Timing Optimization of Logic Network Using Gate Duplication

Chun-hong Chen[#] and Chi-ying Tsui*

[#] Department of Information Engineering,
Zhejiang University of Technology,
HangZhou, Zhejiang
People Republic of China

*Department of EEE,
The Hong Kong University of Science & Technology
Clear Water Bay, Kowloon,
Hong Kong

Abstract

We present a timing optimization algorithm based on the concept of gate duplication on the technology-decomposed network. We first examine the relationship between gate duplication and delay reduction, and then introduce the notion of duplication gain for selecting the good candidate gates to be duplicated. The objective is to obtain the maximum delay reduction with the minimum duplications. The performance of the algorithm is demonstrated with experiments on benchmark circuits. Our approach can also be combined with other technology-independent timing optimizers (such as *speed-up*) to achieve further delay improvement.

1 Introduction

Timing optimization has been an important goal in logic synthesis. In the technology-independent phase, the internal structure of a Boolean network is *restructured* to obtain a logically equivalent network with the reduced maximum logic level or reduced longest path delay under a given delay model. In the technology-dependent phase, technology mapping for minimum delay improves the circuit timing by selecting gates with appropriate size and driving capability from the library, and building fanout-trees for the gates with heavy loads to reduce the delay. After technology mapping, timing driven layout and/or transistor sizing can be used to further reduce the delay.

In the technology-independent phase, the exact delay information is unknown, but experience shows that the structure of the input Boolean network constrains, to a large extent, the technology-dependent phase in terms of the circuit performance. The simplified delay models used in the technology-independent phase can also give reasonably good correlations with the actual delays after technology mapping. With these in mind, many approaches for technology-independent timing optimization have been proposed [1-4]. They are typically based on a variety of transformations to speed up the network, with the goal of obtaining the maximum delay improvement with a minimum area penalty. The level reduction [2,3], and delay-driven clustering and collapsing [4] are examples of such transformations (more detailed survey of the timing optimization techniques can be found in [5]). However, most existing transformations do not consider the effect of the large fanout nodes of the transformed Boolean network on the timing delay of the final mapped circuit. For the technology-decomposed network which is the immediate input to the technology mapping process, if there are many gates on the critical paths having large number of fanouts, the mapper becomes less effective in improving circuit performance since potentially the large fanout nodes will remain in the mapped circuit. Large fanout is generally associated with heavy loading and long delay. Therefore, it is desirable to reduce the large fanouts right before technology mapping.

In this paper we look at how to reduce the fanouts of the critical nodes in the technology-decomposed network of an optimized Boolean function. We propose to use *gate duplication* to reduce the number of fanouts driven by the critical nodes. For a critical node driving large fanouts, some of the fanouts lie on the critical path while the rest do not. By duplicating the node to reduce the fanout load of the node that drives the critical path, the delay of the critical path can be reduced. Here, given a network of 2-input gates and inverters, we develop an effective algorithm for choosing the *good* candidate nodes, which have large fanouts to duplicate, based on the analysis of the slacks at all, gates in the network. The algorithm uses the *unit-fanout delay model*, and trades area for speed by modifying the topological structure of the network step by step. Our results show that the proposed technique can improve the circuit performance and can be combined with other timing optimizers such as *speed-up* [3] to further reduce the delay.

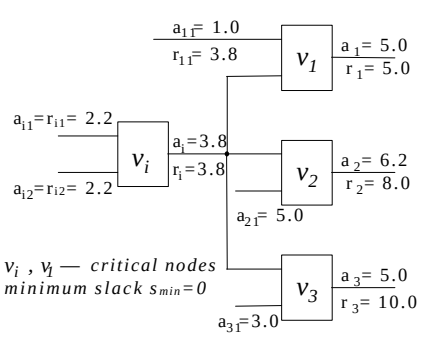
In the next section we first introduce some definitions. Then, we describe our gate duplication algorithm in Section 3, and report the experimental results in Section 4. Finally, in Section 5, we give the conclusion together with the possible future work.

2 Definitions

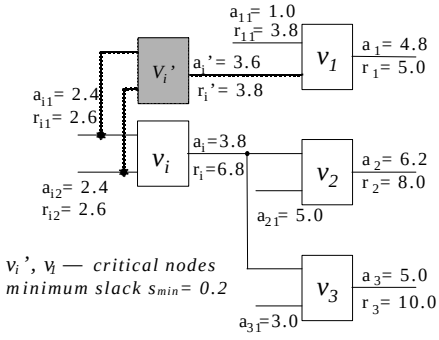
A combinational circuit can be represented as a Boolean network where each *node*, v, denotes a gate (or a primary input/output) in the circuit, and each directed *arc* from v_i to v_j denotes a connection from the output of v_i to an input of v_j. In this case, v_i is called a *fanin* of v_j , and v_j a *fanout* of v_i . The fanin set and fanout set of v_i are denoted by FI_i and FO_i , respectively. If there is a path from v_i to v_j , v_i is called a *transitive fanin* of v_j , and v_j a *transitive fanout* of v_i . The gates in the network are assumed to be single output gates. In the following, gate and node are used interchangeably.

For a network represented in terms of 2-input gates, it is reasonable to assign unit delay to each gate in the technology-independent stage. To account for the effect of fanout loads on the signal delay, each gate can be associated with an additional delay which, under the

unit-fanout delay model [3], is δ units for each fanout. The *node-delay* for (internal) node v_i is defined to be $d_i = 1+\delta \cdot f_i$, where f_i is the number of its fanouts. Given the *required time* at each primary output and the *arrival time* at each primary input in the network, the arrival time a_i and the required time r_i of node v_i are given by



(a) before duplication



(b) after duplication

Figure 1. The Example of Gate Duplication

$$\left.\begin{aligned} a_i &= \max_{v_j \in FI_i}(a_j + d_i) \\ r_i &= \min_{v_j \in FO_i}(r_j - d_j) \end{aligned}\right\} \qquad (1)$$

respectively. This recursive computation is called a *delay trace*. The *slack* at node v_i , denoted by s_i , is the difference between r_i and a_i , i.e., $s_i = r_i - a_i$. If the slack at a node is non-positive, the node is said to be on a *critical path* and, hence, called a *critical node*. Similarly, the minimum slack, s_{min} , corresponds to the *most critical path* and the *most critical node* (the *false path problem* [6] is ignored here). An ε-*network* is a sub-network in which all the nodes have a slack no more than $s_{min}+\varepsilon$, where $\varepsilon \geq 0$. The ε-network is the circuit region to be restructured for timing optimization.

3 Gate Duplication for Timing Optimization

3.1 Gate duplication and delay reduction

We first show how gate duplication can reduce the delay. The rationale is to decrease the arrival times at some critical nodes and/or increase the required times at the others, both of which can lead to the increase of the slack time at the critical nodes and hence reduce the critical path delay. The penalty is the possible circuit area increase. Figure 1 shows an example. Assuming that the arrival times at the input and the required times at the outputs are available, we can derive a_is and r_is at other nodes using equation (1) under the unit-fanout delay model (assuming δ = 0.2), as shown in Figure 1(a). v_i and v_1 are the critical nodes because their slack times are the minimum of zero. If a new gate v_i' is duplicated from gate v_i , as shown in Figure 1(b), the arrival time of v_1 is reduced, and the required time of v_i is increased. As a result, the minimum slack time is increased by 0.2, which reduces the critical path delay.

3.2 The gate duplication algorithm

The outline of our *Gate Duplication Algorithm* is shown in Figure 2. Input to the algorithm is a technology-decomposed Boolean network. The output of the algorithm is an optimized decomposed network. The main idea of the algorithm is to obtain the maximum delay reduction by duplicating as few gates as possible.

```
gate_duplication (network, ε)
  do {
      delay_trace( );
      generate an ε-network;
      smin = current_minimum_slack;
      critical_node_set = all nodes in ε-network;
      foreach node ∈ critical_node_set
          compute_duplication_gain (node);
      ordered_node_set = sort (critical_node_set);
      foreach node ∈ ordered_node_set
          do {
              node_duplicate (node);
              modify_network_structure ( );
              update_time ( );
          } while (duplication_gain (node) > 0
                  && slack (node) ≤ (smin+ε))
  } while (delay decreases && runtime allows)
```

Figure 2. Outline of the Gate Duplication Algorithm

To do this, each critical node is associated with a *duplication gain*, which denotes the potential delay reduction, if this node is duplicated. The nodes with the maximum duplication gain are good candidates to be duplicated first. When a gate is duplicated, the structure of the network is *locally* modified by changing the fanins/fanouts at the related nodes. The arrival/required times at all nodes of the network are *globally* updated using the *update-time* procedure. The process of delay trace, duplication gain computation, gate duplication and delay information updating is iterated until no further delay improvement can be made.

To reduce the delay by gate duplication effectively, a critical node (i.e., v_1 in Figure 1) is *isolated* (or *split*)

from the fanouts of v_i as the only one fanout of duplicated gate (i.e., v_i' in Figure 1(b)). That will, to the largest extent, reduce the arrival time at v_i' by $\delta \cdot (f_i - 2)$, where f_i is the number of fanouts of v_i . Since the arrival time at v_1 depends on the maximum of its two fanin's arrival times, the difference between the two fanin's arrival times prior to gate duplication is another important factor that affects the arrival time at v_1 after duplication. For example, when this difference value is less than $\delta \cdot (f_i - 2)$, the reduction of the arrival time at v_1 is determined by the former instead of the latter. Similarly, the potential increase of the required times at v_i's fanins due to gate duplication depends on f_i as well as the difference between the required times at v_i's fanouts prior to duplication. More specifically, assume that v_{min1} and v_{min2} are such two fanouts of the critical node v_i that

$$\left.\begin{aligned} r_{\min 1} &= \min_{v_j \in FO_i} (r_j - d_j) \\ r_{\min 2} &= \min_{v_j \in (FO_i - v_{\min 1})} (r_j - d_j) \end{aligned}\right\} \quad (2)$$

We define the *required time difference* of v_{min2} and v_{min1} as $r_{dif} = r_{min2} - r_{min1} \geq 0$. Also, let v_n be the other fanin (apart from v_i) of v_{min1}. We define the *arrival time difference* of v_{min1}'s fanins as $a_{dif} = a_i - a_n$ (obviously, $a_{dif} \geq -\varepsilon$ because v_i is critical). It can be seen that, if $f_i < 3$, no delay improvement can be made using gate duplication at node v_i . As long as $f_i \geq 3$ and gate duplication is performed on v_i (see Figure 1), the minimum of a_{dif} and $\delta \cdot (f_i - 2)$ is the decrease of the arrival time of node v_{min1} . If v_{min1} has only one fanin, $a_{dif} = \infty$ which means the number of v_i's fanouts dominates the arrival time of v_{min1} during gate duplication. On the other hand, gate duplication at v_i potentially increases the required times at v_i's fanins by the minimum of $(r_{dif} + \delta)$ and $\delta \cdot (f_i - 1)$ and, at the same time, increases the arrival times at v_i's fanins by δ . Therefore, the net effect of gate duplication on the slack times at v_i's fanins depends upon $min\{r_{dif}, \delta \cdot (f_i - 2)\}$ instead of $min\{r_{dif} + \delta, \delta \cdot (f_i - 1)\}$. The above observations lead us to the following definition for the *duplication gain*:

$$\begin{aligned} duplication-gain(v_i) = {} & \alpha \cdot \min\{r_{dif}, \delta \cdot (f_i - 2)\} \\ & + \beta \cdot \min\{a_{dif}, \delta \cdot (f_i - 2)\} \end{aligned} \quad (3)$$

where α and β accounts for the contribution of the increased required times at v_i's fanins and the reduced arrival time at v_{min1} to the circuit delay improvement, respectively. Both α and β provide the duplication gain with a good global view. Intuitively, we can use the numbers of v_i's transitive fanins and v_{min1}'s transitive fanouts which are critical nodes to account for the contributions. For the computational simplicity, we set α and β to be the number of levels of v_i from the primary inputs and the primary outputs, respectively.

In order to select just the *true* critical nodes for gate duplication, it is necessary to update the arrival/required times of the nodes in the network dynamically. Without loss of generality, we assume that, as a result of gate duplication, the arrival time (a_j) at some node v_j is decreased by Δa, and the required time (r_k) at some node v_k is increased by Δr. The updating operation is performed recursively using the *forward/backward propagation* algorithms as shown in Figure 3.

```
update_time
  /* do forward arrival-time propagation */
  forward_propagate (v_j , Δa);
  /* do backward required-time propagation */
  backward_propagate (v_k , Δr);

forward_propagate (v_j , Δa)
  if (v_j is primary_output || Δa = 0)  return;
  a_j = a_j − Δa ; s_j = s_j − Δa ;
  foreach node v_n ∈ FO_j
     if (|FI_n| < 2)
        forward_propagate (v_n , Δa);
        continue;
     endif
     new_Δa = a_n − max_{v_i ∈ FI_n} (a_i + d_n);
     forward_propagate (v_n , new_Δa);

backward_propagate (v_k , Δr)
  if (v_k is primary_input || Δr = 0)  return;
  r_k = r_k + Δr ; s_k = s_k + Δr ;
  foreach node v_m ∈ FI_k
     if (|FO_m| < 2 )
        backward_propagate (v_m , Δr);
        continue;
     endif
     new_Δr = min_{v_i ∈ FO_m} (r_i − d_i) − r_m ;
     backward_propagate (v_m , new_Δr);
```

Figure 3. Updating the Arrival/Required Times

4 Results

Our gate duplication algorithm was implemented in the *SIS* framework [7]. In the experiments, we used example circuits from the MCNC benchmark set. We first optimized each circuit using the standard script (*script.rugged*) and then technology-decomposed it into network of 2-input NAND gates and inverters. We then applied our gate duplication algorithm to reduce the circuit delay. We compared our results with the networks that are optimized by the *speed-up* algorithm [3]. Also, we compared the results of optimizing the circuit first by *speed-up* and followed by using gate duplication. The delay-optimized networks are finally mapped by the *SIS* technology mapping (under the minimum delay mode) to get the circuit area and delay.

Table 1 shows the circuit delay and area obtained before and after using our algorithm. We see that, on average, our algorithm reduces the circuit delay by 13.8% with the area increase of 5.5%. Also listed in this Table are the results obtained with *speed-up* algorithm for comparison. On average, *speed-up* is about 2% better than ours in terms of the delay. However, the area penalty is high, on average 17.6% more than the original

Table 1. Comparison of Our Algorithm and *Speed-up* Algorithm on Delay/Area

Circuit	Before opt.		Opt. by *gate duplication* algorithm				Opt. by *speed-up* algorithm			
	delay	area	delay	area	delay improv.	area penalty	delay	area	delay improv.	area penalty
9symml	14.6	24956	12.8	29578	12.3%	18.5%	13.1	29040	10.3%	16.4%
C432	44.9	35717	39.6	45702	11.8%	28.0%	36.4	39440	18.9%	10.4%
C880	40.1	45862	24.5	46562	38.9%	1.5%	21.0	69167	47.6%	50.8%
C1908	29.1	56285	26.5	58230	8.9%	3.5%	23.3	71852	19.9%	27.7%
C3540	44.6	194649	49.5	208503	–11.0%	7.1%	36.1	208469	19.1%	7.1%
C5315	31.1	169862	26.4	169879	15.1%	0.0%	26.7	205544	14.1%	21.0%
C7552	48.3	284461	46.0	290257	4.8%	2.0%	46.5	273693	3.7%	-5.0%
alu2	30.2	44032	26.4	44657	12.6%	1.4%	28.4	57504	6.0%	30.6%
alu4	34.5	80106	30.9	82914	10.4%	3.5%	31.1	97793	9.9%	22.1%
apex6	20.2	83652	17.3	84183	14.4%	0.6%	20.4	81315	–1.0%	–2.8%
apex7	12.1	24677	9.5	25319	21.5%	2.6%	11.1	24999	8.3%	1.3%
f51m	18.7	13023	17.7	13912	5.3%	6.8%	16.8	20031	10.2%	53.8%
frg1	14.7	21784	9.6	20972	34.7%	–3.7%	8.7	20686	40.8%	–5.0%
Average	–	–	–	–	**13.8%**	**5.5%**	–	–	**16.0%**	**17.6%**

one. Actually, our algorithm is not intended to serve as a substitute for *speed-up*. Instead, it can be combined with *speed-up* as well as any other timing optimization algorithm to provide a further delay reduction. The results of using our gate duplication algorithm based on the network optimized by *speed-up* are shown in Table 2. It can be shown that, on average, our algorithm achieves a further delay reduction of 8.9% with an area penalty of 3.5%. The maximum delay improvement is as high as 24%. This indicates that our gate duplication algorithm is very effective for reducing the circuit delay.

5 Conclusion and Future Work

We proposed a gate duplication algorithm for delay optimization at the technology-independent logic synthesis stage. The basic relations between the gate duplication and the delay reduction were described. Our approach, especially when combined with other timing optimization algorithms, produces significant reductions in circuit delay with a small area penalty. Currently, we plan to extend our gate duplication scenario to technology-dependent timing optimization.

Table 2. Performance of the Combination of *Speed-up* and Our Algorithm

Circuit	*Speed-up*	*Speed-up* + Our Algorithm		
	delay / area	delay / area	delay improv.	area penalty
9symml	13.1 / 29040	11.9 / 31040	10.7%	6.9 %
C432	36.4 / 39440	36.3 / 41267	0.3%	4.6%
C880	21.0 / 69167	20.8 / 71606	1.0%	3.5%
C1908	23.3 / 71852	21.9 / 75499	6.0%	5.1%
C3540	36.1 / 208469	34.9 / 211566	3.3%	1.5%
C5315	26.7 / 205544	24.6 / 210860	7.9%	2.6%
C7552	46.5 / 273693	43.2 / 273639	7.1%	0.0%
alu2	28.4 / 57504	21.7 / 62641	23.6%	8.9%
alu4	31.1 / 97793	28.4 / 99017	8.7%	1.3%
apex6	20.4 / 81315	15.5 / 82866	24.0%	1.9%
apex7	11.1 / 24999	9.8 / 25286	11.7%	1.1%
f51m	16.8 / 20031	15.3 / 20561	8.9%	2.6%
frg1	8.7 / 20686	8.5 / 21777	2.3%	5.3%
Average	–	–	**8.9%**	**3.5%**

References

[1] K. C. Chen and S. Muroga. Timing Optimization for Multi-Level Combinational Circuits. In Proceedings of the Design Automation Conference, pp.339-344, 1990.

[2] G. De Micheli. Performance-Oriented Synthesis of Large Scale Domino CMOS Circuits. IEEE Trans. on Computer-Aided Design, vol. CAD-6, pp.751-765, 1987.

[3] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. In Proceedings of the International Conference on Computer-Aided Design, pp.282-285, 1988.

[4] H. J. Touati, H. Savoj, and R. K. Brayton. Delay Optimization of Combinational Logic circuits by Clustering and Partial Collapsing. In Proceedings of the International Conference on Computer-Aided Design, pp.188-191, 1991.

[5] M. Fujita and R. Murgai. Delay Estimation and Optimization of Logic Circuits: A survey. In Proceedings of Asia and South Pacific Design Automation Conference, pp.25-30, 1997.

[6] H. Chen and D. H. C. Du. Path Sensitization in Critical Path Problem. IEEE Trans. on Computer-Aided Design, vol. 12, pp.196-207, 1993.

[7] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Memorandum No. UCB/ERL M92/41, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.