Combined Architecture/Algorithm Approach to Fast FPGA Routing

Marcel Gort and Jason H. Anderson, *Member, IEEE*

***Abstract*— We propose a new field-programmable gate array (FPGA) routing approach, which, when combined with a low-cost architecture change, results in a 40% reduction in router runtime, at the cost of a 6% area overhead and with no increase in critical path delay. Our approach begins with PathFinder-style routing, which we run on a coarsened representation of the routing architecture. This leads to fast generation of a partial routing solution where the signals are assigned to groups of wire segments rather than individual wire segments. A Boolean satisfiability (SAT)-based stage follows, generating a legal routing solution from the partial solution. We explore approximately 165 000 FPGA switch block architectures, showing that the choice of the architecture has a significant impact on the complexity of the SAT formulation, and by extension, on routing runtime. Our approach points to a new research direction, namely, reducing FPGA computer-aided design runtime by exploring FPGA architectures and algorithms together.**

***Index Terms*— Computer-aided design (CAD), field-programmable gate array (FPGA) architecture, FPGA routing, satisfiability (SAT).**

I. Introduction

As FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs) become larger, the runtime required to execute the associated computer-aided design (CAD) tools gets worse. It can now take days to compile the largest industrial FPGA designs from hardware description level to bitstream. In the past, worsening CAD tool runtime was mitigated by increases in the uniprocessor clock speed. However, this is no longer the case. The high current density in modern processors has created a "power wall," which restricts increases in processor clock speeds. The gap between the size of FPGA devices and the ability of CAD tools to handle them is widening with every process generation. In addition to decreasing productivity for hardware engineers, long runtimes are an impediment to the adoption of FPGAs by software developers, who are used to compilation times measured in seconds or minutes, not hours or days.

Previous efforts to reduce FPGA CAD runtimes have focused on algorithmic changes [1]–[5] or parallelization [6]–[8]. In this paper, we reduce router runtime via a combined CAD and architecture approach. Interactions between CAD and architecture are known to affect FPGA speed, area, and power, however, the impact of these interactions on runtime has not been well studied. We propose a new two-stage FPGA routing algorithm that takes advantage of increased flexibility in the FPGA switch block (SB), resulting in reduced router runtime.

At a high level, our routing approach works as follows: first, a PathFinder-style router [9] assigns signals to small groups of wire segments, called *wide wires*, rather than to single wire segments. This is made possible by *coarsening* the routing resource (RR) graph, which allows PathFinder to terminate early. Second, an embedding stage assigns each signal from a wide wire onto one of the wire segments contained within. We express the embedding problem as a Boolean satisfiability (SAT) problem and use a SAT solver (MiniSat [10]) to solve it. The combination of this new routing approach with a modified routing architecture leads to router runtime reductions of up to 40% compared to an accelerated version of VPR [11] routing, with only 6% area overhead, and no increase to the critical path delay or wire length.

Our approach bears some resemblance to two-stage global-detailed routing, where global routing assigns signals to entire FPGA channels, and detailed routing assigns signals to wire segments within those channels. In our case, however, the first stage assigns signals to small groups of wire segments rather than to entire channels, meaning that routing decisions can be made using detailed timing and congestion information, leading to high-quality routes. A preliminary version of this paper [12] described our two-stage routing approach and presented a new *grouped* routing architecture that allows embedding to find legal signal-to-wire segment assignments. A preliminary architecture exploration for this grouped routing architecture was also described. In this paper, we build on the prior work, making several contributions, including the following.

1) A detailed treatment of the SAT formulation used for the embedding step.
2) An exploration and analysis of approximately 165 000 permutations of the grouped routing architecture are carried out.
3) An analysis of the robustness of SB architectures to changes in logic block (LB) placements is presented.
4) Modifications to the PathFinder algorithm are made to make it aware of the embedding stage.

We believe that our approach of addressing routing algorithms and architecture together represents an entirely new direction for reducing tool runtime—one with fertile ground for further research. The remainder of this paper is organized as follows. Section II provides the relevant background on FPGA routing algorithms and architecture, SAT, and previous work related

Manuscript received November 9, 2011; revised March 1, 2012; accepted May 18, 2012.

The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: gortmarc@eecg.utoronto.ca; janders@eecg.utoronto.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2202326

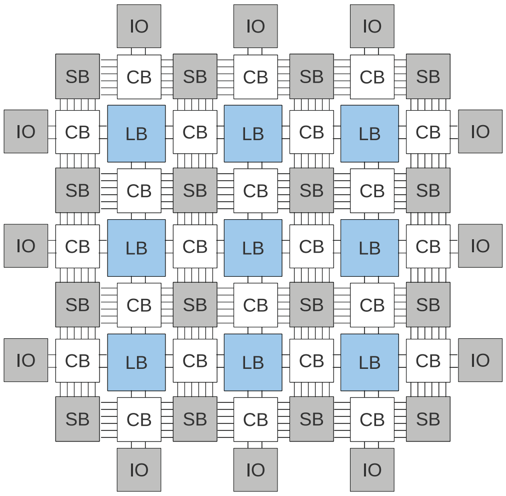


Fig. 1. FPGA architecture.

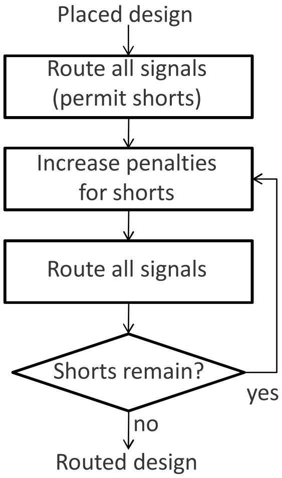

Placed design
Route all signals (permit shorts)
Increase penalties for shorts
Route all signals
Shorts remain?
yes
no
Routed design

Fig. 2. Negotiated congestion routing flow.

to runtime-driven FPGA architecture design. The proposed two-stage routing approach is described in Section III. In Section IV, we describe our grouped routing architecture and present results from an extensive architecture study. In Section V, we outline modifications to VPR PathFinder that generate coarse routes more likely to lead to SAT. An experimental study appears in Section VI. Conclusions and suggestions for future work are offered in Section VII.

II. Background

A. FPGA Architecture

Fig. 1 depicts a basic island-style FPGA. Each LB has a capacity, which represents the number of lookup tables (LUTs) and flip-flops that can be contained therein. LBs connect to wire segments through programmable switches in the connection blocks (CBs). Wire segments connect to other wire segments using programmable switches in the SBs. Connections between LBs are made by turning on the appropriate switches in the SBs and CBs. The degree of connectivity inside SBs and CBs is described using the parameters F_s and F_c, respectively. F_s describes the number of wire segments coming out of an SB than can be reached by each wire segment coming into an SB, on average. In other words, it represents the average fanout of each wire coming into an SB. F_c describes the fraction of wire segments in a channel that can be directly reached by an LB pin through a CB. $F_{c_{\text{in}}}$ refers to the connectivity to LB input pins while $F_{c_{\text{out}}}$ refers to the connectivity to LB output pins.

B. FPGA Routing

FPGA routing is one of the most time-consuming stages in the CAD flow. It is responsible for finding routes for connections between LBs, using wire segments and programmable switches. Naturally, no two signals may use the same wire segment. The two largest FPGA vendors, Xilinx and Altera, use a variant of the PathFinder negotiated congestion routing algorithm [9] in their commercial routers [13], [14]. PathFinder is also used in the publicly available VPR FPGA placement and routing framework [11], which we modified and used in this paper. Fig. 2 gives an overview of the negotiated congestion approach used in the VPR PathFinder. First, all signals in a placed design are routed in the best manner possible (e.g., minimum delay), permitting shorts between the signals (two or more signals may use the *same* wire). Then, the penalties associated with the shorts are increased, and the signals are rerouted, avoiding shorts where possible. The process of increasing the penalties for shorts and rerouting the signals continues iteratively until all shorts are removed and the routing is feasible.

VPR PathFinder uses a RR graph to represent the FPGA interconnect. Graph nodes represent wire segments, input pins, or output pins. Programmable switches between wires or pins are represented as edges between the nodes. Each node has a capacity, which indicates the number of signals it can accommodate, and an occupancy, which indicates the number of signals that currently occupy the RR node. Routing continues until the occupancy of each node is not greater than its capacity, at which point the routing is feasible.

C. Enhancements to Baseline Router

VPR routing was used for all experiments reported in this paper. However, in order to bring the router more in line with modern industrial routers, we made two modifications, which were first described in [5]. The first modification reduces the contention for LB output pins by forcing a multisink signal to use the same LB output pin for all of its sinks. The second modification allows a PathFinder iteration to skip the rerouting of signals that use no congested RR nodes. When combined, these two modifications result in a 3.3× speedup in router runtime at the cost of a 2% increase in both critical path delay and wire length [5]. We use this enhanced version of VPR routing as our baseline in this paper. In other words, the runtime improvements reported in this paper are on top of the "fast" baseline.

D. Architectures for Fast CAD

A limited number of papers have explored the interaction of FPGA architecture and CAD runtime. In [15], the PLASMA

FPGA architecture was proposed, which was intended as a prototyping platform. The entire routing process takes 3 seconds because of a very flexible routing fabric, which leads to a much simplified routing problem. The cost of fast routing is a dramatically higher silicon area. In [16], Lysecky *et al.* use a simplified planar FPGA routing architecture to reduce the number of possible routing paths. The authors split the routing into a global PathFinder-based routing stage and a detailed routing stage, which can be represented as a graph coloring problem owing to the planar architecture. The authors also modify their implementation of PathFinder to only rip up and reroute signals with shorts. A 3× improvement in router runtime is reported compared to VPR, at the cost of 30% increase in critical path delay [17]. No comparison is made with a standard (nonplanar) routing architecture.

In [18], Chin *et al.* explore the interaction between routing architecture parameters and place and route runtime. They show that increasing the LB capacity significantly impacts the place and route runtime by reducing the number of LBs that have to be placed and the number of connections that must be made to connect them. In [19], the same authors explore how changes to the LB architecture can affect the CAD runtime. In this paper, we explore the CAD runtime impact of FPGA architecture changes in conjunction with CAD algorithm changes. By tailoring CAD algorithms to the architectures for which they are being run, new insights can be obtained on the interaction between FPGA architectures, CAD algorithms, and CAD runtime.

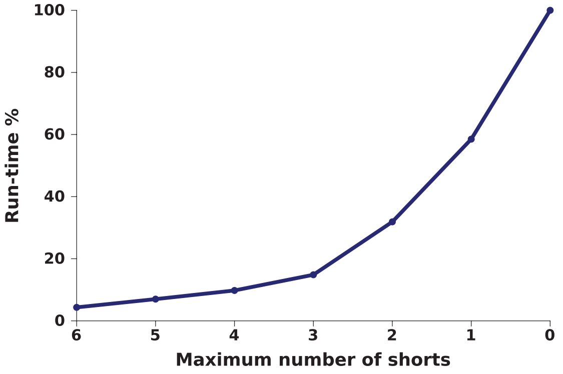


Fig. 3. Percentage of router runtime versus maximum RR node shorts.

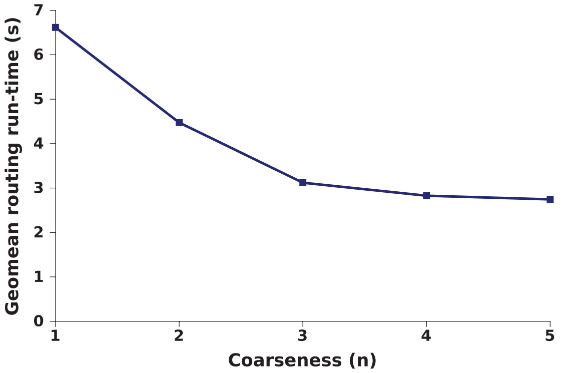


Fig. 4. Geometric mean router runtime versus coarseness.

E. Boolean SAT

A SAT problem is a Boolean formula typically expressed in conjunctive normal form (CNF), which essentially is a product-of-sums functional representation. The solution to the SAT problem is an assignment of TRUE or FALSE to each of the variables such that the Boolean formula evaluates to TRUE. If a solution exists, the problem is said to be SAT; otherwise, it is unsatisfiable (UNSAT). Several efficient academic SAT solvers exist, the most popular of which is the MiniSat [10], which we use in this paper (ver. 2.2.0). We chose SAT because of the rapid advances in SAT runtime in the last decade. One alternative approach, which we did not explore, is to use binary integer linear programming to solve a SAT formulation, as in [20].

III. New Two-Stage Routing Approach

A. Motivating Experiment

For all experiments in this paper, we target an FPGA with single-driver wire segments that span two LB tiles, though our approach can be adapted to work with mixed wire segment lengths. The 16 benchmark circuits with the longest router runtime were selected from the 20 largest MCNC benchmarks commonly used in FPGA CAD research, as well as the set of benchmarks that are packaged with VPR 5.0 [11]. Circuits were mapped into four-input LUTs using ABC [21], then clustered using T-Vpack [22] into LBs with 10 four-LUTs and 22 inputs, then placed to the smallest m row × m column FPGA that would fit the circuit. Across all runs, each circuit was routed using a fixed channel width of 1.2× the minimum channel width needed to route the circuit.

Our routing approach is based on the observation that the unmodified PathFinder spends a significant amount of time attempting to legalize *almost legal* routing solutions. Fig. 3 shows the normalized geometric mean runtime of PathFinder (across all circuits) as a function of the maximum number of signals shorted on any single wire segment. The figure shows that approximately 40% of the runtime is spent resolving the congestion of a routing solution where no RR node is overutilized by more than one signal. This suggests that PathFinder quickly reduces congestion in general, but is slow at fine-tuning the routing solution to make it entirely legal. Our routing approach takes advantage of this property by first terminating PathFinder when the routing solution is almost legal, and then using a SAT-based approach to generate a fully legal solution, providing an overall runtime reduction.

B. Coarse Routing Stage

Our first routing stage leverages the observation made above by routing with a coarsened RR graph, where some low-level details are abstracted away. The RR nodes that represent sets of adjacent wire segments are collapsed together, creating super nodes. While RR nodes typically have a capacity of 1, our super nodes have capacities greater than 1, meaning that they can accommodate more than one signal's route. We refer to these super nodes as wide wires. Wire segments may be grouped only with other wire segments of the same length,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

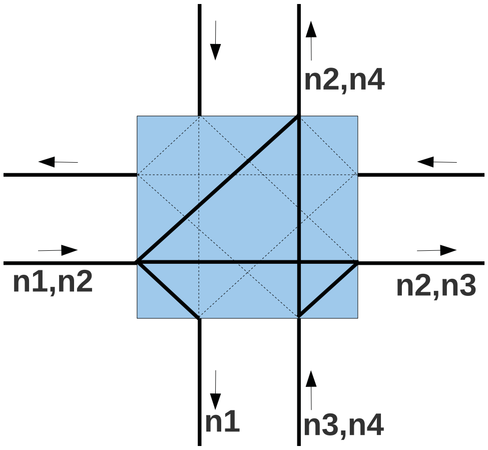


Fig. 5. Coarse routing solution for four signals: $n1$, $n2$, $n3$, and $n4$.

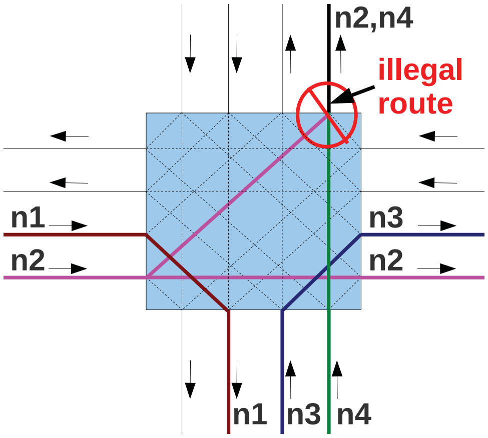


Fig. 6. Best embedding solution based on the coarse routing in Fig. 5.

with end points at the same locations, and that drive in the same direction.

Edges in the RR graph are also coarsened to create super edges. A super edge connects two wide wires if there are any connections between their constituent wire segments. The bold lines going through the SB in Fig. 5 represent super edges connecting wide wires. Fig. 6 shows the uncoarsened edges that correspond to these super edges. Since details of the flat edges are abstracted away, our coarsening method is lossy. Additionally, signals that are grouped together on one wide wire are not tied together for the entirety of their routes (as in bus-based routing). Rather, different signals can follow different edges out of a wide wire.

As expected, the impact of RR graph coarsening on router runtime varies depending on the number of wire segments in a wide wire, which we term coarseness, n. At one extreme, when the coarseness is equal to the channel width ($n = W$), our coarse routing is equivalent to global routing. Conversely, setting $n = 1$ leads to an uncoarsened RR graph (standard PathFinder). The points in between can be viewed as partial global routing. In Fig. 4, we vary n and show the geometric mean runtime of PathFinder across all benchmarks. W is set to 1.2 times the minimum W needed to route the circuit, rounded to the nearest even factor of n to avoid having wide wires with different capacities. Router runtime decreases when n is increased, with a 60% runtime reduction observed with $n = 5$ compared to when $n = 1$. Note that the runtime values shown in Fig. 4 do not consider the embedding time that would be necessary to finalize the solution. After the coarse routing phase, routes are not fully defined in terms of individual wire segments on the FPGA; rather, they are defined in terms of wide wires. We refer to the not yet fully defined routes as "illegal" for the remainder of this paper. In the next section, we show that a net runtime improvement can be achieved by using a fast embedding approach.

C. Embedding Stage

The output of the coarsened routing stage is not a complete routing solution because signals are not assigned to individual wire segments. A decoarsening or embedding stage must be included in order to generate a legal detailed routing solution from the partial solution, assigning each signal to a single wire segment. The embedding difficulty depends on the connection patterns of the underlying routing architecture. If the routing architecture has full connectivity between wires within each wide wire, embedding is trivial, since any assignment of signals to tracks is legal. On the other hand, if only one-to-one connectivity exists, meaning that a wire segment within a wide wire only connects to a single wire within a different wide wire, then it is possible that no legal embedding exists. In such a case, PathFinder could be rerun on an uncoarsened RR graph.

We provide an example of a routing with no legal embedding in Figs. 5 and 6. Fig. 5 shows the routes through an SB for four signals ($n1$, $n2$, $n3$, $n4$). In this figure, solid lines indicate wide wires with $n = 2$. Each is labeled with the signals (up to 2) that use it. Fig. 6 shows the best embedding solution constrained to the routes chosen in Fig. 5. The wide wires have been expanded (flattened) so that, in this figure, each solid line represents one wire segment. Observe that no assignment of signals to wire segments can avoid shorts. However, by adding some flexibility to the connectivity between the wide wires, it is possible to arrive at a legal embedding. We speculate that an intermediate amount of connectivity will allow the vast majority of coarse routing solutions to be embedded while avoiding the significant area overhead of having full connectivity between constituent wire segments of connected wide wires. We test this hypothesis in Section IV.

1) SAT Formulation: We formulate the embedding problem as a Boolean SAT problem in the CNF form. Our formulation is inspired by the SAT-based detailed routing approach presented in [23], which, given a global routing, formulates detailed routing as a SAT problem instance. The authors generate two types of clauses to represent routing constraints: exclusivity constraints and liveness constraints. Exclusivity constraints ensure that no shorts exist on any routing track. Liveness constraints ensure that, for each signal, there exists a path from the signal's source to each of its sinks. Our SAT formulation uses the same types of constraints, though we express them differently to best serve our embedding problem. In our formulation, each variable represents a signal-to-wire

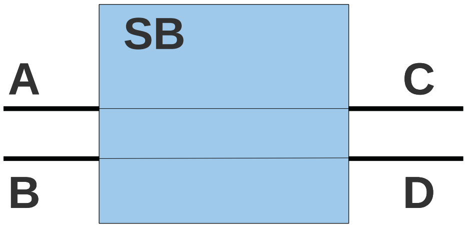


Fig. 7. Flattened representation of one-to-one connections between two wide wires.

segment assignment. If the variable is TRUE, then the signal is routed through the wire segment. We express exclusivity and liveness constraints in terms of these variables.

Exclusivity constraints express that, if a variable is TRUE, all other variables for the same wire segment must be FALSE. For example, if $n = 3$, and signals 1, 2, and 3 are on a wide wire with a constituent wire segment A, we generate the clauses $(\overline{1A} \vee \overline{2A}) \wedge (\overline{1A} \vee \overline{3A}) \wedge (\overline{2A} \vee \overline{3A})$ for that wire segment, where variable $1A$ corresponds to signal 1 using wire segment A. These clauses ensure that at most one of $1A$, $2A$, and $3A$ is TRUE.

Liveness constraints ensure that at least one legal connection is made through each CB and SB on a signal's path, thereby ensuring that there is a path from that signal's source to each of its sinks. They are represented using two types of clauses. The first type ensures that at least one wire segment within each wide wire through which a signal passes is used by the signal. In other words, at least one of the variables tied to the signal/wide wire pair must be TRUE. This type of clause is simply an OR of these variables. For example, to ensure that signal 1 uses least one of wire segments A, B, or C (all within the same wide wire), the clause $(1A \vee 1B \vee 1C)$ is generated. The second type of clause ensures that connection patterns in SBs are honored. Each super edge has an underlying connection pattern that is not represented in the coarsened routing stage, but must be considered in the embedding stage. Clauses are generated to ensure that variables corresponding to wire segments in adjacent wide wires (connected by a super edge) are not both TRUE when no connection exists between them.

For example, consider a RR graph with $n = 2$. Suppose that signal 1 passes through two adjacent wide wires. The connection pattern between these two wide wires is shown in Fig. 7. The following clauses are generated to ensure that only legal connections are formed: $(\overline{1A} \vee \overline{1D}) \wedge (\overline{1B} \vee \overline{1C})$.

The main differences between our SAT-based embedding approach and the detailed routing approach presented by Nam *et al.* [23] are as follows. 1) Our liveness constraints through the SBs ensure that illegal connections are not made, rather than ensure that legal connections are made. This results in fewer SAT clauses when there are many possible legal connections. 2) We support multiterminal signals. 3) Our formulation supports any arbitrary connectivity patterns between wide wires.

We now describe the full SAT formulation to solve the embedding problem shown in Fig. 9. For this example, we would like to embed signals 1 and 2, which have been mapped onto wide wires with $n = 2$, onto individual wire segments A, B, C, and D. The SAT formulation described below can be used as input to a SAT solver to obtain a legal embedding for this problem.

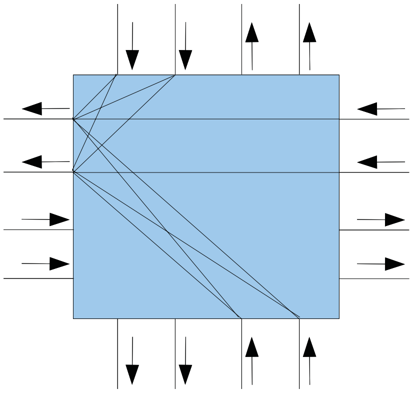


Fig. 8. Connectivity pattern (flattened) guaranteed to lead to SAT assignment.

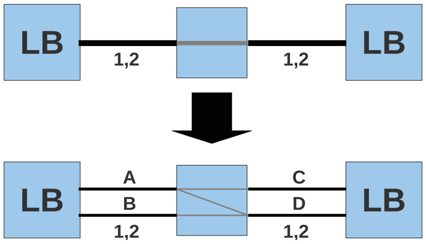


Fig. 9. Embedding example of assigning signals 1 and 2 to wire segments A, B, C, and D.

The exclusivity clauses ensure that no two signals embed onto the same wire. The clauses below ensure that only one of the signals 1 and 2 can use wire segments A, B, C, or D

$$(\overline{1A} \vee \overline{2A}) \wedge (\overline{1B} \vee \overline{2B}) \wedge (\overline{1C} \vee \overline{2C}) \wedge (\overline{1D} \vee \overline{2D}). \quad (1)$$

The first type of liveness constraints ensures that each signal uses at least one of the constituent wire segments within each used wide wire

$$(1A \vee 1B) \wedge (2A \vee 2B) \wedge (1C \vee 1D) \wedge (2C \vee 2D). \quad (2)$$

The second type of liveness constraints ensures that no invalid connections are made through the SBs. In the example, this amounts to ensuring that, for a given signal, the variables associated with wire segments B are not TRUE while variables associated with wire segments C are also TRUE. Connections between all other wire segment pairs are valid, so no clauses need be generated for those pairs. The liveness constraints are given by

$$(\overline{1B} \vee \overline{1C}) \wedge (\overline{2B} \vee \overline{2C}). \quad (3)$$

This means that signal 1 cannot use both wire segments B and C, and signal 2 cannot use both wire segments B and C. The above clauses can be AND'ed together to make a Boolean formula. A legal embedding can be generated by finding an assignment of TRUE or FALSE for each of the variables in this formula that makes the formula evaluate to TRUE.

The overall formula, as well as a SAT solution, is

$$\begin{aligned} F = {} & (\overline{1A} \vee \overline{2A}) \wedge (\overline{1B} \vee \overline{2B}) \wedge (\overline{1C} \vee \overline{2C}) \wedge (\overline{1D} \vee \overline{2D}) \\ & \wedge (1A \vee 1B) \wedge (2A \vee 2B) \wedge (1C \vee 1D) \wedge (2C \vee 2D) \\ & \wedge (\overline{1B} \vee \overline{1C}) \wedge (\overline{2B} \vee \overline{2C}) \end{aligned} \quad (4)$$

$$\begin{aligned} & (1A = T, 1B = F, 1C = T, 1D = F, 2A = F, 2B = T, \\ & 2C = F, 2D = T). \end{aligned} \quad (5)$$

Interestingly, we also tried using PathFinder for our embedding stage, with routes constrained to those chosen during the coarse routing stage. PathFinder was usually *unable* to find a legal embedding, except where full connectivity existed on all super edges. This makes intuitive sense because the embedding problem is highly constrained, which makes it difficult to solve using heuristic approaches, such as those in PathFinder. In the embedding stage, PathFinder cannot encourage routes to avoid entire congested areas of the FPGA, since the routes have already been heavily constrained by the coarse routing step. It appears as though the embedding step must be solved by directly considering conflicts between individual routes. Our SAT-based embedding approach discovers viable solutions that cannot be found by PathFinder when constrained by the coarse routing.

IV. Routing Architecture Study

In this section, we describe a grouped routing architecture with connection patterns that match those required at each stage of our routing approach. We chose to make connections between wide wires in an SB in accordance with the well-known Wilton connectivity pattern [24], as it has been shown to result in good routability. In the Wilton architecture, each wide wire end point entering an SB connects to three other wide wires ($F_s = 3$).

Regarding the low-level connectivity *between* the constituent wire segments of two connected wide wires, a broad range of options is available. Increasing the amount of connectivity causes a higher area overhead, but also makes finding a satisfactory assignment more likely, as well as faster. In a previous study [12], we explored a small fraction of all potential connection patterns, approximately 2000 in total. We allowed for different connectivity patterns between wide wires by making connections in each of the six directions through an SB (N↔S, E↔W, N↔E, N↔W, S↔E, S↔W).[1] In total, there are 2^{6n^2} possible low-level connectivity patterns for a particular value of n (2^{n^2} for each direction, to the power of six directions), or 1.4×10^{45} connectivity patterns in total for n ranging from 2 to 5. Although some interesting SBs were found in our original experiments (e.g. Fig. 8, which is guaranteed to lead to SAT for any benchmark), the search space was left largely unexplored.

For this paper, we explore approximately 165 000 randomly generated SB connectivity patterns, using a Monte Carlo approach. For any wide edge existing between two wide wires, there exist n^2 switches that can potentially exist in the underlying connectivity pattern. We randomly generate SB patterns by assigning each of these potential switches a certain probability of existing. For example, with $n = 2$, there are four (2^2) potential switches in each wide edge. If we set the existence probability to 0.5, a wide edge with two underlying switches is most likely, but other wide edges are also possible. Fig. 7 shows a wide edge with two out of four potential edges existing. Using higher or lower existence probabilities will cause our SB generator to focus on SB architectures with more or less area overhead, respectively. To generate a wide range of SB architectures, we use a range of existence probabilities for each value of n.

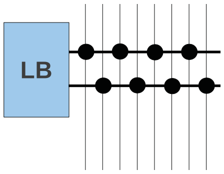


Fig. 10. CB pattern with $n = 1$.

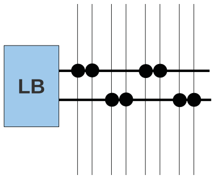


Fig. 11. CB pattern with $n = 2$.

To constrain the solution space, all wide edges going in the same direction have the same connectivity pattern. To generate a connectivity pattern for an SB, we randomly generate connectivity for six wide edges (one for each direction). Additionally, SBs that are very unlikely to lead to SAT solutions are not explored. For example, to have a reasonable chance of leading to a SAT solution, at least one switch in a wide edge must connect to each constituent wire segment to which it is adjacent. Another way of avoiding the exploration of UNSAT SBs is to check against known UNSAT SBs before running experiments. If the switches in a potential SB are a subset of the switches in an SB known to be UNSAT, then it is guaranteed that the potential SB will also be UNSAT, so it can be left unexplored.

For all connectivity patterns, we assume full connectivity in the super edges within CB:[2] a choice which does not increase the overall number of switches in the CB, as shown in Figs. 10 and 11. An Fc value of 0.5 is used in these figures for simplicity. Our experiments, however, are run with $Fc_{\text{in}} = 0.2$ and $Fc_{\text{out}} = 0.1$.

A. Experimental Methodology

Fig. 12 shows the flow used to evaluate connectivity patterns. Before beginning this flow, we generate coarse routes for each benchmark using VPR routing, for each value of n that we explore. Pregenerating coarse routes saves experimentation time, since the first phase of routing does not have to be

[1] We do not split wire segments that span multiple FPGA tiles. This means that connectivity added to the straight connections of a given wide wire are only added at the end points.

[2] A pin connecting to a wide wire through a super edge has a switch to all of the wire's constituent wire segments.

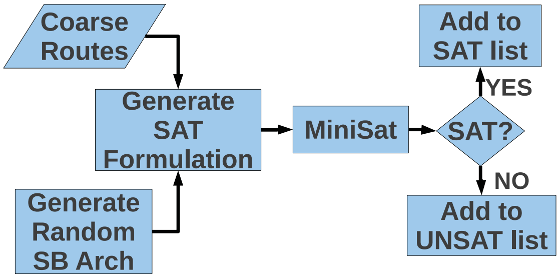


Fig. 12. Monte Carlo SB exploration flow.

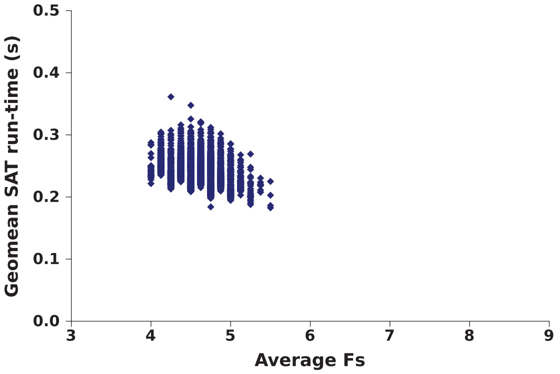


Fig. 13. Scatter plot showing the SAT runtime versus the area for all SAT SBs with $n = 2$.

performed for every SB connectivity pattern that we explore. The first step in the flow, after having generated coarse routes, is to generate a new SB architecture using our Monte Carlo SB generator. A SAT formulation is then generated using the coarse route of each benchmark and the SB architecture. The SAT formulation is then provided as input to MiniSat, which then produces either a legal routing or else an UNSAT result.

Because of the large number of SB connectivity patterns that we wanted to explore, we required a large amount of computation. We made extensive use of a large-scale cluster run by SciNet [25]. The SciNet general-purpose cluster is composed of 30 240 cores of Intel Xeon E5540, with 2 GB of RAM per core. In total, the results presented here represent approximately 167 000 core-hours (19.1 core-years) of computing time.

Figs. 13–16 show the tradeoff between area and SAT runtime for $n = 2$, $n = 3$, $n = 4$, and $n = 5$, respectively, for all SB connectivity patterns that we considered in our experiments. Each point in the graph represents the geomean routing runtime of 16 benchmarks routed using a distinct SB architecture. In these figures, we use average F_s as a proxy for area overhead because it is easy to calculate for a large number of architectures. We use actual area overhead for our final results. Of all connectivity patterns considered, we only show those that led to satisfactory assignments for *all* benchmarks. For these connectivity patterns, the SAT runtime varied, with richer connectivity generally leading to lower SAT time. The figures show that using the minimum area connectivity pattern tends to lead to higher SAT runtime, especially for $n = 4$ and $n = 5$. The figures show that it is possible to use a connectivity pattern with slightly more area to achieve the majority of the runtime benefit of a high-area connectivity pattern. The figures also show that, as n increases, the area necessary to have a reasonable SAT runtime also increases. As a point of illustration, Fig. 17 shows an SB pattern for $n = 3$, which offers a good tradeoff between area overhead and SAT runtime.

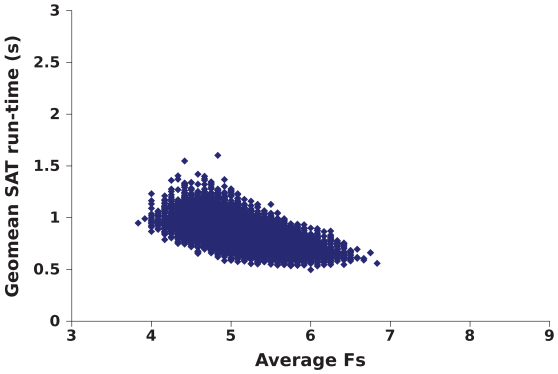


Fig. 14. Scatter plot showing the SAT runtime versus the area for all SAT SBs with $n = 3$.

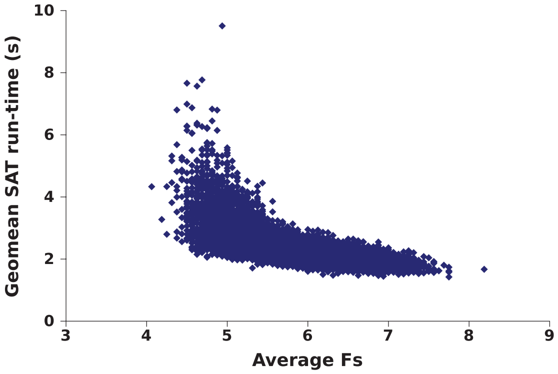


Fig. 15. Scatter plot showing the SAT runtime versus the area for all SAT SBs with $n = 4$.

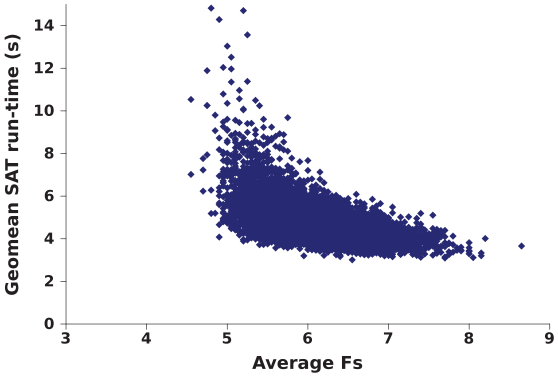


Fig. 16. Scatter plot showing the SAT runtime versus the area for all SAT SBs with $n = 5$.

B. SB Robustness Analysis

While all SBs included in Figs. 13–16 lead to SAT embeddings for the 16 benchmarks that we considered, they may

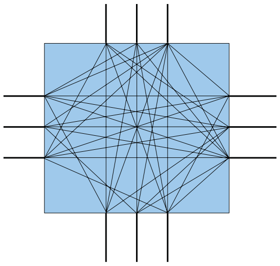


Fig. 17. Example connectivity pattern for $n = 3$.

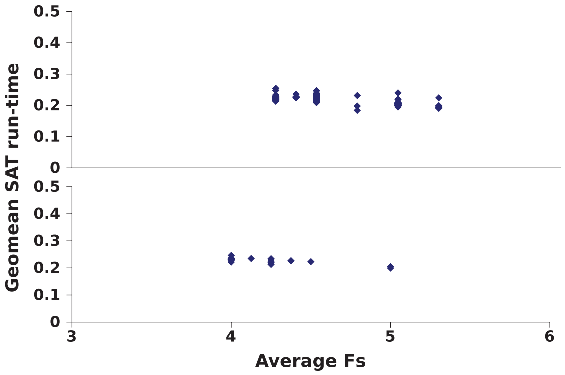


Fig. 18. Scatter plot showing the SAT runtime versus the area for robust and fragile SBs near the Pareto optimal curve with $n = 2$.

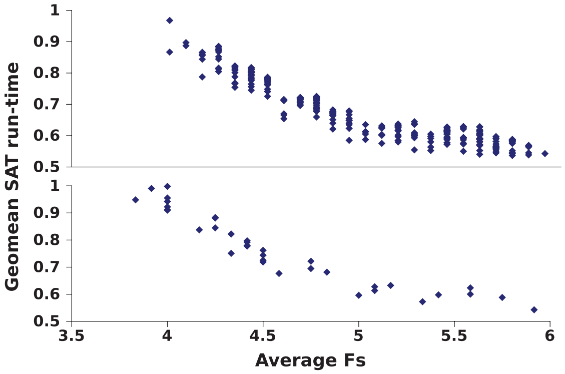


Fig. 19. Scatter plot showing the SAT runtime versus the area for robust and fragile SBs near the Pareto optimal curve with $n = 3$.

not necessarily do so for the universe of all designs. It is useful to determine which of the SBs are *robust* to changes. If a pattern is *robust*, it will lead to satisfactory assignments for a wide range of benchmarks. Otherwise, it is said to be *fragile*. In this section, we present the results of experiments that aim to determine the robustness of otherwise desirable SB patterns, meaning those that provide a good tradeoff between area and SAT runtime. In Figs. 13–16, this corresponds to the SBs near the Pareto optimal curve, where low runtime and low area overhead are both desirable properties. To generate a set of desirable SBs, we first determine the points in the scatter plots that are on the Pareto optimal curve as follows.

1) Add the lowest area point to the set of Pareto points.
2) Add the point to the Pareto set which results in the most negative slope from the lowest area point, where slope is δruntime$/\delta$area.

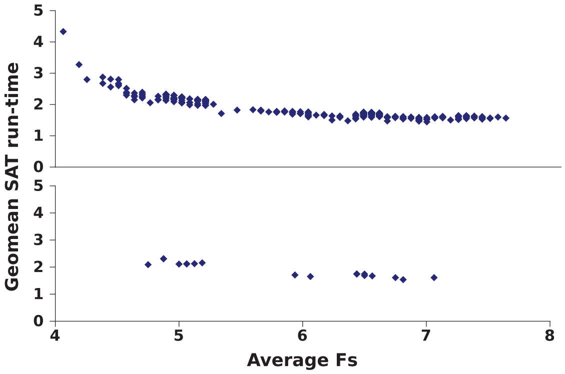


Fig. 20. Scatter plot showing the SAT runtime versus the area for robust and fragile SBs near the Pareto optimal curve with $n = 4$.

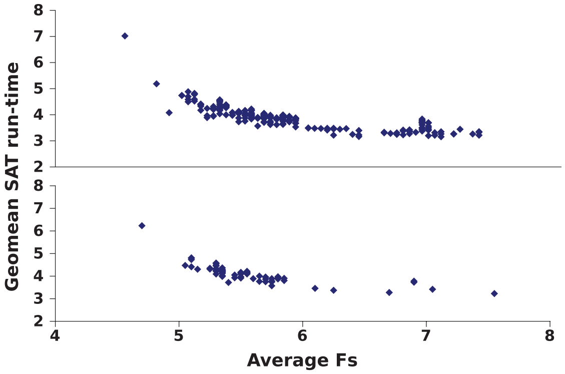


Fig. 21. Scatter plot showing the SAT runtime versus the area for robust and fragile SBs near the Pareto optimal curve with $n = 5$.

3) Repeatedly add the point to the Pareto set with the most negative slope from the most recently added point in the Pareto set until points with negative slopes no longer exist.

Next, a Pareto optimal curve is interpolated by drawing straight lines between points in the Pareto set. To increase the number of architectures in our robustness analysis, we then add points that have a runtime within 10% of an area-equivalent point on the Pareto optimal curve. For each of $n = 2$, $n = 3$, $n = 4$, and $n = 5$, this creates a set of approximately 320 SB patterns. For each of these patterns, we would ideally test robustness by routing a new set of benchmarks. However, because we do not have access to many large benchmarks, we generate three different placements of LBs for the same 16 benchmarks, leading to 48 new benchmark–placement combinations. SB patterns that lead to satisfactory assignments for each of the 48 new inputs (three placements for each of the 16 benchmarks) are considered robust, while those that do not are considered fragile.

Figs. 18–21 show the SB patterns on or near the Pareto optimal curve. The robust patterns are shown in the upper half of the figures, while the fragile ones are shown in the lower half. Table I shows the percentage of robust and fragile SB architectures in the Pareto optimal set for each value of n.

These results show that, on average, for $n = 2, 3, 4,$ and 5, 88% of SB architectures that show the best tradeoff

TABLE I
PERCENTAGE OF ROBUST AND SB ARCHITECTURES FOR $n = 2, 3, 4, 5$

n	Robust	Fragile	Percent UNSAT benchmarks
2	91.9%	8.1%	0.23%
3	88.8%	11.2%	0.45%
4	94.4%	5.6%	0.12%
5	78.4%	21.6%	0.45%
avg	88.4%	11.6%	0.31%

between area and runtime are completely robust, meaning that SAT is achieved for all benchmark–placement combinations. Additionally, for all SB architectures considered, across all benchmarks and placements, only 0.31% of benchmarks resulted in UNSAT, on average. For these 0.31% of runs, PathFinder could be rerun on an uncoarsened RR graph, without the embedding step, to get a legal routing solution.

V. PathFinder UNSAT Avoidance

In the previous experiments, no consideration was given to how the PathFinder routing algorithm itself could be altered to reduce the probability of an UNSAT condition occurring. In this section, we examine one method of doing so and discuss the associated results.

The conditions that cause UNSAT are complex and it is difficult to predict whether a coarse route will be SAT or UNSAT. Nevertheless, we have created a heuristic that aims to reduce the likelihood of UNSAT conditions. Consider two adjacent wide wires connected by a wide edge and a signal that uses both these wide wires, traversing the wide edge. During embedding, the SAT solver must find unoccupied wire segments for this signal to use within each of these wide wires. Intuitively, if these wide wires on either side of a wide edge have high occupancy, the SAT solver has less flexibility and has a greater chance of failing, potentially resulting in UNSAT. Wide edges that connect underutilized wide wires will have a smaller chance of causing UNSAT than edges that connect fully utilized wide wires.

We have created a cost associated with each wide edge that predicts how likely an UNSAT condition is at that edge based on *edge occupancy*. The cost is as follows:

$$\text{EdgeCost} = \alpha * \frac{\text{occ}_1 + \text{occ}_2}{\text{cap}_1 + \text{cap}_2} \tag{6}$$

where occ_1 is the occupancy of the RR node on one side of the edge, and cap_2 is the capacity of the RR node on the other side of the edge.

The cost is essentially the average utilization of the two wide wires connected by a wide edge. When routing a signal, we multiply the path cost from the source to the current RR node by this cost. Ideally, the EdgeCost would, on average, be equal to 1, so that it does not alter the average of all costs. Unfortunately, for a legal routing, the utilization fraction will, on average, be less than 1. To compensate for this, the utilization fraction is multiplied by correction factor (α), which we set to the empirically determined value of 1.5.

TABLE II
PERCENTAGE OF ROBUST AND FRAGILE SB ARCHITECTURES FOR $n = 3, 4,$ AND 5 USING EDGECOST

n	Robust	Fragile	Percent UNSAT benchmarks
2	89.3%	11.7%	0.65%
3	95.6%	4.4%	0.11%
4	97.5%	2.5%	0.06%
5	97.8%	2.2%	0.05%
avg	95.1%	4.9%	0.22%

Table II shows that SB robustness is greatly improved by incorporating EdgeCost into RR costs in PathFinder. Unfortunately, using EdgeCost also results in significantly longer routing runtimes. This is likely because, when using EdgeCost, instead of using a heuristic that simply attempts to make a legal coarse routing, PathFinder is now also attempting to spread signals out beyond the legality requirement. Extra effort, and hence runtime, is required to accomplish this goal. Because of the extra runtime penalty incurred when using EdgeCost in Pathfinder, we do not use it. Regardless, we believe it is still interesting to consider that it is possible to generate coarse routes that are more likely to result in SAT.

VI. Overall Results

Table III shows the total two-stage router runtime for each circuit. For each of $n = 2, 3, 4,$ and 5 the table shows results for the lowest area, the lowest runtime, and the lowest area/runtime product SB architectures from our experiments. The geometric mean of the runtimes across all benchmarks is shown near the bottom of the table. Also included in this table are these runtimes compared against three baseline flows. The first baseline, indicated as "versus standard" refers to the runtimes using our flow divided by the runtimes achieved using a standard VPR flow using a standard architecture with $W = 1.2 * W_{\text{min}}$. The next two baselines refer to area-neutral comparisons, which we will describe in Section VI-A. The results show that the best combined two-stage router runtime occurs when using $n = 3$ and when using the lowest runtime SB architecture (44% lower runtime than the standard flow). The lowest area/runtime product SB architecture also produces a very low runtime (43% lower). All area results were generated using VPR's area model by running unmodified VPR on the final routing results produced by our flow. The area results include new switches added to SBs. Details on the area models used by VPR can be found in [26].

Table IV shows the critical path delay values for our approach using the lowest area/runtime product SB architectures. These delay values were generated by reading the final routing solutions into VPR for timing analysis using a flattened RR graph. All reported critical path delay values account for the extra capacitive loading due to the additional switches in the grouped architecture. The results show that our two-stage routing approach has a negligible impact on critical path delay. In fact, on average, the critical path is slightly improved. Table V shows that the wire length is also not negatively affected by our approach. It improves slightly

TABLE III

COMBINED ROUTING AND SAT RUNTIMES IN SECONDS FOR RR GRAPH COARSENESS BETWEEN 1 AND 5. THE COLUMNS LABELED "AREA" CORRESPOND TO THE ROBUST SB ARCHITECTURE WITH THE LOWEST AREA FOR A PARTICULAR VALUE OF n. THE COLUMNS LABELED "rt" CORRESPOND TO THE ROBUST SB ARCHITECTURE WITH THE LOWEST SAT RUNTIME. THE COLUMNS LABELED "rt*AREA" CORRESPOND TO THE SB ARCHITECTURE WITH THE LOWEST RUNTIME/AREA PRODUCT

Benchmark	**Coarseness (*n*)**												
	1	**2**			**3**			**4**			**5**		
		Area	rt*area	rt	Area	rt*area	rt	Area	rt*area	rt	Area	rt*area	rt
cf_cordic_ v_18_18_18	2.757	2.37	2.34	2.37	1.84	1.73	1.73	3.08	2.92	2.70	3.36	2.91	2.91
cf_fir_24_16_16	4.41	2.24	2.24	2.26	2.39	2.32	2.31	2.61	2.48	2.31	3.84	3.64	3.67
clma	7.373	4.67	4.66	4.69	4.58	4.11	4.04	23.94	4.81	4.57	9.58	6.24	6.24
des_perf	2.129	1.57	1.57	1.57	1.45	1.30	1.28	2.54	2.24	2.07	3.21	2.74	2.70
ex1010	3.743	2.50	2.49	2.51	2.94	2.72	2.68	3.91	2.88	2.77	5.68	4.74	4.72
frisc	1.711	1.25	1.25	1.26	1.34	1.22	1.19	1.71	1.44	1.37	3.33	2.11	2.15
mac2	19.734	5.52	5.53	5.56	6.19	5.76	5.70	7.58	5.12	4.81	15.58	7.71	7.55
paj_raygentop	4.465	2.09	2.09	2.08	1.41	1.32	1.30	2.44	2.34	2.18	2.11	1.90	1.86
paj_top	36.299	156.29	156.21	156.44	22.93	21.63	21.15	112.71	36.36	32.22	129.53	69.72	65.78
pdc	6.537	3.60	3.60	3.61	3.56	3.08	3.02	79.03	3.93	3.77	202.75	4.53	4.68
rs_decoder_2	2.152	1.27	1.26	1.28	1.31	1.24	1.23	1.54	1.29	1.25	2.52	2.13	2.09
s38417	2.29	1.70	1.71	1.71	2.04	1.93	1.91	1.90	1.75	1.63	2.58	2.26	2.29
spla	2.648	2.00	2.00	2.01	1.87	1.70	1.68	12.64	2.12	2.06	153.84	3.08	3.04
sv_chip0	2.869	2.95	2.97	2.99	2.56	2.34	2.29	4.87	4.21	3.83	5.53	4.63	4.71
sv_chip1	55.604	33.69	33.67	33.76	32.20	31.83	31.70	28.23	25.69	24.96	28.54	24.16	25.07
sv_chip2	253.004	178.03	177.90	178.32	156.11	154.43	154.20	219.71	178.51	175.02	239.80	197.23	195.86
geomean	6.62	4.72	4.71	4.74	4.05	3.77	3.72	8.39	4.67	4.40	12.22	5.99	5.98
versus standard	1.00	0.71	0.71	0.72	0.61	0.57	0.56	1.27	0.71	0.67	1.85	0.91	0.90
versus higher W	1.00	0.94	0.94	0.95	0.84	0.78	0.77	1.86	1.04	0.98	2.81	1.38	1.37
versus flattened RR	1.00	0.74	0.74	0.74	0.59	0.55	0.54	1.29	0.72	0.68	1.60	0.78	0.78

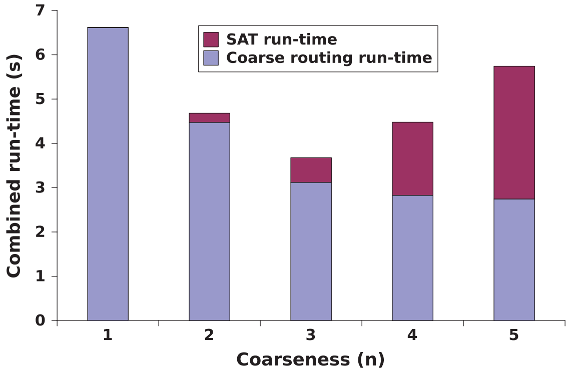


Fig. 22. Runtime versus coarseness (n).

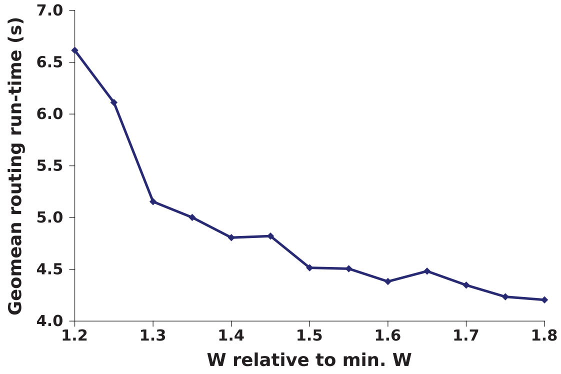


Fig. 23. Runtime versus channel width (W) relative to the minimum channel width (min. W) required to route each circuit.

because PathFinder has an easier routing problem, and so is able to use less circuitous routes.

Fig. 22 breaks down the runtime of the coarse routing and embedding stages using the best area/runtime product SB architecture for values of $n = 2, 3, 4$, and 5. Observe that $n = 3$ leads to the best overall runtime, and that, as expected, SAT runtime increases with n owing to larger and more complex SAT problem instances. We examine the scalability of SAT runtime with respect to circuit size later in this section.

A. Area Equivalent Comparisons

To ensure that the runtime improvements were not due solely to the extra flexibility provided by the changes that we made to the SB architecture, we ran two sets of experiments that examine the tradeoff between area and routing runtime. The goal of these experiments was to compare our routing flow and SB architecture against other routing flows that use the *same* amount of area. In the first set of experiments, we ran unmodified VPR routing on the grouped routing architectures that led to the best area–runtime tradeoff for $n = 2, 3, 4$, and 5. In other words, we did not use a coarsened RR graph. The second set of experiments compares our area–runtime tradeoff against a more conventional approach to trading off area for runtime, namely, varying the channel width (W). Increasing W leads to an easier routing problem, which generally leads to reduced runtime, as shown in Fig. 23. To compare against the runtime benefits provided by increasing W, we set the W

TABLE IV
CRITICAL PATH DELAY FOR COARSENESS BETWEEN 1 AND 5 USING THE SB ARCHITECTURES WITH THE LOWEST RUNTIME/AREA PRODUCT

	Coarseness (*n*)				
Benchmark	**1**	**2**	**3**	**4**	**5**
cf_cordic_ v_18_18_18	5.09E-09	5.09E-09	5.09E-09	5.09E-09	5.09E-09
cf_fir_24_16_16	1.21E-08	1.21E-08	1.22E-08	1.21E-08	1.22E-08
clma	9.94E-09	1.00E-08	9.94E-09	9.94E-09	9.94E-09
des_perf	4.95E-09	4.85E-09	4.85E-09	4.93E-09	4.85E-09
ex1010	7.91E-09	7.19E-09	7.27E-09	7.27E-09	7.43E-09
frisc	9.57E-09	9.14E-09	9.06E-09	9.14E-09	8.98E-09
mac2	2.55E-08	2.55E-08	2.55E-08	2.55E-08	2.55E-08
paj_raygentop	1.09E-08	1.08E-08	1.08E-08	1.08E-08	1.09E-08
paj_top	6.10E-08	6.08E-08	6.07E-08	6.06E-08	6.06E-08
pdc	7.96E-09	7.61E-09	6.97E-09	6.88E-09	6.96E-09
rs_decoder_2	1.88E-08	1.74E-08	1.76E-08	1.76E-08	1.74E-08
s38417	5.85E-09	5.85E-09	5.85E-09	5.85E-09	5.85E-09
spla	6.79E-09	6.71E-09	6.39E-09	6.39E-09	6.39E-09
sv_chip0	3.82E-09	3.74E-09	3.74E-09	3.74E-09	3.82E-09
sv_chip1	1.24E-08	1.24E-08	1.25E-08	1.25E-08	1.25E-08
sv_chip2	2.73E-08	2.75E-08	2.77E-08	2.74E-08	2.74E-08
Geomean	1.07E-08	1.05E-08	1.04E-08	1.04E-08	1.04E-08

TABLE V
WIRELENGTH FOR COARSENESS BETWEEN 1 AND 5 USING THE SB ARCHITECTURES WITH THE LOWEST RUNTIME/AREA PRODUCT

	Coarseness (*n*)				
Benchmark	**1**	**2**	**3**	**4**	**5**
cf_cordic_ v_18_18_18	40 038	40 306	38 781	39 520	38 910
cf_fir_24_16_16	35 202	33 497	33 691	33 688	34 233
clma	79 601	76 642	77 582	74 249	74 626
des_perf	48 145	47 347	46 451	48 024	46 773
ex1010	48 260	46 000	45 077	43 999	46 957
frisc	30 251	28 810	28 518	27 078	28 760
mac2	90 928	86 218	86 120	83 938	86 380
paj_raygentop	29 263	28 155	26 966	28 312	27 093
paj_top	589 494	624 688	596 748	582 497	573 596
pdc	61 146	56 515	54 235	54 392	53 418
rs_decoder_2	23 068	21 344	21 308	20 664	22 221
s38417	35 394	34 392	35 502	34 363	34 093
spla	37 400	35 825	34 928	33 981	35 605
sv_chip0	81 782	81 594	79 581	83 096	80 282
sv_chip1	159 519	156 842	157 193	155 750	154 005
sv_chip2	522 984	514 474	510 383	511 672	512 940
Geomean	66 810	64 787	63 873	63 394	63 887

value of an $n = 1$ standard routing architecture (i.e., with no additional SB flexibility) so that its area was equivalent to the area of a $W = 1.2 \cdot W_{\min}$ grouped architecture with values of n ranging from 2 to 5. This permitted us to evaluate router runtime on different interconnect architectures that occupy about the same area.

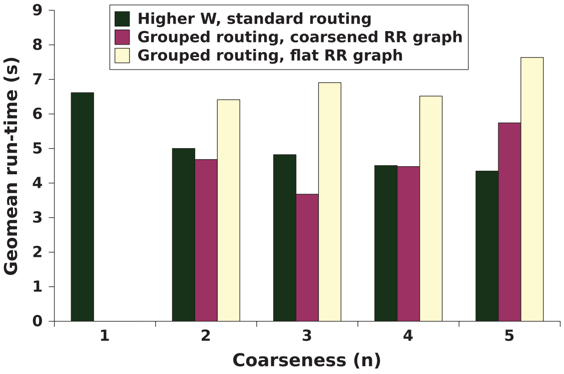


Fig. 24. Area-neutral runtime versus coarseness (n).

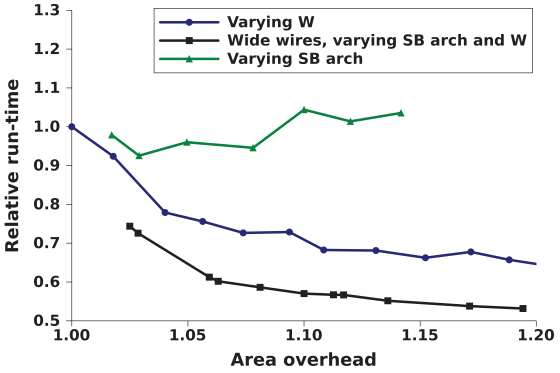


Fig. 25. Runtime–area tradeoff using three methods: varying channel width (W). Varying SB arch without using wide wires. Varying both W and SB arch using wide wires.

Figs. 24 and 25 show the results. Fig. 24 shows one area-neutral comparison for each of $n = 2, 3, 4$, and 5 between our flow and the two area-neutral flows described above. Our flow is labeled "grouped routing, coarsened RR graph," while the two area-neutral flows are labeled "higher W, standard routing," and "grouped routing, flat RR graph." For $n = 3$, the runtime of our approach, including SAT time, is 22% lower than the runtime that results from increasing the channel width until the area is equivalent. It is interesting that standard VPR PathFinder routing run on an uncoarsened routing graph cannot take advantage of the additional flexibility in the SB to reduce router runtime. In fact, the runtime increases as a result of the increased number of RR graph edges that result from more SB connectivity. Fig. 25 examines the tradeoff between area and routing runtime for each of the three approaches. To generate points that correspond to a variety of area overheads, we ran our flow with $n = 3$ and $W = 1.1 * W_{\min}, 1.2 * W_{\min}, 1.3 * W_{\min}$. This figure shows that our approach, labeled "Wide wires, varying SB arch and W," offers the best tradeoff between area and runtime. Compared to a standard VPR flow using W = 1.2 (labeled "varying W"), we can get a 26% reduction in routing runtime for 2% area overhead, a 40% reduction in routing runtime for 6% area overhead, and a 45% reduction in routing runtime for 14% area overhead. We conclude that our combined routing/architecture approach is a more effective way in tackling runtime than simply increasing W, though the two methods are somewhat orthogonal. This figure also shows that increasing SB flexibility without using our two-step routing approach (labeled "Varying SB arch") leads to a poor area–runtime tradeoff.

TABLE VI

SAT SCALABILITY. NUMBER OF VARIABLES, NUMBER OF CLAUSES, AND SAT RUNTIMES VERSUS COARSENESS OF THE RR GRAPH

	RR_nodes	**Number of Variables**				**Number of Clauses**				**Run-time (s)**			
Benchmark		$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
cf_cordic_ v_18_18_18	77 337	52 624	91 006	70 632	110 254	77 028	154 175	268 144	392 440	0.16	0.40	1.07	1.52
cf_fir_24_16_16	58 965	41 836	58 296	74 498	91 551	56 782	121 371	209 167	311 514	0.12	0.33	0.79	1.20
clma	127 419	92 256	131 735	164 754	201 773	136 090	294 654	472 497	676 525	0.28	0.91	2.09	2.79
des_perf	88 822	58 958	80 975	107 252	127 609	83 608	169 210	30 941	425 005	0.17	0.47	1.27	1.64
ex1010	70 644	58 415	80 292	100 629	129 812	88 136	187 258	299 732	491 126	0.18	0.54	1.23	1.94
frisc	46 998	35 411	50 039	62 273	79 169	53 404	115 932	185 015	292 348	0.11	0.34	0.72	1.13
mac2	142 480	102 235	143 038	180 681	228 840	143 964	307 279	488 687	778 793	0.30	0.94	1.95	3.26
paj_raygentop	61 180	35 965	48 938	64 751	76 207	52 920	110 379	201 843	276 242	0.10	0.30	0.75	1.01
paj_top	1 075 857	65 348	90 717	117 526	143 009	99 734	206 613	349 768	502 022	0.21	0.66	1.53	2.00
pdc	81 460	28 733	39 403	49 775	63 533	41 396	89 975	150 549	242 573	0.08	0.25	0.52	0.84
rs_decoder_2	42 042	44 616	63 314	78 718	95 278	62 836	139 389	219 898	320 957	0.12	0.38	0.80	1.14
s38417	71 937	43 818	61 039	76 204	98 031	67 386	141 971	231 382	377 111	0.14	0.42	0.94	1.38
spla	61 269	102 481	141 537	186 953	223 174	139 510	268 180	473 842	654677	0.30	0.74	1.77	2.52
sv_chip0	173 482	192 865	270 597	350 607	427 808	263 312	506 019	865 990	1 251 348	0.58	1.51	3.64	5.36
sv_chip1	288 747	703 775	987 007	1 288 041	1 554 515	950 630	1 849 323	3 205 286	4 462 850	2.20	5.68	12.89	19.88
sv_chip2	1 092 041	603 968	851 422	1 101 928	1 353 648	816 836	1 530 141	2 610 245	3 715 664	1.90	4.74	11.63	17.12
geomean	117 538	79 989	111 233	142 427	175 285	114 639	236 693	398 714	590 309	0.24	0.68	1.59	2.33

For completeness, Table VI shows the number of variables in the SAT formulation, number of clauses in the SAT formulation, and SAT runtimes for values of n from 2 to 5. The number of flattened RR graph nodes is also provided to show how SAT scales with circuit size. The ratio between SAT runtime and the number of RR nodes does not increase with larger circuits, indicating that SAT-based embedding should work well for larger industrial benchmarks. This is especially true for $n = 2$ because there are no clauses with more than two variables. Solving this type of problem, called 2-SAT, can be done in linear time [27].

VII. CONCLUSION

In this paper, we showed that CAD runtime can be reduced by considering FPGA architecture and algorithms together. We described a two-stage routing approach that first routes to a coarsened representation of the FPGA architecture, and then uses SAT to convert the coarse routing solution to a legal detailed routing solution. We also presented a grouped FPGA routing architecture that supports our new routing approach. We explored a total of approximately 165 000 SB architectures, generated using a Monte Carlo methodology, highlighting an interesting tradeoff between area and embedding runtime. Additionally, we examined the robustness of these architectures to changes in placements and benchmarks. Using the combined architecture and algorithm, we could decrease the router runtime by 40% at the cost of 6% area overhead, with no impact on circuit critical path delay.

We also presented a method of making the PathFinder algorithm aware of the likeliness of SAT or UNSAT resulting from a coarse route. Unfortunately, changing the costs to promote coarse routes that are more likely to result in SAT increases router runtime substantially. The methods could, however, be used in a limited capacity to modify coarse routes in order to resolve UNSAT conditions in a feedback loop with the SAT solver. This is an interesting avenue of future research, as it would avoid having to do a full rerouting in the rare case of an UNSAT coarse routing.

ACKNOWLEDGMENT

The authors would like to thank H. Bian for her helpful comments on this paper, and H. Mangassarian for useful discussions on the satisfiability formulation.

REFERENCES

[1] J. Swartz, V. Betz, and J. Rose, “A fast routability-driven router for FPGAs,” in *Proc. ACM/SIGDA FPGA*, 1998, pp. 140–149.

[2] Y. Sankar and J. Rose, “Trading quality for compile time: Ultrafast placement for FPGAs,” in *Proc. ACM/SIGDA FPGA*, 1999, pp. 157–166.

[3] R. Tessier, “Fast placement approaches for FPGAs,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 7, no. 2, pp. 284–305, 2002.

[4] P. Maidee, C. Ababei, and K. Bazargan, “Fast timing-driven partitioning-based placement for island style FPGAs,” in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 2003, pp. 598–603.

[5] M. Gort and J. H. Anderson, “Accelerating FPGA routing through parallelization and engineering enhancements,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 1, pp. 61–74, Jan. 2012.

[6] M. Gort and J. H. Anderson, “Deterministic multi-core parallel routing for FPGAs,” in *Proc. IEEE Field-Program. Technol.*, Dec. 2010, pp. 78–86.

[7] V. Betz, A. Ludwin, and K. Padalia, “High-quality, determinstic parallel placement for FPGAs on commodity hardware,” in *Proc. ACM/SIGDA FPGA*, 2008, pp. 14–23.

[8] C. C. Wang and G. G. Lemieux, “Scalable and deterministic timing-driven parallel placement for FPGAs,” in *Proc. ACM/SIGDA FPGA*, 2011, pp. 153–162.

[9] L. McMurchie and C. Ebeling, “PathFinder: A negotiation-based performance-driven router for FPGAs,” in *Proc. ACM/SIGDA FPGA*, 1995, pp. 111–117.

[10] N. Eén and N. Sörensson. (2011). *MiniSAT* [Online]. Available: http://minisat.se

[11] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, M. Fang, and J. Rose, “VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling,” in *Proc. ACM/SIGDA FPGA*, 2009, pp. 133–142.

[12] M. Gort and J. H. Anderson, “Reducing FPGA router run-time through algorithm and architecture,” in *Proc. IEEE Field Program. Logic Appl.*, Sep. 2011, pp. 336–342.

[13] S. Gupta, J. Anderson, L. Farragher, and Q. Wang, “CAD techniques for power optimization in virtex-5 FPGAs,” in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2007, pp. 85–88.

[14] W. C. R. Fung and V. Betz, “Simultaneous short-path and long-path timing optimization for FPGAs,” in *Proc. Int. Conf. Comput.-Aided Design*, 2004, pp. 838–845.

[15] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider, and L. Albertson, "Plasma: An FPGA for million gate systems," in *Proc. ACM/SIGDA FPGA*, 1996, pp. 10–16.

[16] R. Lysecky, F. Vahid, and S. Tan, "Dynamic FPGA routing for just-in-time FPGA compilation," in *Proc. ACM/IEEE Design Autom. Conf.*, Jul. 2004, pp. 954–959.

[17] R. Lysecky, F. Vahid, and S. X. D. Tan, "A study of the scalability of on-chip routing for just-in-time FPGA compilation," in *Proc. 13th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2005, pp. 57–62.

[18] S. Y. Chin and S. J. Wilton, "An analytical model relating FPGA architecture and place and route runtime," in *Proc. IEEE Field Program. Logic Appl.*, Sep. 2009, pp. 146–153.

[19] S. Y. Chin and S. J. Wilton, "Toward scalable FPGA CAD through architecture," in *Proc. ACM/SIGDA FPGA*, 2011, pp. 143–152.

[20] R. Li, D. Zhou, and D. Du, "Satisfiability and integer programming as complementary tools," in *Proc. Asia South Pacific Design Autom. Conf.*, 2004, pp. 879–882.

[21] Berkeley Logic Synthesis and Verification Group. (2011). *ABC: A System for Sequential Synthesis and Verification*, Berkeley, CA [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[22] A. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *Proc. ACM/SIGDA FPGA*, 1999, pp. 37–46.

[23] G.-J. Nam, K. A. Sakallah, and R. A. Rutenbar, "Satisfiability-based layout revisited: Detailed routing of complex FPGAs via search-based Boolean SAT," in *Proc. ACM/SIGDA FPGA*, 1999, pp. 167–175.

[24] S. J. Wilton, "Architectures and algorithms for field-programmable gate arrays with em-bedded memories," Ph.D. dissertation, Dept. Electron. Commun. Eng., Univ. Toronto, Toronto, ON, Canada, 1997.

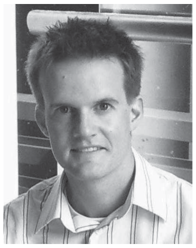
[25] C. Loken, D. Gruner, L. Groer, R. Peltier, N. Bunn, M. Craig, T. Henriques, J. Dempsey, C. H. Yu, J. Chen, L. J. Dursi, J. Chong, S. Northrup, J. Pinto, N. Knecht, and R. V. Zon, "SciNet: Lessons learned from building a power-efficient top-20 system and data centre," *J. Phys., Conf. Ser.*, vol. 256, no. 1, p. 012026, 2010.

[26] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Boston, MA: Kluwer, 1999.

[27] B. Aspvall, M. F. Plass, and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas," *J. Inf. Process. Lett.*, vol. 8, no. 3, pp. 121–123, 1979.

Marcel Gort received the B.A.Sc. degree in computer engineering from the University of Western Ontario, London, ON, Canada, in 2007, and the M.A.Sc. degree in computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2009. He is currently pursuing the Ph.D. degree with the University of Toronto, Toronto, ON.

He was with the IBM Toronto Software Laboratory, working in the compiler group. His current research interests include fast and scalable computer-aided design algorithms for FPGAs as well as FPGA architectures amenable to these algorithms.



Jason H. Anderson (S'96–M'05) received the B.Sc. degree in computer engineering from the University of Manitoba, Winnipeg, MB, Canada, and the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto (U of T), Toronto, ON, Canada.

He joined the FPGA Implementation Tools Group, Xilinx, Inc., San Jose, CA, working on placement, routing, and synthesis, in 1997. From 2005 to 2008, he managed various groups with Xilinx, focusing on strategic research and development projects. He became a Principal Engineer with Xilinx in 2007. He joined the ECE Department, U of T in 2008, where he is currently an Assistant Professor. He has authored numerous papers published in refereed conference proceedings and journals, and holds 20 issued U.S. patents. His current research interests include computer-aided design and architecture for FPGAs.

Dr. Anderson was a recipient of the Ross Freeman Award for Technical Innovation, the highest innovation award from Xilinx, for his contributions to the Xilinx placer technology in 2000. He received four awards for excellence in undergraduate teaching from 2009 to 2012. He serves on the technical program committees of various conferences, including the ACM International Symposium on Field Programmable Gate Arrays and the IEEE International Conference on Field Programmable Technology.