Macro-Driven Circuit Design Methodology for High-Performance Datapaths

Mahadevamurty Nemani and **Vivek Tiwari**

Intel Corporation, 3600 Juliette Lane, Santa Clara, CA 95052. Ph: 408-765-0589

Email contact: Vivek.Tiwari@intel.com

ABSTRACT

Datapath design is one of the most critical elements in the design of a high performance microprocessor. However datapath design is typically done manually, and is often custom style. This adversely impacts the overall productivity of the design team, as well as the quality of the design. In spite of this, very little automation has been available to the designers of high performance datapaths. In this paper we present a new "macro-driven" approach to the design of datapath circuits. Our approach, referred to as *SMART* (Smart Macro Design Advisor), is based on automatic generation of regular datapath components such as muxes, comparators, adders etc., which we refer to as *datapath macros*. The generated solution is based on designer provided constraints such as delay, load and slope, and is optimized for a designer provided cost metric such as power, area. Results on datapath circuits of a high-performance microprocessor show that this approach is very effective for both designer productivity as well as design quality.

1. INTRODUCTION

We believe that datapath design for high-performance applications (such as microprocessors) presents unique challenges. We also believe that this domain is currently under-served by commercial synthesis CAD. Besides the business realities regarding the relative size of the high-performance vs. general VLSI market, the reasons for this are to do with the basic nature and limitations of synthesis tools [10][11]. These tools are good at searching and optimizing within a defined search space. They cannot innovate, i.e. extend the search space on their own. However, continuous innovation is necessary in the high-performance microprocessor domain. Only part of the traditional 50% performance increase per generation comes through process technology evolution. The rest comes from ever more aggressive micro-architectures. These micro-architectures demand ever more from circuit designers – wider datapaths in shorter pipeline stages. New circuit families and structures are needed each generation. This can only be done by experienced and innovative designers, not CAD tools.

Ironically datapath design is also the area where some kind of help from tools can yield the highest returns. This is because datapath design is the most tedious, critical and time-consuming aspect of the design of high-performance microprocessors. Consider the following:

a) The datapath performance decides the overall performance of the processor. Most of the critical paths in the overall design go through the datapath.

b) Most of the chip power is dissipated in the datapath blocks (and their clocks) [8].

c) Large portions of the datapath have to be manually designed to meet the aggressive performance goals. Given aggressive time-to-market constraints, this implies large design teams.

Aside from the fundamental limitation of tools (described above), why is it that the designers of high-performance datapaths do not try to extract at least some benefit from the automation offered by traditional logic synthesis? We believe this is due to the following reasons:

a) Logic synthesis tools are limited to the gate level. Designers have to go down to the transistor level to extract the maximum performance.

b) Logic synthesis tools have been primarily restricted to static CMOS implementations. However, CPU designers heavily employ pass, dynamic logic (and continuously look for newer derivatives) in order to meet performance goals.

c) Logic synthesis tools produce flattened netlists by default, and do not respect the implicit hierarchies that a designer intuitively works with. Limiting a tool's scope through "black-boxes" significantly limits its optimization capabilities.

In light of all of the above, we believe that the best way to address this disconnect between the requirements of custom datapath design and the random-logic oriented synthesis tools is through *advisory* tools. These tools do not intend to replace the designer but to advise and assist him/her. In this paper, we present a specific realization of this paradigm. We call it SMART (Smart Macro Design Advisor).

The paper is organized as follows. The motivation for SMART is further discussed in Section 2. Section 3 provides an overview of the SMART flow, and Sections 4 and 5 discuss the components of SMART in greater detail. The results of some experiments on datapath circuits from a recent in-house high-performance microprocessor design are presented in Section 6.

2. MACRO-DRIVEN AUTOMATION

The SMART methodology is motivated by the following observations:

a) Datapath logic basically consists of a set of regular structures like multiplexors (muxes), shifters, adders, comparators, decoders, encoders, zero-detects, register files etc. We refer to these as datapath *macros*. A design methodology centered on an a-priori designed macro database that's available to the designer, could potentially offer significant productivity benefits.

b) Typical hand design of a datapath involves, for each instance of a macro, manual design space exploration followed by manual optimization to meet the performance, noise etc. requirements. Most of this latter work is in transistor sizing. An automatic sizer that can consistently size transistors on even the most performance-critical blocks is thus extremely valuable to a designer.

c) Tight schedule constraints limit design space exploration, thus, resulting in over-design. This implies wastage of silicon area and power. Automated exploration over a high-quality design database can thus lead to higher design quality.

In a real design, a macro may not always be realized in exactly the same way it exists in the database. A few structural changes to the schematic (e.g., merging in of a few gates of condition logic) may have to be performed to match RTL or to extract performance gains out of the design. A macro-based design environment should therefore support editing of macros in the design database. In addition, the designer should be allowed to control transistor sizes of portions of the macro while letting the automatic sizer size the rest of the macro. This is important; for instance, to allow the designer to improve the noise immunity of the circuit based on the local operating conditions.

The SMART macro methodology described in this paper has been developed with the aim of satisfying these requirements.

3. SMART METHODOLOGY OVERVIEW

The basic components of SMART of are summarized as follows:

(i) A large expandable database of the best available tried and tested topologies for the basic set of macros. Whenever a designer comes up with an implementation not available in the database, it can be incorporated into the database. This expandability is a key element of SMART's design database.

(ii) A specially designed transistor sizing engine. The motivation behind the design of the sizing engine is that it should be tunable for each macro, and extendable to different logic families. It is not aimed as a traditional general sizer [1-5] that gives reasonable results for all kinds of circuits, but may or may not meet the specified constraints all the time. Instead it aims to do the best it can for each specific macro, even at the cost of some additional customization.

(iii) A topology optimizer. The aim of this component is to automatically tune a topology for a specific macro instance starting from a general topology. (The topology optimizer component is currently under development and is not discussed further in this paper).

The basic design flow of SMART is as shown in Figure 1. Given a macro instance with its local constraints like delays, slopes and loads, SMART searches over the provided design space, and provides sized solutions. It can automatically pick the best solution based on a specified cost function (area, power) or let the designer make his/her own choice. At that point, the designer can further tune the design if needed. For instance, on a particularly noisy portion of the chip, the designer may like to manually tune certain transistor sizes.

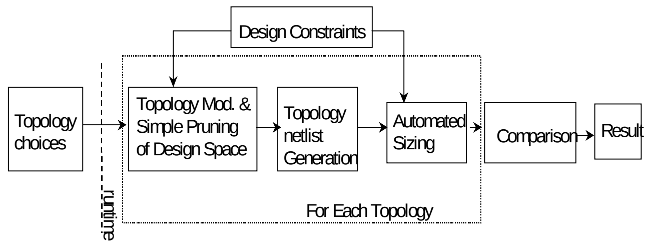

Figure 1: Design flow in SMART

While there is a lot of prior work on macro generators in the context of DSPs, we believe that there is no published work that directly addresses the problem domain targeted by SMART. Traditional optimization strategies are gate level and synthesis based. SMART on the other hand, is targeted at a custom design environment and is intended to support circuit families and topologies normally encountered in high performance microprocessors. Circuit innovations and logic optimizations are not automated but are captured in the design database, which is created by expert circuit designers based on project specific circuit design methodologies. The automated optimizations occur at the transistor level through an innovative sizing approach that can handle many different circuit styles, and hence generate full custom, high performance designs.

4. SMART DESIGN DATABASE

An important component of the SMART design advisory methodology is its design database. As mentioned earlier, this database contains many of the frequently used implementations of various macros. The schematics of these implementations are unsized. Instead, the transistors in the design are labeled with size variables. It must be noted that labeling directly impacts the quality of the final solution and the speed of the optimization procedure. While associating every transistor with a unique size variable may generate the solution with least transistor width, this may not be practical from a layout regularity perspective. Another important aspect is the hierarchy in the macro, which is important also important for layout. The schematics in the SMART database are designed keeping hierarchy in mind, which one may not be able to get through traditional logic synthesis tools. Regularity and hierarchy planning is where the expertise of a circuit designer plays an important role. This expertise is automatically incorporated into the schematics in the SMART database. In the remainder of this section we describe an example database of multiplexor macros.

Strongly mutexed N-first pass gate mux (Figure 2(a)) In this topology, we assume that the select signals are strongly mutexed (i.e., at all times one and only one signal is high. The others are low.) In the default labeling, the input drivers are labeled as P_1 (PMOS) and N_1 (NMOS), the pass-gates are labeled N_2 (assuming both PMOS and NMOS devices are of the same size, and the size of the inverter in the pass-gate is a fixed relation of N_2). The output driver is sized as P_3 (PMOS) and N_3 (NMOS).

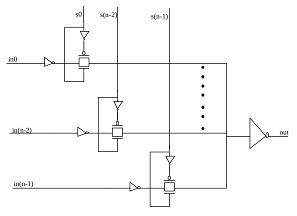

Figure 2(a): Strongly mutexed N-first pass gate mux

Weakly mutexed N-first pass gate mux (Figure 2(b)). Here, the input select signals are not strongly mutexed. The input select signals are combined to generate a select signal, which would make the select signal set strongly mutexed. This introduces additional delay from select to output. The labeling here is similar to that in (a), except that the NOR gate is labeled as P_4 (PMOS) and N_4 (NMOS).

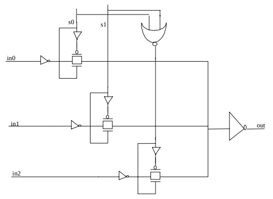

Figure 2(b): 3x1 weakly mutexed N-first pass gate mux

2-input pass gate mux with encoded select (Figure 2(c)). This topology is frequently used for 2 input muxes. In addition to an N-first pass gate, this topology has a P-first pass gate. An encoded select signal is used to avoid additional delay from select to output. The labeling here follows along the lines of (a).

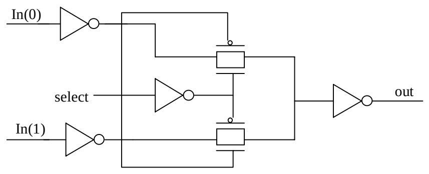

Figure 2(c): 2x1 Pass Gate Mux with Encoded Select

Tri-state muxes (Figure 2(d)). This topology is used when the load to be driven is very large or when the input signals travel over long inter-connects. Otherwise, this topology is typically slower than (a). In the default labeling, the tri-states are sized as P_1 (PMOS) and N_1 (NMOS) and the size of the inverter in the tri-state is a fixed relation of P_1 and N_1. The output driver is sized as P_2 (PMOS) and N_2 (NMOS).

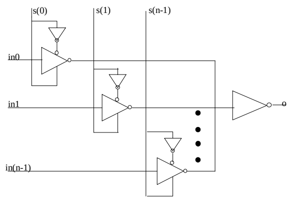

Figure 2(d): Tristate mux

Un-split domino mux (Figure 2(e)). In this topology all the product terms are connected to a single domino node. The output inverter is typically a high skew gate. The clock power is an important design metric in the selection of this topology. In the default labeling, the precharge (PMOS) device is labeled as P_1 (PMOS), evaluate device (clk-NMOS) is labeled N_2 and the remaining NMOS transistors (connected to data ports) are labeled N_1. The output driver is labeled as P_3 (PMOS) and N_3 (NMOS).

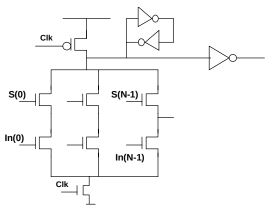

Figure 2(e): Nx1 Un-split domino mux

***(m, N – m)* Partitioned domino mux (Figure 2(f)).** This topology is typically better than (f), in terms of area and power when the size of the mux is large. A good choice of *m* is *m = floor(n/2).* This topology is used when high performance is required. If the two partitions are of the same size, they can be labeled identically, if not, we have to label them differently. Assuming the latter, we would label the PMOS precharge, NMOS data and NMOS evaluate devices of the top partition as P_1, N_1 and N_2 respectively, and the corresponding labels on the bottom partition are P_3, N_3 and N_4 respectively. The output driver is labeled as P_5 (PMOS) and N_5 (NMOS).

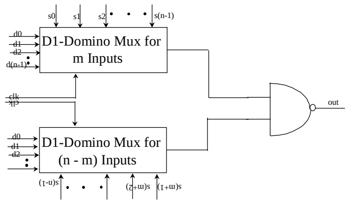

Figure 2(f): partitioned domino mux with size *m* and *(n – m)*

Figure 3 illustrates how SMART can be extended to new datapath macros. As can be seen from this figure, there is a customization cost associated with each macro. We believe that this is justified, in order to bring the productivity benefits of automation to the ever-increasing challenges of datapath design in high-performance microprocessors.

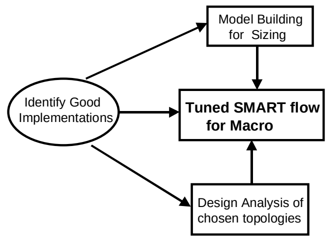
Model Building for Sizing
Identify Good Implementations
Tuned SMART flow for Macro
Design Analysis of chosen topologies

Figure 3: Flow development for a new macro

5. SMART SIZING METHODOLOGY

One of the key components in the SMART flow is the novel sizing engine used to optimize each topology. Since each topology is optimized through sizing, and then compared with other topologies using some metric, the choice of a "good implementation" depends significantly on the performance of the sizer. An important aspect to note about our sizing approach is that it is tuned to work well on datapath macros.

There is a lot of prior work on the general problem of transistor sizing. Continuous transistor sizing, in which the transistor sizes are allowed to vary continuously between a minimum and maximum size, has been tackled before in [1-4]. A related problem is that of discrete of library-sizing [5]. This is a combinatorial problem that is NP-complete. The continuous transistor sizing methodology used in SMART has significant differences from prior approaches. As mentioned earlier, the sizer has been developed with the aim of performing better on macros than a general sizer would. It also has to be extendable to different logic families. This has been achieved by keeping the optimization engine of the sizer independent of circuit topology and style, but customizing the other parts (modeling, path extraction, path compaction, and constraint generation) for different macro topologies and logic families. The macro-specific customization of the sizer is in accordance with the philosophy of incurring non-recurring customization costs for recurring benefits of better design quality (area, power), as is the up-front analysis of macro topologies (cf. Section 4).

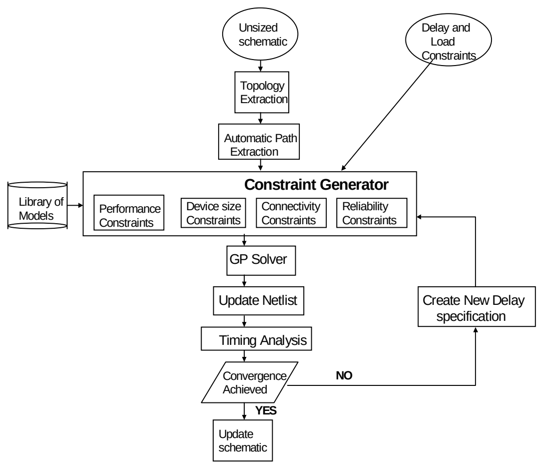

Figure 4: Overall flow of the SMART sizer

The overall flow of the sizer, implemented in our in-house design environment, is shown in Figure 4 and is fully automated. SMART takes in as input a description of the topology along with the design constraints like delays, slopes, external load etc. Using the database of models, it translates the extracted paths into constraints for timing, slopes (important for timing and reliability) and noise. These constraints are posynomial. SMART minimizes a cost function (power, area) under the above constraints. The quality of the optimization results depends on the quality of modeling. Thus, one would like the models to be as accurate as possible in relating the behavior of a gate to its device sizes. However, since we need to explore the design space as rapidly as possible, we need the sizer to be *fast*, in addition to being *effective*. This in turn places a restriction on the complexity of the modeling process. Keeping these issues of accuracy versus speed in mind, we have chosen to keep the component model *"posynomial"* (positive polynomial) [6]. This makes the optimization problem a Geometric Program [3]. The advantage of such programs is that they can be transformed into convex problems [6, 7] that can be solved efficiently and quickly, in a numerically stable fashion.

The optimal sizes for the netlist are generated by solving the above optimization problem using a geometric program solver. The delay of the generated netlist is then measured using a static timing analysis tool. If the actual timing of the generated netlist differs from the specification, "new" timing constraints based on the mismatch between the current delay specification and the current delay are generated. This is iterated until the original performance constraints are satisfied by the design.

Given the importance of the sizing methodology to the overall SMART flow, we discuss its key components in greater detail in the following subsections.

5.1 Sizer modeling

As mentioned in the previous section, the SMART sizer requires delay and slope models for basic circuit components. A necessary constraint on our models is that they be posynomial. A model for a component relates its timing and output slope behavior to its device sizes and input slopes. By components we could mean simple gates like inverters, NANDs, NORs, AOIs, etc., (implemented in static or domino style), pass-gates and tri-states, or, complex designs like domino muxes, pass-gate muxes etc.. This enables us to handle a wide variety of circuits. A typical equation template for rise delay is shown in (1) and (2).

$$t_{rise} = f(t_{int}, t_{in\_slope}, C_{ext}, W) \qquad (1)$$

$$t_{out\_slope} = g(t_{in\_slope}, C_{ext}, W) \qquad (2)$$

Here, W is the size of the pull-up transistors of the gate, C_{ext} is the external load, t_{int} is the intrinsic delay, t_{in_slope} is the input fall slope and *f(.)* and *g(.)* are posynomial functions. These timing models need not be exact, since they are only used within the inner optimization loop of the flow. The solution is always analyzed through a timing analysis tool. Better model accuracy leads to faster convergence.

5.2 Path extraction and complexity reduction

Among the several different types of constraints we generate for sizing a given circuit, the most important ones are the timing constraints. While there are several approaches to generating timing constraints, [9], the approach taken by us has been to specify these as constraints on the topological paths through the network. However, a combinational circuit can have an extremely large set of paths through it. Specifying constraints on all the paths would generate a very large problem size. We describe some of the techniques adopted by us to reduce the problem size.

In order to contain the number of paths generated, we make use of the regularity in datapath design. Using the regularity feature, we can prune out a large number of paths generated. Regularity allows us to constrain several nodes in the design to have the same size properties, and thus *identical* to each other. Thus paths containing *identical* nodes are reduced to one path during optimization. Efficiently exploiting this feature allows us to significantly reduce the path space.

In order to reduce the size of the problem, we model each node in the design with their worst case pin-to-pin delay models. However, for wide gates, there can be significant difference in the delays between the slow and fast input pins. To make sure that, as far as possible, a node is represented through its worst case input pin, we impose static precedence constraints on the input pins. This corresponds to partitioning the input pins into two sets, *fast* and *slow*, with pins in *fast* set representing the faster pins, while the pins in the *slow* set representing the slower ones. We use this information to prune the path space by eliminating the *fast* paths (when an equivalent *slow* path exists) from the path space. This allows us to capture a small set of meaningful paths for generation of constraints.

It is possible for two identical nodes (A and B) in a design to have different number of fanouts. We can impose a dominance relation on these nodes based on the amount of capacitance they drive, and in turn use this information to prune our path space. We heuristically decide the dominance based on the fanout of the nodes, as the capacitance information is an unknown during sizing. A more accurate approach is to compare the fanout space of two nodes when determining the dominance relationship.

Using the above approaches significantly reduces the size of the optimization problem. E.g., on a 64 bit dynamic adder, an exhaustive timing analysis revealed over 32,000 paths. However, the above techniques reduced the problem size to 120 paths, i.e., a factor of over 250 reduction in the problem size.

5.3 Constraint generation

High-performance designs tend to exploit different circuit styles. Hence it is possible for a given circuit to have a mixture of static, dynamic and pass logic in them. In static circuits, the flow of logic through a gate takes place only in one direction. Thus, each "static" path generates two timing constraints, namely, rise and fall at the output. However, in the case of pass logic we have one set of constraints through the data port, while we have another set of constraints through the control port. Every path through the data port corresponds to two timing constraints, as in a simple static path. However, for generating *control* to *out* constraints, we will have two delay constraints for a given path through the control port. These correspond to the rise and fall delays at the output. In both constraints the direction of signal transitions is the same up to the pass gate, i.e. what is needed to turn on the pass gate. However, downstream of the pass gate, we must consider the two the different signal directions. Thus, we will generate two paths and four constraints for every pass gate.

Dynamic circuits need separate constraints for precharge and evaluate paths. This is also tied to the clocking methodology and whether domino stages use clocked evaluate (*D1*) or not (*D2*).

As can be seen from the above discussion, several issues arise when we handle multiple circuit families and these must be carefully handled during flow customization for different macros. An interesting feature of SMART sizer for dynamic circuits is that the problem formulation automatically takes into account OTB (Opportunistic Time Borrowing) [12]. This allows its application on even some of the most critical circuits.

6. RESULTS

In this section, we discuss a few results obtained using SMART.

6.1 Sizing results

In order to study the effectiveness of our sizing methodology, we conducted several experiments on macros in a real microprocessor design. In these experiments, we extracted each macro from the design and measured its loading. The delay through it was measured using PathMill. We used the SMART sizer to produce a design with the same topology and performance. We re-ran PathMill to verify the performance of the SMART solution.

In Figures 5(a), 5(b), 5(c) we present the normalized total transistor width improvements we obtained on some of the macros (incrementors, zero-detects and decoders of bit-widths). Reduction in total transistor width directly results in area and power savings. It can be seen that large improvements in area and power can be obtained using the SMART sizing methodology. In all cases, the timing of the SMART solution was within a few pico-seconds of the original design.

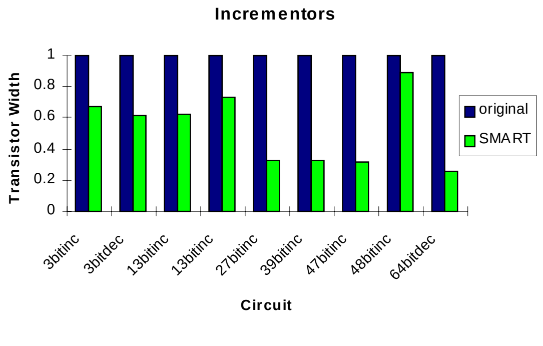

Figure 5(a): Results for incrementors

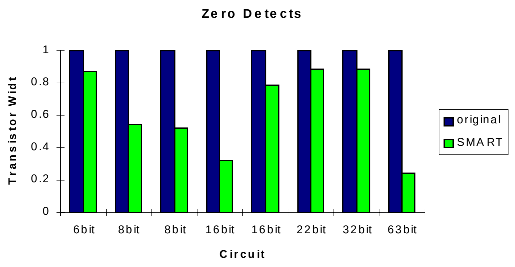

Figure 5(b): Results for zero detects

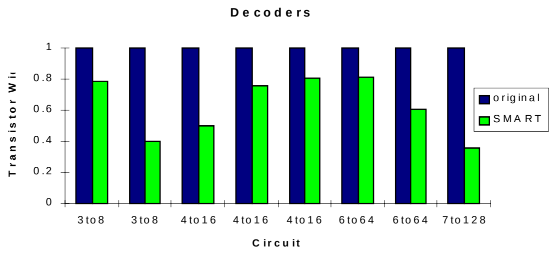

Figure 5(c): Results for decoders

In Table 1, the savings obtained for several muxes with different topologies (*cf.* Section 4) are presented. For each topology we considered multiple instances – the average savings are reported.

Topology	Xtor Width Savings	Clock Load Savings
Strongly Mutex Passgate	**15%**	n/a
2-Input Passgate Mux with encoded select	**25%**	n/a
Tri-state Mux	**16%**	n/a
Un-split Domino	**45%**	**39%**
Split Domino	**42%**	**28%**

Table 1: Results for different mux topologies

6.2 Results for adders

As an experiment with large and complex macros, we designed a 64 bit dual-rail carry-look-ahead adder and applied the SMART flow to it. The trade-off curve generated by SMART for this particular topology of the 64-bit adder is shown in Figure 6.

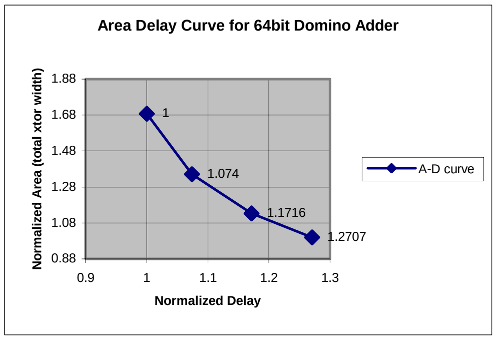

Figure 6: Results for a high-performance dynamic adder

6.3 Topology exploration example

We next present example results from a 32-bit high performance, 2-stage dynamic (D1-D2) comparator. We explored several alternate topologies to see if the chosen topology was in fact optimal. As shown in Figure 7, using SMART, we found that, the original topology performed better than the other alternatives. Moreover, the clock was reduced by 31%, without sacrificing performance. However, it must be remembered that under different design constraints, the original topology may not be the optimal one. With SMART, the exploration at a different design constraint is very easy, but to do this manually is an extremely tedious job.

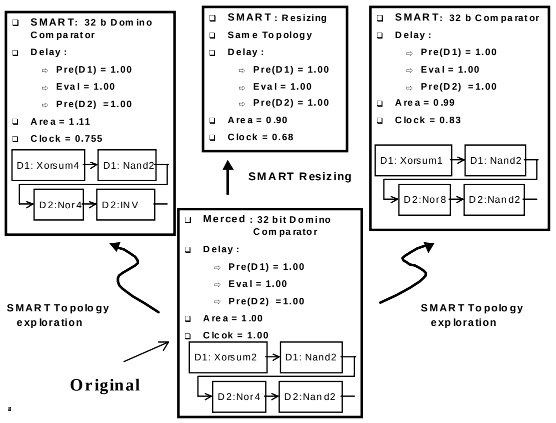

Figure 7: Topology exploration example

6.4 Results for complete design blocks

Finally, we present example results from using SMART for an entire functional block from the datapath of the same processor. This particular block has over 13,800 transistors in it, and datapath macros accounted for 22% of the total transistor width, and 36% of the total power. On applying SMART to the macros in the design, we achieved about 8% reduction in the total transistor width along with 8% power reduction on the overall design (measured using PowerMill). A timing analysis on the new design showed no performance penalty.

We recently used SMART as a part of the power reduction effort on one of the steppings of a high-performance microprocessor. We worked on four functional blocks: an instruction alignment block (Block1), bypass blocks from the instruction execution unit (Blocks 2 and 3), and a block from the instruction fetch unit (Block4). We have summarized the post-layout power savings obtained in Table 2. These savings were achieved with a short turn around time.

Functional Block	Power savings with SMART
Block1	**41%**
Block2	**22%**
Block3	**19%**
Block4	**7%**

Table 2: Results for power reduction on design blocks

7. CONCLUSIONS

In this paper we have presented our approach for extending the scope of CAD to datapath circuits in high performance microprocessors. This domain has been the toughest for traditional CAD to deal with. We prescribe advisory tools as the way to get the best of both worlds. And describe one such tool called SMART. SMART utilizes the vast design experience available in a company to generate the best possible design space of topologies for datapath macros. The proposed flow uses a novel quick sizing methodology (and an optional topology tuning stage) to generate various solutions that meet the designer's performance requirements. Using the design database of an actual microprocessor, we have presented results for several datapath macros namely, muxes, comparators, incrementors, decoders, adders etc., which demonstrate the utility of our approach.

ACKNOWLEDGEMENTS

We would like to thank the other members of the SMART team (Nithya Nagarajan, Franklin Baez, and Rakesh Patel) for all their efforts.

REFERENCES

[1] J. Fishburn and A. Dunlop, " TILOS: A posynomial programming approach to transistor sizing," in *Proc. ACM/IEEE ICCAD* , pp. 326-328, 1985.

[2] S. S. Sapatnekar, V. B. Rao and P. M. Vaidya and Sung-Mo Kang, "An exact solution to transistor sizing Problem for CMOS circuits using convex optimization," *IEEE Transactions on CAD*, vol. 12, pp. 1621-1634, Nov. 1993.

[3] J.-M. Shyu, A. Sangiovanni-Vincentelli, J. Fishburn and A. Dunlop, "Optimization-based transistor sizing," *IEEE Journal of Solid-State Circuits*, vol.23, Apr. 1988

[4] M. R. Berkelaar and J. A. Jess, "Gate-Sizing in MOS digital circuits with linear programming," *Proc. European Design Automation Conference*, pp. 217-221, 1990.

[5] W. Chuang, S. S. Sapatnekar and I. N. Hajj, "Timing and area optimization for standard-cell VLSI circuit design," *IEEE Transactions on CAD*, vol. 14, March 1995.

[6] J. G. Ecker, "Geometric programming: Methods, computations & applications," *SIAM Review*, vol.22, pp. 338-362, July 1980.

[7] K. O. Kortanek, X. Xu and Y. Ye, "An infeasible interior-point algorithm for solving primal and dual geometric programs," *Mathematical Programming*, vol.76, 1996.

[8] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel and F. Baez, "Reducing Power in High Performance Microprocessors", *Proc. DAC*, pp. 732-737, June 1998.

[9] A. R. Conn, I. M. Elfadel, W. W. Molzen, P. R. O'Brien, P. N. Strenski, C. Visweswariah and C. B. Whan, "Gradient-Based Optimization of Custom Circuits Using a Static-Timing Formulation", *Proc DAC*, June 1999.

[10] R. K. Brayton et. al. "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[11] G. DeMicheli, "Synthesis and Optimization of Digital Circuits", New Jersey: McGraw Hill, Inc., 1994.

[12] D. Harris et al., "Opportunistic time-borrowing domino logic", U.S. Patent No. 5,517,136, May 14, 1996.