Performance Optimization Using Separator Sets

Yutaka Tamiya
FUJITSU LABORATORIES LTD.
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, JAPAN 211-8588
tamy@flab.fujitsu.co.jp

Abstract

In this paper, we propose a new method to optimize a performance of a very large circuit. We find the best set of local transformations to be applied to the circuit, by inserting "padding nodes" on non-critical edges of the circuit, and calculating separator sets of the circuit using separator sets. Our method is robust for very large circuits, because its memory usage and calculation time are linear and polynomial order with the size of the circuit.

According to our experimental results, our method has accomplished all circuits, while K. J. Singh's selection function method has aborted with three large circuits because of memory overflow. The results also shows our method has a comparable capability in delay optimization to Singh's method.

1 Introduction

In order to optimize a performance of a very large circuit, it is impossible to re-synthesize whole the circuit at a time. The best known strategy to date is repetition of local transformations: extracting a portion of a large circuit as a sub-circuit, which is critical in delay, re-synthesizing the sub-circuit, and restoring the sub-circuit to its original portion (Fig. 1). In order to achieve maximum improvement of a circuit performance, it is important to find a "good"set of local transformations. We call a set of local transformations for re-synthesis a "selection set". However, it is conjectured an NP hard problem to find the optimum selection set with the minimum area and/or power with satisfying all timing constraints[1].

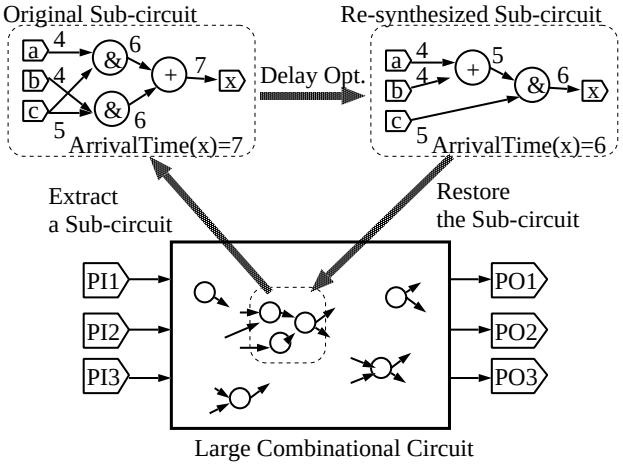

Figure 1: Local Transformation for Performance Optimization

Singh et al. have proposed a very efficient framework for performance optimization using iterative local transformations[1]. They use a selection function to represent a set of selection sets. A selection function is a characteristic function with a set of variables $\{x_i\}$, each corresponding to a local transformation i. The selection function holds true for a sub-set of $\{x_i\}$, whose corresponding local transformations improve all outputs of the circuit in delay.

They implement a selection function with a BDD[2]. One selection set corresponds to a one-path of the BDD (a path from the root node to the constant "one" node). The optimum selection set corresponds to the minimum weighted path among all one-paths, and can be calculated in linear time with the size of the BDD. However, BDD has a problem of memory explosion. In the worst case, both memory usage and calculation time grow exponentially with the size of a circuit. Thus the selection function is not robust for very large circuits.

Another method of finding a selection set uses separator set[3][4]. A separator set is a node set, which cuts all paths from inputs to outputs. Using a network flow algorithm the separator set with the minimum cost is calculated with time of polynomial order, and memory of linear order to the size of a circuit.

A separator set, however, does not always imply the optimum selection set. Only single-output tree-structured circuits hold it, but multi-output DAG circuits does not[1].

We show an example of delay optimization in Fig. 2(a). We want to find the optimum selection set, which reduces the slack at the output O_1 by 3. We first calculate three variables for each node i: $s(i)$ (slack at i), $D(i)$ (amount of node delay reduction by the local transformation at i), and $A(i)$ (cost of the local transformation at i. e.g.: increase of area and/or power).

In this example, only three nodes, T, W, and X, are re-synthesizable. The optimum selection set is $\{T, X\}$ with cost of 9. The separator set, however, is $\{W, X\}$ with cost of 11. Thus the separator set is not the optimum selection set.

The selection function at O_1 is $\mathcal{F}(O_1) = X \cdot (W + T)$. According to the on-set of the function, selection sets are $\{W, X\}$ and $\{T, X\}$. The latter has less cost and is chosen as the optimum selection set.

We propose a new method to find the optimum selection set using separator sets. By inserting "padding nodes" on non-critical edges of the circuit, the optimum selection set can be derived from separator sets even for multi-output DAG circuits. Our method is robust for very large circuits, because its memory usage and calculation time are linear and polynomial order with the size of the circuit.

2 Algorithm

We assume that a circuit is a combinational Boolean network, and $s(i)$'s, $D(i)$'s, and $A(i)$'s at all node i are calculated. If $D(i)$ is equal to or less than 0, $A(i)$ should be set to $+\infty$.

First, we transform a multi-output circuit to a single-output one, by adding a "virtual output". We replace all output nodes with buffers with a node delay of 0, and connect those buffers' outputs to inputs of the virtual output. As a result, the slack at the virtual

0-7803-5832-X /99/$10.00

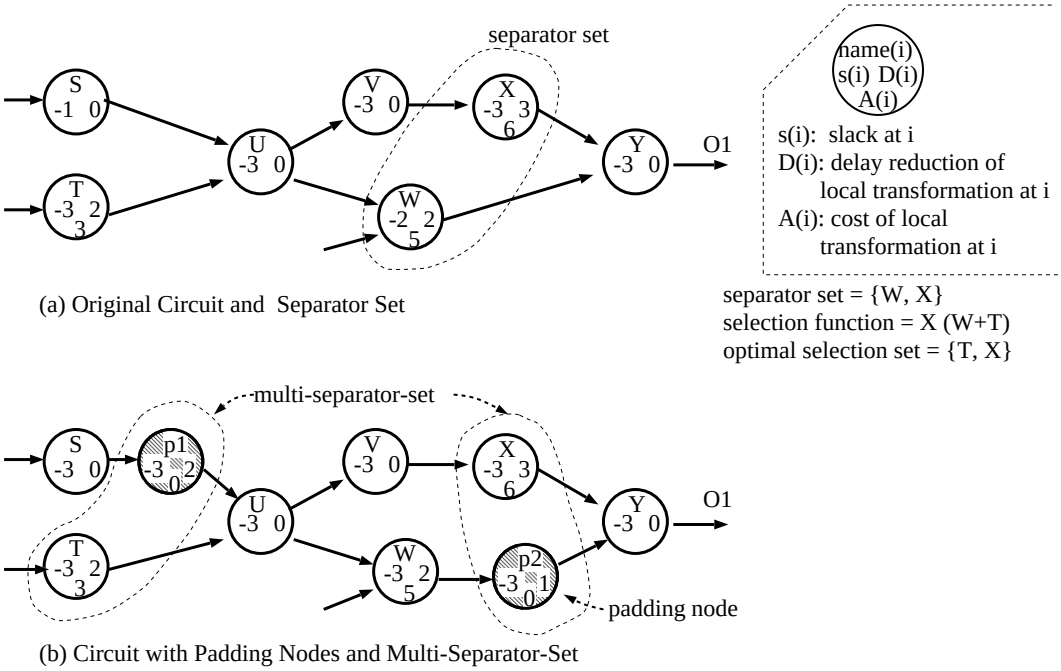

Figure 2: Finding a Selection Set

output is equal to the minimum slack of all original outputs.

For all edge e, we calculate $ds(e)$, a difference of node slacks at e's both ends:

$$ds(e) = s(tail\_node(e)) - s(head\_node(e)) \quad (1)$$

And if $ds(e) > 0$, we insert a "padding node" at e whose node delay is $ds(e)$. And delay reduction and cost of its local transformation are set $ds(e)$ and 0, respectively (i.e.: $D(e) = ds(e)$ and $A(e) = 0$). Note that padding nodes are inserted at all non-critical fanin edges of all nodes (including the virtual output).

From the viewpoint of delay analysis, slacks of all nodes become equal. Padding nodes eliminate all differences in arrival times at all inputs, required times at all outputs, and path delays among all reconvergent paths.

From the viewpoint of delay optimization, we treat padding nodes re-synthesizable, such that their node delays can be reduced to 0 with no cost. Thus, padding nodes are more likely chosen in a selection set than any ordinary node with positive cost. Note that if a padding node is chosen in a selection set, we do not need to apply any local transformations to the padding node.

A separator set of a circuit with padding nodes has a very important property as follows:

Lemma 1 (Delay Reduction by A Separator Set)
If s is any separator set of a circuit with padding nodes, and s has finite cost, then s has at least one ordinary node (i.e.: non-padding node), and it reduces the delay of the circuit by:

$$D(s) = \min_{i \in s} D(i) \quad (2)$$

□

The first half of the lemma is proven by the facts that a separator set must include at least one node on the most critical path, and padding nodes are not inserted on critical paths. The latter half of the lemma is trivial.

An example of padding nodes is shown in Fig. 2(b). There are two padding nodes $p1$ and $p2$, with delay reduction of 2 and 1, respectively. First, we find the separator set with the minimum cost: $\{T, p1\}$. But delay reduction of this separator set is 2, and it is less than the objective delay reduction (3). Then we find the separator set with the second minimum cost: $\{X, p2\}$. Combination of those separator sets results in delay reduction of 3 and cost of 9, which is the same result of the selection function.

As seen in this example, combination of more than one separator sets is efficient. In this situation, it is important how to select a set of separator sets at a time with the minimum total cost achieving the objective delay reduction. We call such a set of separator sets a "multi-separator-set".

Definition 1 (Multi-Separator-Set)
A "multi-separator-set" is a set of one or more than one separator sets of the same circuit. Those separator sets can share the same node in the circuit, if a sum of delay reduction of separator sets sharing the node is equal to or less than the delay reduction of the node.

We call a multi-separator-set "optimum", if total cost of nodes belonging to it is the minimum.
□

According to the following theorem, multi-separator-set and selection set are equivalent in terms of delay reduction and cost. Thus if we find the optimum multi-separator-set, we can get the optimum selection set by removing padding nodes from the multi-separator-set.

Theorem 2 (Selection Set and Multi-Separator Set)
Let η be a circuit, and η' be the circuit of η with padding nodes inserted. Any selection set t in η, which reduces the circuit delay by $D(t) > 0$, has its corresponding multi-separator-set m in η', such that:

- m consists of nodes in t and padding nodes.

- Delay reductions of t and m are the same: $D(t) = D(m)$.

□

Proof
At first, we prove t and padding nodes contain at least one separator set in η'.

Suppose that t and padding nodes do not contain any separator sets in η'. Then there exists a path p from one input to one output in η', all whose nodes are not padding nodes and included in t. This means p is the most critical path in η, and its path delay cannot be reduced by t. This is contradict to $D(t) > 0$.

Therefore, if $D(t) > 0$, we can find a separator set s, which consists of nodes in t and padding nodes. When $D(t) > D(s)$, we update the delay reduction $D(i)$ for each node i in s: $D(i) \Leftarrow D(i) - D(s)$. Note that this process also updates $D(t)$: $D(t) \Leftarrow D(t) - D(s)$.

Until $D(t) = 0$, we can iterate finding a separator set in η' and updating node delay reductions. This iteration should terminate since each iteration finds a different separator set and η' has finite separator sets.

It is trivial that all found separator sets contain only nodes in t and padding nodes, and their total circuit delay reduction is equal to the original $D(t)$.

□

According to Theorem 2, finding the optimum multi-separator-set has the same time complexity as finding the optimum selection set: i.e., NP hard. Thus, we approximate it by finding separator sets one by one like the proof of Theorem 2 (Fig. 3).

Step 1: Set a variable $m\_separator\_set$ empty, which represents a multi-separator-set.

Step 2: Make a flow network from the circuit with padding nodes. Note that each node n in the circuit has its corresponding edge in the flow network, whose flow capacitance is equal to the node cost ($A(n)$). We give each node (including a padding node) a variable of "rest of delay reduction". Its value is initially set to the delay reduction of the node, and decreases each time the node is chosen as a member of a separator set.

Step 3: Calculate the maximum flow on the flow network by a flow algorithm. If there exist no flows with finite cost, the procedure ends and returns $m\_separator\_set$.

Step 4: Find a separator set, which corresponds to the cut-set (set of edges) of the maximum flow, and merge the separator set to $m\_separator\_set$.

Step 5: For each node in the separator set obtained at Step 4, reduce its rest of delay reduction by the delay reduction of the separator set. If the rest of delay reduction becomes 0, we set the flow capacitance of the corresponding edge $+\infty$, so that we cannot any more choose the node as a member of a separator set.

Step 6: Return to Step 3.

Figure 3: Approximation of Multi-Separator-Set

Let us consider the time complexity of the multi-separator-set approximation. Finding the maximum flow takes $O(n^2 m)$ time by Dinic's algorithm[1], where the flow network has n nodes and m edges. If a multi-separator-set contains c separator sets, finding the multi-separator-set takes $O(c \cdot n^2 m)$ time.

But this is overestimate in practice, because Dinic's algorithm is iteration of finding an argumenting path on a flow network. In Fig. 3 we raise or keep flow capacitance for all edges at Step 5. Then, returning back to Step 3, we do not need to cancel the previously calculated maximum flow, but, get the new maximum flow by finding argumenting paths along the edges, whose capacitance we have raised. Thus, we conclude there is no big difference among times for calculating a single separator set and a multi-separator-set.

3 Experimental Results

We have compared delay optimizing capabilities of our method with Singh's separator set[3] and selection function[1] methods. Singh's methods are implemented as "speed_up -f " and "speed_up" commands of SIS-1.2, respectively. We have implemented our method on our logic synthesis platform "Magus", which is written in C++ and Tcl/Tk.

We have generated initial circuits, by area optimization and timing-driven 2-input gate decomposition on SIS (script.rugged and "speed_up -i" command).

For all three methods, we have used a unit-gate-delay model for delay estimation, a sum of literals for the optimization cost. A local transformation has been a collapse-and-decomposition: collapsing nodes in three levels and decomposing with two-cube-kernels in timing-driven mode[6][3].

Our experimental results are shown in Table 1. The platform is a Linux PC with Pentium II 400MHz, and 1GB memory. In the table, "$-\Delta$Delay" and "ΔLit" mean decrease in the circuit delay reduction and increase in the number of literals, respectively.

For all circuits, our method has successfully terminated with reasonable CPU time and delay reduction. But Singh's separator set method (speed_up -f) has resulted in zero or too small delay reductions, and the selection function method (speed_up) has aborted with three large circuits: s13207, s38417, and s38584. BDD's for the selection functions cannot be constructed even with 1GB memory. We have found speed_up (SIS-1.2) does not use a good BDD variable ordering for the selection function. Thus we have modified speed_up to use a depth first BDD variable ordering[7], and obtained new results of Table 2. However, speed_up still aborts with s13207. A dynamic variable ordering might accomplish the circuit, but it would take much more CPU time.

The maximum memory usage of our method has been 85MB for s38417. Since memory usage of our method grows linearly, our method can handle more than ten times larger circuits than speed_up with the same memory. This shows robustness of our method for finding a selection set.

For almost all circuits, our delay reduction is the same or a little bit better than speed_up. We conclude capability of delay reduction of our method is comparable to speed_up.

But ours tends to take more CPU time than speed_up. We think two reasons. One reason concerns with an implement issue: our method takes many iteration loops, so that it can obtain maximal delay reduction. Another reason is our poor resolution procedure for a conflicting selection set. The local transformation "collapse-and-decomposition" may cause a conflicting selection set: i.e, two or more than two local transformations in the same selection set can not be applied simultaneously because their collapsed node sets are overlapped to each other.

Algorithm finding a selection set (both of ours and speed_up) assumes all local transformations can be independently applied. Thus

[1] $n = 2 \cdot N + 2$, $m = E + N$, if the circuit has N nodes and E pin-to-pin nets

Circuit	Initial		Ours (Magus)			Separator Set (speed_up -f [3])			Selection Function (speed_up[1])		
	Delay	Lit	$-\Delta$Delay	ΔLit	CPU†	$-\Delta$Delay	ΔLit	CPU†	$-\Delta$Delay	ΔLit	CPU†
C432	27	339	6	87	25	0	0	0.1	5	107	6251
C499	19	820	4	192	58	4	352	1.6	4	144	23
C880	35	684	17	317	138	6	64	0.6	15	219	133
C1355	19	820	4	192	58	4	352	1.6	4	144	23
C1908	30	812	8	234	156	3	168	0.7	7	143	26
C2670	20	1379	5	66	60	2	30	0.9	4	24	63
C3540	40	2222	9	68	200	3	68	1.4	9	53	434
C5315	32	2911	11	337	321	5	80	3.0	9	148	55
C6288	92	4962	30	1064	1075	0	0	1.7	21	169	364
C7552	35	3976	16	999	953	0	0	1.2	13	427	129
s13207*	31	5045	7	-75	228	3	42	4.8	Memory Overflow		>1812
s15850*	38	6648	13	-188	390	3	95	6.5	13	126	8367
s35932*	11	16302	3	2460	776	0	0	24.7	2	2848	409
s38417*	26	23828	6	-2067	1306	0	0	10.9	Memory Overflow		>858
s38584*	29	23178	7	-1348	1984	2	52	49.5	Memory Overflow		>15937

* : Combinational logic extracted from a sequential circuit.

† : CPU time in seconds on Pentium-II 400MHz with 1GB memory.

Table 1: Results of Delay Optimization

a resolution procedure are needed for a conflicting selection set. Our method employs a greedy heuristic: choosing the first local transformation and canceling all the latter conflicting local transformations.

On the other hand, speed_up has a smarter resolution procedure: when it finds a conflicting selection set, it makes a new selection function by AND-ing the original selection function and negation of the conflicting selection set (a function of a selection set is an AND of all positive literals each corresponding to a local transformation in the selection set). At next time to find a selection set, optimum non-conflicting selection set should be obtained since previously found conflicting selection sets are disabled in the selection function. But, this resolution procedure makes the BDD size of the selection function much larger, and more likely results in memory overflow.

Circuit	speed_up with BDD Var. Ordering		
	$-\Delta$Delay	ΔLit	CPU
s13207*	Memory Overflow		1684
s38417*	4	-2081	495
s38584*	8	292	29740

Table 2: Results of Selection Function Method with Depth First BDD Variable Ordering

4 Conclusions

In this paper we propose a new method of finding the optimum selection set for re-synthesizing a large circuit. Our idea of a "padding node" is simple, but a combination with a network flow algorithm is very effective and robust. Comparing with Singh's methods, our method has reasonable capability in delay optimizing, and is much more robust.

In our future work, we will investigate a smarter resolution procedure for conflicting selection sets. We also will evaluate our method with technology dependent circuits.

References

[1] K. J. Singh, "Performance Optimization of Digital Circuits", PhD thesis, Univ. of California, Berkeley, 1992.

[2] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation", IEEE Trans. on Computers, C-35, August 1986.

[3] K. J. Singh, R. K. Brayton, A. Sangiovanni-Vincentelli, "Timing Optimization of Combinational Logic", Proc. of ICCAD-88, pp. 282–285, 1988.

[4] H. Savoj, K. Xiang, K. Pan, and A. Domic, "Technology Dependent Timing Optimization", Workshop Note of IWLS '97, Session 2–10, 1997.

[5] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in a network with power estimation", Soviet Mathematics Doklady, 11, pp. 1277–1280, 1970.

[6] J. Vasudevamurthy, J. Rajski, "A Method for Concurrent Decomposition and Factorization of Boolean Expressions", Proc. of ICCAD-90, pp. 510–513, 1990.

[7] M. Fujita, H. Fujisawa, N. Kawato, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams", Proc. of ICCAD-88, pp. 2–5, 1988.