Don't Care Wires in Logical/Physical Design

Philip Chong, Yunjian Jiang, Sunil Khatri, Fan Mo, Subarna Sinha, Robert Brayton
EECS, U.C. Berkeley

March 12, 2000

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 [12, 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 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 multi-level 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**, i.e. can be used independently, just like don't cares in logic synthesis must be able to be used independently. Once an assignment of sources to pins is chosen, based on some wiring optimization, the logic within each block is determined.

We can make the final assignment of sources to pins dependent on the ease with which the resulting netlist can be placed and wired, or a combination of this and some delay constraints. Thus a wireplan is chosen for the blocks, and then the logic functions in the blocks are finalized.

We show how to use SPFDs (Sets of Pairs of Functions to be Distinguished) [12, 1, 9] to find compatible sets of alternate wires. We use this wiring flexibility to construct a wireplan. After some notation in Section 2, Section 3 gives some methods for the initial logic decomposition and the motivation for the particular approach used. Section 4 gives a brief discussion of SPFDs and their relation to information flow, and then gives a method for constructing compatible wire sets. Section 5 poses an assignment problem of sources to pins which minimizes the total wire length, given a placement of modules. We show that this "alternate wire choice" problem is NP-complete and give a heuristic that works well with the problem instances that arise. Section 6 gives two methods for using don't care wires in placement algorithms that we have experimented with so far. Section 7 gives the details and results on some experiments on a few benchmark examples. Section 8 concludes and describes future work that needs to be done in this area.

2 Notation

We focus on the implementation of a combinational logic module. The logic is assumed to have an initial decomposition into a multilevel Boolean network which is a netlist with primary inputs denoted x_i and primary outputs, z_k. The network consists of nodes j with immediate inputs, FI_j, the fanins of j, and a single output. Associated with a node j is a logic function f_j and a Boolean variable y_j whose value is given by $y_j = f_j(y^j)$. The y^j is a vector of inputs, called the fanins of node j. A node k such that there is a path from k to j is said to be in the transitive fanin of j, TFI_j. Similarly, j is said to be in the transitive fanout of k. All nodes for which there is a path from k to that node is the transitive fanout of k, TFO_k. Variables y_j are called intermediate variables. The graph of the network is assumed to be acyclic. The function $f_j(y^j)$ can be rewritten in terms of the primary inputs only, by recursively substituting its intermediate fanin variables by their functions until only primary input variables are left. This gives a function $g_j(x)$ called the global function at node j.

We measure information flowing through a function by the number of input minterm pairs "distinguished" by the function. Thus if m_i and m_k are two minterms in the input space $X = \{x_1, \ldots, x_n\}$, and $g_j(m_i) \neq g_j(m_k)$, we consider this a bit of information provided by g_j. This information must be provided to node j either directly by a primary input or indirectly by one of the fanins from an intermediate function. The specification for the network module can be considered to be the information required at each of the network's primary outputs. If a particular input minterm m_k does not need to be distinguished at any of the outputs from all other minterms, then it is a don't care for the network.

3 Networks of PLAs and Initial Decompositions

Recent work on noiseless fabrics [5] led to a re-examination of the use of multi-level networks where each logic node is implemented as a PLA. This is a general logic synthesis technique, and has been shown to have advantages even for implementations where noise is not a concern. At the same time, we observed that a technique that quantifies the information flowing through a network, called SPFDs, were more effective when given logic nodes that have more logic in them. In some sense the PLAs are similar to the initial application of SPFDs to FPGAs; each node contains a significant logic function, and if that logic function changes, the area requirement for implementing the function does not change much. For FPGAs, the area does not change at all if the number of inputs does not change. For PLAs, the area may change, but typically if the number of inputs does not change we can control the area (we can use bit pairing, folding, etc to keep the area within bounds). PLAs offer some additional advantages in that the layout for each PLA is regular and can be accurately characterized in terms of its electrical characteristics (delay, noise, etc.).

An initial network can be minimized using the usual logic synthesis methods. Then it is decomposed and clustered into PLAs with the target of absorbing as many connections as possible internally in each PLA with the constraint that the resulting PLA network has no cycles. During this clustering no placement information is known, so a heuristic is used 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. Then the result is assessed for being within given bounds in the number of rows and columns (dictated mainly by delay and noise constraints within the PLA).

After clustering, a set of compatible alternates is generated for each local input connection. We use the concepts of SPFDs and information flow to generate alternate wires. These alternates are used in a floorplanning algorithm where during each move the best choice of alternate wires for each local input is used to evaluate the move. Once the final placement and final netlist are chosen, the logic inside a PLA may change and may no longer fit within the row and column bounds; however, we experimentally determined that the PLA areas are usually well controlled in this process. Note that the number of inputs and outputs does not change for a given PLA.

We experimented with the following method as shown in Figure 1: after the usual logic synthesis (we use script.rugged in SIS), we decompose and cluster a network into a set of PLAs of medium size (e.g. 10–15 inputs, 1–5 outputs, 15–25 cubes). These PLAs then are the nodes in our logic network. A set of compatible alternate 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 of source wires 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 a final optimization of the placement.

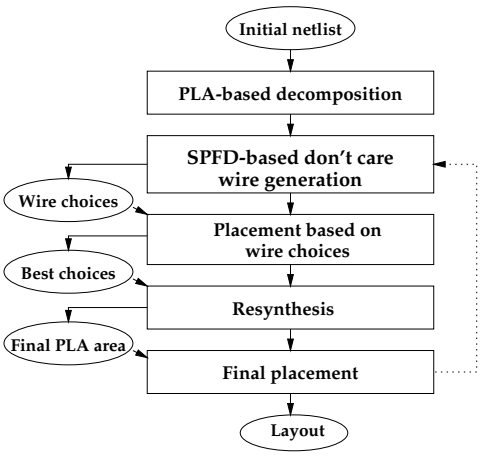

Figure 1: Don't care wire-based logic/physical design flow

4 SPFDs and Compatible Wire Sets

We give an intuitive description of the concept of SPFDs (for a formal discussion, refer to [12, 1, 9]). Consider a multi-level network of combinational nodes. Each fanin of a node provides information to that node. The output of a node has a similar role; it must provide information to 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 fanins covers the pairs of minterms required to be distinguished at the output of that node. An SPFD for an input wire (pin) of a node is a particular choice of pairs of minterms which are assigned to be distinguished by that input. An SPFD is simply the set of all minterm pairs which must be distinguished by a wire or node. It can be represented by a graph whose nodes are minterms and an edge connects two minterms if the function associated with the SPFD (wire or node) must have different values for the two minterms. An SPFD is just a way to represent the information required of a wire or node.

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 for different sources for a pin and has the same sense as don't cares for logic in that we can choose one of several elements from a set for optimizing an implementation. For don't cares the set is the set of values $\{0, 1\}$ and for a pin, we have a set of sources to choose from. Thus we call the set of sources "don't care wires".

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 by each pin is known, the logic functions inside the node (PLA) can be determined. Even then, there is a lot of flexibility for the function, since there are still many logic don't cares that can be used. Further flexibility is available if the SPFD graph is disconnected. For example, suppose there are only 4 minterms in an SPFD graph given by the pairs, $\{(m_1, m_2), (m_3, m_4)\}$. Then a function implementing this can have m_1, m_3 in its onset and m_2, m_4 in its offset or vice versa for the complement, or m_1, m_4 in the onset and m_2, m_3 in the offset or vice versa.

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**, like logic don't cares, 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 union of information coming into a node through its input pins is **always** enough to supply the information required for that node's information output requirements.

Let R_k denote the set of alternate sources for pin k.

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 of sets $R = \{R_k\}$ is* **compatible** *if for each selection, there exist logic functions at each node such that the implied netlist for that selection can implement the specifications at the primary outputs.*

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, assign SPFDs, $SPFD_\eta$ and $SPFD_\sigma$ so that the required information is supplied. Once this is done, each SPFD represents the set of minterms which must be distinguished by that node or pin. In this paper, we will not go into the details of this computation (see [9]) of distributing the output information to the input pins.*

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 initial choice of source for pin 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 1 *For any selection of $\{R_\sigma\}$, the resulting network is acyclic.*

Although the above procedure produces a particular set $\{R_\sigma\}$, any set with the following two properties will suffice for compatibility.

Theorem 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.

Lastly, we would like to illustrate why we chose SPFDs as a technique to generate alternate wires as opposed to previous approaches to find alternate wires based on redundancy addition and/or removal [11, 10]. We believe SPFDs provide more flexibility than any of the previous approaches. Let us consider the following example.

$$\begin{aligned} z_1 &= \overline{g}b + g\overline{b} \\ g &= \overline{a}b + a\overline{b} \\ z_2 &= b + c \end{aligned}$$

Let us find alternates for the wire, (g, z_1). The SPFD of the wire (g, z_1) is given by the set $A = \{(00, 10), (01, 11)\}$. (In the set A, each minterm is of the form gb). Now, if we express the minterms of A in terms of the primary inputs, a and b, we get $A' = \{(00, 10), (11, 01)\}$. (The minterms in A' are of the form ab). Both the edges can be distinguished by the primary input, a. Thus, (a, z_1) can be a compatible alternate for (g, z_1). On the other hand, the techniques in [11, 10] cannot find this alternate. This is because of the presence of the XOR gates. Redundancy removal based techniques work by trying to make a path untestable. This is not possible if there is an XOR (or XNOR) gate along the path because the output of an XOR is sensitive to the both its inputs. SPFDs, on the other hand, look for the actual information content and are not affected by the kinds of gates. Moreover, SPFDs provide more flexibility for implementing the function. Any function that distinguishes all the edges of the SPFD is a suitable implementation. This is a major source of flexibility in SPFDs as we can swap onset and offset minterms in the original function to get many different functions, many of which cannot be obtained by redundancy removal techniques.

5 An Assignment Problem

In our initial experiments, we used total wire length as the cost function to be minimized. Note that indirectly, this controls total area, routability, and power, but not necessarily worst case delay. For example, a significant area increase will result in an increase in total wire length. Evaluating the total wire length of a placement requires that a best selection of alternate wires be made for that placement. 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 pin, find the selection which minimizes the sum of the half perimeters of the bounding boxes of the nets.*

Theorem 2 *The Alternate Wire Choice (AWC) Problem is NP-complete.*

Proof. (see appendix) □

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

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 nonnegative.*

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.

6 Two Placement Algorithms

For this work, we chose total wirelength, measured by the half-perimeter bounding box for each net, as the metric for the final design. By minimizing overall wirelength, we reduce the total wiring utilization for the design, and hence minimize overall congestion.

Note that the half-perimeter bounding box metric is affected by the locations of the pins on the PLAs. However, since we do not have exact pin locations, we estimate these locations by the center points of blocks.

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.

6.1 Mincut Placement Approach

One of our approaches uses a mincut placement algorithm [2] to evaluate the placement of a netlist with alternate wires. Our method 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 wire length minimization heuristics are used to guide the placement.

After mincut partitioning, a low-temperature simulated annealing, based on a sequence pair representation [7], is used to further improve the layout. In the annealing process, after each random move on the sequence pair, the layout is derived and the greedy AWC algorithm is applied to give the best wire length based on the wire choices. Once annealing is done, a greedy compaction method is applied, which evaluates the best location for each cell for minimal wire length. Finally, the AWC problem is solved for this layout using branch and bound to obtain the final wire choices.

6.2 Force-Directed Approach

We also implemented a force-directed placement algorithm. Our force-directed placer is incremental, so the AWC subroutine can be easily embedded. At each step the new position of the cells is computed in terms of the forces acting on the cells, where the forces are generated

from the existing wires attached to each PLA. Then AWC is invoked to determine if better wire choices exist. All input and output ports are fixed on the chip boundary so that no trivial solution (all cells collapse into one single point) will be derived. To overcome cell overlaps, we adopt the algorithm introduced in [13] while some modifications are made to improve speed. The basic idea is to form a density field in the chip area. Cells in this field tend to move towards those areas with lower density and away from areas with higher density.

7 Experimental Results

We performed two experiments to investigate the contribution of alternate wires.

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 II 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: In the results for Experiment I, we see 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 since resynthesis was not done. As expected, the results obtained correlate with the increased number of wires with alternates.

7.1 Discussion

Although not presented in Tables 2 and 3, we also noted the change in total areas of the placed designs when alternate wires are used. For all experiments, the worst-case final placed area increase was 8%. This small increase in area is partly due to our choice of total wirelength as a metric; since we are not actively minimizing area, we expect the final design area to vary somewhat. As well, after selection of alternate wires we must resynthesize the network, and so we may have a change in the PLA areas at this stage.

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 7.5% to 8.7% for the mincut placement, and 5.1% to 5.4% for the force-directed technique.

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 do not guarantee a global minimum, so different local minima can be obtained. Second, using the mincut placement technique, there is no direct relation between the cut sizes in the recursive bipartitioning and the final placement wirelengths. Thus a selection of alternate wires which reduces the cost of a cut may in fact increase the total wirelength. For cases where using alternate wires

Design	PLAs/ IPins	Regular APins #(%)	Regular Alts #	Maximum APins #(%)	Maximum 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	Max
alu2-5	6143.5	19.8	16.8	20.9
apex6-5	18053.5	0.0	3.3	0.0
apex7-4	2843.5	2.2	13.9	13.3
apex7-5	2512.0	6.2	15.5	7.0
count-4	758.0	4.2	8.0	0.0
count-5	849.0	0.0	0.0	0.0
term1-4	4748.0	8.9	34.3	14.2
term1-5	4057.0	4.2	16.4	16.0
ttt2-4	1251.0	22.4	23.1	22.4
ttt2-5	1116.0	14.2	0.0	0.0
x4-5	4590.5	0.0	0.0	0.0
average	4265.6	7.5	12.0	8.7

Table 2: % Wirelength Improvement, Mincut

increases the total wirelength, we ignore the results and instead use the initial placement generated without alternate wires.

8 Conclusions and Future Work

We presented initial experiments using don't care wires. We used a combined sequence-pair simulated annealing and mincut partitioning approach to placement to take advantage of don't care wires. We also modified a force directed placement method which also solves an AWC problem at each step. In the future we will look at larger examples; currently we are constrained by our SPFD implementation which uses BDDs, but a different approach using SAT is now being evaluated and looks promising. We will also experiment with different methods for clustering to get the initial decomposition. For now we have used total wire length as the cost metric, but a more realistic metric would be a combination of this and worst case delay. Finally, the greatest advantage will come from the ability to generate more alternate candidate wires. However, the results obtained so far are interesting and definitely show an advantage in using don't care wires.

To generate more alternate wires, one possibility is

Design	Init	Reg	Resyn	Max
alu2-5	6492.0	6.4	7.6	7.4
apex6-5	22253.0	7.7	1.1	9.9
apex7-4	3097.0	1.8	0.7	3.3
apex7-5	2688.0	9.5	5.7	10.4
count-4	788.0	5.1	4.7	3.9
count-5	823.0	0.8	1.5	1.1
term1-4	5374.0	11.9	2.8	4.2
term1-5	6112.0	0.8	1.5	1.8
ttt2-4	1111.0	3.5	3.1	11.4
ttt2-5	1649.0	5.3	12.4	3.8
x4-5	6148.0	3.4	1.5	1.9
average	5139.5	5.1	3.9	5.4

Table 3: % Wirelength Improvement, Force Directed

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 all 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 in the sense that they contain all the information required, they may contain most of the information. This would lead to a type of *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.

Acknowledgements

This research was supported partially by the SRC, the GSRC/Marco Center at Berkeley, and the California Micro program with our industrial sponsors, Motorola, Fujitsu, Synopsys, and Cadence.

Appendix 1.

Theorem 3 *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. □

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. Chang, K.T. Cheng, N. Woo and M. Marek-Sadowska, "Layout Driven Logic Synthesis for FPGAs", in *Proceedings of Design Automation Conference*, pp. 308–13, June 1994.

[11] L.A. Entera and K.T. Cheng, "Sequential logic optimization by redundancy addition and removal",in *Proceedings of the International Conference on Computer-Aided Design*, pp. 310–15, Nov 1993.

[12] 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.

[13] H. Eisenmann, F.M. Johannes. Generic Global Placement and Floorplanning. In *Proceedings of Design Automation Conference*, pp. 269–274, 1998.