Concurrent Flip-Flop and Repeater Insertion for High Performance Integrated Circuits

Pasquale Cocchini
pasquale.cocchini@intel.com, Intel Labs, CAD Research

Abstract
For many years, CMOS process scaling has allowed a steady increase in the operating frequency and integration density of integrated circuits. Only recently, however, have we reached a point where it takes several clock cycles for global signals to traverse a complex digital system such as a modern microprocessor. Thus, interconnect latency must be taken into account in current and future design tools at the architectural as well as synthesis level. To this purpose, this work proposes a new latency-aware technique for the performance-driven concurrent insertion of flip-flops and repeaters in VLSI circuits. Overwhelming evidence showing an exponential increase in the number of pipelined interconnects with process scaling, for high-performance microprocessors as well as high-end ASICs, is also presented. This increase indicates a radical change in current design methodologies to cope with this new emerging problem.

1 Introduction

Repeater insertion is a technique extensively used to reduce the delay of interconnects and improve their noise characteristic particularly when signals are distributed over long distances on a chip. An elegant dynamic programming algorithm was proposed in [1] to determine optimal repeater assignments of the candidate locations of a given interconnect topology. Several other works based on the same technique, [3-7] to cite a few, have also been proposed incorporating other optimization steps such as noise or area minimization, wire sizing, etc. All of these works, however, only consider the case where a signal is required to arrive at its destination within one single clock cycle. On the other hand, in complex digital systems with relatively large die area operating at very high frequency, as in the case of modern high-performance microprocessors such as the Itanium® processor [9], many global signals traveling across the chip need several clock cycles to reach their destinations, thus requiring the adoption of *pipelined* interconnects, i.e. latent wiring structures in which normal repeaters are interleaved with sequential elements such as latches and flip-flops. Current scaling trends indicate that this phenomenon will be accentuated in future process generations. As can be seen in Figure 1, the frequency of high-performance microprocessors approximately doubles every process generation, in part due to shorter gate pipelines [10].

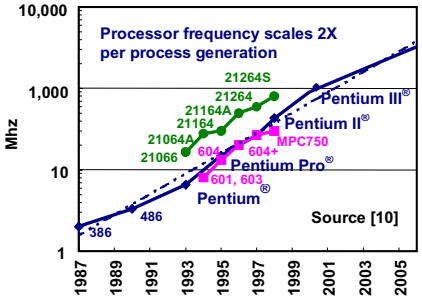

Figure 1. Frequency Trend.

Moreover, as showed in Figure 2, the die size also tends to increase by about 25% per generation, taking advantage of increased complexity and level of integration. As a result, the numbers of clock cycles needed to cross the die is bound to increase. In this situation, at least two new challenges are faced by micro-architects and circuit designers: a) the accurate prediction of the minimum latency that can be achieved between the blocks of a design, given the available routing resources of a CMOS process, and b) the performance-driven insertion of repeaters and flip-flops in a large number of pipelined nets where interconnect and functional latency constraints are specified by the micro-architects.

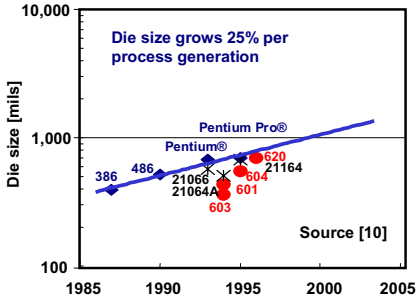

Figure 2. Die Size Trend.

To this purpose, in this paper we propose a new methodology for the performance-driven concurrent flip-flop and repeater insertion in latency constrained and unconstrained VLSI interconnects. In the case of unconstrained latency the problem is solved optimally with the goal of minimizing the overall interconnect latency, i.e. the latency at the most latent receiver. In the constrained case the insertion problem is solved specifying the target latency for each driver-receiver pair of a net.
This technique effectively extends the work of [1] to handle the broader case of pipelined interconnects by simultaneously inserting normal repeaters and flip-flop gates, also referred to in this paper as *clocked* repeaters, maintaining the optimality of the original algorithm. Moreover, flip-flops are inserted while taking into account the distribution of the skew in a clock distribution network modeled as a grid of independent clock domains. To our knowledge, this is the first work that addresses in details the problem of repeater insertion while also considering interconnection latency. For simplicity, interconnects are modeled in the paper using Elmore [2] delay. Nevertheless, our technique can also be used with other hierarchical delay models such as those based on moment matching as in [5][6].

2 Definitions

2.1 Routing Topology

We model the topology of an interconnect as a tree $\Theta = (N, B)$ composed of a set N of nodes n_i with branching degree at most two, and a set B of directed branches $b_{u,v}$ connecting node pairs (u, v), with $u < v$. The root and the leaves of the tree host the interconnect driver and receivers, respectively, whereas other intermediate nodes are the candidate locations for the insertion of repeaters. Each node n_i can also be thought of as the root of a sub-tree $\theta_i \subseteq \Theta$ composed of all nodes and branches of Θ that can be reached from n_i, with $\theta_i \equiv \Theta$ at the root of Θ. At a branch point n_i of degree 2, $\theta_{i,v}$ and $\theta_{i,z}$ are the sub-trees rooted at n_i composed of the portions of θ_i departing from branches $b_{i,v}$ and $b_{i,z}$, respectively. An example of such a topology is shown in Figure 3, where nodes are enumerated following the order of a depth first traversal, starting from the root n_1.

0-7803-7607-2/02/$17.00 ©2002 IEEE

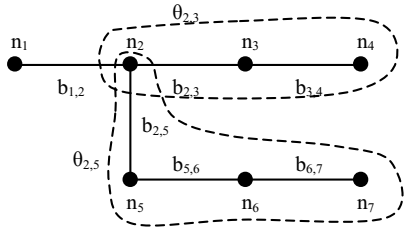

Figure 3. Example of routing tree topology.

2.2 Clock and Routing Grids

The clock distribution network of a chip is modeled as a regular grid of n independent domains Δ_i distributed over the die. The clock skew is then represented by an upper triangular matrix Σ where an element $\sigma_{i,i}$ indicates the skew within domain Δ_i and an element $\sigma_{i,j}$ indicates the skew between domains Δ_i and Δ_j. The location of the nodes of every Θ is then constrained to the center of the tiles of a finer regular routing grid superimposed to the clock one. This scenario is depicted graphically in Figure 4. An important requirement on the routing grid is that the size l_{tile} of its tiles should be short enough to allow an effective insertion of repeaters and flip-flops. In particular this can be achieved by choosing a value two or more times shorter than a given process dependent repeater critical length l_{crit}, defined here as the typical distance between the repeaters of a delay optimized two-pin interconnect routed over the most resistive metal layer used. Let $L_{crit} = l_{crit} / l_{tile}$ be such distance measured in terms of routing tiles. It has been shown [8] that repeater delay is quite insensitive to local displacement, therefore the center of a relatively small grid tile provides a good approximation for the real location of the repeater to be positioned anywhere within the tile.

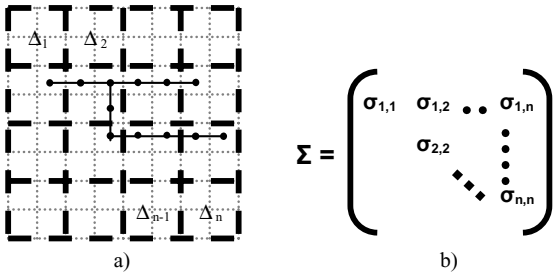

Figure 4. a) Routing grid and clock domains. b) Skew matrix.

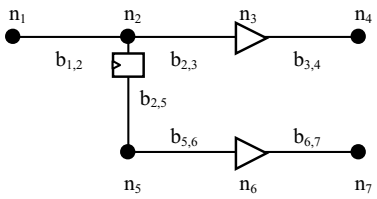

Figure 5. Assignment for the topology Θ of Figure 3. $A_\Theta = \{a_1, a_{2,3}, a_3, a_4, a_{2,5}, a_5, a_6, a_7\}$, where $a_1 = 0$, $a_{2,3} = 0$, $a_3 = bb$, $a_4 = 0$, $a_{2,5} = ff$, $a_5 = 0$, $a_6 = bb$, $a_7 = 0$.

2.3 Routing Resource Allocation

Interconnects routed over a topology Θ can be designed by allocating in its nodes and branches routing resources such as wires of given metal layer, length, width, and repeater gates of given size. Since this paper focuses on the repeater insertion problem, for simplicity we assume all branches $b_{u,v}$ as allocating wires of given length $l_{u,v} = l_{tile}$ with same metal layer and fixed width. A repeater assignment A_Θ over topology Θ is then defined as a set of labels $a_{u,v}$ and a_u where $a_{u,v} = g_k$ corresponds to the assignment of a repeater g_k, taken from a given gate library G, to branch $b_{u,v}$ right after node n_u. On the other hand, a value $a_{u,v} = 0$ indicates that no repeater is inserted. For a node n_u branching to two children n_v and n_z through branches $b_{u,v}$ and $b_{u,z}$, $a_u \equiv a_{u,v} \cup a_{u,z}$, whereas if n_u has one child n_v only, $a_u \equiv a_{u,v}$. Finally if n_u is a leaf $a_u = 0$. A possible assignment for the topology of Figure 3 is showed in Figure 5.

2.4 Wire and Gate Modeling

A wire of length $l_{u,v}$ routed over branch $b_{u,v}$, is modeled with a resistance $R_{u,v}$ connected between nodes n_u and n_v, and two capacitances of value $C_{u,v}$ connected between n_u and ground and n_v and ground, respectively. If the wire has distributed resistance R_l and distributed capacitance C_l, the lumped $R_{u,v}$ and $C_{u,v}$ can be calculated as $R_{u,v} = R_l\, l_{u,v}$, and $C_{u,v} = 0.5\, C_l\, l_{u,v}$. The wire model of a simple topology with three nodes and two branches is showed in Figure 6. A repeater g_k is modeled as a buffer gate with input capacitance **load**(g_k) and input to output delay **delay**(g_k, C_{out}), expressed as a function of the capacitance C_{out} present at its output. Similarly, if g_k is a clocked device it is modeled as a D-type flip-flop with input capacitance **load**(g_k), clock to output delay **delay**(g_k, C_{out}), and set-up time $T_{set\text{-}up}(g_k)$.

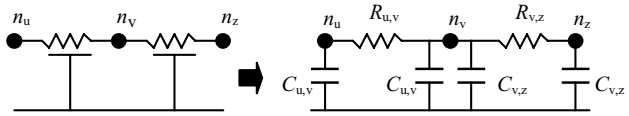

Figure 6. Wire modeling.

2.5 Interconnect Cover

Let us assume that a repeater assignment is given for a routing tree Θ along with timing constraints at its leaves in terms of input load capacitance and propagation required time. Under these circumstances, the timing at the root of each sub-tree θ_i of Θ needed to satisfy those constraints can be expressed by a 4-tuple $\gamma_i = (c_i, r_i, \lambda_i, a_i)$. Here, c_i is the input capacitance seen at the root, r_i is the required arrival time after the positive edge of a clock signal φ with period T_φ, λ_i is the interconnect latency defined as the max number of clocked repeaters crossed when going from the root of θ_i to its leaves, and a_i is the repeater assignment at n_i. Since every leaf n_u of Θ is also the elementary tree $\theta_u \equiv (\{n_u\}, \emptyset)$, the interconnect constraints at the receivers are also specified using a 4-tuple $\gamma_u = (c_u, r_u, 0, 0)$. Because γ specifies the timing constraints and the allocated resources of an interconnect mapped onto the topology of sub-tree θ, we call γ interconnect *implementation* or *cover* of θ. In general a sub-tree θ_i will have multiple feasible covers specifying different timing and resource assignments. For convenience, these covers are grouped per latency in the ordered set $\Gamma_i = \{\Gamma_i^m, \ldots, \Gamma_i^n\}$ with $m, n \geq 0$, and $m < n$, where $\Gamma_i^k = \{(c_1, r_1, k, a_1), \ldots, (c_n, r_n, k, a_n)\}$ is a set of covers of θ_i with same latency k. In the case of a sub-tree θ_u rooted at a branching point of degree two, $\gamma_{u,v}$ and $\gamma_{u,z}$ are used to denote the covers of the sub-tree components $\theta_{u,v}$ and $\theta_{u,z}$, respectively.

3 Cover Computation

If we assume that the covers at the leaves of Θ are given as constraints and that an assignment A_Θ is known, the cover of every sub-tree θ_i can be recursively determined, from leaves to root, using a hierarchical delay model such as Elmore [2]. This is accomplished here by the three operations **wire**, **repeat** and **join** sketched in Figure 7. Specifically, If a node n_u branches to node n_v with a known γ_v through branch $b_{u,v}$, cover $\gamma_u \equiv \gamma_{u,v}$ is first calculated through operation **wire**, where $\gamma_{u,v}$ is back propagated to n_u inserting a wire on branch $b_{u,v}$ using Elmore delay. It must be noted that, unlike [1], covers γ are here constrained by a fixed given clock cycle T_φ. Therefore, only covers with non-negative required time are generated. After operation **wire**, if a gate must be inserted at n_u, operation **repeat** is called. For the reason just stated, a non-clocked repeater is inserted only if the required time at its input is zero or

positive. In that case the new required time and the input capacitance of g are stored in the cover while the latency remains unchanged. Similarly, a clocked repeater is inserted only if the slack at its output is non-negative. In particular, this slack is computed from the required time of the wire by subtracting the delay of the flip-flop and the term $\sigma_{m,n}$ that models the skew of the clock signal φ as defined in Section 2.2.

```
// Compute γ_{u,v} from γ_v inserting a wire on branch b_{u,v}
wire (b_{u,v}, γ_v)
1.    γ_{u,v} = Ø
2.    if slack = r_v – R_{u,v} (C_{u,v} + c_v)    0
2.1       γ_{u,v} = (2C_{u,v} + c_v, slack, λ_v, 0)
3.    return γ_{u,v}
```

```
// Compute γ_u joining γ_{u,v} and γ_{u,z}
join (γ_{u,v}, γ_{u,z})
1.  γ_u = (c_{u,v} + c_{u,z}, min(r_{u,v}, r_{u,z}), max(λ_{u,v}, λ_{u,z}), a_{u,v} ∪ a_{u,z})
2.  return γ_u
```

```
// Compute a new γ from γ_{u,v} inserting repeater g
repeat (γ_{u,v}, g)
1.    γ = Ø
2.    if slack = r_{u,v} – delay(g, c_{u,v})    0
2.1       if g is not clocked
2.1.1         γ = (load(g), slack, λ_{u,v}, g)
2.2       else if slack – σ_{m,n}    0
2.2.1         γ = (load(g), T_φ – T_{set-up}(g), λ_{u,v} + 1, g)
3.    return γ
```

Figure 7. Operations for cover computation.

Here, Δ_m is the clock domain where the flip-flop g is located and Δ_n is chosen among the domains of the upstream flip-flops so as to consider the most pessimistic value of $\sigma_{m,n}$. If the slack is not negative, the required time at the input of the gate is set to the period of the clock minus the set-up time of gate g, and the latency of the new cover is increased by one. When two covers $\gamma_{u,v}$ and $\gamma_{u,z}$ are back propagated to a branch node n_u of degree two via operations **wire** and **repeat**, cover γ_u is calculated by means of operation **join**. Here, the input capacitance is the sum of the load seen at the two branches and the required time is the minimum of that at $\gamma_{u,v}$ and $\gamma_{u,z}$ to account for the worst case, whereas according to its definition the latency of the joined cover is the maximum of the latencies at the merging branches.

4 Cover Optimality

When multiple covers are computed for a sub-tree θ, only those *non-inferior* covers that can lead to optimal solutions at the root of Θ need to be saved. A principle for cover optimality was introduced in [1] to prune cover sets of their inferior solutions. We extend here those concepts to the general case of non-zero latency in the following property.

Property 4: Cover Inferiority

$\forall\gamma \in \Gamma$, γ is inferior in Γ if $\exists\gamma' \in \Gamma$ such that at least one of the following is true:

a. $\lambda = \lambda'$, $c \quad c'$, $r < r'$
b. $\lambda = \lambda'$, $c > c'$, $r = r'$
c. $\lambda = \lambda'$, $c = c'$, $r = r'$, $\text{cost}(\gamma) > \text{cost}(\gamma')$
d. $\lambda > \lambda'$, $c \quad c'$, $r \quad r'$

In properties 4.a and 4.b it is easy to see that γ is an inferior cover since any gate driving a sub-tree θ with cover γ will have input required time always worse than that of the same gate driving θ with γ', while having the same input capacitance and the same latency. On the other hand, when γ and γ' have identical input capacitance and required time, as in property 4.c, γ is also inferior if the value of a user specified cost function associated with the routing resources allocated in θ by γ, e.g. repeater area, is greater than that of γ'. Finally, when γ has latency higher than that of γ', as in property 4.d, γ is inferior for the same reasons as in 4.a, 4.b and, when it has identical input capacitance and required time, because γ covers sub-tree θ with same timing as in γ' but wasting an unnecessary extra clock cycle.

5 Assignment for Minimum Latency

The repeater insertion problem for the design of interconnects with minimum latency, under the definitions of Section 2, is stated here with the following formulation. Given an interconnect topology Θ mapped onto a routing grid and a clock grid with skew matrix Σ, timing constraints at the receivers in terms of $\gamma_u \equiv (c_u, r_u, 0, 0)$, a library G of clocked and non-clocked repeaters, find a set of optimal covers Γ_1 with minimum latency at the driver of $\theta_1 \equiv \Theta$, according to property 4. This is accomplished here by calling the algorithm **MiLa**, whose pseudo-code is outlined in Figure 8, with argument θ_1. Notice that minimizing the latency in Γ_1 corresponds to minimizing the signal latency at the most latent receiver of the net. Also, please notice that the latency values λ_u at the receivers are here set to zero only for convenience. The optimal covers at each node of the tree are computed recursively after multiple nested calls, starting from the leaves and ending at the root, traversing the tree in a depth first fashion. At any call, if θ_u is a leaf the given constraint at the corresponding receiver $\Gamma_u = \{(c_u, r_u, 0, 0)\}$ is returned.

```
// Compute optimal covers Γ_u of sub-tree θ_u for min latency
MiLa(θ_u)
1.    if θ_u is a leaf then Γ_u = (c_u, r_u, 0, 0)
2.    else if θ_u root branches once to b_{u,v}
2.1       Γ_v = MiLa(θ_v)
2.2       Γ_u = ∪_{γ∈Γv} (wire(b_{u,v}, γ))        // insert |Γ_v| covers
2.3       Γ_g = Ø
2.4       for each g in G                          // insert |G| covers
2.4.1         Γ = ∪_{γ∈Γu} (repeat(γ_{u,v}, g))
2.4.2         apply property 4. to Γ ⇒ ∀ |Γ^k| = 1
2.4.3         Γ_g = Γ_g ∪ Γ
2.5       Γ_u = Γ_u ∪ Γ_g
3.    else if θ_u root branches twice to b_{u,v} and b_{u,z}
3.1       Γ_{u,v} = MiLa(θ_{u,v}), Γ_{u,z} = MiLa(θ_{u,z})
3.2       // Γ_{u,v} ≡ {Γ^x, …, Γ^y}, Γ_{u,z} ≡ {Γ^m, …, Γ^n}
3.3       if y < n then swap(Γ_{u,v}, Γ_{u,z})
3.4       for k = x – n to y – m   // latency shift operation
3.4.1         Γ_u = Γ_u ∪ merge (Γ_{u,v}, {Γ^{m+k}, …, Γ^{n+k}})
4.    apply property 4. to Γ_u
5.    if θ_u = θ_1 then Traverse the tree from root up and
                        compute the latency at each receiver
6.    return Γ_u
```

Figure 8. The MiLa algorithm.

If the root of θ_u is connected to a single branch $b_{u,v}$, in line 2.1 the algorithm is called again to compute the optimal covers Γ_v of the next sub-tree θ_v, then in line 2.2 such covers are propagated to node n_u inserting wires. Next, in loop 2.4 an additional cover is inserted in Γ_u for each repeater in G. To do this, all the covers previously computed in line 2.2 are repeated using the same repeater gate g calling operation **repeat** thus generating a new set Γ. Inferior covers are then deleted according to property 4 leaving Γ with only one optimal cover for every available sub-set $\Gamma^k \subseteq \Gamma$ with latency k, since its covers originated from the same repeater. Finally, on line 2.5, Γ_u is updated adding the repeated covers. Section 3 of the **MiLa** algorithm computes the optimal covers of sub-tree θ_u when its root is connected to two branches $b_{u,v}$ and $b_{u,z}$. A methodology for joining covers without taking into account latency was given in [1] and more formally in [3]. The same method, extended to the latency case, is implemented in the function **merge** outlined in Figure 9. Here, only the elements of Γ_u and Γ_v with the same latency are joined using a technique similar to the merging of two sorted lists. In this case we assume that covers in sets Γ_u and Γ_v are sorted first by increasing latency and then, within each latency, by increasing required time and

capacitance. It is apparent that a cover not featuring such a monotonic increasing behavior would be an inferior one according to property 4 and would then be deleted from its set prior to the merge.

```
// Join covers with same latency from Γu and Γv in Γ
// max |Γ| = |Γu| + |Γv|
merge (Γu, Γv)
1.      // γj^i ≡ i-th element of Γj, λj^i = latency of γj^i
2.      Γ = Ø, x = y = 1
3.      while x ≤ |Γu| and y ≤ |Γv|
3.1         if λu^x > λv^y then y = y + 1, goto 3.
3.2         if λu^x < λv^y then x = x + 1, goto 3.
3.3         Γ = Γ ∪ join(γu^x, γv^y) // from Fig. 3
3.4         if ru^x ≠ rv^y then x = x + 1
3.5         if rv^x ≠ ru^y then y = y + 1
4.      return Γ
```

Figure 9. Function merge.

On line 3.1 **MiLa** calls itself twice to compute the covers of sub-trees $\theta_{u,v}$ and $\theta_{u,z}$. As illustrated in line 3.2, the corresponding sets $\Gamma_{u,v}$ and $\Gamma_{u,z}$ will be in general composed of an arbitrary number of sub-sets Γ^k of different latency k, where k is equal to the max latency at the root of each branch. Similarly to the *cover latency* λ defined in section 2, the *signal latency* at any node of an interconnect can be defined as the number of flip-flops crossed to reach the node starting from the driver where the signal latency is zero. While the algorithm is geared towards minimizing the signal latency at the most latent receiver, it must also determine the signal latency at all other receivers such that optimal covers are obtained and propagated back to the driver. To do so, we must join all combinations of the sub-sets $\Gamma^k \subset \Gamma_{u,v}$ and $\Gamma^h \subset \Gamma_{u,z}$ so that for each couple (Γ^k, Γ^h) a new joined sub-set $\Gamma^q \subset \Gamma_u$ is generated with function **merge** where $q = \max(k, h)$. For example, a value $q = k$ would correspond to the case of joined covers of Γ_u of latency k where the h-latency covers of $\Gamma_{u,z}$ have been shifted in time by latency $k - h$. The general case is implemented in loop 3.4 of **MiLa** where all possible shifts in latency are generated and the joined covers of Γ_u computed by function **merge**. Here, because of line 3.3, set $\Gamma_{u,v}$ is the one which contains covers with maximum latency. Therefore, on line 3.4.1, only the covers of $\Gamma_{u,z}$ need to be shifted by latency k to consider all possible cases. After determining the optimal covers Γ_u for every case of branching degree at the root of θ_u, in line 4 set Γ_u is pruned of its inferior elements according to property 4. Finally, in line 5 the optimal set Γ_u is returned. It is worth noting that the application of property 4 has the important consequence of limiting the returned set Γ_u to having covers with at most two values of latency, that is $\Gamma_u = \{\Gamma^k, \Gamma^{k+1}\}$. Such a property, also experimentally verified, holds assuming that library G contains at least one flip-flop with input capacitance equal or lower than the input capacitance of all non-clocked repeaters. After the last call to **MiLa** returns, the optimal covers γ_i in Γ_1 for the whole interconnect are computed and repeater assignments and the signal latencies at every receiver found by traversing the tree from root on.

The optimality of the algorithm is proved by induction on the sequence of recursive computations of cover sets Γ_u generated by the depth first traversal induced by the first call to θ_1. Therefore, assuming that the given covers Γ at the receivers are optimal, we only have to prove the optimality of the covers produced by one recursive call to section 2 or 3 of the algorithm. However, both sections 2 and 3 produce a cover set Γ_u containing among its elements all possible optimal solutions according to the problem formulation. The optimality of all the covers of Γ_u is then ensured by property 4, which eliminates any inferior element.

6 Assignment for Given Latency

Algorithm **MiLa** is here modified into a new algorithm called **GiLa**, outlined in Figure 10, to accept latency constraints and perform repeater insertion with the same underlying methodology, using a simple and effective heuristic to resolve latency mismatch at intermediate points. Similarly to **MiLa**, timing constraints at the receivers are given in terms of $\gamma_u \equiv (c_u, r_u, \lambda_u, 0)$, but now λ_u specifies the latency constraint between the receiver at γ_u and the driver at the root, with inverted sign.

```
// Compute optimal covers Γu of sub-tree θu given latency
// constraints λu at each driver-receiver pair
GiLa(θu)
1.      if θu is a leaf then Γu = (cu, ru, λu, 0)
2.      else if θu root branches once to bu,v
2.1         Γv = GiLa(θv)
2.2         Γu = ∪γ∈Γv (wire(bu,v, γ))    // insert |Γv| covers
2.3         Γg = Ø
2.4         for each g in G               // insert |G| covers
2.4.1           Γ = ∪γ∈Γu (repeat(γu,v, g))
2.4.2           apply property 4. to Γ ⇒ ∀ |Γ^k| = 1
2.4.3           Γg = Γg ∪ Γ
2.5         Γu = Γu ∪ Γg                  // Γu ≡ {Γ^x, …, Γ^y}
2.6         if θu = θ1 then
2.6.1           if x > 0 then exit: the net is not feasible
2.6.2           if y < 0 then     // insert –y more flops in Γu
2.6.2.1             Γu = ReFlop(θu, –y)
3.      else if θu root branches twice to bu,v and bu,z
3.1         Γu,v = GiLa(θu,v), Γu,z = GiLa(θu,z)
3.2         // Γu,v ≡ {Γ^x, …, Γ^y}, Γu,z ≡ {Γ^m, …, Γ^n}
3.3         if y < m then     // insert m – y more flops in Γu,v
3.3.1           Γu,v = ReFlop(θu,v, m – y)
3.4         if n < x then     // insert x – n more flops in Γu,z
3.4.1           Γu,z = ReFlop(θu,z, x – n)
3.5         Γu = Γu ∪ merge (Γu,v, Γu,z)
4.      apply property 4. to Γu
5.      return Γu
```

Figure 10. The GiLa algorithm.

The latency λ_1 at the driver is set to be always zero. For example, the latency constraints of the assignment of Figure 5 are $\lambda_1 = 0$, $\lambda_4 = 0$, and $\lambda_7 = -1$. Using negative numbers to express the latency at the receivers allows us to reuse operations **join** and **repeat** in **GiLa** without modification. As in the case of **MiLa**, the covers Γ_1 solutions of the problem are computed calling **GiLa** with argument θ_1. **GiLa** proceeds then exactly in the same way of **MiLa** except when branch points of degree two are reached in Section 3 of the algorithm.

```
// Insert extra flops in branch rooted by sub-tree θu
ReFlop(θu, extra_flops)
1. Traverse the tree from θu up removing sets Γ along the way
   and computing the number crossed_flops of crossed flip-flops
   until either a leaf or a branch point of degree 2 is reached.
2. Traverse the tree down back to θu generating new sets Γ using
   the wire and repeat functions but this time forcing the
   insertion in the branch of an exact number of flip-flops equal
   to crossed_flops + extra_flops. In particular, flip-flops are
   equally spaced along the branch so as to equally distribute
   the extra positive slack introduced. If there are more flip-flops
   to be inserted than available locations, extra flip-flops are
   inserted in already occupied locations.
3. return Γu
```

Figure 11. Function ReFlop.

Here, if the computed sets $\Gamma_{u,v}$ and $\Gamma_{u,z}$ have covers with same latency, then function merge is called in line 3.5 and the merged set Γ_u returned after being pruned of inferior solutions. However, if no such covers exist the difference in latency between the two branches is computed and the sub-tree with lowest latency recomputed in line 3.3.1 for $\theta_{u,v}$, or 3.4.1

for $\theta_{u,z}$, by function **ReFlop**, outlined in Figure 11, so that its latency is augmented by the corresponding difference. **ReFlop** implements a simple heuristic to insert the needed extra flip-flops in the processed branch. Its effectiveness relies on the following observation: if the covers Γ_u computed for minimum latency at the root of a sub-tree θ_u meet their timing constraints with less latency than requested, it is always possible to meet the same timing constraints also inserting extra flip-flops to increase the latency of the branch rooted at θ_u. Furthermore, to avoid wasting unnecessary area, extra flip-flops are not inserted after a branching point of degree two as that would lead to the use of two flip-flops for the gain of a latency value of one only. After reaching the root of Θ, in Section 2.6 **GiLa** also checks that the solution set Γ_1 has covers with latency zero, corresponding to meeting the latency constraints specified at the receivers. If all covers have latency greater than 0, then the latency constraints are infeasible as no solution can be achieved with fewer flip-flops. On the other hand, if all covers have latency less than 0 **ReFlop** is called to insert the needed extra flip-flops.

7 Experimental Results

Instead of directly implementing the basic algorithms, we have chosen to verify our methodology by applying it to a more complex case where repeater insertion is simultaneously performed with interconnect topology synthesis. To this purpose we have incorporated our repeater insertion strategy and the P-Tree$_{AT}$ [3] routing tree construction technique in two new algorithms referred here as to **FloP-Tree-ML** and **FloP-Tree-GL** based on the **MiLa** and **GiLa** algorithms, respectively. Aside from testing the correctness of the proposed methodology this has also served us in verifying that our technique is indeed amenable to being employed in other interconnect design algorithms based on the same dynamic programming style of [1] to extend them to the broader case of clocked repeater insertion. In the following both **FloP-Tree-ML** and **FloP-Tree-GL** are experimentally verified.

7.1 Set-up

In our experiments, the proposed algorithms are applied to perform concurrent topology synthesis and repeater insertion in a test case composed of 10972 nets with various pin count, representing the global nets of a previous generation commercial microprocessor based on a 0.18µm CMOS process. The repeater library *G* used in the experiments is composed of two repeaters, a single-stage inverter and a D-type flip-flop, with sizes chosen to optimize the delay of an infinite repeated line [11]. The use of an inverting gate is possible since **FloP-Tree-ML** and **FloP-Tree-GL** keep track of the polarity of the repeated signals. For every net, a cover constraint Γ specified at each receiver is composed of two covers γ corresponding to two different available gate sizes. To consider the case of root covers with negative polarity two choices are also specified at the driver: a first given initial gate and a second gate with inverted logic function. Topology synthesis is then constrained to produce routing trees mapped onto a routing grid satisfying the requirements of Section 2.2, using only two metal layers. Moreover, the intra-domain σ_{ii} and the inter-domain σ_{ij} clock skews values of matrix Σ are equal to 10% and 25% of the target clock period, respectively. Finally, all experiments are run on an IBM ThinkPad equipped with a 1GHz Intel Pentium III processor and 512MB of memory.

7.2 Scaling Trends of Pipelined Interconnects

The goal of this experiment is the study of the latency scaling properties of the global nets of a high-performance microprocessor across different CMOS process generations. To this purpose, algorithm **FloP-Tree-ML** is applied to all the nets of our test case to produce routing solutions with minimum latency following different scaling scenarios created by:

- Scaling the devices and interconnects by a factor S=0.7 from the original 0.18µm process, first to a 0.13µm process and then to a 0.09µm one. Parasitics are derived from the ITRS [12] roadmap. Notice that using a 0.18µm design as a base for this study leads to optimistic projections since the number of global signals is likely to increase with newer processes leading to larger repeater counts.
- Scaling the die size and clock frequency following: N) *nominal* scaling, representative of microprocessor shrinks, where die size scales down by S^2 and frequency scales by S^{-1}, and T) *trend* scaling, where die size scales by 1.25 and frequency scales by 2 as indicated by the current microprocessor trends of Figure 1 and Figure 2 [10]. In addition, we also follow a third scaling rule C) *constant* scaling, representative of high-performance ASICs, in which the die size remains the same.

The combination of these rules generates six distinct scaling patterns for which the results of **FloP-Tree-ML** are shown in Table 1. Patterns are indicated with two letters corresponding to their component rules. For example, CN corresponds to a scaling pattern with constant die size and nominal frequency. A first run marked as REF, corresponding to the original 0.18µm process and a target clock frequency of 1GHz, is used to normalize the values of subsequent runs. Since **FloP-Tree-ML** produces a set of multiple net implementations with different topologies and repeater assignments, only one solution per set, the one with minimum number of repeaters and highest positive slack at the driver, is considered to generate the experiment results. As expected, for all scaling patterns, both the total number of flip-flops **Flops** and their percentage **FlopsR** with respect to the total number of clocked and non-clocked repeaters **Rptrs** increase. For brevity, let us consider in more details the two extreme scaling patterns NN and TT and the intermediate case CN. In the case of pattern NN, even though the die size shrinks, the total number of repeaters increases by about 30% each generation due to sheer process scaling, i.e. since in a delay optimized repeated line, repeaters scale faster than interconnects [11].

Table 1. Repeater and flip-flop insertion for minimum latency performed across three process generations for six different scaling patterns.

Scaling Pattern	Proc [µm]	Die Size	Freq [GHz]	Rptrs	FlopsR [%]	Flops	Area [%]	λ > 0 [%]	Covers	cpu [s]
REF	0.18	1	1	1	1.7	1	100	1	17	679
NN	0.13	0.5	1.4	1.29	2.2	1.7	56	3	13	367
	0.09	0.25	2	1.66	2.4	2.4	33	4	11	240
CN	0.13	1	1.4	2.16	2.4	2.9	79	6	15	761
	0.09	1	2	4.17	3.5	8.8	67	10	10	440
TN	0.13	1.25	1.4	2.5	3.1	4.6	89	6	13	589
	0.09	1.56	2	5.32	4.3	13.7	84	15	8	378
NT	0.13	0.5	2	1.3	4.2	3.2	57	5	13	382
	0.09	0.25	4	1.77	10.2	10.8	37	12	10	252
CT	0.13	1	2	2.17	5	6.5	81	8	15	757
	0.09	1	4	4.24	13.8	34.9	75	32	9	414
TT	0.13	1.25	2	2.5	5.2	7.7	90	9	13	579
	0.09	1.56	4	5.46	15.4	50	97	42	7	354

More interestingly, the total number of flip-flops increases by about 70% each process generation, but in this case also because of frequency scaling. Moreover, the total area of all repeaters decreases following the die size shrink. As can be seen, the run time of **FloP-Tree-ML** tends to decrease as the number of flip-flops goes up. This is due to the fact that in general more covers (the average number of covers per net is reported in column 10) are pruned as a consequence of the insertion of a flip-flop than because of the insertion of a non-clocked repeater. At the other extreme, using scaling pattern TT, which corresponds to a scaling trend that high-performance microprocessors have been able to sustain so far, the die size increases by 25% and the frequency doubles each generation. For this reason, the increase in repeaters is here more dramatic: every process generation the total number of repeaters goes up

by a factor of about 2.5X, while the number of flip-flops increases by about 7 times! Moreover, the percentage of pipelined interconnects (column **λ > 0**) is 9% and 42% for the 0.13µm and 0.09µm processes, respectively. For the scaling pattern CN, representative of high-end ASICs, the increase in the number of flip-flops is 2.16X and 2.9X for the scaled 0.13µm and 0.09µm processes, respectively.

For the convenience of the reader the total number of inserted clocked and non-clocked repeaters **Rptrs** and the total number of flip-flops **Flops** are plotted in logarithmic scale in Figure 12 and Figure 13, respectively, for the six scaling patterns of Table 1. In Figure 12, as expected, for all scaling patterns the total number of repeaters increases exponentially, independently of frequency, with increase rate growing from a minimum corresponding to the *nominal* die size scaling rule, to a maximum corresponding to the *trend* die size scaling rule. Similarly, Figure 13 shows that the number of inserted flip-flops also grows exponentially for all scaling patterns. This time, however, the increase rate depends on both die size and frequency.

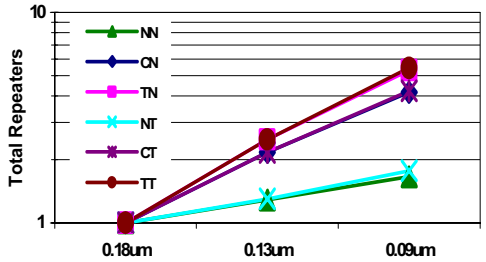

Figure 12. Increase in total number of repeaters and flip-flops for six scaling patterns across three process generations.

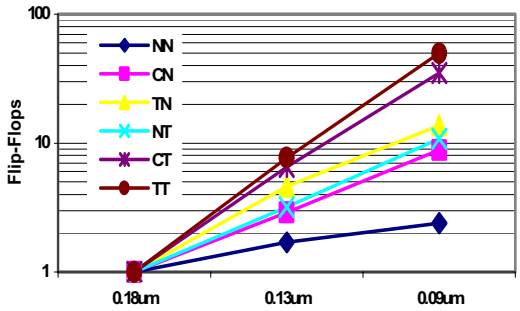

Figure 13. Increase in number of flip-flops for six scaling patterns across three process generations.

7.3 Latency Constrained Pipelined Interconnect Synthesis

In this experiment we test the functionality of **FloP-Tree-GL** by applying it to the latency constrained repeater insertion of the same test case used in the previous experiment. The set-up of the experiment is the same as the previous one with the exception that now latency constraints are given at each net driver-receiver pair. In particular, the constrained repeater insertion is performed on the scaled test case corresponding to the scaling pattern TT in Table 1. The latency constraints are generated by randomly adding 1 or 2 extra clock cycles to the receiver latency values computed by **MiLa** in the previous experiment in all nets with λ > 0. In practice, in the case of a microprocessor, such increase could be attributed to conservative latency values set at the architectural level. The results of the experiment are reported in Table 2, where normalized values refer to the REF experiment of Table 1. As can be seen, in all three runs the number of inserted flip-flops increases due to the higher latency constraints with respect to the values of Table 1. Nevertheless, the total number of flip-flops and repeaters does not substantially vary. Intuitively, this can be explained by the fact the inserted extra flip-flops take the place of normal non-clocked repeaters. As expected, for all runs, the run-time increases due to the extra calls to routine **ReFlop**.

Table 2. Latency constrained repeater and flip-flop insertion across three process generations for nominal scaling pattern.

Scaling Pattern	Proc [µm]	Die Size	Freq [GHz]	Rptrs	FlopsR [%]	Flops	Area [%]	λ > 0 [%]	Covers	cpu [s]
NN	0.18	1	1	1.00	3.2	1.9	100	1	14	738
	0.13	0.5	1.4	1.29	4.0	3.0	56	3	10	385
	0.09	0.25	2	1.67	4.2	4.2	33	4	8	257

8 Conclusion

We have presented a new methodology for the simultaneous insertion of repeaters and flip-flops in VLSI circuits. Pipelined interconnects are designed so as to minimized the overall signal latency or to satisfy latency constraints. Experimental results demonstrated the applicability of the methodology to a large industrial test case, where our study on the scaling properties of pipelined interconnects showed an exponential increase in the number of inserted clocked repeaters. The increase is dramatic enough to indicate that radically new design methodologies that are able to cope with this emerging problem must be adopted if current scaling trends are to be sustained.

References

[1] L.P.P.P. van Ginneken, "Buffer Placement in Distributed RC-Tree Networks for Minimal Elmore Delay". *Proc. IEEE Int. Symp. Circuits Syst.*, 1990, pp. 865-868.

[2] W.C. Elmore, "The Transient Response of Damped Linear Networks", *Journal of Applied Physics*, vol. 19, pp. 55-63, Jan 1948.

[3] J. Lillis, C.-K. Cheng, T.-T. Lin, C.-Y. Ho, "New Techniques for Performance Driven Routing with Explicit Area/Delay Trade-off and Simultaneous Wire Sizing", *Proc. IEEE/ACM Design Automation Conference*, 1996, pp. 395-400.

[4] C.J. Alpert, J. Hu, S.S. Sapatnekar, P.G. Villarubia, "A practical Methodology for Early Buffer and Wire Resource Allocation", *Proc. IEEE/ACM Design Automation Conference*, 2001, pp. 189-194.

[5] C.-P. Chen, N. Menezes, "Noise-Aware Repeater Insertion and Wire Sizing for On-Chip Interconnect Using Hierarchical Moment-Matching", *Proc. IEEE/ACM Design Automation Conference*, 1999, pp. 502-506.

[6] C.J. Alpert, A. Devgan, S.T. Quay, "Buffer Insertion With Accurate Gate and Interconnect Delay Computation", *Proc. IEEE/ACM DAC* 1999, pp. 479-484.

[7] T. Okamoto, J. Cong, "Interconnect Layout Optimization by Simultaneous Steiner Tree Construction and Buffer Insertion", *ACM/SIGDA Physical Design Workshop*, 1996, pp 1-6.

[8] J. Cong, T. Kong, D.Z. Pan, "Buffer Block Planning for Interconnect-Driven Floor-planning", *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 1999, pp. 358-363.

[9] R. McInerney, M. Page, K. Leeper, T. Hillie, H. Chan, and B. Basaran, "Methodology for repeater insertion management in the RTL, layout, floorplan and fullchip timing databases of the Itanium™ microprocessor", *Proc. Int. Symp. Physical Design*, San Diego, CA, Apr. 2000, pp. 99-104.

[10] Vivek De and Shekhar Borkar, "Low power and high performance design challenges in future technologies", Proc. Great Lakes Symp. On VLSI, IL, March 2000, pp 1-6.

[11] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI.* Reading, MA: Addison-Wesley, 1990.

[12] International Technology Roadmap for Semiconductors, http://public.itrs.net.