Integrating Logic Retiming and Register Placement *

Tzu-Chieh Tien Hsiao-Pin Su Yu-Wen Tsay
Yih-Chih Chou Youn-Long Lin
Department of Computer Science, Tsing Hua University
Hsin-Chu, Taiwan 300, R.O.C.
{trash,robin,ywtsay,ycchou,ylin}@theda28.cs.nthu.edu.tw

Abstract

Retiming relocates registers in a circuit to shorten the clock cycle time. In deep sub-micron era, conventional pre-layout retiming cannot work properly because of dominant interconnection delay that is not available before layout. Although some retiming algorithms incorporating interconnection delay have been proposed, layout information is still not utilized effectively nor efficiently. Retiming and layout is combined for the first time in this paper. We present heuristics for two key problems: interconnection delay estimation and post-retiming incremental placement. An efficient retiming algorithm incorporating interconnection delay is also proposed. Experimental results show that on the average we can improve the circuit speed by 5.4% targeted toward a $0.5um$ CMOS technology. Scaling down the technology to $0.1um$, as much as 25.6% improvement have been achieved.

1 Introduction

Retiming is a sequential logic optimization technique proposed by Leiserson and Saxe [1]. It relocates registers to reduce the cycle time and/or area while preserving the functionality. Much effort has been made for retiming. Some applied retiming to reduce power[6], to improve testability[7], or for latch-based circuits[8]. Some effort made retiming practical by controlling the initial state of the circuit[9] or reducing the run time[4].

However, without accurate interconnection delay, above-mentioned approach cannot get the best circuit performance when the process technology is down to half-micron or below. This is because the interconnection delay has become the dominant part of the path delay and the interconnection delay is difficult to measure before placement and routing.

Post-layout retiming is a possible approach to take interconnection delay into account. With layout information back-annotated, we can estimate the delay and retime the circuit accordingly. Incremental placement and routing techniques must also be developed in order to retain most P&R decision from the previous iteration.

*Supported in part by the National Science Council, R.O.C., under a contract no. NSC86-2221-E-007-019.

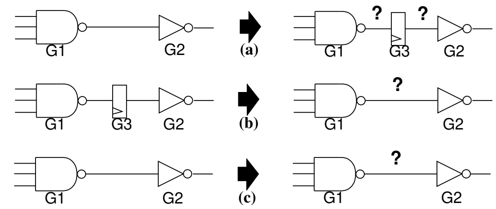

Figure 1: Estimation for three kinds of interconnection delay: (a) adding a register, (b) deleting a register, and (c) an unchanged wire.

In this paper, we propose a post-layout retiming technique. Heuristics have been developed for interconnection delay estimation, retiming incorporating interconnection delay, and post-retiming placement.

The rest of this paper is organized as follows. Three key problems and previous work are described in Section 2. Our system is introduced in Section 3. The retiming algorithm is proposed in Section 4. Heuristics for interconnection delay estimation and post-retiming placement are proposed in Section 5 and 6, respectively. Experimental results are described in Section 7. Finally, Section 8 concludes this paper.

2 Key Problems and Previous Work

We have to deal with three key problems: retiming incorporating interconnection delay, interconnection delay estimation, and post-retiming placement.

Several retiming algorithms taking into account interconnection delay have been proposed [2][3]. If we can annotate a circuit with the interconnection delay value changed after retiming as shown in Figure 1, algorithms in [2][3] would give the optimal retiming solutions. If the path delay monotonicity constraint or the one-way extendable property cannot be satisfied, the algorithms used in [2] and [3] are time-consuming. Nevertheless, due to the placement variation, neither the path delay monotonicity constraint nor the one-way extendable property can be easily satisfied. Using faster retiming algorithms with sub-optimal solutions may be acceptable because, after post-retiming placement, the optimal retiming solution might no longer be optimal.

How to estimate the interconnection delay for further retiming is more important. Three kinds of interconnection delay as shown in Figure 1 need to be estimated: adding a register to a wire, deleting a register from a wire, and an unchanged wire. The delay of an unchanged wire can simply be assumed as the original delay. The delay of a register-deleted wire is also easily estimated by generating a routing.

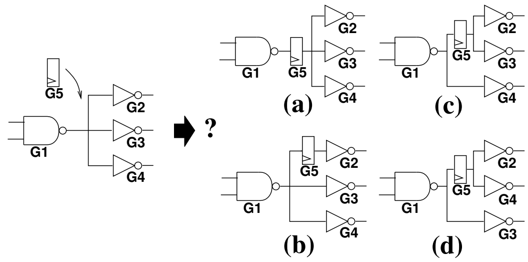

Figure 2: Four possible configurations when a register is placed between $G1$ and $G2$.

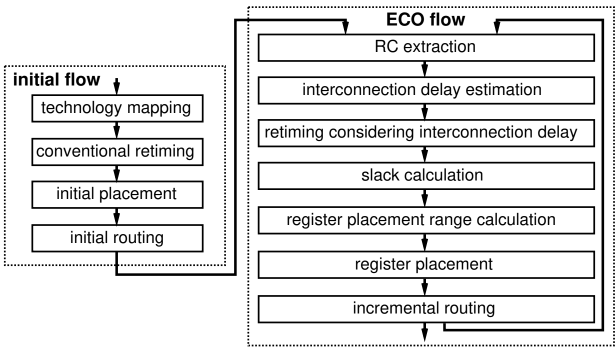

Figure 3: Design flow.

The interconnection delay of a register-added wire is more difficult to estimate. One possible method is to predict the placed location for the added register, generate a routing, and calculate the delay. Methods proposed in [11] can be used to predict the location for the added register, but this needs a powerful incremental post-retiming placement to resolve the possible congestion of registers.

Another difficult problem for interconnection delay estimation is to decide on which fan-out configuration will occur since different configurations will lead to different delay characteristics. For a three-fan-out structure as shown in Figure 2, when estimating the interconnection delay for the wire after inserting a register $G5$ between $G1$ and $G2$, we have to choose among configurations **(a)**,**(b)**, **(c)**, and **(d)**. Before the final retiming is done, we do not know which configuration will occur.

The third key problem for post-layout retiming is to design an effective post-retiming placement procedure. Though some incremental placement heuristics such as those in [10] have been proposed, methods specific to the post-retiming will perform better.

3 System Flow

The design flow is shown in Figure 3. The circuit is technology mapped using **SIS**[13]. A retiming transformation ignoring wire delay is applied on the circuit. After initial placement and routing, the design enters the ECO (engineering change order) flow. The distributed Elmore interconnection delay[12] is back-annotated from the RC extractor. Then the unknown interconnection delay is estimated. Retiming is applied according to the estimated delay. If it succeeds, delay slacks are computed for forward-annotation. The range for placing a register is calculated according to the slack. Then a bipartite matching is carried out to decide on where to place each register. Locations of all gate cells are retained. The iteration of the ECO flow stops when retiming fails or the user-given time limit expires.

4 Retiming Algorithm

The algorithms proposed in [2] and [3] can be used in the post-retiming design flow. However, because neither the path delay monotonicity constraint nor the one-way extendable property is easily satisfied, we develop a new efficient retiming algorithm incorporating interconnection delay as shown in Figure 4. Signals are propagated from primary inputs for each path to identify the new location of each register. A function $a(v)$ is used for the signal propagation and the retiming value $r(v)$ can be obtained from $a(v)$. $a(v)$ is dynamically updated by queuing and dequeuing gates when traversing the whole circuit. Detail description for this algorithm can be seen in [5] and http://theda28.cs.nthu.edu.tw/~trash.

5 Interconnection Delay Estimation

Before or during the retiming process, the unknown interconnection delay must be estimated. As mentioned in Section 2, three kinds of delay should be measured: adding a register to a wire, deleting a register from a wire, and the unchanged wire. The interconnection delay of an unchanged wire is set to the original value since the post-retiming physical synthesis is done incrementally. In our approach, the interconnection delay of a register-deleted wire is calculated according to the relative cell locations.

For the interconnection delay due to adding a register, we first decide on where to place the register on the layout. We employ a heuristic that simply put the register at the geometric center of its fan-in and fan-out cells as depicted in Figure 5. Then, to be explained in the next section, we round the location to the slot occupied by a register nearest the geometric center. The interconnection delay is estimated according to the slot position.

Another problem we have introduced in Section 2 is the fan-out configuration decision. Before retiming is finalized, we cannot predict which configuration will be chosen. Different configurations lead to different interconnection delays and may cause different retiming solutions. We conservatively use the configuration with the largest interconnection delay. For the example of a three-fan-out node shown in Figure 2, the interconnection delay from gate $G1$ to register $G5$ is estimated based on configuration **(b)**, and the interconnection delay from register $G5$ to gate $G2$ is calculated using configuration **(a)**.

6 Post-Retiming Placement

Whether a predicted cycle time could be achieved is up to the post-retiming placement.

It is difficult to simultaneously place registers and alter gate cell locations according to the assigned slacks. So we decide to keep all locations of gate cells unchanged. The interconnection delay estimated for the unchanged wire or the register-deleted wire is easily met. The location for the added register is naturally predicted to the slot nearest the geometric center as shown in Figure 5.

After analyzing many circuits, we observe that most registers are not necessary to be placed at the estimated locations. Only the register that starts a path or ends a path with delay close the cycle time should be placed near the estimated location. The range to place a register is controlled according to slacks.

A slack value is assigned for each fan-in and fan-out node of a register. It is the upper bound of the interconnection delay between the node and the register, and is defined as

$$(\textit{cycle time} - \textit{path delay}) \;/\; 2 \;+$$

```
asap_q_int_retiming(c)
      /* initialization */
1.    for each v ∈ V
2.         r(v) ← 0
3.         a(v) ← −∞
4.    endfor
5.    for each v ∈ PI
6.         a(v) ← 0
7.         enqueue(queue,v)
8.    endfor
9.    for each v ∈ PO
10.        a(v) ← 0
11.   endfor
      /* traverse the circuit */
12.   while queue ≠ ∅
13.        node u ← dequeue(queue)
14.        for each v is a fan-out node of u
15.             a_int ← a(u) + d(u) − w(u,v) * c
16.             r_int ← ⌈ a_int / c ⌉ − 1
17.             a_v ← a_int + est_d(u,v)
18.             r_v ← ⌈ ( a_v + d(v) ) / c ⌉ − 1
19.             if r_v < r(v)
20.                  r_v ← r(v)
21.             endif
                /* failure condition */
22.             if v ∈ PO
23.                  if r_v > 0
24.                       return FAILURE
25.                  endif
26.             endif
27.             if r_v > r_int
28.                  a_reg ← a_int + est_d(u,v,0)
29.                  r_reg ← ⌈ a_reg / c ⌉ − 1
30.                  if r_reg > r_int
                     /* a register would be put
                        in the edge from t to u */
31.                       r(u) ← r(u) + 1
32.                       for each t is a fan-in node of u
33.                            a_u ← r(u) * c + est_d(t,u,w(t,u))
34.                            a(u) ← max(a(u),a_u)
35.                       endfor
36.                       recover_queue_and_value_a_for_fanouts_of_u()
                          /* break the loop of line 14 */
37.                       break
38.                  endif
                     /* else a register would be put
                        in the edge from u to v */
39.                  for each t is a fan-in node of v
40.                       a ← r_v * c + est_d(t,v,w(t,v))
41.                       a_v ← max(a_v,a)
42.                  endfor
43.             endif
                /* update a(v) */
44.             if a_v > a(v)
45.                  a(v) ← a_v
46.                  r(v) ← r_v
47.                  if v ∉ queue
48.                       enqueue(queue,v)
49.                  endif
50.             endif
51.        endfor
52.   endwhile
53.   return SUCCESS
```

v:	vertex corresponding to a gate in the circuit
$d(v)$:	gate delay of v
$w(u,v)$:	number of registers between vertices u and v
$est\_d(u,v)$:	estimated interconnection delay between u and v
$est\_d(u,v,i)$:	estimated interconnection delay of ith section of edge (u,v) when $w(u,v) > 0$
$r(v)$:	retiming value on v
$a(v)$:	a propagation function to calculate $r(v)$ by $r(v) = 0$ if $v \in PI$; $r(v) = \lceil (a(v)+d(v))/c \rceil - 1$ if $v \notin PI$
c:	desired cycle time

Figure 4: The **asap_q_int_retiming** algorithm.

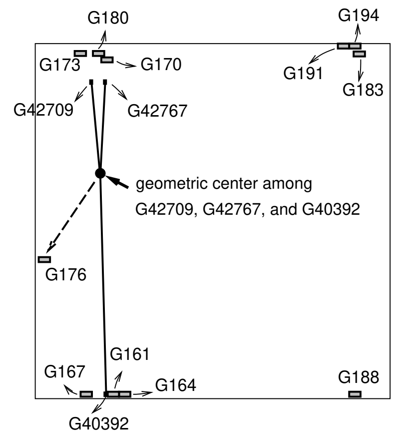

Figure 5: Geometric center of nodes and the slot nearest the geometric center.

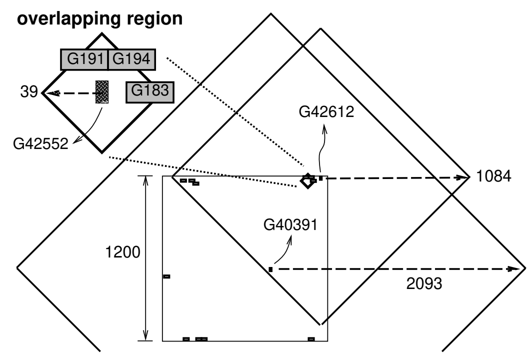

Figure 6: Register placement range calculation according to slacks.

$$(\textit{ the estimated interconnection delay }) \qquad (1)$$

(*cycle time* − *path delay*) is the total slack for the path, and is divided by two for the starting and ending parts of the path. Under this definition, the slack for the node on the critical path connected to the starting or the ending register is exactly the estimated interconnection delay. This makes the starting register or ending register very possible to be placed to the estimated location.

According to the forward-annotated slack, the placement range for each register could be calculated. A region for each node connected to a register is calculated for placing the register with interconnection delay smaller than the slack. As shown in Figure 6, the overlapping region for all nodes connected to the register is the final placement range for the register.

Since all gate cell locations are unchanged and register cells are removed, there leave many slots on the layout. All registers after logic retiming will be placed into these slots. If the number of registers increases after retiming, we add more slots to both ends of each cell row.

A register may have more than one slot to be placed into. Decision for register-to-slot assignment is done by a bipartite matching as shown in Figure 7.

7 Experimental Results

We conduct experiment using some large circuits from the ISCAS89 and the LGSynth93 benchmark sets. The circuits with name extension ".i" are from the ISCAS89 benchmark set and those with ".l" are from the LGSynth93 benchmark

Circuit	0.1um				0.25um				0.5um			
	Init	Final	CPR	I/G	Init	Final	CPR	I/G	Init	Final	CPR	I/G
s5378.i	2.419	1.839	23.98%	1.41	2.965	2.724	8.12%	0.66	4.716	4.342	7.93%	0.37
s5378.l	1.872	1.851	1.11%	1.02	3.175	2.688	15.34%	0.47	5.213	4.853	6.89%	0.26
s9234.i	2.177	1.818	16.47%	1.34	2.991	2.646	11.51%	0.69	4.366	3.973	9.01%	0.39
s9234.n.l	2.652	2.482	6.41%	1.26	4.158	4.037	2.91%	0.57	7.308	6.883	5.81%	0.42
scf.l	1.644	1.359	17.29%	2.04	2.934	-	-	0.38	5.415	-	-	0.22
bbara_bbtas.l	2.046	1.578	22.89%	0.81	2.752	-	-	0.97	5.920	5.842	1.32%	0.23
dsip.l	1.245	-	-	0.96	2.135	-	-	0.71	3.750	-	-	0.52
s298.l	1.613	-	-	1.85	3.326	3.153	5.18%	1.07	5.555	5.358	3.54%	0.64
bigkey.l	1.796	1.335	25.64%	2.28	2.558	2.458	3.92%	1.91	3.660	-	-	1.66
s35932.i	2.282	2.145	6.02%	1.72	3.773	3.330	11.75%	0.99	5.815	5.560	4.39%	0.63
s38417.i	3.279	-	-	1.59	5.235	4.698	10.24%	0.75	8.965	8.551	4.62%	0.33
s38417.l	3.183	-	-	1.20	5.337	4.773	10.57%	0.58	8.546	7.954	6.93%	0.32
clma.l	6.544	5.797	11.41%	3.10	9.746	9.467	2.87%	1.38	17.013	16.400	3.60%	0.78
average			14.58%				8.24%				5.40%	

I/G : average interconnection delay over average gate delay   CPR : Clock Period Reduction

Table 1: Performance Improvement

register	slots
G170	G170,G173,G176,G180 G183,G191,G194
G173	G161,G164,G167,G170,G173 G176,G180,G183,G191,G194
G176	G161,G164,G167,G170,G173 G176,G180,G183,G191,G194
G180	G161,G164,G167,G170,G173 G176,G180,G183,G191,G194
G183	G183,G191,G194
G188	G161,G164,G167,G188
G191	G183,G191
G218	G161,G164,G167,G170 G173,G176,G180,G188

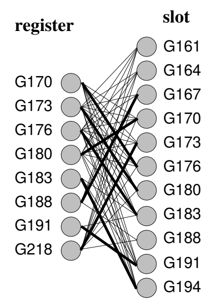

Figure 7: Register assignment by bipartite matching.

set. All are run on SUN SPARC 5, SPARC 10, or SPARC 20 workstations.

Each circuit is synthesized for three kinds of process technologies to compare the retiming results under the different weights of interconnection delay. The 0.5um technology is from the 0.5um CMOS cell library of TSMC[14], and the 0.25um and 0.1um technologies are obtained by scaling down the 0.5um technology. We compare the retiming results with and without interconnection delay considerations.

Cycle time improvements are listed in Table 1. The initial clock periods are listed in columns 2, 6, and 10. The final clock periods are listed in columns 3, 7, and 11. A field is filled with "-" if the circuit cannot be retimed with interconnection delay consideration. The ratio of the average interconnection delay over average gate delay of a circuit is also listed in Table 1 for reference. The results show that our approach successfully reduces the clock period by 5.4% on the average for 0.5um technology. When the interconnection delay become more dominant in 0.25um and 0.1um technologies, our approach achieves more improvements of 8.2% and 14.6%, respectively.

8 Conclusion

We have presented a post-layout retiming technique attempting to close the gap between conventional retiming and physical layout. Heuristics are proposed for two key problems of post-layout retiming: interconnection delay estimation and post-retiming placement. And an efficient retiming algorithm incorporating interconnection delay is also developed.

Experimental results showed an average of 5.4% improvement in cycle time targeted towards a 0.5um technology compared with that considering only gate delay. Experiments were also made for scaled down 0.25um and 0.1um technologies and the results showed our capability of further performance improvement in future technology.

References

[1] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, vol.6, pp.5-35, 1991.

[2] T. Soyata, E. Friedman, and J. H. Mulligan, Jr., "Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process," *IEEE Trans. on Computer-Aided Design*, vol.16, pp.105-120, Jan. 1997.

[3] K. N. Lalgudi and M. C. Papaefthymiou, "Retiming Edge-Triggered Circuits Under General Delay Models," *IEEE Trans. on Computer-Aided Design*, vol.16, pp.1393-1408, Dec. 1997.

[4] N. Shenoy and R. Rudell, "Efficient Implementation of Retiming," in *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design*, pp.226-223, Nov. 1994.

[5] W.-J. Chen, W.-K. Cheng, T.-F. Lee, C.-H. Wu, and Y.-L. Lin, "On the Relationship between Sequential Logic Retiming and Loop Folding," in *Proc. of Synthesis and Simulation Meeting and International Interchange, SASIMI'93, Japan*, pp.384-393, Oct. 1993.

[6] C. V. Schimpfle, Sven Simon, and Josef A Nossek, "Optimal placement of registers in data paths for low power design," in *Proc. of the 1997 IEEE Int. Symp. on Circuits and Systems, ISCAS'97*, vol.3, pp.2160-2163, June 1997.

[7] A. El-Maleh, T. E. Marchok, J. Rajski, and W. Maly, "Behavior and Testability Preservation Under the Retiming Transformation," *IEEE Trans. on Computer-Aided Design*, vol. 16, pp.528-542, May 1997.

[8] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of Circuits with Single Phase Transparent Latches," in *Proc. of the Int. Conf. on Computer Design*, pp.86-89, Oct. 1991.

[9] H. J. Touati, and R. K. Brayton, "Computing the Initial States of Retimed Circuits," *IEEE Trans. on Computer-Aided Design*, vol.12, pp.157-162, Jan. 1993.

[10] C.-S. Choy, T.-S. Cheung, and K.-K. Wong, "Incremental Layout Placement Modification Algorithms," *IEEE Trans. on Computer-Aided Design*, vol.15, pp.437-445, Apr. 1996.

[11] M. Pedram, and N. Bhat, "Layout Driven Technology Mapping," in *Proc. of the ACM/IEEE Design Autom. Conf.*, pp.99-105, June 1991.

[12] W. C. Elmore, "The transient response of dampen linear networks with particular regard to wide band amplifiers," *J. Appl. Phys.*, vol.19, pp.55-63, 1948.

[13] 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, May 1992.

[14] *TCB650 Library, 0.5um Standard Cell Data Book*, TSMC, Apr. 1996.