Performance-Driven Simultaneous Place and Route for Row-Based FPGAs

Sudip K. Nag and Rob A. Rutenbar

Department of Electrical and Computer Engineering,

Carnegie Mellon University,

Pittsburgh, PA -15213

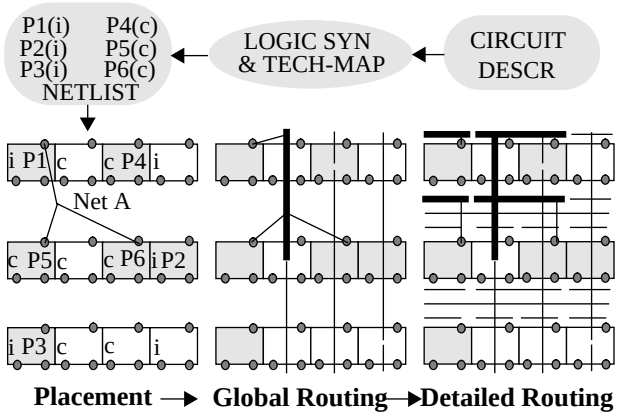
Abstract

Sequential place and route tools for FPGAs are inherently weak at addressing both wirability and timing optimizations. This is primarily due to the difficulty in predicting these at the placement level. A new performance-driven simultaneous placement / routing technique has been developed for row-based designs. Up to 28% improvements in timing and 33% in wirability have been achieved over a traditional sequential place and route system in use at Texas Instruments for several MCNC benchmark examples.

1 Introduction

Field Programmable Gate Arrays provide a means of drastically reducing the turn-around time for digital ICs, with a relatively small degradation in performance (*e.g.*, maximum achievable clock speed). For a variety of application-specific integrated circuit (ASIC) applications, where time-to-market is most critical and the performance requirements do not mandate a custom or semi-custom approach, FPGAs are an increasingly popular alternative. This has prompted a substantial amount of specialized synthesis and layout research focused on maximizing density, minimizing delay, and minimizing design time. The style of FPGA that this research targets is the row-based style (*e.g.*, ACTEL-style parts [1]).

Figure 1. Layout Flow for Row-Based FPGAs



Row-based FPGAs comprise rows of logic modules separated by channels. Each logic module can be configured to perform a variety of functions. Horizontal routing resources are available in the form of segments in channels. Adjacent segments can be connected by programming the *horizontal antifuse* between them. Ports of logic modules can be connected to segments by programming a *cross antifuse*. Vertical routing resources are similarly available for wires that span channels; these paths may themselves be segmented. Depending on the technology, the antifuses can cause a nonnegligible delay. Small segment sizes are desirable for wirability because they maximize segment usage. However, this tends to increase the number of antifuses on each signal path, which is detrimental for timing. Hence, there is usually a mix of small and large segments. The spatial distribution of segments in a channel is known as the *segmentation* of the channel.

The design flow for a typical row-based FPGA is shown in Figure 1. Logic synthesis and technology mapping tools convert a high level circuit description into a net-list of FPGA logic block sized cells [9]. These blocks may be of different types, *e.g.*, I/O blocks ("i" blocks), combinational logic blocks ("c" blocks), *etc*. A placer maps these cells onto valid module locations. Because accurate routing information is absent during placement optimization, typical placers optimize based on estimated net-length and congestion criteria, using formulations essentially similar to row-based semi-custom standard cell layouts [6]. A global router then assigns vertical feedthroughs to nets which must span multiple channels [7]. In Figure 1., for example, net A needs a feedthrough which will be assigned at the global routing stage. Once the global routing is done, the channel problems are defined. A detailed router then allocates horizontal routing resources, assigning specific segments to each net in each channel [8].

The problem with this fairly traditional layout flow is the increasing need to achieve two incompatible optimizations: *dense routable* placements which meet aggressive *timing* requirements. Failure to pack a single design onto the smallest feasible FPGA carries a substantial cost penalty: the cost of moving to a larger FPGA and then only sparsely populating it. Unfortunately, optimizing for 100% wirability is often at odds with optimization for speed. Critical paths must be respected and the cells and nets which define these paths must be given priority during placement. Given the extremely granular, rigid nature of the routing resources here, optimizations for delay minimization are very difficult to estimate during placement, yet an overly conservative estimate may compromise overall routability.

We suggest that the core of this problem is the assumption that FPGA placement, global routing, and detailed routing under delay constraints must be *separate, sequential* steps. We suggest that the more reasonable approach is *simultaneous* placement, global routing and detailed routing. Of course, in general, such a strategy is likely

to be computationally unaffordable. However, the same granular/rigid problem structure that makes difficult the creation of the "downstream" estimators necessary in a sequential place-then-route scheme is a positive advantage in a simultaneous place/route scheme: it limits the possibilities that must be searched. Indeed, row-based FPGAs are an ideal problem domain for an aggressive combinatorial formulation in which *all* the design variables are manipulated simultaneously. FPGAs are of modest size (thousands of cells and nets) with an extremely limited palette of geometric alternatives for any given cell or wire. With appropriate algorithm design, all of these difficulties can be turned to our advantage.

This paper presents a simultaneous placement, global routing, detailed routing algorithm for row-based designs which maximizes density and minimizes worst-case critical path delay. We present an optimization-based formulation which uses simulated annealing, combined with constructive global routing and detailed routing "in the loop" to achieve this. The paper is organized as follows. Section 2 discusses in somewhat more detail the FPGA design concerns to which we have just alluded, and summarizes relevant previous layout approaches specific to FPGAs. Section 3 then presents our simultaneous placement and routing approach in detail. Section 4 presents results on some MCNC benchmark examples, and compares these with results from production FPGA layout tools in use at Texas Instruments. Finally, Section 5 offers concluding remarks.

2 FPGA Design Concerns and Approaches

2.1 Design Concerns

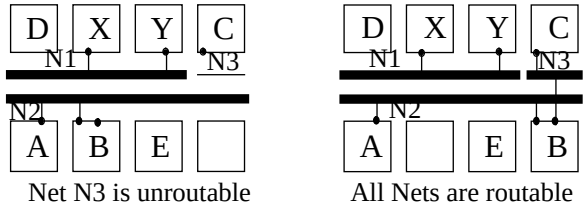
We can broadly summarize the three areas of concern in FPGA layout to be:

- **Routability**: fixed routing resources complicate our ability to achieve 100% automatic wiring, especially if we try to pack the largest possible design onto a particular (fixed size) FPGA.
- **Delay:** wiring paths necessarily pass through several antifuses, accruing delay (via loading) with each antifuse; optimizing delay is often at odds with optimizing routability.
- **Leverage:** by which we mean the ability of any particular step in the traditional sequential layout flow to impact the routability and delay characteristics of the final placed/routed FPGA.

We discuss these briefly, below.

For semi-custom row-based standard cell designs, flexibility in terms of variable channel height can be used to alleviate routability problems. In contrast, row-based FPGAs, with their rigidly proscribed channel structure (tracks and track segmentation) see no such flexibility. For example, antifuse locations usually allow only adjacent segments on the same track to be connected; this constrains each connection to be routed on a single track in its passage through one channel. Wirability predictions at the placement level based on net-length and estimated congestion are thus especially prone to error for FPGAs. The problem is that these rigid routing resources and their fine-grain connectivity constraints are *invisible* at the placement level. This is illustrated in Figure 2. with a small example comprising 7 cells (X, Y, A, B, C, D and E), 3 routing segments, and 3 nets (N1 connecting X and Y, N2 connecting A and B, and N3 connecting B and C). The total net-length and congestion is less for the placement on the left. Nevertheless, the placement on the left is actually unroutable due to the segmentation, whereas the alternative can be wired.

Figure 2. Length and Congestion-based Predictor Inaccuracy



The timing problem involves the identification and minimization of critical path delay. If interconnect delays could be ignored, the path delays could be estimated accurately at the placement level by the block delays on paths between latches and primary inputs/outputs. However, since interconnect delays are not insignificant even for conventional layout styles, placers often use initial critical path/net estimates to prioritize the nets. There are two basic problems in applying this scheme to FPGAs. The first is that the interconnect delays are significant due to the existence of antifuses connecting disjoint path segments. Therefore, it is very probable that a path with less "depth" (passing through fewer blocks) than a statically identified critical path might become critical because of the large interconnect delays of its constituent nets. The second problem concerns basic interconnect delay estimation at the placement level. The common assumption of monotonic scaling of delay, *i.e.*, the *longer* a net, the *larger* is the delay, is not always true. Given non-uniform segmentation schemes created to optimize routability, this becomes a particularly dangerous assumption; the interconnect delay is influenced strongly by the *number* of antifuses in the path, not just the path length.

While our ability to *predict* wirability and timing behavior for FPGAs during placement is rather weak, it continues to be true that the ability to *effect a substantial change* in the wirability and timing behavior is much greater at the placement stage. Compared to the routing stage, where the flexibility is very much limited due to the rigid routing resources and fixed placement, placement offers much more *leverage* in controlling the density and delay of the final layout. An illustration of this fact can be seen again from Figure 2. The placement at the left in Figure 2. is unroutable no matter what routing algorithm is used. However, a small change in the placement (moving cell B), makes it routable.

2.2 Previous Approaches

Existing work in the areas of FPGA synthesis and layout can be divided into four major areas: *partitioning*, *technology mapping*, *placement* and *routing*.

Partitioning is motivated by the fact that very large logic circuits require multiple FPGA chips. Partitioning affects the wirability of each FPGA chip and the timing behavior of the circuit. The partitioner therefore needs to comprehend that the routing resources are fixed and the relative magnitudes of intra-chip and inter-chip delays. Most previous partitioning work is based on the Kernighan-Lin bipartitioning technique [19] with the Fiduccia-Matheyses modifications [20]. The bipartitioning technique is either modified [14] or used iteratively [15] to achieve *k*-way partitions. Some methods also employ simulated annealing [3]for partition optimization [18].

The *technology mapping* phase transforms an optimized, but generic logic input (*e.g.*, comprised of only nand gates) into the logic structures specific to the type of FPGA. The typical objectives in this phase are to minimize area [16] or delay in terms of number of logic levels [17], the assumption being that all interconnects have similar

delay. In an effort to mitigate the deleterious effects of this assumption, [10] uses post-mapping timing-driven placement to derive net-criticality constraints consistent with the particular mapping, for driving the subsequent placement and routing phases.

Although the placement phase is arguably the most critical to ensuring the ultimate success of the final FPGA layout, it has received relatively less specialized attention than FPGA routing. Most industrial placers of which we are aware use some variant of annealing-based semi-custom placement [6], with modifications to suit specific architectures. Interestingly, [22] recently revived for FPGAs some earlier stochastic wirability prediction techniques originally designed to evaluate how probable it was that a particular netlist could be wired in a given gate array architecture. Again, we regard this as a reaction to the continuing difficulty of ensuring that complex designs can be packed onto a specific FPGA architecture with 100% routability.

The two requirements of the *routing* phase are to achieve 100% wirability and to meet the timing requirements. [8] considers routing for the segmented channel architecture of row-based FPGAs via search and dynamic programming algorithms; the approach seeks to maximize routability, but only handles timing via prioritizing critical nets in the cost function. To address performance more directly, [13] introduces a cost-based maze-router for island-style FPGAs [2] that iteratively improves performance by modifying the delay bounds on connections, and rerouting.

2.3 Analysis

We believe that the biggest obstacle to achieving dense, high-performance FPGA layouts is lack of communication among the various phases of the design process, *i.e.*, technology mapping may compromise place/route, placement may compromise routability or delay. The primary cause here is the difficulty to predict accurate wirability and timing behavior at the placement level. The more successful attempts to handle these problems resort to (1) constructive estimators of the downstream design phases, and (2) iterative improvement to undo/redo suboptimal early decisions in the design flow. The overall trend is to combine previously separated, sequential design steps. Examples include [14], which attempts to iterate between partitioning and technology mapping to optimize the timing and wirability. [10] uses a technology mapping solution to generate net constraints for guiding the placement. [23] similarly combines the mapping and placement phases, using rectilinear Steiner tree-based global routing, and relocating the contents of logic blocks in congested areas in an annealing framework. (In the gate array rather than FPGA arena, [21] follows placement with limited re-synthesis and re-placement to optimize the solution.). For row-based FPGAs, we suggest that simultaneous placement, global routing and detailed routing under timing constraints is the necessary next step in this direction.

3 Simultaneous Placement, Global and Detailed Routing

3.1 Algorithmic Strategy

Our strategy for efficiently combining the placement, global and detailed routing steps for row-based FPGAs comprises the following key elements:

- **Optimization-based approach:** The scale of the row-based FPGA layout problem–10^2 to 10^3cells and nets–and the highly rigid geometric alternatives imposed by FPGA cell and wiring resources both lend themselves to a combinatorial optimization formulation of the combined layout problem. Our scheme of choice is simulated annealing [3], which is robust in a variety of related layout problems, suitable for the scale of this problem, and accommodating of complex cost functions.
- **Cells and Nets Uniformly Malleable:** This means that, conceptually, all the cells and all the nets are manipulated concurrently throughout the optimization process, under routability and timing constraints. The problem of reduced flexibility at the routing level, caused by the existence of fixed placement, is mitigated because it is always possible to change the placement.
- **Incomplete Intermediate Layout Configurations:** We do not attempt to move from complete layout to complete layout as our annealer generates and evaluates new solution candidates. We do insist that all cells are legally placed in each intermediate state, but not that all nets be routed. Some nets may be completely unroutable; others may be globally routed but not detail routed; still others may be completely embedded. Our annealing formulation tracks all the actors in this layout process–cells, globally routed nets, detail routed nets–and accrues costs based on the "level of completeness" of the evolving layout.
- **Incremental Global Routing:** For efficiency, we avoid doing complete global routing of all nets at intermediate states of optimization. Instead, we maintain a partial global routing at intermediate states with some nets being unroutable. At any intermediate state, only the existing unroutable nets and a few perturbed nets are globally routed. This has the twin advantages of efficiency and independence of final solution from the routing heuristics used.
- **Incremental Detailed Routing:** Guided by the same reasoning as mentioned for the global routing, we do incremental detailed routing. At any intermediate state of optimization, a partial detailed routing solution exists with certain nets being unroutable in certain channels. Only the existing unroutable nets and a few perturbed nets are being detail routed in relevant channels.
- **Critical Path Calculation:** We address the timing issue by explicitly (re)determining the static critical paths as the layout evolves. Again, as for the routing calculations, we perform only incremental critical path-related calculations for efficiency. At any intermediate stage of optimization, the cell/net perturbation information is used to efficiently determine the change in critical path delay.

The remainder of this section expands on these key components of our formulation.

3.2 Annealing Formulation

Our central goal is an iterative-improvement layout algorithm in which placement perturbations, global routing perturbations, and detail routing perturbations are all feasible concurrently. Obviously, we expect that the frequency of each sort of perturbation will change over the evolution of the annealing cooling process: in the hot regime we expect mostly placement decisions to be made; in the warm regime we expect small placement changes accompanied by large-scale global routing decisions; in the cold regime we expect fine placement changes, modest global routing changes, and substantial attention to detail routing choices. The key technical question is how to accommodate all these actors in the layout process.

We can describe any annealing formulation by describing its

four key components: the *state representation* of the evolving solution; the set of perturbations, or *move-set*, that moves from one state to the next; the *cost function* that measures the quality of each visited state; and the *cooling schedule* that determines how we move from initial large-scale random search to local, fine-grain optimization. This section describes these components.

Our state representation for evolving layouts has 3 components:

- **Cell Placement Assignments:** each cell is always assigned a feasible location. We do not allow illegal intermediate states (*e.g.*, overlapping or unassigned cells).
- **Cell Pin Assignments:** since each cell is based on some arrangement of programmable lookup tables (LUTs), any given cell-level function can be realized using many different pin assignments, usually referred to as *pinmaps* in this context. Each movable cell is always assigned a particular legal pinmap, which affects the nets to which it is connected. We assume it is possible at compile time to generate a manageable palette of pinmap alternatives from which to select a pin assignment during layout.
- **Net Segment Assignments:** since we manipulate nets as well as cells during layout, the disposition of each net must also be accounted for. Nets may appear in three distinct states: (1) completely unrouted, (2) globally routed but not detail routed, (3) globally and detail routed. The distinction involves whether each net has associated with it a set of free vertical and horizontal routing segments which complete its necessary port connections. Each net n can be regarded as a pair of sets (V_n=$\{v_1, v_2,...v_k\}$, H_n=$\{H_1, H_2, ... H_l\}$), where each v_i is a vertical routing segment used to span channels, and each h_i is a horizontal segment in a particular channel. An unrouted net has no assigned segments, *i.e.*, (V_n=ϕ, H_n=ϕ). A globally routed net has vertical segments assigned, but not horizontal segments: (V_n=$\{v_1, v_2, ...v_k\}$, H_n=ϕ). A completely routed net has its vertical and horizontal segments assigned.

Moves are the mechanisms for initiating state changes in the course of optimization. Our move-set is actually quite simple, comprising only two orthogonal classes of moves: *cell swaps*, and *pinmap reassignments*. Swaps randomly exchange the contents at two different logic module locations. Since one of these locations may be empty, we also support single cell translations. Pinmap reassignments randomly change the pin assignments for a particular cell from a palette of fixed, legal alternatives. An important point here is that there are *no* moves that specifically alter nets. Rather, each move that alters cells *removes* any routing associated with the pins on the moved cells. Then, fast heuristics attempt to reroute the ripped up nets globally and detailed. Thus, a single placement move may cause a set of routed nets to be rerouted differently, or a set of previously routed nets to become unroutable, or a set of previously unrouted nets to become routable. Our strategy is to employ a vigorous placement optimization, each of whose atomic steps sets off a cascade of local net ripup and repair attempts. By allowing this routing process to fail, *i.e.*, to be incomplete after any given placement perturbation, we free the overall layout optimization to evolve both placement and routing at a rate determined by the inherent difficulty of the problem.

The cost function in an annealer controls the acceptance of new states, and measures the overall quality of the evolving solution. Our cost function must address both routability and timing concerns for the evolving placement, global routing and detailed routing. We use the following weighted cost function:

$$Cost = W_g \times G + W_d \times D + W_t \times T \quad (1)$$

G counts the number of globally unrouted nets. Similarly, D counts the number of nets that lack a complete detailed routing. T measures the worst-case delay on the slowest path in the current placement, using a detailed timing model that accounts for physical segment and antifuse delays. Roughly speaking, the cost function penalizes the *unroutability* of the current placement and the *worst-case delay* through the layout, as determined by an up-to-date critical path analysis. Perhaps most interestingly, there is *no* wirelength estimation term. Wirelength minimization happens *constructively*, in the sense that the fast heuristics we employ for incremental global and detailed routing after each placement move are strongly biased toward short paths, where possible. The weights, W_g, W_d and W_t are determined adaptively at runtime so as to normalize the components of the cost function so that each term contributes approximately equally to the cost function.

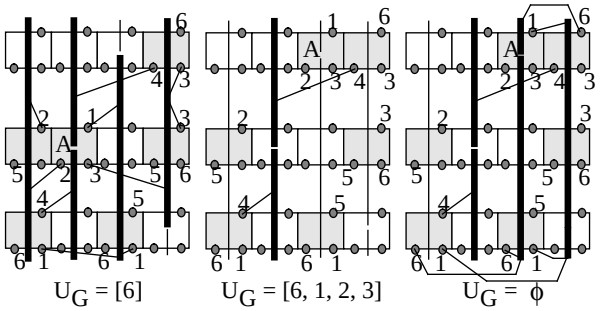
Our cooling schedule is based on the scheme proposed in [4] which determines starting temperature, time spent at a temperature and temperature decrements adaptively, as the annealing progresses. In the following sections we return to the details of the incremental routing and delay calculations necessary to compute the cost function introduced in this section.

3.3 Incremental Global Rerouting

A move that alters the placement produces a change in cost ΔC. One component of this cost is the incremental change in the number of globally unroutable nets, δG. Global routing for row-based FPGAs consists primarily of assigning feedthroughs to nets that need them. Of course, we would ideally like to consider a full global routing optimization after each cell perturbation, *e.g.*, by annealing, [25], or by heuristic cost-based search [24], but this is computationally infeasible. Hence, we employ an incremental global router based on fast, simple heuristics. At any intermediate stage of annealing, U_G represents the set of nets that are globally unroutable. When a cell is moved, all the nets connected to it are ripped up (*i.e.*, their vertical segment assignments are removed) and these nets are added to U_G. U_G itself is sorted based on the estimated length of its contents. After each move, we then work our way down U_G, attempting to globally route each unroutable net, thus giving priority to the longer unroutable nets.

The heuristics used for globally routing a net are simple: we assign the available set of vertical segments that are closest to the center of a net's bounding box. After an initial start-up transient wherein many of the nets find some (poor) global path, each new move affects a manageable number of nets. Moreover, since the heuristic used is simple, the time requirement is minimal. Perhaps most importantly, we note that we are relying not on *one* exhaustive search for a good global route for each net, but rather, on *many* simple searches for global routes, each undertaken in a new, possibly more compliant placement. Nets are continuously being ripped up and re-

Figure 3. Incremental global routing



routed, rendering the final solution less dependent on the ability of the global routing heuristic to find the *best* paths.

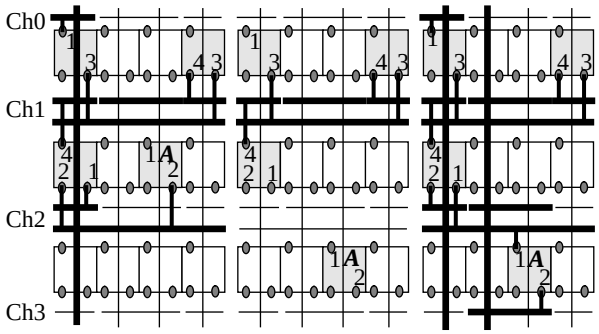
The incremental global routing mechanism is illustrated by the 6 nets (numbered 1-6) in Figure 3. Initially, only net 6 is unroutable, U_G = [6]. After the illustrated move, the nets connected to the cell perturbed (nets 1, 2 and 3) are ripped up, thus freeing some vertical segments. Nets 1, 2 and 3 are now added to U_G. Working through U_G, we attempt to find a global route for net 6, which succeeds, and then similarly route net 1. Nets 2 and 3 no longer need vertical resources (*i.e.*, a trivially null global routing now suffices). So, at the end of incremental global routing, all nets are globally routed and the contributed δG for this move is -1.

3.4 Incremental Detailed Rerouting

Any placement move may also alter the number of nets without a detailed routing, in a contribution to ΔC of δD. It should be noted here that if a net cannot be globally routed, it automatically cannot be detail routed. Following the same reasoning as was discussed for global routing, we employ a set of fast heuristics for incremental detailed routing. In addition to maintaining U_G for each intermediate layout configuration, we also maintain for each channel *R*, a set U_{DR} of unroutable nets–that is, nets for which there are insufficient horizontal segments to complete *detailed* routing in channel *R* for the current placement. Note that when any cell is moved or has its pin-map perturbed, we remove *all* connected nets, both the *vertical* segments (global routing) and the *horizontal* segments (detailed routing). These nets are initially deposited in U_G and relevant U_{DR}. Following incremental global routing, we proceed through each of the *P* total channels, and try to detail route the unrouted nets in each U_{DR}, 1 R P. As before, each U_{DR} is actually a queue sorted on estimated net length.

The incremental detailed router assigns available tracks to unrouted nets based on two terms: segment-wastage and number of segments used [11]. In this manner, we indirectly optimize for minimum net length; our router constructively prefers short paths. Large segment usage for small-span nets may cause wirability problems in the channel. If too many segments are used for a net, then this path will pass through many horizontal antifuses, and likely accrue unacceptable delay. Of course, a move's acceptance depends on the complete cost function comprising both wirability and timing terms.

Figure 4. Incremental detailed routing



The incremental detailed routing procedure is illustrated in Figure 4. Initially (Figure 4.a), all nets have been globally routed. Also, in channels 0, 1 and 3, all nets (that are present in these channels) can be detailed routed. However, in channel 2, net 1 cannot be routed for lack of horizontal routing resources. When cell A is moved (Figure 4.b), nets connected to cell A (nets 1, 2) are ripped up (*i.e.*, vertical and horizontal segments connected to nets 1 and 2 are freed). Therefore, now nets 1 and 2 need global routing and there are *detail*-unrouted nets in all channels. Eventually, all these unrouted nets are rerouted both globally, and then in the channels (Figure 4.c). Note that only a small subset of nets gets rerouted on any placement perturbation.

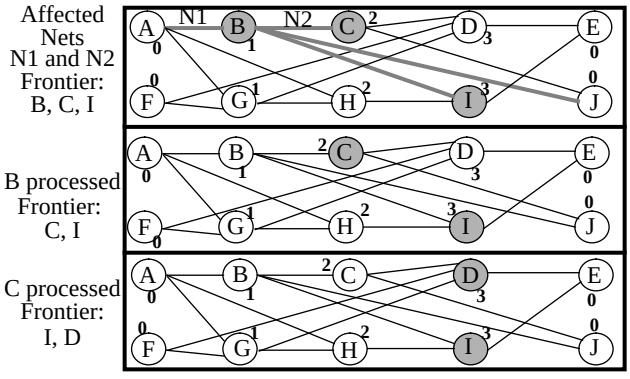
3.5 Incremental Worst-Case Delay Calculation

A placement perturbation can affect the global and detailed routability of a subset of the nets, which is reflected in terms of the cost function. Similarly, we must reflect the impact of placement changes on delay. Rather than relying on the user to supply a set of critical paths to evaluate, the worst-case critical path is incrementally updated after each perturbation. Any change here contributes an incremental change in worst-case delay δT to the overall cost function. Critical paths are defined between the boundaries formed by primary inputs, outputs and sequential blocks (or flip-flops). We consider the long-path delay problem and assume that all paths are sensitizable. Of course, this is a simplification, and our timing estimates are therefore quite conservative. Nevertheless, by maintaining *pressure* on the worst path delay as placement evolves, we do bound all other delays.

Initially the cells are levelized. Boundary elements have a level of 0. The level of any other cell is one more than the maximum of the levels of all its inputs. The levels determine the order in which the cells should be processed while propagating delays across paths. Since levels are determined only by connectivity and not the location of cells, levelization needs to be done only once.

Figure 5. illustrates the delay propagation mechanism. The levels of the cells are shown beside each. When a cell is moved (for example, cell B in Figure 5.), all nets connected to it may be rerouted. Therefore, the interconnect delays for these nets (driver to sinks) have to be recalculated. In addition, the delay change needs to be propagated to a boundary while respecting the levels of the cells concerned. To do this, a frontier of affected cells is maintained: affected cells are those cells which are either connected to affected nets, or which lie on the path of affected cells to boundaries. Initially the frontier has only non-boundary cells which are connected to the affected nets as inputs (*e.g.*, cells B, C and I in our example). At any

Figure 5. Incremental Critical Path Calculation



stage, the cell in the frontier with the minimum level is processed. Therefore, cell B is processed. Processing a cell involves two parts: updating the output delay of the cell based on the new input delays, and if output delay changes, putting new cells in the frontier by examining the fanout cells. If a fanout cell is already in the frontier or if it happens to be a boundary element, then that cell is not added to the frontier. The expansion stops when the frontier is empty. The maximum delay at an input of a boundary cell is a measure of the most critical path delay.

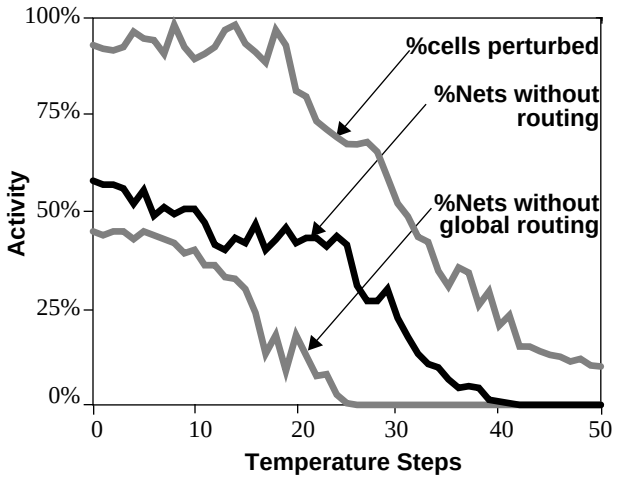
To sharpen the worst-case delay estimate, we use a detailed RC tree model for the interconnect–when the nets contributing to this worst path are physically embedded. Since the exact antifuse usage is known for such nets, we calculate the Elmore delay [5]. Of course, in our simultaneous place and route strategy, not all nets are physically embedded at all times. For such nets we resort to crude estimators that relate the known spatial extent of the net (based on its current port locations) to the probable number of antifuses it will encounter, to create a rough delay estimate. This is, of course, inaccurate, but suffices early in the layout process. Recall that other terms of the cost function put pressure on the number of unrouted nets, coercing these nets to take feasible paths for which we can more accurately estimate delay.

4 Implementation and Results

We have implemented these ideas in a prototype simultaneous placement, global and detailed routing tool for row-based FPGAs. Our goals here are (1) to illustrate the dynamics of fully simultaneous placement, global and detailed routing optimizations; (2) to demonstrate the impact of this layout strategy on delay optimization; (3) to demonstrate the impact on wirability.

We begin by illustrating the dynamics of the annealing layout process in Figure 6. We plot the fraction of cells and nets whose layout is changing as annealing optimization is proceeding. Our goal here is demonstrate that our formulation does, in fact, allow placement, global and detailed routing to proceed simultaneously. To be precise, we plot, at each temperature: %cells perturbed; % nets globally unrouted; % nets unrouted. The difference between the last two (%nets unrouted - % nets globally unrouted), represents the fraction of nets that are globally routed but detailed unrouted. As can be seen, placement activity starts aggressively, and falls off until only small, local perturbations are being attempted for local routability improvement. Many nets start globally unrouted, but by the middle of the annealing process have found acceptable vertical routing resources. Initially the number of nets globally routed but detailed unrouted (represented by the difference of the plots for unrouted nets and globally unrouted nets) is small. However, the aggressive changes in the middle of the layout process, which result in all nets being globally routed, raises the number of nets that are globally routed but detailed unrouted. In the second half of the optimization process, all nets continue to be globally routed (although their vertical segment assignments do change) and the number of unrouted nets (caused primarily by detailed unroutability in this phase) converges to zero, thus producing a fully routed solution. The dynamics here are as we described in our initial goals: vigorous placement optimization, followed by a focus on global routing, followed by graceful convergence to full detailed routing.

Figure 6. Illustrating layout evolution during optimization



To measure the impact of this strategy on delay optimization, our tool has been tested on 5 MCNC benchmark examples. The results were compared with those of a proprietary sequential place and route system for row-based FPGAs in use at Texas Instruments. The custom placer is based on TimberWolfSC [6], the global router is from [7] and the detailed router from [11]. Table 1 shows the results. The critical paths were determined using Texas Instruments' timing analyzer. Post-layout interconnect delays were determined using the RICE [12] AWE-based delay evaluation tool. The critical path delays determined by the post-layout timing analyzer were very close (within 90%) of that determined internally by our simultaneous place and route tool. Results appear in Table 1. Most importantly, we achieved improvements in worst-case timing from 16% to 28% over the sequential layout tool.

Table 1. Timing Improvement

Design	# cells	% improvement
s1	181	28
cse	156	16
ex1	227	23
bw	158	25
s1a	163	21

Table 2. Wirability Improvement

Design	# cells	Tracks/Channel reqd	
		Seq. P&R	Sim. P&R
s1	181	23	18
cse	156	22	17
ex1	227	26	21
bw	158	15	10
s1a	163	22	17

To measure the wirability improvement that can be achieved with the simultaneous approach, the number of tracks per channel in these designs was reduced for each example to the point that our simultaneous tool, and the sequential tool failed to meet 100% wirabil-

ity. By this process we determined the minimum number of tracks required in each channel of each row-based design to successfully route the layout. Results appear in Table 2. Wirability improvements (track count reductions) ranging from 20% to 33% were achieved. The primary reason for this was that the placer in the sequential case packed the cells based on the connectivity while being ignorant of the routability of such a configuration. The time required for sequential layout was roughly 1 hour compared to 3-4 hours for simultaneous layout on an HP 425 workstation.

Finally, Figure 7. illustrates a larger 529 cell design completed with 100% routing in roughly 8 hours on an IBM RS6000 by our tool.

5 Conclusions

A technique for simultaneous placement and routing has been developed for row-based FPGAs. Efficient incremental global and detailed routing techniques coupled with an aggressive combinatorial optimization formulation enables our tool to optimize routability and delay simultaneously. Preliminary results show the merits of this approach over traditional sequential approaches, achieved at some cost in runtime. Our current work focuses on technical improvements to the core of the annealing formulation for increased speed, and larger benchmarks.

Acknowledgments

This research was funded by the Semiconductor Research Corporation. We would like to thank Texas Instruments for providing the opportunity to compare our tool with their place and route system. We would also like to thank Kaushik Roy, Mark Harward and N.S. Nagaraj for helping with the comparison.

References

[1] A. El Gamal *et. al.*, "An Architecture for Electrically Configurable Gate Arrays," *IEEE Journal of Solid-State Circuits*, April 1989.

[2] W. Carter *et. al.*, "A User Programmable Reconfigurable Gate Array," *Proc. 1986 IEEE CICC*, May 1986.

[3] S. Kirkpatrick *et. al.*, "*Optimization by Simulated Annealing*," Science, 1983

[4] M. D. Huang, F. Romeo and A. Sangiovanni-Vincentelli, "An Efficient Cooling Schedule for Simulated Annealing," *Proc. of ICCAD*, 1986.

[5] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers", *Journal of Applied Physics*, 1948.

[6] C. Sechen and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," *Proc. of ICCAD*, 1988.

[7] R. Rao, "A Global Router for Channelled FPGAs," *Proc. MCNC Logic Synthesis Workshop*, 1992.

[8] J. Greene *et. al.*, "Segmented Channel Routing", *Proc. of DAC*, 1990.

[9] S. Ercolani and G. De Michelli, "Technology Mapping for Electrically Programmable Gate Arrays", *Proc. of 28th DAC*, 1991.

[10] P. S. Sawkar and D. E. Thomas, "Performance Directed Technology Mapping for Table-Look-Up Based FPGAs," *Proc. of 30th DAC*, 1993.

[11] K. Roy, "Detailed Routing for row-based FPGAs", *IEEE Transactions on CAD*, 1994.

[12] C. L. Ratzlaff, N. Gopal and L. T. Pillage, "RICE: Rapid Interconnect Circuit Evaluator", *Proc. of 28th DAC*, 1991.

[13] J. Frankle, "Iterative and Adaptive Slack Allocation for Performance-driven Layout and FPGA Routing", *Proc. of DAC*, 1992.

[14] P. S. Sawkar, "Performance Directed Synthesis for FPGAs", Prospectus document, 1993.

[15] R. Kuznar, F. Brglez, K. Kozminski, "Cost Minimization of Partitions into Multiple Devices", *Proc. of DAC*, 1993.

[16] R. Murgai *et. al.*, "Improved Logic Synthesis Algorithms for Table Look Up Architectures", *Proc. of ICCAD* 1991.

[17] R. J. Francis *et. al.*, "Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays", *Proc of 27th DAC*, 1990.

[18] K. Roy and C. Sechen, "A Timing-driven N-Way Chip and Multi-Chip Partitioner", *Proc. of ICCAD 1993*.

[19] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Systems Technical Journal*, vol 49, 1970.

[20] C. M. Fiduccia and R. M. Mattheyses, "A Linear-time Heuristic for Improving Network Partitions", *Proc. of 19th DAC*, 1982.

[21] M. Pedram and N. Bhatt, "Layout Driven Logic Restructuring/Decomposition", *Proc.Proc. of ICCAD*, 1991.

[22] P. K. Chan *et. al.*, "On Routability Prediction for Field Programmable gate Arrays," *Proc. 30th DAC*, June 1993.

[23] N. Bhatt and D. Hill, "Routable Technology Mapping for FPGAs", *FPGA Workshop*, 1992.

[24] J. Rose, "Parallel Global Routing for Standard Cells", *IEEE Transactions on CAD*, September 1990.

[25] K. W. Lee and C. Sechen, "A New Global Router for Row-Based Layout", Proc. of ICCAD, 1988.

Figure 7. Output of example with 529 cells