Automating the Layout of Reconfigurable Subsystems Via Template Reduction

Shawn Phillips, Akshay Sharma, Scott Hauck
Department of Electrical Engineering
University of Washington, Seattle, WA
{phillips, akshay, hauck}@ee.washington.edu

1. Introduction

In the traditional FPGA design space there is a limit to the number and variety of FPGAs that can be supported – large NREs due to custom fabrication costs and design complexity means that only the most widely applicable devices are commercially viable. However, a unique opportunity exists in the system-on-a-chip (SoC) design space. Here, an entire system, including memories, processors, DSPs, and ASIC logic are fabricated together on a single silicon die. FPGAs have a role in this design space as well, providing a region of programmability in the SoC that can be used for run-time reconfigurability, bug-fixes, functionality improvements, multi-function SoCs, and other situations that require post-fabrication customization of a hardware subsystem. This gives rise to an interesting opportunity: since the reconfigurable logic will need to be custom fabricated along with the overall SoC, that reconfigurable logic can be optimized to the specific demands of the design.

The goal of the Totem project [1, 2, 3] is to reduce the design time and effort in the creation of a custom reconfigurable architecture. The architectures that are created by Totem are based upon the applications and constraints specified by the designer. Since the custom architecture is optimized for a particular set of applications and constraints, the designs are smaller in area and perform better than a standard FPGA while retaining enough flexibility to support the specified application set, with the possibility to support applications not foreseen by the designer.

2. Totem

The goal of the Totem project is to create tools to generate domain-specific reconfigurable architectures based on designers' needs. One way the Totem project can achieve its goal is to remove as much flexibility as possible from a reconfigurable device, while still supporting the particular algorithms or domain that concerns a designer. While the gains of removing unneeded overhead are apparent, creating a custom reconfigurable architecture is a time consuming and costly endeavor; thus, another goal of the Totem project is to automate the creation of these custom architectures. The overall Totem design flow can be broken into three parts: high-level architecture generation, VLSI layout generation, and place-and-route tool generation.

The focus of this work is the automatic generation of mask layouts, which is performed by the VLSI layout generator. The layout generator will receive, as input from the high-level architecture generator, the Verilog representation of the custom circuit. We are currently investigating three possible methods of automating the layout process: standard-cell generation [1], circuit generators, and template reduction. Here we present the template reduction method.

3. Template Reduction Method

The idea behind template reduction is to start with a full-custom layout that provides a superset of the required resources, and the removal of those resources that are not needed by a given domain. The goal of the Template Reduction Method is not only the removal of unneeded routing resource, but also the removal of unneeded functional units.

During template reduction, the removal of resources is done by automatically editing the layout to eliminate the transistors and wires that form the unused resources, as well as automatically replacing programmable connections with fixed connections or breaks for flexibility that is not needed. In this way, we can get most of the advantage of a full custom layout, while still optimizing towards the actual intended usage of the array.

Template reduction has been broken into three tasks. The first is the creation of a feature rich macro cell, which is used as an initial template that will be reduced and compacted to form the final circuit. The second is the creation of the reduction list that identifies the resources that should be removed. The final task is the implementation of the reductions on the template, followed by the compaction of the resultant circuit.

4. Results on Benchmarks

We are using five sets of netlist to evaluate the template reduction method. All of the netlist sets have been compiled using the RaPiD compiler [4]. The five benchmark sets are:

- Radar – used to observe the atmosphere using FM signals
- Image Processing – a minimal image processing library
- FIR –six different FIR filters, two of which time-multiplex use of multipliers
- Matrix Multiply – five different matrix multipliers
- Sorters – two 1D sorting netlists and two 2D sorting netlists

The template reduction method is able to reduce the number of functional units by an average of 45%, and the routing resources by an average of 75%. Through these reductions, we have found that the template reduction method produces circuits that are on average 53.4% smaller and 13.9% faster than the unreduced template.

5. Conclusions and Future Work

With the advent of SoCs, it is now possible to reduce the NRE cost of creating custom reconfigurable devices. This presents some interesting possibilities for high performance reconfigurable circuits that are targeted at specific application domains, instead of random logic. Automation of the design flow is required if these new custom architectures are to be designed in a timely fashion.

The template reduction method is able to leverage full custom designs, while still removing unneeded resources. This enables it to create circuits that perform at or better than that of the initial full custom template. In this work we have shown that the automation of the layout portion of the design flow is possible using a template reduction methodology. Through profiling we have created a feature rich macro cell as our template. We have found that the template reduction method produces circuits that are 53.4% smaller and 13.9% faster than the unreduced template.

It is becoming evident that no single method is able to produce architectures that meet a designer's constraints in all cases. Template reduction works well when a specified architecture is a subset of an existing full-custom template, and thus is able to leverage the benefits of full-custom design. The standard-cell method, on the other hand, can support any arbitrary FPGA design, even though circuits created by this method have decreased performance and an area penalty when compared to full-custom designs.

In future work we will be investigating the circuit generator method, which will complement the template reduction and standard-cell methods. The combination of these three methods will provide a wide spectrum of approaches, with each being the appropriate method in different situations. In the end, we hope to provide designers with the ability to create reconfigurable devices that were historically the province of ASICs.

6. Acknowledgements

The authors would like to thank the RaPiD group, especially Carl Ebeling and Chris Fisher, for the RaPiD-I layout used in this research. We also are indebted to Larry McMurchie for support on the Cadence tool-suite. This work was funded in part by grants from NSF and NASA. Scott Hauck was supported in part by an NSF CAREER award and an Alfred P. Sloan Research Fellowship.

7. References

[1] S. Phillips, S. Hauck, "Automatic Layout of Domain-Specific Reconfigurable Subsystems for System-on-a-Chip", ACM/SIGDA Symposium on Field-Programmable Gate Arrays, pp. 165-173, 2002.

[2] K. Compton, S. Hauck, "Totem: Custom Reconfigurable Array Generation", IEEE Symposium on FPGAs for Custom Computing Machines Conference, 2001.

[3] K. Compton, A. Sharma, S. Phillips, S. Hauck, "Flexible Routing Architecture Generation for Domain-Specific Reconfigurable Subsystems", International Conference on Field Programmable Logic and Applications, pp. 59-68, 2002.

[4]. D. C. Cronquist, P. Franklin, S.G. Berg, C. Ebeling, "Specifying and Compiling Applications for RaPiD", *IEEE Symposium on FPGAs for Custom Computing Machines* 1998