Permanent Fault Repair for FPGAs with Limited Redundant Area

Shu-Yi Yu and Edward J. McCluskey
Center for Reliable Computing – Stanford University, Stanford, California
{syu, ejm}@crc.stanford.edu

Abstract

FPGA fault repair schemes remove faulty elements from designs through reconfiguration. In designs with high FPGA utilization, a sufficient number of routable fault-free elements may not be available for permanent fault repair. We present a new permanent fault repair scheme, in which the original design is reconfigured into another fault tolerant design that has smaller area, so the damaged element can be avoided. Three new schemes that fully utilize available fault-free area and provide low impact on availability are presented. Analytical results show that our schemes improve availability compared to a module removal approach, which removes a redundant module when it becomes faulty.

1. Introduction

Field-Programmable Gate Arrays (FPGAs) provide a solution to permanent fault repair in finer granularity because of their regular structure and reconfiguration capability. In FPGAs, a faulty module can be repaired by reconfiguring the chip so that a damaged configurable logic block (CLB) or routing resource is not used by a design. Many techniques have been presented to provide permanent fault removal for FPGAs through reconfiguration. One approach is to generate a new configuration after permanent faults are detected in computing systems. CAD tools and algorithms for fast FPGA re-routing and re-mapping were developed [1-2]. Another approach is to generate pre-compiled alternative FPGA configurations and store the configuration bit maps in non-volatile memory, so that when permanent faults are present, a new configuration can be chosen without the delay of re-routing and re-mapping [3-4].

With the repair schemes mentioned above, permanent faults in an FPGA can be repaired as long as there are sufficient routable fault-free elements on the chip so that designs can avoid using faulty elements. However, for designs that use high percentage of FPGA resource, all routable fault-free elements could be exhausted. In addition, for very long mission times, routable fault-free elements can be exhausted due to multiple permanent fault repairs. In these cases, further permanent damage can no longer be repaired. To solve this problem, the design has to be reconfigured into a new fault tolerant design with smaller area, so that the permanent faults can be avoided. A traditional technique is to remove design elements at the module level, such as TMR/Simplex [5] or self-purging redundancy [6]. However, a system with redundant modules removed may have much lower availability than the original system. For example, consider a TMR system with error detection and a duplex system, which is obtained by removing a module from TMR because of permanent faults. When a transient fault occurs in each system, the TMR system can still function, but the duplex system has to be stopped for recovery. Hence, a duplex system has lower availability than a TMR system. In this paper, we present a new method of permanent fault repair in FPGAs to reduce impact on system availability by reconfiguring the original design into a new fault tolerant design that fully utilizes the reduced available fault-free FPGA area.

This paper is organized as follows. Section 2 defines the problem we are addressing. Section 3 presents three design methods for permanent fault repair. Section 4 describes an analytic model of a system with error recovery, and describes metrics we use to evaluate the designs. Section 5 shows the analytical results and compares these design methods. Section 6 concludes the paper.

2. Problem definition

The problem we are solving here is how to find a configuration for an FPGA-based design when available chip area is reduced because of permanent faults. The new configuration needs to have small availability reduction compared to the module removal approach. Since TMR is widely used in fault tolerant systems [7] and can be used for longer mission times when it is implemented with recovery mechanisms [11], we focus on area reduction in TMR systems.

To explain the features required of an FPGA-based design, Fig. 1 illustrates the error recovery steps for FPGA-based systems. First, concurrent error detection (CED) mechanisms detect an error. If an error occurs for the first time, it is treated as a transient error; otherwise, it is treated as a permanent error. When a transient error occurs, the system recovers from corrupt data and resumes normal operation. When a permanent fault occurs, fault diagnosis is initiated to determine the location of the damaged resource, and a suitable configuration is chosen according to the available area. Then computation is resumed.

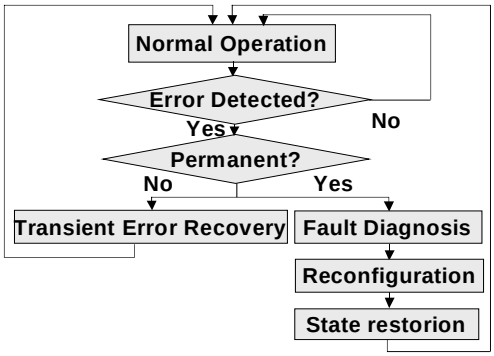


Figure 1. Error recovery in FPGAs.

Possible transient recovery schemes include *rollback* [8] and *roll-forward* recovery [9-11]. In rollback recovery, the system state is backed up to some point in its processing, which is called a *checkpoint*. When an error occurs and is detected during operation, the system will restore its state back to the previous checkpoint and re-compute. However, re-computation has time overhead. In roll-forward recovery, the system will recover the corrupt data by copying correct data from fault-free redundant modules to the faulty module. Hence, re-computation delay is avoided.

From the error recovery flow, it is clear that CED and transient error recovery schemes are required for FPGA-based designs. We assume that fault diagnosis and reconfiguration control mechanisms are available off-chip. These mechanisms can be implemented in another FPGA for fault tolerance and recovery purposes [12]. Configuration files can be pre-compiled and stored in memory. States and data for recovery after permanent fault repair are stored by the other FPGA, too. CED and transient recovery mechanisms are built on-chip within an FPGA-based design. States and data for transient error recovery are handled by the design itself. In this paper, we present designs that have less area than TMR but still have error recovery capability.

3. Design candidates

We present three fault tolerant design techniques: (1) hybrid TMR-Simplex-CED, (2) duplex with a checking block, and (3) duplex with two CED blocks. Figure 2 (a) shows a TMR system, and Figs. 2 (b), (c), and (d) illustrate the design techniques respectively. These designs all contain the features required for error recovery and are able to recover from single faults. Unlike conventional fault tolerant designs, these designs are adjusted according to the available FPGA area. When a transient fault occurs, rollback or roll-forward recovery is used depending

on location of faults and design structure. When a permanent fault occurs, an error indication is raised. The choice of rollback or roll-forward will be explained in the following paragraphs.

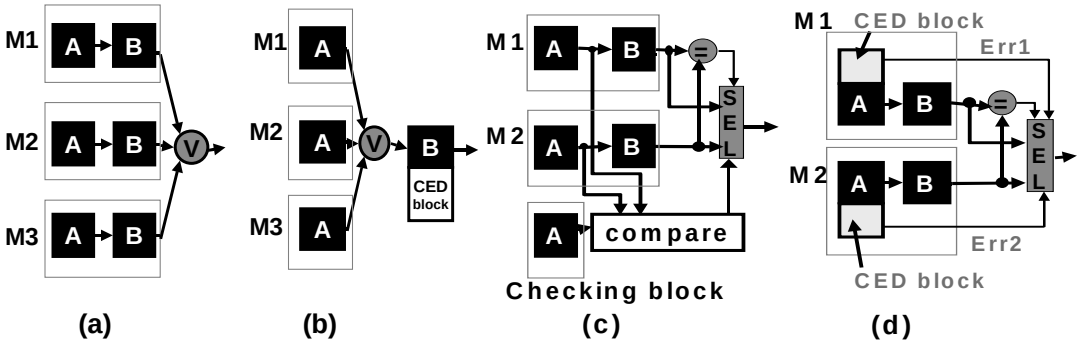


Figure 2. Design candidates modified from TMR. (a) The original TMR design. (b) A hybrid TMR-Simplex-CED design. (c) A duplex system with a checking block. (d) A duplex system with two CED blocks.

In a hybrid TMR-Simplex-CED design, we partition the original simplex design into two parts. As shown in the example of Fig. 2 (b), the design is partitioned into blocks A and B. A is triplicated; and CED, such as a parity checking block or diversed duplication [13], is added to B. A and B can have signals connected to each other. The area reduced by the design compared with TMR is approximately the area of block B. The partition depends on the available FPGA area and design constraints. When errors occur in the partition with CED, rollback is used. Otherwise, when errors occur in the partition with TMR, roll-forward is used [11]. The partition choice depends on design characteristics and block usage. In a real design, a more frequently used block should be protected by TMR since its fault is more likely to affect the whole system. However, block usage needs further measurement [14] and is not part of our paper.

In a duplex system with a checking block, one of the original three modules in TMR is changed to a checking block. The checking block implements a portion of the normal function of a module, and the portion size depends on the available FPGA area. As shown in Fig 2 (c), the original M3 is changed to block A, and the area saved is approximately the area of block B. The outputs of the checking block are used to check the outputs from blocks A in M1 and M2. The two modules, M1 and M2, are compared with each other. When there is a mismatch between the duplex modules, the module that agrees with the checking block is selected, and roll-forward is used to restore data in the other one. If there is no mismatch between the two modules but they disagree with the checking block, roll-forward is used to restore data in the checking block. When there is a mismatch between the duplex modules and they both agree with the checking block, rollback is used. This situation happens when faults occur in block B.

A duplex system with two CED blocks is shown in Fig. 2 (d). The outputs from the duplex are compared to each other. CED blocks are built in both modules. When the remaining area on an FPGA is no longer sufficient for implementing CED for two whole modules, CED is only added for part of the design. For example, in Fig. 2 (d), CED is added to detect only errors in blocks A, but not in blocks B. Recovery approaches depend on CED results and the comparison between the duplex. When there is a mismatch between the two modules and only one of the CED blocks indicates error, the outputs from the module without error indication is selected as final outputs, and roll-forward is used to restore data in the other module. When there is a mismatch between the duplex but both or neither of the CED blocks indicates an error, rollback recovery is used. When there is no mismatch between the two modules but both of the CED blocks indicates errors, rollback is used. When there is no mismatch between the duplex but one of the CED blocks indicates an error, no recovery action is taken.

We are going to compare our designs with a traditional module removal approach. For a TMR system, one approach to module removal for permanent fault repair is to remove the faulty module and change it to a duplex system [15]. Roll-forward can be used for transient error recovery in TMR, and rollback can be used in a duplex. We denote the module removal approach as *TMR-Duplex* throughout the paper. Figure 3 illustrates the idea of TMR-Duplex.

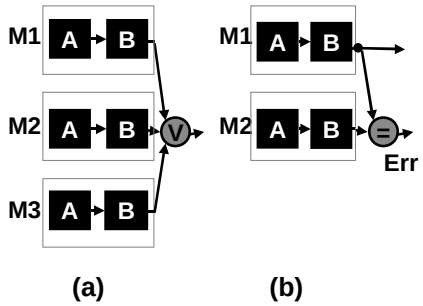


Figure 3. Permanent fault repair through a module removal approach, TMR-Duplex: (a) the original TMR design, and (b) duplex.

4. Evaluation metrics

In this section, we describe metrics used to evaluate the designs in Sec. 3. Consider a computing system with both rollback and roll-forward recovery capability. When a transient fault occurs, the choice of rollback or roll-forward depends on the location of the fault. Errors are assumed to occur independently in each unit area in unit time. We use the following metrics to evaluate a design:

Rollback probability: it is the probability of initiating rollback recovery when a system is under normal function. It represents the overhead of recovery. For each rollback recovery, the computation process rollbacks to its previous checkpoint. Hence, the average re-computation time for each checkpoint is the product of the rollback probability and the time between two consecutive checkpoints. In a real-time application, a high rollback probability may not be acceptable, since a deadline may be missed because of re-computation. Rollback probabilities are dominated by the effect of single transient errors happening in blocks that only have an extra copy but no CED or checking blocks, which are blocks B in Figs. 2 (b) through (d). Since we assume errors happen independently in unit area, rollback probabilities can be approximated as probabilities of transient errors happening in the blocks B, that is,

Rollback probability = Transient error rate per unit area × area of blocks B.

Expected lifetime: it is the expected time from the moment that a system starts running to the moment the system fails. The relationship between *Mean Time To Failure* (*MTTF*) and expected lifetime is shown in Fig. 4. MTTF is the average time from start to failure. In our schemes, since repair is taken into account, we use the expected lifetime as our metric.

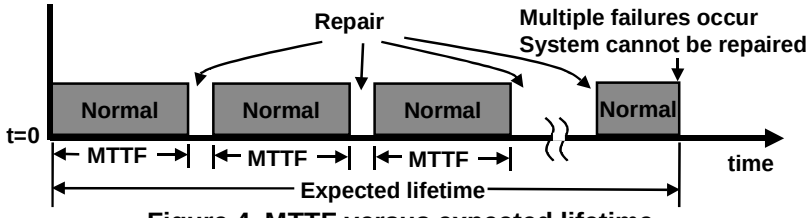


Figure 4. MTTF versus expected lifetime.

When the permanent error rate is much smaller than the rate of system repair and when the probability of failure during repair process is smaller than the probability of system failure during normal operation, expected lifetime can be approximated as

Expected lifetime ≈ (1 / Prob (system fails between two checkpoints)) checkpoint interval time. The derivation is in [16]. Expected lifetime depends mainly on the probability of double errors. This is because single errors do not cause failure of repair or recovery, and double errors happen more frequently than errors more than two. For different designs, a system failure occurs in different fault scenarios. We list the scenarios as follows:

(1) Hybrid TMR-Simplex-CED: a failure occurs when faults occur in the simplex block and are not detected, or the simplex block is fault-free but two or three blocks of TMR are faulty. Therefore, the probability of system failure is

Prob(block B fails)×(1 – CED coverage)
+Prob(block B is fault-free)×Prob(2 or 3 of blocks A fail)

(2) A duplex with a checking block: in Table 1, all the cases that cause failure for the design are listed. M1 and M2 are the two modules of the duplex system. The notation of A and B are the blocks A and B in Fig. 2 (c). In each entry, "0" means the block is faulty, "1" means it is fault-free, and "d" means don't care. The "Result" columns indicate if the overall module is faulty. These scenarios can be divided into three categories. In category I, two or three blocks A (including the checking block) are faulty, therefore the checking block cannot make correct comparison. In category II, both blocks B are faulty, and none or one of blocks A is faulty, therefore no rollback is taken action to correct faults in blocks B. In category III, M1 and M2 are both faulty but one has faults in block A and the other has faults in block B. In this case, the checking block will decide to roll-forward, which cannot recover the system from failure. Therefore, the probability of system failure is

Prob(2 or 3 of blocks A fail) +Prob(0 or 1 of blocks A fails, but both of blocks B fail)
+Prob(The checking block is fault-free)
×Prob(One of the duplex fails in block A and the other fails in block B).

Table 1. Fault scenarios that cause failure in a duplex with a checking block.

Category	M1			M2			Checking Block (A)
	A	B	Result	A	B	Result	
I	0	d	0	0	d	0	0
	0	d	0	0	d	0	1
	0	d	0	1	d	D	0
	1	d	0	0	d	0	0
II	0	0	0	1	0	0	1
	1	0	0	0	0	0	1
	1	0	0	1	0	0	0
	1	0	0	1	0	0	1
III	1	0	0	0	1	0	1
	0	1	0	1	0	0	1

(3) A duplex system with two CED blocks: a failure occurs when both the modules are faulty but at least one of them is undetected. To explain this, Table 2 lists all possible cases of detection when both the modules are faulty. The two modules of the duplex system are denoted as M1 and M2. The probability of system failure is

Prob(both modules are faulty)×(1-Prob(Faults in both modules are detected)).

Table 2. Scenarios that cause failure in a duplex with two CED blocks.

Faults in M1	Faults in M2	Consequence
Detected	Detected	Rollback is used for recovery. → No failure
Detected	Undetected	Data in M2 is used for roll-forward recovery. → Failure
Undetected	Detected	Data in M2 is used for roll-forward recovery. → Failure
Undetected	Undetected	No recovery action is taken. → Failure

5. Comparison

In this section, we compute the evaluation metrics, rollback probability and expected lifetime, for different design techniques and compare them with the module removal approach, TMR-Duplex. CED overhead depends on the error detection mechanism and applications. Parity prediction and duplication are widely used in general applications [17-18]. Duplication takes 100% overhead plus the overhead for comparison between duplicated hardware. The overhead of parity checking block can be as high as 90% or even more than 100% [19]. Many CED techniques are also developed to minimize overhead for specific applications. For example, arithmetic codes are commonly used in arithmetic operations [20]. In [21], the overhead of residue code with mod 7 checking for a 16-bit multiplier is as low as 31.1%. Because CED overhead varies from very small to over 100%, we examine the design candidates with three different CEDs as examples: (1) duplication with 100% overhead, (2) parity prediction with 90% overhead, and (3) arithmetic code with 30% overhead. Our schemes require circuits to be partitioned into two parts. A partitioned multiplier may not have CED overhead as low as 30%. However, for a circuit consisting of arithmetic units, partition can be made along boundaries of these units instead of partitioning each arithmetic unit. Since these arithmetic units are not partitioned, low CED overhead may still be achieved by implementing arithmetic error detecting codes within each unit.

For duplication, we assume that faults can be detected when they occur only in one of the duplicated blocks. To calculate the lower bound on the metrics, we assume that faults cannot be detected for all other cases. Hence, the detection coverage for duplication is

Prob (faults only happen in one of the duplicated blocks) / Prob (any fault occurs).

In real designs, design diversity [13] can be used to improve detection coverage for duplication. For other CED techniques, we assume they have 100% single fault detection coverage. To calculate the lower bound on the metrics, we assume that multiple faults are not detectable. Hence, the detection coverage for CED is

Prob(single faults occur in either the original logic or the CED logic)/Prob(any fault occurs).

Multiple-fault detection techniques can be used to increase CED coverage.

Figures 5 through 7 are the metrics plotted with respect to the total available FPGA area. The metrics are normalized to those of a duplex system. The area axis is normalized to the area of a simplex system. All designs fully utilize the available area except TMR-Duplex. In TMR-Duplex, because the design is changed to a duplex when the available area is smaller than area of a TMR, the design area is twice the simplex area.

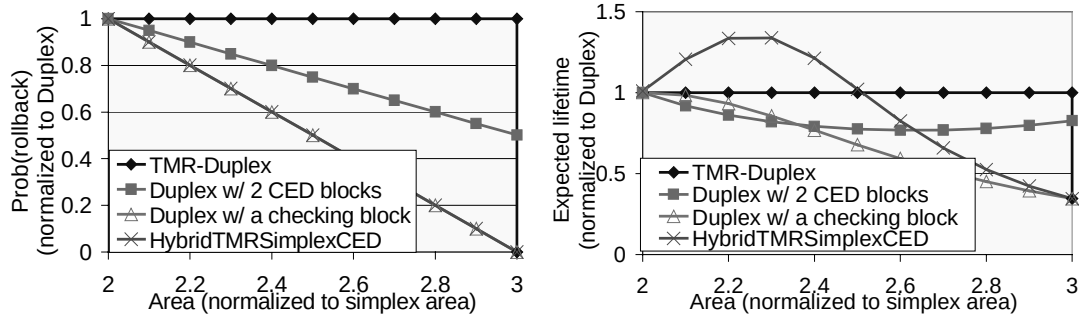


Figure 5. Duplication, 100% overhead: (a) rollback probability, and (b) expected lifetime.

In Fig. 5 (a), the hybrid TMR-Simplex-CED has the same rollback probability as the duplex with a checking block. The reason is that when duplication is used, the hybrid TMR-Simplex-CED has the same block B area as the duplex with a checking block. The duplex with two CED blocks has higher rollback probability. This is because the design uses more area for CEDs than other designs, and as a result, its block B has smaller area.

In Fig. 5 (b), TMR-Duplex has the longest lifetime when the available area is close to 3 times the simplex area. This is because although other designs have more fault tolerant features, the TMR-Duplex has the smallest area among all and is less vulnerable to multiple faults. When the available area is closer to twice the simplex area, the hybrid TMR-Simplex-CED has the longest lifetime. This is because when duplication is used for CED, the hybrid-TMR-Simplex-CED is the series connection of a TMR and a duplex. It can recover double faults when one of them occurs in the TMR part and the other occurs in the duplex part. However, in a plain duplex design, double faults can cause failure when they occur in different modules. When the total available area is smaller, we can see the hybrid-TMR-Simplex-CED design as a 5-block design (3 for TMR, 2 for simplex-CED) where each block is small; and a plain duplex design as a 2-block design where each block is large. Hence, the probability of two-block failure, which will cause system failure, in a 2-large-block design can be larger than that in a 5-small-block design. The hybrid TMR-Simplex-CED design has equal or longer lifetime than duplex with a checking block, regardless how much the available area is. The reason is that in the hybrid TMR-Simplex-CED design, errors in one block of TMR will not be propagated to module outputs. However, in the duplex with a checking block, errors will always be propagated to module outputs unless they occur in the checking block. The lifetime of a duplex with two CED blocks first decreases then increases when the available area decreases. With high detection coverage, a duplex with two CED blocks can have fault tolerance capability very close to a *duplex pair*, that is, a system with double duplicated modules (which are totally four modules.) However, when the CED overhead is high, the CED cannot be built over the whole module when the available area is not sufficient. When the area first decreases, the CED area decreases and the overall detection capability decreases. As a result, the average lifetime decreases. As the area decreases further, the overall CED coverage does not decrease much, but the overall area is small so that it is less vulnerable to multiple faults; hence, the average lifetime increases.

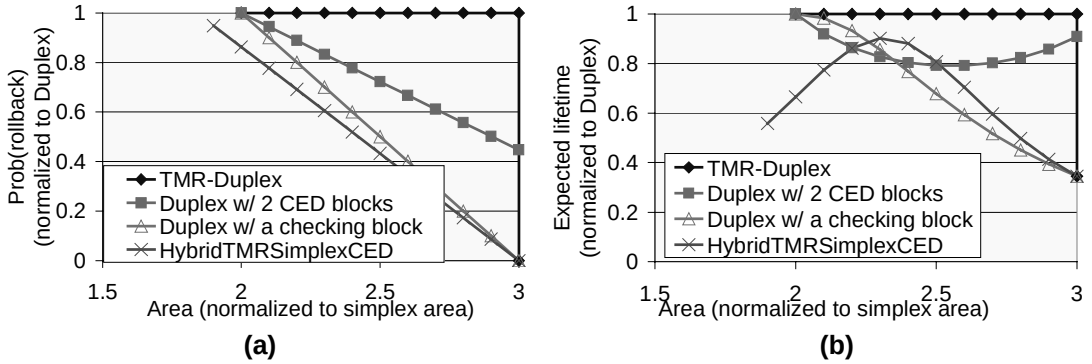
In Fig. 6 (a), the hybrid TMR-Simplex-CED has the lowest rollback probability among all. The reason is that when CED overhead is lower than 100%, the hybrid TMR-Simplex-CED has the smaller portion of block B than the duplex with a checking block (the checking block can be seen as 100% overhead), and the duplex with two CED blocks always have larger area used for CED than other designs.

In Fig. 6 (b), TMR-Duplex has the longest lifetime among all. Although the CED overhead is 90% and is not much smaller than duplication, but the results are quite different. The reason is that in our pessimistic assumption, duplication has higher detection coverage than other single-fault detection techniques. Therefore, the designs including hybrid TMR-Simplex-CED and duplex with two CED blocks, which use CED with single fault detection capability, do not have lifetime as long as in Fig. 5 (b). However, a plain duplex design has less design area so that it is has longer lifetime than other designs.

In Fig. 7 (a), the results are similar to Fig. 6 (a) except for the duplex with two CED blocks. When the area is closer to 3 times the simplex area, the duplex with two CED blocks has the lowest rollback probability among all. This is because when the CED overhead is small, the design can still have CED blocks that detect errors in most of the area even the available area is decreasing. In this case, the design has similar fault tolerance capability as a *duplex pair*, which has two pairs of duplex modules. Therefore, the design can have very small rollback probability. The hybrid TMR-Simplex-CED has lower rollback probability than Fig. 5 (a) because of smaller CED overhead.

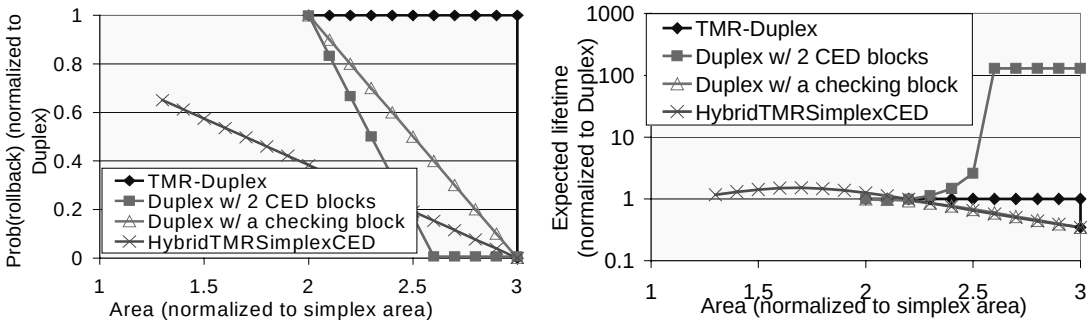
In Fig. 7 (b), when the area is closer to 3 times the simplex area, the duplex with two CED blocks has the longest lifetime among all. This is because the design has similar fault tolerance capability as a duplex pair, as discussed in the previous paragraph. When the area is close to 2 times the simplex area, hybrid TMR-Simplex-CED has the longest lifetime because it has the largest blocks A than any other design with the same design area.

In summary, we find that the metric values are CED dependent, and the choice of designs depends on CED techniques and available FPGA area. For non-real-time applications, the average lifetime is the critical criterion for choosing a suitable design. For real-time applications, the choice depends on rollback probability since timing is a critical issue. Table 3 lists the choice of designs with different available area and CED techniques.



(a) **(b)**

Figure 6. 90% overhead CED: (a) rollback probability, and (b) expected lifetime.



(a) **(b)**

Figure 7. 30% overhead CED: (a) rollback probability, and (b) expected lifetime.

Table 3. Design selection with different available area and CED techniques.

CED	Applications	Area	
		~3×Simplex area	~2×Simplex area
Duplex	Real-time	Hybrid-TMR-Simplex-CED	Hybrid-TMR-Simplex-CED
	Non-real-time	TMR-Duplex	Hybrid-TMR-Simplex-CED
90%	Real-time	Hybrid-TMR-Simplex-CED	Hybrid-TMR-Simplex-CED
	Non-real-time	TMR-Duplex	TMR-Duplex
30%	Real-time	Duplex w/ Two CED blocks	Hybrid-TMR-Simplex-CED
	Non-real-time	Duplex w/ Two CED blocks	Hybrid-TMR-Simplex-CED

6. Conclusion

In this paper we presented three new permanent fault repair schemes for FPGA-based computing systems. When no routable fault-free elements are available for conventional permanent fault repair schemes, our schemes reconfigure the design into another fault tolerance design, which needs smaller area. We have examined three design structures to adapt to limited FPGA area and to provide reliability and availability. These designs are found to have improved availability compared to the module removal approach. Hence, they are more suitable for real-time applications where availability is a critical issue. For non-real-time applications, the module removal approach may be more suitable than the presented three techniques since it has longer lifetime. The choice of designs depends on the CED overhead. For an actual implementation, information of multiple error detectability of CED and design diversity for

module redundancy needs to be obtained to select suitable designs according to the available area.

Acknowledgements

The authors would like to thank Dr. Nirmal Saxena, Dr. Subhasish Mitra, Dr. Wei-Je Huang, and Siegrid Munda for their useful feedback. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-97-C-0024.

References

[1] Emmert, J.M., and D. Bhatia, "Incremental Routing in FPGAs," *Proc. of 11th Annual IEEE Int. ASIC Conf.*, pp. 217-221, 1998.

[2] Dutt, S., V. Shanmugavel, and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconfiguration in Field Programmable Gate Arrays," *Digest of Technical Papers, IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 173-176, 1999.

[3] Lach, J., W.H. Mangione-Smith, and M. Potkonjak, "Algorithms for Efficient Runtime Faulty Recovery on Diverse FPGA Architectures", *DFT'99*, pp. 386-394, 1999.

[4] Huang, W.-J. and E.J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA Fault Tolerance," to appear in *FCCM'01*, 2001.

[5] Mathur F. P. and P. DeSousa, "Reliability Modeling and Analysis of General Modular Redundant Systems," *IEEE Trans. Rel.*, R-24, No. 5, pp. 296-9, 1975.

[6] Losq, J. "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy," *IEEE Trans. Comput.*, C-25, No. 6, pp. 269-78, 1976.

[7] Siewiorek, D.P. and R.S. Swarz, *Reliable Computer Systems – Design and Evaluation*, 3rd Ed, Digital Press, 2000.

[8] Chandy, K. M. et al., "Rollback and recovery strategies for computer programs," *IEEE Trans. on Computers*, vol. C-21, No. 6, pp. 546-56, 1972.

[9] Long, J., W., W. K. Fuchs, and J. A. Abraham, "A Forward Recovery Using Checkpointing in Parallel Systems," *Proc. of the 1990 Int. Conf. on Parallel Processing*, Vol. 1, pp. 272-275, 1990.

[10] Pradhan, D. K., and N. Vaidya, "Roll-forward Checkpointing Scheme: Concurrent Retry with Nondedicated Spares," *IEEE Workshop on Fault Tolerant Parallel and Distributed Systems*, pp. 166-74, 1992.

[11] Yu, S.-Y. and E.J. McCluskey, "On-line Testing and Recovery in TMR Systems for Real-Time Applications," to appear in *ITC'01*, 2001.

[12] Huang, W.-J. and E.J. McCluskey, "A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations," *FPGA'01*, pp. 183-192, 2001.

[13] Mitra, S., N.R. Saxena, and E.J. McCluskey, "A Design Diversity Metric and Reliability Analysis for Redundant Systems," *Proc. 1999 Int. Test Conf.*, pp. 662-671, 1999.

[14] Elder, J.H., Osborn, J., Kolasinski, W.A., and Koga, R., "A Method for Characterizing a Microprocessor's Vulnerability to SEU," *IEEE Trans. on Nuclear Science*, Vol. 35, No. 6, pp. 1678-81, 1988.

[15] Jing, M.-H. and L.J.C. Woolliscroft, "A Fault-Tolerant Multiple Processor System for Space Instrumentation," *Int'l Conf. on Control*, Wol. 1, pp. 411-416, 1991.

[16] Yu, S.-Y. and E. J. McCluskey, "Permanent Fault Repair for FPGAs with Limited Redundant Area," Stanford CRC TR 01-2, 2001.

[17] Spainhower, L. and t. A. Gregg, "S/390 Parallel Enterprise Server G5 Fault Tolerance," *IBM Journal of Research Development*, Vol. 43, pp. 863-873, Sept./Nov. 1999.

[18] Zeng, C., N.R. Saxena, and E.J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection," *Proc. 1999 Int'l Test Conf.*, pp. 672-680, 1999.

[19] Mitra, S. and E.J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" *Proc. 2000 Int. Test Conf.*, pp. 985-994, 2000.

[20] Lin, M.-B. and A.Y. Oruc, "A Fault-Tolerant Permutation Network Modulo Arithmetic Processor," *IEEE Trans. on VLSI Systems*, Vol. 2, No. 3, Sept. 1994.

[21] Sparmann, U., "On the Effectiveness of Residue Code Checking for Parallel Two's Complement Multipliers," *IEEE Trans. on VLSI Systems*, Vol. 4, No. 2, pp. 227-39, 1996.