A Heuristic Algorithm for the Fanout Problem

Kanwar Jit Singh* Alberto Sangiovanni-Vincentelli

University of California, Berkeley - CA

Abstract

We present an algorithm to optimally distribute a signal to its required destinations. The choice of the buffers and the topology of the distribution tree depends on the availability of different strength gates and on the load and the required times at the destinations. The general problem is to construct a fanout-tree for a signal so that the required time constraint at the source node is met and the fanout-tree has a minimum area. Since the area constrained fanout problem is NP-complete and area is not a major consideration in present high density designs, we restrict our attention to the simpler problem of designing fast fanout circuits without any area constraint. The proposed algorithm builds the fanout tree by partitioning the fanout signals into subsets and then recursively solving each sub-problem. At each stage the algorithm generates a fanout tree that is an improvement over the previous stage. This feature allows the user to specify the improvement desired by the fanout correction process. The performance of the algorithm, when run on randomly generated distributions of required times and on real design examples, is very promising.

1 Introduction

As fast turn-around time becomes the major driving force in electronic industry, there is an increasing need to consider performance aspects during the synthesis process. No longer can a designer postpone looking at the circuit performance until after he has a functionally correct circuit. Doing so results in the designer spending a great deal of time trying to manually increase the circuit speed. Timing analysis on a circuit can isolate slow combinational sub-circuits that cause the circuit to fail performance tests. Such combinational sub-circuits need to be resynthesized either by reducing the length of the critical paths [2, 7] or by appropriately distributing the load driven by gates on the failing paths. As device sizes on integrated-circuits continue to shrink, capacitive effects of parasitics and routing become increasingly significant. In many present-day technologies the delay through a logic gate is dominated by the loading (extrinsic delay) as opposed to the delay from the input to output pin (intrinsic delay). This paper proposes methods of improving performance by tackling the problems introduced by high fanout gates.

Hoover and Pippinger [3] presented an algorithm to bound the fanouts in an arbitrary network with only a constant factor increase in circuit size and depth. The delay of the network is computed as the number of levels of fanin bounded gates. The algorithm does not take into account the availability of different versions of gates and consequently does not provide an adequate method for reducing the delay in a *mapped circuit*. It is not known how to choose the best value for the fanout bound (except trying all values and keeping the best).

More recent work [1] has tried to solve the problem of building a fanout tree so that the required time at the root of the tree (the gate where the signal originated) meets the timing constraint and the area of the tree is minimum. The input to the algorithm is a set of required times at signal destinations and a target technology containing the gates that can be used to implement the fanout tree. The constructive algorithm they propose uses dynamic programming techniques to select the gates/buffers to drive subsets of signals and build up a tree structure that satisfies required time constraints with the minimum area. Optimum results are achieved on a restricted class of circuits. They also propose a heuristic algorithm for practical implementation. The restriction of the problem to non-inverting buffers and the fact that the signal is distributed in the same phase as the root, we feel is a major limitation of the algorithm. Furthermore, an entry in the technology represents a specified gate driving a specified number of fanout loads. When the loads in the fanout set are different, this assumption is inaccurate.

Our approach is motivated by the work of [6] on the optimal decomposition of large gates into smaller ones. Under the assumption that the delay through some gates (NAND, AND) is proportional to the number of fanins, they develop a bi-recursive algorithm, based on the distribution of arrival times, to come up with an optimal decomposition of such large gates. The inputs of the gate are distributed into subsets and the maximum saving that results from doing so is computed. These smaller gates are then recursively decomposed. We observed that the delay of a gate is proportional to the number of outputs it drives. This led to the realization that a recursive approach, similar to that in [6], can be used to build fanout trees. The similarity between the two algorithms is that both evaluate delay improving transformations and accept the best. They then recursively work on the sub-problems that result. The main difference between the two algorithms (apart from the fact that they solve different problems) is is that our algorithm exploits the phase relationships between the signals to develop small fanout-trees. This makes our algorithm more computationally intensive.

2 The buffering algorithm

In Section 2.1 we introduce some basic definitions in order to ease the task of describing the algorithm. We explain the general outline of the buffering process in Section 2.2. We will present a methodology to use the algorithm for buffering a single node in order to reduce the constraint violation in a circuit. In Section 2.3 we introduce the basic ideas in the creation of the fanout tree for a single node. The problem being handled models a general buffering scenario. The general case corresponds to both complemented and uncomplemented signals being required at the destinations which can have different capacitive loads.

2.1 Definitions

A *combinational circuit* can be represented as a DAG (directed acyclic graph). Each vertex in the graph corresponds to a gate in the circuit. The gates in the circuit are assumed to be single output gates. There is a directed arc from g to h if gate h depends directly on gate g (there is a connection from the output of gate g to an input of gate h). The set of gates(vertices) that have a connection (directed arc) from themselves to h is the fanin set of h and is denoted by FI_h. Similarly the fanout set of h, FO_h, is the set of gates that is driven by gate h.

A parameterized linear delay model is used to compute the delay between an input pin and the output pin of a gate. For a gate g with n inputs, the model consists of a set of n triplets. The triplet (c_k^g, i_k^g, e_k^g), $k = 1, \ldots, n$ characterizes the delay between the k-th input and the gate output. The elements of the triplet are the capacitance of the input pin, the intrinsic delay and the extrinsic delay (delay per unit load). We use the notation I_g^h to denote the index of the input pin of gate h that is driven by gate g. To account for the routing capacitance we associate an additional capacitance C_w with each fanout. The delay between the output and the k-th input of gate g is computed as

$$d_k^g = i_k^g + e_k^g . \sum_{h \in FO_g} (C_w + c_{I_g^h}^h)$$

The notation r_{gh} denotes the *required time* (also called the expected arrival time) of a signal along the connection from gate g to gate h. We use r_g to denote the time a signal is required to arrive at the output of gate g. The *arrival time* at gate g is denoted by a_g and is the time that the signal actually arrives at the output of the gate. The required time at the primary outputs and arrival times at primary inputs are known (either specified by the user or determined automatically by a timing analysis).

*This research was supported by DARPA under contract 442427-25885, DEC, SGS-Thomson, Texas Instruments, Philips and Intel.

A *delay trace* consists of determining the arrival and required times at all nodes in the circuit after evaluating the delay through all the gates in the circuit. The arrival time at the output of gate g is computed after the arrival time of its inputs has been computed. The recursive formula is

$$a_g = \max_{h \in FI_g} (a_h + d^g_{I^g_h})$$

The formulae for recursively computing the required time at gate g are

$$r_{gh} = r_h - d^h_{I^h_g} \quad \text{and} \quad r_g = \min_{h \in FO_g} r_{gh}$$

The *slack* at a gate, g, is the difference between the required and arrival times at the gate output ($r_g - a_g$). If the slack at a gate is non-positive (the signal at the node output arrives no earlier than it is required) the node is said to lie on a *critical path*. The minimum value of slack is denoted by $S_{\min}$ and it corresponds to the most critical path. A path from the input to output on which all nodes have a slack less than ($S_{\min} + \epsilon$) is called an $\epsilon - critical$ path. The set of $\epsilon - critical$ paths forms a sub-circuit called the *ϵ-network*. Loosely speaking, this sub-circuit is the section that violates the timing constraints by the most.

2.2 Outline of the buffering process

The algorithm **buffer_network** used to reduce circuit delay is presented in this section. Input to the algorithm is a combinational circuit η consisting of gates implemented in a target technology. The primary inputs are assumed to be valid at the user-specified arrival times and the circuit is deemed to meet timing constraints if the primary outputs arrive before the specified required times. If the circuit does not meet these timing constraints the algorithm iteratively reduces the violation in the constraints by applying fanout correction at multiple fanout gates.

As with any procedure that aims at reducing circuit delay through a series of local transformations, we recognize the fact that buffering a single node may not reduce the delay through the circuit. There can be other paths that are still critical and not affected by this local transformation. One method to overcome this problem is to apply the local procedure on a set of gates that forms a node-cutset (separator set) of the critical paths. If the local procedure results in an improvement in delay on each gate in the cutset, an improvement in the delay of the entire circuit is achieved. This paradigm of applying a local procedure along a cutset to improve circuit speed was suggested by DeMicheli [2] and has since been used in other delay-reducing techniques [7].

For the purposes of applying fanout correction on the circuit, gates with a large number of fanouts on the critical path are good candidates. We weight the gates on the ϵ-critical paths and find a cutset of nodes. The choice of ϵ and the function used to weight the nodes is crucial in reducing the number of iterations required to meet the timing constraints. A weighting scheme that favors (gives small weights) nodes with a large fanout allows us to use a minimum-weighted cutset algorithm [5] to find the set of good nodes to buffer. For each gate on the cutset we apply the algorithm **buffer_node** described in Section 2.3 to generate the appropriate fanout tree. Improving the slack along a selected path eventually causes other paths to become more critical than the one under correction. Further improvement along the selected path does not improve circuit performance. This imposes an upper limit on the amount by which the required time of a gate input may improved and be reflected in the performance enhancement of the circuit. The algorithm accepts a parameter TR (the minimum value of required time at the gate input that will improve circuit performance) and terminates when the desired required time is achieved. Figure 2 explains how TR is computed for a node g on the critical path. The basic outline of the buffering algorithm is shown in Figure 1. The process of delay trace, cutset evaluation and buffering nodes along the critical cutset is repeated until the timing constraints are met or there is no improvement in the circuit delay. At this stage the buffering process terminates and the user may use other techniques — critical-path resynthesis, decomposition or gate selection — to reduce circuit delay.

2.3 Buffering a single node

In this section we introduce the algorithm **buffer_node** used to build fanout trees. We wish to build a fanout tree at the output of gate g in the combinational circuit. Assume that the signal g is distributed in the positive and negative phases to various destinations. The fanout set of gate g is FO and it consists of two parts — FO^+ and FO^-. FO^+ consists of gates to which the signal g is distributed. The elements in FO^- receive the complemented signal $\overline{g}$. The inverted signal is generated by an inverting buffer denoted by g_I.

```
buffer_network( network, $\epsilon$)
  do
      delay_trace();
      generate an $\epsilon$-network; /* $\epsilon$ is user defined */
      assign-node-weights to nodes in $\epsilon$-network;
      cutset = minimum-weight-node-cutset of $\epsilon$-network;
      foreach $n \in$ cutset
          $TR$ = Target value of required time at $n$;
          ($FO^+, n_I, FO^-$) = Fanout data for node $n$;
          buffer_node( $n, n_I, FO^+, FO^-, TR$);
  while ( timing constraints not met && delay decreases )
```

Figure 1: Outline of the Buffering Algorithm

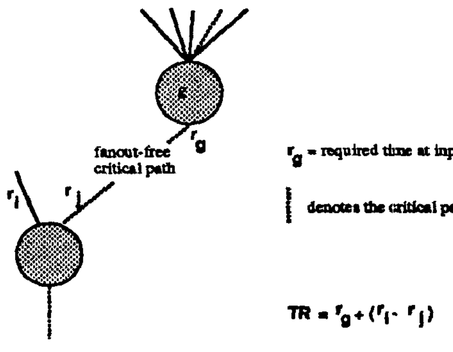
g
r_g
fanout-free critical path
r_i
r_j
r_g = required time at input of g
denotes the critical path
$TR = r_g + (r_i - r_j)$

Figure 2: Computing the target performance

For each gate h in the fanout set FO we know
1) capacitance c_h associated with the input pin of h,
2) required time r_h for the signal at the input pin of h.
Let the cardinality of the fanout sets be $|FO^-| = m$ and $|FO^+| = n$. For convenience we use the superscripts + and - to denote values associated with the sets FO^+ and FO^- respectively. The fanouts in both partitions are sorted by their required times i.e. $r^+_1 \leq r^+_2 \leq \cdots \leq r^+_n$ and $r^-_1 \leq r^-_2 \leq \cdots \leq r^-_m$. Assume that there are M inverting buffers in the target technology to be used for buffering. Non-inverting buffers can be made up as cascades of inverting buffers.

The algorithm **buffer_node** builds a fanout tree by selecting a delay reducing transformation at the current gate depending on the distribution of the loads and required times. The transformations aim to reduce the required time at the input of the selected gate. The three transformations that are used (named *repower*, *trans1* and *trans2*) are illustrated in Figure 3. $R0$, $R1$ and $R2$ denote the required time at the input of gate g on the application of these transformations.

The algorithm **buffer_node** is described in Figure 4. Please refer to Figure 3 while reading Figure 4. We first evaluate the required time $R0$ resulting from choosing the best available version of the gates g and g_I (the *repower* transformation). Having done the obvious powering-up of gate g, the distribution of fanout signals into two sets of early and late required signals is attempted according to *trans1*. The root gate is used to drive the early required signals and a smaller load. $R1$ is the maximum required time that is achieved using this transformation. If there is no saving ($R1 < R0$) as a result of *trans1* we conclude that the required times have a small spread. In that case the balanced decomposition according to *trans2* is evaluated. The best configuration of buffers added is found and the required time $R2$ is computed. We accept *trans2* only if $R2 > R0$. Having selected either *trans1* or *trans2*, the algorithm will recur on the new nodes created and then again on the root node. The routine $recursion\_will\_help()$ is used to prune recursive calls that will not help improve the required time at gate g.

The gate g is part of a network and consequently the point where the delay value is computed must be clearly defined. When we apply buffering transformations on gate g we need to consider not only the delay reduction of g but also the possible increase in the delay of the previous gate since g may now have a larger input capacitance. In our algorithm we take into account the drive (e^f) of the gate that feeds gate

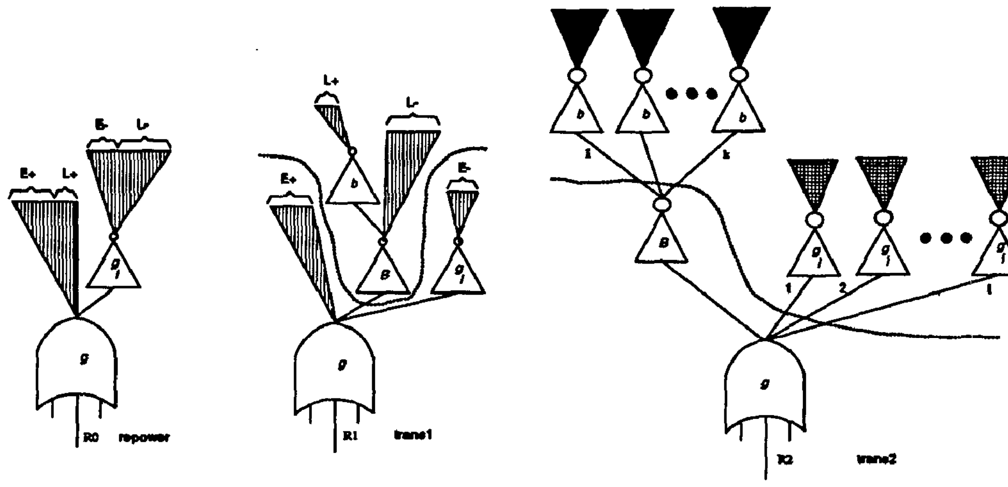

Figure 3: The basic buffering transformations

```
buffer_node( $g, g_I$ $FO^+, FO^-, TR$)
    $R0$ = evaluate_repower($g, g_I, FO^+, FO^-$);
    if ($R0 > TR$) return;
    ($R1, E^+, E^-$) = evaluate_trans1($g, g_I, FO^+, FO^-$);
    if ( $R1 > R0$) then
        create-configuration-trans1($g, g_I, E^+, E^-$);
        if ($R1 < TR$ && $recursion\_will\_help()$)
            buffer_node($B, b, L^-, L^+, TR$);
            buffer_node($g, g_I, E^+ \cup \{B\}, E^-, TR$);
        endif
    else
        ($R2, k, l$) = evaluate_trans2($g, g_I, FO^+, FO^-$);
        if ($R2 > R0$) then
            create-configuration-trans2($g, g_I, k, l$);
            if ($R2 > TR$)
                buffer_node($g, B, \bigcup_l g_I, \bigcup_k b, TR$);
        endif
    endif
```

Figure 4: Buffering of a single node

g, say f, when computing the savings as a result of the transformations. This captures the relevant environment for gate g and accurate evaluation of the savings can be made. Different values for e^f may lead to different fanout trees. We will now explain how the various transformations described in Figure 3 are evaluated and what is the criterion for accepting a transformation. We will also briefly discuss the complexity of applying the transformations.

We first evaluate the saving resulting from choosing the best available version of the gates (the *repower* transformation). This is the obvious way to get a faster circuit when different versions of the gate being buffered are available in the target technology. This transformation does not change the structure of the fanout tree (all signals in FO^+ are still driven by gate g and those in FO^- by g_I). The attempt here is to reduce the delay by choosing replacement gates for g and g_I that have the capability of driving large loads. The *repower* transformation results in a required time $R0$ being achieved at the input of gate g.

The cumulative capacitance of any subset of the fanout set, $A \subset FO$ is $C_A = \sum_{h \in A}(c^h_{I^h_g})$. Let r_{g_I} denote the required time of the signal at the input of the inverter g_I.

$$r_{g_I} = r_1^- - (i^{g_I} + e^{g_I}.C_{FO^-})$$

The required time at the input of gate g after *repower* is

$$R0 = \min[r_1^+, r_{g_I}] - (i^g + e^g.(c^{g_I} + C_{FO^+})) - e^f.c^g$$

The value $R0$ is what a simple algorithm for fanout correction, such as the one in [4], would achieve. Our objective here is to do better and so we try to restructure the fanout tree based on the distribution of required times.

Applying *trans1* involves partitioning the sorted set of fanouts into two sets E and L of early required signals and late required (less critical) signals respectively. Denote by E^+ and E^- the subsets of FO^+ and FO^- that comprise the early required signals. These are driven by the gates g and g_I respectively as shown in Figure 3. The late required subsets are therefore $L^+ = FO^+ \backslash E^+$ and $L^- = FO^- \backslash E^-$ which are driven by inverting buffers b and B respectively. For each partition we have different choices for the buffers g_I, b and B. The required time at the input of gate g is therefore denoted as $R1(E^+, E^-, b, B, g_I)$. We denote by r_{g_I}, r_b and r_B the required times at the inputs of inverting buffers g_I, b and B respectively.

The choice of the added buffers b and B, the gate g_I and the sets L^+ and L^- is made so as to maximize the required time at the input of gate g. The best required time that can be achieved by an application of *trans1* is denoted by $R1$. If an application of *trans1* enables us to meet the target required time TR, we implement the transformation and exit the recursion. If the target required time is not met and $R1 > R0$ (there is a saving compared to the *repower* transformation) we accept the transformation and recur on the problems of buffering gate B and then gate g. Recursive calls that cannot improve circuit performance are eliminated by the routine *recursion_will_help()* as described in Figure 4. An example of pruning recursive branches occurs in transformation *trans1*. Whenever the required time at the input of buffer B (Figure 3), namely r_B exceeds $\min[r_1^+, r_{g_I}]$ we can terminate the recursion. Buffering node B in an effort to increase r_B beyond its current value cannot increase the required time at the input of gate g. In addition, at each stage of the recursion we know the maximum amount that we need to save (derived from the target required time TR) and recursion terminates when this saving is achieved. These pruning strategies result in a significant improvement in the run times for the fanout correction process.

By appropriate precomputation of the cumulative capacitance values we can compute the required time at the input of gate g in constant time for a given configuration. Thus $O(M^3mn)$ computations are needed to find the maximum value of $R1$. Let the time to recursively apply the balanced decomposition for a problem with m signals in negative phase and n signals in the positive phase be denoted by $T1(m,n)$. If $|E^+| = s$ and $|E^-| = t$, the recursive application of *trans1* can be shown by the following equation —

$$T1(m,n) = O(M^3mn) + T1(n-s, m-t) + T1(t,s)$$

The solution to this equation that results in the largest run-time is $T1(m,n) = O(M^3m^2n^2)$.

When there is no saving by using the transformation *trans1*, one can easily infer that the required times in the sets FO^+ and FO^- are clustered together. At this stage a balanced tree as per transformation *trans2* is attempted to improve the required time at the input of g. The transformation evaluates a multi-way split of the positive and negative fanouts so as to result in clusters of similar cumulative capacitances. For a $k-$way balanced decomposition of FO^+ and a l-way partitioning of the negative fanout set FO^- we can compute the required time $R2$ at the input of gate g. For a given $(k,l)-$way decomposition of the fanout set FO (shown in Figure 3) we evaluate the required time $R2(l,k,g_I,b,B)$ at the input of gate g for the possible choices of inverting buffers b, B and g_I. We are interested in finding the configuration that yields the maximum value of $R2$.

With appropriate precomputations, $O(M^3mn)$ computations are needed to find the maximum value of $R2$. Let the time to recursively apply the transformation *trans2* for a problem with m signals in negative phase and n signals in the positive phase be denoted by $T2(m,n)$. $T2(m,n)$ can be computed by solving the equation —

$$T2(m,n) = O(M^3mn) + T2(k,l)$$

where $2 \leq l \leq \frac{m}{2}$ and $2 \leq k \leq \frac{n}{2}$. The solution to this equation is $T2(m,n) = O(M^3mn\log(mn))$.

The algorithm **buffer_node** is polynomial in the number of fanouts and, due to the pruning strategies that avoid needless recursions, it is fast. At each stage in the recursion it preserves a valid buffer tree which is better than the one at the previous stage. This allows the buffering to be done on a *need-to* basis. The amount of improvement desired can be used to stop the recursion when the desired saving has been obtained. This leads to improved run times and prevents the unwarranted area increase that could result from building the best fanout tree. However, if the best fanout tree is desired, the target required time TR can be set to be a large number.

3 Results

The algorithm described in Section 2 has been implemented as a part of **misII** — the logic synthesis system at Berkeley. The input to the algorithm is a network of mapped gates along with the library of gates that are available to implement the network. The mapping is carried out for a minimum area circuit so that none of the high power gates are chosen at this stage. The routine **buffer_network** is used to power up gates and to build fanout trees where required. Table 1 shows the results on some examples from the MCNC and ISCAS benchmark set. The standard cell library provided alongwith the MCNC'89 benchmark set was the target technology. Required times at the primary outputs are set to be the arrival time of the latest output and so any reduction in required times also results in reducing circuit delay. The wire load (C_w) associated with every fanout was set to be three times the input load on the smallest inverter. The drive at the primary input and load at the primary output was that of a small inverter. A small value of ϵ ($\epsilon = 0.5$) was chosen during this experiment to restrict attention to nodes close to the critical paths.

Table 1 shows the largest number of fanouts on any gate on the critical path. The initial area and delay of the circuit is shown. We compare these to the area and delay that results from application of the *repower* transformation. Since the target technology does not have different versions of many cells, this transformation sometimes does not lead to large saving. The comparison of the area and delay of the circuit after it has been exposed to the iterative application of the **buffer_network()** algorithm is also shown. From the experiment we see that buffering leads to significant reduction in delays with small area increase. We also see that simple repowering of gates very sub-optimal in almost all cases. As expected, the greatest saving occurs in circuits with large fanouts on the critical path.

Name	Max fanout	Before Buffering		Simple repowering		After Buffering		
		Area	Delay	Area	Delay	Area	Delay	Time
9sym	16	332	82.8	1.00	1.00	1.23	0.70	11
9symml	10	257	80.0	1.00	1.00	1.16	0.81	9
C1355	7	696	88.2	1.01	0.97	1.15	0.90	24
C432	21	408	218.1	1.00	1.00	1.20	0.51	16
C6288	24	4632	488.4	1.00	0.99	1.10	0.76	284
C7552	116	3207	389.8	1.00	0.71	1.08	0.57	173
C880	8	588	113.5	1.00	1.00	1.07	0.81	16
alu4	16	501	150.6	1.01	0.97	1.08	0.73	18
apex1	138	2923	429.0	1.00	0.74	1.05	0.61	632
apex2	25	908	105.0	1.01	0.80	1.03	0.76	33
apex3	113	2823	512.6	1.01	0.83	1.08	0.56	501
apex4	137	3762	1375.6	1.01	0.85	1.10	0.54	946
apex6	23	1108	98.6	1.00	81.6	1.12	0.54	46
apex7	7	362	73.0	1.01	0.91	1.04	0.86	9
clip	7	170	54.3	1.00	1.00	1.07	0.86	5
des	190	5129	426.1	1.00	1.00	1.15	0.26	526
duke2	9	553	77.8	1.01	0.94	1.10	0.73	16
e64	39	2924	105.1	1.02	0.81	1.04	0.80	78
misex2	5	145	39.6	1.02	0.91	1.19	0.86	4
rd73	5	99	59.4	1.01	0.99	1.17	0.88	3
seq	93	2569	377.0	1.01	0.77	1.07	0.65	270
z4ml	4	70	47.0	1.01	0.90	1.11	0.90	3

run-times are in seconds on a DEC-3100

Table 1: Results of Gate Buffering

4 Conclusions and future work

We have presented a heuristic solution to the problem of reducing delay in a circuit by reducing the load at gate outputs. The algorithm makes a selection of the fanout distribution network based on the loads and required times of the destinations, and by making use of the full range of buffers available in the given technology. Results from the use of the algorithm on a number of industrial designs give a significant reduction in circuit delay.

Further extensions to the buffering algorithm such as gate duplication and critical-path isolation are being looked into. We would also like to investigate the interaction between the buffering algorithm and other techniques to reduce delay, mainly the tree mapping and the critical-path resynthesis algorithms. This interaction results from the fact that the critical-path resynthesis depends on delay data which may be modified by the buffering process e.g. a path that is critical due to large fanout may be selected for resynthesis. In this case buffering, and not the restructuring of the critical path, is the preferred method to reduce circuit delay. Furthermore, given a circuit and timing constraints that need to be satisfied, we want to be able to determine the sequence of operations — buffering and resynthesis — that would meet the constraints with a minimal area increase. We are in the process of investigating the coupling between technology mapping (using DAG covering) and fanout correction in order to develop a method to smoothly explore the area-delay tradeoff in designs.

References

[1] C. L. Berman, J. L. Carter, and K. F. Day. The Fanout Problem: From Theory to Practice. In C. L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99. MIT Press, March 1989.

[2] G. De Micheli. Performance-Oriented Synthesis of Large-Scale Domino CMOS Circuits. *IEEE Transactions on CAD*, CAD-6(5):751–765, 1987.

[3] H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding Fanout in Logical Networks. *Journal of the Association for Computing Machinery*, 31(1):13–18, January 1984.

[4] K. Keutzer and M. Vancura. Timing Optimization in a Logic Synthesis System. In G. Saucier, editor, *Proceedings of International Workshop on Logic and Arch. Synthesis for Silicon Compilers*, pages 1–13, Grenoble, France, May 1988. Inst. Nat. Polytechnique.

[5] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[6] P. G. Paulin and F. Poirot. Logic Decomposition Algorithms for the Timing Optimization of Multi-Level Logic. In *Proceedings of ICCD 89*, pages 329–333, 1989.

[7] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. In *ICCAD-88*, pages 282–285. IEEE, 1988.