EPIC-Explorer: A Parameterized VLIW-based Platform Framework for Design Space Exploration

Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi and Davide Patti
Department of Computer Science and Telecommunications Engineering — University of Catania
Viale A. Doria, 6 — 95125 Catania, Italy
{gascia,vcatania,mpalesi,dpatti}@diit.unict.it

***Abstract*— The constant increase in levels of integration and the reduction of the time-to-market have led to the definition of new methodologies stressing reuse. This involves not only the reuse of pre-designed processing components in the form of intellectual properties (IPs) but also that of pre-designed architectures. For such architectures to be reused for various applications they have to be heavily parameterized. Several manufacturers, in fact, produce pre-packed solutions for various classes of applications, in the form of parameterized system-on-a-chip (SOC) platforms. In this paper we present EPIC-Explorer, a framework to simulate a parameterized VLIW-based platform that will allow an embedded system designer to evaluate any instance of the platform in terms of performance, area and power consumption. The results obtained show that the framework can be effectively used to explore the space of possible configurations to evaluate the area/performance/power trade-off.**

I. Introduction

The increase in levels of integration forecast for the coming decade indicate an enormous increase in the number of transistors as compared with the previous decade and the implementation of a whole system on a single chip. Unfortunately, the evolution of design methodologies cannot compete with the increase in the capacity for integration, thus leading to a "production gap" and under-use of transistors. Automatic synthesis tools represent a first response to these problems. In the past, the *capture-and-simulate* design paradigm [1] was the most widely used and consisted of capturing the logic ports or register-transfer components in a structural scheme which was then simulated to verify its correctness. This approach was recently replaced by the *describe-and-synthesize* paradigm, in which the functions of a system are first described using an appropriate language and then synthesised into an electronic circuit using automatic synthesis tools.

The production gap continues to widen, however, and many researchers propose approaches based on the reuse of previously designed and verified modules (design reuse). This is essentially a return to the old *capture-and-simulate* approach, but this time, instead of using logic ports as the building blocks for a design, custom and/or standard modules known as intellectual properties (IP) or simply *cores* are used. However, when applied to the design of a system-on-a-chip (SOC), this approach has created new problems that make application very expensive and reserved for certain specific niches.

Platform-based design [2] was introduced several years ago as a concept that would revolutionize chip design and redefine the future of SOC. The basic idea behind the platform-based design approach is to avoid designing a chip from scratch. Some portion of the chip's architecture is predefined for a specific type of application. Usually there is a processor, a real-time operating system (RTOS), peripheral intellectual property (IP) blocks, some memory and bus structure. This implies that the basic micro-architecture of the implementation is essentially "fixed", i.e., the principal components should remain the same within a certain degree of parameterization [3]. Many researchers predict that platforms will take the lion's share of the IC market [2], [3], [4], [5].

Prefabricated configurable platforms have many advantages, including time-to-market and cost advantages through extensive design reuse. The disadvantages are due to the fact that a platform limits choices by reducing flexibility and performance compared with a traditional ASIC or full-custom design methodology.

On the one hand these platforms have to be sufficiently general as to be used across several different applications and on the other they have to be sufficiently simple to customize to meet the constraints and requirements of the specific application to be mapped on them. One solution to this problem is to make the platform parametric [6]. Therefore, these parameterized SOC platforms must be optimally configured to meet varied power and performance requirements of a large class of applications.

There are two main roles in this context: that of the *platform developer* and that of the *platform user*. The platform developer develops a platform oriented towards a certain class of applications (e.g. television set-top boxes, digital cameras, network switches, etc.) and makes it highly configurable or programmable, in order to adapt to different applications and design constraints. Programmability can come in several forms [7], like general-purpose processors, field-programmable logic, and tunable architecture parameters like reshapable memory hierarchy, segmentable bus architectures, and variable bit widths. Of course this configurability and programmability uses far more transistors than more customized designs, but with current and future integrated circuit (IC) technologies, those transistors are readily available [8], [9]. The platform user becomes the real embedded system designer, whose role shifts from that of designing a whole IC to that of programming functions on the platform.

In this paper we present a framework for analysis in terms

of area, performance and power of any instance deriving from a parameterized platform based on a VLIW processor. Having established the configuration of the system (having, that is, defined the instance of the platform), any C application (for the specific instance of the platform) is compiled and simulated, the output obtained being the silicon area occupied, the total number of clock cycles required to execute the application and the total amount of energy consumed. The contribution the paper intends to make is to provide the embedded system designer with a tool that operates at the system level and allows the designer to perform rapid evaluation at a high level of abstraction of the impact of various architectural choices on the variables to be optimized (area, power and performance). The accuracy of the results obtained is guaranteed by the use of estimation models widely tested and discussed in the literature.

The paper is organised as follows. In Section II we discuss some of the most important contributions towards defining and developing simulation software platforms. In Section III we present the architecture of our platform, while Sections IV and V describe the models used to estimate power and area. In Section VI we demonstrate how the platform can be effectively used for design space exploration, giving some experimental results. Finally, Section VII provides our conclusions and indications as to future developments.

II. Previous Work

One of the main objectives of a system designer is to assess the impact of certain architectural choices on the variables to be optimized, from the highest levels of the design flow downwards. Often, however, the design variables to be optimized are typically observed at the lowest levels of abstraction. These are some of the reasons that have driven research towards the definition of estimation methodologies that will allow designers to obtain predictions of the final characteristics of a system at the higher levels of the design flow. The relevant literature is full of contributions in this context, ranging from characterization of the macroblocks making up a system (e.g. cache, memories, buses, peripherals, register files, microprocessors) [10], [11], [12], [13], [14] to frameworks for the analysis of a system as a whole. Most of the contributions belonging to this second category mainly address evaluation of the impact on performance of the architectural and micro-architectural features of the system. SimpleScalar [15], for instance, provides an infrastructure for computer system modeling that simplifies implementing hardware models capable simulating complete applications. During simulation, model instrumentation measures the dynamic characteristics of the hardware model and the performance of the software running on it. Several platforms have been built using SimpleScalar to obtain estimates of power consumption as well as performance. Among these, Wattch [16] is a framework for analyzing and optimizing microprocessor power dissipation at the architecture level. It is based on a suite of parameterizable power models for different hardware structures and on per-cycle resource usage counts generated through cycle-level simulation. Another example is SimplePower [17] an execution-driven, cycle-accurate, RT level power estimation tool that allow to evaluate the effect of high-level algorithmic, architectural, and compilation trade-offs on energy. It also provides the energy consumed in the memory system and on-chip buses using analytical energy model.

Wisconsin Architectural Research Tool Set (WARTS) [18] is a collection of tools for profiling and tracing programs, analyzing program traces, and simulating computer architectures. It contains a tool for profiling and tracing system, a cache performance profiler, a cache simulators, a library for editing executable files, a framework for memory system simulators, and a fast and portable parallel architecture simulator.

Simics [19] is a full system simulation platform, that allow to build your own virtual computer, or use pre-configured models, based on microprocessors such as Alpha, ARM, Itanium, MIPS, Pentium, PowerPC, SPARC, and x86-64. It simulates both uniprocessor and multiprocessor systems, provides a common infrastructure for a broad variety of tasks, including: microprocessor design; memory hierarchy design; component development and testing; automated software quality testing; SoC virtual prototypes; hardware-software co-simulation; and the development of firmware, drivers, and operating systems.

SimOS [20] is a complete machine simulation environment designed for the efficient and accurate study of both uniprocessor and multiprocessor computer systems. SimOS simulates computer hardware in enough detail to boot and run commercial operating systems. In addition to the CPU, it simulates caches, multiprocessor memory busses, disk drives, ethernet, consoles, etc. The SimOS environment also contains powerful mechanisms for exploiting the high degree of visibility and flexibility afforded by software simulation. The simulation models of SimOS are heavily instrumented to collect statistics about the simulated behavior.

ML-RSIM [21] is an event-driven cycle-accurate simulator that integrates detailed processor and cache models with a complete I/O subsystem. Combined with the Unix-compatible Lamix operating system, ML-RSIM provides a unique tool that allows researchers to study the interaction of computer architecture, I/O activity, system software and applications.

RSIM [22] simulates shared-memory multiprocessors (and uniprocessors) built from processors that aggressively exploit instruction-level parallelism (ILP). RSIM is execution-driven and models state-of-the-art ILP processors, an aggressive memory system, and a multiprocessor coherence protocol and interconnect, including contention at all resources. Most processor parameters are user configurable for example, instruction issue width, instruction window size, and number of functional units. It supports a two-level cache hierarchy with separate first-level data and instruction caches and a unified second-level cache. Most cache and memory system parameters including the number of L1 cache ports, the number of L1 or L2 MSHRs, cache sizes, and all latencies are user configurable.

Finally Platune [23], a framework for performance and power tuning of a SOC platform. The SOC platform consits

of a MIPS R3000 processor, a parameterized instruction and data caches, an universal asynchronous receiver and transmitter (UART) peripheral, a discrete cosine transform (DCT) CODEC peripheral, a parameterized peripheral bus and a parameterized system local bus connected by a bridge. Platune is used to simulate an embedded application that is mapped onto the SOC platform and output performance and power metrics for any configuration of the SOC platform.

III. The EPIC-Explorer Platform

The main aim of the EPIC-Explorer platform presented here is to provide the embedded system designer with a framework for evaluating the impact of the architectural and micro-architectural features of a hardware/software system on area, power dissipation and overall system performance. The platform can be used by both software and hardware designers. The former are provided with a tool that will allow an application to be evaluated as regards not only performance but also power consumption, and thus permits optimisation in this direction. The hardware designer can use it to determine an optimal system configuration for a specific application (or class of applications) in terms of area, performance and power consumption.

In the following subsections we will describe the interfacing between the estimation engines (for area and power dissipation) and design space exploration and the Trimaran [24] framework, and we will present the reference architecture.

A. The Data Flow

A key feature of how the EPIC Explorer platform operates is the interface with Trimaran [24], a framework that provides a parameterized compiler for EPIC/VLIW architectures and a library for the generation of the dynamic execution statistics on which the estimate of performance and power consumption is to be based. Illustrating the details of the various components of Trimaran lies beyond the scope of the paper. However, it is sufficient to view Trimaran as a compiler that takes as input the source code of an application and the description of the architecture of the VLIW processor (number of functional units, size of the register file, etc.), and performs the compilation, that is, static scheduling of the operations inside the sequence of bundles of instructions.

Following execution of the benchmark, Trimaran generates a binary file (that can be executed on the host machine) that simulates execution of the benchmark on the VLIW processor. In order to consider the impact in terms of performance, area and power of the memory hierarchy, a cache simulator based on Dinero has been added to the platform. In addition, we have also added models to estimate the area occupied and power dissipated by the memory hierarchy and the power dissipation due to switching activity on the interconnection buses and along the memory hierarchy.

Figure 1 shows details of the flow of information exchanged between Trimaran and EPIC-Explorer. A generic application written in C represents the input to the Trimaran framework. The first Trimaran component, IMPACT, performs classical

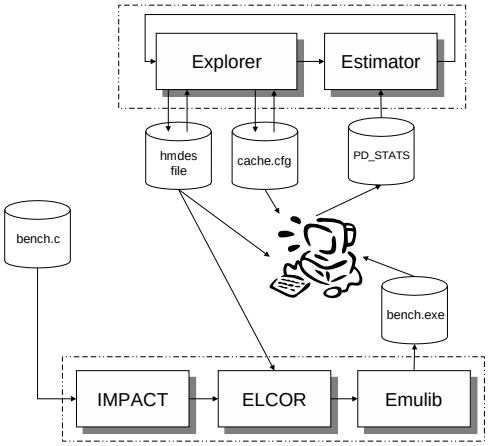

Fig. 1. Interfacing between Trimaran and EPIC-Explorer.

optimization operations on the code, while the Elcor component performs the actual scheduling of the instructions for the specific VLIW architecture. This will depend on the micro-architecture of the processor, which will be described in the `processor.hmdes`[1] file by means of a high-level description language. Subsequently, according to the scheduling performed by Elcor, an executable file is generated via linking with the emulation library Emulib, which provides the code needed to simulate each of the instruction scheduled by Elcor. The end result is a file which, when executed on the host machine, generates a file of statistics`PD_STATS`. These statistics comprise for example the instruction mix, the number of hits/misses per cache, etc. In order to include the cache parameters among those explored, Emulib has been modified in such a way as to consider the configuration of the hierarchy of memories (cache size, associativity, block size, replacement policy, write policy, etc.) described in the file `cache.cfg`. The statistics thus obtained are used by the estimation models implemented in EPIC-Explorer to assess the impact in terms of area/power/performance of the configuration being considered.

Once this evaluation has been carried out, the Explorer component will establish the next configuration to be evaluated, modifying the description/configuration files `processor.hmdes` and `cache.cfg`. The way in which this decision is made naturally depends on the exploration algorithm being executed. As the exploration proceeds, the configurations visited (together with the area, power and performance values estimated) are stored in a database which, on conclusion of the exploration, will be processed to extract only the Pareto-optimal configurations.

B. Description of the Architecture

Figure 2 is a block diagram of the parameterized SOC platform considered. It comprises a VLIW microprocessor core and a two-level memory hierarchy. The exploration parameters taken into consideration for each of the caches L1D, L1I and L2U are size, associativity and block size.

[1]See [25] for a detailed specification of hmdes machine description language

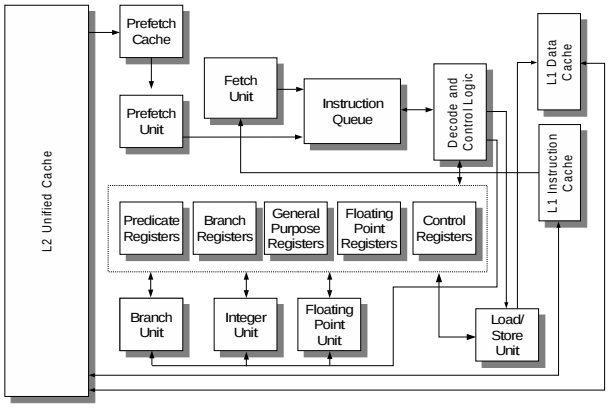

Fig. 2. Reference architecture based on VLIW core.

The processor parameters visible at the architectural level, which are therefore the ones that will be explored, can be classified in two main categories:

- The size of the register files, in terms of the number of registers contained in each of them.
- The number of functional units for each type of unit supported.

As far as the former are concerned, five different types of register file can be identified:

- GPR: 32-bit registers for integers with or without sign.
- FPR: 64-bit registers for floating point values (with single and double precision).
- PR: 1-bit registers used to store the Boolean values of instructions using predication.
- BTR: 64-bit registers containing information about possible future branches.
- CR: 32-bit control registers containing information about the internal state of the processor, for example the program counter.

The functional units involved are:

- Integer Units: execution of integer operations.
- Float Units: execution of floating point operations.
- Memory Units: associated with load/store operations.
- Branch Units: associated with branch operations.

Table I gives the set of values each system parameter can take. There are a total of 18 parameters, 9 relating to the memory hierarchy and 9 to the processor, giving a total of 1.47×10^{13} possible combinations. It should be pointed out, however, that not all the combinations can be mapped in a configuration of the system (unfeasible configurations). For example, a configuration in which the first-level cache has less capacity than the second-level cache is considered unfeasible and will not be part of the space of possible configurations. At any rate, the space of feasible configurations still remains impossible to explore exhaustively.

A fixed parameter of the reference architecture is n_p, that is the number of slots in the instruction bundle, which has been fixed to a max of 6 operations per instruction. Other fixed parameters of the considered architecture are the number of read and write ports for the register files, which have been assumed respectively equal to $2 * n_p$ and n_p.

TABLE I

PARAMETER SPACE.

Parameter	Parameter space
GPR	16,24,32,40,48,56,64
FPR	8,16,24,32,40,48,56,64
PR	8,16,24,32,40,48,56,64,128,256
CR	8,16,24,32,40,48,56,64
BTR	8,12,16
integer_units	1,2,3,4,5,6
fbat_units	1,2,3,4
memory_units	1,2,3
branch_units	1,2,3
L1D_size	128,256,512,1024,2048,4096,8192 16384,32768,65536,131072
L1D_block	8,16,32,64
L1D_assoc	1,2,4,8,16
L1I_size	128,256,512,1024,2048,4096,8192 16384,32768,65536,131072
L1I_block	8,16,32,64
L1I_assoc	1,2,4,8,16
L2U_size	8192,16384,32768,65536,131072 262144,524288
L2U_block	8,16,32,64,128
L2U_assoc	1,2,4,8,16

IV. POWER ESTIMATION MODELS

In this section we will present the power consumption estimation models used in the platform described in Section III.

For the case study reported here we assumed $V_{dd} = 3.3V$ and a clock rate of 500 MHz, which is appropriate since measurements [10] for critical path of $0.1\mu m$ caches showed a cycle time always $\leq 1ns$. Note that parameters of the various models adopted here have been properly scaled to refer uniformly to the $0.1\mu m$ technology size.

A. Processor

To estimate the power consumed by the processor it was decided to use an adaptation of the Cai-Lim model [26] to the VLIW processor. As shown in [27] the model possesses a discrete degree of accuracy and it is designed to demonstrate relative power savings between designs.

The model subdivides the architecture into a set of functional blocks called FBUs (Functional Block Units), associated with various elements of the architecture.

1) Adaptation for Functional Units: The FBUs considered for the architecture being investigated comprise instruction decoding, the integer, floating point,memory and branch functional units. Each of these FBUs is considered to be made up of four different types of circuit: *static*, *dynamic*, *clock* and *SRAM*. The model, the parameters of which were characterised using SPICE on a $0.25\mu m$ technology, gives two measures for each type of circuit:

- *Active Power Density*: average power consumption per area unit when active.
- *Inactive Power Density*: average power consumption per area unit when inactive. This quantity is mainly due to

static power consumption (usually set to 10% of the active power density).

So, the power contribution for a particular FBU can be computed considering the relative area occupation of each type of circuit mentioned above and the activity/inactivity cycles of the FBU.

Indicating the clock period with τ, the energy consumption of a generic FBU can be estimated as:

$$E_{FBU} = \left(\sum_{n=1}^{4} A_n \times P_a(n) \times clk_a + A_n \times P_i(n) \times clk_i \right) \times \tau$$

where n indicates the type of circuit referred to and $P_a(n)$ ($P_i(n)$) and clk_a (clk_i) respectively are the active (inactive) power density and the total number of clock cycles during which the FBU is active (inactive). (The cycles of activity and inactivity are obtained from the execution statistics). The total amount of power dissipated will therefore be the sum of the energy contributions made by the various FBUs divided by the total exection time, that is:

$$P_{average} = \frac{E_{total}}{cycles \times \tau}$$

2) Adaptation for Register Files: The model adaptation for register files is slightly different from the one presented for functional units. The reason is simple: the exploration parameters associated with register files modify their sizes, so we cannot consider a priori a fixed area A_i for each of the 4 circuit types seen above. Based on their SPICE simulations for SRAMs with different sizes, Liao *et al.* proposed a model [28] to estimate the static and dynamic power consumption of a register file. In particular, they obtained equations that take into account:

- *word size*: number of bit for each register in the register file.
- *words*: number of registers in a specific instance of the register file.

As shown in the previous section, each of these two quantities are available from architectural model: the first stricly depends on the type of register file considered, the second is directly associated to the relative register file size exploration parameter. Once static and dynamic power consumpion have been computed, we can obtain active (P_a) and inactive (P_i) power, because, by definition we have:

$$P_a = P_{dynamic} + P_{static}$$
$$P_i \simeq P_{static}$$

So for each register file we can compute the energy dissipation as usual:

$$E_{rf} = (clk_a \times P_a + clk_i \times P_i) \times \tau$$

where τ is the clock period lenght. Then, to get average power consumption we can simply divide the sum of all energy contributions by the total execution lenght $total\_cycles \times \tau$.

B. Memory Hierarchy and Buses

The contribution to power consumption made by the memory hierarchy was estimated using the analytical model presented in [29] based on the characterisation performed by Wilton and Jouppi in [10]. The total amount of power dissipated by a cache is expressed as the sum of four contributions:

$$P_{cache} = P_{bitlines} + P_{wordlines} + P_{output} + P_{ainput}$$

where:

- $P_{bitlines}$ is the power dissipated on account of transitions in the single cell due to pre-loading for eventual access, reading and writing.
- $P_{wordlines}$ is the power dissipated on account of the selection by the drivers of the wordlines for reading and writing operations.
- P_{output} is the power dissipated following transitions of the external interconnection lines driven by the cache. It takes into consideration transitions in the address and data lines connecting the cache to the lower and upper levels.
- P_{ainput} is the power associated with transitions in the address lines at the cache decoder input.

A fundamental aspect of the model being considered is that it is based on estimation of the number of transitions for the various circuit elements involved in the activity of the cache. These transitions are estimated using the dynamic statistics and the equations described in [29].

The contribution towards power consumption made by the interconnection system was calculated by counting the number of transitions on the bus lines and applying the following formula:

$$P_{bus} = \frac{1}{2} V_{dd}^2 \alpha f C_l$$

where V_{dd} is the supply voltage, α is the switching activity (i.e. the ratio between the total number of transitions on the bus and the number of patterns transmitted), f is the clock frequency and C_l is the capacity of a bus line (assuming that all the lines have the same capacity).

V. Area Estimation Models

In this section we will present the area estimation models for the parts making up the platform described in Section III.

A. Processor

The area occupied by the processor with varying architectural and micro-architectural parameters was estimated using the analytical model proposed by Miyaoka *et al.* in [30]. The area is estimated as the sum of a *kernel* and other hardware units. By *minimum kernel* we mean the nucleus of the processor that implements the generic, essential functions, for example the pipeline stages (fetch, decode, execution memory access, write back), a bus for the instructions memory, a bus for the data memory, an ALU unit, a shifter, etc. For varying numbers of instructions that can be executed simultaneously, expressed by the parallelism factor n_p, this minimum generic kernel was mapped in the Miyaoka model and synthesized.

Indicating with $c_k^0(n_p)$ the area of this minimum kernel for various parallelism factors, it is necessary to add the contributions made by the other components of the processor. These additional contributions can be classified as follows:

- c_k^1: increase due to the number of possible instructions and the number of bits per instruction.
- c_k^2: increase due memory interfacing buses
- c_k^3: increase due to the presence of register files
- c_k^4: increase due to the presence of functional units hu.

The total area of the kernel can thus be expressed as:

$$c_k = c_k^0(n_p) + c_k^1(b_{inst}, I) + c_k^2 + c_k^3(n_p, b_{opr}) + \sum_{hu \in HU} c_k^4(hu)$$

Where b_{inst} is the number of bit per instruction (fixed to 32) and I is the number of instructions (fixed to 60). It should be pointed out that these are not the area contributions made by the additional components but represent the increase in the minimum kernel area that management of these components involves.

To the minimum kernel area we have to add the additional contributions due to the area occupied by the register files (c_{rf}) and the hardware units (c_{HU}). As regards the former, if b_r is the size expressed in bits and n_r the number of registers, the area of a given register file will be indicated by the expression $c_r(b_r, n_r, n_p)$. The total contribution made by the register files is therefore:

$$c_{rf} = \sum_{i=1}^{N_{RF}} c_{r_i}(b_{r_i}, n_{r_i}, n_p)$$

where N_{RF} is the number of register files present (in our case study $N_{RF} = 5$).

For the hardware units (e.g. adders, comparators, shifters, etc.) reference will be made to a library of modules already synthesised, optimised and characterised as regards area. So if c_{hu_i} indicates the area occupied by the hardware unit hu_i, the area occupied by N_{HU} will be:

$$c_{hu} = \sum_{i=1}^{N_{HU}} hu_i$$

In short, the area of the processor will be estimated by summing the contributions made by the kernel, the register files and the hardware units:

$$A_P = c_k + c_{rf} + c_{hu}$$

B. Memory Hierarchy and Buses

To estimate the area occupied by the caches we used the model described in [10]. CACTI tool presents a well tested implementation of this model, so it was it sufficient to interface EPIC-Explorer to CACTI estimation functions to get an area occupation value for each of the three caches.

VI. Using the Platform

In this section we will use the platform described previously to study the area/performance/power trade-off surfaces. As stated more than once, it is computationally unfeasible to evaluate each single configuration of the platform, so the configuration space was explored using the approach proposed by Givargis *et al.* in [31]. The parameter dependency graph for our platform is shown in Figure 3.

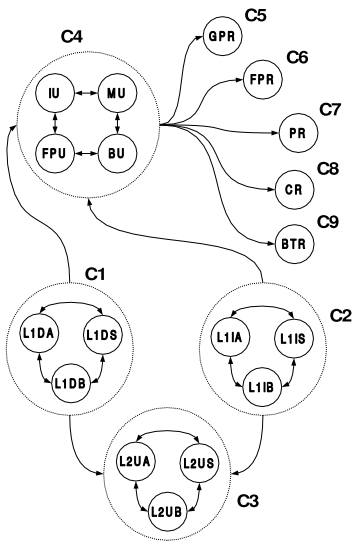

Fig. 3. Dependency graph of the reference architecture.

Given the high level of complexity of the architecture (due to the density of the clusters and the high degree of correlation between them), it is not practical to apply the method in its original for. The algorithm was therefore modified so as to speed up the exploration, at the expense of less accuracy in the solutions obtained. The modification consists of taking into account only the interdependence between parameters belonging to the same cluster, and not of that between clusters. More specifically, with reference to the dependency graph in Figure 3 this is equivalent to eliminating the one-way arrows connecting clusters c3 and c4 to c1 and c2, and all those going from c4 to clusters c5-c9.

Figure 4 shows the power/execution cycle trade-offs for two applications from the Motorola Powerstone suite [32], which contains a collection of embedded and portable applications, including paging, automobile control, signal processing, imaging and fax applications. The first `fir` is an integer FIR filter code from a text book, while the second, `jpeg`, is a JPEG 24-bit image decompression standard. Discontinuities in the trade-off for the `fir` application occurred when the size, line or set-associativity of the caches crossed its working set. Figure 5 shows the area/execution cycle/power trade-off for the `jpeg` application.

Table II summarizes the results obtained. The first column gives the benchmark used: the first two rows refer to exploration in order to optimize power and execution cycles (`jpeg` (2D), `fir` 2D), whereas the last one ((`jpeg` (3D)) refers to exploration aiming at optimizing area, execution cycles and power. The second column gives the number of configurations visited (i.e. the number of simulations carried out) to complete the exploration. The third column gives the time required to complete the exploration. Of course it is not

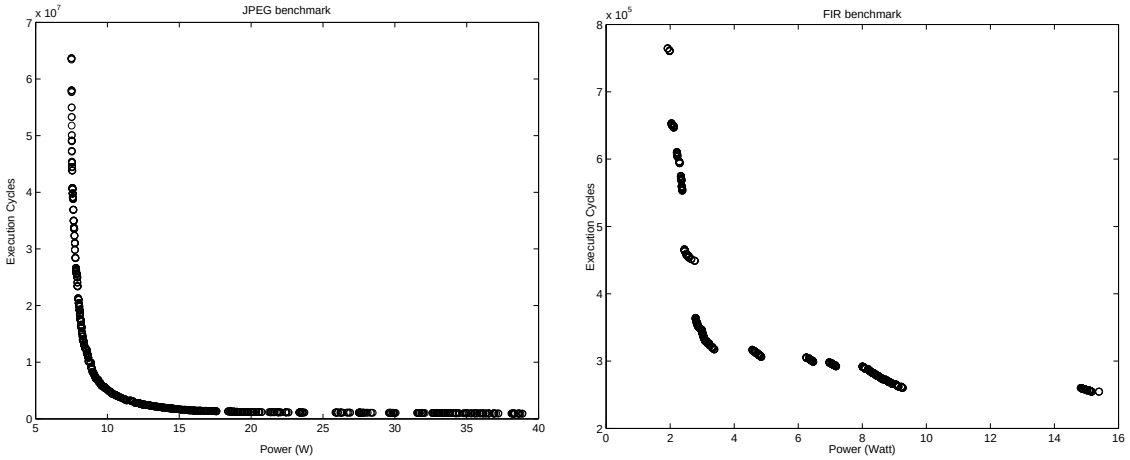

Fig. 4. Trade-off power/execution cycles for the `jpeg` and `fir` application.

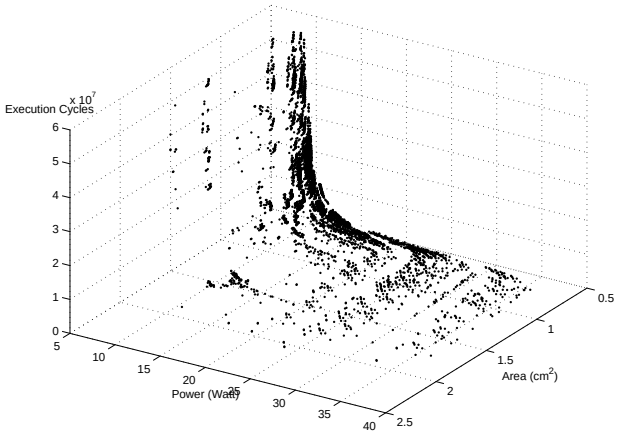

Fig. 5. Trade-off area/power/execution cycles for the `jpeg` application.

directly proportional to the number of configurations visited, as not all simulations require re-compilation of the benchmark. If, for example, the configuration of the processor remains unchanged, it is not necessary to repeat compilation of the benchmark but only the phase referring to its execution. The fourth column gives the number of Pareto-optimal configurations obtained. Finally, the remaining columns give the maximum difference in the area, power and execution cycle trade-offs. As can be seen in the last column, in the case of execution cycles the values differ considerably between the two benchmarks considered. This is essentially due to the lower degree of computational complexity in `fir`. It was, in fact observed that with this benchmark an increase in the size of the caches beyond 1KB did not lead to any substantial variation in the execution cycles.

VII. Conclusions and Future Developments

In this paper we have presented EPIC-Explorer, a framework for the simulation of a parameterized SOC platform based on a VLIW processor. The platform, which can be freely downloadable from the Internet [33], allows the designer to evaluate any application written in C and compiled for any instance of the platform in terms of area, dissipated power and clock cycles. The main use the platform has been designed for is to provide a powerful, flexible simulation and estimation framework that can be used to develop design space exploration algorithms. The high degree of parameterization of the platform generates an enormous configuration space, exhaustive exploration of which would be computationally unfeasible, and so it is an excellent testbed for comparison between different design space exploration algorithms. The aim of the paper was to present the platform as a simulation framework. Future developments will address the integration in EPIC-Explorer of the various design space exploration algorithms proposed in the literature and the implementation of new ones. At the same time the platform will be extended by adding a hierarchic interconnection system and parameterized peripheral IPs

References

[1] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1991.

[2] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, *Surviving the SOC Revolution A Guide to Platform-Based Design*. Kluwer Academic Publishers, 1999.

[3] K. Keutzer, S. Malik, R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.

[4] J. Xu and W. Wolf, "Platform-based design and the first generation dilemma," in *Ninth IEEE/DATC Electronic Design Processes Workshop*, Apr. 2002.

[5] G. Martin and J.-Y. Brunel, "Platform-based co-design and co-development: Experience, methodology and trends," in *Ninth IEEE/DATC Electronic Design Processes Workshop*, Apr. 2002.

[6] F. Vahid and T. Givargis, "Platform tuning for embedded systems design," *IEEE Computer*, vol. 34, no. 3, pp. 112–114, Mar. 2001.

[7] F. Vahid, "Making the best of those extra transistors," *IEEE Design & Test of Computers*, vol. 20, no. 1, p. 96, 2003.

[8] K. Kiefendorff, "Transistor budgets go ballistic," *Microprocessor Report*, vol. 12, no. 10, pp. 14–18, Aug. 3 1998.

[9] "International thechnology roadmap for semiconductors," Semiconductor Industry Association, 1999.

TABLE II

RIASSUNTIVE TABLE.

Benchmark	Visited configurations	Elapsed time	Pareto configurations	Area tradeoff	Power tradeoff	Cycles tradeff
`jpeg` (2D)	23514	6.5 days	2492	-	5x	68x
`fir` (2D)	7357	1 day	994	-	15x	3x
`jpeg` (3D)	55907	4 days	4991	4x	4x	61x

[10] P. Shivakumar and N. P. Jouppi, "CACTI 3.0: An integrated cache timing, power, and area model," COMPAQ Western Research Lab, Palo Alto, California 94301 USA, Tech. Rep., 1999.

[11] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Transactions on Very Large Scale Integration*, vol. 2, no. 4, 1994.

[12] W. Fornaciari, D. Sciuto, and C. Silvano, "Power estimation of system-level buses for microprocessor-based architectures: A case study," in *ICCD*, Austin, Texas, Oct. 1999.

[13] M. Nemani and F. N. Najm, "High-level area and power estimation for VLSI circuits," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 697–713, June 1997.

[14] A. Srinivasan, G. D. Huber, and D. P. LaPotin, "Accurate area and delay estimation from RTL descriptions," *IEEE Transactions on Very Large Scale Integration*, vol. 6, no. 1, pp. 168–172, 1998.

[15] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[16] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *International Symposium on Computer Architecture (ISCA)*, 2000, pp. 83–94.

[17] W. Ye, N. Vijaykrishnan, M. T. Kandemir, and M. J. Irwin, "The design and use of SimplePower: A cycle-accurate energy estimation tool," in *Design Automation Conference*, 2000, pp. 340–345.

[18] M. D. Hill, J. R. Larus, A. R. Lebeck, M. Talluri, and D. A. Wood, "Wisconsin architectural research tool set," *Computer Architecture News*, Aug. 1993.

[19] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hllberg, J. Hgberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer*, pp. 50–58, Feb. 2002.

[20] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the simos machine simulator to study complex computer systems," *ACM TOMACS Special Issue on Computer Simulation*, 1997.

[21] L. Schaelicke and M. Parker, "Ml-rsim reference manual," Department of Computer Science and Engineering, University of Notre Dame, Tech. Rep., 2002.

[22] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve, "Rsim: Simulating shared-memory multiprocessors with ilp processors," *IEEE Computer*, vol. 35, no. 2, pp. 40–49, Feb. 2002.

[23] T. Givargis and F. Vahid, "Platune: A tuning framework for system-on-a-cip platforms," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, Nov. 2002.

[24] "An infrastructure for research in instruction-level parallelism," http://www.trimaran.org/.

[25] J. Gyllenhaal, "A machine description language for compilation," Master's thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana IL, Sept. 1994.

[26] G. Cai and C. H. Lim, "Architectural level power/performance optimization and dynamic power estimation," in *Cool Chips Tutorial colocated with MICRO32*, Nov. 1999.

[27] S. Ghiasi and D. Grunwald, "A comparison of two architectural power models," *Lecture Notes in Computer Science*, vol. 2008, 2001.

[28] W. Liao, J. Basile, and L. He, "Leakage power modeling and reduction with data retention," in *IEEE/ACM International Conference on Computer-Aided Design*, Nov. 2002.

[29] M. B. Kamble and K. Ghose, "Analytical energy dissipation models for low power caches," in *IEEE International Symposium on Low Power Electronics and Design*, Aug. 1997.

[30] Y. Miyaoka, Y. Kataoka, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Area/delay estimation for digital signal processor cores," in *Asia and South Pacific Design Automation Conference*, 2001, pp. 156–161.

[31] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized systems-on-a-chip," in *International Conference on Computer Aided Design*, Nov. 2001.

[32] J. Scott, L. H. Lee, J. Arends, and B. Moyer, "Designing the low-power M*CORE architecture," in *Power Driven Microarchitecture Workshop at ISCA98*, Barcelona, Spain, June 1998.

[33] D. Patti and M. Palesi, "Epic-explorer," http://epic-explorer.sourceforge.net/, July 2003.