Exploiting Binary Translation for Fast ASIP Design Space Exploration on FPGAs

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

Menno Lindwer
Intel Corp.
Eindhoven
email: menno.lindwer@intel.com

***Abstract*—Complex Application Specific Instruction-set Processors (ASIPs) expose to the designer a large number of degrees of freedom, posing the need for highly accurate and rapid simulation environments. FPGA-based emulators represent an alternative to software cycle-accurate simulators, preserving maximum accuracy and reasonable simulation times. The work presented in this paper aims at exploiting FPGA emulation within technology aware design space exploration of ASIPs. The potential speedup provided by reconfigurable logic is reduced by the overhead of RTL synthesis/implementation. This overhead can be mitigated by reducing the number of FPGA implementation processes, through the adoption of binary-level translation. Hereby we present a prototyping method that, given a set of candidate ASIP configurations, defines an overdimensioned ASIP architecture, capable of emulating all the design space points under evaluation. This approach is then evaluated with a design space exploration case study. Along with execution time, by coupling FPGA emulation with activity-based physical modeling, we can extract area/power/energy figures.**

I. Introduction

A common feature of modern embedded systems is the need for highly optimized application-specific processing elements, such as Application Specific Instruction-set Processors (ASIPs). To efficiently explore the hardware-software customization of such systems, appropriate emulation techniques are required to provide fast but accurate performance estimates. For the purpose of architectural exploration, software based simulators, such as those presented in [1] and [2], do not providecycle-accuracy or require unaffordable simulation times for complex designs. To efficiently explore the hardware-software customization of such systems, appropriate emulation techniques are required to provide fast but accurate performance estimates. Approaches relying on FPGA-based emulation techniques have been proposed in the recent past as alternative solutions, such as the RAMP project ([3]), [4], and [5].

However, some further steps are needed before they can be effectively exploited within architectural design space exploration. Firstly, some kind of technology-awareness must be introduced, to enable the translation of the emulation results into a pre-estimation of a prospective ASIC implementation of the design. Moreover, countermeasures are needed to counterbalance the time needed to go through the physical synthesis and implementation flow. As major contribution of our work, we propose a design methodology overcoming 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. We automatically identify what we call a *worst case* configuration (WCC), i.e. a processor configuration over-dimensioned with the hardware resources necessary to emulate all the configurations included in the predefined set of candidates.
To evaluate different design points, we run on the WCC implemented on FPGA the binaries obtained compiling the target application for each candidate configuration and *adapted* by means of a custom-defined manipulation algorithm. During the execution, we obtain, by means of dedicated counters automatically instantiated inside the HDL code before synthesis, performance and switching activity metrics, assuring the obtained results to be perfectly equivalent to those obtainable from its "single-configuration" prototyping.

Several approaches aim at increasing the speed-up reducing the number of necessary synthesis/implementations, by looking at FPGA reconfiguration and programmability capabilities ([6], [7]). Other works [8] [9] [10] exploit binary translation to convert sequential code compiled for single-issue processors into code executable on VLIW architectures. However, we are not aware of any previous work combining binary manipulation and FPGAs for providing support in the design of completely configurable application-specific processing elements.

In order to provide technology-awareness, we also propose the possibility of back-annotating the emulation results with technology-dependent energy, power and area models, to translate the activity figures into a pre-estimation of the area-obstruction and power contribution of each functional block of the architecture implemented on ASIC.

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 JU - 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".

In this work we refer to an industrial ASIP design and programming flow [11]. It includes a cycle-accurate simulation tool that is used for evaluating the execution of an application on the considered ASIP architecture. We compare our approach with such baseline flow structure in order to assess the benefits that can be achieved by means of the proposed techniques.

II. Reference Architectural Template and Exploration Strategy

This work targets in general VLIW ASIPs, which are instances of industrial IPs, based on a flexible *Processor Architecture Template* and providing an automatically retargeting compiler (see Fig. 1). According to the template, every processor consists of a composition of *processor slices* (PS). PS are complete vertical datapaths through the Processor Architecture Template, 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;
- logical memories: container for hardware implementing memory functionality;
- interconnect: automatically instantiated and configured, implementing the required connectivity within the processor.

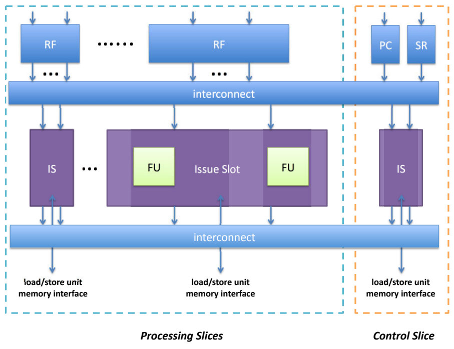
RF
......
RF
PC
SR
interconnect
IS
FU
Issue Slot
FU
IS
interconnect
load/store unit memory interface
load/store unit memory interface
load/store unit memory interface
Processing Slices
Control Slice

Fig. 1. Reference VLIW ASIP template

In this work, we enable a design space exploration that covers the most important degrees of freedom exposed by the Processor Architecture Template. The only assertion we impose is every candidate processor configuration to be composed of one *control* PS (in charge of managing the program flow and interacting with the program memory) plus a variable number of additional *processing* PS providing computational capabilities to the processor.

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.

III. The Reference Design Flow

The overview of the baseline flow, employed in a prospective evaluation cycle, is depicted in Fig. 2. Every configuration to be evaluated during the Design Space Exploration (DSE) process is described using a proprietary description format. Each configuration description is passed to an 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. On the software side, 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.

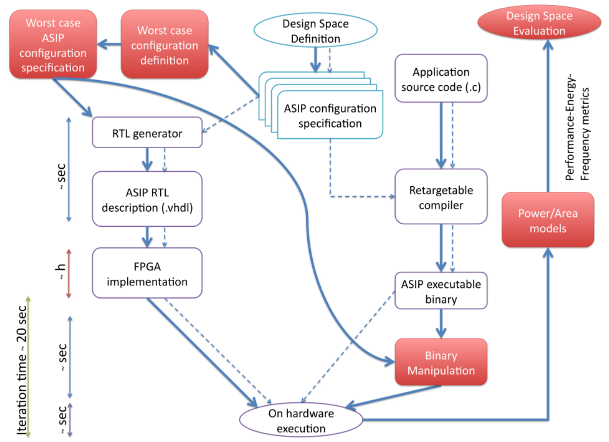

Fig. 2. Reference flow and extended flow with binary translation capabilities. Reference flow steps are marked with dashed arrows, while extended flow ones are denoted with solid arrows. With extended flow, evaluation time for N candidate architectures is approximately 1 hour and $N \times 20$ seconds.

IV. The Extended Prototyping Flow

Within this work, the baseline flow has been extended to enable FPGA-based evaluation of a set of design points. Firstly, the hardware structures of the template building blocks have been instrumented with dedicated activity probes and counters. Secondly, to introduce technology awareness, the collected emulation data and the processor description are translated to physical metrics by means of dedicated area, power and energy models, to be fed back to the optimization algorithm. The translation is performed by means of a set of analytical expressions and tables that allow the evaluation of the energy and area contributions of the functional blocks inside the library, on the basis of their parameterization and switching activity. The models refer to a target technology library. In a third place, in order to achieve faster prototyping of multiple candidate configurations on one single overdimensioned platform, the baseline flow is extended with additional features provided by a in-house developed utility, in charge of identifying and synthesizing the WCC, and able to adapt the binaries compiled for each configuration to allow execution on WCC.

A. The WCC synthesis algorithm

In the extended flow, all the design points under test are provided to the flow at the beginning of the iterative process. The WCC is defined by updating it 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:
  $$FU\_set(x, WCC) =$$
  $$FU\_set(x, c) \cup FU\_set(x, i) \text{ for } i = 1, ..., N;$$

B. The binary manipulation algorithm

For each candidate architecture, knowing the architectural parameters, the instruction bits can be partitioned in sub-ranges that identify specific control directives to the datapath. The width and the position of each range are statically dependent on the architectural configuration that must execute the instruction. For each field, a disabling configuration is defined, able 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, each parsed candidate architectural description is analyzed by the tool, and compared to the WCC description, to identify: the position and the size of the field inside the candidate instruction word, the position and the size of the field in the WCC, and an "offset" indication to be considered during adaptation.

A strict one-to-one relationship is thus established between each processor slice (and the related instruction fields) in the candidate architecture and a corresponding processor slice miming it in the WCC. The information contained inside the "offset" structure indicates how the value in the related candidate instruction must be modified, taking into account hardware structures instantiated in the WCC but not involved in the prototyping of the considered design point.

For every instruction in the candidate binary, a WCC instruction word is then *populated* according to this information. An example is depicted in Figure 3.

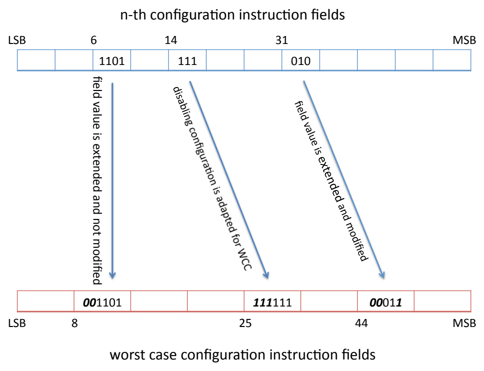
n-th configuration instruction fields
LSB
6
14
31
MSB
1101
111
010
field value is extended and not modified
disabling configuration is adapted for WCC
field value is extended and modified
001101
111111
0001***1***
LSB
8
25
44
MSB
worst case configuration instruction fields

Fig. 3. Example of manipulation of an instruction word. First field is left-extended to obtain the same length of corresponding field on the WCC; second field represents a disabling configuration adapted for WCC; third field is left-extended and modified, due to the presence of an offset value.

V. Power and Area Models

The pre-estimation of a prospective ASIC implementation of a prototyped design point is done according to a pre-built table of "normalized values". The area and the power contribution of every functional block in the processor has been analyzed and its dependency on the architectural parameters and on the activity rate has been studied. The identified expressions are the actual models to be used. Normalized values are the coefficients of the models, while architectural parameters and activity rate are variables. While area is activity-independent, power and energy dissipation depend on the number of cycles when the considered functional block was accessed. Such activity trace is obtained from the counters connected in FPGA prototype to the relevant signals. Normalized values are derived from results of conventional post-layout power simulation through the usage of PrimeTime PX. Proposed results were obtained referring to a 40 nm low power TSMC technology. Despite all the possible factors introducing unpredictability, according to the experiments that were performed, power and area estimation shows an accuracy of 90%. The details about the model expression and the considered dependencies are not reported in this paper for the sake of brevity.

VI. Use Case

In this section we present a use case of the previously described binary manipulation techniques. We plot the results obtained while performing the architecture selection process over a set of 30 different ASIP configurations. The explored design points were identified considering different permutations of the following parameter values:

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

- $FU\_set(x,c)$: from 3 to 10 FUs per issue slot;
- $RF\_size(x,c)$: 8 or 16 or 32 entries, each 32-bits wide;
- $n\_mem(c)$: 2 or 3 or 4 or 5.

A filtering kernel was compiled for every candidate configuration and the resulting binaries were executed on the WCC prototype. In order to execute the chosen application and obtain relevant metrics, a system with a host processor and one 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, triggering the execution on ASIP core and then fetching from it the results of the execution. The adopted hardware FPGA-based platform features a Xilinx Virtex5 XC5VLX330 device, counting over 2M equivalent gates. In Fig. 4 we show the results of the evaluation obtained with respect to total execution time, to total latency, and to total energy and power dissipation. Multi-constraint optimization can be effectively performed. For example, imposing a constraint on maximum execution time (e.g. 200K cycles), the user could identify a subset of candidates that do not satisfy the constraint (gray bullets). Then, among the remaining design points, one could choose the best configuration with respect to power or area (white bullet).

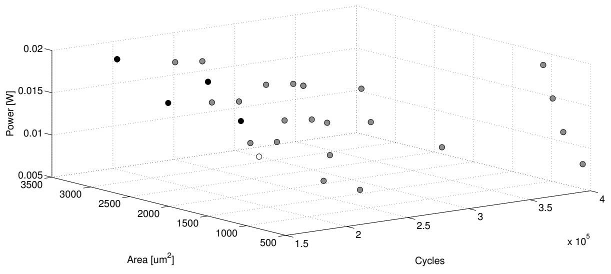

Fig. 4. Use case results. Every configuration could be represented by a different 4-tuple, whose elements represent total number of issue slots, register file capacity (in 32-bit words), number of fully-featured issue slots, number of data memories. Execution cycles, modeled power consumption and modeled area occupation are evaluated for the different configurations under emulation.

All the presented data are obtained after traversing only one synthesis/implementation flow. In order to evaluate the accuracy of the proposed approach and the achievable speedup, we compared the emulation results obtained by means of our prototyping strategy with those obtained using a cycle-accurate software-based simulator referring to the same image-filtering kernel. Both the functional results and the cycle counts (Fig. 4) obtained with the two methods were completely equivalent. But, while cycle-accurate simulation required few minutes (roughly five on average per configuration), onboard execution on the FPGA prototype required only few seconds (roughly two) to emulate each candidate architecture. A synthesis/implementation flow, performed on an Intel Quad-Core machine with commercial tools, required less than half an hour to complete. Such time obviously depends on the size of the system, but can be estimated in the order of one hour for moderately complex systems. 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 the mentioned numbers, as depicted in Fig. 2, the presented approach allows a time saving that increases with the number of candidate topologies under prototyping, outperforming soon (for approximately ten candidate design points involved in the design process) software-based simulation.

VII. Conclusions

In this work, an approach to ASIP configuration selection, based on FPGA-based emulation platforms, is presented and evaluated. To reduce the amount of time needed for standard FPGA exploration flows, we exploited manipulation of the application binaries to show that different VLIW ASIP architectures can be emulated on-hardware by *mapping* them via software on a larger *worst case* configuration.
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 extend the presented binary manipulation tool to support for adaptiveness and fault tolerance in ASIP single- and multi-core platforms.

References

[1] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "Mparm: Exploring the multi-processor soc design space with systemc," *J. VLSI Signal Process. Syst.*, vol. 41, pp. 169–182, September 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1072958.1072962

[2] G. Ascia, V. Catania, M. Palesi, and D. Patti, "A system-level framework for evaluating area/performance/power trade-offs of vliw-based embedded systems," in *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, vol. 2, jan. 2005, pp. 940 – 943 Vol. 2.

[3] 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.

[4] 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

[5] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. H. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *MICRO*, 2007, pp. 249–261.

[6] Y. E. Krasteva, F. Criado, E. d. l. Torre, and T. Riesgo, "A fast emulation-based NoC prototyping framework," in *RECONFIG '08: Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 211–216.

[7] 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.

[8] K. Ebcioglu, E. Altman, M. Gschwind, and S. Sathaye, "Dynamic binary translation and optimization," *Computers, IEEE Transactions on*, vol. 50, no. 6, pp. 529 –548, jun 2001.

[9] J. Dehnert, B. Grant, J. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, "The transmeta code morphing trade; software: using speculation, recovery, and adaptive retranslation to address real-life challenges," in *Code Generation and Optimization, 2003. CGO 2003. International Symposium on*, march 2003, pp. 15 – 24.

[10] M. Gschwind, E. Altman, S. Sathaye, P. Ledak, and D. Appenzeller, "Dynamic and transparent binary translation," *Computer*, vol. 33, no. 3, pp. 54 –59, mar 2000.

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