FPGA Routing and Routability Estimation Via Boolean Satisfiability

R. Glenn Wood and Rob A. Rutenbar, *Fellow, IEEE*

***Abstract*— Guaranteeing or even estimating the routability of a portion of a placed field programmable gate array (FPGA) remains difficult or impossible in most practical applications. In this paper, we develop a novel formulation of both routing and routability estimation that relies on a rendering of the routing constraints as a single large Boolean equation. Any satisfying assignment to this equation specifies a complete detailed routing. By representing the equation as a binary decision diagram (BDD), we represent all possible routes for all nets simultaneously. Routability estimation is transformed to Boolean satisfiability, which is trivial for BDD's. We use the technique in the context of a perfect routability estimator for a global router. Experimental results from a standard FPGA benchmark suite suggest the technique is feasible for realistic circuits, but refinements are needed for very large designs.**

***Index Terms*— Computer-aided design, field programmable gate array (FPGA), placement, rapid prototype, routing.**

I. Introduction

LAYOUT for field programmable gate arrays (FPGA's) is difficult primarily because of the rigid, discrete interconnect structure of current programmable parts. Placement and routing techniques imported and adapted from the world of conventional application specific integrated circuits (ASIC's) have been quite successful in automating FPGA layout. Nevertheless, some layout issues are poorly dealt with in FPGA's, most notably, problems of routability estimation. Answering *exactly* a simple question such as "is this placement routable in this FPGA routing fabric?" is usually impossible. Routability and congestion estimators adapted from ASIC's often fare poorly here because the geometric "slack" allowable in freeform routing on bare silicon is simply not present when we can only embed routes in the finite interconnect patterns available in an FPGA.

A wide variety of tactics have been tried to solve the FPGA routing problem. Brown [7] and Chan [10] both elaborate stochastic wirability theories to handle FPGA structures. Rao [25] extends ASIC global routing heuristics for FPGA's. Alexander [1] describes a router that solves approximations to a sequence of Steiner-Tree-on-a-Graph problems for all nets in a multiweighted graph depicting the FPGA architecture. Aggressive rip-up-and-retry has been developed to deal with the net ordering problem [22] and with delay optimization [13]. Brown [6] describes a detailed router which expands the output from a global router into a graph of possible detailed routes for a particular two-point connection; Lemieux [19] extends this technique to the case of partially segmented channels. Greene [15] and Roy [26] describe search-based strategies for segmented channel routing. For very large designs, [18] and [28] retarget ASIC floorplanning techniques to FPGA's to improve routability and delay. While often successful, none of these techniques can *guarantee* that a routable placement will indeed be routed.

It has also been argued that the highly constrained nature of the interconnect in FPGA's makes them amenable to unique solution strategies. One such approach is described by Nag and Rutenbar [23], [24] which attacks both the placement and routing problems simultaneously by allowing placement changes to be evaluated for their routability and ultimate net delay using an actual, incremental detailed router. Bhatt and Hill [3] look at the role of technology mapping plays in influencing routability. Wu [32] observes that simplified variants of the detailed routing problem for FPGA's are reducable to the two-dimensional (2-D) interval packing problem, and to graph coloring. These constructions were used to assert the NP-hardness of FPGA routing in [33]. Wong *et al.* [11], [29] introduce a global router that uses an off-line, integer linear programming technique to enumerate all possible uses of a switch-block architecture. A global router uses this information as an exact congestion estimator for each small, atomic block of the routing fabric. However, the global router must still handle the complexity of managing the competition for routes between the atomic blocks in the fabric. Again, none of these techniques can predict or guarantee *exactly* the routability of a nontrivial region of an arbitrary FPGA.

In this paper, we offer a novel formulation for both routing and routability estimation that provides exact routability guarantees. The key idea, extending earlier work in ASIC routing by Devadas [12], is to render the routing problem as a set of interacting assignments of nets to available resources which can be expressed exactly as a large set of Boolean equations. Given a netlist, a description of the routing fabric, a region to be routed, and boundary constraints as determined by a global router, we show how to generate automatically the necessary Boolean equation that determines the routability of this region. Any assignment of values to input variables that satisfies this Boolean routability equation (i.e., that makes

Manuscript received March 31, 1997; revised October 14, 1997. This work was supported in part by the SRC under Contract DC-068 and by the Army Research Office.

R. G. Wood was with the Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA 15213 USA. He is now with Hewlett-Packard, Colorado Springs Division, Colorado Springs, CO 80907 USA.

R. A. Rutenbar is with the Department of Electrial and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Publisher Item Identifier S 1063-8210(98)02957-6.

the output 1) specifies precisely a complete, detailed routing. Although Boolean satisfiability is itself an NP-hard problem, there is a growing body of successful, practical attacks on even large satisfiability problems, e.g., decision diagrams [8] and heuristic search [20]. In our attack on the problem, we represent this Boolean function using binary decision diagrams (BDD's, [8]) which allows us to represent all possible routings for a region, for all nets, *simultaneously*. The efficient BDD-based implementation allows not only for rapid construction of the necessary routability functions, but also for very fast incremental updates of the routability, for example, as the result of perturbing a global route.

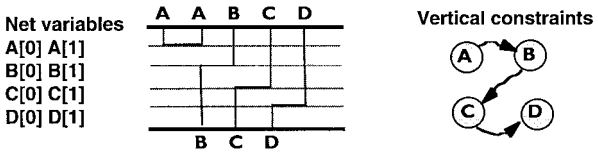
We refer to this strategy as *routing via Boolean satisfiability*. In the remainder of the paper we develop and illustrate this idea, extending the original treatment of [30], [31]. Section II develops the basic satisfiability formulation for routing. Section III describes implementation issues, including treatment of several practical issues associated with solving large satisfiability problems. Section IV shows some promising initial results for a subset of the Xilinx 4000-series routing fabric. Finally, Section V offers some concluding remarks.

II. Routing Via Boolean Satisfiability: Formulation

In [12], Devadas showed a simple but elegant formulation of conventional two-layer channel routing as Boolean satisfiability. This formulation encodes the information present in a channels vertical constraint graph, horizontal constraint graph, and anticipated channel width into Boolean constraints on n-bit vectors, one per net to be routed. Fig. 1 shows a simple example of a channel with four nets and four tracks represented with a transformation into four 2-bit Boolean vectors. Each vector can be assigned a Boolean value encoding the track assignment for the corresponding net; in this example the conventional dense encoding using $\log_2$ (*tracks*) bits per vector is used. The *exclusivity* constraints ensure that the horizontal segments of no two nets overlap in the same track. The *vertical* constraints likewise ensure no overlap between nets' vertical segments. The resulting *routability* constraint is satisfiable just if the conjunction of all the other horizontal and vertical constraints are satisfiable. This resulting Boolean equation fully specifies the set of feasible net-to-track mappings via its satisfying variable assignments.

More precisely, any satisfying assignment is a *complete* routing of the problem—not merely a set of feasible net-to-track assignments for each net in isolation—assigning all nets simultaneously to feasible tracks. The problem with this application is that n, which must be known to execute the formulation, is a function of the required number of tracks in the final solution—a parameter which is necessarily an output of the problem. Thus, one is forced to continuously reformulate the problem with increasing guesses at the final track density until a satisfiable Boolean formulation exists. Worse, the formulation requires a set of fairly severe simplifications concerning wire width, via size, etc. For real ASIC channel routing, the formulation is impractical.

But this Boolean satisfiability approach is much more amenable to FPGA's, which really do have completely rigid



Exclusivity Constraints

$$\text{ExcCon1} = ((A[0] \oplus B[0]) + (A[1] \oplus B[1]))$$
$$\text{ExcCon2} = ((B[0] \oplus C[0]) + (B[1] \oplus C[1]))$$
$$\text{ExcCon3} = ((C[0] \oplus D[0]) + (C[1] \oplus D[1]))$$

Vertical Constraints

$$\text{VCGCon1} = \vec{A} > \vec{B}$$
$$\text{VCGCon2} = \vec{B} > \vec{C} \quad \text{(i.e., as binary numbers)}$$
$$\text{VCGCon3} = \vec{C} > \vec{D}$$

Routability Constraint

$$\text{Routable} = \text{ExcCon1} \wedge \text{ExcCon2} \wedge \text{ExcCon3} \wedge \text{VCGCon1} \wedge \text{VCGCon2} \wedge \text{VCGCon3}$$

Fig. 1. Example of Devadas' satisfiability formulation for simple two-layer channel routing.

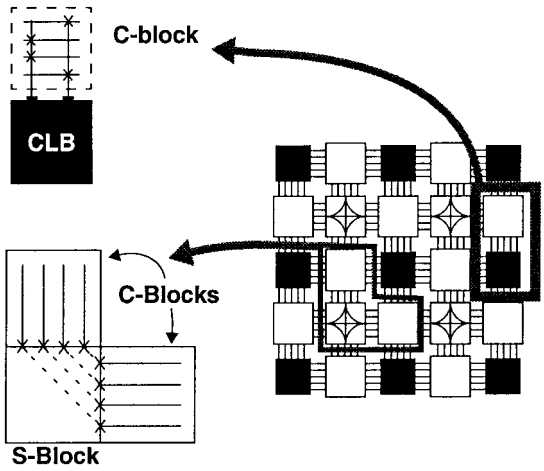


Fig. 2. Island-style FPGA model.

routing resources. In this work, we restrict ourselves to an island-style (i.e., Xilinx type, see [7]) FPGA fabric, illustrated in Fig. 2. Configurable logic blocks (CLB's) connect to connection blocks (C-blocks) which allow signals egress into the global routing fabric. Switch blocks (S-blocks) allow signals to turn corners between rows and columns of S-blocks and CLB's. In this routing problem, there is no need to model a Vertical Constraint Graph because track stubs in an FPGA are prefabricated to be electrically disjoint. However, FPGA's do add a different type of constraint: ensuring that signals maintain continuity. In a *connectivity* constraint a net must negotiate turns through the finite set of resources provided in switch blocks, and also must connect to its sources/sinks through the finite set of connections provided in connection blocks.

To recast this routing problem as a satisfiability problem, we need to do the following.

1) Invoke a global router which assigns each net a path through regions of the routing fabric composed of several S-blocks and C-blocks in the FPGA.
2) Assign to each net a vector of Boolean variables for each region of routing fabric through which it passes. This vector of variables encodes each possible decision for how to assign the net to physical resources in the region.

3) Formulate a Boolean *connectivity* constraint (i.e., a Boolean function) that ensures that each net actually connects through a set of legal, contiguous routing resources from source to sink. This function is essentially a characteristic function over the encoding of net-to-resource assignments that is "1" for connected paths.
4) Formulate a Boolean *exclusivity* constraint (i.e., another Boolean function) that ensures that no two electrically distinct nets try to use a common routing resource in any block of the routing fabric. This function is essentially a characteristic function over the net-to-resource assignments that is "1" for sets of noninterfering paths.
5) Formulate the final Boolean *routability* function that determines all the possible routes for these nets; this is just the intersection of the constraints of steps 3 and 4.

There are several tradeoffs inherent in this strategy. First, although it is conceptually straightforward to create the necessary Boolean functions for direct detailed routing for *all* the nets across the *entire* fabric for a complete FPGA, the complexity of the resulting functions is intractable. In our early formulations of the problem, direct representation of all detailed routes across the entire fabric of even a small FPGA produced satisfiability equations too large to represent and solve. Hence, we chose to move some degrees of freedom out of the satisfiability formulation: we assume a global router assigns signals to *regions* of routing fabric, and we also assume that we will use the satisfiability formulation on these regions of the fabric, instead of the whole chip. Thus, the existence of the global router here is simply an engineering compromise. Given a region to be routed or estimated for routability, the global router assigns nets to regions of the fabric, and fixes entry/exit points on the overall perimeter of the region. These form our boundary constraints for satisfiability-based routing.

For our particular experiments, we assumed simple island-style fabric, and chose to treat the FPGA as an array of vertical channels. Other region shapes are possible, but channels are perhaps the simplest for a global router to deal with. Also, the choice of vertical channels across the entire height of a candidate FPGA stresses the fact that we wish to represent reasonably large, global portions of the routing fabric as the atomic units for global routing. The global router not only picks a feasible coarse graph for each net, but also provides a *channel port specification* for each channel. This specification is just a set of triples (n, ι, flag) where each triple contains the following. The first field is the net ID. Next is the connection point where this net intersects this channel: a row and column block in the array, and a track number. This connection point must occur on either a CLB or a C-block. Finally, *flag* can take on one of two values: *entrance* or *exit*. *Entrance* means that the net intersects the channel from either the west or east side and makes a turn heading south. *Exit* means that the net intersects the channel from either the east or west and makes a turn north. This specification can be created via a simple scanline-style horizontal pass through the tracks in the channel.

Fig. 3 gives a detailed example of a channel-style region of an FPGA with three columns and ten rows. The architecture is described in terms of these standard parameters [7, ch. 6]:

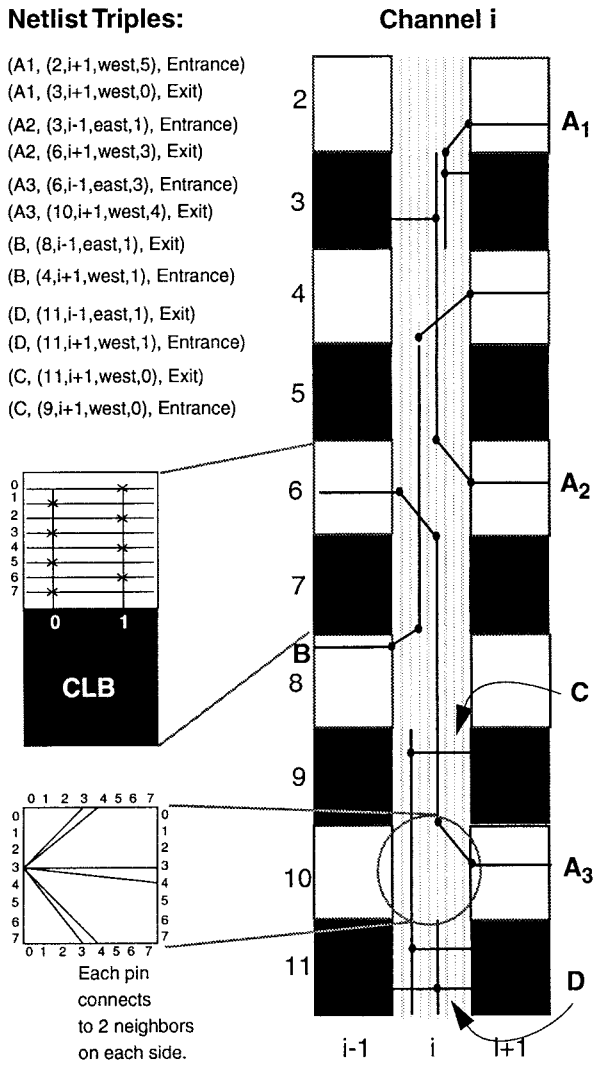


Fig. 3. Detailed FPGA example channel with routes and netlist triples shown.

W is the number of tracks per channel, Fc is the number of tracks to which each CLB port can actually connect, and Fs is the number of turns that each wire entering a switch-block port can make inside the switch block. For the example, the architecture has $W = 8, Fc = 4, Fs = 6$. The figure also shows a netlist in this triples-format for six nets. One example set of detailed paths for these nets is embedded in the channel.

To understand the satisfiability formulation, it is instructive simply to construct the relevant Boolean equations for this simple example in Fig. 3. The formulation requires that six vectors be allocated

$$\vec{A}_{1,i}, \vec{A}_{2,i}, \vec{A}_{3,i}, \vec{B}_i, \vec{C}_i, \vec{D}_i.$$

These correspond to the six two-point connections of interest. We encode each of these vectors in 3 bits, since each vector encodes one of at most eight distinct net-to-resource decisions. (Other encodings are possible, e.g., see [21].) The channel subscript i is used to emphasize that these bit vectors only belong to the formulation for channel i. Upon determining a feasible route, these vectors will contain the binary-encoded track assignment for each two-point connection. There are two

$$\text{ConnA}_{1,i} = [(\vec{A}_{1,i} \equiv 5) \vee (\vec{A}_{1,i} \equiv 6)] \wedge [(\vec{A}_{1,i} \equiv 1) \vee (\vec{A}_{1,i} \equiv 3) \vee (\vec{A}_{1,i} \equiv 5) \vee (\vec{A}_{1,i} \equiv 7)]$$

$$\text{ConnA}_{2,i} = [(\vec{A}_{2,i} \equiv 0) \vee (\vec{A}_{2,i} \equiv 2) \vee (\vec{A}_{2,i} \equiv 4) \vee (\vec{A}_{2,i} \equiv 6)] \wedge [(\vec{A}_{2,i} \equiv 3) \vee (\vec{A}_{2,i} \equiv 4)]$$

$$\text{ConnA}_{3,i} = [(\vec{A}_{3,i} \equiv 3) \vee (\vec{A}_{3,i} \equiv 4)] \wedge [(\vec{A}_{3,i} \equiv 4) \vee (\vec{A}_{3,i} \equiv 5)]$$

$$\text{ConnB}_i = [(\vec{B}_i \equiv 1) \vee (\vec{B}_i \equiv 2)] \wedge [(\vec{B}_i \equiv 1) \vee (\vec{B}_i \equiv 2)]$$

$$\text{ConnC}_i = [(\vec{C}_i \equiv 1) \vee (\vec{C}_i \equiv 3) \vee (\vec{C}_i \equiv 5) \vee (\vec{C}_i \equiv 7)] \wedge [(\vec{C}_i \equiv 1) \vee (\vec{C}_i \equiv 3) \vee (\vec{C}_i \equiv 5) \vee (\vec{C}_i \equiv 7)]$$

$$\text{ConnD}_i = [(\vec{D}_i \equiv 0) \vee (\vec{D}_i \equiv 2) \vee (\vec{D}_i \equiv 4) \vee (\vec{D}_i \equiv 6)] \wedge [(\vec{D}_i \equiv 0) \vee (\vec{D}_i \equiv 2) \vee (\vec{D}_i \equiv 4) \vee (\vec{D}_i \equiv 6)]$$

The seeming redundancies are an artifact of the constraint generation which starts independently from each end-port of each 2-point wire and generates the relevant constraintn

$$\text{Excl}_i = (\vec{B}_i \oplus \vec{A}_{2,i}) \wedge (\vec{B}_i \oplus \vec{A}_{3,i}) \wedge (\vec{A}_{3,i} \oplus \vec{C}_i) \wedge (\vec{C}_i \oplus \vec{D}_i)$$

$$\text{Routable}_i = \text{ConnA}_{1,i} \wedge \text{ConnA}_{2,i} \wedge \text{ConnA}_{3,i} \wedge \text{ConnB}_i \wedge \text{ConnC}_i \wedge \text{ConnD}_i \wedge \text{Excl}_i$$

Fig. 4. Satisfiability formulation for example of Fig. 3.

different types of routing violations that we must be careful to avoid in our formulation. We must ensure no *connectivity* constraints are violated by checking that proposed pin entries and exits on each atomic region of the fabric are actually feasible. Also, we must ensure that no *exclusivity* constraints are violated; these result when two or more distinct nets try to use the same routing resource.

To formulate the solution, these two classes of routing violations must be prevented. First, each two-point connection has a connectivity constraint associated with it whose purpose is to ensure that the connection makes a contiguous path through the channel. Also, exclusivity constraints are needed for all pairs of connections of distinct nets that interact: that is, if the interval defined by their endpoints overlaps in one or more C-blocks. The exclusivity constraint between any two bit vectors is just the *vectored* exclusive-or between them. This ensures that at least one bit differs in their respective vectors. On the other hand, we sometimes choose *not* to generate an exclusivity constraint between two nets: this is the mechanism for handling multipoint nets. By allowing electrically common two-point nets to "overlap," multiple two-point connections of the same signal can be merged automatically. In this way, we can handle multipoint nets (i.e., those that fan out inside the routing fabric to multiple sinks) directly and naturally. Fig. 4 shows the complete set of equations. For convenience, the equations are shown in a shorthand where only the net vectors are shown, using the obvious decimal encoding, as opposed to each of their individual bits.

Each of the connectivity constraints can be broken down into two intersected clauses, each arising from one of the paired triples. For instance, in $\text{ConnA}_{1,i}$ the two component triples are $(A, (2, i+1,$ west, $5)$, Entrance) and $(A, (3, i+1,$ west, $0)$, Exit). The first engenders the query: $find\_tracks\_\text{S}\_block(2, i,$ east, 5, south) which looks at the S-block architecture present in S-block $(2, i)$ and determines to what pins on the south of this S-block the pin labeled "5" on the east side can turn. The second triple results in the query: $find\_tracks\_\text{C}\_block(3, i,$ east, $0)$ which queries the C-block at location $(3, i)$ and returns the set of tracks that pin 0 on the east side can connect to. In this instance, $find\_tracks\_\text{S}\_block$ will return tracks 5 and 6 while $find\_tracks\_\text{C}\_block$ will return tracks 1, 3, 5, and 7. So, for the connection to be complete, it requires that the results from these two function calls be ANDed together. The exclusivity constraints are built between all interacting pairs of two-point connections of differing nets. This ensures that the route obeys the one-net-per-segment constraints.

Note that the construction process for the equations allows us to prune some obvious interactions that cannot happen. For example, we do not need to check for exclusion between nets whose global routes cannot possibly share resources, or create clauses that check connectivity for block-to-block paths that cannot connect in our architecture. In this way we reduce the complexity of the intermediate connectivity equations, the single exclusivity equation, and the final routability equation. Recall that $\mathbf{Routable}_i$ is, at the end, a Boolean function of the per-net bit-vectors which specifies the assignment of net segments to individual routing resources in this region. If $\mathbf{Routable}_i$ is identically "0," then the region is unroutable under the boundary conditions established by the current global routing. In contrast, if there exist satisfying assignments for the inputs to $\mathbf{Routable}_i$, then any such assignment is a valid routing. The need to manipulate these formulae to create $\mathbf{Routable}_i$, and to test it for satisfiability motivates our decision to represent all these equations as BDD's.

III. Solution Implementation: BDD Management

We use reduced ordered binary decision diagrams (ROBDD's) [8], [9] as introduced by Bryant, to represent

and solve our satisfiability formulation. ROBDD's constitute a canonical DAG-based representation for arbitrary Boolean functions, unique for a given variable ordering. In contrast to other search-based satisfiability solution techniques, e.g., [20], a BDD-based solution actually represents *all* possible satisfying assignments explicitly. The obvious disadvantage here is the size of the resulting BDD, which in our application represents explicitly all possible routing solutions for all the nets in a region of FPGA fabric. The advantages, however, are twofold.

1) The structure of the BDD itself offers insight into the structure of the routing problem. A large BDD means a large number of routing solutions are possible; a small BDD means few feasible routings. (Interestingly, when the number of solutions grows very large, the size of the BDD again begins to decrease since it becomes easier to represent the *smaller* set of nonroutable assignments; see [31].) A null "0" BDD of course means *no* solution. This was in fact our original motivation for pursuing satisfiability as a routability *estimation* strategy.
2) BDD-based representation allows us to perturb the routing solution incrementally, by moving pins on the entry/exit boundary conditions of the routing region. Since all possible routes are represented explicitly, all interactions are correctly captured when perturbations are to be made. (See Section IV.)

Since BDD's can grow quite large for some functions, and are extremely sensitive to the order in which intermediate results are computed and the global variable ordering, several issues had to be addressed.

In the examples shown so far, nothing has been said about the *order* of final intersection of all the constraints for a particular BDD. To keep the BDD's of reasonable size while they are being assembled, it is important to intersect the most restrictive constraints first. So, in the process of generating the final Boolean function (which itself may ultimately be of manageable size) intermediate results may grow too large if the exclusivity constraints are intersected first because these are least restrictive. For this reason, the connectivity constraints, which are generally more restrictive, are ANDed together first, followed by the exclusivity constraints. This improvement proved to be sufficient for most designs. However, for some channels of the most complex designs, another level of constraint ordering had to be used.

The set of connectivity constraints can be broken down into two classes: those that involve two-point connections both terminating on CLB's, and those that do not. Because C-blocks are typically much more flexible than S-blocks, the first class of connections are much less restrictive; so, they should be ANDed in after the second class but still before the exclusivity constraints. Returning to the simple example of Fig. 3, if we AND the constraints in the example in this order: $\mathrm{Excl}_i, \mathrm{ConnD}_i, \mathrm{ConnC}_i, \mathrm{ConnB}_i, \mathrm{ConnA}_{3,i}, \mathrm{ConnA}_{2,i}, \mathrm{ConnA}_{1,i}$ which corresponds to a monotonic decrease in flexibility and, hence, the worst possible ordering, the intermediate BDD sizes are 226, 217, 179, 82, 35, 23, and 26 nodes. In contrast, when the constraints are intersected in the opposite order, the BDD grows as 3, 6, 9, 12, 14, 15, and finally 26 nodes. Thus, as opposed to the poor constraint ordering, which requires space to represent an intermediate BDD of size 226, the intelligent ordering only ramps up to a BDD of maximum size 26, the final solution size. This difference would be even more significant if a nonoptimal variable ordering were chosen.

Fig. 5 shows the connectivity constraint BDD's followed by the final routability BDD for the example of Fig. 3. The BDD corresponding to the exclusivity constraint is not shown because it is far too large to fit. All edges incident to the "0" node along with the "0" node for the routability constraint are omitted to avoid clutter. (Note that in actual implementation we use a standard multirooted DAG representation with negation arcs [5] to minimize storage; the examples are shown in the simpler nonshared style for clarity.)

As the constraints are intersected in the optimal order, the intermediate solutions are just a concatenation of the small BDD's until the exclusivity constraint is intersected. Then the BDD has the potential to grow substantially depending on the variable ordering and the number of feasible solutions. In this example, it does not grow very much because there are only two interacting two-point connections that have multiple solutions, and the bit vectors corresponding to these two connections happen to be adjacent to each other in the ordering.

In addition to constraint ordering, we must also be concerned with BDD variable ordering. Unfortunately, the size of a BDD is also known to be extremely sensitive to its variable ordering. As a result, there has been extensive research into determining an optimal ordering for a given BDD representation. This application is no exception.

In [2], Berman showed that the size of a single function BDD can be bounded from above by the expression $n2^{\omega_T(\chi)}$ where n is the number of inputs of the representative circuit χ and $\omega_T(\chi)$ is the so-called T-*width* of the circuit under some topological ordering of the gates, T. $\omega_T(\chi)$ is simply the largest number of connections—also called the bandwidth—cut by any partition on this linear ordering; Fig. 6 illustrates the idea for a gate-level version of the exclusivity constraint of Fig. 4. We omit the "i" channel subscript on the variables for simplicity here. The top of the figure illustrates the ordering $T = A_2, A_3, C, B, D$ and the corresponding T-width is three. The resulting BDD would have no more than 5×2^3 nodes. The bottom circuit in the figure shows a second ordering $T' = D, C, A_3, B, A_2$ that achieves the width of 1. So it follows that this BDD should require no more than 5×2^1 nodes. Because the size of the BDD grows super-polynomially with the T-width, one heuristic for BDD ordering is to select an initial ordering with minimal T-width.

If we actually regard these linear orderings as placements in the sense of classical physical design, then the total amount of wire required to "wire" one of these channels can be approximated by summing the number of wires crossing the cut between each of the gates. So, in the first circuit, this would be $3 + 3 + 1 = 7$ while on the bottom, it would be $1 + 1 + 1 = 3$. In our application, we can use this as the starting point for a novel domain-specific variable ordering heuristic: rather than try to create a gate ordering with optimal

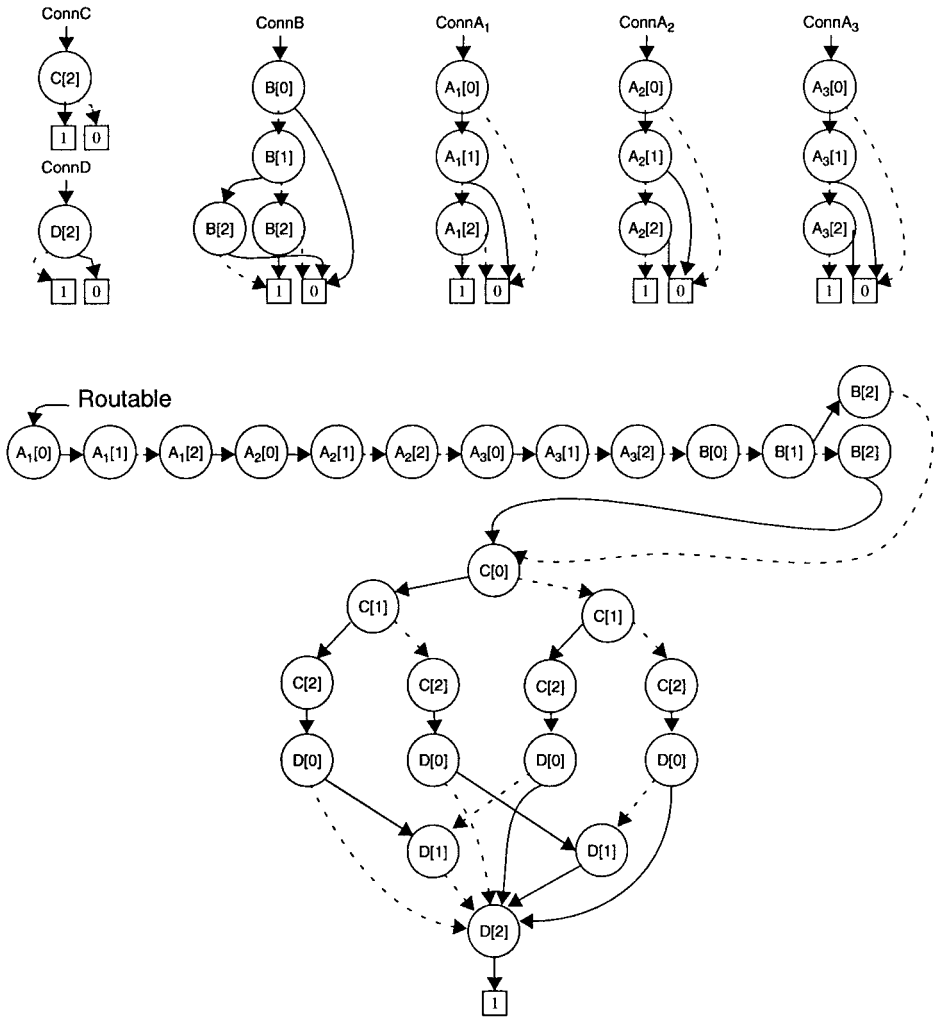


Fig. 5. BDD's for connectivity and routability constraints for channel of Fig. 3.

T-width, we recast the problem as a placement problem and strive for a placement with minimal wirelength. This heuristic has three basic assumptions.

1) We do a coarse-grain variable ordering in which we assume that the bit-vectors themselves are the atomic elements to be ordered, not the individual variables comprising each vector. This ordering is accomplished via a linear placement heuristic.
2) We assume that the individual variables in each vector will be contiguously ordered, based on their index, once the overall coarse ordering is found.
3) We only order the variables based on the structure of the single aggregate exclusivity constraint for the region to be routed.

Hall in [16] first noted that the optimal placement of connected point objects in an R-dimensional space corresponds to the R smallest eigenvectors of the Laplacian of the graph of interest. The Laplacian of a graph is a function its $N \times N$ adjacency matrix, $C = [c_{ij}]$. Let the degree matrix, D, be the $N \times N$ matrix with zeroes as nondiagonal elements and diagonal elements having values equal to the sum of the corresponding row in the adjacency matrix. Then, the Laplacian L is just $D - C$. We are interested in the *linear minimum total quadratic wirelength placement* problem which only requires computation of the eigenvector corresponding to the smallest nonzero eigenvalue. This eigenvalue will then correspond to the total wirelength. The eigenvector represents an embedding of the nodes in a 1-D space such that quadratic wirelength (e.g., $\Sigma_{i,j}\ c_{ij}(x_i - x_j)^2$) is directly minimized, under the constraint that the variance of the coordinates is equal to one. Our heuristic assumes that this "wire" minimization also *indirectly* minimizes the T-width of the circuit; this is similar to the assumptions underlying spectral partitioning techniques, e.g., [17]. The virtue of this formulation in our case is that it offers a simple and direct method for the creation of the initial variable ordering.

In our BDD-based routing, this method is used to find a suitable ordering on the bit-vectors representing each portion of each net, but not on the individual element variables of each of these vectors. Referring again to the two orderings illustrated in Fig. 6, note that each circuit corresponds to a gate-level version of the exclusivity constraint of Fig. 4 at the level of the bit-vectors for each net. The objective of the linear placement is to order these gates—and thus the variables—on a line such that wirelength is minimized. Here, variables

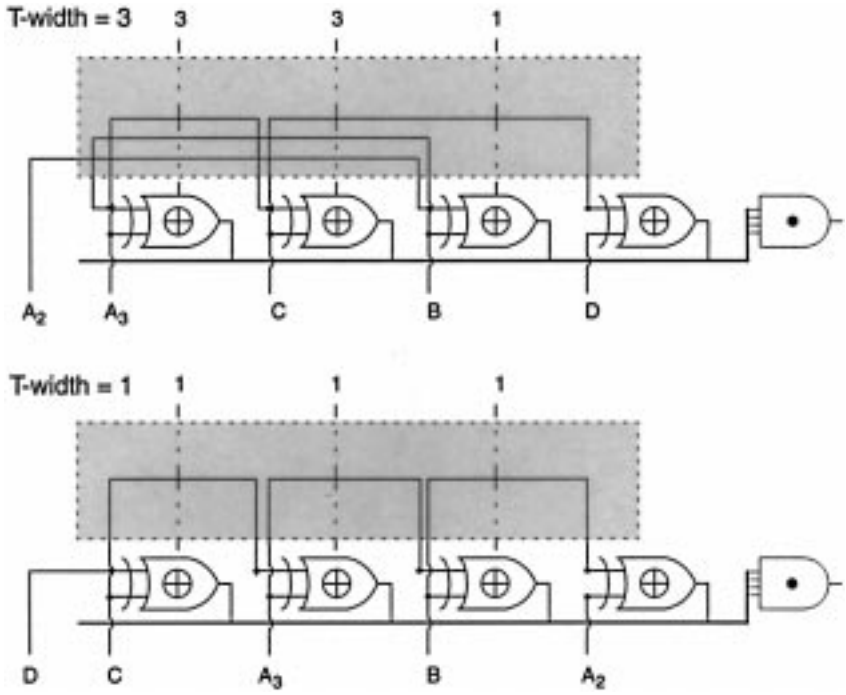


Fig. 6. T-width for two topological gate-level orders for exclusivity constraint of Fig. 4.

are actually bit vectors, and exclusive-or gates are actually vectored exclusive-or gates. The eigenvector corresponding to the smallest nonzero eigenvalue represents the x-locations of where the bit-vectors should be placed on a line between $[-1, 1]$ so that if arcs were drawn between interacting bit vectors, then the sum of the squares of the lengths of each arc is minimum. Such a quadratically minimum ordering for these variables is

$$\vec{D}, \vec{C}, \vec{A_3}, \vec{B}, \vec{A_2}, \vec{A_1}.$$

To derive the final ordering for all the relevant BDD variables, we assume that the individual component variables comprising each bit vector remain contiguous under this ordering. In other words, the global structure of the variable ordering is determined by the eigensolution, and the local structure is determined by the index on each component variable in each bit vector.

Finally, note that our T-width construction targets only the empirically observed structure of the BDD's created to represent just one type of Boolean constraint, the exclusivity constraint. It can be intuitively seen by observing the example BDD's in Fig. 5 that the connectivity constraints are smaller when the bit vector variables are contiguous (assumption 2, above). The string of nodes running horizontally in the routability BDD in this picture can be roughly thought of as the component of the final BDD that comes from connectivity constraints, while the dense, "bushy" part at the bottom contains the *exclusivity* information. Our heuristic works to optimize the structure of this exclusivity-related part of the DAG (assumption 3).

Empirically, this scheme was found to work reasonably well for our BDD-based routing problem. It performed significantly better than other *ad hoc*, domain-specific ordering techniques we tried, and rendered final BDD's 10–100 times smaller (as would be expected) than a random ordering on the examples we tried Several additional implementation details appear in [31].

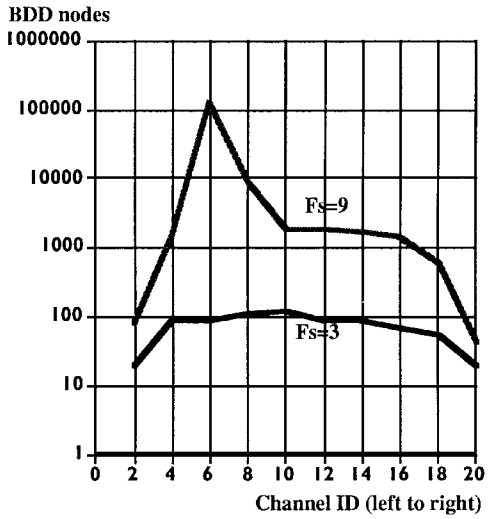


Fig. 7. BDD size by channel for benchmark 9 symml and for flexibility $Fs = 3$ and $Fs = 9$.

IV. Incremental Updates of Routing

As mentioned earlier, to fit this technique into the context of a complete routing solution, e.g., a perfect congestion estimator for a global router, our satisfiability formulation must have the added quality that it can be quickly updated. As it turns out, it is easy to incrementally update the Boolean formulation for the routing of a region when we use BDD's as the basic representation. The central questions are the following.

1) How do we *remove* a net from the routability function?
2) How do we *add* a new net at new port/block locations?

Consider the following situation (refer again to Fig. 3). Two-point connection $\vec{A}_{3,i}$ is modified from its original triplet pair of $(A,(6,i-1,\text{east},3),\text{Entrance})$, $(A,(10,i+1,\text{west},4),\text{Exit})$ to $(A,(8,i-1,\text{west},4),\text{Entrance})$, $(A,(10,i+1,\text{west},4),\text{Exit})$. Clearly, this new connection impacts the feasibility constraints. So, first, the previous bit vector associated with the connection, $\vec{A}_{3,i}$ is *smoothed* [9] out of the BDD by computing a new BDD which has the same onset as the previous for *any value* of $\vec{A}_{3,i}$. This is just the *existential quantification* operation over all of the bits in the bit vector $\vec{A}_{3,i}$. The *new* routability function for this region of FPGA fabric, *minus* this net, is just

$$\left(\exists \vec{A}_{3,i}\ \text{routable}_i\right)\left(\vec{A}_{1,i},\vec{A}_{2,i},\vec{B}_i,\vec{C}_i,\vec{D}_i\right)$$

which can be computed in time proportional to the number of nodes in the BDD [9]. Then, the new connectivity constraint is generated by simply adding (via AND, OR, EXOR's) the clauses that represent interaction with the new net at its new location. Finally, the new exclusivity constraints are built corresponding to all of the variables that $\vec{A}_{3,i}$ intersects with. These are intersected with the existentially quantified root BDD in the order just described.

The result is that we can complete a geometric perturbation using only Boolean operations: we can erase a connection specified by a particular set of boundary conditions, and then perturb those boundary conditions to specify a new connection. This is a singular advantage of a BDD-based formulation of the routing problem: because we represent explicitly *all* possible routing solutions for the region, *all* the necessary information for correct and efficient perturbations of the geometric boundary constraints is already available to us. Assuming that we can afford the time and space to build the initial BDD's for the satisfiability constraints for a given region, subsequent exploration of "nearby" routing solutions, obtained by moving pins, adding/deleting individual nets, etc., can be done quite efficiently.

V. Experimental Results

All experiments were conducted on an IBM Power Series 850 based on a 100-MHz PPC604 chip running AIX 4.1.3 and equipped with 64 MB of RAM. All coding was done in C++, and our satisfiability-based region routing code amounted to roughly 7500 lines. A global router of the type in [6] and [19] produced coarse graphs. To generate triple sets for each of the channels, a simple left-edge channel routing algorithm was used to assign nets to tracks followed by backmapping of tracks to channel border pins assuming diagonal switch blocks. Because the channel is fully segmented, the left-edge algorithm produces an optimal track assignment [15]. Of course, due to fact that the channels are considered separately, this does not represent a legitimate route *globally*; however, for the purposes of experimentation, it sufficiently represents the class of triple sets that our satisfiability-based region router is likely to encounter. The BDD package from [5] was used. Each BDD node requires 18–22 bytes depending upon how close the BDD is to the maximum memory limit. Simplified Xilinx-style FPGA's were investigated with varying switch block architectures and array dimensions, no long-line global interconnect, and a constant C-block architecture with $Fc = W$.

TABLE I
Benchmark Circuits

Benchmark	Dimension rows X cols	W	# Nets	# 2-point connects
9symml	10 x 11	9	79	259
alu2	13 x 14	10	153	511
alu4	17 x 19	13	256	851
apex7	10 x 12	12	126	300
example2	12 x 14	16	205	444
2large	14 x 15	11	145	519
k2	20 x 22	16	404	1256
vda	16 x 17	14	225	722

Table I describes the benchmarks. The benchmarks are those from Alexander [1] at the University of Virginia, Charlottesville. W is the minimum C-block track count that the global router has determined can be used to complete the detailed embedding of its coarse graphs. We use this in our subsequent experiments to make the satisfiability problems suitably challenging: if we use many fewer than W tracks per channel, the regions are trivially unroutable and each BDD quickly collapses to the null "0" BDD. Conversely, if we use many more than W tracks, the regions are easily routable, and the BDD-based formulation simply produces very large representations of the constraints. Neither of these scenarios require the use of an exact estimator for routability.

Fig. 7 presents the number of nodes required to represent each of the BDD's for each of the channels for the single benchmark 9symml. The bottom curve corresponds to an FPGA with $W = 9$ and diagonal switch blocks, i.e., $Fs = 3$ and each pin has only one option for turning to each of the other three sides. The top curve plots the same information for a switch block architecture with $Fs = 9$. Notice how some of the channel BDD's increase in size greatly. This is due to the fact that their satisfiability sets increased in size with the more flexible S-blocks. Since our BDD representation explicitly represents all routing solutions, it grows in size with the flexibility of the FPGA fabric.

Table II describes the result of an experiment in which we build routability BDD's for each channel for each of the benchmarks, assuming that $Fs = 3$ for each switchbox.The table offers the following data.

- Column 2 is the total BDD size (BDD nodes) when summed over all of the channels in each FPGA. Large problems with more nets have, unsurprisingly, larger BDD's. When the constraints are ordered carefully as described in Section III, the intermediate sizes of the BDD's tend to increase monotonically to this final size.
- Column 3 gives the mean time required for an incremental routing update averaged over 20 random triple set perturbations. This means the geometric boundary

TABLE II
AGGREGATE BENCHMARK RESULTS

Benchmark	Total BDD Nodes	Incremental Time	Initial Time
9symml	740	12.9 ms	105 ms
alu2	2069	25.5 ms	229 ms
alu4	3542	31.1 ms	448 ms
apex7	1559	15.6 ms	166 ms
example2	7014	84 ms	439 ms
2large	2582	25.3 ms	301 ms
k2	1229464	50.5 ms	1238 ms
vda	10444	46.1 ms	726 ms

TABLE III
AGGREGATE RESULTS; MORE FLEXIBLE SWITCH BLOCKS

Benchmark	Total BDD Size	Incremental Time	Initial Time
9symml	6202	35.8 ms	245 ms
alu4	29561	183.3 ms	2.54 s
apex7	11016	75 ms	537 ms
term1	6867	55 ms	380 ms
example2	171377	2.37 s	22.3 s
2large	31003	198 ms	2.76 s
k2*	N.C.	N.C.	N.C.
vda*	169702	4.25 s	96.0 s

conditions for signal entry/exit to each channel were randomly perturbed (in this case, pins were swapped) 20 times. Note that given the complete BDD for all the routability constraints, this is quite fast.

- Finally, column 4 shows the mean time required to build the initial routability BDD's for each channel, averaged over all of the channels in each design. Once again, this assumes $Fs = 3$, and W is taken from Table I.

Perhaps the most interesting observation from this data is simply the speed with which these satisfiability formulations capture all routing solutions for all the channels in these designs: typically 100–1000 ms to create the suite of BDD solutions for each channel; typically 10–100 ms to perturb a geometric boundary condition on pins on any channel.

It is also important to note how the above parameters vary with more flexible routing architectures. Table III repeats Table II, but uses FPGA's possessing switch blocks with $Fs = 6$ (much like those in Fig. 3). As can be seen, once again, the problem complexity grows quickly with more flexible interconnect structure, with some examples taking as much as 24× more space and 28–50× more time for incremental/initial BDD build time. (Note that vda and k2 had to be run on a different machine that had more memory but also had an extremely high process load; so, the times are not comparable.) Still k2 exhausted all memory and was not able to complete. Even here, the worst case set of total routing solutions we constructed required under 2 min.

This experiment nicely captures the tradeoffs of the BDD-based solution strategy. Since we represent explicitly all possible routing solutions, incremental geometric perturbations (of the type a global router might be expected to attempt) are efficient, fast, and exact in the sense that the new BDD again represents all possible solutions under the new boundary conditions. However, some very large problems—with many nets or tracks, or large flexibility—may require BDD's that are difficult to construct. If we need only to answer the question "is there any routing for this region?" we might try other search-based techniques for satisfiability that do not represent all solutions [20]. Of course, if we merely want to find *one* good routing solution for a segmented channel in an FPGA, many techniques exist, e.g., [15], [26] which do not represent *explicitly* all solutions. Also, while the channel abstraction is convenient, nothing in the satisfiability-based strategy precludes other region shapes than channels. We note that alternative decision diagram structures and construction techniques exist which may be more suited to these large satisfiability problems. The side effect of the satisfiability-based strategy that we find most intriguing is the possibility of using the *structure* of the rendered BDD itself—for example, its size, or its density of satisfying assignments—as an estimator for the *difficulty* of the final detailed routing problem for the region. This remains an interesting open problem for us.

VI. CONCLUSIONS

In this paper we developed a novel formulation of the FPGA routing and routability estimation problems that reduced these problems to Boolean Satisfiability, extending early ideas from [12]. Given an FPGA routing architecture, a region of the routing fabric, and boundary constraints on signal I/O's and coarse paths as produced by a global router, this formulation can answer exactly the question "can we route these nets in this region?" By representing the resulting satisfiability problem using BDD's, we can not only determine exact routability, but also determine all feasible routing assignments for nets in the region, and support incremental perturbations of the global routing constraints. A preliminary implementation suggests that the technique is workable, but that great care must be taken in the construction and solution of the satisfiability problem, since real FPGA's can easily generate very large satisfiability problems.

One obvious application of such a technique is as a perfect routability estimator for a global router. It is worth noting, however, how very different is the behavior of a satisfiability-based estimator. When the routing flexibility is low, when space is tight, when signals contend for limited paths, the estimator works *best*. This is because the number of feasible routing alternatives is small, which makes for simpler BDD's. When routing is highly flexible, space is easily available, nets interact little, the estimator works poorly since all paths must be represented and there are *many* feasible paths. But in such circumstances routing is generally easier, and estimation is less critical. Another application is estimating the routability of a

"difficult" bounded region of a larger design with a substantial number of net incompletions. This technique may make it possible to make more intelligent decisions about whether to try more vigorous routing (if the region is routable), or simply replace this region (if it is intrinsically unroutable).

In summary, we believe that this formulation is both and elegant and potentially practical. However, there is significant opportunity for improvement in our implementation. Improvements in net-to-resource variable encoding (e.g., [21]), Boolean constraint clause ordering, decision diagram variable ordering (notably dynamic ordering [27]), and decision diagram structure (e.g., smaller structures such as free-BDD's [4]) could have a substantial improvement on the range of practical application for the technique.

ACKNOWLEDGMENT

The authors are grateful to R. Bryant of Carnegie Mellon University, Pittsburgh, PA, for the BDD package and for several enlightening conversations about satisfiability. They are also grateful to the reviewers for several helpful comments that improved the quality of this presentation.

REFERENCES

[1] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins, "An architecture-independent approach to FPGA routing based on multi-weighted graphs," in *Proc. European Design Automat. Conf.*, Sept. 1994, pp. 259–264.
[2] C. L. Berman, "Ordered binary decision diagrams and circuit structure," *Proc. IEEE ICCD*, Oct. 1989, pp. 392–395.
[3] N. Bhatt and D. Hill, "Routable technology mapping for FPGA's," in *Proc. FPGA Workshop*, 1992.
[4] J. Bern, C. Meinel, and A. Slobodova, "Efficient OBDD-based Boolean manipulation in CAD beyond current limits," in *Proc. ACM/IEEE DAC*, June 1995, pp. 408–413.
[5] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. ACM/IEEE DAC*, June 1990, pp. 40–45.
[6] S. Brown, J. Rose, and Z. G. Vranesic, "A detailed router for field programmable gate arrays," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 620–628, May 1992.
[7] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*. Boston, MA: Kluwer-Academic, 1992.
[8] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, 1986, pp. 677–691.
[9] ——, "Symbolic Boolean manipulation with ordered binary decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, Sept. 1992.
[10] P. K. Chan *et al.*, "On routability prediction for field programmable gate arrays," in *Proc. IEEE DAC*, June 1993.
[11] Y-W. Chang, S. Thakur, K. Zhu, and D. F. Wong, "A new global routing algorithm for FPGA's," in *Proc. ACM/IEEE ICCAD*, 1994, pp. 356–361.
[12] S. Devadas, "Optimal layout via Boolean satisfiability," in *Proc. ACM/IEEE ICCAD*, 1989, pp. 294–297.
[13] J. Frankle, "Iterative and adaptive slack allocation for performancedriven layout and FPGA routing," in *Proc. ACM/IEEE DAC*, 1992.
[14] H. Fujii, G. Ootomo, and C. Hori, "Interleaving based variable ordering for ordered binary decision diagrams," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 38–41.
[15] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, "Segmented channel routing," in *Proc. ACM/IEEE DAC*, 1990, pp. 567–572.
[16] K. M. Hall, "An R-dimensional quadratic placement algorithm," *Management Sci.*, vol. 17, pp. 219–229, Nov. 1970.
[17] L. Hagen and A. B. Kahng, "A new approach to effective circuit clustering," *IEEE Trans. Computer-Aided Design*, vol. 11, Sept. 1992.
[18] H. Krupnova, C. Rabedaoro, and G. Saucier, "Synthesis and floorplanning for large hierarchical FPGA's," in *Proc. ACM Int. Symp. FPGAs*, Feb. 1997.
[19] G. Lemieux and S. Brown, "A detailed router for allocating wire segments in FPGA's," *Proc. ACM Physical Design Workshop*, Apr. 1993.
[20] J. P. Marques Silva and K. A. Sakallah, "GRASP—A new search algorithm for satisfiability," in *Proc. ACM/IEEE ICCAD*, Nov. 1997.
[21] S-i. Minato, "Zero-suppressed BDD's for set manipulation in combinatorial problems," in *Proc. ACM/IEEE DAC*, June 1993, pp. 272–277.
[22] L. E. McMurchie and C. Ebeling, "PathFinder: A negotiation-based path-driven router for FPGA's," in *Proc. ACM/IEEE Int. Symp. Field Programmable Gate Arrays*, Feb. 1995.
[23] S. K. Nag and R. A. Rutenbar, "Performance-driven simultaneous place and route for island-style FPGA's," in *Proc. ACM/IEEE ICCAD*, Nov. 1995.
[24] S. K. Nag and R. A. Rutenbar, "Performance-driven simultaneous place and route for row-based FPGA's," in *Proc. ACM/IEEE DAC*, June 1994.
[25] R. Rao, "A global router for channelled FPGA's," in *Proc. MCNC Physic. Design Workshop*, 1992.
[26] K. Roy, "Detailed routing for row-based FPGA's," *IEEE Trans. Computer-Aided Design*, 1994.
[27] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. ACM/IEEE ICCAD*, Nov. 1993, pp. 42–47.
[28] J. Shi and D. Bhatia, "Performance driven floorplanning for FPGA based designs," in *Proc. ACM Int. Symp. FPGAs*, Feb. 1997.
[29] S. Thakur, Y.-W. Chang, D. F. Wong, and S. Muthukrishnan, "Algorithms for an FPGA switch module problem with applications to global routing," *IEEE Trans. Computer-Aided Design*, vol. 16, Jan. 1997.
[30] R. G. Wood and R. A. Rutenbar, "FPGA routing and routability estimation via Boolean satisfiability," in *Proc. ACM Int. Symp. FPGAs*, Feb. 1997.
[31] R. Glenn Wood, "Routing for FPGA's via Boolean satisfiability," M.S. thesis, Dep. Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, May 1996.
[32] Y-L. Wu and M. Marek-Sadowska, "Graph based analysis of FPGA routing," in *Proc. European Design Auto. Conf.*, 1993, pp. 104–109.
[33] Y-L. Wu and D. Chang, "On the NP-completeness of regular 2-D FPGA routing architectures and a novel solution," in *Proc. ACM/IEEE ICCAD*, 1994, pp. 362–367.

R. Glenn Wood received the Bachelor's degree in electrical engineering from the University of Texas, Austin, in 1994. He received the Master's degree in electrical engineering from Carnegie Mellon University (CMU), Pittsburgh, PA, in 1996, where he studied physical design automation.

He is currently employed by Hewlett-Packard Colorado Springs Division, Colorado Springs, CO, where he designs high-speed logic analysis systems.

Rob A. Rutenbar (S'77–M'84–SM'90–F'98) received the Ph.D. degree from the University of Michigan, Ann Arbor, in 1984.

He subsequently joined the faculty of Carnegie Mellon University (CMU), Pittsburgh, PA. He is currently Professor of Electrical and Computer Engineering, and (by courtesy) of Computer Science, and since 1993, Director of the CMU Center for Electronic Design Automation. His research interests focus on circuit and layout synthesis algorithms for mixed-signal ASIC's, high-speed digital systems, and FPGA's.

Dr. Rutenbar received a Presidential Young Investigator Award from the National Science Foundation in 1987. He has been on the program committees for the IEEE International Conference on Computer-Aided Design, the ACM/IEEE Design Automation Conference and European Design and Test Conference, and the Editorial Board of IEEE SPECTRUM. He was General Chair of the 1996 ICCAD, and is currently on the program committee of the ACM International Symposium on FPGA's. He Chaired the Analog Technical Advisory Board for Cadence Design Systems from 1992 through 1996. He is a member of the ACM.