A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints

Gi-Joon Nam, *Member*, *IEEE*, Fadi Aloul,
Karem A. Sakallah, *Fellow*, *IEEE*, and Rob A. Rutenbar, *Fellow*, *IEEE*

Abstract—This paper presents empirical analyses of two Boolean Satisfiability (SAT) formulations of FPGA (Field Programmable Gate Array) detailed routing constraints. Boolean SAT-based routing transforms a routing problem into a Boolean SAT instance by rendering geometric routing constraints as an atomic Boolean function. The generated Boolean function is satisfiable if and only if the corresponding routing is possible. Two different Boolean SAT-based routing models are analyzed: the *track-based* and the *route-based* routing constraint model. The track-based routing model transforms a routing task into a net-to-track assignment problem, whereas the route-based routing model reduces it into a routability-checking problem with explicitly enumerated set of detailed routes for nets. In both models, routing constraints are represented as CNF Boolean Satisfiability clauses. Through comparative experiments, we demonstrate that the route-based formulation yields an easier-to-evaluate and more scalable routability Boolean function than the track-based method. This is empirical evidence that a smart/efficient Boolean formulation can achieve significant performance improvement in real-world applications.

Index Terms—Boolean Satisfiability, FPGAs, routing, physical design.

1 Introduction

The Boolean Satisfiability problem (SAT) involves finding an assignment to binary variables that satisfies a given set of Boolean constraints. Typically, these constraints are presented in CNF (Conjunctive Normal Form), where each constraint is a disjunction of literals (either positive or negative instances of the binary variables). The Boolean Satisfiability is one of the most fundamental problems in computer science and a variety of algorithms—including the well-known *search-based SAT algorithms*—have been proposed. The search-based Boolean SAT solvers explore the Boolean space of the input variables to find just one satisfying assignment, or search exhaustively to conclude that no satisfying assignment exists. Most search-based solvers are basically variations of the Davis-Putnam procedure [7], [8] and the best well-known version is based on a backtracking search algorithm that, at each node in the search tree, elects an assignment and prunes subsequent search by iteratively applying *unit clause* and *pure literal* rules [31]. These algorithmic advances led to the appearance of numerous systematic and complete Boolean SAT solvers such as GRASP [18], RelSAT [3], SATO [32], Li's solver [17], BerkMin [11], Chaff [20], and made remarkable contributions to various fields including EDA (Electronic Design Automation). A variety of problems in the EDA domain—combinational equivalence checking [13], automatic test pattern generation [14], processor verification [30], bounded model checking [5], FPGA detailed routing [21], etc.—were already tackled with efficient Boolean SAT solvers and showed great promises.

In this paper, we attack one of EDA problems; FPGA detailed routing via Boolean SAT. This method is called a Boolean SAT-based routing. Two Boolean SAT-based routing models are presented: the *track-based* and the *route-based* routing constraint model. In both models, routing constraints are represented as CNF Boolean Satisfiability clauses. Through comparative experiments, we demonstrate that the route-based formulation yields an easier-to-evaluate and more scalable routability Boolean function than the track-based method. This is empirical evidence that a smart/efficient Boolean formulation can achieve significant performance improvement in real-world applications. We also show that the *route-based* FPGA SAT routing produces very competitive routing results compared to traditional *one-net-at-a-time* routing approaches.

The rest of the paper is organized as follows: Section 2 reviews FPGA routing terminologies, conventional FPGA routing methods, and the basic concept of Boolean SAT-based routing as preliminaries. In Section 3, the track-based and route-based routing formulations are described in detail. The comparative experimental results and analyses are presented in Section 4 and the conclusion will follow in Section 5.

2 Preliminaries

Routing is the final process of VLSI design implementation and is concerned with assigning appropriate routing

- *G.-J. Nam is with IBM Austin Research Lab, Bldg. 904-6H006, 1140 Burnet Rd., Austin, TX 78758. E-mail: gnam@us.ibm.com.*
- *F. Aloul is with the School of Computer Engineering, American University, Dubai, UAE. E-mail: faloul@aud.edu.*
- *K.A. Sakallah is with the Department of Electrical Engineering and Computer Science, 1301 Beal Ave., Ann Arbor, MI 48109. E-mail: karem@umich.edu.*
- *R.A. Rutenbar is with the Department of Electrical and Computer Engineering, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: rutenbar@ece.cmu.edu.*

Manuscript received 6 June 2003; revised 6 Nov. 2003; accepted 16 Jan. 2004. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0057-0603.

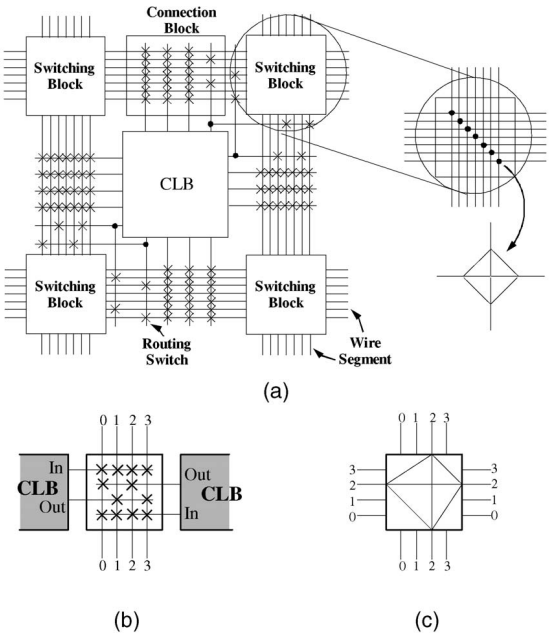


Fig. 1. FPGA routing configuration and its modeling. (a) FPGA routing configuration. (b) Connect block model. Input logic pin: Fc = W = 4. Output logic pin: FC = 2. (c) Switching block model: Fc = 3.

resources (i.e., metal wire segments and vias) to each net in the placed netlist to establish the required interconnections [25]. In this paper, we focus on Boolean SAT-based detailed routing formulations on FPGAs (Field Programmable Gate Arrays). To facilitate understanding of the rest of the paper, we first introduce the modeling of FPGA routing architecture and conventional FPGA routing methods. Then, the general concept of Boolean SAT-based routing formulation is presented.

2.1 FPGA Architecture and Associated Terminology

A Field-Programmable Gate Array (FPGA) is a programmable logic device that can implement a variety of circuit functions. An FPGA consists of an array of prefabricated logic blocks and wiring resources which can be easily configured by end users. The routing configuration of an FPGA is depicted in Fig. 1. FPGA can be modeled as a two-dimensional array of *Configurable Logic Blocks* (*CLBs*), *Connection Blocks* (*C-Blocks*), and *Switching Blocks* (*S-Blocks*). CLBs are the places where any arbitrary functionality of a circuit is implemented with combinational and sequential logics. A vertical (horizontal) channel is defined as a set of *tracks* between two consecutive columns (rows) of CLBs; wire segments connecting CLB pins are aligned into tracks running in the channel. C-blocks and S-blocks contain programmable switches and form the routing resources: C-blocks connect CLB pins to channel tracks; S-blocks are surrounded by C-blocks and allow signals to either pass straight through or to make 90-degree turns. Thus, C-blocks and S-blocks form the basic routing resources. I/O blocks reside on the boundary of the array. The routing capacity of a given FPGA architecture is conveniently expressed by three parameters, *W*, *Fc*, *Fs* [6]. The channel width *W* is the number of tracks in a vertical or horizontal channel. The C-block flexibility *Fc* (<= *W*) is the number of tracks in adjacent channels that each CLB logic pin may connect to. The S-block flexibility *Fs* is the total number of other tracks

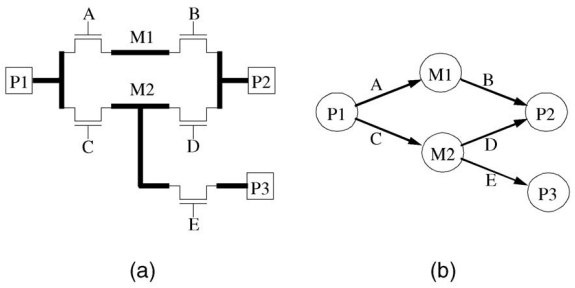


Fig. 2. Routing graph representation in conventional FPGA routers. The bold lines in (a) represent metal wire segments. (a) FPGA circuit segment. (b) Routing graph.

that each wire segment entering an S-block can connect to. For the example FPGA in Fig. 1b and Fig. 1c, these parameters are $W = 4$, $Fc = 2$ for output pins, $Fc = 4$ for input pins, and $Fs = 3$. For the more detailed FPGA architectures, please refer to [6].

A *net* is a set of CLB and/or I/O pins that must be electrically connected and consists of a source (driver) pin and one or more sink pins. In case a net has *n* sink pins, this net can be further decomposed into *n* different (source pin, sink pin) pairs which we call *two-pin connections*. A *global route* of a two-pin connection is a specification of routing regions that forms an uninterrupted alternating sequence of C and S-blocks. A global route of a two-pin connection is further decomposed into one or more horizontal and/or vertical *net-segments*, each of which is an alternating sequence of C and S-blocks within a single channel. The *detailed route* of a two-pin connection is a set of wire segments and routing switches within the restricted routing area determined by the global router. Thus, a detailed router has to assign wire segments and corresponding routing switches following the topology specified by the global router such that the detailed routes of different nets do not overlap.

2.2 Overview of Conventional FPGA Detailed Routers

Most conventional FPGA routers start by constructing a routing graph whose topology mirrors the complete FPGA routing architecture. Fig. 2a shows part of the routing fabric in an FPGA. It has three logic pins (P1, P2, and P3) and five transistors that provide programmable interconnection points (PIPs) between wire segments (metal conductors) shown in bold lines. Fig. 2b illustrates the corresponding routing graphs: Logic pins and wire segments are represented as vertices that are connected by edges which represent the programmable transistors. Thus, paths in this graph correspond to feasible routes in the FPGA. Using this graph, conventional routers use sophisticated cost functions to search for routing solutions (i.e., paths in the graph) for all the nets in a circuit.

For example, PathFinder [19], which is one of the most advanced FPGA routers in the literature, uses an iterative algorithm that attempts to balance the competing goals of eliminating congestion and minimizing delay of critical paths. This algorithm initially routes each net using the shortest (i.e., lowest cost) path algorithm. The cost of a routing resource consists of two components: overusage history and its propagation delay. At the first iteration, any routing resources can be shared among different nets. However, any

resource sharing (overusage) affects the cost of the corresponding resource in the subsequent iteration. By gradually increasing the cost of overused resources, PathFinder forces nets to take alternative detour routes within permissible timing delay budgets. Eventually, only the timing critical net is assigned high cost resources to avoid any detouring. One iteration of the router consists of sequential ripping up and rerouting all the nets in the circuit.

If a routing solution exists for a given circuit and placement, Pathfinder will eventually converge toward it after a sufficient number of iterations. If the problem is unroutable, however, the algorithm may not converge and execution must be aborted after a suitable time limit. This convergence problem is inherent to all conventional *one-net-at-a-time* routing algorithms: These algorithms cannot decide the routability of the given circuit placement unless they explore all the possible ordering of nets. Even when only a few nets are not routed in a circuit, these algorithms must resort to time-consuming rip-up-reroute procedures which may or may not lead to a feasible routing solution.

2.3 Boolean SAT-Based VLSI Routing

Boolean SAT-based routing transforms the geometric routing task into a Boolean Satisfiability (SAT) problem by rendering the routing constraints as an atomic Boolean function. The generated Boolean function is satisfiable (has an assignment of input variables such that the generated function evaluates to constant "1") if and only if the design is routable. Any satisfying assignment to the binary variables of the Boolean function represents a legal routing solution. Moreover, by demonstrating the absence of satisfying assignments for a generated routing Boolean function, we can prove that no routing solution exists.[1] A particular virtue of this method is that much of the geometric complexity of the interaction among objects (i.e., nets in routing) is hidden and rendered implicitly in the Boolean constraint functions so that all the objects are considered simultaneously. In other words, Boolean-based routing is a concurrent method allowing higher degrees of freedom for each object in contrast to the conventional one-net-at-a-time approach.

In spite of these unique properties, the use of Boolean Satisfiability to solve VLSI routing is not as common as other methods using integer linear programming or heuristic search. To the best of our knowledge, Szymanski [27] was the first to establish a link between geometric layouts and Boolean formulations. In this work, he proves that a general dogleg channel routing problem belongs to the *NP-complete* class by reducing the 3-satisfiability problem to it, i.e., given a 3-satisfiability formula, he showed how to construct an instance of the channel routing problem that can be routed in a certain number of tracks if and only if the original formula is satisfiable. Capitalizing on this idea, Devadas [9] devised a formulation of conventional 2-layer channel routing as a generic Boolean SAT problem by encoding the information present in a channel's vertical constraint graph, horizontal constraint graph, and the anticipated channel width into Boolean constraint formulas on a set of n-bit Boolean vectors, one per net to be routed. Thus, if the generated Boolean formula is satisfiable, then any satisfying assignment corresponds to a feasible routing

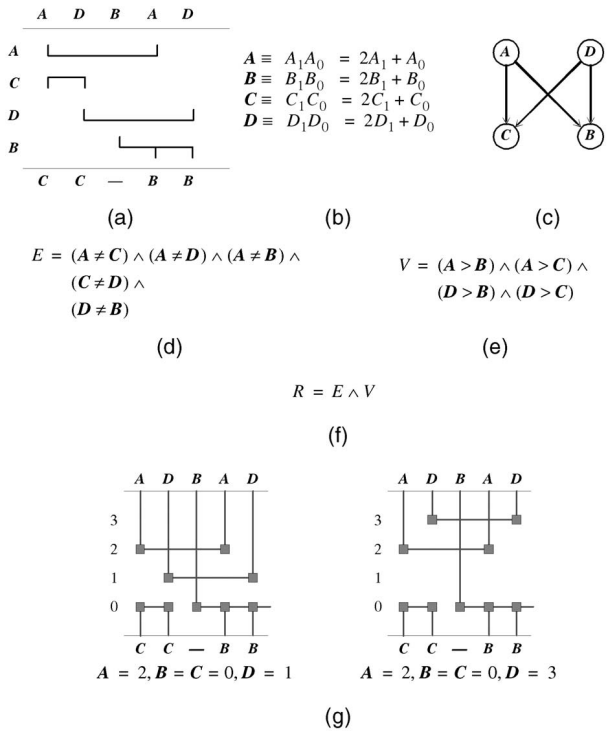


Fig. 3. Boolean SAT modeling of channel routing problem. (a) Channel to be routed. (b) Binary encoding of track numbers. (c) Vertical Constraint Graph (VCG). (d) Exclusivity constraint. (e) vertical ordering constraint. (f) Channel routability constraint. (g) Two feasible solutions.

of the channel; otherwise (i.e., the function is proven to be unsatisfiable) the channel is provably unroutable with the anticipated channel width. Later, Sulimma and Kuntz [26] showed that the grid-based channel routing with the restricted two-layer model is a fixed-parameter tractable problem which can be solved in linear time with the fixed channel length. The idea was to record a net which leaves each track per column while sweeping columns from left to right in the channel. To represent this information, they used MDDs [12].

The core idea of applying a Boolean SAT technique to a simple routing problem is illustrated in Fig. 3. It is a channel routing problem with four nets labeled ***A***, ***B***, ***C***, and ***D***. The goal is to assign a track number to each net such that distinct nets are nonoverlapping both horizontally and vertically. Assuming a 4-track channel, each net needs two binary variables (for example, A_1, A_0 for net ***A***) to encode its track number (Fig. 3b). Two types of constraints are defined to guarantee a legal channel routing solution. First, an *exclusivity constraint* insures that nets whose horizontal spans overlap are assigned to different tracks. This constraint is typically represented by a horizontal constraint graph and can be conveniently expressed as a Boolean function (Fig. 3d). The other constraint insures that, for any two nets with pins in the same column on opposite sides of the channel, the net associated with the top pin is assigned a higher track number. This constraint is conveniently captured by the vertical constraint graph (VCG) shown in Fig. 3c, which in turn is equivalent to the Boolean function V of Fig. 3e. The conjunction (AND) of these two functions is the complete routability constraint Boolean function R for the channel (Fig. 3f). Any binary assignment to the variable $A_1, A_0, \ldots, D_0$ that makes $R = 1$ corresponds to a feasible

1. Therefore, for this application, we need a complete systematic solver rather than a stochastic solver such as WalkSAT [24].

routing solution and completely specifies the net-to-track mapping. Two feasible assignments are shown in Fig. 3g.

Boolean-based routing, in general, has the following advantages over conventional one-net-at-a-time routing approaches:

- **Simultaneous net embedding**: The conventional *one-net-at-a-time* routing approach is notorious for being dependent on net ordering because previously routed nets act as obstacles to the yet-to-be-routed nets. In Boolean-based routing, all routing constraints are considered concurrently by a Boolean SAT solver, making net ordering irrelevant.
- **Routability decision**: The unsatisfiability of the routing Boolean constraint function, as proven by a Boolean SAT solver, directly implies that there is no feasible routing solution with the given placement and global routing configuration. On the other hand, any assignment to the Boolean variable vector that satisfies the routability Boolean function corresponds to a complete feasible detailed routing solution.

However, Boolean SAT-based routing—similarly to other mathematical formulation-based methods such as ILP (Integer Linear Programming)[2]—is known to be less scalable than conventional routing approaches. In other words, the size of problems that can be attacked by Boolean SAT-based routing is smaller than that of conventional routing. This is why conventional heuristic search-based methods are more popular in practice.

3 Boolean SAT-Based FPGA Detailed Routing Formulations

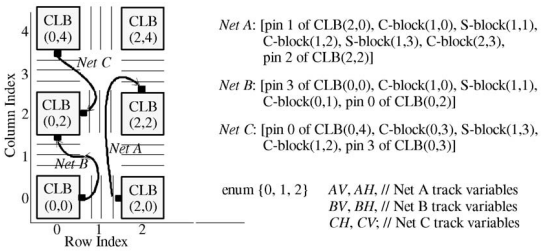
In this section, we review two SAT formulations which are transformed from the exactly same FPGA routing instance. These are called the *track-based* and the *route-based* routing constraint model, respectively.

3.1 Formulation 1: Track-Based Routing Constraint Model

The track-based routing constraint model transforms FPGA detailed routing problem into a net-to-track assignment problem [21]. Each net is represented by a set of "track" variables that indicate the indices of the horizontal and vertical tracks over which the net might be routed. A routability Boolean function is then defined over these variables to enforce two types of constraints:

- **Connectivity constraints** to insure the existence of a conductive path for each two-pin connection through the sequence of C and S-blocks specified by a global router. Basically, these constraints model the routing flexibility available in the C and S-blocks.
- **Exclusivity constraints** to guarantee that electrically distinct nets with overlapping vertical or horizontal spans in the same channel are assigned to different tracks. These constraints are essentially instances of channel routing problems in Fig. 3.

2. We have performed another comparative analysis work between SAT-based and ILP-based FPGA routing methods. Please refer to [21] for detailed results.



(a)

$$Conn(A) = [(AV \equiv 0) \vee (AV \equiv 1) \vee (AV \equiv 2)] \wedge \quad \text{// Vertical channel 1}$$
$$[AV = AH] \wedge \quad \text{// S-block(1,3)}$$
$$[(AH \equiv 0) \vee (AH \equiv 1) \vee (AH \equiv 2)] \quad \text{// Horizontal channel 3}$$

$$Conn(B) = [(BV \equiv 0) \vee (BV \equiv 1) \vee (BV \equiv 2)] \wedge \quad \text{// Vertical channel 1}$$
$$[BV = BH] \wedge \quad \text{// S-block(1,1)}$$
$$[(BH \equiv 0) \vee (BH \equiv 1) \vee (BH \equiv 2)] \quad \text{// Horizontal channel 1}$$

$$Conn(C) = [(CH \equiv 0) \vee (CH \equiv 1) \vee (CH \equiv 2)] \wedge \quad \text{// Horizontal channel 3}$$
$$[CH = CV] \wedge \quad \text{// S-block(1,3)}$$
$$[(CV \equiv 0) \vee (CV \equiv 1) \vee (CV \equiv 2)] \quad \text{// Vertical channel 1}$$

(b)

$$Excl(V1) = (AV \neq BV) \wedge (AV \neq CV)$$

(c)

Fig. 4. Example of track-based detailed routing constraint formulation. Prior to constraint evaluation, the integer-valued net variables are encoded using binary variables and the constraints are transformed to CNF. (a) Global routing configuration for nets A, B, and C and corresponding variable declarations. (b) Connectivity constraints. (c) Exclusivity constraints.

This formulation is named the *track-based routing constraint model* to emphasize the fact that it is defined over a set of variables that represent the tracks available for routing.

An example that illustrates this formulation is shown in Fig. 4 for an FPGA with $W = Fc = Fs = 3$. Each net is assumed to have been assigned a global route (sequence of C and S-blocks) by a global router (Fig. 4a). Track variables are then created for each net to model its possible assignment to specific tracks in each of the channels specified by its global route. For instance, two track variables are associated with net A, AV and AH to indicate their track assignments in vertical channel *1* and horizontal channel *3*, respectively. These track variables are multivalued with a domain $\{0, \ldots, W-1\}$. In the actual formulation, each of these multivalued variables is encoded by $\lceil \log_2 W \rceil$ Boolean variables using the standard decimal-to-binary encoding.

The construction of the connectivity and exclusivity constraints is depicted in Fig. 4b. The connectivity constraint for a given net restricts the net's track variables to those values that insure a continuous conductive path between the net's pins. For example, net B can be assigned to any track in vertical channel *1* as well as horizontal channel *1* as long as the same track number is used in both channels. The constraint of the same track number reflects the requirement through S-block which has a flexibility Fs = 3. The exclusivity constraints in this example insure that net pairs (A, B) and (A, C) are assigned to different track numbers in vertical channel *1*. Fig. 5 shows the actual CNF representation of connectivity and exclusivity constraints between two 3-valued track variables. The number of CNF clauses required to express a connectivity constraint type-1 (i.e., a

$$X = [X_0, X_1] \equiv 2X_1 + X_0$$
$$Y = [Y_0, Y_1] \equiv 2Y_1 + Y_0$$

(a)

$$\begin{aligned} f(X) &= (X \equiv 0) \vee (X \equiv 1) \vee (X \equiv 2) \\ &= \overline{(X \equiv 3)} \\ &= \overline{(X_0 \wedge X_1)} \\ &= (\overline{X_0} \vee \overline{X_1}) \end{aligned}$$

(b)

$$\begin{aligned} f(X, Y) &= (X \equiv Y) \\ &= [X_0 \equiv Y_0] \wedge [X_1 \equiv Y_1] \\ &= [(X_0 \rightarrow Y_0) \wedge (Y_0 \rightarrow X_0)] \wedge [(X_1 \rightarrow Y_1) \wedge (Y_1 \rightarrow X_1)] \\ &= [(\overline{X_0} \vee Y_0) \wedge (X_0 \wedge \overline{Y_0})] \wedge [(\overline{X_1} \vee Y_1) \wedge (X_1 \wedge \overline{Y_1})] \end{aligned}$$

(c)

$$\begin{aligned} f(X, Y) &= (X \neq Y) \\ &= \overline{(X \equiv 0 \wedge Y \equiv 0) \vee (X \equiv 1 \wedge Y \equiv 1) \vee (X \equiv 2 \wedge Y \equiv 2)} \\ &= \overline{(X \equiv 0 \wedge Y \equiv 0)} \wedge \overline{(X \equiv 1 \wedge Y \equiv 1)} \wedge \overline{(X \equiv 2 \wedge Y \equiv 2)} \\ &= (X_0 \vee X_1 \vee Y_0 \vee Y_1) \wedge (\overline{X_0} \vee X_1 \vee \overline{Y_0} \vee Y_1) \wedge (X_0 \vee \overline{X_1} \vee Y_0 \vee \overline{Y_1}) \end{aligned}$$

(d)

Fig. 5. CNF representation of track-based routing constraints. (a) Track variable X, Y, and corresponding Boolean variables. (b) Connectivity constraint (type 1) in CNF. (c) Connectivity constraint (type 2) in CNF. (d) Exclusivity constraint in CNF.

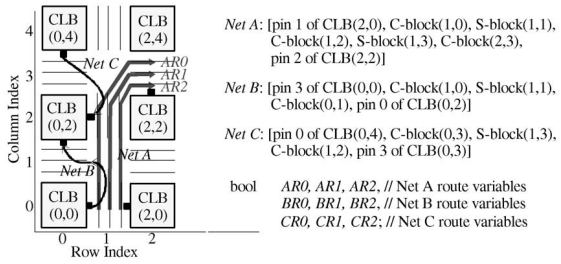
constraint from C-block) is dependent on the actual value of W and hard to formulate simply in one equation. For most cases, a type-1 connectivity constraint requires fewer than three CNF clauses, each having at most $2\lceil \log_2 W \rceil$ literals. The type-2 connectivity constraint from S-block whose form is an equality between two track variables, however, can be expressed with exactly $2\lceil \log_2 W \rceil$ 2-literal CNF clauses. In general, exclusivity constraints, which are basically inequalities between track variables, can be represented with W CNF clauses, each having $2\lceil \log_2 W \rceil$ literals. The Boolean function that models the routability of these three nets is simply the conjunction of all the connectivity and exclusivity requirements:

$$R(X) = Conn(A) \wedge Conn(B) \wedge Excl(H1),$$

where X is a vector of Boolean variables that encode the track variables $AH, AV, \ldots, CH$, and CV.

3.2 Formulation 2: Route-Based Routing Constraint Model

In the route-based formulation [22], the routability of a netlist is directly modeled in terms of Boolean variables that represent all of the detailed routes admissible by the given global routing solution. This choice of variables leads to a simpler set of constraints than those described above and enables the solution of larger routing instances. This formulation is illustrated in Fig. 6. Within the global routing region specified for net A, for example, there are only three possible detailed routes, indicated by the three Boolean variables *AR0*, *AR1*, and *AR2*. This is in contrast to the track-based formulation where net A was encoded by two Boolean vector AV and AH, which consists of two Boolean variables each, to capture the exactly same set of detailed routes. A similar set of routes and corresponding route variables is created for net B and C. A particular route is included in the final routing solution if its corresponding Boolean variable is assigned the value 1 and is excluded as a routing option otherwise. With this choice of variables, the FPGA detailed routing problem is transformed from a track assignment to a "routability checking" problem. The routability of a netlist in terms of these "route" variables can now be expressed with two types of constraints:



Net A: [pin 1 of CLB(2,0), C-block(1,0), S-block(1,1), C-block(1,2), S-block(1,3), C-block(2,3), pin 2 of CLB(2,2)]

Net B: [pin 3 of CLB(0,0), C-block(1,0), S-block(1,1), C-block(0,1), pin 0 of CLB(0,2)]

Net C: [pin 0 of CLB(0,4), C-block(0,3), S-block(1,3), C-block(1,2), pin 3 of CLB(0,3)]

(a)

bool *AR0, AR1, AR2,* // Net A route variables
BR0, BR1, BR2, // Net B route variables
CR0, CR1, CR2; // Net C route variables

$Live(A) = (AR0 \vee AR1 \vee AR2)$ // At least one detailed route for net A must be selected
$Live(B) = (BR0 \vee BR1 \vee BR2)$ // Ditto for net B,
$Live(C) = (CR0 \vee CR1 \vee CR2)$ // and net C.

(b)

$Excl(Resource(1,1,0)) = (\overline{AR0} \vee \overline{BR0})$ // Track 0 of C-block(1,1)
$Excl(Resource(1,1,1)) = (\overline{AR1} \vee \overline{BR1})$ // Track 1 of C-block(1,1)
$Excl(Resource(1,1,2)) = (\overline{AR2} \vee \overline{BR2})$ // Track 2 of C-block(1,1)
$Excl(Resource(1,2,0)) = (\overline{AR0} \vee \overline{CR0})$ // Track 0 of C-block(1,2)
$Excl(Resource(1,2,1)) = (\overline{AR1} \vee \overline{CR1})$ // Track 1 of C-block(1,2)
$Excl(Resource(1,2,2)) = (\overline{AR2} \vee \overline{CR2})$ // Track 2 of C-block(1,2)

(c)

Fig. 6. Example of route-based formulation. A Boolean variable is assigned for each detailed route enumerated for a net. (a) Global routing configuration for nets A, B, and C with the three possible detailed routes for net A. The detailed routes for each net are represented by a set of Boolean route variables. (b) Liveness constraints. (c) Exclusivity constraints.

- **Liveness constraints** to ensure that each two-pin connection has at least one detailed route selected in the final routing solution. The liveness constraint for a given two-pin connection has a simple form, namely, an OR over the connection's Fc route variables (see Fig. 6b). For a netlist with n two-pin connections, liveness constraints yield a set of n CNF clauses, each containing Fc positive literals.
- **Exclusivity constraints** to guarantee that electrically distinct nets with overlapping vertical or horizontal spans in the same channel are assigned to different tracks. These constraints are semantically identical to those described earlier for the track-based formulation, but have a much simpler CNF representation (see Fig. 6c). For example, $Excl(Resource(1,1,0)) = (\neg AR0 \vee \neg BR0)$ indicates that the routing resource, tack segment 0 of C-block (1, 1), can only be used by either detailed route 0 of net A or detailed route 0 of net B, but not both. In general, if k different detailed routes from different nets are competing for the same routing resource, a set of $k(k-1)/2$ exclusivity constraints is created to insure that at most one of those detailed routes is assigned to that resource. Each of those constraints, in turn, is a simple CNF clause consisting of two complemented literals.

Since each two-pin connection forms a complete path from a source pin to a sink pin, no further constraint is needed to stitch them together into a single multipin net.

The routability of a netlist for a given placement and global routing configuration is expressed by a single Boolean function which is the conjunction of all liveness and exclusivity constraints:

$$Routable(X) = \left[\begin{array}{ccc} \wedge & Live_i(X) \wedge & \wedge \\ 1 \leq i \leq n & & 1 \leq j \leq r \end{array} \begin{array}{c} \\ \\ Excl_j(X) \end{array} \right],$$

where, assuming there are n nets and r routing resources in total, $Live_i(X)$ is a liveness constraint of two-pin connection i, $Excl_j(X)$ is an exclusivity constraint of routing resource j, and X is a vector of Boolean route variables that represent the possible detailed routes for each of the nets. With $Fc = W$ and $Fs = 3$ architectural assumption, the route-based formulation is truly equivalent to the track-based formulation since both of them capture exactly the same set of detailed routes within given global routing regions. In addition, for most circuits, it requires fewer variables and is expressed in terms of a simpler set of CNF constraints.

Currently, the route-based formulation is able to consider only a FPGA routing architecture with a S-block flexibility *Fs = 3*. With a more flexible S-block routing architecture (*Fs > 3*), the number of detailed routes enumerated will grow exponentially. Thus, the number of Boolean variables required in the route-based formulation also grows exponentially, which makes the method inapplicable. Although the majority of current FPGA routing architecture employs *Fs = 3*, we need to show a practical solution with routing architectures of *Fs >= 3*. As a matter of fact, the route-based formulation doesn't have to capture all the admissible detailed routes within a given global routing region. Instead, multiple detailed routes from several global routing solutions can be considered for each net simultaneously. In other words, for each net, first generate a set of possible global routing solutions. For each global routing solution, we can pick a couple of promising detailed routes[3] with the global region. Then, all the enumerated detailed routes from various global routing solutions become a set of candidate detailed routes for a net under consideration. The benefits of this scheme are two-fold: 1) More timing appropriate detailed routes can be chosen for critical nets and 2) the detailed router is able to escape from a decision, particularly a bad one, made from global routing solutions. This remains as future work.

For a fair comparison with the track-based formulation, however, we only consider all the admissible detailed routes within global routing regions for experiments.

4 Experimental Results

We experimentally tested the effectiveness of the route-based formulation method on the standard MCNC benchmark circuits. The relevant properties of these circuits, listed in Table 1, include the size of the target FPGA CLB arrays (column "X x Y"), the actual number of CLBs used by the circuit ("#CLBs"), the number of multipin nets ("#Nets"), the corresponding number of two-pin connections that are routed individually ("#2pin Conns"), and the average number of channel segments per 2-pin connection (column "Ave. Channel Segs/2pin-Net"). The average number of channel segments per two-pin connection is important because, in track-based formulation, the number of Boolean variables required to formulate routing problems is dependent on this factor.

TABLE 1
Benchmark Circuits

Circuit	X x Y	#CLBs	#Nets	#2Pin Conns	Ave. Channel Segs/2PinConn
9symml	9 x 9	70	79	259	10.99
alu2	12 x 12	143	153	510	8.46
apex7	11 x 11	77	126	300	5.25
C499	10 x 10	74	115	312	6.00
C880	14 x 14	174	234	656	6.59
C1355	10 x 10	74	115	312	6.31
example2	19 x 19	120	205	444	5.86
k2	19 x 19	358	404	1257	8.15
term1	8 x 8	54	88	202	4.24
too_large	13 x 13	148	186	519	7.12
vda	15 x 15	208	225	722	8.26
Average	*	*	176	499	*

The first experiment we conducted compares the performance of the track and route-based detailed routing formulations. Using the placements and global routing solutions generated by VPR [4], routing functions from the track and route-based formulations were produced for decreasing values of the channel width *W* until we found an unroutable routing instance. These routability Boolean functions were subsequently evaluated by the GRASP SAT solver [18].

Table 2 summarizes the results of the last two "iterations" (satisfiable and unsatisfiable) for five of the 11 benchmark circuits: the minimum channel width for which the benchmark circuit was still routable and the maximum channel width for which it was proven to be unroutable. For the remaining benchmarks not listed in Table 2, the minimum width for routability could not be found using the track-based formulation; the performance of the route-based formulation on these benchmarks is discussed later.

The columns in this table record, for each benchmark circuit, the following data: the assumed channel width ("W"), the number of Boolean variables and CNF clauses in the routability function ("V" and "CL"), the number of decisions and conflicts during the SAT search for a solution ("Dec" and "Conf"), and the CPU time in seconds spent for actual SAT search by GRASP ("Time"). Column "R?" indicates whether the routability function had a feasible solution; the computational advantage of the route-based formulation is shown in column "Speedup" which is the ratio of the CPU times taken by GRASP to solve the respective routability functions. The experiment was conducted on a Pentium III PC running Debian Linux 2.2.18 with 512 MB of physical memory. The GRASP SAT solver was configured to use the "DLCS" decision heuristic.

We can observe immediately that typical solution times of the route-based formulation are much faster than the track-based method, achieving 66x speedups on average. The actual numbers of decisions and conflicts during the SAT search validate the achieved speedups, as shown in Fig. 7. The graph illustrates the normalized values of decisions, conflicts, and SAT solving time of route-based formulation when those of track-based formulation are

3. This is possible in real FPGAs because, typically, there are various lengths of wire segments per channel.

TABLE 2
Performance Comparison between Two Boolean Formulations of FPGA Detailed Routing: Time Unit Is in Seconds

Circ.	W	Track-based Formulation					Route-based Formulation					R?	Speed up
		V	CL	Dec	Conf	Time[s]	V	CL	Dec	Conf	Time[s]		
9sym	6	2604	36994	11883	9910	471.12	1554	29119	347	0	1.94	Yes	243
	5	2604	32450	13896	11687	521.39	1295	24309	344	272	6.11	No	85
apex	5	1983	15358	606	220	3.63	1500	11695	568	0	1.51	Yes	2
	4	1322	10940	445	336	3.59	1200	9416	293	191	1.38	No	3
exam	6	3603	41023	1347	708	24.12	2664	27684	993	1	5.65	Yes	4
	5	3603	36344	12613	11063	531.42	2220	23144	1331	1245	25.16	No	21
term1	4	746	3964	322	134	0.51	808	3290	207	6	0.11	Yes	5
	3	746	3517	71	48	0.19	606	2518	12	12	0.03	No	6
C499	6	2070	22470	11381	10141	255.34	1872	18870	395	18	1.51	Yes	169
	5	2070	19908	11755	10696	322.76	1560	15777	372	355	4.66	No	69

considered 100. In both formulations, the number of Boolean variables of the routability functions can be computed in the following way: Assuming that the benchmark circuit has a total n two-pin connections, the total number of Boolean variables of the route-based formulation is exactly nW, whereas it is approximately $n\lceil \log_2 W \rceil m$ in the track-based formulation, where m is the average number of channels each global route of a two-pin connection passes through (i.e., the last column in Table 1). Fig. 8 also empirically demonstrates that the route-based formulation generates the routability function with fewer variables and clauses except *term1*. The circuit *term1* turns out to have a smallest m value (average number of channel segments per 2-pin connection) leading to fewer variables with the track-based formulation.

The numbers of variables and clauses, however, are not sufficient to explain the order-of-magnitude reduction in runtime. The analysis of clause size distribution (Fig. 9) reveals that the route-based formulation is mostly composed of 2-literal CNF clauses (more than 95 percent on average), whereas the track-based formulation is composed mostly of clauses containing three or more literals (i.e., less than 20 percent of 2-literal CNFs). It is a well-known fact that 2-SAT has *P* class complexity while 3-SAT or higher SAT problems are *NP-complete* [10]. This is because, in 2-SAT, it is feasible to construct an implication graph to deduce solutions efficiently, which is impossible with higher order of SAT problems. Accordingly, the performance gain can be

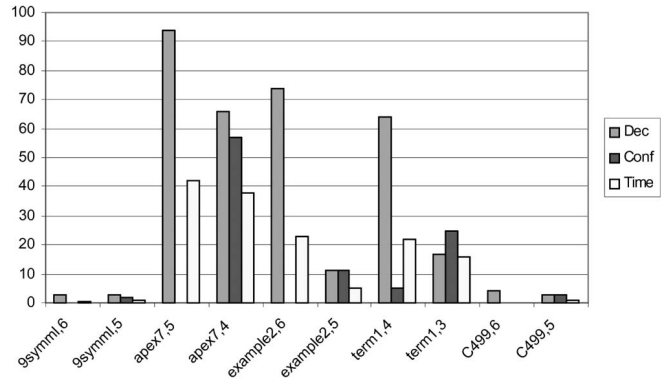


Fig. 7. The normalized number of decisions, conflicts, and normalized SAT solving time of route-based formulation over those of track-based formulation. That is, 100 represent the decision, conflict, and runtime of the track-based formulation.

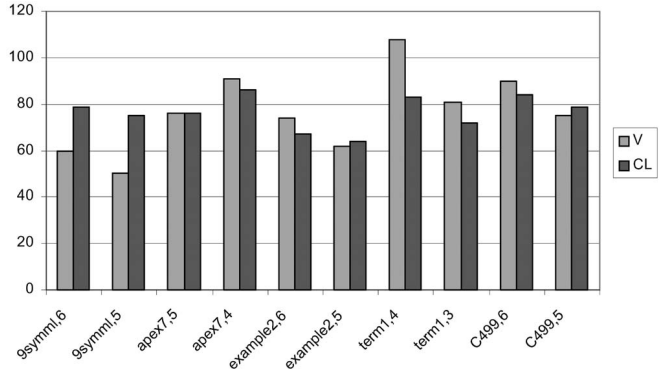


Fig. 8. The normalized number of CNF variable and classes of route-based formulation over those of track-based formulation. Again, 100 represents the number of variables and clauses of the corresponding track-based formulation.

justified by the CNF clauses structure of the route-based constraint function, which is close to a 2-SAT problem (albeit it is not exactly 2-SAT due to liveness constraint CNF clauses). Overall, this experiment suggests that it is more straightforward to find solutions for Boolean SAT instances from the route-based formulation.

In Table 3, we show the performance of the route-based formulation method over routing problems where the track-based method was not able to solve them within a 24 hour limit—i.e., no conclusion was drawn whether given circuits are routable or not. Each column heading has the same meaning as those in Table 2. "G.Time" and "S.Time" represent "Boolean SAT generation time" and "Solution search time," all in seconds. The number of decisions and conflicts are larger than those numbers in Table 2, suggesting that these are harder SAT instances. For only two cases—benchmark *k2* with $W = 9$ and $W = 8$—even the route-based method could not solve the routability Boolean SAT problems. One interesting observation is that it is most difficult to draw the routability decision (either routable or unroutable) with marginal track counts per channel W. This is not surprising because, when a target FPGA has ample routing tracks per channel, the corresponding FPGA detailed routing is an easy one due to abundant routing resources. Similarly, if there are significantly insufficient routing resources, it should be straightforward to prove unroutability.

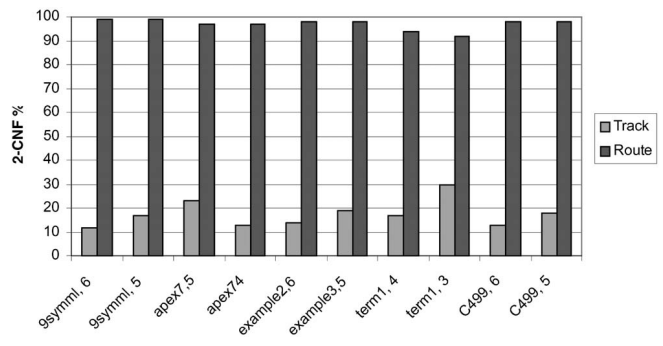


Fig. 9. The ratio comparison of size 2 CNF clauses over total CNF clauses.

TABLE 3
The Route-Based Formulation Results for Cases where the Track-Based One Was Unable to Solve

Circuit	W	#Vars	#Clause	#Dec	#Conf	G.Time [s]	S.Time [s]	R?
alu2	8	4080	83902	986	2	1.87	13.02	Yes
	7	3570	73478	9014	8968	1.66	1191.6	No
C880	7	4592	61745	1143	81	1.41	13.82	Yes
	6	3936	53018	40327	39546	1.21	52364.70	No
	5	3280	44291	612	598	1.02	21.83	No
k2	11	13827	372694	4199	16	8.93	280.61	Yes
	10	12570	338927	2902	51	8.01	186.08	Yes
	9	11313	305160	N.C[a]	N.C	7.19	N.C	N.C
	8	10056	271393	N.C	N.C	6.33	N.C	N.C
	7	8799	237626	17823	9873	5.81	4870.56	No
too_lrg	7	3633	50373	828	19	1.11	8.63	Yes
	6	3114	43251	1669	1184	0.93	59.01	No
	5	2595	36129	1163	1122	0.82	36.93	No
vda	9	6498	130997	1387	3	3.00	31.78	Yes
	8	5776	116522	25924	24861	2.72	6146.20	Yes
	7	5054	102547	23333	22098	2.28	12383.50	No

a. N.C stands for "Not Complete". Thus, the data is not available.

TABLE 4
Track Number Comparison with Other Conventional Routers

Placer	VPR [4]			SPLACE	ALTOR [23]			FPR [1]
G. Router				SROUT [28]	TRACER [15]	IKMB [2]	GBP [29]	
D. Router	R-SAT	VPR	SEGA[16]					
9symml	6	5	6	7	6	8	9	9
alu2	7	6	9	8	9	9	11	10
apex7	5	5	6	6	8	10	11	9
exam2	6	5	6	7	10	11	13	13
term1	4	4	6	5	7	8	10	8
too_lrg	7	7	9	8	9	10	12	11
k2	10	9	11	11	14	15	17	17
vda	8	8	10	10	11	12	13	13
C499	6	6	7	*[a]	*	*	*	*
C880	7	7	8	*	*	*	*	*
C1355	6	6	7	*	*	*	*	*
Total Σ W	**72**	**68**	**85**	**81**	**93**	**102**	**115**	**109**

a. "*" indicates the related data is not available. For "Total W" calculation, we used the corresponding track values of the route-based formulations.

Finally, in Table 4, we compare the performance of various FPGA detailed routers, including our route-based formulation shown under the "R-SAT" heading. The table shows the number of tracks required to successfully route each benchmark circuit with the specified placement and global routing programs under the same track minimization scenario. The most meaningful comparison in this table is between the route-based method, VPR, and SEGA because they differ only in how detailed routing was done. The data for the rest of the routers are provided only for reference. Our route-based routing method shows the second best results among them, next to VPR [4] while beating the rest of them. Interestingly however, for four cases out of those five circuits (shaded cells in the table), our method proved *unroutability*, whereas VPR was able to successfully route them with the same numbers of tracks per channel. This difference is due to the fact that our method performs only detailed routing, while VPR does both global and detailed routing. Thus, VPR can change the global routing configuration easily when it cannot find a detailed routing solution. A more fair comparison is to extract the final detailed routing results from VPR and use them as a global routing solution for the route-based detailed routing formulation. Indeed, the route-based formulation method was able to produce exactly the same results as VPR except *k2* with nine tracks per channel where it couldn't find a solution after 24 hours.

5 Conclusion

In this paper, we conducted comparative analyses of two Boolean SAT-based FPGA detailed routing methods: track-based and route-based formulations. The route-based formulation differs from the track-based formulation because the routability constraints are expressed in terms of a set of "route" variables, each of which designated a specific detailed route for a given net. The experiments demonstrate that the route-based formulation yields an easier to evaluate and more scalable routability Boolean function than the track-based method. This fact shows that a more efficient Boolean formulation can achieve significantly better solution search performance improvement in real-world applications.

Acknowledgments

This work was conducted while Gi-Joon Nam and Fadi Aloul were at the University of Michigan and was supported by the US National Science Foundation under grant 9404632.

References

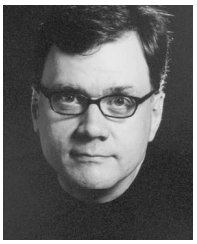
[1] M.J. Alexander, J.P. Cohoon, J.L. Ganley, and G. Robins, "Performance-Oriented Placement and Routing for Field-Programmable Gate Arrays," *Proc. European Design Automation Conf.*, 1995.
[2] M.J. Alexander and G. Robins, "New Performance-Driven FPGA Routing Algorithms," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 12, pp. 1505-1517, Dec. 1996.
[3] R. Bayardo Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real World SAT Instances," *Proc. 14th Nat'l Conf. Artificial Intelligence*, pp. 203-208, 1997.
[4] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Proc. Seventh Ann. Workshop Field Programmable Logic and Applications*, pp. 213-222, 1997.
[5] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," *Proc. Tools and Algorithms for the Construction and Analysis of Systems*, 1999.
[6] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field Programmable Gate Arrays*. Boston: Kluwer Academic, 1992.
[7] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, vol. 7, pp. 201-215, 1960.
[8] M. Davis, G. Longeman, and D. Loveland, "A Machine Program for Theorem Proving," *Comm. ACM*, vol. 5, no. 7, 1962.
[9] S. Devadas, "Optimal Layout via Boolean Satisfiability," *Proc. ACM/IEEE Int'l Conf. Computer-Aided Design*, pp. 294-297, 1989.
[10] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
[11] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust SAT-solver," *Proc. Design, Automation, and Test in Europe (DATE '02*, pp. 142-149, Mar. 2002.
[12] T.Y.K. Kam and R.K. Brayton, "Multi-Valued Decision Diagrams," Technical Report UCB/ERL M90/125, Univ. of California at Berkeley, 1990.
[13] W. Kunz and D. Stoffel, *Reasoning in Boolean Networks*. Kluwer Academic, 1997.
[14] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 1, pp. 4-15, 1992.
[15] Y.-S. Lee and A. Wu, "A Performance and Routability Driven Router for FPGAs Considering Path Delays," *Proc. Design Automation Conf.*, pp. 557-561, 1995.
[16] G. Lemieux and S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs," *Proc. ACM Physical Design Workshop*, Apr. 1993.

[17] C.-M. Li, "Integrating Equivalency Reasoning into Davis-Putnam Procedure," *Proc. Nat'l Conf. Artificial Intelligence,* 2000.
[18] J.P. Marques-Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Trans. Computers,* vol. 48, no. 5, May 1999.
[19] L.E. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Path-Driven Router for FPGAs," *Proc. ACM/IEEE Int'l Symp. Field Programmable Gate Arrays,* Feb. 1995.
[20] M. Moskewicz, C. Madigan, Y. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *Proc. Design Automation Conf.,* pp. 530-535, 2001.
[21] G.-J. Nam, K.A. Sakallah, and R.A. Rutenbar, "A New FPGA Detailed Routing Approach via Search-Based Boolean Satisfiability," *IEEE Trans. Computer-Aided Design,* vol. 21, no. 6, June 2002.
[22] G.-J. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, "A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints," *Proc. Int'l Symp. Physical Design,* 2001.
[23] J. Rose, W. Snelgrove, and Z. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," *Proc. Canadian Conf. Very Large Scale Integration,* pp. 169-173, Nov. 1985.
[24] B. Selman, H. Kautz, and B. Cohen, "Local Search Strategies for Satisfiability Testing," *DIMACS Series in Discrete Math. and Theoretical Computer Science,* vol. 26, pp. 521-532, 1996.
[25] N. Sherwani, *Algorithms for VLSI Physical Design Automation.* Boston: Kluwer Academic, 2000.
[26] K. Sulimma and W. Kunz, "An Exact Algorithm for Solving Difficult Detailed Routing Problems," *Proc. ACM/IEEE Int'l Symp. Physical Design,* Apr. 2001.
[27] T. Szymanski, "Dogleg Channel Routing Is NP-Complete," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 4, no. 1, 1985.
[28] S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories," PhD dissertation, Univ. of Toronto, 1997.
[29] Y.-L. Wu and M. Marek-Sadowska, "Routing for Array-Type FPGAs," *IEEE Trans. Computer-Aided Design,* vol. 16, no. 5, pp. 506-518, May 1997.
[30] M.V. Velev and R.E. Bryant, "Superscalar Processor Verification Using Efficient Reductions from the Logic of Equality with Uninterpreted Functions to Propositional Logic,", *Proc. Correct Hardware Design and Verification Methods,* pp. 37-53, 1999.
[31] R. Zabih and D.A. Mcallester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," *Proc. Nat'l Conf. Artificial Intelligence,* pp. 155-160, 1988.
[32] H. Zhang, "SATO: An Efficient Propositional Prover," *Proc. Int'l Conf. Automated Deduction,* pp. 272-275, 1997.

Gi-Joon Nam received the BS degree in computer engineering from Seoul National University, Seoul, Korea, in 1995 and the MS and PhD degrees in computer science and engineering from the University of Michigan, Ann Arbor, in 1999 and 2001, respectively. Since August 2001, he has been with IBM Austin Research and is currently working in the physical design space, particularly placement/routing and timing closure tools. His general interests are computer-aided design algorithms, combinatorial optimizations, VLSI system designs, and computer architecture. He received a first prize (and the best paper application) in the VLSI Design Contest of the 2001 ACM/IEEE Design Automation Conference. In 2004, he is serving on the technical program committee for the ACM/IEEE International symposium on Physical Design (ISPD), Great Lake Symposium on VLSI (GLSVLSI), International Conference on Computer Design (ICCD), and IEEE International System-On-Chip Conference (SOCC). He is a member of the IEEE and the ACM.

Fadi Aloul received the BS degree in electrical engineering summa cum laude from Lawrence Technological University (LTU) in 1997 and the MS and PhD degrees in computer science and engineering from the University of Michigan, Ann Arbor, in 1999 and 2003, respectively. He was a postdoctoral research fellow at the University of Michigan during the summer of 2003. He is currently an assistant professor of computer engineering at the American University in Dubai. He has received a number of awards, including the Agere/SRC research fellowship, GANN fellowship, and the LTU presidential scholarship. He is currently serving on the technical program committee of the 2004 International Workshop on Logic Synthesis (IWLS). He has published more than 25 papers in international journals, conferences, and workshops. His current research interests are in the area of computer-aided design, verification, and Boolean satisfiability.

Karem A. Sakallah received the BE degree in electrical engineering from the American University of Beirut, Beirut, Lebanon, in 1975 and the MSEE and PhD degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1977 and 1981, respectively. In 1981, he was with the Department of Electrical Engineering at Carnegie Mellon University as a visiting assistant professor. From 1982 to 1988, he was with the Semiconductor Engineering Computer-Aided Design Group at Digital Equipment Corporation in Hudson, Massachusetts, where he headed the Analysis and Simulation Advanced Development Team. Since September 1988, he has been with the University of Michigan, Ann Arbor, as a professor of electrical engineering and computer science. From September 1994 to March 1995, he was with the Cadence Berkeley Laboratory in Berkeley, California, on a six-month sabbatical leave. He has authored or coauthored more than 130 papers and has presented seminars and tutorials at many professional meetings and various industrial sites. His research interests include the area of computer-aided design with emphasis on simulation, timing verification and optimal clocking, logic and layout synthesis, Boolean satisfiability, and hardware and software verification. He is currently an associate editor of the *IEEE Transactions on Computers*. He is a fellow of the IEEE and a member of the ACM and Sigma Xi.



Rob A. Rutenbar received the PhD degree from the University of Michigan in 1984 and subsequently joined the faculty of Carnegie Mellon University (CMU). He is currently the Stephen J. Jatras Professor of Electrical and Computer Engineering and (by courtesy) of Computer Science. From 1993 to 1998, he was director of the CMU Center for Electronic Design Automation. He cofounded Neolinear, Inc., in 1998 and currently serves as its chief scientist. He is the founding director of the MARCO/DARPA Center for Circuit & System Solutions (C2S2), a national consortium of US universities chartered in 2001 to explore long-term solutions for next-generation circuit challenges. His research interests focus on circuit and layout synthesis algorithms for mixed-signal ASICs, for large digital ICs. In 1987, he received a Presidential Young Investigator Award from the US National Science Foundation. He has won Best/Distinguished Paper awards from the Design Automation Conference (1987 and 2002), the International Conference on CAD (1991), and the Semiconductor Research Corporation TECHCON (1993, 2000, and 2003). In 2001, he was cowinner of the Semiconductor Research Corporation's Aristotle Award for contributions to graduate education. He is a fellow of the IEEE and a member of the ACM and Eta Kappa Nu.