SETmap: A Soft Error Tolerant Mapping Algorithm for FPGA Designs with Low Power

Chi-Chen Peng, Chen Dong, Deming Chen

Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign

ABSTRACT: *Field programmable gate arrays (FPGAs) are widely used in VLSI applications due to their flexibility to implement logical functions, fast total turn-around time and low none-recurring engineering cost. SRAM-based FPGAs are the most popular FPGAs in the market. However, as process technologies advance to nanometer-scale regime, the issue of reliability of devices becomes critical. Soft errors are increasingly becoming a reliability concern because of the shrinking process dimensions. In this paper we study the technology mapping problem for FPGA circuits to reduce the occurrence of soft errors under the chip performance constraint and power reduction. Compared to two power-optimization mapping algorithms, SVmap [17] and Emap [15] respectively, we reduce the soft error rate by 40.6% with a 2.22% power overhead and 48.0% with a 2.18% power overhead using 6-LUTs.*

1. Introduction

Soft errors have received much attention in the research community in recent years. A soft error occurs when a cosmic particle, such as a neutron, strikes a portion of the circuit causing the state of a node to change from 1→0 or 0→1. Soft errors are becoming a serious problem in circuit design due to shrinking process dimensions. The smaller dimensions create a situation where the capacitance at each node in the circuit is lower, consequently requiring a smaller amount of charge to cause a glitch. This glitch can propagate through a logic network provided: 1) the glitch occurs on a sensitized path, i.e., there is no *logical masking*, 2) the glitch propagates un-attenuated or even amplified, i.e., there is no *electrical masking*, and 3) the glitch arrives at the data input of a storage element during the latching window, i.e., there is no *latching-window masking*.

Soft error can occur in the memory cell or logic circuit. Traditionally, soft errors in memories have a greater impact than in logic circuits because memories have smaller cell size and a bit flip resulting in SEUs (single event upsets) becomes permanent before reprogramming takes place. Now, soft errors in logic have become a major concern as well. Previous works attempting to reduce soft errors thus have focused on these two areas: enhancing memory cells and modifying logic circuits. For example, IBM and NASA [7] presented several SRAM architectures to resist SEUs. In terms of logic, one of the famous structures is Triple Modular Redundancy (TMR) [7][9], but the area penalty (200%) is large for this approach. Moreover, TMR architecture needs a majority voting circuit to output the correct data, thus the depth of a circuit will increase. Work [6] tries to minimize area and reduce error rate at the same time. The authors follow a TMR method but only replicate the most susceptible gates for soft error protection. However, its area overhead is still very high (more than 100%).

In [2], circuit re-synthesis for improving soft-error reliability is presented. It assesses the impact of individual gates on the circuit's soft-error rate based on logic masking and *don't cares*. Then, it increases reliability through addition of single gates. In [3], three different schemes for detecting and correcting soft errors in configuration bits of the LUTs (look-up table) of FPGAs were proposed. The smallest area overhead among the three is about 48 percent. Work [12] proposed an FPGA architecture to detect 100% and correct 96% of SEUs with about 40% less transistors than the TMR-based hardened memory cell architecture. Lee [8] presents a re-synthesis work to reduce soft error rate for FPGAs. Their work targets dual-output LUTs architecture, which is supported in Xilinx's Virtex-5 FPGA [10] and Altera's Stratix II FPGA [11]. Yet, the applicability of this work may be limited. None of the above works worked on power minimization.

Power minimization is an important task especially for FPGAs because FPGA chips are intrinsically power-inefficient due to the significant amount of additional logic added for providing reconfigurability. One effective way to perform power minimization is at the logic synthesis level; more specifically, with technology mapping, which is a critical synthesis step for FPGAs. There are previous works such as [18,19], PowerMap [20], PowerMinMap [21], Emap [15], and SVmap/DVmap [17] on low power FPGA technology mapping. Techniques including bin packing, dynamic programming, greedy algorithm, binate covering, network flow algorithm, and cut-enumeration algorithm have been applied to hide the nodes of high-switching activity into LUTs so the overall dynamic power can be reduced. However, none of the above works considered fault tolerant and reliability issues.

In this paper, we present a new *soft error tolerant* mapping algorithm, *SETmap*, for FPGA designs with low power. We adopt a cut-enumeration-based method that consists of cut generation and cut selection. Our essential goal is to reduce soft error rate. To achieve that, we design a novel approach to effectively masking out soft errors during the mapping process. Meanwhile, to make the mapper power aware, we consider switch activity in the cost function. Experimental results show that our algorithm produces significant error rate reduction over previous low-power mapping algorithms, SVmap [17] and Emap [15], across a series of MCNC and ICSAS'89 benchmarks. To the best of our knowledge, this is the first technology mapping algorithm that targets both soft error reduction and low power for FPGAs.

The rest of the paper is organized as follows. In Section 2, we provide some basic definitions and formulate the fault-tolerant mapping problem on SRAM-based FPGAs. Section 3 reviews the cut-enumeration procedure and power model. Section 4 presents a detailed description of our algorithm. Section 5 provides the experimental results, and Section 6 concludes this work.

2. Definition and Problem Formulation

2.1 Definition

A DAG (directed acyclic graph) can represent a Boolean network. In a DAG, each node represents a logic gate and a directed edge *(i, j)* exists if the output of gate *i* is an input of gate *j*. A PI node has no incoming edges and a PO node has no outgoing edges. We use *input(v)* to denote the set of nodes which are *fanins* of gate *v*. Given a Boolean network *N*, we use O_v to denote a *logic cone* rooted on node *v* in *N*. O_v is a sub-network of *N* consisting of *v* and some of its predecessors, such that for any node $w \in O_v$, there is a path from *w* to *v* that lies entirely in O_v. The maximum cone of *v*, consisting of all the predecessors of *v*, is called a *fanin cone* of *v*, denoted as F_v. A *cut C* is a partitioning of a cone O_v such that the logic between *v* and the cutline form a smaller cone of *v*. The *cut-set* of the cut *C* consists of the signals on the cutline, which can be represented as *input(C)*. A cut is *K-feasible* if the cardinality of the cut-set is $\leq K$. We also call the cardinality of the cut-set the *cutsize* of the cut. The *level* of a node *v* is the length of the longest path from any PI node to *v*. The level of a PI node is zero. The *depth* of a network is the largest node level in the network. A Boolean network is *l-bounded* if $|input(v)| \leq l$ for each node *v*.

2.2 Problem Formulation

The mapping problem for soft error tolerance on SRAM-based FPGAs is to cover a given *l*-bounded Boolean network with *K*-feasible cones so that soft error tolerance after mapping is maximized while the optimal mapping depth is guaranteed under the unit delay model. We also strive to minimize the area and power overhead during such a mapping process. Our initial networks are all 2-bounded and *K* is 5 and 6 in this study because most of the state-of-the-art FPGA devices use these *K* values currently. Therefore, our final mapping solution is a DAG in which each node is a 5-LUT or 6-LUT and the edge (LUT_u, LUT_v) exists if LUT_u is in $input(LUT_v)$.

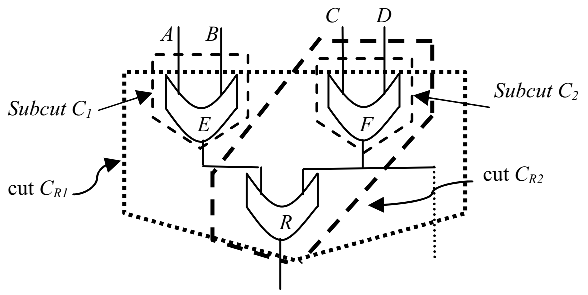

Figure 1: Example of cut generation, cost function, and global duplication adjustment

3. Cut-Enumeration and Power Model

3.1 Review of Cut-Enumeration

Cut enumeration is an effective method for finding all the possible ways of the *K*-feasible cones rooted on a node [1,2,3]. We use a simple example to illustrate the cut enumeration process. We use *{A,B,C,...}* to represent a cut with cut-set *A,B,C...*, where *{A,B,C,...}* are either internal signals or PIs. In Figure 1, all the cuts rooted on node *R* can be generated by combining the cuts rooted on its *fanin nodes E* and *F*. For this purpose, we can call the cuts on the fanin nodes *subcuts*. Combining C_1 with C_2 forms a new cut C_{R1} = *{A,B,C,D}* rooted on *R*. The cut-enumeration process combines each subcut (or the fanin node *E* or *F* itself, e.g., cut C_{R2}) on one of the fanin nodes with each counterpart from the other fanin node to form new cuts for the root node. If the input of the new cut exceeds *K*, the cut is discarded. During this enumeration process, the arrival time and truth table for each cut can be calculated. The arrival time of PI nodes is 0. The arrival time propagates through the cuts from PIs to POs, where each cut (implementable by a *K*-LUT) on the paths represents one unit delay. To get the minimum arrival time for a node *v*, we have [1,5]:

$$Arr_v = \underset{\forall C \, on \, v}{MIN}\left[\underset{i \in input(C)}{MAX}(Arr_i) + 1\right] \quad (1)$$

where *C* represents every cut generated for *v* through cut enumeration. Here, the arrival time of *C* is $MAX(Arr_i) + 1$, where Arr_i is the minimum arrival time on input signal *i* of *C*. All the cuts that can provide the minimum arrival time Arr_v form a set MA_v. Thus, the minimum arrival time for each node in the network is propagated from the PIs through cuts and iteratively calculated until all the POs are reached by a topological order. The longest minimum arrival time of the POs is the minimum arrival time of the circuit, i.e., the optimal mapping depth of the circuit.

Similarly, the mapping cost can be propagated along the process of cut enumeration. The cost for a cut *C* can be calculated as follows [1,5]:

$$A_C = \sum_{i=input(C)} [A_i / f(i)] + U_C \quad (2)$$

where U_C is the cost contributed by cut *C* itself. A_i is the estimated cost (e.g., mapping area in [5]) of a fanin cone rooted on signal *i* and *f(i)* is the fanout number of signal *i*. Therefore, the cost on *i*, i.e., the propagated cost for the fanin cone of *i*, F_i, is shared and distributed into other fanout nodes of *i*. Once the outputs reconverge, the total cost of the shared fanin cones can be summed up. This idea tries to estimate the mapping cost more accurately, considering the effects of gate fanout. Otherwise, the cost of F_i may be counted multiple times while processing the different fanouts of node *i* [1,5]. However, we show in Section 4 that this estimation model is no longer applicable to soft error cost.

3.2 Power Model

We model the dynamic power for an LUT as follows:

$$P_{LUT} = 0.5 f \cdot V_{dd}^2 \left(\sum_i^k \alpha_i \cdot C_{in} + \alpha_o \cdot C_{net}\right) \quad (3)$$

where α_i is the switching activity on input *i* of the LUT. C_{in} is the input capacitance on an LUT (a constant). α_o is the switching activity at the LUT output. C_{net} is the estimated output capacitance of wires and buffers contained in the net driven by the LUT, which can be estimated using a wire-load model with good accuracy [17]. We do not specifically model the static power but we try to reduce the total number of LUTs in our mapping solution and with a smaller area, the static power would be reduced as well. To obtain an accurate power evaluation, the gate-level FPGA power estimator *fpgaEva_LP2* [16] will be used in this study to obtain post layout power analysis. In fpgaEva_LP2, the capacitances of devices, interconnects and programmable switches are extracted after routing to calculate dynamic power during signal transition. The static power is estimated based on macro-modeling using SPICE simulation. FpgaEVA-LP2 achieved high fidelity compared to SPICE simulation, and the absolute error is merely 8% on average [16].

4. Algorithm Description

Based on the cut-enumeration framework, we first present our solutions in terms of soft error cost propagation (Section 4.1), cost function for a cut (Section 4.2), power cost and cost adjustment (Section 4.3), and cut selection (Section 4.4). Then, we present the overall algorithm in Section 4.5.

4.1 Soft Error Cost Propagation under the Timing Constraints

Cut enumeration can efficiently find all possible *K*-feasible cuts rooted on each node. While enumerating cuts, we calculate and store the truth table of each cut. Then we use the iterative procedure mentioned in Section 3 to estimate the soft error cost for each cut and each node in the network. A cut has a higher soft error cost if it has a bigger probability to propagate a soft error (bit flip) occurring at one of its inputs. The cost is smaller if a cut has a bigger chance to mask such a soft error to propagate from its inputs. Once the soft error cost for the cut itself can be estimated, the soft error for a fanin cone and the propagated cost for a cut C can be estimated using a similar idea shown in Equation (2).

However, soft error cost should not be divided by the fanout number. The error can propagate through all the fanouts of a node and it is equivalent that the cost is duplicated by the amount equal to the fanout number. Therefore, the propagated soft error cost for a cut becomes:

$$S_C = (\sum_{i=input(C)} s_i) + Cost_C \qquad (4)$$

where $Cost_C$ is the soft error cost contributed by the cut C itself. S_i is the estimated soft error cost of the fanin cone rooted on signal i. We propagate the soft error cost with the propagation process of the minimum arrival time to guarantee the optimal mapping depth. After we calculate the soft error cost for every possible cut rooted on the node v, the lowest propagated soft error cost S_v in the fanin cone F_v is below:

$$S_v = \underset{C \in MA_v}{MIN}(S_c) \qquad (5)$$

S_v is the smallest propagated soft error cost up to node v under the constraint of an optimal mapping depth. We use equations (4) and (5) to calculate the soft error costs of the cuts and nodes iteratively and go through all the nodes from PIs to POs. Note Equation (4) will be enhanced to include power cost and duplication cost in Section 4.3. Next, we present our cost estimation method of soft error for a cut itself.

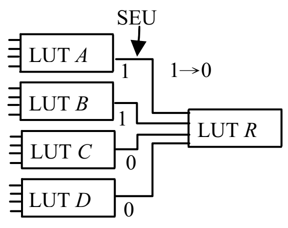

Figure 2: A mapping example

4.2 Calculation of Cost Function for a Cut Itself

The purpose of our cost function is to find a cut which has better logic masking effect for a soft error. Figure 2 and Figure 3(a) can explain the main idea. Figure 2 is a mapping solution and Figure 3(a) is the truth table of the cut that implements LUT R in the Karnaugh map format. Nodes {A, B, C, D} are the fanin LUTs of node R. Assume nodes A and B output logic "1", and C and D output logic "0"; according to Figure 3(a), the node R produces logic "1". If a soft error occurs at node A (node A outputs "0"), node R still outputs "1" according to the truth table and is not affected by the soft error. As a result, the soft error is masked. Meanwhile, a cosmic particle may not just flip one bit but also two or more neighbouring bits in the subcircuit [22]. In the example, assume there are two bit flips at neighbouring nodes A and B (i.e., 1100=>0000), then, node R would produce "0" and is indeed affected by the soft error. However, if the two-bit flip is 1100=>1001, then the soft error is still masked. Note that this masking effect is the intrinsic property of the logic itself and is directly related to how LUT R is mapped. Driven by this observation, our mapping result tries to find as many cuts as possible that have this property so we can filter out more soft errors through technology mapping.

CD / AB	**00**	**01**	**11**	**10**
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

(a)

D / EC	**0**	**1**
00	0	1
01	1	1
11	1	1
10	1	1

(b)

Figure 3: Truth tables of cuts C_{R1} (a) and C_{R2} (b)

Our cost function for a cut consists of two components. Both are related to the input combinations or input vectors in the truth table of an LUT. For example, in Figure 3, for the truth table in (a), there are 16 possible input combinations (vectors). To compute the cost for a cut, we compute two probability values first. One is the occurrence probability of the m-th input vector in the truth table of the cut, named as P_m, and the other is the probability of a bit flip for the LUT when the m-th input vector occurs, named as $P_{flip\text{-}m}$. Figure 1 illustrates how these two components work together and how our cost function can select a better cut in terms of soft error reduction. In Figure 1, two cuts C_{R1} and C_{R2} are rooted on the node R. Two truth tables in Figure 3 represent the functionalities of C_{R1} and C_{R2} respectively. Assume the probability of being logic "1" (P_{one}) for the signals {A, B, C, D} are all 0.5:

$$P_{one}(\mathrm{A}) = P_{one}(\mathrm{B}) = P_{one}(\mathrm{C}) = P_{one}(\mathrm{D}) = 0.5$$

Then, P_{one} of signals {E, F} are 0.75 since nodes E and F are OR gates. In the truth table C_{R1}, there are 16 possible input vectors. If we assume that the inputs are not correlated, the probability for each vector to occur is a simple product of the probabilities of the inputs being logic "1" (P_{one}) or "0" (P_{zero}). For example, the probability for vector (1110) to occur in Figure 3(a) would be:

$$P_{one}(\mathrm{A}) \times P_{one}(\mathrm{B}) \times P_{one}(\mathrm{C}) \times P_{zero}(\mathrm{D}) = 0.0625$$

According to [22], the probability of a double-bit soft error event in the circuit is ~5% and the sum of single-bit and double-bit soft error events covers >99% of all the soft error events. Since three or more bit flips only represent a very small percentage in a soft error event, we ignore these types of multi-bit errors in this study and focus on single-bit and double-bit soft error reduction. However, our approach can be easily extended to handle higher order multi-bit errors.

Before we explain how $P_{flip\text{-}m}$ is computed, we define *distance-1 vectors* and *distance-2 vectors*. In the truth table, if two outputs are immediate neighbors (distance-1 neighbors), then the vectors to generate these two outputs are distance-1 vectors to each other. Meanwhile, the vectors that generate outputs that are distance-2 neighbors in the truth table are called distance-2 vectors. In the truth table for cut C_{R1}, output 0 is a distance-1 neighbor of four other outputs highlighted in the dark background and is a distance-2 neighbor of six other outputs highlighted in the gray background. The physical meaning for distance-1 (distance-2) vectors is that if a single-bit (double-bit) soft error occurs, one of those distance-1 (distance-2) vectors would be the original vector that experiences a bit flip (two bit flips or a double-bit soft error) to produce the current vector. To compute $P_{flip\text{-}m}$, we need to examine the truth table to find out the neighboring outputs with different logic values. Then, we can retrieve the corresponding vectors for these outputs. Among these input vectors, each vector

```
Algorithm SETmap
input: network, $K$ (LUT input size)
output: mapping solution $S$.

Monte Carlo simulation to estimate $P_{one}$ and switching activities;
/* cut enumeration; delay and cost propagation. */
for each node u in topological order do
    if u is a primary input then
        $f(K; u) = \{u\}$;
    else
        $v1; v2 \Leftarrow$ fanins of $u$;
        $f(K; u) = \{K$-feasible combination of $C1, C2$
                where $C1 \in f(K; v1); C2 \in f(K; v2)\}$;
    end   /* $f(K; u)$ is a $K$-feasible cut rooted at $u$. */
    for each cut rooted on u do
        $Cost_C$ = Compute_Soft_Error_Cost(cut);
        $Cost_{power}$ = Compute_Power_Cost(cut);
        Power_cost_adjustment;
    end
    propagate_delay_and_cost;
end
$D$ = optimum mapping depth;
/* cut selection */
Push all PO nodes into a queue $L$;
while L is not empty do
    Pop $u$ from $L$;
        best_cost = $\infty$;
        for each timing_feasible cut $C$ on $u$ do
            $propagate\_cost_C := S_C$; /* equation (9) */
            if ($best\_cost > propagate\_cost_C$) then
                $best\_cost := propagate\_cost_C$;
                $LUT_u := C$;
            end-if
        end-for
    for each $v \in input(LUT_u)$ do
        if v has not been pushed into L then
            Push $v$ into $L$;
    end
    $S := S \cup \{LUT_u\}$;
end
output $S$;
```

Figure 4. SETmap algorithm

has a probability to generate an incorrect output logic value when one or more of the inputs in the vector had a bit flip (i.e., soft error propagated) because bit flip(s) in the input(s) would make this vector change into one of the distance-1 or distance-2 vectors which could produce a different output value. Therefore, we call these types of vectors *error-propagation vectors*. It is obvious that some vectors are not error-propagation vectors. In Figure 3(a), the vectors corresponding to the white cells with logic "1" will not propagate a soft error when single-bit or double-bit flips occur in their inputs. On the other hand, the error-propagation vectors are (0000) and the vectors for the dark and gray cells. To evaluate the probability of error propagation for a cut, we need to evaluate how many error-propagation vectors are there for this cut. An error-propagation vector has at most *cut_size* number of distance-1 vectors and at most $Max\_NF_{cut\_size}$ number of distance-2 vectors. The $Max\_NF_{cut\_size}$ for a cutsize between 2 to 6 can be easily computed, which are 1, 3, 6, 10, and 15, respectively. We compute the probability to propagate a soft error for a cut due to the *m*-th input vector as:

$$P_{flip-m} = \alpha * \frac{NC_m}{cut\_size} + (1-\alpha) * \frac{NF_m}{Max\_NF_{cut\_size}} \qquad (6)$$

where NC_m and NF_m are the number of distance-1 and distance-2 error-propagation vectors of the *m*-th input vector respectively, which generate a different output value from that of the *m*-th input vector. α is the percentage of single-bit flips in all failure events. We set α as 0.95, which counts a 5% of the chance for a double-bit soft error.

Continuing with the example, to get the soft error cost of C_{R1} in Figure 3(a), we only need to deal with the error-propagation vectors. Each vector may have a different probability to output the wrong data. In Figure 3(a), if {ABCD} is supposed to be (0000) and a soft error occurs at one or two input signals, the output of node *R* certainly produces the wrong data, which is "1". According to Equation (6), $P_{flip-(oooo)}$ = 0.95*4/4 + 0.05*6/6 = 1. But for the other ten error-propagation vectors, there is only one vector (0000) with a different output value. Then P_{flip-m} for the four distance-1 vectors are all 0.95*1/4 = 0.2375, and for the six distance-2 vectors are all 0.05*1/6 = 0.0083. The rest of the vectors in the truth table never flip outputs. Based on above ideas, our soft error cost function is computed as follows:

$$P_m = \prod_{x \in m} (P_{one}(x),\ \text{if } x{=}1 \mid P_{zero}(x),\ \text{if } x{=}0) \qquad (7)$$

$$Cost_C = \sum_{m=1}^{2^{cut\_size}} P_m * P_{flip-m} \qquad (8)$$

P_m is the probability of the occurrence of the vector *m*. $Cost_C$ is the soft error cost contributed by the cut *C* itself (refer to Equation (4)). The rationale behind Equation (8) to compute $Cost_C$ is that we enumerate every input vector in cut *C* to evaluate its probability to propagate a soft error. The total probability for soft error propagation is a summation of the probability for each input vector. This cost thus has a physical meaning. A smaller cost indicates that when one or two input signals are flipped, cut *C* has a larger chance to mask it out. Therefore, the smaller the value of $Cost_C$ is, the better the cut *C* is for soft error reduction. In Figure 3(a), given the input order of {ABCD}, the cost is:

$$\begin{aligned}
Cost_{CR1} &= P_{(0000)} \times P_{flip\text{-}(0000)} + \text{distance1\_vectors\_cost} \\
&+ \text{distance2\_vectors\_cost} \\
&= (0.5 \times 0.5 \times 0.5 \times 0.5) \times (1 + 0.2375 \times 4 + 0.0083 \times 6) \\
&= 0.1249875
\end{aligned}$$

In Figure 3(b), given the input order of {ECD}, the cost is (note $P_{one}(E)$ is 0.75):

$$\begin{aligned}
Cost_{CR2} &= P_{(000)} \times P_{flip\text{-}(000)} + P_{(001)} \times P_{flip\text{-}(001)} + \\
&P_{(010)} \times P_{flip\text{-}(010)} + P_{(100)} \times P_{flip\text{-}(100)} + P_{(011)} \times P_{flip\text{-}(011)} + \\
&P_{(110)} \times P_{flip\text{-}(110)} + P_{(101)} \times P_{flip\text{-}(101)} \\
&= (0.25 \times 0.5 \times 0.5) \times (0.95*3/3 + 0.05*3/3) + \\
&(0.25 \times 0.5 \times 0.5) \times (0.95*1/3) + (0.25 \times 0.5 \times 0.5) \times (0.95*1/3) + \\
&(0.75 \times 0.5 \times 0.5) \times (0.95*1/3) + (0.25 \times 0.5 \times 0.5) \times (0.05*1/3) + \\
&(0.75 \times 0.5 \times 0.5) \times (0.05*1/3) + (0.75 \times 0.5 \times 0.5) \times (0.05*1/3) \\
&= 0.1688
\end{aligned}$$

Therefore, Cut C_{R1} is the better choice in this example.

To compute the probability P_{one} or P_{zero}, existing analytical algorithms can be adopted [13][14]. However, we found that these heuristics are not accurate when the circuits contain reconvergent paths, especially when the path of reconvergence is long. For example, when we use the method proposed in [13], the average estimation error compared to the result of Monte Carlo simulation is 6.5%. Therefore, to accurately evaluate our mapping results, Monte Carlo simulation is used to obtain P_{one} and P_{zero} in our work. We use 10,000 random input vectors at PIs for the original 2-bounded network and count how many times each signal is evaluated as logic "1" during the simulation. Then, the probability

	SETmap		SVmap [17]		Emap [15]		Error Rate Reduction		Power Comparison	
	Err Rate	Power (w)	Err Rate	Power (w)	Err Rate	Power (w)	vs SVmap	vs Emap	vs SVmap	vs Emap
alu2	0.78%	0.0264	1.38%	0.0312	1.87%	0.0323	-43.3%	-58.2%	-15.43%	-18.29%
alu4	0.63%	0.1741	1.12%	0.1920	1.56%	0.1912	-43.8%	-59.4%	-9.35%	-8.97%
apex2	0.25%	0.2988	0.53%	0.2999	0.47%	0.2720	-53.2%	-48.2%	-0.36%	9.86%
apex4	2.23%	0.1981	5.49%	0.1940	5.29%	0.1901	-59.4%	-58.0%	2.12%	4.21%
apex6	2.26%	0.1309	3.32%	0.1307	6.87%	0.1174	-31.9%	-67.1%	0.13%	11.46%
dalu	0.67%	0.0552	0.55%	0.0578	0.55%	0.0518	23.6%	22.6%	-4.46%	6.49%
ex1010	2.52%	0.7089	4.01%	0.6260	4.03%	0.6879	-37.2%	-37.4%	13.25%	3.06%
ex5p	2.54%	0.1674	4.97%	0.1403	4.51%	0.1448	-48.9%	-43.8%	19.29%	15.62%
frg2	6.33%	0.1663	8.03%	0.1883	8.33%	0.1681	-21.2%	-24.0%	-11.67%	-1.07%
i10	3.34%	0.7183	6.47%	0.7188	8.07%	0.6379	-48.4%	-58.6%	-0.08%	12.60%
misex3	0.59%	0.1941	1.21%	0.2144	0.92%	0.2096	-50.8%	-35.8%	-9.48%	-7.39%
pdc	0.38%	0.6269	2.46%	0.6057	2.24%	0.6604	-84.7%	-83.2%	3.49%	-5.08%
rot	3.07%	0.1560	3.41%	0.1667	4.19%	0.1650	-10.0%	-26.7%	-6.44%	-5.50%
s3330_	3.41%	0.3195	6.20%	0.2843	7.94%	0.2902	-45.0%	-57.1%	12.37%	10.08%
s3384_	9.59%	0.3413	11.84%	0.3367	16.22%	0.3334	-19.0%	-40.9%	1.38%	2.38%
seq	0.66%	0.2629	1.20%	0.2632	1.42%	0.2533	-44.9%	-53.3%	-0.12%	3.78%
spla	0.83%	0.5299	2.18%	0.4771	2.00%	0.5647	-61.9%	-58.5%	11.08%	-6.17%
vda	0.88%	0.0849	2.04%	0.0725	2.87%	0.0763	-57.0%	-69.4%	17.12%	11.26%
C3540	1.18%	0.0690	1.92%	0.0709	2.23%	0.0703	-38.5%	-47.0%	-2.71%	-1.81%
C7552	3.47%	0.3045	5.47%	0.2449	7.90%	0.2843	-36.4%	-56.0%	24.36%	7.12%
AVE.	2.28%	0.2767	3.69%	0.2658	4.47%	0.2701	**-40.60%**	**-48.00%**	**+2.22%**	**+2.18%**

Table 1. Soft error reduction and power comparison results

P_{one} for a signal is this number divided by 10,000, and $P_{zero} = 1 - P_{one}$ for a signal. For the same reason, we can compute the switch activity by Monte Carlo simulation for each node in the network and obtain more accurate estimation results.

4.3 Power Cost & Cost Adjustment

We accumulate all the switching activity values on the input nodes of a cut and use this sum ($Cost_{power}$) to penalize cuts that incur larger switching power. The smaller this sum is, the bigger the chance that the cut can be picked. This naturally selects cuts that hide highly switching nodes in LUTs to reduce power. Simply adding the power cost in our cost function, however, is not accurate because of node duplication. In this paper, we carried out duplication cost adjustment, considering the specific characteristics of power cost. We use Figure 1 to illustrate our solution. When cut C_{R1} is formed by combining subcut C_1 and C_2, node F needs to be duplicated in the mapping solution due to an extra fanout going out of C_{R1}. The duplicated node F has its own cost and should be added to the cost of cut C_{R1}. The propagated soft error cost for a cut is refined from equation (4) as follows:

$$S_C = (\sum_{i=input(C)} S_i) + Cost_C + \beta * Cost_{power} + P_{f1} + P_{f2}$$

$$P_f = \begin{cases} \gamma \times Cost_{subcut} & \text{if fanout(subcut)} > 1 \\ 0 & \text{otherwise} \end{cases} \qquad (9)$$

where $f1$ and $f2$ are the two fanin nodes of the root node, and C is formed by the two subcuts rooted on $f1$ and $f2$ respectively. $Cost_{subcut}$ is the power cost of a subcut itself. β and γ are normalizing constants (we set β=0.3 and γ=0.1 through empirical study because these values balance the contributions of power cost and soft error cost in the cost function). This cost adjustment has a global impact for the cost propagation process and makes the power cost estimation more closely related to the actual implementation. We observe that such an adjustment can reduce both dynamic power and total LUT number of the mapping solution. Note that [5] used a similar cost adjustment strategy to deal with duplications but was purely for area reduction.

4.4 Cut Selection

Cut selection needs to select the best cuts to cover the entire circuits to complete the mapping. To map a critical node v, only the cut that provides MA_v (refer to Section 3.1) is picked to implement the LUT to guarantee the optimal mapping depth. For the nodes that are on the non-critical paths, we can use a cut that has smaller soft error cost S_C as long as the cut can still fulfil the timing requirement on the node to guarantee the optimal mapping depth.

4.5 Overall Algorithm

Overall our algorithm can be summarized in Figure 4. It mainly includes three stages. It first estimates P_{one} values and switching activities for each node in the original network. It then uses cut-enumeration to compute all possible cuts of every node and the functionality of every cut. During the cut-enumeration, delay and cost values are propagated. The *Compute_Soft_Error_Cost* function calculates the soft error cost for every cut following Equation (9). Meanwhile, the *Compute_Power_Cost* function and power cost adjustment (Section 4.3) can be carried out for every cut. After cut-enumeration, an optimal mapping depth is obtained for the circuit, and the cost for each cut is also available. Based on this framework, we can then pick the best cut for a node driven by the timing constraint to generate the final mapping solution.

5. Experimental Results

SETmap is implemented in C and merged with SIS [4] system. We show the detailed comparison results between SETmap, SVmap, and Emap in terms of the power consumption and the soft error rate using both MCNC and ISCAS'89 benchmarks.

Monte Carlo simulation is used to evaluate the soft error occurrence and propagation; and fpgaEVA-LP2 [16] has been applied to obtain accurate post layout power measurement. 20,000 random input vectors have been generated first. The Monte Carlo simulation randomly flips one bit or two bits in the circuit and then evaluates these 20,000 input vectors to obtain the output data. For double-bit flip simulation, we consider FPGA architecture to flip two bits simultaneously among the LUTs in the same logic cluster guided by its occurrence probability. For each benchmark, we carried out 500 separate runs with random bit flips within each run (each simulation is driven by different 20,000 input vectors to get more stable results). The output vectors at the POs are compared with correct output data (the golden model) and calculate the total number of propagated errors for the benchmark.

	Err Rate Reduction	Area Overhead	Power Overhead
vs SVmap	-36.2%	+4.8%	+7.4%
vs Emap	-47.9%	+1.5%	+7.2%

Table 2. Results of 5-LUTs in a nutshell

	K=5			*K*=6		
cut size	no. of cuts	sum of soft error cost	ave. of soft error cost	no. of cuts	sum of soft error cost	ave. of soft error cost
2	203	82.61	0.407	197	79.6	0.404
3	395	92.99	0.235	365	89.74	0.246
4	790	89.74	0.114	494	58.93	0.119
5	1344	308.27	0.229	540	55.81	0.103
6	-	-	-	1136	156.66	0.138
AVE.		143.4	**0.21**		88.148	**0.16**

Table 3. Soft error cost comparison between $K = 5$ and 6

Table 1 shows the final results. "Err Rate" columns show the error rate, which indicates the percentage where a soft error propagates all the way to a PO. Power columns show the power consumption reported by fpgaEVA-LP2. We compare to two previously published low-power technology mappers, SVmap [17] and Emap [15]. Both guarantee optimal mapping depth. Comparing to SVmap (the "vs SVmap" column) and Emap (the "vs Emap" column), SETmap shows 40.6% and 48.0% improvement respectively for soft error reduction with 2.22% and 2.18% power penalty on average using 6-LUTs. The error rate reduction is calculated as *(Err(SETmap)-Err(map2))/Err(map2)*. Power overhead is calculated as *(Power(SETmap)-Power(map2)) /Power(map2)*. We also did a comparison on area (LUT number after mapping). On average, SETmap uses 5.0% more LUTs than SVmap and 2.0% less LUTs than Emap.

In addition, we carried out experiments using 5-LUTs. The final comparison results are shown in Table 2. The comparison results of error rates and power consumption of SETmap for K = 6 are better than those for K = 5, especially comparing to SVmap. Intuitively, a larger cutsize will produce more cuts rooted on a node than a smaller cutsize, so there is a better chance to find a low-cost cut. Table 3 shows the detailed information about this point on the soft error cost estimation.

The "no. of cuts" column shows the number of cuts for cut sizes ranging from 2 to K generated during cut enumeration for the benchmark ALU4, where K is either 5 or 6. We only list the number of cuts for each cutsize when this cut represents the lowest cost on a node. The "sum of soft error cost" column shows the total soft error cost for each cutsize, and the "ave. of soft error cost" shows the average of soft error cost for each cutsize. Overall, the final average of soft error cost is smaller for the $K = 6$ case compared to the $K = 5$ case. A similar conclusion can be drawn for the power cost.

6. Conclusions and Future Work

In this work, we presented a mapping algorithm to reduce soft errors for FPGAs. Our solution offered excellent soft error reduction while guaranteeing optimal mapping depth under the unit delay model. In addition, we consider power optimization to reduce the power overhead. Experimental results showed that, compared to SVmap and Emap respectively, our algorithm SETmap produced 40.6% and 48.0% more soft error rate reduction with 2.22% and 2.18% power penalty. The future work would include studying the electrical and latching-window masking effects of the FPGA routing interconnects. Layout-driven technology mapping will also be studied to further improve circuit reliability against soft errors.

7. Acknowledgement

This work is partially supported by NSF grants CCF 07-46608 and CCF 07-02501.

8. References

[1] J. Cong, C. Wu, and E. Ding, "Cut Ranking and Pruning: Enabling A general and Efficient FPGA Mapping Solution," *Intl. Sym. on FPGA,* 1999.

[2] S. Krishnaswamy, S. Plaza, I. Markov, and J. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," *ICCAD*, 2007.

[3] H. Zarandi, S. Miremadi, C. Argyrides, and D. Pradhan, "Fast SEU Detection and Correction in LUT Configuration Bits of SRAM-based FPGAs," *Parallel and Distributed Processing Symp*., 2007.

[4] E. M. Sentovich et. al., "SIS: A System for Sequential Circuit Synthesis," *Dept. of Elec. Engineering and Computer Science, UC Berkeley*, 1992.

[5] D. Chen and J. Cong, "DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs," *ICCAD*, Nov. 2004.

[6] K. Mohanram and N. A. Touba, "Partial error masking to reduce soft error failure rate in logic circuits*," Intl. Symp. on Defect and Fault Tolerance of VLSI Systems*, 2003.

[7] F. L. Kastensmidt, L. Carro and R. REIS, *Fault-tolerance techniques for SRAM-based FPGAs*, Dordrecht, Netherland: Springer, 2006.

[8] J. Y. Lee, et. al., "Fault-tolerant resynthesis for dual-output LUTs," *ASPDAC*, 2010.

[9] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development,* 1962.

[10] A. Cosoroaba and F. Rivoallon, "Achieving higher system performance with the Virtex-5 family of FPGAs," In *http://www.xilinx.com*.

[11] "Altera stratix II device handbook," In *http://www.altera.com*, 2007.

[12] E. S. Sundar et. al., "A Novel CLB Architecture to Detect and Correct SEU in LUTs of SRAM-based FPGAs," *Intl. Conf. on Field-Programmable Technology*, 2004.

[13] B. Krishnamurthy and I. G. Tollis, "Improved Techniques for Estimating Signal Probabilities," *IEEE Trans. On Computers*, 38(7):1041–1045, 1989.

[14] M. A. Al-Kharji and S. A. Al-Arian, "A New Heuristic Algorithm for Estimating Signal and Detection Probabilities," *Great Lake Sym. for VLSI*, 1997.

[15] J. Lamoureux and S.J.E. Wilton, "On the Interaction between Power-Aware CAD Algorithms for FPGAs," *ICCAD*, 2003.

[16] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power Modeling and Characteristics of Field Programmable Gate Arrays," *TCAD*, vol. 24, Issue 11, pp. 1712-1724, Nov. 2005.

[17] D. Chen, et. al. "Low-Power Technology Mapping for FPGA Architectures with Dual Supply Voltages," *FPGA*, 2004.

[18] A. H. Farrahi and M. Sarrafzadeh. "FPGA Technology Mapping for Power Minimization," *Field-Programmable Technology*, 1994.

[19] J. Anderson and F. N. Najm, "Power-Aware Technology Mapping for LUT-Based FPGAs," *Field-Programmable Technology*, 2002.

[20] Z. H. Wang, et. al. "Power Minimization in LUT-Based FPGA Technology Mapping," *ASPDAC*, 2001.

[21] H. Li, W. Mak, and S. Katkoori, "Efficient LUT-Based FPGA Technology Mapping for Power Minimization," *ASPDAC*, 2003.

[22] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. "Characterization of Multi-bit Soft Error events in advanced SRAMs," *Intl. Electron Devices Meeting*, 2003.