Rewiring For Robustness

Manu Jose[1], Yu Hu[3], Rupak Majumdar[1] and Lei He[2]
1. Computer Science Department, University of California, Los Angeles
2. Electrical Engineering Department, University of California, Los Angeles
3. Electrical and Computer Engineering Department, University of Alberta *

ABSTRACT

Logic synthesis for soft error mitigation is increasingly important in a wide range of applications of FPGAs. We present R2, an algorithm for rewiring a post-layout LUT-based circuit that reduces the overall criticality of the circuit, where criticality is the fraction of primary inputs that lead to observable errors at the primary outputs if an single event upset inverts a configuration bit. Our algorithm explicitly optimizes the robustness of the interconnect, the dominant component of FPGAs. The key idea of R2 is to exploit Boolean flexibilities in the circuit implementation to replace wires with high criticality with those with lower criticality while preserving the circuit functionality. We estimate criticalities using a Monte Carlo fault simulation. We represent flexibilities using SPFDs (Set of Pairs of Functions to be Distinguished), and use criticality information to choose candidates for rewiring, assigning the maximum flexibility to high criticality wires. Compared to IPR, a recent robust logic optimization, our implementation increases MTTF (Mean Time to Failure) by 24%, showing for the first time, the advantages of exploiting Boolean flexibilities in optimizing for robustness. In addition, R2 achieves 5% and 2% more reduction on the number of wires and LUTs in an FPGA than that obtained by the existing rewiring algorithm for area minimization.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated circuits – Design aids

General Terms

Algorithms, Reliability

*Manu Jose was supported in part by the DARPA grant HR0011-09-1-0037. The project was partially funded by the discovery program in National Sciences and Engineering Research Council of Canada. Part of Yu Hu's work was performed when he was with Electrical Engineering Department, UCLA, and part of Lei He's work was performed during his visit to the State Key Laboratory for ASIC, Fudan University, Shanghai China.

DAC 2010, June 13-18, 2010, Anaheim, California, USA.

Keywords

SPFD, FPGA, Rewiring, Logic synthesis, Soft Errors

1. INTRODUCTION

Field programmable gate arrays (FPGAs) are built upon memory elements that are particularly vulnerable to soft errors (e.g., caused by single event upsets (SEUs)). Due to the aggressive CMOS scaling and the emerging nano devices, soft error mitigation for SRAM-based FPGAs has become increasingly important for a wide range of applications from internet line cards to enterprise servers [17]. An effective approach for reducing the impact of the soft errors can lead to higher mean-time-to-failure (MTTF), reduced maintenance cost, and increased QoS (quality of service).

Among all memory elements in an FPGA, 99% are configuration bits (the rest are user bits, e.g., flip-flops and on-chip memory data) [15]. An analysis [11] of the failure modes of FPGA-based designs shows that failures due to routing structures (i.e. interconnect) is dominant, about 80% of all failures. In this paper, we specifically take the impact of soft error on interconnect into consideration, and focus on error mitigation techniques based on *logic resynthesis* that interact with physical design in the CAD flow.

The idea of *robust logic synthesis*, which performs logic resynthesis with reliability as the primary optimization criterion, has been explored in several recent papers [16, 18, 20, 21]. The main insight in these papers is *logic masking insertion*, in which a logic block is rewritten for maximal logic masking in order to prevent the propagation of faults through the block. For example, [18] introduce logic masking by a template with reconvergent paths for technology mapping and resynthesis, and [20] rewrites LUT configuration bits for local logic masking while maintaining the circuit topology. Compared with Triple Modular Redundancy (TMR) [14], a widely applied technique for soft error mitigation, logic masking-based fault tolerance has much lower area, performance and power overhead.

However, in most existing fault-tolerant resynthesis algorithms, a logic block to be restructured is expressed by the completed specified function (CSF) derived from the LUTs inside this block. In other words, the Boolean *flexibility* due to the rest of the network outside the logic block is not taken into consideration, and therefore the opportunity of resynthesis is limited. In contrast, we in this paper explore the opportunity of using Boolean flexibility in robust logic synthesis. Specifically, we propose a rewiring algorithm for a post-layout circuit to improve reliability. Since rewiring optimizes interconnect, the most soft error-susceptible com-

ponent in an FPGA, the intuition is that a rewiring optimization will increase reliability.

We use Set of Pairs of Functions to be Distinguished (SPFD) [5]) as our representation for Boolean flexibilities. Our choice is driven by two factors. First, SPFDs can represent more flexibilities than don't-cares [7], and therefore give us more potential to insert logic masking. Second, SPFDs have previously been used successfully for rewiring of LUT-based circuits for area [5, 9], delay and routability optimization [10].

Our main contribution is an algorithm for *robust rewiring*, called R2. The R2 algorithm takes a post-layout circuit and performs rewiring, i.e., finds alternatives for a wire and replaces it with the one for maximal robustness. R2 uses Monte Carlo simulation to estimate the *criticality* for each configuration bit (i.e., the fraction of primary inputs that lead to observable errors at the primary outputs if an SEU inverts the configuration bit). The key idea in R2 is to replace the most critical wires with those with lower criticality, where the candidates for replacement are found based on SPFD computations.

Unlike SPFD-based rewiring for area optimization [5], during the course of resynthesis, R2 re-assigns the SPFD according to the criticality of a wire to maximize the opportunity of the replacement of the highly critical wires. We compare our results against a baseline produced by IPR [20], a recent robust resynthesis for LUT reconfiguration. The R2 algorithm increases the MTTF by 24% and decreases the numbers of wires and LUTs by 5% and 2% respectively at the same time, over the IPR baseline. In fact, we see slight decreases in wire and LUT numbers (2% and 1%) over an area-minimizing rewiring technique [5] as well.

Flexibilities have been widely used in various resynthesis algorithms for area [9], power [6], or floorplanning and placement [8] optimization. To the best of our knowledge, this paper is the first to apply the design freedom offered by flexibilities to optimize for reliability in FPGA designs.

The remainder of this paper is organized as follows. Section 2 describes the preliminaries. Section 3 introduces the proposed R2 algorithm. Section 4 shows the experimental results, and the paper is concluded in Section 5.

2. PRELIMINARIES

2.1 Boolean Network

A LUT-based *Boolean network* is represented as a directed acyclic graph (DAG) whose nodes represent LUTs and whose directed edges correspond to wires connecting the LUTs. For a wire (n, n'), we say n is the *source* and n' the *target* of the wire. A sequential circuit can be treated as a combinational one by removing the registers. The nodes in the lowest level of the DAG are called *circuit inputs* (CIs), which include the *primary inputs* (PIs) and the outputs of registers. The nodes in the highest level are called *circuit outputs* (COs), which include *primary outputs* (POs) and the inputs to registers. The transitive *fanin* (TFI) (resp. *fanout* (TFO)) *cone* of node n is a sub-network whose nodes can reach the fanin edges of n (resp. can be reached from the fanout edges of n).

A circuit is *K-LUT mapped* iff each node has a single output pin p_0 and up to K input pins $p_1, \cdots, p_u$, for $u \leq K$. Each node has an internal logic function $p_0 = f(p_1, \cdots, p_u)$ that defines the logic relationship between the output and input pins of the node. Each pin can be associated with a global logic function $g_1, \cdots, g_u$, respectively, in terms of the CIs of the circuits, defined inductively as follows. The global function associated with a CI x is x itself. The global function associated with the output pin p_0 of a node with input pins $p_1, \cdots, p_u$ is $f(g_1, \cdots, g_u)$, where $g_1, \cdots, g_u$ are the global functions associated with pins $p_1, \cdots, p_u$ respectively and f is the logic function for the node.

2.2 SPFD

For a Boolean function f, we write $\overline{f}$ for the function such that $\overline{f}(x) = 0$ iff $f(x) = 1$ for each input x. For two Boolean functions f_1 and f_2, we say f_2 *covers* f_1, written $f_1 \leq f_2$, if whenever $f_1 = 1$ we have $f_2 = 1$. Let (π_1, π_0) be a pair of logic functions such that π_0 and π_1 are non empty and they do not overlap ($\pi_1 \neq 0$, $\pi_0 \neq 0$, and $\pi_1 \wedge \pi_0 = 0$). A function f can *distinguish* (π_1, π_0) if $\pi_1 \leq f \leq \overline{\pi_0}$ or $\pi_0 \leq f \leq \overline{\pi_1}$. I.e., there is no overlap between on-set of $\pi_1(\pi_0)$ and off-set of $\pi_0(\pi_1)$.

An SPFD [5]

$$R(x, x') = \{(\pi_{11}(x), \pi_{10}(x')), \cdots, (\pi_{m1}(x), \pi_{m0}(x'))\},$$

is a set of pairs of Boolean functions, where variable x ranges over some space X. A Boolean function f satisfies an SPFD $R(x, x')$ iff f distinguishes all m functions pairs in $R(x, x')$. Note that SPFD is a way to express the flexibility in a Boolean network.

Given a Boolean network, [5] proposed the following algorithm to compute the SPFD for each node and each wire (i.e., an edge in the Boolean network graph). There are two passes in the computation. The first, forward traversal (from CIs to COs) of the network is used to compute the global logic function of each node. The second, backward traversal computes the SPFDs at each pin.. At each pin, the SPFD is computed as follows.

1. At each CO node, O_j, the SPFD can be obtained as $R_j(X, X') = (f_j^{on}, f_j^{off})$, where f^{on} and f^{off} represents the on-set and off-set of function f, respectively, and X represents the vector in the CI space.

2. At the output pin of a node, the SPFD is the union of its fanout pins' SPFDs.

3. For an input wire (i, j) of a node j, its SPFD $R_{ij}(X, X')$ is obtained by decomposing its output SPFD $R_j(X, X')$ into minterms and assigning the function pairs backwards to input pins.

Rewiring is a technique that replaces a wire in the Boolean network with another while maintaining functional equivalence. An SPFD-based rewiring algorithm was proposed in [5, 9]. A wire (i, j) can be replaced by the output of a node q without changing the logic functionality of the COs if the SPFD of the output pin of node q satisfies $R_{ij}(X, X')$, the SPFD of (i, j). After the wire replacement, the logic function of LUT j might need be changed according to the logic functionality of q. When a wire (i, j) does not provide any unique information to the sink node j, it can be removed from the network. Again, the function at LUT j has to be changed to account for the different flow of information. The algorithm proceeds in topological order from primary inputs to primary outputs and attempts to change the wiring of each node in the network, if some gain is obtained by the replacement.

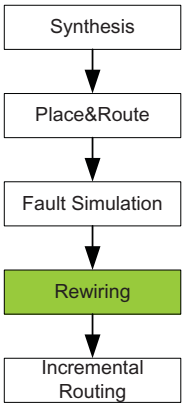

Figure 1: CAD flow

3. ROBUST REWIRING

We propose a rewiring algorithm whose objective is to improve the robustness of a circuit to single-event upsets. As shown in Figure 1, our proposed algorithm is applied after placement and routing, since we can get accurate fault information (particularly for interconnect) only after that phase.

3.1 Fault Model and Fault Simulation

Following [20, 18], we assume a *single fault* model, i.e., at most one single event upset (SEU) occurs in a clock period. This assumption is valid, given the real SER for commercial SRAM-based FPGAs [19]. We consider SEUs on both LUT configuration bits and routing. An SEU that occurs in an LUT configuration bit results in the flip of the logic value (i.e., "0" ("1") changes to "1" ("0")) in the corresponding SRAM bit, and consequently changes the logic function encoded by the LUT.

An SEU that occurs in a configuration bit of a connection box or a switch box for the routing resource can cause the following three faults: (1) stuck-at fault (mis-connected to a "0" or "1" signal), (2) stuck-open fault (two originally connected wires are broken), (3) bridging fault (multiple wires are mis-connected) [15]. Given a FPGA-based design, a detailed analysis of the impact of these faults at the physical level is expensive. For example, the logic value associated with a faulty routing signal due to a bridging fault depends on both the logic value and the individual driving strength of the bridged nets. In our experiments, we make the following simplified assumption: *an SEU that occurs in a routing configuration always causes the flip of the logic value of the corresponding routing signal*. In other words, an SEU in a routing signal always changes the logic value that it carries. Note that this is a pessimistic assumption and estimates an upper bound of the fault rate.

While we make this simplifying assumption for efficient fault simulation, our proposed algorithm can be applied to more sophisticated fault models (e.g., [15]) by replacing our fault simulator with a more sophisticated one.

Using the fault model, we perform fault simulation using Monte Carlo sampling to estimate the *criticality* of a configuration bit. The criticality of a configuration bit c is the percentage of CI vectors that cause erroneous values for COs due to the flip of the logic value of c.

Intuitively, the criticality of a configuration bit shows how likely is the flip of this bit observed at the COs. The higher the criticality of a configuration bit is, the more input vectors may sensitize the SEU that occurs at this bit. This definition quantitatively measures the impact of the SEU in the bit level on the *mean time to failure* (MTTF). It is easy to show that MTTF is inversely proportional to the average criticality of all configuration bits of a design.

3.2 The R2 Algorithm

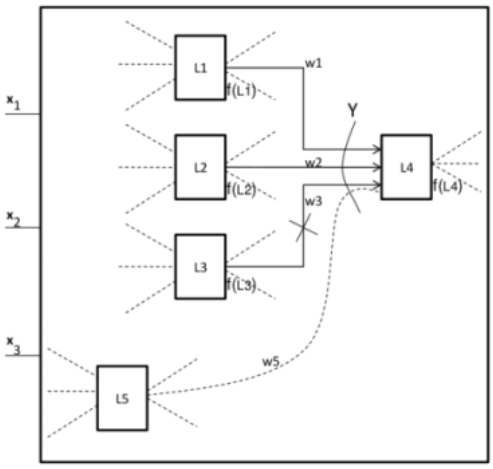

Figure 2: A LUT Network

Given a K-LUT mapped circuit, the objective of the *rewiring for robustness* algorithm is to find alternatives for potential interconnects to minimize the overall criticality of configuration bits in the entire circuit, while keeping the total number of wires under a threshold.

We present an algorithm, called R2, for rewiring for robustness. We start with an example to illustrate the intuition behind the algorithm. Figure 2 shows a fraction of a circuit during the course of the resynthesis. This region contains five 3-LUTs $(L_1, \cdots, L_5)$, where L_1, L_2, L_3 are the fanins of L_4 with wire connections w_1, w_2, w_3, respectively. Suppose that there are three primary inputs $X = \{x_1, x_2, x_3\}$ and that for each LUT L, the function $F(L)$ at the output of L is represented based on X space as shown in Figure 3. The SPFD at the output of L_4 on Y space is

$$R_4(Y, Y') = \{m_{001}, m_{111}\} \otimes \{m_{000}, m_{011}, m_{101}, m_{110}\},$$

where $m_{y_1 y_2 y_3}$ is a minterm in Y-space of L4 and $\otimes$ is the cartesian product of the two sets of minterms.

Assume the criticalities of w_1, w_2, and w_3 are c_1, c_2, and c_3 respectively, and that $c_3 > \max(c_1, c_2)$. R2 identifies wires such as w_3 with the maximum criticality in the region of resynthesis, and tries to find a replacement wire for w_3 which will result in a lower criticality of the network.

Formally, R2 works as follows.

1. Perform fault simulation to estimate criticality for each wire and sort wires according to criticality.

2. Choose the wire w with the maximum criticality as a potential target for replacement.

3. Compute the flexibility of w. In contrast to the method in [5], in the course of backward propagation of the SPFD computation, we try to minimize the size of w (i.e., the number of minterms needed to be distinguished by w), giving it the maximal flexibility. Intuitively, the more flexibility a wire has, the more potential replacement nodes in its transitive fanin (TFI) can be found.

4. Search nodes in the transitive fanin that could replace w, i.e., find a node whose function satisfies the SPFD of w. Let n' be such a node. Make a temporary connection w' from n' to the target of w and remove w. This may require reprogramming the LUTs in the transitive fanout (TFO) of n'.

   As an example, consider the LUT L_5 with function $F(L_5)$, which satisfies the SPFD of w_3 as shown in Figure 3. We make a temporary connection w_5 and remove w_3. As a consequence, we may need to modify the internal logic of L_4. Such programming can be always carried out as SPFD of w_5 inherenlty assumes that the internal logic of L_4 can be freely changed.

5. Update the criticality of the changed wires and nodes through fault simulation, and reject the rewiring if the total criticality increases.

Step 2 is the key difference between the area-minimizing SPFD computation and the one used in R2. Essentially, to calculate the SPFD at the fanin of a node j, we assume that the SPFD R_j, has been computed. We need to give an ordering on the fanin pins of n_i. Such an arbitrary ordering is decided in the area-minimizing SPFD computation proposed in [5]. However, in robust rewiring we take the advantage of the known information of the criticality of a wire and use it to give the privilege (highest flexibility) to the most critical wire, and therefore a replacement for this wire is more likely to be found. Specifically, suppose that the fanin wire (i, j) of node j has the highest criticality, its SPFD is formally given as

$$R_{ij} = R_j \wedge (y_i \neq y_i') \wedge \prod_{u \in \text{fanin(j)} \wedge u \neq i} (y_u = y_u').$$

That is, the SPFD of fanin wire (i, j) is the set of minterms that can only be distinguished by fanin wire (i, j) but not by any of the fanin wires of node j.

Now back to the example in Figure 2, the area-minimizing SPFD computation might result in

$$R_{3,4}^{\text{area}} = \{(m_{001}, m_{000}), (m_{111}, m_{110}), (m_{111}, m_{000})\}.$$

Note that the pair (m_{111}, m_{000}) can be distinguished by w_1 or w_2, and therefore it does not have to be assigned to the SPFD of w_3. Thus, our SPFD computation for R2 gives the SPFD for w_3,

$$R_{3,4}^{\text{robust}} = \{(m_{001}, m_{000}), (m_{111}, m_{110})\},$$

which is the minimum set of minterms that have to be distinguished by w_3. Obviously, function of L_5 does not satisfy $R_{3,4}^{area}$ and therefore we lose the opportunity of replacing the highly critical wire w_3 in area-minimizing SPFD computation. On the other hand, the new SPFD computation in R2 makes such a replacement possible.

During the SPFD computation, if we find a wire (i, j) with empty SPFD with respect to the primary outputs, then that wire is redundant and can be removed from the network. After this removal, if node i had only a single fanout and it is not a primary output then node i can be removed from the network. The removal of wires and nodes during the robust rewiring helps to minimize the area of the network.

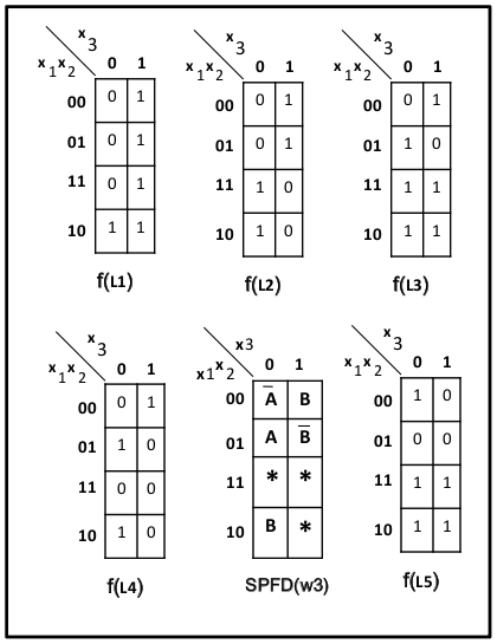

Figure 3: Functions of LUT in the network

4. EXPERIMENTAL RESULTS

We have implemented the R2 algorithm and tested it on MCNC [2] and IWLS [12] benchmarks. We consider the combinational logic in the benchmark circuits. Using the same settings as [5], a circuit is synthesized, optimized and mapped to 5-LUTs using SIS [3] with the following command:

> "eliminate 2 gkx -ac; simplify -d; xl_part_coll -m -g 2; xl_coll_ck; xl_partition -m; simplify; xl_imp; xl_partition -t; xl_cover -e 30 -u 200; xl_coll_ck -k;".

Taking the mapped combinational circuit, we first perform in-place reconfiguration (IPR) using the algorithm in [20] to reconfigure the LUTs for logic masking. The resulting circuit is referred as "IPR" in our comparison (see Table 1). Then, we perform a fault simulation to obtain the criticality of the wires (edges) and LUTs (nodes) assuming that there is one configuration bit in each wire. The result of this simulation is used as a guidance to choose wires during the optimization where 2,000 random input vectors are used in each run of the fault simulator. The total criticality for a circuit is computed by adding the criticality of all

Table 1: Summary of results

	Total criticality			Wire#			Node#		
Circuit	IPR	Yamashita	R2	IPR	Yamashita	R2	IPR	Yamashita	R2
Apex6	839.95	704.82	646.34	897	896	889	329	329	328
x3	1167.7	1088.32	1081	935	914	887	340	338	333
example2	1153	932.45	921	451	425	449	190	186	190
C1908	1874.32	1971.13	1794.03	430	428	430	136	136	136
too_large	1230.5	839,47	778.79	883	875	882	227	227	227
apex7	669.87	575.96	492.8	294	284	273	122	121	119
frg2	3843.98	3396.59	2977.17	1319	1234	1193	482	481	477
ttt2	369.77	364.87	310.75	238	234	211	77	77	72
dalu	2316.32	2316.85	2139.97	1381	1230	1254	404	390	393
apex2	8125.03	7641.13	5958.61	4743	4663	4718	1162	1162	1159
x4	948.3	845.39	801.61	598	570	491	234	233	213
t481	2849.17	2449.03	2218.67	1722	1688	1721	416	416	416
seq	10162.70	9231.63	8008.22	4490	4409	4468	1109	1108	1107
pci_spoci_ctrl	1801.05	2476.14	1722.06	1438	1359	1422	431	425	429
ss_pcm	418.91	377.70	356.41	466	465	457	226	226	226
usb_phy	471.53	409.00	406.19	552	549	547	278	278	278
mem_ctrl	31540.17	19180.03	18730.20	16031	14580	12844	5182	4633	4576
GEOMEAN	1756.34	1568.11	1410.84	1052.34	1016.26	996.43	353.39	349.39	345.76
	Estimated MTTF Ratio			Ratio of Num. of Wires			Ratio of Num. of Nodes		
	1.00	1.12	1.24	1.00	0.97	0.95	1.00	0.99	0.98
		1.00	1.11		1.00	0.98		1.00	0.99

wire and LUT configuration bits, where the criticality of a configuration bit is the percentage of random input vectors that showed a difference in some primary output when the configuration bit at the wire or the LUT was flipped. Two different rewiring algorithms, the area-minimizing rewiring of [5] (the resulting circuit is referred as "Yamashita") and our proposed R2 are performed. After the rewiring, the fault simulator with more (100,000 in our experiment) random input vectors are used to calculate the final criticality of wires and LUTs in the resynthesized circuits. The detailed comparisons are shown in Table 1.

According to [20], the IPR algorithm can achieve up to 50% criticality reduction compared to the state of the art logic synthesis tool ABC [13]. R2 reduces the total criticality of the IPR-optimized circuit (called "IPR" in Table 1), by 20% while also reducing the total number of wires and nodes by 5% and 2%, respectively. Compared with Yamashita's rewiring, R2 reduces the total criticality by 11%. It is interesting that our R2 results in fewer wires and nodes compared with the area-minimizing Yamashita's algorithm. The reason might be due to the heuristic of R2 that re-assigns the SPFD to nodes with high criticality in exchange for more replacement opportunity, and therefore more wires/nodes are removed. The running time of R2 is comparable to that of Yamashita's rewiring approach.

In Table 1, we also compare the estimated MTTF, which is the commonly used system-level metric for robustness. Based on [20], MTTF is inversely proportional to the fault rate, i.e., the average criticality of all configuration bits of the wires and the LUTs. Therefore MTTF is inversely proportional to the total criticality, assuming the same FPGA chip (with the same amount of configuration bits) and the uniform distribution of single soft error. Our MTTF ignores the reduction of LUTs (nodes) and wires, and therefore it is underestimated. We report MTTF normalized with respect to that for IPR. The criticality reduction by R2 is equivalent to 24% and 11% MTTF increase compared to IPR and Yamashita respectively. It should be noted that Yamshita method also increases the MTTF by 12%, this is becuase Yamashita method reduces the number of LUTs and wires in the circuit and this can lead to a reduction in the total criticality of the circuit.

5. CONCLUSIONS AND FUTURE WORK

We have presented R2, a robust rewiring algorithm that performs wire replacement in order to reduce the criticality of the interconnects in an FPGA-based design. The preliminary experimental results show that the proposed algorithm increases MTTF (mean time to failure) by 24% compared with IPR [20], a recently proposed LUT reconfiguration approach. This clearly shows the need of taking Boolean flexibility into consideration for fault tolerance in order to fully explore the design space.

Compared to both IPR and exiting rewiring for area reduction, R2 reduces area as it reduces LUT and wire numbers in addition to increasing MTTF. Note that our reported MTTF increase is underestimated, as we do not consider area reduction by R2 in MTTF calculation. Furthermore, we assume that there is only one configuration bit per wire in our experimental settings. This may also underestimate MTTF improvement by R2.

In the future, we will further test R2 with a more fine-grained physical-level fault model, where the increased number of configuration bits per wire potentially lead to more improvement. It is interesting to see the improvement even at logic level, and since R2 leads to lower area and reduction in the number of wires, it is quite possible that this could lead to better routability and timing. Our current SPFD calculation uses BDDs and it cannot accommodate bigger circuits. We plan to to do a SAT-based SPFD calculation which would allow us to rewire bigger circuits with poten-

tially more room for rewiring.

6. REFERENCES

[1] H. Savoj and R. K. Brayton. The Use of Observability and External Don't Cares for Simplification of Multi-Level Networks. In *Proc. Design Automation Conf*, 1990.

[2] S. Yang. Logic synthesis and optimization benchmarks, Version 3.0. Technical report, Microelectronics Center of North Carolina (MCNC), 1991.

[3] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

[4] Ellen M. Sentovich, Vigyan Singhal, and Robert K. Brayton. Multiple boolean relations. In *International Workshop on Logic Synthesis*, 1993.

[5] S. Yamashita, H. Sawada, and A. Nagoya. A new method to express functional permissibilities for LUT based FPGAs and its applications. In *Proc. Int. Conf. on Computer Aided Design*, 1996.

[6] J.-M. Hwang, F.-Y. Chiang, and T.-T. Hwang. A Reengineering Approach to Low Power FPGA Design Using SPFD. In *Proc. Design Automation Conf*, 1998.

[7] S. Sinha, R and K. Brayton. Implementation and Use of SPFDs in Optimizing boolean Networks. In *Proc. Int. Conf. on Computer Aided Design*, 1998.

[8] P. Chong, Y. Jiang, S. Khatri, F. Mo, S. Sinha, and R. Brayton. Don't Care Wires in Logical/Physical Design. In *International Workshop on Logic Synthesis*, 2000.

[9] J. Cong, Y. Lin, and W. Long. SPFD-Based Global Rewiring. In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2002.

[10] J. Cong, Y. Lin and W. Long. A New Enhanced SPFD Rewiring Algorithm. In *Proc. Int. Conf. on Computer Aided Design*, 2002.

[11] P. Graham, M. Caffrey, J. Zimmerman, D. E. Johnson, P. Sundararajan, and C. Patterson. Consequences and categories of SRAM FPGA configuration SEUs. In *Proceedings of the Military and Aerospace Applications of Programmable Logic Devices* , 2003

[12] IWLS 2005 benchmarks. In *http://iwls.org/iwls2005/benchmarks.html*.

[13] ABC: A system for sequential synthesis and verification. In *http://www.eecs.berkeley.edu/ alanmi/abc/*.

[14] Xilinx TMRTool. Product brief. In *Xilinx Corporation*, 2006.

[15] H. Asadi, M.B. Tahoori, B. Mullins, D. Kaeli, and K. Granlund. oft Error Susceptibility Analysis of SRAM-Based FPGAs in High-Performance Information Systems. In *IEEE Transactions on Nuclear Science (TNS)*, December 2007.

[16] S. Krishnaswamy, S.M. Plaza, I.L.Markov and J.P.Hayes. Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation. In *Proc. Int. Conf. on Computer Aided Design*, 2007.

[17] S. Mukherjee. *Architecture design for soft errors*. Morgan-Kaufman, 2008.

[18] Y. Hu, Z. Feng, R. Majumdar, and L. He. Robust FPGA resynthesis based on fault tolerant boolean matching. In *Proc. Int. Conf. on Computer Aided Design*, 2008.

[19] K. Chapman and L. Jones. SEU Strategies for Virtex-5 Devices. In *Xilinx Corporation, XAPP864*, 2009.

[20] Z. Feng, Y. Hu, R. Majumdar, and L. He. IPR: In-Place Reconfiguration for FPGA Fault Tolerance. In *Proc. Int. Conf. on Computer Aided Design*, 2009.

[21] Ju-Yueh Roy Lee, Yu Hu, Rupak Majumdar, Lei He, and Minming Li. Fault-tolerant resynthesis with dual-output luts. In *Proc. Asia South Pacific Design Automation Conf.*, 2010.