Combining on-hardware prototyping and high-level simulation for DSE of multi-ASIP systems

Paolo Meloni, Sebastiano Pomata, Luigi Raffo
Department of Electrical and Electronic Engineering
University of Cagliari
email: {paolo.meloni, sebastiano.pomata, luigi}@diee.unica.it

Roberta Piscitelli, Andy D. Pimentel
Computer Systems Architecture Group
University of Amsterdam
email: {roberta.piscitelli, a.d.pimentel}@uva.nl

***Abstract*—Modern heterogeneous multi-processor embedded systems very often expose to the designer a large number of degrees of freedom, related to the application partitioning/mapping and to the component- and system-level architecture composition. The number is even larger when the designer targets systems based on configurable Application Specific Instruction-set Processors, due to the fine customizability of their internal architecture. This poses the need for effective and user-friendly design tools, capable to deal with the extremely wide system-level design space exposed by multi-processor architecture and, at the same time, with an extended variety of processing element architectural configurations, to be evaluated in detail and in reasonable times. As a possible solution, within the MADNESS project [1], an integrated toolset has been proposed, combining the benefits of novel fast FPGA-based prototyping techniques with those provided by high-level simulation. In the toolset, the resulting evaluation platform serves as an underlying layer for a Design Space search algorithm. The paper presents the individual tools included in the toolset and their interaction strategy. The approach is then evaluated with a design space exploration case study, taking as a target application a video compression kernel. The integrated toolset has been used to produce a Pareto front of evaluated system-level configurations.**

I. Introduction

Modern embedded systems are parallel, component-based, heterogeneous and finely tuned on the basis of the workload that must be executed on them [2]. To improve design reuse, Application Specific Instruction-set Processors (ASIPs) are often employed as building blocks in such systems, as a solution capable of satisfying the required functional and physical constraints (e.g. throughput, latency, power or energy consumption etc.), while providing, at the same time, high flexibility and adaptability. Composing a multi-processor architecture including ASIPs and mapping parallel applications onto it is a design activity that require an extensive Design Space Exploration process (DSE from now on), to result in cost-effective systems. The MADNESS Project [1] aims at defining novel methodologies for the application-driven customizations of such highly heterogeneous embedded systems. The issue is tackled at different levels, integrating different tools. As a main content of this paper we present part of the overall MADNESS framework: a tool-set in charge of exploring the mentioned design space, according to a search algorithm, relying for the evaluation of the design points under tests on the combination of reconfigurable hardware prototyping and event-based simulation.

The research leading to these results has received funding from the European Community Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 248424, MADNESS Project, from ARTEMIS ART Call 2009 grant no. 100265 - ASAM Project, and by the Region of Sardinia, Young Researchers Grant, PO Sardegna FSE 2007-2013, L.R.7/2007 "Promotion of the scientific research and technological innovation in Sardinia".

Event-based simulation is a widely used technique, offering flexibility and speed, but on the other hand it needs calibration data, as a preliminary input and periodically in the iterative process. These data must be acquired through more accurate evaluation methods, tipically instruction-level simulators, that however turn out to be slow especially when compared to FPGA approaches.

FPGA-based emulation techniques have been proposed in the recent past as an alternative solution to the software-based simulation approach, but some further steps are needed before they can be effectively exploited within architectural design space exploration. When performing architectural DSE, a significant number of different candidate design points has to be evaluated and compared. In this case, if no countermeasures are taken, the advantages achievable with FPGAs, in terms of emulation speed, are counterbalanced by the overhead introduced by the time needed to go through the physical synthesis and implementation flow.

The FPGA-based prototyping platform developed within MADNESS overcomes such limitations, enabling the use of FPGA-based prototyping for micro-architectural design space exploration of ASIP processors. In our approach, to increase the emulation speed-up, we exploit translation of application binary code, compiled for a custom VLIW ASIP architecture, into code executable on a different configuration. This allows to prototype a whole set of ASIP solutions after one single FPGA implementation flow, mitigating the afore-mentioned overhead.

II. Related work

The process of system-level DSE is typically performed exploiting support from two different kind of tools [3]: an evaluation platform that examines the design points in the design space, using for example analytical models or (system-level) simulation, and an exploration engine that iteratively searches and decides which points have to be evaluated. There is a significant variety of approaches that aim at defining novel methods to perform either one or the other step in a time-effective manner, [4], [5], [6], especially targeting heterogeneous MPSoCs [7], [6], [8].

The majority of the approaches rely on system-level simulation to do the evaluation [3]. Basing on the Y-Chart principle we can find simulation tools that work at a high level of abstraction like [9], [10].

Modular system-level MP-SoC DSE framework are proposed in [11], [12], [8], for DSE of embedded systems. The MultiCube project [13] has similar objectives, but it mostly targets micro-architectural exploration of multiprocessors rather than system-level architectural exploration.

[14] present a framework specifically targeting ASIPs, integrating a design tool-chain with a virtual platform to explore a number of axes of the MP-SoC configuration space.

However, none of the mentioned approaches, to the best of our knowledge, experiments the integration of high-level simulation and FPGA prototyping. In literature, the use of emulation on reconfigurable hardware has been often limited to the analysis and exploration of high-performance computing systems, mainly enabling the prototyping of "static" architectural templates to speed-up the evaluation of architectural design techniques on complex applications. The most important contribution to the field of large hardware multi-FPGA platforms for simulation of complex systems is brought by the RAMP project [15].
Examples of hardware-based full-system emulators are [16] and [17], in which the FPGA-based layer is employed to accelerate the extraction of several metrics of the considered architecture, specified and automatically instantiated in a modular fashion. Such papers report a speedup achievable with the use of FPGA prototyping of three/four orders of magnitude in emulation speed, when compared with software-based simulators. The Daedalus framework [18] can be considered a baseline for the work presented in this paper. In Daedalus, on-hardware evaluation is used for DSE purposes. FPGA-based evaluation platforms are automatically created using the ESPAM tool. However, support for prototyping of highly heterogeneous (e.g. ASIP-based) architectures was not completely provided, since configuration at component-level was not allowed and no countermeasures are taken to balance the overhead related with the synthesis and implementation flow. Some works can be found in literature, that aim at the reduction of the number of necessary synthesis/implementations, by looking at FPGA reconfiguration and programmability capabilities. RAMP Gold, a framework developed within the RAMP project, also provides some capabilities of changing at runtime the cache-related parameters during the emulation. In [19], relying on partial reconfigurability techniques, FPGAs are used to optimize register file size in a soft-core VLIW processor. In our previous work [20] we implemented a hardware reconfigurable prototyping platform to allow fast ASIP design space exploration. We partially modified and extended the previous work, exploiting a full software approach in order to avoid the penalties and overheads due to hardware implementation.

As major contribution of our work we want to present a toolset, developed within the MADNESS project, that provides the needed extensions to the Daedalus framework, to seamlessly and time-effectively couple FPGA-based prototyping with DSE and simulation techniques. The toolset can be comfortably used for the optimization of MPSoC systems based on configurable processing elements. We outline in the paper the interfacing between the components and we assess the achievable results with a use case.

III. Reference architectural template and exploration strategy

In this work and in the MADNESS Project, the developed methods are tested referring to an architectural template that exposes most degrees of freedom that may be experienced in modern embedded systems. The main aim is to optimize multi-processor systems, arbitrarily interconnected by means of custom-tuned communication structures (FIFO-, bus- or NoC-based). IP cores that can be used as building blocks are PEs, Memories, interconnect modules, I/O peripherals. Within the project a library of IPs has been collected and integrated, in order to allow the prototyping of almost completely arbitrarily heterogeneous architectures. The DSE process presented in this work is aimed at defining, for a generic multi-processor architecture, the following architectural parameters:

- Number of processing elements
- Kind of processing elements
- Mapping of the application tasks on the processing elements

Other exploration directions are also envisioned and supported within the project, such as interconnect infrastructure, level of support for fault tolerance and adaptive behaviour, but they are not discussed in this paper for the sake of brevity. Number of processors can vary in a range that can be defined at the beginning of the optimization process, as input directive to the DSE engine. The tasks to be mapped are defined in the application code, and are described using a task graph. The PEs can be static architectures (e.g. GP RISCs) or customized processor configurations. Such processors can be constructed using an industrial flow for the implementation of VLIW ASIPs, based on a flexible *Processor Architecture Template* (see Fig. 1). According to the template, every ASIP consists of a composition of sub-structures called *processor slices*, that are complete vertical datapaths, composed of elementary functional elements called *Template Building Blocks*, such as:

- register files: holding intermediary data in between processing operations, configurable in terms of width, depth, number of read and write ports;
- issue slots: basic unit of operation within processor; every issue slot includes a set of function units (FUs), that implement the operations actually executable. Every issue slot receives an operation code from the instruction decoding and, accordingly, accesses the register files and activates the function units;
- logical memories: container for hardware implementing memory functionality;

- interconnect: automatically instantiated and configured, implementing the required connectivity within the processor.

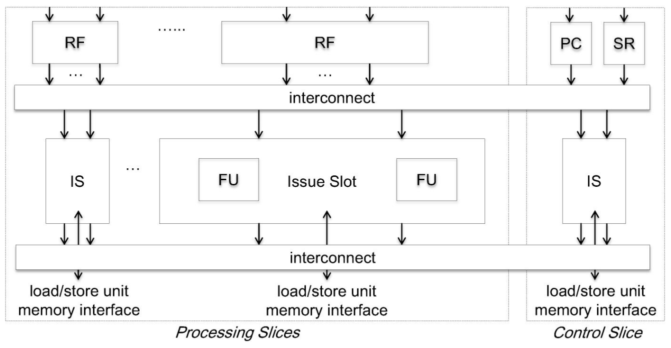

RF
......
RF
PC
SR
...
...
interconnect
IS
...
FU
Issue Slot
FU
IS
interconnect
load/store unit memory interface
load/store unit memory interface
load/store unit memory interface
Processing Slices
Control Slice

Fig. 1. Reference VLIW ASIP template

We enable a design space exploration covering the main degrees of freedom exposed by the Processor Architecture Template (the only assertion we impose is that a *control* slice handling the program counter and status register update must be instantiated in the processor together with an arbitrary number of *processing* slices). Design points are processor configurations that instantiate an arbitrary number of processing slices and different parameterizations of the building blocks included in them. The design space under consideration is thus determined by the following degrees of freedom:

- $N_{IS}(c)$: the number of slices in configuration c;
- $FU\_set(x, c)$: the set of function units in issue slot x, in configuration c;
- $RF\_size(x, c)$: the size (depth) of the register file associated with issue slot x, in configuration c;
- $n\_mem(c)$: number of memories in configuration c.

IV. General toolset description

The toolset that we are presenting in this paper integrates three main components: a search engine, a simulation tool and a FPGA-based prototyping platform. The interaction between the tools is depicted in Figure 2.

The main input to the toolset is the application code, along with the specifications of the constraints that must be considered during the optimization process. Moreover all the tools can receive some input directives related to the settings of their operation mode. The search of the optimal design point is an iterative process that is driven by the *Design space search engine* (search engine hereafter). This tool, that will be described more in detail in section V, embeds novel techniques for effectively pruning the design space by means of heuristic search algorithms and techniques for avoiding the use of relatively time-consuming simulations during DSE. When the search engine requires the evaluation of a (set of) design point(s), it produces a system-level description of those design points (*Design point system-level description* in the figure). At the output of the search engine, such description is expressed using a very abstract format to specify the design

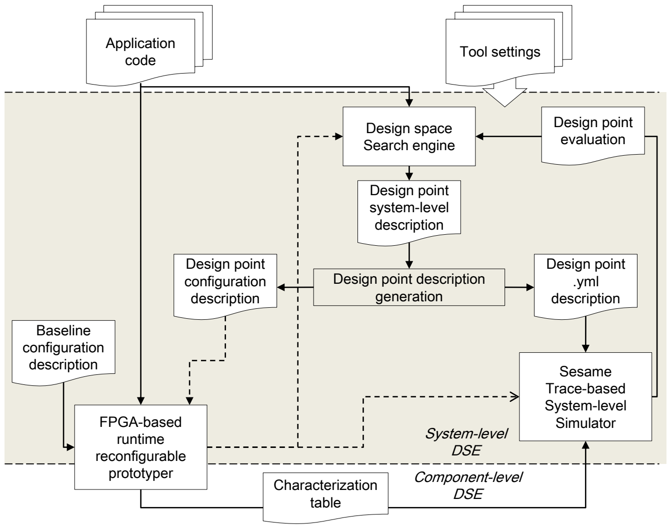

Application code
Tool settings
Design space Search engine
Design point evaluation
Design point system-level description
Design point configuration description
Design point description generation
Design point .yml description
Baseline configuration description
Sesame Trace-based System-level Simulator
FPGA-based runtime reconfigurable prototyper
System-level DSE
Component-level DSE
Characterization table

Fig. 2. General toolset overview

point to be evaluated and, adequately translated by a utility (*Design point description generation* in the figure), can be elaborated by the lower level of the toolchain. At this level, the FPGA-based prototyping platform and the simulator cooperate in different ways during the evaluation process.

More in detail, two use cases are typically possible:

- preliminary calibration;
  At the starting point, the FPGA-based prototyper is exploited for calibrating the simulation model. The execution of the application tasks is emulated on a baseline single-ASIP hardware prototype, to perform a detailed component-level (processor-level) DSE. According to such emulation, the tasks are conveniently characterized in terms of their computation latency over different processor configurations. Once this detailed characterization has taken place, the resulting numbers are passed as input to the simulation model, as a *characterization table*, so that it can start serving as an evaluation platform for the search engine.
- periodic tuning and detailed analysis;
  When needed, during the iterative process, the search engine is able to directly ask the prototyping for a (set of) customized multi-ASIP design point(s) under evaluation (dashed line in Figure 2). The prototyper is exploited, in this case, for system-level DSE, obtaining a detailed characterization that may be provided as a feedback to the search engine and to refine the tuning of the simulator.

At the end, a Pareto front is provided to the user, to be considered when choosing the optimal application-specific architecture.

V. Design Space Exploration: search engine

To optimally explore the design space for optimum design points, a search engine that utilizes heuristic search techniques, such as multi-objective Genetic Algorithms (GAs), has been developed. Such GAs prune the design space by only performing a finite number of design-point evaluations during the search, evaluating a *population* of design points (solutions) over several iterations, called *generations*. With the help of genetic operators, a GA progresses iteratively generating new populations towards the best possible solutions. In the search engine, the design space is explored in an iterative fashion using the NSGA II evolutionary algorithm. This module constructs a chromosome, a string of values representing the architectural- and mapping-related, which, for the sake of the exploration that is considered in this paper, may be defined as follows:

$$[p_1, p_2, \cdots p_j \cdots p_N, k_1, k_2, \cdots k_j \cdots k_N]$$

where the position j refers to a specific application process, the value p_j indicates respectively the ID of the processing unit in the system onto which the application process is mapped, and the k_j indicates the architectural configuration chosen for the processor (obviously if $p_i = p_j$ (task i and j are mapped on the same processor), then $k_i = k_j$). As mentioned, such string is analyzed by the *Design point description generation* utility to produce two different design description formats: the input for the prototyping platform (expressed using an industrial proprietary format) and the input for the simulation tool. Both formats will be described more in detail in the following sections.

To further optimize the DSE process, the search engine also allows for hybrid DSE in which fast but slightly less accurate analytical performance estimations are interleaved with more accurate but slower Sesame system-level simulations to evaluate design points during DSE. Evidently, the aim is to interleave the analytical evaluations with the simulative evaluations in a way such that most evaluations are performed analytically. As a consequence, such an approach could significantly improve the efficiency of the DSE process, allowing for searching a much larger design space. The hybrid DSE part of the Search module is, however, beyond the scope of this paper. The interested reader is referred to [21] for more details.

VI. System-level simulation

For simulative evaluation of design points during the DSE, we deploy the Sesame MPSoC simulation framework [22]. Sesame is a modeling and simulation environment for the efficient design space exploration of heterogeneous embedded systems. According to the Y- chart design approach, it recognizes separate application and architecture models within a system simulation. An application model describes the functional behavior of a (set of) concurrent application(s). An architecture model defines architecture resources and captures their performance characteristics. Subsequently, using a mapping model, an application model is explicitly mapped onto an architecture model (i.e., the mapping specifies which application tasks and communications are performed by which architectural resources in an MPSoC), after which the application and architecture models are co-simulated to qualitatively study the performance consequences of the chosen mapping. For application modeling, Sesame uses the Kahn Process Network (KPN) model of computation in which parallel processes implemented in a high-level language communicate with each other via unbounded FIFO channels. Hence, the KPN model unveils the inherent task-level parallelism available in the application and makes the communication explicit. Furthermore, the code of each Kahn process is instrumented with annotations describing the application's computational actions, which allows to capture the computational behavior of an application. The reading from and writing to FIFO channels represent the communication behavior of a process within the application model. When the Kahn model is executed, each process records its computational and communication actions, generating a trace of application events, an abstract representation of the application behavior, necessary for driving the architecture model. Application events are generally coarse grained. Typical examples are:

- *read(channel id, pixel block)* that represents a communication event, in this case a data read from a FIFO channel
- *execute(DCT)* that represents an atomic computation event, in this case the execution of a DCT kernel.

The architecture model simulates the performance consequences of such computation and communication events generated by the application model. It is parameterized with an event table (the previously mentioned *calibration table*), that contains latency values that are associated to a given event. A table entry could include, for example, the number of cycles needed by a given processor architecture to complete a DCT function, that is the computation latency associated with the event *execute(DCT)*. Other kind of events, such as the previously mentioned communication actions or remote memory access can be modeled, associating a latency to a different architectural component, but are not strictly related with the scope of this paper that mainly discusses about processor characterization. To realize trace-driven co-simulation of application and architecture models, Sesame has an intermediate mapping layer that controls the mapping of Kahn processes (i.e., their event traces) onto architecture model components by dispatching application events to the correct architecture model component.

This description (*Design point .yml description* in Figure 2) is specified in YML (Y-chart Modeling Language), an XML-based format that consists of three parts: a high-level architectural description of the design point, an application graph description and a description of the mapping of application processes and communication channels onto the architecture resources. This information is automatically generated (by the *Design point description generation*). Moreover, it receives the *calibration table* by the FPGA-based prototyping environment.

VII. FPGA-based prototyping platform

The FPGA-based prototyping platform, as mentioned, is used to provide detailed characterization numbers when needed by the optimization process. The evaluation can be done at component-level on a single ASIP (during preliminary characterization) or on a complete system-level configuration. The inputs to the prototyping phase are:

- the partitioned application code (coded in plain C)
- a set of ASIP architectural specifications, describing the processor configurations to be evaluated, expressing number and kind of template building blocks and their connectivity, according to the reference architectural template previously described;
- a system-level specification, describing the system architecture in terms of number and kind of processing elements and defining their connectivity. When performing the preliminary calibration step, the system-level specification is a default single-ASIP system that includes a host processor in charge of uploading the binaries in the ASIP memories. When performing a system-level evaluation, the multi-ASIP specification is automatically created by the *Design point description generation* tool shown in Figure 2.

Both the ASIP- and the system-level specifications (both indicated as *Design point configuration description* in Figure 2) are expressed in a proprietary industrial format. This enables to exploit the tools in the industrial flow ([23]) that is taken as reference within the project, aimed at the design and the programming of ASIP architectures compliant with the previously described general template. The tool suite includes HDL generators and a retargeting compiler, and envisions a typical ASIP design flow. A configuration description is passed to the RTL constructor, that analyzes it and provides as output the VHDL description of the whole architecture. This HDL code is used as input for the FPGA implementation phase, that can be performed with standard commercial tools. The target application code is compiled by means of an adequate compiler, retargeting itself according to the instruction set and the architectural features of the processor under prototyping. After compilation, the program can be executed on the ASIP implemented on FPGA. In order to enable fast on-FPGA evaluation of multiple design points, such flow has been extended within the project as shown in Figure 3.

The prototyping speed-up technique developed within MADNESS focuses on the identification of what we call a *worst case* configuration (WCC), i.e. a processor configuration that is over-dimensioned with the hardware resources necessary to emulate all the configurations included in the predefined set of candidates. Once the WCC has been implemented on FPGA, to evaluate different design points, we run on top of it the binaries obtained compiling the target application for each candidate configuration and *adapted* by means of a custom-defined manipulation algorithm. The manipulation is aimed at activating only the needed subset of the WCC circuitry, to mime the prototyping of the considered design point after a stand-alone implementation and programming flow. During the execution, we obtain, by means of dedicated counters automatically instantiated inside the HDL code before synthesis, performance and switching activity metrics. To evaluate every candidate architecture we take into account only

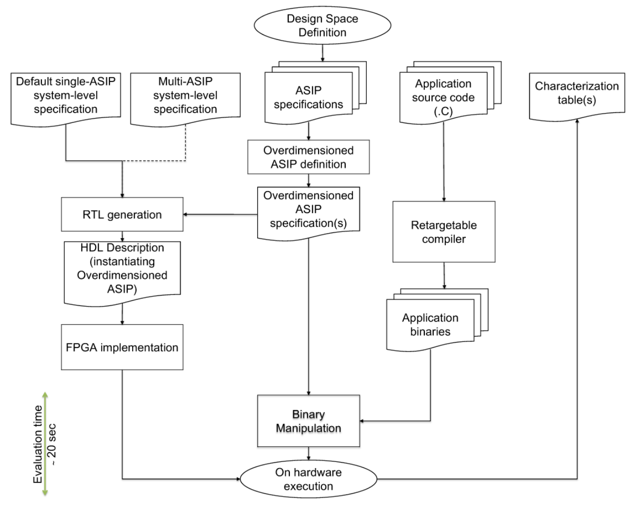

Design Space Definition
Default single-ASIP system-level specification
Multi-ASIP system-level specification
ASIP specifications
Application source code (.C)
Characterization table(s)
Overdimensioned ASIP definition
RTL generation
Overdimensioned ASIP specification(s)
Retargetable compiler
HDL Description (instantiating Overdimensioned ASIP)
Application binaries
FPGA implementation
Evaluation time ~20 sec
Binary Manipulation
On hardware execution

Fig. 3. Prototyper block diagram

the meaningful counters inside the WCC, assuring the obtained results to be perfectly equivalent to those obtainable from its “single-configuration” prototyping.

A. Support for fast prototyping

In order to exploit the support for fast prototyping, all the design points under test must be provided before the beginning of the iterative process. As a first step, we define the WCC by updating its structure at each iteration according to the design point under analysis. At iteration N (i.e. parsing the $N-th$ candidate configuration under test c)

- The number of issue slots inside c is identified and compared with previous iterations. A maximum search is performed, then, if needed, the WCC is modified to instantiate $N_{IS}(WCC)$ issue slots, where
$$N_{IS}(WCC) = max\{N_{IS}(i)\} \text{ for } i = 1, ..., N;$$
- For every issue slot x inside c, the size of the associated register file is identified and compared with previous iterations. A maximum search is performed, then, if needed, the register file related to the issue slot x inside the WCC is resized to have $RF\_s(x, WCC)$ locations, where
$$RF\_s(x, WCC) = max\{RF\_s(x, i)\} \text{ for } i = 1, ..., N;$$
- For every issue slot x inside c, the set of FUs is identified and compared with previous iterations. The issue slot x inside the WCC is modified, if needed, to instantiate a set of FUs being the minimum superset of FUs used in previous configurations:
$$FU\_set(x, WCC) =$$
$$FU\_set(x, c) \cup FU\_set(x, i) \text{ for } i = 1, ..., N;$$

Since we know the architectural parameters for each candidate design point, we can partition the instruction bits in ranges corresponding to control directives to the datapath, like operation codes (selecting specific function units and specific operations inside issue slots), index values for register files read/write operations, and configuration patterns for the connectivity matrices in charge of driving the propagation of the computing data through the datapath. For each range, its width and position are dependent on the architectural configuration that must execute the instruction. For each field, also a disabling configuration is defined, allowing to determine a no-operation for the related datapath part. The general idea is then to manipulate each single instruction field of a candidate configuration, in order to fit it (modified in position, size and value) in the WCC instruction format.

First, the tool parses and analyzes each architectural description, comparing it to the WCC definition, in order to identify: position and size of the field inside the candidate instruction word, position and size of the field in the WCC, and an "offset" indication to be considered during adaptation.

Then, a one-to-one constraint is determined between each processing slice (and related instruction fields) in the candidate architecture and a processing slice in the WCC that is in charge of miming it. The information inside the "offset" structure indicates how to handle and eventually modify the value in the related candidate instruction, considering hardware structures instantiated in the WCC but not in the prototyping of the considered design point. For each instruction in the candidate architecture binary file, a WCC instruction word is then *populated* according to what we know about the architecture. An example is depicted in Figure 4.

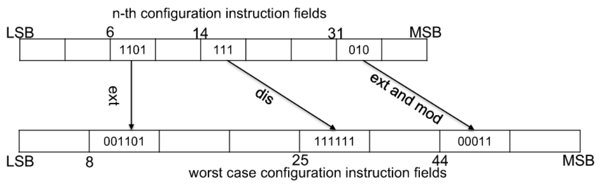

n-th configuration instruction fields
LSB
6
14
31
MSB
1101
111
010
ext
dis
ext and mod
001101
111111
00011
LSB
8
25
44
MSB
worst case configuration instruction fields

Fig. 4. Example manipulation of instruction word. First field (*ext*) is left-extended to obtain the same length of corresponding field on the WCC. The second (*dis*) represents a disabling configuration and it gets adapted for the target worst case configuration. Third field (*ext and mod*) is left-extended and modified, to enable the execution of the binary despite the presence of additional hardware on the WCC.

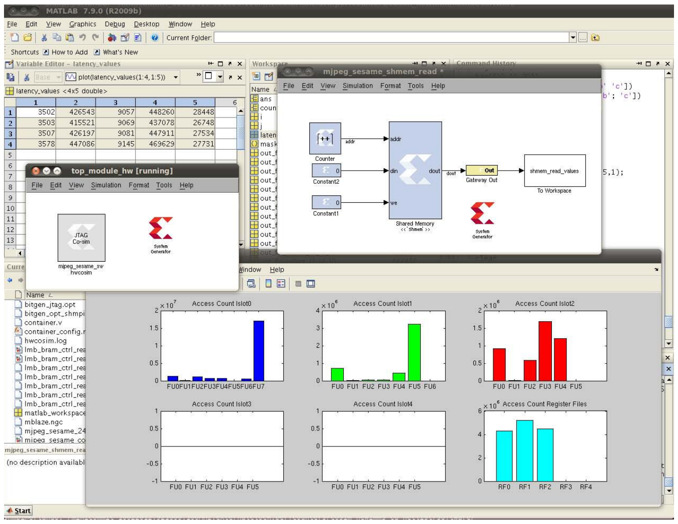
At this point, all the obtained translated binaries can be loaded on FPGA on-board memory and executed on the ASIP prototyping platform only by invoking a custom C function in the application flow, in charge of selecting the correct binary code for the desired emulated configuration and uploading it to ASIP program memory. At the end of each on-ASIP execution, metrics are automatically extracted from the platform, accessing memory-mapped counters, obviously excluding those related to hardware elements that are instantiated within the WCC but are not involved in the prototyping of a specific configuration under test.

VIII. Interfacing the tools through co-simulation

In order to enable the calibration data to be comfortably accessed by the DSE environment, we implemented a dedicated support for extracting the emulation results from the FPGA, exploiting Xilinx SysGen toolbox for Matlab.

The toolbox enables to define shared memories that can be accessed either by the hardware modules implemented on the FPGA, or by a Simulink instance running on a host workstation. In this way, it is sufficient to connect the performance counters inside the processors and the other modules in the system to such memories, to have a user-friendly interface to the evaluation platform. The HDL generator was enhanced in order to automatically set-up the needed connections and wirings to support the counter values fetching.

Simulink objects and Matlab funtions can, at the other end, read from the shared memory activity values and counts to make them available for plotting or in the workspace. Being the DSE environment also implemented using Matlab, this results in an efficient method implementing the exchange of data between the tools (i.e. to implement the transfer of the previously described *calibration table*). In Figure 5 we show a screenshot of the framework user interface.



	1	2	3	4	5
1	3502	426543	9057	448260	28448
2	3503	415521	9069	437078	26748
3	3507	426197	9081	447911	27534
4	3578	447086	9145	469629	27731

Fig. 5. Screenshot of the Matlab GUI after a prototyping of a 4-tasks application kernel executed on 4 ASIP configurations. On the top-left (in the workspace) the *calibration table* of latency numbers to be passed in input to the simulation tool. On the right the Simulink model used to access the shared memories implementing the interface with the FPGA prototype. At the bottom detailed emulation data plotted as histograms. Below the latency values, the FPGA seen as a black box by the Matlab/Simulink environment.

IX. Use Case

In this section we present a typical use case of the previously described integrated toolset, where the FPGA-based prototyping platform is used for preliminary calibration of the simulation model. The target application is a motion-JPEG (M-JPEG in the following) video compression kernel. The use-case that we are presenting is a DSE process that optimizes the mapping of the application parallel tasks on a selected set of ASIP configurations. During the calibration, we explored the component-level design space exposed by the ASIP template, evaluating 18 different ASIP configurations under the workload related with the execution of the parallel tasks inside the M-JPEG task-graph. The explored design points were identified considering different permutations of the following parameter values:

- $N_{IS}(c)$: 2 or 3 or 4 or 5;

- $FU\_set(x, c)$: two different sets of FUs were considered, basically differing only for the inclusion of a *multiply-and-accumulate* FU (MAC), that is the most power/area hungry inside the library;
- $RF\_size(x, c)$: 8 or 32 entries, each 32-bits wide;

A filtering kernel was compiled for every candidate configuration and the resulting binaries were executed on the WCC prototype, after being adequately manipulated. During the preliminary calibration phase, as previously explained, a system with a host processor and one over-dimensioned ASIP core was designed. The host processor is in charge of reading the adapted binary from its local memory and upload it to ASIP program memory. After the binary file has been uploaded, host processor triggers the ASIP core for its start and wait until the end of its execution, to fetch from ASIP local memory the results of the execution and eventually to check them for the presence of any errors.

The adopted hardware FPGA-based platform features a Xilinx Virtex5 XC5VLX330 device, counting over 2M equivalent gates.

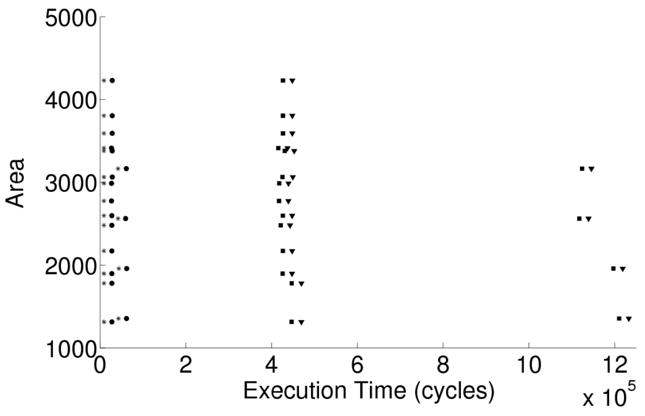


Fig. 6. Pareto plot representing the latency values included in the *calibration table* annotated with the area value corresponding to the related ASIP configurations. Different symbols are used to represent values associated to different tasks in the M-JPEG kernel

The synthesis/implementation flow, performed on an Intel Quad-Core machine with commercial tools, required less than half an hour to complete. Binary translation was also performed on the same machine, but the related overhead in terms of emulation time is negligible (less than a second). According to this numbers, the presented approach allows a time saving that increases with the number of candidate topologies under prototyping, easily outperforming software-based simulation. The results of the preliminary component-level DSE are plotted in Figure 6. All the presented data are obtained after traversing only one synthesis/implementation flow. Area numbers are evaluated according to the ASIP configuration features and to area models provided by the industrial partners in the MADNESS project. Similar models are also available for energy, but the possibility of evaluating power consumption, even if enabled at both simulation- and prototyping-level, is not discussed in this paper. As may be noticed, all the tasks show to have similar behavior with respect to the fitting to the different candidate ASIP architectures. Design points that, for all the tasks, experience an execution time much longer than the others (right end of the graph) are those that do not feature any MAC, that, evidently, is intensively exploited for the kind workload in the M-JPEG kernel. Besides estimating computation latency for the tasks in the target application, the prototyping phase can be used to identify computation bottlenecks and congestion hot-spots inside the architecture. As an example, we show in Figure 7 a graph reporting number of accesses to every function unit and register file in a candidate ASIP configuration, during the execution of the M-JPEG kernel.

As may be noticed, in the presented example, the WCC is used to evaluate a design point featuring only 3 issue slots, thus the activity counters related to issue slots 3 and 4 are never stimulated and must not been considered when evaluating the design point.

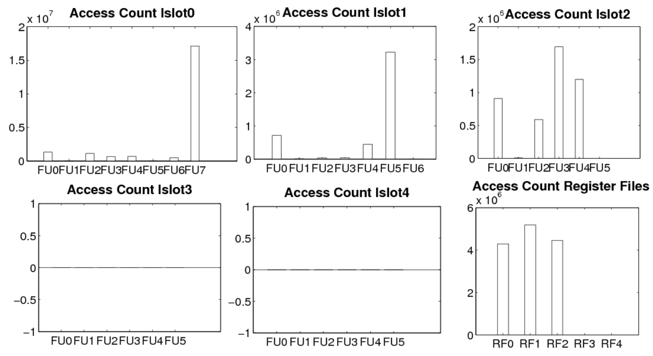
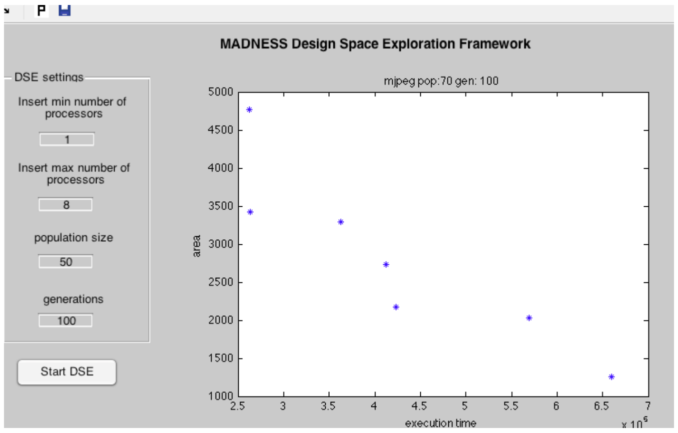


Fig. 7. Detailed calibration results at functional block level

After the calibration step, the system-level DSE process can be initiated. The DSE engine can start evaluating different design points using the simulation model. The simulation model is able to tune itself by reading, directly from the Matlab workspace, the data inside the previously mentioned *characterization table*. As an example of the achievable results, we show in Figure 8 the Pareto graph obtained after an exploration process that involved an iterative evaluation of 100 generations with each population composed of 50 solutions each.

This implies 500 evaluations performed by the toolset. The whole DSE experiment, after calibration, required 35 minutes on a the previously mentioned Intel Quad-Core computer.

As may be noticed in the Pareto graph, after the DSE process, a set of design points has been identified, showing different performance (execution time) vs cost (area) trade off. The fastest and more area-hungry Pareto point (top-left of the graph) features one host processor (executing the *VideoIn* task) and three ASIP processors. The three selected configurations are different, featuring respectively 4 ISs (3 equipped with MAC, executing the *DCT* task), 4 ISs (2 equipped with MAC, executing both *Vle* and *Q*) and 3 ISs (1 with MAC executing *Vout*). However a solution providing the same execution time but requiring less hardware is also identified, mapping *DCT* and *Vout* on the same processors, but using two different


MADNESS Design Space Exploration Framework

DSE settings

Insert min number of processors: 1

Insert max number of processors: 8

population size: 50

generations: 100

Fig. 8. GUI of the DSE framework, plotting the results obtained for the M-JPEG use case

smaller processors for *Vle* and *Q* (respectively featuring 1 IS without MAC and 2 ISs with one MAC) The slowest and cheapest solution (the Pareto point at the bottom-right corner of the plot) is a single-ASIP featuring only one instance of the cheapest processor (1 *processing* issue slot without MAC), that is in charge of executing all the tasks in the target application kernel. Besides providing the plot in the figure, the process collects the simulation results for all the evaluated design points and the related HDL system-level description, in order to enable the protoyping of a multi-ASIP design point on the FPGA platform.

X. Conclusions

In this work, an approach to the application-driven configuration and programming of multi-ASIP systems, based on a combination of trace-based high-level simulation and FPGA-based emulation, is presented and evaluated. The main point of strength of the proposed approach relies on the complementarity of the two methods. While simulation, once duly calibrated, is capable of exploring vast design spaces in reasonable times, FPGAs, if the overhead related with on-hardware implementation is adequately reduced, are a convenient method for rapidly evaluating sets of design points with component-level detail and complete accuracy. The presented use case validates the usefulness of the framework as an effective support to quantitative design space exploration or simply as an environment for rapid prototyping of complex ASIP-based platforms. Future developments of this work will go towards providing, by extending and improving the fundamental mechanisms presented in this article, support for adaptiveness and fault tolerance in ASIP single- and multi-core platforms.

References

[1] “Madness,” www.madnessproject.org.

[2] G. Martin, “Overview of the mpsoc design challenge,” in *Design Automation Conference, 2006 43rd ACM/IEEE*, 0-0 2006, pp. 274 –279.

[3] M. Gries, “Methods for evaluating and covering the design space during early design development,” *Integr. VLSI J.*, vol. 38, pp. 131–183, December 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1056204.1056205

[4] V. Reyes, T. Bautista, G. Marrero, P. Carballo, and W. Kruijtzer, “Casse: a system-level modeling and design-space exploration tool for multiprocessor systems-on-chip,” in *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, aug.-3 sept. 2004, pp. 476 – 483.

[5] C. Lee, S. Kim, and S. Ha, “A systematic design space exploration of mpsoc based on synchronous data flow specification,” *J. Signal Process. Syst.*, vol. 58, pp. 193–213, February 2010. [Online]. Available: http://dx.doi.org/10.1007/s11265-009-0351-6

[6] Z. Jia, T. Bautista, A. Nunez, C. Guerra, and M. Hernandez, “Design space exploration and performance analysis for the modular design of cvs in a heterogeneous mpsoc,” in *Reconfigurable Computing and FPGAs, 2008. ReConFig ’08. International Conference on*, dec. 2008, pp. 193 –198.

[7] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, “Mapping applications to tiled multiprocessor embedded systems,” in *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*, july 2007, pp. 29 –40.

[8] J. Madsen, T. K. Stidsen, P. Kjaerulf, and S. Mahadevan, “Multi-objective design space exploration of embedded system platforms,” in *DIPES*, 2006, pp. 185–194.

[9] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: orthogonalization of concerns and platform-based design,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 12, pp. 1523 –1543, dec 2000.

[10] B. Kienhuis, E. Deprettere, K. Vissers, and P. Van Der Wolf, “An approach for quantitative analysis of application-specific dataflow architectures,” in *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*, jul 1997, pp. 338 –349.

[11] G. Palermo, C. Silvano, and V. Zaccaria, “A flexible framework for fast multi-objective design space exploration of embedded systems,” in *PATMOS*, 2003, pp. 249–258.

[12] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, “Pisa: A platform and programming language independent interface for search algorithms,” in *EMO*, 2003, pp. 494–508.

[13] “Multicube,” www.multicube.eu.

[14] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini, “An integrated open framework for heterogeneous mpsoc design space exploration,” in *Design, Automation and Test in Europe, 2006. DATE ’06. Proceedings*, vol. 1, march 2006, pp. 1 –6.

[15] J. Wawrzynek, M. Oskin, C. Kozyrakis, D. Chiou, D. A. Patterson, S. lien Lu, J. C. Hoe, and K. Asanovic, “Ramp: Research accelerator for multiple processors,” 2006.

[16] P. Del Valle, D. Atienza, I. Magan, J. Flores, E. Perez, J. Mendias, L. Benini, and G. De Micheli, “Architectural exploration of mpsoc designs based on an fpga emulation framework,” 2006. [Online]. Available: http://infoscience.epfl.ch/record/100054

[17] S. Lukovic and L. Fiorin, “An automated design flow for NoC-based MPSoCs on FPGA,” in *RSP ’08: Proceedings of the 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 58–64.

[18] H. Nikolov and et al., “Daedalus: Toward Composable Multimedia MP-SoC Design,” in *DAC ’08: Proceedings of the 45th annual Design Automation Conference*. New York, NY, USA: ACM, 2008, pp. 574–579.

[19] S. Wong, F. Anjam, and F. Nadeem, “Dynamically reconfigurable register file for a softcore vliw processor,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 969 –972.

[20] P. Meloni, S. Pomata, G. Tuveri, S. Secchi, L. Raffo, and M. Lindwer, “Enabling fast asip design space exploration: an fpga-based runtime reconfigurable prototyper,” *VLSI Design*, vol. 2012, no. Article ID 580584, February 2012.

[21] A. D. Pimentel and R. Piscitelli, “Design space pruning through hybrid analysis in system-level design space exploration,” in *Design, Automation and Test in Europe, 2012. DATE ’12. Proceedings*.

[22] A. D. Pimentel, C. Erbas, and S. Polstra, “A systematic approach to exploring embedded system architectures at multiple abstraction levels,” *IEEE Trans. Comput.*, vol. 55, no. 2, 2006.

[23] SiliconHive. (2010) Hivelogic configurable parallel processing platform. [Online]. Available: http://www.siliconhive.com/flex/site/Page.aspx?PageID=17604