On Power and Fault-Tolerance Optimization in FPGA Physical Synthesis *

Manu Jose[1] Yu Hu[2] Rupak Majumdar[1,3]
1. Computer Science Department, University of California, Los Angeles
2. Electrical and Computer Engineering Department, University of Alberta
3. Max-Planck Institute for Software Systems, Kaiserslautern

ABSTRACT

Power and fault tolerance are deemed to be two orthogonal optimization objectives in FPGA synthesis, with independent attempts to develop algorithms and CAD tools to optimize each objective. In this paper, we study the relationship between these two optimizations and show empirically that there are strong ties between them. Specifically, we analyze the power and reliability optimization problems in FPGA physical synthesis (i.e., packing, placement, and routing), and show that the intrinsic structures of these two problems are very similar. Supported by the post routing results with detailed power and reliability analysis for a wide selection of benchmark circuits, we show that with minimal changes —fewer than one hundred lines of C code— an existing power-aware physical synthesis tool can be used to minimize the fault rate of a circuit under SEU faults. As a by-product of this study, we also show that one can improve the mean-time-to-failure by 100% with negligible area and delay overhead by performing fault-tolerant physical synthesis for FPGAs. The results from this study show a great potential to develop CAD systems co-optimized for power and fault-tolerance.

Categories and Subject Descriptors

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

General Terms

Design, Reliability, Performance

Keywords

Fault Tolerance, Low Power, Physical Synthesis, FPGA

1. INTRODUCTION

Optimizing for power is one of the main criteria in FPGA synthesis, with an extensive body of research and commercial tool support in the FPGA CAD flow. More recently, there is a growing interest in optimizing FPGA designs with respect to faults arising out of soft errors, motivated by increasing vulnerability of SRAM-based FPGAs to soft errors due to aggressive CMOS scaling. Various techniques to minimize fault rates have emerged to increase the mean time to failure (MTTF) for applications such as enterprise servers and Internet routers [22].

Power and fault tolerance are studied as two unrelated, or even mutually competitive, optimization objectives in FPGA synthesis. For instance, fault tolerance based on triple modular redundancy (TMR) requires over $3\times$ power overhead. As a result, a common problem formulation for fault-tolerance optimization is to maximize the robustness (e.g., MTTF) under a specific power constraint [18] or minimize the power overhead under a target fault rate. Compared to power optimizations, synthesis for fault-tolerance is less mature, both in the scope and scalability of tools and in commercial integration in CAD tools.

In this paper, we empirically demonstrate several, perhaps surprising, intrinsic connections between power and fault rate optimizations. First, we experimentally show that the existing power-aware physical synthesis already simultaneously optimizes for fault tolerance, and provide a theoretical justification. Further, we show we can minimally modify an existing power-aware physical synthesis tool to optimize for fault tolerance: we obtain a consistent improvement in MTTF (100% MTTF improvement on average) by slightly changing the interpretation of the cost function in an existing power-aware physical synthesis tool. Overall, this required fewer than 100 lines of change in the source code! Practically, our results indicate that optimizing for fault tolerance can take advantage of the relatively mature power optimization techniques.

Our empirical observations are based on a case study in FPGA physical synthesis (clustering, placement and routing) exploring the relationship between the two optimizations. The rationale of using physical synthesis to study power-reliability relationship is two-fold: (a) physical synthesis has been proven to be effective for power optimization [19, 15]; (b) physical (layout) information is necessary to accurately capture the bit-level SEU effect, which can be used to guide the fault-tolerant optimization.

Unlike power-aware physical synthesis [19], a thorough study of fault-tolerant physical synthesis was absent so far. Most of the existing work has been focused on redundancy-based methods (e.g., TMR [24] or partial TMR [16]). Recent attention has been moved to sensitivity-based methods, which consider the functional or physical sensitivity of a configuration bit to SEU and try to reduce the overall sensitivity of a design with minimal overhead. There are discrete pieces of research using sensitivity-based fault tolerance, e.g., logic masking-based re-synthesis [14, 12], defect-aware placement [4], and physical-sensitivity-aware routing [13,

*This research was sponsored in part by the DARPA grant HR0011-09-1-0037, the NSF grant CCF-0953994, and an NSERC discovery program.

.

17]). However, there has been no recorded, existing systematic study on the effect of fault-tolerance optimizations at various physical synthesis steps.

In this paper, we perform a cross-layer study of sensitivity-based fault tolerance on conventional physical synthesis flow. Following the methodology used in the power-aware physical synthesis [19], our fault-tolerant physical synthesis incorporates the SEU-sensitivity into the cost functions in all physical synthesis phases. Instead of devising a new set of cost functions for fault tolerance, we discover the intrinsic connection between power-aware and fault-tolerant physical synthesis, which allows us to employ the identical structures of the cost function used in power-aware physical synthesis to enhance fault tolerance. So instead of switching activity, if we feed SEU-sensitivity into the cost function of power-aware physical synthesis optimization we get fault tolerance optimization. This observation further suggests a great potential of simultaneously optimizing both power and fault tolerance. Our fault-tolerant physical synthesis shows that placement is the most effective phase for fault-tolerant optimization, and the gains obtained from different physical synthesis phases are cumulative.

Our study provides initial evidence of the intrinsic connection between power and fault tolerance, and suggests the potential of simultaneous optimization for these two objectives, perhaps also in other phases of design.

2. EXPERIMENTAL FRAMEWORK

2.1 Fault, Delay, and Power Model

We assume the *single fault* model, i.e., at most one single event upset (SEU) occurs in a clock period, based on the real SER in commercial SRAM-based FPGAs [11]. Across an FPGA, we assume that the SER on all configuration bits is uniformly distributed, i.e., all bits have an identical probability to be faulty. The same model has been used in recent work [12, 14]. We consider SEUs on configuration bits of both LUTs and routing. An SEU that occurs in an LUT configuration bit results in the flip of the logic value and consequently changes the logic function encoded by the LUT.

Given an FPGA-based design, a detailed analysis of the impact of faults at the physical level is expensive. For example, the logic value associated with a faulty routing signal due to a bridging fault [5] 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*. 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., [6]) by replacing our fault simulator with a more sophisticated one.

To quantify the impact of the SEU of an configuration bit, we define its *sensitivity* as follows. The sensitivity, $s(c)$, of a configuration bit c is the percentage of primary input vectors that cause erroneous values at primary outputs of the circuit due to the flip of the logic value of c. Intuitively, the sensitivity of a configuration bit shows how likely the flip of this bit is observed at the circuit outputs. The higher the sensitivity of a configuration bit is, the more input vectors may sensitize the SEU that occurs at this bit. The overall *fault rate* F_R of a circuit is

$$F_R = \frac{\sum \{s(c) \mid \text{configuration bit } c\}}{\text{total number of configuration bits}} \quad (1)$$

Using the above fault model, we perform a fault simulation using Monte Carlo sampling to estimate the sensitivity of a single bit and then evaluate the overall fault rate of the entire circuit. In all experimental results reported in the rest of this paper, the fault rate of a circuit is evaluated based on a physical-level fault simulation which returns the sensitivity of each configuration bit after placement and routing by VPR [9, 10, 21]. The fault-tolerance of a circuit is often measured using *mean time to failure* (MTTF), which is inversely proportional to the fault rate [23, 12, 14].

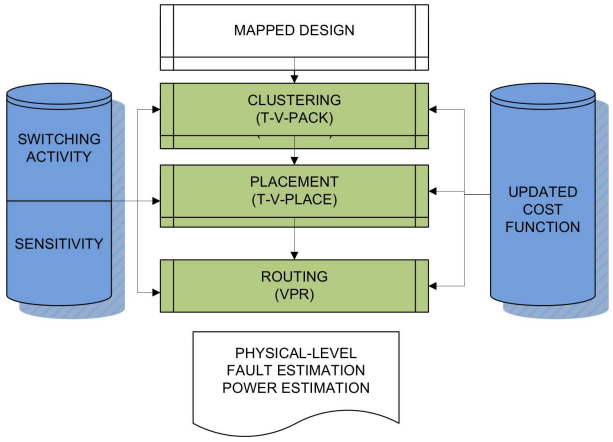

Figure 1: FPGA Synthesis Flow.

We use the same delay and power model as [19], and only dynamic power is considered in this work. The switch activities are collected using random simulation. In all experimental results reported in the rest of this paper, the power and delay values are produced based on resistance and capacitance extracted from designs after placement and routing.

2.2 CAD Flow and Experimental Settings

The baseline FPGA CAD flow is as shown in Figure 1. A design is first synthesized and mapped to LUTs using Berkeley ABC [1]. Then three physical synthesis phases, including clustering (using T-VPACK [3, 10]), placement and routing (using VPR [9]), are performed. For the fault-tolerant flow, a logic-level fault simulation with 10K random vectors is performed in the mapped circuit, and the sensitivity of each LUT configuration bit and each logic connection is extracted. For power-aware flow, the switching activity of each net is obtained by the Monte Carlo simulation with 10K random vectors. 18 circuits from IWLS [2] and MCNC [25] benchmarks are tested. The characteristics of the benchmark circuits are shown in Table 1.

Table 1: Characteristics of benchmark circuits

Name	CI#	CO#	NETS#	Name	CI#	CO #	NETS#
dalu	75	16	1497	spi	276	274	4592
pci_spoci_ctrl	85	73	1813	ss_pcm	106	96	428
C5315	178	123	1798	steppermotordrive	29	29	237
des	256	245	5195	systemcdes	322	255	3765
pdc	16	40	13414	C499	41	32	280
aes_core	789	659	31606	C7552	207	108	2110
des_area	368	192	7613	k2	45	45	2761
sasc	133	129	703	pair	173	137	1786
simple_spi	148	144	1046	tv80	373	391	12012

An island-style FPGA architecture is assumed in this work. This architecture includes an array of clustered logic blocks (CLB)s interconnected by programmable routing. Each CLB has I inputs and N outputs. It contains N basic logic elements (BLEs), each of which includes an LUT and a FF. In our experiments, we test the

Table 2: Architectures used in the experiments

name	cluster_size(N)	LUT_input#	cluster_input#(I)
4x4	4	4	10
4x10	10	4	22
6x4	4	6	15
6x10	10	6	33

algorithms under the four architecture settings shown in Table 2.

3. CLUSTERING

3.1 Review of Power-Aware Clustering

A popular framework of FPGA clustering algorithm is a greedy approach [8]. It iteratively selects a LUT, which is used as the seed of a new cluster. Based on this seed LUT, other LUTs are added into this cluster according to an *attractive function* between these LUTs and the seed LUT.

The power-aware clustering, i.e., P-T-VPack, [19] attracts high-activity nets inside CLBs by penalizing the global interconnect (nets outside CLBs) with high switching activities. Specifically it modifies the attraction function in the following ways:

1. An LUT whose input and output wires have the highest switching activity is selected as the seed LUT.

2. Given a cluster C containing the seed LUT, the attraction function that selects the remaining LUTs to be packed into this cluster is defined as

$$AF_p(B) = \alpha \cdot Crit(B) + (1-\alpha) \cdot \frac{1}{G}[\beta \cdot (\sum\{Weight(i) \mid i \in Nets(B) \cap Nets(C)\}) + (1-\beta) \cdot \frac{1}{f_{Avg}} \sum\{f(i) | i \in Nets(B) \cap Nets(C)\}]$$

where $Crit(B)$ measures how close LUT B is to be on the critical path, $Net(B)$ is the set of nets connected to LUT B, $Net(C)$ is the set of nets connected to those LUTs already selected in cluster C, $f(i)$ is the estimated switching activity of net i, f_{Avg} is the average switching activity of all nets in the circuit, α and β are user-defined constants and G is a normalizing factor. A more detailed explanation of this formula can be found in Section 5.1 of [19].

3.2 Morphing P-T-VPack for Fault Tolerance

In the optimization for fault-tolerance, the fault rate at a global interconnect is proportional to the average sensitivity of SRAMs along its routing. Therefore, to minimize the overall fault rate (shown in (2)), a fault-tolerant clustering should encapsulate interconnects with high sensitivities inside CLBs, which results in a very similar objective to power-aware clustering. Based on this observation, we make the following modification in the power-aware attraction function

1. An LUT whose input and output wires have the highest sensitivity is selected as the seed LUT.

2. In $AF_p(B)$, we replace the switching activity $f(i)$ of a net i with its sensitivity $s(i)$. Note that the sensitivity of a net is the sum of sensitivities of all source-to-sink edges and the source node.

Such modification of the attraction function requires very small changes of the source code in a power-aware clustering. For a modularized implementation, changes in the source code are not necessary since the switching activity annotated network is given as an input file and it can be replaced by the one with annotated sensitivity values. In the rest of this paper, this adapted Clustering for fault tolerance is called *F-T-VPack*. In (2), we experimentally found the best α and β values to be 0 and 0.6, respectively.

Table 3: Fault-tolerant clustering results

	Total sensitivity		Crit_Delay(s)	
Arch	T-VPack	F-T-VPack	T-VPack	F-T-VPack
4x4	672.45	604.62 (-10%)	4.34E-08	4.33E-08
4x10	633.47	430.84 (-32%)	3.80E-08	4.08E-08
6x4	466.15	429.98 (-8%)	3.61E-08	3.52E-08
6x10	402.12	259.50 (-36%)	3.24E-08	3.24E-08

Table 3 shows the geomean of the total sensitivity[1] and critical path delay for 18 benchmark circuits under each architecture setting. For the cluster size of 4, F-T-VPack reduces the fault rate by 8%-10%, while for the cluster size of 10, F-T-VPack reduces the fault rate by over 30%. Intuitively, the larger the cluster size is, the more highly-sensitive LUTs can be encapsulated in one CLB and therefore the more effective F-T-VPack is. While F-T-VPack reduces the fault rate, it generally does not degrade the performance, i.e., the critical path delay of the circuits synthesized by F-T-VPack is comparable to that by the timing-driven clustering, T-VPack. In addition, F-T-VPack results in almost the same number of clusters as T-VPack, which is not shown in detail due to the space limit.

3.3 Cross Domain Optimization

With the success of the adaption from power-aware clustering to fault-tolerant one, this subsection studies the potential of cross optimization between power and fault tolerance, i.e., using the power-aware clustering to optimize a design and measuring the change of the fault rate using MTTF, or vice versa.

The only difference between the power-aware clustering and the fault-tolerant clustering is the third term in the attraction function AF_p, where the switching activity and sensitivity values are used for power and fault tolerance respectively. Although these two values have different physical meaning, it is worthwhile to investigate their inherent relationship. Essentially, the switching activity of a node (i.e., an LUT) in the Boolean network is determined by its SDC (satisfiability don't-cares), which constrains the input patterns that can sensitize the node. On the other hand, the sensitivity of a node is determined by both its SDC and ODC (Observability don't-cares), since a bit flip in a node may be masked by either the SDC or the ODC of this node and will not propagate to the primary outputs. Therefore, the switching activity and sensitivity are two closely related values. Particularly, for those nodes close to the primary outputs or latch inputs, these two values should be highly correlated since their observability's are 100% and SDC becomes the only factor to determine their both values.

Such intrinsic connection between the switching activity and sensitivity values implies the possibility of simultaneous optimization for both power and fault-tolerance in the clustering, i.e., we expect improvement of fault tolerance by taking the original objective function AF_p in power-aware clustering without any changes!

To verify this hypothesis, two clustering algorithms, i.e., P-T-VPack and F-T-VPack, are performed on 18 benchmark circuits for four architectures. Table 4 shows the average power reduction and MTTF increase, both compared with T-VPack. Since our optimization does not change the area of an FPGA chip, MTTF is inversely proportional to the total sensitivity based on equation (2). In the table, the positive (negative) percent means we reduce (in-

[1]Assuming the same FPGA device, the fault rate is proportional to the total sensitivity based on (2).

Table 4: Simultaneous power and fault tolerance optimization in clustering (compared with T-VPack)

	P-T-VPack		F-T-VPack	
Arch	Power Reduction	MTTF Increase	Power Reduction	MTTF Increase
4x4	1%	4%	-11%	11%
4x10	10%	2%	12%	47%
6x4	5%	1%	5%	8%
6x10	5%	4%	4%	55%

crease) power or increase (reduce) MTTF. Using P-T-VPack, we can consistently simultaneously reduce power and increase MTTF. However, the increase in MTTF obtained by P-T-VPack is significantly less than that obtained by F-T-VPack. On the other hand, F-T-VPack almost always simultaneously reduces power and increases MTTF, except for 4x4 architecture. It is interesting to see that F-T-VPack gives higher power reduction for 4x10 architecture compared with P-T-VPack, due to the interconnect uncertainty shown in [20].

4. PLACEMENT

4.1 Review of Power-Aware Placement

The power-aware placement, P-T-VPlace [19], follows the simulated annealing-based placement algorithm used in VPR [9]. The P-T-Vplace cost function includes the following three components:

1. Wiring cost: the sum of the bounding box dimensions of all the nets. That is, if there are N_{nets}, and $bb_x(i)$ and $bb_y(i)$ are the x and y dimensions of the bounding box of net i,

$$WiringCost = \sum_{i=1}^{N_{nets}} q(i) \cdot [bb_x(i) + bb_y(i)]$$

where $q(i)$ is used to scale the bounding boxes to better estimate wire length for nets with more than 3 terminals,

2. Timing cost: the sum of the product of the delay and the timing slack of nets.

$$TimingCost = \sum_{\forall i,j \in circuit} Delay(i,j) Crit(i,j)^{CE}$$

where the $Delay(i,j)$ is the estimated delay of the connection from source i to sink j, CE is a constant, and $Crit(i,j)$ is an indication of how close to the critical path is the connection.

3. Power cost: the sum of the product of the bounding box size and the switching activity of nets.

$$PowerCost = \sum_{i=1}^{N_{nets}} q(i) \cdot [bb_x(i) + bb_y(i)] \cdot f(i)$$

where $f(i)$ is the switching activity of net i. The total cost of a placement is the sum of the wiring cost and the timing cost for all the nets and is given by :

$$\Delta C = \alpha \cdot \frac{\Delta\, TimingCost}{PrevTimingCost} + (1-\alpha) \cdot \frac{\Delta\, WiringCost}{PrevWiringCost} + \gamma \cdot \frac{\Delta\, PowerCost}{PrevPowerCost}$$

where $PrevTimingCost$, $PrevWiringCost$ and $PrevPowerCost$ are the auto-normalizing factors which are computed every temperature, and α is an user defined constant used to control the relative importance of these two factors.

Table 5: Fault-tolerant placement results

	Total sensitivity		Crit_Delay(s)	
Arch	T-VPlace	F-T-VPlace	T-VPlace	F-T-VPlace
4x4	672.45	535.44 (-20%)	4.34E-08	4.24E-08
4x10	633.47	441.02 (-30%)	3.81E-08	3.86E-08
6x4	466.15	336.94 (-28%)	3.62E-08	3.68E-08
6x10	431.02	272.74 (-37%)	3.39E-08	3.32E-08

Table 6: Simultaneous power and fault tolerance optimization in placement (compared with T-VPlace)

	P-T-VPlace		F-T-VPlace	
Arch	Power Reduction	MTTF Increase	Power Reduction	MTTF Increase
4x4	18%	13%	18%	26%
4x10	28%	27%	26%	44%
6x4	25%	14%	20%	38%
6x10	31%	46%	32%	58%

4.2 Morphing P-T-VPlace for Fault Tolerance

To reduce the fault rate of the circuit, we have to place the highly-sensitive nets which connects between the clusters as close as possible to minimize the total sensitivity along the global interconnect.[2] Essentially, this is the same intuition behind the power-aware placement. Therefore, our fault-tolerant placement, namely *F-T-VPlace*, takes the same cost function as *P-T-VPlace* and replaces the switching activity term, $f(i)$, with the sensitivity of a net. Note that the bounding box terms in this cost function effectively models the number of configuration bits required by a net. We experimentally decide that α=0.5 and $\gamma = 1$, which give the best results. Like the adaption in the clustering, such a replacement requires minor changes in the source code for a modularized implementation.

Table 5 shows the geomean of the total sensitivity and critical path delay for 18 benchmark circuits under each architecture setting. For different architectures, *F-T-VPlace* consistently reduces the fault rate by 20%-30%. Similar to *F-T-VPack*, *F-T-VPlace* does not degrade the performance, i.e., the critical path delay of the circuits synthesized by *F-T-VPlace* is comparable to that by the timing-driven placement, *T-VPlace*.

4.3 Cross-Domain Optimization

This section further examines the possibility of simultaneously optimizing power and fault tolerance in the placement. Particularly, we perform P-T-VPlace and F-T-Vplace to measure both power and MTTF. Table 6 shows the power reduction and the MTTF increase achieved by these two placement algorithms, both compared with T-VPlace on 18 benchmark circuits for four architectures. In the table, the positive (negative) percent means we reduce (increase) power or increase (reduce) MTTF. Overall, both F-T-VPlace and P-T-VPlace simultaneously reduces power and increases MTTF across all architecture settings. It is interesting that F-T-VPlace achieves comparable power reduction as P-T-VPlace. Compared with the increase in MTTF obtained by F-T-VPlace, P-T-VPlace is 30%-40% less effective.

5. ROUTING

[2]Although the internal configuration bits inside CLBs are also sensitive to SEU, the global interconnect tends to require much larger number of configuration bits and therefore it increases overall SEU sensitivity.

5.1 Review of Power-Aware Routing

P-T-VRoute follows the Pathfinder algorithm [7] used in VPR router [9]. Each net is iteratively routed based on the weight assigned to routing resource components. As pointed out in [19], routing is the least effective phase for power optimization.

The power-aware router, P-T-VRoute [19], uses the cost function which has a delay term and a congestion term to evaluate a routing track n while forming a connection from source i to sink j as follows:

$$\begin{aligned} Cost(n) = \quad & Crit(i,j) \cdot delay_{Elmore}(n) + \\ & (1 - Crit(i,j)) \cdot [ActCrit(i) \cdot cap(n) + \\ & (1 - ActCrit(i)) \cdot b(n) \cdot h(n) \cdot p(n)] \end{aligned}$$

The first term (delay) is the product of the delay of node n and $Crit(i,j)$. The second term (power) includes $cap(n)$, the capacitance associated with routing resource node n, and $ActCrit(i)$, the activity criticality of net i. The third term (congestion), which has more weight when the criticality is low, has three components: $b(n)$ is the "base cost", $h(n)$ is the historical congestion cost, and $p(n)$ is increased gradually as the algorithm progresses to discourage node sharing, allowing the algorithm to produce a legal solution.

5.2 Morphing P-T-VRoute for Fault Tolerance

Similar to our adaption of power-aware placement, we replace the activity criticality term $ActCrit(i)$ by *sensitivity criticality*, which is a normalized sensitivity defined in the same form as $ActCrit(i)$. We keep the term $cap(n)$ since the accumulated capacitance is approximately proportional to the number of configuration bits. Such an approximation allows the minimal change of the source code when morphing a power-aware routing to a fault-tolerant version.

Table 7 shows the geometric mean of the total sensitivity and critical path delay for 18 benchmark circuits under each architecture setting. For different architectures, F-T-VRoute consistently reduces the fault rate by 2%-14%. Similar to F-T-VPack and F-T-VPlace, F-T-VRoute does not degrade the performance, i.e., the critical path delay of the circuits synthesized by F-T-VRoute is comparable to that by the timing-driven routing, T-VRoute. Compared to the placement and packing phases, fault-tolerant routing gives less fault rate reduction because of the complicated routing resource constraints.

Table 7: Fault-tolerant routing results

	Total sensitivity		Crit_Delay(s)	
Arch	T-VRoute	F-T-VRoute	T-VRoute	F-T-VRoute
4x4	672.45	575.74 (-14%)	4.34E-08	4.23E-08
4x10	633.47	613.69 (-3%)	3.80E-08	3.89E-08
6x4	466.15	450.79 (-3%)	3.61E-08	3.53E-08
6x10	431.02	421.9 (-2%)	3.38E-08	3.32E-08

5.3 Cross-Domain Optimization

P-T-VRoute and F-T-VRoute are individually performed on 18 benchmark circuits for four architectures. Table 8 shows the average power reduction and increase in MTTF, both compared with the original timing-driven router in VPR. In the table, the positive (negative) percent means we reduce (increase) power and increase (reduce) MTTF. Using the F-T-VRoute, we can consistently simultaneously reduce power and increase MTTF. Interestingly, the power reduction obtained by F-T-VRoute is comparable to that obtained by P-T-VRoute. On the other hand, simultaneous optimization for power and fault tolerance is not observed in P-T-VRoute except for 4x4 architecture.

Table 8: Simultaneous power and fault tolerance optimization in routing (compared with T-VRoute)

	P-T-VRoute		F-T-VRoute	
Arch	Power Reduction	MTTF Increase	Power Reduction	MTTF Increase
4x4	11%	2%	1%	17%
4x10	7%	-7%	15%	3%
6x4	10%	-6%	16%	3%
6x10	1%	-10%	11%	2%

Table 9: Combined fault-tolerant physical synthesis phases (power reduction and MTTF increase compared with the corresponding timing-driven flow (T-VPack and VPR)

Arch	4x4		4x10		6x4		6x10	
	MTTF increase	Power reduction	MTTF increase	Power reduction	MTTF increase	Power reduction	MTTF increase	Power reduction
CBB	11%	-11%	47%	12%	8%	5%	55%	4%
BPB	26%	18%	44%	26%	38%	20%	58%	32%
BBR	17%	1%	3%	15%	3%	16%	2%	11%
CPB	61%	18%	92%	26%	62%	20%	119%	21%
CBR	23%	1%	49%	15%	27%	16%	56%	11%
BPR	45%	20%	52%	23%	57%	23%	68%	21%
CPR	81%	25%	114%	29%	80%	30%	122%	25%

6. COMBINED RESULTS

Finally, we study the interaction among the above three physical synthesis phases in order to answer the following two questions:

- Which phases are effective for simultaneously optimizing power and fault tolerance?
- Is the optimization obtained by different phases cumulative? I.e., what is the overlap between different combinations of optimization?

Table 9 shows the power reduction and MTTF increase (compared with the corresponding timing-driven flow) obtained by fault-tolerant physical synthesis. All different combinations of F-T-VPack (denoted by "C"), F-T-VPlace (denoted by "P") and F-T-VRoute (denoted by "R") are compared in the table. Note that "B" denotes the baseline algorithm (T-VPack, T-VPlace or T-VRoute). For example, the row "BPR" denotes the the following CAD flow: T-VPack, F-T-VPlace and F-T-VRoute. From this table, we have the following observations:

- Among the three physical synthesis phases, placement is most effective for optimizing fault tolerance across all architecture settings. Packing is the second most effective and achieves comparable fault rate reduction to placement for architectures with large cluster size (4x10 and 6x10). Routing is less effective for fault tolerant optimization by itself.
- The fault tolerance optimization obtained from different synthesis phases is cumulative and there is little overlap between different phases. Sometimes the combination of multiple phases achieves more improvement than the sum of the gains obtained by individuals, e.g., the combined clustering and placement increases the MTTF by 61% while the clustering and placement individually increases the MTTF by only 11% and 26%, respectively, for 4x4 architecture.
- Using our fault-tolerant physical synthesis, we can almost consistently obtain simultaneous reduction for power and increase

Table 10: Combined power-aware physical synthesis phases (Power reduction and MTTF increase compared with the corresponding timing-driven flow (T-VPack and VPR)

Arch	4x4		4x10		6x4		6x10	
	MTTF increase	Power reduction	MTTF increase	Power reduction	MTTF increase	Power reduction	MTTF increase	Power reduction
CBB	1%	4%	1%	2%	5%	1%	5%	4%
BPB	18%	13%	28%	27%	25%	14%	31%	46%
BBR	11%	2%	7%	-7%	10%	-6%	1%	-10%
CPB	21%	23%	38%	32%	25%	26%	43%	64%
CBR	6%	9%	12%	6%	12%	11%	8%	7%
BPR	29%	30%	43%	29%	33%	29%	33%	43%
CPR	31%	38%	40%	39%	36%	41%	42%	65%

in MTTF. Among three individual synthesis phases, placement is the most effective one for simultaneous optimization of power and fault tolerance. However, the power reduction obtained by fault-tolerant physical synthesis is often not cumulative, i.e., F-T-VPlace reduces the power by 32% but the combined F-T-VPack, F-T-VPlace and F-T-VRoute only give 25% power reduction for 6x10 architecture.

- The combination of the three phases gives 100% increase in MTTF on average over the four architectures and 27% reduction in power which is the best result among all these combinations.

To further examine the potential of using existing power-aware physical synthesis for simultaneous optimization, Table 10 shows power reduction and MTTF increase (compared with the corresponding timing-driven flow) obtained by the power-aware physical synthesis. All different combinations of P-T-VPack (denoted by "C"), P-T-VPlace (denoted by "P") and P-T-VRoute (denoted by "R") are compared in the table.

- Although P-T-VRoute does not simultaneously optimize for power and fault tolerance, the combination of P-T-VRoute and other power-aware synthesis phases gives consistent simultaneous power reduction and increased MTTF.
- The MTTF increase obtained by the power-aware physical synthesis is cumulative and sometimes super-linear. Note that this is a different observation from the power reduction obtained by fault-tolerant physical synthesis. The combination of three phases gives the best results for simultaneous power reduction and increased MTTF.

7. CONCLUSIONS AND FUTURE WORK

We have presented an experimental study exploring the interaction between power and fault tolerance optimization in FPGA physical synthesis. We adapt an existing power-aware physical synthesis tool chain [19], with fewer than 100 lines of change, to reduce the soft error-induced fault rate. Our study shows that power and fault tolerance are two closely connected optimization objectives, and they can be concurrently optimized. Existing power-aware synthesis flows already optimize for fault tolerance to a certain extent, and minor changes in the implementation can result in much higher fault tolerance. In our experiments, our fault-tolerant physical synthesis tool chain increases the MTTF by 100% on average, while simultaneously reducing the power dissipation by 25% on average, both compared with the baseline physical synthesis (T-VPack and VPR). Without any changes, an existing power-aware physical synthesis tool simultaneously reduces the power dissipation by 35% and increases the MTTF by 46% on average.

In future, we plan a similar study for other design phases and using different optimization techniques.

8. REFERENCES

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

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

[3] V. Betz A. Marquardt and J. Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 1999.

[4] A.K. Agarwal, J. Cong, and B. Tagiku. Fault tolerant placement and defect reconfiguration for nano-FPGAs. In *Proc. Int. Conf. on Computer Aided Design*, 2008.

[5] G. Asadi and M. Tahoori. Soft error rate estimation and mitigation for SRAM-based FPGAs. In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, Feb 2005.

[6] H. Asadi, M.B. Tahoori, D. Kaeli B. Mullins, and K. Granlund. Soft error susceptibility analysis of sram-based FPGAs in high-performance information systems. *IEEE Transactions on Nuclear Science (TNS)*, 2007.

[7] V. Betz and J. Rose. Directional bias and non-uniformity in FPGA global routing architectures. In *Proc. Int. Conf. on Computer Aided Design*, 1996.

[8] V. Betz and J. Rose. Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size. *IEEE Custom Integrated Circuits Conference, Santa Clara, CA, 1997, pp. 551 - 55*, pp:551–554, 1997.

[9] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *FPL*, 1997.

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

[11] K. Chapman and L. Jones. SEU strategies for Virtex-5 devices. In *XAPP864*, 2009.

[12] Z. Feng, Y. Hu, L. He, and R. Majumdar. IPR: In-place reconfiguration for FPGA fault tolerance. In *Proc. Int. Conf. on Computer Aided Design*, 2009.

[13] S. Golshan and E. Bozorgzadeh. Single-Event-Upset (SEU) Awareness in FPGA Routing. In *Proc. Design Automation Conf*, pages 330–333, June 2007.

[14] 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.

[15] Y. Hu, Y. Lin, L. He, and T. Tuan. Physical synthesis for FPGA interconnect power reduction by dual-vdd budgeting and retiming. In *ACM Trans. on Design Automation of Electronics Systems*, 2008.

[16] Jonathan Johnson and Michael Wirthlin. Voter insertion algorithms for FPGA designs using triple modular redundancy. In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2010.

[17] M. Jose, Y. Hu, R. Majumdar, and L. He. Rewiring for robustness. In *Proc. Design Automation Conf*, 2010.

[18] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 6:212–222, 1998.

[19] J. Lamoureux and S.J.E. Wilton. On the interaction between power-aware FPGA cad algorithms. In *Proc. Int. Conf. on Computer Aided Design*, 2003.

[20] Y. Lin and L. He. Stochastic physical synthesis for FPGAs with pre-routing interconnect uncertainty and process variation. In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2007.

[21] A. Marquardt, V. Betz, and J.Rose. Timing-driven placement for FPGAs. In *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pages 203–213, 2000.

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

[23] S. Mukherjee, J. Emer, and S.K. Reinhardt. Radiation-induced soft errors: An architectural perspective. In *Intl. Symp. on High-Performance Computer Architecture (HPCA)*, 2005.

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

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