

Delay Driven AIG Restructuring using Slack Budget Management

Andrew C. Ling
Department of Electrical and
Computer Engineering
University of Toronto
Toronto, Canada
aling@eecg.toronto.edu

Jianwen Zhu
Department of Electrical and
Computer Engineering
University of Toronto
Toronto, Canada
jzhu@eecg.toronto.edu

Stephen D. Brown
Altera Corporation Toronto
Technology Centre
Toronto, Canada
sbrown@altera.com

ABSTRACT

Timing optimizations during logic synthesis has become a necessary step to achieve timing closure in VLSI designs. This often involves “shortening” all paths found in the circuit at a cost of increasing the circuit area. In contrast, we present a synthesis approach which leverages slack budgeting to effectively minimize the critical path length without increasing the area of the design. Our results confirm that this is an effective method to control area while optimizing for delay. When compared to an area driven logic synthesis flow, we achieve a 32% reduction in logic depth and an 11% reduction in circuit delay when placed by VPR [1]; and when compared against a depth controlled logic synthesis flow without slack budgeting, we achieve an 8% reduction in logic depth and a 3% reduction in circuit delay when placed by VPR [1]. In both cases, the area penalty is negligible.

Categories and Subject Descriptors

B.6.3 [Hardware]: Logic Design - *Design Aids*

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

Delay optimizations in CAD are critical to allow designers achieve timing closure on their designs. There have been several pieces of work that have applied delay optimizations in all stages of the CAD flow: from high-level synthesis and logic synthesis [2] to placement [1]. Delay optimizations late in the CAD flow are attractive since delay metrics within the circuit can be accurately modeled. Hence, timing optimizations can correctly focus on critical areas in the circuit. However, optimizations late in the flow are fairly restrictive since netlist transformations at this stage must obey physical architectural constraints. Furthermore, the runtime penalty of optimizations late in the CAD flow is expensive since all restructuring techniques must be legalized to conform to the architectural requirements.

In contrast, technology independent optimizations early in the flow are attractive since architectural restrictions can be ignored. However, often simple delay models must be used since there is a large disconnect between the circuit representation and the physical implementation. Specifically, logic depth has been a primary minimization metric when dealing with delay driven optimization

at the technology independent logic synthesis level. Although circuit depth does not correlate well with the actual delay of a given path, the fidelity between circuit depth and circuit performance has shown that depth can be an effective metric to improve circuit performance.

When minimizing the overall depth of a circuit, area is usually sacrificed. This is due to the reduction in shared logic when optimizing for depth. For example, Fig. 1 shows two different decompositions of two functions. In Fig. 1(a), no logic is duplicated where function f is shared. The number of gates in this case is 5 and the circuit depth is 4. In contrast, Fig. 1(b) shows the same two functions, however, function f is no longer shared. In this implementation, the number of gates is 7 and the circuit depth is 3.

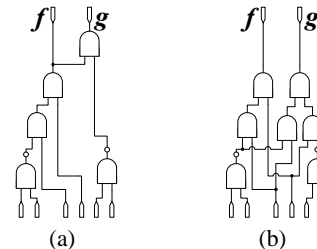


Figure 1: Area to depth tradeoff example. (a) Decomposition of f and g which minimizes area, gate cost = 5, depth = 4. (b) Decomposition of f and g which minimizes depth, gate cost = 7, depth = 3.

The increase in area shown in Fig. 1(b) must be controlled during depth minimization optimizations. If not, the resulting netlist will have significantly more area in the final implementation which increases its fabrication costs. Furthermore, a large area penalty will also lead to more congestion in the later stages of the flow which will degrade the timing performance. Previous work has only addressed depth minimization at the logic synthesis level in a localized manner [2]. When optimizing for depth, heuristics are used to minimize the depth on small clusters. When a depth bound is met, the optimizations focus on area. The limitation with this approach is that it is greedy in nature and depth optimizations can be too aggressive in localized regions. A possible solution to this is applying depth minimization at key points throughout a given path such that depth is reduced without a large area penalty.

In this work, we follow this idea and show how we can aggressively reduce depth while minimizing the area penalty. We do so by gathering global information to guide our depth optimizations. The depth minimization transformations we apply are similar to what is presented in [2]. However, unlike previous work [2], we drive our depth optimization with global information derived from slack budget management [3]. This allows us to trade area for depth at key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GLSVLSI'08, May 4–6, 2008, Orlando, Florida, USA.
Copyright 2008 ACM 978-1-59593-999-9/08/05 ...\$5.00.

locations throughout a given path without increasing the area significantly. We will show how this can complement optimizations found in the ABC [4] framework: a state-of-the-art synthesis engine that is an order of magnitude faster than SIS with competitive area. When combined with synthesis rewriting [4], budget driven depth optimizations can reduce logic depth while keeping the area penalty minimal resulting in a significant area to depth tradeoff. This is shown empirically where our technique reduces logic depth by 32% when compared to an area driven flow with a 2.7% area penalty. When compared to a depth controlled flow, our technique reduces logic depth by 8% with a 0.5% area penalty.

The rest of the paper is organized as follows: Section 2 gives some background information and related work; section 3 illustrates our slack budget management scheme and our budget driven delay optimizations; section 4 highlights our results; and section 5 concludes the paper.

2. BACKGROUND

For our purposes, each edge has a unit delay of 1 while nodes have no delay. We define the height of a node, h_i , as the longest distance to a PO and the depth of a node, d_i , is the longest distance to a PI. We will often use the terms depth and delay interchangeably. The depth of an AIG is the length of the longest path from a PI to PO. Given a depth constraint, D_{req} , the slack along a given path is defined as $D_{req} - d_{PI-PO}$, where d_{PI-PO} is the length of the path from PI to PO. The slack along an edge, e_{ij} , is denoted s_{ij} and is defined as $s_{ij} = h_i - h_j - d_{ij}$ where d_{ij} is the delay on the edge, and h_i and h_j are the heights assigned to node i and j respectively.

In this work, we adapt the budgeting formulation in [3] and apply it to timing driven logic synthesis. Given a logic depth constraint, we attempt to distribute the slack throughout a given path. Using this information, we are able to drive localized optimizations in a guided manner and as a result, minimize the overall depth of the circuit while keeping the area penalty minimal.

3. DEPTH DRIVEN OPTIMIZATIONS

The first step of our delay minimization algorithm is to isolate localized regions for resynthesis. This is done by applying a variant of the covering problem which attempts to minimize the overall cut size of the final covered netlist. The resulting netlist will be a set of covers where each cover encapsulates a localized region of the netlist. The clustering phase is illustrated in Fig. 2. For a detailed description of the cut based clustering algorithm, please refer to [5].

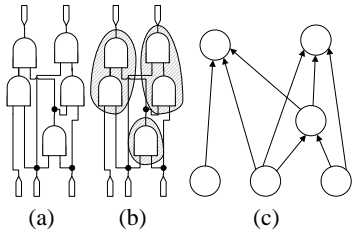


Figure 2: Clustering of a simple AIG. (a) Original netlist (b) Covering of original netlist. (c) Representing each cover and PI as a single node.

Following the clustering phase, slack must be distributed throughout the netlist such that each cluster has an associated slack along each fanin edge. When distributing slack, we want to maximize the number of clusters whose input edges have a large degree of slack.

This is because clusters whose input edges have a large degree of slack do not have to reduce its logic depth at the cost of increasing area. Thus, our goal is to maximize the sum of all edge slacks, while evenly distributing the slack to all clusters. To achieve this, we adopt the method presented in [3]. Here, the authors represent the budgeting problem as a linear programming formulation and transform it into network flow which is reshown here for completeness. Before we can represent our problem as a linear program, we must first describe our slack budgeting problem more formally.

PROBLEM 3.1. Slack Budgeting Problem. *Given a clustered AIG representing our logic circuit and a required delay D_{req} , allocate slack along each edge for all paths such that the overall slack allocated within the netlist is evenly distributed and maximized.*

Note that when trying to solve this problem, the summation of slack and delay along any path from a PI to a PO should not exceed D_{req} . To simplify this constraint, we can connect all POs to a single node SO , and connect all PIs to a single node SI . The edges used to connect to the SO and SI node will have a zero delay. This is illustrated in Fig. 3(a). Doing so will unify all PI and PO timing constraints into one timing constraint where the summation of the total slack and delay on any path between SI and SO must be less than or equal to D_{req} . Problem 3.1 and its constraints can be formulated as an integer linear program (ILP) shown in equation 1.

$$MAX \sum_{e_{ij} \in E} s_{ij} \quad (1)$$

$$h_i = h_j + d_{ij} + s_{ij} \quad (2)$$

$$h_{SI} \leq D_{req} \quad (3)$$

Constraint $h_i = h_j + d_{ij} + s_{ij}$ states that for a given edge e_{ij} , the height on node i must be equal to the height on node j plus the delay and slack on edge e_{ij} , while constraint $h_{SI} \leq D_{req}$ states that the longest path to SI must be less than or equal to D_{req} . To remove the constraint $h_{SI} \leq D_{req}$, we need to attach a back edge between the SO and SI node with a delay of $-D_{req}$. This is shown in Fig. 3(b) where a back edge is added between SO and SI . Finally, we can eliminate the slack variables s_{ij} by using the

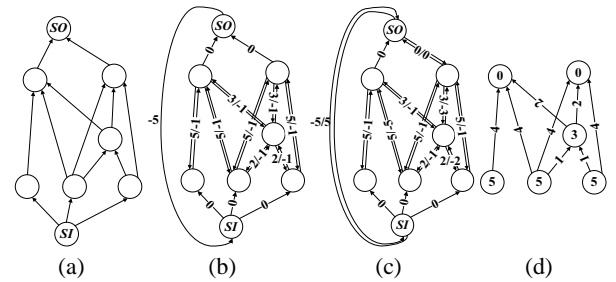


Figure 3: Conversion of clustered graph in Fig. 2 to a network flow problem for maximum slack budget allocation. (a) Addition of a "super" input and output node, SI and SO . (b) Creation of network flow graph. Removal of $h_{SI} \leq D_{req}$ timing constraint with the addition of a back edge between SO and SI . Also addition of lower and upper bound edge constraints. (c) Conversion of network flow graph into a residual graph. (d) Assignment of node heights and slack budgets on each edge.

relation $s_{ij} = h_i - h_j - d_{ij}$ as shown below.

$$\begin{aligned} \sum_{e_{ij} \in E} s_{ij} &= \sum_{e_{ij} \in E} h_i - h_j - d_{ij} \\ &= \sum_{e_{ij} \in E} h_i - h_j - \sum_{e_{ij} \in E} d_{ij} \\ &= \sum_{v_i \in V} h_i (\|fanout(v_i)\| - \|fanin(v_i)\|) - \sum_{e_{ij} \in E} d_{ij} \\ &= \sum_{v_i \in V} h_i (\rho_i) - \sum_{e_{ij} \in E} d_{ij} \end{aligned}$$

Here, we represent the value $\|fanout(v_i)\| - \|fanin(v_i)\|$ as ρ_i . This translates equation 1 to the following (note that the term $(-\sum_{e_{ij} \in E} d_{ij})$ can be dropped since it is a constant):

$$\begin{aligned} \text{MAX} \quad & \sum_{v_i \in V} h_i \rho_i \\ & h_j - h_i \leq -d_{ij} \end{aligned} \quad (4)$$

Equation 4 has a single constraint $h_j - h_i \leq -d_{ij}$ which represents a lower bound of total delay that can be allocated to an edge e_{ij} . However, we also must add an upper bound since there is a maximum amount of slack that a given cluster can leverage. For example, a cluster with 2 inputs cannot have a logic depth of more than 2 if decomposed into an AIG. Thus, to prevent any slack from being wasted, a bound is placed on each edge such that the delay and slack on an edge cannot exceed 2. This can be added with the following constraint:

$$h_i - h_j \leq u_{ij} \quad (5)$$

The dual of equation 4 with the additional constraint 5 can be derived using Lagrangian multipliers and is shown in the following equation:

$$\text{MIN} \quad \sum_{e_{ij} \in E} u_{ij} z_{ij} - d_{ij} y_{ij} \quad (6)$$

$$\sum_{e_{ki} \in E} (y_{ki} - z_{ki}) - \sum_{e_{ij} \in E} (y_{ij} - z_{ij}) = \rho_i \quad (7)$$

Note that equations 6 and 7 can be solved using network flow where u_{ij} and $-d_{ij}$ represent the cost of each edge in the network flow graph, and ρ_i represents the demand of each node. The construction of the network flow graph is illustrated in Fig. 3(b) where edges are labeled with their cost. For each edge, the first number lists the backward edge cost, while the last number lists the forward edge cost. Here, the delay constraint of each edge is modeled as a negative cost on the forward edge between i and j while the upper bound constraint is modeled as a positive cost on a backward edge between nodes i and j . Solving the network flow problem illustrated in Fig. 3(b) will create a residual graph. Using this residual graph, the slack budgets can be found as described in the following section.

Finding Slack Budget on Edges. Finding the slack budget on each edge is possible after the network flow problem is solved in the previous section. This is done by first creating a residual graph, then deriving the h_i values for each node. The residual graph is created by adding a reverse edge e_{ji} to every edge that has flow through it. This is shown in Fig. 3(c) (we have simplified the graph such that if two e_{ij} edges exist, the higher cost edge is omitted). The cost of the reverse edge, e_{ji} , is assigned the negative cost found on e_{ij} . Using the residual graph, h_i is calculated as the shortest distance from SO to every node i . Finding h_i for every node is equivalent to the single-source shortest path algorithm with SO as the source node. This can be solved using well known shortest-path

algorithms [6]. After the shortest-path algorithm is run, each node is assigned a height, h_i . Using the height values, the slack budgets can be derived using $s_{ij} = h_i - h_j - d_{ij}$. The assignment of node heights and slack budgets is shown in Fig. 3(d). A more detailed explanation of this procedure with proofs can be found in [3].

Assigning Upper and Lower Bound. Equation 4 and 5 illustrates a lower bound and upper bound on the slack budget assigned to each edge. In an AIG, since a path from a given input edge to the root node of the cluster must at least go through one AND gate, we set the lower bound delay of an edge as 1. To set the upper bound, we assume that the maximum possible depth of a logic function decomposition will be equivalent to the number of input variables to the function, N .

3.1 Depth Reduction and Area Recovery

Once we have a slack budget assigned to each incoming edge of a cluster, we can apply synthesis transformations such that delay is minimized. Although numerous Boolean depth minimization techniques can be applied, we choose a tree-height minimization technique since it can be directly applied to AIGs. Our transformation consists of two steps as follows: Depth minimization using AIG tree-height minimization; Area recovery using AIG rewriting [4].

The depth transformations we apply are similar to what is presented in [2]. This consists of two basic AIG restructuring techniques we will refer as tree-balancing and inverted-edge pushing. Tree-balancing consists of extracting a tree of AND gates and balancing the tree such that its depth is minimized. An example of this is shown in Fig. 4(a) and 4(b). Tree-balancing alone is limited since

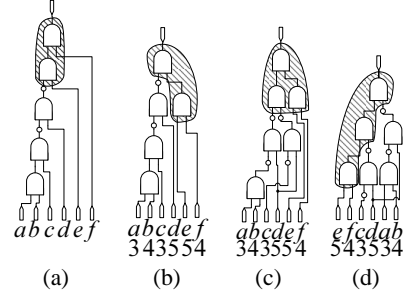


Figure 4: Depth minimization restructuring techniques for an AIG, highlighted cluster can be legally transformed using tree balancing. (a) Original netlist. (b) Tree-balancing to reduce cluster depth. (c) Inverted-edge push. (d) Tree-balancing to further reduce depth and meet slack budgets.

it depends on the existence of AND trees in the network. For example, in Fig. 4(b) it is impossible to reduce the depth of the AIG network any further due to the inverted edges which are represented as bubbled edges. The inverted edges, however, can be “pushed” further down the graph with a simple transformation shown in Fig. 5.

This effectively expands a given AND tree such that it has more

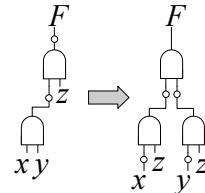


Figure 5: Example of inverted-edge push.

freedom for restructuring. This is shown in Fig. 4(c) where the

original AND tree is expanded due to the inverter push allowing for the depth of the cluster to be potentially reduced further.

As Fig. 5 illustrates, the transformation due to the inverter push has an overhead of an additional AND gate. Furthermore, the transformation will only lead to a performance gain if a better successive tree-balancing transformation is possible, such as the case shown in Fig. 4(d). Thus, it is necessary to predict where to apply the optimization such that it is worth the area penalty. Unlike the work done in [2], which generates several decompositions to choose from, we choose a single node to apply our inverted-edge push. The choice is driven by the slack budget information described in the previous section. For example, consider Fig. 4(b). In Fig. 4(b), each input edge has been annotated with the sum of the slack budget plus the lower bound delay ($s_{ij} + d_{ij}$) to form the total allocated delay for the edge. Notice that input a only has 3 units of delay assigned to it, but currently violates this. Tree-balancing alone cannot help in this case, nor can input permuting since input c also only has an allocated delay of 3 units. However, using our inverted-edge push technique, followed by tree-balancing, we can achieve our allocated delay for all inputs edges as shown in Fig. 4(d).

Another benefit of slack budgeting is that we can apply area recovery techniques on the portions within the cluster that have a large amount of budgeted slack. However, from our experience, running ABC’s synthesis rewriting to recover area works better than recovering area locally within a cluster. AIG rewriting is explained in [4] and is not discussed here.

4. RESULTS

We synthesize a large set of benchmark circuits using our depth driven flow and compare our results against four state-of-the-art synthesis flows. Following logic synthesis, our circuits are technology mapped to 4-LUTs using ABC [4]. The technology mapped circuits are then clustered into an FPGA architecture containing a cluster size of 10. These circuits are then place and routed with the VPR place and route tool [1].

We will refer to our depth driven flow as `depth_resyn2`. To see if our `depth_resyn2` flow can minimize logic depth without impacting area, we measure against four of ABC’s standard synthesis flows: an area driven flow (`compress2` [4]), a depth controlled flow (`resyn2`), a depth driven flow (`resyn2` with duplication), and a depth controlled flow with choice nodes (`choice` [7]). Details of these flows can be found in [8, 7, 4]. Our flow, `depth_resyn2`, is most similar to the `resyn2` flow, however, instead of tree-balancing, we apply our depth minimization flow which uses slack budgeting.

<i>flow</i>	4-LUTs	Gate	LUT	P&R
<code>compress2</code>	1.027	0.68	0.77	0.89
<code>resyn2</code>	1.005	0.92	0.94	0.97
<code>choice</code>	1.034	0.93	0.96	0.98
<code>resyn2 dup</code>	0.782	0.96	0.98	0.94

Table 1: Summarized comparison of delay and area for 20 IWLS circuits.

A summarized set of results are shown in Table 1. Here, we compare our slack budget approach against the four flows mentioned previously for 20 IWLS circuits and show the average results. Column *flow* lists the flow being compared; column 4-LUT lists the area comparison; column Gate lists the gate depth comparison; column LUT lists the LUT depth comparison; and column P&R lists the place and route delay comparison. Numbers less than 1 indicate that our approach is better. The trend shown in Table 1 shows that the most improvement in our approach is seen at the gate

level and gradually reduces after technology mapping and place and route. However, the resulting area to depth tradeoff in each case is very significant when we achieve a significant performance improvement against the area driven flow with a 32% reduction in gate depth with a small increase in area of 2.7%. A similar trend is seen when compared against the depth controlled flow `resyn2` where we can reduce the logic depth by 8% with an area increase of only 0.2%.

When compared against the depth driven flow `resyn2 dup`, we only achieve a 2% reduction in LUT depth. However, due to the large area overhead incurred in `resyn2 dup`, our place and route delay improvement is 6%. This shows the importance of slack budgeting to control area during depth driven optimizations in order to realize a place and route improvement in circuit performance.

Interestingly, rewriting has a difficult problem recovering area during the depth driven flow in ABC. We suspect that this is because of the large perturbation caused by the duplications created by the aggressive tree-balancing (`balance -s`) which makes it very difficult for synthesis rewriting to recover from. In contrast, our budget driven inverted-edge push occurs at key locations such that there is minimal disruption to the netlist. When compared against the choice flow, our relative improvement is minimal. This shows the power of choice networks during technology mapping to recover area and depth due to the larger solution space available to the mapper.

It should also be noted that the runtime was not an issue when running the network flow formulation using the Goldberg network optimization library [9]. On a 3.2 GHz machine with 2GB of RAM, none of the runs took more than a few seconds.

5. CONCLUSION

We have shown a novel application of the maximal budgeting formulation applied to slack allocation in the synthesis flow. Our results confirm that this is a scalable and practical means for depth minimization on AIGs where we can achieve an 8% reduction in logic depth with negligible area overhead.

6. REFERENCES

- [1] V. Betz and J. Rose, “VPR: A new packing, placement and routing tool for FPGA research,” in *FPL*, 1997, pp. 213–222.
- [2] J. Cortadella, “Timing-driven logic bi-decomposition,” *IEEE Journal on Technology in Computer Aided Design*, vol. 22, no. 6, pp. 675–685, June 2003.
- [3] S. Ghiasi, E. Bozorgzadeh, S. Choudhuri, and M. Sarrafzadeh, “A unified theory of timing budget management,” in *ICCAD*, Washington, DC, USA, 2004, pp. 653–659.
- [4] A. Mishchenko, S. Chatterjee, and R. Brayton, “DAG-aware AIG Rewriting: A fresh look at combinational logic synthesis,” in *DAC*, 2006, pp. 532–536. [Online]. Available: <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [5] A. Ling, J. Zhu, and S. Brown, “BddCut: Towards scalable symbolic cut enumeration,” in *ASP-DAC*, Jan. 2007, pp. 408–413.
- [6] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, “Shortest paths algorithms: Theory and experimental evaluation,” in *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1994.
- [7] A. Mishchenko, S. Chatterjee, and R. Brayton, “Improvements to technology mapping for LUT-based FPGAs,” in *FPGA*. ACM Press, 2006.
- [8] A. Mishchenko and R. Brayton, “Scalable logic synthesis using a simple circuit structure,” in *IWLS*, 2006, pp. 15–22.
- [9] A. V. Goldberg, “An efficient implementation of a scaling minimum-cost flow algorithm,” *J. Algorithms*, vol. 22, no. 1, pp. 1–29, 1997.