ON THE AUTOMATIC INTEGRATION OF HARDWARE ACCELERATORS INTO FPGA-BASED EMBEDDED SYSTEMS

Christian Pilato[†], *Andrea Cazzaniga*[†], *Gianluca Durelli*[†],
Andres Otero[⋆], *Donatella Sciuto*[†], *Marco D. Santambrogio*[†]

[†]Politecnico di Milano – Dipartimento di Elettronica ed Informazione – Milano (Italy)
{*pilato,acazzaniga,sciuto,santambr*}*@elet.polimi.it, gianluca.durelli@mail.polimi.it*

[⋆]Universidad Politecnica de Madrid – Centro de Electrónica Industrial – Madrid (Spain)
joseandres.otero@upm.es

ABSTRACT

This paper proposes an automatic framework for the seamless integration of hardware accelerators, starting from an OpenMP-based application and an XML file describing the HW/SW partitioning. It extends a fully software architecture by generating and integrating the cores, along with the proper interfaces, and the code for scheduling and synchronization. Experimental results show that it is possible to validate different solutions only by varying the input code.

1. INTRODUCTION

Heterogenous embedded systems [1] are becoming very popular to perform one or a few dedicated functions, often with strong requirements concerning reliability, performance, low power, real-time and cost. They allow to accelerate different parts of the applications with several general purpose, digital signal and hardware accelerators (realized using Field Programmable Gate Arrays - FPGAs - or dedicated Application Specific Integrated Circuits), interconnected through various communication mechanisms.

Nowadays, the optimization of applications for such complex embedded systems is a difficult problem. In fact, according to the traditional design flow, the hardware and software parts of an application are developed separately, often by hand, and their final integration is based on ad-hoc methods, usually requiring a long time and a high expertise by the designer. Research in hardware/software co-design aims at providing a systematic and concurrent development of the hardware and software components, providing interaction and exploration of trade-offs along the whole design process. We strongly believe that a great improvement in system-level design can be obtained when the designer will mainly focus on the development of the application and the high-level specification of the target architecture. Then, the low-level implementation (e.g., generation of cores, drivers and synchronization directives) should be assisted by automatic design flows.

In this paper, we propose an automated design flow for the integration of hardware cores that aims at bridging the gap between high-level synthesis and system-level design. The proposed framework receives as input the application to be implemented and its decomposition in tasks, with their hardware/software partitioning. It also receives the description of the desired target platform and its components, such as the number of software processors, the interconnection topology and the information about the memory (e.g., addressing spaces). As output, it generates the hardware specification of the entire target architecture and the corresponding software code. As a proof of concept, we developed a backend for targeting Xilinx Virtex5 FPGA devices with Xilinx EDK 12.1 and generating different architectures for a representative test case. The resulting architectures can be directly synthesized, deployed and executed onto the target boards without any additional effort for the designer.

The rest of the paper is organized as follows. Section 2 discusses the related work in the area. Section 3 introduces and details the proposed automatic framework that is then validated in Section 4. Finally we draw some concluding remarks and outline future directions of work in Section 5.

2. RELATED WORK

The automatic generation of hardware platforms is a hot topic in system-level design. For example, Impulse CoDeveloper [2] proposes a design framework that can be configured via XML file, but it leaves to the designer the generation of the synchronization directives. On the other hand, in our opinion, one of the most challenging tasks to create heterogeneous platforms is the generation and the integration hardware cores, along with the development of the corresponding modified application. This work is usually performed manually and it requires a high expertise by the

designer. For example, in [3], the authors propose a framework to automatically generate FPGA-based architectures, but focusing on the prototyping of different interconnection topologies and building cost models. However, the initial application has to be enriched with the driver code to manage the execution of the different cores. Finally, if the application is decomposed in parallel tasks, the execution has to be properly synchronized. On the other hand, in recent years, different algorithms, tools and frameworks for High-Level Synthesis (HLS) have been proposed to leverage the designer from the manual generation of hardware cores (e.g., [4, 5, 6]), but without considering the complete system generation. The interested reader can find a detailed analysis of the different approaches in [7].

In our work, we adopt the structure of the cores proposed in [8] since it allows a seamless integration with software code, and we generate a wrapper for interfacing with the memory as, for example, in [5]. Then, we also generate the complete hardware and software specifications, including software directives for scheduling and synchronization. The framework is built in a modular way, such as it is possible to target different architectures or target devices simply by implementing the related *platform-dependent* part.

3. PROPOSED FRAMEWORK

The overview of the proposed design flow is shown in Figure 1. It receives in input the C code of the application, annotated with OpenMP [9] for describing the parallelism, along with XML files containing the decisions about the HW/SW partitioning and the specification of the target platform. Then, it produces the complete hardware and software specifications for the entire system, ready for the synthesis with commercial tools. The proposed framework allows the user to test applications on the top of different master-slave architectural models, where both the number of master elements (i.e. the processors) and slave elements (i.e. the hardware accelerators) can be customized. All the considered models allow to map the application tasks both in software, by means of general purpose soft processors (i.e., Xilinx Microblaze), or in hardware, by generating dedicated accelerators for specific tasks.

The design flow is mainly composed of three different parts: (1) the platform-independent analysis, (2) the platform-dependent synthesis and (3) the system synthesis. In the follows, we will describe the first two phases that are in our framework. In fact, in the last part (represented by the gray box in Figure 1), we use the generated specification to interface with third-party commercial tools for the generation of the bitstream to configure the target device. As a result, the proposed design flow results complementary to existing methods for system synthesis, allowing to raise the specification of the architecture to a higher level of abstraction.

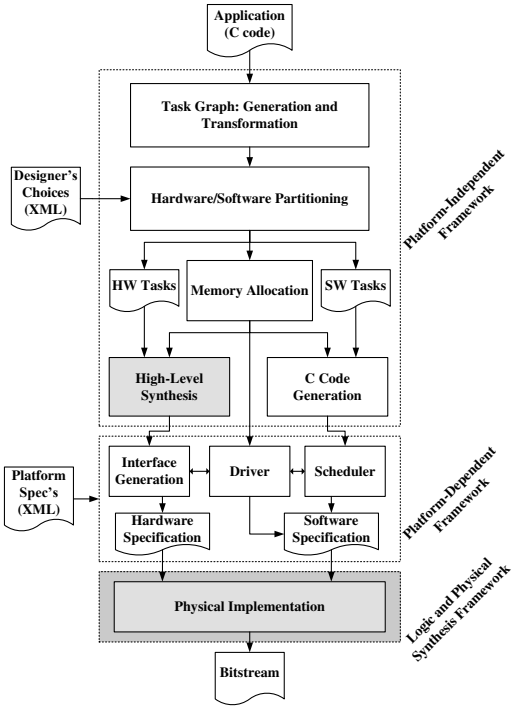

Fig. 1. Overview of the proposed design flow, where gray boxes can represent external tools.

3.1. Platform-Independent Analysis

In this phase, we analyze the input application to derive information about the task graph and the tasks to be executed on the different processing elements. Then we also identify the variables to be allocated in the shared memory and we generate both the hardware and software cores based on this information.

Task Graph Generation: As a first step, we extract the Hierarchical Task Graph [10] (HTG) (i.e., the representation we adopted for the partitioned application) for the original source code, annotated with OpenMP pragmas, and we represent each task with a function. Then, additional transformations may be performed on the HTG. For example, a task needs to be split up if it contains a sequence of function calls that, accordingly to the mapping, have to be run on different processing elements. At this point, we integrate the decisions about the hardware/software partitioning and the tasks are classified with respect to the processing elements assigned for their execution. The hardware tasks require the generation of the corresponding modules, while the code for the software ones can be directly generated. However, in both the cases, it is necessary to define the allocation of shared variables in advance.

Allocation: After determining the tasks to be executed and integrating the information about hardware/software partitioning, the subsequent step consists in the identification of the variables that require an allocation in the shared mem-

ory. In particular, the allocation strictly follows the classical C semantics. In fact, let us consider the following example:

```
int foo(int a, int b, int *c);
...
foo(x, y, &z);
```

The first two parameters are passed by value and, thus, their value can be simply transmitted to the inputs of the cores (e.g., through the system bus or the message passing). On the other hand, the third parameter is passed by reference and, thus, the core `foo` could potentially access the corresponding memory location. For this reason, the actual parameter `z` needs to be allocated into the shared memory and, then, the corresponding address passed to the task. For this reason, we perform an analysis of the variable uses, reserving a space in the memory and assigning the corresponding address to the selected variables.

Hardware Core Generation: The hardware cores to be generated require to receive the parameters both by value and reference. Even if different tools or methods can be integrated (see Section 2), in this work, we adopt the structure of the cores described as follows, since it can easily interface with the memory following the C semantics. Each core contains one input port for each of the input parameters, as declared in the corresponding software code, and an output port for the return value, if any. Additional signals are required to trigger the execution of the core (*start*) or to determine its completion (*done*). Such ports will be connected to the software processor by a direct connection or through the bus. On the other hand, if the corresponding function requires a memory access, the core needs to interface with the shared memory. For this reason, the corresponding ports have to be generated and connected to the proper memory controller. After generating a generic core with the structure described above, we need to design proper interfaces to wrap each generated core, based on the characteristics of the target platform. For example, to connect the memory interface with the Xilinx MPMC, we can design a wrapper based on the Xilinx Native Port Interface (NPI).

3.2. Platform-Dependent Synthesis

In this phase, we generate the parts of the application that are dependent on the target platform, such as the core interfaces, the driver for transferring the data and the runtime manager code, including directives for the synchronization. At this end, we finally generate the specification for both the hardware and software parts of the system. Due to lack of space, we refer the interested reader to [11] for a detailed description of this phase.

Note that the information about the memory addresses determined in the previous phase are here combined with the characteristics of the shared memory contained into the architecture information given as input to this phase. As an example, the variable `z` could be initialized as:

```
int *z = 0x9A000AA;
```

where `0xAA` is the offset inside the memory, as determined by the allocation step, while `0x9A00000` is the actual base address of the shared memory.

4. EXPERIMENTAL EVALUATION

We developed a C++ prototype for validating the first two parts of the framework shown in Figure 1. The former interfaces with the GCC compiler [12] and it is built on the top of its internal representation, while the latter targets the Xilinx XUPV5-LX110T FPGA device (17,280 slices). Xilinx EDK 12.1 is adopted in the last part of the design flow to obtain the bitstream from the generated specifications. Note that, for targeting different FPGA vendors or tools, it is only required to implement the proper platform-dependent part. The hardware cores have been generated with *bambu* [6], a free framework for HLS, under development at Politecnico di Milano. Note that adopting alternative HLS tools (e.g., ImpulseC [2], GAUT [4], xPilot [5]) only requires to generate the proper driver code for data transfers. The resulting architecture is composed of two MicroBlaze processors, working at 100MHz, each of them connected to a dedicated PLB bus, as shown in Figure 2. The Xilinx Mailbox has been adopted to ensure the message passing between the processors, acting as a FIFO to support the consecutive execution of software tasks when this is not directly controller by the scheduler. A timer has been also introduced to measure the execution time of the computation.

To validate the proposed flow, we implemented a pointwise filtering algorithm to compute the edge detection. The algorithm iterates over all the pixels of the input image and computes the convolution between the neighbors of the pixel under analysis in a 3×3 region by computing the discrete Laplacian of a 2-D function. We created different variants of

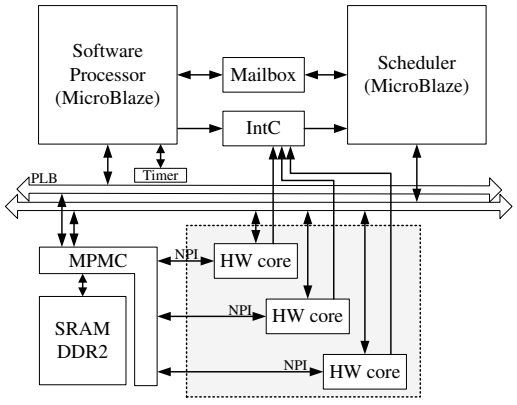

Fig. 2. Block diagram of the implemented target architecture. The gray box represents the part that is integrated by the proposed framework.

Table 1. Area occupation and execution time of the kernel computation for the different experiments. S_p represents the speed-up with respect to the sequential software execution.

# Cores			Area		Time	
	SW	HW	(# slices)	(%)	(# cycles)	S_p
1	1	0	4,102	23.74	6,593,760	1.00
	0	1	5,088	29.44	2,663,764	2.48
2	1	1	4,452	25.76	3,348,185	1.97
	0	2	5,709	33.04	1,362,660	4.84
4	1	3	6,977	40,38	1,777,853	3.71
	0	4	7,908	45.76	1,244,516	5.30
6	1	5	8,408	48.66	1,396,466	4.72
	0	6	9,432	54.58	1,331,082	4.95

this application by splitting up the computations in different cores, from 1 up to 6, differently assigned to the software processors or the dedicated hardware cores, with an ASAP dynamic scheduling algorithm.

The kernel execution time for the different experiments is shown in Table 1, along with their area occupation. We also reported the speed-up S_p with respect to the sequential software execution. As expected, the best execution times are obtained when the tasks are totally executed in hardware, reaching a speed up of 5.3x with four cores. In fact, when one of the tasks is executed in software, this usually limits the performance of the system if the the core iterations are not properly balanced. On the other hand, as shown in Figure 3, increasing the number of the cores results in a contention when accessing the shared memory. This can lead to better explore the memory allocation or to rewrite the application to reduce the memory accesses.

Note that these experiments have been obtained just by varying the input source code and the XML file describing the HW/SW partitioning. Then, each of these architectures required few seconds to be generated by the proposed design flow, ready for the synthesis.

5. CONCLUSIONS AND FUTURE WORK

This paper presented an automated framework for integrating hardware accelerators in FPGA-based embedded systems directly from the C code of the application. It only requires information about the hardware/software partitioning and it can automatically interface commercial tools for the generation of the configuration bitstream.

The proposed framework will serve to evaluate the different techniques that we are developing in system-level design. In particular, the integration of the support for dynamic partial reconfiguration will allow to deploy architectures with a larger part of the application to be executed in hardware.

Acknowledgments

This work was partially funded by the European Commission in the context of the FP7 FASTER project (#287804).

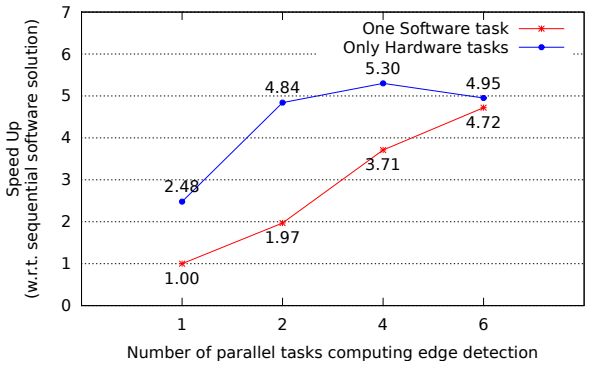

Fig. 3. Speed-up resulting from the hardware acceleration.

6. REFERENCES

[1] M. Duranton, "The Challenges for High Performance Embedded Systems," in *Proceedings of DSD*, 2006, pp. 3–7.

[2] R. Bodenner, "Creating Platform Support Packages," *Impulse Accelerated technologies, Inc., Application Note: IATAPP-109*, 2011.

[3] S. Secchi, P. Meloni, and L. Raffo, "Exploiting FPGAs for technology-aware system-level evaluation of multi-core architectures," in *Proceedings of ISPASS 2010*, march 2010, pp. 194 –202.

[4] P. Coussy, C. Chavet, P. Bomel, D. Heller, E. Senn, and E. Martin, "GAUT: A high-level synthesis tool for DSP applications," in *High-Level Synthesis: From Algorithm to Digital Circuit*, P. Coussy and A. Morawiec, Eds. Springer, 2008.

[5] J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "Platform-Based Behavior-Level and System-Level Synthesis," in *IEEE International SOC Conference*, 2006, pp. 199 –202.

[6] Politecnico di Milano, "Bambu web site." [Online]. Available: http://panda.dei.polimi.it

[7] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473 –491, April 2011.

[8] C. Pilato, F. Ferrandi, and D. Sciuto, "A design methodology to implement memory accesses in high-level synthesis," in *Proceedings of CODES+ISSS '11*, 2011, pp. 49–58.

[9] M. Sato, "OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors," in *ISSS '02*, 2002, pp. 109–111.

[10] M. Girkar and C. D. Polychronopoulos, "The hierarchical task graph as a universal intermediate representation," *International Journal of Parallel Program*, vol. 22, no. 5, pp. 519–551, 1994.

[11] G. Durelli, A. Cazzaniga, C. Pilato, D. Sciuto, and M. D. Santambrogio, "Automatic run-time manager generation for reconfigurable mpsoc architectures," *Proceedings of ReCoSoC '12*, 2012.

[12] GNU Compiler Collection, "GCC, version 4.5." [Online]. Available: http://gcc.gnu.org/