A Fault Masking Dual Module Redundancy Method for FPGA

Meisong Zheng*, Zilong Wang*, and Lijian Li *
* Insitute of Automation, Chinese Academy of Sciences
Beijing, 100190, China, meisong.zheng@ia.ac.cn

***Abstract*—In order to solve the problem of single-event upset (SEU) in static-random access memory (SRAM) based field-programmable gate arrays (FPGAs), a *Fault Masking Dual Module Redundancy* (FMDMR) structure is proposed in this paper. The FMDMR method make use of AND/OR logic as dual-module redundancy (DMR) voter. The AND/OR logic are built with unoccupied carry-chains in FPGA; hence no additional hardware overhead are brought about by the insertion of voters. Experiments on MCNC'91 benchmarks show that the FMDMR method can reduce 70% SEU faults on average, with a 2x hardware overhead. It balances between area and reliability, and fits for applications with no rigorous require for reliability.**

I. INTRODUCTION

Field-programmable gate arrays (FPGAs) based on static-random access memory (SRAM) can be reprogrammed by users as many times as necessary, this flexibility makes it more and more widely used in different applications[1, 2]. But such flexibility relies on the programming of SRAM cells, which are very sensitive to various perturbations; especially single-event upsets (SEUs).

Traditional FPGA hardening techniques include triple-module redundancy (TMR) [3], periodicity scrubbing [4], configuration bits verify and readback [5], and dual-module redundancy (DMR) [6]. Recently, in-place fault mitigation algorithms [7, 8], making use of different logic masking techniques, emerges to reduce fault rate in FPGAs. Those methods bring about low or no cost in area, but the fault masking effect is not obvious either.

TMR is the most straightforward fault-tolerant technique[9], but the hardware overhead of TMR method are excessive, generally over 200%, This huge overhead also give rise to high power dissipation and low working frequency. Selective TMR [10-12] can reduce the area overhead with a small loss of SEU immunity, is being used as an alternative to applications with no need of too much require of reliability.

Dual-module redundancy (DMR) can reduce hardware overhead to only 100%, but previous DMR techniques were mostly used for fault detection. A different comparison result means that there is something wrong in one of the twin module and the FPGA system has to stop to repair the error. The major disadvantage of DMR is that it can offer neither fault localization nor fault-free module auto switch when the fault is discovered, which will cause a great decline on the working efficiency. To overcome this problem, a method combines DMR and concurrent-error detection (CED) is proposed [13], when error occurs it needs only one clock cycle in hold operation to detect the faulty module, and after that it will operate normally again without performance penalties. But the designing of CED encode and decode circuit is very difficult, and for complex circuit it is an impracticable task. Therefore [13] is not a suitable option for general use.

This paper provides a *fault masking module redundancy* (FMDMR) structure carried on lookup-table (LUT) level, which adds an AND or OR logic voter after each duplicated LUT pair. This architecture can mask most FPGA errors induced by SEUs on the basis of logic gates nature: AND gate output remains 0 once one of the gate inputs is 0; OR gate output remains 1 once one of the gate inputs is 1. Different LUT outputs have different 0/1 preference, this fact determines whether an AND or OR logic voter will be added to the output of LUT pairs. With the help of abundant dedicated carry-chain resources in Xilinx FPGAs, the insertion of logic voter causes no additional hardware overhead to the system. MCNC'91 benchmarks are applied to validate the SEU mitigation capability of FMDMR.

II. METHOD STATEMENT

The FMDMR method is based on this principle: Logic 0 is control value for AND gates, logic 1 is control value for OR gates. Control value means that the output of a gate is totally controlled by certain logic value of one of its input. For example when one input of an AND gate is 0, the output of the gate will remain 0 no matter what other inputs are. We make use of this property of AND gates to mask 0→1 faults in DMR circuits, and OR gates to 1→0 faults in a similar way.

A. AND/OR Logic Act as DMR Voters

Without faults, the output of a duplicated LUT pair is

This work was supported by National Nature Science Foundation (61073035) under Grant and Beijing Natural Science Foundation (4092013) under Grant.

either <0,0> or <1,1>, both of them can be voted by an AND or OR logic voter. Therefore makes up four types of combinations of DMR outputs and their voters as shown in Fig. 1.

The FMDMR method aims at mitigating SEU induced faults in FPGA, generally only on configuration bit will be affected by SEU, hence it is reasonable to assume that fault happens in one LUT once a time.

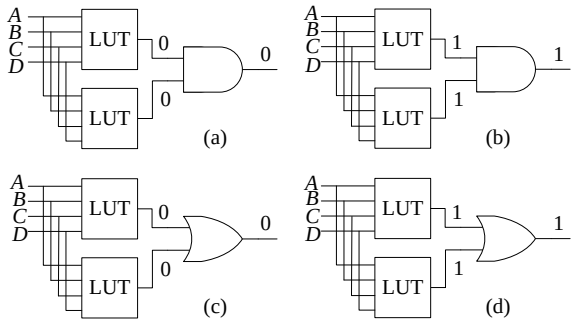

Fig.1 Four types of DMR outputs and their voters

The changed output and masking effect of AND/OR logic voter is shown in Fig. 2. In Fig. 2(a), because both of the inputs hold the control value 0, the 0→1 fault happens on one of the input can be masked by the AND gate. The same case happens on Fig. 2(d). But in the cases of Fig. 2(b) and Fig.2(c) faults happened on inputs propagated to the outputs. A conclusion can be extracted based on those cases: for a duplicated LUT pair, 0 → 1 fault can be masked by an AND logic, and 1 → 0 fault can be masked by an OR logic.

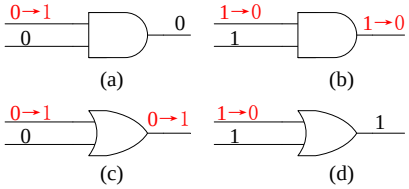

Fig.2 Fault masked by AND/OR logic voter

With the conclusion derived from Fig. 2, the FMDMR fault masking scheme is designed as follows:

Make a copy of every LUT in the circuit (DMR);

Insert an AND logic as DMR voters for 0-preference LUTs;

Insert an OR logic as DMR voters for 1-preference LUTs.

So the problem becomes which LUTs are 0-preference and which are 1-preference. A distinction between 0 and 1 preference for every LUT is made by means of signal probability calculation, which will be described in the next section.

B. Logic Value Preferences of LUT Output Lines

Firstly, the concept of *Signal probability* (P_s) is introduced as the probability of each LUT output line to be sensitized to 1. Then, a threshold H is used to determine whether an LUT is 0-preference or 1-preference. That is:

An LUT is 0-preference if its output O has a signal probability $P_s(O) \leqslant H$; 1- preference if $P_s(O) > H$.

Depends on the functions and application scenarios, different circuits have different input probabilities. For an LUT with its input signal probabilities already known, *access probability* (P_a) of each configuration bit can be obtained, so the output signal probability can be calculated as:

$$P_s(O) = \sum P_a(\text{cell} = 1),$$

in which cell=1 means the configuration bit contains a logic value of 1. For example a 3-input LUT E with input signal probabilities of $P_s(I_2)$=0.4, $P_s(I_1)$=0.5, $P_s(I_0)$=0.6, and a truth table shown as Table I; the output signal probability $P_s(E)$ can be calculated as:

$$P_s(E) = P_a(C_3) + P_a(C_4) + P_a(C_5) + P_a(C_6) + P_a(C_7)$$
$$= (1 - P_s(I_2))P_s(I_1)P_s(I_0) + P_s(I_2) = 0.78.$$

TABLE I
TRUTH TABLE OF AN EXAMPLE LUT

I_2	I_1	I_0	E	Accessed Cell
0	0	0	0	C_0
0	0	1	0	C_1
0	1	0	0	C_2
0	1	1	1	C_3
1	0	0	1	C_4
1	0	1	1	C_5
1	1	0	1	C_6
1	1	1	1	C_7

We assume all primary inputs have signal probabilities of $P_s(I_p)$=0.5 for simplicity, and then propagate them to primary outputs level by level; hence the logic value preferences of each LUT can be obtained. We set H=0.5 in this paper, so if the signal probability of a given LUT is greater than 0.5 it is a 1-preference LUT, otherwise it is 0-preference. Based on this, LUT E shown in Table I is 1-preference.

III. PROPOSED METHOD

A. AND/OR Logic Voters Designed with Carry-Chain

Since the FMDMR is carried on LUT level, this method needs lots of DMR voters to vote the LUT outputs. We make use of dedicated carry-chains in Xilinx FPGAs to build the AND/OR logic. Since the carry-chains are generally not used in most applications, the insertion of voters does not bring about additional

overhead. A simplified diagram of carry-chain logic in one slice is shown in Fig. 3, it is comprised with a multiplexer and exclusive-or gate. Real FPGAs contain more complex data selections and relative connections for the multiplexer and exclusive-or gate than Fig. 3.

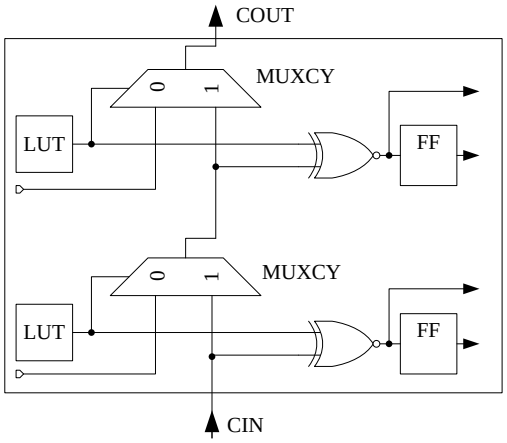

Fig. 3 A simplified carry-chain in one slice

As shown in Fig. 4(a), an AND logic voter is built with a multiplexer:

$$O_{\mathrm{AND}} = \overline{O_1} \cdot 0 + O_1 \cdot O_2 = O_1 O_2.$$

As shown in Fig. 4(b), an OR logic voter is built with a multiplexer combined with an exclusive-or gate:

$$O_{\mathrm{OR}} = \overline{O_2} \cdot (O_1 \oplus O_2) + O_2 \cdot O_2 = O_1 + O_2.$$

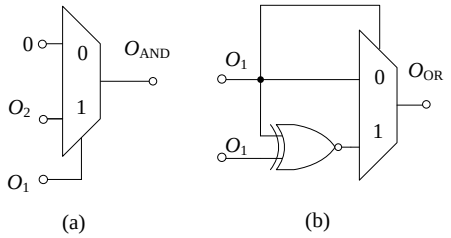

Fig. 4 AND OR logic constructed by carry-chains

B. FMDMR Method Implementation

As stated earlier, an efficient way to reduce the final output error is to insert AND logic voter for 0-prefence LUTs and OR logic voter for 1-prefence LUTs. In order to make the best use of the dedicated carry-chain, original LUTs should be constrained in separate slices. In Xilinx FPGAs, one slice contains 2 LUTs, so that we can set the duplicated LUT pairs exactly in one slice and use the dedicated carry-chain in that slice to build the logic voter. This is realized by the ucf constrains additionally attached to the project. Based on these the FMDMR method is implemented as follows:

1) Map the circuit into LUT structure;

2) Calculate LUT logic value preferences level by level, and mark 0-preference LUTs with AND voter, 1-preference LUTs with OR voter;

3) Copy LUTs in the same slice and build voter as they have been marked in step 2, using the dedicated carry-chain in that slice;

4) Connect the voted output to the original connections.

Step1 is implemented using RASP (Rapid System Prototyping) synthesis and mapping tool[14], Netlists synthesized by RASP boasts more trimly structure and more concise format, hence are easier to be read and disposed by our C++ procedure. What's more the same netlists can also be realized in Xilinx FPGAs using Xilinx Primitives. step2 is a C++ procedure runs on a PC with a 3.2GHz quad core CPU and 4GB memory. Step 3 is realized in verilog accompany with a ucf file indicating the area constraint for each LUT. Step 4 is accomplished using Xilinx synthesize, place and routing tool.

The following is an example of step 3 operates on one LUT:

```
LUT3 #(
  .INIT(8'he0)              // Specify LUT Contents
)LUT3_INST_L2_1 (
  .O(inter2_1),             // LUT general output
  .I0(inter16),             // LUT input
  .I1(inter15),             // LUT input
  .I2(pi)                   // LUT input
);
LUT3 #(
  .INIT(8'he0)
)LUT3_INST_L2_2 (
  .O(inter2_2),
  .I0(inter16),
  .I1(inter15),
  .I2(pi),
);
MUXCY_L MUXCY_L_inst2 (
  .LO(inter2),      // Carry local output signal
  .CI(inter2_2),    // Carry input signal
  .DI(0),           // Data input signal
  .S(inter2_1)      // MUX select, tie to '1' or LUT4 out
);
```

with the ucf statements:

```
INST "LUT4_INST_L2_1"  LOC = slice_X5Y10 | BEL = F;
INST "LUT4_INST_L2_2"  LOC = slice_X5Y10 | BEL = G;
```

LUT3_INST_L2_1 and LUT3_INST_L2_2 are same functional 3-LUT pairs (with configuration bits 0xe0) with same inputs (inter16, inter15, pi); the output of them is inter2_1 and inter2_2, respectively. An AND logic voter is inserted by the implementation of multiplexer MUXCY_L_inst2, it votes inter2_1 and inter2_2 with an output inter2:

$$\begin{aligned} \text{inter2} &= \overline{\text{inter2\_1}} \cdot 0 + \text{inter2\_1} \cdot \text{inter2\_2} \\ &= \text{inter2\_1} \cdot \text{inter2\_2}. \end{aligned}$$

The ucf statements are used to constrain the LUT pair resides in the same slice.

IV. Experimental Results

In this section, the proposed FMDMR method is tested on standard MCNC'91 benchmarks by means of logic-simulation fault injection mechanism. The algorithm was implemented in C++ and tested on a PC with a 3.2GHz quad core CPU and 4GB memory. In the end of this section, the simulation results are validated on a real FPGA.

A. Fault Injection Method

The logic simulation fault injection flow is shown in Fig. 5.

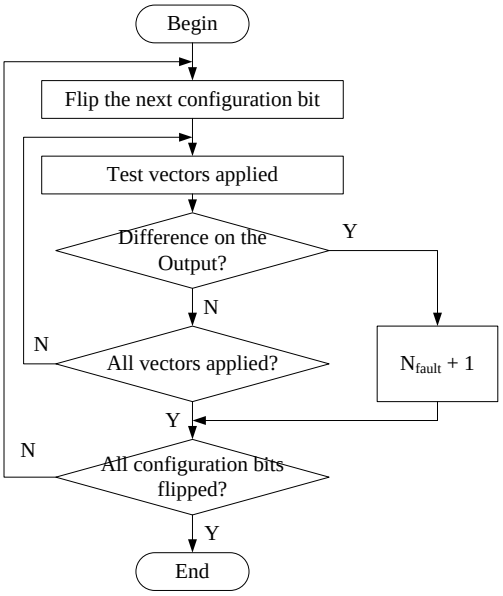

Fig. 5 Logic-simulation fault injection experiment flow

We reverse the configuration bit once a time for every LUT. Once a configuration bit is flipped, test vectors are applied to circuit inputs and the outputs are compared with an error-free circuit. The number of fault (N_{fault}) will be added with one when a difference is detected. The next configuration bit will be reversed after all test vectors have been applied or a fault is detected on this reversed configuration bit.

Once a fault injected to an LUT, the fault effect depends on its redundancy property. Only on the following two cases will the injected fault really reverse the LUT configuration bit:

1) LUT pair voted by AND logic whereas the fault is 1 →0.

2) LUT pair voted by OR logic whereas the fault is 0 →1.

Otherwise the fault point LUT remains its configuration memory, means that it is immune to the injected faults.

For circuits with input number less than 12, such as *cm152a* and *alu2*, we applied full input space simulation for every injected fault. For circuit with input number more than 12, such as *c1355* and *c432*, we applied 1,000 test vectors generated by a linear-feedback shift register (LFSR).

B. Experimental Results and Analysis

Some results of combinational MCNC'91 benchmark circuits are shown in Table II. Column 2 titled with "N_{inputs}" shows the number of primary inputs of each circuit; column 3 titled with "N_{lut}" shows the number of LUTs of each circuit; and column 4 titled with "N_{bits}" shows the total number of configuration bits of each circuit. Columns 5 titled with "F_{ORI}" shows the number of configuration bits propagates fault to the output while flipped in the original circuit; Columns 6 titled with "F_{FMDMR}" shows the number of configuration bits propagates fault to the output in the FMDMR hardened circuit. The last column titled with "Reduced" shows the percent of faults reduced by the FMDMR method compared with the original circuits.

TABLE II
Experimental Results

Circuits	N_{inputs}	N_{lut}	N_{bits}	F_{ORI}	F_{FMDMR}	Reduced
cm82a	5	4	64	32	16	50.00
cm42a	4	10	160	160	10	93.75
cm138a	6	10	160	96	10	89.58
cm152a	11	6	96	79	23	70.89
cc	21	26	416	237	61	74.26
c8	28	39	624	410	122	70.24
c1355	41	74	1184	1081	467	56.80
c499	41	74	1184	1077	464	56.92
ttt2	24	75	1200	737	231	68.66
term1	34	88	1408	656	174	73.48
c432	36	124	1984	1128	448	60.28
x1	51	143	2288	1277	370	71.03
c880	60	174	2784	1712	500	70.79
alu2	10	197	3152	1666	400	75.99
Average			1193	739	235	70.19

From Table II, we can see that 62% (739/1193) of the configuration bits are SEU sensitive without protection; while the RTMR hardened FPGA reduced that proportion to 20% (235/1193). Compared with the original circuit, FMDMR method can reduce 70% faults on average, for some circuits like *cm42a* and *cm138a* even 90%.

C. Verification Test on Real FPGA

We also implemented the FMDMR method on a real FPGA in order to verify the validity. The test platform is

based on Xilinx XC4VSX35 with a 100MHz oscillator and 4 LEDs. We implemented the cm152a circuit on the FPGA and inject faults on its configuration bits.

The experiment scheme is shown in Fig. 6. The Golden part is error-free circuit without fault injection. The ORI and FMDMR part represents the original cm152a circuit and the FMDMR hardened cm152a circuit, both of the ORI and FMDMR circuit is configured with one configuration bit fault in one LUT. The configuration bit faults are injected at the time of FPGA configuration. When FPGA configuration is finished, Input vectors are applied to the three circuits with the frequency of 100MHz on the test board, and the outputs of ORI and FMDMR are compared with the Golden one. Differences between the outputs drive the test board LEDs, LED1 represents error detected in ORI circuit; while LED0 represents error detected in FMDMR circuit.

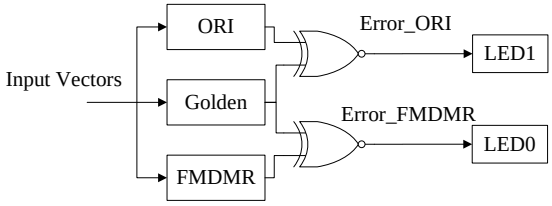

Fig. 6 Real FPGA test scheme

The cm152a circuit is composed with one 3-input LUT and five 4-input LUTs, hence the total LUT configuration bits equals to $5\times2^4+1\times2^3=88$. So the FPGA is reconfigured 96 times, the state of LED1 and LED0 are recorded after each reconfiguration. Lightening of LED1 represents an error detected in ORI circuit; while lightening of LED0 represents an error detected in FMDMR circuit.

The experiment showed 79 times of ORI circuit errors ($N_{\text{Error\_ORI}}$=79) and 23 times of FMDMR circuit errors ($N_{\text{Error\_FMDMR}}$=23). This result behaves the same with the logic-simulation fault injection mechanism described in Section IV-C, further demonstrates the reasonability of FMDMR.

Because of no additional hardware overhead is brought about by the insertion of voters, the total overhead of FMDMR is only 2x of the original circuit while TMR method uses 3x of the original area. In conclusion, FMDMR is an effective FPGA reinforce method.

V. Conclusion

This paper presents a fault masking DMR technique (FMDMR) with AND/OR logic voter for FPGA SEU mitigation. Compared to traditional DMR methods, this architecture needs neither reset to recover from errors nor additional error judgment circuit to switch to the fault-free part. What's more, the insertion of voters does not induce additional hardware overhead.

Results on MCNC'91 benchmark show that circuits hardened by FMDMR can reach a relatively high SEU immunity level at a reasonable area overhead. Since SEUs are probability events, FMDMR can guarantee the stability of system operation combine with a certain frequency of FPGA scrubbing. It could be applicable to scenarios in which a very high reliability is not mandatory, while area is constrained.

References

[1] G. Ge, M. E. Kaye, and Z. Yun, "Enhancement of Gaussian background modelling algorithm for moving object detection & its implementation on FPGA," in *IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2015, pp. 118-122.

[2] C. E. Kennedy and M. Mozaffari-Kermani, "Generalized parallel CRC computation on FPGA," in *IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2015, pp. 107-113.

[3] C. Carmichael, "XAPP197.Triple modular redundancy design techniques for Virtex FPGA's (DRAFT)," 2006.

[4] C. C. Brendan Bridgford, Chen Wei Tseng, "XAPP779.Correcting Single-Event Upsets in Virtex-II Platform FPGA Configuration Memory," February 19 2007.

[5] K. Chapman, " XAPP864. SEU Strategies for Virtex-5 Devices," V2.0, April 1, 2010.

[6] H. H. Ng, " XAPP564. PPC405 Lockstep System on ML310," January 29, 2007.

[7] U. Kretzschmar, A. Astarloa, J. Lazaro, M. Garay, and J. Del Ser, "Robustness of different TMR granularities in shared wishbone architectures on SRAM FPGA," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)* , 2012, pp. 1-6.

[8] H. Keheng, H. Yu, and L. Xiaowei, "Reliability-Oriented Placement and Routing Algorithm for SRAM-Based FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 256-269, 2014.

[9] H. El-Razouk and Z. Abid, "A New Transistor-Redundant Voter for Defect-Tolerant Digital Circuits," in *Canadian Conference on Electrical and Computer Engineering(CCECE'06)*, 2006, pp. 1078-1081.

[10] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, pp. 2957-2969, 2004.

[11] V. Chandrasekhar, S. N. Mahammad, V. Muralidaran, and V. Kamakoti, "Reduced Triple

Modular Redundancy for Tolerating SEUs in SRAM-based FPGAs," in *Proceedings of NASA International Conference on Military Applications in Programmable Logic Devices (MAPLD)*, 2005.

[12] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-Grain SEU Mitigation for FPGAs Using Partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, pp. 2274-2280, 2008.

[13] F. G. de Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis, "Designing fault-tolerant techniques for SRAM-based FPGAs," *IEEE transactions on Design & Test of Computers*, vol. 21, no. 6, pp. 552-562, 2004.

[14] J. Cong, J. Peck, and Y. Ding, "RASP: A general logic synthesis system for SRAM-based FPGAs," in *Proceedings of the ACM fourth international symposium on Field-programmable gate arrays*, 1996, pp. 137-143.