Placement Driven Retiming with a Coupled Edge Timing Model

Ingmar Neumann *Wolfgang Kunz*

University of Frankfurt/Main
Department of Computer Science
Electronic Design Automation Group
60054 Frankfurt/Main, Germany

Abstract

Retiming is a widely investigated technique for performance optimization. It performs powerful modifications on a circuit netlist. However, often it is not clear, whether the predicted performance improvement will still be valid after placement has been performed. This paper presents a new retiming algorithm using a highly accurate timing model taking into account the effect of retiming on capacitive loads of single wires as well as fanout systems. We propose the integration of retiming into a timing-driven standard cell placement environment based on simulated annealing. Retiming is used as an optimization technique throughout the whole placement process. The experimental results show the benefit of the proposed approach. In comparison with the conventional design flow based on standard FEAS our approach achieved an improvement in cycle time of up to 34% and 17% on the average.

1 Introduction

Retiming, originally proposed by Leiserson and Saxe [1][2], is a powerful and well-known technique for performance optimization of digital circuits. It is based on relocating registers while preserving the functionality of the circuit. Many improvements and extensions to the original ideas have been developed, like acceleration techniques [3] dramatically speeding up execution time, concepts for integrating retiming into logic synthesis [4], algorithms for retiming level clocked circuits [5][6], algorithms taking register setup and hold times into account [7][8], algorithms for retiming registers with enable inputs [9] as well as algorithms that can improve testability [10].

When optimizing large sequential circuits the use of retiming is very attractive. Conventional state encoding techniques suffer from the state explosion problem and usually fail for circuits containing more than a couple of hundred registers. Retiming on the other hand, does not require an explicit representation of the state set. It operates directly on a netlist description of the circuit and can handle circuits with thousands of registers.

Nevertheless, retiming has encountered only limited acceptance in industrial practice. This is mainly for two reasons. Firstly, a retimed circuit is very hard to verify against the original circuit. However, in recent years there have been advances in the area of sequential equivalence checking for retimed circuits [11], [12]. There is promise that industrial tools capable of verifying retimed circuits will become available within the next years. This paper deals with a second problem affecting the acceptance of retiming in practice. The choice of an accurate timing model in combination with an appropriate retiming algorithm is a delicate issue. With conventional timing models and retiming techniques it often remains unclear whether the predicted performance improvement will still be valid after placement has been performed.

The original FEAS-algorithm developed by Leiserson and Saxe finds a retiming for a circuit such that a given cycle time is met if such a retiming exists. It is based on a simple timing model assuming gate delays to be load independent. Unfortunately, for CMOS technology this model is not accurate enough as gate delays cannot be considered to be load independent and retiming registers changes the loads of the gates.

In [13]-[15], more sophisticated timing models are used. For each edge multiple delay values are calculated covering the two cases that this edge can contain none or at least one register. This is already a strong improvement over previous models. However, these models do not correctly describe situations given at fanout trees, as shown in Fig. 1.

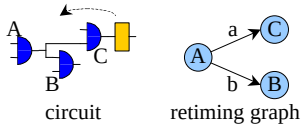

Figure 1: Retiming example

In real circuits, retiming gate *C* changes the load seen by gate *A* and therefore also changes the delay of *A*. This however, does not only affect the arrival time at gate *C* but also at gate *B*. In practice, retiming of registers into fanout trees may change the topology of the affected nets dramatically and can change arrival times even on paths where no registers have been moved. Ignoring this effect may lead to unpredictable results.

The advent of deep-submicron technologies introduced additional difficulties by increasing the influence of wire length on the total delay. Loads resulting from wires are affected by retiming even more than loads resulting from gate inputs and, above all, are not known before placement.

An interesting approach for integrating retiming into the design flow was presented in [16]. Retiming is coupled with partitioning based floor planning allowing performance optimization during an early physical design stage.

An approach to integrate retiming into detailed placement was presented in [17]. After performing a conventional

placement and routing procedure an optimization loop consisting of wire length estimation, retiming and register insertion is entered. Even though this approach produces promising results it does not fully exploit the potential of coupling placement and retiming. Note that a timing-driven placer aggressively tries to shorten wires on critical paths while paying less attention to less critical wires. This can lead to a balance of path lengths reducing the optimization potential for retiming.

In order to take all of the above into account we propose a more accurate timing model for retiming and describe a much tighter coupling between retiming and detailed placement. Our approach does not use retiming for a post-placement optimization, but employs it as an optimization technique throughout the whole placement process.

2 Retiming with Accurate Timing Model

A powerful and efficient retiming algorithm for cycle time minimization is FEAS [1],[2]. The FEAS-algorithm has many attractive properties and therefore we wish to adopt the general FEAS strategy also in this work. However, combining FEAS with an accurate timing model is a delicate issue. The difficulty arises from the fact that FEAS is based on an assumption called *path delay monotonicity constraint* in [14]. It is assumed that for any path of the circuit the data arrival time at a register can never grow if the register is moved backwards in the circuit. The simple timing model of the original FEAS always fulfills this assumption. Unfortunately, this can change if more complex timing models are used. If we consider the wire loads in gate netlists being mapped to typical standard cell libraries, it turns out that in practice the monotonicity assumption holds for two terminal nets in the vast majority of the cases. However, the problem occurs at the point where we model situations as shown in Fig. 1 more realistically. In such cases, the monotonicity assumption indeed may be violated. There is also a second problem with multi-sink nets. Even if monotonicity isn't violated, the FEAS strategy of retiming a critical vertex when its arrival time becomes too large will often lead to suboptimal results. Note that shifting a register into one branch of a multi terminal net will also affect the data arrival time on paths leading through the other branches of that net.

Trying to maintain the efficiency of FEAS on one hand and using a realistic timing model on the other hand requires a more sophisticated solution for retiming critical vertices.

Our new retiming algorithm follows the same strategy as the well known FEAS-algorithm. We perform an alternating sequence of running timing analysis and eliminating constraint violations locally by retiming critical vertices. However, because we use a more complex and accurate timing model extensive modifications were necessary both of the arrival time calculation and of the strategy of deciding when to retime a vertex.

2.1 Timing Model

The circuit is mapped onto a weighted directed graph $G = (V, E)$. Each logic gate is mapped onto a vertex v, being assigned a delay $t_d(v)$ and a retiming value $r(v)$ which initially is 0 and can be incremented during retiming. The logic gate corresponding to a vertex v is denoted by $g(v)$ in the following.

A net driven by gate $g(u)$ with n driven gates, $n \geq 1$, is modeled as a bundle of edges ("branches") $B = \{b_i\} = \{(u, v_i) \subseteq E,\ 1 \leq i \leq n\}$. Each edge $e = (u, v) \in E$ is assigned a weight $w(e)$ denoting its initial number of registers. The number of registers on e during or after retiming is denoted by $w_r(e) = w(e) + r(v) - r(u)$.

Like in [2] G is extended by a hostnode representing the environment of the circuit.

2.1.1 Single Sink Net Edges

For edges of bundles modeling nets with one single sink our timing model is similar to the model proposed in [13]. Each edge $e = (u, v)$ is assigned three delay values as shown in Fig. 2:

- t_w: delay for a signal propagating from u to v in the case that there is no register on e, i.e., $w_r(e) = 0$
- t_i: delay for a signal propagating from u to the data input of a register on e if there is at least one register on e, i.e., $w_r(e) > 0$
- t_o: delay for a signal propagating from the output of a register on e to v, if $w_r(e) > 0$

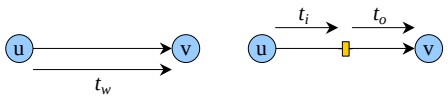

Figure 2: Timing model for single sink net edge

2.1.2 Multi Sink Net Edges

As already mentioned, a net is modeled by a bundle of edges $B = \{b_i\} = \{(u, v_i) \subseteq E\ ,\ 1 \leq i \leq n\}$.

In our model, the actual weight w_r of an edge $b_i \in B$ has an influence on the parameters t_i, t_o, t_w of other edges $b_j \in B$, $j \neq i$. Therefore, we assign to each edge three tables. Each table contains different values for t_i, t_o or t_w, respectively, according to distinguishable register arrangements in B. This allows us to model the fact that retiming one particular gate will change the arrival time of other gates driven by the same net.

We have developed two models differing in table size to be described in the following sections.

2.1.2.1 Complex Model

Let us first consider the situation for the net at the inputs of the registers. We distinguish for each branch $b \in B$, whether $w_r(b) = 0$ or $w_r(b) \neq 0$. By doing so we can model the fact that driving different combinations of cells leads to different sums of gate input capacitances and requires nets of different topologies having different lengths. This leads to different delay values for a signal starting at v implying different values for t_w and t_i for every branch.

For B consisting of n branches we can distinguish 2^n different cases for the calculation of the load that $g(u)$ has to drive. The t_w- and t_i- tables consist of key-value pairs, where the key denotes an identifier for a register arrangement and the value denotes the corresponding t_w- and t_i-value, respectively. Since t_w is only defined for branches without registers and t_i only for branches with at least one register, this leads to 2^{n-1} entries for the t_w- and t_i- tables of each branch.

Fig. 3 shows an example with a bundle consisting of two branches. We can distinguish four different register arrangements leading to two different values for t_i and t_w for each branch. For illustration, in this example, the timing values are given two indices. The first index denotes the branch number, the second index denotes its position in the table.

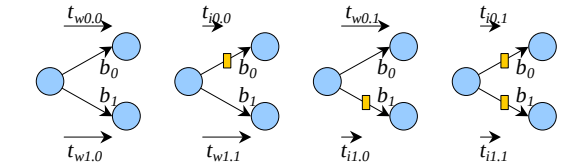

Figure 3: Possible register arrangements for $n = 2$

For the register arrangements (arr.) in Fig. 3 the tables of timing values are as follows:

at branch b_0:

arr.	t_w
1	$t_{w0.0}$
3	$t_{w0.1}$

arr.	t_i
2	$t_{i0.0}$
4	$t_{i0.1}$

at branch b_1:

arr.	t_w
1	$t_{w1.0}$
2	$t_{w1.1}$

arr.	t_i
3	$t_{i1.0}$
4	$t_{i1.1}$

For each branch we have a t_i- and t_w - delay table. Each table row represents a delay value being valid for a particular register arrangement. The second column contains the delay values itself, and the first column contains the number of the register arrangement for which the delay is valid.

Next, we consider the nets at the outputs of the registers and analyze the different cases for the values of t_o. We make the assumption that a bundle B is realized with a minimal number of registers (*register sharing*) when the netlist is created from the retiming graph, as shown in Fig. 4.

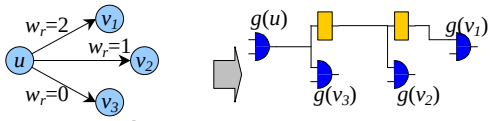

Figure 4: Creating circuit from retiming graph

We believe this assumption to be reasonable because in typical standard cell libraries latch cells in general have a much higher area requirement than simple logic cells. Therefore, if the driving force of a register isn't sufficient to drive a long, widely spread net in most cases it will be more effective to buffer the net and to replicate buffers rather than to replicate area consuming registers.

Consequently, for the values t_o of a branch $b_i \in B$ we need to distinguish for each branch $b_j \in B$, $i \neq j$, whether $w_r(b_i) = w_r(b_j)$ or $w_r(b_i) \neq w_r(b_j)$. This models the fact that under the assumption described above two gates $g(v_i)$ and $g(v_j)$ with $w_r(b_i) = w_r(b_j)$ are driven by the same register. We conclude that also for the tables for t_o we have to distinguish 2^{n-1} different cases leading to 2^{n-1} table entries.

2.1.2.2 Simple Model

The complex model permits an accurate modeling of the real situation but has the drawback of an exponential table growth making it impossible to use this model for large fanout systems. Therefore, we developed a second model with linear growth of the table size.

To reach linear growth, we assume that the values of t_i and the values of t_w, respectively, of a particular branch $b_i \in B$ are the same for all configurations containing the same number of branches b_j with $w_r(b_j) > 0$, $b_j \in B$, $i \neq j$.

In other words, for the calculation of the delay values of a particular branch we only care about how many of the other branches of the bundle carry at least one register and not in which other branches these registers are located. This ignores the differences in the input capacitances of different gate types as well as in the wire loads for the different register arrangements. However, this inaccuracy diminishes as the number of branches in a net increases.

Similarly, for the t_o-values of b_i we assume that they are the same for all configurations containing the same number of branches b_j with $w_r(b_j) = w_r(b_i)$, $b_j \in B$, $i \neq j$.

This limits the size of each table for each branch in a bundle B to $|B|$. In our implementation we use the complex model for bundles with at most four branches. For the circuits examined in our experiments, it turned out that more than 90% of all nets can be described using the complex model. For the few larger nets with more than four sinks we use the simple model.

2.2 Parameter Calculation

This section explains, how the values of t_i, t_o, t_w and t_d are determined from a netlist with a given placement.

The delay $t_d(v)$ of a vertex v denotes the load independent delay of gate $g(v)$. To calculate the edge delays t_i, t_o, t_w we have to determine the loads seen by gates for different register arrangements. A particular load consists of the input capacitances of the driven gates and the capacitance of the net connecting them. The net lengths are predicted for each register arrangement using different methods for the simple and the complex delay model.

2.2.1 Complex Model

In this model, we assume specific register positions for the estimation of the length of the nets.

For the calculation of the t_i and t_w values in a branch bundle $B = \{b_i\} = \{(u, v_i)\}$, $1 \leq i \leq n$, the length $l(w)$ of net w driven by $g(u)$ is required. To estimate $l(w)$ we determine a (minimum size) rectangle R_0 containing $g(u)$ and those gates $g(v)$ that are driven by $g(u)$. If no register is present we use

the half perimeter of R_0 as $l(w)$. If registers are present we determine a rectangle R_1 containing the gates $g(v)$ that are driven by a register. If R_0 and R_1 overlap we assume the register to be positioned inside R_0 and we use the half perimeter of R_0 as $l(w)$. If they don't overlap we determine a minimum size rectangle R_r which touches R_0 and R_1. The center of R_r gives the assumed location for the register. As $l(w)$ we use the sum of the half perimeter of R_0 and the quarter perimeter of R_r. Fig. 5 shows the estimation of $l(w)$ for the register arrangements shown in Fig. 3.

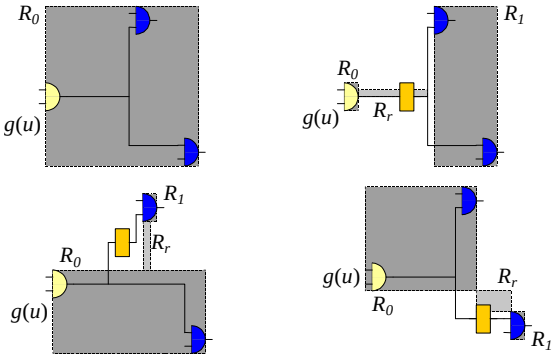

Figure 5: Estimation of length $l(w)$ of a net w driven by $g(u)$ for different cases

For a particular t_w and t_i respectively we obtain

$$t_w = a \cdot \left(l(w) \cdot cpl + \sum_{\text{inputs ports connected to } w} c_{in} \right)$$

and

$$t_i = a \cdot \left(l(w) \cdot cpl + \sum_{\text{inputs ports connected to } w} c_{in} \right) + t_{setup}$$

where a denotes the driving force of $g(u)$, *cpl* stands for the *capacitance per length*, c_{in} denotes the input capacitance of a port and t_{setup} denotes the setup time of the register.

For the calculation of a t_o value of a branch for a particular register arrangement the length $l(w')$ of net w' driven by a register is required. For the estimation of $l(w')$ we determine a rectangle R_0 containing all gates $g(v)$ that are driven by this register and a rectangle R_1 containing $g(u)$ and those gates $g(v)$ driven by $g(u)$ or by other registers. If R_0 and R_1 overlap we assume the register to be positioned inside R_0 and we use the half perimeter of R_0 as $l(w')$. Otherwise, we determine a minimum size rectangle R_r touching R_0 and R_1 and use the center of R_r as assumed location for the register. $l(w')$ is calculated using the sum of the half perimeter of R_0 and the quarter perimeter of R_r. For a particular t_o we obtain

$$t_o = a_{reg} \cdot \left(l(w') \cdot cpl + \sum_{\text{inputs ports connected to } w} c_{in} \right) + b_{reg}$$

where a_{reg} denotes the driving force and b_{reg} the load independent clock to output delay of the register.

2.2.2 Simple Model

Because in the simple model we care only about how many branches of a bundle $B = \{b_i\} = \{(u, v_i)\}$, $1 \le i \le n$ carry a certain number of registers but not in which branches these registers are located, we do not determine register positions for the length estimation. Instead we use a simplified length estimation method using the half perimeter of the rectangle R containing $g(u)$ and all $g(v_i)$ with $(u, v_i) \in B$.

For the calculation of the t_i- and t_w- values of B the length $l(w)$ of net w driven by $g(u)$ is required. Our estimation relies on the observation that the net length grows with increasing number of gates $g(v)$ driven by $g(u)$. The more gates $g(v)$ are driven by registers the fewer need to be driven by $g(u)$. For a register arrangement containing m branches b_j with $w_r(b_j) = 0$, $0 \le m \le n$, we estimate $l(w)$ as follows:

$$l(w) = \frac{\text{perimeter}(R)}{2} \cdot \sqrt{\frac{m+1}{n+1}}$$

We observed, that in most cases taking the square root gives more realistic values than a linear model.

For the load capacitance seen by $g(u)$ we obtain

$$c_{load} = l(w) * cpl + \begin{cases} m \cdot c_{av} & m = n \\ m \cdot c_{av} + c_{reg} & m < n \end{cases}$$

where c_{av} denotes the average input capacitance of all input ports driven by $g(u)$ when no registers occur on any branch, i.e., $w_r(b_i) = 0$ for all $b_i \in B$. For $m < n$, at least one gate $g(v)$ is driven by a register, so $g(u)$ has to drive one register additionally to the m logic gates, and we have to add the input capacitance c_{reg} of the register. (Remember that we assume register sharing as described in Section 2.1.2.1.)

For a particular t_w and t_i we obtain:

$$t_w = a \cdot c_{load} \qquad \text{and} \qquad t_i = a \cdot c_{load} + t_{setup}$$

For determining a particular value of t_o the length $l(w')$ of net w' driven by the register is required. For a particular branch $b_i \in B$ let m' denote the number of branches $b_j \in B$, $j \ne i$, showing the same register count as b_i, i.e., $w_r(b_i) = w_r(b_j)$. We calculate $l(w')$ as follows:

$$l(w') = \frac{\text{perimeter}(R)}{2} \cdot \sqrt{\frac{m'+1}{n+1}}$$

The load seen by the register is

$$c_{load} = l(w') \cdot cpl + m' \cdot c_{av}$$

and the corresponding value of t_o results to

$$t_o = a_{reg} \cdot c_{load} + b_{reg} .$$

Note that our algorithm can also be used in the case when no placement is given, e.g. during logic synthesis. In that case, wire capacitances are assumed to have zero value.

2.3 Algorithm

Like the *FEAS*-algorithm, we basically perform an alternating sequence of calculating arrival times and retiming critical vertices.

We have already explained, that retiming one particular end vertex of a branch bundle has an influence on the delay of paths leading through other end vertices of the same bundle. We also showed, that our coupled edge timing model enables us to take this effect into account. Now, we will explain how to exploit the resulting optimization potential. For this purpose, we have to extend the basic FEAS-strategy of retiming critical vertices. To identify feasible register arrangements in a branch bundle, we have to consider all end vertices of the bundle at one glance and we have to consider data arrival times at registers positioned on edges of that bundle as well as data arrival times at registers positioned on outgoing edges of end vertices of the bundle. The details are given in the following subsections.

2.3.1 Arrival Time Calculation

Recall that our timing model from Section 2.1 assigns distinct delays both to vertices and edges. Consequently, we can associate different arrival times with a vertex and its outgoing edge. First, for each edge, we determine t_i, t_o, t_w for the actual situation. The arrival time $t_{ar}(v)$ of a vertex v is calculated from the delay values of the incoming edges $e_i = (u_i, v)$ and the arrival time of the predecessor vertices u_i of v as follows:

$$t_{ar}(v) = t_d(v) + \max \begin{cases} t_o(e_i) & w_r(e_i) > 0 \\ t_{ar}(u_i) + t_w(e_i) & w_r(e_i) = 0 \end{cases}$$

For an edge $e = (u, v)$ we define an edge arrival time t_{ear}:

$$t_{ear}(e) = t_{ar}(u) + \begin{cases} t_i & w_r(e) > 0 \\ t_w + t_d(v) + t_{tr}(v) & w_r(e) = 0 \end{cases}$$

The parameter $t_{tr}(u)$ denotes the *time to register* for a vertex u and is calculated as

$$t_{tr}(u) = \max(t_i(e_j)),\ e_j = (u, v_j)$$

using the t_i values for the case that $w_r(e_j) > 0$. In other words, for the case that there is a register on e, t_{ear} is the signal arrival time at the input of that register. Otherwise, if no register is present, t_{ear} is the latest arrival time at an assumed (not necessarily yet present) register on an outgoing edge of v. This model is motivated by the retiming procedure of the following subsection. This procedure takes into account that a previously critical end vertex v_1 of a bundle may become uncritical by retiming another end vertex v_2 of the same bundle. In this case, however, since this effect is of limited strength, we assume that an immediate successor of v_1 remains critical and needs to be retimed.

2.3.2 Retiming Critical Vertices

In contrary to the *FEAS*-algorithm which inspects each vertex separately when deciding whether or not to retime it, we consider all end vertices of a branch bundle $B = \{b_i\} = \{(u, v_i)\}$ at one glance. If $t_{ear}(b) \le t_{max}$ holds for each edge $b \in B$, then locally no timing constraint is violated.

If at least one edge violates the timing constraint, we try to find a register arrangement on the edges $b \in B$ that is reachable by backward retiming some vertices v_i. If an appropriate arrangement is found all vertices are marked that need to be retimed in order to achieve this arrangement. If more than one arrangement is found we choose one heuristically as shown in Fig. 6. If no arrangement is found, then all vertices v_i with $w_r(b_i) = 0$, $b_i = (u, v_i)$, are marked.

```
analyze_nets(tmax) {
  for each branch bundle B = [bi] = [(u,vi)]
    if (tear(b) > tmax for any b ∈ B) {
      find reachable retimings of vertices vi that satisfy
        tear(b) < tmax for each b ∈ B;
      among the candidates requiring a minimum number of vi
        to be retimed, choose the one minimizing
        (tmax – max(tear(b)) and mark all vi that must be retimed;

      if (no candidate is found)
        mark all successors vi with wr(bi)==0;
    }
}
```

Figure 6: Algorithm *analyze_nets()*

The basic philosophy of our procedure is to exploit load coupling between branches in order to extend the number of feasible retimings in a fanout system. By exploring these additional possibilities our approach may find a feasible retiming where the conventional approach fails.

Our modified *FEAS*-algorithm using the coupled edge timing model (*FEAS_CTM*) tries to find a retiming for a given cycle time t_{max}, as shown in Fig. 7.

```
FEAS_CTM(tmax) {
  for (i = 0; i < |E|, i = i+1)
    calculate_arrivaltimes();
    if (checkcycletime(tmax) == true)
      return true;
    analyze_nets(tmax);
    for each (vertex v)
      if (v is marked)
        r(v) = r(v) + 1;
  return false;
}
```

Figure 7: Algorithm *FEAS_CTM()*

If no feasible retiming exists the original *FEAS*-algorithm needs $|V|$ iterations of its inner loop to detect that. Standard FEAS tries to satisfy violated timing constraints locally by retiming a critical vertex u. *FEAS_CTM()*, however, first attempts to resolve the violation by retiming one or more successors v_i of u, while u itself eventually may be retimed during a later iteration. Therefore *FEAS_CTM()* needs

$$\sum_{u \in V} |\{(u, v_i)\}| = |E|$$

iterations to test if it can reach a feasible retiming.

The new timing model allows a more precise representation of a circuit than previous models and consequently allows to find feasible solutions that cannot be detected using simpler models. However, because of the coupling between the edges, our heuristics for selecting vertices for retiming cannot

guarantee that local decisions will always lead to the global optimum. Hence, compared to standard FEAS we loose an attractive theoretical property: our algorithm does not guarantee to find the optimum solution anymore. On the other hand, it turns out beneficial to sacrifice this theoretical property. Known approaches are only optimal within their inexact timing model. In this work, we propose to replace the accurate FEAS on an inaccurate timing model by an inaccurate FEAS_CTM on a more accurate timing model. In fact, it is possible to prove that our new approach always achieves a smaller or at least the same cycle time compared to the conventional FEAS. This is based on the following observation:

If we compare FEAS_CTM to algorithms without edge coupling, we see that FEAS_CTM attempts to remove a constraint violation using a more conservative retiming. If we consider the example shown in Figure 8, we see that without edge coupling it would not be possible to detect that the solution shown in the middle is feasible. Consequently, gate *A* would be retimed instead of gate *C*. This solution however, if needed, can still be produced also by FEAS_CTM in a later step. Hence we don't miss anything compared to the conventional FEAS. If we examine all cases we note that FEAS_CTM is always more conservative than FEAS and explores a solution space that contains the solution space of FEAS. Along these lines it is possible to prove that if FEAS or the Bellman-Ford-algorithm using the Soyata-Friedman model reach a particular cycle time FEAS_CTM at least reaches the same cycle time.

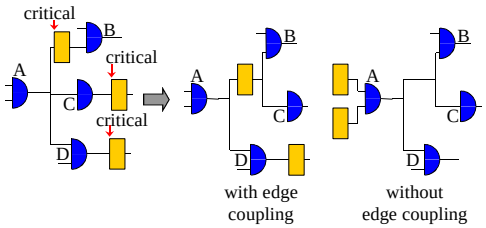

Figure 8: Retiming with and without edge coupling

2.4 Overview

The core of our approach is a timing driven simulated annealing-based standard cell placement algorithm following the philosophy of common placement tools such as [18]. Figure 9 gives an overview of the placement procedure at a particular temperature level. First, a static timing analysis is performed. For this analysis wire length estimations obtained from the actual placement are used. If timing constraints are already met we continue with the placement process immediately. Theoretically it would be possible to halt the placement process if timing constraints are already met at the beginning. However, continuing with the placement process in general makes sense because a further reduction of total wire length often can lead to a more compact solution. If timing constraints are not met at this point a retiming based optimization step is performed. Afterwards, the newly created registers are inserted into the placement using a fast placement approach. Then, wire lengths are re-estimated and the cycle time is calculated again. If constraints are met now, or at least an improvement has been achieved, the new configuration is accepted, otherwise all modifications of the netlist structure and the placement will be rejected. Afterwards, net weights are recalculated and the placer begins another iteration.

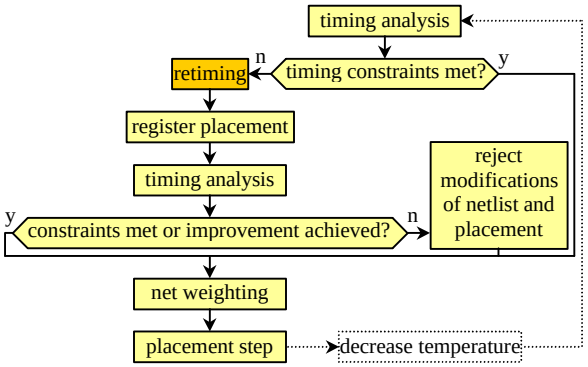

Figure 9: Placement at a particular temperature level

2.5 Aborting Retiming

In our experiments we observed that in those cases where *FEAS_CTM* has been able to identify a feasible retiming it always needed only a very small number of iterations ($<<|E|$) of its inner loop. The same observation was made by Shenoy and Rudell [3] for the original *FEAS*-algorithm. Therefore, we limited the number of trials to $|E|^{0.5}$ to save computation time.

2.6 Register Placement

In general, a simulated annealing-based placer will be able to find good positions for the newly created registers, independent from their initial position. But this process will take a lot of time if registers are inserted randomly, making it impossible to verify immediately after register insertion whether or not a cycle time improvement has been really achieved.

Furthermore, it will save a lot of work for the placer, if those registers are inserted at "reasonable" locations, especially at low temperatures, when cells aren't allowed to make large jumps.

Therefore, we use a separate register insertion step to provide the timing analyzer quickly with realistic assumptions about the wire lengths after retiming has been performed. For each new register a position is determined such that the sum of the lengths of the nets connected to this register is minimized. In many situations, the result will not be a particular vertex, but a target area of rectangular shape. In the latter case, we look for the most suitable cell gap inside this area and position the register there. This helps to keep the modifications of the original placement as small as possible. If the gap isn't large enough, neighbor cells are pushed aside first. By doing so it is always guaranteed that no cell overlapping occurs. At this point, no further work is done to reuse gaps left by deleted registers. No work is done either to balance the total row length, because these tasks are performed by the simulated annealing placer later. An example of inserting a single additional register is shown in Fig. 10.

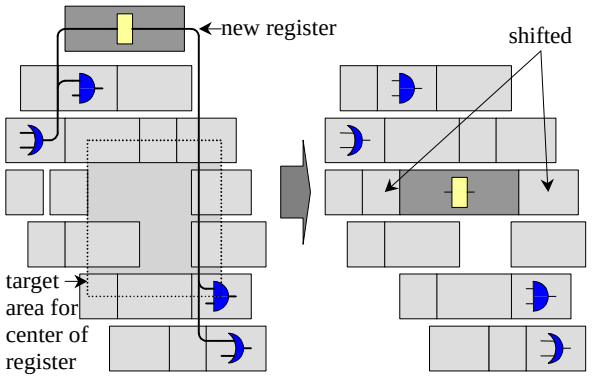

Figure 10: Single register placement example

3 Experimental Results

For the evaluation of the benefit of our new timing model and a tight integration of timing-driven placement and retiming a comparison of three different design flows is of interest:

- A conventional design flow consisting of retiming a logic netlist, followed by timing-driven placement
- A design flow as in [17] consisting of timing-driven placement, followed by retiming using the delay values calculated from the final placement. After performing a register insertion step as described Section 2.6, additionally some placement steps at very low temperatures are performed to achieve uniform row lengths again.
- A tight integration of retiming and placement as described in this paper.

For our experiments we mapped the larger circuits of the ISCAS-89 benchmark set onto a 0.18 µm standard cell library. The results are shown in Table 1. Column 2 contains the achieved cycle time for a timing-driven placement approach without any application of retiming. Then, for each of the previously described design flows using retiming, the achieved cycle time (c.t.) in nanoseconds and the final number of registers (#FF) are shown. Columns 3 and 4 contain results for pre-placement retiming, columns 5 and 6 contain results for post-placement retiming, and columns 7 and 8 show the results for the approach presented in this paper. The wire length values used for cycle time calculation have been estimated using the half perimeter bounding box method commonly used in placement tools.

circuit	none	pre-placement		post-placement		tight integration	
	c.t.	c.t.	#FF	c.t.	#FF	c.t.	#FF
S1423	12.4	10.6	113	10.3	111	9.48	111
S1488	4.39	4.15	42	3.83	21	3.25	13
S1494	4.46	3.77	39	4.01	22	3.18	12
S5387	3.95	4.12	319	3.94	164	3.90	164
S9234	10.2	7.29	447	7.24	228	6.40	451
S9234.1	10.3	8.04	425	7.70	211	7.31	438
S13207	10.0	8.42	862	9.81	669	7.39	872
S13207.1	10.5	9.68	640	9.38	641	9.35	641
S15850	15.3	11.3	960	14.5	597	8.21	1088
S15850.1	14.8	12.8	587	11.4	586	10.7	605
S35932	10.6	10.4	1728	10.6	1728	8.34	2659
S38584	17.0	17.6	3205	16.2	1452	14.6	1452
S38584.1	14.1	15.3	3413	13.4	1426	12.9	1428
S38417	15.2	12.1	2213	10.7	2193	10.4	2171

Table 1: Cycle time and register counts for FEAS_CTM

For comparison, the results obtained by using a standard FEAS algorithm are shown in Table 2.

circuit	pre-placement		Post-placement		tight integration	
	c.t.	#FF	c.t.	#FF	c.t.	#FF
S1423	10.6	113	10.7	112	10.6	114
S1488	4.53	7	4.39	6	4.30	6
S1494	4.53	7	4.46	6	4.45	6
S5387	4.37	325	3.95	179	3.85	348
S9234	7.31	268	7.08	249	6.26	462
S9234.1	7.34	269	7.66	242	7.36	553
S13207	9.31	950	10.0	669	8.30	943
S13207.1	9.79	640	9.38	641	9.18	641
S15850	12.5	962	15.3	597	12.6	3355
S15850.1	13.4	586	11.4	610	10.3	659
S35932	10.4	2826	10.5	2193	8.98	2841
S38584	19.0	3379	17.0	1452	16.2	3330
S38584.1	13.0	1428	13.4	1428	12.8	1429
S38417	12.5	2006	10.7	2193	10.2	2479

Table 2: Results for standard FEAS

The experimental results show that in most cases the use of retiming only before placement achieved the smallest performance improvement of all strategies. In a few cases cycle time was even larger after placement. If retiming was applied after placement we achieved somewhat better results, and in no case there was an increase of cycle time. However, this approach was outperformed by our new approach using tight integration, which produced equal or better results for each benchmark. Using standard FEAS instead of FEAS_CTM produced similar results but clearly achieved smaller improvements. Table 3 gives a summarizing overview of the approaches by comparing the achieved improvements in cycle time. We note that both, the coupled edge time model and the integration of retiming into placement substantially contribute to the quality of our results.

	pre-placement retiming		post-placement retiming		tight integration	
	FEAS	FEAS CTM	FEAS	FEAS CTM	FEAS	FEAS CTM
min	-11.8%	-8.5%	0%	0.25%	0.02%	1.2%
max	28.7%	28.5%	31.2%	29.6%	38.6%	46.3%
average	6.7%	11.0%	9.98%	12.2%	15.9%	23.7%

Table 3: Achieved cycle time improvements

Table 4 contains the CPU run times in seconds on a Sun Ultra Sparc 5 Workstation for a conventional timing-driven placement without retiming and for a tight integration of placement and retiming with FEAS_CTM. The results show that the increase in run time caused by integrating retiming is moderate. Despite the fact, that retiming is performed numerous times, the overall run time of our approach is still dominated by the simulated annealing-based placer core.

circuit	placem. only	tight integration	circuit	placem. only	Tight integration
S1423	100	162	S13207.1	5747	7841
S1488	115	174	S15850	7073	12134
S1494	115	211	S15850.1	7665	8233
S5387	1066	1536	S35932	29107	40948
S9234	3227	3854	S38584	22249	26253
S9234.1	3148	4234	S38584.1	21518	30393
S13207	5464	8265	S38417	37614	43566

Table 4: CPU runtimes

Finally, we observe that it is indeed of great interest to investigate accurate timing models for retiming as well as the integration of retiming into placement. In comparison with the conventional design flow (pre-placement standard FEAS, followed by timing driven placement) our new approach (tight coupling of FEAS_CTM and placement) achieved an improvement in cycle time of up to 34% and 17% on the average.

4 Conclusion

A new retiming algorithm has been developed using a highly accurate timing model. This allows us to model timing at fanout trees correctly. In general, our approach pursues the same basic retiming strategy as the conventional *FEAS*-algorithm leading to low complexity of the overall procedure. However, a more detailed local analysis at fanout systems improves the accuracy of the timing data being available and makes it possible to identify better register locations than in previous approaches. Furthermore, we present an approach for integrating retiming into the physical design process. Instead of using retiming as a pre- or a post-placement optimization method, it is applied as a cycle time improvement technique throughout the whole placement process. The experimental results show that our integrated approach exploits the optimization potential of retiming and placement significantly better than applying retiming only before or after placement.

5 References

[1] Leiserson C., Saxe B., “Optimizing Synchronous Systems”, Journal of VLSI and Computers Systems, pp. 41-67, 1983.

[2] Leiserson C., Saxe B., “Retiming Synchronous Circuitry”, pp.5-35, Algorithmica 6(1) 1991.

[3] Shenoy N., Rudell R., “Efficient Implementation of Retiming”, Proc. ICCAD-94, pp. 226-233, 1994

[4] Malik S. et al., “Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques”, IEEE Transactions on CAD, vol. 10, no. 1, pp. 74-84, 1991

[5] Lockyear B., Ebeling C., “Optimal retiming of level-clocked circuits using symmetric clock schedules”, IEEE Transactions on CAD, vol. 13, no. 9, pp. 1097-1109 1994.

[6] Ishii A., Leiserson C., Papaefthymiou M., “Optimizing two-phase, level-clocked circuitry”, Advanced Research in VLSI and Parallel Systems, Proc. of the 1992 Brown/MIT Conference, pp. 246-264, 1992

[7] Papaefthymiou M., “Asymptotically Efficient Retiming Under Setup and Hold Constraints”, Proc. ICCAD-98 pp.288-295, 1998

[8] Sundararajan V., Sapatnekar S., Parhi K., “MARSH: Min-Area Retiming with Setup and Hold Constraints”, Proc. ICCAD-99, pp. 2-13, 1999

[9] Eckl K., Madre J., Zepter P., Legl C., “A Practical Approach to Multiple-Class Retiming”, Proc. DAC-99, pp. 237-242, 1999

[10] El-Maleh A., Marchok T., Rajski J., Maly W., “Behavior and Testability Preservation Under the Retiming Transformation”, IEEE Transactions on CAD, vol. 16., no. 5, pp. 528-542, 1997

[11] Stoffel D., Kunz W., "Record & Play: A Structural Fixed Point Iteration for Sequential Circuit Verification", Proc. ICCAD-97, pp. 394-399, 1997.

[12] van Eijk C., “Sequential Equivalence Checking without State Space Traversal, Proc. DATE-98 pp. 618-623, 1998

[13] Soyata T., Friedman E., “Retiming with Non-Zero Clock Skew, Variable Register, and Interconnect Delay”, Proc. ICCAD-94, pp. 234-241, 1994

[14] Soyata T., Friedman E., Mulligan J., “Incorporating Interconnect, Register, and Clock Distribution Delays into the Retiming Process”, IEEE Transactions on CAD, vol. 16, no. 1, pp.105-120, 1997

[15] Lalgudi K., Papaefthymiou M., “DELAY: An Efficient Tool for Retiming with Realistic Delay Modeling”, Proc. DAC-95, pp. 304-309, 1995

[16] Cong J., Lim S.K., “Physical Planning with Retiming”, Proc. ICCAD-2000, pp. 2-7, 2000

[17] Tien T. et al., “Integrating Logic Retiming and Register Placement”, Proc. ICCAD-98, pp. 136-139, 1998

[18] Sechen C., “VLSI Placement and Global Routing using Simulated Annealing”, Kluwer Academic Publishers, 1988