

LUT-Based FPGA Technology Mapping for Reliability

Jason Cong and Kirill Minkovich

Computer Science Department, University of California, Los Angeles
Los Angeles, CA 90095, USA
{cong, cory_m}@cs.ucla.edu

ABSTRACT

As device size shrinks to the nanometer range, FPGAs are increasingly prone to manufacturing defects. We anticipate that the ability to tolerate multiple defects will be very important at 45nm and beyond. One common defect point is in the lookup table (LUT) configuration bits, which are crucial to the correct operation of FPGAs. In this work we will present an error analysis technique that is able to efficiently calculate the number of critical bits needed to implement each LUT. We will perform this analysis using a scalable overlapping window-based method called DCOW (Don't-care Computation with Overlapping Windows), which allows for accurate and efficient don't-care lower bound calculations. This new windowing technique can approximate the complete don't cares within 2.34%, and can be used for many logic synthesis operations. In particular, we apply DCOW to our FPGA mapping algorithm to reduce the number of possible faults. This will allow the design to have a much higher success of functioning correctly when implemented on a faulty FPGA. By using our algorithm, we are able to reduce the number of possible faults by more than 12% with no area increase.

Categories and Subject Descriptors

B.6.3 [Hardware]: Logic Design – Optimization

General Terms

Algorithms, Measurement, Design, Reliability, Experimentation.

Keywords

Error Analysis, Windowing, Don't Cares, Logic Synthesis, Technology Mapping, FPGA Lookup Table.

1. INTRODUCTION

As device size shrinks to the nanometer range, device failures are almost guaranteed. Circuits based on nanotechnology are expected to outperform current technology in terms of density (area), power consumption, and computation speed. These circuits will not only more susceptible to errors, but will also have high defect rates. Systems employing nano components will presumably have to deal with non-negligible error rates [1], [2], and [7]. The error effects can be corrected using techniques such as hardware/time redundancy [16], error-correcting information encoding [8] and [19], software-based fault tolerance [20], or combinations thereof [15] and [18].

Even at the current device size, the defect rate is significant, and it would be extremely useful to have a secondary use for the defective chips. One approach to using partly defective FPGAs is

to choose designs that avoid the defects. This approach is called the Application-Specific FPGA (ASFPGA) and is provided by Xilinx in their EasyPath program [23]. The idea is that the designer creates a bitstream as usual, and then Xilinx selects defective FPGAs that are compatible with the given bitstream. The advantage here is that there is no need for any change in the FPGA architecture, the tool chain, the final bitstream, or methods used when developing the design.

Currently, algorithms for improving reliability [6], [9], [17] are only evaluated using Monte Carlo simulation where a single new fault is injected for each set of inputs. This simulation attempts to evaluate whether a fault will get masked, and in turn these reliability improvement algorithms try to reduce the detectability of an error. However, reducing the detectability of a fault does not guarantee the prevention of the fault; i.e., just because a fault is highly unlikely to be detected, it does not mean that it will not occur. For most applications, a circuit which has a fault that only appears every 10,000 clock cycles is just as broken as a circuit that has a fault every clock cycle in the sense that both circuits are not functional. This is especially true when it comes to optimizing FPGA configuration bit errors, as in [6] and [9], since it is very unlikely that a configuration bit error will correct itself.

In this work we will show that we can efficiently determine which bits of the bitstream are critical, and we can efficiently map the designs to significantly reduce the number of critical configuration bits (defined in the next section). First, in Sections 2 we will present the background and problem formulation. Then, in Section 3 we will present a study of how configuration bit errors propagate in LUT-based FPGAs. This will show that most errors can be captured using a windowing technique. In Section 4 we present a novel windowing technique, DCOW (Don't-care Computation with Overlapping Windows), that can efficiently estimate the complete don't care (DC) set for each node in the circuit. In Section 5 we will use DCOW to create a technology mapper that can improve the reliability of a circuit. The results from the technology mapper will be presented in Section 6.

2. BACKGROUND AND PROBLEM FORMULATION

There are several important factors involved in determining how errors in the configuration bits can be observed on the outputs. For an error to be detectable, it has to be both controllable and observable. The controllability set [11] (whose complement is the controllability don't care set (CDC)) represents all the input patterns that are produced by the environment. The observability set [11] (whose complement is the observability don't care set (ODC)) consists of all the input patterns that represent situations where an output is observed by the environment. Combining the CDC and the ODC sets together determines the complete set of don't cares (DC). This combined set can be used to determine which configuration bits are not critical. A LUT configuration bit is a *critical bit* if the corresponding LUT input pattern is controllable,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC'10, June 13–18, 2010, Anaheim, California, USA

Copyright 2010 ACM 978-1-4503-0002-5 /10/06...\$10.00

and the inversion of this bit can lead to an observable error on at least one of the circuit outputs.

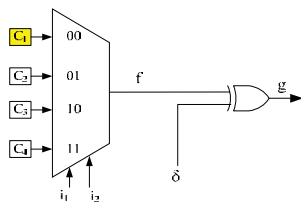


Figure 1. Configuration bits of a 2-LUT FPGA

Let us examine Figure 1 to see how controllability and observability affect the configuration bits of an FPGA. For an error on configuration bit C_1 in Figure 1 to be controllable, there has to exist an input to the circuit which results in $i_1=0$ and $i_2=0$. For an error on configuration bit C_1 to be observable at the circuit outputs, f should first be replaced by $f \oplus \delta$ to test both the correct ($\delta=0$) and faulty functioning ($\delta=1$) of this LUT. Therefore, an input to the circuit has to exist which results in $i_1=0$ and $i_2=0$, while at the same time toggling δ leads to a toggling of at least one primary output.

In our work we present an effective mapping algorithm to improve the probability that a design will function correctly when implemented on a faulty FPGA. More concretely, we can formulate the FPGA reliability improvement problem as follows:

Problem: Given a circuit, use technology mapping to improve reliability by reducing the number of critical bits in the LUT configuration bits used to implement the circuit. The number of critical bits provides a way to estimate the likelihood that a design will function correctly when implemented on a faulty FPGA, in which some configuration bits are faulty due to transient and permanent errors. As in [6] and [9], in this paper, we address only FPGA LUT configuration bit errors, but we plan to address interconnect errors in our future work.

3. MOTIVATION FOR WINDOWING

To be able to quickly estimate controllability and observability of configuration bit errors, it is important to analyze how they relate to different properties in the circuit. To evaluate these properties, we simulated a fault on every configuration bit of MCNC circuits with less than 25 combination logic inputs (latches were treated as primary inputs). The restriction on the number of inputs allowed every possible circuit input to be simulated. The set consisted of the following 8 circuits: alu4, apex4, mixex3, pdc, s298, spla, ex1010, and ex5p. These benchmarks were each mapped depth optimally to the 6-LUT architecture using the ABC FPGA mapper, “fpga” [13]. ABC performs mapping by first converting the design into an AND-INV (AIG) graph, and then performs several iterations of cut selection (depth-optimal followed by area recovery).

We first analyzed how many nodes can be affected by a configuration bit error. The data in Figure 2 shows that the errors do not propagate very far. Over 82% of the time, the error propagates to ten or fewer nodes. This means it is possible to correctly calculate (82% of the time) the observability by only examining ten of the transitive fanout nodes. The transitive fanout (TFO) node n can be defined recursively (PO to PI) using $TFO(n) = \{n\} \cup (\bigcup_{f \in fanouts(n)} TFO(f))$, where $fanouts(n)$ are the fanouts of n . Similarly, the transitive fanin (TFI) of node n can be defined recursively (PI to PO) using $TFI(n) = \{n\} \cup (\bigcup_{f \in fanins(n)} TFI(f))$, where $fanins(n)$ are the fanins of n .

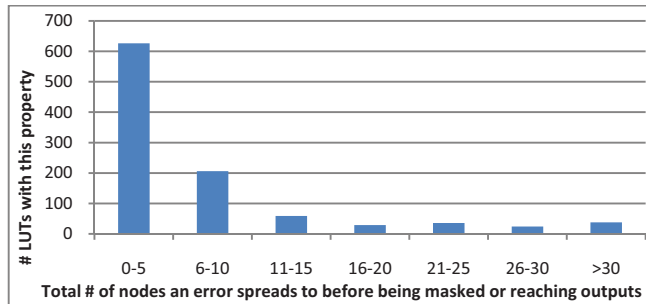


Figure 2. Error spreading count

When the error spreads to a lot of nodes, it is useful to see if it also spreads to the outputs and becomes observable. Figure 3 shows how far the error spreads compared to the amount of masking. In this figure, each dot represents a LUT. The x-coordinate represents the percent of critical configuration bits of this LUT that are *not* masked, and the y-coordinate represents the total number of other LUTs reached by all possible configuration errors in this LUT. We can see that if the error spreads to more than ten LUTs, then it is likely that all the corresponding configuration bits are critical.

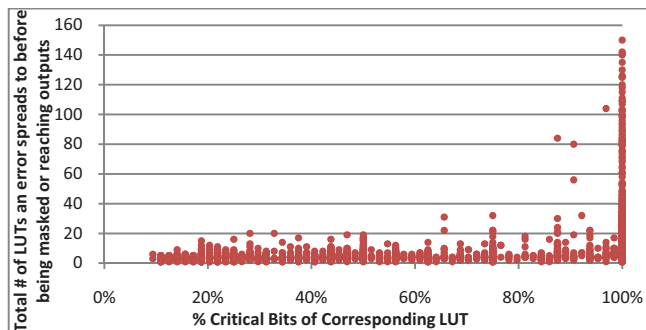


Figure 3. Error spreading statistics

In order to evaluate where all the possible faults can occur, the complete DC for each node has to be calculated. If the circuit is small enough, a complete simulation can be performed to compute the complete DC computation. But for most circuits, the number of inputs is too large to perform a full simulation, and a full DC computation is very costly [12]. In these cases we will use a windowing technique [14] to perform the evaluation. Windowing works by creating a small environment (defined by a TFI and a TFO set) around the node under test and performing a complete simulation in this environment. By restricting the size of the circuit under test using the windowing technique, we can simplify the fault coverage problem size. Since the window is significantly smaller than the whole circuit, the window-based fault coverage will result in an overestimate of the real faults by underestimating the complete DC of the LUT node in the window. Unlike Monte Carlo simulation-based methods [6], [9], [17], if a configuration bit is determined not to be critical in a window, then it is guaranteed to be not critical in the circuit. So, we never have false negatives using the windowing approach.

The quality of the windowing solution is determined by how accurately it can be used to generate the complete DC set for each node in the circuit. Since windowing is guaranteed to produce a DC lower bound and simulation produces a DC upper bound, the difference (scaled to total number of configuration bits in the circuit) can be used to evaluate the quality of the solution. Specifically, the quality of the solution can be evaluated using the following equation:

$$Gap = \frac{\sum |DC_{sim}| - \sum |DC_{win}|}{\sum 2^{inputs}} \quad (1)$$

For a LUT node with $|inputs|$ inputs, $|DC_{win}|$ ($|DC_{sim}|$) is the size of the DC set computed by the windowing formulation (Monte Carlo simulation), Σ is a summation over all the nodes in the circuit, and 2^{inputs} is the number of configuration bits of the node and also the maximum possible size of the DC. Thus, the gap is the difference between the two DC evaluations scaled to the maximum possible size of the DC. For the simulation-based DC calculation, we used a complete simulation for circuits with less than 25 inputs and 100M random simulation for the rest of the circuits. To generate a baseline for previous methods, we evaluated a quick 100K Monte Carlo simulation and the windowing from [14]. The windowing in [14] creates windows by combining a fanout cone and a fanin cone with the intermediate nodes. The results of using these methods to calculate the DCs for the MCNC benchmarks are shown in Table 1. Assuming the large 100M simulation provides a good estimation of the sum of the DCs, a quick simulation is overestimating by almost 3.31%, while the windowing from [14] is underestimating by 6.65%. This is an encouraging starting point which shows that the baseline windowing technique can be greatly improved.

Table 1. Baseline evaluation

Circuit	100M or Full Simulation		100k Quick Simulation		[14] Windowing		Total Possible Critical Bits
	$\Sigma DC $	$\Sigma DC $	Gap	$\Sigma DC $	Gap		
alu4	6062	6062	0%	5307	-4%	19752	
apex2	6676	11675	20%	3151	-14%	25484	
apex4	10699	10699	0%	10522	-1%	21273	
clma	62960	94317	27%	29267	-29%	117790	
misex3	5704	5704	0%	5188	-3%	17844	
pdc	38036	38659	1%	35501	-4%	57692	
s298	63	63	0%	60	0%	668	
s38417	8497	13579	7%	1704	-9%	75060	
s38584.1	10691	11171	1%	7920	-4%	77010	
seq	7739	8606	3%	5177	-9%	27176	
spla	35445	35945	1%	33196	-4%	54832	
bigkey	2832	2832	0%	2688	-1%	22424	
des	4968	4968	0%	4968	0%	25244	
diffeq	2221	2337	0%	764	-6%	23272	
dsip	7168	7168	0%	7168	0%	18744	
elliptic	986	1334	2%	56	-6%	16900	
ex1010	11687	11687	0%	11659	0%	24560	
ex5p	7856	7856	0%	6646	-9%	12792	
frisc	19433	22957	5%	1326	-23%	77464	
tseng	4079	4120	0%	2564	-8%	17837	
Average			3.31%		-6.65%		

4. DON'T-CARE COMPUTATION WITH OVERLAPPING WINDOWS (DCOW)

To improve the DC computation of the previous methods, we introduce a new windowing method, called DCOW (Don't-care Computation with Overlapping Windows), which has three major innovations: basic window creation (Section 4.1), information passing between windows (Section 4.2), and growing windows in an intelligent manner (Section 4.3).

4.1 Input-Bounded Windowing

Our main improvement relates to how we generate the window. Instead of adding a set number of fanin and fanout levels, as in [12] and [14], we construct an input-bounded window which tries to

maximize the volume (as defined by the number of nodes covered). Since the window has to grow in both directions, the maximum volume K-feasible cut method from FlowMap [4] cannot be used directly, and we grow the window using a heuristic. For each node that can be added to the window, we select the one that adds the least amount of new inputs to the window. This allows the window to grow as large as possible while maintaining the limit on the number of inputs. This volume maximizing idea is effective for two reasons. First, the more nodes the window covers the more likely it is to capture the complete DC information. Second, this heuristic attempts to limit the fanins of the window by selecting reconvergent nodes. This in turn helps restrict the controllability set size by reducing the possible inputs to the node. These two properties allow DCOW to estimate the DC within 3.19%, which correlates to reducing the estimated gap by over 2X when compared to the baseline windowing in [14].

The windowing heuristic generates a lot of ties by only examining nodes that are immediately connected to the window. This is especially a problem when selecting the first node to add to the window. But by performing a simple lookahead of one node, we can further reduce the average DC estimate gap to 2.66%, a further reducing the estimated gap by 19.6% when compared to the baseline windowing in [14].

4.2 Overlapping Windows

The first step in evaluating whether a configuration bit is in the DC set is to determine if it is **not** controllable. The controllability set can be minimized by overlapping the windows to further restrict the controllability set of each window. Let us define controllability(n, W) to be the set of possible inputs to LUT n when performing a complete simulation of window W . If two windows disagree on the controllability of node n , i.e., $p \in \text{controllability}(n, W2)$ but $p \notin \text{controllability}(n, W1)$, then the window input which corresponded to $p \in \text{controllability}(n, W2)$ was not a valid input to that $W2$. Thus, the final controllability of node n is defined as:

$$\text{controllability}(n) = \bigcap_{n \in \text{window } W} \text{controllability}(n, W) \quad (2)$$

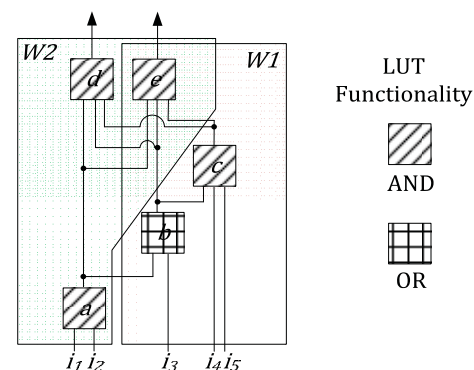


Figure 4. Overlapping window example

For example, consider Figure 4. It has two 4-input windows defined by their inputs, $W1 = \{a, i_3, i_4, i_5\}$ and $W2 = \{i_1, i_2, b, c\}$ (where a, b , and c are outputs from LUT nodes labeled a, b , and c) with a common node e . Both windows must agree on the controllability set for e . Consider input $p = \{a, b, c\} = \{0,0,1\}$ to node e . We can derive that $p \in \text{controllability}(e, W2)$ but $p \notin \text{controllability}(e, W1)$; then the following inputs to the $W2$

are not valid: {0,0,0,1}, {0,1,0,1}, {1,0,0,1}, and {1,1,0,1}. The more overlapping windows we use, the more accurate our solution will become. This can further reduce the estimated gap of the DCOW-based DC calculations method by 2.7% when compared to the baseline windowing in [14].

4.3 Simulation Guided Growing

After computing the controllability for each node, the step to evaluating whether a configuration bit is critical is to determine if it is observable. The observability set is much more difficult to represent accurately. The main difficulty of a window-based observability calculation is that it is important to cover all the fanouts of a node. This is the only way to determine if the error will get masked. If not enough nodes are encompassed by the window, then the window will not be very effective. From the results in Figure 2 and Figure 3, we see that we do not need to cover very many TFO (transitive fanout) nodes to determine if the error is observable. In fact, if the fanout window captures at least five TFO nodes, it then has an 85.4% chance that any error escaping the window will reach the outputs. If the window can at least cover ten TFO nodes, then this probability is increased to 99.4%.

Unlike previous methods which selected several levels of TFO nodes [12], [14], and [21], we use simulation to help select which TFO nodes should be included in the window. We first perform a quick simulation to see where the errors can spread to without reaching the outputs (false positive errors), and then we guide windowing to cover the TFO nodes which have the most false positive errors. Overall, this technique can further improve the DCOW-based DC calculation gap by 20.3% when compared to the baseline windowing in [14].

4.4 Summary of Windowing

The summary in Figure 5 shows how our DCOW method compares to the currently available techniques. Initially, obtaining a solution that is close to simulation might seem like a lot of work. But, the real DC calculation lies between the “DCOW” line and the “Sim 100M” line, and windowing is a scalable way to get an accurate lower bound of the complete DC set. And a DC lower bound is much more valuable than an upper bound, as shown in the following theorem:

Theorem 1: Given a LUT, let f be its original function and f' be the resulting function after a configuration bit error. If $f \oplus f'$ is covered by the DC lower bound computed by DCOW, then the configuration bit error is not critical.

The proof of this theorem is omitted due to the page limit. Note that the DC lower bound computed by DCOW can be used in other logic optimization procedures, such as resynthesis and rewiring.



Figure 5. Summary of methods

The results in Table 2 show that the improved windowing technique can measure the DC of a node within 2.34% on average. When comparing the estimated DC gap, DCOW amounts to a 2.88X improvement over the baseline windowing from [14] and a 41% improvement over the 100K simulation-based DC calculation.

In terms of runtime, it took less than 10 minutes to perform the windowing estimation for all circuits, compared to 48 minutes for the 100K Monte Carlo simulation.

Table 2. Evaluation of the DCOW method

Circuit	100M or Full Simulation	ABC [14] Windowing		DCOW		Total Possible Critical Bits
	$\Sigma DC $	$\Sigma DC $	Gap	$\Sigma DC $	Gap	
alu4	6062	5307	-4%	6062	0%	19752
apex2	6676	3151	-14%	3995	-11%	25484
apex4	10699	10522	-1%	10699	0%	21273
clma	62960	29267	-29%	47228	-13%	117790
misex3	5704	5188	-3%	5704	0%	17844
pdc	38036	35501	-4%	38036	0%	57692
s298	63	60	0%	63	0%	668
s38417	8497	1704	-9%	5918	-3%	75060
s38584.1	10691	7920	-4%	9753	-1%	77010
seq	7739	5177	-9%	6337	-5%	27176
spla	35445	33196	-4%	35445	0%	54832
bigkey	2832	2688	-1%	2800	0%	22424
des	4968	4968	0%	4968	0%	25244
diffeq	2221	764	-6%	1338	-4%	23272
dsip	7168	7168	0%	7168	0%	18744
elliptic	986	56	-6%	662	-2%	16900
ex1010	11687	11659	0%	11687	0%	24560
ex5p	7856	6646	-9%	7856	0%	12792
frisc	19433	1326	-23%	15779	-5%	77464
tseng	4079	2564	-8%	3639	-2%	17837
Average			-6.74%		-2.34%	

In the next section we will present a technology mapping extension to ABC that uses windowing to improve the probability that a design will function correctly when implemented on a faulty FPGA.

5. RELIABILITY IMPROVEMENT IN MAPPING

Using our newly created DC computation method, DCOW, we developed, DCOWMap, a method for improving reliability through technology mapping. This method extends ABC [13] using a windowing technique to simplify reliability evaluation (shown in Figure 6). Since technology mapping is just a covering problem, it does not affect the functionality of internal nodes. This property ensures that the mapping of one node does not affect the DC set of any other nodes. This independence integrates very well in the dynamic programming based technology mapping used in ABC.

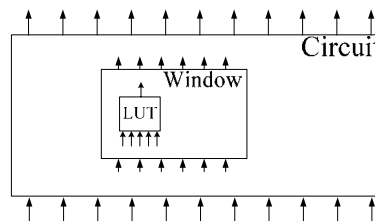


Figure 6. Windowing technique for FPGAs

By performing this technology mapping-based reduction we can evaluate the amount of reliability gains that can be achieved through a localized search. We implemented DCOWMap by adding an extra two steps to the ABC FPGA mapper (command “fpga”). Each step works by creating a DCOW-based window around each node in topological order (PI to PO), performing a

complete simulation on the window and then selecting the best cut implementation for the node. The two steps of mapping differ only in how they perform cut selection. In the first step we use a critical bit flow formulation, and in the second step we use an exact critical bit formulation.

5.1 Critical Bit Flow Cut Selection

The first step selects the best cut implementation by using a flow heuristic commonly used for area flow calculations [5],[10],[13]. We recursively define critical bit flow (CBF) as follows:

$$CBF(n) = CriticalBits(n) + \sum_{f \in fanins(n)} \frac{CBF(f)}{NumFanout(f)} \quad (3)$$

where $CriticalBits(n)$ is the number of critical bits of the LUT implementation of n (also equal to the complement of n 's DC set), $fanins(n)$ are the fanins of n , and $NumFanout(f)$ is the number of fanouts of node f in the current selected mapping. The notion of a flow captures the fact that each incoming flow is shared by the fanouts of the fanins, which gives a global view of the cost function during technology mapping similar to that used in [13] and [5]. This global view is then complemented with the use of the exact critical bit cut selection in the second step.

5.2 Exact Critical Bit Cut Selection

The second step selects the best cut implementation by recursively calculating the exact number of critical configuration bits it would take to implement each cut. This is based on the local view heuristic in [10], known as the exact area. The exact number of critical bits (ECB) of a cut is defined as the sum of the number of critical bits of the LUTs in the maximum fanout free cone (MFFC) of the cut, and can be recursively defined as follows:

$$ECB(n) = CriticalBits(n) + \sum_{\substack{f \in fanins(n) \wedge \\ f \in MFFC(n)}} ECB(f) \quad (4)$$

where $CriticalBits(n)$ is the number of critical bits of the LUT implementation of n , $fanins(n)$ are the fanins of n , and $MFFC(n)$ is the set of nodes in the MFFC rooted at n . This can be calculated in a DFS order very efficiently since the selection of a single cut does not usually make a drastic change to the mapping solution of the circuit.

6. RESULTS

We evaluated DCOWMap using the 20 largest MCNC benchmark sets. The results in Table 3 were evaluated using either a full simulation of all possible inputs (latches are treated as primary inputs) for circuits with less than 25 combinational logic inputs (alu4, apex4, misex3, pdc, s298, spla, ex1010, and ex5p) or using a Monte Carlo simulation with 100M random inputs for the rest of the circuits. These benchmarks were each mapped depth optimally to the 6-LUT architecture. Since the number of inputs to each window is relatively small, DCOWMap was able to map the largest circuit, clma, in less than 200 seconds. This corresponds to evaluating 50 nodes each second. These results show that with no area increase, the critical bits can be reduced by over 12% (spla, des, and ex5p were reduced by over 20%). Since the area was unaffected, both versions (ABC baseline and DCOWMap) would be implemented on the same FPGA with the same number of

possible failures. Thus, on average, the optimized version can tolerate 12% more permanent faults in the LUT configuration bits.

To verify that our mapping solution did not degrade the placement solution, we placed and routed the mapped circuits using VPR [3]. We observed no increase in routing failure and no critical path delay increase (in fact, critical path delay decreased by 0.4%).

Table 3. Technology mapping results

Circuit	ABC Baseline [13]		DCOWMap	
	LUTs	# Critical Bits	LUTs	# Critical Bits
alu4	507	13690	506	11365
apex2	687	18808	687	17075
apex4	594	10573	591	9155
clma	2830	54816	2822	47693
misex3	490	12140	489	10165
pdc	1515	19656	1508	15890
s298	25	605	25	576
s38417	2496	66563	2501	62415
s38584.1	2334	66291	2332	55805
seq	705	19437	705	16015
spla	1436	19387	1435	14899
bigkey	578	19592	578	19592
des	556	20276	556	15308
diffeq	556	21051	556	19898
dsip	873	11576	873	11576
elliptic	329	15914	329	15646
ex1010	668	12873	667	12181
ex5p	384	4936	383	3694
frisc	1997	58039	1997	49757
tseng	756	13757	756	12872
Geomean	716.56	16686.14	715.84	14638.26
Ratio	1.000	1.000	0.999	0.877

To verify that the DCOW-based critical bit evaluation maintains the ratio seen in Table 3, the mapped MCNC benchmarks were reevaluated using DCOW-based windowing. We observed no significant change in the ratio of the number of critical bits (changed from 0.877 to 0.878) but the run time was reduced to 10 minutes from 31 days when using DCOW instead of a 100M simulation. Using the DCOW-based evaluation allows us to *evaluate* much larger designs, for which a Monte Carlo evaluation of 100M would not be feasible (specifically, the IWLS benchmark suite [22] with circuits that have up to 107K simple gates). When DCOWMap optimized the number of critical bits of the 2005 IWLS OpenCore benchmark suite [22], we observed that, using a DCOW-based evaluation, DCOWMap was able to reduce the number of critical bits by 10.6% with no area increase (due to space constraints the results are not included here but can be seen on the project website indicated at the conclusion of this paper). This shows that the quality of the solution does not degrade when scaled to real-world size problems.

These results might seem to be less impressive when compared to ROSE [9] (which claims to reduce the fault rate by 25%), but the fault rate estimation used by ROSE cannot accurately measure permanent configuration bit failures. The ROSE fault estimation works by injecting one fault into a new randomly chosen configuration bit for each one of the 20K inputs, and then testing to see if the fault can be observed. But there are two problems with this approach. First, 20K is an extremely small sample for evaluating a solution. If the circuit has 300 nodes, then a 20K simulation corresponds to testing each configuration bit with roughly only one possible input vector (300 nodes * 64 configuration bits per node = 19200 total configuration bits). This misleads the algorithm so that it optimizes the reliability metric for

each configuration bit for that specific input vector instead of all possible input vectors. Second, by selecting random configuration bits to test instead of testing all of them, the fault rate stays artificially low. For example, this would allow a minor change, like the masking of 200 configuration bits, to appear like a 2X improvement ($20K * 2\% = 400$ versus $20K * 1\% = 200$). In the best case, the ROSE fault evaluation calculates the amount of masking that exists for transient configuration bit faults, but configuration bit faults are not transient. In this case we need to ensure that the circuit functions correctly under all inputs, not just one. In our evaluation we use a new 100M input vector to evaluate each node and report the total number of critical bits.

7. CONCLUSION

The contributions of this work can be broken down into three parts. First, we presented a quantitative error propagation study where we identified several critical properties that enable efficient DC calculations. Second, we presented a scalable DC computation method using an improved windowing method (DCOW), with a 2.88X improvement over the windowing in [14]. Third, we used the windowing technique to create a reliability-driven technology mapper, DCOWMap, which is able to reduce the total number of critical configuration bits by 12.3% with no area overhead. Our work shows that it is possible to quickly increase the probability that a design will function correctly when implemented on a faulty FPGA.

The source code for the technology mapping algorithm and all the reliability evaluators can be downloaded at:
http://cadlab.cs.ucla.edu/software_release/reliability/

8. ACKNOWLEDGEMENTS

Financial support from Altera, Magma, and NSF grant CNS-0725354 are greatly acknowledged.

9. REFERENCES

- [1] R. I. Bahar, D. Hammerstrom, J. Harlow, W. H. Joyner, C. Lau, D. Marculescu, A. Orailoglu, and M. Pedram, "Architectures for Silicon Nanoelectronics and Beyond," *IEEE Computer*, pp. 25–33, Jan. 2007.
- [2] P. Beckett and A. Jennings, "Towards Nanocomputer Architecture," *Asia-Pacific Computer System Architecture Conference*, pp. 141–150, Feb. 2002.
- [3] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *International Workshop on Field-Programmable Logic and Applications*, 1997.
- [4] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. Computer-Aided Design*, pp. 1-12, 1994.
- [5] J. Cong, C. Wu, and E. Ding, "Cut Ranking and Pruning: Enabling a General and Efficient FPGA Mapping Solution," *International Symposium on Field-Programmable Gate Arrays*, pp. 29-35, Feb. 1999.
- [6] Z. Feng, Y. Hu, L. He, and R. Majumdar, "IPR: In-Place Reconfiguration for FPGA Fault Tolerance," *International Conference on Computer-Aided Design*, Nov. 2009.
- [7] M. Forshaw, R. Stadler, D. Crawley, and K. Nikolic, "A Short Review of Nanoelectronic Architectures," *Nanotechnology*, volume 15, pp. 220–223, Sept. 2003.
- [8] J. Han, J. Gao, Y. Qi, P. Jonker, and J. A. B. Fortes, "Toward Hardware-Redundant, Fault-Tolerant Logic for Nanoelectronics," *IEEE Design and Test of Computers*, 22(4):328–339, Aug. 2005.
- [9] Y. Hu, Z. Feng, L. He, and R. Majumdar, "Robust FPGA Resynthesis Based on Fault Tolerant Boolean Matching," *International Conference on Computer-Aided Design*, pp. 706-713, 2008.
- [10] S. Jang, B. Chan, K. Chung, and A. Mishchenko, "WireMap: FPGA Technology Mapping for Improved Routability and Enhanced LUT Merging," *ACM Trans. Reconfigurable Technology and Systems (TRETS)*, Vol. 2(2), Jun. 2009.
- [11] G. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw Hill, 1994.
- [12] A. Mishchenko and R. Brayton, "SAT-Based Complete Don't-Care Computation for Network Optimization," *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 412-417, Mar. 2005.
- [13] A. Mishchenko, S. Chatterjee, R. Brayton, and M. Ciesielski, "An Integrated Technology Mapping Environment," *Proceedings International Workshop on Logic & Synthesis*, pp. 383-390, Jun. 2005.
- [14] A. Mishchenko, J. Zhang, S. Sinha, J. Burch, R. Brayton, and M. Chrzanowska-Jeske, "Using Simulation and Satisfiability to Compute Flexibilities in Boolean Networks," *IEEE Trans. CAD*, Vol. 25(5), pp. 743-755, May 2006.
- [15] K. Nikolic, A. Sadek, and M. Forshaw, "Fault-Tolerant Techniques for Nanocomputers," *Nanotechnology*, pp. 357–362, 2002.
- [16] J. H. Patel and L. Y. Fung, "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands," *IEEE Transactions on Computers*, 31:589–592, Dec. 1982.
- [17] I. Polian and W. Rao, "Selective Hardening of NanoPLA Circuits," *International Symposium on Defect and Fault Tolerance of VLSI Systems*, pp. 263-271, Oct. 2008.
- [18] Y. Qi, J. Gao, and J. Fortes, "Markov Chains and Probabilistic Computation - a General Framework for Multiplexed Nanoelectronic Systems," *IEEE Transactions on Nanotechnology*, 4(2):194–205, Mar. 2005.
- [19] W. Rao, R. Karri, and A. Orailoglu, "Fault Tolerant Arithmetic with Applications in Nanotechnology Based Systems," *International Test Conference*, pp. 472–478, 2004.
- [20] W. Rao, A. Orailoglu, and R. Karri, "Towards Nanoelectronics Processor Architectures," *JETTA Special Issue on Test, Defect Tolerance, and Reliability of Nanoscale Devices*, 23:235–254, 2007.
- [21] Q. Zhu, N. Kitchen, A. Kuehlmann, A. Sangiovanni-Vincentelli, "SAT Sweeping with Local Observability Don't-Cares," *Design Automation Conference*, pp. 229-234, Jun. 2006.
- [22] IWLS 2005 Benchmarks,
<http://iwls.org/iwls2005/benchmarks.html>
- [23] Xilinx, EasyPath FPGAs,
<http://www.xilinx.com/products/easypath>