DON'T CARE WIRES IN LOGICAL/PHYSICAL DESIGN

by

Philip Chong, Yunjian Jiang, Sunil Khatri, Subarna Sinha
and Robert Brayton
CAD Research Group

Memorandum No. UCB/ERL M99/52

1 November 1999

DON'T CARE WIRES IN LOGICAL/PHYSICAL DESIGN

by

Philip Chong, Yunjian Jiang, Sunil Khatri, Subarna Sinha and Robert Brayton
CAD Research Group

Memorandum No. UCB/ERL M99/52

1 November 1999

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Don't Care Wires in Logical/Physical Design

Philip Chong, Yunjian Jiang, Sunil Khatri, Subarna Sinha, Robert Brayton
CAD Research Group, U.C. Berkeley

November 1, 1999

Abstract

A layout is obtained before the complete functionalities of individual cells are set. In particular, given an initial decomposition of a logic module, we determine, for each input to the nodes of the multi-level network, a set of "compatible" alternate wires, i.e. for every pin of the network, there is a **set** *of sources. These sets are compatible, like don't cares in logic; any combination can be used (as long as there is exactly one source for each pin) and each combination determines a different set of functionalities within the cells. SPFDs [10, 1, 9] are used to find compatible sets of alternate wires. We use this wiring flexibility to construct a wireplan; the final assignment of sources to pins depends on the ease with which the resulting netlist can be placed and wired. Once an assignment is chosen, the logic within each cell is determined. This partially reverses the classical approach of doing logic synthesis first, followed by physical layout and leads to an "information" driven placement methodology. We show results demonstrating total wire length improvements using this new flexibility.*

1 Introduction

Recent studies and projections point to the growing importance of wire delays as IC technologies continue to advance. This leads to the notion of **wireplanning** in place of floorplanning [8]; i.e. create a plan about how individual blocks may communicate, perhaps without fully knowing the functionality of those blocks. In **pure** wireplanning, this concept is taken to the extreme, by merely looking at dependencies to determine what connections ultimately have to be made, but not deciding *a priori* what particular computations are made in a block or even which blocks are to be built. After a plan for the information flow is made, logic functionalities are determined. An example of this is the work of Gosti *et al.* [4].

In this paper, we explore the concept of **don't care wires** which partially reverses the classical approach of doing logic synthesis first, followed by physical layout. A layout is obtained before the complete functionalities of individual blocks are set. In particular, given an initial decomposition of a logic module, we determine, for each input to the nodes of the multilevel network, a set of alternate wires. Thus for every pin of the network, there is a **set** of sources. These sets are like don't cares in logic; any combination can be used (as long as there is exactly one source for each pin) and each combination determines a different set of functionalities. In this sense the sets are **compatible**. Once an assignment of sources to pins is chosen, the logic within each block is determined.

We make the final assignment of sources to pins dependent on the ease with which the resulting netlist can be placed and wired. Thus a wireplan is chosen for the blocks, and then the logic is finalized.

We show how to use SPFDs (sets of pairs of functions to be distinguished) [10, 1, 9] to find compatible sets of alternate wires. We use this wiring flexibility to construct a wireplan. In Section 2, we give some details about SPFDs and how compatible wire sets can be constructed. Section 3 gives some methods for the initial logic decomposition and the motivation for the particular approach used. Section 4 poses an assignment problem of sources to pins which minimizes the total wire length, given a placement of modules. We show that the problem is NP-complete and give a heuristic that works well with the problem instances that arise. Section 5 gives two methods that we have experimented with, for using don't care wires in placement algorithms. Section 6 gives the details and results of the experiments we conducted. Section 7 concludes and describes future work that needs to be done in this area.

2 SPFDs and Compatible Wire Sets

We give an intuitive description of the concept of SPFDs (for a formal discussion on SPFDs, refer to

[10, 1, 9]). Consider a multi-level network of combinational nodes. Each node is associated with a logic function expressed in terms of its immediate fanins. Each fanin provides information to the node by giving different values for certain combinations of primary inputs to the module; it **distinguishes** different pairs of input minterms. The output of a node has a similar role; it must also distinguish different pairs of minterms for its fanouts. As long as each node plays its role, the particular logic function at the node is irrelevant for the functionality of the module. Several inputs of a node often distinguish the same pair of minterms. Hence there is some freedom in choosing which pairs to assign to which inputs; the only requirement is that the union of all the pairs of minterms distinguished by all the inputs covers the pairs of minterms required to be distinguished for that node. An SPFD for an input is a particular choice of pairs of minterms which must be distinguished by that input. Once the SPFD choices have been made, there may be different sources (nodes in the circuit) which can supply the required information, i.e. have enough distinguishing power. This allows different sources for each pin. In other words, we can choose one of several don't care wires for any input.

The actual logic function at the node will depend on which sources were assigned to which input pins since the information coming through each pin depends on the source chosen. However, once the information supplied to each pin is known, the logic function can be determined. Even then, there is a lot of flexibility for the function, since there are still many don't cares that can be used.

Since we want the freedom to choose the source for each pin independent of the choices made for the other pins, we need the sets of alternate wires to be **compatible**. This is similar to compatible don't cares for the nodes of a logic module, where we want to be able to choose the value for a particular node independent of the values chosen for the other nodes. So compatible sets must guarantee that the total information coming into a node through its input pins is enough to supply the information required for that node's required distinguishing power.

Definition 1 *Given a set of sets of nodes $R = \{R_k\}$, a* **selection** *is a ordered set of nodes $\{\eta_1, \ldots, \eta_{|R|}\}$ such that $\eta_k \in R_k$.*

Definition 2 *A set $\{R_k\}$ is* **compatible** *if for any selection of $\{R_k\}$, there exist logic functions at each node such that the implied netlist implements the primary output functions.*

We can create compatible sets of alternate wires using the following procedure.

Procedure 1 *(Constructing a Compatible Set)*

1. *Starting from the outputs and proceeding in a backward topological order, for each node η in the network, and each of its input pins, σ, compute their SPFDs, $SPFD_\eta$ and $SPFD_\sigma$. Once this is done, each SPFD represents the set of minterms which must be distinguished by that node or input. See [9] for the details of this computation.*
2. *Initialize for each node η,*

   $$ex(\eta) = \{\eta\} \cup TFO(\eta)$$

   and for each input pin, σ of η, let $R_\sigma = \{\eta'\}$ where η' is the current source of σ. R_σ will eventually represent the set of alternate wires for σ.
3. *Starting from the inputs and proceeding in some topological order, at each node η, do the following:*
   (a) *Let $C = \overline{ex(\eta)}$*
   (b) *For each fanin wire σ of η:*
   - *Find an $\eta' \in C$ such that $SPFD_\sigma \subseteq SPFD_{\eta'}$.*
   - *Include η' in R_σ, $R_\sigma = R_\sigma \cup \{\eta'\}$.*
   - *Update $ex(\eta') = ex(\eta) \cup ex(\eta')$. (This is done to avoid cycles in the resulting network.)*
   - *This continues until no more nodes can be added to R_σ.*

The set $\{R_\sigma\}$ obtained by the procedure has the following property.

Lemma 2.1 *For any selection of $\{R_\sigma\}$, the resulting network is acyclic.*

Although the above procedure produces a particular set, any set with the following two properties will suffice for compatibility.

Theorem 2.1 *Any set of sets of nodes $\{R_\sigma\}$ satisfying*

1. *for any selection, the resulting netlist is acyclic, and*
2. *$\eta' \in R_\sigma \longrightarrow SPFD_\sigma \subseteq SPFD_{\eta'}$.*

is compatible.

Thus to form a compatible set of alternate wires it is sufficient to make sure that whatever netlist is chosen, it is acyclic, and that each alternate's SPFD covers the original input's SPFD.

3 Initial Decompositions

Recent work on noiseless fabrics [5] led to a re-examination of the use of multi-level PLAs as a general logic synthesis technique, even for implementations where noise is not a major concern. The network is decomposed and clustered into PLAs with the target of absorbing the most wires internally in each PLA such that the resulting PLA network has no cycles. During this clustering no placement information is known, so we use the heuristic that the smaller the number of connections, the better the decomposition. This usually leads to a smaller number of PLAs. During clustering, the logic is minimized and the PLA folded. The result is constrained to be within given bounds (dictated mainly by delay and noise constraints within the PLA) in the number of rows and columns,

After clustering, a set of compatible alternates is generated for each input connection. Once the final netlist is chosen, the logic inside a PLA may change and no longer fit within the bounds. We experimentally determined how much the resulting PLAs can change. Note that the number of inputs and outputs does not change for a given PLA.

We experimented with the following method for initial decomposition: we decompose a network into a set of PLAs of medium size (e.g. 10–15 inputs, 1–5 outputs, 15–25 cubes). These PLAs are the nodes in our logic network. A set of compatible wire sets is generated using SPFDs. These are fed to a placement tool that selects the best set of alternate wires. The final placement and the best choices are fed back to the logic synthesis program which determines the logic functionality of each PLA, and then minimizes and folds it. The final areas and logic functionalities are returned to placement for final optimization of the placement.

4 An Assignment Problem

Evaluating the total wire length of a placement requires that a selection of alternate wires be made. We thus have the following problem.

Alternate Wire Choice Problem (AWC) *Given a point placement of pins, and a set $\{R_\sigma\}$ of candidate sources for each input pin, find the selection which minimizes the sum of the half perimeters of the bounding boxes of the nets.*

Theorem 4.1 *The Alternate Wire Choice (AWC) Problem is NP-complete.*

Proof: The associated decision problem is to: determine if there exists a choice of alternate wires such that the sum of wire lengths is at most k. Clearly this problem lies in NP; a certificate for this problem is the choice of alternates for each input pin. Given this, the total wire length can be evaluated in polynomial time and compared to k.

To show NP-completeness, consider a reduction of SAT to AWC. Take a SAT formula F in conjunctive normal form. Let $v_1, \ldots, v_n$ be the variables which appear in F. In the corresponding AWC problem, create a cell located at coordinates $(0,0)$ with outputs $o_1, o_1', \ldots, o_n, o_n'$ corresponding to the literals v_i and their complements. Now create a cell at coordinates $(2,0)$ with inputs $a_1, \ldots, a_n$, and for each input a_i construct two alternate wires, one from o_i and one from o_i'. This structure represents the assignment of values to the variables v_i; the wire from o_i to a_i is chosen if v_i is assigned to 1, or the wire from o_i' to a_i is chosen if v_i is assigned to 0.

Finally create a cell at coordinates $(1,0)$ with inputs $b_1, \ldots, b_m$; each input b_i corresponds to a clause C_i in F. For each input b_i, construct alternate wires corresponding to the literals in C_i; b_i has an alternate wire from o_j (o_j') iff C_i contains the literal v_j (v_j').

Now consider the choices of alternates for each b_i, given some choice of alternates for the a_js, i.e. under some assignment to the variables v_j. If some literal v_j (v_j') of C_i was given a value of 1 under the assignment, then the alternate for b_i may be chosen to be the corresponding o_j (o_j'). Thus, if all the clauses can be satisfied under the given assignment of variables, then there exists a choice of alternates for the inputs b_i such that for all $1 \leq j \leq n$, either output o_j or o_j' has no connecting wire chosen. Then there is a choice of alternate wires for b_i which gives a total wire length of exactly $2n$.

Conversely, if the given assignment does not satisfy C_i, then some alternate wire for b_i must be chosen from an unselected output, say o_j'. The net attached to o_j' then has length 1, and the length of the net attached to o_j is 2, so the total wire length must be greater than $2n$.

Thus a satisfying assignment of the variables exists iff the constructed AWC problem has a solution with wire length less than or equal to $2n$. This reduction can be done in polynomial time, so AWC is NP-complete. ∎

Branch and bound techniques can be applied to solve AWC exhaustively. However, for efficiency we propose the following algorithm to solve the AWC problem.

Procedure 2 *(Semi-greedy Algorithm for AWC)*
PHASE I

1. *For each pin with alternate wires, temporarily*

disconnect it from the current net.

2. *For each net form the bounding boxes of the currently connected pins. These partial bounding boxes form a lower bound on the total wire length.*

3. *For each pin with alternate wires, if its pin position is inside one of the partial bounding boxes for its candidate wires (the original wire plus its alternates), assign it to that net. No increase has been caused by this assignment, and hence the partial assignment seen so far must be part of an optimum assignment.*

4. *For each remaining pin with alternate wires, compute the "delta" costs if it is assigned to each of the candidate nets. There is a net assignment which increases the total net length by the least amount. Choose this assignment and update the chosen net.*

5. *Continue step 4 until all pins have been assigned.*

PHASE II

1. *For each pin which is an extreme of the bounding box of its currently assigned net, temporarily release it from its assignment, and compute the best net to put it in and its delta decrease cost in doing this. Note that the delta decrease is non-negative.*

2. *Choose the pin with the maximum delta decrease and reassign the pin to the new net.*

3. *Repeat 1 and 2 until the best delta is 0.*

Notes:

- After PHASE I, there may be pins that can be moved to different nets to improve the total cost.
- After step 2 in PHASE II, the deltas need to be updated efficiently.
- During PHASE II, a pin may be reassigned more than once. To speed up the process, one may want to "lock" a pin once it is reassigned once.
- After PHASE II (with no locking), the solution is locally optimal, in that there is no pin which can be moved to a new net such that the total cost is decreased. However, there might be a set of pins that can be reassigned all at once which decreases the cost.

5 Two Placement Algorithms

If we consider the alternate wire choices for each input pin and optimize for both area and total wire length, we have a two dimensional solution space: physical placement and logical connection. Given a physical placement of the blocks as points in the physical dimension, choosing the best set of logical connections is NP-complete (Section 4). Similarly, given a set of connections, choosing the best placement is also hard. Here we give two approaches to tackling this combined problem.

Throughout this discussion the wire length for a net is estimated by the half perimeter of the bounding box for all terminals connected to the net. Exact pin positions are not considered; pin locations are estimated by the center points of blocks.

5.1 Sequence-Pair Approach

We explored a modified sequence pair-based simulated annealing placement algorithm. A sequence pair is a representation of the physical relations of rectangular blocks placed on a two dimensional plane [7]. Given a sequence pair (Γ_+, Γ_-), the horizontal constraint graph and the vertical constraint graph of the rectangles are implicitly contained in the representation. One of the area-optimal packings under the constraints can thus be obtained in $O(n^2)$ time by applying the longest path algorithm. There are $(n!)^2$ possible different sequence pairs for n blocks. If block orientation is also considered, the solution space is $(n!)^2 2^n$. Thus simulated annealing is generally applied to search the space.

The approach we have experimented with is to give up some optimality in the logical connection space, and heuristically choose a good set of connections for each placement move. The heuristics described in Section 4 can be evaluated in linear time in terms of the number of nets that have alternate wires. This is in the worst case $O(n^2)$ in terms of the number of blocks, which is comparable to the evaluation of the layout for sequence pairs. After the placement converges, one can afford to search for the best set of connections in the other dimension.

Three types of simulated annealing moves are used: swapping two blocks in one sequence, swapping two blocks in both sequences, and 90 degree rotation. A type-one move disturbs the placement significantly, and thus is given higher probability at high temperature. A type-three move is given higher probability at low temperature. For each move, the layout is legalized and minimized for area, and the semi-greedy wire selection is applied to give the new cost. After

convergence, we apply a branch and bound method to solve the AWC problem exactly.

5.2 Mincut Placement Approach

We also examined the use of a mincut placement algorithm [2] to evaluate the placement of a netlist with alternate wires. Our approach differs from traditional mincut placement techniques by using alternate wires to change the cut costs during the recursive bipartitioning of the design. Choosing alternates for wires on a cut net may prevent that particular net from being cut at all, thus reducing the cost. We therefore evaluate the cost of a partition by accounting for such effects. This reduction in cut cost will generally translate to a reduction in wire length for the final placement; alternate choices which prevent nets from being cut during bipartitioning will generally correspond to the selection of shorter local wires.

We modified the FM partitioning algorithm [3] to account for alternate wires. After recursive bipartitioning is applied to a design, partitions are adjoined in a quadrature fashion [2] to obtain a placement. As well, additional wirelength minimization heuristics are used to guide the placement. Both low-temperature simulated annealing and a greedy compaction method are used to further improve the final layout. Finally, the AWC problem is solved for this layout using branch and bound to obtain the wire choices.

6 Experimental Results

We performed three experiments to demonstrate the power of alternate wires. In all cases, our cost function was $area + 2 \cdot wirelength$ where wirelength was measured as the total of the half-perimeter bounding boxes of the nets, and area was the total layout area. Our main objective was wire length, motivated by DSM concerns, with some control on the area.

Experiment I The first experiment was to decompose each example into a set of PLAs as described in Section 3. Table 1 shows the results of this decomposition. The number following the design name is related to the maximum physical width allowed for each PLA in the decomposition [5]. The resulting number of PLAs for each design is shown in the *PLAs* column, and the total number of input pins on these PLAs is shown in the *IPins* column.

We then generated alternate wire sets for each of these examples. The number of pins with alternate wires for each example is shown in the *APins* column under the *Regular* heading (the *Maximum* columns are described in Experiment III below). The percentage (in parentheses) of input pins which have alternate choices is also shown. The average number of alternate choices for each of these pins is shown in the *Alts* column.

We then did the following comparisons:

1. We placed the PLAs without using alternate wires. The total wire lengths for these initial placements (using the two placement methods) are shown in the *Init* column of Tables 2 and 3.

2. We applied the same placement algorithms on the PLAs using alternate wires. The percentage improvement in wire length over the initial placement is shown in the *Reg* column in the two tables.

3. The chosen best wires were returned to logic synthesis and the functionalities of the PLAs were determined according to the wire choices. Another placement was performed using the new PLA areas, and the resulting wire lengths were compared to the initial results. The improvement in wire lengths over the initial placement is shown in the *Resyn* column in the tables.

Experiment II Additionally, when the functionality of the PLAs are finalized, we can do "wire-removal" [9, 6] and place the final netlist. This gives a further reduction in total wire length, shown in the *Final* column in Tables 2 and 3. These numbers are with respect to the *Init* column. To fairly assess the power of alternate wires alone, we performed wire removal on the initial netlist (without alternate wires); the corresponding comparison of wire length gains for the placed designs are shown in the *WR* column, which gives the percentage gain of wire removal versus the initial placement without wire removal.

Experiment III In the above results, there is a fairly high correlation between the improvement in wire length and the percentage of wires that have alternates. Note that the percentage of wires with alternates for the examples is small (on average about 7.5%). As an additional experiment, we wanted to see what would happen if there were more wires with alternates. To this end, we ignored the acyclic constraint when generating alternates. In addition, for each wire, we computed its minimum SPFD [9] and designated another wire as an alternate if its SPFD covered this minimum SPFD. The resulting number of pins with alternate wires and the average number of choices for each of these is shown in the *Maximum* columns of Table 1. This generated only a few more wires with alternates (their average increased to 9%),

although the average number of alternates on wires with at least one alternate increased substantially.

The wire length improvement over the initial placement using these extended sets of alternate wires are shown in the *Max* columns of Tables 2 and 3. This figure loosely indicates an upper bound on the possible improvement due to alternate wires alone, and should be compared to the *Reg* column. As expected, the results obtained correlate with the increased number of wires with alternates.

6.1 Discussion

We also examined the change in total areas of the placed designs when alternate wires are used. For Experiment I (*Resyn* column), the worst-case final placed area increase was 26%. For Experiment II (*Final* column), the worst-case area increase was 30%. However, on average the area increase was less than 3% for each experiment. Thus for the average case the use of alternate wires does not affect the final layout area greatly.

We experimented with different cost functions for sequence pair based placement. For a cost function of the form: $\alpha \cdot area + \beta \cdot wirelength$, we collected placement results for $(\alpha, \beta) = \{(1,0),(1,1),(1,2),(0,1)\}$. As expected, higher α yields better area and higher β yields better wire length. For each combination of α and β, the results with alternate wire choices always give roughly the same gain in wire length. The results shown in Table 2 are that of $(\alpha, \beta) = (1,2)$.

As noted, the gains in wirelength achieved is very much correlated with the percentage of pins which have alternates. When these were increased from 7.5% (*Regular*) to 9% (*Maximum*) in Experiment III, the gain in wire length went from 6.8% to 10.7% for sequence pair, and 6.3% to 9.7% for mincut.

In some cases, there is an increase in wire length when alternate wires are introduced. There are two explanations for this. First, the placement algorithms used are non-deterministic, so random variations will occur. Second, for the mincut placement, the correlation between the cut sizes in the recursive bipartitioning and the final placement wirelengths is not exact. Thus, having the alternate wires affect the cut costs may in fact mislead the mincut algorithm.

The sequence pair approach gives better wirelengths on small examples, while the mincut technique does well for the large examples. The sequence pair approach gave a smaller area overall, though.

		Regular		Maximum	
Design	PLAs/ IPins	APins #(%)	Alts #	APins #(%)	Alts #
alu2-5	18/233	32(13.7)	28.44	37(15.9)	37.43
apex6-5	37/553	21(3.8)	16.10	27(4.9)	81.56
apex7-4	12/157	9(5.7)	22.22	12(7.6)	38.75
apex7-5	11/146	5(3.4)	14.40	6(4.1)	55.83
count-4	6/67	4(6.0)	12.75	4(6.0)	30.25
count-5	6/68	3(4.4)	21.67	3(4.4)	35.00
term1-4	15/186	23(12.4)	19.61	29(15.6)	37.03
term1-5	12/170	11(6.5)	32.55	15(8.8)	44.00
ttt2-4	7/73	7(9.6)	14.00	7(9.6)	15.29
ttt2-5	8/85	9(10.6)	15.22	10(11.8)	18.30
x4-5	24/269	19(7.1)	34.05	28(10.4)	32.64

Table 1: Characterization of Examples

Design	Init	Reg	Resyn	Final	WR	Max
alu2-5	4003.0	18.2	22.8	28.0	8.5	26.4
apex6-5	14120.5	1.0	3.6	3.6	-0.8	1.2
apex7-4	2051.5	3.9	22.5	29.3	15.4	7.5
apex7-5	1810.5	7.1	14.3	14.3	4.7	7.6
count-4	448.0	10.4	9.7	9.7	9.0	10.0
count-5	470.5	4.7	2.7	2.7	2.7	4.7
term1-4	2902.5	12.7	32.5	32.5	18.0	23.2
term1-5	2810.0	5.6	26.3	22.6	16.7	14.6
ttt2-4	609.0	7.0	8.2	8.2	7.2	6.7
ttt2-5	792.5	7.8	3.7	3.7	11.5	7.7
x4-5	4105.0	-3.0	9.3	9.3	5.8	8.5
average	3102.1	6.8	14.1	14.9	9.0	10.7

Table 2: % Wirelength Improvement, Sequence Pair

Design	Init	Reg	Resyn	Final	WR	Max
alu2-5	4048.0	27.7	26.2	22.8	10.9	25.7
apex6-5	11079.0	-0.8	-0.1	4.5	-0.9	3.9
apex7-4	2085.0	4.9	24.7	31.1	14.5	8.4
apex7-5	1801.5	-4.2	7.2	7.2	3.7	5.6
count-4	450.5	10.3	12.5	12.5	9.5	11.7
count-5	482.5	6.0	5.1	5.1	3.2	6.0
term1-4	2901.5	17.0	36.0	36.0	12.7	17.9
term1-5	2825.5	7.2	19.0	21.6	13.3	10.9
ttt2-4	633.0	6.9	8.6	8.6	8.5	6.9
ttt2-5	790.0	1.1	4.7	4.6	6.6	8.3
x4-5	3386.5	-6.4	2.0	-0.4	6.2	1.3
average	2771.2	6.3	13.3	14.0	8.0	9.7

Table 3: % Wirelength Improvement, Mincut

7 Conclusions and Future Work

We presented initial experiments using don't care wires. We used both sequence-pair simulated annealing and mincut partitioning approaches to placement to take advantage of don't care wires. In the future we will look at larger examples, as well as variations on placement, decompositions, and generation of alternate wire sets. However, the results obtained so far are quite encouraging and definitely show an advantage in using don't care wires.

The biggest gain will come from being able to generate more alternate wires. One possibility is to decrease the size of the PLAs. In our current experiments, the PLAs have up to five outputs each. This means that each input furnishes this information to five outputs. Hence a wire can be replaced only if another wire also can furnish all this information. It is as if for each input there are five wires internal to the PLA, and an input can be replaced only if all five internal wires can be replaced by another wire.

Relaxing the acyclic constraint and using the minimum SPFD merits further exploration. The rationale is that although the alternate wires generated are not valid, they contain useful information. In some sense, these are used during placement to obtain an "information" driven placement. Thus a placement is seen to be good if for each node, the information it requires is nearby. Once a good placement is obtained, a node can be implemented by gathering enough of the nearby information to cover the information required. One can imagine possibly increasing the number of inputs to a node but still improving the wiring, since the inputs are local.

We noticed that of our two methods for placement (with total wire length as the dominant factor in the cost), the bipartitioning followed by a low temperature annealing gave the best results on the larger examples compared to pure simulated annealing. This suggests that simulated annealing is not well suited to placement with wire length minimization.

Future experiments include:

1. Using SPFDs to do wire removal using the final placement information. For example, we may want to replace a long wire with a shorter one. The wire removal/replacement done in the present experiments did not do wire replacement using placement information.

2. Since simulated annealing seems to be not well adapted for wirelength as a cost function (we speculate that the underlying cost surface is too jagged), we want to experiment with a new placement method based on nonlinear programming applied to a "smoothed" version of the problem.

3. Although we have confined our studies to decomposition into PLAs, we are not limited to this. Another possibility is to keep the logic nodes as single functions, do a placement of these using alternate wires, and then a technology mapping which takes into account placement information.

We know how to generate alternate wire sets for combinational logic modules, but ultimately we would like to use don't care wires at the chip level. Thus we need to either extend the methods of SPFDs, or explore other methods for generating alternate wire sets for sequential circuits.

Acknowledgements

This research was supported partially by the SRC (under grant number 683), the GSRC/Marco center at Berkeley, and the California micro program with our industrial sponsors, Motorola, Fujitsu, Synopsys, and Cadence.

References

[1] R. Brayton. Understanding SPFDs: A new method for specifying flexibility. In *Workshop Notes, International Workshop on Logic Synthesis*, 1997.

[2] Melvin A. Breuer. Min-cut placement. *Journal of Design Automation and Fault-Tolerant Computing*, 1(4):343–362, October 1977.

[3] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *IEEE Design Automation Conference*, pages 175–181, 1982.

[4] W. Gosti, A. Narayan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Wireplanning in logic synthesis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 26–33, 1998.

[5] S. Khatri, R. Brayton, and A. Sangiovanni-Vincentelli. A VLSI design methodology using a network of PLAs embedded in a regular layout fabric. Technical Report UCB/ERL M99/50, Electronics Research Laboratory, University of California, Berkeley, May 1999.

[6] S. Khatri, S. Sinha, A. Kuehlmann, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. SPFD based wire removal in a network of PLAs. In *International Workshop on Logic Synthesis*, 1999.

[7] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transaction on CAD*, 1996.

[8] R.H.J.M. Otten and R.K. Brayton. Planning for performance. In *Proceedings of the 35th Design Automation Conference*, pages 122–127, 1998.

[9] S. Sinha and R. K. Brayton. Implementation and use of SPFDs. In *Proceedings of the International Conference on Computer-Aided Design*, 1998.

[10] S. Yamashita, H. Sawada, and A. Nagoya. A new method to express functional permissibilities for LUT based FPGAs and its applications. In *Proceedings of the International Conference on Computer-Aided Design*, 1996.