Migration : A new technique to improve synthesized designs through incremental customization

Rajendran Panda, Abhijit Dharchoudhury, Tim Edwards, Joe Norton, and David Blaauw
Advanced Tools Group, Advanced System Technologies Lab., Motorola, Austin, TX 78730
email: panda@adttx.sps.mot.com

Abstract - A novel technique to explore the *performance vs design effort* trade-off is proposed. Starting from an optimally synthesized design, performance-critical cells are incrementally and optimally selected and custom-sized to generate this trade-off. Efficient algorithms for the optimal selection, and for improving the reuse of custom-sized cells in the design are given. Significant performance gains are shown in several real circuits through the addition of very few customized cells.

Keywords - migration, custom sizing, timing optimization

1. Introduction

Circuit designers today face two challenges of conflicting objectives - designing for optimum performance, and designing in a minimum time to reduce the 'time to market'. Though full-custom solutions are ideal for achieving optimum performance, they require the most time and manpower resources. On the other hand, automatically synthesized solutions are attractive in the required design effort, but often fall short of the performance goals. The common design practice of identifying *a priori* what portions of a design are to be custom-designed and what are to be synthesized can be very sub-optimal, in terms of both the achieved performance and the design resources spent. There is currently no systematic way of automatically, and optimally, exploring the trade-off between these two metrics. Addressing this need, we provide a technique to generate design solutions that lie on an optimal *performance vs design effort* trade-off, and let the designer select the solution that is appropriate for the situation.

Despite the advancements in automatic synthesis, full-custom solutions still show superior performance than their synthesized counterparts, as illustrated by a real design in Figure 1. The curve in the figure is the optimal *area vs performance* trade-off generated by a mature custom sizing tool using a TILOS[1]-like algorithm. The x, y co-ordinates are performance (in terms of slack) and transistor area respectively, with points toward the left indicating better performance.

Also shown in the figure to the right of the curve is a standard-cell solution synthesized for optimum performance, using a commercial synthesis tool and an industrial library of 163 cells, and 2 - 4 drive strengths per cell type. Note the large performance difference between the two design methods. This is clearly due to the extreme flexibility in custom-sizing the devices, since the circuit structure is the same for these solutions. This is despite the fairly large size of the library used in synthesis. We have observed such performance differences consistently across numerous designs, and with different libraries. The library cells which are optimized for a wide range of loads may be suitable to instantiate a major chunk of logic which is not critical. However, while instantiating some logic on critical paths, the finite selection of drive strengths in the library, their fixed transistor sizes, and the fixed P/N ratio for a given drive strength become severe bottlenecks from the performance point of view.

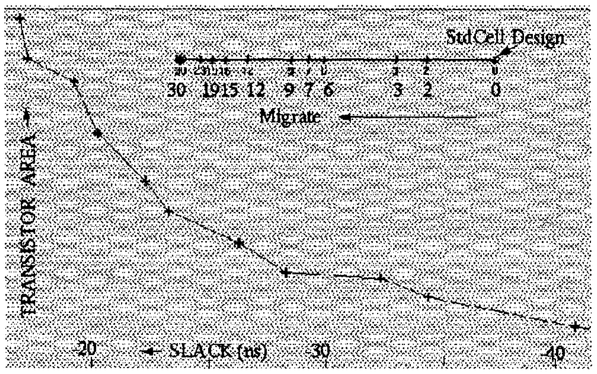

Figure 1. Performance gap between custom and standard cell designs.

The key idea in our proposal to optimize both the performance and the design effort is to stay close to the standard cell design paradigm, while also adding the flexibility of fine-tuning the device sizes of a selected few logic instances. This is very similar to what the designers have been doing, but with two important differences: (i) the custom-synthesis partition is now fine-grained, which is done at the cell (i.e. gate) level, and (ii) the partitioning decision is automatic and nearly optimal. The starting point is a synthesized design that has been optimized for performance and a given area. The performance is then improved in incremental steps, while maintaining the same area, by replacing a selected few critical cell instances with optimally sized custom cells. In Figure 1, this process is shown by the horizontal trade-off line which is migrating the design from the standard cell solution point to the left toward the full-custom trade-off curve. The number of custom cells needed at each step of migration is shown below the solution point. The main steps in the process are the optimal selection of the cell instances to be customized, and the optimal sizing of the selected instances. The latter step can be accomplished using sizing techniques proposed earlier, such as [1]. We address here the first part, viz. the optimal selection.

The remainder of the paper is organized as follows. In Section 2, we review the state-of-the-art in circuit optimization. Section 3 presents the proposed technique and the algorithms. Finally, Section 4 presents the results of several case studies and the conclusion.

2. Related Work

The topic of transistor and gate sizing for optimal performance/area/power has been very actively addressed. [1] proposed an optimization based on the posynomial relation between device sizes and circuit delays. [2] optimizes delay in a convex non-linear problem. Optimal gate sizing is addressed in [3] using linear programming techniques. [4] addresses the power-delay trade-off in gate sizing. [5] combines iterative and linear programming techniques for gate sizing. [6] proposes transistor sizing based on dynamic timing, and using non-linear solution techniques. A related problem of simultaneously optimizing the devices and wires has been studied in [7] and [8].

Earlier approaches assumed either custom or standard cell design paradigm and targeted at optimal *performance vs area/power* trade-off. In contrast, our work uniquely extends sizing techniques to explore the '*performance vs design effort*' trade-off.

3. Custom Cell Addition

We shall formulate the task as a constrained optimization problem, discuss its complexity, motivate our approach and then present effective heuristics. First, we introduce the relevant concepts and terminologies.

The overall objective is to improve the *delay graph* of a circuit, which is a DAG of timing arcs between nodes after all feedback arcs are broken. Every arc is marked with a value which is the propagation delay between its nodes, and there are distinct arcs for rising and falling transitions. A *path* is a sequence of edges in the delay graph. Associated with each arc is a *trigger* transistor, the device whose gate node is the source of the timing arc.

The sizing optimization is done iteratively on the transistor with the largest figure of merit, the merit being determined by a combination of *slack* and *sensitivity* metrics, given below :

Slack of a node is the difference between the required and the actual arrival times for the node, with large negative slack signifying large timing criticality.

Delay sensitivity, $\delta(m,a)$, of a transistor m w.r.t. an arc a is a measure of change in a's delay to a small change in the size of m[1], a large negative value indicating a favorable area/delay relation. Delay sensitivities are calculated directly from analytical timing relations.

The size of a device affects the delays of several arcs - all arcs through the device, and arcs from or to those nodes whose loading is affected by the size of that device. Therefore the merit of a transistor should consider its sensitivities w.r.t. every arc it can affect.

In our discussion, the term *cell* refers to a logic gate in the library, and *instance* refers to a logic block that can be mapped to a cell.

3.1. Formulation

We assume that the design effort (layout, extraction and characterization) needed to generate a new library cell is constant, and is independent of its logic functionality. Then the additional design effort to improve a given standard cell design can be measured by the number of new cells required. We can now formally pose the problem as a constrained optimization problem, as follows.

Given an initial standard-cell design, and a performance improvement target, find a minimum cardinality set of customized cells with which a subset of the cell instances of the design can be replaced to realize that performance improvement.

Let $\mathcal{G}$ be the delay graph under optimization and d be the delay of the longest (slowest) path in the circuit. To improve the performance of the circuit by a small amount, say, Δd, we need to improve a set P of the most critical paths whose delays are in the range $[d,\ d - \Delta d]$, such that their delays become equal to or less than $d - \Delta d$.

The problem is inherently complex for several reasons. (1) More than one arc in $\mathcal{G}$ has delay dependencies on the size of a transistor. So it becomes difficult to study the optimization of the identified critical paths in isolation. Changing the size of a transistor to improve a critical path in P can increase the delays in paths outside P. (2) The relative criticality of paths need to be considered in deciding which arcs of a path should be improved.

We consider a hypothetical, simplified, version of the above problem, purely to get a handle on the theory and to motivate our approach.

Consider a reduced delay graph $\mathcal{G}' \subset \mathcal{G}$ comprising only of timing arcs lying on the paths in P. Assume that (1) $\mathcal{G}'$ can be optimized without changing the delays in $\mathcal{G} - \mathcal{G}'$, (2) for every $a_i \in \mathcal{G}'$, there is a unique transistor m_i such that $\delta(m_i,\ a_j) = K$ for $i = j$ and $= 0$ otherwise in the region of improvement (Δd), and where K is a negative constant, and let M be the union of such transistors, corresponding to all the arcs in $\mathcal{G}'$, (3) for any $m \notin M$, and any arc $a \in \mathcal{G}'$, $\delta(m,\ a) = 0$ and (4) the transistors in M each belong to distinct cell instances in the design.

(1) above ignores the effect of sizing on the delays of arcs outside the selected critical paths. (2) and (3) together identify *one and only one* transistor for every arc on the critical paths that can be sized so that the arc's timing will

be improved. The equal delay sensitivity (K) eliminates the relative area merits of improving one arc over other. (4) establishes a simple cardinality relation between the instances sized and the transistors sized.

It is clear that, to improve $\mathcal{G}'$, every path in P must be covered (i.e. improved), by improving one or more edges in each path. Since improving an edge corresponds to sizing its corresponding transistor, this cover must be optimal to minimize the design effort. We refer to this problem as the *optimal selection problem.* We can easily show that the above simplified version of the problem is NP-complete, by reducing the *Hitting Set Problem*[9], known to be NP-complete, to this problem in polynomial time.

3.2. Optimal Selection

The foregoing formulation motivates an approach that improves the delays of a set of edges s.t. every path in P is covered (improved). This edge set is in fact the *hitting set* for P. The delay of each edge in the hitting set can be reduced by resizing the instance containing the trigger of that edge. Although resizing other instances may also improve an edge, the rationale for sizing the instance of the trigger is that the trigger and the transistors in its dc-component usually have larger sensitivities w.r.t. that edge than the transistors in other instances. Only in rare extreme cases when the trigger is already over-sized and is also driving no load or light load, this is not true.

To minimize the number of cells resized, the hitting set must be optimal. A good heuristic for the hitting set problem is the greedy covering of P using edges that lie on the most number of paths in P. We use such a greedy approach but with certain modifications so that the deviations from the above simplifying assumptions are properly accounted for. The simple greedy measure of an edge, which is the number of paths it covers, is modified by weights based on the slacks and sensitivities associated with the edges.

Without going into detail, we give below the merit function of a transistor that is used in determining the optimal set of instances to be resized. For an edge a, and a transistor m, let $S(a)$ be the slack of a, $\delta(m,a)$ be the delay-sensitivity of m w.r.t. a, $\gamma(m)$ be the input-capacitance-sensitivity of m, $R_{in}(m)$ be the drive resistance at the input (gate) of m, $S_{in}(m)$ be the slack at the input (gate) of m, and $r(m)$ be the total number of paths covered by the arcs in $\mathcal{G}'$ for which m is a trigger.

Then, the merit function $F(m)$ is taken as :

$$F(m) = r(m) \times w(m)$$
$$w(m) = \sum_{a \in \mathcal{G}'} S(a) \times \delta(m,a) + \gamma(m) \times R_{in}(m) \times S_{in}(m)$$

The $S(a)$ weights account for the relative criticality of the edges. Likewise, $\delta(m,a)$ considers relative gains of sizing, and $\gamma(m)$ considers the loading effects on the driver. The slacks are forced to be negative by suitably shifting their raw values so that the weighting function is not misinterpreted for an arc with positive slack and positive sensitivity.

3.3. Auto-grouping

Suppose multiple instances of a cell have identical drivers and loads, and the logic structure surrounding these instances are 'identical'. An example of such identical structures is a battery of inverters driving a bus, as shown in Figure 2. In such situations, if one instance is performance-critical, it is very likely that the other copies are also equally critical. Moreover, it is possible to resize all of them identically so that several signal paths in the design are improved with just one custom cell. This is the basic idea in improving reuse of customized cells. The designer can explicitly specify which are similar instances that can be grouped for sizing. When no explicit information is available, the auto-grouping algorithm detects such situations automatically.

As soon as a cell is selected for sizing, the auto-grouping mechanism looks at all other instances of this cell in the design, and verifies a set of conditions to detect if two instances are similar. The conditions verified are related to sizing. Specifically, we check to see if the instances have the same cell type, strength, loads and slacks at the output pins, and slopes at the input pins, within a specified tolerance. If so, the similar instances are grouped.

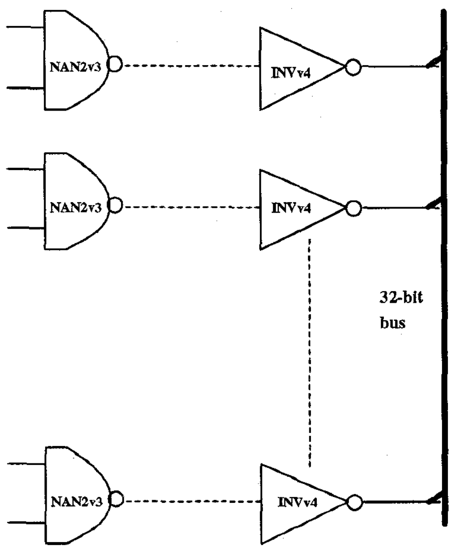

Figure 2. Multiple similar instantiation of cells

If the similar instances were not grouped, then the sizing algorithm would size them differently, though approximately equally, since the factors influencing the sizing algorithm, such as slacks, sensitivities, loads, etc. will change from one iteration to another, and are also dependent on the order in which the instances are sized. This will lead to more than one custom cell. The auto-grouping criteria are easily tested on sorted lists of instances. Thus the algorithm is very fast. □

The overall algorithm for migration is:

(1) Identify a set of top critical paths as the target set to improve. A minimum of 5 – 50 paths are targeted at each step of migration.
(2) Identify the triggers on the targeted critical paths.
(3) Merit the triggers using the cost function in Section 3.2.
(4) Greedily select the instances containing the triggers with the highest merit, until all the targeted paths are covered.
(5) For each selected instance, auto-group it with other identical instances elsewhere in the design.
(6) Resize the transistors of selected instances for optimal performance.

4. Results and Conclusion

The proposed technique was benchmarked using four real circuits drawn from different designs - two control blocks, one decoder logic, and one large microprocessor core. The initial synthesized circuits were obtained using a commercial synthesis tool, and running production quality scripts to optimize the delay under area constraints. Three different libraries were used in synthesizing the above four circuits. The proposed migration algorithm was then applied to these circuits, specifying a maximum of 20-50 cells in the design to be custom sized. The optimal selection and grouping of cells was done as described in Section 3, and the optimal sizing was done using a mature sizing optimization tool. For the migrated circuits, only the selected cells were allowed to be resized. The same sizing tool was used also to completely resize the circuits to ascertain the maximum performance gain achievable through sizing. The results are reported in Table 1. Due to the large size of the microprocessor core, a fully resized solution could not be obtained for this case.

In the table, column 2 shows the size of the circuit. Columns 3 and 4 show the size of the library used for synthesizing the initial circuit. Columns 5 – 7 show the delays respectively of the synthesized, fully resized, and migrated circuits. Column 10 shows the number of cells customized, and its percentage w.r.t. the total cell instances. Column 8 shows the percentage reduction in the delay of the migrated circuit w.r.t. the synthesized design. Column 9 compares the improvement obtained by migration to the best improvement possible by resizing the entire circuit. All delay comparisons are w.r.t. a fixed area, which is the area of the initial design.

Despite the high quality libraries used, the synthesized designs were 1.4 times to 2.1 times slower than the fully resized versions, indicating the sizing-related performance limitation of the standard cell design. But by adding very few (0.3% to 17%) custom cells, the migration obtained significant speed improvements (11.8% - 38.1%) over the synthesized design. The 11.8% delay reduction in the large microprocessor core is significant considering the small (0.3%) part of the design that required to be customized. This suggests that another application of migration can be in a design situation when timing problems creep up after placement and routing. In this situation, migration can supplement the in-place optimization efforts.

The effectiveness of the algorithm is evident in the fact that 45% to 71.4% of the performance gap between the full-custom and synthesis solutions could be reclaimed at the expense of as little as 0.3% - 17% custom design effort.

The above results indicate the usefulness of this approach in today's design situation, where minimizing design effort has become crucial to the success of a product. The technique adds another dimension to the trade-offs that a designer can exercise automatically.

REFERENCES

[1] Fishburn et. al. Tilos: A posynomial programming approach to transistor sizing. *ICCAD*, pp 326–328, 1985.

[2] Chuang, et. al. Timing and area optimization for standard-cell vlsi circuit design. *IEEE Trans. on CAD*, pp 308–320, 1995.

[3] Berkelaar et. al. Gate sizing in mos digital circuits with linear programming. *EDAC*, pp 217–221, 1990.

[4] Sapatnekar et. al. Power vs delay in gate sizing : Conflicting objectives? *ICCAD*, pp 463–466, 1995.

[5] Chen, et. al. An iterative gate sizing approach with accurate delay evaluation. *ICCAD*, pp 422–427, 1995.

[6] Conn, et. al. Optimization of custom mos circuits by transistor sizing. *ICCAD*, pp 174–180, 1996.

[7] Cong et. al. Simultaneous driver and wire sizing for performance and power optimization. *IEEE Trans. on VLSI Systems*, pp 408–425, 1996.

[8] Menezes, et. al. A sequential quadratic programming approach to concurrent gate and wire sizing. *ICCAD*, pp 144–151, 1995.

[9] Garey et. al. *Computers and Intractability*, 1979.

TABLE 1. Results of Migration

Circuit Name	No. of Trans.	Synthesized Design: Library Cells	Synthesized Design: Library Size Strengths	Synthesized Design: Delay ns	Custom Delay ns	Migrated Design: Delay ns	Migrated Design: Improvement % over synthesized	Migrated Design: Improvement % of best gain	Migrated Design: No. of New Cells
Control 1	1059	163	2-4	37.3	17.4	23.1	38.1%	71.4%	30(17%)
Control 2	6239	163	2-4	41.0	28.5	34.5	15.8%	52.0%	20(2%)
Decoder	2926	430	3-4	12.0	7.1	9.8	18.3%	45.0%	30(5.5%)
Core	80542	175	2-4	22.8	n/a	20.1	11.8%	n/a	51(0.3%)