LAYOUT AWARE RETIMING

A. Ranjan\(^1\)  A. Srivastava\(^2\)  V. Karnam\(^3\)  M. Sarrafzadeh\(^4\)

\(^1\) Monterey Design Systems, Sunnyvale, CA.
\(^2\) Computer Science Department, University of California at Los Angeles, CA.
\(^3\) Advanced Micro Devices, Sunnyvale, CA.
\(^4\) Computer Science Department, University of California at Los Angeles, CA.

\(^1,2,3,4\) (ranjan@mondes.com, apple@cs.ucla.edu, krv@cmdmail.amd.com, majid@cs.ucla.edu)

ABSTRACT

Retiming is a technique for optimizing sequential circuits. Up till now, all the major work on retiming has concentrated on retiming at pre-layout stage where interconnect delays of the circuit are not taken into account while doing retiming. In this work we propose a tighter integration of layout and retiming stages. We propose in-place incremental retiming and placement methodology, in which retiming and placement proceed incrementally. By doing so we show that a more accurate assessment of number of registers needed to achieve minimum or given cycle time can be made. Our results show that incorporating wire delays at the retiming level achieves the required cycle time by adding \(\approx 60\%\) lesser registers and hence saving \(\approx 10\%\) chip area as compared to the case when no wire delay information is used during retiming.

1. INTRODUCTION

Retiming is a sequential logic optimization technique. Leiserson and Saxe provided the first formulation and theoretical solution to the retiming problem[1][2]. Retiming uses the flexibility provided by repositioning registers (memory elements) to optimize a circuit to achieve one of several objectives: minimize clock period, minimize number of registers, minimize registers for a given clock period.

Major work on the aforementioned three variations have been discussed in [1][2]. [3] and [4] report minimum area retiming algorithms. Retiming for low power designing has been reported in [5] and [6]. Retiming to improve testability has been discussed in [7] and [8].

However, in the absence of interconnect delay estimation, the above approaches will fail to accurately achieve the best circuit performance. It is quite possible that the placed and routed circuit performs much worse than that predicted by pre-layout retiming. Many researchers have proposed several methodologies to model and include interconnect delays while retiming. Prominent among these are [9], [10], [12], [13] and [14]. Though all these techniques (except [12]) incorporate interconnect delay into retiming routine, they suffer from lack of accurate modeling of interconnect delay.

In [12] the authors have tried to apply retiming on the post-layout circuit. Though they try to take interconnect delay into account by doing a post-layout retiming, it still falls short of providing a tight coupling between the two. Their work deals with refining the solution of initial retiming. They also do not discuss the impact of their post-layout retiming approach on the final chip area.

In our work, we propose to integrate the retiming and layout stages by providing a feedback from layout to the retiming routine. One major advantage of this approach is that we are not working with the post-layout circuit and have the flexibility of integrating combinational logic synthesis at the retiming level.

The rest of the paper is organized as follows. In the next section, Section 2, we will discuss retiming problem formulation and our proposed methodology. Interconnect modeling and lumping of wire delays to gate delays will be discussed in Section 3. In Section 4, we present our incremental retiming and placement algorithm in detail. The integrated retiming and placement flow is discussed in Section 5. Section 6 is for experimental setup and results. We conclude our paper with Section 7 by providing directions for future work.

2. PROBLEM FORMULATION

In the following sub-sections we will discuss the retiming problem, retiming with interconnects taken into account and our proposed methodology.

2.1. Retiming Problem

![An example circuit](figure1.png)
To provide a motivation into the retiming problem we will give a simple example shown in Figure 1 and discussed in [17]. We assume that each gate has the delay shown above it. The solid black rectangles are the registers. The best clock period for such a circuit is given by the maximum delay of a path consisting of gates. The clock period for the circuit shown in Figure 1 is 6 units.

![Figure 2. Retiming for minimum registers](image1)

An equivalent circuit with 3 registers and a clock period of 4 units is shown in Figure 2. Another equivalent circuit with a clock period of 2 units is shown in Figure 3. The circuit in Figure 2 has the minimum number of registers. Whereas the circuit in Figure 3 has minimum clock period but at the cost of adding an extra register in the circuit.

![Figure 3. Retiming for minimum clock period](image2)

This simple example itself shows a trade-off between minimum clock period achievable and the number of registers in the circuit. Thus a simple re-configuration of registers yields different designs with varying area costs and clock periods.

2.2. Retiming Problem with Delays on the Nets

In the previous subsection we looked at how repositioning registers may achieve significantly different results in terms of cycle time and number of registers in the optimized circuit. Here we will discuss the impact of wire delays on retiming. Once again we will consider the example circuit of Figure 1 with added wire delays which has been reproduced for convenience as Figure 4.

![Figure 4. Retiming with interconnect delays incorporated and an example placement](image3)

The numbers on the wires in this circuit denote the delay in propagating a signal across this net. Hence the first five nets have delays of 2 each and rest of the nets have delays of 1 each.

Now let us once again explore the configurations of circuits in figures 2 and 3. With the wire delays taken into account, the clock period of circuit in Figure 2 is 7 units (assume that a register on a net divides the net into two nets, each having delay half of the original net). Also the clock period for circuit in Figure 3 is 7 units. Both of these are significantly different from the previous case when no wire delay was taken into account. There is one more interesting observation. The circuit in Figure 3 has more registers than the circuit in Figure 2 for the same clock period. This simple example demonstrates that by ignoring interconnect delay at the retiming stage, we might be obtaining wrong optimization results. Here we are adding extra registers in circuit of Figure 3 in the hope that we have reduced the clock period of the circuit. Whereas at the layout level we have increased the area by adding more registers without any gain in the clock period over that of Figure 2.

2.3. Our Proposed Methodology

The proposed flow is shown in Figure 8. As is clear from the figure, our main objective is to feed-back correct wire delays (by constructing a layout) to the Retiming Engine. Based on these new delay estimates, the Retiming Engine may decide to change the previous locations of registers in the netlist. This information will be propagated to the Placement Engine to assign new locations to the registers (and add whatever new registers have been added by the Retiming stage). The updated wire delay information will be fed back to the Retiming Engine. This process continues till we can find no more improvement in the cycle-time (for minimum cycle time objective) or if the given cycle time is achieved (for constrained cycle time objective).

3. INTERCONNECT MODELING

3.1. Wire delay model

In the absence of actual net routing, we need to use a net model to estimate the net delay. Again we use the following model proposed in [15]: For the given coordinates of the terminals of the net (centers of gates in our case), the star point is calculated as the center of gravity of all the terminals on the net. The star point is connected to the driver terminal and all the driven terminals of the net.

Let $X_i$ and $Y_i$ denote the Manhattan distance of terminal $i$ from the star point in $x$ and $y$ directions respectively. Based on this net model, the equivalent RC-model is calculated. In this model each edge has an output resistance $R_o$, a series resistance $R_i$, and a parallel capacitance $c_i$ and each driven terminal has a load capacitance of $C_i$. Based on the Manhattan distance from the star point, these lumped resistances and capacitances are

$$R_i = r_x X_i + r_y Y_i; \quad c_i = c_x X_i + c_y Y_i. \quad (1)$$
where $r_x$, $r_y$, $c_x$, and $c_y$ are the resistances and capacitances (per unit length) in $x$ and $y$ directions respectively.

The Elmore path delay from driver terminal $d$ to the driven terminal $i$ will be estimated as

$$D(d, i) = (R_d + R_i)[c_d + \sum_{j \neq d}(c_j + C_j)] + R_i(c_i + C_i) \quad (2)$$

For our work we have chosen the values for $r_x$, $r_y$, $c_x$, $c_y$, $R_x$, and $C_i$ cited in [18][19].

3.2. Lumping Wire Delay with Gate Delay

Major limitation of all the retiming approaches till now has been their inability to include wire delay while doing retiming. At the logic level, wire delays can not be estimated accurately. We propose to overcome this limitation by lumping the wire delays (obtained at the layout retiming) to the gate delay. At the logic level, wire delays can not be driven terminal has been their inability to include wire delay while doing retiming. Major limitation of all the retiming approaches till now has been their inability to include wire delay while doing retiming. At the logic level, wire delays can not be estimated accurately. We propose to overcome this limitation by lumping the wire delays (obtained at the layout stage) with the gate delays in an intelligent fashion.

The cases where we need to lump wire delay to the gate delay are explained below.

3.2.1. No register at the fanin or fanout of the gate

![Figure 5. No register at the fanin or fanout of the gate](image)

The delay of the gate $A$ is modified as

$$\text{Delay}_{\text{final}}(A) = \text{Delay}_{\text{initial}}(A) + \max[D(A, \text{fanout}(A))]$$

Where $\text{Delay}_{\text{initial}}(A)$ is the initial delay of the gate $A$, $\max[D(A, \text{fanout}(A))]$ is the maximum of all the net delays from the gate $A$ to its fanout gates, given by equation 2. We are using the maximum of all such delays to ensure that we have an upper estimate of the delay.

3.2.2. Registers at one or more fanins

![Figure 6. Registers at the fanin](image)

$$\text{Delay}_{\text{final}}(A) = \text{Delay}_{\text{initial}}(A) + \max[D(A, \text{fanout}(A))] + \max[D(\text{fanin}_v(A), A)]$$

Here $\max[D(\text{fanin}_v(A), A)]$ is the maximum of all the net (which have a register) delays from the fanins of gate $A$ to $A$. Once again we are adding an upper estimate of all such fanin and fanout delays to the gate delay. Register is assumed to be located at the center of the net.

3.2.3. Registers at one or more fanouts

$$\text{Delay}_{\text{final}}(A) = \text{Delay}_{\text{initial}}(A) + \max[D(A, \text{fanout}(A))]$$

Here $\max[D(A, \text{fanout}(A))]$ is the maximum of all the net delays from the gate $A$ to the fanouts of the gate. If a fanout net has a register then only the delay from the gate $A$ to that register is considered.

4. IN-PLACE INCREMENTAL RETIMING AND PLACEMENT

In-place Incremental Retiming and Placement is a methodology in which retiming and placement stages proceed incrementally. The incremental changes done at the retiming stage are reflected in the incremental placement and the updated delay information is fed-back to the retiming stage.

Here first we describe an algorithm for Incremental Retiming. The details of this algorithm and proofs for retiming feasibility can be found in [20]. The network is modeled by the synchronous network graph, $G(V, E, W)$, whose vertex set $V = V' \cup V'' \cup V''\prime$, is partitioned into input, internal and output vertices. Each vertex $v_i$ has a data ready time $t_i$, that is the time at which the signal generated by the corresponding gate is ready with respect to the clock edge. The algorithm is iterative in nature. At each step, the vertices whose data ready time is larger than the required cycle time $T$ are flagged and put in a temporary set $M$. The vertices of set $M$ are retimed by $r = +1$. These steps are repeated until the network is timing-feasible for $T$ or procedure exit returns TRUE. The pseudo-code of the algorithm is shown below.

Of particular interest to the Incremental Placement algorithm is the Retime-by-1 function in the retiming algorithm. That is the only time when registers in the network are being moved. All the other vertices of the network are static if no restructuring is being done for the combinational logic. In Section 3 we have already discussed the methodology for lumping wire delays to the vertex delays. The thing to note is that after Retime-by-1 function decides to move register(s) from the fanout of a vertex to its fanin, there is a change in the delays associated with the corresponding nets. This in turn means that the delays of the vertices at the fanin and fanout of this vertex will incur a change (because of lumping of wire delays with the vertex delays). However, we need to make sure that these are the only vertices (the vertex under consideration, its fanins and fanouts) for which the delay needs to be updated. In the following lemma we reason out that this indeed is the case.

Lemma: If a vertex $v$ is being retimed, only the delay of vertex $v$ and delays of the vertices connected at the fanin or fanout of $v$ need to be updated.

Proof: It is easy to see that the delay of a vertex gets changed only if register(s) are placed or removed from its
Algorithm 1 Incremental Retiming Algorithm

```
retire \{ 
  for(k = 1; k + +) \{ 
  Compute \( t_i \) for each vertex \( v \in V \); 
  \( M = \{v \in V \mid t_m > T\} \); 
  if(\( M = \emptyset \)) return TRUE; 
  else \{ 
  if(exit) return FALSE; 
  Retime-by-1 all vertices in \( M \); 
  if(\( M \cap V^O \neq \emptyset \)) set-outputs; 
  \} 
  \} 
set-outputs \{ 
  Retime-by-1 all primary output vertices not in \( M \); 
  \( S = \{v \in V^G \mid \exists \) a zero weight path from an input vertex to \( v \}\); 
  Retime-by-1 all the input vertices and those in \( S \); 
  \} 
Retime-by-1 \{ 
  If there is a register at the fanout of vertex, move it towards the fans. 
  \} 
exit \{ 
  return \( (k \geq |V|) \); 
  \}
```

fanin or fanout nets (the cases are discussed in Section 3.2).

When a vertex \( v \) is being retimed, the register(s) at its fanout are being moved to its fanin. Hence delay of vertex \( v \) needs to be updated. By retiming vertex \( v \) we are changing register(s) placed at the fanins of its fanout vertices and those at fanouts of its fanin vertices. Hence the delays of the fanin and fanout vertices of vertex \( v \) need to be updated.

The lemma provides a guideline for Incremental Placement algorithm. The idea is that we do not need to construct a full placement to accommodate for new changes in the network. When retiming a vertex, only the positions of the register(s) at its fanin or fanout change and we need to change our placement incrementally to assign new places to these registers. Hence incorporating incremental placement into incremental retiming will change Retime-by-1 function as given in Algorithm 2.

The combination of algorithms 1 and 2 constitutes the algorithm for Layout Aware Retiming.

Algorithm 2 Incremental Retiming and Placement

```
Retime-by-1 \{ 
  If there is a register at the fanout of vertex \( v \), move it towards the fans. 
  \} 
Assign-new-positions to the registers in the layout.
Update-delay-of-vertex(v) 
  for-each-fanin(v) \{ 
  Update-delay-of-vertex(fanin(v)) 
  \} 
for-each-fanout(v) \{ 
  Update-delay-of-vertex(fanout(v)) 
  \} 
Assign-new-positions \{ 
  The moved register is assigned a position in the placement which corresponds to the geometric mean of the terminals of the net on which the register lies. 
  \}
```

SIS library format to TimberWolf1.4.0’s library format. We then generate an initial placement using the Placement Tool described in [21]. The wire delays calculated from this initial placement are supplied to the Incremental Retiming Algorithm (a part of SIS package). All the gate delays are updated as discussed in Sections 3 and 4.

The retiming routine (discussed in Section 4) picks one gate at a time and retimes it by moving registers from its fanout towards its fanin. Based on the new locations of the registers, the associated gate delays (those at the fanin and fanout of the picked vertex) are updated and the information is fed back to the retiming routine. This process continues till a certain objective is met (minimum possible cycle time or constrained cycle time).

6. EXPERIMENTAL RESULTS

We have worked on eleven MCNC benchmark circuits. The description of these circuits can be found in [16]. There are two sets of results which correspond to two different approaches. In the first approach, traditional, we do not supply wire delays to the Retiming Engine. Hence the retiming is done considering only the gate delays. In the second approach, LAR (Layout Aware Retiming), the retiming is done considering wire delays along with the gate delays.

The results associated with our work are shown in Tables 1 and 2 and the corresponding bar charts. The bold-faced results show which approach is performing better in terms of cycle time or the number of added registers. The column % imp clk shows the improvement in clock cycle time achieved by either of the approaches. % increase area column shows the increase in area of the final layout after detailed placement and global routing. We have extracted results for two sets of objectives.

The results in Table 1 correspond to the case when minimum cycle time is the objective for retiming. The set of results in Table 2 are for constrained cycle time objective.

From the results in Table 1 it can be concluded that LAR approach performs better or as good as, in terms of reducing cycle time, the case when no wire delay information is taken into account. One major observation is that in all cases LAR achieves significantly lesser number of registers than that achieved by traditional approach. This may look surprising but has a very logical


Table 1. Comparing LAR with traditional for min-cycle objective

<table>
<thead>
<tr>
<th>clk</th>
<th>initial clk (ns)</th>
<th>initial reg</th>
<th>final clk (ns)</th>
<th>% imp clk</th>
<th>final reg</th>
<th>% increase area</th>
</tr>
</thead>
<tbody>
<tr>
<td>s208</td>
<td>13.1</td>
<td>8</td>
<td>11.4</td>
<td>12.98%</td>
<td>16</td>
<td>17.33%</td>
</tr>
<tr>
<td>s208</td>
<td>17.15</td>
<td>14</td>
<td>14.37</td>
<td>16.21%</td>
<td>46</td>
<td>14.88%</td>
</tr>
<tr>
<td>s441</td>
<td>19.22</td>
<td>41</td>
<td>19.26</td>
<td>29.63%</td>
<td>12</td>
<td>30.46%</td>
</tr>
<tr>
<td>s208</td>
<td>21.7</td>
<td>29</td>
<td>21.6</td>
<td>28.66%</td>
<td>48</td>
<td>28.66%</td>
</tr>
<tr>
<td>s400</td>
<td>19.8</td>
<td>21</td>
<td>19.99</td>
<td>29.22%</td>
<td>49</td>
<td>26.17%</td>
</tr>
<tr>
<td>s410</td>
<td>20.2</td>
<td>30</td>
<td>20.09</td>
<td>21.40%</td>
<td>37</td>
<td>21.40%</td>
</tr>
<tr>
<td>s208</td>
<td>22.71</td>
<td>6</td>
<td>21.41</td>
<td>3.31%</td>
<td>24</td>
<td>24.85%</td>
</tr>
<tr>
<td>s326</td>
<td>17.9</td>
<td>21</td>
<td>14.93</td>
<td>14.7%</td>
<td>63</td>
<td>37.11%</td>
</tr>
<tr>
<td>s441</td>
<td>40.38</td>
<td>32</td>
<td>26.51</td>
<td>24.55%</td>
<td>73</td>
<td>20.58%</td>
</tr>
<tr>
<td>s488</td>
<td>35.14</td>
<td>32</td>
<td>26.51</td>
<td>24.55%</td>
<td>73</td>
<td>20.58%</td>
</tr>
<tr>
<td>s1488</td>
<td>45.17</td>
<td>6</td>
<td>45.9</td>
<td>-1.61%</td>
<td>13</td>
<td>2.18%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>clk</th>
<th>initial clk (ns)</th>
<th>initial reg</th>
<th>final clk (ns)</th>
<th>% imp clk</th>
<th>final reg</th>
<th>% increase area</th>
</tr>
</thead>
<tbody>
<tr>
<td>s208</td>
<td>13.1</td>
<td>8</td>
<td>11.4</td>
<td>12.98%</td>
<td>16</td>
<td>17.33%</td>
</tr>
<tr>
<td>s208</td>
<td>17.15</td>
<td>14</td>
<td>14.37</td>
<td>16.21%</td>
<td>46</td>
<td>14.88%</td>
</tr>
<tr>
<td>s441</td>
<td>19.22</td>
<td>41</td>
<td>19.26</td>
<td>29.63%</td>
<td>12</td>
<td>30.46%</td>
</tr>
<tr>
<td>s208</td>
<td>21.7</td>
<td>29</td>
<td>21.6</td>
<td>28.66%</td>
<td>48</td>
<td>28.66%</td>
</tr>
<tr>
<td>s400</td>
<td>19.8</td>
<td>21</td>
<td>19.99</td>
<td>29.22%</td>
<td>49</td>
<td>26.17%</td>
</tr>
<tr>
<td>s410</td>
<td>20.2</td>
<td>30</td>
<td>20.09</td>
<td>21.40%</td>
<td>37</td>
<td>21.40%</td>
</tr>
<tr>
<td>s208</td>
<td>22.71</td>
<td>6</td>
<td>21.41</td>
<td>3.31%</td>
<td>24</td>
<td>24.85%</td>
</tr>
<tr>
<td>s326</td>
<td>17.9</td>
<td>21</td>
<td>14.93</td>
<td>14.7%</td>
<td>63</td>
<td>37.11%</td>
</tr>
<tr>
<td>s441</td>
<td>40.38</td>
<td>32</td>
<td>26.51</td>
<td>24.55%</td>
<td>73</td>
<td>20.58%</td>
</tr>
<tr>
<td>s488</td>
<td>35.14</td>
<td>32</td>
<td>26.51</td>
<td>24.55%</td>
<td>73</td>
<td>20.58%</td>
</tr>
<tr>
<td>s1488</td>
<td>45.17</td>
<td>6</td>
<td>45.9</td>
<td>-1.61%</td>
<td>13</td>
<td>2.18%</td>
</tr>
</tbody>
</table>

average | 16.67 | 18.04% | 39.5 | 22.31% | 19.28% | 27.33 | 11.62% |

Figure 8. Details of Our Approach

Read BLIF

tech decompp

Generate Initial Placement

Reftime Updated Network

Incremental Placement

Final Placement

Compute Delays

New locations for registers

Read BLIF tech decompp tech mapping Generate Initial Placement Reftime Updated Network Incremental Placement Results achieved ? Final Placement

explanation. With better wire delay information, the Retiming Routine may not add that many new registers and can achieve similar cycle time reduction by relocating existing registers. On the average, LAR adds only 64% more registers to achieve the same reduction in cycle time when compared to traditional which requires addition of 137% more registers. This means that after detailed placement and global routing, LAR would increase the chip area by 11% whereas traditional one results in an increase of 22% chip area.

For the case when constrained cycle time is the objective (second column of Table 2), LAR approach achieves the required cycle time by adding very few registers. Whereas, the traditional approach adds a lot more registers to achieve the required cycle time. On the whole LAR adds 50% lesser registers resulting in a saving of 8% chip area on the average. Once again it shows that in the absence of actual wire delays the circuit optimization can be very inaccurate.

7. CONCLUSION AND FUTURE WORK

In this paper we have presented a methodology to integrate Retiming stage with the Layout stage. Our results show that for both set of objectives, minimizing cycle time or constrained cycle time, our approach can achieve similar or better cycle times and at the same time adding ≈ 60% lesser registers when compared to the traditional approaches. Our approach is better than post-layout retiming approach in that it can very easily integrate other synthesis level optimizations at the retiming stage.

As a future work in this direction we intend to look into concurrent logic optimization, retiming and placement.
Table 2. Comparing LAR with traditional for constrained-cycle objective

<table>
<thead>
<tr>
<th>ckt</th>
<th>desired clk (ns)</th>
<th>initial reg</th>
<th>traditional</th>
<th>final reg</th>
<th>% increase area</th>
<th>final reg</th>
<th>% increase area</th>
</tr>
</thead>
<tbody>
<tr>
<td>s208</td>
<td>2.0</td>
<td>8</td>
<td>11</td>
<td>9</td>
<td>6.3%</td>
<td>8</td>
<td>0%</td>
</tr>
<tr>
<td>s238</td>
<td>15.3</td>
<td>24</td>
<td>29</td>
<td>30</td>
<td>20.9%</td>
<td>15</td>
<td>3.5%</td>
</tr>
<tr>
<td>s441</td>
<td>14.8</td>
<td>10</td>
<td>28</td>
<td>31</td>
<td>21.9%</td>
<td>11</td>
<td>24.7%</td>
</tr>
<tr>
<td>s382</td>
<td>6.8</td>
<td>21</td>
<td>33</td>
<td>21</td>
<td>12.7%</td>
<td>21</td>
<td>0%</td>
</tr>
<tr>
<td>s100</td>
<td>6.0</td>
<td>21</td>
<td>31</td>
<td>25</td>
<td>10.0%</td>
<td>25</td>
<td>4.0%</td>
</tr>
<tr>
<td>s790</td>
<td>37.3</td>
<td>55</td>
<td>55</td>
<td>55</td>
<td>0%</td>
<td>55</td>
<td>0%</td>
</tr>
<tr>
<td>s790</td>
<td>5.0</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>0%</td>
<td>25</td>
<td>0%</td>
</tr>
<tr>
<td>s841</td>
<td>40.7</td>
<td>29</td>
<td>24</td>
<td>24</td>
<td>9.1%</td>
<td>24</td>
<td>0%</td>
</tr>
<tr>
<td>s538</td>
<td>32.9</td>
<td>32</td>
<td>33</td>
<td>32</td>
<td>3.0%</td>
<td>32</td>
<td>0%</td>
</tr>
<tr>
<td>s1488</td>
<td>45.3</td>
<td>6</td>
<td>15</td>
<td>14</td>
<td>2.18%</td>
<td>14</td>
<td>0%</td>
</tr>
</tbody>
</table>

average: 16.67 (29.67) 12.63% (20.5) 4.12%

REFERENCES