# Embedded Reconfigurable Computing: the ERA Approach

Georgios Keramidas
Industrial Systems Institute
Patras, Greece

Stephan Wong, Fakhar Anjam, Anthony Brandon, Roel Seedorf
Delft University of Technology
Delft, The Netherlands

Claudio Scordino
Evidence Srl.
Pisa, Italy

Luigi Carro,
Debora Matos
UFRGS
Porto Alegre, Brazil

Roberto Giorgi,
Stamatis Kavvadias
University of Siena
Siena, Italy

Sally A. Mckee,
Bhavishya Goel
Chalmers University of Technology
Göteburg, Sweden

Vasileios
Spiliopoulos
Uppsala University
Uppsala, Sweden

***Abstract*—The growing complexity and diversity of embedded systems—combined with continuing demands for higher performance and lower power consumption—places increasing pressure on embedded platforms designers. The target of the ERA project is to offer a holistic, multi-dimensional methodology to address these problems in a unified framework exploiting the inter- and intra-synergism between the reconfigurable hardware (core, memory, and network resources), the reconfigurable software (compiler and tools), and the run-time system. Starting from the hardware level, we design our platform via a structured approach that allows integration of reconfigurable computing elements, network fabrics, and memory hierarchy components. These hardware elements can adapt their composition, organization, and even instruction-set architectures to exploit tradeoffs in performance and power. Appropriate hardware resources can be selected both statically at design time and dynamically at run time. Hardware details are exposed to our custom operating system, our custom runtime system, and our adaptive compiler, and are even visible all the way up to the application level. The design philosophy followed in the ERA project proved efficient enough not only to enable a better choice of power/performance trade-offs but also to support fast platform prototyping of high-efficiency embedded system designs. In this paper, we present a brief overview of the design approach, the major outcomes, and the lessons learned in the ERA project.**

***Keywords: adaptive embedded platform; hardware-software codesign; reconfigurable computing.***

## I. INTRODUCTION

The embedded systems market has become a main focus in Europe, distinguishing the European high technology sector from the more high-performance systems market in the USA and the consumer electronics and semiconductor markets in Asia. The strong application areas in Europe are spread among different application markets such as automotive, aerospace, industrial automation, medical/healthcare, telecommunication, and audio/video processing. The target Objective ICT-2009.3.4 "Embedded Systems Design", to which the ERA project was submitted, puts a strong focus on the development of a novel (generic) embedded systems design methodology that can be applied to several application areas.

In the ERA (Embedded Reconfigurable Architectures) project [1], we develop a platform that can adapt itself through coarse-grain reconfigurable hardware to tailor the hardware itself to the changing needs of the applications running it to respond to different application markets, platform usage, or user objectives. The proposed ERA platform can provide adaptability at different abstraction levels: optimization of application software at design time, OS control and optimization at run time to cope with changing conditions, and hardware adaptation at run time to efficiently tune its performance to the application or OS needs, taking power budgets into account.

However, harnessing a highly reconfigurable hardware fabric requires innovative design solutions and methodologies given the many problems that have to be addressed. First, power consumption of reconfigurable devices is generally high, since reprogramming them requires reading from an external memory. Second, support for reconfiguration necessarily requires additional wiring and extra hardware complexity. Finally, design flows must remain sufficiently simple to hide optimization bottlenecks from the user. The ERA project addresses these issues by introducing a reconfigurable fabric that can be adapted at design time, at application deployment, and even during execution (and thus over the product's entire lifetime).

Finally, to exploit the adaptability of the proposed ERA platform, we deployed specific software tools (working either at complier or OS level) and we developed our own benchmark suite (and a corresponding benchmark characterization methodology). The applications comprising the ERA benchmark suite were carefully selected to represent workloads of new-generation smart phones.

## II. PROJECT OVERVIEW

To cope with the power and performance problems that reconfiguration presents, the ERA approach emphasizes

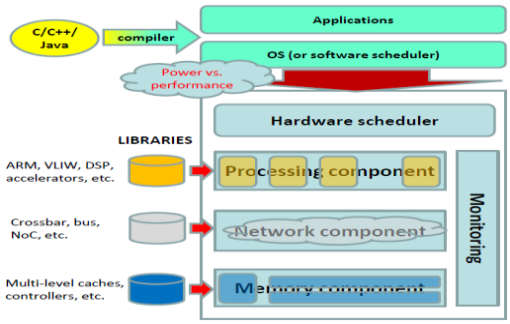



Fig. 1. High-level view of the ERA system.

accelerator development within a coarse-grain reconfigurable fabric. The ERA family of architectures combines the ρ-VEX reconfigurable VLIW processor, flexible memory organizations, and a configurable interconnection network that provides better power management by distributing routing resources (Fig. 1). The software stack is comprised of a compiler and an OS (including a run-time management system) that can drive both static and dynamic reconfiguration decisions according to application characteristics and user power and performance objectives.

The choice to use one adaptable processor avoids the need to create a different accelerator for each new application or application domain, which is costly in terms of design time and time-to-market. Adapting the processor organization on-the-fly avoids these pitfalls while supporting flexible power and performance management. VLIW processors represent an excellent design point with respect to this management strategy because of their excellent power/performance trade-offs. The issue-width and dimensionality can be varied to meet size and power design constraints. The variable issue-width can increase ILP at the cost of heavier pressure on the memory subsystem (which we therefore adapt, as well). Knowing specific application behaviors and requirements—along with user objectives—allows us to precisely tailor the memory hierarchy organization and management to better match the processor's data-consumption needs. On-chip memory can be effectively reorganized; targeted turned-off policies (e.g., decay) can reduce power consumption; shared-memory communication can be minimized; and data placement and replacement can be controlled in hardware and software. Such malleable memory systems leverage information from the application and the compiler together with hardware monitoring and profile-directed feedback. Just as memory needs change from phase to phase and from application to application, so do communication requirements. We thus implement a reconfigurable NoC to manage changing communication needs. Finally, dedicated monitoring hardware is responsible for conveying information to the OS level to support global reconfiguration decisions.

TABLE I. The ERA Benchmark Suite (EBS)

| Application | Characteristics | Domain | Source |
|---|---|---|---|
| ***cjpeg/djppeg*** | Image compression / decompression using JPEG method ([5]) | image processing | C |
| ***X.264*** | H.264 encoder and decoder ([6]) | video processing | C |
| ***EC-DS*** | Elliptic Curve Digital Signature [7] | security | C |
| ***tesseract*** | Perform OCR recognition | OCR | C++ |
| ***mpeg2*** | MPEG2 video decoder | video processing | C |
| ***ac3*** | AC3 audio decoder | audio processing | C |
| ***susan*** | Image smoothing, corner, and edge detection ([5]) | image processing | C |

## III. ERA Benchmarks & Characterization

### A. The ERA Benchmark Suite (EBS)

In the context of the ERA project, the partners have put together a number of benchmarks representative of the embedded domain. In order to select interesting applications, the partners chose a new generation mobile phone as the target appliance, although considerations that emerged during the project lifetime extend beyond that domain. The selected benchmarks, shown in TABLE I, include image, video, and sound processing applications, as well as security and text recognition (OCR) applications. Some of the benchmarks were taken from other suites, as indicated in the table. The mobile phone scenario also allows us to take into account the presence of a complex operating system, since it has a fundamental role in the management of the hardware platform, and thus in the adaptation process.

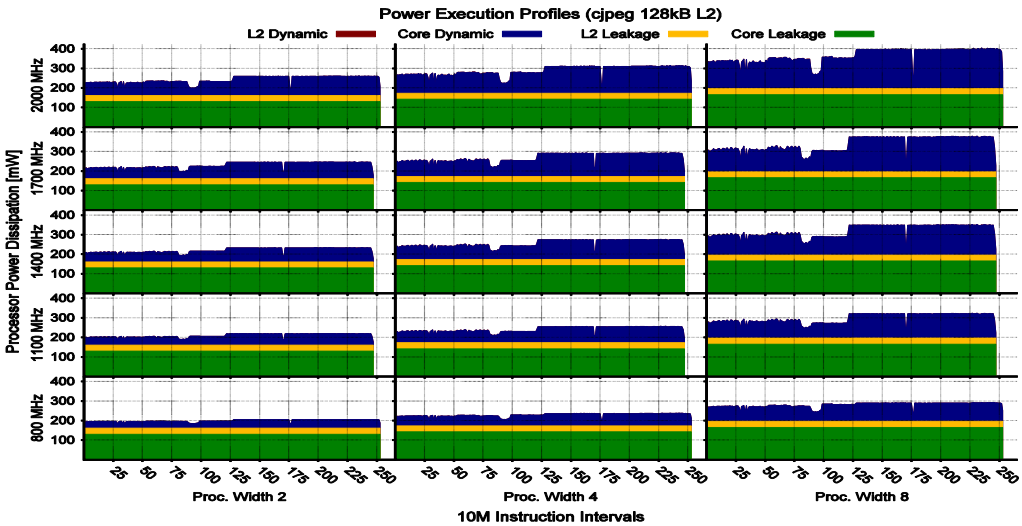
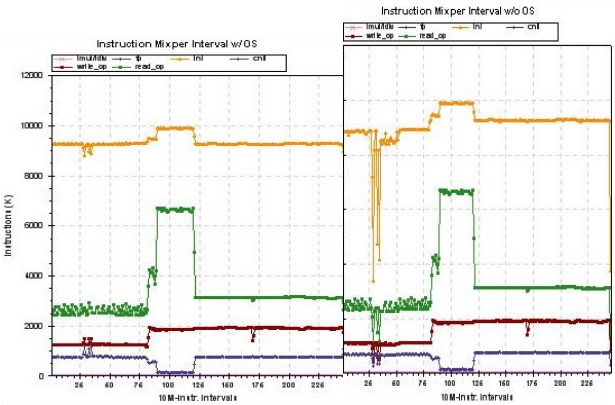
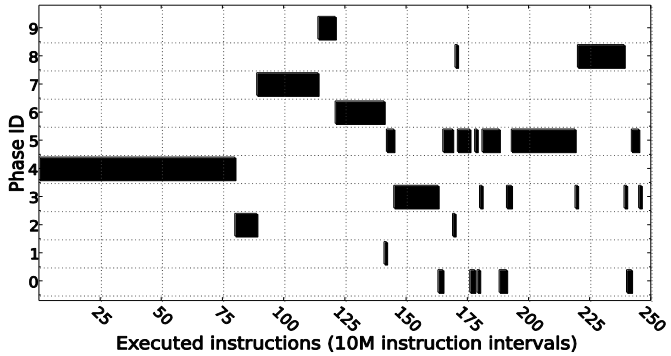



Fig. 2. Power profiles of the execution of *cjpeg* with different frequencies and processor issue widths.




**(a)** Instruction mix per interval of *cjpeg*, plus memory read and write operations (x86 ISA), including the OS (left) and excluding the OS (right). (int instrs: orange, cntrl instrs: blue, read ops: green, write ops red)




**(b)** Phase classification for *cjpeg* using Simpoint (Kmax=10)

Fig. 3. Per interval detailed instruction mix profile (above) and phase classification (below) of *cjpeg*.

### B. Benchmark Characterization

We performed an in-depth and diverse characterization of the EBS, exploring the design space for reconfiguration, but also in terms of configuration-independent metrics (with and without OS instructions), and phase classification. Our results include detailed execution profiles of the applications. Representative results are shown in Fig. 2 and Fig. 3. For all experiments, we use the COTSon/SimNow [3] simulation infrastructure, and we conduct further processing with Simpoint [4] frequency vectors of instructions executed in functions and loops in each interval for phase classification. For power, we use McPAT [2], modified for efficient per-interval measurements.

## IV. Hardware Platform

### A. Core Microarchitecture: The ρ-VEX VLIW Processor

The ρ-VEX [13][14] is an extensible and reconfigurable softcore VLIW processor based on the VEX ISA [15]. The parameterized processor is implemented in VHDL. A VEX development toolchain including a C compiler and a cycle-accurate simulator is freely available from Hewlett-Packard (HP) for architectural exploration and code generation.

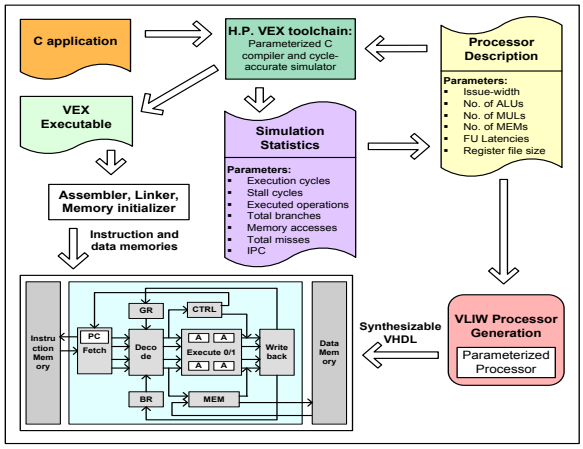



Fig. 4. Methodology to generate and utilize the ρ-VEX processor.

Fig. 4 depicts the methodology to generate and utilize the design-time configurability of the ρ-VEX processor. The process starts with a C application, which is simulated and profiled to determine a good processor configuration for it. The VEX simulator [16] reads the processor's configuration parameters, such as issue-width; number of ALUs, multipliers (MULs), and load/store or memory units (MEMs); latencies for the different functional units (FUs); and register file size. It simulates the application and then generates a detailed log file with statistics such as execution cycles, total executed operations, total branches, stall cycles, memory operations (accesses, misses), and IPC. The process can be repeated until a good processor configuration is found. The optimized processor parameters are then used to generate a synthesizable VHDL description for the processor. We do not require any commercial tools for VHDL generation. The parameters are set in a configuration file and included with the design files when the processor is being synthesized. The compiler is also provided with the same description to generate an executable for the processor. Hence, optimized processors can be implemented in FPGAs quickly for different applications, reducing the development time and the associated costs.

Furthermore, utilizing the 2-issue ρ-VEX processor, we implemented a run-time reconfigurable multi-core processor, called the 2-4-8-issue ρ-VEX processor [17]. The processor has four 2-issue cores, each of which can run independently. If not in use, each core can be taken to a low power mode by gating off its source clock to reduce the power consumption. Multiple (up to four) 2-issue cores can be combined at run-time to make a larger issue-width core. The possible configurations for the processor are: (1) four 2-issue cores, (2) one 4-issue and two 2-issue cores, (3) two 4-issue cores, and (4) one 8-issue core. The configuration and issue-width are changed in a single cycle after the configuration bits are written. The parameters that can be changed at run-time include the issue-width; the number of ALU, MUL, and MEM units; and the size of the register file. The processor can exploit both instruction and/or thread level parallelism. Individual cores can be interrupted (and their states saved), merged, or split, and their running tasks can be migrated to other cores in order to improve performance and power consumption characteristics.

TABLE II. Implementation results for the ρ-VEX processors.

| Processor | Registers | LUTs | DSP48E1s | RAMB36E1s |
|---|---|---|---|---|
| **2-issue** | 844 | 2429 | 4 | 4 |
| **4-issue** | 1451 | 4988 | 4 | 16 |
| **8-issue** | 2754 | 12088 | 8 | 64 |
| **2-4-8-issue** | 3187 | 16790 | 16 | 64 |

Table II presents the implementation results for the 2-, 4-, 8-, and 2-4-8-issue ρ-VEX processors. We used the Xilinx ISE version 13.2 and the Virtex-6 XC6VLX240T-1FF1156 FPGA for the implementation. All the processors run at above 100 MHz. The 2-, 4-, and 8-issue processors have the same number of ALUs as their issue-width; have only 1 MEM unit each; and have 2, 2, and 4 MUL units, respectively. The 2-4-8-issue processor has 8 ALUs, 8 MULs, and 4 MEM units.

### *B. Memory-Driven Reconfiguration Policies*

One of the main contributions of the ERA project is the formulation of a new, three-dimensional reconfiguration approach according to which it is possible to carefully budget the available system power between the core and the last-level cache (LLC) in unicore and multicore reconfigurable scenarios. Figure 5 depicts a high-level view of the proposed reconfiguration framework. The idea behind our approach is that there is a great synergy between the following system parameters: i) Voltage/frequency level of the core (using core DVFS [8]), and ii) processor instruction window (leveraging dynamic core issue-width resizing), and iii) effective LLC size (controlled by cache resizing techniques [9]). As shown in Fig. 5, those three points formulate a triangle in which all vertices are tightly coupled and complex interactions between the three components exist. This reconfiguration triangle can orchestrate informed reconfiguration policies driven either by the application characteristics (memory or CPU bound programs, memory access behavior or ILP characteristics), the available power budget (in terms of any power related metric, e.g., EDP), or the required system performance (e.g., soft or real-time applications, QoS).

Let us present a working scenario: starting from the core, it is possible to lower the core frequency by simultaneously increasing the core parallelism (issue-width) without hurting performance (*case 1* in Fig. 5), if the application exhibits high ILP and is CPU bound. For CPU-bound programs we can further reduce the effective size of the LLC (*case 2*) using dynamic LLC resizing [9]. In contrast, in memory bound programs we can further reduce the core Voltage/frequency setting (using DVFS), since most of the time in those programs the core remains idle waiting while misses are serviced (*case 3*). Going one step further, by lowering the effective LLC size, the miss ratio of the program may increase (*case 4*), but by increasing the core parallelism we also increase the processor's ability to tolerate the extra miss latencies (*case 5*). Finally, the bottom-line idea in our framework is that an orthogonal parameter in this reconfiguration triangle is the notion of memory-level parallelism (MLP) in LLC misses (*case 6*). The importance of MLP is highlighted in a separate publication [8].

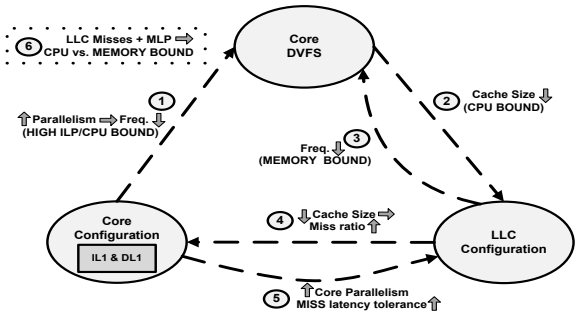


Fig. 5. Memory-driven reconfiguration triangle.

### *C. Adaptive Network*

In the same manner, as processor-cache/memory bandwidth and throughput vary, communication needs also vary. Devising a single strategy/organization for the communication network would mean either spending extra power without reaping equivalent performance gains or suffering degradation in the quality of services. Therefore, in the ERA project an adaptable NoC is designed to manage changing communication needs. We chose to apply our reconfigurable approach to the router buffers due to their high power dissipation. We rely on a simple feedback-based mechanism to monitor and redistribute the buffer resources among the NoC channels at run time. Our solution basically adds multiplexers to each channel, allowing the allocation of buffer slots according to the current communication rates in each router port. Our architecture is called AR (Adaptive Router). More specifically, if a channel has a lower communication rate than its neighbor, it can lend some of its unused buffer slots to a neighbor having a higher communication rate. When a different communication pattern is detected, the roles may be reversed or modified at run time, with no need for a redesign step. The proposed architecture is able to sustain performance due to the fact that not all buffers are completely used at any given time. We have verified that our proposal offers significant advantages over similar approaches in both power and performance [18] [19].

## V. SOFTWARE STACK

The ERA software stack is comprised by the ρ-VEX reconfigurable compiler, an application API, and an innovative run-time system (including a supervisor module). The whole system is also supported by a customized Linux-based OS designed and implemented for the purposes of the ERA project.

### *A. Adaptive Compiler*

The use of the variable issue-width ρ-VEX VLIW processor necessitates corresponding support from the compiler side, which was not provided by existing VLIW compiler solutions (given the static scheduling used in VLIW cores). The ρ-VEX compiler toolchain consists of the HP VEX compiler, a port of the gnu binutils package with assembler and linker, and a tool to convert object files into synthesizable VHDL code for the instruction ROM and data memory. Furthermore, we investigated compiler algorithms that are able to offer the required flexibility and capacity to the

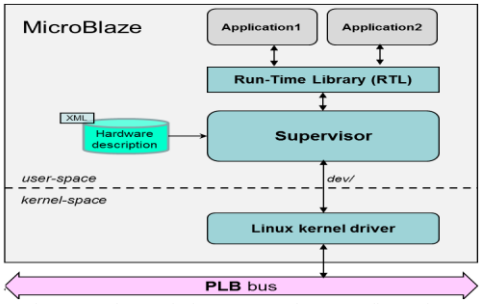


Fig. 6: Software infrastructure for reconfiguration.

ρ-VEX core in a single VLIW schedule (even in the case of a multicore system). The compiler's awareness of the dynamic underpinnings of the ρ-VEX is combined with hardware techniques to enable the generation of generic binaries that can be executed by different-issue VLIW processor cores. This is achieved without losing much performance while maintaining full flexibility at run time to enable dynamic decisions based on other factors (e.g., other threads, energy/power budget).

### B. OS, Application API and Run-Time System

The software stack is comprised of not only the compiler but also the application API needed to guide reconfiguration, an OS, and a run-time support system to manage the transformation process. The run-time system is a critical component because it is responsible for monitoring, allocating, and reconfiguring the underlying hardware resources. Several challenges must be addressed for an efficient implementation of a run-time system. For example, one must take into account the classic trade-off between the reconfiguration benefits (in power or performance) and the reconfiguration overheads (time or energy lost for selecting and instantiating a new system setup). Furthermore, in case of a multithreading workload, proper scheduling decisions must be issued. In contrast to traditional scheduling policies, our scheduler has the additional burden of reconfiguring the hardware fabric.

Previous approaches in the area [10], [11] focused on run-time instantiation of specific hardware components and suitable programming models, rather than considering run-time mechanisms to issue reconfiguration decisions. As a result, in the ERA project, we have designed and implemented an appropriate software infrastructure to ease the investigation of innovative reconfiguration mechanisms and policies. The main goal of the software infrastructure is to reconfigure the hardware resources based on the information gathered by the monitoring hardware at run-time, the compiler, or even the phase detection approach described in Section III. Moreover, the software infrastructure includes a scheduler for allocating the reconfigurable processors and a basic programming model (API) to allow the applications or even the programmer to interact directly with the hardware in an abstract way.

The ERA reference platform consists of a Xilinx Virtex-6 FPGA board including a MicroBlaze softcore (to act as work dispatcher) and the reconfigurable fabric (an array of ρ-VEX cores, memory and network resources). The host core (MicroBlaze) is powered by an Evelin BSP embedded Linux distribution provided by Evidence Srl. Figure 6 depicts the main three components of the software infrastructure. The

**START_PHASE** (*QoS*, *phase_type*, *deadline*)

*QoS*: value of Quality of Service (SMALL, MEDIUM, HIGH)

*phase_type*: type of the phase; can be either NOTYPE or one or more flags (i.e., OR-ed) among PARALLEL, STRIDED, MANY_REFS, LOOP.

*deadline* (optional): expected duration of the phase

**END_PHASE** ()

Fig. 7: API for triggering reconfiguration.

overall software stack has been fully implemented and validated on the ERA FPGA platform utilizing workloads where one part of them is executed on the host while the other part is executed on the ρ-VEX cores.

*Run-Time Library:* the Run-Time Library is a typical C library invoked at compile time. It facilitates the communication between the application layer and the supervisor (see below) hiding all communication details. Its main goal is to let applications specify information about the next program phase via the API as shown in Fig. 7. The library also allows applications or even the users to dispatch work to the reconfigurable cores and gather the result of the computations via the provided API. An example is shown in Fig. 8.

*Supervisor*: the supervisor is a user-level Linux daemon implemented in C++. It collects run-time information through the specified monitoring hardware, schedules requests on the ρ-VEX cores, and issues reconfiguration decisions. The supervisor can be programmed to follow specific optimization policies, e.g., to save power and/or to deliver more performance. The architecture-dependent information is stored in XML files for portability. More specifically, there are two XML files. The first file describes the hardware architecture. Each device is characterized by a set of properties that can be reconfigured independently by writing in special files exposed to user-level software by the kernel driver. This XML file specifies the available devices, their properties, and which files are available to trigger the reconfiguration. The file also contains information regarding the energy used by each configuration. The second XML file contains a "reconfiguration table" that holds extra information about the program phases and QoS requirements.

```
struct task_arg {
        void* ptr; // Pointer to data
        size_t size; // Size of data
};

vproc_t run_task (const char* processor_type,
        const char* program_file,
        const char* data_file,
        const struct task_arg* in,
        const struct task_arg* out,
        enum queue_t queue,
        const char* reconftable_file,
        unsigned int priority);

int get_result(vproc_t vproc,
        enum block_t block,
        const struct task_arg* out);
```

Fig. 8: API for programming reconfigurable processors.

*Linux kernel driver:* the last component of the infrastructure is a Linux kernel driver that exports to user-level a set of files in the dev/ directory to enable the supervisor to interact with the reconfigurable hardware at run-time (e.g., to load data in a reconfigurable core or to resize the core issue-width). In addition, the driver can also manage thread migration between different cores and preempt the execution to run higher-priority tasks.

## VI. PROTOTYPE SYSTEM

### A. Case Study: Reconfigurable Caches

Finally, in this section we also present some additional information about the memory system implemented in our FPGA prototype platform. The ρ-VEX cores are equipped with our own RTL implementation of level-1 (L1) instruction and data caches. This hardware implementation allowed us to study the effects of cache configurations in a fully functional system. As a reference point, we relied on the cache implementation presented in [12]. The cache implementation was performed in a way to support a variety of configuration parameters which can be performed either at compile time or even at run-time. Static reconfigurations include all the major cache parameters (cache size, associativity, and block size) and policies (write policies and replacement policies).

In addition to static reconfigurations, the data cache supports dynamic enabling/disabling of individual cache ways. This cache resizing is managed by the supervisor via the Linux kernel driver. The ρ-VEX core and cache design include 32-bit counters that can track cache performance statistics. The supervisor can use these performance numbers to guide informed way-resizing decisions. The cache associativity is governed by a control register that can be accessed by the supervisor. The current design supports dynamic reconfiguration of cache associativity only for write-through caches, since write-back caches would incur many costly write-backs on reconfiguration. When a cache way is disabled, the corresponding tag and data memory banks are clock gated to prevent further switching and power dissipation.

## VII. CONCLUSIONS

In the ERA project, we focused on many different aspects of the design of reconfigurable embedded systems that allow for not only static determination of the (design) parameters, but, more importantly, dynamic adaption of these parameters at run time. The parameterization encompasses the processor core, on-chip memories, and NoC. We performed in-depth characterization and profiling of a representative set of benchmarks to drive the static design space exploration and the dynamic reconfiguration of these system components. We not only investigated and measured the benefits of our adaptive system for the different components separately, but also determined their interplay. In addition, we developed our own tools and adapted the GCC compiler to target our adaptive platform, which now allows (static and dynamic) reconfiguration decisions to be made by the application designer, compiler, OS, and hardware scheduler. In particular, we integrated the run-time support in a supervisor that considers the "wishes" from all of these sources to make run-time reconfiguration decisions. In conclusion, the ERA project defined a new adaptive embedded system platform that can flexibly adapt itself to the different requirements of the application and the underlying hardware. We demonstrated that performance and energy consumption can be traded off at a much finer level of granularity than in previous systems.

## ACKNOWLEDGEMENT

This work was supported by the European Commission in the context of the ERA (Embedded Reconfigurable Architectures) collaborative project #249059 (FP7).

## REFERENCES

[1] ERA – Embedded Reconfigurable Architectures. [Online]. Available: http://www.era-project.eu, 2010.

[2] S. Li, J.H. Ahn et al. *McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures*. Proc. of the International Symposium on Microarchitecture, 2009.

[3] E. Argollo, A. Falcón et al. *COTSon: Infrastructure for Full System Simulation*. SIGOPS Operating Systems, 2009.

[4] T. Sherwood, E. Perelman et al. *Automatically Characterizing Large Scale Program Behavior*. Proc. of the Architectural Support for Programming Languages and Operating Systems Conference, 2002.

[5] M.R. Guthaus, J.R. Ringenberg et al. *MiBench: A Free, Commercially Representative Embedded Benchmark Suite.* Proc. of the Workload Characterization Workshop, 2001.

[6] C. Bienia, S. Kumar, et al. *The PARSEC Benchmark Suite: Characterization and Architectural Implications.* Proc. of the Parallel Archiytectures and Compiler Techniques, 2008.

[7] S. Bartolini, I. Branovic et al. *Effects of Instruction-set Extensions on an Embedded Processor: a Case Study on Elliptic Curve Cryptography over GF($2^m$)*. IEEE Transactions on Computers, 2008.

[8] G. Keramidas, V. Spiliopoulos, and S. Kaxiras. *Interval Based Models for Run-time DVFS Orchestration in Superscalar Processors*. Proc. of the Computing Frontiers Conference, 2010.

[9] G. Keramidas, C. Datsios, S. Kaxiras. *A Framework for Efficient Cache Resizing*. Proc. of the Symposium on Systems, Architectures, Modeling and Simulation, 2012.

[10] K. Kosciuszkiewicz, F. Morgan, K. Kepa. *Run-Time Management of Reconfigurable Hardware Tasks Using Embedded Linux.* Proc. of the Field-Programmable Technology Conference, 2007.

[11] E. Lubbers, M. Platzner. *A Portable Abstraction Layer for Hardware Threads.* Proc of the Field Programmable Logic and Application Conference, 2008.

[12] V. Saljooghi, A. Bardizbanyan et al. *Configurable RTL Model for Level-1 Caches.* Proc. of the Norchip Conference, 2012.

[13] S. Wong, T. van As, and G. Brown. *ρ-VEX: A Reconfigurable and Extensible Softcore VLIW Processor*. Proc. of the Field-Programmable Technologies Conference, 2008.

[14] S. Wong and F. Anjam. *The Delft Reconfigurable VLIW Processor*. Proc. of the Advanced Computing and Communications, 2009.

[15] J. Fisher, P. Faraboschi, and C. Young. *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*. Morgan Kaufmann, 2004.

[16] Hewlett-Packard Laboratories. *VEX Toolchain*. [Online]. Available: http://www.hpl.hp.com/downloads/vex/.

[17] F. Anjam, M. Nadeem, and S. Wong. *Targeting Code Diversity with Run-time Adjustable Issue-slots in a Chip Multiprocessor*. Proc. of the Design, Automation, and Test in Europe Conference, 2011.

[18] D. Matos, C. Concatto et al. *A NoC Closed-loop Performance Monitor and Adapter*. Journal of Microprocessors and Microsystems, 2011.

[19] D. Matos. et al. *Reconfigurable Routers for Low Power and High Performance*. IEEE Trans. on Very Large Scale Integr. Systems, 2011.