SETMAP: A SOFT ERROR TOLERANT MAPPING ALGORITHM FOR FPGA DESIGNS WITH LOW POWER

BY

CHI-CHEN PENG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Assistant Professor Deming Chen

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 non-recurring engineering cost. The most popular FPGAs in the market are SRAM-based FPGAs. However, as process technologies advance to nanometer-scale regimes, the issue of reliability of devices becomes critical. Soft errors are increasingly becoming a reliability concern because of the shrinking process dimensions. In this thesis, 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 and Emap, respectively, we reduce the single event upset (SEU) rate by 30.5% with a 3.7% power overhead penalty and 50.1% with a 4.7% power overhead penalty using six-input LUTs. When multi-event upset (MEU) occurs, our work reduces the soft error rate by 33% and 31.5% for double bit flips and triple bit flips, respectively, compared to SVmap, and by 52.9% and 50.3% for double bit flips and triple bit flips, respectively, compared to Emap.

ACKNOWLEDGMENTS

I am heartily thankful to my adviser, Deming Chen, whose encouragement, guidance and support from the beginning to the end enabled me to develop an understanding of the research subject. He continually and convincingly conveyed a spirit of adventure with regard to my research. Also I would like to thank my friends in the lab for assisting me in this research.

In addition, I would like to thank my wife, Huei-Lien Hsu, who takes on more responsibility to take care of our baby. Her hard work helps me to focus on my research. This thesis could not have been completed in time without her support.

TABLE OF CONTENTS

CHAPTER 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 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 [1] presented several SRAM architectures to resist SEUs. Figure 1 is an example to show how an enhancing memory cell works [2]. When a memory cell holds a value, either P1 and N2 or P2 and N1 are in the "on" state. Therefore, there are always two SEU sensitive nodes in the cell. The decoupling resistor slows the regenerative response of the cell, so the cell can tolerate a voltage transient pulse due to particle hit.

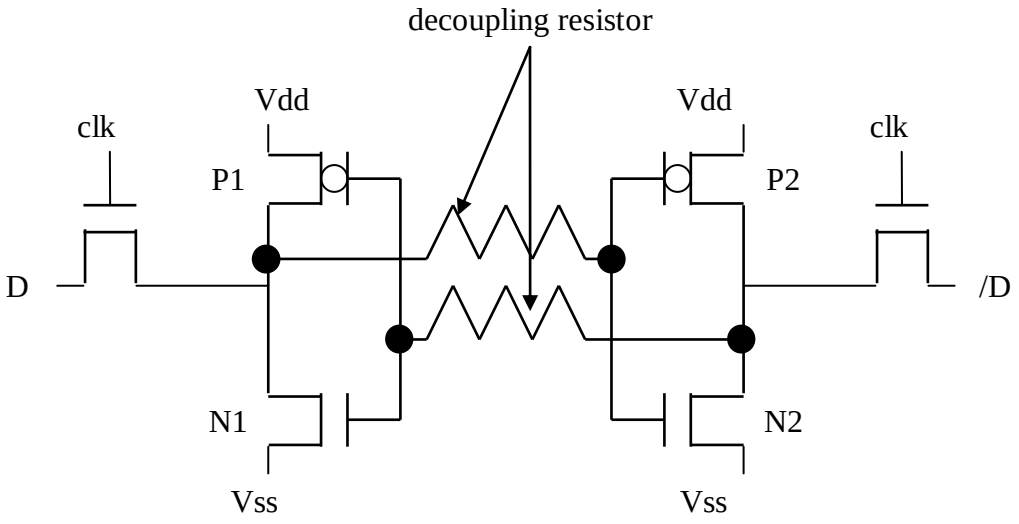


Figure 1: Example of enhancing SRAM for avoiding SEU.

In terms of logic, one of the famous structures is triple modular redundancy (TMR) [1], [3], but the area penalty (200%) is large for this approach. Moreover, TMR architecture needs a majority voting circuit (MAJ) to output the correct data; thus, the depth of a circuit will increase. Figure 2 shows a TMR example. Figure 2(a) is the original circuit and Figure 2(b) is the TMR of Figure 2(a). In Figure 2(b), combinational logic 1 is copied to combinational logic 2 and 3, and one MAJ is connected to the output of these three combinational logic components. When one SEU occurs in one of the three combinational logic components, the MAJ still can output the correct data. Mohanram et al. [4] try to minimize area and reduce error rate at the same time. They 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 [5], 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*.

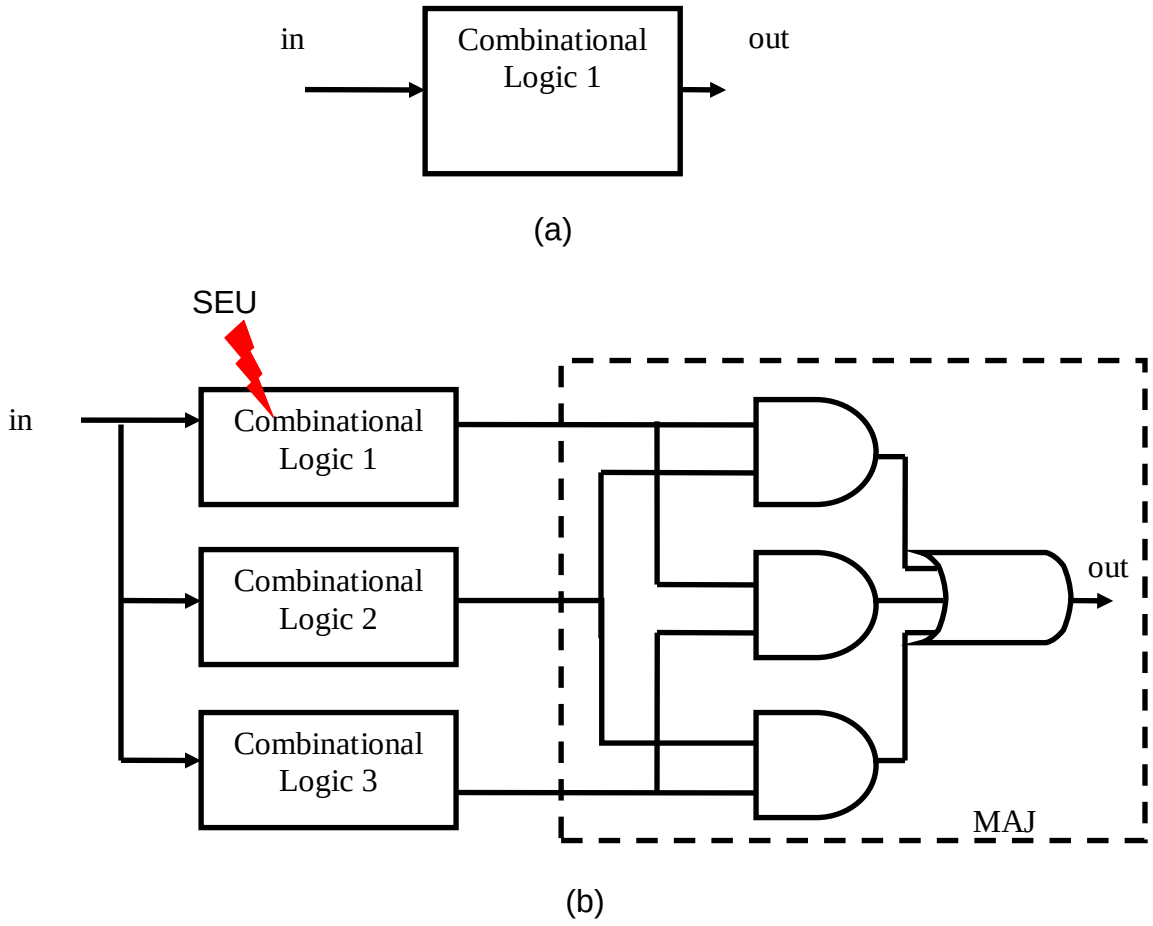


Figure 2: Triple modular redundancy including majority voter schematic.

Then, it increases reliability through addition of single gates. In [6], three different schemes for detecting and correcting soft errors in configuration bits of the LUTs (look-up tables) of FPGAs were proposed. The smallest area overhead among the three is about 48%. Reddy et al. [7] proposed a FPGA architecture to detect 100% and correct 96% of SEUs with about 40% fewer transistors than the TMR-based hardened memory cell architecture. Lee et al. [8] present a re-synthesis work to reduce the soft-error rate for FPGAs. Their work targets dual-output LUT architecture, which is supported in Xilinx's Virtex-5 FPGA [9] and Altera's Stratix II FPGA [10]. Yet, the applicability of this work may be limited. None of the above research 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 is previous work such as [11], [12], PowerMap [13], PowerMinMap [14], Emap [15], and SVmap/DVmap [16] 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 thesis, 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 the 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 [16] and Emap [15], across a series of MCNC and ICSAS'89 benchmarks with ignorable power overhead. 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 thesis is organized as follows. In Chapter 2, we provide some basic definitions and formulate the fault-tolerant mapping problem on SRAM-based FPGAs. Chapter 3 reviews the cut-enumeration procedure and power model. Chapter 4 presents a detailed

description of our algorithm. Chapter 5 provides the experimental results, and Chapter 6 concludes this thesis.

CHAPTER 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 that 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*. The logic cone 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 [17]. The maximum cone of *v*, consisting of all the predecessors of *v*, is called a *fanin cone* of *v*, denoted as F_v [17]. A *cut C* is a partitioning of a cone O_v, such that the logic between *v* and the cutline forms 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 thesis. 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 $input(LUT_v)$.

CHAPTER 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 [5], [6], [18]. 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 3, 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 *CR1 = {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 *CR2*) 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 [17], [18]:

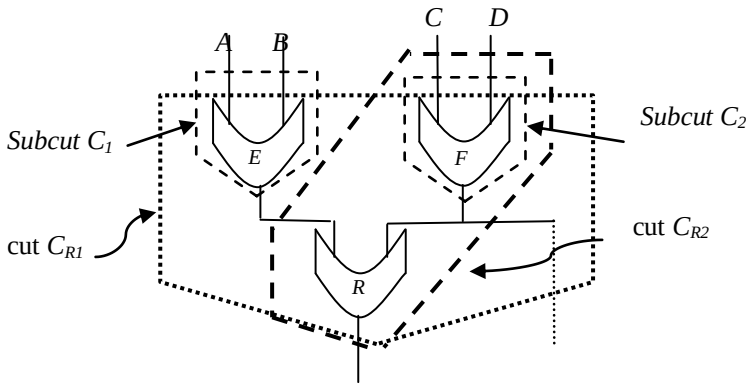


Figure 3 : Example of cut generation, cost function, and global duplication adjustment.

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

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 [17], [18]:

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

where U_C is the cost contributed by cut C itself. A_i is the estimated cost (e.g., mapping area in [17]) 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 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. 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 [17], [18]. However, we show in Chapter 4 that this estimation is no longer applicable to soft-error cost.

3.2 Power Model

We model the dynamic power for an LUT as follows:

$$P_{LUT} = 0.5 f \cdot V_{dd}^2 \cdot (\sum_{i}^{k} \alpha_i \cdot C_{in} + \alpha_0 \cdot C_{net}) \tag{3}$$

where α_i is the switching activity on input i of the LUT, C_{in} is the input capacitance on an LUT (a constant), and α_o is the switching activity at the LUT output. We define C_{net} as the estimated

output capacitance of wires and buffers contained in the net driven by the LUT; C_{net} is changeable LUT by LUT. Since we do not have wire capacitance information during mapping, we simply use $\sum_{i}^{k} \alpha_i$ as the switching cost in our cost function and try to minimize this quantity.

We do not specifically model the static power, but we try to reduce the total number of LUTs in our mapping solution. 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* [19] will be used in this thesis 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. The power estimator *fpgaEVA-LP2* achieved high fidelity compared to SPICE simulation, and the absolute error is merely 8% on average [19].

CHAPTER 4

ALGORITHM DESCRIPTION

On the basis of 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 global power 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 Chapter 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

$$S_C = \left(\sum_{i=input(C)} S_i \right) + Cost_C \tag{4}$$

where $Cost_C$ is the soft error cost contributed by the cut C itself. The estimated soft error cost of the fanin cone rooted on signal i is S_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:

$$S_v = \underset{C \in MA_v}{MIN}(S_C) \tag{5}$$

The term 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 that Equation (4) will be enhanced to include power cost and global duplication cost in Section 4.3. Next, we present our cost estimation method of soft error for a cut itself.

4.2 Calculation of Cost Function for a Cut Itself

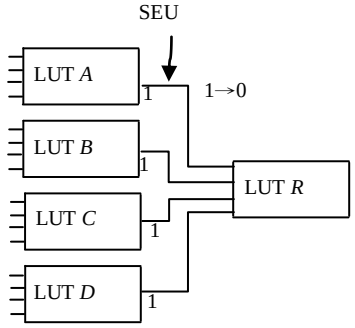


Figure 4 : Mapping result.

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 5 : Truth tables of cuts C_{R1} (a) and C_{R2} (b).

The purpose of our cost function is to find a cut which has better logic masking effect for a soft error. Figure 4 and Figure 5(a) can explain the main idea. Figure 4 is a mapping solution and Figure 5(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, B, C,* and *D* output logic "1"; according to Figure 5(a), the node *R* outputs "1." When 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. 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.

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 5, 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 3 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 3, two cuts C_{R1} and C_{R2} are rooted on the node *R*. Two truth tables in Figure 5 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:

$$P_{one}(\mathrm{A}) = P_{one}(\mathrm{B}) = P_{one}(\mathrm{C}) = P_{zero}(\mathrm{D}) = 0.5 \tag{6}$$

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 5(a) would be:

$$P_{one}(\mathrm{A}) \times P_{one}(\mathrm{B}) \times P_{one}(\mathrm{C}) \times P_{zero}(\mathrm{D}) = 0.0625 \tag{7}$$

Before we explain how $P_{flip\text{-}m}$ is computed, we define a concept called *neighboring vectors*. In the truth table, if two outputs are neighbors, then the vectors to generate these two outputs are neighboring vectors. In the truth table for cut C_{R1}, output 0 is a neighbor of four other outputs highlighted in bold color and italic font type. Then, the input vector for output 0 is a

neighboring vector for the other four vectors that generate the four outputs respectively. 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 neighboring vectors for these outputs. Among these input vectors, each vector has a probability to generate an incorrect logic value if one of the inputs in the vector had a bit flip (soft error propagated) because a bit flip in that input would make this vector change into one of the neighboring vectors, which would produce a different output value. We call these types of vectors *error-propagation vectors*. In Figure 5(a), the error-propagation vectors are (0000), (0001), (0010), (0100), and (1000). In Figure 5(b), such vectors are (000), (001), (010), and (100). An error-propagation vector has at most *cut_size* number of neighboring vectors that generate different output values. Therefore, the probability to propagate a bit flip for a cut due to the *m*th input vector is:

$$P_{flip\text{-}m} = N_m / cut\_size \tag{8}$$

where N_m is the number of neighboring vectors, which generate a different output value from the *m*th input vector.

Continuing with the example, to get the soft error cost of C_{R1} in Figure 5(a), we only need to deal with the five error-propagation vectors since only they have a chance to output wrong data. Each vector may have a different probability to output the wrong data. In Figure 5(a), if {ABCD} is supposed to be (0000) and a soft error occurs at one of these input signals, the output of node *R* certainly produces the wrong data, which is "1." As a result, the probability to propagate a bit is $P_{flip\text{-}(oooo)} = 4/4 = 1$. But for the other error-propagation vectors (0001), (0010), (0100), and (1000), there is only one neighboring vector (0000) with a different output value. Then $P_{flip\text{-}m}$ for these four vectors are all $1/4 = 0.25$. The rest of the vectors in the truth table

never flip outputs. On the basis of the above concepts, our soft error cost function is computed as follows:

$$P_m = \prod_{x \in m} (P_{one}(x),\ \text{if } x=1 \mid P_{zero}(x),\ \text{if } x=0) \tag{9}$$

$$Cost_C = \sum_{m=1}^{2^{cut\_size}} P_m * P_{flip\text{-}m} \tag{10}$$

The term P_m is the probability of vector m, and $Cost_C$ is the soft error cost contributed by the cut C (Equation (4)). The rationale behind Equation (10) 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 of the input signals is 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 5(a), given the input order of {ABCD}, the cost is

$$\begin{aligned} Cost_{CR1} &= P_{(0000)} \times P_{flip\text{-}(0000)} + P_{(0001)} \times P_{flip\text{-}(0001)} \\ &+ P_{(0010)} \times P_{flip\text{-}(0010)} + P_{(0100)} \times P_{flip\text{-}(0100)} + P_{(1000)} \times P_{flip\text{-}(1000)} \\ &= (0.5 \times 0.5 \times 0.5 \times 0.5) \times (4/4 + 1/4 + 1/4 + 1/4 + 1/4) \\ &= 0.125 \end{aligned} \tag{11}$$

And, in Figure 5(b), given the input order of {ECD}, the cost is

$$\begin{aligned} Cost_{CR2} &= P_{(000)} \times P_{flip\text{-}(000)} + P_{(001)} \times P_{flip\text{-}(001)} + P_{(010)} \times P_{flip\text{-}(010)} + P_{(100)} \times P_{flip\text{-}(100)} \\ &= (0.25 \times 0.5 \times 0.5) \times (3/3) + (0.25 \times 0.5 \times 0.5) \times (1/3) \\ &+ (0.25 \times 0.5 \times 0.5) \times (1/3) + (0.75 \times 0.5 \times 0.5) \times (1/3) \\ &= 0.167 \end{aligned} \tag{12}$$

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 [20], [21]. 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 [20], 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 and count how many times each signal is evaluated as logic "1" during the simulation. Then, the probability 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 and obtain more accurate estimation results.

4.3 Power Cost and Global Power 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 thesis, we carried out duplication cost adjustment, considering the specific characteristics of power cost. We use Figure 3 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}. However, we cannot directly sum these two costs because the total cost can be over-estimated by doing so. The propagated soft error cost for a cut is refined as follows:

$$S_C = (\sum_{i=input(C)} S_i) + Cost_C + Cost_{power} + P_{f1} + P_{f2} \tag{13}$$

$$P_f = \begin{cases} \beta \times Cost_{subcut} & \text{if fanout(subcut)} > 1 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

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. The power cost of a subcut itself is $Cost_{subcut}$, and β is a positive constant. We set β=0.01 through empirical study because this value balances the contributions of power cost and soft error cost in the cost function.) Once S_C is adjusted, it starts to influence the cost of the following fanout cones. Thus, 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. Such a duplication model would have an effect of reducing logic duplication during mapping, thus reducing the total number of LUTs.

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 6. At the beginning, we use cut-enumeration to compute all possible cuts of every node and the functionality of every cut (Section 4.1). After cut-enumeration, an optimal mapping depth is obtained. Then we estimate P_{one} for every node in the program. The *Compute_Soft_Error_Cost* function calculates the soft error cost for every cut following Equation (13) and Equation (14) (Section 4.2.) Meanwhile, the *Compute_Power_Cost* function and global power cost adjustment (Section 4.3) can be carried out for every cut. After the costs of all possible cuts are computed, we propagate the lowest

```
algorithm SETmap
input: network, K (LUT input size)
output: mapping solution S.
/* cut enumeration. f(K; u) is a K-feasible cut rooted at u. */
for each node u in topological order do
      if u is a primary input then
         f(K; u) = {u};
      else
         v1; v2 ⇐ fanins of u;
         f(K; u) = {K-feasible combination of C1, C2 where C1∈ f(K; v1); C2 ∈ f(K; v2)};
      end
end
D = optimum mapping depth;

/* cost calculation and propagation */
Read P_one for every node;
for each node u in topological order do
       for each cut rooted on u do
         Cost_C = Compute_Soft_Error_Cost(cut);
         Cost_power = Compute_Power_Cost(cut);
         Glocal_power_cost_adjustment;
       end
       propagate_cost;
end

/* cut selection */
Push all PO nodes into a queue L;
while L is not empty do
      Pop u from L;
         best_cost = ∞;
         for each timing_feasible cut C on u do
                  propagate_cost_C := S_C;
                  if (best_cost > propagate_cost_C) then
                           best_cost := propagate_cost_C;
                           LUT_u := C;
                  end-if
         end-for
       for each v ∈ input(LUT_u) do
         if v has not been pushed into L then
                  Push v into L;
       end
       S := S ∪ {LUT_u};
end
output S;
```

Figure 6: SETmap algorithm.

costs. On the base of this framework, we can pick the best cut for a node driven by the timing constraint to generate the final mapping solution (Section 4.4).

CHAPTER 5

EXPERIMENTAL RESULTS

SETmap is implemented in C and merged with the SIS [22] system. We show the detailed comparison results between SETmap, SVmap [16], and Emap [15] in terms of the power consumption and the soft error rate using both MCNC and ICSAS'89 benchmarks.

Monte Carlo simulation is used to evaluate the soft error occurrence and propagation; and *fpgaEVA-LP2* [19] has been applied to obtain accurate post-layout power measurement. Twenty thousand random input vectors have been generated first. The Monte Carlo simulation randomly flips one bit in the circuit and then evaluates these 20,000 input vectors to obtain the output data. For each benchmark, we carried out 500 separate runs with one random bit flip within each run. Each simulation is driven by 20,000 different input vectors to get more stable results. The output vectors at the POs are compared with correct output data (the golden model), and the total number of propagated errors for the benchmark is calculated.

Table 1. SEU reduction and power comparison result (k = 6)

	SETmap		SVmap [16]		Emap [15]		SEU Rate Reduction		Power Comparison	
	SEU rate	**Power (w)**	**SEU rate**	**Power (w)**	**SEU rate**	**Power (w)**	**vs SVmap**	**vs Emap**	**vs SVmap**	**vs Emap**
alu2	0.90%	0.048	1.35%	0.049	1.48%	0.050	-33.8%	-39.5%	-2.0%	-4.0%
alu4	0.60%	0.235	1.11%	0.239	1.52%	0.245	-45.7%	-60.4%	-1.7%	-4.0%
apex2	0.69%	0.346	1.17%	0.327	1.00%	0.319	-41.1%	-31.3%	5.8%	8.6%
apex4	1.32%	0.230	5.83%	0.214	7.36%	0.195	-77.3%	-82.0%	7.2%	17.7%
apex6	2.10%	0.331	3.03%	0.331	5.29%	0.319	-30.9%	-60.4%	-0.2%	3.6%
dalu	0.78%	0.079	0.60%	0.081	0.85%	0.077	29.0%	-9.0%	-2.8%	2.6%
ex1010	0.45%	0.662	3.43%	0.608	2.95%	0.605	-86.8%	-84.7%	8.8%	9.4%
ex5p	0.55%	0.194	3.46%	0.175	3.50%	0.156	-84.1%	-84.3%	11.3%	24.9%
frg2	5.49%	0.447	5.31%	0.437	7.93%	0.445	3.5%	-30.8%	2.2%	0.5%
i10	3.97%	1.289	7.19%	1.297	8.13%	1.272	-44.8%	-51.2%	-0.6%	1.3%
misex3	0.96%	0.252	1.01%	0.246	1.85%	0.238	-5.4%	-48.3%	2.4%	5.9%
pdc	0.42%	0.572	0.40%	0.500	2.01%	0.535	5.4%	-79.0%	14.4%	6.9%
rot	3.72%	0.367	3.83%	0.369	3.56%	0.365	-2.9%	4.6%	-0.6%	0.4%
s3330_	4.42%	0.772	7.24%	0.756	8.69%	0.773	-39.0%	-49.1%	2.1%	-0.1%
s3384_	10.56%	0.990	11.46%	0.995	14.34%	1.004	-7.9%	-26.3%	-0.5%	-1.4%
seq	0.72%	0.309	1.05%	0.289	1.98%	0.284	-31.6%	-63.6%	7.2%	9.0%
spla	0.63%	0.506	1.42%	0.480	2.10%	0.508	-55.1%	-69.7%	5.4%	-0.5%
vda	1.43%	0.117	2.68%	0.101	3.67%	0.102	-46.8%	-61.1%	16.6%	14.8%
C3540	1.49%	0.090	1.44%	0.093	1.94%	0.092	3.4%	-23.2%	-3.7%	-2.7%
C7552	2.79%	0.586	3.38%	0.573	5.84%	0.583	-17.6%	-52.3%	2.2%	0.5%
AVE.	2.20%	0.421	3.32%	0.408	4.30%	0.408	**-30.5%**	**-50.1%**	**3.7%**	**4.7%**

To compute the soft error rate, we divide the total number of propagated errors in the simulation by ten million (500 runs * 20,000 input vectors) for each benchmark. Table 1 shows the final results where *K* is 6. The SEU rate column shows the error rate, which indicates the percentage where a soft error propagates all the way to the PO. The power column is the power consumption reported by *fpgaEVA-LP2*. We compare to two previously published low-power technology mappers, SVmap [16] and Emap [15]. Both guarantee optimal mapping depth. Comparing to SVmap (the "vs SVmap" column) and Emap (the "vs Emap" column), SETmap shows 30.5% and 50.1% improvement, respectively, for soft-error reduction with 3.7% and 4.7% penalty on average. The error rate reduction is calculated as *(SEU(SETmap) - SEU(map2)) / SEU(map2)*. Power overhead is calculated as *(Power(SETmap) - Power(map2)) / Power(map2)*.

Table 2. Two and three bit flips comparison result (k = 6)

	Error rate for 2 flip-bits			SEU Rate Reduction		Error rate for 3 flip-bits			SEU Rate Reduction	
	setmap	**svmap**	**setmap**	**vs SVmap**	**vs Emap**	**svmap**	**emap**	**emap**	**vs SVmap**	**vs Emap**
alu2	2.11%	2.56%	3.11%	-17.5%	-32.2%	3.23%	3.57%	3.98%	-9.6%	-19.0%
alu4	1.06%	2.01%	3.12%	-47.1%	-65.9%	1.75%	2.91%	4.64%	-40.0%	-62.4%
apex2	1.19%	1.60%	3.06%	-25.7%	-61.0%	1.68%	2.30%	4.02%	-26.7%	-58.1%
apex4	2.92%	11.37%	14.13%	-74.3%	-79.3%	3.88%	16.76%	21.63%	-76.8%	-82.1%
apex6	4.30%	5.23%	11.19%	-17.9%	-61.6%	7.07%	9.14%	16.00%	-22.6%	-55.8%
dalu	1.46%	1.13%	1.56%	29.0%	-6.3%	2.19%	1.63%	2.53%	34.2%	-13.4%
ex1010	0.78%	6.33%	6.49%	-87.7%	-88.0%	1.31%	8.66%	10.92%	-84.9%	-88.0%
ex5p	1.73%	4.81%	7.57%	-64.0%	-77.1%	2.12%	9.56%	9.69%	-77.8%	-78.1%
frg2	10.67%	11.14%	14.78%	-4.2%	-27.8%	16.01%	18.81%	22.23%	-14.9%	-28.0%
i10	8.02%	11.75%	14.85%	-31.7%	-46.0%	11.52%	18.51%	20.02%	-37.8%	-42.5%
misex3	1.18%	1.81%	3.51%	-34.5%	-66.3%	2.65%	4.72%	5.77%	-43.9%	-54.1%
pdc	1.13%	1.76%	3.53%	-36.0%	-68.1%	1.93%	3.55%	5.32%	-45.8%	-63.8%
rot	7.38%	6.83%	8.36%	8.0%	-11.7%	9.83%	11.42%	13.55%	-14.0%	-27.5%
s3330_ com	8.40%	11.31%	16.12%	-25.7%	-47.8%	12.41%	17.58%	21.16%	-29.4%	-41.4%
s3384_ com	18.27%	22.64%	25.31%	-19.3%	-27.8%	26.76%	32.57%	35.86%	-17.8%	-25.4%
seq	1.30%	2.74%	4.12%	-52.4%	-68.4%	3.16%	4.96%	6.96%	-36.3%	-54.7%
spla	1.13%	3.28%	4.16%	-65.6%	-72.9%	2.48%	4.08%	5.64%	-39.1%	-56.0%
vda	3.15%	7.57%	8.99%	-58.4%	-64.9%	3.88%	10.35%	13.09%	-62.5%	-70.4%
C3540	3.56%	3.33%	4.04%	6.8%	-11.9%	5.49%	3.44%	5.85%	59.7%	-6.1%
C7552	5.91%	8.03%	12.49%	-26.5%	-52.7%	8.81%	11.31%	17.05%	-22.2%	-48.3%
AVE.	4.28%	6.36%	8.52%	**-32.2%**	**-51.9%**	6.41%	9.79%	12.30%	**-30.4%**	**-48.7%**

We also carried out experiments in Table 2 where each chip will experience two bit flips and three bit flips. Overall, for the two-bit-flip case, SETmap is 32.2% and 51.9% better for soft-error reduction compared to SVmap and Emap, respectively. For the three-bit-flip case, SETmap is 30.4% and 48.7% better for soft-error reduction compared to SVmap and Emap, respectively. More comparisons between soft error rate and the number of bit flips will be discussed in the end of this chapter.

Table 3. SEU reduction and power comparison result (k = 5)

	SETmap		SVmap [16]		Emap [15]		SEU Rate Reduction		Power Comparison	
	SEU rate	**Power (w)**	**SEU rate**	**Power (w)**	**SEU rate**	**Power (w)**	**vs SVmap**	**vs Emap**	**vs SVmap**	**vs Emap**
alu2	1.21%	0.039	1.85%	0.038	1.77%	0.038	-34.4%	-31.5%	3.2%	1.8%
alu4	1.12%	0.221	1.72%	0.220	1.77%	0.220	-35.1%	-36.9%	0.8%	0.5%
apex2	1.15%	0.301	1.71%	0.256	1.73%	0.245	-32.8%	-33.6%	17.5%	22.9%
apex4	2.68%	0.200	6.80%	0.181	7.17%	0.167	-60.6%	-62.6%	10.5%	19.6%
apex6	2.96%	0.242	3.66%	0.239	4.75%	0.242	-19.0%	-37.7%	1.2%	-0.1%
dalu	1.14%	0.079	0.90%	0.084	1.19%	0.078	27.2%	-4.1%	-6.1%	1.2%
ex1010	0.95%	0.586	2.52%	0.521	2.64%	0.553	-62.3%	-64.0%	12.5%	5.9%
ex5p	1.87%	0.153	4.46%	0.149	4.74%	0.141	-58.0%	-60.5%	2.8%	8.6%
frg2	4.40%	0.336	6.58%	0.330	8.17%	0.330	-33.2%	-46.2%	1.9%	1.8%
i10	5.16%	0.948	6.07%	0.944	8.71%	0.929	-15.1%	-40.8%	0.4%	2.0%
misex3	1.52%	0.221	3.07%	0.210	3.81%	0.206	-50.7%	-60.3%	4.9%	7.3%
pdc	1.19%	0.494	1.34%	0.482	2.04%	0.475	-11.1%	-41.8%	2.4%	3.9%
rot	4.07%	0.256	4.65%	0.272	4.62%	0.270	-12.4%	-12.0%	-6.0%	-5.1%
s3330_	4.59%	0.579	7.54%	0.575	9.37%	0.565	-39.2%	-51.0%	0.7%	2.5%
s3384_	11.63%	0.727	13.27%	0.730	13.77%	0.732	-12.3%	-15.5%	-0.4%	-0.7%
seq	1.68%	0.271	3.02%	0.245	3.49%	0.247	-44.2%	-51.8%	10.8%	9.6%
spla	2.50%	0.429	3.81%	0.397	3.91%	0.428	-34.3%	-36.1%	8.0%	0.2%
vda	4.27%	0.076	4.44%	0.073	5.26%	0.078	-3.9%	-18.9%	3.5%	-2.6%
C3540	1.95%	0.070	2.02%	0.072	2.48%	0.069	-3.6%	-21.4%	-2.1%	1.3%
C7552	3.60%	0.435	4.85%	0.428	7.43%	0.436	-25.7%	-51.5%	1.8%	-0.2%
AVE.	2.98%	0.333	4.21%	0.322	4.94%	0.323	**-28.04%**	**-38.90%**	**3.41%**	**4.02%**

In addition, we carried out experiments when K is 5. The final comparison results are shown in Table 3 and Table 4. The error rates of SETmap when K = 6 are better than those when K = 5 since a large cut size will produce more cuts rooted on a node than a smaller cut size, so there is a better chance to find a low-cost cut. Table 5 shows the detailed information about the reason why the error rate of the K = 6 case is better than the K = 5 case.

Table 4. Two and three bit flips comparison result (k = 5)

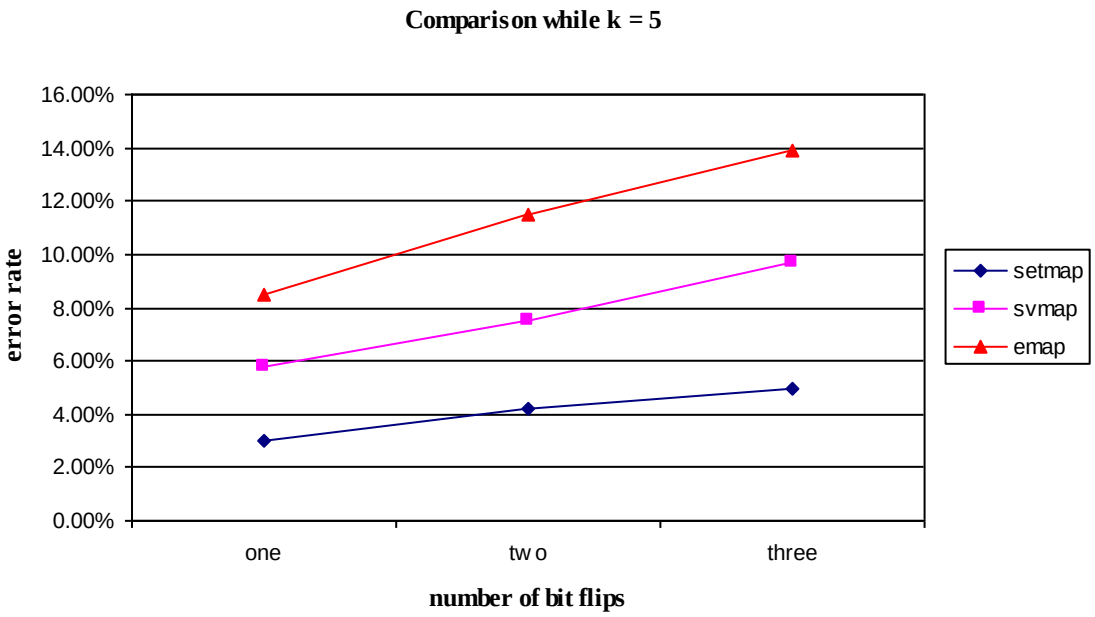
	Error rate for 2 flip-bits			SEU Rate Reduction		Error rate for 3 flip-bits			SEU Rate Reduction	
	setmap	**svmap**	**setmap**	**vs SVmap**	**vs Emap**	**svmap**	**emap**	**emap**	**vs SVmap**	**vs Emap**
alu2	2.77%	2.90%	3.58%	-4.5%	-22.8%	4.01%	4.85%	5.11%	-17.3%	-21.5%
alu4	1.91%	3.28%	3.48%	-41.9%	-45.2%	3.19%	4.21%	6.00%	-24.3%	-46.9%
apex2	2.64%	3.20%	3.45%	-17.3%	-23.4%	2.98%	4.67%	4.79%	-36.3%	-37.9%
apex4	5.81%	12.89%	14.57%	-54.9%	-60.1%	7.15%	19.19%	21.26%	-62.7%	-66.3%
apex6	5.70%	6.44%	10.63%	-11.6%	-46.4%	7.86%	9.81%	15.55%	-19.9%	-49.4%
dalu	1.91%	1.83%	2.57%	4.8%	-25.6%	3.51%	2.83%	3.39%	23.8%	3.4%
ex1010	1.83%	5.08%	6.12%	-64.0%	-70.1%	2.87%	7.33%	8.94%	-60.8%	-67.9%
ex5p	4.26%	7.24%	10.91%	-41.2%	-60.9%	5.62%	12.90%	15.62%	-56.5%	-64.1%
frg2	11.16%	12.18%	14.48%	-8.4%	-22.9%	14.60%	17.46%	19.30%	-16.4%	-24.4%
i10	10.96%	11.38%	16.89%	-3.7%	-35.1%	15.89%	17.53%	20.61%	-9.4%	-22.9%
misex3	3.40%	5.53%	6.52%	-38.6%	-47.9%	5.61%	8.26%	11.08%	-32.0%	-49.3%
pdc	2.67%	3.23%	4.08%	-17.4%	-34.5%	3.00%	5.01%	6.52%	-40.1%	-54.0%
rot	8.38%	7.54%	10.57%	11.3%	-20.7%	11.28%	13.34%	13.85%	-15.4%	-18.5%
s3330_ com	8.71%	12.39%	16.39%	-29.7%	-46.8%	14.07%	19.69%	22.67%	-28.5%	-37.9%
s3384_ com	20.32%	25.24%	26.73%	-19.5%	-24.0%	27.42%	35.71%	38.76%	-23.2%	-29.3%
seq	3.08%	4.72%	6.31%	-34.8%	-51.2%	6.24%	8.45%	9.90%	-26.1%	-36.9%
spla	3.24%	4.68%	7.35%	-30.8%	-55.9%	7.12%	9.17%	10.88%	-22.4%	-34.6%
vda	6.21%	6.95%	10.01%	-10.6%	-37.9%	10.89%	11.55%	15.74%	-5.7%	-30.8%
C3540	4.11%	3.69%	5.20%	11.5%	-20.9%	5.91%	5.95%	8.18%	-0.7%	-27.8%
C7552	6.33%	9.93%	14.41%	-36.3%	-56.1%	10.24%	12.31%	19.54%	-16.8%	-47.6%
AVE.	5.77%	7.52%	9.71%	**-21.9%**	**-40.4%**	8.47%	11.51%	13.89%	**-24.5%**	**-38.2%**

Table 5. Soft error cost of cuts rooted on each node in ALU4

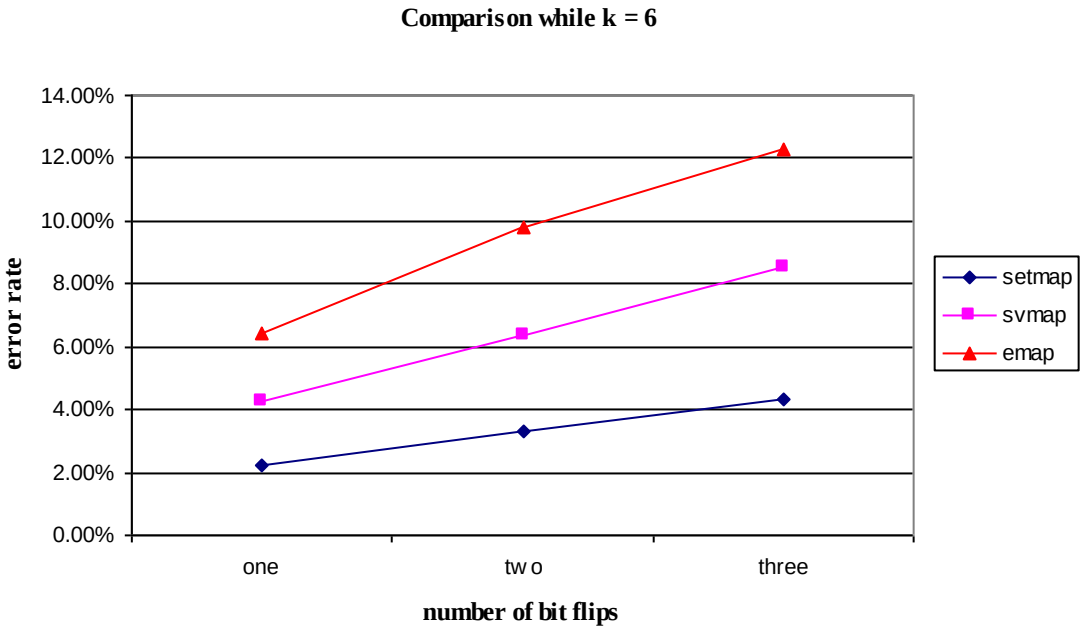
	K=5			*K*=6		
Cut size	number of cut	sum of soft error cost	average of soft error cost	number of cut	sum of soft error cost	average of soft error cost
2	205	90.9995	0.4439	187	83.0093	0.4439
3	434	109.6718	0.2527	381	98.4504	0.2584
4	850	107.185	0.1261	560	68.6	0.1225
5	1243	297.6985	0.2395	583	74.4491	0.1277
6	-	-	-	1021	194.8068	0.1908
average	683	151.3887	**0.2656**	546.4	103.8631	**0.2287**

The “number of cut” column shows the number of cuts for cut sizes ranging from 2 to *K*, where *K* is either 5 or 6. The “sum of soft error cost” column shows the total soft error cost for each cut size. To get Table 5, we first find out the cut with the lowest cost rooted on each node in the ALU4 benchmark. The cut size of the lowest cost cut rooted on each node is different from 2 to either 5 or 6. Since our cost function includes soft error cost and power cost, we extract the soft error cost from the lowest cost of the cut rooted on each node. The results of ALU4 in Table 1, Table 3, and Table 5 show that the average of soft error cost and error rate when $K = 6$ are 13.9% and 25.6% better than those when $K = 5$. This result shows that both the soft error cost and the soft error rate are related to *K*.

Figure 7 shows further error rate comparison results between SETmap, SVmap, and Emap. The slopes of increasing error rate in SETmap are smaller than those in SVmap and Emap. The error rates increase by 3.94% and 4.24% from one bit flip to three bit flips in SVmap when $K = 5$ and 6, respectively. The error rates increase by 5.42% and 5.89% from one bit flip to three bit flip in Emap when $K = 5$ and 6, respectively. However, in our algorithm, the error rates only increase by 1.96% and 2.1% from one bit flip to three bit flips for these studies.



(a)



(b)

Figure 7: Error rate comparison results between SETmap, SVmap, and Emap.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, 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 30.5% and 50.1% soft error rate reduction with 3.7% and 4.7% power penalty for the SEU case. For multiple bit upsets, SETmap is 33% and 52.9% better for the two-bit-flip case, and 31.5% and 50.3% better for the three-bit-flip case. Also, from one bit flip to three bit flips, the error rate in our algorithm just increased by 2.1%, while it was 4.24% in SVmap and 5.89% in Emap. The result shows that our algorithm is also very effective dealing with multiple bit flips. 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.

REFERENCES

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

[2] H. T. Weaver et al., "An SEU tolerant memory cell derived from fundamental studies of SEU mechanisms in SRAM," *IEEE Transactions on Nuclear Science*, vol. NS-34, no. 6, pp. 1281-1286, 1987.

[3] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200-209, 1962.

[4] K. Mohanram and N. A. Touba, "Partial error masking to reduce soft error failure rate in logic circuits," in *International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2003, pp. 433-440.

[5] S. Krishnaswamy, S. Plaza, I. Markov, and J. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *International Conference on Computer-Aided Design*, 2007, pp. 149-154.

[6] H. Zarandi, S. Miremadi, C. Argyrides, and D. Pradhan, "Fast SEU detection and correction in LUT configuration bits of SRAM-based FPGAs," in *International Parallel and Distributed Processing Symposium*, 2007, p. 6.

[7] E. S. S. Reddy, V. Chandrasekhar, M. Sashikanth, V. Kamakoti, and N. Vijaykrishnan, "A novel CLB architecture to detect and correct SEU in LUTs of SRAM-based FPGAs," in *International Conference on Field-Programmable Technology*, 2004, pp. 121-128.

[8] J.Y. Lee, Y. Hu, R. Majumdar, L. He, and M. Li, "Fault-tolerant resynthesis for dual-output LUTs," in *Asia and South Pacific Design Automation Conference*, 2010, pp. 325-330.

[9] A. Cosoroaba and F. Rivoallon, "Achieving higher system performance with the Virtex-5 family of FPGAs," Xilinx Corporation, CA. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp245.pdf

[10] *Altera Stratix II Device Handbook*, Altera Corporation, CA, 2007. [Online]. Available: http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf?GSA_pos=10&WT.oss_r=1&WT.oss=handbook stratix device

[11] A. H. Farrahi and M. Sarrafzadeh, "FPGA technology mapping for power minimization," in *International Conference on Field-Programmable Technology*, 1994, no. 849, p. 66.

[12] J. Anderson and F. N. Najm, "Power-aware technology mapping for LUT-based FPGAs," in *International Conference on Field-Programmable Technology*, 2002, pp. 211-218.

[13] Z. H. Wang, E.C. Liu, J. Lai, and T.C. Wang, "Power minimization in LUT-based FPGA technology mapping," in *Asia and South Pacific Design Automation Conference*, 2001, pp. 635-640.

[14] H. Li, W.K. Mak, and S. Katkoori, "Efficient LUT-based FPGA technology mapping for power minimization," in *Asia and South Pacific Design Automation Conference*, 2003, pp. 353-358.

[15] J. Lamoureux and S.J.E. Wilton, "On the interaction between power-aware CAD algorithms for FPGAs," in *International Conference on Computer-Aided Design*, 2003, pp. 701-708.

[16] D. Chen, J. Cong, F. Li, and L. He , "Low-power technology mapping for FPGA architectures with dual supply Voltages," in *International Symposium on FPGA*, 2004, pp. 109-117.

[17] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *International Conference on Computer-Aided Design*, 2004, pp. 752-759.

[18] J. Cong, C. Wu, and E. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *International Symposium on FPGA*, 1999, pp. 29-35.

[19] F. Li, Y. Lin, L. He, D. Chen, and J. Cong, "Power modeling and characteristics of field programmable gate arrays," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1712-1724, 2005.

[20] B. Krishnamurthy and I. G. Tollis, "Improved techniques for estimating signal probabilities," *IEEE Transactions on Computers*, vol. 38, no. 7, pp. 1041-1045, 1989.

[21] M. A. Al-Kharji and S. A. Al-Arian, "A new heuristic algorithm for estimating signal and detection probabilities," in *Great Lakes Symposium for VLSI*, 1997, pp. 26-31.

[22] E. M. Sentovich et al. (1992), SIS: A system for sequential circuit synthesis. Department of Electrical Engineering and Computer Science, University of California at Berkeley, CA. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html