Predicting Interconnect Delay for Physical Synthesis in a FPGA CAD Flow

Valavan Manohararajah, Gordon R. Chiu, Deshanand P. Singh and Stephen D. Brown
Altera Toronto Technology Center
vmanohar|gchiu|dsingh|sbrown@altera.com

ABSTRACT

This paper studies the prediction of interconnect delay in an industrial setting. Industrial circuits and two industrial FPGA architectures were used in the study. We show that there is a large amount of inherent randomness in a state-of-the-art FPGA placement algorithm. Thus, it is impossible to predict interconnect delay with a high degree of accuracy. Futhermore, we show that a simple timing model can be used to predict some aspects of interconnect timing with just as much accuracy as predictions obtained by running the placement tool itself. Using this simple timing model in a two-phase timing driven physical synthesis flow can both improve quality of results and decrease runtime. Next, we present a metric for predicting the accuracy of our interconnect delay model and show how this metric can be used to reduce the runtime of a timing driven physical synthesis flow. Finally, we examine the benefits of using the simple timing model in a timing driven physical synthesis flow, and attempt to establish an upper bound on these possible gains, given the difficulty of interconnect delay prediction.

1. INTRODUCTION

Most of the delay in an FPGA is due to its programmable interconnect. In academic FPGA architectures, the interconnect is responsible for approximately 60% of the total circuit delay [1]. Our experience with commercial FPGA architectures indicates that the interconnect delay is responsible for an even higher fraction of the total delay (60% to 70%). Once a circuit has been placed and routed, interconnect delay is known exactly. However, the prediction of interconnect delay even before the physical design steps (placement and routing) in an FPGA CAD flow are executed is a difficult yet important problem. The ability to predict interconnect delay early in the CAD flow offers two advantages. First, the timing driven restructuring operations carried out during the early CAD steps (synthesis and technology mapping) can be made much more effective if interconnect delay can be predicted with reasonable accuracy. Second, the delay predictions can be used to provide feedback to the user during design restructuring or floorplanning operations without having to go through lengthy place and route steps.

There are a number of methods for predicting wire length before placement and routing have taken place [2, 3, 4]. Most of these methods apply to ASIC CAD flows. We know of only one method that treats the version of the problem found in FPGAs [5]. In an FPGA with buffered routing switches, the wire length and delay of a connection are highly correlated. Thus, it can be expected that a method that predicts wire length can be used to predict delay. However, methods that predict wire length do so by examining the local structure of a connection. While this may produce reasonably accurate wire lengths in a CAD flow where the minimization of wire length is the primary objective, in a timing driven CAD flow, the primary objective is to minimize the delay of critical connections, and examination of local structure is not adequate to identify the critical connections in the circuit or the type of optimizations typically performed by timing driven place and route tools.

Our work's focus is on the predictability of interconnect delay before placement has taken place. We view placement to be the key step in the predictability problem. Once placement is completed, every logic element in the circuit has been assigned to a particular location on the FPGA and the routing step is left the task of determining the routes needed to realize the circuit connections. The routing step is quite predictable because the router will often use the fastest possible routes for the most critical connections and use the slower routes for the non-critical connections.

This work presents five results on the predictability of interconnect delay in FPGAs. First, it is shown that the placement step, in a state-of-the-art FPGA CAD tool, has a large amount of inherent randomness present. Second, a simple delay model is shown to predict some aspects of interconnect timing with just as much accuracy as the placement tool itself. Third, a method of overcoming the limited predictability present in the FPGA CAD flow is described for a physical synthesis tool. Fourth, a metric for measuring the predictability of interconnect delays and techniques for improving the runtime without sacrificing the quality of a timing-driven restructuring flow are presented. Finally, we establish an upper bound on the gains that are possible with early timing-driven restructuring operations by using an ideal predictor.

2. TARGET FPGA ARCHITECTURES

Altera's Stratix [6] (a 4-input look-up-table architecture) and Stratix II [7] (a 6-input adaptive look-up-table architecture) chips were used to study the predictability of interconnect delay. As illustrated in Figure 1, the high level structure of both chips is similar. Both chips are comprised of I/O elements (IOEs), logic array blocks (LABs), digital signal processing blocks (DSPs) and memory elements (M512, M4K and M-RAM). While DSPs and memory elements perform very specific roles in the FPGA, the LABs can be configured to perform arbitrary logic functions. The

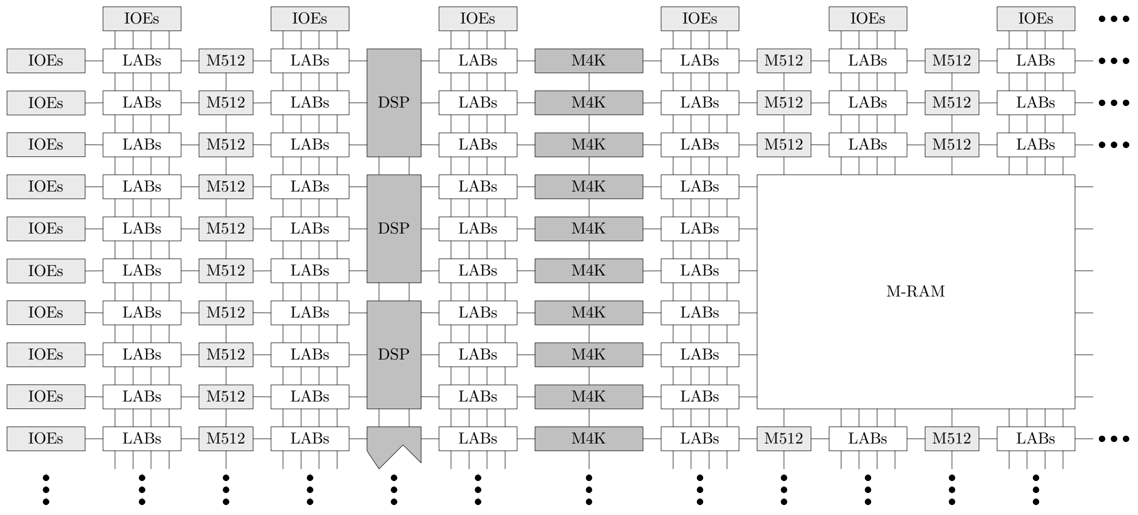


Figure 1: Structure of the FPGA architectures.

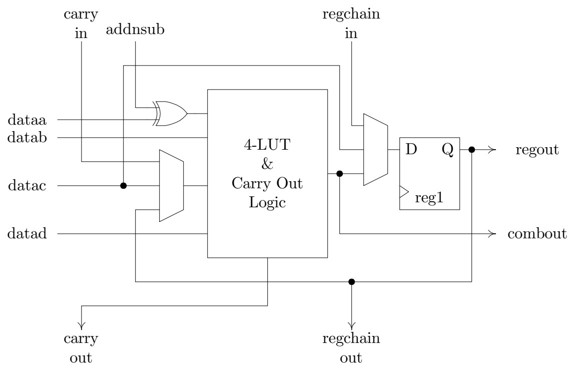


Figure 2: The 4-input logic element.

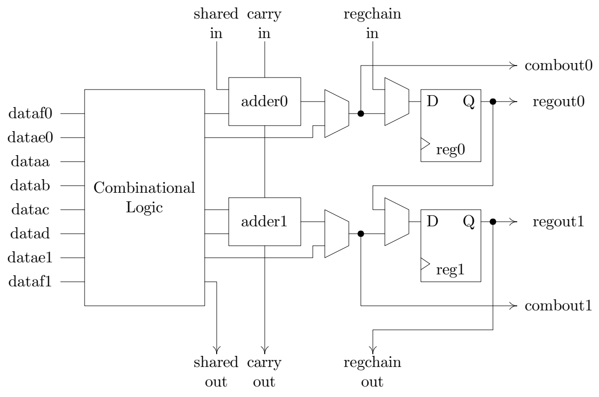


Figure 3: The 6-input adaptive logic module.

LABs are also the source of significant differences between the two architectures. A LAB in a 4-LUT device contains 10 logic elements (LEs) while a LAB in an adaptive-LUT device contains 8 adaptive logic modules (ALMs). The LE on a 4-LUT device, illustrated in Figure 2, contains a four-input lookup table (4-LUT), a register and some logic that facilitates the creation of arithmetic circuits. Figure 3 illustrates the ALM. It contains two registers, two sets of addition circuitry and a combinational logic module that can implement two functions of varying complexity. The combinational logic module can be configured to implement a single 6-LUT, or two LUTs with five or fewer inputs. If the module is configured to implement two 5-LUTs, the LUTs must share at least two of their inputs as there are only 8 inputs connected to the module.

3. FPGA CAD FLOW

Figure 4 illustrates the CAD flow used in this work. The first two steps of the CAD flow were performed by a leading third-party synthesis tool capable of targeting Altera devices and the last three steps were performed by Altera's Quartus II CAD software [8]. In the first step, design entry, the design is described in terms of a hardware description language such as VHDL or Verilog. Synthesis optimizes the circuit obtained from design entry. During synthesis the netlist is represented in terms of a generic gate library. A technology mapping stage within synthesis converts the netlist into the logic elements available in the target FPGA architecture. Placement determines a location for each of the logic elements in the circuit and routing determines the wires that will be used to connect up the elements making up the circuit. A configuration file that can be used to program the target FPGA is produced at the end of the CAD flow.

4. PRELIMINARIES

If the delay within circuit components and the delay of connections between circuit components are known, timing analysis can be used to establish the *slack* [9] of every connection in the circuit. The slack of a connection is defined to be the amount of delay that can be added to the connection before it becomes *critical*. A connection is critical if the length of a path it belongs to is equal to the length of the longest path in the circuit. Timing analysis also establishes a *criticality* [10] for each connection. The criticality for a connection i, $crit_i$, is defined to be

$$crit_i = 1 - \frac{slack_i}{maxslack} \tag{1}$$

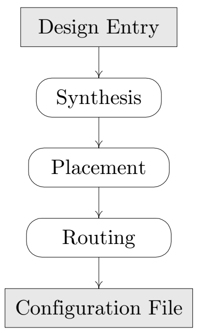


Figure 4: The FPGA CAD flow.

where $slack_i$ is the slack of i and $maxslack$ is the maximum slack observed in the circuit. Criticality is a value between 0 and 1 which indicates the relative importance of each connection to overall circuit timing. Connections that have a significant effect on circuit timing have criticalities closer to 1 while connections that have negligible effect on circuit timing have criticalities closer to 0.

Fifty industrial circuits were used to study the predictability of interconnect delay. The circuits ranged in size from 200 to 10000 LEs with an average size of approximately 3300 LEs. The circuits will be identified by the names *circuit1*, *circuit2*, ..., and *circuit50*.

5. PLACEMENT, SELF PREDICTION

The goal of placement is to produce a circuit that optimizes circuit timing and minimizes the amount of wiring required by the circuit. In a timing-driven CAD flow, the optimization process that takes place during placement is guided by a cost function that places a heavier emphasis on the former (timing) rather than the latter (wirelength). Of primary importance to the cost function is the notion of criticality defined earlier. Logic is placed in a way that minimizes the distances spanned by connections with large criticalities.

A simple experiment can be performed to determine the upper bound on the predictability of interconnect delay. The behavior of the placer can be changed by choosing a different initial seed. Different seeds usually lead to different placement solutions, but the quality of the solutions are often similar. An upper bound on the predictability of interconnect delay can be obtained by running placement with two different seeds. The delays obtained using the first seed are used as a prediction of the delays that will be obtained using the second seed. Since the algorithm places a heavy emphasis on the connections with high criticality, we cannot expect the delay of connections with low criticality to be predictable. Thus, as a measure of predictability, we compute the correlation coefficient (r^2) of the delays for connections with a criticality higher than 0.5 in either of the two runs. Our justification for the choice of 0.5 as the threshold separating those connections important to predictability from those that are not is as follows. Experimentally, we have observed that 99% of the connections on the critical path (criticality of 1) for one seed have a criticality in the range 0.5–1.0 for an alternate seed. Thus, from a delay prediction perspective, it is imperative that we are able to consistently predict the behaviour of those connections with criticality above 0.5.

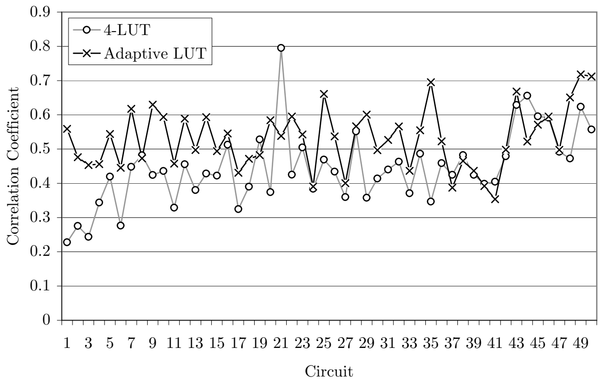


Figure 5: Correlation coefficient of the delays obtained from two different seeds.

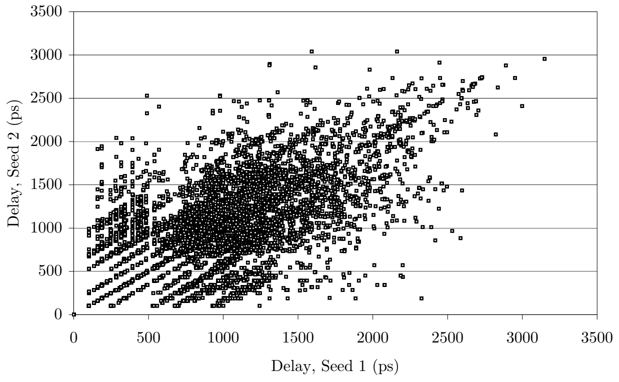


Figure 6: A plot of the delays obtained using two different seeds on *circuit26* (4-LUT device).

Figure 5 presents the values of r^2 for fifty industrial circuits and two FPGA architectures. The average value of r^2 on 4-LUT devices was 0.43 and the average value of r^2 on adaptive-LUT devices was 0.52. Although there are few circuits whose delay is highly predictable, as indicated by the average, the predictability for most circuits is fairly low. This is more apparent if we take a closer look at a circuit with an r^2 value close to the average for each device. Figure 6 presents a plot of the delays from *circuit26* which has an r^2 value of 0.43 (4-LUT device) and Figure 7 presents a plot of the delays from *circuit44* which has an r^2 value of 0.52 (adaptive-LUT device). In both figures, the delays obtained using the first seed are not well correlated to the delays obtained using the second seed. In Figure 6, a connection with a delay of 1000ps on the first seed may have a delay ranging from 100 to 2500ps on the second seed. Similarly, in Figure 7, a connection with a delay of 500ps on the first seed may have a delay ranging from 0 to 1750ps on the second seed. Clearly, interconnect delay cannot be be predicted well enough to be used for early timing-driven synthesis.

An experiment, much like one performed for interconnect delays, can be performed for interconnect criticalities. Placement uses criticalities to guide its optimization decisions. A timing driven synthesis tool can use criticalities

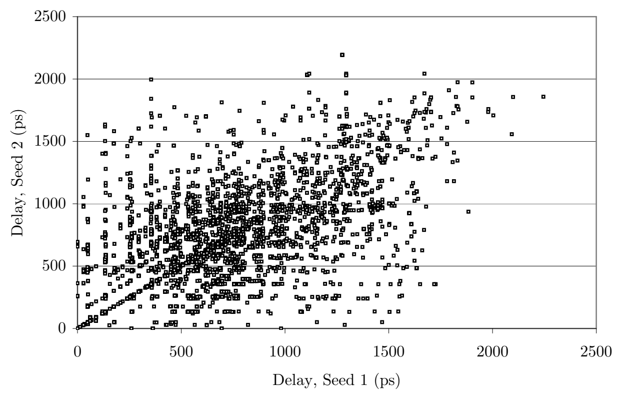


Figure 7: A plot of the delays obtained using two different seeds on *circuit44* (adaptive-LUT device).

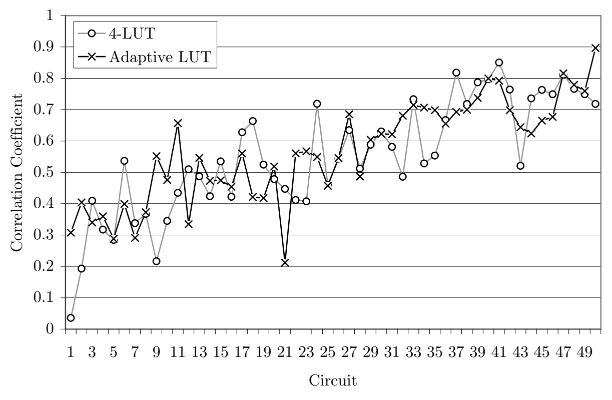


Figure 8: Correlation coefficient of the criticalities obtained from two different seeds.

to identify the parts of the circuit that are on or near the critical path. Thus, the prediction of interconnect criticality is just as important as the prediction of interconnect delay. Note that the prediction of criticalities is quite different from the prediction of delays — delay is a property of an individual connection while criticality is a property of the longest path through a connection. Since circuit timing is determined by the length of paths rather than the delays of individual connections, we expect criticalities to be more predictable than delays. Figure 8 presents the correlation coefficient of the criticalities obtained from running each circuit with two seeds. The average value of the correlation coefficient was 0.51 and 0.54 on 4 and adaptive-LUT devices, respectively. Although the predictability of criticalities is a little better than the predictability of delays, the effect is most significant on 4-LUT devices.

Note that delays and criticalities are more predictable on adaptive-LUT devices than on 4-LUT devices. This is primarily due to the greater amount of logic being packed into each adaptive-LUT device LAB. Recall that an adaptive-LUT device LAB contains 8 ALMs (compared to 10 LEs in a 4-LUT device LAB), and each of these ALMs contain at least as much functionality as two 4-LUTs. With more logic being packed into each LAB, many connections will use the highly predictable LAB interconnect and will not need

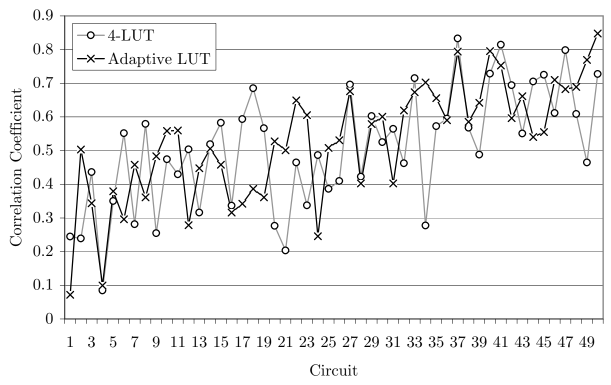


Figure 9: Correlation coefficient of the criticalities obtained using the simple delay model with respect to the criticalities from actual placement.

to traverse the less predictable programmable interconnect connecting LABs.

6. A SIMPLE EARLY TIMING MODEL

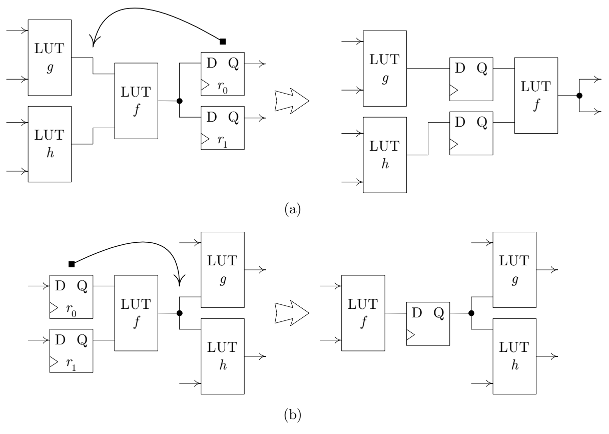
We now propose a simple delay model that can be used by early timing-driven synthesis tools. As demonstrated in the preceding section, interconnect delay is hard to predict even if an identical placement algorithm is used. Our model makes no attempt at predicting delays. It simply looks up a single delay value for each *connection type* in a table. A connection's type is determined by four components: type of node and port at the connection's source, and type of node and port at the connection's destination. For example, logic elements/modules (such as those in Figures 2 and 3), memory elements, DSP blocks and I/O elements are all considered to be different node types. Furthermore there are several distinct port types that enter and leave each node type. For example, for the 4-LUT in Figure 2 there are 7 input port types and 4 output port types. To generate a table of delay values, a large number of experiments were run on a variety of circuits and the delays observed for each connection type was recorded. For each connection type, an average weighted delay is then computed from the recorded results. The weight of a delay d is given by $\frac{1}{(1+d)}$. This weighting scheme favors the low delay values when computing the average because we expect timing driven tools to focus on the critical connections which will most likely use the fastest available routes.

This model is far too simplistic to produce accurate predictions of interconnect delay. However, the criticalities computed using this model are almost as good as those obtained from running placement. Figure 9 illustrates the correlation coefficient of the criticalities obtained using the simple delay model with respect to the final criticalities obtained from placement. The average correlation coefficient on 4-LUT devices was 0.47 (versus 0.51 computed using two seeds) and the average correlation coefficient on adaptive-LUT devices was 0.49 (versus 0.54 computed using two seeds).

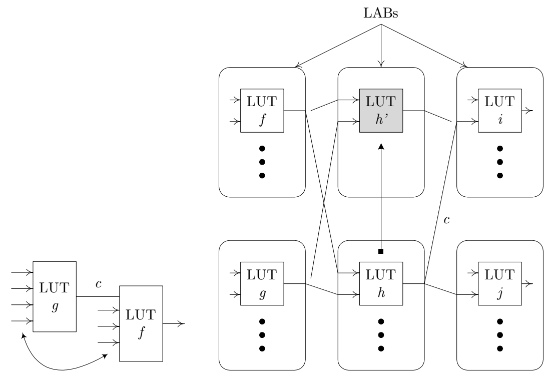
7. USE OF PREDICTION IN EARLY PHYSICAL SYNTHESIS

Physical synthesis is often applied after the completion of placement. Placement is sufficiently close to the routing step that reasonably accurate interconnect delays are known and small changes to the circuit can still be made without too much difficulty. During physical synthesis, estimates of interconnect delay obtained from the existing placement guide the synthesis transformations that restructure the logic on or near the critical path. These transformations will often leave the circuit in a illegal state and an incremental placement step [11] is used to resolve any illegalities that are introduced.

In earlier versions of the place and route CAD flow, physical synthesis was performed exclusively after placement. Transformations include register retiming [12], timing-driven functional decomposition [13], local rewiring [14], and logic replication.



(a) Retiming.



(b) Local Rewiring. (c) Logic Replication

Figure 10: Retiming, Rewiring, and Replication.

Figure 10(a) shows register retiming, a powerful logic optimization technique for synchronous circuits. Register retiming uses the property that registers can be taken from the outputs of gates and moved to their inputs, or vice versa. Using these moves in combination, one can attempt to maximize circuit speed and minimize area.

Figure 10(b) illustrates the local rewiring optimization. We identify a pair of LUTs f and g connected by a critical signal c. Using functional decomposition techniques, we determine if the overall timing of the two LUTs can be improved by swapping some of the non-critical signals attached to f with some of the critical signals attached to g.

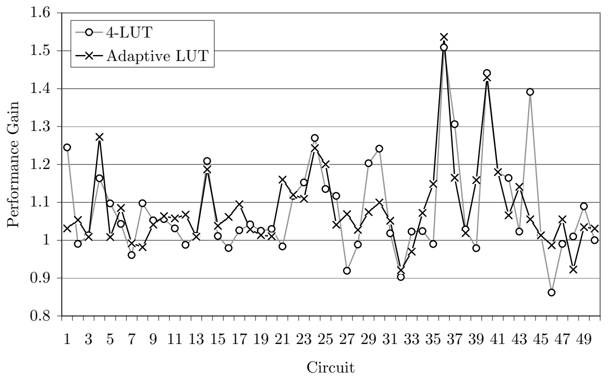


Figure 11: Performance improvements obtained using early physical synthesis.

Following placement, a LUT that drives a signal with several fanouts may be placed at a location that is not ideally suited for any of its fanouts. For example, in Figure 10(c), LUT h drives two LUTs i and j, and it has been placed at a location that minimizes its delay to destination j. However, if connection c is critical, we can replicate h to produce a new LUT h' which can be placed closer to the target of c. The logic replication algorithm performs this transformation on critical signals driven by multi-fanout sources.

These transformations, when performed after placement, were used to obtain average performance improvements of 11% on both 4- and adaptive-LUT devices. However, in obtaining these performance improvements, CAD tool runtime (for the final three steps in the CAD flow) was increased by approximately 200%.

A closer examination reveals that incremental placement is responsible for a large fraction of the runtime overhead. Modern FPGAs have a large number of architectural constraints, and resolving any illegalities involving these constraints while perturbing the existing placement as little as possible is a difficult problem. If we can predict interconnect delay with reasonable accuracy before placement has taken place, we can perform many of the physical synthesis transformations much earlier in the CAD flow and avoid the computational cost of performing placement legalization.

To use this pre-placement delay prediction, we split physical synthesis into an *early* and *late* stage [15]. Early physical synthesis takes place before placement and late physical synthesis takes place after placement. Early physical synthesis uses register retiming and timing-driven functional decomposition to restructure the logic on or near the critical path, and the delay model described in the preceding section is used to estimate interconnect delays, slacks, and criticalities. Late physical synthesis uses all transformations available, but constrains the transformations so as to prevent a large number of placement illegalities from being created. This two-stage physical synthesis flow obtains an average performance improvement of 13% while increasing runtime by approximately 150%.

To determine the benefits of the simple delay model proposed in the preceding section, early physical synthesis was run in isolation. The resulting performance gains are presented in Figure 11. An average performance gain of 7.5%

and 8.0% was observed on 4-LUT and adaptive-LUT devices, respectively. Clearly, the model predicts post-placement timing with reasonable accuracy to allow some timing-driven optimizations to take place even before placement have taken place. On 4-LUT devices, 37 circuits show an improvement, and on adaptive-LUT devices, 44 circuits show an improvement. This result is in agreement with our earlier observation that delays and criticalities are more predictable on adaptive-LUT devices as compared to 4-LUT devices. Even though physical synthesis is being performed with a simple delay model, there are several circuits (13 4-LUT device circuits and 10 adaptive-LUT device circuits) whose performance is improved by over 15%. Furthermore, there is no need for legalization during early physical synthesis and runtime overhead is reduced to approximately 60%.

The results from the above variants of the physical synthesis flow are summarized in Table 7.

Physical Synthesis Flow	$\Delta f_{\max}$	**Δ runtime**
Late Only	+11%	+200%
Early Only	+8%	+60%
Both *Early* and *Late*	+13%	+150%

Table 1: Summary of results for a physical synthesis flow.

8. ADVANCED PREDICTION IN PHYSICAL SYNTHESIS

Our modified version of the timing-driven physical synthesis CAD flow splits optimizations performed into two stages: *early* physical synthesis, which occurs prior to placement, and *late* physical synthesis, which occurs after placement [15]. Early physical synthesis performs timing-driven optimizations with a coarse estimate of delays. Runtime is reduced during this stage because the need to legalize placement is eliminated. Late physical synthesis performs optimizations with a more accurate estimate of delays, but at a higher cost due to incremental placement legalization.

In order to leverage the prediction accuracy in physical synthesis, we must first define a metric for evaluating the accuracy and quality of a prediction. We then present two techniques. These techniques demonstrate how this metric can be used to leverage the different accuracies of prediction in early physical synthesis. By changing the partitioning of operations between early and late physical synthesis depending on the prediction model accuracy, quality of results can be improved.

8.1 A Metric for Prediction Accuracy in Physical Synthesis

We define a *wire-criticality mapping function* as a function that maps interconnect to a criticality for the wire, i.e., a delay mapping function A maps interconnect i to criticality c: $A : i \rightarrow c$. To generate the *wire-criticality mapping function* for a given prediction, we first annotate all estimated interconnect delays onto the circuit, and perform a timing analysis. The timing analysis provides slacks for each interconnect edge in the circuit; from these interconnect wires we can calculate a criticality as described in Section 4.

We define $d(i)$, the criticality difference, as the absolute difference between the criticality of a wire i in two mapping functions. We define $w(i)$ as a weighting function chosen to place more emphasis on certain wires in the circuit. For the purposes of physical synthesis, prediction of critical wires is much more important than prediction of non-critical wires. Thus the weighting function $w(i)$ is designed to assign a higher weight to those wires with a higher criticality. Our chosen formulation for $w(i)$ and $d(i)$ is

$$\begin{aligned} w(i) &= (1 - \textsc{max}(A(i), B(i)))^{\alpha} \quad (\alpha > 1) \\ d(i) &= \textsc{abs}(A(i) - B(i)) \end{aligned}$$

and for our experiments we have selected the value $\alpha = 8$.

Given two delay mapping functions A and B, we can define a metric for the similarity of the two functions. We call our metric the *weighted average criticality difference* which we define as the weighted average of the *criticality differences* between two criticality estimates $A(i)$ and $B(i)$ of a wire i:

$$avgC_{diff}(A, B) = \frac{\sum\limits_{i \in interconnect} (w(i)d(i))}{\sum\limits_{i \in interconnect} w(i)}$$

where $w(i)$ and $d(i)$ are the weight and criticality difference of wire i.

Given a criticality estimate E generated from a model for interconnect delay, and the actual criticality A, the weighted average criticality difference $x = avgC_{diff}(A, E)$ provides an estimate of the quality of the given interconnect delay model. A value of x close to 0 indicates that the model provides a very accurate prediction of criticality. A large value of x indicates that the estimate and actual criticalities differ in their prediction of critical paths.

8.2 Leveraging Prediction Accuracy in Late Physical Synthesis

In this section, we use the previously defined metric to improve the partitioning between *early* and *late* physical synthesis, in the case when our delay model during early physical synthesis is accurate. By performing more optimizations during early physical synthesis, runtime is reduced because there is no need for costly placement legalization.

At the beginning of the late physical synthesis stage, we determine the accuracy of the delay model used to drive the timing-driven optimizations performed during the early physical synthesis stage. If the delay model from early physical synthesis is very similar to the current post-placement estimated delays, very little benefit will result from running optimizations during late physical synthesis. However, if the delay model from early physical synthesis is drastically different from the current post-placement delays, this implies that the previous early physical synthesis optimizations were performed using a poor delay model. Thus, extra timing driven optimizations may be necessary during late physical synthesis.

The calculations are performed as follows. We note the predicted criticality of all interconnect wires at the end of *early* physical synthesis (A), and also note the criticality of all interconnect wires at the beginning of *late* physical synthesis (E). We then compute $x = avgC_{diff}(A, E)$. We define two thresholds, x_{low} and x_{high}. If $x < x_{low}$, the timing model used to perform *early* physical synthesis timing-driven optimizations is very similar to the timing model available to *late* physical synthesis. Thus there is little benefit from performing further timing driven optimizations. Only a reduced set of optimizations are then performed in *late* physical synthesis. On the contrary, if $x > x_{high}$, the timing model used to perform early physical synthesis timing-driven optimizations is drastically different from the more accurate timing model currently available. Thus extensive optimizations need to be performed in *late* physical synthesis.

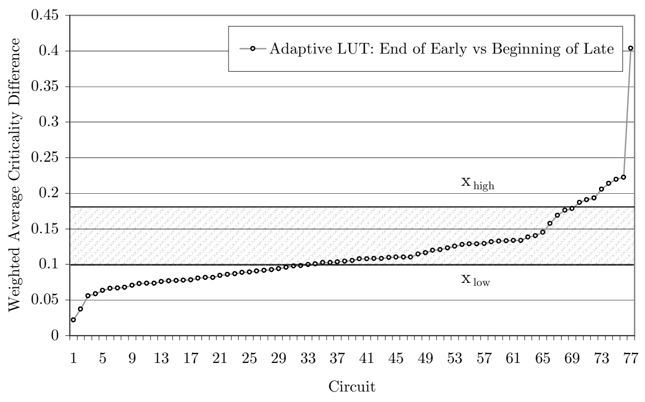


Figure 12: Weighted Average Criticality Difference - Early vs. Late

Figure 12 shows the computed weighted average criticality difference between the end of early physical synthesis and the beginning of late physical synthesis for a seventy-seven circuit benchmark set. The shaded region highlights all circuit whose weighted average criticality difference is between x_{low} and x_{high} and for which normal late physical synthesis was executed. Quick late physical synthesis was run for circuits below x_{low}, and extra late physical synthesis was run for circuits above x_{high}.

8.3 Leveraging Prediction Stability in Early Physical Synthesis

In this section, we use the previously defined metric to improve the partitioning between *early* and *late* physical synthesis, in the case when our delay model during early physical synthesis is inaccurate.

At the beginning of the early physical synthesis, we can make a prediction as to the quality of the delay model available. If the delay model from early physical synthesis is poor, very little benefit will result from running optimizations during early physical synthesis. In this case, we bypass early physical synthesis and perform extra operations during late physical synthesis. We test the *stability* of the early physical synthesis delay model by adding noise and determining its effect on the prediction of critical paths.

The calculations are performed as follows. We note the predicted criticality of all interconnect wires at the beginning of *early* physical synthesis (M). We then add small amounts of noise to the predicted delay of each interconnect wire in our circuit. The amplitude and distribution of the noise added approximate the error inherent in our prediction, and is determined statistically from experimental data collected from the benchmark set. With the noise added, we re-calculate the predicted criticality (M'). We can then compute $x = avgC_{diff}(M, M')$. This procedure is repeated several times, and an average x_{avg} is calculated. We call x_{avg} the stability of the timing model. We define a threshold, $x_{threshold}$. If $x_{avg} < x_{threshold}$, the timing model used to perform *early* physical synthesis timing-driven optimizations is stable under the addition of noise. That is, the critical path does not change very much even with the addition of the expected prediction error. Early physical synthesis can proceed as normal. On the contrary, if $x \geq x_{threshold}$, the timing model used to perform *early* physical synthesis timing-driven optimizations is very unstable. The addition of noise causes the model to predict contradicting critical paths. Thus, *early* physical synthesis should be skipped, and extra optimizations should be performed in *late* physical synthesis.

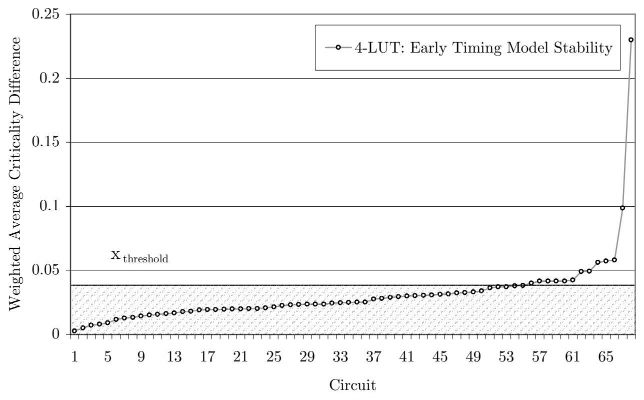


Figure 13: Weighted Average Criticality Difference - Early Model Stability

Figure 13 shows the computed average stability of the early model at the beginning of early physical synthesis for a sixty-nine circuit benchmark set. The shaded region highlights all circuit whose stability is less than the threshold $x_{threshold}$ and for which normal early physical synthesis was executed. Early physical synthesis was skipped and extra late physical synthesis was run for circuits above $x_{threshold}$.

By more intelligently partitioning runtime between early and late physical synthesis according to the accuracy of the early delay model, we can achieve significant runtime reductions. The application of these two techniques to the two-stage physical synthesis flow reduces the runtime increase from 150% to 110% with no change in the 13% performance gain.

9. AN UPPER BOUND ON EARLY PHYSICAL SYNTHESIS GAINS

Previously, an upper bound on the predictability of interconnect delay was established using *self-prediction* where the delays from placement with one seed was used to predict the delays from placement with another seed. An experiment can be performed to determine an upper bound on the gains available from the early physical synthesis flow

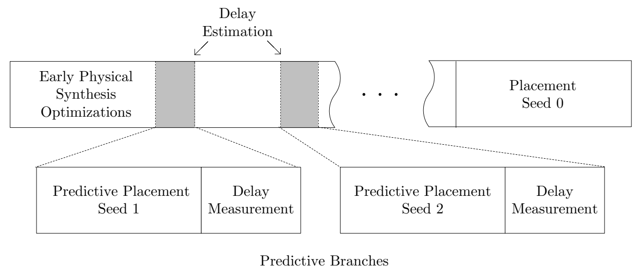

Delay Estimation
Early Physical Synthesis Optimizations
. . .
Placement Seed 0
Predictive Placement Seed 1
Delay Measurement
Predictive Placement Seed 2
Delay Measurement
Predictive Branches

Figure 14: Upper bound estimate flow using self-predictive delay model.

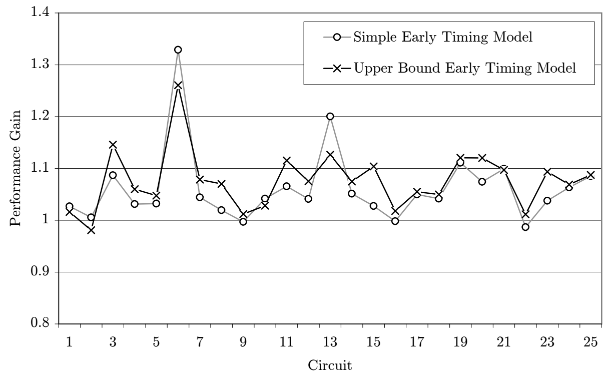


Figure 15: Performance improvements obtained from upper bound delay model.

using an early delay model derived from the self-predicted delays.

In early physical synthesis, a series of circuit transformations are performed, driven by estimated interconnect delays, slacks, and criticalities. Periodically, the slack and criticality values on the paths of the circuit need to be updated. Estimated interconnect delays are annotated onto the circuit using an early timing model.

To establish an upper bound on performance gains, we replace the existing early timing model with one based on actual self-predicted delays ("the upper bound timing model"). When interconnect estimations are required, the flow is branched and a prediction flow is run. In the prediction flow, placement is immediately performed on the circuit (with a different seed). At the end of placement, for the prediction branch, the actual delays observed for each path is noted and back-annotated onto the original circuit branch. Optimizations then proceed as normal on the original circuit branch until future interconnect estimations are required. This prediction flow is depicted in Figure 14.

It is important to note that a different seed is used for each prediction branch. Without a different seed for the original and prediction branch, each predicted delay would be identical to the actual delay. With a different seed, the interconnect delay predictability of the early timing model is equal to the upper bound on predictability – as predictable as the placement self-prediction.

The resulting performance gains from using the upper bound timing model during early physical synthesis is contrasted with the performance gains from using the simple early timing model (of Section 6) in Figure 15. Across a set of 25 circuits, an average performance gain of 7.6% was observed when using the upper bound timing model, compared to 6.0% with the simple early timing model. The upper bound model outperforms the simple model by 1.6%, with 19 circuits (76%) showing an improvement. Clearly, the upper bound model provides a better post-placement timing estimation.

Of note, the simple early timing model is able to achieve 80% of the gains available from the upper bound timing model. This result would seem to argue that only marginally increased performance gains are possible from an improved early timing model.

10. CONCLUSION

We presented five results on the problem of interconnect delay prediction in a timing driven FPGA CAD flow. First, we showed that there is a large amount of inherent randomness present in a state-of-the art FPGA placement algorithm. Second, we showed that a simple delay model can be used to estimate interconnect criticalities with just as much accuracy as running the placement tool itself. Third, we showed that the simple model can be used in conjunction with an early physical synthesis step to improve circuit performance by 7–8% on average. Fourth, we use techniques to predict the quality of the delay model and more efficiently allocate compilation time between the two stages of our physical synthesis flow. Finally, we showed that even with a delay model that achieves the upper bound in predictability, only small additional gains of 1–2% on average are available.

11. REFERENCES

[1] M. Sheng and J. Rose. Mixing Buffers and Pass Transistors in FPGA Routing Architectures. In *Proceedings of the ACM Symposium on FPGAs*, Monterey, CA, February 2001, pp. 75–84.

[2] C. Sechen. Average Interconnection Length Estimation for Random and Optimized Placements. In *Proceedings of the International Conference on Computer Aided Design*, November 1987, pp. 190–193.

[3] T. Hamada, C.-K. Cheng, and P. M. Chau. A Wire Length Estimation Technique Utilizing Neighborhood Density Equations. *IEEE Transactions on Computer-Aided Design*, Vol. 15, No. 8, August 1996, pp. 912–922.

[4] S. Bodapati and F. N. Najm. Pre-Layout Estimation of Individual Wire Lengths. In *Proceedings of the Conference on System Level Interconnect Prediction*, San Diego, CA, April 2000, pp. 93–98.

[5] S. Balachandran and D. Bhatia. A-Priori Wirelength and Interconnect Estimation Based on Circuit Characteristics. In *Proceedings of the Conference on System Level Interconnect Prediction*, Monterey, CA, April 2003, pp. 77–84.

[6] Altera. *Stratix Device Handbook (Complete Two-Volume Set)*. v3.1, Sept. 2004.

[7] Altera. *Stratix II Device Handbook (Complete Two-Volume Set)*. v1.2, Oct. 2004.

[8] Altera. *Quartus II Development Software Handbook v5.0 (Complete Four-Volume Set)*. v5.0, May 2005.

[9] R. Hitchcock, G. Smith and D. Cheng. Timing Analysis of Computer-Hardware. *IBM Journal of Research and Development*, Jan. 1983, pp. 100–105.

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

[11] D. P. Singh and S. D. Brown. Incremental Placement for Layout-Driven Optimizations on FPGAs. In *Proceedings of the International Conference on Computer-Aided Design*, San Jose, CA, 2002, pp. 752–759.

[12] D. P. Singh, V. Manohararajah, and S. D. Brown. Incremental Retiming for FPGA Physical Synthesis. In *Proceedings of the*

Design Automation Conference, Anaheim, CA, June 2005, pp. 433–438.

[13] V. Manohararajah, D. P. Singh, and S. D. Brown. Timing Driven Functional Decomposition for FPGAs. In *Proceedings of the International Workshop on Logic and Synthesis*, Lake Arrowhead, CA, June 2005, pp. 415–422.

[14] V. Manohararajah, D. P. Singh, S. D. Brown, and Z. G. Vranesic. Post-Placement Functional Decomposition for FPGAs. In *Proceedings of the International Workshop on Logic and Synthesis*, Temecula Creek, CA, June 2005, pp. 114–118.

[15] D. P. Singh, V. Manohararajah, and S. D. Brown. Two-Stage Physical Synthesis for FPGAs. In *Proceedings of the Custom Integrated Circuits Conference*, San Jose, CA, September 2005, pp. 171–178.