

Accelerating Retiming Under the Coupled-Edge Timing Model

Ingmar Neumann Kolja Sulimma Wolfgang Kunz
University of Kaiserslautern/Germany
Department of Electrical Engineering and Information Technology
Electronic Design Automation Group
email: neumann@eit.uni-kl.de

Abstract

Retiming has been shown to be a powerful technique for improving the performance of synchronous circuits. However, even though retiming algorithms of polynomial time complexity have been developed the runtimes still may become prohibitively long for large circuits. For the original FEAS algorithm proposed by Leiserson and Saxe, acceleration techniques have been developed solving this problem in practice. However, FEAS uses a simple circuit model being fairly inaccurate for gate level net lists mapped onto actual technologies. Recently a retiming algorithm FEAS_CTM based on a new timing model tackling this problem has been proposed. In this paper we present a technique for speeding up execution time of FEAS_CTM. This technique is also suitable for a variety of published algorithms based on the circuit model proposed by Soyata and Friedman. In this work the approach has been integrated into FEAS_CTM and its benefit has been proven by experimental results.

1. Introduction

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

The original FEAS-algorithm [2] 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 circuit model assuming gate delays to be load independent.

In [11]-[13], more sophisticated timing models are used. For each edge in the retiming graph multiple delay values are calculated covering the two cases that this edge can contain none or at least one register. Branches of multi sink nets are considered independently from each other.

The problem of modeling fanout trees accurately has been addressed in [14]. The problem is illustrated in Figure 1.

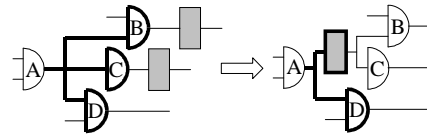


Figure 1: Retiming end gates of a fanout tree

In real circuits, retiming gate *B* and gate *C* will change the load (drawn in bold lines) seen by gate *A* and therefore also changes the delay of gate *A*. This however, does not only affect the arrival time at gate *B* and *C* but also at gate *D*.

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.

To be able to take dependencies between branches of a fanout system into account, delay tables are assigned to each edge in the retiming graph. Each table entry represents a delay value being valid for one particular register arrangement in the fanout system. Prior to each arrival time calculation step the delay values being valid for the actual register arrangement are selected from the delay tables. This timing model allows a very realistic modeling of circuits in the retiming graph.

For each of the timing models described above powerful retiming algorithms of polynomial time complexity have been developed. However, for large circuits with thousands of registers runtimes may still be unacceptably large impeding the use of these algorithms in practice.

For the original FEAS algorithm this problem has been addressed in [3]. The authors presented a very efficient

technique for detecting unreachable target cycle times. Experiments showed that integrating this technique into FEAS can dramatically speed up execution time and generally leads to near linear time complexity when retiming typical VLSI circuits. Unfortunately, there is no straightforward way to integrate this technique into retiming algorithms using more complex timing models. The reason for this is that this technique is based on the assumption that the sum of the delays of paths forming a loop in the retiming graph does not depend on the actual positions of the register. This assumption however, does not hold for more accurate timing models as described in [11] and [14].

To make retiming more practical, we propose an acceleration technique for the FEAS_CTM retiming algorithm presented in [14]. Our experimental results show a dramatic acceleration for most of the test cases and demonstrate that accurate timing model based retiming is possible even for very large circuits.

The remainder of this paper starts with a brief description of the FEAS_CTM timing model and retiming algorithm. For a more detailed description we refer the reader to [14]. Afterwards we describe our new acceleration technique in detail. The paper closes with experimental results and a short conclusion.

2. The FEAS_CTM algorithm

2.1. Circuit Model

A circuit is mapped onto a weighted directed retiming 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. A net with n sinks, $n \geq 1$, is modeled as a bundle of edges ("branches") $B = \{b_i\} = \{(u, v_i) \in E, 1 \leq i \leq n\}$. 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 $w_r(e) = w(e) + r(v) - r(u)$.

2.1.1. Single Sink Net Edges. A net with one single sink is modeled by an edge bundle containing one single edge. This edge e is assigned three delay values modeling two different cases as shown in Figure 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: Delay for a two terminal net edge

2.1.2. Multi Sink Net Edges. For an edge $b_i \in B$ modeling a branch of a multi sink fanout system the delay values described above depend on the weights of the other edges belonging to B . To model these dependencies three delay tables are assigned to b . Each table contains different values for t_i , t_o or t_w , respectively, according to distinguishable register arrangements in the fanout system.

For the calculation of the entries of the t_i - and the t_w -table of b_i for each other edge $b_j \in B$, $i \neq j$, it is distinguished whether $w_r(b_j) = 0$ or $w_r(b_j) \neq 0$. For B consisting of n branches there are 2^n different register arrangements to be distinguished. Because t_w is only defined for $w(b_i) = 0$ and t_i is only defined for $w(b_i) > 0$ this leads to 2^{n-1} entries in the t_i - and the t_w -table.

The calculation of the entries of the t_o -table of an edge b_i is based on the assumption that a bundle B is realized with a minimal number of registers when the net list is created from the retiming graph. Consequently, for building the t_o table of an edge $b_i \in B$ for each edge $b_j \in B$, $i \neq j$, has to be distinguished, 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 modeled by two vertices v_i and v_j with $w_r(b_i) = w_r(b_j)$ are driven by the same register. Because t_o is only defined for $w(b_i) > 0$ there are 2^{n-1} different cases to be distinguished.

2.2. Algorithm

Like FEAS, FEAS_CTM performs an alternating sequence of calculating data arrival times and retiming critical vertices. These steps are repeated until either a solution is found or it is detected that no solution can be reached.

2.2.1. Arrival Time Calculation. During a preliminary step for each edge the delay value(s) being valid for the actual register arrangements are selected from the corresponding delay tables in accordance to the weight of the edge.

The arrival time $t_{ar}(v)$ of a vertex v is calculated recursively 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 $e = (u, v)$ an edge arrival time t_{ear} is defined:

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

The parameter $t_r(u)$ denotes the *time to register* for a vertex u and is calculated as

$$t_r(u) = \max(t_i(e_j)), e_j = (u, v_j)$$

using the t_i values for the case that $w_r(e_j) > 0$. For the case that there is a register on e , t_{ear} denotes the data arrival time at the input of that register. Otherwise, if no register is present, t_{ear} denotes the latest arrival time at an assumed (not necessarily yet present) register on an outgoing edge of v .

2.2.2. Retiming Critical Vertices. At the core of FEAS_CTM, function *analyze_nets*, shown in Figure 3, marks critical vertices for retimings. In this step all end vertices of an edge bundle $B = \{b_i\} = \{(u, v_i)\}$ are considered at one glance.

```

analyze_nets( $t_{max}$ ) {
  for each branch bundle  $B = [b_i] = [(u, v_i)]$ 
    if ( $t_{ear}(b) > t_{max}$  for any  $b \in B$ ) {
      find reachable retimings of vertices  $v_i$  that satisfies
         $t_{ear}(b) < t_{max}$  for each  $b \in B$ ;
      among the candidates requiring a minimum number
        of  $v_i$  to be retimed, choose the one minimizing
        ( $t_{max} - \max(t_{ear}(b))$ ) and mark all  $v_i$  that must
        be retimed;
      if (no candidate is found)
        mark all successors  $v_i$  with  $w_r(b_i) == 0$ ;
    }
}

```

Figure 3: Function for retiming critical vertices

Figure 4 shows the main loop of the retiming algorithm FEAS_CTM.

```

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

```

Figure 4: Retiming algorithm FEAS_CTM

If no feasible retiming can be found *FEAS_CTM* needs $|E|$ iterations of its inner loop to detect this.

3. Accelerating Retiming

Retiming based cycle time minimization in general performs a binary search for the minimum reachable cycle

time $t_{optimal}$. During this process the retiming algorithm is called several times trying to find retimed circuits for various target cycle times t_{target} . It has been observed that in the cases where FEAS_CTM has been able to find a feasible retiming, it only required a very small number ($\ll |E|$) of iterations of its inner loop. The same observation has been made for the original FEAS Algorithm in [3]. Consequently speeding up retiming has to concentrate on those cases where no feasible solution could be found. In those cases FEAS_CTM requires $|E|$ iterations of its inner loop.

In the following we propose an efficient and effective technique for testing whether a particular cycle time t_{target} can *not* be reached. This check is performed during each iteration of FEAS_CTM. As soon as this test succeeds the first time, we can abort the retiming process for t_{target} .

At first we give an informal outline. Afterwards our approach is explained in more detail.

A common strategy for computing bounds in timing analysis is to consider circuit loops[15]. Our acceleration technique is based on the following fact: If there is at least one loop in the retiming graph we can not determine a feasible retiming for, there will be no feasible retiming for the whole circuit. Consequently, we can abort the retiming process, if any such loop is found. This has already been observed in [3] and delivers the basis for our acceleration technique. During each iteration of the inner loop of FEAS_CTM we determine a set of loops we will analyze in detail.

In [3], this analysis is very simple because the sum of the delays of paths forming a loop in does not depend on the actual positions of the register. For the more sophisticated timing models considered here, however, this assumption is wrong so that a more complex analysis is required.

For each selected loop we perform three tests. Test 1 determines a lower bound t_{lb} for the cycle time that may be reached in the loop we are considering. If $t_{lb} > t_{target}$ holds then we can abort the whole retiming process. Test 2 determines an upper bound t_{ub} for the minimum cycle time that may be reached in this loop. If $t_{ub} < t_{target}$ holds then there exists a timing feasible register placement for this loop and we can proceed with the next loop. If we find that $t_{lb} \leq t_{target} \leq t_{ub}$ holds we perform test 3. This test tries to determine a register placement for the loop using a dedicated register placement technique. If we can not determine a timing feasible register placement for the loop we can abort the overall retiming process. If no loop violating the timing constraint is found we proceed with the retiming process. Figure 5 shows retiming algorithm FEAS_CTM with our acceleration technique integrated.

```

FEAS_CTM( $t_{target}$ ) {
  for ( $i = 0; i < |E|, i = i+1$ )
    calculate_arrivaltimes();
  /* test whether  $t_{target}$  is not reachable */
  determine set of critical loops C;
  for (each loop  $L \in C$ )
    if ( $t_{lb}(L) < t_{target}$ )
      return false;
    if ( $t_{target} \leq t_{ub}(L)$ )
      if (no feasible register placement for  $L$  found)
        return false;
  /* retime critical vertices */
  if (checkcycletime( $t_{target}$ ) == true)
    return true;
  analyze_nets( $t_{target}$ );
  for each (vertex  $v$ )
    if ( $v$  is marked)
       $r(v) = r(v) + 1$ ;
  return false;
}

```

Figure 5: Accelerated FEAS_CTM

3.1. Building the loop set

As an addition to the FEAS_CTM circuit model each graph vertex v is supplemented with an additional pointer *crit_edge*. During the arrival time calculation step this pointer is set to the incoming edge of v that is part of the longest path leading through v . Let E^* denote the set of those edges $e \in E$ a *crit_edge* pointer points to. Then E^* and the vertices $v \in V$ form a subgraph $G^* = (V, E^*)$ of G containing only vertices with exactly one predecessor. The set of loops in G^* form the set of loops we will investigate further. Because there are no reconverging paths in G^* , all loops in G^* can be determined with a simple graph traversal in linear time.

3.2. Determining a lower bound

In this section we describe how to determine the bound t_{lb} for a loop $L = \{(v_i, v_{i+1}) \in E^*, 1 \leq i \leq n, v_1 = v_n\}$.

As already explained, the FEAS_CTM circuit model assigns tables of delay values to each edge in the retiming graph to reflect the fact that retiming an end vertex of a fanout tree has an influence on the delay of a paths leading through neighbor branches. Consequently the sum of all path delays in L , denoted by $t_{loop}(L)$ in the following, will also depend on the weights of edges that are not part of the loop. However during the unreachability detection step we consider each loop isolated. To be able to determine bounds for the cycle time reachable by retiming we analyze the delay tables and determine for each edge $e \in G^*$ the minimum values for t_i , t_o , t_w . The minimum value found for t_w is denoted by t_{wmin} in the following. The sum of the minimum t_i and the minimum t_o is denoted by t_{iomin} .

In other words, t_{wmin} denotes the minimum delay that edge e contributes to $t_{loop}(L)$ for the case that e does not carry a register. Similarly, t_{iomin} denotes the minimum delay that e contributes to $t_{loop}(L)$ for the case that e carries at least one register.

Further we determine the total sum of registers in L , denoted by r , by adding the weights of the edges forming L . Note that retiming will not change r for any loop. However, the number of edges in L with at least one register may change, because an edge can carry more than one register. The maximum number of edges carrying at least one register for any particular register placement in L results to $r^* = \min(|L|, r)$.

Consequently, any possible register arrangement in L will consist of n edges with $w > 0$ and $(|L| - n)$ edges with $w = 0$, $1 \leq n \leq r^*$.

Selecting n t_{iomin} - values and $(|L| - n) - t_{wmin}$ values, $1 \leq n \leq r^*$, in a way that the resulting sum is minimized and adding them to the sum of the delays of the vertices in L results in $t_{loop}^n(L)$. $t_{loop}^n(L)$ delivers a lower bound for $t_{loop}(L)$ for the case n edges in L carry at least one register.

Minimizing $t_{loop}^n(L)/n$, $1 \leq n \leq r^*$ gives a lower bound t_{lb} for the cycle time that may be reached by retiming if we consider $|L|$ isolated.

If we sort the t_{iomin} and the t_{wmin} values of the edges of L in two separated lists in rising order t_{lb} can be determined very efficiently in $|L|$ steps.

If $t_{lb} > t_{target}$ holds for L we can abort the retiming process because it will not be possible to find a retimed circuit for t_{target} .

3.3. Determining an upper bound

If we are combining n t_{iomin} values and $(|L|-n) t_{wmin}$ values, $n = 1$, in a way that $t_{loop}^n(L)/n$ is maximized, analogously to section 3.2 we derive an upper bound t_{ub} for the cycle time that can be reached if we consider L isolated. In other words, the upper bound calculation considers the worst case situation, where all registers are located on a single edge. If $t_{ub} < t_{target}$ holds then there exists a register placement in L reachable by retiming which satisfies the timing constraint. If this is the case we can skip the further investigation of L and may proceed with the next loop.

3.4. Retiming a loop

The analysis described above determines bounds for the cycle time that can be reached in L . These tests can be performed very efficiently and in many cases will already detect whether a cycle time t_{target} may be reachable or not. In practice, this will especially be the case when t_{target} is not too close to time $t_{optimal}$. However, if we find that

$t_{ub} < t_{target} < t_{lb}$ holds then we investigate L further. In this case we look for a register placement in L satisfying the timing constraint in L . If no such register placement is found we can abort the whole retiming process because there will be no retiming satisfying the timing constraint for the whole circuit.

For general graphs determining a register placement would require the use of a general retiming algorithm. However since we know that L represents a loop and we are considering L isolated we can use a dedicated technique.

A straightforward approach would be to place the first register on an arbitrarily chosen edge. For the remaining registers, positions would be determined by stepping forward through L and placing each register as far away as possible from its predecessor register. If the register placement determined will not be timing feasible, as shown in Figure 6 on the left side, this does not yet necessarily mean that no feasible solution exists. A feasible solution might be found repeating the process placing the initial register on a different edge, as shown in Figure 6 on the right side. The three numbers assigned to each edge denote the t_{imin} , t_{omin} , t_{vmin} -delays of the edge.

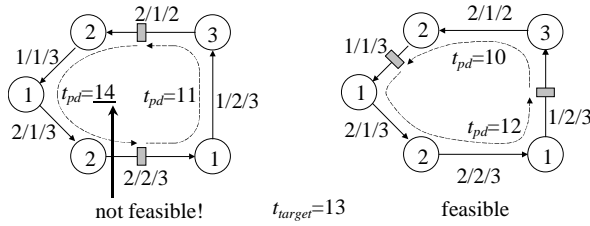


Figure 6: Retiming example

If m denotes the minimum distance measured in number of edges between two registers in the register placement found in the first trial, we have to perform the register placement step another m times. Because m may reach at most $|L|/r$ this leads to a time complexity of $O(|L|^2/r)$ for this approach.

This simple approach would achieve satisfactory run times only for large r ($r \approx L$). For small r we have a time complexity of $O(|L|^2)$. However, we will see that we can do better.

The first register is positioned arbitrarily as explained above. As a preliminary step we determine for each edge the data arrival time at the input of an assumed register on this edge. These data arrival times are stored in an auxiliary data structure in sorted order. This can be done in $|L| \cdot \ln(|L|)$ steps.

Using this data structure we do not have to step through the loop to determine the positions of the $(r-1)$ subsequent register as described above but we can

determine each position in $\ln|L|$ steps. This leads to a time complexity of $O(|L| \cdot \ln(|L|))$ for analyzing L .

It should be noted that our analysis may not always succeed in detecting an unreachable cycle time. The reason is that we consider each loop independently. Our analysis might find a timing feasible register placement for each loop. However, these register placements need not necessarily be compatible with each other, so the possibility that no timing feasible register placement for the whole circuit exists. However, our experimental results showed, that despite of this deficiency the proposed approach can still considerably speed up the retiming process.

The acceleration technique described here can also be used in combination with the timing model used in [11]-[13]. Since, in this model, we have no tables containing multiple delay values, the delay values of each edge itself are used as t_{wmin} and t_{iommin} .

4. Experimental Results

For the evaluation of the benefit of our acceleration technique we mapped the circuits from the ISCAS-89 benchmark set onto a 0.18 μm standard cell library. For each benchmark we performed numerous optimization runs using wire length data that has been derived from different placements. For each trial that did not succeed (e.g., no feasible retiming has been found) we reported the following:

- the target cycle time t_{target} being used
- the cycle time $t_{optimal}$ that finally has been reached using a binary search based optimization approach
- whether or not our approach detected that t_{target} could not be reached; if this was the case we additionally reported during which iteration of the inner loop of FEAS_CTM this happened.

In our experiments we observed, that in those cases our technique detected that a particular cycle time was not reachable only a very small number of iterations (for the benchmarks examined in no case more than 5) had been necessary. This enables a considerable reduction of the cpu runtimes in those cases. The maximum number of iterations of the inner loop of FEAS_CTM has been limited to $|E|^{0.5}$ as proposed in [14].

We already mentioned that our analysis may not always succeed in detecting an unreachable cycle time. However this happened only in those cases t_{target} has been chosen very close to $t_{optimal}$. In the majority of the test cases our approach succeeded. The results of our experiments are shown in Table 1. For each benchmark it contains two cycle times t_a and t_b , expressed in percentage of $t_{optimal}$. These two cycle times are defined as follows. For all experiments performed with $t_{target} < t_a$ our approach succeeded in detecting unreachable cycle times. For all experiments performed with $t_{optimal} > t_{target} > t_b$ the

approach failed. For $t_a < t_{target} < t_b$ both happened. Column 4 contains the average cpu time saving of a full binary search based timing optimization run for each benchmark.

circuit	t_a	t_b	cpu time saving(%)
S38417	0.87	0.88	38
S35932	0.94	0.99	58
S38584.1	0.75	0.82	21
S38584	0.76	0.86	24
S1238	0.77	0.87	33
S1423	0.88	0.94	34
S1488	0.76	0.88	22
S1494	0.75	0.87	23
S208.1	0.81	0.86	50
S27	0.75	0.88	30
S298	0.76	0.82	18
S344	0.95	0.96	50
S349	0.94	0.96	55
S382	0.77	0.79	14
S386	0.89	0.93	34
S400	0.82	0.84	16
S420.1	0.82	0.85	20
S444	0.75	0.87	15
S510	0.84	0.89	31
S526	0.76	0.80	15
S526n	0.69	0.87	14
S641	0.88	0.94	19
S713	0.94	0.97	36
S820	0.75	0.88	24
S832	0.77	0.87	25
S838.1	0.88	0.92	22
S953	0.91	0.94	35
S5378	0.77	0.82	30
S15850.1	0.76	0.85	31
S15850	0.90	0.92	22
S13207.1	0.79	0.85	29
S13207	0.94	0.96	57
S9234.1	0.76	0.80	26
S9234	0.88	0.90	41

Table 1: Experimental results

On average our approach has been able to detect all unreachable target cycle times that are 86% of the cycle time that finally has been reached.

From a theoretical point of view, it is important to mention that if our method identifies an unreachable target cycle time then this cycle time is indeed unreachable by retiming with FEAS_CTM. This is an improvement over the purely heuristic abortion criterion (aborting after $|E|^{0.5}$ iterations) of the original approach.

5. Conclusion

In this paper we have presented an acceleration technique for the FEAS_CTM retiming algorithm. This

technique is based on detecting that a particular target cycle time may be unreachable during a very early phase in the retiming process. Experiments showed that in those cases the technique detected that a target cycle time will not be reachable, the number of iterations necessary to detect this has been dramatically reduced from the number of edges in the retiming graph to a number which in no case was larger than 5. On average this has been the case for all target cycle times up to 86% of the optimal cycle time.

6. 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] Soyata T., Friedman E., "Retiming with Non-Zero Clock Skew, Variable Register, and Interconnect Delay", Proc. ICCAD-94, pp. 234-241, 1994
- [12] 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
- [13] Lalgudi K., Papaefthymiou M., "DELAY: An Efficient Tool for Retiming with Realistic Delay Modeling", Proc. DAC-95, pp. 304-309, 1995
- [14] Neumann I., Kunz W. "Placement Driven Retiming with a Coupled Edge Timing Model", Proc. ICCAD-2001, pp95-102, 2001
- [15] Papaefthymiou M., "Understanding retiming through maximum average-delay cycles", Mathematical Systems Theory, 27, pp. 65-84, 1994