FPGA Routing and Routability Estimation Via Boolean Satisfiability

R. Glenn Wood and Rob A. Rutenbar

Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213 U.S.A.
email: glennw@col.hp.com rutenbar@ece.cmu.edu

Abstract*

Guaranteeing or even estimating the routability of a portion of a placed 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 BDDs. 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.

1 Introduction

Layout for FPGAs 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 ASICs have been quite successful in automating FPGA layout. Nevertheless, some layout issues are poorly dealt with in FPGAs, 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 ASICs often fare poorly here because the geometric "slack" allowable in free-form 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. [Brow92b] and [Chan93] both elaborate stochastic wirability theories to handle FPGA structures. In [Alex94], a router solves approximations to a sequence of Steiner-Tree-on-a-Graph problems for all nets in a multi-weighted graph depicting the FPGA architecture; congestion weights prevent signal overlap while net cost weights are used to build optimal routes. Aggressive rip-up-and-retry has been developed to deal with the net ordering problem [McMu95]. [Brow92b] describes a detailed router which expands the output from a global router into a graph of all possible detailed routes for a particular two-point connection; these expanded graphs are then pruned for efficiency and greedy local techniques used to search for exact detailed routes. [Lemi93] extends this technique to the case of partially segmented channels. 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 FPGAs makes them amenable to unique solution strategies. One such approach is described in [Nag94,Nag95], which attack 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. [Wu93] observes that simplified variants of the detailed routing problem for FPGAs are reducable to the 2D interval packing problem, and to the graph coloring problem. These constructions were used to assert the NP-hardness of FPGA routing in [Wu94]. [Chan94] introduces 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 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 [Deva89], 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 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 [Brya86] and heuristic search [Marq97]. In our attack on the problem, we represent this Boolean function using Binary Decision Diagrams (BDDs, [Brya86]) 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 the basic formulation, describe our implementation, and show some promising initial results for a subset of the Xilinx 4000-series routing fabric.

* This work was supported in part by the Semiconductor Research Corporation and the Army Research Office.

2 Routing via Boolean Satisfiability: Formulation

In [Deva89], Devadas showed a simple but elegant formulation of conventional 2-layer channel routing as Boolean satisfiability. This formulation encodes the information present in a channel's Vertical Constraint Graph, Horizontal Constraint Graph, and anticipated channel width into Boolean constraints on *n*-bit vectors, one per net to be routed. The resulting Boolean equation for these constraints 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 channel routing, the formulation is impractical.

But this Boolean satisfiability approach is much more amenable to FPGAs, which really do have completely rigid routing resources. In this work we restrict ourselves to an island-style (*i.e.*, Xilinx-type, see [Brow92b]) FPGA fabric, illustrated in Figure 1. Configurable logic blocks (CLBs) 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 CLBs. In this routing problem, there is no need to model a Vertical Constraint Graph because track stubs in an FPGA are pre-fabricated to be electrically disjoint. However, FPGAs 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 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 non-interfering 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**,**4** above.

There are several trade-offs inherent in this strategy. First, although it is conceptually straightforward to create the necessary

Figure 1. Island-style FPGA model

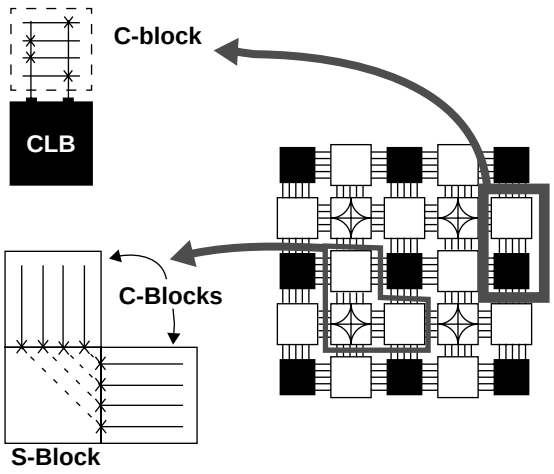
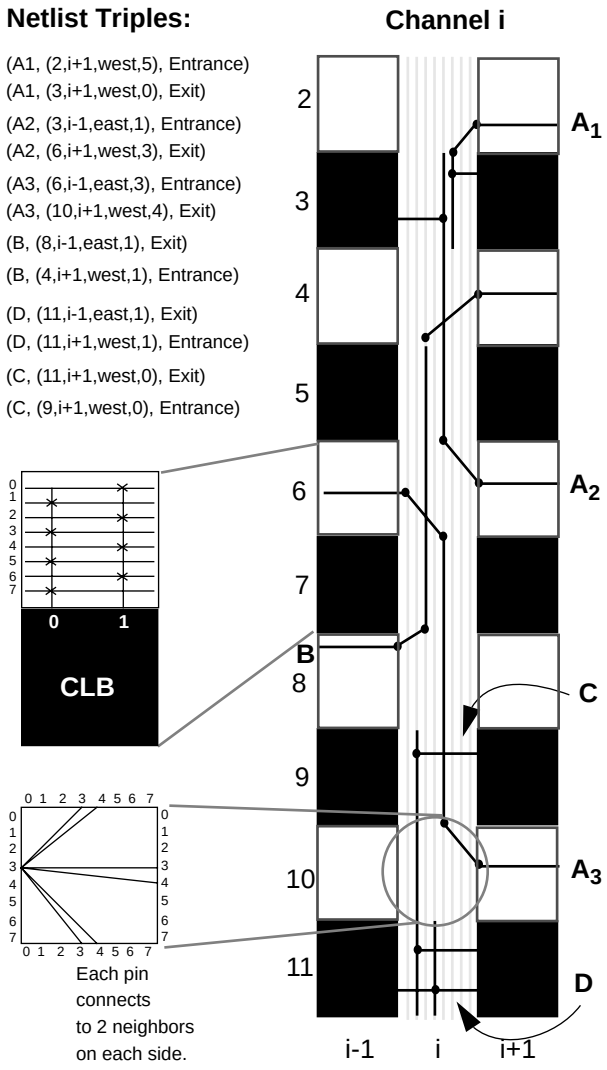


Figure 2. Detailed FPGA example with routes and netlist triples shown



Boolean functions for direct detailed routing for *all* the nets for an entire FPGA, the complexity of the resulting functions is intractable; we cannot represent these functions even for small designs. 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 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.) 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. Figure 2 gives a detailed example of a channel-style region of an FPGA with 3 columns and 10 rows. The architecture is described in terms of (W=8, Fc=4, Fs=6) as in [Brow92b, Ch. 6]. The figure also shows a netlist in this triples-format for 6 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 Figure 2. The formulation requires that 6 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 6 two-point connections of interest. We encode each of these vectors in 3 bits, since each vector encodes one of at most 8 distinct net-to-resource decisions. (Other encodings are possible, given different representations of the satisfiability computation; *e.g.*, see [Mina93] for use of zero-suppressed BDDs for large sparse set representations.) 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. Figure 3 illustrates two different types of routing violations we must be careful to avoid. The left channel depicts the violation of a *connectivity* constraint for the two-point connection involving net B. We know that net B must enter the channel on S-block (4,i+1) via the pin indexed 1 on its west side. According to the architecture of Figure 2, the net should be able to enter the channel only on tracks 1 or 2; thus, the route is invalid because it has the net on track 3. The right channel shows an *exclusivity* constraint being violated. Here, the two-point connections involving nets C and D are mapped to the same track. Figure 2 actually shows a valid route corresponding to the assignment:

$$\vec{A}_{1,i} = 101\ ,\ \vec{A}_{2,i} = 100\ ,\ \vec{A}_{3,i} = 100\ ,$$
$$\vec{B}_i = 010\ ,\ \vec{C}_i = 001\ ,\ \vec{D}_i = 100\ .$$

Figure 3. Routability constraint violations

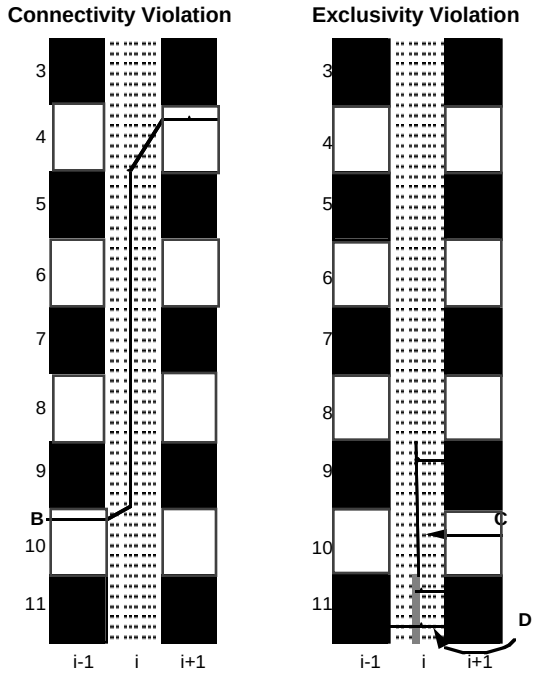


Figure 4. Satisfiability formulation for example of Figure 2

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

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

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

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

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

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

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

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

The seeming redundancies are an artifact of the constraint-generation process, which starts independently from each end–port constraint–of each 2-point connection and generates the relevant constraint.

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, 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 multi-points. 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 multi-point nets (*i.e.*, those that fan out inside the routing fabric to multiple sinks) directly and naturally. Figure 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_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_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_S_block* will return tracks 5 and 6 while *find_tracks_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, and we do not need to 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 BDDs.

3 Solution Implementation: BDD Management

We use Reduced Ordered Binary Decision Diagrams (ROBDDs) [Brya86,Brya92] as introduced by Bryant, to represent and solve our satisfiability formulation. ROBDDs constitute a canonical DAG-based representation for arbitrary Boolean functions, unique for a given a variable ordering. In contrast to other search-based satisfiability solution techniques, *e.g.*, [Marq97], 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 two-fold:

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. 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 made. (See Section 4.)

Since BDDs 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 BDDs 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 CLBs, 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 Figure 4, if we AND the constraints in the example in this order: Excl_i, ConnD_i, ConnC_i, ConnB_i, $\text{ConnA}_{3,i}$, $\text{ConnA}_{2,i}$, $\text{ConnA}_{1,i}$ 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 non-optimal variable ordering were chosen.

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. In [Berm89], Berman showed that the size of a BDD representation of a circuit c has an upper bound of $n \cdot 2^{wT(c)}$ nodes where n is the number of inputs variables and $wT(c)$ is the so-called *T-width* of the circuit under some topological ordering of the gates. The T-width is the number of "tracks" needed to "route" a circuit when its gates are placed on a line ac-

cording to the order T. The bound is extremely sensitive to this T-width; we have developed a novel heuristic to minimize this value.

Hall in [Hall70] 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. We are interested in the minimum total quadratic wirelength placement problem, which only requires computation of the eigenvector corresponding to the smallest non-zero 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. $(x_i - x_j)^2$) is directly minimized and T-width is *indirectly* minimized under the constraint that the variance of the coordinates is equal to 1. So, by producing the adjacency matrix corresponding to the exclusivity constraints for a particular channel, we should be able to determine a good variable ordering via a simple eigenvector extraction. However, this will in general destroy the topological ordering of the gatelist. This is not a factor in our case because we are only interested in the relative locations of the bit vectors which all happen to be primary inputs and hence have no topological dependencies. An ordering based on this simple eigen-solution strategy proved sufficient for us to create BDDs for most of our routing problems.

4 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 BDDs as the basic representation. The central questions are:

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

Consider the following situation (refer again to Figure 2). Two-point connection $\vec{A}_3$ is modified from its original triplet pair of (A,(6,i-1,east,3),Entrance), (A,(10,i+1,west,4),Exit) to (A, (8,i-1,west,4), Entrance), (A, (10,i+1,west,4), Exit). Clearly, this new connection impacts the feasibility constraints. So, first, the previous bit vector associated with the connection, $\vec{A}_3$ is *smoothed* [Brya92] out of the BDD by computing a new BDD which has the same on-set as the previous for *any value* of $\vec{A}_3$. This is just the *existential quantification* operation over all of the bits in the bit vector $\vec{A}_3$. The *new* routability function for this region of FPGA fabric, *minus* this net, is just:

$$(\exists \vec{A}_3 \ \text{routable}_i) \ (\vec{A}_{1,i}, \vec{A}_{2,i}, \vec{B}_i, \vec{C}_i, \vec{D}_i)$$

which can be computed in time proportional to the number of nodes in the BDD [Brya92]. Then, the new connectivity constraint is generated by simply adding (via AND, OR, EXORs) 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 $\vec{A}_3$ intersects with. These are intersected with the existentially quantified root BDD in the order just described. (See [Wood96] for more details of the incremental update process.)

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 BDDs 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.

5 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 [Brow92a,Lemi93] 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 back-mapping 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 [Gree90]. 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 [Brac90] was used. Each BDD node requires 18-22 bytes depending upon how close the BDD is to the maximum memory limit. Simplified Xilinx-style FPGAs 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 1 describes the benchmarks. The benchmarks are those from Alexander [Alex94] at the University of Virginia. 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 repre-

Table 1. 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

sentations of the constraints. Neither of these scenarios require the use of an exact estimator or routability.

Figure 5 presents the number of nodes required to represent each of the BDDs 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 3 sides. The top curve plots the same information for a switch block architecture with Fs=9. Notice how some of the channel BDDs 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 2 describes the result of an experiment in which we build routability BDDs 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 BDDs. When the constraints are ordered carefully as described in the previous section, the intermediate sizes of the BDDs tend to increase monotically 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 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 BDDs for each channel, averaged over all of the channels in each design. Once again, this assumes Fs=3, and W is taken from Table 1.

It is interesting to note how the above parameters vary with more flexible routing architectures. Table 3 repeats Table 2, but uses FPGAs possessing switch blocks with Fs=6 (much like those in Figure 2). As can be seen, once again, the problem complexity grows quickly with more flexible interconnect structure, with some examples taking as much as 24X more space and 28/50X 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.

This experiment nicely captures the trade-offs 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 BDDs 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 [Marq97]. Of course, if we merely want to find one routing solution for a segmented channel in an FPGA, many techniques exist, *e.g.*, [Gree90] which again uses a search technique, but does not represent explicitly all solutions. On the other hand, 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

Figure 5. BDD size by channel for benchmark 9symml, for flexibility Fs=3 and Fs=9

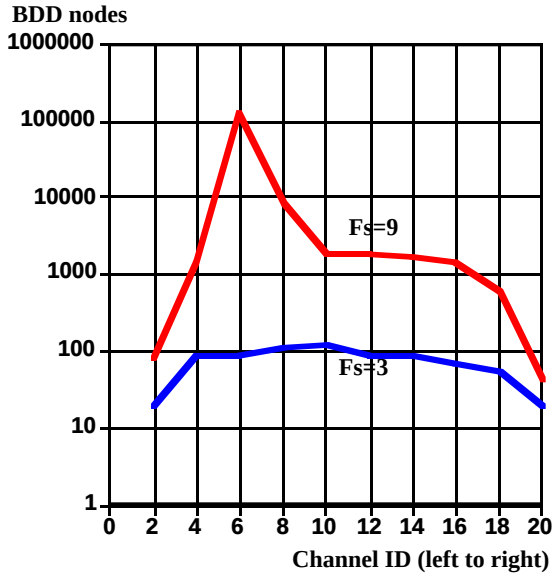


Table 2. Aggregate Benchmark Results

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 3. 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

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.

6 Conclusions

In this paper we developed a novel formulation of the FPGA routing and routability estimation problems that reduced these problems to Boolean Satisfiability. Given an FPGA routing architecture, a region of the routing fabric, and boundary constraints on signal I/Os 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 BDDs, 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 FPGAs 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 BDDs. 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.

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., 1-hot sparse set encodings [Mina93]), Boolean constraint clause ordering, decision diagram variable ordering (notably dynamic ordering techniques [Rude93], and decision diagram structure (e.g., smaller structures such as free-BDDs [Bern95] which support the satisfiability computations we need but not canonicity which we do not, or zero-suppressed BDDs [Mina93] for large sparse set problems) could have a substantial improvement on the range of practical application for the technique.

Acknowledgments

We are grateful to Randy Bryant for the BDD package and for several enlightening conversations about satisfiability. We are also grateful to the reviewers from the ISFPGA'97 program committee for several comments helpful in improving the clarity and accuracy of the presentation.

References

[Alex94] M.J. Alexander, J.P. Cohoon, J.L. Ganley, and G. Robins, "An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs," *Proc. European Design Auto. Conf.*, pp. 259-264, Sept. 1994.

[Berm89] C.L. Berman, "Ordered Binary Decision Diagrams and Circuit Structure," *Proc. IEEE ICCD*, pp. 392-395, Oct. 1989.

[Bern95] J. Bern, C. Meinel, and A. Slobodova, "Efficient OBDD-Based Boolean Manipulation in CAD Beyond Current Limits," *Proc ACM/IEEE DAC*, pp. 408-413, June 1995.

[Brac90] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient Implementation of a BDD package," *Proc. ACM/IEEE DAC*, pp. 40-45, June 1990.

[Brow92a] S. Brown, J. Rose, and Z.G. Vranesic, "A Detailed Router for Field Programmable Gate Arrays," *IEEE Trans. CAD*, pp. 620-628, vol. 11, no. 5, May 1992.

[Brow92b] S.D. Brown, R.J. Francis, J. Rose, and Z.G.Vranesic, *Field Programmable Gate Arrays*, Boston, Kluwer Acad. Publishers, 1992.

[Brya86] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, pp. 677-691, 1986.

[Brya92] R.E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293-318, Sept. 1992.

[Chan93] P.K. Chan *et. al.*, "On Routability Prediction for Field Programmable gate Arrays," *Proc. IEEE DAC*, June 1993.

[Chan94] Y-W. Chang, S. Thakur, K. Zhu, and D.F. Wong, "A New Global Routing Algorithm for FPGAs," *Proc. ACM/IEEE ICCAD*, pp. 356-361, 1994.

[Deva89] S. Devadas, "Optimal Layout Via Boolean Satisfiability," *Proc. ACM/IEEE ICCAD*, pp. 294-297, 1989.

[Gree90] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. El Gamal, "Segmented Channel Routing," *Proc. ACM/IEEE DAC*, pp. 567-572, 1990.

[Hall70] K.M. Hall, "An R-Dimensional Quadratic Placement Algorithm," *Management Sci.*, vol. 17, pp. 219-229, Nov. 1970.

[Lemi93] G. Lemieux and S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs," *Proc. ACM Physical Design Workshop*, California, Apr. 1993.

[Marq97] J.P. Marques Silva and K.A. Sakallah, "GRASP--A New Search Algorithm for Satisfiability," *Proc. ACM/IEEE ICCAD*, Nov. 1997.

[Mina93] S-i. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *Proc. ACM/IEEE DAC*, pp. 272-277, June 1993.

[McMu95] L.E. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Path-Driven Router for FPGAs," Proc. *ACM/IEEE International Symp. Field Programmable Gate Arrays*, Feb. 1995.

[Nag95] S.K. Nag and R.A. Rutenbar, "Performance-Driven Simultaneous Place and Route for Island-Style FPGAs," *Proc. ACM/IEEE ICCAD*, Nov. 1995.

[Nag94] S.K. Nag and R.A. Rutenbar, "Performance-Driven Simultaneous Place and Route for Row-Based FPGAs," *Proc. ACM/IEEE DAC*, June 1994.

[Rude93] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Proc. ACM/IEEE ICCAD*, pp. 42-47, Nov. 1993.

[Wood96] R. Glenn Wood, *Routing for FPGAs via Boolean Satisfiability*, M.S. Thesis, Dept. of ECE, Carnegie Mellon University, May 1996.

[Wu93] Y-L. Wu and M. Marek-Sadowska, "Graph Based Analysis of FPGA Routing," *Proc. European Design Auto. Conf.*, pp. 104-109, 1993.

[Wu94] Y-L. Wu and D. Chang, "On the NP-Completeness of Regular 2-D FPGA Routing Architectures and a Novel Solution," *Proc. ACM/IEEE ICCAD*, pp. 362-367, 1994.