Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder

Raphael Rubin
Computer and Information Systems
University of Pennsylvania
3330 Walnut Street
Philadelphia, PA 19104
rafi@seas.upenn.edu

André DeHon
Electrical and Systems Engineering
University of Pennsylvania
200 S. 33rd Street
Philadelphia, PA 19104
andre@acm.org

ABSTRACT

We show that, with the VPR implementation of Pathfinder, perturbations of initial conditions may cause critical paths to vary over ranges of 17–110%. We further show that it is not uncommon for VPR/Pathfinder to settle for solutions that are >33% slower than necessary. These results suggest there is room for additional innovation and improvement in FPGA routing. As one step in this direction, we show how delay-targeted routing can reduce delay noise to 13% for our worst-case design and below 1% for most designs. Anyone who uses VPR as part of architecture or CAD research should be aware of this noise phenomena and the techniques available to reduce its impact.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aides—*placement and routing*; B.6.1 [**Logic Design**]: Design Styles—*logic arrays*

General Terms

Algorithms, Measurement

Keywords

Timing-Driven Routing, VPR, Pathfinder, Noise, Sensivitity

1. INTRODUCTION

While exploring techniques to tolerate variation [9], using VPR [1, 2], our efforts were severely hampered by results that contradicted intuition. As we added controls to check for and prevent systematic errors, it became increasingly apparent that the results were not self consistent; they were erratic and *noisy*. After seeking advice and tuning VPR, we devised a simple test to asses the stability of the delay results from the router: adding tiny variations to the resource graph several orders of magnitude smaller than the nominal delays. The result—a critical path delay spread greater than 50% of the average—was so shocking and initially unbelievable that we devised several more approaches to evaluate the sensitivity of the router. We also tried randomly breaking ties during routing and randomly shuffling the order of nets in the netlist file (this does not effect the structure, but does alter the routing order). For each of these tests, one would expect to see little or no change in the critical path delay across the samples; however, in every case not only did we get spreads in the 20–100% range, the distributions were similar to the initial approach.

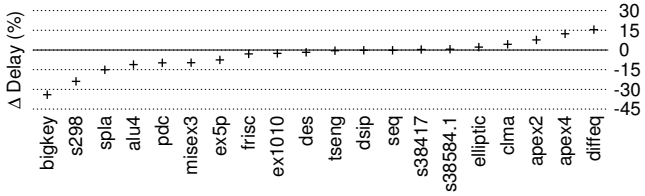


Figure 1: Percent delay improvement for faster-wire architecture over uniform architecture for the Toronto 20 benchmarks

We hope to raise awareness of the current state of the published and publicly available versions of the Pathfinder [6] algorithm. VPR is the defacto standard router for academic FPGA architecture and CAD research, and anyone who uses it should be aware of noise inherent in its operation as well as steps one can take to reduce the noise. We review and quantify the effects of tuning with the pressure factor multiplier to reduce delay noise (Section 4) and present a small modifications to VPR/Pathfinder to target a particular delay that can further reduce delay noise (Section 5). These results suggest that, while VPR/Pathfinder is very good, FPGA routing is not a solved problem and, deeper analysis and refinement of the Pathfinder algorithm is well warranted.

2. TRY THIS AT HOME

Our observations may be surprising. We encourage readers to see these effects for themselves. Input files, external tools (*e.g.* our netlist shuffler), scripts, and patches to VPR 4.3 [1] and 5.0.2 [7] to reproduce most of the techniques and results presented in this paper are available at: `http://www.seas.upenn.edu/~icgroup/publications/pf_fpga_2011`

We start with a simple example comparing two architectures. We use the architecture `k4-n4.xml` from the VPR 5.0.2 distribution where all wires have the same character-

FPGA'11, February 27–March 1, 2011, Monterey, California, USA.

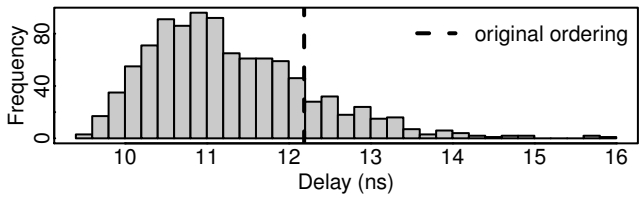


Figure 2: Distribution of Delay from VPR when routing 1000 Netlist Shuffles of `alu4` at Channel Width 28

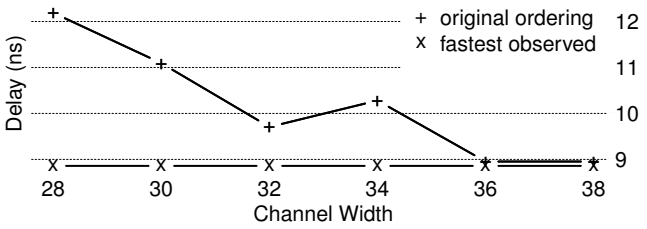


Figure 3: VPR routed and feasible delays as a function of channel width for `alu4`. Maximum iterations is set to 1000 to ensure success at all widths routed.

istics for the first; in the second architecture, half the wires are slightly faster such that if all bottleneck signals utilize the faster resources we scale the critical path delay by 0.995. We route each netlist on both architectures with the same placement and channel width target, determined by running VPR with defaults and the `-verify_binary_search` option, to get the results in Figure 1. This graph is surprising for two reasons: (1) the second architecture is slower than the first on one quarter of the designs, (2) while the improvement should only have a 0.5% impact on delay, we see delay changes from -34% to +15%. This magnitude of effect cannot be a result of the architectural change compared.

Next, we keep the architecture (`k4-n4`), placement, and channel width the same but alter the ordering of the CLBs within the input netlist file. This *netlist shuffling* transformation does not alter the structure of the problem, only the order in which VPR routes signals. Figure 2 shows the results of routing with 1000 unique permutations of `alu4` along with the routed delay from the original netlist ordering. This shows that VPR/Pathfinder is highly sensitive to the order in which nets are routed, with some net orderings being 67% worse than others. The result VPR produces with the original ordering is 29% worse than the best result in this range.

A common architectural parameter to vary during experiments is the channel width. Channel width may affect delay when there are insufficient resources to allocate least delay paths for all nets, forcing some nets to take non-minimal paths to avoid congestion. Adding tracks to each channel allows more critical signals to take shorter paths, at least until the critical signals all utilize the least delay routes. However, it is not uncommon for VPR to produce graphs like Figure 3 where the delay descent is not monotonic in channel width. We define the feasible delay, $T_{feasible}$, as the smallest delay found by any of our routing attempts; this delay is known to be achievable and is an upper bound on the true minimum delay, T_{true}, for the routing task. Note that the feasible delay line is flat, suggesting that the decrease in delay is **not** because the architecture cannot be routed at minimum delay for low channel widths, but that the router is doing an increasingly poor job of finding low delay paths.

3. IMPLICATIONS

These results suggest that the critical path delay that the VPR/Pathfinder router will find for any netlist, placement, and architecture includes a large, random component in addition to the true delay, T_{true}, the architecture supports.

$$T_{vpr\_rt}(\text{net,place,arch}) = T_{true}(\text{net,place,arch}) + Noise \quad (1)$$

We observe $Noise$ that ranges from 0–67% with a median of 17% (Figure 2) compared to $T_{feasible}$, our upper bound estimate on T_{true} ($T_{feasible} \geq T_{true}$).

Graphs like Figure 1 are common ways of showing the varying effects of an architecture or CAD optimization across a benchmark set. Prior to seeing results like Figure 2, we thought these graphs were primarily capturing the fact that some netlists could benefit more from the optimization than others. It is now clear that at least part of the difference in benefits among the netlists is this random $Noise$ component.

Anyone using VPR/Pathfinder in the CAD flow to analyze architectures or pre-routing CAD optimizations should be aware of this noise level. Changes that have an impact below this noise level will be difficult to quantify reliably and will demand more work than simply reporting the mean speedup on 20 benchmarks. Larger changes can be measured, but the precision of the result is limited by the noise effects.

These results suggest both a need to better understand Pathfinder behavior to be able to interpret its results and a need to continue to improve our routing techniques to reduce this noise effect.

4. PRESSURE FACTOR MULTIPLIER

Lemieux previously reported that the VPR router would produce results across a large delay range and demonstrated how tuning `pres_fac_mult` (pfm) can shrink that range in a trade off with runtime [5]. pfm controls the rate at which the Pathfinder edge cost function shifts priority to congestion minimization instead of delay minimization. The edge cost for net $i \rightarrow j$ to use node n is:

$$C(n,i,j) = \alpha_{ij} * d_n + (1 - \alpha_{ij}) * (b_n + h_{n,t}) * p_n \quad (2)$$

$$p_n = 1 + \max(0, (1 + occupancy - capacity))\, pf_t$$

$$pf_t = pf_1 \times (pfm)^{t-1} \quad (3)$$

$$\alpha_{ij} = \min\left(\frac{T_{ij}}{T_{crit}}, max\_crit\right) \quad (4)$$

Where d_n is the delay of the node, b_n is a base cost for using the node, $h_{n,t}$ is the congestion history term for the node, T_{ij} is the longest unregistered path containing the 2-point net $i \rightarrow j$, pf_1 is the initial pressure factor, and pfm is the pressure factor multiplier. To quantify this effect, we compare the normal VPR router run with two different values pfm values: 1.3 (the default for VPR 5.0.2) and 1.1. For each of these cases, we routed 1000 different permutations of each of the Toronto 20 benchmarks [3]. A single placement and channel width, is used for all routing trials. The results in Table 1 confirm that reducing pfm does improve delay and shrink the range of values observed. However, the noise is still significant ($\geq$25%) and uncontrolled for most designs (14 of 20).

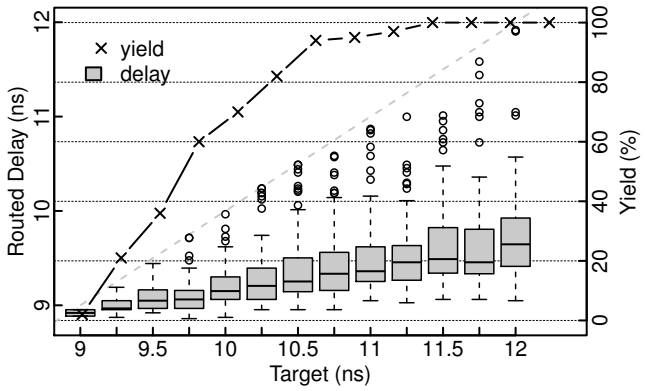


Figure 4: Routed Delay versus Specified Delay Target for 100 Netlist Shuffles of Delay-Targeted Routing of `alu4` at Channel Width 28

5. DELAY TARGETED ROUTING

So [8] demonstrated that routing with a delay target and pruning paths that did not meet that target could often achieve the congestion-oblivious lower bound route—the delay achievable when all nets are allowed to take the minimum delay path ignoring congestion. So used a sophisticated scheme for slack allocation. We exploit the same basic idea of delay targets but use a simpler strategy that trivially integrates with VPR/Pathfinder and works with any target bound. Rather than allow the critical path delay in VPR to float, we target a fixed delay goal, T_{target}. This is achieved by replacing the critical delay (T_{crit}) in the VPR/Pathfinder criticality computations (Equation 4) by T_{target}, giving us:

$$\alpha_{ij} = \min\left(\frac{T_{ij}}{T_{target}}, max\_crit\right) \quad (5)$$

VPR's imposed upper bound (max_crit) on allowed criticality prevents the criticality term from exceeding 1.0 on super critical paths. The VPR 5.0.2 default for max_crit is 0.99. We also add delay target satisfaction to the termination conditions of the main Pathfinder loop. This approach avoids the arduous task of slack budgeting and, like Pathfinder's congestion negotiation, permits potentially valuable super critical intermediate states.

Figure 4 shows that delay-targeted routing can consistently achieve the requested target, T_{target}, when the target is slighly above $T_{feasible}$. When the target is not close to the $T_{feasible}$, it will often return routes faster than requested.

We can perform a single route for a particular T_{target} in time comparable to a timing-driven VPR route at a fixed channel width, whereas So's technique takes an order of magnitude more time per target.

A single delay target does not address the delay minimization problem; for that we use the search in Algorithm 1. In the initial route, congestion is disabled, allowing each path to takes its delay minimizing path, thus computing the congestion-oblivious lower bound on delay. We determine an upper bound by exponentially increasing our target until one succeeds—the same technique used by VPR when searching for a minimum channel width. The reduction is a binary search with two modifications. Upon a successful route trial we pivot on the resulting delay instead of the input target since Figure 4 showed that the delay-target search often find

Algorithm 1: Delay Target Search

```
T_current=CongestionObliviousRoute
max = min = T_current        /* Initial lower bound */
repeat                       /* Find initial upper bound */
    max *= 2
until try_route(max)
stage=0
repeat                                          /* Refine */
    retry = 0
    stage++
    success=false
    repeat
        T_target = (max + min)/2
        if ((T_current = try_route(T_target))!=FAIL) then
            T_target+ = (max − T_target)/1000
            success=true
    until retry++ >= retries or success
    if success then
        max = T_current
    else
        min = T_target
until max <= min * (1 + target precision)
      or stage >= max_stages
```

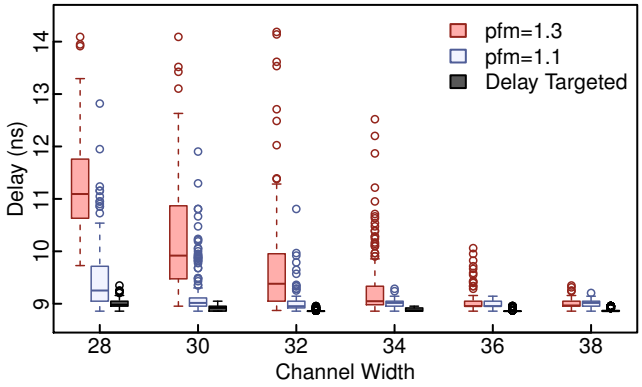


Figure 5: Spread of Routed Delays vs. Channel width for *pfm* Tuning and Delay-Targeted Routing for 200 netlist shuffles of `alu4`

delays well below its target. In practice this provides a significant acceleration in the reduction of the search window. Secondly: we are still limited by the heuristic nature of the router and cannot trust a negative result (Figure 5). To compensate we slightly shift the target to perturb the route and sample again from the noise distribution.

One consequence of using the delay target in Equation 5 is that T_{target} also limits criticality when T_{target} is even slightly larger than T_{true}. This allows delay-targeted routing to run with larger max_crit values that do not necessarily converge in default VPR.

To test the efficacy of this technique, we performed the same netlist shuffle delay experiment we performed for pfm (Section 4) for the delay-targeted router (with $pfm = 1.1$ and $max\_crit = 0.999$) and included the results in Table 1. Delay targeted-routing, though not perfect, reduces the noise to a tolerable level for all but a few designs (four designs with ranges around 10%).

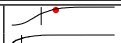
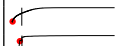
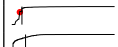
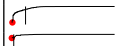
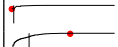
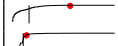
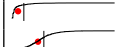
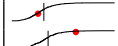
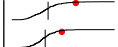
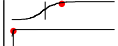
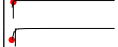
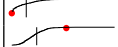
Design	W	VPR 5.0.2 $pfm = 1.3$ CDF	med.	%	$pfm = 1.1$ med.	%	Delay Targeted med.	%
alu4	28		11.0	67	9.3	45	9.0	7.3
apex2	34		12.0	65	12.0	32	12.0	0.0
apex4	36		12.0	79	9.7	84	9.3	13.0
bigkey	20		6.0	74	5.6	58	5.6	0.9
clma	42		28.0	57	28.0	11	28.0	0.1
des	20		12.0	40	12.0	5	12.0	3.2
diffeq	22		8.0	110	7.4	59	7.4	0.0
dsip	20		7.3	30	7.3	6	7.3	0.0
elliptic	34		15.0	59	14.0	32	14.0	0.0
ex1010	36		20.0	17	20.0	3	20.0	1.9
ex5p	36		11.0	66	10.0	25	10.0	0.0
frisc	38		15.0	58	15.0	28	15.0	0.0
misex3	30		12.0	83	9.5	72	9.1	10.0
pdc	50		20.0	60	17.0	63	17.0	9.3
s298	22		19.0	88	15.0	50	14.0	4.0
s38417	30		16.0	39	16.0	2	16.0	0.0
s38584.1	28		12.0	38	12.0	2	11.0	0.0
seq	34		12.0	63	11.0	31	11.0	0.0
spla	44		16.0	61	14.0	43	14.0	2.0
tseng	20		7.5	72	7.4	39	7.3	0.9

Table 1: Comparison of Routed Delay Range for VPR pfm Tuning and Delay-Targeted Routing. Each result is based on 1000 netlist shuffles. W is the target channel width. The dots mark the result found by VPR (original ordering). Vertical lines in Cumulative Distribution Function (CDF) column mark the median.

Figure 5 revisits the channel width noise in Figure 3. To provide perspective, the boxplots represent the distributions for delay-targeted routing and unmodified VPR routing at each channel width. By including the distributions, we expose the non-monotonicity of the curve not as a meaningful effect, but as set of random points selected from the *Noise* distribution at each channel width. The distributions better fit our expectations, the range shrinks and shifts downward with each additional pair of tracks. The delay-targeted results are also a distribution, just one that is much tighter than the *pfm*=1.1 case. Even at 28 tracks per channel the range is just 7.3%, showing that delay-targeted routing significantly reduces the anomalies we saw in Figure 3.

We see the improvements in routing quality offered by our delay-targeted routing not as a final solution to the quality and noise problems raised but more as further demonstration of the opportunity to improve Pathfinder-based routing. Future work is needed to better characterize or guarantee quality and noise bounds and to reduce the additional runtime required to achieve high quality.

6. RELATED WORK

Yan [10] explored the sensitivity of architecture results to CAD tools. The changes that result from different pre-routing tool flows (logic mapping, placement) also present different tasks into routing, but Yan only presents a few distinct cases into routing for each benchmark and makes no attempt to differentiate the impact of *systematic* improvement from better placements or mappings from the *random* impacts of routing. In that sense, our experiments more cleanly characterize just the sensitivity of routing to changes that should **not** change the results. Nonetheless, the highest level intent is the same—to alert users to the potential pitfalls and suggest approaches to mitigate these effects.

Our work is also related to the known optimums and lower bound work from Cong (*e.g.* [4]). We demonstrate that our most popular tools (VPR/Pathfinder) are not finding the best solutions and quantify how far from optimum they are. As with Cong's work, this highlights the need for future work in algorithms and implementations to close the gap.

Acknowledgments Benjamin Gojman helped create the tables and graphs. This research was funded in part by National Science Foundation grant CCF-0904577. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] V. Betz. VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs. <http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html>, March 27 1999. Version 4.30.

[2] V. Betz and J. Rose. VPR: A new packing, placement, and routing tool for FPGA research. In W. Luk, P. Y. K. Cheung, and M. Glesner, editors, *Proceedings of the International Conference on Field-Programmable Logic and Applications*, number 1304 in LNCS, pages 213–222. Springer, August 1997.

[3] V. Betz and J. Rose. FPGA Place-and-Route Challenge. <http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html>, 1999.

[4] J. Cong and K. Minkovich. Optimality study of logic synthesis for LUT-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):230–239, February 2007.

[5] G. Lemieux and D. Lewis. Using sparse crossbars within LUT clusters. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 59–68, 2001.

[6] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 111–117, 1995.

[7] J. Rose et al. VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs. <http://www.eecg.utoronto.ca/vpr/>, 2008.

[8] K. So. Enforcing long-path timing closure for FPGA routing with path searches on clamped lexicographic spirals. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 24–33, 2008.

[9] J. S. J. Wong, P. Sedcole, and P. Y. K. Cheung. Self-measurement of combinatorial circuit delays in FPGAs. *Transactions on Reconfigurable Technology and Systems*, 2(2):1–22, 2009.

[10] A. Yan, R. Cheng, and S. J. E. Wilton. On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 147–156, 2002.