A2B: a Framework for the Fast Prototyping of Reconfigurable Systems

Christian Pilato, Riccardo Cattaneo, Gianluca Durelli, Alessandro A. Nacci, Marco D. Santambrogio, and Donatella Sciuto

Politecnico di Milano,
P.zza Leonardo da Vinci, 32
20133 - Milano (Italy)
pilato@elet.polimi.it

Abstract. The constantly growing complexity of heterogeneous systems requires effective methods for supporting the designer both during the development of the application and the implementation of the architecture. Unfortunately, existing tools still require that the designer develops large parts by hand, especially when hardware accelerators and partial dynamic reconfiguration are taken into account.
This paper presents *A2B*, an ongoing project at Politecnico di Milano, about a semi-automatic framework to assist the designer during the development of reconfigurable heterogeneous systems. It allows to start from a C-based description of the behavioral specification to be implemented and to perform a progressive refinement of both the designed application and the hardware architecture. It offers the possibility to specify decisions either by an interactive environment or by automatic algorithms, hiding most of the implementation details to the designer.

1 Introduction and Related Work

Nowadays, embedded systems are very widespread and the continuous growing request of computational power is supported by technology scaling that allows to create larger and larger devices at lower cost. Moreover, devices based on *Field Programmable Logic Array* (FPGA) are becoming very popular and integrated with general purpose processors (e.g., Intel Stellarton, Apple Macbook Pro and Maxeler Workstation), but they are also used to create custom architectures themselves, both for embedded systems and high-performance computing.

This opens new challenges for *Electronic System Level* (ESL) design [1] and, in particular, for hardware/software co-design and architectural exploration. In fact, according to the traditional design flow, the hardware and software parts of an application are usually 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 developing methodologies and tools for a systematic and concurrent development of the hardware and software components, providing exploration of alternatives along the whole design process.

In this scenario, platform-based design and orthogonalization of concerns [2] are very common in the system design since they allow to decouple the development of the application from the implementation of the architecture. Following this principle, Gerstlauer *et al.* [3] claim that the system-level design flow (shown in Figure 1) is basically composed of two steps: *decision making* and *refinement*. In particular, decision making computes an allocation of resources, a spatial binding and a temporal scheduling, while refinement automatically generates an implementation, consisting of a structural model (e.g., TLM or RTL descriptions) and quality numbers (timing, area and power consumption). These steps can be separately approached, but they require to share several information to converge to high-quality solutions.

In recent years, there is also a renovated interest for *High-Level Synthesis* (HLS): both academic (e.g., LegUp [4]) and commercial (e.g., Xilinx Vivado HLS) solutions have been proposed to easily create more and more hardware cores directly from high-level descriptions (e.g., C/C++/SystemC/Java). The proper use of *Partial Dynamic Reconfiguration* (PDR) is becoming crucial to reduce the area occupation by assigning more than one core to the same FPGA region. However, to obtain efficient systems, it is necessary to properly hide the reconfiguration overhead and, in general, to take these aspects into account from the early stages of the design process. Unfortunately, PDR is still lacking of complete and integrated design solutions, usually requiring complex manual steps.

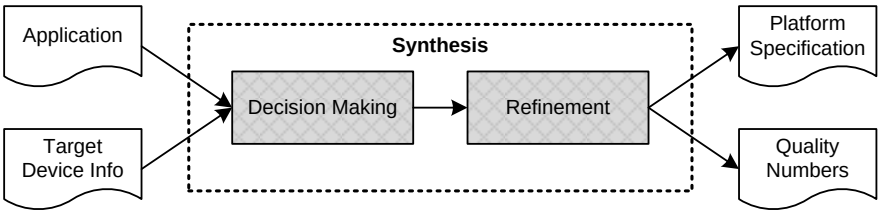

Fig. 1: System-level design methodology, as proposed in [3]

Daedalus [5] is an example of modular framework for hardware/software codesign, but it focuses only on loop affine applications and dataflow graphs. It includes different tools for the automatic partitioning, the system-level exploration and the automatic generation of the architecture, sharing information through a file based on on *eXtensible Markup Language* (XML). However, the integration of hardware cores and aspects related to PDR are not taken into account. Different techniques (e.g., [6, 7]) have been proposed instead to take multiple hardware implementations into account (i.e., trade-offs between usage of resources and execution time), but without proposing an automated flow up to the final implementation of the system. In the same way, [8] is able to automatically partition the application in hardware cores and software processes, but most of the system generation is still done by hand. On the other hand, com-

mercial tools can assist during the refinement phase (with manual intervention) up to the logic and physical synthesis (automatically performed, like in Xilinx ISE/Vivado Design Suite). However, the decision making phase is still left to the designer and it requires a high expertise. In summary, most of the existing solutions cover only specific aspects of the design. We strongly believe that a great improvement in the design of reconfigurable systems can be obtained when the designer will mainly focus on the high-level development of the application and the architecture in an integrated environment, while the low-level implementation (e.g., generation of hardware and software specifications) should be assisted by automatic and integrated design flows.

This paper proposes $A2B$, a modular and semi-automatic framework for the development of reconfigurable systems. It starts from the application description (currently in C language with OpenMP pragmas [9] to suggest the kernels to be potentially implemented in hardware) and a minimal description of the target architecture, along with information about the target device. Such information is used to properly develop the application by efficiently taking into account the characteristics of the target platform (e.g., communication bandwidth/delay, resources available for hardware implementations, ...). Different phases have been identified to cover the different aspects of the design. Moreover, since we defined a common exchange format based on XML and the framework is organized in a modular way, it results very simple to integrate and compare alternative solutions (either automated algorithms or commercial tools) for each phase without affecting the others. The output is a system specification (both hardware and software) that can be directly used with commercial toolchains (e.g., Xilinx ISE Design Suite) to actually generate the bitstream and program the target device.

The main contributions of this paper are:

- it presents a semi-automatic framework to easily generate reconfigurable systems on the top of existing commercial tools;
- it shows how to integrate different solutions for each of the phases, hiding most of the low-level implementation details to the designer;
- it allows to perform co-exploration of application and architecture to generate customized reconfigurable systems.

At the time being, we are working to support both the Xilinx XUPV5 device (for FPGA-based embedded systems) and the Maxeler MaxWorkstation (for High-Performance Computing). We performed different experiments to validate the proposed framework by generating alternative solutions for the same application up the actual execution on the target device and the results look promising. Moreover, we also developed a practical Graphical User Interface (GUI) to assist the designer during the development, suggesting the proper decisions and disabling the ones that would lead to unfeasible solutions.

The rest of this paper is organized as follows. Section 2 describes the proposed framework and its organization. Then, Section 3 and Section 4 detail the different phases of the design, that are the decision making and the refinement, respectively. Preliminary results are presented in Section 5, while Section 6 concludes the paper and outlines future directions of work.

2 A2B: Framework Overview

The aim of the proposed framework is to support the design for three different classes of users: the *application designer*, the *platform architect* and the *system designer*. The application designer needs to evaluate different solutions of partitioning, mapping and scheduling, including the evaluation of the reconfiguration overhead. In such a way, he/she can focus on the optimization of the application and, then, directly evaluate the effects of the decisions on the target platform. The platform architect can evaluate different architectural solutions with respect to the input application. Also in this case, he/she can focus only on the exploration of the architecture and its parameters, having the possibility to directly generate the corresponding platform specification. Finally, the system designer covers both the phases, needing to concurrently develop the application and the architecture. In this case, an integrated environment is definitively required to proper share the information between the two phases of the design.

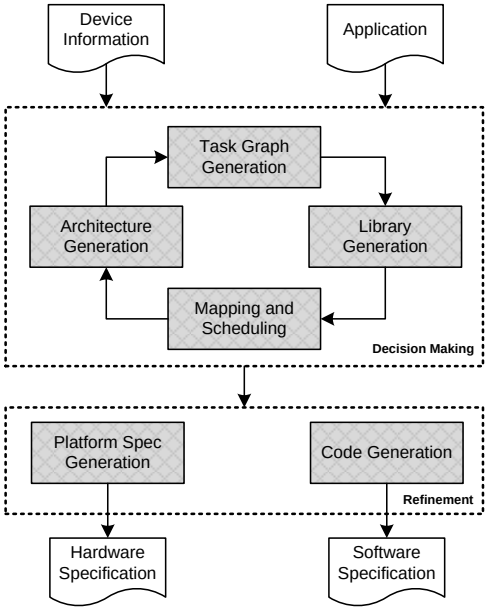

Fig. 2: Overview of the proposed framework, namely $A2B$. Gray boxes represent the identified steps.

The proposed framework, namely $A2B$, follows the idea proposed in [3] and it is thus composed of two different parts, shown in Figure 2: the *decision making* and the *refinement*. The former includes all the phases that are needed for determining the organization of the architecture and the partitioned application, including its mapping and scheduling with respect to the available components of the architecture. The latter includes the generation of the elements for the

actual synthesis of the architecture (i.e., both hardware and software specifications). In details, *A2B* starts from the application specification (in C language with OpenMP pragmas [9]) and a minimal description of the target architecture (in XML), along with information about the target device. As output, it produces the specification of both the hardware and software specifications. The former includes the definition of the architecture and the hardware descriptions of the cores and their interfaces. The latter includes the software that runs on the top of each general purpose processor. The produced system is fully compatible with commercial tools (e.g., Xilinx ISE Design Suite) for the actual synthesis and generation of the bitstream.

In order to easily share the information among the different phases, *A2B* adopts an XML file with the following structure:

- *architecture*: the description (at different levels of abstraction) of processors, hardware logic, memories, IP blocks and their interconnections;
- *application*: the description (e.g., reference to the source code files) and the characterization (e.g., profiling data) of the application;
- *library*: list of hardware and software implementations that are available for each of the application's tasks;
- *partitions*: the description of the partitioned solution (task graph) and its mapping and scheduling.

A2B is thus highly integrated and it effectively allows the co-exploration of architecture and application, if needed.

3 Decision Making

According to the framework proposed in Figure 2, the decision making step starts from the description of the input application and the information about the target device. As output, it produces an XML file that contains the information about the target architecture and the decisions about the application. Then, it is composed of the following phases:

- **Task Graph Generation**: it extracts and optimizes the task graph for the input application.
- **Library Generation**: it generates admissible implementations (both hardware and software) for each of the tasks.
- **Mapping and Scheduling**: it assigns an implementation and a component for the execution of each task.
- **Architecture Generation**: it determines the topology and the high-level parameters of the target architecture (if the architecture exploration has to be performed);

Note that, if the architecture has been already generated and characterized, the design making phase coincides with the classical platform-based design [2]. On the contrary, it is also possible to perform the design of a custom architecture on the basis of the application to be deployed.

Task Graph Generation: In our current implementation, this part starts from the code of the input application and it extracts the task graph based on the OpenMP pragma annotations, as defined in [10], with a compiler-based approach. Automatic partitioning methodologies ([8, 11]) can be also adopted to generate the task graph, provided to generate the proper output description in the *partitions* section. In this case, this part may include further steps, as high-level analysis or profiling of the application behavior, to annotate each of the extracted kernels, represented as *functions* in the rest of the framework. Additionally, different transformations (e.g., kernel splitting/merging) are performed to determine the proper granularity of the extracted tasks and trade-off performance and requirement of resources. The tasks and the task graph are then reported into the XML, associated with the corresponding C-based description to be further processed by methods for library generation. Meta-information, such as the number of repetitions for each of the tasks, is also stored along with the required communications. This will be necessary to properly generate the module interconnections and the run-time system.

Library Generation: For each task, it is possible to generate multiple implementations for the available processing elements. Moreover, considering hardware implementations, it is also possible to derive multiple variants of the same task with different execution time and requirement of resources. In this phase, it is thus possible to interface with HLS tools (e.g., Xilinx Vivado HLS) for the automatic generation and synthesis of the hardware implementation on the target device to determine the latency and the requirements of resources, along with high-level estimation methods. The characterization of software implementations can be obtained instead by estimation or profiling methods [12]. All the results are then reported into the *library* section for further use.

Mapping and Scheduling: To deploy an application on the target architecture, we need to select, for each task, an implementation for its synthesis and a processing element for its execution. Automatic methodologies (e.g., [6, 7]) can be easily integrated in *A2B*: the implementations are contained into the *library* and this phase updates the *partitions* section with the information about the mapping decisions. For each of the hardware components, the framework is able to determine the proper level of reconfiguration. In particular, if only one core is assigned to one portion of the FPGA, it is implemented as a static IP component since no reconfiguration is required. Otherwise, if multiple components are assigned to the same FPGA portion, PDR will be adopted to switch between the different implementations. Finally, if two or more tasks share the same implementation on a component, a reconfiguration is not required to switch between the tasks and *hardware reuse* can be exploited. In such a way, the designer (or the corresponding automated methodology) simply determines the mapping of the tasks, without taking care of the reconfiguration details that are automatically managed by the framework. Note that this part is tightly connected with the floorplacing phase [13], implemented as an inner loop in our methodology. In fact, it is necessary to determine the actual physical regions and evaluate the corresponding task assignment to early identify any violation of the constraints.

Architecture Generation: This part includes all the necessary steps to determine the high-level description of the target architecture, that is the description only with components, interconnections and memories, along with their most relevant parameters (i.e., number of instances, size, ...). Different algorithms can be integrated for the exploration of the architecture based on the application's characteristics. In particular, considering the requirements of the application, it is possible to determine the processors number and area of the FPGA dedicated to hardware cores. Then, it is also possible to explore the interconnection topology (e.g., bus-based, NoC-based or point-to-point) and the memories size as in [14]. This phase thus generates an updated version of the *architecture* section in the XML file to be taken into account by the rest of the flow, allowing a progressive refinement of both application and architecture.

4 Refinement

The refinement step starts from the solution generated in the previous phase (represented into the XML file) and generates, as output, the hardware and software specifications. This part is thus composed of the following phases:

- **Platform Specification Generation**: it generates the specifications of the hardware part (i.e., component instances and their interconnections), compliant with the backend tools and thus ready for the synthesis;
- **Code Generation**: it generates the specifications of the software part, that is the software code that has to run on each of the general purpose processor.

Note that this part is highly target-dependent since it is based not only on the target device, but also on the tools that will adopted for the synthesis.

Platform Specification Generation: It generates the specification of the hardware part in order to interface with commercial tools for the synthesis. This part is highly target-dependent since it requires to have information also about the software tools that will be used for the synthesis. We thus include a database of components classified by the software tool and the corresponding version. This phase then requires to map the high-level components of the architecture that have been selected in the previous phase to the physical components (e.g., IP cores) in the database. At the time being, we support the generation of the project files for Xilinx ISE Design Suite 14.3 to target Xilinx XUPV5 and for Maxeler MaxCompiler to target Maxeler MaxWorkstation.

Code Generation: This step generates the software code to be executed by the general purpose processors, including the software tasks, the run-time manager, including the drivers to control the execution of the cores, the data transfers and the dynamic reconfigurations. Note that, if one task/core requires a dedicated communication protocol to transfer the data, it generates the proper software code according to the protocol description. This is possible since information is shared between the library generation (where the core is created), the mapping (where the implementation is selected) and this part (where the core is instantiated and connected to both the software and the hardware sides).

5 Case Study: Edge Detection

We are implementing *A2B* in C++ as an extension of the framework proposed in [15]. At the moment, we support the specification for architectures to be synthesized with Xilinx ISE Design Suite ver 14.3 and Maxeler MaxCompiler 2012.1. Note that extending the framework to support different toolchains only requires to properly customize the refinement steps. This allows to decompose the problem, with different teams that can work in parallel.

In the following, we will show how the proposed framework can be adopted to easily design a reconfigurable system for a point-wise filtering algorithm to compute the edge detection, where the designer has only to take care of the application development and the toolchain hides some implementation details. The test application is composed of four main steps: a gray scale conversion (GS), a Gaussian blur filter (GB), an edge detection filter (ED) and finally a threshold phase (TH).

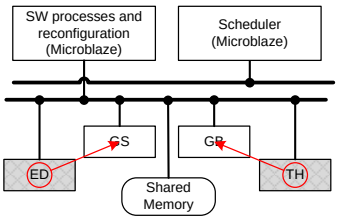

Fig. 3: High-level schema of the evaluated architecture: gray regions are automatically removed in the second experiment.

We initially designed a static architecture, working at 100MHz, implementing all the four computational steps in hardware (designed by hand with a specific protocol represented in the XML file as in [16]), while the reading and the writing of the image have been performed by a software task (assigned to one Microblaze). For the sake of simplicity, the runtime execution is managed by another Microblaze processor. The resulting architecture is shown in Figure 3 and it occupies 5,641 slices of the target device, that is a Xilinx Virtex5 LX110T FPGA (17,280 slices). Then, we profiled only the execution of these four kernel steps that completed in 151,792,313 clock cycles for an image of 1,024x768 pixels.

After that, we would like to reduce the area occupation of the system by exploiting PDR. For this reason, we analyzed the execution time of each phase, reported in Table 1, and we decided to move ED and TH to the regions where GS and GB were executed, respectively. In such a way, we aim at masking the reconfiguration of one region during the execution of the other one. For doing this, through the GUI, we simply changed the mapping of these tasks to the components where the other ones are assigned. *A2B* automatically infers that there are multiple tasks that have been assigned to the same region. Thus the *architecture generation* phase determines the minimal size for each of them, to correct constraints for allowing the implementation. Finally, it implements

Table 1: Execution time and area occupation of each kernel, apart from the core interface that does not require to be reconfigured.

Task	Execution Time [cycles]	Slices
GS	26,824,247	128
GB	58,399,544	276
ED	39,524,613	213
TH	26,896,272	134

the necessary steps to reconfigure those regions and switch the functionality at run-time (including the necessary descriptions in the output project file for partial bitstream generation). It also adds the proper elements to the architecture (e.g., ICAP) and it accordingly modifies the software run-time manager to properly issue both the execution and the reconfiguration. We thus generated the new system in few seconds (except for the time required for synthesis and bitstream generation) and it now occupies 5,321 slices since $A2B$ automatically removed the unused regions, but it requires some extra logic to manage the reconfiguration. Concerning the performance, the two implementations are almost equivalent in terms of overall execution time since, with these decisions, we are effectively able to mask the reconfiguration overhead.

The results show that $A2B$ can be efficiently used to design and prototype reconfigurable systems, hiding most of the details to the designer and opening new possibilities for collaborative research thanks to its modular organization.

6 Conclusions and Future Work

This paper proposed an integrated framework that is currently under development at Politecnico di Milano for the design of reconfigurable systems. It is composed of different steps for the exploration of both architecture and application. It also includes different methodologies and tools to cover the two classical aspects of the system-level design: the decision making and the refinement in order to support the designer as much as possible in the generation of systems ready for the synthesis and the deployment onto the target device. We described how it is possible to integrate different existing methodologies for each of the steps, including the manual intervention of the designer through a practical GUI.

We are currently working on automated methodologies and exploration algorithms for each of the phases (e.g., task graph generation, mapping and scheduling, architecture generation) to develop high-performance systems customized with respect to the applications, along with the integration of commercial tools for HLS (i.e., library generation of the hardware implementations). Finally, we are also evaluating the proposed framework on more complex test cases.

Acknowledgments

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

References

1. Bailey, B., Martin, G., Piziali, A.: ESL Design and Verification: A Prescription for Electronic System Level Methodology. Morgan Kaufmann (March 2007)
2. Sangiovanni-Vincentelli, A., Carloni, L., Bernardinis, F.D., Sgroi, M.: Benefits and challenges for platform-based design. In: Proceedings of the 41st Design Automation Conference (DAC 2004). (2004) 409–414
3. Gerstlauer, A., Haubelt, C., Pimentel, A., Stefanov, T., Gajski, D., Teich, J.: Electronic system-level synthesis methodologies. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **28**(10) (oct. 2009) 1517–1530
4. Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J.H., Brown, S., Czajkowski, T.: LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In: Proceedings of FPGA '11. (2011) 33–36
5. Thompson, M., Nikolov, H., Stefanov, T., Pimentel, A.D., Erbas, C., Polstra, S., Deprettere, E.F.: A framework for rapid system-level exploration, synthesis, and programming of multimedia mp-socs. In: Proceedings of CODES+ISSS '07. (2007) 9–14
6. Banerjee, S., Bozorgzadeh, E., Dutt, N.D.: Integrating physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration. IEEE Trans. Very Large Scale Integr. Syst. **14**(11) (November 2006) 1189–1202
7. Ferrandi, F., Lanzi, P., Pilato, C., Sciuto, D., Tumeo, A.: Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **29**(6) (june 2010) 911–924
8. Gohringer, D., Hubner, M., Benz, M., Becker, J.: A design methodology for application partitioning and architecture development of reconfigurable multiprocessor systems-on-chip. In: Proceedings of FCCM '10. (may 2010) 259 –262
9. Sato, M.: OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors. In: Proceedings of ISSS '02. (2002) 109–111
10. Bertels, K., Sima, V., Yankova, Y., Kuzmanov, G., Luk, W., Coutinho, G., Ferrandi, F., Pilato, C., Lattuada, M., Sciuto, D., Michelotti, A.: Hartes: Hardware-software codesign for heterogeneous multicore platforms. IEEE Micro **30** (2010) 88–97
11. Cordes, D., Marwedel, P., Mallik, A.: Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming. In: Proceedings of CODES/ISSS '10. (2010) 267–276
12. Lattuada, M., Pilato, C., Tumeo, A., Ferrandi, F.: Performance modeling of parallel applications on MPSoCs. In: Proceedings of SOC'09. (2009) 64–67
13. Bonetto, A., Cazzaniga, A., Durelli, G., Pilato, C., Sciuto, D., Santambrogio, M.D.: An open-source design and validation platform for reconfigurable systems. In: Proceedings of FPL '12. (2012) 707 –710
14. Beltrame, G., Fossati, L., Sciuto, D.: Decision-theoretic design space exploration of multiprocessor platforms. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **29**(7) (july 2010) 1083 –1095
15. Pilato, C., Cazzaniga, A., Durelli, G., Otero, A., Sciuto, D., Santambrogio, M.D.: On the automatic integration of hardware accelerators into FPGA-based embedded systems. In: Proceedings of FPL '12. (2012) 607–610
16. Nane, R., van Haastregt, S., Stefanov, T., Kienhuis, B., Sima, V.M., Bertels, K.: IP-XACT extensions for Reconfigurable Computing. In: Proceedings of ASAP '11. (2011) 215–218