Analyzing and Predicting the Impact of CAD Algorithm Noise on FPGA Speed Performance and Power

Warren Shum and Jason H. Anderson
Dept. of ECE, University of Toronto, Toronto, ON, Canada
{shumwarr,janders}@eecg.toronto.edu

ABSTRACT

FPGA CAD algorithms are heuristic, and generally make use of cost functions to gauge the value of one potential circuit implementation over another. At times, such algorithms must decide between two or more implementation options of apparently equal cost. This work explores the variations in circuit quality, i.e. *noise*, that arise when CAD algorithms are altered to choose randomly when faced with such equal-cost alternatives. Noise sources are identified in logic synthesis and technology mapping algorithms, and experimental results are presented which show standard deviations of 3.3% and 3.7% from the mean in post-routed delay and power. As a means of dealing with this variation, early timing and power prediction metrics can be applied after technology mapping to find the best circuits in the presence of noise. When applied to designs with over 1.5% variation in delay and power, the best prediction models have a 40% probability of capturing the best circuit when predicting the top 10% of circuits in a group of noise-injected circuits.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids

Keywords

FPGAs, CAD, noise, prediction

1. INTRODUCTION

The typical FPGA CAD flow proceeds in a series of steps, with different heuristic algorithms being used in each step. In such algorithms, it is common to encounter situations where a choice must be made between two (or more) alternatives that appear to have identical quality. For example, in logic synthesis, a logic function might be implemented in multiple ways, each having the same estimated area, delay, or power. However, the choice of how that function is implemented may affect the post-routed circuit delay or power in an unpredictable way. In practice, choices between alternatives are arbitrarily made (e.g. always select the first alternative) or are controlled with the use of a random number generator. By running an algorithm multiple times using different seeds for the random number generator, one can obtain a set of circuits with different characteristics. The variation in circuit quality (area, performance, power) through seemingly neutral changes in CAD algorithms is what we call algorithmic **noise**.

The practice of executing CAD algorithms with multiple seeds in the hope of producing a higher-quality implementation is well-established for placement and routing [3, 14]. In this work, we show that noise also exists earlier in the CAD flow: in logic synthesis and technology mapping. By exposing the noise in these earlier stages, we allow seed sweeping to take place earlier, in less time-consuming stages.

This work makes the following contributions:

- An investigation of noise sources in logic synthesis and technology mapping.
- Experimental data showing the amount of power and performance noise present in CAD algorithms, using a commercial FPGA architecture.
- Timing and power prediction metrics for early identification of the best circuits in the presence of noise.

2. NOISE INJECTION IN FPGA CAD

This work focuses on the logic synthesis and technology mapping stages. In particular, we inject noise into the algorithms implemented in the academic tool, ABC [4].

2.1 Logic Synthesis

The primary data structure in ABC is an `AND`-Inverter Graph (AIG) which is a representation of a logic circuit using two-input `AND` gates and inverters. The general goal of logic synthesis algorithms is to reduce the number of nodes in the AIG and the AIG's depth (i.e. the number of logic levels from any combinational input to a combinational output). We use the *resyn2* script in ABC for technology independent logic synthesis. The script loops repeatedly through three synthesis algorithms, each of which contains an opportunity for noise injection. The algorithms are as follows:

AIG balancing: Balancing is a technique that aims to reduce the number of levels in an AIG [10]. It is done in two main steps. The **tree covering** step identifies multi-input `AND` gates in the AIG. The second step is **tree balancing**. For each multi-input `AND` gate identified by the tree covering stage, the tree balancing stage decomposes it into a balanced tree of two-input `AND` gates.

By default, ABC does AIG balancing in a deterministic way, making the same (arbitrary) balancing decisions every time. However, there are multiple ways to perform the balancing. To see this, consider the example in Fig. 1. With different random choices, balancing might produce either of the AIG subgraphs shown. The tree balancing stage may not have a unique optimal solution, and it is therefore a source of noise. We modified the algorithm to choose randomly in scenarios where there exist two or more equal-depth options.

AIG rewriting: AIG rewriting is an algorithm that reduces the number of nodes/logic levels in an AIG by examining subgraphs of nodes and replacing them with lower-cost

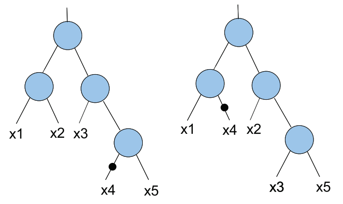

Figure 1: Examples of balanced AIGs.

substitutes [11]. The rewriting algorithm walks through the AIG and for each node n enumerates all 4-input cuts of n. A *cut* of a node n is a set of nodes (*leaves*) such that each path from a primary input to n passes through at least one node of the cut. Each cut is replaced with equivalent AIG subgraphs from a hash table of pre-computed "good" subgraphs. If the subgraph replacement candidate leads to a reduction in AIG nodes, it is accepted.

To add randomness at this stage, we modified the algorithm to allow changes even when the replacement leads to no change in the AIG node count (a *zero-cost* replacement). In our modified ABC implementation, a candidate zero-cost replacement is accepted with a 50% probability.

AIG refactoring: This technique involves computing one large cut for each AIG node, then replacing it with a factored form with fewer nodes [9]. The cuts are chosen based on how much reconvergence they contain, which is an indicator of redundancy that can be exploited by refactoring. Refactoring differs from rewriting in that it acts on a larger scale (refactoring can handle up to 16-input cuts). The noise injection in this stage is similar to the method used in AIG rewriting.

2.2 Technology Mapping

We use the priority cut-based technology mapping algorithm in ABC [12] to map the AIG logic into K-input LUTs (K-LUTs). The mapper does this by first evaluating a set of cuts for each node in an AIG, representing potential LUT implementations of that node and a subset of its predecessors in the AIG. The cuts are selected and sorted in terms of delay, number of inputs, and area. FPGA technology mapping algorithms use logic depth as a proxy for delay, and use the number of LUTs as a proxy for area.

The priority cuts for each node are evaluated based on several criteria, depending on the mapping parameters. These criteria include depth and cut size. The sorting makes use of a cut comparison function which compares two cuts. We introduce random noise in this stage when deciding between cuts with the same values for each of the specified metrics. If the cuts are tied for each of these metrics, our modification to the algorithm makes a random selection between them.

The mapping algorithm [12] makes several passes over the netlist, stitching together the best results (using depth-optimal mappings on critical paths, and area-oriented mappings elsewhere). We introduce noise in both types of passes: depth-oriented mapping and area-oriented mapping.

3. NOISE ANALYSIS

In order to evaluate noise over many random seeds, a large number of circuit compilations (over 10,000) were executed using a high-performance cloud computing system [5]. We use Altera's Quartus 10.1 to pack, place and route the circuits and perform timing and power analysis. The target FPGA family is Altera Stratix III [2]. Altera's QUIP (Quartus University Interface Program) was used to bring designs from ABC into Quartus II [1]. Modelsim 6.3e was used for simulation to get toggle rates for the power estimation, done

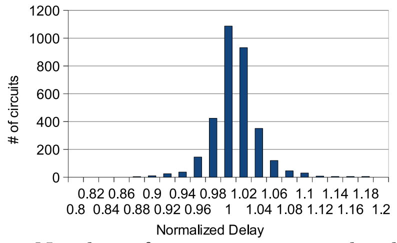

Figure 2: Number of circuits vs. normalized delay.

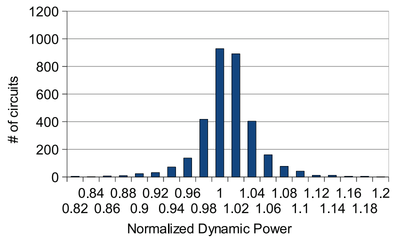

Figure 3: Number of circuits vs. normalized power.

using Quartus PowerPlay. 5,000 random input vectors were applied to each circuit. Placement and routing were performed using the *standard fit* setting (maximum effort) in Quartus II. The critical path delay was obtained using the TimeQuest timing analyzer. The benchmark set consists of 20 circuits from the MCNC benchmark set and 7 circuits from the VPR 5.0 benchmark set, in order to have data for some larger circuits. For the following sections, the word **design** will be used to refer to all circuits having the same original source file (e.g. '`alu4`' is a design). A **circuit** will refer to a particular compilation of the design using certain seeds (e.g. '`alu4`' compiled with synthesis seed 1 and mapping seed 2 is a circuit).

We begin by showing the amount of noise present when all noise sources are activated (balancing, rewriting, refactoring, delay and area-oriented technology mapping, and placement). Five seeds are used in each of synthesis, mapping and placement for 27 designs, making a total of $5 \cdot 5 \cdot 5 \cdot 27 = 3,375$ circuits. For these experiments, circuits with logic depth greater than the minimum depth observed for that design were removed. This is because we would like to find ways of differentiating between minimal-depth mappings, which are likely to be selected over non-minimal-depth ones.

Fig. 2 shows the number of compiled circuits vs. their critical path delay, normalized to the average for each design. Analogous power results are given in Fig. 3. The results appear as a roughly normal distribution, with the power noise being slightly greater than delay noise. The standard deviation of critical path delay is 3.3% and dynamic power is 3.7%. This is a fairly significant amount, enough to drive the use of seed sweeping to find the circuit compilations with the best results in this distribution.

We now describe our effort to isolate the effect of logic synthesis noise. 25 synthesis seeds were used for each design. All synthesis noise sources were activated (balancing, rewriting, and refactoring). For each circuit, the delay and power were averaged across five Quartus compilations using different place-and-route seeds. This was done in order to reduce the impact of placement and routing noise, in an attempt to isolate the synthesis noise.

The standard deviations of delay and power due to synthesis noise alone are 1.8% and 2.7% for delay and power, respectively. This indicates the degree to which random, zero-cost changes in the logic synthesis stage affect the overall quality of the circuit. For technology mapping, the standard deviations are 0.9% and 1.4% for delay and power, respectively, which is less than in synthesis. This is likely due to good tiebreaking mechanisms in the mapper, as well as the fact that there are fewer downstream CAD stages to be affected by noise introduced in mapping.

4. EARLY DELAY/POWER PREDICTION

While engineers commonly perform multiple placement and routing compilations using different seeds to find a high quality implementation of their design, it is a long process, taking hours or even days for the largest designs [7]. Synthesis and mapping, on the other hand, are relatively quick, leading us to question whether we can sweep seeds in the synthesis and mapping stages instead of in placement and routing. This would require early timing and power prediction metrics at the post-mapping stage, in order to find the best candidate circuits for placement and routing.

Early power and performance estimation has been done at various stages of the CAD flow. At the high-level synthesis stage, power estimation has been done to drive low-power resource allocation and binding techniques [6]. At the preplacement stage, work has been done to predict interconnect wirelength and delay [8, 13]. The work by Manohararajah et al. [8] proposed a simple timing model based on using a single delay value for each connection depending on its source and destination node type and port (e.g. logic, I/O, memory). The work by Pandit and Akoglu [13] attempts to estimate wirelengths using structural metrics at the preplacement stage – metrics taken from works in the ASIC domain and applied to FPGAs.

4.1 Delay Prediction

We examine ways to predict the circuit with the lowest critical path delay at the post-mapping stage, given a set of circuits compiled with different synthesis/mapping seeds. Our delay prediction model assigns each node (LUT) a certain delay, then traverses the circuit graph in topological order and computes the arrival time at each node. We examine numerous timing models, sweeping several parameters.

4.1.1 Varying Pin Delays

Due to the tree-like structure of the multiplexer within a LUT, the delay from each of its input pins to its output pin varies. In general, commercial FPGA tools automatically assign the slowest-arriving inputs to the LUT input pins with smaller delays. On this front, we investigated two timing prediction models. The **pin utilization** model bases the LUT delay of the number of used input pins on the LUT. A K-LUT with n used inputs is assigned a delay of n/K. The **pin order** model bases the LUT delay on a prediction of the ordering of the input pins, which is based on their estimated arrival times. Specifically, the model assumes that the i^{th}-fastest input pin on a K-LUT has a delay of i/K and that the latest arriving input signal is assigned to the fastest LUT input pin; the next-latest arriving input is assigned to the second-fastest LUT input, and so on.

4.1.2 Logic, Routing and Constant Factors

The overall delay model for a LUT n is as follows:

$$\begin{aligned} Delay(n) =& const + logic\_factor \cdot logic\_delay(n) \\ & + fanout\_factor \cdot fanout(n) \end{aligned} \tag{1}$$

- *const*: A constant value for each LUT, which can be set to 0 or 1. If set to 1, it represents a unit delay for the LUT.
- $logic\_factor$: A scaling factor for the LUT delay, $logic\_delay$, calculated using one of the pin-based timing models above. We examined factors ranging from 0 to 5.
- $fanout\_factor$: A scaling factor for the fanout of the node, which represents the routing delay. We examined values from 0 to 5. Fanout was capped at 10 to prevent high-fanout nodes from dominating the delay.

4.1.3 Maximum/Scaled Metrics

A logic circuit contains a single (or a few) critical path(s). Early in the CAD flow, where delay estimates are used, it is difficult to predict the paths that will be critical at the post-routing stage. Therefore, we propose a scaled model in which we take the sum of the top L maximum arrival times of delay paths in a circuit, each scaled by an exponentially decaying factor. The scaled delay model for a circuit is expressed as:

$$scaled\_delay = \sum_{i=1}^{L} max\_arrival\_time(node_i) \cdot factor^i \tag{2}$$

where $max\_arrival\_time(node_i)$ is the maximum *total* delay to reach the node with the i^{th} latest arrival time, and $0 < factor < 1$. For our experiments, a factor of 0.95 was chosen empirically since it decays quickly enough to ignore path endpoints that are unlikely to be critical, yet not so quickly that it considers only a few.

4.2 Power Prediction

The dynamic power consumption of a circuit can be calculated as:

$$P_{dyn} = \frac{1}{2} \sum_{m=1}^{M} S(m) \cdot C(m) \cdot f \cdot V_{dd}^2 \tag{3}$$

where M is the number of nets in the circuit, $S(m)$ is the switching activity of a net m, $C(m)$ is the capacitance of net m, f is the frequency of the circuit, and V_{dd} is the supply voltage. The switching activity of a net is estimated using a fast vector simulation implemented in ABC with 1000 random input vectors. The simulator can be run in two modes:

- **Zero delay**, in which the LUTs/wires are assumed to have zero delay (i.e. on each clock cycle, all signals immediately settle into their final state).
- **Unit delay**, in which each LUT is assumed to have a delay of one. This allows for some modeling of glitches.

The overall power model is a product of switching activity and fanout over all nets:

$$P_{est} = \sum_{m=1}^{M} S(m) \cdot fanout(m) \tag{4}$$

Fanout is used as a substitute for capacitance (capped as in Section 4.1.2). V_{dd} and f from (3) are ignored since they are constant for each circuit.

5. RESULTS

We ran our prediction methods on the noise-injected circuits from Section 3. Each of the 27 benchmark designs was synthesized and mapped with different seeds to create 25 mapped candidates. Each candidate was placed and routed using Quartus using 5 different seeds. The results were averaged across the 5 placement seeds to obtain a representative average result for each mapped candidate circuit. The prediction algorithns were all implemented in ABC [4]. For each mapped candidate circuit, all prediction models were run, sweeping all parameter combinations. Designs with low swing (no circuit with more than 1.5% deviation from the average) were ignored. For the delay portion of this study, the designs were arbitrarily split into training and test sets.

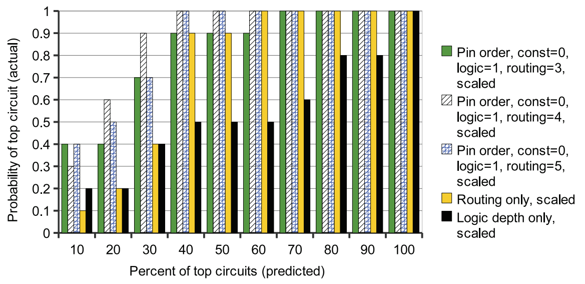

Figure 4: Probability of finding the top circuit vs. percentage of top modeled circuits considered (delay).

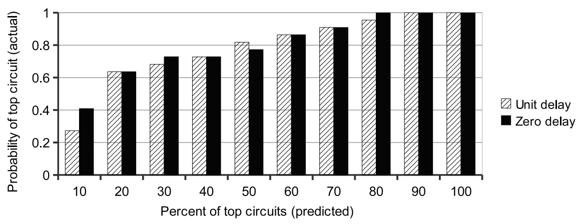

Figure 5: Probability of finding the top circuit vs. percentage of top modeled circuits considered (power).

Model parameters were chosen using data from the training design set; model predictive accuracy was evaluated using data from the test designs.

We first consider results for delay prediction. Fig. 4 shows the probability of finding the best circuit implementation in terms of delay (or one within a 0.1% margin of it). The top three models (based on the ranking of the top predicted circuit for each design) are shown, along with two simple models: "Routing only" ($fanout\_factor = 1$ in Eqn. 1, all others 0, scaled) and "Logic depth only" ($const = 1$, all others 0, scaled). The top models were chosen by taking the best *predicted* circuit for each design and summing its *actual* ranking (best=1, worst=25) across all designs. The models with the lowest sums were chosen as the best ones. The x-axis shows the percentage of the top circuits (by model score) considered, while the y-axis shows the probability of finding the actual top circuit within this group. The legend shows the timing model used.

The best model was found to be "Pin order, const=0, logic=1, routing=3, scaled" which refers to the pin order model (Section 4.1.1) with ($const = 0, logic\_factor = 1$, and $fanout\_factor = 3$) as the factor settings (Section 4.1.2), and the exponentially decaying scaling factor (Section 4.1.3). Using this model, if we take the top 10% of circuits (according to the model) we have approximately a 40% chance of selecting the actual best one (i.e. post-routing best).

Fig. 5 shows analogous results for power. "Zero delay" and "Unit delay" represent the zero delay and unit delay simulation models. Both the unit and zero delay models appear to offer similar results. Simply using a zero delay simulation and taking the top 10% of predicted circuits, the probability of capturing the top circuit is over 40%.

Table 1: Average benefit of prediction models.

Delay		
Model	% improv.	% improv. (full)
Best prediction	1.3	1.8
Logic depth only	0.6	0.3
Fanout only	0.9	0.9
Power		
Model	% improv.	
Unit delay	1.4	
Zero delay	1.1	

Table 1 shows the average delay/power savings from our prediction models if the top 10% of predicted best circuits are carried forward to placement and routing. Looking at the "% improv." column, we see that the best prediction model gives an average benefit of 1.3% in post-routed delay and 1.4% in power (compared to the average delay/power for a design) – this is more than the benefit offered by the simpler models of logic depth and fanout. It should be noted that these results were generated using relatively small training and test sets (for delay). If we instead use the entire benchmark set for *both* training and test, we can obtain improvements of up to 1.8% in delay (column "% improv. (full)").

6. CONCLUSIONS

In this paper, we studied FPGA CAD algorithm noise – variations in circuit quality due to cost-neutral changes in the compilation flow. We identified noise sources in logic synthesis and technology mapping algorithms, and we described our method of noise injection in those algorithms. Under the influence of synthesis noise, standard deviations of critical path delay and dynamic power were 1.8% and 2.7%, respectively, while the results for technology mapping were 0.9% and 1.4%, respectively. Under the influence of noise in all CAD stages, the standard deviations were 3.3% in delay and 3.7% in power.

We evaluated ways of predicting the impact of noise on delay and power at the post-technology mapping stage. Results show that the circuits predicted to be in the top 10% at the technology mapping stage in terms of delay or power, have a 40% probability of being the best circuit at the post-routing stage.

7. REFERENCES

[1] Altera Corp. Quartus university interface program. `www.altera.com/education/univ/research/quip/unv-quip.html`, 2009.

[2] Altera Corp. Stratix III device handbook. `http://www.altera.com/literature/lit-stx3.jsp`, 2011.

[3] Altera Corp. Timing closure methodology for advanced FPGA designs. `www.altera.com/literature/an/an584.pdf`, 2011.

[4] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. `www.eecs.berkeley.edu/~alanmi/abc/`, Release 00406.

[5] C. Loken et al. SciNet: Lessons learned from building a power-efficient Top-20 system and data centre. *Journal of Physics: Conference Series*, 256(1):012026, 2010.

[6] D. Chen, J. Cong, Y. Fan, and Z. Zhang. High-level power estimation and low-power design space exploration for FPGAs. In *IEEE/ACM ASP-DAC*, pages 529 –534, 2007.

[7] M. Gort and J. Anderson. Deterministic multi-core parallel routing for FPGAs. In *IEEE FPT*, pages 78 –86, 2010.

[8] V. Manohararajah, G. Chiu, D. Singh, and S. Brown. Predicting interconnect delay for physical synthesis in a FPGA CAD flow. *IEEE TVLSI*, 15(8):895 –903, Aug. 2007.

[9] A. Mishchenko and R. Brayton. Scalable logic synthesis using a simple circuit structure. In *Proc. IWLS*, pages 15–22, 2006.

[10] A. Mishchenko, R. Brayton, S. Jang, and V. Kravets. Delay optimization using SOP balancing. In *Proc. IWLS*, pages 75–82, 2011.

[11] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In *ACM/IEEE DAC*, pages 532 –535, 2006.

[12] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *IEEE/ACM ICCAD*, pages 354 –361, 2007.

[13] A. Pandit and A. Akoglu. Wirelength prediction for FPGAs. In *IEEE FPL*, pages 749 –752, 2007.

[14] R. Y. Rubin and A. M. DeHon. Timing-driven pathfinder pathology and remediation: quantifying and reducing delay noise in VPR-pathfinder. In *ACM FPGA*, pages 173–176, 2011.