Reducing FPGA Router Run-Time Through Algorithm and Architecture

Marcel Gort and Jason H. Anderson
Dept. of Electrical and Computer Engineering, University of Toronto
Toronto, Ontario, Canada
{gortmarc|janders}@eecg.utoronto.ca

***Abstract*—We propose a new FPGA routing approach that, when combined with a low-cost architecture change, results in a 34% reduction in router run-time, at the cost of a 3% area overhead, with no increase in critical path delay. Our approach begins with traditional 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 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. Our approach points to a new research direction: reducing FPGA CAD run-time by exploring FPGA architectures and algorithms together.**

I. Introduction

As field-programable gate arrays (FPGA) become larger, the run-time required to execute the associated CAD tools is worsening. It can now take days to compile the largest industrial FPGA designs from HDL-to-bitstream. In the past, worsening CAD tool run-time was mitigated by increases in uniprocessor clock speed. However, this is no longer the case. 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 run-times 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.

Prior efforts to reduce FPGA CAD run-times have focused on algorithmic changes [1], [2], [3], [4] or parallelization [5], [6], [7]. In this work, we reduce router run-time 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 run-time is not well-studied. We propose a new two-stage FPGA routing algorithm that takes advantage of increased flexibility in the FPGA switch block, resulting in reduced router run-time.

At a high-level, our routing approach works as follows: first, a PathFinder-style router 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 [8]) to solve it. The combination of this new routing approach with a modified routing architecture leads to router run-time reductions of up to 34% compared to standard PathFinder routing using VPR, with only 3% area overhead, and no increase to critical path delay.

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. We also introduce a new *grouped* routing architecture that allows embedding to find legal signal-to-wire segment assignments. In this paper, we make several contributions:

- A new two-stage routing approach that combines PathFinder with a SAT-based embedding.
- A new grouped FPGA routing architecture.
- A preliminary architecture exploration for the grouped routing architecture. We demonstrate architectures that, when combined with our new router, result in a 34% improvement in router run-time.

We believe that our combined routing algorithm/architecture approach represents an entirely new direction for reducing tool run-time – one with fertile ground for further research. The remainder of this paper is organized as follows: Section II provides relevant background on FPGA routing algorithms and architecture, SAT, and previous work related to run-time-driven FPGA architecture design. The proposed two-stage routing approach is described in Section III. In Section IV, we describe our grouped routing architecture. An experimental study appears in Section V. Conclusions and suggestions for future work are offered in Section VI.

II. Background

A. FPGA Routing

Fig. 1 depicts a basic island-style FPGA. Logic blocks (LB) connect to wire segments through programmable switches in connection blocks (CB). Wire segments connect to other

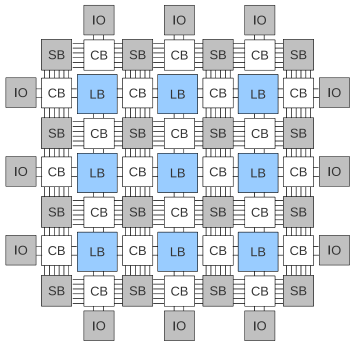

Fig. 1: FPGA architecture.

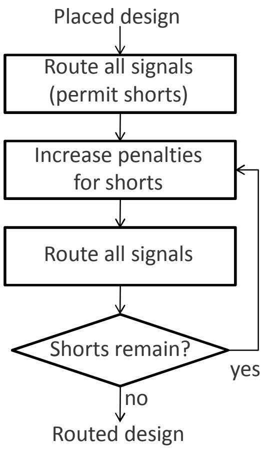
Placed design
Route all signals (permit shorts)
Increase penalties for shorts
Route all signals
Shorts remain?
yes
no
Routed design

Fig. 2: Negotiated congestion routing flow.

wire segments using programmable switches in switch blocks (SB). Connections between LBs are made by turning on the appropriate switches in SBs and CBs.

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 [10], [11]. PathFinder is also used in the publicly-available VPR FPGA placement and routing framework [12], which we modified and used for this work. Fig. 2 gives an overview of the negotiated congestion approach used in 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 signals may use the *same* wire). Then, the penalties associated with shorts are increased, and the signals are re-routed, avoiding shorts where possible. The process of increasing the penalties for shorts and re-routing the signals continues iteratively until all shorts are removed and the routing is feasible.

VPR PathFinder uses an 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.

B. Architectures for Fast CAD

A limited amount of prior work has explored the interaction of FPGA architecture and CAD run-time. In [13], the PLASMA FPGA architecture was proposed, which was intended as a prototyping platform. The entire routing process takes three seconds due to the very flexible routing fabric, which leads to a much simplified routing problem. The cost of having fast routing is dramatically higher silicon area. In [14], Lysecky et al. use a simplified planar FPGA routing architecture to reduce the number of possible routing paths. The authors split 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 re-route signals with shorts. A $3\times$ improvement in router run-time is reported compared to VPR, at the cost of 30% increase in critical path delay [15]. No comparison is made with a standard (non-planar) routing architecture. In [16], Chin et al. explore the interaction between routing architecture parameters and place and route run-time. They show that varying LB capacity significantly impacts router run-time by reducing the overall problem size. In [17], the same authors explore how changes to logic block architecture can affect CAD run-time.

C. Boolean Satisfiability

A SAT problem is a Boolean formula typically expressed in conjunctive normal form (CNF), which essentially is product-of-sums (POS) 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 satisfiable (SAT); otherwise, it is unsatisfiable (UNSAT). Several efficient academic SAT solvers exist, the most popular of which is MiniSat [8], which we use in this work (ver. 2.2.0).

III. A New Two-Stage Routing Approach

A. A Motivating Experiment

For all experiments in this paper, we target an FPGA with single-driver wire segments that span 2 LB tiles, though our approach can be adapted to work with mixed wire segment lengths. The 16 benchmark circuits with the longest router run-time 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 [12]. Circuits were mapped into 4-input LUTs using ABC [18], then clustered using T-Vpack [19] into LBs with 10 4-LUTs and 22 inputs. 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 PathFinder spends a significant amount of time attempting to legalize *almost legal* routing solutions. Fig. 3 shows the geometric mean run-time of PathFinder (across all circuits) as a function of the maximum number signals shorted on any single wire segment. The figure shows that approximately 40% of run-time is spent resolving the congestion of a routing solution where no RR node is over-utilized by more than 1 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, then using a SAT-based approach to generate a fully legal solution, providing an overall run-time 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 only be grouped with other wire segments of the same length, 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. 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 run-time 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 run-time of PathFinder across all benchmarks. *W* is set to 1.2× 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 run-time decreases when n is increased, with a 50% run-time reduction observed with $n = 5$ compared to $n = 1$ (without considering the embedding run-time).

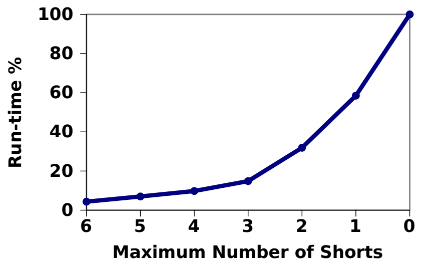

Fig. 3: % of router run-time vs. maximum RR node shorts.

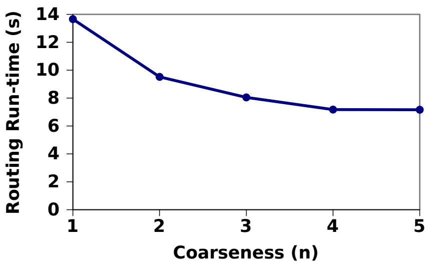

Fig. 4: Geometric mean router run-time vs. coarseness.

C. Embedding Stage

The output of the coarsened routing stage is not a complete routing solution, since signals are not assigned to individual wire segments. A de-coarsening, or embedding stage, must be performed 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 1-to-1 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 re-run on an uncoarsened RR graph.

We provide an example of a routing with no legal embedding in Figs. 5 and 6. Fig. 5 shows routes through a SB for four signals ($n1$,$n2$,$n3$,$n4$). In this figure, solid lines indicate wide wires with $n = 2$. Each is labelled 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 wide wires. We test this hypothesis in Section IV.

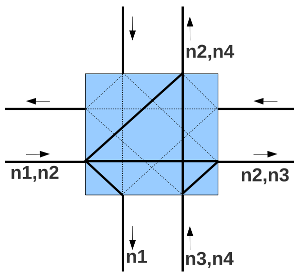

Fig. 5: Coarse routing solution for four signals: n1,n2,n3,n4.

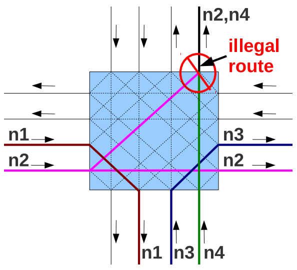

Fig. 6: Best embedding solution based on the coarse routing in Fig. 5.

1) SAT formulation: We formulate the embedding problem as a Boolean satisfiability problem in CNF form. Our formulation is inspired by the SAT-based detailed routing approach presented in [20], 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-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 on 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 $3A$ are 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 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, where variables are named such that variable $1A$ represents signal 1 being on wire segment A. The following clauses are generated to ensure that only legal connections are formed: $(\overline{1A} \vee \overline{1D}) \wedge (\overline{1B} \vee \overline{1C})$.

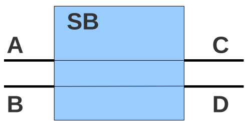

Fig. 7: Flattened representation of 1-to-1 connections between two super nodes

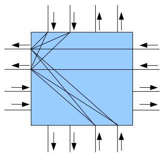
Fig. 8: Connectivity pattern (flattened) guaranteed to lead to SAT assignment.

The main differences between our SAT-based embedding approach and the detailed routing approach presented by Nam et. al [20] are as follows: 1) Our liveness constraints through SBs ensure that illegal connections are not made, rather than ensuring that legal connections are made. This results in fewer SAT clauses when there are many possible legal connections; 2) we support multi-terminal signals; and 3) our formulation supports any arbitrary connectivity patterns between wide wires.

IV. Routing Architecture

In this section, we describe a grouped routing architecture with connection patterns that match those required at each stage of our routing approach. Connections between wide wires in a SB follow the well-known Wilton connectivity pattern [21]. Each wide wire end point entering a switch block 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 are available. Increasing the amount of connectivity incurs a higher area overhead, but also makes finding a satisfying assignment more likely, as well as faster. We explore only a fraction of all potential connection patterns since the search space is very large. We allow for different connectivity patterns between wide wires making connections in each of the 6 directions through a 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*, or $1.4 * 10^{45}$ connectivity patterns in total for n ranging from 2 to 5. A more thorough exploration of this architectural space is a fertile area for future research.

We explored a broad range of low-level connectivity patterns. The search space was pruned in two ways. First, we eliminated patterns that would almost certainly lead to UNSAT. At a minimum, all connectivity patterns have 1-to-1

[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.

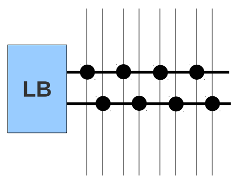

Fig. 9: Connection block pattern with $n = 1$.

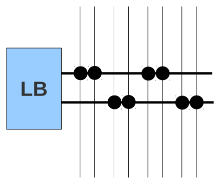

Fig. 10: Connection block pattern with $n = 2$.

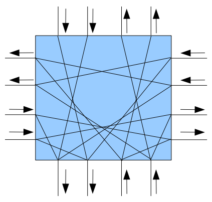

Fig. 12: Connectivity pattern for $n = 2$.

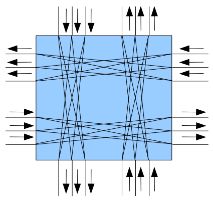

Fig. 13: Connectivity pattern for $n = 3$.

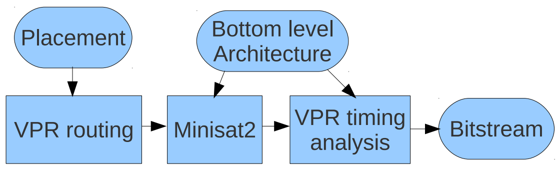

Fig. 11: Routing flow using PathFinder and SAT-based embedding.

connections between wire segments in connected wide wires, as in Fig. 7. Second, we have identified a connectivity pattern that *guarantees* SAT without requiring full connectivity, so we do not explore patterns with more than this amount of connectivity. Our guaranteed pattern has full connectivity in all turning directions and 1-to-1 connectivity for all straight connections. Fig. 8 shows part of this connectivity pattern for $n = 2$. Any constraints imposed by the lack of connectivity in the East↔West super edge can be overcome using the flexibility in the other (turning) super edges.

We evaluated nearly 2000 patterns with more connectivity than the infeasible sparse case, but less than a full connectivity pattern. Starting with the minimum 1-to-1 connectivity patterns, we generated new connectivity patterns by adding switches in multiples of n, in order to limit the search space. For all connectivity patterns, we assumed full connectivity in the super edges within *connection blocks*[2]: a choice which does not increase the overall number of switches in the connection block, as shown in Figs. 9 and 10. An F_c value of 0.5 is used in these figures for simplicity. Our experiments, however, are run with $F_{c\_in} = 0.2$ and $F_{c\_out} = 0.1$. Fig. 11 shows the routing flow used to evaluate connectivity patterns. We first generate a placement using standard VPR 5.0. We then perform routing using modified VPR PathFinder (timing-driven) on the coarsened RR graph. Our tool then creates a SAT formulation using the coarse routing solution and the low-level connectivity patterns, for which MiniSat attempts to find a satisfying assignment. Using this assignment, our tool generates a routing solution in VPR's routing output format. We then modified VPR to be able to accept a complete routing solution as input and perform only timing analysis, and verify the legality of all routes.

Of all connectivity patterns considered, we only used those that led to satisfying assignments for *all* benchmarks. For these connectivity patterns, the SAT run-time varied, with richer connectivity leading to lower SAT time. For each value of n, we chose a connectivity pattern that offered a good trade-off between area overhead and SAT run-time. Unfortunately, a comprehensive description of all patterns considered could not be included due to page limits[3], however, Figs. 12 and 13 show the connectivity patterns that we chose for $n = 2$ and $n = 3$, respectively. We omit the 1-to-1 connections in the figures for clarity. We found that in general, extra connectivity was most helpful when it was added to the East↔West and North↔South connections. When investigating the reason for this, we discovered that VPR's PathFinder routing prefers making straight connections over turning connections by a factor of about 5:1. Since horizontal and vertical connections are more often utilized, adding connectivity here helps the most. It is important to note than this property may change as the high level routing architecture changes. For example, using longer wires may lead PathFinder to use fewer straight connections. We leave the exploration of different high level routing architectures as future work.

Interestingly, we also tried using PathFinder for our embedding stage, with routes contrained to those routes 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.

V. Results

Table I shows the total two-stage router run-time for each circuit. Fig. 14 shows the geometric mean router run-time, area, and critical path delay normalized to the standard VPR flow run on an uncoarsened RR graph. The critical path delay values for our approach were generated by reading the final routing solution 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

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

[3]A paper describing the detailed architectural study is in preparation.

TABLE I: Combined routing and SAT run-times in seconds for RR graph coarseness between 1 and 5.

	Coarseness (n)				
Benchmark	**1**	**2**	**3**	**4**	**5**
cf_cordic_v_18_18_18	4.27	3.36	3.14	4.87	4.65
cf_fir_24_16_16	13.42	8.51	6.11	6.26	5.64
clma	10.89	7.29	7.05	7.07	7.61
des_perf	3.62	3.09	2.72	3.70	3.85
ex1010	7.55	5.17	4.58	4.56	5.87
frisc	4.03	3.73	2.82	3.30	4.74
mac2	20.03	16.39	13.04	12.40	16.41
paj_raygentop	10.62	5.24	4.66	4.83	5.28
paj_top	86.04	70.13	66.68	67.57	59.50
pdc	13.31	6.30	6.02	4.31	4.41
rs_decoder_2	5.57	4.67	3.76	3.52	3.85
s38417	2.87	2.47	2.93	3.23	3.48
spla	7.87	4.79	4.94	5.08	6.39
sv_chip0	7.14	5.22	5.61	8.07	9.41
sv_chip1	136.67	96.81	93.58	74.72	75.33
sv_chip2	716.29	609.45	552.90	393.19	420.59
Geomean	13.66	9.86	8.99	9.25	10.14
Run-time reduction		27.79%	34.15%	32.29%	25.79%

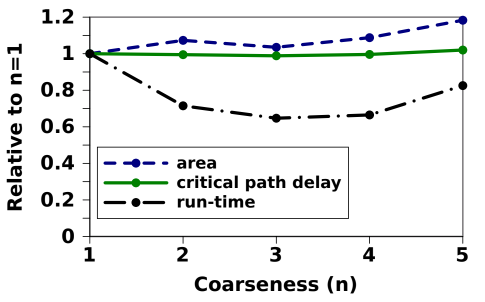

Fig. 14: Run-time, area, and crit. path delay vs. (n).

the grouped architecture. The area values are expressed in the number of minimum width transistors per FPGA tile, as reported by VPR. The results show that our two-stage routing approach has a negligible impact on critical path delay. For $n = 3$, we observe a 34.2% improvement in run-time at the cost of a 3% area overhead incurred due to increased flexibility in the SB. For the coarseness values from $n = 2$ to $n = 5$, Fig. 15 breaks down the run-time of the coarse routing and embedding stages. Observe that $n = 3$ leads to the best overall run-time, and that as expected, SAT run-time increases with n owing to larger and more complex SAT problem instances. We examine the scalability of SAT run-time with respect to circuit size later in this section.

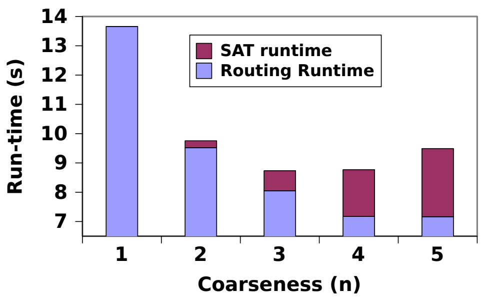

Fig. 15: Run-time break-down vs. coarseness (n).

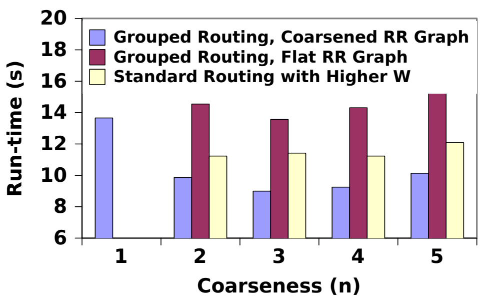

Fig. 16: Run-time comparisons with area-equivalent FPGA architecture/CAD flows for n from 2 to 5.

To ensure that the run-time improvements were not due solely to the proposed changes in routing architecture, we ran standard VPR PathFinder routing on the grouped architectures with n from 2 to 5, without coarsening the RR graph. We then compared the run-time against our two-stage approach. We also compared our area/run-time trade-off against a more conventional approach to trading off area for run-time, namely, varying the channel width (W). Increasing W leads to an easier routing problem, which generally leads to reduced run-time. To compare against the run-time benefits provided by increasing W, we set the W 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 run-time on different interconnect architectures that consume about the same area.

The results of both experiments are shown in Fig. 16. The "Grouped Routing, Coarsened RR Graph" bars refer to our two-stage routing approach run on the grouped FPGA routing architecture. The "Grouped Routing, Flat RR Graph" bars refer to the standard PathFinder routing algorithm run on the same grouped FPGA routing architecture (uncoarsened). The "Standard Routing with Higher W" bars refer to the standard PathFinder routing algorithm being run on a standard routing architecture with W chosen so that its area is equivalent to our grouped architecture.

The results show that standard VPR PathFinder routing run on an uncoarsened routing graph cannot take advantage of the additional flexibility in the SB to reduce router run-time. In fact, run-time increases due to the increased number of RR graph edges that result from more SB connectivity. The results also show that we offer a better trade-off between area and run-time compared to simply increasing the channel width, W. For $n = 3$, the run-time of our approach, including SAT time, is 21% lower than the run-time that results from increasing the channel width until the area is equivalent. We conclude that our combined routing/architecture approach is a more effective way at tackling run-time than simply increasing W. For completeness, Table II shows the number of variables in the SAT formulation, number of clauses in the SAT formulation, and SAT run-times for values of n from 2 to 5. The number of flattened RR graph nodes is also provided to show how

TABLE II: SAT scalability: Number of variables, number of clauses, and SAT run-times vs 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	77337	52624	70632	91006	110254	77028	154175	268144	392440	0.16	0.40	1.07	1.52
cf_fir_24_16_16	58965	41836	58296	74498	91551	56782	121371	209167	311514	0.12	0.33	0.79	1.20
clma	127419	92256	131735	164754	201773	136090	294654	472497	676525	0.28	0.91	2.09	2.79
des_perf	88822	58958	80975	107252	127609	83608	169210	305941	425005	0.17	0.47	1.27	1.64
ex1010	70644	58415	80292	100629	129812	88136	187258	299732	491126	0.18	0.54	1.23	1.94
frisc	46998	35411	50039	62273	79169	53404	115932	185015	292348	0.11	0.34	0.72	1.13
mac2	142480	102235	143038	180681	228840	143964	307279	488687	778793	0.30	0.94	1.95	3.26
paj_raygentop	61180	35965	48938	64751	76207	52920	110379	201843	276242	0.10	0.30	0.75	1.01
paj_top	1075857	65348	90717	117526	143009	99734	206613	349768	502022	0.21	0.66	1.53	2.00
pdc	81460	28733	39403	49775	63533	41396	89975	150549	242573	0.08	0.25	0.52	0.84
rs_decoder_2	42042	44616	63314	78718	95278	62836	139389	219898	320957	0.12	0.38	0.80	1.14
s38417	71937	43818	61039	76204	98031	67386	141971	231382	377111	0.14	0.42	0.94	1.38
spla	61269	102481	141537	186953	223174	139510	268180	473842	654677	0.30	0.74	1.77	2.52
sv_chip0	173482	192865	270597	350607	427808	263312	506019	865990	1251348	0.58	1.51	3.64	5.36
sv_chip1	288747	703775	987007	1288041	1554515	950630	1849323	3205286	4462850	2.20	5.68	12.89	19.88
sv_chip2	1092041	603968	851422	1101928	1353648	816836	1530141	2610245	3715664	1.90	4.74	11.63	17.12
geomean	117538	79989	111233	142427	175285	114639	236693	398714	590309	0.24	0.68	1.59	2.33

SAT scales with circuit size. The ratios between SAT run-time 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 2 variables. Solving this type of problem, called 2-SAT, can be done in linear time [22].

VI. Conclusions and Future Work

In this work, we showed that CAD run-time 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, 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. Using the combined architecture and algorithm, we decrease router run-time by 34% at the cost of 3% area overhead, with no impact on circuit critical path delay.

As future work, we suggest performing a more detailed exploration of low level routing architecture, specifically examining whether some SB connection patterns more often lead to satisfying assignments in the embedding stage. We also plan on exploring how changes to the high level routing architecture (e.g. different mixes of wire segment lengths) affect the choice of low level routing architecture. Additionally, in our work, we have not made changes to the PathFinder algorithm; yet, it is possible that we could avoid coarse routes that lead to unsatisfiable embedding problems by cost function changes in PathFinder.

Acknowledgments

The authors thank Hannah Bian for her helpful comments on the manuscript, and Hratch Mangassarian for useful discussions on the SAT formulation.

References

[1] J. Swartz, V. Betz, and J. Rose, "A fast routability-driven router for FPGAs," in *ACM/SIGDA FPGA*, 1998.

[2] Y. Sankar and J. Rose, "Trading quality for compile time: ultra-fast placement for FPGAs," in *ACM/SIGDA FPGA*, 1999.

[3] R. Tessier, "Fast placement approaches for FPGAs," *ACM TODAES*, vol. 7, no. 2, 2002.

[4] P. Maidee, C. Ababei, and K. Bazargan, "Fast timing-driven partitioning-based placement for island style FPGAs," in *ACM/IEEE DAC*, 2003.

[5] M. Gort and J. H. Anderson, "multi-core deterministic parallel routing for FPGAs," in *FPT*, 2010.

[6] V. Betz, A. Ludwin and K. Padalia, "High-quality, determinstic parallel placement for FPGAs on commodity hardware," in *ACM/SIGDA FPGA*, 2008.

[7] C. C. Wang and G. G. Lemieux, "Scalable and deterministic timing-driven parallel placement for FPGAs," in *ACM/SIGDA FPGA*, 2011.

[8] N. Eén and N. Sörensson, "Minisat," 2011, `http://minisat.se`.

[9] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *ACM/SIGDA FPGA*, 1995.

[10] S. Gupta, J. Anderson, L. Farragher, and Q. Wang, "CAD techniques for power optimization in Virtex-5 FPGAs," in *CICC*, 2007.

[11] W. C. R. Fung, V. Betz, "Simultaneous short-path and long-path timing optimization for FPGAs," in *ICCAD*, 2004.

[12] 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 *ACM/SIGDA FPGA*, 2009.

[13] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider, and L. Albertson, "Plasma: an FPGA for million gate systems," in *ACM/SIGDA FPGA*, 1996.

[14] R. Lysecky, F. Vahid, and S. Tan, "Dynamic FPGA routing for just-in-time FPGA compilation," in *ACM/IEEE DAC*, 2004.

[15] 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 *IEEE FCCM*, 2005.

[16] S. Y. Chin and S. J. Wilton, "An analytical model relating FPGA architecture and place and route runtime," in *FPL*, 2009.

[17] ——, "Towards scalable FPGA CAD through architecture," in *ACM/SIGDA FPGA*, 2011.

[18] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification," Release 70930, `http://www.eecs.berkeley.edu/~alanmi/abc/`.

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

[20] 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 *ACM/SIGDA FPGA*, 1999.

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

[22] B. Aspvall, M. F. Plass, and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas," 1978, CS Dept., Stanford University.