Scientia Iranica D (2011) 18 (6), 1425–1434

Sharif University of Technology

Scientia Iranica

Transactions D: Computer Science & Engineering and Electrical Engineering

www.sciencedirect.com

An adaptive method to tolerate soft errors in SRAM-based FPGAs

S. Bahramnejad, H.R. Zarandi *

Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, P.O. Box 15875-4413, Iran

Received 23 May 2010; revised 23 January 2011; accepted 17 May 2011

KEYWORDS

SRAM-based FPGA;
Soft error;
Error propagation probability;
System failure rate.

Abstract In this paper, we present an adaptive method that is a combination of SEU-avoidance in CAD flow and adaptive redundancy to tolerate soft error effects in SRAM-based FPGAs. This method is based on the modification of T-VPack and VPR tools. Three different steps of these tools are modified for SEU-awareness: (1) clustering, (2) placement and (3) routing. Then we use the unused resources as redundancy. We have investigated the effect of this method on several MCNC benchmarks. This investigation has been performed using three experiments: (1) SEU-awareness in clustering with redundancy, (2) SEU-awareness in clustering and placement with redundancy and (3) SEU-awareness in clustering, placement and routing with redundancy. With a confidence level of 95%, the results show that, using each of these three experiments, the system failure rate of ten MCNC circuits has been decreased between 4.52% and 10.42%, between 10.25% and 21.63%, and between 10.48% and 24.39%, respectively.

1. Introduction

SRAM-based FPGAs are used in industry, spacecraft, networking, processing and prototyping [1,2]. The main reasons for FPGAs popularity in implementing systems are:

1. No non-recurring engineering costs.
2. Having megabits of RAM modules and many reconfigurable resources that make them suitable for large circuit implementation.
3. Fast time to market because of reduced manufacturing design time.
4. Re-configurability, remote programming and re-programmability for the sake of modifying its behavior or correcting it in case of any problems [2–5].

SRAM-based FPGAs are sensitive to transient faults, so called soft errors, which corrupt the memory content and lead to system misbehaving. These are the main disadvantages of these FPGAs. Implemented circuits in FPGAs are controlled by configuration memory, which is composed of SRAM cells [6]. So, the contents of the configuration memory are vital for the correct operation of the circuit. The primary source of transient faults is single event upsets (SEUs), which are the result of energized particles striking the circuit. These particles may deposit or remove sufficient charges and change the state of logical elements temporarily, which leads to system failure [7]. Having smaller feature size, higher integration level, smaller noise margin and lower operating voltage make the tolerance to transient faults a key challenge in SRAM-based FPGAs [8].

Several fault-tolerant methods to tolerate SEU effects have been proposed previously. In [9], two configuration-based methods, *Readback* and *Scrubbing*, are presented. Although these methods can correct any errors, they impose a time penalty and interruption on the normal execution of circuits implemented on FPGAs. Some redundancy-based methods have been presented in [2,10–12]. A TMR method with dynamic partial reconfiguration is presented in [2]. A new reliability-oriented placement and routing algorithm (RoRA), based on the TMR technique, is proposed in [11]. Moreover, an optimal design for TMR is presented in [12]. Although the TMR technique decreases the effects of SEUs significantly, this enforces a high area overhead, three times more input and output pins, and high performance penalties [13]. Hence, it is not applicable to digital circuits, which occupy a large fraction of FPGA resources by themselves. Presented in [8,13] are two SEU-avoidance methods, which are non-redundant. These methods are classified into SEU avoidance methods; they cannot correct any SEUs in circuits implemented on FPGAs.

* Corresponding author.
E-mail address: h_zarandi@aut.ac.ir (H.R. Zarandi).

1026-3098
Peer review under responsibility of Sharif University of Technology.

doi:10.1016/j.scient.2011.08.024

The main idea of this work is to use unused FPGA resources for a fault-tolerant issue. Although this idea may seem without novelty, please note that all previous methods are not like ours, as they used circuit redundancy for fault-tolerant issues. They are however limited in utilization, since their methods firstly make redundancy in the circuit and secondly they program the FPGAs. This may not always be possible because of the following problems:

1. The redundant circuit is too big to download onto the FPGA, since methods like DWC and TMR impose two and three times area, and power more than the non-redundant simple circuit, respectively.
2. When applying previous methods and downloading onto FPGAs, empirical results show that there are still some unused resources in the FPGAs, which are left without any consideration.

In such situations, we offer a method having the following advantages:

1. It improves the fault-tolerant attributes of a given circuit based on the limited available resources designers can specify.
2. It is adaptive and based on iterations to achieve the requested fault-tolerant attribute; in each iteration, we check how much we become closer to the goal.
3. Each iteration of the method uses a few remaining FPGA resources to improve fault-tolerance, so it can be continued until all resources in the FPGA are used.
4. All fault-tolerant methods impose some overheads, and they should clarify as to why using their method has a good trade-off considering tolerating faults and overheads. But in the FPGA applications, nobody used the method in reverse: First, look at how much space and how many overheads are allowed. Second, choose an effective method, based on the allowable overheads, to reach the FT.
5. This method is orthogonal to most previous methods, since our work can be applied beside them when the goal is to get more fault tolerance than before.

This paper presents a method based on SEU-avoidance, which is applied to all clustering, placement and routing steps, combined with adaptive redundancy, using unused resources of FPGA. In our method, SEU-avoidance is performed at all design steps (clustering, placement and routing), whereas in the mentioned SEU-avoidance methods, one step (routing) or two steps (placement and routing) are SEU-aware. Also, in our method, redundancy is performed based on the amount of unused resources in FPGA, whereas in previous methods that are based on full TMR, the circuit is triplicated, and the used FPGA should be three times greater than the FPGA we use. So our method can be applied to any digital circuits that occupy a large fraction of FPGA resources.

This method is applied to three different parts of two popular tools, T-VPack and VPR. These tools have been selected as typical FPGA CAD tools in this paper, because of the following reasons [13].

1. These tools are popular and well-known in FPGA CAD tools. They have been used in many FPGA research studies.
2. Without any attention to the type of FPGA, they provide a general extensible and flexible tool for describing FPGA architecture and therefore are well suited for evaluating FPGA concerns.
3. The tools can also estimate the area, delay and power consumption of the circuits implemented on a given FPGA using the given physical design characteristics of FPGA architecture.

The modification of these tools is based on a new property of circuit nets, called the Error Propagation Probability (EPP). EPP is the probability of observing failures on system outputs, provided the net is faulty. The redundancy is based on a property of circuit nets called System Failure Rate (SFR). The SFR of a given net is the probability of system failure when the net is faulty. The experiments are performed in three cases with adaptive redundancy:

1. SEU-aware clustering.
2. SEU-aware clustering and placement.
3. SEU-aware clustering, placement and routing.

In all these experiments, unused resources of FPGA are used for redundancy and the maximum tolerable overhead of redundancy has been assumed to be 20%. Results of experiments of the above mentioned three cases show system failure rate is decreased about 7.47%, 15.94% and 17.43%, and the average overhead in these experiments is about 14.24%, 12.17% and 14.17%, respectively.

The rest of this paper is organized as follows. In Section 2, previous work has been reviewed. Section 3 shows the presented SEU-tolerance method. Section 4 shows experimental results and finally Section 5 concludes this paper.

2. Related work

There have been some related works about SEU tolerance for SRAM-based FPGAs. The work related to our study can be categorized into three main topics including reconfiguration-based, redundancy-based and fault avoidance techniques.

One method based on checksum and CRC is presented in [1]. The method reloads correct configuration into the FPGA. Although the method can correct any detected single bit-flip, it requires an extra redundant FPGA for saving correct configuration frames. Another method proposes a so called *Readback* method to detect upsets that occur in the configuration memory, and then correction of the data frame that contains the affected bit. This method requires a hardware implementation of algorithms for reading and evaluating each data frame, and a memory space to store constants and variables [6,9]. Another method for SEU correction is *Scrubbing* that reloads the entire CLB frame segment at a chosen interval. This method introduces a limited overhead that essentially corresponds to the circuit needed to control the bitstream loading process and also a mechanism to determine scrub rate [6,9].

The presented method in [2] is an approach based on the TMR technique coupled with partial dynamic reconfiguration to tolerate the effects of soft errors in order to both reduce the time necessary for re-programming and to avoid the necessity of halting the entire system. In [10], the TMR technique in the presence of SEU is evaluated. The evaluation measures the reliability, area cost and speed of three different TMR styles on two different counter designs. These three TMR techniques are TMR with one voter, TMR with three voters and feedback TMR. The results of this evaluation show that in the case of feedback TMR, it is possible to have a counter design that is insensitive to any single event upset. In [11], the effects of SEUs in the configuration memory of SRAM-based FPGAs are analyzed and a reliability oriented placement and routing algorithm, coupled with TMR, is presented. As a result, fault injection experiments show that the capability of tolerating SEU effects in the FPGA configuration memory increases considerably. The represented technique in [12] investigates a design for TMR

that demonstrates the optimal number and place of voters in the TMR design, and shows that the number and placement of voters in the TMR design can directly affect the number of upsets in the routing resources. Although the TMR technique decreases the effects of SEUs significantly, this enforces a high area overhead, and three times more input and output pins [8, 13]. Hence, it is not applicable to digital circuits, which occupy a large fraction of FPGA resources.

The presented method in [8] shows that care bit reduction is an additive function, along with the routing of a net, and proves that a maze routing-based router is suitable for achieving immunity to soft errors. The presented method in [13] is an SEU-mitigative placement and routing of circuits in FPGAs, which is based on the VPR tool. In this work, the VPR tool is modified. Placement and routing decisions are made with an awareness of SEU. The advantage of these methods is that no redundancies during the placement and routing are used and the algorithms are based on SEU avoidance. But in [8], the method is applied only to the routing step of the FPGA CAD tool and so there are no fault-tolerant methods for clustering and placement steps. The method presented in [13] is only for placement and routing steps and a no fault-tolerant method is utilized for the clustering step. Moreover, their placement and routing algorithm aims at switch boxes and does not consider connection boxes, which are not negligible.

3. The proposed method

The proposed method is based on T-VPack and VPR tools and adaptive redundancy including five steps as illustrated in the gray blocks of Figure 1. These are:

1. Signal Probability (SP) and Error Propagation Probability (EPP) Computing.
2. SEU-aware clustering by modified T-VPack.
3. SEU-aware placement and routing by modified VPR.
4. System Failure Rate (SFR) Computing.
5. Redundancy Generation.

As Figure 1 shows, the gray boxes specify the modifications made to the traditional CAD flow. In this flow, the *EPP computing* process takes the *.blif* file of the circuit as an input, and then computes the signal probabilities and error propagation probabilities of all nets. The results of this step are two separated files:

1. Error Propagation Probability (EPP) file,
2. Signal Probability (SP) file.

After EPP and SP computing, the modified T-VPack tool takes the EPP file, as well as the *.blif* file. This tool packs the look-up tables and flip-flops together to form more coarse-grained logic blocks, based on the EPP property of circuit nets. A *netlist* file in *.net* extension is the output of the modified T-VPack tool. Next, a modified VPR is used to place and route the *netlist* file. This process generates placement and routing files based on FPGA architecture, the *netlist* and EPP file. After placement and routing, the *SFR computing* process computes the system failure rate based on routing information and EPP and SP files. Outputs of this process are System Failure Rate (SFR) and priority files. The SFR file is the overall system failure rate and the priority file is a descending sorted file based on the system failure rate in terms of fault occurrence in each net. The next step is *Redundancy Generation*, which takes *.blif* and priority files as inputs and triplicates the first net in the priority file. The redundancy is repeated until one of the following conditions is true:

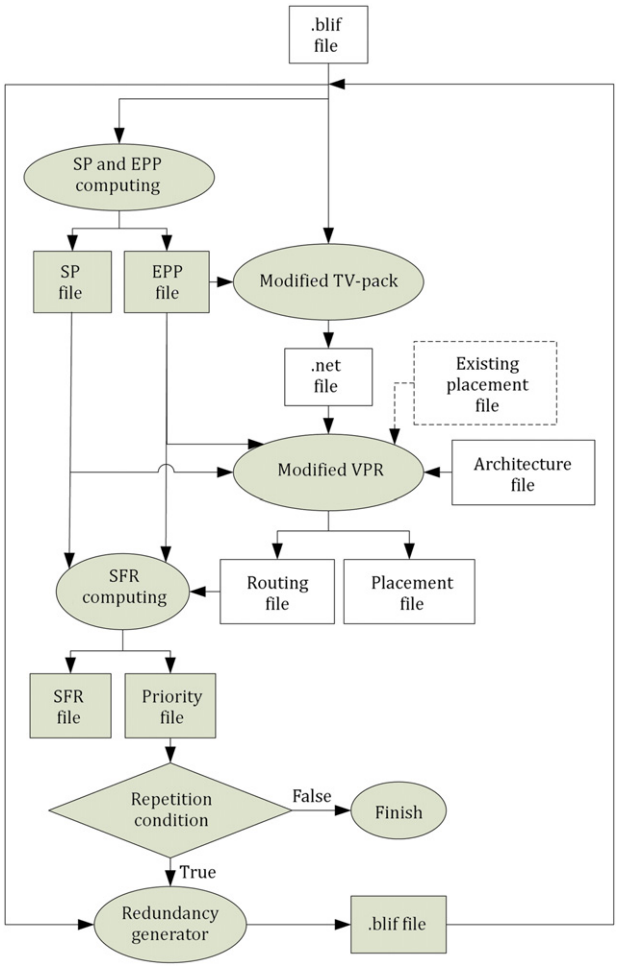

Figure 1: SEU-tolerance method in CAD flow and SFR computing and redundancy generation. The gray blocks correspond to the modification made in traditional flow.

1. FPGA has not enough unused resources for redundancy.
2. An acceptable amount of system failure rate improvement is achieved.
3. There is not a non-redundant net for redundancy.
4. The amount of overhead is not tolerable for users whose assumed tolerable overhead is 20%.

3.1. SP and EPP computing

In this step, the signal probability and error propagation probability of all circuit nets are computed according to [14, 15], which both depend only upon the gate-level *netlist* of the design. So, for extracting the gate level circuit, the *.blif* format of the circuit is used.

Signal probability is the logic "1" occurrence probability for a signal [16]. The signal probability of a net can be computed according to the gates that are between circuit inputs and the net. For example, if we have two signals A and B with $SP(A) = a$ and $SP(B) = b$ probabilities, rules for computing the signal probability of basic gate outputs, i.e. NOT, AND and OR gates which are shown in Eqs. (1)–(3) [14].

Signal probability for NOT gate:

$$SP(\bar{A}) = 1 - SP(A) = 1 - a. \quad (1)$$

```
Algorithm SP_Computing()
{
  For each input net Ni
    Ni_SP=0.5
    Ni_SPNOT=0.5
  End If
 Next
For other nets Nj
 If Gate=LATCH
   Nj_SP=Gate_Input_SP
 ElseIf Gate=NOT
   Nj_SP=Gate_Input_SPNOT
 ElseIf Gate=AND
   Nj_SP=1
    For Gate_Inputs Inputi
      Nj_SP=Nj_SP× Inputi_SP
    Next
 ElseIf Gate=OR
   Product=1
    For Gate_Inputs Inputi
      Product=Product× Inputi_SPNOT
    Next
   Nj_SP=1-Product
 End If
 Next
}
```

Figure 2: Signal probability computing algorithm.

Signal probability for AND gate:

$$SP(AB) = SP(A) \times SP(B) = a \times b. \tag{2}$$

Signal probability for OR gate:

$$\begin{aligned} SP(A+B) &= SP(\overline{\overline{A}\,\overline{B}}) = 1 - SP(\overline{A}\,\overline{B}) \\ &= 1 - (1-a) \times (1-b) = a + b - ab. \end{aligned} \tag{3}$$

The reason why Eqs. (2) and (3) are a good estimation for signal probability calculation is explained in Appendix. Figure 2 shows a signal probability computing algorithm that uses the rules in [14]. This algorithm sets the signal probability of circuit input nets to 0.5. Then, according to Eqs. (1)–(3), based on gates between circuit inputs and each net, it computes the signal probability of each net. *SP* values will be used for system failure rate computing.

Figure 3 shows the error propagation probability computing algorithm based on the gate level circuit. This algorithm first initializes the fault and signal probabilities of all nets. It sets the fault probability of the faulty net to "1" and its signal probability to "0". It sets the fault probability of other nets to "0" and extracts their signal probability from the SP file. Next, it finds all output nets where there is a path between them and the faulty net. This algorithm computes the probability of error propagation from the faulty net to each circuit output net according to the rules presented in [15]. Net error propagation probability computing is according to the gate level circuit,

```
Algorithm EPP_Computing (netlist file F)
{
   While Exists a net for EPP computing
    //Initial signal and fault probabilities
    For each net Ni
        If Ni is faulty
            Ni_FP = 1 //fault probability
            Ni_SP = 0
            Ni_SPNOT =1-(Ni_FP+Ni_SP)
        Else
            Ni_FP = 0
            Ni_SP = SP //from SP file
            Ni_SPNOT =1-Ni_SP
        End if
    Next
      Out_nets = Find_Outputs;
//find outputs that there is a path between faulty net and them
   For each net Oi in Out_nets
     //Gate is the gate that neti is its output
      While Gate_output ≠ Oi
        Gate_EPP(Gate_output)
      While End
     Gate_EPP(Oi)
    Next
    Product=1
    For each net Oi in Out_nets
            Product= Product×(1 − Oi_FP)
    Next
      Neti_EPP=1-Product
   While End
}
Gate_EPP(Gate_output)
{
  If Gate = LATCH
    Gate_output_FP = latch_Input_FP
  ElseIf Gate=NOT
   Gate_output_FP = Gate_Input_FP
  ElseIf Gate = AND
   Product=1
   For each gate input Inputi
     Product=Product×(Inputi_SP+Inputi_FP)
   Next
    Gate_Out_FP=Product-Gate_Out_SP
  ElseIf Gate = OR
   Product=1
    For each gate input Inputi
       Product= Product×(Inputi_SPNOT+Inputi_FP)
    Next
     Gate_Out_FP=Product-Gate_Out_SPNOT
  EndIf
}
```

Figure 3: Error propagation probability computing algorithm.

which depends on the gates between the error site and the output. This process is repeated, until the EPP of all nets are computed. EPP values will be used for SEU-avoidance in packing, placement and routing, and for system failure rate computing. Outputs of this step are SP and EPP files that are used as inputs of other steps.

3.2. SEU-aware clustering

The original T-VPack gets a technology mapped *netlist* of look-up tables and flip-flops in *.blif* format as an input. The first block placed into a cluster is the un-clustered block, which is driven by the most critical net in the circuit. Next, the most attractive blocks are added to the cluster based on the attraction function in Eq. (4). This operation is repeated until either no more blocks will fit into the cluster, or all cluster inputs are used. Once a cluster is full, another cluster is started with a new seed, and the process is repeated until there are no un-clustered blocks left in the circuit [17].

$$Attraction = \alpha \times Criticality + (1 - \alpha) \times Input\_Sharing, \quad (4)$$

where α is a trade-off variable, which determines how much we wish the attraction to be affected by *criticality* vs. *input_sharing*. *Criticality* is used for determining which block should be used as a cluster seed, and *input_sharing* is a normalized factor, which is the number of inputs that logic blocks in a cluster have in common normalized by the maximum number of nets to which a logic block can connect [17].

The criticality of a block is determined by three factors:

1. The most important and highly weighted factor is the slack on the nets.
2. The number of paths on the input side and output side of each block.
3. If there is more than one net with the same slack, and the same number of paths, then the third and least weighted factor is the distance from the sources [17].

We have added another factor to the criticality of a block, which is the error propagation probability of connected nets to the block. The modified T-VPack tool, in addition to the *.blif* file, takes the output of *EPP computing* (EPP file) as an input. We added a function to the T-VPack source code that extracts the EPP of each net from the EPP file and uses this value for computing the criticality of each logic block. The new value of criticality that we use instead of the *criticality* value in Eq. (4) is computed according to Eq. (5):

$$Criticality_{new} = \beta \times EPP + (1 - \beta) \times Criticality_{old}, \quad (5)$$

where β is a user-defined constant, which determines the relative importance of the previous criticality components and the EPP.

3.3. SEU-aware placement

VPR is a placement and routing tool for FPGAs that takes a *netlist* of the circuit in *.net* format and FPGA architecture file, and then places and routes the circuit.

VPR placement is based on the *Simulated Annealing* algorithm. This placer initially places logic blocks and I/Os randomly into the FPGA. Then, the placement is iteratively improved by randomly swapping blocks. Acceptance of each swap is based on a cost function [18]. This cost function is composed of two components: wiring cost and timing cost. These two components are presented in Eqs. (6) and (7), respectively:

$$Wiring\_Cost = \sum_{i=1}^{nets} q(i).\left[\frac{bb_x(i)}{C_{av,x}(i)} + \frac{bb_y(i)}{C_{av,y}(i)}\right], \quad (6)$$

where $q(i)$ is used to scale the bounding boxes in order to better estimate the wire length of nets that have more than three terminals. $C_{av,x}(i)$ and $C_{av,y}(i)$ are the average channel capacities (in track) in x and y directions, respectively, over the bounding box of net i [13].

$$Timing\_Cost = \sum_{\forall\, i,j \in Circuit} Delay(i,j) \times Criticality^{CE}, \quad (7)$$

where $Delay\ (i,j)$ is the estimated delay of the connection from source i to sink j, CE is a constant, and $Criticality\ (i,j)$ is an indication of how close to the critical path the connection is [13]. We have added an EPP factor for computing $Criticality\ (i,j)$, which is in addition to the closeness to the critical path, the probability of fault propagation from the (i,j) connection to the circuit output is important.

The modified VPR tool, in addition to *netlist* and FPGA architecture files, takes the output of *EPP Computing* (EPP file). We have added a function to the VPR source code that extracts the EPP of each net from the EPP file and uses this value for computing the criticality of nets. For SEU-awareness in placement, we have used the *EPP* of nets for criticality computing. The new criticality value computed, according to Eq. (5), will be used in the *Timing_Cost* in Eq. (7).

3.4. SEU-aware routing

VPR routing is based on the Pathfinder negotiated congestion algorithm [19]. Any iteration of the router consists of sequentially ripping-up and re-routing every net in the circuit. The cost of using a routing resource is a function of the current overuse of that resource and any overuse that occurred in the previous router iteration. During the initial iterations, an overuse of routing resources is allowed. However, in later iterations, the penalty for this overuse is increased until every net uses only one wire [13]. The VPR timing driven router uses the cost function in Eq. (8):

$$Cost = Criticality \times Delay + (1 - Criticality) \times Congestion_{cost}. \quad (8)$$

In this equation, the first expression presents the delay of nets and the second expression presents the congestion cost. In SEU-aware routing, we have used the EPP of nets (extracted from the EPP file) for criticality computing, according to Eq. (5), and we have used this new value instead of the criticality in Eq. (8). With this modification, criticality is a function of the slack and the EPP of nets. So, Eq. (8) is trying to reduce the delay of nets with high slack and high EPP, thus making these nets shorter. The shorter nets have a fewer number of SRAMs, so SEU effects on these nets are reduced. It should be noted that β in Eq. (5) may have different values to those used in Eqs. (7) and (8).

3.5. SFR computing

After the placement and routing of the circuit, the *SFR computing* process computes the system failure rate due to each net and then for the entire circuit based on the presented technique in [20]. According to this technique, SFR is computed by Eq. (9):

$$SFR_{chip} = \sum_{i=1}^{nets} SFR_i = \sum_{i=1}^{nets} NER_i \times NIP_i, \quad (9)$$

where NIP_i is the *netlist* impact probability, which is the activation probability of error site i by the inputs and its propagation to the outputs. For open and stuck-at faults, NIP_i

can be computed according to Eq. (10). The first expression of this equation accounts for the erroneous value being "0", and the second expression accounts for the erroneous value being "1". Each part expresses that the erroneous value should be first activated (signal probability, SP_i, is used for the activation probability) and then propagates it to the outputs (EPP_i). Note that $EPP_i(0) = EPP_i(1)$ [20].

$$NIP_i = SP_i \times EPP_i(0) + (1 - SP_i) \times EPP_i(1) = EPP_i. \quad (10)$$

For wired-AND bridging faults (between nets i and j), NIP_i can be computed by Eq. (11). The first term of the equation expresses the probability of node i being "1", and node j being "0". The second term calculates the probability of node i being "0" and node j being "1".

$$NIP_i = [SP_i \times (1 - SP_j) \times EPP_i(0)] + [(1 - SP_i) \times SP_j \times EPP_i(0)]_i. \quad (11)$$

For wired-OR bridging faults between nets i and j, NIP_i can be computed in the same way as wired-AND bridging faults according to Eq. (12):

$$NIP_i = [SP_i \times (1 - SP_j) \times EPP_i(1)] + [(1 - SP_i) \times SP_j \times EPP_i(1)]. \quad (12)$$

NER_i is node i error rate, which is computed by Eq. (13):

$$NER_i = fpb \times N_{SRAM}, \quad (13)$$

where N_{SRAM} is the total number of SRAM cells on each net, and fpb (FIT rate per bit) is the raw FIT rate of an SRAM cell that depends on the device characteristics and the flux encountered by the device.

We compute NER_i, NIP_i and then SFR_i for both open and bridging faults and add SFR_i(open) and SFR_i(bridge) for the computing failure rate of the system in terms of fault occurrence on net i.

3.6. Redundancy generation

In the proposed method, the most susceptible net is selected for applying redundancy. Net susceptibility is defined as the system failure rate due to any fault occurrence on each net. Priority and *.blif* files are inputs of this process. Therefore, in this step, the most susceptible net has been triplicated. For triplication of the net, we triplicate its source and sink(s) logic blocks. Then, we add a voter for sink output voting. The output of this process is a new *.blif* file. This file can be used as an input for *EPP computing* and the T-VPack for repeating the CAD process until all repetition conditions are true.

4. Experimental results

Evaluation of the proposed method has been performed on ten MCNC benchmarks, and we measured the System Failure Rate Improvement (SFRI) due to open and bridging faults and overall SFRI compared to area, timing, power and overall overheads. For System Failure Rate (SFR) computing, we use the presented method in [20] explained in Section 3.5. The overall overhead is composed of three components: routing area increment (area overhead), critical path delay increment (timing overhead) and power consumption increment (power overhead), that is computed by Eq. (14). If one of these components is negative, we consider it "0", that is this component has no effect on overall overhead.

$$Overhead = \alpha \times Area\_Overhead + \beta \times Timing\_Overhead + \gamma \times Power\_Overhead, \quad (14)$$

where α, β and γ are user-defined constants between "0" and "1", which determine the relative importance of the overhead components, such that the summation of α, β and γ is equal to "1". Without loss of generality, we have chosen the value of 1/3 for all constants in our experiments. This means that all overheads have the same importance in our experiments.

Simulations have been conducted in three cases with redundancy:

1. Only SEU-aware clustering,
2. SEU-aware clustering and placement,
3. SEU-aware clustering, placement and routing.

We have compared the system failure rate and overhead in these three cases with a base experiment. The base experiment is mapping the circuits into FPGA without any redundancy and without any SEU-awareness in clustering, placement and routing of the circuits. In these three cases, after routing a circuit, the net with the greatest error propagation probability is selected for redundancy and after performing the net redundancy, the new circuit is again mapped into the FPGA. The redundancy process is repeated until one of these conditions is not true:

1. FPGA has enough unused resources for redundancy,
2. An acceptable amount of system failure rate improvement is not achieved,
3. There is a non-redundant net for redundancy,
4. The amount of overhead is tolerable for the user.

In these experiments, we supposed that the maximum tolerable overhead for a user is 20%, and redundancy is repeated until the overall overhead is less than, or equal to 20%. In all experiments, the system failure rate has been measured due to soft errors that result in open and bridging faults on the circuit nets.

In the case of SEU-aware clustering, the new criticality value in the attraction function of the T-VPack tool has been used and the β coefficient of new criticality in placement and routing cost functions has been set to zero, that is placement and routing are not SEU-aware. Table 1 shows the results of experiments by only SEU-aware clustering. According to these results, the average of system failure rate due to open and bridging faults decreases about 5.34% and 22.74%, respectively, and the overall decrement in the system failure rate is about 7.47% on average. With a confidence level of 95%, the system failure rate is in confidence intervals of 4.52% and 10.42%. Also, the average of area, timing and power overheads are about 17.95%, 3.65% and 13.31%. So, according to Eq. (14), the overall overhead is about 14.24% on average.

In the case of SEU-aware clustering and placement, the new criticality value in the attraction function of the T-VPack tool and the cost function of VPR placement have been used. The β coefficient of new criticality in the VPR routing cost function has been set to zero, that is routing is not SEU-aware. Table 2 presents results of SEU-aware clustering and placement. According to these results, the average of the system failure rate due to open and bridging faults decreases by about 13.51% and 29.54%, respectively, and the overall decrement in the system failure rate is about 15.94% on average. With a confidence level of 95%, the system failure rate is at a confidence interval of 10.25% and 21.63%. Area, timing and power overheads are about

Table 1: System failure rate improvement compared to overhead by redundancy and SEU-aware clustering.

Circuit	Open faults imp. (%)	Bridging faults imp. (%)	SFRI (%)	Area overhead (%)	Timing overhead (%)	Power overhead (%)	Overhead (%)
alu4	8.412	18.014	11.731	33.363	1.015	26.835	20.000
apex2	6.203	29.742	9.485	12.203	3.486	9.134	8.274
apex4	0.058	18.087	1.570	13.777	−6.994	4.137	5.971
des	4.498	−0.428	3.210	28.889	5.056	21.305	18.416
ex5p	5.367	50.104	9.247	11.436	32.674	−31.682	14.703
misex3	6.773	32.791	12.433	23.370	−12.478	33.887	19.085
pdc	12.246	4.549	12.063	18.168	−13.111	34.806	17.658
seq	2.782	31.853	6.749	19.446	9.489	20.509	16.481
ex1010	2.192	7.039	2.452	4.031	23.791	−8.604	9.274
spla	4.847	35.637	5.764	14.860	−6.423	22.818	12.559
Avg.	5.338	22.739	7.470	17.954	3.650	13.314	14.242

Table 2: System failure rate improvement compared to overhead by redundancy and SEU-aware clustering and placement.

Circuit	Open faults imp. (%)	Bridging faults imp. (%)	SFRI (%)	Area overhead (%)	Timing overhead (%)	Power overhead (%)	Overhead (%)
alu4	15.356	26.284	19.133	33.561	5.595	19.707	19.621
apex2	18.882	36.299	21.311	16.298	10.130	4.262	10.229
apex4	4.421	19.129	5.655	12.654	8.202	0.809	7.222
des	5.263	17.177	8.378	10.252	21.902	−9.175	10.718
ex5p	6.284	45.581	9.692	6.356	4.959	−32.387	3.772
misex3	15.155	42.505	21.105	28.728	9.291	12.819	16.945
pdc	23.127	12.211	22.867	26.964	1.292	26.747	18.334
seq	15.940	43.001	19.633	22.359	5.199	11.757	13.105
ex1010	4.360	13.273	4.837	8.154	6.696	−2.910	4.950
spla	26.358	39.968	26.764	26.611	6.362	17.592	16.855
Avg.	13.515	29.543	15.937	19.194	7.963	4.922	12.175

Table 3: System failure rate improvement compared to overhead by redundancy and SEU-aware clustering, placement and routing.

Circuit	Open faults imp. (%)	Bridging faults imp. (%)	SFRI (%)	Area overhead (%)	Timing overhead (%)	Power overhead (%)	Overhead (%)
alu4	22.556	39.242	28.324	36.926	22.226	6.101	20.000
apex2	19.491	40.565	22.430	15.518	15.192	4.262	11.657
apex4	6.535	21.950	7.828	21.542	8.717	4.336	11.531
des	4.684	0.520	3.595	28.516	5.616	20.662	18.264
ex5p	6.249	49.865	10.031	11.179	12.933	−39.178	8.037
misex3	18.098	46.207	24.213	20.277	−10.885	35.098	18.458
pdc	23.573	15.772	23.387	25.908	24.197	3.375	17.826
seq	16.922	45.681	20.847	21.527	20.229	0.398	14.051
ex1010	4.546	15.537	5.135	7.949	8.292	−4.345	5.414
spla	28.114	42.852	28.552	24.124	5.164	18.933	16.073
Avg.	15.077	31.819	17.434	21.346	11.168	4.964	14.131

19.19%, 7.96% and 4.92% on average. So, according to Eq. (14), the overall overhead is about 12.17% on average.

In the case of SEU-aware clustering, placement and routing, the new criticality value has been used in the attraction function of the T-VPack tool and in cost functions of VPR placement and routing. Table 3 presents results of SEU-aware clustering, placement and routing. According to these results, the average of the system failure rate due to open and bridging faults decreases by about 15.08% and 31.82%, respectively, and the overall decrement in the system failure rate is about 17.43% on average. With a confidence level of 95%, the system failure rate is at a confidence interval of 10.48% and 24.39%. Area, timing and power overheads are about 21.35%, 11.17% and 4.96% on average. So according to Eq. (14), the overall overhead is about 14.13% on average.

Tables 1–3 show that in these three experiments, the average of SFR reduction due to bridging faults is more than for open faults. Also, these tables show that an increment in the number of SEU-aware design steps leads to area and timing overhead increments and power overhead decrements.

Figure 4 tells us that in most experiments, the system failure rate decrement by SEU-awareness in three steps of CAD flow (clustering, placement and routing) is more than SEU-awareness in two steps (clustering and placement), and the system failure rate decrement by SEU-awareness in two steps (clustering and placement) of CAD flow is more than SEU-awareness in one step (clustering).

We could not compare our work with some previous work, such as [6,9], which are cost-effective and impose very low overheads. The reasons are:

1. The work presented in [9] uses partial reconfigurations in the case of any fault detection. Their methods do not use any available resources and just focus on a feature of the FPGA for FPGA configuration memories. The work presented here, however, is based on the remaining resources from the synthesis of a given design on FPGA, and can tolerate errors in both FPGA configuration memory and user designed memory. Moreover, our work is orthogonal to that work, since their method can be applied beside ours.
2. The work presented in [6] uses TMR in Place and Route stages of CAD flow. However, our work can be applied to the Clustering stage (a step behind), as well as Place and Route stages. Moreover, their method is based on

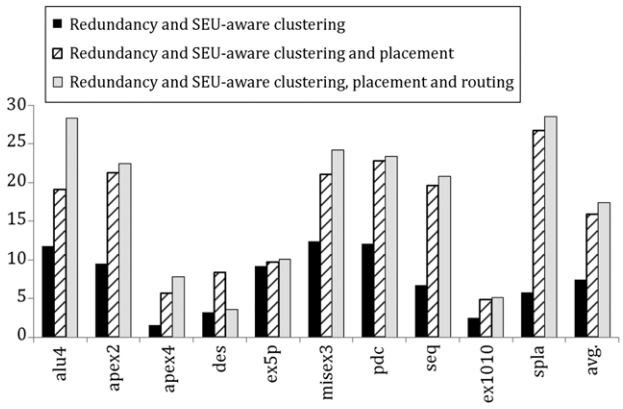

Figure 4: System failure rate improvement by SEU-avoidance in different design steps with adaptive redundancy.

triplicating modules implemented on FPGA, which may not be possible and applicable because of high area and power overheads that prevent fitting circuits into FPGAs. However, our work can increase the fault-tolerance of a design with low granularity, which makes it possible to perform the steps in many iterations.

3. Concerning quantitative comparisons, the work presented in [9] is based on a feature of FPGA and can be implemented by FPGA vendors only. Therefore, we are unable to implement and obtain results, as the work presented in [6] is based on given sources from Xilinx, which are not applicable for us. Furthermore, implementing their ideas for available CAD tools like VPR and T-VPack takes a huge effort to implement and is much more time-consuming than ours. So, at present, we are unable to implement them.

5. Conclusions

This paper presented a fault-tolerant technique based on T-VPack and VPR modification, with adaptive redundancy. The method performed the clustering, placement and routing of circuits inside FPGAs in three cases: (1) only SEU-aware clustering, (2) SEU-aware clustering and placement, and (3) SEU-aware clustering, placement and routing. In all cases, redundancy was continued until the overhead was equal to, or less than 20%. We evaluated the proposed method on ten MCNC benchmarks. Results of experiments show that the system failure rate improvement, using SEU-aware clustering, placement and routing is about 1.50% more than SEU-aware clustering and placement; moreover, it is about 9.96% more than SEU-aware clustering. Also, results showed that the overall overhead is about 14.24%, 12.17% and 14.13% on average for each of these experiments.

According to the results, in spite of the overhead of redundancy, the amount of system failure rate reduction is desirable. Moreover, the method is adaptive and can be used for any circuits based on its free remaining resources inside FPGAs. So, this is a cost effective and flexible method that uses unused resources of FPGA for redundancy.

Moreover, this work made possible some realizations of theoretical issues in given applications that have valuable outcomes for designers and engineers of the applications. This work showed:

1. How much the methods presented in this paper are applicable to given large scale applications like FPGAs,
2. How much extra fault tolerance someone would earn when applying the methods to a specific design.

This work showed that the proposed method can improve the fault-tolerance of circuits downloaded into FPGA without any concerns regarding sufficient hardware boards, and any concerns of fitting FT-circuits into FPGA.

Appendix

Following two well-known equations from elementary probability texts shows how the probabilities of the union and intersection of two sets, A and B, should be calculated:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B), \tag{A.1}$$

$$P(A \cap B) = P(A) * P(B|A). \tag{A.2}$$

However, we have used Eqs. (2) and (3) for each of the above equations. These are some reasons why Eqs. (2) and (3) are used in this paper:

1. We know the fact that Eqs. (2) and (3) may not be exact values of probability, and we would say that we do not consider "Signal Correlations", i.e. signal dependencies. Considering signal correlations (or signal dependencies) is exactly the same as much work done for observability calculations in the testability research fields. All these issues lead to the Boolean Satisfactory Problem (known as the SAT problem) [21–23], which was proved by Cook [21] in 1971 this problem is NP-complete. This means that finding out how much portion of time is a primary output "1" (finding the SP of primary outputs), and is an NP-complete problem, which should definitely be solved in polynomial time.

   Therefore, researchers have paid attention to solving the problem by partial computer programs or estimations in linear time until now [24,25]. This is the way in which we tried to find the problem by estimation in our equations.
2. Based on Lemmas 1 and 2, our probability estimation equations present a linear estimation of actual values with upper and lower boundaries. This shows that our estimations are logical and reasonable.

Definition 1. Let ϕ be an empty set and U a universal set, i.e. $P(\phi) = 1 - P(U)$.

Definition 2. Let $\hat{P}(X)$ be our estimation for the actual value of $P(X)$.

Lemma 1. *$\hat{P}(A \cap B) = P(A) \cdot P(B)$ is a linear estimation of the actual value of $P(A \cap B)$.*

Proof. $P(A \cap B)$ has minimum value when two sets, A and B, do not have intersections, so:

$$P_{\min}(A \cap B) = 0.$$

Moreover, $P(A \cap B)$ has maximum value when two sets, A and B, are identical, so:

$$P_{\max}(A \cap B) = P(A).$$

It can easily be proved that our estimation for $P(A \cap B)$ is between $P_{\min}(A \cap B)$ and $P_{\max}(A \cap B)$:

$$\begin{aligned} P_{\min}(A \cap B) &\leq \hat{P}(A \cap B) = P(A) \cdot P(B) \\ &\leq P_{\max}(A \cap B). \end{aligned} \tag{A.3}$$

This equation shows that our estimation has upper and lower bounds. This is a key point that should be noted, which says that our estimation error has some boundaries.

Now, in order to estimate linearly the actual value, suppose set A is fixed and set B is variable in space. Therefore, we have the following facts:

Point	Set space	X	$Y = P(A \cap B)$
1	$B = \phi$	$P(\phi) = 0$	0
2	$B = U$	$P(U) = 1$	$P(A)$

Based on linear estimation, we can find a line to meet the above equations, i.e. a line to pass through two points:

$$(X1, Y1), \qquad (X2, Y2), \tag{A.4}$$

where $X1$ is $P\{\phi\} = 0, Y1 = 0, X2 = P(U) = 1$ and $Y2 = P(A)$. The line passing through these two points can be expressed as:

$$Y = \frac{Y2 - Y1}{X2 - X1}(X - X1) + Y1, \tag{A.5}$$

by putting values in the above equation, we can reach the following expression:

$$\hat{P}(A \cap B) = \frac{P(A) - 0}{1 - 0}(P(B) - 0) + 0 = P(A) \cdot P(B). \tag{A.6}$$

This linear estimation shows that our estimation not only has upper and lower bounds, but also has a reasonable estimation that meets two points. □

Lemma 2. *$\hat{P}(A \cup B) = P(A) + P(B) - P(A).P(B)$ is a linear estimation of the actual value of $P(A \cup B)$.*

Proof. $P(A \cup B)$ has minimum value when one set is the subset of another e.g., $B \subseteq A$, so:

$$P_{\min}(A \cup B) = P(A).$$

Moreover, $P(A \cup B)$ has maximum value when two sets, A and B, are disjoint, so:

$$P_{\max}(A \cup B) = P(A) + P(B).$$

It can easily be proved that our estimation for $P(A \cup B)$ is between $P_{\min}(A \cup B)$ and $P_{\max}(A \cup B)$;

$$\begin{aligned} P(A) &= P_{\min}(A \cup B) \leq \hat{P}(A \cup B) \\ &= P(A) + P(B) - P(A).P(B) \leq P_{\max}(A \cup B) \\ &= P(A) + P(B). \end{aligned} \tag{A.7}$$

This equation shows that our estimation has upper and lower bounds. This is a key point that should be noted, which says our estimation error has some boundary.

Now, in order to estimate linearly the actual value, suppose set A is fixed and set B is variable in space.

Therefore, we have the following facts.

Point	Set space	X	$Y = P(A \cup B)$
1	$B = \phi$	$P(\phi) = 0$	$P(A)$
2	$B = U$	$P(U) = 1$	1

Based on linear estimation, we can find a line to meet the above equations, i.e. a line to pass through the following points:

$$(X1, Y1), \qquad (X2, Y2), \tag{A.8}$$

where $X1$ is $P\{\phi\} = 0, Y1 = P(A), X2 = P(U) = 1$ and $Y2 = 1$. The line passing through these two points can be expressed as Eq. (A.5). By putting values in the above equation, we can reach the following expression:

$$\begin{aligned} \hat{P}(A \cup B) &= \frac{P(A) - 1}{0 - 1}(P(B) - 0) + P(A) \\ &= P(A) + P(B) - P(A) \cdot P(B). \end{aligned} \tag{A.9}$$

This linear estimation shows that our estimation not only has upper and lower bounds, but also has a reasonable estimation, which meets two points. □

References

[1] Asadi, H. and Tahoori, M.B. "Soft error rate estimation and mitigation for SRAM-based FPGAs", *Proceedings of the 2005 ACM/SIGDA 13th international Symposium on Field-Programmable Gate Arrays*, Monterey, California, USA, pp. 149–160 (2005).

[2] Bolchini, C., et al. "TMR and partial dynamic reconfiguration to mitigate SEU faults in FPGAs", *Proceedings of the 22nd IEEE international Symposium on Defect and Fault-Tolerance in VLSI Systems*, Washington, DC, pp. 87–95 (2007).

[3] Zarandi, H.R., et al. "Soft error mitigation in switch modules of SRAM-Based FPGAs", *The Proceedings of IEEE International Symposium on Circuits and Systems*, New Orleans, Louisiana, USA, pp. 141–144 (2007).

[4] Srinivasana, S., et al. "Improving soft-error tolerance of FPGA configuration bits", *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Washington, DC, pp. 107–110 (2004).

[5] Huang, J., et al. "Routability and fault tolerance of FPGA interconnect architectures", *Proceedings of IEEE International Test Conference*, pp. 479–488 (2004).

[6] Sterpone, L. and Violante, M. "Hardening FPGA-based systems against SEUs: a new design methodology", *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, Washington, DC, pp. 436–444 (2005).

[7] Weaver, C. and Austin, T.M. "A fault-tolerant approach to microprocessor design", *Proceedings of the 2001 International Conference on Dependable Systems and Networks*, Washington, DC, pp. 411–420 (2001).

[8] Golshan, S. and Bozorgzadeh, E. "Single-event-upset (SEU) awareness in FPGA routing", *Proceedings of the 44th Annual Design Automation Conference*, San Diego, California, pp. 330–333 (2007).

[9] Xilinx Application Notes XAPP216. "Correcting single-event upset through virtex partial reconfiguration", Xilinx Corporation Publisher (2000).

[10] Rollins, N., Wirthlin, M., Caffrey, M. and Graham, P. "Evaluating TMR techniques in the presence of single event upsets", *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, D,C., Richard Katz, Ed., NASA Office of Logic Design and AIAA, pp. P63.1–6 (September 2003).

[11] Sterpone, L. and Violante, M. "A new reliability-oriented place and route algorithm for SRAM-based FPGAs", *IEEE Transactions on Computers*, 55(6), pp. 732–744 (2006).

[12] Lima Kastensmidt, F., et al. "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs", *Proceedings of the Conference on Design, Automation and Test in Europe*, Washington, DC, pp. 1290–1295 (2005).

[13] Zarandi, H.R., et al. "SEU-mitigation placement and routing algorithms and their impact in SRAM-based FPGAs", *Proceedings of the 8th International Symposium on Quality Electronic Design*, Washington, DC, pp. 380–385 (2007).

[14] Parker, K.P. and McCluskey, E.J. "Probabilistic treatment of general combinational networks", *IEEE Transactions on Computers*, 4(6), pp. 668–670 (1976).

[15] Asadi, H. and Tahoori, M.B. "An accurate SER estimation method based on propagation probability", *Proceedings of the Conference on Design, Automation and Test in Europe*, Washington, DC, pp. 306–307 (2005).

[16] Liu, B. "Signal probability based statistical timing analysis", *The Proceedings of IEEE Design, Automation and Test in Europe*, Munich, Germany, pp. 562–567 (2008).

[17] Marquardt, A., et al. "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density", *ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, Monterey, California, United States, pp. 37–46 (1999).

[18] Marquardt, A., et al. "Timing-driven placement for FPGAs", *ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, Monterey, California, United States, pp. 203–213 (2000).

[19] Betz, V. and Rose, J. "VPR: A new packing, placement and routing tool for FPGA research", *International Workshop on FPL*, pp. 213–222 (1997).

[20] Asadi, H. and Tahoori, M.B. "Soft error susceptibility analysis of SRAM-based FPGAs in high-performance information systems", *IEEE Transaction on Nuclear Science*, 54(6), pp. 2714–2726 (2007).

[21] Cook, S. "The complexity of theorem proving procedures", *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158 (1971).

[22] Rodriguez, C., et al. "Asynchronous team algorithms for Boolean satisfiability", *BioInspired Models of Network Information and Computing Systems*, Hungary, pp. 66–69 (2007).
[23] Davis, M. and Putnam, H. "A computing procedure for quantification theory", *Journal of the ACM*, 7(3), pp. 201–205 (1960).
[24] Babic, D., et al. "B-cubing: new possibilities for efficient SAT-solving", *IEEE Transactions on Computers*, 55(11), pp. 1315–1324 (2006).
[25] Nam Gi-Joon, K.A., et al. "A new FPGA detailed routing approach via search-based Boolean satisfiability", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(6), pp. 674–684 (2002).

Somayeh Bahramnejad received her B.S. degree from the Department of Computer Engineering, Ferdowsi University of Mashhad, in 2004, and her M.S. degree from the Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran, in 2009. Her research interests are fault-tolerant computing, computer architecture, test and testability of digital circuits, VLSI design and quantum computing.

Hamid-Reza Zarandi received his Ph.D., M.S. and B.S. degrees from Sharif University of Technology, Iran, in 2006, 2002 and 2000, respectively. He has been Assistant Professor in the Department of Computer Engineering and Information Technology at Amirkabir University of Technology since 2006. Dr. Zarandi has done research into high performance computing, dependable computer architecture, fault-tolerant computing, and embedded systems. He is a member of the IEEE Computer Society.