

Run-Time-Conscious Automatic Timing-Driven FPGA Layout Synthesis

Jason Anderson¹, Sudip Nag^{2*}, Kamal Chaudhary², Sandor Kalman²,
Chari Madabhushi², and Paul Cheng³

¹ Xilinx Inc., Toronto, ON, Canada

² Xilinx Inc., San Jose, CA, USA

³ Xilinx Inc., Longmont, CO, USA

{janders,kamal,sandor,chari,pcheng}@xilinx.com

Abstract. Layout tools for FPGAs can typically be run in two different modes: non-timing-driven and timing-driven. Non-timing-driven mode produces a solution quickly, without consideration of design performance. Timing-driven mode requires that a designer specify performance constraints and then produces a performance-optimized layout solution. The task of generating constraints is burdensome since design performance is difficult to gauge at the pre-layout stage and the relationship between the constraints supplied and tool execution time is unpredictable. In this paper, we propose a new mode for layout tools, called “automatic timing-driven” mode that produces a performance-optimized layout, without requiring any constraint specification. A key feature of this mode is a novel and practical method for automatic constraint generation that creates constraints that result in *predictable* and *controlled* layout execution time. The automatic constraint generation approach has been integrated into commercial FPGA layout tools and tuned to provide layouts having 28% better performance than non-timing-driven mode, on average. Results show that the ratio of the automatic to non-timing-driven layout execution time is consistent and predictable across a suite of designs.

1 Introduction

State-of-the-art field-programmable gate arrays (FPGAs) can implement systems with millions of gates that operate at speeds in the hundreds of megahertz. An important part of the FPGA CAD flow is the layout step, comprised of placing and routing a design on a target FPGA device. Place and route systems for FPGAs typically support two modes of operation: non-timing-driven and timing-driven. The former mode ignores delays, producing a valid placed and routed design as quickly as possible. Since delays are ignored, the resultant solution can have poor performance characteristics. In contrast, timing-driven mode typically requires that the user (designer) provide performance constraints. Layout tools then perform delay optimization in order to meet the specified constraints.

* Sudip Nag was with Xilinx when this work was conducted and is now with Flexlogics.

In this paper, we propose a new mode of operation for FPGA layout tools in which a performance-optimized layout solution is produced without the need for a user to provide performance constraints. A relevant question that arises relates to the definition of “performance”. The aspect of performance we choose for automatic optimization is a design’s clock frequency, which represents the most common optimization goal in digital circuit design. We refer to the new layout mode as *automatic timing-driven layout*.

The proposed automatic mode offers a number of benefits. First, as we will demonstrate, the layouts produced by the automatic mode have better performance than those produced by non-timing-driven mode. Often, the automatic layout performance will be sufficient to meet design requirements and no constraint specification and additional layout passes will be needed. Second, and perhaps most important, the automatic mode offers *predictable* and *controlled* layout execution time. In particular, the automatic approach can be tuned to produce performance-driven layout solutions in execution times that are consistently longer by a *pre-specified* percentage than those associated with non-timing-driven layout. Thus, its aim is to offer the best possible performance within a given execution time. This differs considerably from other automatic performance-driven FPGA layout systems, such as [1], which attempt to optimize design performance to the maximum extent possible. Rather, our approach provides a user with a reasonably good performance-optimized layout, as well as offers the run-time predictability that is a crucial part of a quality “push-button” tool experience – a primary concern for commercial FPGA tool vendors.

To optimize performance in today’s high-speed FPGAs, users employ a relatively “ad hoc” process. A trial-and-error approach is typically used for constraint generation whereby a user specifies performance constraints, executes layout tools and then analyzes the results. If the constraints are met, new and more aggressive constraints are specified and the process is repeated. On the other hand, if constraints are not met, long and unpredictable tool run-times are incurred, the constraints must be relaxed and layout tools re-executed. Each iteration of this process can take hours or days for a large design, leading to a lengthy design cycle and increased cost. An important application of the proposed automatic approach is as an initial layout pass that requires no user intervention and takes limited execution time. The resultant solution can then be used as a starting point for further optimization, leading to a reduction in the number of passes through the design flow.

One of the key contributions of this paper is a novel method for performance constraint generation that chooses design constraints in a way that is conscious of layout execution time. The approach works by doing a careful analysis of the distribution of delay slacks in circuits and produces realistic and feasible performance constraints. The aggressiveness of the generated constraints and the related layout execution time are managed elegantly through a single parameter. The proposed automatic approach has been integrated into a commercial FPGA layout system and used for performance optimization of industrial FPGA designs. The rest of this paper is organized as follows. In Section 2, we present preliminary background material and discuss related work. The automatic layout

approach is described in Section 3. Experimental results are given in Section 4. Conclusions are offered in Section 5.

2 Background

For the purpose of static timing analysis, the combinational part of a digital circuit can be represented using a timing graph, $G(V, E)$, which is a directed acyclic graph (DAG) with nodes, V , corresponding to circuit blocks and edges (connections), E , between nodes corresponding to electrical connections between blocks. Delay values are associated with each node and edge in a circuit's timing graph. Let $input(v)$ and $output(v)$ represent the predecessors and successors of a node v ($v \in V$), respectively. A node v with $input(v) = \emptyset$ corresponds to a primary input or register output. Similarly, a node v with $output(v) = \emptyset$ corresponds to a primary output or register input. The maximum performance of a layout solution (clock frequency) is limited by the delay of the longest-delay path in the timing graph, as given by:

$$T_{period} = \max_{\pi \in \Pi} \left\{ \sum_{b \in nodes(\pi)} Delay(b) + \sum_{c \in edges(\pi)} Delay(c) \right\} \quad (1)$$

where $nodes(\pi)$ and $edges(\pi)$ represents the circuit blocks and connections on a path π , respectively, and Π represents the set of *all* paths in the timing graph. An important property of a combinational path π is its *slack*, which is defined to be the difference between the path's required delay (RT_{π}) and its actual delay (AT_{π}):

$$slack(\pi) = RT_{\pi} - AT_{\pi} \quad (2)$$

The required delays for paths are generally fixed by user constraints specified prior to layout synthesis. Paths with negative slack are referred to as *critical*; such paths have excessive delay that must be reduced if performance constraints are to be met. Layout tools usually operate at the level of individual driver/load connections (edges) rather than entire paths. The slack of a connection, c , is defined to be the *minimum* slack of any path through c :

$$slack(c) = \min_{\pi \in \Pi_c} \{slack(\pi)\} \quad (3)$$

where Π_c represents the set of all paths through connection c . An early work by Hitchcock et. al. showed how path delays, path slacks and connection slacks can be computed in $O(|V|)$ time [2]. Other related work has focused on distributing the slack of a path amongst its constituent connections to yield an upper bound on the allowable delay of each connection, for use by layout tools [3,4,5].

Substantial research has been dedicated to performance-driven layout synthesis for FPGAs (e.g., [1,6,7,8]). Generally, the approach taken has been to optimize performance by giving higher priority to connections with negative slack, versus non-critical connections. The priority notion can have different meanings, depending on the context in which it is applied. In the placement phase for example, nets with negative slack connections are given priority for wirelength and/or

delay minimization, which can be estimated using metrics such as half-perimeter bounding box length. In the routing phase, steps can be taken to ensure that negative slack connections are given preference for allocation of low-delay FPGA routing resources [7].

3 Automatic Timing-Driven Layout Synthesis

In this section, we present the proposed automatic timing-driven layout approach. We first describe how performance constraints are automatically generated and subsequently, we outline how the constraint generation process is integrated into layout tools.

3.1 Constraint Generation

A straight forward way to handle automatic constraint generation is to analyze the circuit prior to placement and routing, estimate the achievable clock frequency and then perform timing-driven placement and routing with the estimated target. The problem with this approach is that the ability to predict performance at the pre-layout stage is quite limited. In FPGAs, path delay is often dominated by interconnect delay rather than logic block delay. Interconnect delays are difficult to predict and are known accurately only *after* layout is complete. In pre-layout constraint estimation/generation, if the estimated design frequency is too low, then the resultant design performance will be far from the best achievable. Conversely, if the estimated performance is too aggressive, long and unpredictable layout run-time will be incurred and the target frequency may not be feasible. To address these issues, in our approach, we dynamically adjust the performance target throughout the flow, based on the current layout status.

Our constraint generation approach is based on the crucial observation that the run-time of layout tools is highly dependent on the number of critical connections rather than the absolute performance target. We will demonstrate this assertion in our experimental results; however, it makes sense intuitively since greater numbers of critical connections imply greater competition for the FPGA resources with the best delay characteristics, leading to more complex trade-offs and increased run-time. Furthermore, critical connections must be routed in a “delay-driven” manner. Delay-driven routing is compute-intensive since it involves detailed RC delay calculations, for example using [9] or [10], to find minimum delay paths through the FPGA routing fabric from a critical connection’s source to load pin. Thus, a good constraint generation approach must be cognizant of how many connections are made critical by the constraints imposed.

The goal of tightly controlling the number of critical connections eliminates the possibility of employing a “naive” constraint generation approach, which uses an intermediate layout solution’s maximum path delay to derive a performance constraint. For example, consider the timing graphs for two circuits shown in Fig. 1(a)¹. The top of the figure shows a timing graph for a circuit with four

¹ Block delays are assumed to be zero in these examples.

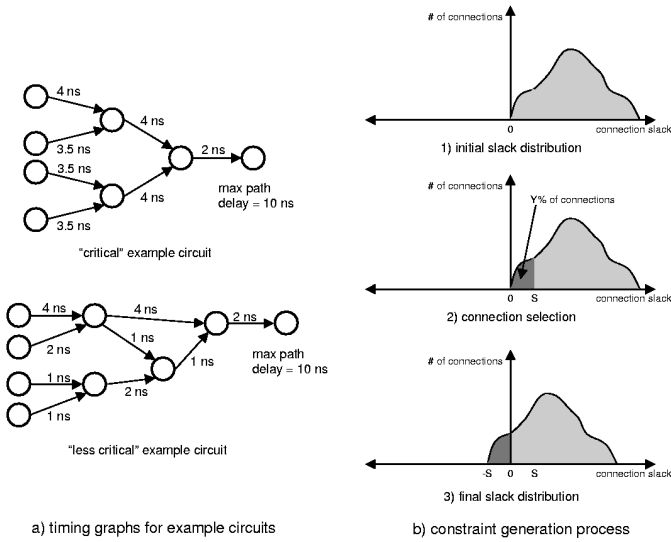


Fig. 1. (a) Timing graphs; (b) Constraint generation.

paths. The maximum path delay is 10 ns. The bottom of the figure shows a timing graph for a circuit with six paths. The maximum path delay is also 10 ns. If, for example, we generated path delay constraints by taking 90% of the maximum path delay, then the maximum allowable path delay for both circuits would be constrained to 9 ns. For the circuit in the top of Fig. 1(a), this would result in *all* its paths and connections becoming critical, since the delays of all paths in this circuit lie between 9 and 10 ns. On the other hand, a 9 ns constraint applied to the circuit in the bottom of Fig. 1(a) would result in only a single path with 3 connections being critical. Thus, the naive approach lacks control over the number of connections that become critical and is unsuitable for automatic constraint generation.

The novel aspect of our constraint generation approach is that it selects a performance target in a way that carefully manages the number of critical connections. Fig. 1(b) gives an abstract view of the constraint generation process. The input to the process is a delay for each connection in the design being optimized. At the placement stage, connection delay estimates are used; at the routing stage, accurate delays are available. Let T_{layout} be the maximum delay of any path in the current layout. We begin by computing the slacks of all connections in the circuit based on a performance constraint of T_{layout} . Part 1 of Fig. 1(b) shows the distribution of connection slacks at this stage (top of figure). The horizontal axis represents connection slack; the vertical axis represents the number of connections having a given slack. Observe that since slacks are computed based on a constraint that is equal to the current maximum path delay, there are no connections with negative slack.

Our goal is to choose a constraint that results in a specific fraction of connections becoming critical. In this example, assume we aim to make $Y\%$ of the

design’s connections critical. Larger values for Y imply increasingly hard-to-meet delay constraints and longer run-times. We analyze the slack distribution to identify the slack, S , such that $Y\%$ of connections have a slack less than or equal to S . This set of connections is represented by the dark, shaded region in part 2 of Fig. 1(b). We set the new path delay constraint, $T_{constraint}$, for the circuit as follows:

$$T_{constraint} = T_{layout} - S \quad (4)$$

The new slack distribution, based on a constraint of $T_{constraint}$, is shown in in part 3 of Fig. 1(b). Precisely $Y\%$ of connections are critical in the layout solution for constraint $T_{constraint}$. This form of constraint determination is used throughout the layout flow, with varying values for parameter Y . Note that in this discussion, we have restricted ourselves to designs having a single clock. However, extending the approach to designs with multiple clock domains is straight forward as slack distributions and period constraints can be generated independently for each clock domain in the same manner.

Observe that because our constraint generation approach is connection-based, it does not require enumerating the paths in a circuit (an operation with exponential time complexity). Rather, the proposed approach simply involves determining connection slacks using [2] and then “binning” the slacks to generate a histogram, similar to those in Fig. 1(b). Histogram generation and connection slack computation are both $O(|V|)$ operations; therefore, the complexity of the constraint generation approach is $O(|V|)$.

3.2 Integration into Layout Tools

We implemented the constraint generation approach and integrated it into a commercial FPGA layout system. The specific algorithms used in the system are proprietary; however, the constraint generation procedure described above is not limited for use with any particular layout algorithm. The placement step of the layout system proceeds in phases. Early phases optimize the design at a high-level of abstraction, permitting large changes in the placement solution. Later phases are mainly concerned with finer-grained placement refinement. Each placement phase prioritizes connections based on their delay slacks and the most critical connections are afforded preference for wirelength minimization. Prior to each placement phase, the circuit’s interconnect delays are estimated and annotated onto the timing graph. At this point, the automatic constraint generation process is invoked. We empirically determined that setting parameter Y to 3% (3% of connections are made critical) achieves our objective of producing reasonably good quality performance-optimized layout solutions in a predictable and relatively low execution time. The placement phase then commences with consideration of the newly generated constraint (and its associated connection slacks). We have observed that this approach results in successively aggressive, but realistic constraints as placement proceeds through its phases.

The routing tool operates in two phases. In the initial phase, a design’s connections are routed without timing considerations. The goal is to produce a routing solution with minimal resource usage. After initial routing, automatic

constraint generation is executed and 2% of connections are made critical. To meet the constraint, critical connections are re-routed in a delay-driven manner and their delays are reduced. If, after delay-driven routing, the constraint is met, the constraint generation process is invoked again with parameter $Y = 2\%$. This causes 2% more connections to be added to the total pool of connections that must be routed in delay-driven mode. The iterative process of successive constraint generation and delay-driven routing continues until one of two conditions is true: (1) a performance constraint that is difficult or impossible to meet is identified, or (2) a fixed iteration limit is exceeded (3 iterations). We have observed that router run-time is highly sensitive to the number of connections that must be routed in delay-driven mode. Run-time considerations are managed through condition (2), which places strict limits on the fraction of a design's connections that are permitted to become critical. Following constraint generation and delay-driven routing, the final routing phase proceeds, which refines the initial routing solution and removes any remaining infeasibilities.

One of the elegant features of our approach is that no timing constraint or performance data is passed between the various stages of the layout tool, for example, between the placer and the router. Instead, the various phases dynamically determine the performance of the current layout and a performance target is chosen accordingly. This greatly simplified the integration of the proposed approach into existing timing-driven layout tools.

A second attractive feature of the approach is that the "aggressiveness" of the constraint generation process is controlled by a single parameter (Y). As discussed above, specific values for Y have been selected for use in our layout framework. This particular tuning reflects performance results and execution times that we believe to be acceptable to users of the automatic timing-driven flow. Of course, higher (lower) values for Y will lead to better (worse) performance at the expense of longer (shorter) tool execution time. In the next section, we validate our choices for this parameter experimentally through an analysis of tool run-time and layout quality.

4 Experimental Study

To evaluate our approach, we compare it with the layout solutions that represent the extremes in the run-time/performance trade-off space. Specifically, we compare the automatic approach with two different scenarios: *non-timing-driven* layout and *best performance* layout.

In the non-timing-driven scenario, the placement and routing tools are run without a performance objective. A layout solution is generated in as little time as possible, without regard for design performance. In the best performance scenario, layout tools are run with a difficult-to-meet performance objective (clock frequency constraint), which must be selected individually for each design. We determined the performance objective for each design in an iterative manner by starting with an easy-to-meet objective and then increasing it gradually until eventually, a performance objective that could not be met was discovered. The

highest, meet-able objective was then selected as the performance objective for the best performance layout scenario.

In our experiments, we use 35 industrial benchmark circuits collected from Xilinx® customers and target a popular commercial FPGA (Xilinx Virtex™-II) [11]. The primary combinational logic element in the target FPGA is a 4-input look-up-table, which is a small memory capable of implementing any logic function requiring less than or equal to 4 inputs. FPGA logic blocks are referred to as slices; each slice contains two 4-input look-up-tables, two registers as well as arithmetic and other circuitry. The sizes of the circuits in our study range from 109 slices to 14334 slices. The FPGA’s interconnection network is comprised of variable length wire segments that connect to one another through programmable buffered switches.

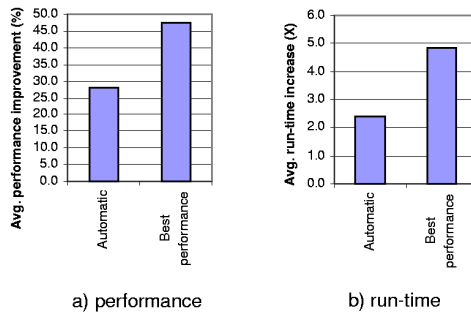


Fig. 2. (a) Performance vs. non-timing-driven; (b) run-time vs. non-timing-driven.

4.1 Experimental Results

We begin by summarizing the performance of the three layout solutions. Fig. 2(a) shows the average percentage improvement in clock frequency for the automatic approach and the best performance solution versus the non-timing-driven layout solution. The average performance improvement offered by the automatic solution is about 28%. The best performance layout solutions have clock frequencies that are 48% faster than the non-timing-driven solutions, on average. The results underscore the huge benefits of timing-driven layout: the performance improvements over non-timing-driven amount to the equivalent of several speed grades. Note that the disparity in performance between the automatic and best performance layouts simply reflects our tuning preferences, described in Section 3.2. Below, we show that other tunings are also possible.

Fig. 2(b) summarizes the run-time results. The average increase in the layout tool’s run-time is shown for the automatic solution and the best performance solution, in comparison with the non-timing-driven solution. Observe that the run-time hit associated with both forms of performance-driven optimization is considerable. On average, the run-time of automatic timing-driven layout synthesis is about 2.4 times (X) longer than the run-time for non-timing-driven

layout. Producing the layout solution with the best performance takes 4.9 times longer, on average, than non-timing-driven layout.

Table 1 gives detailed performance results for a subset of the circuits used in the experiments. Columns 2 through 4 of the table present performance data. The clock frequency for each circuit is shown in megahertz; the percentage improvement versus the non-timing-driven layout solution is shown in parentheses. Observe that the performance gap between the automatic and best performance solution is fairly design dependent. For example, for the circuit *industry2*, the automatic solution performance is within 5% of the best performance result. Conversely, for the circuit *industry5*, the automatic solution performance is 31% better than non-timing-driven performance, and the best performance solution is superior by 68% to the non-timing-driven solution. The variability in the performance gap between the automatic and the best performance solutions across the design suite is explained by considering the approach taken to automatic constraint generation. Performance constraints are generated based on making a specific fraction of a design’s connections critical rather than on the basis of a design’s maximum potential performance.

Table 1. Performance results for individual circuits.

Circuit	Non-timing-driven (MHz)	Automatic (MHz) (%)	Best perf. (MHz) (%)
industry1	79.9	110.7 (38.5)	127.7 (59.8)
industry2	111.9	149.8 (33.9)	156.9 (40.2)
industry3	167.1	174.2 (4.2)	190.4 (13.9)
industry4	92.7	111.3 (20.1)	116.3 (25.5)
industry5	109.2	142.7 (30.7)	183.9 (68.4)
industry6	88.8	150.3 (69.3)	162.8 (83.3)
industry7	84.3	110.1 (30.6)	146.4 (73.7)
industry8	133.4	179.3 (34.4)	201.1 (50.7)
industry9	158.5	182.2 (15.0)	198.8 (25.4)
industry10	56.7	89.9 (58.6)	97.9 (72.7)
industry11	73.9	100.4 (35.9)	113 (52.9)
industry12	26.4	32.3 (22.3)	35.5 (34.5)
industry13	60.1	70.9 (18.0)	83.1 (38.3)
industry14	123.2	153.8 (24.8)	179 (45.3)
industry15	112.5	133.3 (18.5)	156.8 (39.4)
	Avg. % impr. (these circuits)	30.3%	48.3%
	Avg. % impr. (all circuits)	28.0%	47.50%

Figs. 3(a) and (b) show the run-time results for the individual circuits and demonstrate a key benefit of our approach: execution time predictability. Fig. 3(a) gives results for the best performance layout runs; Fig. 3(b) gives results for the automatic timing-driven runs. Each point in these figures represents the run-time increase (vs. non-timing-driven layout run-time) for one of the 35 benchmark circuits. Fig. 3(a) shows that the time needed to generate the layout solution with the best performance is highly variable and strongly design dependent. The standard deviation in run-time for this case is 2.03X the non-timing-driven run-time. In addition to the variability apparent in Fig. 3(a), the number of layout runs needed to determine the best performance (as described above) was also variable, and generally involved between 3 and 10 iterations of layout execution and constraint tightening. In contrast, Fig. 3(b) clearly illustrates the

power of the proposed constraint generator to deliver performance-driven layouts in a *predictable* amount of execution time. For the automatic timing-driven runs, the standard deviation in run-time is 0.4X the non-timing-driven run-time. Predictable run-time strongly influences user perception, making the proposed approach well-suited for use in a push-button FPGA layout tool.

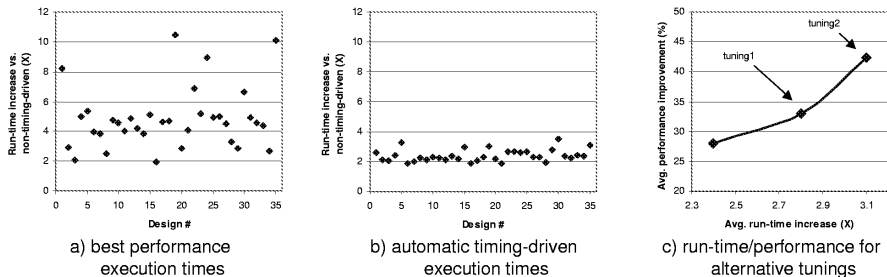


Fig. 3. (a),(b) Run-time results; (c) Results for alternative tunings.

For the results presented so far, the automatic timing-driven solution was tuned based on our desire to provide layout solutions with reasonably good performance within a consistent and relatively low execution time. Here however, we demonstrate that by increasing the aggressiveness (parameter Y) of the constraint generation, we can tune the automatic approach to produce solutions that represent other points in run-time/performance trade-off space. We investigated more aggressive constraint generation and the results are shown in Fig. 3(c) for two alternative tunings, labeled *tuning1* and *tuning2*. For *tuning1*, parameter Y was increased to 10% for constraint generation in placement. For *tuning2*, parameter Y was increased to 20% for placement and 3% for routing. The bottom-left data point in Fig. 3(c) reflects the results presented above. The data in Fig. 3(c) demonstrates the effectiveness with which parameter Y controls layout performance and run-time. Observe that the performance characteristics for *tuning2* layout solutions are quite close to those of the best performance layout solutions studied above, but require considerably less run-time. From this, we conclude that the *extra* run-time needed to move from a “close to” best performance layout to a “true” best performance layout is substantial.

5 Conclusions

Performance-driven layout synthesis is a mandatory component of modern high-speed FPGA design. A key task in the design process is that of determining appropriate performance objectives to supply to layout tools as constraints. In this paper, we presented a new approach to FPGA layout synthesis, called automatic timing-driven layout. The proposed layout approach produces performance optimized layout solutions, without requiring constraints to be specified. Constraints are generated automatically as layout progresses, in a manner that results in a specific fraction of a circuit’s connections becoming delay critical.

This method permits performance-optimized layout solutions to be generated, while providing predictable layout execution time. The approach has been integrated into a commercial FPGA layout tool, where it has been tuned to produce solutions having a 28% performance advantage (on average) over non-timing-driven layout solutions, in execution times that are predictably 2.4 times that required for generating a non-timing-driven layout.

Acknowledgements. The authors thank Walt Manaker, Tom Wurtz and Nick Mezei for their helpful comments and their contributions to the implementation.

References

1. Betz, V., Rose, J.: VPR: A new packing, placement and routing tool for FPGA research. In: Proc. of FPL. (1997) 213–222
2. Hitchcock, R.B., Smith, G.L., Cheng, D.D.: Timing analysis of computer hardware. *IBM Jour. of Research and Development* **26** (1982) 100–105
3. Nair, R., Berman, C.L., Hauge, P.S., Yoffa, E.J.: Generation of performance constraints for layout. *IEEE Transactions on CAD* **8** (1989) 860–874
4. Youssef, H., Shragowitz, E.: Timing constraints for correct performance. In: Proc. of IEEE/ACM ICCAD. (1990) 24–27
5. Frankle, J.: Iterative and adaptive slack allocation for performance-driven layout and FPGA routing. In: Proc. of ACM/IEEE DAC. (1992) 536–542
6. Marquardt, A., Betz, V., Rose, J.: Timing-driven placement for FPGAs. In: Proc. of ACM/SIGDA FPGA. (2000) 203–213
7. McMurchie, L., Ebeling, C.: Pathfinder: A negotiation-based performance-driven router for FPGAs. In: Proc. of ACM/SIGDA FPGA. (1995) 111–117
8. Nag, S., Rutenbar, R.: Performance-driven simultaneous place and router for row-based FPGAs. In: Proc. of IEEE/ACM ICCAD. (1995) 332–338
9. Elmore, W.: The transient response of damped linear networks with particular regard to wideband amplifiers. *Jour. of Applied Physics* **19** (1948) 55–62
10. Rubinstein, J., Penfield, P., Horowitz, M.: Signal delay in RC tree networks. *IEEE Transactions on CAD* **2** (1983) 202–211
11. Xilinx: Virtex-II FPGA Data Book. (2004)