Enhancing Timing-Driven FPGA Placement for Pipelined Netlists

Ken Eguro Scott Hauck

Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA

{eguro, hauck}@ee.washington.edu

ABSTRACT

FPGA application developers often attempt to use pipelining, C-slowing and retiming to improve the performance of their designs. Unfortunately, such registered netlists present a fundamentally different problem to the CAD tools, limiting the benefit of these techniques. In this paper we discuss some of the inherent quality and runtime issues pipelined netlists present to existing timing-driven placement approaches. We then present some algorithmic modifications that reduce post-compilation critical path delay by an average of 40%.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids – *Placement and routing.*

General Terms

Algorithms, Design.

Keywords

Reconfigurable logic, FPGA, placement, simulated annealing, timing-driven, pipelined.

1. INTRODUCTION

Although the widespread popularity of FPGAs is a testament to their unique blend of flexibility and ease of use, this adaptability generally comes at a price. As discussed in [3], circuits implemented on an FPGA can be expected to run three to four times slower than their ASIC counterparts. To minimize the effect of this intrinsic performance penalty, specialized timing-driven placement and routing techniques have been developed for reconfigurable devices [5] [6]. Although applied to completely different phases of the mapping process, the most successful timing-aware placement and routing algorithms share a general approach to maximizing the achievable clock rate: they first identify the portions of the circuit that might be timing-critical, then give these sensitive sections special priority over less critical portions of the netlist during compilation.

Although these timing-driven CAD tools perform well given classical netlists, to further improve performance FPGA application developers often apply a variety of pipelining, C-slowing and retiming techniques to their circuits. However, these heavily registered netlists may not always reach their maximum potential. As we will discuss in this paper, registered netlists have some fundamentally different characteristics that can limit the efficacy of existing timing-driven placement approaches.

2. Classical Timing-Driven Placement

VPR [5] is one of the most popular academic FPGA place and route tool suites. As the de facto standard, it has served as both a building platform and comparison target for countless other research efforts. VPR includes T-VPlace, a simulated annealing based timing-driven placement algorithm. T-VPlace considers both a net's wirelength and delay contribution during placement to achieve a good balance between overall netlist routability and critical path delay. During simulated annealing, it calculates the cost of a move using Eq. 1.

$$\Delta C = \lambda * \frac{\Delta Timing\_Cost}{Previous\_Timing\_Cost} + (1-\lambda) * \frac{\Delta Wiring\_Cost}{Previous\_Wiring\_Cost} \qquad (1)$$

In this way, VPR can emphasize maximum routability ($\lambda = 0.0$), minimum critical path delay ($\lambda = 1.0$) or, most likely, strike a balance between the two. While the *Wiring_Cost* is essentially just a summation of all nets' bounding boxes, calculating the *Timing_Cost* is a bit more complex.

Before placement on a given architecture is started, VPR builds a table that estimates the shortest path delay from each logic block and I/O pad in the array to every other logic block and I/O pad. VPR uses this table throughout the annealing process to determine the source/sink delay of each connection in the netlist.

Calculating the timing cost of the current placement begins by performing a static timing analysis on the initial random placement. This gives us both D_{max}, the overall maximum critical path delay of the current placement, and $Slack(i, j)$, the amount of delay we could add to the connection between source i and sink j without increasing the critical path delay.

As shown in Eq. 2, we can calculate the relative criticality of each link in the netlist based upon this information. Using Eq. 3, VPR then weights the impact of the delay between each source-sink pair based upon its criticality. That is, delay along a path that has lots of timing slack is relatively cheap, while delay anywhere along the critical path is expensive. Finally, Eq. 4 shows that the overall placement timing cost is calculated as the summation of the timing cost of each source/sink pair.

$$Criticality(i,j) = 1 - \frac{Slack(i,j)}{D_{\max}} \qquad (2)$$

$$Timing\_Cost(i,j) = Delay(i,j) * Criticality(i,j)^{Crit\_Exp} \qquad (3)$$

$$Timing\_Cost = \sum Timing\_Cost(i,j) \qquad (4)$$

Under review for *DAC 2008*, June 9–13, 2008, Anaheim, CA,USA.

3. Implications

Although VPR's T-VPlace formulation showed a dramatic critical path delay improvement as compared to a purely congestion-driven placement tool in [5], relatively little is known about the absolute performance of the algorithm. In this section we will discuss some potential shortcomings of the T-VPlace approach that can eventually lead to instabilities in the simulated annealing placement itself.

3.1 Criticality Accuracy & Runtime

If we focus on Eq. 2 and 3, we can see that VPR's timing cost function is based upon the source/sink criticalities calculated during static timing analysis. Unfortunately, static timing analysis is far too computationally expensive to perform after each annealing move. Thus, VPR's default settings only perform a single timing analysis at the beginning of each temperature iteration, and then use these criticalities to calculate the quality of subsequent moves. This means that VPR performs less than a few hundred timing analyses, instead of potentially several million.

If we revisit VPR's basic cost function, we can capture this optimization formally. We can see in Eq. 5 that VPR calculates the criticality of each source/sink pair (*i*, *j*) at the beginning of temperature iteration *k*. For any given placement within a temperature iteration, we use Eq. 6 to calculate the timing cost. This is simply the delay of the source/sink pair (*i*, *j*) at temperature iteration *k*, move number *l*, multiplied by the criticality of the link as calculated at the beginning of the temperature iteration. This makes the incremental timing cost simply the change in delay between move (*l*-1) and move *l*, multiplied by the criticality of the link at the beginning of the temperature iteration.

$$Criticality(i,j,k) = 1 - \frac{Slack(i,j,k)}{D_{\max}(k)} \tag{5}$$

$$Timing\_Cost(i,j,k,l) = Delay(i,j,k,l) * Criticality(i,j,k)^{Crit\_Exp} \tag{6}$$

$$\Delta TC(i,j,k,l) = \left[Delay(i,j,k,l) - Delay(i,j,k,l-1)\right] * Criticality(i,j,k)^{Crit\_Exp} \tag{7}$$

Unfortunately, while this optimization does make placement several orders of magnitude faster, since we do not update the criticality nor critical path delay within a temperature iteration, we slowly get less and less accurate timing information. This can lead to less than optimal final results.

Consider the example shown in Figure 1. At the beginning of the annealing we calculate the critical path delay. We then use this value to calculate the slack and criticality of each source/sink pair. However, as we move blocks around, a gap forms between the real criticalities of the current placement and the values we use to calculate the timing cost. Since a single temperature iteration might attempt tens or hundreds of thousands of moves, the optimizations we attempt towards the end of a temperature iteration can extremely inaccurate.

In Figure 1 we believe that we will make the system better if we move block *a* to reduce the delay on the critical path (*a*, *c*). However, this move only accomplishes this by adding delay to the previously non-critical path (*a*, *b*). Although this actually increases the circuit's real critical path delay, the placement tool is unaware this is a poor choice.

Assuming for the moment that we are willing to ignore algorithm runtime, we can demonstrate the advantages of more up-to-date criticality information. In Figure 2 we show two placement runs of the benchmark *ex5p* on the single 4-LUT, single FF *4lut_sanitized* architecture. Indicated with squares is the wire cost and critical path delay recalculated at the end of each temperature iteration when we perform one static timing analysis (STA) per temperature iteration. Indicated with triangles are the results when we perform 1000 static timing analyses per temperature. In the case of *ex5p*, this equates to roughly one static timing analysis for every 100 attempted simulated annealing moves.

As we can see, while the wire costs for both placement runs are smoothly declining, the critical path delay for the placement performed with the default settings fluctuates considerably. This is particularly concerning since this oscillation persists even as we near the end of the annealing process. This is likely due to the fact that, with stale criticality information, the placement tool does not realize when it is increasing the critical path delay of the system. On the other hand, the placement performed with frequent static timing analysis shows a much more stably decreasing critical path delay.

In Table 1 we show results for static timing analysis testing for the full suite of VPR netlists, 22 of the largest combinational and sequential MCNC benchmarks. We report normalized geometric mean placement wire cost and routed critical path delay. Testing was performed on the *4lut_sanitized* architecture using the most commonly used methodology: minimum sized arrays with 1.2x the minimum channel width. Placements were routed using the

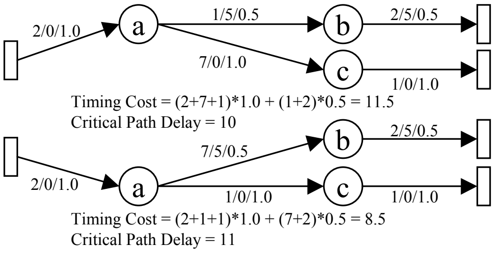


Figure 1. Effect of stale criticality information. Notation: delay / slack / criticality

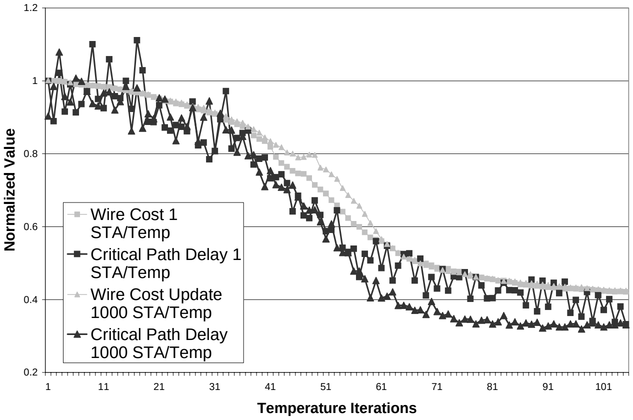


Figure 2. Netlist *ex5p* and accurate criticality information

built in timing-driven routing tool with *A** optimizations turned off. We did not use *A** during any of our testing because we noticed that VPR's aggressive implementation affected routing quality unpredictably. Here we can see a relatively clear average critical path delay benefit of 14% when we perform 100-1000 static timing analyses per temperature. Updating more frequently than that does not seem to have measurable additional benefit.

Although this critical path delay benefit is nice, there is the matter of placement runtime. While CPU time is notoriously difficult to equitably report (doubly so in our case since we utilized a Condor cluster to perform testing), we expect that such placements would take 100-1000x longer to produce. This is because the time required to perform static timing analysis largely eclipses any of the other necessary calculations associated with placement.

3.2 Registered Netlists & Placement Stability

Unfortunately, our concerns surrounding timing-driven placement go beyond computational complexity when we consider pipelined netlists. If we attempt to repeat our testing, but with heavily registered circuits, we can see that frequent static timing analysis does not necessarily improve our results. Rather, it can induce serious placement convergence problems.

Table 1. Benefits of frequent static timing analysis – Conventional MCNC netlists

λ,Crit_Exp	Static Timing Analysis/Temp	Normalized Wire Cost	Normalized Routed CPD
0.5, 8.0	1	1.00	1.00
	10	1.03	0.90
	100	1.03	0.86
	1000	1.03	0.86
	10000	1.04	0.87

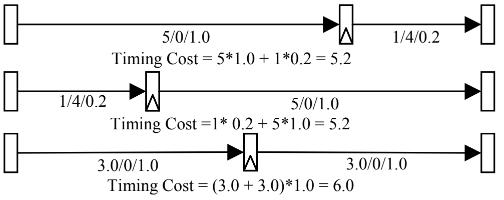

5/0/1.0
1/4/0.2
Timing Cost = 5*1.0 + 1*0.2 = 5.2
1/4/0.2
5/0/1.0
Timing Cost =1* 0.2 + 5*1.0 = 5.2
3.0/0/1.0
3.0/0/1.0
Timing Cost = (3.0 + 3.0)*1.0 = 6.0

Figure 3. Registered netlists & placement oscillation

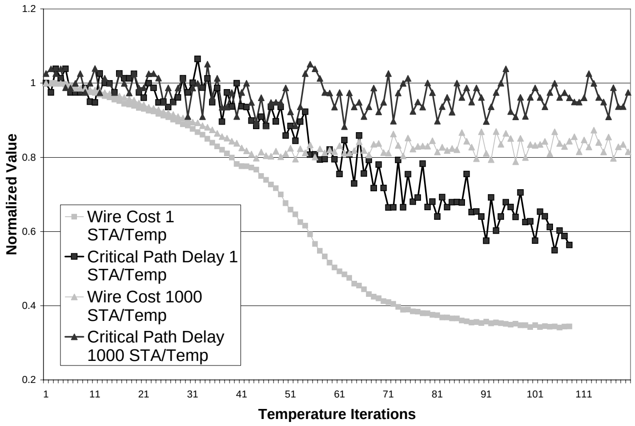


Figure 4. Pipelined/retimed netlist ex5p and placement convergence problems (λ = 0.5, *Crit_Exp* = 8.0)

If we consider the example in Figure 3, we see that if the critical path delay, slack and source/sink criticalities are updated after each move, we encourage the system to place the register off-center, actively avoiding the placement with the best critical path delay. Of even greater concern, though, is that the register can freely move between being off-center to the left and off-center to the right. For a placement tool concerned with trying to optimize both wirelength and delay simultaneously, this makes for a difficult, constantly moving target. This destabilizes the placement and can cause the annealing not to converge. When we were not updating criticalities very often under the classic VPR scheme, although we were not necessarily optimizing towards the correct goal, at least the guiding forces in the placement within a given temperature iteration were consistent. In that way we would always make forward progress, albeit towards a potentially less than optimal destination.

We can see this problem clearly manifest itself when we repeat our static timing analysis testing on fully pipelined/C-slowed and retimed netlists. All of the original 22 VPR benchmarks were pipelined and/or C-slowed, then processed via Leiserson/Saxe retiming [4] to create circuits with a maximum logic depth of one LUT. These netlists were then packed into CLBs with T-VPack and, again, placed onto minimum-sized *4lut_sanitized* architectures with 1.2x the minimum channel width as found by default VPR. As with our earlier testing, routing was handled by the VPR timing-driven routing tool with *A** disabled.

In Figure 4 we show two placement runs of the fully pipelined *ex5p* netlist. Indicated with squares is the wire cost and critical path delay for placement when we perform static timing analysis once per temperature iteration. Indicated with triangles are the results when we perform 1000 static timing analyses per temperature iteration. From this graph, the placement performed with very frequent timing analysis clearly suffers from convergence issues.

We found this kind of behavior typical among our pipelined MCNC netlists. When performing placement on these circuits with more frequent static timing analysis and the default values VPR suggests for Eq. 1 and 3 (λ = 0.5, *Crit_Exp* = 8.0), we found that many of the netlists' registers oscillate in a futile attempt to reduce their critical path delay. This results in poor placements with unusually high wire cost. We can combat this oscillation somewhat by lowering the emphasis we place on timing.

Unfortunately, while this allows us to get stable placements more reliably, we also somewhat compromise our search for good critical path delay. Shown in Table 2 are the results we obtained from VPR using (λ = 0.5, *Crit_Exp* = 1.0). For these fully pipelined netlists we are able to improve critical path delay by 16% when we perform 100 static timing analyses per temperature. However, even our attempts at stabilizing the annealing are not entirely successful when we attempt to perform more than 100 timing analyses per temperature. Although we were unable to complete our testing of VPR varying both λ and Crit_Exp due to the extremely large computational requirements, preliminary testing indicates that even the best placement parameters would only improve VPR's achievable critical path delay by approximately 5-10%.

If we take a step back for a moment, we should not be surprised by the difficulties we encountered producing high-quality timing-driven placements for regular netlists, much less pipelined netlists. Heavily registered circuits are well known for posing

Table 2. Limitations of frequent static timing analysis - Fully pipelined MCNC netlists. Asterisk indicate that some placements were unroutable and not included in the average.

λ, Crit_Exp	Static Timing Analysis/Temp	Normalized Wire Cost	Normalized Routed CPD
0.5, 8.0	1	1.00	1.00
	10	1.06*	0.95*
	100	1.06*	0.75*
	1000	1.73*	0.68*
0.5, 1.0	1	0.96	0.89
	10	0.96	0.86
	100	0.96	0.84
	1000	0.96*	0.86*
	10000	0.96*	0.85*

unique problems to CAD tools. For example, much of our discussion so far is very reminiscent of the difficulties encountered in [2] which attempted to tackle the issues of timing-driven routing for pipelined netlists. This holds particularly true given the issue raised by Figure 3, where VPR's placement algorithm actually discourages the best placement.

Furthermore, placement for pipelined netlists has been a known difficult problem for some time. For example, the deeply pipelined radio cross-correlator in [7] was laboriously hand-placed by the author to achieve good performance. This painstaking process even inspired the authors of [1] to develop a specific tool to assist in manual pipelining and placement. The extreme difficulty of such an endeavor, given the scale of even relatively small FPGA designs, is likely indicative of the complexities these netlists present to the design tools.

4. Efficient and Stable Placement

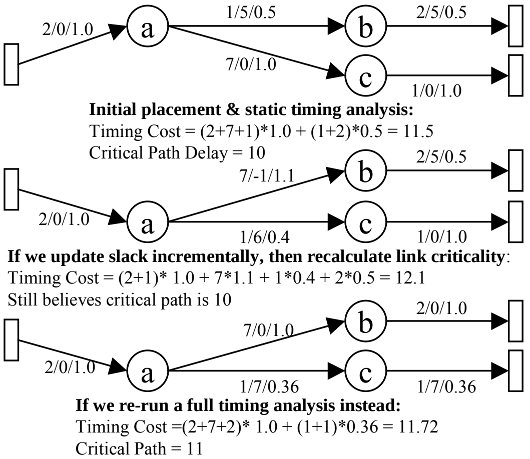
Looking back at the problems we encountered during placement we can identify two primary issues. First, to produce high quality placements we need to have up-to-date criticality information. How do we get this data without resorting to the computationally impractical solution of performing a full static timing analysis after each move? Second, worrisome instability develops during the annealing process when we attempted to use fresh timing information during the placement of registered netlists. What can we do to stabilize the system?

4.1 Updating Criticality Incrementally

We believe we can provide current timing information with low computational effort by tracking incremental changes to link slack. Although this methodology can only estimate criticality, it does provide enough information to the placement tool to reveal shifts in timing significance. While nothing can replace a full static timing analysis performed at the beginning of each temperature iteration, we can attempt to maintain the relevance of this information by reflecting changes in link delay on link slack.

Each time an annealing move is made, VPR's timing-driven placement algorithm already evaluates the change in link delay for all sources and sinks connected to the migrated blocks. As seen in Eq. 8 and 9, if we simply subtract the change in link delay from the link slack, we can easily recompute an estimated source/sink criticality for the new placement.

$$Slack(i,j,k,l) = Slack(i,j,k,l-1) - \Delta Delay(i,j,k,l) \qquad (8)$$



Initial placement & static timing analysis:
Timing Cost = (2+7+1)*1.0 + (1+2)*0.5 = 11.5
Critical Path Delay = 10

If we update slack incrementally, then recalculate link criticality:
Timing Cost = (2+1)* 1.0 + 7*1.1 + 1*0.4 + 2*0.5 = 12.1
Still believes critical path is 10

If we re-run a full timing analysis instead:
Timing Cost =(2+7+2)* 1.0 + (1+1)*0.36 = 11.72
Critical Path = 11

Figure 5. Accuracy of incremental slack and criticality update versus full timing analysis

$$Criticality(i,j,k,l) = 1 - \frac{Slack(i,j,k,l)}{D_{\max}(k)} \qquad (9)$$

While less accurate than a complete timing analysis, this only requires two additional add/subtracts and one multiplication/division to preserve the majority of the accuracy of the netlist's criticality information. Looking at the top and middle illustrations of Figure 5, we can see this technique in action. Here we revisit the example fromFigure 1, but incrementally update the slack and link criticality information.. The suggested move decreases the delay on (*a*, *c*) by six units from 7 to 1 and increases the delay on (*a*, *b*) by six units from 1 to 7. To evaluate the quality of the new placement, we reflect this change on the links' slacks. Since (*a*, *c*) was on the critical path, the original slack was 0. Thus, we account for the six unit drop in delay and calculate the new slack on this link to be (0 + 6 = 6). We then use this updated slack to recalculate the criticality of this link. In this case, we still believe the critical path to be 10 units, so we get a criticality of 0.4. Similarly we account for the six unit increase in delay on (*a*, *b*) by updating the slack to (5 - 6 = -1). This makes the criticality of this link 1.1. Finally, we compute the timing cost of this new placement based upon our incrementally updated timing information. From this we can see that the annealer is now aware that the new placement is not as good as the previous one.

Although this methodology does effectively address the large-scale problem of placement in the face of inaccurate timing information, we should note that this technique cannot guarantee perfect criticality data – that would require true static timing analysis. In the bottom diagram of Figure 5 we show the link slack, criticality and timing cost of the new placement as calculated with exact static timing analysis information. If we compare the results of the two techniques, we notice that not only do we not realize that the current critical path has changed, the emphasis placed on the links between blocks *b* and *c* and the output pads is also completely incorrect. That said, our estimates do track relatively well, especially considering the extremely low computational requirements.

4.2 Reformulated Cost Function

If we re-examine the problems that we encountered in Section 3, we realize that a good timing-driven placement cost function should have three qualities. First, it should account for the change in criticality that occurs on a given link when we increase or decrease the link's delay. Without this characteristic we can unwittingly decrease delay on already fast links while adding delay to already slow links. Second, the cost function must prefer balanced delay to unbalanced delay. That is, if the placement tool believes it is a better idea to place a register off-center in terms of pre-register and post-register delay, the register may oscillate between the equally unbalance positions on either side. Third, the cost function should be able to recognize situations in which we have reduced the overall critical path delay of the system. Although a move might make nearly all connections in a circuit critical, this could be a positive step if the new placement has a lower critical path delay.

We believe that the timing cost methodology shown in Eq. 8-12 fulfills all of these requirements. Given the incremental slack update methodology discussed above, we calculate the new criticality of each source/sink link after each move based upon the critical path delay of the system found at the beginning of the temperature iteration. Since we already calculate the delay of each source/sink pair after each move, we can then define the timing cost of a given placement as the summation of all source/sink delays multiplied by their current estimated criticality.

$$\begin{aligned} \mathit{Timing\_Cost}(i,j,k,l) = \mathit{Delay}(i,j,k,l) *& \\ \mathit{Criticality}(i,j,k,l)^{\mathit{Crit\_Exp}}& \end{aligned} \tag{10}$$

$$\mathit{Timing\_Cost}(\mathrm{k,l}) = \sum \mathit{Timing\_Cost}(i,j,k,l) \tag{11}$$

If we look at how this affects the way differences between two placements actually manifest themselves, we see that the timing cost delta is now calculated in an inherently different way.

$$\begin{aligned} \Delta TC(i,j,k,l) = &\begin{bmatrix} \mathit{Delay}(i,j,k,l) * \\ \mathit{Criticality}(i,j,k,l)^{\mathit{Crit\_Exp}} \end{bmatrix} - \\ &\begin{bmatrix} \mathit{Delay}(i,j,k,l-1) * \\ \mathit{Criticality}(i,j,k,l-1)^{\mathit{Crit\_Exp}} \end{bmatrix} \end{aligned} \tag{12}$$

Here we see that the previous delay is multiplied by the previous criticality and the new delay is multiplied by the new criticality. This is quite different from the timing cost delta shown in Eq. 7.

The effect of this during placement is dramatic. For example, if we return to the problematic example in Figure 3, we can see that our modified cost function now correctly identifies the optimal placement of the register. This is shown in Figure 6.

5. Testing

We tested this improved timing-driven placement technique using the same set of classical MCNC and pipelined/C-slowed/retimed MCNC netlists as mentioned earlier. With the exception of the placement tool, all other testing considerations were kept the same.

After some brief exploratory tuning, we used (λ = 0.1, *Crit_Exp* = 12.0) to place the unpipelined MCNC netlists and (λ = 0.025, *Crit_Exp* = 12.0) to place the pipelined circuits. Although we are still investigating the exact relationship, we know that our new cost formulation requires much smaller values of λ to produce good results. If we look at Eq. 9 and 10, we can see why. When we reduce delay on a given link, we actually, from the standpoint of the classical VPR framework, double-count this reduction. This is because, unlike what VPR is expecting, the criticality of this link will also be updated to reflect the smaller delay. Thus, when we multiply the two factors together, our delta timing costs naturally become much larger than the range that the existing VPR framework is expecting. A similar situation holds true for when we increase the delay on a given link.

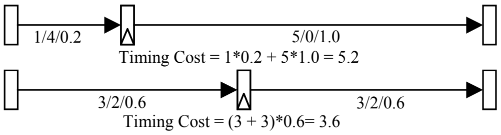

1/4/0.2
5/0/1.0
Timing Cost = 1*0.2 + 5*1.0 = 5.2
3/2/0.6
3/2/0.6
Timing Cost = (3 + 3)*0.6= 3.6

Figure 6. Stability of the reformulated cost function

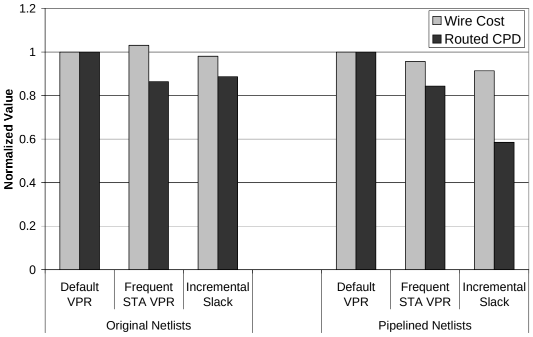


Figure 7. Wire cost and post-routing critical path delay comparison for all examined tools

Figure 7 shows a summary of our results, and further details can be found in Table 3. Taking at look at these values, we can see that our incremental slack approach combined with our reformulated cost function produces very good placements. While the wire cost for placements performed on the purely combinational or lightly registered original MCNC netlists is comparable to that of VPR, we achieve an average of 10% better post-routing critical path delay. Even if ignore the several orders of magnitude difference in computational complexity between our tool and VPR with frequent static timing analysis, the achieved wire cost and critical path delay is very similar, differing by no more than a few percent.

If we look at the results for deeply pipelined MCNC netlists, we see an even more dramatic benefit to our approach. Here we achieve an average 41% better critical path delay compared to the original VPR formulation, with 9% better wire cost. Compared to VPR with frequent static timing analysis, we achieve 25% better critical path delay with 5% better wire cost – all accomplished with none of the concerns regarding the large computational complexity of static timing analysis.

Although not shown in detail due to space constraints, we also tested our approach with netlists pipelined/C-slowed/retimed to a maximum logic depth of two LUTs. Similar to what is shown in Table 3, if we normalize to the results gathered by default VPR, VPR with frequent static timing analysis (λ = 0.5, *Crit_Exp* = 2.0) produced an average wire cost of 1.00 and a critical path delay of 0.77. Although still incomplete, our preliminary testing indicates that our incremental timing approach will produce an average wire cost of 0.96 and a critical path delay of 0.63.

6. Future Work

We have performed all of our testing thus far on the *4lut_sanitized* architecture. While this very simple FPGA is a useful comparison target due to its ubiquitous nature, modern reconfigurable devices generally include clustered CLBs and multiple lengths of long-distance wires. Although, based upon our discussion and the results seen so far, we expect our approach to produce superior placements as compared to existing placement tools, we would like to investigate exactly how our technique scales to more sophisticated architectures.

7. Conclusions

In this paper we have identified some longstanding but poorly understood problems that surround CAD for pipelined netlists. We have shown that although timing-driven placement can improve critical path delay for conventional netlists, existing methodologies have fundamental shortcoming when attempting to deal with registered circuits. Not only are there multiple concerns regarding solution quality and algorithm runtime, we must be aware that some of the inherent characteristics of pipelined circuits might interfere with the convergence of the placement tool.

We have presented two modifications to the classic timing-driven placement methodology that address these issues. First, our incremental slack and source/sink criticality update approach combines the low computational complexity of conventional timing-driven placement with the much more accurate criticality information of placement with frequent static timing analysis. This allows the second portion of our approach, a reformulated cost function, to accurately guide the system towards better overall placements. Combining these techniques allows us to produce much higher quality placements without significant computational overhead. For conventional netlists we produce placements that are on average 10% faster in terms critical path delay with no degradation in routability. For heavily pipelined netlists we generate placements that are 41% faster with 9% better wire cost. For netlists somewhere between the two extremes, we generate placements that are 37% faster with similar wire cost.

8. Acknowledgments

This work was sponsored in part by grant *CCF0426147* from the National Science Foundation, and grant *DE-FG52-06NA27507* from the Department of Energy. The concepts in this paper are covered by U.S. Provisional Patent Application No. 61/012,728, December 10, 2007

9. References

[1] Chow, W. and J. Rose. “EVE: A CAD Tool for Manual Placement and Pipelining Assistance of FPGA Circuits”, *Intl. Symp. on Field-Programmable Gate Arrays*, 2002: 85-94.

[2] Eguro, K. and S. Hauck. “Armada: Timing-Driven Pipeline-Aware Routing for FPGAs", *Intl. Symp. on Field-Programmable Gate Arrays*, 2006: 169-78.

[3] Kuon, I. and J. Rose, "Measuring the Gap between FPGAs and ASICs." *IEEE Trans. on Computer-Aided Design*, Vol. 26, No. 2, Feb. 2007: 203 - 15.

[4] Leiserson, C. and J. Saxe, “Retiming Synchronous Circuitry”, *Algorithmica*, Vol. 6, 1991: 5-35.

[5] Marquardt, A., V. Betz, and J. Rose. “Timing-Driven Placement for FPGAs.” *Intl. Symp. on Field Programmable Gate Arrays*, 2000: 203-13.

[6] McMurchie, L. and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs." *Intl. Symp. on Field-Programmable Gate Arrays*, 1995: 111-7.

[7] Von Herzen, B. “Signal Processing at 250MHz using High-Performance FPGA’s”, *Intl. Symp. on Field-Programmable Gate Arrays*,1997: 62-8.

Table 3. Complete wire cost and post-routing critical path delay results

	Original MCNC Netlists						Pipelined/C-Slowed & Retimed MCNC Netlists					
	Default VPR $\lambda = 0.5$, $CritExp = 8.0$		Frequent STA VPR $\lambda = 0.5$, $CritExp = 8.0$		Incremental Slack $\lambda = 0.1$, $CritExp = 12.0$		Default VPR $\lambda = 0.5$, $CritExp = 8.0$		Frequent STA VPR $\lambda = 0.5$, $CritExp = 1.0$		Incremental Slack $\lambda = 0.025$, $CritExp = 12.0$	
Netlist	**Wire**	**CPD**	**Wire**	**CPD**	**Wire**	**CPD**	**Wire**	**CPD**	**Wire**	**CPD**	**Wire**	**CPD**
alu4	201.10	7.83E-08	207.61	7.23E-08	199.37	7.58E-08	291.84	3.83E-08	268.85	3.07E-08	253.27	3.12E-08
apex2	280.18	9.66E-08	280.98	8.38E-08	272.07	8.61E-08	407.76	4.03E-08	375.01	3.30E-08	364.76	2.42E-08
apex4	192.57	7.74E-08	193.54	7.56E-08	184.70	8.33E-08	213.56	3.24E-08	207.68	3.07E-08	200.03	2.00E-08
bigkey	206.92	7.56E-08	242.81	4.42E-08	237.22	4.32E-08	269.51	4.70E-08	250.53	3.85E-08	242.04	2.89E-08
clma	1481.57	2.42E-07	1595.86	1.56E-07	1424.74	1.59E-07	2414.58	9.42E-08	2520.12	8.72E-08	2103.43	6.70E-08
des	249.48	9.12E-08	262.28	8.61E-08	258.01	7.16E-08	352.68	4.65E-08	349.63	3.14E-08	339.55	2.05E-08
diffeq	157.43	6.24E-08	158.04	5.94E-08	147.88	6.24E-08	485.70	5.38E-08	472.34	4.30E-08	459.45	2.87E-08
dsip	199.69	7.34E-08	230.60	3.95E-08	228.00	4.82E-08	259.39	4.31E-08	229.13	3.42E-08	203.01	3.36E-08
e64	30.21	3.12E-08	30.62	2.88E-08	29.81	3.18E-08	44.35	1.99E-08	40.98	1.46E-08	40.02	1.17E-08
elliptic	502.36	1.11E-07	519.86	1.04E-07	465.58	9.55E-08	1430.86	8.41E-08	1314.98	6.32E-08	1324.43	4.58E-08
ex1010	678.37	1.81E-07	676.88	1.49E-07	663.71	1.48E-07	876.20	5.40E-08	825.75	5.10E-08	796.05	3.64E-08
ex5p	178.17	6.75E-08	180.24	6.43E-08	169.60	6.99E-08	224.83	2.65E-08	216.80	3.18E-08	211.40	1.70E-08
frisc	584.86	1.62E-07	611.27	1.35E-07	536.85	1.29E-07	1427.26	7.17E-08	1438.06	7.32E-08	1382.66	2.82E-08
misex3	199.39	7.34E-08	202.35	6.42E-08	194.78	6.82E-08	269.73	3.53E-08	252.47	3.00E-08	239.27	2.23E-08
pdc	934.04	1.49E-07	958.96	1.43E-07	916.11	1.58E-07	1185.60	7.69E-08	1175.54	7.16E-08	1108.00	3.84E-08
s1423	16.37	5.82E-08	16.33	6.17E-08	15.56	7.05E-08	75.38	2.24E-08	68.70	1.58E-08	69.63	9.34E-09
s298	228.22	1.32E-07	228.34	1.33E-07	211.04	1.33E-07	456.04	4.85E-08	484.41	3.91E-08	417.54	2.82E-08
s38417	693.47	1.02E-07	706.32	7.84E-08	663.02	8.10E-08	1976.38	7.21E-08	1872.36	5.52E-08	1898.11	3.30E-08
s38584.	678.84	1.06E-07	687.54	7.19E-08	686.95	7.14E-08	1721.81	1.19E-07	1494.88	1.05E-07	1513.75	9.22E-08
seq	259.92	7.90E-08	264.26	7.61E-08	254.12	7.59E-08	355.04	4.49E-08	341.66	3.30E-08	325.21	3.07E-08
spla	625.59	1.35E-07	638.53	1.53E-07	627.76	1.41E-07	846.56	5.33E-08	828.61	5.04E-08	782.97	3.29E-08
tseng	102.62	5.53E-08	102.31	5.23E-08	95.51	5.58E-08	308.53	4.55E-08	314.08	4.02E-08	300.79	2.34E-08
Norm	**1.00**	**1.00**	**1.03**	**0.86**	**0.98**	**0.90**	**1.77**	**0.53**	**1.70**	**0.44**	**1.62**	**0.31**
							1.00	**1.00**	**0.96**	**0.84**	**0.91**	**0.59**