Performance and Wirability Driven Layout for Row-Based FPGAs

SUDIP NAG^a and KAUSHIK ROY^{b,*}

^aXilinx Inc., 2100 Logic Drive, San Jose, CA 95124; ^bElectrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285

(Received 22 February 1994; In final form 10 July 1995)

In FPGAs the routing resources are fixed and their usage is constrained by the location of antifuses. In addition, the antifuses affect the layout performance significantly, depending on the technology. Hence, simplistic placement level assumptions turn out to be grossly inadequate in predicting the timing and wirability behavior of a layout. There is a need, therefore, for a layout technique which changes the layout at placement level based on accurate post-layout timing analysis and net wirability. In this paper we consider such a wirability and performance driven layout flow for row-based FPGAs. Timing information from a post-layout timing analyzer and wirability information from global and channel routers are used by an incremental placer to effectively perturb the placement. A large improvement (up to 29%) in timing has been obtained (compared to non-iterative FPGA layout) for a set of industrial designs and benchmark examples.

Keywords: Placement, routing, wirability, critical path, FPGA, layout

1. INTRODUCTION

The primary requirements of most digital circuit layout tools are ensuring 100% wirability and meeting the timing requirements. A class of such tools partition the layout problem into three distinct steps: placement, global routing and detailed routing. At the placement stage, wirability requirements translate into requirements to place highly-connected blocks closely so as to reduce the size of nets. Efforts are also made to estimate and

reduce excess congestion. To handle timing requirements at the placement level, some placers use pre-placement critical path information [1, 2]. However, much better results can be achieved by dynamically updating the net or path timing criticalities in the course of performing min-cut [3, 4] or other placement techniques [5, 6, 7]. A path is defined as an alternating sequence of logic and nets. And a set of paths are said to be critical if the signal delay through the paths are above a given threshold value provided by the user. At

*Corresponding author.

intermediate stages, path criticalities and net delays are used to determine the slacks on the nets [8, 13, 14]. Other approaches employ linear programming for dynamic path constraint determination [15] while partitioning or formulate the timing-driven placement problem as a quadratic programming problem [16, 17].

At the placement level, wirability requirements are handled by the radiation of net spans denoted by the net bounding boxes and timing requirements are handled by keeping the critical paths short. There are two underlying assumptions for these placement level heuristics:

1. The wirability of a net is a function of its length and the congestion of its routing region.
2. The delay across a path is a function of its length.

In FPGA layout, however, these simple assumptions do not suffice. A sample row-based FPGA architecture [18] is shown in Figure 1. It comprises of rows of logic modules separated by channels. The logic modules could be of different types (input/output, combinational etc.). Horizontal routing resources are available in the form of segments. These can be electrically connected to adjacent segments by programming *horizontal* antifuses. Vertical routing resources are available in the form of segmented feedthroughs running across the channels. If net *A* is connected as shown in Figure 1, four cross antifuses (**C**), two horizontal antifuses (**H**) and one feedthrough antifuse (**F**) have to be programmed to make the connection. Depending on the technology, each of

FIGURE 1 Row-based FPGA architecture.

these antifuses can cause a significant delay. Small segment sizes are desirable for wirability. However, they increase the number of horizontal antifuses on signal paths. Usually, there is a mixture of small and large segments in a channel. The spatial distribution of segments in a channel is known as the *segmentation* of the channel.

In the case of FPGAs, therefore, in addition to the inaccuracies in wirability prediction at the placement level caused by the inability to predict the congestion and signal path accurately, a further inaccuracy is caused by the inability to predict the effect of segmentation. This deteriorates the validity of assumption 1. The validity of assumption 2 also deteriorates since the number of antifuses on a path significantly contributes to the delay across it. It is therefore possible that a short path (length-wise) could be much worse (delay-wise), than a long path mapped onto one segment. It must be noted here, that the general methodology followed by detailed routers for FPGAs is to minimize the horizontal antifuse wastage and segment wastage *i.e.* it tries to match the length of a net to the lengths of the possibly multiple segments to which it is mapped. This causes it to process long nets first assuming all nets have the same criticality. Therefore, the case just mentioned above could arise more often than not.

However, it continues to be true that the ability to change the wirability and timing behavior of a layout is much greater at the placement stage compared to the detailed routing stage where the flexibility to reach new states is limited by the fixed placement and global routing. The need for an incremental placer which changes the layout at the placement stage based on post-layout timing and wirability analysis is therefore imperative. It should, however, be noted that if the placement is changed substantially, the post-layout wirability and timing information cease to be valid. Therefore, only a small incremental change can be allowed at the placement stage.

The paper is organized as follows. Section 2 describes the details of the performance and wirability driven FPGA layout flow. The results

of applying the layout algorithms on some MCNC benchmark examples and some industrial designs are given in Section 4. Finally, the conclusions are given in Section 5.

2. THE FPGA LAYOUT FLOW

The logic modules and the segmented routing channels are laid out in the unprogrammed FPGAs, and hence, the layout problem is different from standard cell or gate array layout. Figure 2 shows our wirability and performance driven FPGA layout flow. The logic circuits can be described using a high level language such as HDL. Logic and/or state machine synthesis (which will not be described here) produce a modified net list description using a technology mapping tool which efficiently maps boolean functions to the proper FPGA logic modules. The modified net list description is used by the placement tool to place the cells in the FPGA base array. Wirability

information from the global and channel routers, and timing information from our post-layout timing analyzer are used by the incremental placer to perturb the placement for performance and wirability. The resultant modified placement is then rerouted. This process continues till a stopping criterion (described later) is met. In the following subsections the layout flow will be described in details.

2.1. Initial Placer

Initial placement is done using modified Timber Wolf [9, 10], a simulated annealing based tool for gate array placement. The FPGA architecture details and constraints are incorporated in the array template. Unlike the gate arrays, the feedthrough cells cannot be inserted for vertical routing. Whenever a pin cannot be reached by a module output through its dedicated vertical tracks, uncommitted feedthrough segments have to be used. This appears as an extra cost in the cost function for placement optimization.

The placer cost function (C) consists of total wire length (W), timing path penalty (P), and extra cost (F) for using uncommitted feedthrough. The complete expression for the cost function is given by

$$C = W + aP + bF \quad (1)$$

where a and b are the relative weights (determined empirically) of the three terms in the cost function. The wire length of a net is estimated as half the perimeter of the minimum rectangle, a bounding box, that encompasses the net. For each critical timing path, an upper bound is put on the wire length of all the nets in the path. A penalty is assigned for a path that has the wire length beyond this upper bound.

2.2. Global Router

With predefined vertical routing resources in FPGAs, a global router is constrained to utilize

FIGURE 2 Our FPGA layout flow.

only the available routing segments. Global routing consists of two passes [11]. In the first pass, channel congestion is estimated and the nearest neighbor placement refinements are performed for minimizing net length and assigning nets to the less congested channels. In the second pass, the uncommitted feedthroughs are assigned with channel congestion considered. For each net, the pins are grouped into partitions. In each partition, all pins can be connected without using uncommitted feedthroughs. A minimum length of feedthrough is chosen to connect between partitions. With all partitions of the net connected, a minimum spanning tree (MST) is constructed for all pins of the net. The construction of MST also takes channel congestion into consideration. The MST specifies pins that are to be connected in a channel. A group number is assigned to these pins for the channel router.

2.3. Channel Router

The channel routing problem is formulated as an assignment problem where each net within a channel is assigned to one or more unassigned segments. Each net is allowed to use at most one track due to a technology constraint which does not allow programming of antifuses connected in an *L*-shaped fashion. The programming of such antifuses can lead to programming two antifuses at the same time which can degrade the performance of the programmed antifuses.

The cost of routing is determined by the number of segments used by the critical nets in a channel, and the length of the segments assigned to different nets. The number (H_x) of horizontal antifuses to be programmed is one less than the number of segments assigned to the corresponding net (x). The use of these antifuses increases the critical path delays of the nets. If a net x of length L_x (net length within a horizontal channel) is routed with p ($1 \leq p \leq K$, maximum of K segments allowed) segments, each of length L_j ($j = 1, \dots, p$), then $[(\sum_{j=1}^p L_j) - L_x]$ gives a measure of the unused

segment (s) length. For K -segment routing we define the cost C of routing a net x as

$$C = w_1 \cdot \alpha + w_2 \cdot \beta$$

where

$$\alpha = \frac{\sum_{j=1}^p L_j - L_x}{\sum_{j=1}^p L_j}$$

$$\beta = \frac{H_x}{K + 1}$$

where α and β are penalties for segment length wastage and *hfuse* usage, respectively, and are both positive and less than 1. The weights w_1 , w_2 assigned to the wastage factor and the antifuse usage factor respectively, are determined by the technology under consideration. For example, the metal-metal horizontal antifuse has a much lower programmed resistance than a programmed polymetal antifuse, and hence, w_2 for the latter technology should be higher than the metal-metal antifuse technology. Usually, w_2 is greater than w_1 . Values for w_1 and w_2 are determined experimentally based on the technology used.

The routing algorithm uses a greedy search to assign nets to the unassigned segment (s). The nets are ordered in increasing order of length. Assuming that the longer nets are more critical, they are routed first. Therefore, the longer nets can be assigned to single segments and with least segment wastage. Once a net has been assigned to one or more segments, the corresponding segments are unavailable for further routing, decreasing the problem size for the subsequent nets.

2.4. Delay Generator

Once the channel routing is done, the complete layout knowledge enables accurate delays to be generated for all interconnections. These interconnect delays are then used by our timing analyzer to determine the critical paths. For the FPGA interconnects, the three primary sources of delays are horizontal antifuses, cross antifuses and

the usual delay due to routing in metal. The first step towards generating the interconnect delays is transforming the nets to RC-networks [12]. We represent the RC-network as a graph with the nodes being cell pins. Each net segment and the horizontal and the cross antifuses are modeled by a lumped RC delay. For a particular net spanning channels Ch_a to Ch_b , initially all pins including feedthroughs and equivalent pins are given a unique node. Then, the nodes corresponding to feedthroughs are collapsed. Finally additional nodes are added where required to account for the presence of horizontal antifuses. Elmore delay calculations [12] are then made by propagating information from the driver to the pins using breadth-first search on the graph. Figure 3 shows a net having a driver pin (Y) and two other input pins $p1$ and $p2$. Routing of this net requires programming three cross antifuses and an horizontal antifuse. The RC tree generated out of this net is shown adjacent to it. Programmed cross antifuses are modeled by a resistance R_c . Similarly,

the programmed horizontal antifuses are modeled by a resistance R_h . Rl_i and Cl_i respectively represent the lumped resistance and capacitance of the metal lines which also include any capacitance due to unprogrammed antifuses on the metal lines. Figure 3 also shows how Elmore's first order delay calculations are made from driver pin Y to pin $p2$.

2.5. Timing Analyzer

Once the interconnect delays have been estimated, we generate a set of critical paths having delays above a threshold value using a timing analyzer that we developed. The threshold can be determined to be a certain percentage of the maximum delay path through the circuit, or can be user specified.

The timing analyzer assumes that the circuit is represented by a directed acyclic graph where each node represents a combinational cell (or logic). The sequential cells such as the flip-flops are

$$\text{Elmore Delay (P2)} = \ln(2)[R11 * (Cl1 + Cl2 + Ci + Cl3 + Cl4) + Rc * (Cl2 + Ci + Cl3 + Cl4 + Ci) + \dots + R15 * Ci]$$

FIGURE 3 Delay Generation Process.

removed from the circuit and are replaced by their corresponding inputs and outputs. These inputs and outputs act as *pseudo* primary inputs and *pseudo* primary outputs. The directed acyclic graph is leveled such that the *level* of a node v is given by $\lceil \max(\text{levels of fanin nodes of } v) \rceil + 1$. Each input, primary or pseudo, is assigned a level of 0. Let us consider Figure 4. There are two primary inputs A and F having level of 0. Nodes B , E , and G have a level of 1; C and H are assigned a level of 1; C and H are assigned a level of 2; and D gets level 3. The longest delay $dist_j(v)$ from each node v to each primary output j is calculated. At each node v , $dist_j(v)$ is initialized to 0. The directed graph is traversed from each output j to its fanins until a primary input or a pseudo input is reached. At each node v , $dist_j(v)$ is the maximum of $dist_j(v)$ already present at node v and the newly calculated one. The newly calculated $dist_j(v)$ is given by $[dist_j(v') + weight_{v,v'} + Node\ Delay(v)]$ where v' is the node from where v is reached, and is an immediate fanout of node v . The interconnect delay between node v and v' is given by $weight_{v,v'}$. For example, Figure 4 shows a directed acyclic graph where each edge between v and w has a weight which represents the interconnect delay. If all node delays are neglected then $dist_D(A)$ is 7 and $dist_D(B)$ is 3.

At each node v its fanouts are sorted in decreasing order of $[weight_{v,v_1}] + dist_j(v_1)$, which indicates the length of the longest path from v via

FIGURE 4 Timing Analysis Process.

edge $v - v_1$. Sorting takes $O(m \log(m))$ time where m is the number of fanouts from v . The longest path through the graph can be easily obtained by traversing the graph from input to output. From each node v visit only those nodes at the top of the fanout list of each node v . Continue this process until the primary output is reached. The maximum delay path, MAX_d , through the graph is the maximum value of $dist_j(v)$ over all input nodes v and for all outputs j . The user defined threshold can be defined to be $T = \theta * MAX_d$, where $0 < \theta < 1$.

The graph is now traversed from input to the output in a breadth first manner to obtain all paths above threshold value T . All partial paths up to node v are stored in node v . A maximum of N most critical paths are stored at each node. Let us assume that all partial paths up to node v have been generated. The fanout nodes of v are included in the set of critical paths if the *partial path delays up to node v plus $dist_j(v)$* is greater than T . It can be observed that the search can be pruned because all the fanout nodes of v are sorted in order of $dist_j$. For Figure 4, if $\theta = 0.9$, the set of critical paths are $A-E-D$, $A-E-C-D$, $F-E-D$, and $F-E-C-D$, each of length 7 units.

The complexity of the algorithm is $O(m \log(m) + n(k))$, where m is the number of edges and $n(k)$ is the number of nodes on k paths which exceed the given threshold limit [19].

The net criticalities are generated from the interconnect delays, such that each net is assigned a value between 1 and p , proportional to the net delay. Net delay is defined as the maximum delay from the driver pin to any input pin comprising that net.

3. THE INCREMENTAL PLACER

In general, it is non-trivial to change a placement incrementally. This invariably results in cell overlaps, the removal of which might cause a change in a substantial portion of the chip. However, for structured layouts, like gate-arrays and FPGAs,

an incremental change in the placement can be brought about by swapping or moving modules to valid locations. This is exactly the method followed by our incremental placer. In order to arrive at a linear-time algorithm, we first select a set of η critical cells. Then, for each of these, the best swapping candidate is selected. Finally, the best of the η swap pairs determines the swap. This process is repeated a few times during the incremental placement phase. This is in spirit similar to the procedure for selecting the best set of swaps in a Kernighan-Lin style bipartitioning [20]. The rest of this section describes the components of the incremental placer in detail.

3.1. Information Handling

The global and the channel routers provide the wirability information in terms of two m by n matrices—the *vertical wiring matrix* (V) and the *horizontal wiring matrix* (H), as shown in Figure 5. m is the number of channels and n is the number of zones per channel (each channel is divided into n zones of widths like that of logic modules). H_{ij} represents the number of segments assigned to nets in zone j of channel i . Similarly, V_{ij} represents the number of feedthroughs assigned to nets in zone j of channel i . For the example of Figure 5, there are

three channels and three zones. The middle channel has only one feedthrough in zone 2, while there is a segment assigned to net A in all three zones of the middle channel (shown by highlighted lines). Therefore, the second row of matrix H has all 1's.

The timing analyzer provides the path criticality of the k most critical paths, and the net criticalities based on maximum driver-to-sink delay (α_{ds}) for nets.

3.2. Selecting Critical Cells

The criterion for selecting the η most critical cells is their predicted ability to cause a substantial change. The cell criticality is governed by the following equation for every cell P :

$$PC = \omega_{pc} \times Pcr + \omega_{nc} \times Ncr + \omega_{wc} \times Wcr + \omega_p \times Per$$

Pcr is the maximum criticality of the sub-set of k critical paths of which P is a constituent. If P does not happen to be in any of the k critical paths, Pcr of P would be zero. This causes the cells on the critical paths to be likely candidates for swapping.

Ncr of P is the summation of the net-criticalities of all nets connected to it. This is similar to the first term in its objective of handling timing issues.

FIGURE 5 Wirability information.

However, Ncr addresses timing at a coarser and more global level than the path specific term. Also, among the constituent nets of the most critical path, the resizing of a few key nets may contribute more than resizing the remaining nets of the path. This distinction is also achieved by the second term.

Wcr of P determines the ability of P to effect a change in wirability. Essentially, if a cell happens to be in a congested area, its movement has very high probability of improving the wirability. Wcr is measured by adding the Nw (wirability criticality) of nets connected to cell P . Nw of a net comprises two parts, HNw and VNw . HNw of net N is found by mapping its bounding box onto the horizontal wiring matrix described earlier, and then adding the matrix element values corresponding to the bounding box. This gives a measure of how congested the net bounding box region is in terms of horizontal wiring. VNw is determined similarly by using the vertical wiring matrix. However, since the driver port of a logic module can span a few channels up and down, and the global router utilizes these hardwired spans as vertical routing resources, the effective bounding box used is smaller than the bounding box of the net itself by the driver span amount.

Per of P determines how perturbed the area around P is. If a cell or nets connected to it are moved by a large amount, all the timing and wirability data concerning that cell and its connected nets cease to be accurate. Therefore, the intent is to try to move cells from an undisturbed area, if possible. Per of P is determined by the summation of the $Nper$ of the nets connected to P . $Nper$ of a net N comprises two parts: wirability related perturbation and timing related perturbation. Wirability related perturbation is a function of the change in the bounding box of a net from its original bounding box; timing related perturbation is determined as a function of the change in the maximum driver-to-sink distance (α_{ds}) for net N .

The symbols ω_{pc} , ω_{nc} , ω_{wc} and ω_p represent the weights associated with the four different criticality measures and were determined empirically.

3.3. Selecting Best Move

For every critical cell, the best swap is found by explicitly trying out all the feasible alternatives, i.e., template locations, either vacant or occupied, whose type matches that of the critical cell's template location. The swapping cost is determined according to the following:

$$\delta C = \omega_{pc} \times \delta Pcr + \omega_{nc} \times \delta Ncr + \omega_{wc} \times \delta Wcr$$

δPcr determines the expected improvement in the maximum critical path delay as a result of the move. Hence, if the cell moved or the cells swapped do not form a part of the most critical path, then this term would be zero. For a path X , the new delay is approximated using the old delay and the ratio of the new path length and the old path length. For the case where a target clock speed is specified, once the target speed is met, further reduction is not required. However, we still have to concentrate on other optimization terms like wirability.

δNcr also measures the improvement in timing as a result of the swap/move, as does the term δPcr . However, it handles this issue at a coarser and more global level using the timing criticalities of nets. The new timing criticality of a net is derived from its old one using the ratio of its new α_{ds} and its old α_{ds} . This has the effect of reducing the delay across more critical nets.

δWcr measures the effect of the swap/move on wirability. The mechanism of finding wirability of a net is the same as outlined earlier. The old and new bounding boxes are mapped onto the wirability matrices. The improvement in wirability is just the difference between the new and old wirability.

3.4. Updating the Information

As a result of the acceptance of a move/swap, the timing and the wirability information have to be updated. However, as mentioned before, since timing and wirability information are intricately

related to the way ports get finally connected at the detailed routing stage, only a heuristic guess can be made for updating them. However, assuming that more than one move needs to be done at the incremental placement level, it is imperative that such updates be done in order to capture at least the global effects for later moves. An example of such a global effect, observable even at the placement stage, is the case when a substantial reduction in the length of one element of and extremely critical path results in that path ceasing to be as critical.

As a result of a move/swap, a set of net bounding boxes change. These changes are used to update the horizontal and vertical wirability matrices. The path delays are updated by using the ratios of old and new path lengths. The net criticalities are updated by the ratio of the old and new α_{ds} of the corresponding net.

3.5. Number of Moves

In every iteration, a certain number of swaps/moves are made by the incremental placer. This number should be small in order that the wirability and the timing information should continue to provide useful information. If too few moves are made at the incremental placer, then a large number of iterations across the flow of Figure 2 will be required to get a substantial improvement. On the other hand, if too many moves are made, the wirability and timing information cease to be accurate resulting in the possibility of later moves *looking* good but being detrimental. To avoid the problems of these two extremes, we adopt a dynamic determination approach. In the first iteration, the number of swaps/moves is set to a small user-defined percentage (default 5%) of the total number of cells. If it is found that the performance (found after routing and timing analysis) improves then the maximum number of swaps/moves for the second iteration is increased by a user-defined percentage (default set to 10%) and if it degrades, then it is reduced by the same.

3.6. Stopping Criterion

It can be observed that the critical path delay through a circuit may increase slightly after an iteration. This is partly due to the fact that placement moves out of local minimas to move into another minima, and also partly due to the fact that though the incremental placer moves the blocks with the timing and wirability information, however, it is not absolutely coupled with the global and channel routers. Hence, the best result is stored. The number of iterations through the loop can be user defined and can be used as a stopping criteria.

4. IMPLEMENTATION AND RESULTS

The performance and wirability driven FPGA layout algorithms were implemented in C on an Apollo 425 workstation. We experimented with three designs presently used in the industry and two MCNC benchmark examples. Texas Instruments' TPC1010 logic modules with corresponding channel segmentation scheme were used in our study. The logic netlist, optimized from logic synthesis, was converted to layout netlist using pinmap information for each macro. A predefined, two-dimensional layout array template that consists of rows of logic modules is used as the placement template. The TPC1010 series of FPGAs have a 44×8 template size (8 rows, each having 44 logic modules). However, the template size was varied to fit the larger designs. Table I shows the result of applying our layout scheme to some examples. *bw* and *duke2* are the two MCNC synthesis benchmarks and the rest are industrial designs. Column 2 shows the template size. The number of blocks/nets for each design are shown next. It can be observed that the number of blocks is equal to the number of nets because each block has exactly one output. A net consists of a driver pin (output) and a set of input pins connected together. The interconnect delays were obtained by modeling the nets and antifuses as RC trees as

TABLE I Results for MCNC benchmark and industrial design examples with 1ns block delay

design name	template size	#blocks/ nets	Initial layout		Final layout		% improvement in delay
			hfuses	%wastage	hfuses	%wastage	
f104667	44×8	262/262	16	48.3	18	46.9	29.12
bw	44×8	144/144	6	45.3	6	45.6	3.57
duke2	44×8	318/318	12	44.9	12	43.7	23.05
f104780	44×18	671/671	29	44.3	27	45.7	23.91
cf92382a	44×18	668/668	28	52.2	26	49.8	19.97

shown before. For Table I, a logic module delay of 1ns (which is approximately the TPC1010 logic module delay) was used. For such a case, the interconnect delays were dominant. Columns 4 through 7 show the layout statistics (the total number of horizontal antifuses *hfuses* used and the total segment wastage (Section 2.3) over all the channels) for both initial and final layout. Initial layout is defined as the layout obtained after one pass through the layout flow of Figure 2. The percentage improvement in critical path delays are shown in the last column. The improvements were compared with the initial FPGA layout. Up to

29% improvement in critical path delays were obtained. The improvement was small for example *bw* because of its smaller size. For all designs we obtained good results within 6 iterations.

Table II and Table III show channel by channel layout results for two examples-*f104667* and *duke2*. The number of nets in each channel, the number of horizontal antifuses (*hfuses*), and the percentage segment wastage in each channel are shown for both initial and final layouts. The incremental placer moves a small number of logic blocks depending on the present layout information. Hence, the number of nets present in each

TABLE II Detailed layout results for f104667

channel #	Initial layout			Final layout		
	nets	hfuses	% seg. wastage	nets	hfuses	%seg. wastage
1	22	1	41.3	25	2	39.1
2	30	1	48.1	32	1	51.7
3	50	1	45.1	50	1	45.3
4	55	1	47.6	56	1	46.7
5	66	3	52.8	69	4	54.8
6	61	4	52.2	60	5	52.2
7	40	2	49.5	39	0	49.2
8	40	2	46.9	39	3	48.5
9	19	1	21.9	21	1	41.7

TABLE III Detailed layout results for duke2

channel #	Initial layout			Final layout		
	nets	hfuses	% seg. wastage	nets	hfuses	%seg. wastage
1	25	1	40.0	28	0	37.1
2	46	1	45.3	45	1	43.7
3	68	0	41.5	68	0	41.5
4	85	3	45.1	85	2	44.7
5	77	2	51.9	77	2	47.1
6	61	3	43.7	63	4	46.8
7	49	0	39.4	52	1	39.7
8	53	1	44.9	56	0	43.7
9	32	1	47.1	34	2	38.1

TABLE IV Results for MCNC benchmark and industrial design examples with 3ns block delay

design name	Initial layout		Final layout		% improvement in delay
	hfuses	% wastage	hfuses	%wastage	
f104667	16	48.3	18	49.9	12.35
bw	6	45.3	—	—	—
duke2	12	44.9	11	45.2	14.42
f104780	29	44.9	30	45.0	5.83
cf92382a	28	52.2	29	50.8	10.17

channel after the initial and the final layouts were different. Though the initial and final layout results of Table II and Table III look very similar, the incremental placer was able to move blocks so that the channel congestion in different regions were minimized, and the critical path delays were reduced.

If the module delays increase, a different set of paths might become critical. And when module delays start dominating the critical path delays a large improvement in critical path delays may not be achievable. However, routability or wirability improvements can still be achieved. Table IV shows the results of our experiments with a larger block delay of 3ns. The percentage improvements were lower in this case because of larger block delays. An interesting observation is the relatively small improvement for example *bw*. This is because of its small size compared to the template: when lots of resources are available, there is not much to be optimized anyway.

The wirability considerations for the layout flow were experimented by using lesser number of tracks such that one or two nets were not routable for every design. In all the cases, the layout tool was able to perturb the placement so that routability was achieved within few iterations.

5. CONCLUSIONS

Fixed routing resources and the presence of antifuse severely deteriorate the validity of placement level predictions of wirability and timing in the case of FPGAs. However, the ability to effect a change in these behaviors continues to be much

more at the placement level compared to the routing level. There is a need, therefore, for a flow where the post-layout information is effectively used to change the placement. Such an iterative wirability and performance improvement scheme has been developed for row-based FPGAs.

A likely problem with such an iterative improvement approach could be convergence. Specifically, the algorithm could keep on switching between good and bad states. If the amount of layout change in an iteration (determined by number of moves in each iteration before accurate evaluation at the end of iteration) is large, then the chances of convergence problem increases. At the same time, making very few moves at each iteration will result in minimal improvement of layout. In order to tackle these two competing issues, we use a dynamic mechanism to determine amount of layout change in an iteration as discussed before. Nonetheless, convergence can still be a limitation of this approach.

Acknowledgment

The authors would like to thank Prof. Rob A. Rutenbar for providing many useful suggestions. The authors would also like to thank Mahesh Mehendale, Raghu, Dan Edmondson, Ching-Hao Shaw, and Ashwin Shah of Texas Instruments for many interesting discussions and constant encouragement.

References

- [1] Terai, M., Takahashi, K. and Sato, K. (1990). "A New Min-Cut Placement Algorithm for Timing Assurance Layout Design Meeting Net Length Constraint," *Proc. of 27th Design Automation Conference*, pp. 96–102.

- [2] Lin, I. and Du, D. H. C. (1990). "Performance-Driven Constructive Placement," *Proc. of 27th Design Automation Conference*, pp. 103–106.
- [3] Garbers, B., Korte, B., Promel, H. J., Schwietzke, E. and Steger, A. (1990). "VLSI-Placement Based on Routing and Timing Information," *Proc. of ICCAD*, pp. 317–321.
- [4] Marek-Sadowska, M. and Lin, S. P. (1989). "Timing Driven Placement," *Proc. of ICCAD*, pp. 94–97.
- [5] Sutanthavibul, S. and Shragowitz, E. (1991). "Dynamic Prediction of Critical paths and Nets for Constructive Timing-driven Placement," *Proc. of 28th Design Automation Conference*, pp. 632–635.
- [6] Donath, W. (1990). "Timing Driven Placement Using Complete Path Delays," *Proc. of 27th Design Automation Conference*, pp. 84–89.
- [7] Sutanthavibul, S. and Shragowitz, E. (1990). "An Adaptive Timing-Driven Layout for High Speed VLSI," *Proc. of 27th Design Automation Conference*, pp. 90–95.
- [8] Hauge, P. S., Nair, R. and Yoffa, E. J. "Circuit Placement for Predictable Performance," *Proc. of ICCAD*, pp. 88–91.
- [9] Sechen, C. and Sangiovanni-Vincentelli, A. (1984). "The TimberWolf Placement and Routing Package," *IEEE Custom Integrated Circuits Conf.*
- [10] Shaw, C., Mehendale, M., Edmondson, D., Roy, K., Rao, R., Wilmoth, D., Harward, M., Landers, R. and Shah, A. (1992). "An FPGA Architecture Evaluation Framework," *FPGA-92 Workshop*, Berkeley, pp. 15–19.
- [11] Rao, R. (1992). "A Global Router for Channelled FPGAs," *MCNC Logic Synthesis Workshop*, pp. 119–129.
- [12] Elmore, W. (1948). "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *Journal of Applied Physics*, pp. 55–63.
- [13] Youssef, H. and Shragowitz, E. (1990). "Timing Constraints for Correct Performance," *Proc. of ICCAD*, pp. 24–27.
- [14] Frankle, J. (1992). "Iterative and Adaptive Slack Allocation for Performance-Driven Layout and FPGA Routing," *Proc. of 29th Design Automation Conference*, pp. 536–542.
- [15] Jackson, M. A. B. and Kuh, E. S. (1989). "Performance-Driven Placement of Cell Based IC's," *Proc. of 26th Design Automation Conference*, pp. 370–375.
- [16] Srinivasan, A. (1991). "An Algorithm for Performance-Driven Initial Placement of Small-Cell ICs," *Proc. of 28th Design Automation Conference*, pp. 636–639.
- [17] Jackson, M. A. B., Srinivasan, A. and Kuh, E. S. "A Fast Algorithm for Performance-Driven Placement," *Proc. of ICCAD*, pp. 328–331.
- [18] Gammal, A. E. *et al.* (1989). "An Architecture for Electrically Configurable Gate Array," *IEEE Journal of Solid State Circuits*, **24**(2), 394–398.
- [19] Roy, K. and Abraham, J. (1991). "The Use of RTL Descriptions in Accurate Timing Verification and Test Generation," *IEEE Journal of Solid State Circuits*.
- [20] Kernighan, B. and Lin, S. "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Sys Tech Journal*, **49**(2), 291–307, 2/70.

Authors' Biographies

Sudip Nag received the B.Tech., degree in electronics and electrical communications engineering from Indian Institute of Technology, Kharagpur, India in 1987 and M.S., and Ph.D., degrees in electrical engineering from Carnegie Mellon University in 1992 and 1995 respectively. He worked at Texas Instruments (India) from 1987–1990. Currently he is working at Xilinx Inc., San Jose. His research interests are in the general area of CAD development tools for VLSI circuits, including layout, timing analysis and FPGA architecture synthesis.

Kaushik Roy received his B.Tech., degree in electronics and electrical communications engineering from the Indian Institute of Technology, Kharagpur, in 1983, and his Ph.D., from the Electrical and Computer Engineering department of the University of Illinois at Urbana-Champaign in 1990. He was with the Semiconductor Process and Design Center of Texas Instruments, Dallas, from 1988, where he worked on FPGA architecture development and low power circuit design. He joined the electrical engineering faculty at Purdue University in 1993, where he is currently an Associate Professor. He was a guest-editor for the special issue on low-power VLSI in the December 1994 issue of *IEEE Design and Test of Computers*. He is Associate Editors for IEEE Trans. on Circuits & Systems and IEEE Design and Test of Computers. He is in the program committee of several conferences. Kaushik is a recipient of 1995 NSF CAREER development award. His current research interests are in low-power VLSI design, VLSI testing and fault tolerance, and FPGA's.