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

Abstract. When designing SoCs, a unique opportunity exists to generate custom FPGA architectures that are specific to the application domain in which the device will be used. The inclusion of such devices provides an efficient compromise between the flexibility of software and the performance of hardware, while at the same time allowing for post-fabrication modification of circuits. To automate the layout of reconfigurable subsystems for systems-on-a-chip, we present template reduction. Template reduction enables a designer to eliminate resources from a template that are unnecessary to support the specified application domain. To facilitate this, we have created a feature rich template, from which we automatically generate application specific reconfigurable circuits. Compared to the full template, we achieve designs that are 53.4% smaller and 13.9% faster, while continuing to support the algorithms in a particular application domain.

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. 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, 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, 4] 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 Background

2.1 RaPiD

The reconfigurable-pipelined datapath (RaPiD) [5] has been chosen as a starting point for the architectures that we will be generating. The goal of the RaPiD-I architecture is to provide performance at or above the level of a dedicated ASIC, while also retaining the flexibility that reconfigurability provides. RaPiD-I is able to achieve these goals through the use of course-grain components, such as memories, ALUs, multipliers, and pipelined data registers.

Along with coarse-grain components, the initial RaPiD-I architecture consists of a one-dimensional routing structure, instead of a standard FPGA's two-dimensional interconnect. The RaPiD-I architecture is able to take advantage of the reduction in complexity that a one-dimensional routing structure provides because all of its computational units are word-width devices. This structure has proven effective in supporting high-performance signal processing applications [5].

2.1 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 remove those resources that are not needed by a given domain. One example of a similar approach to template reduction in industry

is eASIC's FlexASIC™ [6]. Their approach enables designers to remove unneeded routing resources by the elimination of vias, creating a reconfigurable device that is similar to an anti-fuse based design. The goal of the Template Reduction Method is to not only remove unneeded routing resource, but to also remove 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.

3.1 Template Cell

Extensive profiling has been performed to create the feature rich template. This led us to the cell, called RaPiD-II, which is a more feature rich version than the original RaPiD-I cell. The increase in resources is required, since we have found that the original full-custom RaPiD-I cell does not have enough interconnect resources to handle some of the benchmarks intended for the architecture [4]. RaPiD-II addresses this issue because it has 24-buses and three bus-connectors per functional unit, compared to RaPiD-I, which has 14-buses and one bus-connector per functional unit. In addition to the increase in routing resources, the RaPiD-II cell that was chosen had to have a rich enough resource mix of functional units to support a large set of applications. Note that RaPiD-II isn't an architecture chosen just for template reduction, but instead is the RaPiD tile we believe is the best for implementation in any methodology, including full custom tiles. Template reduction will work on RaPiD-II, the original RaPiD-I, or any other premade tile that has at least enough resources to support the desired circuits.

3.2 Reduction List Generation

The next task in template reduction is the creation of the reduction list. Towards this end we have implemented a subtractive scheme that eliminates as many functional units and routing resources, collectively called "resources", as possible while placing and routing a set of netlists onto the template architecture. Individual netlists are placed using a simulated annealing approach [2], and routed using the Pathfinder algorithm [7]. Initially, all netlists in the set are individually placed and routed on the template architecture. At the end of this first run, the fraction of netlists that used each resource in the template is recorded, and a cost (referred to as *usage_cost*) is

assigned to each resource based on the fraction of netlists that used the resource during the previous run.

After completion of the first run on all netlists, a second run is commenced during which the netlists in the set are individually placed and routed again on the template architecture. However, for any given netlist, the cost of using a resource during the second run is influenced by the *usage_cost* of that resource. At the end of the second run, the *usage_cost* of each resource is again adjusted in a manner identical to that at the end of the first run, and a third run is begun. Once the three runs are completed, we have a list of the resources that can be eliminated from the template architecture.

3.3 Reduction and Compaction

Once the reduction list is generated, the final task is to automatically edit the template, followed by a compaction step to reduce the template size. To reduce the template, the layouts were automatically edited within the Cadence CAD tools. To achieve the required automation Cadence SKILL [8] routines were written for each reduction that the subtractive method performs.

First among the reductions is the elimination of any unused cells (that is, complete RaPiD-II tiles). The next reduction is the elimination of any functional units in any cell that are not needed. Next, we remove any of the bidirectional bus-connectors that are not needed in the interconnect. The final reduction is the removal of any unused wires. When an unused wire is removed, the corresponding transistors and programming bits in any muxes and drivers on the wire are also removed. Once all of the reductions have been preformed the final design is compacted.

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 [9]. 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

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. We have created the RaPiD-II cell, since the RaPiD-I cell was not able to implement all of the circuits from the benchmark suite, circuits that it was targeted to support. We have found that the template reduction method produces circuits that are 53.4% smaller and 13.9% faster than RaPiD-II.

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.

References

1. S. Phillips, Automatic Layout of Domain-Specific Reconfigurable Subsystems for System-on-a-Chip, M.S. Thesis, Northwestern University, Dept. of ECE, July 2001.
2. A. Sharma, Development of a Place and Route Tool for the RaPiD Architecture, M.S. Thesis, University of Washington, 2001.
3. K. Compton, S. Hauck, "Totem: Custom Reconfigurable Array Generation", IEEE Symposium on FPGAs for Custom Computing Machines Conference, 2001.
4. 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.
5. C. Ebeling, D. C. Cronquist, P. Franklin, "RaPiD – Reconfigurable Pipelined Datapath", 6th Annual Workshop on Field Programmable Logic and Applications, 1996.
6. Application Specific Programmable Platform using eASICore® Whitepaper Version 1.0, http://www.easic.com/technolgy/whitepapers.html, March 2004.
7. L. E. McMurchie and C. Ebeling "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs", Symposium on Field-Programmable Gate Arrays, pp.111-117, 1995.
8. Cadence Design Systems, Inc., "Openbook", version 4.1, release IC 4.4.5, 1999.
9. 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.