# A Boolean Satisfiability-Based Incremental Rerouting Approach with Application to FPGAs

Gi-Joon Nam, Karem Sakallah and Rob Rutenbar†
Department of Electrical Engineering and Computer Science, University of Michigan
†Department of Electrical and Computer Engineering, Carnegie Mellon University

## Abstract

*Incremental redesign is an increasingly essential step in any complex design. Late changes or corrections in functional specifications (so-called "engineering change orders" or ECOs) force us to search for a minimal perturbation that achieves the desired repair. In reconfigurable design scenarios, these incremental repairs may be in response to physical faults: the goal is to "design around" the fault. For FPGAs, incremental rerouting is an essential component of this repair problem. We develop a new incremental rerouting algorithm for FPGAs using techniques from Boolean Satisfiability (SAT). In this application, these techniques have the twin virtues that they (1) represent all possible routing (and rerouting) constraints simultaneously and exactly, and (2) search for rerouting solutions by perturbing all nets concurrently. Preliminary results are promising. For several FPGA benchmarks, we were able to reroute fault reconfigurations that perturb up to 5.74% of all nets for a small number of fault sets (one to four faults) with only 1.55 track overhead per channel on average, with CPU time 0.76 to 4.91 seconds/fault.*

## 1. Introduction

As designs become more complex and design times continue to shrink, complex designs are increasingly subject to small-scale changes, typically late in their design cycles in order to correct errors found, or to meet late changes in specification. For these *Engineering Change Orders* (ECOs), we require a local, minimal-perturbation solution that incrementally changes as little of the design as possible. Since designers have already spent considerable effort to realize the design, we should avoid methods that need to rearrange large sections of the design in response to these small change requests. In particular, we want to avoid creation of additional errors or unanticipated consequences in modules already correct and validated thus far. Incremental redesign techniques have only recently been studied for a few EDA problems [5, 6, 7, 10], notably in logic synthesis and physical design.

There are many scenarios where incremental redesign is useful. In this paper, we focus on the FPGA fault reconfiguration scenario introduced in [6, 7]. In a large FPGA, a few faulty logic cells may occur as a result of a manufacturing defect, or from operational faults in the field. Given the reconfigurable routing fabric, these can conceivably be mapped out by "routing around" the faulty cells--assuming we have sufficient spare logical and wiring resources. The technique exploited in [6, 7] achieves reconfigurations by constructing a chain of replacements from faulty logic cells to adjacent correct logic cells, until a reserved spare/unused cell is reached.

Reservation of these necessary spare resources can be done *statically* or *dynamically*. In a static approach, extra interconnections are reserved as part of the initial routing process so that a specific fault pattern (e.g., one fault per row) can be tolerated. The drawback of static techniques is the large number of statically allocated spares that must be reserved [7]. In a dynamic approach [6], necessary interconnect resources are searched for "on the fly" whenever an assumed fault pattern occurs. In both approaches, however, a spare (unused) logic cell was reserved per row in addition to extra interconnect resources. An unfortunate consequence is the need (in our opinion) to reserve a relatively large number of logic cells and wiring tracks globally, to accommodate a quite small set of anticipated faults. We argue that part of the problem here is the limited *scale* of the redesign perturbations that current techniques can successfully manage. If a small number of nets need to be rewired to insert a logic spare and reconfigure, the problem is manageable. But if a large number of nets must be slightly perturbed to reconfigure in a dense, resource-constrained design, then we believe it is worth considering rerouting techniques with a more global view.

In this paper, we introduce a new incremental rerouting method for island-style FPGAs based on ideas from Boolean Satisfiability (SAT) [4, 12]. The key idea is to transform the geometric FPGA rerouting task into a single Boolean function which represents all the possible routing constraints simultaneously over the existing solution. The generated Boolean function has the property that any *satisfying assignment* (variable assignments that render the function identi-

cally "1") specifies a valid routing solution. This approach has several novel characteristics which distinguish itself from traditional routing approaches [1, 2, 3, 6, 7, 9]. First, it embeds all the nets that should be rerouted *concurrently*. Second, for each net to be rerouted, it considers *multiple* global routing topologies at the same time. Finally, it is able to determine *exactly* the feasibility of solutions; once the generated Boolean function is proven to be unsatisfiable, then we can conclude that there is no feasible routing solution at all. The large SAT problems that result from this rerouting formulation are solved rapidly using an efficient search-based SAT engine called *GRASP* [12].

The rest of paper is organized as follows. Section 2 develops our new SAT-based incremental rerouting formulation algorithm called MSDR_ECO. Section 3 illustrates how to formulate the necessary routing constraints in this SAT style with a small example. Section 4 presents extensive experimental results showing the performance of MSDR_ECO in fault reconfiguration applications. Finally, Section 5 offers some concluding remarks.

## 2. SAT-Based Incremental Rerouting: Basic Boolean Formulation

We base our modeling on the *island-style* FPGA architecture adopted in CGE [2], SEGA [9], VPR [1] and SDR [11]. This is one of the most commonly used layout models in the literature and can be easily adapted to reflect a variety of commercial FPGAs. An island-style FPGA is comprised of a two-dimensional array of *Configurable Logic Blocks (CLBs)*, *Connection Blocks (C-blocks)* and *Switching Blocks (S-blocks)*. IO cells reside on the boundary of the array. The routing capacity of a given FPGA architecture is conveniently expressed by 3 parameters, $W, F_c, F_s$ [3]. The channel width $W$ is the number of tracks in a vertical or horizontal channel. The C-block flexibility $F_c$ is defined to be the number of tracks that each logic pin can connect to. The S-block flexibility $F_s$ denotes the number of other tracks that each wire segment entering an S-block can connect to. In the sequel, we assume that each routing track is fully segmented. That is, each wire segment spans only one block.

We configured this general layout model to mimic the Xilinx XC4000E/X series architecture for some of our experiments [13]. In this specific FPGA architecture, each logic pin in one CLB can connect to any tracks in the channel (i.e., $F_c = W$) and each wire segment entering an S-block can connect to one track on each of the other three sides (i.e, $F_s = 3$) and they take the same track numbers.

A *net* is a set of CLB and/or IO pins that must be electrically connected, which 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. In our approach, a global router is allowed to generate a *set* of *different* global routes for each two-pin connection; each of these will be referred to as a *global route alternative*. 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 routing switches following the topology specified by the global router such that no overlapping of routing resources among detailed routes of different nets occurs.

In [11], we presented a novel FPGA detailed routing formulation method using Boolean Satisfiability (SAT). The basic idea was that we construct a set of Boolean functions representing routing constraints over the entire FPGA, and invoke an Boolean SAT solver on the generated function to find any satisfying assignments. Finally the found SAT solution determines precisely a full FPGA detailed routing solution. Our new, incremental router, **MSDR_ECO** (**M**ultiple-paths **S**atisfiability-based **D**etailed **R**outer for **E**ngineering **C**hange **O**rders) is based on an extended formulation that targets only incremental change. Thus, we assume that an FPGA detailed routing solution is already given, and we need to perturb some minimal fraction of the current solution to finalize the perturbed connections.

For each two-pin connection that should be perturbed, the existing routing solution is ripped up first, and then a global router produces a set of individually feasible global route alternatives. Of course, when considering all perturbed nets concurrently, not all of these promising global alternatives can survive due to detailed route conflicts we will subsequently discover. MSDR_ECO then considers the current detailed routing solution as well as the multiple global route alternatives per each target two-pin connection to generate a Boolean *routability function* $R(X)$ which captures *all* the possible routing constraints over the existing routing solution simultaneously. Finally, a Boolean SAT solver is invoked on the routability function to determine if there exists any legal detailed routing solution.

Our Boolean routability function $R(X)$, where $X$ is a suitable Boolean vector of binary variables that encode the track number for each two-pin connection, can be expressed as the conjunction $R(X) = L(X) \wedge E(X)$ where:

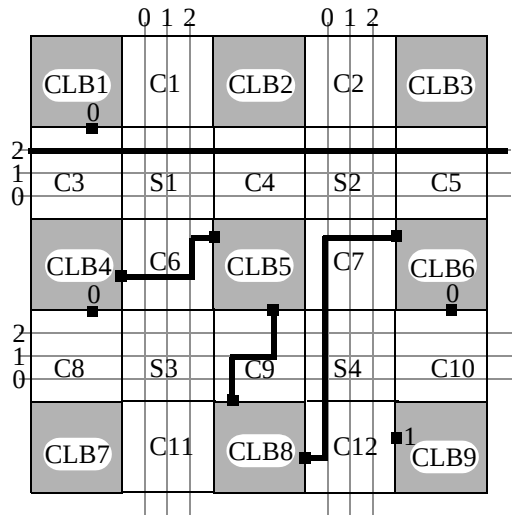
- *Liveness constraint function* $L(X)$ guarantees that at least *one* global route alternative per two-pin connection should be chosen as a final legal routing solution.
- *Exclusivity constraint function* $E(X)$ ensures that electrically distinct nets with overlapping vertical or horizontal spans in the same channel are always assigned to *different* tracks.

The remainder of this section describes each procedure of the overall algorithm in detail.

1. **Global routing:** Given i) target FPGA architecture information, ii) a detailed routing solution, and iii) a set of target two-pin connections to be rerouted, a global router assigns a set of *n* global route alternatives for each two-pin connection, where *n* is defined by a user. The method of generating global routes per two-pin connection is an independent procedure from the detailed routing formulation. In our implementation, we used a standard maze router algorithm [8].
2. **Routing constraint formulation:** Liveness and exclusivity constraint are generated to yield the routing constraint Boolean function $R(X)$ in conjunctive normal form. We walk through a detailed example in the next section.
3. **Routing constraint evaluation:** A Boolean SAT solver is invoked to find a satisfying assignment for $R(X)$ or to show that $R(X)$ is unsatisfiable (i.e., identical to "0"). Any satisfying SAT solution can be reinterpreted as a legal FPGA detailed routing solution.

## 3. SAT-Based Rerouting: Detailed Constraint Formulation Example

The transformation of routing constraints into a Boolean SAT problem is illustrated in Figure 1 with a simple example. Figure 1 (a) shows the existing detailed routing solution. We assume that each horizontal/vertical channel has 3 different tracks numbered *0, 1, 2*. Bold lines in routing channels represent wire segments already taken by other nets. We have two target two-pin connections *A* and *B*, as shown in Figure 1 (b). For the two-pin connection *A*, two different global route alternatives (*AG1* and *AG2*) will be considered simultaneously. For contrast, only one global route alternative *BG1* will be considered for the two-pin connection *B*. As a first step of the transforming procedure, one Boolean vector is assigned to each global route alternative, labeled ***AG1***, ***AG2*** and ***BG1*** respectively. These Boolean vectors each comprise a set of Boolean variables encoding a possible track number assigned to the corresponding two-pin connections. In addition, one Boolean variable is assigned to each global route alternative, labeled *bAG1*, *bAG2* and *bBG1*, and these Boolean variables determine whether the corresponding global route alternative is included in a final routing solution. The liveness and exclusivity constraint functions are generated from these Boolean variables, and the geometry of the global route alternatives. One liveness constraint CNF clause is formed for each target two-pin connection, which is a disjunctive form of all the Boolean variables corresponding to global route alternatives. In this way, at least one of global route alternatives is forced to be chosen as a final routing solution. In the possible case where multiple global route alternatives are selected as part of a final routing solution, the user (or a simple post-processor) can select the most desirable individual solution from among




(a) Current detailed routing solution

Target two-pin connection:
A = (CLB1, pin0) => (CLB9, pin1)
B = (CLB4, pin0) => (CLB6, pin0)
Global route alternatives:
AG1 = (CLB1 pin0, C3, S1, C4, S2, C7, S4, C12, CLB9 pin1)
AG2 = (CLB1 pin0, C3, S1, C6, S3, C9, S4, C12, CLB9 pin1)
BG1 = (CLB4 pin0, C8, S3, C9, S4, C10, CLB6 pin0)

(b) Target connection and global route alternatives

**Liveness constraints:**

$$\boldsymbol{L(X)} = (\boldsymbol{bAG1} \vee \boldsymbol{bAG2}) \wedge (\boldsymbol{bBG1})$$

**Exclusivity constraints:**

$$\begin{aligned}\boldsymbol{E(X)} = {} & [\boldsymbol{bAG1} \rightarrow \boldsymbol{AG1} \neq \boldsymbol{2}] \wedge [\boldsymbol{bAG1} \rightarrow \boldsymbol{AG1} \neq \boldsymbol{0}] \wedge \\ & [\boldsymbol{bAG2} \rightarrow \boldsymbol{AG2} \neq \boldsymbol{0}] \wedge [\boldsymbol{bAG2} \rightarrow \boldsymbol{AG2} \neq \boldsymbol{1}] \wedge \\ & [\boldsymbol{bAG2} \rightarrow \boldsymbol{AG2} \neq \boldsymbol{2}] \wedge [\boldsymbol{bBG1} \rightarrow \boldsymbol{BG1} \neq \boldsymbol{1}] \wedge \\ & [(\boldsymbol{bBG1} \wedge \boldsymbol{bAG2}) \rightarrow \boldsymbol{BG1} \neq \boldsymbol{AG2}]\end{aligned}$$

**Final routability Boolean function:**

$$\boldsymbol{R(X)} = \boldsymbol{L(X)} \wedge \boldsymbol{E(X)}$$

(c) Liveness & Exclusivity constraints

Routing Solution: Net A = (**AG1** = 1)
Net B = (**BG1** = 0 or 2)

(d) Routing solution

**Figure 1. Example of generating routing constraints Boolean function.**

them. Exclusivity constraints prevent any routing resource from being overused by different nets; there are two types of exclusivity constraints:

- **Background-path avoidance:** this is the case when a global route alternative for a target two-pin connection overlaps with an existing *detailed* routing path. For example, a global route alternative *AG1* in the connection block C3 and C4 should not be assigned track number 2. Thus $(\boldsymbol{AG1} \neq \boldsymbol{2})$. These constraints take the simple form of a vector-constant inequality.
- **Potential-path avoidance:** this is the case when two

different global route alternatives from *different* nets have overlapping connection blocks. For example, the global route alternative *AG2* of a net *A* and alternative *BG1* of net *B* have overlapping C-block C9. Thus $(\boldsymbol{AG2} \neq \boldsymbol{BG1})$. These constraints take the simple form of an inequality between two Boolean vectors.

Both types of exclusivity constraints are, however, enforced only when the corresponding global route alternatives are selected as part of a final routing solution. Thus, every exclusivity constraint takes the implication form with the corresponding Boolean variable such as $[\boldsymbol{bAG1} \rightarrow (\boldsymbol{AG1} \neq \boldsymbol{2})]$.

Between global route alternatives from the *same* net, however, no exclusivity constraints are constructed (for example *AG1* and *AG2* in C-block C3). In this way, they might share the same track in the channel, which is a perfectly legal routing solution. The final routing constraint function $\boldsymbol{R}(\boldsymbol{X})$ is just a conjunction (logical "AND") of all the liveness and exclusivity constraint functions represented in CNF forms. Figure 1 (d) shows the possible routing solution re-interpreted from the SAT solution. It says *net A* should take the global route alternative *AG1* and the assigned track number in the region is 1; and, *net B* can take either track number 0 or 2.

## 4. Experimental Results

We tested the effectiveness of the MSDR_ECO rerouting tool under the FPGA fault reconfiguration scenario described in [6,7]. In the first set of experiments, up to 4 physical faults were assumed to be detected on CLBs which are already mapped into. Reconfiguration proceeds by first relocating the logic elements in the faulty CLBs into nearby, unused, defect-free CLB positions, and then incrementally rerouting only the necessary connections without affecting the logical functionality of original design. In [6,7], it is assumed that there exists one spare (unused) CLB per row, reserved for a fault reconfiguration. Thus only one fault was allowed per row of CLBs. For MSDR_ECO, however, no such special fault model is assumed; faults can happen in any CLB. Also no extra logic cells were reserved for reconfiguration. Thus, as long as there exist *some* unused CLBs, MSDR_ECO will try to reconfigure.

It is difficult to compare the performance between [6,7] and MSDR_ECO exactly, because [6,7] use SEGA [9] for the original detailed routing task, while MSDR_ECO uses VPR [1]. For the target circuits in our experiment, VPR produces more compact detailed routing solutions than SEGA, requiring 43.72% less tracks per channel. Thus, it can be fairly claimed that MSDR_ECO performs reconfigurations in a denser and more difficult environment, with less geometric "slack" for use during rerouting, than [6,7].

The key metrics considered important here are track overhead and reconfiguration time. As mentioned in Section 2, MSDR_ECO is able to consider multiple global routing alternatives per connection. For this experiment, maximum 5 different global route alternatives were considered for each connection. All experiments were conducted on a SUN Ultra Sparc-2 running SunOS with 1 Gb of physical memory, and GRASP [12] is used as a SAT solver. The results of the first experiment are shown in Table 1.

**Table 1. Performance for first fault reconfiguration experiment with MSDR_ECO.**

| Circuit | | FPGA | | MSDR_ECO | | |
|---|---|---|---|---|---|---|
| | | | | Tracks required | | |
| Name | #2pin conns | Size of CLB Arrays | % CLB utilization | Before ECO | After ECO | Reconfiguration time/fault |
| 9symml | 259 | 9 x 9 | 86.42 | 5 | 6.48 | 1.33 |
| apex7 | 300 | 11 x 11 | 63.64 | 5 | 6.08 | 0.76 |
| C499 | 312 | 10 x 10 | 74.00 | 6 | 7.50 | 2.00 |
| C880 | 656 | 14 x 14 | 88.78 | 7 | 8.50 | 3.27 |
| C1355 | 312 | 10 x 10 | 74.00 | 6 | 7.65 | 1.13 |
| example2 | 444 | 19 x 19 | 33.24 | 6 | 7.08 | 2.56 |
| k2[a] | 1257 | 19 x 19 | 99.16 | 10 | 12.33 | 4.91 |
| term1 | 202 | 8 x 8 | 84.38 | 5 | 6.23 | 0.78 |
| too_large | 519 | 13 x 13 | 87.58 | 7 | 8.80 | 4.01 |
| vda | 722 | 15 x 15 | 92.44 | 8 | 9.85 | 3.42 |
| Average | | | | 6.5 | 8.05 | 2.42 |

a. For circuit k2, only up to 3 CLB faults were tried, because there are only 3 unused CLBs before ECO.

- The first two columns describes our benchmark circuits with the number of two-pin connections.
- Column 3 and 4 specify how each circuit mapped into its target FPGA: the size of the overall CLB array, and the CLB utilization. The higher the CLB utilization is, the more difficult the reconfiguration is because a smaller number of spare logic resources are available.
- Columns 5 to 7 summarize the rerouting performance of the overall MSDR_ECO flow. Initial detailed routing solutions from VPR [1] are shown in column 5. Column 6 gives the actual track count required to achieve successful fault reconfiguration. This number is averaged over 40 separate random trials, with up to 4 CLB faults/trial. Column 7 shows the reconfiguration CPU time per fault in seconds.
- The number of two-pin connections perturbed through this experiment varies from 5 to 35, and they account for 0.96% to 5.74% of the original nets (on average 3.65%).

Even with a more liberal fault model, MSDR_ECO was able to successfully reconfigure for up to 4 faults with minor routing track overhead in negligible time.

Of course, delay is also a critical concern for any routing perturbation. We are not yet considering this directly. We do know that for all these cases, incremental rerouting did not increase the *maximum* wire length of the given circuits. Thus, at least the solution is not dramatically meandering

**Table 2. Fault reconfiguration on Xilinx XC4000X-Style FPGAs**

| Circuit | CLB Size | CLB Usage | #2-pin Conns | Reconfig Avg Time/ Fault | ECO Avg #nets |
|---|---|---|---|---|---|
| alu4 | 28 x 28 | 97% | 4235 | 17.6 | 16.07 |
| apex2 | 31 x 31 | 92% | 5405 | 7.71 | 16.43 |
| apex4 | 26 x 26 | 80% | 3604 | 9.91 | 19.97 |
| bigkey | 54 x 54 | 27% | 5064 | 8.31 | 16.73 |
| diffeq | 28 x 28 | 96% | 3986 | 5.45 | 20.60 |
| dsip | 54 x 54 | 22% | 3872 | 10.14 | 23.90 |
| misex3 | 27 x 27 | 89% | 3984 | 6.40 | 20.30 |
| seq | 30 x 30 | 85% | 4986 | 8.84 | 19.70 |
| tseng | 23 x 23 | 91% | 2764 | 3.56 | 21.40 |

the perturbed wires. However, we have not yet assessed the impact on any actual critical paths. This is clearly an area for closer examination for this approach; in general, it appears that the timing impact is dominated not only by the density of the local routing, but also by the proximity of nearby spares in the CLB lattice.

One drawback of this first experiment is that, in any practical situation, we have a *fixed* set of available routing resources, known for our target FPGAs in advance. In other words, the main interest is to know whether designs can fit into target FPGAs or not, rather than to optimize the number of routing resources used. In our second experiment, we employed a Xilinx XC4000X series architecture [13] and similar fault reconfiguration simulations were performed with practical-sized benchmark circuits. In this experiment, a small portion of the tracks (5 tracks, globally, which was the worst case track overhead in the first experiment) were reserved in advance for the possible fault reconfiguration in the future. In other words, initial detailed routing was performed by VPR with 5 less tracks than those available in XC4000X series. Then, MSDR_ECO completed fault reconfigurations assuming up to 4 different CLB faults can occur at random positions.

Table 2 describes results of the second experiment. The first 4 columns describe the benchmark circuits, size of CLB array, usage % of CLBs, and number of two-pin connections. The next column is the average reconfiguration time per fault in seconds, and the final column shows the average number of two-pin connections that were incrementally rerouted. The results were collected after 40 random fault injection trials per circuit. It is not surprising that the reconfiguration times per fault are quite larger than those in the first experiment because the sizes of circuits considered are also far larger. Yet, it seem clear that 5 extra global tracks were large enough to accommodate 4 CLB faults per case and every case was successfully fault-reconfigured within a few minutes. As before, we know that the length of the longest wire does not increase, but not the specific delay impact. Especially for dense, realistic architectures, ensuring timing closure after rerouting may require a different track overhead.

Overall, we regard these as very satisfactory results. The Boolean SAT-based routing formulation approach seems be a suitable method of attack for ECO applications which require some small solution perturbation. With advanced SAT solver technology, it was possible to consider multiple global route topologies per connection concurrently, which appears to pay off in reducing the demand for setting aside spare routing resources.

## 5. Conclusion

In this paper, we described a new strategy for FPGA incremental rerouting, using ideas from Boolean SAT. Experiments with injecting small sets of random faults and reconfiguring the routing as necessary suggest the viability of this approach. Preliminary results are encouraging: we have been to able to reconfigure up to 4 different CLB faults per circuit with modest routing resource overheads in negligible time. Unaddressed as yet is the large-scale impact on timing. However, this seems possible by carefully restricting the allowable alternative global routing paths, and adding extra constraints that structurally prohibit excessively long path detours.

## 6. References


[1] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research", *the 7th Annual Workshop on Field Programmable Logic and Applications*, 1997.

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

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

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

[5] J. Cong, J. Fang and K.-Y. Khoo, "An Implicit Connection Graph Maze Routing Algorithm for ECO Routing," *Proc. IEEE/ACM ICCAD*, Nov. 1999.

[6] S. Dutt, V. Shanmugavel and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconfiguration in FPGAs," *Proc. ACM/IEEE ICCAD*, Nov. 1999.

[7] F. Hanchek and S. Dutt, "Methodologies for Tolerating Cell and Interconnect Faults in FPGAs," *IEEE Trans. on Computers*, vol. 47, no. 1, Jan. 1998.

[8] C. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Electronic Computers*, 1961.

[9] G. Lemieux, S. Brown, and Z. Vranesic, "On Two-Step Routing for FPGAs", *ISPD*, pp. 60-66, April 1997.

[10] C. Lin, K.-C. Chen, S.-C. Chang and M. Marek-Sadowska, "Logic Synthesis for Engineering Change," *the 32nd ACM/ IEEE Design Automation Conference*, June 1995.

[11] G. Nam, K. Sakallah, and R. Rutenbar, "Satisfiability-Based Layout Revisited: Detailed Routing of Complex FPGAs Via Search-Based Boolean SAT," *Intl' Symposium on FPGAs*, Feb. 1999.

[12] J. Silva and K. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability", *IEEE Trans. on Computers*, vol. 48, no. 5, May 1999.

[13] http://www.xilinx.com/partinfo/databook.htm.