AN OPEN-SOURCE DESIGN AND VALIDATION PLATFORM FOR RECONFIGURABLE SYSTEMS

Alessandra Bonetto, Andrea Cazzaniga, Gianluca Durelli,
Christian Pilato, Donatella Sciuto, Marco D. Santambrogio

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

ABSTRACT

Reconfigurable computing is a hot topic for research, as the possibilities and the technology offered by the reconfigurable devices improve year after year both in terms of available configurable logic resources and the possibilities offered to exploit them. This has led CAD tools to grow both in complexity and effectiveness. The expertise required to develop and test a complete system-on-chip using vendors tools has subsequently increased, forcing some designers to create their own tools as support to official development flows. Within this field quite few works have been developed, with respect to the huge effort that has been spent in the exploitation of architectural designs.

ReBit is an open-source tool able to help the designer in exploring different placement solutions in the architecture refinement process and in testing the correct execution of an application on a real device.

1. CONTEXT DEFINITION

To manage the design process of nowadays FPGA-based systems, CAD tools have grown both in complexity and effectiveness. The expertise required to develop and test a complete system-on-chip using vendors tools has subsequently increased, forcing some designers to create their on tools as support to official development flows.

ReBit is not intended to replace Xilinx tools during design and development processes, but to help the user through the validation phase. In particular, Xilinx has released a new version of PlanAhead [1] and ReBit is designed instead on top of it. It aims at increasing the amount of information given to a user as feedback on his work. This work thus addresses the need of a debugging framework for PR architectures, in order to ease the developers' task and give an opportunity for testing real-time issues in a graphical way.

Besides the tools included in the ISE tool suite, other tools have been designed to support the partial reconfiguration flow. An example of such tool is Jbits [2], a Java API which gives an insight into the bitstream configuration files for the Virtex family of devices.

Steiner *et al.* [3] propose an an open-source tool able to read and manipulate both generic and physical netlists, but it does not focus on validation aspects such as testing whether a UCF file is feasible or allowing the user to evaluate the performance of his/her own scheduling and floorplacing solutions. With the same target, Lavin *et al.* [4] implemented *Rapidsmith*, a set of tools and APIs that enable CAD tool creation for Xilinx FPGAs, focusing more on the creation of placers and routers. At the mapped netlist level, Betz and Rose proposed VPR [5], considered the *de facto* place-and-route tool for research, also used for modeling in the development of Altera architectures.

Finally, the previous version of ReBit [6] was mainly focused on validation aspects involved in partial reconfiguration while this new release has been updated removing the BUSMACRO communication infrastructure and it has been extended with two main components: the floorplacer and the scheduler. In such a scenario the main objective of this work is to design an open-source framework able to assist and to guide the designer in implementing a partial dynamic reconfigurable system from its specification to the validation of the runtime support.

The remainder of this paper is organized as follows: Section 2 provides the description of the proposed open-source design and validation framework. Examples on how to use and how to benefit from ReBIT are presented in Section 3, while case studies are introduced in Section 4 and some final remarks are given in the conclusion.

2. THE PROPOSED FRAMEWORK

ReBit is an open-source tool[1] designed to ease the verification process of hardware systems, focused on reconfigurable architectures. The main goal of ReBit is thus to design a framework that is complementary to Xilinx's tools, able to ease the system designers efforts to check their configuration files after the output of the chosen flow and debug possible errors *before* the actual configuration or reconfiguration of the FPGA takes place.

[1] Source code available at http://code.google.com/p/rebit/

PlanAhead [7] gives an automated way to check the configuration files for possible undesired behaviors and design errors. ReBit can be thus used either to perform graphical verification of systems generated, for example, with PlanAhead, or to generate support architectures to speed up the verification of manually partitioned applications.

The tool was entirely developed in C++ and wxWidgets, to ensure portability on different platforms. The graphical interface is organized in panels, aggregating separate functionalities. The available panels are:

- **Project description:** this is the first panel visualized as the tool starts. In this panel it is possible to provide all the input files needed in the other panels.
- **Area Constraints:** if at least two partial bitstreams have been added in the project, it is possible to check if any combination of them generates an area conflict.
- **Relocation:** if a partial bitstream has been added in the project, it is possible to compute and visualize all the possible placements to relocate that functionality.
- **UCF validation:** checks an UCF file for common errors in the area constraints.
- **UCF visualization:** graphical visualization of UCF areas constraints, with the possibility to modify area placements on the fly.
- **Scheduling:** schedules an partitioned application task graph using a specific algorithm.
- **Floorplacing:** floor places a scheduled application to generate a compliant UCF.

Given the data and an architecture description it is possible to automatically check the constraints defined in the Partial Dynamic Reconfiguration Flows and thus relieve the system developer from doing so manually. In addition, the framework has the knowledge of the particular FPGA architecture by means of its RPM grid (RPM stands for Relatively Placed Macro). In addition to the graphical interface, ReBit provides a set of scripts to generate a predefined architecture to be use in an application testing phase. They include ISE/EDK [8] bash scripts and PlanAhead TCL scripts. The user has to run another script, that executes in the correct order all the necessary steps. At first, a base EDK architecture is generated, then XST generates the modules netlists, which are mapped by PlanAhead scripts into the previously generated architecture. Finally, the partial and total bitstreams are generated. HDL implementations of the task cores, along with a compliant UCF file, are needed in this process.

Outputs include updated UCF files, due to the manual UCF modification or the automatic floor placing and updated XML files, modified to include scheduling information. Several hints are provided to the user during the tool usage on the integrated console.

One of the main contribution of ReBit is the modeling of several common components into data structures, implemented as C++ classes. Those classes provide not only structures to store relevant data, but also public APIs to act on those components. Thanks to these APIs, it is possible to easily add new features and algorithms to ReBit and to customize the tool to best fit specific needs.

3. APPLICATIVE SCENARIOS

ReBit includes several features to help the user in validating custom architectures. We identified different tool usages:

- **validation of a manual generated system**: an user manually produced some bitstreams and/or UCF files to be checked for possible errors.
- **validation of an application**: an user implemented an hardware application and he/she wants to verify whether the code execution is correct on a real device, not only in simulation, without having to deal with placement details.
- **custom scheduling and floor placing algorithms verification**: an user wants to implement a custom scheduling or floor placing algorithm and he/she wants to have its work eased by the presence of several API and data structures to deal with the most common tasks, e.g. read an XML file or update an UCF file.

The first two usages are intended for common users who want to use ReBit *as it is*, exploiting its validation features. The latter usage, on the contrary, allows to customize ReBit by means of public APIs and it is intended for those users that want to extend it with new features.

3.1. Validation of a manual generated architecture

Nowadays, a designer that wants to generate an hardware implementation of an application has the availability of many proprietary tools to perform all the steps required. Nevertheless, the steps necessary to generate a complete architecture are very time consuming, and each modification may require to perform the entire process all over again. After the architecture generation, a designer may want to investigate different placement solutions of the generated hardware cores, to try to improve his/her system performance or resource usage. Some area constraints may not be valid and wait until the end of the architecture generation to discover the mistake it is a useless waste of time. ReBit wants to provide a visual interface to perform a preliminary analysis of different solutions and to help less expert users to discover unfeasible placements through a graphical visualization of the proposed constraints. Once a feasible solution is found, the designer can implement its solution to check whether it is

not only feasible, but also more performing. It is possible to discover area conflicts between partial bitstreams or validate their feasible alternative placements, check the UCF constraints for common errors and modify the area constraints through a graphical interface.

3.2. Application validation

Partitioning is a key aspect when dealing with the design of applications for embedded systems. An efficient partitioning is necessary to exploit the parallelism between the different tasks also when using reconfigurability. During the partitioning phase, simulation is the best option to verify the correctness of the work in progress. After that, it is necessary anyway to test the behavior of the partitioned application on a real device. To perform this step, it is necessary to generate all the partial bitstreams related to the different tasks, a test architecture, a scheduler able to reconfigure the regions with the new tasks as needed and a compliant UCF. After all these processes, all the generated artifacts have to be included in a bitstream file to be uploaded on the device.

Those aspects are not related strictly to the application partitioning but they are only related to the application verification process. This verification step can be faster performed using ReBit; the process is composed by the following few steps. The partitioned application, encoded in an XML file, as long as the name of the target device, will be processed by ReBit. The first step is to schedule the application tasks using a defined algorithm, e.g., ASAP or ALAP, to produce a scheduled task graph and the source file of a schedule engine to be run on a Microblaze. This graph will feed the floor plan step that will generate as output an UCF file with the area constraints. The next step is to execute the scripts provided separately with ReBit. Those scripts generate a standard architecture using as input a set of HDL files representing the partitioned application tasks. The UCF file and the scheduler source code are processed and as final output a bitstream for the target device is generated. The user has just to download the bitstream file on the device to test its behavior. Note that the generated architecture does not claim to be the best possible for each application, since it is a multi-purpose generic architecture.

4. USE CASES

In Section 3 we described how ReBit can be used in different scenarios. In this Section, we will propose some examples of how ReBit can be used to test the correct implementation of a hardware application and how it is possible to test the behavior of a custom scheduling technique. The interested reader can find a previous example of bitstream validation in the previous version of this work [6].

Table 1. Estimated execution, reconfiguration times and area occupations provided as inputs to the tool chain

Task	Execution Time [cycles]	Reconfig Time [cycles]	Slices
GS	354853	220376	5120
GB_i	90434	310745	3613
ED_i	78234	290297	2723
TH	190754	160176	3224

4.1. Testing an application

In this test case we want to test the correctness of a hardware implementation of the Canny edge detection algorithm [9]. The application execution is composed of four main steps: a gray scale conversion algorithm (GS), a noise reduction filter that in our case is a Gaussian blur filter (GB), an edge detection filter using Laplace kernel (ED) and finally a threshold phase (TH), that eventually removes pixels that are not clearly part of the edges of the image. It is possible to parallelize some operations. All the four steps are point wise operators, so they could all be replicated and fed with different parts of the input image to perform a parallel computation of the result. In our test case, we have decided to split the two filters (GB and ED) into four parallel nodes that operate separately on each quarter image. The application is thus partitioned into 10 different tasks, corresponding to 10 different task graph nodes. Information about each task includes the execution time, the reconfiguration time and the number of required resources, that are shown in Table 1. A separate HDL file is available for each node.

Since we want to test only the correct execution of the algorithm, we will not impose constraints on the execution time or on the number of area. We will schedule the application task graph using an infinite resources ASAP algorithm. As result, the source code of a scheduler is provided as output, that will execute on a Microblaze processor, and an UCF file with the area constraints.

On a Xilinx Virtex5 VLX110t FPGA, the overall design area occupation is 39,370 slices, equals to 56.9% of the entire target device. The application execution time is 523,521 clock cycles.

4.2. Integrate a new scheduling algorithm

During the application testing, we imposed no limits on the number of areas that could be used to map the application tasks. There are several cases in which such constraints connote be ignored, e.g. if multiple instances of the same application or different application have to be in execution at the same time on the same device. In our specific case, we may want to increase the throughput of images processed by mapping more instances of the edge detector application on the same board. Since the scheduling algorithm available

in ReBit (ASAP and ALAP) are not resource constrained, a designer may want to implement a scheduling custom algorithm that takes into consideration a fixed number of reconfigurable areas. Let AREAS be the vector containing these areas, the scheduling behavior is reported in Algorithm 1.

Algorithm 1 Reconfigurable-aware scheduling algorithm

```
1: while there are still tasks to be scheduled do
2:     area = select(AREAS)
3:     if free(area) then
4:         time = ASAP(t, area)
5:     else
6:         reconfigure(area)
7:         time = ASAP(t, area)
8:     end if
9: end while
```

We have decided to integrate this scheduling algorithm directly into the task graph class. To schedule a task graph using this technique, the method is called *computeRecScheduling* and it takes as argument a fixed number of areas. At first, the application task graph is filled using the *addNode* and *addEdge* functions, using the information retrieved from the XML file. Then, the graph nodes are cleared from previously computed scheduling times and they are set in a topological order, using the corresponding APIs. The number of areas and the availability of each area are taken into consideration. If there are still areas not configured, tasks are scheduled using the ASAP algorithm. If this is not the case, the scheduling time is computed as:

$$T_{start} = max\{(T_{free} + T_{reconf}), T_{dep}\} \qquad (1)$$

where T_{free} is the minimum time when an area becomes ready to be reconfigured, T_{reconf} is the time needed to reconfigure the task and T_{dep} is the time when all parent nodes have finished their execution. Nodes scheduling times are updated consequently and it is possible to print the scheduling using the *printSchedule* function.

Figure 1 shows the results of the execution of the edge detector application, scheduled using the reconfigurable aware algorithm.

5. CONCLUSION

In this paper we have proposed ReBit, an open-source platform to help the user in the verification process of hardware applications. Different possible usages have been proposed, to explore different placements to optimize an architecture performances, to graphically visualize the results of a floor placement, to modify graphically an existing UCF or to test an application behavior without having to deal with scheduling management. In addition, it is possible to customize the tool adding new algorithms or features, exploiting the existing data structures and public APIs.

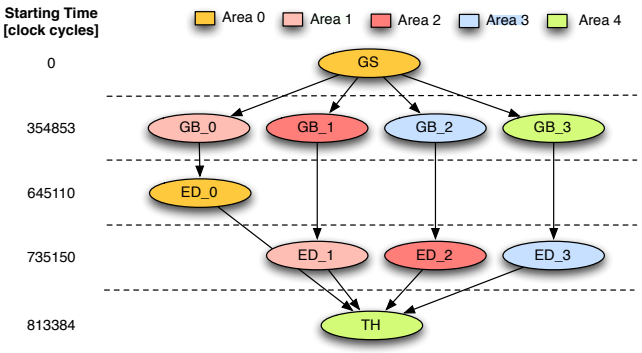

Fig. 1. Execution using the reconfigurable aware schedule and five areas

Acknowledgments

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

6. REFERENCES

[1] Xilinx Inc., *PlanAhead User Guide*, Xilinx Inc., 2012.

[2] S. Guccione, D. Levi, and P. Sundararajan, "Jbits: Java based interface for reconfigurable computing," in *SPIE Proceedings*, vol. 3526, 1998.

[3] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: towards an open-source tool flow," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '11)*, 2011, pp. 41–44.

[4] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapidsmith: Do-it-yourself cad tools for xilinx fpgas," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, sept. 2011, pp. 349 –355.

[5] V. Betz and J. Rose, "Vpr: A new packing, placement and routing tool for fpga research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, ser. FPL '97, 1997, pp. 213–222.

[6] M. Santambrogio, A. Cazzaniga, A. Bonetto, and D. Sciuto, "Rebit: A tool to manage and analyse fpga-based reconfigurable systems," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 220 –227.

[7] X. Inc., "Planahead user guide," Xilinx Inc., Tech. Rep. ug632, December 2009.

[8] ——, "Embedded system tools reference manual, edk 12.1," Xilinx Inc., Tech. Rep. ug111, April 2010.

[9] J. Wu, J. Sun, and W. Liu, "Design and implementation of video image edge detection system based on fpga," in *3rd International Congress on Image and Signal Processing (CISP 2010)*, vol. 1, oct. 2010, pp. 472 –476.