RT Level Reliability Enhancement by Constructing Dynamic TMRs

Naghmeh Karimi
ECE Department
University of Tehran

Shahrzad Mirkhani
ECE Department
University of Tehran

Zainalabedin Navabi
ECE Department
University of Tehran

Fabrizio Lombardi
ECE Department
Northeastern University

{naghmeh, shahrzad} @ cad.ece.ut.ac.ir

{navabi, lombardi} @ ece.neu.edu

ABSTRACT

This paper presents a novel and efficient approach for reliability enhancement at the RT level. The reliability enhancement is performed by utilizing the available resources of a design in their dead intervals. Such resources are used for constructing dynamic TMR structures that can change per clock cycle. In this method all resources participate in constructing TMR structures at least once per a system input to output flow.

To evaluate the proposed fault tolerance technique we consider dependability, and area/latency overhead imposed on a circuit by applying our method. In order to evaluate dependability, faults are injected into our test circuits before and after applying our algorithm and fault coverage is measured. Experimental results show that after applying our method, fault coverage is significantly reduced indicating that the reliability of designs is improved.

Categories and Subject Descriptors: B.8.1 [Performance and Reliability]: Reliability, Testing, and Fault-Tolerance

General Terms: Reliability

Keywords: Reliability, Fault Tolerant, RTL Design, TMR

1. INTRODUCTION

With the growth of VLSI technology and moving toward submicron and nano technologies, reliability has become an important issue. In comparison to the 10^{-9} to 10^{-7} failure rates in CMOS technology, the failure rates in nanotechnologies are projected to be in the order of 10^{-2} to 10^{-1}[15]. In addition, due to device size in nanotechnologies, fabrication yield has dramatically been dropped. Thus the need for a cost-effective reliability enhancement method is more important now than before.

There are two major methods for increasing the reliability of a circuit: fault avoidance and fault tolerance. However considering fault avoidance as the sole provision for reliability enhancement is not appropriate, since this method just deals with design and manufacturing faults, i.e., the faults that occur during the normal operation of the circuit are not considered using fault avoidance method [5]. Hence considering fault tolerance, the ability of a system to operate correctly in spite of the occurrence of faults, becomes an important issue. The ultimate goal of fault tolerance is to prevent system failures from ever occurring. Various fault tolerance schemes are utilized to ensure the reliability of a system.

GLSVLSI'07, March 11–13, 2007, Stresa-Lago Maggiore, Italy.

These schemes detect errors concurrently, and in case of a fault, they utilize a recovery mechanism to recover from errors.

Online and concurrent testing [3] are used for testing a circuit during its normal operation. Unlike offline testing methods, e.g., traditional BIST structures, online testing methods detect a fault in faulty circuits as soon as it occurs. This increases the reliability of the systems, and at the same time, eliminates the need for the system to be halted during the test process.

There are several groups of online test methods. The simplest of these has stored vectors that when matched with normal data, generating circuit responses that are compared with the expected responses of the circuit [3]. In another method, LFSRs and MISRs are used for test vectors and expected outputs [16]. A different method duplicates independently defined components for TMR and other replication methods [11-12]. However, due to the extensive hardware overhead in the resource duplication methods, algorithmic duplication methods have been proposed. In the algorithmic duplication methods, the operations (as opposed to resources) are duplicated. This results in less, but still considerable hardware overhead [12, 4, 13, 14, 6, 2].

In invariant-based methods in addition to computing f(x) for input x of a circuit, function g(x) witch has a well defined relation to f(x) is generated. In these methods the correct operation of the circuit is verified by checking the relation between f(x) and g(x) during the normal operation of the circuit [1, 8].

While duplication methods are utilized in error detection schemes, *N* replication ($N \geq 3$) techniques are considered in error recovery methods. Triple module redundancy (TMR) is widely used in error recovery schemes. In a TMR system each module is replicated three times. While the module and its replicates are feed with the same inputs, their outputs are connected to a majority voter which produces the system output [9]. System reliability in TMR structures can be evaluated by Markov model.

This paper presents an efficient method to reduce the amount of hardware overhead imposed by applying a traditional TMR structure to a design while reliability of design is enhanced simultaneously. In our proposed method, fault detection and recovery is performed by utilizing the available resources in their dead intervals (the clock cycles during which they are inactive). To evaluate the proposed method we utilize fault coverage metric. Unlike test methodologies in which high fault coverage shows high testability, in fault tolerant methods low fault coverage represents high reliability. This means that the goal of fault tolerant methods is to decrease fault coverage of a design.

Besides area overhead reduction, another advantage of this method is that the resulted circuit has 100% resource coverage, i.e., all resources of the circuit participate in a TMR structure at least once per an input to output flow. Since in this method each resource can participate in different TMR structures in different clock cycles, we

call these TMR structures as dynamic TMR structures. This method can be applied to sequential circuits as well as combinational circuits or non-linear and linear circuits.

The remainder of the paper is organized as follows. Section 2 discusses our proposed reliability enhancement algorithm (called RED3) and section 3 shows the results of applying this method to different benchmarks. The performance of RED3 is evaluated by performing fault simulation before and after applying this method on each test case. Section 4 deals with conclusions.

2. RELIABILITY ENHANCEMENT by DYNAMIC TMR (RED3)

This section presents our efficient online testing and reliability enhancement algorithm. RED3 uses the DFG of a design as its input and modifies the given DFG to construct a fault tolerant structure. Since the modification is applied during the RTL synthesis process, the hardware and latency overhead are much lower than the methods which apply the reliability enhancement algorithms to a post synthesis structure [17-18]. In addition, utilizing the available resources of a design to construct dynamic TMR structures results in lower hardware and latency overhead. Utilizing RED3, the faulty unit of a design can also be diagnosed.

2.1 Algorithm Description

RED3 enhances the reliability of a design, linking itself with the synthesis process. In this process, the behavioral description of a circuit is converted to a DFG. The operations of this DFG are scheduled and then bound to available resources. Applying RED3 on the resulted scheduled and bound DFG enhances the reliability of the given circuit, i.e., a fault tolerant circuit is constructed. Constructing a fault tolerant structure depends on the number of busy and free resources of each resource type in different clock cycles of a scheduled and bound DFG. Let's assume that F_k (B_k) represents the number of free (busy) resources of type k in a given clock cycle of a design. Several cases can occur depending on the amount of F_k and B_k in a given clock cycle of a DFG assuming that we have N_k total resources of type k.

Scenario 1 (F_k > 1): In this scenario, there is at least two free k-type resources. We can construct a dynamic TMR structure using each two free k-type resources with a busy one of the same type. These resources are with the same set of inputs while their outputs are connected to a majority voter which produces the TMR structure output. Thus if one of these resources is faulty, i.e., the results of these resources are not equal, the fault is masked.

Scenario 2 (N_k< 3): In this scenario, there is only one (two) k-type resource(s). In this case, we have to add 3- N_k resources to the data path to construct a TMR structure.

Scenario 3 ($\Sigma F_K < \left\lceil \frac{2}{3} N_K \right\rceil$): In this scenario, the sum of k-type free resources in an input to output flow of the given DFG is fewer than $\left\lceil \frac{2}{3} N_K \right\rceil$. In this case, we add at most $\left\lceil 2\frac{n}{m} \right\rceil$ resources to the datapath. n is the number of k-type resources that have not participated in constructing dynamic TMRs after processing the DFG, and m is the latency of the DFG. Adding these resources to the original DFG, results in a fault tolerant DFG using scenario 1.

2.2 Selection of Resources to Test

According to the above discussion, for selecting resources to construct dynamic TMR structures, at each clock cycle we need to know the number of free and busy resources of each type (F_k, B_k of type k). Since a DFG represents only the busy operations and resources during the circuit operation, we present another flow graph, *Resource Usage Graph* (RUG), which can demonstrate the number of free resources as well as busy ones. Using RUG and the above scenarios, certain heuristics decide on timing and mechanisms of constructing dynamic TMR structures.

2.2.1 Resource Usage Graph

RUG is a graph in which vertices represent the available resources and the edges represent the data transfer between these resources.

Consider the scheduled and bound DFG shown in Figure 1. All operations of this circuit are performed during four clock cycles. Shown below are binding of operations of this DFG to adders (*A1*, *A2*, and *A3*) and multipliers (*M1*, *M2,* and *M3*):

- $+_9$ is bound to *A1*
- $+_1$ to $+_4$ are bound to *A2*
- $+_5$ to $+_8$ are bound to *A3*
- $*_1$ to $*_4$ are bound to *M1*
- $*_5$ to $*_7$ are bound to *M2*
- $*_8$ is bound to *M3*

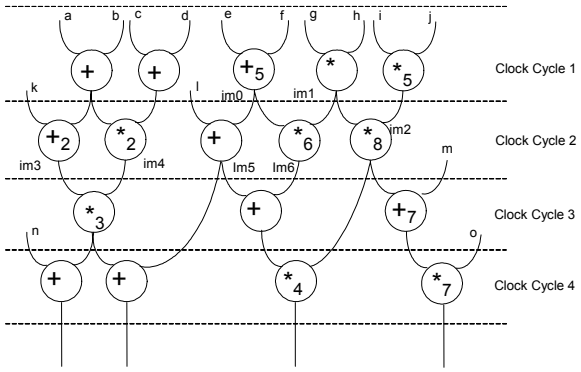

Figure 1. A simple DFG

Figure 2 shows the corresponding RUG of the DFG in Figure 1. This graph shows utilization of available resources versus time. Applying the method in Section 2.1 on the RUG of Figure 2, various scenarios discussed above will be done. In this RUG, in clock cycle 3, *M1* is busy while *M2* and *M3* are free. According to Scenario 1, these three modules can construct a dynamic TMR structure to enhance the reliability of the whole design.

There is at most one free adder in clock cycles 1 to 4. Thus to increase the reliability of design, an extra adder must be added to the available resources (with *Scenario 3*). In this case if *A1*, the extra adder and *A2*, construct a dynamic TMR structure in clock cycle i ($2 \le i \le 4$), *A1*, the extra adder and *A3* can construct another TMR structure in any clock cycle other than i (of course not clock cycle 1 since in this clock cycle adder *A1* is also busy).

2.2.2 RED3 Pseudo Code

Figure 3 shows a pseudo code for our proposed method. The following definitions are used in this pseudo code.

- N_{cc}: Total clock cycles in the original RUG.
- N_{types}: Number of available resource types.
- F_k (B_k) : Number of k-type free (busy) resources in the current clock cycle ($1 \le k \le N_{types}$)
- N_k: Number of k-type resource instances. Note that N_k is constant during all clock cycles. ($N_k=F_k+B_k$ at each clock cycle)

- NTF_k (NTB_k): Number of *k*-type free (busy) resources that have not participated in any TMR structure yet.
- $R_{k,j}$: The j^{th} *k*-type resource instance ($1 \le k \le N_{types}$, $1 \le j \le N_k$).

Note that parameters F_k, B_k, NTF_k, and NTB_k are re-calculated at the beginning of each clock cycle.

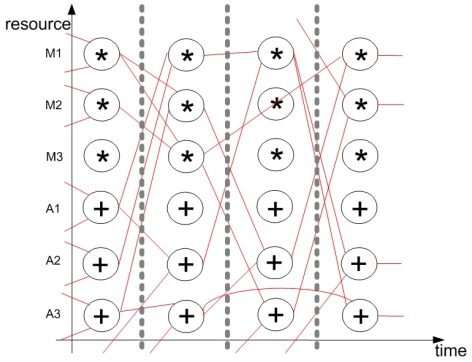

Figure 2. RUG constructed from the DFG of Fig. 1

In the simple RUG shown of Figure 2, *k* is 1 for the adder and 2 for the multiplier. The above parameters are evaluated as follows.

- $N_{cc} = 4$ / $N_{types} = 2$ / N_k: (N1=3 / N2=3)
- F_k: in clock cycles 1 to 4 (F_1: 0, 1, 1, 1 / F_2: 1, 0, 2, 1)
- B_k:in clock cycles 1 to 4 (B1: 3, 2, 2, 2 / B_2: 2, 3, 1, 2)
- NTF_k (NTB_k): Is determined during the algorithm.
- $R_{k,j}$: (Adder: R_{11}, R_{12}, R_{13} / Multiplier: R_{21}, R_{22}, R_{23})

2.2.3 Resource Selection Heuristics

RED3 uses a RUG as its input, processes the RUG starting from clock cycle 1, and produces a modified RUG as its output. In RED3 all function units participate in constructing dynamic TMR structures. Thus the reliability of the resulting RUG is much higher than that of the original RUG. Obviously this algorithm treats each operation independently, and for an operation all of its corresponding resource instances are processed simultaneously.

For each resource type, clock cycles of the given RUG are analyzed consecutively until all instances of that resource type participate at least once in a dynamic TMR structure. Based on the number of free and busy resources, different choices exist for constructing TMR structures. We consider the following heuristics for making these decisions.

Starting with the first clock cycle if any resource instance can participate in a TMR structure (using other available instances), it will become part of that TMR structure. I.e., the process of reliability enhancement is done *as soon as possible*. The priority of constructing a TMR structure using instances not participated in any TMR structure is higher than constructing a TMR structure by use of instances participated in other TMR structures.

2.2.4 Resource Selection Example

Our algorithm implies the modification of the circuit's controller to produce extra control signals in order to enhance the reliability of the design. Figures 4 shows the corresponding RTL datapath after applying our proposed algorithm to the RUG of Figure 2. For the sake of clarity, registers are not shown and only partial routings and the necessary parts of the circuit are depicted here.

The original circuit (Corresponding to RUG of Figure 2) uses three multipliers (*M1*-*M3*) and three adders (*A1*-*A3*). Using the RUG of Figure 2 and scenarios discussed, we decide to construct a TMR using *M1*, *M2*, and *M3*. However, because of utilization of the adders as indicated in the RUG, an extra adder (*EA*) is needed to construct two TMR structures for adder-based reliability enhancement. While *A2* and *A3* participate in one TMR, *A1* participate in two TMRs (Figure 4).

```
for k=1 to N_types loop
Lable1: cc := 1;
  while (true) loop
    Calculate F_k and B_k in clock cycle=cc; Calculate NTF_k from F_k and NTB_k from B_k;
    if (N_k>2) { --N_k=B_k+F_k
      if (NTB_k /= 0) { if (F_k ≥2)
          {Construct TMR structures using each 2 free instances and a busy one;
          Recalculate F_k, B_k, NTF_k, and NTB_k; }}
      if (NTF_k ≥2) { if (B_k /= 0) {
          Construct TMR structures using each 2 free instances with a busy one;
          Recalculate F_k, B_k, NTF_k, and NTB_k;}}
      if (F_k ≥2){ if (B_k /= 0) {
          Construct TMR structures using each 2 free instances with a busy one;
          Recalculate F_k, B_k, NTF_k, and NTB_k;}}}
    else --N_k ≤2
      {N_k <= 3; Goto Lable1;}
    cc++; --go to the next clock cycle
    if (cc > N_cc) break;
  end loop;
  if (not(all R_kj.TMR are true)) – all R_kj do not participate in TMR structures
    Add ⌈2((NTB_K + NTF_K) ÷ N_cc)⌉ resources of type k to N_k; Goto Lable1;}
end loop;
```

Figure 3. The proposed pseudo code for RED3

3. EXPERIMENTAL RESULTS

To evaluate RED3, we have applied the proposed algorithm to three benchmark circuits: a 4^{th} order IIR filter, a 6^{th} order FIR filter, and the Discrete Cosine Transform (DCT) circuit [7].

We have prepared three models from RT level of the above testcases as follows:

- *Normal Model:* This model includes the post synthesis circuit without any reliability enhancement.
- *Dynamic TMR Model:* This model represents the post synthesis circuit after applying RED3 method.
- *Traditional TMR Model:* This model represents the post synthesis circuit of the traditional TMR of the whole circuit, i.e. this model includes three copies of the RTL testcase feeding with the same input. The outputs of these copies are connected to a majority voter.

To evaluate RED3 we consider dependability, and area/latency overhead imposed on a circuit by applying our method. To evaluate dependability, the discussed models have been fault simulated using DSM-FS (a VHDL fault simulator) [10] with 10000 random test vectors, and fault coverage metric has been used for reliability measurement (Table 1). Note that although this method considers transient and intermittent faults, the fault model we have considered in our test cases is *single stuck-at* fault model. Table 1 shows that in our testcases the fault coverage in normal models is much higher than the fault coverage in dynamic TMR models (44% reduction on average). This indicates that the RED3 structure of each design is more reliable than the original design.

Comparing dynamic TMR method with traditional TMR method, Table 1 shows that the area overhead of the dynamic TMR model is considerably lower than the area overhead imposed by the traditional TMR model (lass than 50%). On the other hand, the latency overhead is insignificant (1% on average). Comparing the dynamic TMR model with the normal model, Table 1 shows an increase of 67% average area overhead. This increase is more than 150% if we use traditional TMR models as our reliable models.

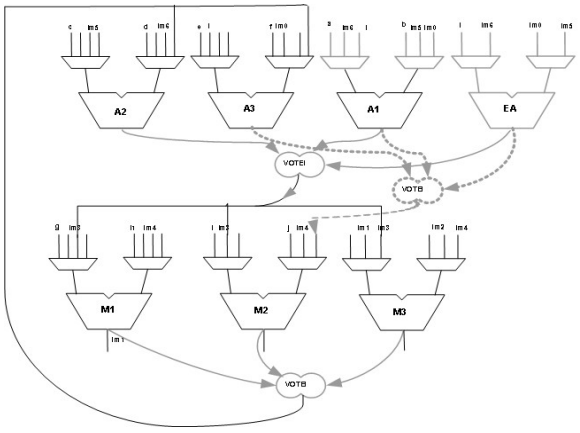

Figure 4. RED3 RTL structure of the RUG in Fig. 2

4. CONCLUSIONS

This paper presents a reliability enhancement method. The proposed method uses free resources in their dead intervals to enhance the reliability of the whole design. The main advantage of the proposed method is its low fault coverage and area overhead as compared with methods using resource replication, e.g., TMR.

This method improves manufacturing yield by dynamically masking manufacturing faults with an acceptable hardware overhead. Furthermore, our method causes a circuit to recover from faults occur during the operation of the circuit.

5. REFERENCES

[1] Bayraktaroglu. I, and Orailoglu. A. Concurrent test for digital linear systems. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, 2001, 1775-791.

[2] Bolchini, C. et al. Concurrent error detection at architectural level. *In Proceedings of the 11th international symposium on System synthesis*, 1998, 72-75.

[3] Bushnell, M. L., and Agrawal, V. D. *Essentials of electronic Testing for Digital Memory and Mixed Signal VLSI Circuits*. Kluwer Academic Publishers, 2000.

[4] Hamilton, S. N., and Orailoglu, A. On-Line test for fault-secure fault identification. *IEEE Trans. VLSI Systems*, Vol. 8, No. 4, August 2000.

[5] Johnson, B. W. *Design and Analysis of Fault-Tolerant Digital Systems.* Addison-Wesley, 1989.

[6] Karri, R., and Iyer, B. Introspection: A register transfer level technique for concurrent error detection and diagnosis in data dominated designs. *ACM Trans. Design Automation of Electronic Systems*, Vol. 6, No. 4, 2001, 501-515.

[7] Lee, M. T. *High-Level Test Synthesis of Digital VLSI Circuits*. Artech House, 1997.

[8] Makris, Y., Bayraktaroglu, I., and Orailoglu, A. Enhancing reliability of RTL controller-datapath circuits via invariant-based concurrent test. *IEEE Trans. Reliability*, Vol. 53, No. 2, 2004.

[9] Mitra, S. *"Diversity Techniques for Concurrent Error Detection.* Ph.D. Thesis, Stanford University, 2000.

[10] Navabi, Z., Mirkhani, S., Lavasani, M., and Lombardi, F. Using RT level component descriptions for single stuck-at hierarchical fault simulation. *Journal of electronic testing-Theory and Applications*, Vol. 20, December 2004, 575-589.

[11] Nicolaidis, M., and Zorian, Y. On-line testing for VLSI- A compendium of approaches. *Journal of electronic testing-Theory and Applications*, Vol. 12, No. 1-2, 1988, 7-20.

[12] Oikonomakos, P., and Zwolinski, M. Using high-level synthesis to implement on-line testability. *IEEE Real-Time Embedded System Workshop*, Dec. 2001.

[13] Oikonomakos, P., Zwolinski, M., and Al-hashimi, B. M. Versatile high-level synthesis of self-checking datapaths using an on-line testability metric. *In Proceedings of. Design Automation and Test in Europe*, March 2003.

[14] Orailoglu, A., and Karri, R. Automatic synthesis of self-recovering VLSI systems. *IEEE Trans. Computers*, Vol. 45, No. 2, Feb. 1996.

[15] Rao, W., Orailoglu, A., and Karri, R. Fault tolerant arithmetic with applications in nanotechnology based systems. In Proceedings of International Test Conference, 2004, 472-47.

[16] Voyiatzis, I.and Paschalis, A. R-CBIST: An effective RAM-based input vector monitoring concurrent BIST technique. In *Proceedings of International Test Conference*, 1998.

[17] Voyiatzis, I., et al. An efficient comparative concurrent built-in self test technique. *In Proceedings of Asian Test Symposium*, 1995.

[18] Voyiatzis, I., et al. A concurrent built-in self test architecture based on a self-testing RAM. *IEEE Trans. Reliability*, Vol. 54, No. 1, March 2005.

Table 1. Applying RED3 on different benchmarks.

Circuit Name	Area (Normal model)	Area (Dynamic TMR model)	Area (Traditional TMR model)	Delay (Normal model)	Delay (Dynamic TMR model)	Delay (Traditional TMR model)	Normal Fault Coverage	RED3 Fault Coverage
6th order FIR filter	1079	1432	3259	46.53	48.65	46.63	87%	28%
4rd order IIR filter	1275	1505	3816	52.81	53.14	52.91	48%	40%
DCT	936	1841	2894	51.3	54.93	53.08	69%	36%