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 \( w_r(e) = w(e) + r(v) \cdot r(u) \).

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:

- \( t_u \): 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](image)

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 \( w_i \) of an edge \( b_i \in B \) has an influence on the parameters \( t_u, t_i, t_o \) of other edges \( b_j \in B, j \neq i \). Therefore, we assign to each edge three tables. Each table contains different values for \( t_u, t_i, t_o \) 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 \( t_u \) and \( t_i \) for every branch.
For branches $i$, we can distinguish $2^i$ different cases for the calculation of the load that $g(u)$ has to drive. The $t_e$- 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 $t_e$- and $t_i$-value, respectively. Since $t_e$ 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 $t_e$- 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 $t_e$ 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 $n = 2$](image)

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

<table>
<thead>
<tr>
<th>at branch $b_0$</th>
<th>at branch $b_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>arr.</td>
<td>$t_e$</td>
</tr>
<tr>
<td>1</td>
<td>$t_{e0.0}$</td>
</tr>
<tr>
<td>3</td>
<td>$t_{e0.1}$</td>
</tr>
</tbody>
</table>

For each branch we have a $t_e$- and $t_i$- 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 $t_e$. 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](image)

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

### 2.1.2 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 $t_e$, respectively, of a particular branch $b_i \in B$ are the same for all configurations containing the same number of branches $b_i$ with $w_i(b_i) > 0, b_j \in B, i \neq j$.

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

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 $t_e, t_i, t_o$, 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 $t_e, t_i, t_o$, 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 $t_e$ values in a branch bundle $B = \{b_i \in \{(u, v)\}, 1 \leq i \leq n\}$, the length $l(w)$ of net $w$ driven by $g(u)$ is required. To estimate $l(w)$ we determine a (minimum size) rectangle $R_o$ 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 \(R_1\) which touches \(R_0\) and \(R_1\). The center of \(R\), 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 \(R_1\). 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](image)

For a particular \(r_t\) and \(l_i\), respectively we obtain

\[
\begin{align*}
  t_u &= \alpha \left( l(w) \cdot cpl + \sum_{\text{ports connected to } u} c_{\text{in}} \right) \\
  t_i &= \alpha \left( l(w) \cdot cpl + \sum_{\text{ports connected to } u} c_{\text{in}} \right) + t_{\text{setup}}
\end{align*}
\]

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

For the calculation of \(t_u\) and \(l_i\) for 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 \(R_1\) touching \(R_0\) and \(R_1\) and use the center of \(R_1\) 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 \(R_1\). For a particular \(t_u\) we obtain

\[
  t_u = \alpha_{\text{reg}} \left( l(w') \cdot cpl + \sum_{\text{ports connected to } w'} c_{\text{in}} \right) + \beta_{\text{reg}}
\]

where \(\alpha_{\text{reg}}\) denotes the driving force and \(\beta_{\text{reg}}\) 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 \(B = \{b_i\} = \{(u, v_i)\}, 1 \leq i \leq n\) 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 \(t_r\) and \(l_r\) 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 \(b_i\) with \(w_i(b_i) = 0\), \(0 \leq m \leq n\), we estimate \(l(w)\) as follows:

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

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

\[
c_{\text{load}} = l(w) \cdot cpl + \begin{cases} 
  m \cdot c_{\text{in}} & m = n \\
  m \cdot c_{\text{in}} + c_{\text{reg}} & m < n
\end{cases}
\]

where \(c_{\text{in}}\) denotes the average input capacitance of all input ports driven by \(g(u)\) when no registers occur on any branch, i.e., \(w_i(b_i) = 0\) 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 \(c_{\text{reg}}\) of the register. (Remember that we assume register sharing as described in Section 2.1.2.1.)

For a particular \(t_u\) and \(l_i\), we obtain:

\[
  t_u = \alpha \cdot c_{\text{load}} \quad \text{and} \quad t_i = \alpha \cdot c_{\text{load}} + t_{\text{setup}}
\]

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

\[
l(w') = \frac{\text{perimeter}(R)}{2} \sqrt{\frac{m'^2 + 1}{n + 1}}
\]

The load seen by the register is

\[
c_{\text{load}} = l(w') \cdot cpl + m' c_{\text{in}}
\]

and the corresponding value of \(t_u\), results to

\[
t_u = \alpha_{\text{reg}} \cdot c_{\text{load}} + \beta_{\text{reg}}.
\]

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 \( t_{\text{arr}} \), \( t_{\text{ext}} \), \( t_{\text{rev}} \) for the actual situation. The arrival time \( t_{\text{arr}}(v) \) of a vertex \( v \) is calculated from the delay values of the incoming edges \( e_i = (u, v) \) and the arrival time of the predecessor vertices \( u_i \) of \( v \) as follows:

\[
\begin{align*}
  t_{\text{arr}}(v) = & \ t_{\text{arr}}(v) + \max \left\{ t_{\text{arr}}(e_i), w(e_i) = 0 \right\} \\
& \text{for an edge } e = (u, v) \text{ we define an arrival edge } t_{\text{arr}}.
\end{align*}
\]

The parameter \( t_{\text{arr}}(u) \) denotes the time to register for a vertex \( u \) and is calculated as

\[
\begin{align*}
  t_{\text{arr}}(u) = & \ \max \{ t_{\text{arr}}(e_i), e_i = (u, v) \}
\end{align*}
\]

using the \( t_i \) values for the case that \( w(e_i) > 0 \). In other words, for the case that there is a register on \( e \), \( t_{\text{arr}} \) is the signal arrival time at the input of that register. Otherwise, if no register is present, \( t_{\text{arr}} \) 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 \( t_{\text{arr}}(b) \leq t_{\text{max}} \) 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 \( w(b_i) = 0 \), \( b_i = (u, v_i) \), are marked.

\[
\begin{align*}
\text{analyze_nets}(t_{\text{max}}) \{ \\
\quad & \text{for each branch bundle } B = \{b\} = \{(u, v)\} \\
\quad & \text{if } (t_{\text{arr}}(b) > t_{\text{max}} \text{ for any } b \in B) \{ \\
\quad & \quad \text{find reachable retimings of vertices } v_i \text{ that satisfy } \\
\quad & \quad \text{among the candidates requiring a minimum number of } v_i \text{ to be retimed, choose the one minimizing } \\
\quad & \quad (t_{\text{max}} - \max(t_{\text{arr}}(b))) \text{ and mark all } v_i \text{ that must be retimed;}
\quad & \quad \text{if } (\text{no candidate is found}) \\
\quad & \quad \text{mark all successors } v_i \text{ with } w(b_i) = 0;
\quad \}
\quad \}
\end{align*}
\]

Figure 6: Algorithm \text{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 \((\text{FEAS}_{\text{CTM}})\) tries to find a retiming for a given cycle time \( t_{\text{max}} \) as shown in Fig. 7.

\[
\begin{align*}
\text{FEAS}_{\text{CTM}}(t_{\text{max}}) \{ \\
\quad & \text{for } (i = 0; i < |E|, i = i+1) \\
\quad & \text{calculate_arrivaltimes();} \\
\quad & \text{if (checkcycletime}(t_{\text{max}}) == \text{true}) \\
\quad & \quad \text{return true;}
\quad & \text{analyze_nets}(t_{\text{max}}); \\
\quad & \text{for each (vertex } v \) \\
\quad & \text{if } ((v \text{ is marked}) \\
\quad & \quad r(v) = r(v) + 1;
\quad & \text{return false;}
\quad \}
\quad \}
\end{align*}
\]

Figure 7: Algorithm \text{FEAS}_{\text{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_CTMs, 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_CTMs needs

\[
\sum_{v_i \in V} f(u, v_i) = |E|
\]

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.

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](image)

### 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](image)

### 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: Register insertion](image)
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 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.

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.

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.

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.

<table>
<thead>
<tr>
<th>circuit</th>
<th>placem. only</th>
<th>tight integration</th>
<th>circuit</th>
<th>placem. only</th>
<th>Tight integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1423</td>
<td>100</td>
<td>162</td>
<td>S13207</td>
<td>1574</td>
<td>7841</td>
</tr>
<tr>
<td>S1488</td>
<td>115</td>
<td>174</td>
<td>S15850</td>
<td>7073</td>
<td>12134</td>
</tr>
<tr>
<td>S1494</td>
<td>115</td>
<td>211</td>
<td>S15850.1</td>
<td>7665</td>
<td>8233</td>
</tr>
<tr>
<td>S5387</td>
<td>1066</td>
<td>1536</td>
<td>S35932</td>
<td>29107</td>
<td>40948</td>
</tr>
<tr>
<td>S9234</td>
<td>3227</td>
<td>3854</td>
<td>S35884</td>
<td>22249</td>
<td>26253</td>
</tr>
<tr>
<td>S9234.1</td>
<td>3148</td>
<td>4234</td>
<td>S35884.1</td>
<td>21518</td>
<td>30393</td>
</tr>
<tr>
<td>S13207</td>
<td>5464</td>
<td>8265</td>
<td>S38417</td>
<td>37614</td>
<td>43566</td>
</tr>
</tbody>
</table>

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