KC705	HiReCookie
1	0.7177	0.4990
2	0.9174	0.8971
3	0.9644	0.9453
4	0.9795	0.9628
5	0.9790	-
6	0.9860	-

The model suffers deviations when performing kernel invocations from different background states, as it can be seen in Fig. 8, where the same execution provides measurements that differ in P_{base}. The model provides the same results in both situations, since it assumes the same power consumption in the static region. Hence, the Dynamic Resource Manager has to ensure that the reference value in the power consumption, i.e. P_{base}, is updated before profiling a new kernel.

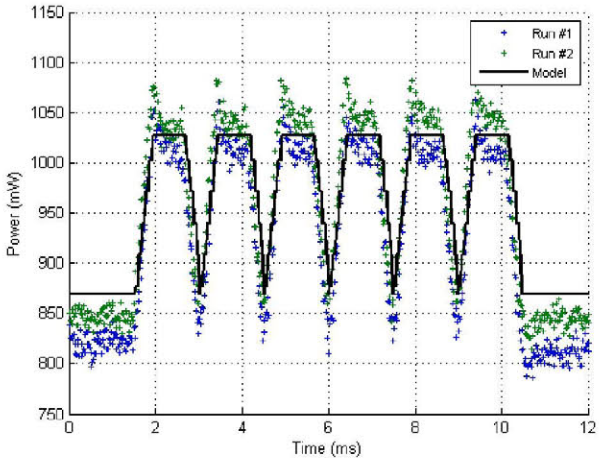

Fig. 8. Comparison between the power consumption model and two different runs in the KC705 development board of an AES256 CTR kernel with 6 thread blocks, 2 threads per block and 1MB input data.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, a simplified yet accurate model of a kernel being executed on ARTICo^3 has been proposed. The model takes into account that kernel execution might involve a combination of three different features provided by the architecture: overlapped execution of thread blocks, parallel execution of redundant thread blocks, and parallel execution of multiple threads within a thread block.

The model has been validated with actual measurements in two different platforms, the HiReCookie node and the KC705 development board, and with a set of kernels that show different behavior during their execution (memory-bounded and computing-bounded). Moreover, an AES256 CTR kernel has been characterized and then analyzed in different scenarios, showing the accuracy of the proposed approach.

The power consumption model, together with the execution and reconfiguration times are powerful tools that the Dynamic Resource Manager of the architecture has to take into account when deciding which kernels are to be executed, or how many thread blocks have to be placed in the reconfigurable slots. Allocation and scheduling policies have to be made analyzing both execution times and energy, specially in platforms with limited resources such as the HiReCookie node. If the speed-up achieved by adding more thread blocks is significantly higher than the increase in terms of power consumption, the system will achieve a more energy-efficient execution, at least up to the point in which execution becomes memory-bounded.

At any rate, reconfiguration must be taken into account, since energy savings obtained by using more computing resources working concurrently might be negligible compared to the amount of energy spent during reconfiguration. This can

be seen, for instance, in the KC705 development board when comparing the time and energy spent loading two AES256 CTR thread blocks (191.4 ms, 114.24 mJ) with the time and energy spent processing 64 kB of raw data (2.68 ms, 1.98 mJ) with those accelerators. Therefore, for small amounts of data there is no overall energy gain. This can be solved either by increasing the amount of data to be processed (in the aforementioned example, moving from one round of 64 kB to hundreds of rounds) or by optimizing the reconfiguration engine, since the reconfiguration throughput does not provide energy-efficient load or switch operations between hardware accelerators. The current work is focused on optimizing the reconfiguration engine so that the overhead due to the reconfiguration process is almost equivalent to the one due to the kernel execution. By achieving that, an additional overlapping, in this case between reconfiguration and execution, is to be implemented.

ACKNOWLEDGMENTS

The authors would like to thank the Spanish Ministry of Education, Culture and Sport for its support under the FPU grant program.

This work was also partially supported by the Spanish Ministry of Economy and Competitiveness under the project REBECCA (Resilient EmBedded Electronic systems for Controlling Cities under Atypical situations), with reference number TEC2014-58036-C4-2-R.

REFERENCES

[1] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Sept 2008, pp. 535–538.

[2] C. Gomez Osuna, M. Sanchez Marcos, P. Ituero, and M. Lopez-Vallejo, "A monitoring infrastructure for fpga self-awareness and dynamic adaptation," in *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, Dec 2012, pp. 765–768.

[3] J. Valverde, A. Rodriguez, J. Camarero, A. Otero, J. Portilla, E. de la Torre, and T. Riesgo, "A dynamically adaptable bus architecture for trading-off among performance, consumption and dependability in cyber-physical systems," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–4.

[4] K. Sekar, K. Lahiri, A. Raghunathan, and S. Dey, "Flexbus: a high-performance system-on-chip communication architecture with a dynamically configurable topology," in *Design Automation Conference, 2005. Proceedings. 42nd*, June 2005, pp. 571–574.

[5] S. Garcia and B. Granado, "Ollaf : A dual plane reconfigurable architecture for os support," in *Design and Test Workshop, 2008. IDT 2008. 3rd International*, Dec 2008, pp. 282–287.

[6] I. Ktata, F. Ghaffari, B. Granado, and M. Abid, "Prediction performance method for dynamic task scheduling, case study: the ollaf architecture," in *Design and Test Workshop (IDT), 2010 5th International*, Dec 2010, pp. 97–102.

[7] G. Dessouky, M. Klaiber, D. Bailey, and S. Simon, "Adaptive dynamic on-chip memory management for fpga-based reconfigurable architectures," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–8.

[8] A. Oetken, S. Wildermann, J. Teich, and D. Koch, "A bus-based soc architecture for flexible module placement on reconfigurable fpgas," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, Aug 2010, pp. 234–239.

[9] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 11, pp. 1712–1724, Nov 2005.

[10] R. Bonamy, D. Chillet, S. Bilavarn, and O. Sentieys, "Power consumption model for partial and dynamic reconfiguration," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, Dec 2012, pp. 1–8.

[11] J. Clarke, A. Gaffar, and G. Constantinides, "Parameterized logic power consumption models for fpga-based arithmetic," in *Field Programmable Logic and Applications, 2005. International Conference on*, Aug 2005, pp. 626–629.

[12] L. Wang, X. Wang, T. Wang, and Q. Yang, "High-level power estimation model for soc with fpga prototyping," in *Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on*, Nov 2012, pp. 491–495.

[13] L. Shannon and P. Chow, "Using reconfigurability to achieve real-time profiling for hardware/software codesign," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '04. New York, NY, USA: ACM, 2004, pp. 190–199.

[14] M. Aldham, J. Anderson, S. Brown, and A. Canis, "Low-cost hardware profiling of run-time and energy in fpga embedded processors," in *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, Sept 2011, pp. 61–68.

[15] H. Hoffmann, J. Eastep, M. Santambrogio, J. Miller, and A. Agarwal, "Application heartbeats for software performance and health," MIT, Tech. Rep. MIT-CSAIL-TR-2009-035, Aug 2009.

[16] H. Hoffmann, M. Maggio, M. Santambrogio, A. Leva, and A. Agarwal, "Seec: A framework for self-aware computing," MIT, Tech. Rep. MIT-CSAIL-TR-2010-049, Oct 2010.

[17] A. Schmidt and R. Sass, "Improving fpga design and evaluation productivity with a hardware performance monitoring infrastructure," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Nov 2011, pp. 422–427.

[18] T. Cervero, J. Dondo, A. Gomez, X. Pena, S. Lopez, F. Rincon, R. Sarmiento, and J. Lopez, "A resource manager for dynamically reconfigurable fpga-based embedded systems," in *Digital System Design (DSD), 2013 Euromicro Conference on*, Sept 2013, pp. 633–640.

[19] D. Göhringer, "Reconfigurable multiprocessor systems: Handling hydras heads – a survey," *SIGARCH Comput. Archit. News*, vol. 42, no. 4, pp. 39–44, Dec. 2014.

[20] A. Rodriguez, J. Valverde, E. de la Torre, and T. Riesgo, "Dynamic management of multikernel multithread accelerators using dynamic partial reconfiguration," in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, May 2014, pp. 1–7.

[21] J. Valverde, A. Otero, M. Lopez, J. Portilla, E. de la Torre, and T. Riesgo, "Using sram based fpgas for power-aware high performance wireless sensor networks," *Sensors*, vol. 12, no. 3, pp. 2667–2692, 2012.