Automatic Layout of Domain-Specific Reconfigurable Subsystems for System-on-a-Chip

Shawn Phillips
University of Washington
Department of Electrical Engineering
Seattle, WA 98195
phillips@ee.washington.edu

Scott Hauck
University of Washington
Department of Electrical Engineering
Seattle, WA 98195
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 a device will provide 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 system-on -a-chip we present template reduction, standard cell, and circuit generator methods. We explore the standard cell method, as well as the creation of FPGA-specific standard cells. Compared to full custom circuits, we achieve designs that are 46% smaller and 36% faster when the application domain is well known in advance. In cases where no reduction from the full functionality is possible, the standard cell approach is 42% larger and 64% slower than full-custom circuits. Standard cells can thus provide competitive implementations, with significantly greater opportunity for adaptation to new domains.

General Terms

Performance, Design, Experimentation

Keywords

Domain-Specific FPGA, Standard Cells, Automatic Layout Generation, System-on a-Chip

1. INTRODUCTION

With the advent of new fabrication technologies, designers now have the ability to create integrated circuits utilizing over one hundred million gates, with operating frequencies in the GHz range. This large increase in transistor count has increased the complexity of devices, but it is also enabling designers to move away from the well known system-on-a-board to a heterogeneous system-on-a-chip (SOC) methodology [5]. This evolution in integration is driven by the need to reduce the overall cost of the design, increase inter-device communication bandwidth, reduce power consumption, and remove pin limitations.

There are several drawbacks to the SOC design methodology. Designers of SOCs have a larger design space to consider, an increase in prototyping costs, a more difficult job of interfacing components, and a longer time to market. There is also a loss in post-fabrication flexibility. In the system-on-a-board approach, designers have the ability to customize the system by careful selection of components, with easy component replacement in late stages of the design cycle. But in the current SOC design methodology framework, in which only ASIC components are used, very tight integration is the goal. Therefore, component changes late in the design cycle are not feasible.

This loss of post-fabrication flexibility can be alleviated with the inclusion of FPGA logic onto the SOC. Unlike application specific integrated circuits (ASICs), by including FPGAs, designers would gain the ability to alter the SOC to meet differing system requirements after the SOC has been fabricated. However, FPGAs are often several times slower, larger, and less energy efficient than ASICs, making them a less ideal choice for high performance, low power designs. Domain-specific FPGAs can be utilized to bridge the gap that exists between flexible FPGAs and high performance ASICs.

A domain-specific FPGA is a reconfigurable array that is targeted at specific application domains, instead of the multiple domains a standard FPGA targets. Creating custom domain-specific FPGAs is possible when designing an SOC, since even early in the design stage designers are aware of the computational domain in which the device will operate. With this knowledge, designers could then remove from the reconfigurable array hardware and programming points that are not needed and would otherwise reduce system performance and increase the design area. Architectures such as RaPiD [4], PipeRench [6], and Pleiades [1], have followed this design methodology in the digital signal processor (DSP) computational domain, and have shown improvements over reconfigurable processors within this space. This ability to utilize custom arrays instead of ASICs in high performance SOC designs will provide the post-fabrication flexibility of FPGAs, while also meeting stringent performance requirements that until now could only be met by ASICs.

Possible application domains could include signal processing, cryptography, image analysis, or any other computationally intensive area. In essence, the more that is known about the target applications, the more inflexible and ASIC-like the custom array can be. On the other end of the spectrum, if the domain space is only vaguely known, then the custom array would need to provide the ability to run a wide range of applications, and thus look and perform more like a standard FPGA.

Since all of the components in an SOC need to be fabricated after integration, this provides designers with a unique opportunity to insert application specific FPGAs into their devices. Unfortunately, if designers were forced to create custom logic for each domain-specific subsystem, it would be impossible to meet any reasonable design cycle. However, by automating the generation of the application specific FPGAs, designers would avoid this increased time to market and would also decrease the overall design cost.

These factors have led us to start the Totem project, which has the ultimate goal of automatically generating custom reconfigurable architectures based upon the perceived application domain in which the device will be used. Since the custom array will be optimized for a particular application domain, we expect that it will have a smaller area and perform better than a standard FPGA, while retaining most of the benefits of reconfigurability.

First we present a short background on RaPiD and Totem. Next, we examine the approach and experimental setup that we have taken to automate the layout of a domain-specific reconfigurable subsystem. Finally, we will show how our approach was able to create circuits that perform within Totem's design specifications, paving the way for future work in providing custom reconfigurable subsystems in SOCs.

2. BACKGROUND

2.1 RaPiD

We are using the reconfigurable-pipelined datapath (RaPiD) architecture as a starting point for the circuits that we will be generating [4]. RaPiD is positioned between standard FPGAs and ASICs. Its goal is to provide the performance of an ASIC while maintaining reconfigurability. RaPiD, like an FPGA, achieves reconfigurability through the use of block components such as memories, adders, multipliers, and pipeline registers. But, unlike a commercial FPGA, RaPiD is not targeted at random logic, but at coarse-grained, computationally intensive functions like arithmetic.

RaPiD utilizes a one-dimensional structure to take advantage of the fact that all of its functional components are word-width computational devices. One of the advantages of a one-dimensional structure is a reduction in complexity, especially in the communications network. Another advantage is the ability to map systolic arrays very efficiently, leveraging all of the research into the compilation of algorithms onto systolic arrays. Finally, while a two dimensional RaPiD is possible, most two-dimensional algorithms can be mapped onto a one-dimensional array through the use of memory elements.

The version of RaPiD that we are benchmarking against, RaPiD I, consists of memories, ALUs, and multipliers that are all connected by a one-dimensional segmented routing structure. Data flows through the array along the horizontal axis, with the vertical axis being used only to provide connections between functional units. To create different versions of RaPiD that target different application domains, the following changes need to be made to the array: modify existing or add new functional units, change the width of the buses or the number of buses present, and modify the routing fabric. Figure 1 shows an example of one possible RaPiD cell. Multiple cells would be tiled along the horizontal axis.

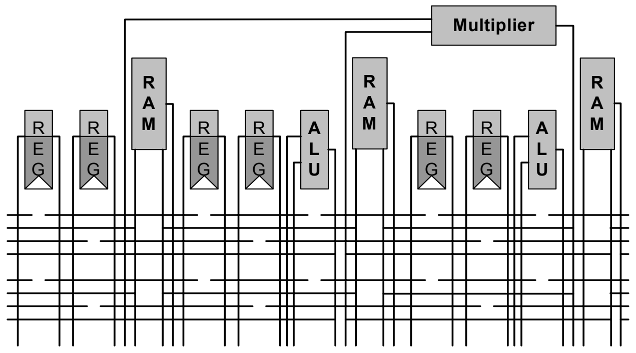


Figure 1. A block diagram of a basic RaPiD cell. Multiple cells are tiled along the horizontal axis.

2.2 Totem

Reconfigurable hardware is a very efficient bridge that fills the gap between software implementations running on general-purpose processors and ASIC implementations. But, standard reconfigurable hardware targets the general case, and therefore must contain a very generic mix of logic elements and routing resources that are capable of supporting all types of applications. This creates a device that is very flexible, allowing for bug fixes, upgrades, and runtime reconfiguration, among others. Yet, if the application domain is known in advance, optimizations can be made to make a compact design that is more efficient than commercial FPGAs. While the benefits of creating a unique FPGA for each application domain are apparent, in practice the design of a new FPGA architecture for each and every application space would require an increase in design time and create significant additional design costs. The goal of the Totem project is the automatic generation of domain-specific FPGAs, giving designers a tool that will enable them to benefit from a unique FPGA architecture without the high cost and lengthy design cycle. The automatic generation of FPGAs can be broken into three major parts: high-level architecture generation, VLSI layout generation, and place-and-route tool creation.

The high-level architecture generator will receive, as input from the designer, information about the set of applications that the SOC will target. The architecture generator will then create a coarse-grained FPGA that consists of block components such as memories, adders, multipliers, and pipeline registers [3]. Routing resources will then connect these components to create a one-dimensional structure. Extending Totem to create two-dimensional arrays of functional units is possible, and will be explored in future research.

The final structure that is created will fall somewhere on the scale between ASICs and commercial FPGAs. Where on this scale the final device will fall depends on how much information the designer is able to provide in advance about the applications that will run on the reconfigurable hardware, and how similar those applications are in composition. If the designer can provide a lot of information, and the applications are similar in composition, then more hardware and connection points can be removed from the FPGA, thus generating a more compact and higher performing design. On the other side of the scale, if the designer does not know which applications will run on the SOC, or the applications that will run on the SOC are very different, then more reconfigurable components will be needed to support a wider range of logic, causing the final design to be more like a commercial FPGA. After the high-level architecture generator creates an architecture that is able to support the applications that will run on it, this information is disseminated to both the VLSI layout generator and the place and route tool generator.

The VLSI layout generator has the task of creating a fabrication-ready layout for the custom device by using the specifications that were provided by the high-level architecture generator. The layout generator will be able to create a layout for any possible architecture that the high-level architecture generator is capable of producing. The difficult task for the layout generator is creating efficient designs, so as not to squander the performance and area gains that the architecture generator was able to achieve

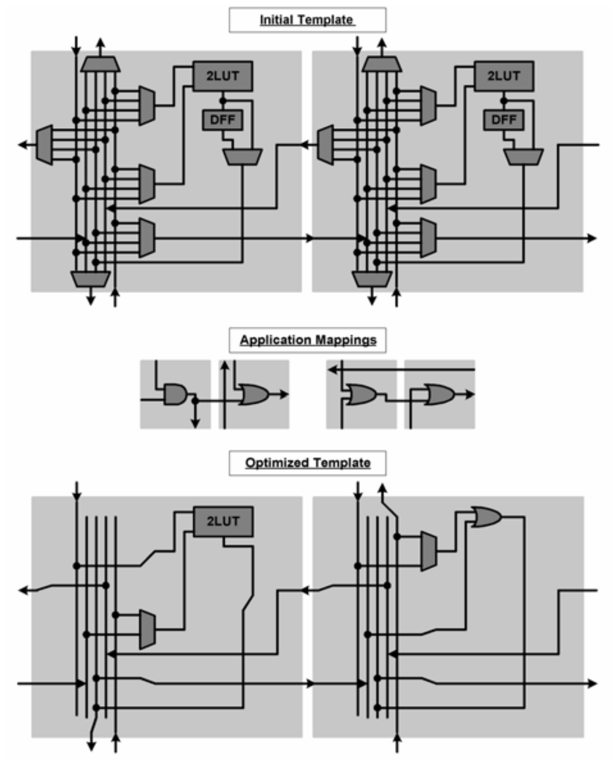


Figure 2. Simplified example of template reduction. The initial template (top) is a modified version of the Xilinx XC6200. The high-level architecture generator has found two target applications (middle). Logic resources that are not needed, including routing resources, are removed from the initial template to create the optimized template (bottom). Notice how both DFFs have been removed and how the 2LUT in the right cell is reduced to an OR gate.

over general-purpose FPGAs. The layout generator will be flexible enough to change over time to take advantage of smaller device sizes as process technology scales down. In addition, the layout generator will produce a bit-stream format that the place and route tools will be able to use to configure the custom FPGA. Three different methods are being explored to automate the layout process: template reduction, standard cells, and FPGA-specific circuit generators. Each of these will be discussed in greater detail later in this paper.

Once the custom architecture is created, the end user will then need a tool set that will automatically generate mappings that target the custom array. The place and route tool generator will create a physical mapping of a user application by using an architecture description that was created by the high-level architecture generator and the bit-stream format that was created by the VLSI layout generator. It does this task by the use of a placement tool, which is based on the simulated annealing algorithm, and a router that is an adaptation of the Pathfinder algorithm [7].

3. APPROACH

Current design methodologies for the layout of circuits typically fall under either full-custom design or standard-cells, with both of these approaches having associated pros and cons. Producing a full-custom circuit is a labor-intensive task, which requires a very long and expensive design cycle. However, the resulting circuit is created is usually the fastest and the smallest that is possible at that time. Generating a standard cell library can be a difficult endeavor, and therefore companies that have extensive libraries vigorously guard them from competitors. However, once the library is created, the ability for indefinite reuse and design automation justifies both the time and expense involved. Unfortunately, circuits that are created using standard cells are larger and slower than full-custom designs [11]. One of the goals of the Totem project is to automate the generation of FPGAs that begin to approach the level of performance that full-custom layouts currently enjoy.

The Totem project has decided to investigate three different approaches to automate the layout process: standard cells, template reduction, and FPGA-specific circuit generators. All three methods will be outlined in the following sections. A goal of the Totem project is to decide which of the three approaches should be used in a particular situation. We may find that one approach is the best for all situations, or that each approach has compelling characteristics that make it the best choice in a particular instance. The objective of this paper is to investigate the standard cell approach. Towards this end, we present the creation of a standard cell tool flow and the investigation of how standard cell templates compare to a full custom RaPiD array.

We further refine the standard cell approach by adding cells to the library that are used extensively in FPGA designs, thus creating an optimized library. These include muxes, demuxes, and SRAM bits, among others. We predict that by adding a few critical cells, the results obtained can be significantly enhanced.

3.1 Template Reduction Method

The template reduction method will leverage the performance edge that full-custom layouts provide, in an automated fashion. This is achieved by using feature rich macro cells as templates that are reduced and compacted to form the final circuit. Providing quality macro cells that can cover a wide range of applications is critical to the success of this method. Therefore, extensive profiling of application domains will be required to establish what resource mixes are needed for each template.

The potential exists to create designs that achieve a performance level that is at parity with that of ASICs. But this is only possible if two conditions are met: the applications that the template needs to support are similar in composition, and the domain that contains the applications is a subset of the initial template. The first condition implies that if a template is required to support a wide range of applications, then it will have to retain most of its reconfigurability. This means that the performance of the final template will be near that of an FPGA, which, as noted earlier, is usually not optimal. While the first condition may mean the final template may not perform well, the second condition, if not met, may mean that the applications specified cannot be mapped onto the initial template. This could occur, for example, if the initial template does not contain a multiplier, but the applications that need to run on the template require one.

With these conditions in mind, if the design specified by the architecture generator does not deviate significantly from the available macro cells, we can use the template reduction method to automate the layout generation in a fast and efficient manner without sacrificing performance. Since our initial focus is on a one-dimensional array as our target FPGA, we will be using variations of the RaPiD architecture as a basis for our feature rich macro template.

3.2 Standard Cell Implementation

The use of template reduction produces very efficient implementations, but it only works well if the proposed architecture does not deviate significantly from the provided macro cells. To fill the gaps that exist between templates' domains, we have implemented a standard cell method of layout generation. This method will provide Totem with the ability to create a reconfigurable subsystem for any application domain.

Using standard cells also creates an opportunity to more aggressively optimize logic than if templates were used, since the circuit can be built from the ground up. It will also allow the designer to easily integrate this method into the normal SOC design flow. In addition, the structures created will retain their reconfigurability since the CLBs and routing interconnect will be programmable. This is achieved by creating a structural Verilog representation of the FPGA, and then generating a standard cell layout based upon that Verilog. Since the Verilog design includes SRAM bits and programmability, the result is a reconfigurable ASIC.

Unfortunately, this method also inherits all of the drawbacks introduced into a design by the use of standard cells, including increased circuit size and reduced performance. To overcome these failings, we will create standard cells that are often used in FPGAs. These cells will include LUTs, SRAM bits, muxes, demuxes, and other typical FPGA components. Some of these units are shown in Figure 3. Since these cells are used extensively in FPGAs, significant improvement could be attained. In this work we compare a standard cell library and a more comprehensive optimized library.

3.3 FPGA-specific Circuit Generators

The standard cell design method is very flexible, and it gives the designer the ability to implement almost any circuit. However, one drawback associated with the flexibility of this design method is its inability to leverage the regularity that exists in FPGAs. By taking advantage of this regularity, a method may produce designs that are of higher quality than standard cell based designs.

One way of creating very regular circuits is through the use of generators. Circuit generators are used to great effect in the memory industry, and it is our belief that we will be able to achieve similar results. FPGA components, like memories, have well-known, constrained structures, positioning them as viable candidates for circuit generators.

Circuit generators will be able to create structures that are of higher quality than those created by the standard cell method. However, unlike the template reduction method, the circuit generators will be able to handle a wider variety of possible architectures. Thus, circuit generators will be positioned to fill the gap between the inflexible, but powerful, template reduction method and the very flexible, but less efficient, standard cell method.

Circuit generators will be implemented to create the parts of the FPGA that inherently have regularity. This includes generators for the routing channels, LUTs, and muxes and demuxes for routing interconnect. To create an entire reconfigurable subsystem out of blocks of logic that circuit generators have created, one would only need to abut the blocks together. Therefore, all of these generators will be combined to create a method that is capable of generating a complete reconfigurable subsystem.

4. EXPERIMENTAL SETUP AND PROCEDURE

4.1 Setup

To retain as much flexibility as possible in our standard cell implementation, behavioral Verilog representations were created

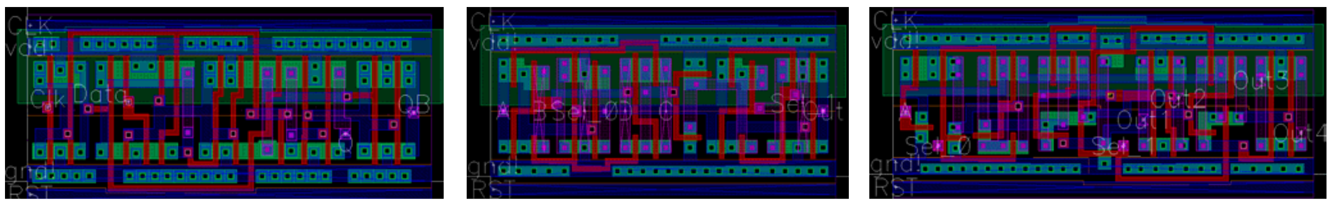


Figure 3. FPGA-specific standard cells: (left) 1-bit DFF, (middle) 4:1 mux, (right) 1:4 demux

for all of the RaPiD components. Synopsys was used to synthesize the behavioral Verilog to produce structural Verilog

Table 1. Template nomenclature and description, reflecting a range of realistic domain-specific optimizations.

Template	Description
Not_reduced	Full template
PA	Removed pipeline registers after the second ALU
PB	Removed pipeline registers before the first ALU
ALU	Converted the ALUs to adders
PB_PA	Removed pipeline registers before the first ALU and after the second ALU
PA_ALU	Removed pipeline registers after the second ALU and reduced the ALUs to adders
PB_ALU	Removed pipeline registers before the first ALU and reduced the ALUs to adders
PB_PA_ALU	Reduced the ALUs to adders and removed pipeline registers before the first ALU and after the second ALU

that uses our standard cells [8]. This will enable us to swap out standard cell libraries, since we would only need to re-synthesize the behavioral Verilog with a new library file generated for the new standard cell library.

Silicon Ensemble (SE) was used to place and route the cells. SE is part of the Cadence Envisia Tool Suite, and is capable of routing multiple layers of metal, including routing over the cells. One powerful feature of SE is its ability to run from macro files, minimizing the amount of user intervention.

Cadence was chosen as our schematic and layout editor because it is a very robust tool set that is widely used in industry [2]. Cadence also has tools for every aspect of the design flow. We are currently using the TSMC 0.25µm design rules for all layouts created in Cadence. As technology changes, we will be able to scale our layouts down without a loss of quality in our results.

The full custom RaPiD components that were used in benchmarking were laid out by Carl Ebeling's group for the RaPiD I powertest. All circuits were laid out using the Magic Layout Editor for the HP 0.35µm process. The designs were ported over to Cadence and the TSMC 0.25µm process.

The choice of a standard cell library was based upon the need to find an industrial strength library that has been laid-out for the TSMC 0.25µm process. This led us to the Tanner standard cell library that is available through the MOSIS prototyping production service [10]. This library has thirty-two basic blocks at its core, which can then be applied to produce any combination of the 1400+ functional blocks that Tanner provides in its Tanner SchemLib symbol library.

We use the Epic Tool Suite to analyze the performance of all of the circuits that have been created. Synopsys has developed the Epic Tool Suite as a robust circuit simulator that enables designers to verify circuit performance at both pre-layout and post-layout without fabrication of the design [8]. The Epic tools use a version of the SPICE engine for circuit simulation, and they also use the SPICE netlist format as circuit input and the SPICE BSIM3V3 as a transistor model format. The tool that we will be mainly using from the tool suite is Pathmill.

4.2 Procedure

The experimental procedure was driven by our use of RaPiD as a starting point, and the use of the tools that were mentioned in the experimental setup section. While there is still considerable manual intervention involved in each step of the flow, our eventual goal is a truly automated process.

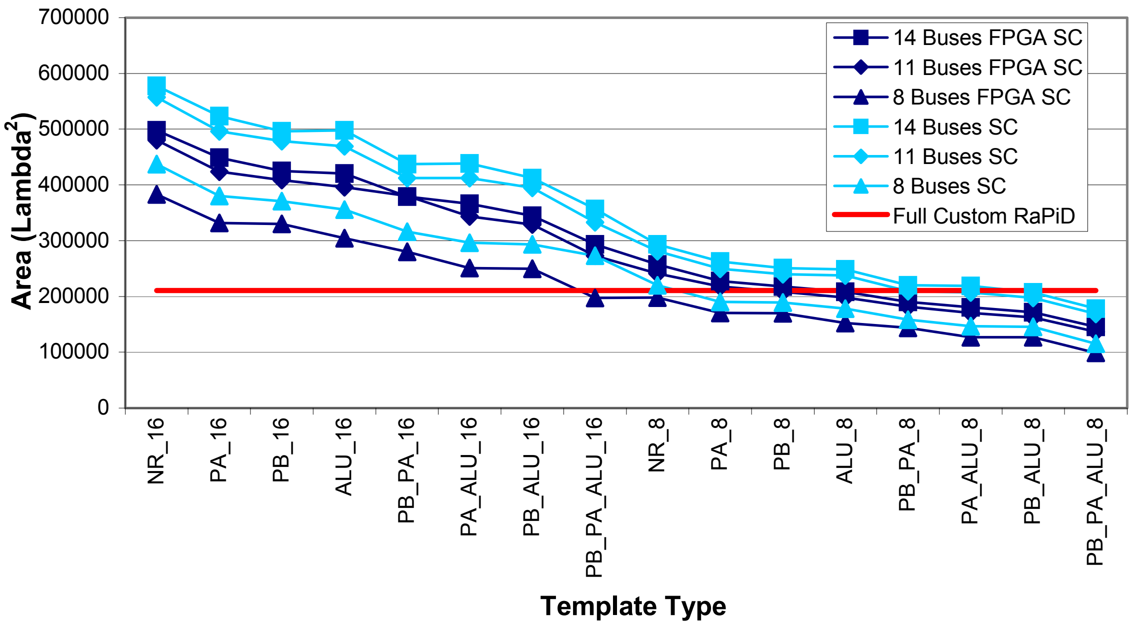
We first imported the RaPiD I powertest components from the Magic Layout Editor using a HP 0.35µm process to Cadence using a TSMC 0.25µm process. To do this, the files were first exported out of the Magic Layout Editor in a CIF file. We then proceeded to modify these CIF files to force compatibility with the TSMC 0.25µm process. Once this was done, the files were then imported into Cadence, and all remaining design errors were corrected by hand. Schematic and Verilog representation of the RaPiD components were also created.

The next step was to find an appropriate standard cell library. As stated above, we settled on the Tanner Standard Cell library. Even though the library was targeted at the TSMC 0.25µm process, the layouts were generated using the more aggressive deep sub-micron version of the process with a lambda of 0.12µm. However, we are currently using the deep-micron version of the TSMC 0.25µm process with a lambda of 0.15µm. This caused some minor problems that were cleaned up by hand using pre-import scripts and some post-import modifications. A library information file representation of the Tanner Cells was also created for Synopsys.

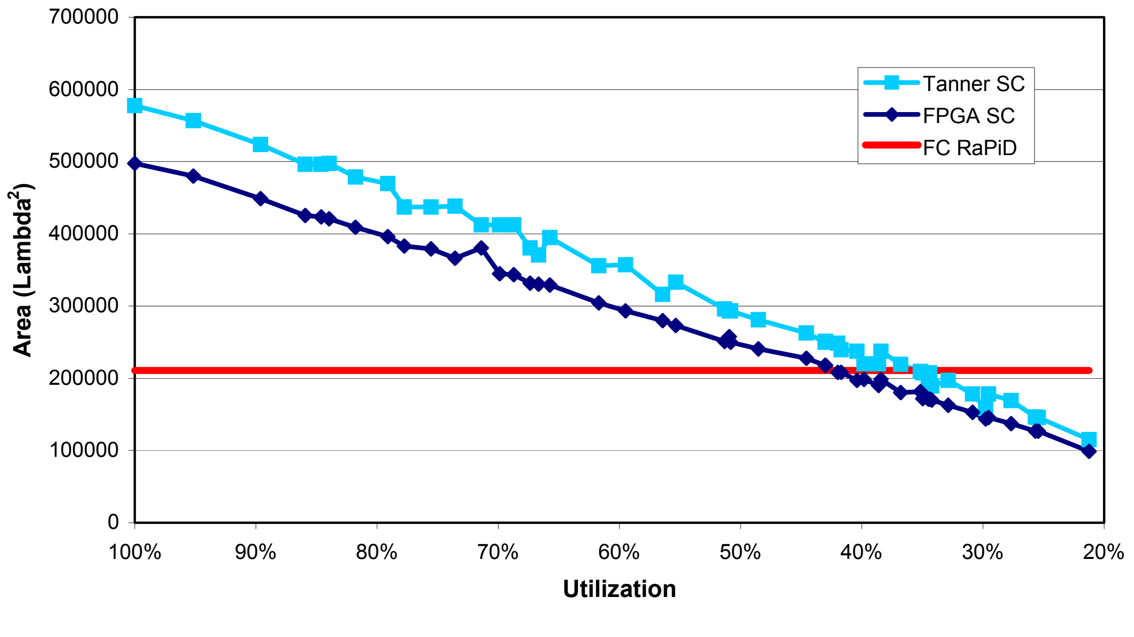
To generate the standard cell version of RaPiD, the tool-flow discussed earlier was used. A behavioral Verilog representation of RaPiD was first created. Synopsys was then used to synthesize this Verilog file to create a structural Verilog file that used the Tanner standard-cells as modules. With this structural Verilog, SE was able to then place-and-route the entire design. The utilization level of SE, which is an indication of how dense cells are packed in the placement array, was increased until the design could not be routed. For most designs this level was set to 90%. The aspect ratio of the chip was also adjusted from 1, which is a square, to 2, which is a rectangle that is twice as long as it is high, to find the smallest layout. For all designs, an aspect ratio of 1 yielded the smallest layout. Once SE was done creating the layout, the EPIC tool-set was used to evaluate the quality of the circuit that was created.

5. RESULTS

The tool-flow and procedure were discussed in detail in the experimental setup and procedure above. The tools were run on nine Sun Ultra Five workstations, four machines with 512MB of memory and 5 machines with 384MB of memory. The runtime



Graph 1. This graph shows the area of full custom RaPiD as well as all of the different versions of the templates in l2 vs. template types. The y-axis is the area of the templates in l2, while the x-axis is a list of each version of template, from most feature rich (left) to least (right).



Graph 2. This graph shows the area of full custom RaPiD, as well as all of the different versions of the templates in l2. The y-axis is the area of the templates in l2, while the x-axis is the percent utilization of the original template, from most feature rich (left) to least (right).

of the entire tool-flow to generate each template was approximately six hours.

As a template is customized towards an application domain, gains in performance and reductions in power and area are possible. To test this, the RaPiD architecture templates were optimized to reflect possible design scenarios. The scenarios included reducing the ALU to an adder, reducing the word size, reducing the number of pipeline registers, and all of the associated cross products. The scenarios were created to reflect reasonable design tradeoffs. For example, if a circuit will be used to support applications that only need to perform arithmetic, the designer could convert the ALUs to adders to increase performance and reduce circuit size. Table 1 describes the eight different designs that were created. These designs were further implemented with 8 buses, 11 buses, and 14 buses for both the Tanner and FPGA standard cells. This reflects varying the richness of the interconnect in the system. Finally, all designs were created with both an eight-bit and a sixteen-bit word size.

5.1 Area

Graph 1 and Graph 2 show the area of the templates as well as full custom RaPiD in units of λ^2. Graph 1 shows the impact of each design scenario, varying from the original RaPiD (left) to a highly reduced version (right). Graph 2 converts this to % utilization of the original, full RaPiD template, measured by the proportion of transistors retained by the reduced templates. As can be seen, the full-custom RaPiD is 2.7x smaller than the standard cell version. However, modifications to the architecture can reduce this impact, achieving up to a 2.1x smaller design with Standard Cells than with the full custom, unreduced RaPiD tile. This demonstrates the benefits that are possible by optimizing the FPGA architecture to the application domain of the SOC. Much of this benefit comes from a reduced word size (switching from 16-bit to 8-bit templates), and by reducing the number of routing registers in the pipelined interconnect. Switching to an optimized standard cell library, by adding 4 FPGA-specific cells to a generic library, achieves an additional reduction of 9% to 18.9%.

5.2 Performance

Graph 3 and Graph 4 are similar to the previous ones, but in this case present the performance of the resulting templates. As described in the experimental setup and procedure section, performance numbers were generated using PathMill to find the longest-path delays.

The performance numbers also scale with reductions in the amount of resources in the FPGA, but in a much less linear manner. While the elimination of each transistor in the design has an approximately equal impact on the area of the overall design, the improvements in performance depend much more heavily on what specific transistors are removed. The most striking feature of the graph, the sudden dips, can be attributed to the fact that the reduction of the number of buses or of the bit width does not affect performance nearly as much as converting the ALU to an adder. Specifically, the replacement of an ALU with an Adder (which improves area on average by only 19%) yields on average a 27% improvement in the performance, while switching to an 8-bit word size (which approximately halves the chip area) only achieves a 13% performance gain. To separate out these effects, the templates were grouped as shown in Graph 5 based on ALU vs. adder and 16-bit vs. 8-bit word-size. Also, while there are not any 16-bit templates that have a smaller size than the full custom RaPiD, there are eight FPGA standard cell templates and three Tanner standard cell templates that have a shorter critical path. These benefits range from 7% to 36%.

6. CONCLUSIONS

As SOCs move into the mainstream, it is likely that FPGAs will play a major role in providing the post-fabrication modification that these devices will require. This presents some interesting opportunities for creating high performance FPGAs that are targeted at specific application domains, instead of random logic. To implement these new architectures in a timely fashion, automation of the design flow is a necessity.

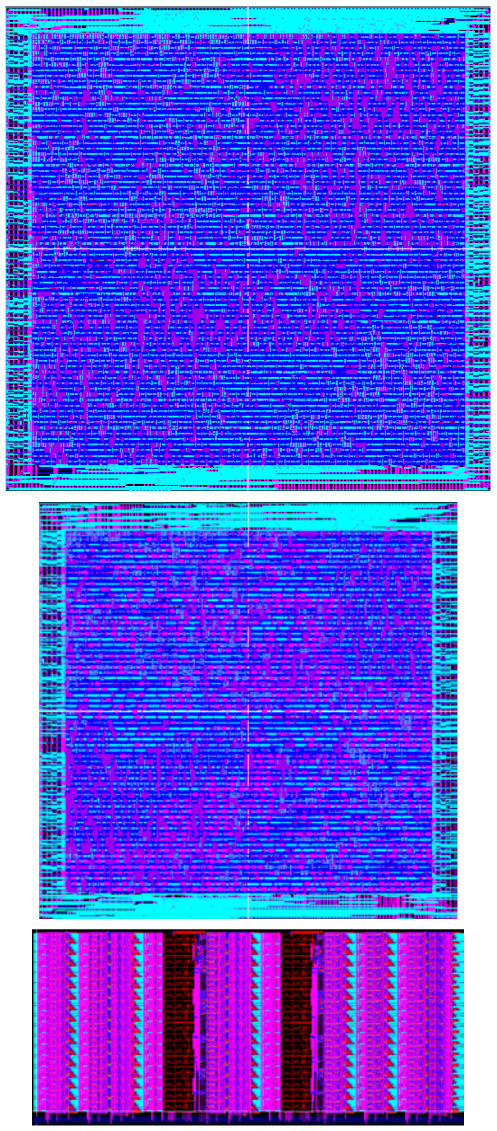
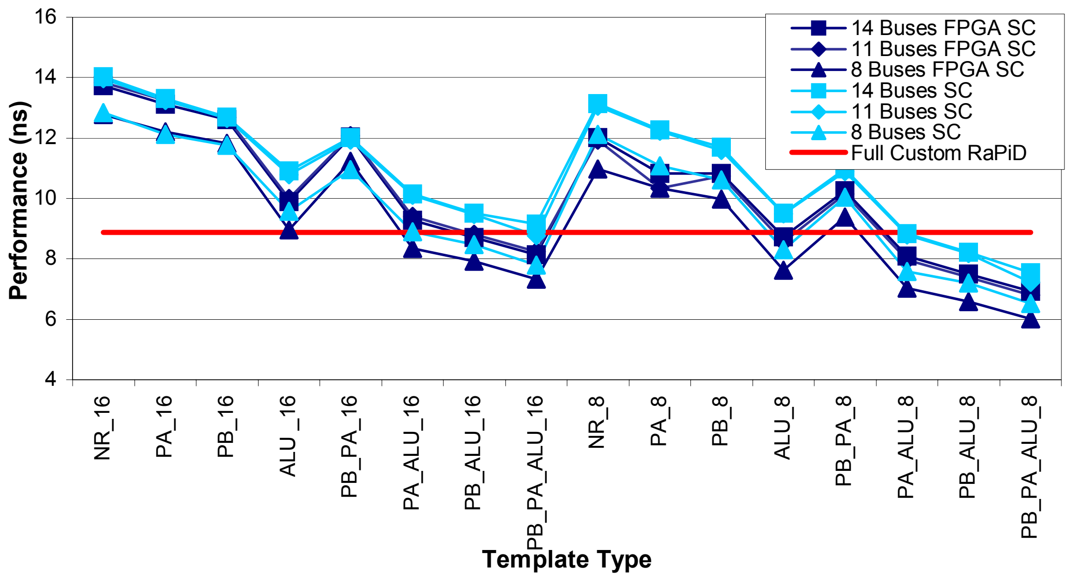


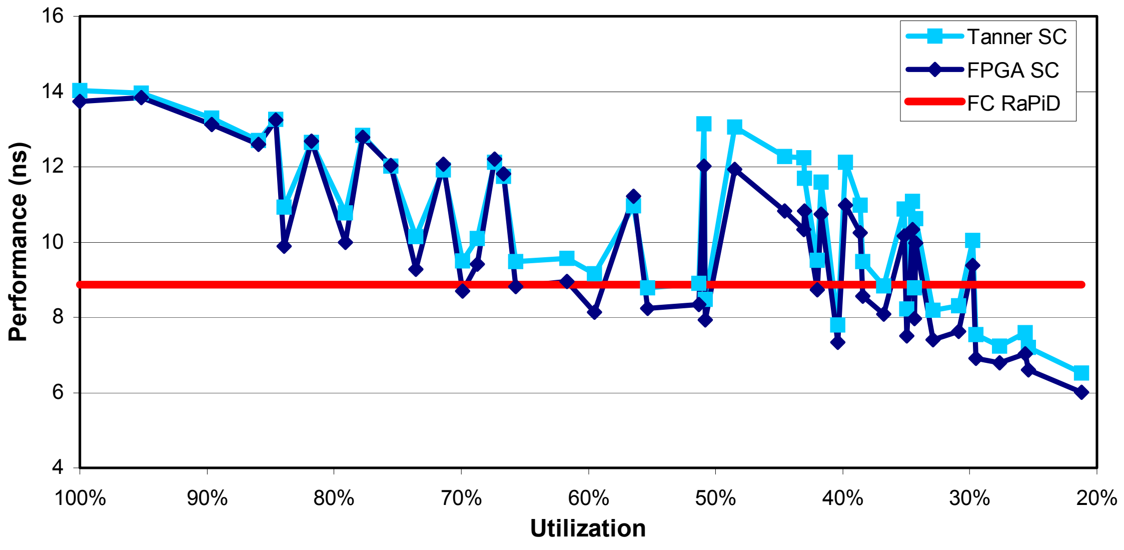
Figure 4. Tanner standard cell (top), FPGA standard cell (middle), and full-custom RaPiD (bottom). The relative size of the various layouts is preserved.

Table 2 summarizes the results of automating the design through the use of standard cells by providing an easy reference for choosing a particular optimization to reach specific design goal. For example, if a design called for a reduction in area, and you do not need all of the functionality that an adder can provide, than reducing ALUs to Adders would give you a 30% reduction in area and a 40% increase in performance.

In this work we have shown that automation of layout generation for domain specific FPGAs is possible. We have further shown that as a target application domain narrows, the savings gained



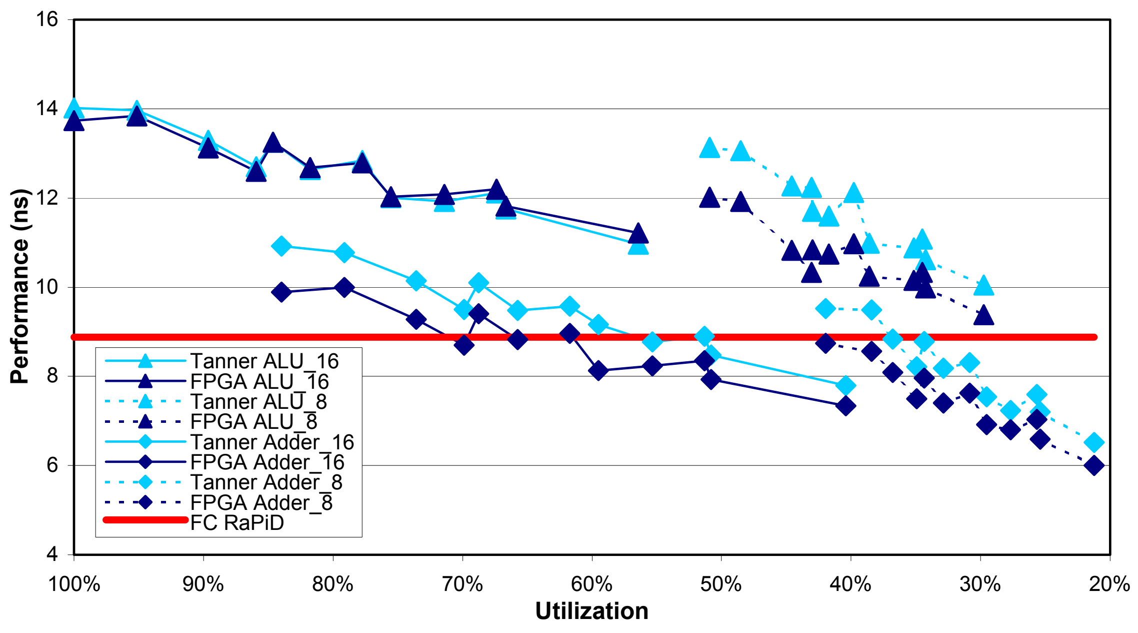
Graph 3. This graph shows the performance of full custom RaPiD as well as all of the different versions of the templates in ns. The y-axis is the performance of the templates in ns, while the x-axis is a list of each version of template, from most feature rich (left) to least (right).



Graph 4. This graph shows the performance of full custom RaPiD as well as all of the different versions of the templates in ns. The y-axis is the performance of the templates in ns, while the x-axis is the percent utilization of the original template, from most feature rich (left) to least (right).

from removing unused logic from a design enables a standard cell method of layout generation to approach that of a full custom layout in area and performance, and in some cases surpass them, with areas ranging from 270% larger to 46% smaller, and performance raging from 157% slower to 36% faster. Finally, by adding to a standard cell library a few key cells that are used extensively in FPGAs, improvements of 9% to 18.9% can be achieved.

The choice of what mechanism to use for implementing domain-specific reconfigurable subsystems is more than just a choice based upon area and performance. A full-custom tile, such as the RaPiD tile used here, provides a highly optimized, but completely inflexible, set of resources. Applications that need different resources, or greater amounts of a given type, may simply be unable to handle these demands. A standard cell methodology allows for any resource mix to be applied, with at



Graph 5. This graph shows the performance of full custom RaPiD as well as all of the different versions of the templates in ns. The templates are grouped depending upon whether they contain ALUs or adders, or whether they are 16-bit or 8-bit. The y-axis is the performance of the templates in ns, while the x-axis is the percent utilization of the templates, from most feature rich (left) to least (right).

most a 42% increase in area, and a 64% increase in performance.

Alternative approaches may help provide the middle-ground between the high quality but inflexible full custom tiles, and the completely inflexible but high overhead standard cell methodologies. In our future efforts we will investigate template reduction and circuit generator approaches. These four approaches combined should provide a spectrum of approaches, with each yielding benefits for some users. Overall, we hope to be able to close the gap between fixed tile-based FPGAs currently being developed in industry for SOC designs, and the traditional benefits of strictly ASIC designs.

Table 2. Average benefit gained by optimization type.

	Improvements	
Basic Optimization	Area	Performance
16 Bit to 8 Bit	2.0x	1.1x
ALU to Adder	1.3x	1.4x
Not Reduced to PA	1.1x	1.1x
Not Reduced to PB	1.2x	1.1x
Tanner SC to FPGA SC	1.2x	1.0x
SC to Full Custom	**1.3x**	**1.2x**

7. ACKNOWLEDGEMENTS

The authors would like to thank the RaPiD group, especially Carl Ebeling and Chris Fisher, for the RaPiD powertest full custom layout used in this research. We also are indebted to Larry McMurchie for support on the Cadence tool-suite. The idea for using standard cells to implement FPGA architectures was originally proposed by Herman Schmit. This work was supported in part by grants from the National Science Foundation and DARPA. Scott Hauck was supported by an NSF CAREER Award and a Sloan Research Fellowship.

8. REFERENCES

[1] Abnous, A. and Rabaey, J. M. "Ultra-low-power domain-specific multimedia processors," *Proc. of IEEE VLSI Signal Processing Workshop*, Oct. 1996.

[2] Cadence Design Systems, Inc., “Openbook”, version 4.1, release IC 4.4.5, 1999.

[3] Compton, K. and Hauck, S. “Totem: Custom Reconfigurable Array Generation”, *IEEE Symposium on FPGAs for Custom Computing Machines*, 2001.

[4] Ebeling, C., Cronquist, D. C. and Franklin, P. “RaPiD– Reconfigurable Pipelined Datapath”, *6th Annual Workshop on Field Programmable Logic and Applications*, 1996.

[5] Glökler, Tilman “System-on-a-Chip Case Study: ADSL-Receiver”, http://www.ert.rwth-aachen.de/Projekte/VLSI/soc.html.

[6] Goldstein, S., Schmit, H., Budiu, M., Cadambi, S., Moe, M and Taylor, R. “PipeRench: An Architecture and Compiler for Reconfigurable Computing”, *IEEE Computer*, Vol. 33, No. 4, pp 70-77, April 2000.

[7] McMurchie, L. E. and Ebeling, C. “PathFinder: A Negotiation-Based Performance-Driven Router for

FPGAs", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 111-117, 1995.

[8] Synopsys, Inc., "Epic Tools User Manual"

[9] Synopsys, Inc., "Synopsys Online Documentation", version 2000.05, 2000.

[10] Tanner Research, Inc., "Tanner CES Products", http://www.tanner.com/CES/products/files_now/dit_std_ce ll.htm.

[11] Weste, Neil H. E. and Eshraghian, Kamran *Principles of CMOS VLSI Design: A Systems Perspective*, Reading, Massachusetts: Addison-Wesely Publishing Company, 1993.