New Switch Box Architecture for SEU Detection in SRAM-based FPGAs

Alireza Rohani
Dept. of Computer Engineering and IT
Amirkabir University of Technology
Tehran, Iran
alirezarohani@aut.ac.ir

Hamid R. Zarandi
Dept. of Computer Engineering and IT
Amirkabir University of Technology
Tehran, Iran
h_zarandi@aut.ac.ir

Mahroo Zandrahimi
Dept. of Computer Engineering and IT
Amirkabir University of Technology
Tehran, Iran
Zandrahimi.m@gmail.com

***Abstract*—this paper proposes a method to detect Single Event Upset (SEU) faults in configuration memory of a Switch Box in SRAM-based FPGAs. A technique for mapping different patterns of a Switch Box into Boolean functions that have an interesting property, identified as self-remap, is presented, and a new architecture that exploits such property for fault detection is introduced. The architecture is validated by simulation-based fault injection tools; moreover, required area, delay and power consumption of the proposed architecture are achieved using synopsis® synthesis tool. Experimental results show that proposed architecture can detect 100% SEU faults while it imposes 150% overhead on area, 55% overhead on delay and 90% overhead on power consumption. The proposed architecture is compared with conventional fault-tolerant architectures to demonstrate its efficiency.**

I. INTRODUCTION

SRAM-based Field Programmable Gate Arrays (SFPGAs) have been increasingly used in digital industry in recent years [1, 2]. They have great advantages like high gate density, fast time to market, no Non-Recurrent Engineering (NRE) cost, and much cheaper cost than ASICs in low volume productions [3, 4]. Moreover, new ones can be reconfigured for several times and can be programmed remotely; these capabilities make them very suitable for applications such as adaptive systems and fault-tolerant design [3].

In spite of above advantages, SFPGAs have an important drawback, which is high susceptibility to some kinds of errors caused by energetic neutrons and alpha particles; is known as Single Event Upset (SEU) [4]. These high energy particles hit the silicon substrate and in some memory elements such as Static Random Access Memory (SRAM) cells, they can complement stored value; this event is identified as soft error [4]. Because the operation of an SFPGA is determined by the values that are stored in SRAM cells, SEU may drastically alter the correct operation of an SFPGA; therefore, designing a system using SFPGAs necessitates a higher degree of fault tolerance against SEU faults.

Memory elements in an SFPGA can be categorized into configuration bits and user defined bits. The Configuration bits are used for specifying mapped circuit in SFPGA, whereas the user defined bits such as Flip-Flop bits hold the current state of the circuit. After mapping a design into an SFPGA, the contents of the configuration memory are supposed to remain unchanged, while the contents of the user bits can be changed at any clock cycle. Because more than 99% of memory bits are configuration bits in modern SFPGAs [4], the probability of soft errors in the configuration bits is more important than user defined bits. As mentioned in [5], SEU faults in configuration memory may lead to serious failures in applications such as bank servers, telecommunication servers, and avionics.

There are some traditional techniques such as Triple Modular Redundancy (TMR) [2, 6] and Duplicating With Comparison (DWC) [2, 6] to increase the dependability of SFPGAs against SEU. These conventional methods impose more than three-time overhead and two-time overhead on area/power consumption, respectively. However, we propose a different architecture to detect SEU in the configuration bits of routing resources in SFPGAs. The Proposed architecture is designed for substituting conventional routing resources which called Switch Box. We have analysed different patterns that can be configured into a Switch Box's configuration memory, and have introduced a method to map these patterns to such Boolean functions that have *self-remap* property. Also, an architecture that exploits *self-remap* property to detect SEU is presented. Experimental results show that the proposed architecture can detect 100% SEU faults in the configuration memory while it imposes 150% area overhead, 55% delay overhead, and 90% power consumption overhead in comparison with the standard Switch Box. The comparison of our results with conventional fault-tolerant architectures highlights the efficiency of the proposed architecture.

The remainder of this paper is organized as follows: some related work is presented in section II, motivation of the proposed architecture is presented in section III, the proposed architecture is introduced in section IV, experimental results

978-1-4244-4946-0/09/$25.00 ©2009 IEEE

are given in section V, and finally, the paper is concluded in section VI.

II. RELATED WORK

Several techniques for mitigating SEU effects in the configuration memory of a Switch Box have been proposed in recent years. All of these techniques imply some kind of redundancy to increase dependability against SEU. Triple Modular Redundancy (TMR) [2, 6] and Duplicating With Comparison (DWC) [2, 6] are two well-known hardware redundancy techniques to mask and detect SEU effects, respectively; However, TMR enforces more than three-time on area overhead as well as on power consumption overhead, while DWC imposes more than two-time on area and on power consumption overheads [2, 6]. Scrubbing [7] is another method based on time redundancy, which reloads the whole content of the configuration memory periodically. In other words, the content of the configuration memory is located in a SEU-immune memory [11] outside of SFPGA, and it is reloaded to the configuration memory periodically.

A method is represented in [3], programs unused SRAM bits in a Switch Box to construct redundant paths between different IO pins. This method can decrease SEU vulnerability up to 50% and imposes acceptable area overhead and power consumption overhead in a Switch Box; however, it is not applicable in all situations and depends on the mapped circuit. Using information redundancy for SEU detection and correction is proposed in [8]. This method uses several Error Detecting And Correcting (EDAC) codes such as parity code and hamming code to detect and correct SEU in the configuration memory of a Switch Box; even though area overhead for such approach is fewer than DWC, its great delay overhead prevents that approach in application like real-time.

III. THE MOTIVATION

A. Self-remap in Boolean Functions

This subsection describes an interesting property identified as *self-remap* in Boolean functions. Let $F(B)$ is a k-input Boolean function, where $B=(b_1, b_2, ...,b_k)$. A *map* for B is a k-tuple $(r_1, r_2, ...,r_k)$ such that $r_i=b_j$ or $r_i=\sim b_j$ for some j, $1 \le i, j \le k$. Given B is $(b_1, b_2, ..., b_k)$, and a *map* R is $(r_1, r_2, ..., r_k)$. Since every r_i can be any of the b_js in its normal or complemented form, there are $(2k)^k$ different possible *maps* for B. Giving a Boolean function $F(b_1, b_2, ...,b_k)$ and a *map* $(r_1, r_2, ..., r_k)$ of $(b_1, b_2, ..., b_k)$, we defined the mapped function $H(b_1, b_2, ..., b_k) = F(r_1, r_2, ..., r_k)$. Fig. 1 illustrates a three-input function $F(A, B, C)$, a *map* of (A, B, C) to $(\sim A, B, \sim C)$, and the mapped function $H(A, B, C)$ where $H(A, B, C) = F(\sim A, B, \sim C)$.

In some cases, it is probable that function F and mapped function H be quite equal; in other words, $H(b_1, b_2, ..., b_k) = F(r_1, r_2, ..., r_k) = F(b_1, b_2, b_k)$. In such cases, F is called a Boolean function with *self-remap* property. An example of such function is shown in Fig. 2. This figure illustrates a three-input function, F, a *map* for its inputs as $(A,B.C)$ to $(\sim A,\sim B,\sim C)$ and the mapped function H. because $H(A,B,C)$ is equal to $F(A,B,C)$, so F is defined as a function with *self-remap* property. It is noteworthy to note that for all combinations of A, B and C, (A, B, C) would not be equal to $(\sim A, \sim B, \sim C)$; in other words:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

F(A,B,C)

~A	B	~C
1	0	1
1	0	0
1	1	1
1	1	0
0	0	1
0	0	0
0	1	1
0	1	0

map (A,B,C)→(~A,B,~C)

A	B	C	H
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

mapped function H(A,B,C)=F(~A,B,~C)

Fig. 1. Example of a map and mapped function

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

F(A,B,C)

~A	~B	~C
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

map (A,B,C) → (~A,~B,~C)

A	B	C	H
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

H(A,B,C)=F(~A,~B,~C)= F(A,B,C)

Fig. 2. Example of a function with self-remap property

A	B	C	F_1	F_2	F_3
0	0	0	1	1	1
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	1
1	1	1	0	0	1

Fig. 3. Example of Three functions without self-remap property

For all A, B, C: (A, B, C) ≠ (~A, ~B, ~C) (1)

Such *map*s are identified as a 0-self map. In fact, we consider only 0-self maps to identify wether a function has *self-remap* property or not. Consequently such *maps* like *(A, B, C)* to *(A, B, C)*, or *(A, B, C)* to *(C, A, B)* are disregarded in evaluating *self-remap* property. It is obvious that if F is a function with *self-remap* property, its output must have some requirements such as the number of ones (zeros) in its output must be an even number. Fig. 3 shows three functions that *self-remap* property can not be assigned them with any different *maps* of A,B and C (only 0-self *maps* are considered). In Section IV, we utilize *self-remap* property for fault detection.

B. Routing Resources

A conventional SFPGA is a 2-D array of routing resources and Logic Blocks. This customary model is common to the architecture of several families of SRAM-based FPGAs [3, 8 and 9] and is used for predefined architecture in VPR tool [9]. The model consists of two kinds of resources: Logic Blocks and Switch Modules. The Logic Blocks contain several four-input LookUp Tables (LUTs) for storing the truth tables of the mapped circuit, while Switch Modules provide interconnection between different Logic blocks and IO pads.

Fig. 4(a) illustrates one of the most well-known types of Switch Modules. This family of Switch Modules consists of four Switch Boxes. As illustrated in Fig. 4(b) Each Switch Box has four input/output wires that can be connected to other wires.

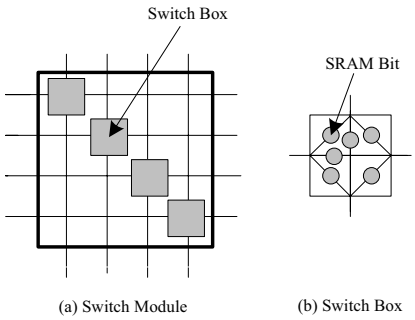

Fig. 4. The structure of Switch Module and Switch Box

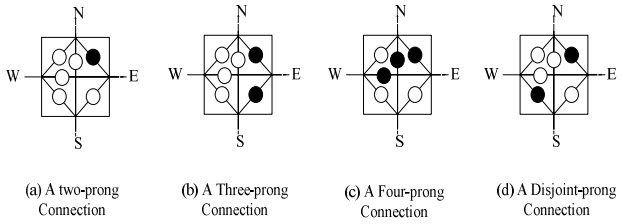

Fig. 5. Types of Switch Box Connection

Also Fig 4(b) illustrates that each Switch Box has six 1-bit SRAM bits. These SRAM bits control connectivity of different wires in a Switch Box. Depending on value that is stored in each 1-bit SRAM cell, the routing structure of an SFPGA is described. Fig. 5 illustrates four different types of connections that are possible based on programming different SRAM bits in a Switch Box. As seen in Fig. 5(a), in two-prong connection type, only one of the SRAM bits is programmed. Three-prong and four-prong connection types (Fig. 5(b) and Fig. 5(c)) are established by programming two and three SRSAM bits, respectively; and finally disjoint connection (Fig. 5(d)) is established by programming two SRAM bits.

To analyse different patterns of SRAM bits in a Switch Box, we develop a routing analysis tool to extract the statistics of each pattern in several standard benchmarks. We have simulated 20 MCNC benchmarks using TVpack and VPR simulation tools [9]. Table I shows the percent of each connection type. The last row of this table shows that 26.58% of connections in the Switch Boxes are disjoint connection type, 45.03% and 10.44% of connections are respectively two-prong and three-prong connection type; 17.00% of connections are crossbar connection type (Crossbar type means a four-prong connection type where its connections are established like a crossbar) and finally 0.95% of connections are four-prong connection type other than crossbar connection type. We use above statistics to convert different patterns of SRAM bits into Boolean functions that have *self-remap* property, then store obtained Boolean functions in an architecture that can detect any SEU in every of SRAM bits.

TABLE I
THE STATISTIC OF CONNECTION TYPES IN MCNC BENCHMARKS

benchmark	Connection types (%)				
	disjoint	two-prong	three-prong	four-prong	
				Crossbar	other
Alu4	27.53	43.77	10.84	12.02	0.87
Apex2	31.74	40.32	10.19	15.08	0.75
Apex4	30.57	42.97	8.82	18.21	0.63
bigkey	17.78	51.98	12.00	17.21	1.25
clma	28.53	43.58	10.06	20.12	0.83
des	18.44	55.23	8.61	21.14	0.72
diffeq	19.82	48.63	13.24	13.00	1.31
dsip	17.91	51.58	11.77	12.25	1.74
E84	23.08	49.76	9.55	15.26	0.64
elliptic	28.64	42.69	10.46	19.26	1.20
Ex1010	26.77	45.75	9.72	24.20	0.76
Ex5p	32.66	41.28	8.47	16.20	0.59
frisc	25.25	48.83	8.09	17.00	0.82
misex	27.88	43.68	10.57	18.21	0.87
pdc	33.04	40.51	8.87	17.32	0.59
S298	29.85	36.69	14.06	14.32	1.87
S38412	24.03	40.20	11.76	21.24	1.01
seq	32.74	56.25	10.18	23.26	0.82
spla	28.70	43.01	10.50	13.00	0.79
average	**26.58**	**45.03**	**10.44**	**17.00**	**0.95**

IV. NEW SWITCH BOX ARCHITECTURE FOR SEU DETECTION

A. Fault detection in functions with self-remap property

Suppose a three-input function, *F*, which is stored in eight SRAM bits. We want to propose an approach to detect SEU faults in any of SRAM bits, if *F* has *self-remap* property. Fig. 6 illustrates the proposed architecture. Please refer Fig. 6 during following descriptions. This architecture consists of eight SRAM bits whose values are streamed to *primary output* depending on *primary inputs*. The *primary output* signal is utilized only for fault tolerant purposes and in cases that all values are required simultaneously, any SRAM bit can directly feed proper destination. There is another multiplexer in Fig. 6 whose select inputs are *mapped inputs*. Upper multiplexer whose select inputs are identified as *primary inputs*, streams one of the stored values into the *primary output* based on the *primary inputs*; at the same time, underneath multiplexer streams another stored value to *remapped output* based on *mapped inputs*. It is very important to notice that two multiplexer can not access the same SRAM bits simultaneously because *mapped inputs* and *primary inputs* are 0-self map.

Let us apply the function *F(A, B, C),* in Fig. 2, to the proposed architecture. In this function*, (A, B, C)* are mapped to *(~A, ~B, ~C)*; then, *primary inputs* are *(A, B, C)* and *mapped inputs* are *(~A, ~B, ~C)*. Because *F* has the *self-remap* property*,* for all *(A, B, C)*, *F(A, B, C)* would be equal to *F(~A, ~B, ~C).* This approach is shown in Fig. 7. It is straightforward that the *primary output* and the *remapped output* would be equal for all A, B and C unless in cases that an SEU has been occurred in on of the SRAM bits.

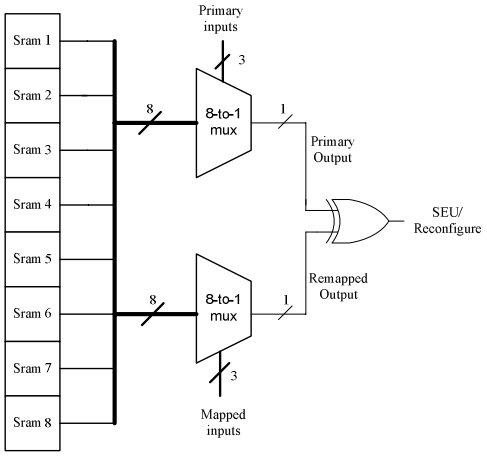

Fig. 6. Fault Detection in three-input function

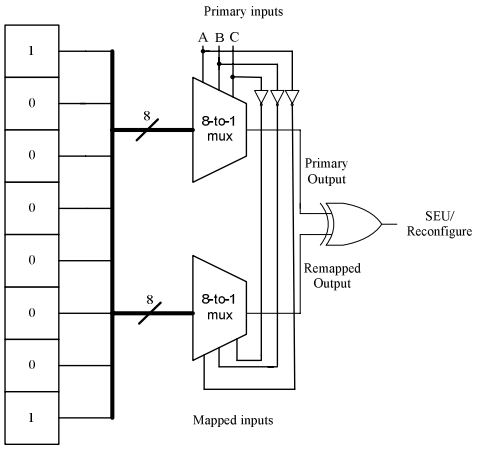

Fig. 7. Mapping of a function to the proposed architecture

So if for all combination of A, B and C the *SEU/reconfigure* would be inactive, we can infer that SRAM bits have not altered by SEU. Let consider another case that an SEU has altered first SRAM cell in Fig. 7. Because *primary inputs* or *A*, *B* and *C* are changing periodically to detect any faults in SRAM cells, when *(A,B,C)* equals to *(0,0,0)*, *mapped inputs* or (*~A*, *~B*, *~C)* would be equal to *(1,1,1)*. Consequently the value of first SRAM cell is streaming into *primary output* (because select lines of upper multiplexer would be equal to 000) and the value of last SRAM cell is streaming into *remapped output* (because select lines of downer multiplexer would be equal to 111). Because the value of first SRAM is complemented by an SEU, the *primary output* and the *remapped output* would not be equal and *SEU/reconfigure* would be active. The effectiveness of this architecture relies on the fact that *primary inputs* and *mapped inputs* access different SRAM cells in same time, therefore, every SEU in SRAM cells can be detected by the proposed architecture. *Primary inputs* are changed periodically to observe all SRAM cells that do not lead to stop the current work of SFPGA, because all SRAM cells connect to proper destinations and primary (mapped) inputs are used to observe all SRAM cells.

The key point for applying this architecture is that the stored function must have *self-remap* property. A method called EDR [5] uses similar property of Boolean functions for mitigating SEU effects in Logic Blocks. Novelty of our architecture is that we add some extra values to the 6-bit patterns of SRAM bits to obtain an 8-bit function that has *self-remap* property, then utilise the architecture like Fig. 6 to detect SEU.

B. SEU-aware architecture

The basic idea for proposing SEU-aware architecture is mapping the stored values of six SRAM bits of a Switch Box to Boolean functions that have *self-remap* property, and utilizing the architecture of Fig. 6 for SEU detection. Fig. 8 shows a Switch Box that a letter is chosen for each SRAM bit. This representation helps us to realize the mapping process

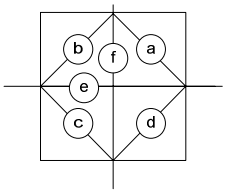

Fig. 8. Naming SRAM bits in a Switch Box

that 6-bit patterns are converted to 8-bit functions with *self-remap* property.

In two-prong connection types, stored patterns can be *(a, b, c, d, e, f) = (1, 0, 0, 0, 0, 0)* or *(0, 1, 0, 0, 0, 0),* and so on. These patterns can be considered as a 6-bit function, F_1, also we can add two extra bits to the end of F_1 and obtain an 8-bit function, F_2. The values of the added cells convert F_1 to a function F_2 that F_2 has *self-remap* property. For example, in such cases that a two-prong connection type like *(a, b, c, d, e, f) = (1, 0, 0, 0, 0, 0),* we add *(0, 1)* to the end of that pattern and obtain *(a, b, c, d, e, f,* ***g***, ***h****) = (1, 0, 0, 0, 0, 0,* ***0***, ***1****),* so F_2*(X, Y, Z) = (1, 0, 0, 0, 0, 0, 0, 1)* (table II, two-prong connection type, column 1) and it is sufficient to map *(X, Y, Z)* to *(~X, ~Y, ~Z)*, in order to detect any SEU faults (F_2 is as same as *F* in fig. 2). In three-prong connection types, stored patterns would be *(a, b, c, d, e, f) = (1, 1, 0, 0, 0, 0)* or *(1, 0, 1, 0, 0, 0),* and so on. Similar method is used to convert these 6-bit patterns to an 8-bit function that has *self-remap* property. For example, if a three-prong connection type like *(a, b, c, d, e, f) = (1, 1, 0, 0, 0, 0),* we add *(0, 0)* to its end and obtain *(a, b, c, d, e, f,* ***g***, ***h****) = (1, 1, 0, 0, 0, 0,* ***0***, ***0****)*, (table II, three-prong connection type, column 1) then it is sufficient to map *(X, Y, Z)* into *(X, Y, ~Z)* in order to detect any SEU faults.

This method also is applicable for disjoint connection type, crossbar connections types and Unused Switch Boxes. Table II illustrates different extra bits that must be added to each connection type to convert it to an 8-bit Boolean function that has *self-remap* property. However, this method can not be applied for all four-prong connection types except crossbar type. To realize, let consider a four-prong connection type such *(a, b, c, d, e, f) = (1, 0, 0, 0, 0, 0, 1, 1)*; in this connection, adding any combination of 0s and 1s does not lead to a function that has *self-remap* property; luckily these patterns are only 0.95% of all patterns in Switch Boxes,

and this problem does insignificant impact on the overall design.

Converting the patterns of Switch Boxes into Boolean functions that has *self-remap* property leads us to propose a new Switch Box architecture, called SEU-aware architecture. For realizing the proposed architecture, six SRAM bits in the traditional architecture (Fig 4(b)) are substituted with eight SRAM. Then obtained 8-bit function that has *self-remap* property is stored in the architecture like Fig. 6. This approach can detect any SEU faults in SRAM bits of a Switch Box. This design can become more efficient by analysing the patterns of added SRAM bits (g and h in table II). In this table, by exhaustive enumeration of all patterns in Switch Boxes, observed that the values of added extra bits are always (0, 0) and (0, 1); thus, we can replace the seventh SRAM bit in our architecture (g in table II) with hardwired zero value. So we can achieve an architecture to detect any SEU by adding only one SRAM bits and two 8-to-1 multiplexers to the traditional architecture. This architecture is shown in Fig. 9. The seventh SRAM bit is replaced by a zero-hardware value based on the above description. This approach descends SEU susceptibility in the proposed architecture and saves a lot of area and power.

The value of added SRAM cell can be stored in a *technology mapper*, when the original configuration bits are given to the *technology mapper* as input, value of the added SRAM cell is produced as output. It is interesting to mention that the proposed architecture is as the same as parity-applied architecture that one extra SRAM bit is added to six bits to detect SEU faults, but in the parity-applied architecture, detector circuit (three levels of two-input XOR gates) imposes large area and delay to the design, but in our proposed architecture, the fault detector is constituting from two 8-to-1 multiplexers that work in parallel and one XOR gate; so the required area, delay and power would be more fewer over parity-applied architecture. In the experimental results, the proposed architecture is validated by a fault injection tool, moreover different parameters such as required area, delay and power consumption are estimated using synopsis® synthesis tool.

V. EXPERIMENTAL RESULTS

This subsection evaluates dependability and efficiency for the proposed architecture. The proposed architecture is described by VHDL language in gate level, and SINJECT fault injection tool [10] is used for fault injection. In experiments, SEU faults are injected in all SRAM cells of the Switch Boxes, these faults might lead a signal getting misrouted or a signal disconnected in the Switch Box. Totally 7000 SEU faults are injected in SRAM cells where all of them are assumed to have an equal probability of being affected by an SEU. Last column of table III shows fault coverage for the proposed architecture. This column illustrates that 100% SEU faults can be detected by the proposed architecture. This is because all *maps* that are used for fault detection are 0-self maps, so the two multiplexers in Fig. 9 access different SRAM cells, therefore all single faults like SEU can be detected immediately. Table III also compares fault coverage for the proposed architecture with three conventional methods that have been proposed a fault-tolerant Switch Box in recent years. It can be seen that all of these methods have 100% fault coverage.

TABLE III
DIFFERENT PARAMETERS IN THE PROPOSED SWITCH BOX

	Overhead (%)			Fault coverage
	Area	Delay	Power	
DWC	192	35	180	100%
Parity method	182	100	100	100%
Our proposed switchbox	150	55	90	100%

For estimating required area, delay and power consumption for the proposed architecture, it is synthesized with synopsis® synthesis tool. The results are presented in table III. It is important to mention that all multiplexers in the proposed Switch Box are implemented by pass transistor gates. That approach leads to reduction in area and power consumption, but it increases delay overhead for the proposed architecture. Table III illustrates that the new Switch Box architecture uses 42% fewer area than the conventional DWC and 32% fewer than parity method, whereas all of these methods have similar fault coverage.

The proposed architecture also saves 90% power consumption in comparison with conventional DWC and 10% in comparison with parity method. High power consumption overhead in DWC is because all SRAM bits must be duplicated in this method. In parity-applied method six XOR gates used as the fault detector circuit that imposes 100% overhead in power consumption. Table III shows delay overhead in our proposed architecture is 20% greater than DWC architecture. This greater delay can be realized because in DWC, The output signal must be passed from one level of gates (six 2-input XOR gates that work in parallel), but in our proposed Switch Box, the output signal must be passed from one multiplexer and one XOR gate that imposes greater delay than DWC method; also notice that multiplexers in our proposed architecture are constructed with pass transistor gates that increases propagation delay; However, delay overhead for our proposed architecture is lower than parity-applied method.

VI. CONCLUSION AND FUTURE WORK

This work presented a new Switch Box architecture that can detect SEU faults in SRAM-based FPGAs. The proposed architecture is based on mapping the patterns of Switch Boxes into Boolean functions that have an interesting property called *self-remap* property, and then proposed a new architecture for utilizing such property for fault detection. Experimental results showed that this method can save power and area over conventional methods and make it applicable where power and area are pivotal majors. Future research will be focused on methods that can raise SEU correction coverage based on the proposed architecture.

TABLE II
MAPPING FUNCTION FOR VARIOUS CONNECTION TYPES

			Connection type																					
			Unused	2-prong						3-prong												Crossbar	Disjoint	
8-bit function	Original SRAM cells	a	**0**	**1**	0	0	0	0	0	**1**	1	1	1	0	0	0	0	0	0	0	0	0	1	0
		b	**0**	**0**	1	0	0	0	0	**1**	0	0	0	1	1	1	0	0	0	0	0	0	0	1
		c	**0**	**0**	0	1	0	0	0	**0**	0	0	0	1	0	0	1	1	1	0	0	0	1	0
		d	**0**	**0**	0	0	1	0	0	**0**	1	0	0	0	0	0	1	0	0	1	1	0	0	1
		e	**0**	**0**	0	0	0	1	0	**0**	0	1	0	0	1	0	0	1	0	1	0	1	0	0
		f	**0**	**0**	0	0	0	0	1	**0**	0	0	1	0	0	1	0	0	1	0	1	1	0	0
	Added SRAM cells	g	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		h	**0**	**1**	1	1	1	1	1	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0

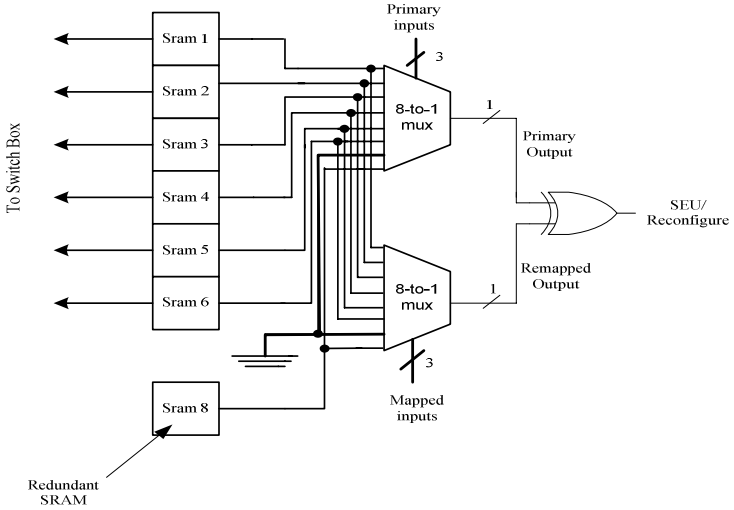

Fig. 9. SEU-aware architecture for Switch Box

REFERENCES

[1] G. Asadi, S. G. Miremadi, H. R. Zarandi, and A. Ejlali, "Fault Injection into SRAM-Based FPGAs for the Analysis of SEU Effects", *In the Proceedings of the IEEE International Conference on Field programmable Technology*, pp. 428-435, 2003.

[2] F. Lima, L. Carro and R. Reis, "Designing Fault Tolerant Systems into SRAM-based FPGAs", *In the Proceedings of International Design Automation Conference*, pp. 650-655, 2003.

[3] H. R. Zarandi, S. G. Miremadi, D. K. Pradhan and J. Mathew, "Soft Error Mitigation in Switch Modules of SRAM-based FPGAs", *In the Proceedings of IEEE International Symposium on Circuits and Systems*, vol. 27, pp. 141-144, 2007.

[4] J. Huang, M. B. Tahoori, and F. Lombardi, "Fault Tolerance of Switch Blocks and Switch Block Arrays in FPGA ", *IEEE Transactions on Very Large Scale Integration Systems*, vol. 13, no. 7, pp. 794-807, 2005.

[5] E. S. Sundar, V. Chandrasekhar, M. Sashikanth and V. Kamakoti, "A Novel CLB Architecture to Detect and Correct SEU in LUTs of SRAM-based FPGAs", *In the Proceedings of the IEEE International Conferences on Field-Programmable Technology*, pp. 121-128, 2004.

[6] F. Lima, G. Neuberger, R. Hentschke, "Designing Fault-tolerant Techniques for SRAM-Based FPGAs", *IEEE Design & Test of Computers*, vol. 21, pp. 552-562, 2004.

[7] S. Srinivasan, A. Gaysen, N. Vijaykrishnan, M.Kandemir, Y.Xie and M. J. Irwin, "Improving Soft-error Tolerance of FPGA Configuration Bits", *In the Proceedings of International Conference on Computer Aided Design*, 2006.

[8] A. Rohani and H. R. Zarandi, "Mitigating and Tolerating SEU Effects in Switch Modules of SRAM-based FPGAs", *In the Proceedings of Southern Programmable Logic Conferences*, 2009.

[9] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research", *International Workshop on Field Programmable Logic and Applications*, 1997.

[10] H. R. Zarandi, G. Miremadi and A. R. Ejlali, "Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models", *In The Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp, 485-492, 2003.

[11] L. R. Rocket, "A Design based on Proven Concepts of an SEU-immune CMOS Configuration Data Cell for Reprogrammable FPGAs", *Journal of Microelectronics,* Vol. 32, No. 1, pp. 99-111, 2001.