SoC yield Improvement

Using TMR Architectures for Manufacturing Defect Tolerance in Logic Cores

Julien Vial Arnaud Virazel Alberto Bosio Luigi Dilillo Patrick Girard Serge Pravossoudovtich
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier – LIRMM
Université de Montpellier II / CNRS
161, rue Ada
34392 Montpellier Cedex 5, France
Email: {vial, virazel, bosio, dilillo, girard, pravo}@lirmm.fr URL: http://www.lirmm.fr/~w3mic

Abstract—**Manufacturing processes in the nanoscale era are less reliable leading to lower yields. As memories are the most important contributor to SoC (System-on-Chip) yield, fault tolerance techniques based on redundancy are generally used to improved memory yield. Conversely, logic cores embedded in SoC usually do not have these important features and manufacturing defects affecting these cores decrease the yield of the entire SoC. Therefore, meaningful techniques for SoC yield improvement must also consider logic cores. In this paper, we propose and investigate the use of TMR (Triple Modular Redundancy) architectures for logic cores to increase the overall SoC yield. We also propose to improve the defect tolerance capabilities of TMR architectures by partitioning logic cores and adding voters on logic core's cuts. Results show that this improvement of the TMR architecture is very fruitful to improve its tolerance capability with a low overhead in term of silicon area. Results obtained on SoC examples (ISCAS'85 and ITC'99 benchmark circuits as logic cores merged with memory cores with different rates) demonstrate the interest of using TMR architectures for logic cores for SoC yield enhancement.**

Keywords-system-on-chip; logic cores; manufacturing defects; yield ramp-up; fault-tolerance; TMR; test of tolerant architecture.

I. INTRODUCTION

System on Chip (SoC) has becoming a widely accepted architecture for complex and heterogeneous systems that include digital, analog, mixed-signal, radio frequency, micromechanical, and other types of components on a single piece of silicon. The context of this study is related to SoCs that are composed of two types of cores: memory and logic cores, where memory cores occupy the major silicon area. This is confirmed by the SIA (Semiconductor Industry Association) roadmap, which forecasts a memory density of up to 90% *w.r.t.* the overall SoC area in the next few years [1].

Despite the efficiency of their design and manufacturing processes, SoCs may be affected by defects leading to yield loss. To increase SoC yield, memory cores usually embed fault tolerance techniques, based on hardware redundancy (spare rows and columns) [2]. With the technology reaching nano dimension even the logic cores, embedded in a SoC, become a challenge for the overall SoC yield level. Thus, to achieve a better yield, also logic cores have to be designed with some fault tolerant techniques in order to tolerate manufacturing defects.

Existing fault tolerant techniques are commonly used to tolerate on-line faults [3]. They use redundancies, *i.e.*, the property of having spare resources that perform a given function and tolerate defects. Fault tolerance techniques are generally classified depending on the type of redundancy. Basically, four types of redundancy are considered: software, information, temporal and hardware [4].

In software redundancy, error detection and recovery are based on replicating application processes on a single or multiple computers [5].

In information redundancy, additional data are used. For example, the use of error-correcting codes requires extra bits that need to be added to the original data bits [4]. Nevertheless, the use of error-correcting codes is widely used in memory context; its application to logic cores requires an important design effort associated to a high area overhead used to predict and compute the code computation.

Temporal redundancy consists in forcing the system to repeat a given operation and then compare the results with those of the previous operation [6]. Such a redundancy is able to tolerate transient or intermittent errors but not permanent errors. Since manufacturing defects lead to permanent errors, this solution is not suitable for the target of our study.

Hardware redundancy consists in modifying the design by adding additional hardware. For example, instead of having a single processor, three processors are embedded to perform the same operation. The failure of one processor is tolerated thanks to a voter that chooses the majority outputs [4].

In [7, 8, 9, 10], we have considered the well-known TMR fault tolerant architecture in order to tolerate manufacturing defects in logic cores. In this paper, we extend this study to the SoC context. We first determine the set of conditions to be satisfied in order to successfully resort to TMR to ramp-up the overall SoC yield. These conditions are related to the testability of the TMR version of logic cores. Therefore, these conditions are evaluated by using an ad-hoc ATPG procedure. The evaluation has been done on several SoC examples with different memory ratio. Based on these results, we first analyze the impact of the robustness of the voter. Then, we propose to improve the fault tolerance of TMR architectures by partitioning logic cores. Results obtained on SoC examples (composed of ISCAS'85 and ITC'99 benchmark circuits as logic cores and different memory ratio) demonstrate the interest of using TMR architectures for logic cores in the SoC context.

The remaining of the paper is organized as follows. In Section II, we detail the TMR approach. Section III shows the impact of using TMR architectures for logic cores on the overall SoC yield. Section IV details the test strategy targeting TMR architectures. Section V presents a set of experimental results. Section VI presents an analysis of voter robustness and an improvement of the TMR structure to make it able to tolerate more defects. Finally, concluding remarks are given in Section VII.

II. The TMR approach

In this section, we first present the TMR architecture and then we show how many defects it can tolerate.

A. *Basic principle*

Several hardware fault tolerant architectures have been proposed in the literature [11]. Generally, the degree of fault tolerance is defined as the maximum number of faults that can be tolerated in the system [12]. The classical hardware redundancy architecture is the NMR (N Modular Redundancy). A NMR structure is a fault tolerant architecture based on N modules performing the same function. The outputs of these modules are compared by using a majority voter. In general, this architecture can tolerate at least $(N - 1) / 2$ defects.

The case of $N = 3$ is called TMR and has been widely studied and used in practical applications [13, 4]. The inputs to three identical modules are tied together receiving thus the same data, and their outputs feed a majority Voter (V) circuit as shown in Figure 1.

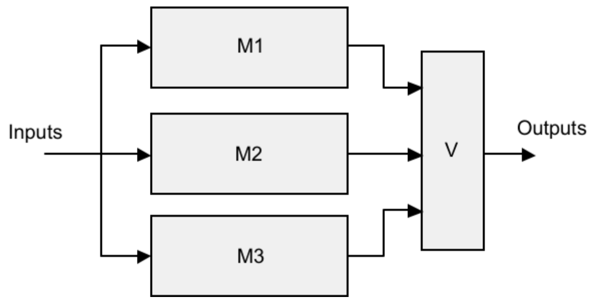

Figure 1. TMR principle

As a result, the TMR architecture significantly reduces the error probability at the primary outputs of the system. The erroneous value propagated by a defective module can be masked thanks to the presence of the two other fault-free modules. In the simplest structure, the voter is the weak point. If a fault appears in the voter, the TMR structure can be possibly faulty. To avoid this type of problem, the voter can be realized in software or with more robust design techniques [14].

B. *How many defects can be tolerated?*

Basically, the TMR architecture can tolerate one defect but, in practice, it can tolerate more than one defect. In fact, if there are two defects, the TMR may function properly depending on the nature and the location of the defects. Two defects are simply tolerated by the TMR if their induced errors cannot simultaneously drive the majority voter. On the other hand, two defects are not tolerated if there are located in two different modules and then propagate an error towards identical outputs on each module.

- Let P be the input pattern.
- Let Ω_i be the set of erroneous outputs in the module i due to the first defect when P is the input (i = 1, 2, 3).
- Let Ω_j be the set of erroneous outputs in the module j due to the second defect when P is the input (j = 1, 2, 3).

Under these constraints, the capability to tolerate two defects is formalized as follows:

- If $i = j$, defects are in the same module and, consequently, are tolerated.
- If $i \neq j$ and $\Omega_i \cap \Omega_j = \emptyset$, defects are tolerated.
- If $i \neq j$ and $\Omega_i \cap \Omega_j \neq \emptyset$, defects are not tolerated.

In Figure 2, two examples are shown with the same pattern P feeding the three modules. The voter has been omitted. Each defect is modeled as a stuck-at fault (f_1 and f_2 respectively). In the case of Figure 2.a, f_1 is propagated towards *O1* in the first module and f_2 is propagated towards *O2* in the second module. The majority voter receives two correct values and one wrong value. The outputs of the TMR are therefore correct and, consequently, faults f_1 and f_2 are tolerated.

In the case of Figure 2.b, f_1 is propagated towards *O1* and *O2* while f_2 is propagated towards *O2*. The voter receives one wrong value for *O1* and two wrong values for *O2*. So, the value on the second output of the TMR is faulty. Consequently, faults f_1 and f_2 are not tolerated.

As a result, two faults are tolerated when there is no pattern able to propagate errors, coming from the two faults in different modules, toward identical outputs in each module.

In the case of more than two defects, multiple defects can be handled by considering all possible fault couples between them. For example, three defects are supposed to behave as three couples of defects. If we assume now that all these fault couples have non-masking behaviors (which is very unlikely in practice), we can handle multiple defects by simply considering all fault couples independently. Moreover, three defects are not observable if the three associated fault couples are not observable individually. Such assumption is reasonable as it relies on the same principles (seldom masking phenomena) than the single stuck-at fault assumption with respect to multiple stuck-at faults.

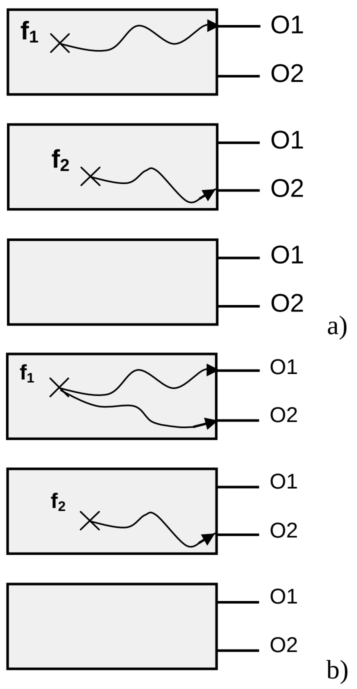

Figure 2. Two stuck-at faults are a) tolerated and b) not tolerated

III. Using TMR architectures in SoC context

In this section, we investigate the interest of using TMR architectures in order to tolerate manufacturing defects in logic cores and consequently improve the overall SoC yield. We therefore analyze the conditions to be fulfilled to achieve benefits in implementing such a TMR architecture instead of using a simple non-tolerant architecture.

A SoC composed of memory and logic cores has a yield expressed by:

$$Y_{SoC} = Y_L \times Y_M \tag{1}$$

where Y_M is the yield of memory cores and Y_L is the yield of logic cores. These two yield factors are computed by considering the contribution of the whole memory cores and the whole logic cores:

$$Y_M = \prod_{i=1}^{mc} Y_{Mi} \quad and \quad Y_L = \prod_{i=1}^{lc} Y_{Li} \tag{2}$$

where *mc* is the number of memory cores and *lc* is the number of logic cores. Assuming that memories embed fault tolerance technique, Y_M can hence be considered close to 100%. Consequently, to further increase the SoC yield, we may resort to TMR architectures for tolerating manufacturing defects in logic cores.

Let A_L be the area of logic cores in the original SoC, A_M be the area of memory cores and A_V the area of voters needed for the TMR implementation in the fault tolerant SoC (SoC_{TMR}). The resulting area overhead of the TMR architecture will be:

$$A_O = \frac{A_M + 3A_L + A_V}{A_M + A_L} = M + \frac{3A_L + A_V}{A_M + A_L} \tag{3}$$

with $M = A_M / (A_M + A_L)$ representing the memory occupancy ratio in the SoC. Thus, if we triplicate all logic cores to implement TMR architectures on a given silicon area (*e.g.*, a wafer) containing several SoCs, we can have n SoC_{TMR} having a yield equal to Y_{SoCTMR} ($Y_{SoCTMR} \times n$ SoC_{TMR} having a correct behavior). On the other hand, without TMR we can have $A_O \times n$ SoC having a yield equal to Y_{SoC} ($Y_{SoC} \times A_O \times n$ defect-free SoC). TMR architectures are therefore worthwhile only if:

$$\frac{Y_{SoCTMR}}{A_o} > Y_{SOC} \tag{4}$$

As $Y_{SoCTMR} \leq 1$, using TMR architectures for yield improvement is interesting only if:

$$Y_{SOC} \leq \frac{1}{A_O} \tag{5}$$

Let us now compute Y_{SoCTMR} and Y_{SoC} by using the Poisson distribution to model the defect distribution on the SoC. It would not be completely accurate to use the Poisson distribution for large circuits due to clustering effects on defects. More accurate calculation could be obtained by using, *e.g.*, the negative binomial distribution model [15, 16]. But, for a first and rough evaluation this is reasonable. In our case, the Poisson distribution is a discrete probability distribution that defines the probability that a number of manufacturing defects occurs in a fixed area if these defects occur with a known probability. Let X be the number of manufacturing defects. Let λ be the average number of expected defects for a given silicon area. Then, $\lambda = n \times p$ with n being the number of logic gates (or transistors) and p being the average defect rate of a gate (or a transistor). Let $P\{X = k\}$ be the probability that the circuit has k manufacturing defects. If n is high and p is low, the binomial distribution becomes the Poisson distribution:

$$P\{X = k\} = e^{\lambda} \times \frac{\lambda^k}{k!} \tag{6}$$

In the original SoC, the presence of a fault in logic cores makes the entire system faulty. So, Y_L is the probability that there is no defect inside logic cores:

$$Y_L = P\{X = 0\} = e^{\lambda_L} \tag{7}$$

Consequently, Y_{SoC} becomes:

$$Y_{SoC} = e^{\lambda_L} \times Y_M \tag{8}$$

The computation of Y_{SoCTMR} is more complex as it depends on *i)* the ability of the TMR to tolerate manufacturing defects and *ii)* the yield of voters. Consequently, Y_{SoCTMR} can be expressed as follows:

$$Y_{SoCTMR} = Y_T \times Y_V \times Y_M \tag{9}$$

where Y_T is the yield of the triplication without considering the voter and Y_V is the yield of voters. Let us first compute Y_T as the probability that no defects occur plus the probability that the TMR tolerates all the defects, Y_T is formally defined as follows:

$$Y_T = \underbrace{P\{X=0\}}_{\text{Probability that there is no defect}} + \underbrace{P\{X=1\}}_{\text{Probability that there is 1 defect}} + R \times \overbrace{P\{X=2\}}^{\text{Probability that there are 2 defects}} + R^3 \times \underbrace{P\{X=3\}}_{\text{Probability that there are 3 defects}} + \ldots$$

n defects are equivalent to C_n^2 couples of defects

$$Y_T = e^{\lambda_T} \times \left(1 + \lambda_T + R\,\tfrac{(\lambda_T)^2}{2!} + R^3\,\tfrac{(\lambda_T)^3}{3!} + \ldots\right) \tag{10}$$

with R being the probability that two defects are tolerated, *i.e.*, R reflects the tolerance capability of the TMR architecture.

There are three times more gates (or transistors) into a TMR architecture than into a non-redundant logic core. So, by substituting λ_T by $3\lambda_L$ we obtain:

$$Y_T = e^{-3\lambda_L} \times \left(1 + 3\lambda_L + \sum_{i=2}^{\infty} R^{C_i^2} \times \tfrac{(3\lambda_L)^i}{i!}\right) \tag{11}$$

Y_V is the probability that there is no defect inside voters:

$$Y_V = P\{X=0\} = e^{\lambda_V} \times \frac{(\lambda_V)^0}{0!} = e^{\lambda_V} \tag{12}$$

and, by substituting λ_V by $\lambda_L \times A_V / A_L$ we obtain:

$$Y_V = e^{\lambda_L \frac{A_V}{A_L}} \tag{13}$$

Consequently, by considering Eq. (3), (9), (11) and (13) and by substituting $\lambda_L = (1-M)\lambda_{SoC}$, Y_{SoCTMR} becomes:

$$Y_{SoCTMR} = e^{-(A_O - M)\lambda_{SoC}} \times \left(1 + 3(1-M)\lambda_{SoC} + \sum_{i=2}^{\infty} R^{C_i^2} \times \tfrac{[3(1-M)\lambda_{SoC}]^i}{i!}\right) \times Y_M \tag{14}$$

and with $Y_{SoC} = e^{\lambda_{SoC}} \Rightarrow \lambda_{SoC} = -\ln Y_{SoC}$:

$$Y_{SoCTMR} = e^{(A_O - M)\ln Y_{SoC}} \times \left(1 - 3(1-M)\ln Y_{SoC} + \sum_{i=2}^{\infty} R^{C_i^2} \times \tfrac{[-3(1-M)\ln Y_{SoC}]^i}{i!}\right) \times Y_M \tag{15}$$

To summarize, implementing TMR architectures for logic cores can improve the overall SoC yield if two conditions are satisfied. First, $Y_{SoC} \leq 1/A_O$; this condition is related to the area overhead needed to implement TMR architectures for logic cores. Secondly, $Y_{SoCTMR} \geq A_O \times Y_{SoC}$ with Y_{SoCTMR} depending on Y_{SoC}, A_O, M and R as shown in Eq. 15. To satisfy these conditions, we have to analyze the impact of A_O, M and R on Y_{SoCTMR}. A_O and M are easily computed but R requires determining the percentage of tolerated couple of defect. In fact, the problem is how can we determine the percentage of tolerated couple of defects for a given TMR architecture of logic cores. As shown in the example of Figure 2, tolerated defects lead to fault-free values at the outputs of the TMR architecture thus corresponding to the untestable defects. Consequently the problem of determining the percentage of couple of tolerated defects is equivalent to determine the untestable ones. In the next section we investigate the test issues related to TMR architectures in order to determine this percentage.

IV. TEST OF TMR ARCHITECTURES

In this section we detail specificities of the test of TMR architectures. Especially, we present how the fault list and the ATPG procedure have been created.

A. *Test specificities*

Testing tolerant architectures should be addressed in a different way compared to the test of standard circuits. Hereafter, we present the peculiarities of TMR testing. For the sake of simplicity, we consider that all modules are structurally identical. Nevertheless, further considerations are still valid also in case of structurally different redundant modules. The first peculiarity of a TMR architecture induced by its redundant nature makes the test approach very different from the test of a classical structure without redundancy. In this type of structure, a fault affecting only one module is masked thank to the two other fault-free modules.

Testing a TMR architecture depends on its final use. In a classical way of use, the goal is to tolerate potential on-line defects (permanent and/or transient). Each module should be fault-free after manufacturing. Due to the intrinsic impossibility to test single stuck-at fault, the architecture has to be modified during test [3] as shown in Figure 3.a. In this modified architecture, redundancy is removed and each single stuck-at fault becomes testable. Each module is individually tested.

In the proposed way of use, the goal is to increase the yield by tolerating permanent defects due to an imperfect manufacturing process. In this case, the test consists in testing globally the TMR structure in order to determine what are the couples of defects which are tolerated or not (determination of the R value). In this case, the architecture is not modified for test purpose like the one presented in Figure 3.b. Finally, we remind that the TMR basically tolerates one single defect and optionally two or more defects as previously discussed. For example, if several defects are located in the same module, the TMR still works properly.

To summarize, in the first approach, the question to be answered during the test of the TMR is: "*Are there one or more than one manufacturing defects in each module of the TMR architecture?*".

In the second approach, the question becomes: "*Does the logic core pass the test despite the presence of one or more than one manufacturing defects?*".

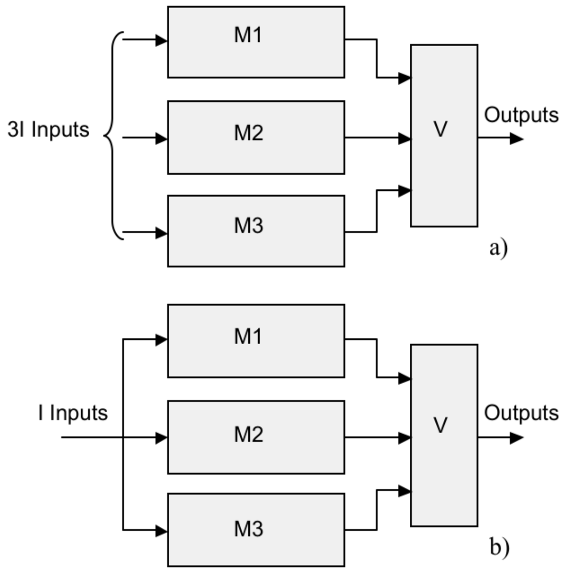

Figure 3. Test of a) an on-line fault tolerant structure and b) a manufacturing fault tolerant structure

B. Fault Model

The fault model widely used in structural testing to detect a manufacturing defect is the single stuck-at fault model. An ATPG based on single stuck-at fault model generates patterns able to detect all single stuck-at faults. Such patterns are not efficient in the context of multiple stuck-at faults occurring in a redundant TMR architecture. Since the single stuck-at fault is by definition tolerated (untestable) in the TMR architecture, a different fault model has to be considered. This fault model must be testable on primary outputs of the TMR, and also be representative of actual manufacturing defects. In Figure 2 we have seen that a couple of stuck-at faults can be observed (not tolerated) on primary outputs of the TMR and also can be testable.

In the following, we refer to a couple of stuck-at fault as fault pair [17]. If there are two stuck-at faults in the structure, these two faults are a fault pair. If there are more than two stuck-at faults, these multiple stuck-at faults can be gathered in fault pairs (under the assumption that no masking effect occurs between faults). For example, three stuck-at faults f_1, f_2, and f_3 are equivalent to three fault pairs $\{f_1, f_2\}$, $\{f_1, f_3\}$ and $\{f_2, f_3\}$. In general, n stuck-at faults are equivalent to C_n^2 fault pairs.

To be testable and thus not tolerated, the two faults composing the fault pair must affect two different modules and propagate toward common module's outputs as shown in Figure 2.

C. Size of the fault list

Let us consider that each module has n single stuck-at faults. The whole number of stuck-at faults in the three modules is $3n$. As $\{f_1, f_2\} = \{f_2, f_1\}$, the total number ψ of fault pairs is:

$$\Psi = C_{3n}^2 = \frac{3n!}{(3n-2)!\times 2!} = \frac{9n^2 - 3n}{2} \qquad (16)$$

We have now to compute the percentage of untestable (tolerated) fault pairs to determine if the TMR architecture is worthwhile for yield improvement. So, we simply generate test patterns able to detect all testable fault pairs. The remaining fault pairs are untestable. Since ψ is quadratic to n, the use of a classical ATPG to detect the entire set of fault pairs will be unfeasible due to a huge CPU time. In the next sub-section, we show how to handle this particular point.

D. ATPG procedure

In order to determine the convenience of using the TMR architecture to handle manufacturing defects, we have first to determine the untestable fault pairs. An ATPG targeting the fault pair model has to be run. Since the goal of this work was not to develop an ATPG for fault pairs, we have adapted an ATPG tool targeting single stuck-at faults to target fault pairs.

Considering that a fault pair is composed of two stuck-at faults (f_1, f_2), the proposed approach is to inject a permanent fault (f_1) in one module of the TMR architecture by modifying the *netlist*. Then, an ATPG is run to test all stuck-at faults in presence of the permanent fault f_1. This process is repeated until we have injected all stuck-at faults in one module.

The simplicity of this approach is obtained at the cost of high simulation time since n (number of single stuck-at faults in one module) full ATPG runs are needed. To reduce this drawback, we have shortened the fault pair list to be handled as follows:

- When there is only one faulty module, the outputs of this module are masked by the voter. Thus, all fault pairs, which impact only one module, are structurally untestable. These fault pairs can be removed from the ATPG fault list.
- Symmetries of the TMR architectures are used to reduce the fault pair list of the ATPG. As the module inputs are tied together during test, fault pairs are equivalent when their two stuck-at faults have the same location in two different modules. Therefore, equivalent fault pairs are removed from the ATPG fault list.

With the help of these simplifications, the size of the fault pair list becomes:

$$Nb\_Faute = \frac{n^2 + n}{2} \tag{17}$$

The list of fault pairs can be further reduced by determining and removing fault pairs that are structurally untestable. Let us consider the fault pair $\{f_1, f_2\}$ and their output cones (list of outputs where the fault effect may be propagated) $\{\phi_1, \phi_2\}$. Due to the presence of the voter, if $\phi_1 \cap \phi_2 = \emptyset$, the fault pair $\{f_1, f_2\}$ is structurally untestable and therefore, can be removed from the fault pair list. In practical cases, the number of structurally untestable fault pairs is quite large leading to a huge improvement of the overall ATPG performance.

To summarize, Figure 4 presents the ATPG algorithm with the fault pair model.

V. TMR INTEREST FOR SOC YIELD IMPROVEMENT

In the previous section we state the constraints to be fulfilled in order to use TMR to increase SoC yield. In this section we evaluate those constraints on a set of benchmark SoCs.

```
create the fault pair list;

for (each fault pair {f1, f2}) {
        Compute output-cones {ϕ1, ϕ2} of f1 and f2;
        if (ϕ1 ∩ ϕ2 ≠ Ø) add fault pairs to the fault list ζ;
}

for (each single stuck-at fault f of module 1) {
        inject the permanent fault f in the VERILOG description;
        add all fault pair Є ζ with (f = f1 or f = f2) to the ATPG fault list;
        run ATPG;
}

create report;
```

Figure 4. ATPG algorithm

For the analysis, both ISCAS'85 and ITC'99 (combinational part only) benchmark circuits are used as logic cores in a SoC. These benchmark circuits are simply cloned three times and majority voters have been added to implement a TMR architecture. Results of the ATPG runs are reported in Table 1 for different values of *M* (Memory occupancy ratio).

The first column lists the benchmark circuit name used as logic core. Second and third columns show the number of stuck-at faults in one logic core (# SAF) and the number of fault pairs in the ATPG fault list (# ATPG FP) using reduction techniques proposed in [8]. The next columns give for three *M* scenarios (50%, 70% and 90%):

- $1/A_O$: Y_{SoC} limit under which it is interesting to implement a TMR as presented in Eq. 5.
- *Rmin*: minimum value of *R* (percentage of untestable fault pairs) above which it is beneficial to use a TMR architecture.
- *R*: tolerance of the TMR architecture. It corresponds to the percentage of untestable fault pairs provided by the ATPG procedure.

TABLE I. INTEREST OF USING TMR ARCHITECTURES FOR SOC YIELD IMPROVEMENT

Logic core	**ATPG data**		**M = 50%**			**M = 70%**			**M = 90%**		
	# SAF	*# ATPG FP*	*1/A_O (%)*	*Rmin (%)*	*R (%)*	*1/A_O (%)*	*Rmin (%)*	*R (%)*	*1/A_O (%)*	*Rmin (%)*	*R (%)*
c1908	1812	1.30M	49.02	90.68	56.42	61.58	86.36	56.42	82.78	65.75	56.42
c2670	2852	1.87M	46.19	95.88	75.95	58.86	94.00	75.95	81.10	84.92	75.95
c3540	3438	4.91M	49.50	89.54	54.09	62.03	84.67	54.09	83.06	61.59	54.09
c5315	4970	3.44M	48.08	92.69	93.20	60.68	89.31	93.20	82.24	73.10	93.20
c6288	6250	18.2M	49.63	89.24	38.02	62.15	84.23	38.02	83.13	60.49	38.02
c7552	7438	7.40M	48.90	90.95	84.92	61.46	86.76	84.92	82.71	66.74	84.92
b04	1477	535k	46.30	95.73	84.32	58.96	93.78	84.32	81.17	84.35	84.32
b05	2553	1.17M	48.08	92.69	88.65	60.68	89.31	88.65	82.24	73.10	88.65
b07	1120	399k	45.66	96.59	81.91	58.34	95.05	81.91	80.78	87.56	81.91
b11	1308	703k	47.73	93.35	74.50	60.35	90.28	74.50	82.03	75.54	74.50
b12	2777	857k	46.51	95.41	95.45	59.17	93.31	95.45	81.30	83.17	95.45
b13	835	59.6k	44.54	97.86	96.94	57.24	96.90	96.94	80.06	92.23	96.94

On the base of these APTG results, we are now able to determine if implementing TMR architectures for logic cores will improve the overall SoC yield. This is simply done by comparing columns *Rmin* and *R* in Table 1 for each *M* scenario. It appears (concerned cases are highlighted by shaded boxes) that using TMR for logic cores in SoC is suitable for 2 logic core examples when *M* = 50%, 3 logic core examples when *M* = 70% and 5 logic core examples when *M* = 90% among the 12 logic core examples. Therefore, TMR architectures are promising solutions for SoC yield improvement. In the next, section we propose to improve the fault tolerance of the TMR to achieve higher yield benefit.

VI. TMR IMPROVEMENT AND DISCUSSION

As shown in the previous section, TMR architecture seems to be a promising fault tolerant structure for logic core to improve the overall SoC yield. Hereafter, we discuss possible improvements of the basic TMR architecture by first analyzing the impact of the yield of the voters and secondly, by partitioning logic cores to increase the tolerance of the TMR architecture. Then, we analyze the SoC yield improvement. Finally, we extend the study to the general SoC context where several logic cores are embedded in a SoC.

A. *Voter yield impact*

For all equations and results presented before, the voters added to implement TMR architectures for logic cores have been considered to be possibly defective with the same defect density than the overall SoC. In this sub-section, we provide results by also considering that voters can be fault-free as they are manufactured with robust design techniques.

Results are reported in Table 2. The first column lists the benchmark circuit names used as logic cores in a SoC with three *M* scenarios (50, 70 and 90%). Then, for each *M* scenario, Table 2 first recalls $1/A_O$, values *Rmin* and *R* of Table 1 and then gives values *Rmin* and *R* obtained when voters are defect-free ($Y_v = 1$). By comparing again columns *Rmin* and *R* in Table 2, using TMR architectures for logic cores is suitable for SoC yield improvement if voters are designed in a robust way (Y_v close to 100%).

B. *Fault tolerance improvement*

To improve the tolerance of TMR architectures, we propose to increase the number of untestable fault-pairs by reducing the combinational depth of logic cores. This is done by partitioning each logic core into two or three equivalent blocks so as to increase the tolerance of the TMR architecture. As shown in Figure 5, majority voters are added on circuit's edge cut.

An important feature of partitioning the circuit is that the tolerance of fault pairs increases when the number of partition increases as well. For example, in the case of double TMR (Figure 5.b), cores (modules) are divided into two equivalent blocks. Each block operates independently and a manufacturing defect in the first block has no impact on the second block.

In a basic TMR architecture, the percentage of untestable fault pairs is always greater than 33.33% as two stuck-at faults in the same module are untestable. In a double TMR architecture, if we consider that the first fault f_1 impacts *M1'*, then the fault pair $\{f_1, f_2\}$ can be detected (not tolerated) if and only if f_2 is in *M2'* or *M3'*. Conversely, if f_2 is in *M1'*, *M1''*, *M2''* or *M3''*, the fault pair is necessarily tolerated due to the presence of the voters. Therefore, the percentage of tolerated fault pairs in a double TMR architecture is always greater than 66.66%.

TABLE II. IMPACT OF THE VOTER YIELD ON THE SOC YIELD IMPROVEMENT

Logic core	M = 50%					M = 70%					M = 90%				
	$1/A_O$ (%)	Rmin (%)	R (%)	$Y_v = 1$		$1/A_O$ (%)	Rmin (%)	R (%)	$Y_v = 1$		$1/A_O$ (%)	Rmin (%)	R (%)	$Y_v = 1$	
				Rmin (%)	R (%)				Rmin (%)	R (%)				Rmin (%)	R (%)
c1908	49.02	90.68	56.42	88.61	56.42	61.58	86.36	56.42	83.35	56.42	82.78	65.75	56.42	58.52	56.42
c2670	46.19	95.88	75.95	89.48	75.95	58.86	94.00	75.95	84.75	75.95	81.10	84.92	75.95	62.40	75.95
c3540	49.50	89.54	54.09	88.46	54.09	62.03	84.67	54.09	83.10	54.09	83.06	61.59	54.09	57.82	54.09
c5315	48.08	92.69	93.20	88.91	93.20	60.68	89.31	93.20	83.83	93.20	82.24	73.10	93.20	59.85	93.20
c6288	49.63	89.24	38.02	88.42	38.02	62.15	84.23	38.02	83.04	38.02	83.13	60.49	38.02	57.64	38.02
c7552	48.90	90.95	84.92	88.65	84.92	61.46	86.76	84.92	83.42	84.92	82.71	66.74	84.92	58.69	84.92
b04	46.30	95.73	84.32	89.45	84.32	58.96	93.78	84.32	84.70	84.32	81.17	84.35	84.32	62.26	84.32
b05	48.08	92.69	88.65	88.91	88.65	60.68	89.31	88.65	83.83	88.65	82.24	73.10	88.65	59.84	88.65
b07	45.66	96.59	81.91	89.63	81.91	58.34	95.05	81.91	84.99	81.91	80.78	87.56	81.91	63.08	81.91
b11	47.73	93.35	74.50	89.02	74.50	60.35	90.28	74.50	84.01	74.50	82.03	75.54	74.50	60.33	74.50
b12	46.51	95.41	95.45	89.39	95.45	59.17	93.31	95.45	84.60	95.45	81.30	83.17	95.45	61.97	95.45
b13	44.54	97.86	96.94	89.95	96.94	57.24	96.90	96.94	85.50	96.94	80.06	92.23	96.94	64.46	96.94

In the case of a triple TMR architecture (see Figure 5.c), circuits (modules) are divided into three equivalent blocks. If f_1 impacts *M1'*, the fault pair $\{f_1, f_2\}$ is necessarily tolerated if f_2 impacts *M1'*, *M1''*, *M2''*, *M3''*, *M1'''*, *M2'''* and *M3'''*. Consequently, the percentage of tolerated fault pairs is always higher than 77.77%.

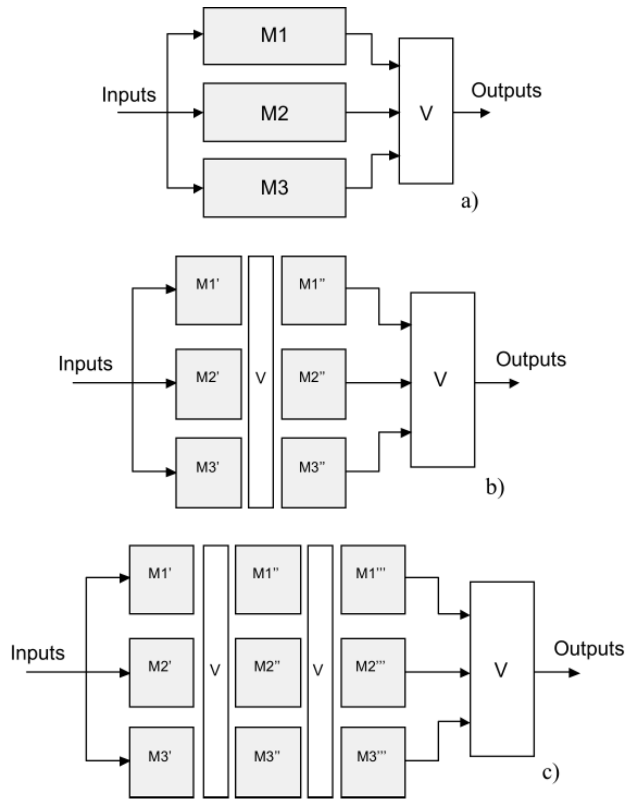

Figure 5. Improvement of the TMR architecture
a) basic, b) double and c) triple TMR architectures

More formally, if circuits are partitioned into k blocks, then the percentage of tolerated fault pairs is always higher than:

$$Tol\_FP = 100 \times \frac{3k-2}{3k} \tag{18}$$

With this technique, the tolerance of the TMR architecture increases with the number of partitions. The main drawback of this technique is the requirement of additional voters. In the next sub-section, we show that the overhead of silicon area due to additional voters still remains very low (less that 3% for large benchmark circuits).

The circuit partitioning consists in three steps. First, we transform the circuit into a hypergraph. A hypergraph is a generalization of a graph, where the set of edges is replaced by a set of hyperedges. A hyperedge extends the notion of an edge by allowing more than two vertices to be connected together. Formally, a hypergraph $H = (V, Eh)$ is defined as a set of vertices V and a set of hyperedges Eh, where each hyperedge is a subset of the vertices set V, and the size of an hyperedge is the cardinality of this subset. Hypergraphs can be used to naturally represent a gate-level VLSI circuit. The vertices of the hypergraph can be used to represent the gates of the circuit, and the hyperedges can be used to represent the lines connecting these gates.

In a second step, we make the hypergraph partitioning. The proposed technique of circuit partitioning for TMR architecture improvement utilizes sh-METIS, a multilevel hypergraph partitioning algorithm based upon the multilevel paradigm [18]. The quality of the partitioning produced by sh-METIS in terms of cut size (the size of the hyperedge cut is representative of the area overhead of the proposed solution), the computational time needed to partition large combinational circuits, and the availability of sh-METIS in a free access on the Web site of the University of Minnesota have motivated our choice for this academic tool. In a performance comparison of their multilevel partitioning algorithm with other state-of-the-art partitioning schemes, the authors of sh-METIS reported that their algorithm produces partitioning that are on the average 6% to 23% better (in terms of cut size) than existing algorithms, and often requires 4 to 10 times less time than that required by other schemes.

The final step consists in adding voters in the place of hyper-cuts. The number of added voters is equal to number of hyper-cuts that is optimal using sh-METIS software.

The ATPG procedure presented in Section IV.D has been run on *Double TMR* architecture, *i.e.,* logic cores have been partitioned into two modules (having the same size). Experimental results are reported in Table 3. The first column of the table lists the benchmark circuit names used as logic cores in a SoC with three M scenarios (50, 70 and 90%). Then, for each M scenario, Table 3 (see the last page of the paper) provides results for both *Basic TMR* and *Double TMR*. As previously, each sub-column gives values $1/A_O$, *Rmin* and *R*.

A first comment regarding these results is that the partitioning of logic cores increases the probability R. For example, when M = 90%, 11 logic core examples make the use of TMR architectures suitable for SoC yield improvement. Of course, if the number of partitions increases, the tolerance of the TMR increases too. This is shown in Table 4 that provides results using *Basic*, *Double* and *Triple* TMR for a unique M scenario (70%).

A second comment is that we can analyze the impact of the logic core partitioning on the area overhead (A_O). From data in Table 3 (column $1/A_O$) we have computed the area overhead needed to implement a *Double TMR* compared to a *Basic TMR* and done it for the different memory ratio. From these results, we notice that the area overhead is low for large benchmark circuits: less than 4.53% when M = 50%, less than 3.45% when M = 70% and less than 1.55% when M = 90%.

TABLE III. INTEREST OF USING LOGIC CORE PARTITIONING FOR SOC YIELD IMPROVEMENT

Logic core	**M = 50%**		**M = 70%**		**M = 90%**	
	Basic TMR	***Double TMR***	***Basic TMR***	***Double TMR***	***Basic TMR***	***Double TMR***

	$1/A_O$ (%)	Rmin (%)	R (%)	$1/A_O$ (%)	Rmin (%)	R (%)	$1/A_O$ (%)	Rmin (%)	R (%)	$1/A_O$ (%)	Rmin (%)	R (%)	$1/A_O$ (%)	Rmin (%)	R (%)	$1/A_O$ (%)	Rmin (%)	R (%)
c1908	49.02	90.68	56.42	47.85	93.13	89.49	61.58	86.36	56.42	60.46	89.96	89.49	82.78	65.75	56.42	82.10	74.74	89.49
c2670	46.19	95.88	75.95	45.77	96.46	92.46	58.86	94.00	75.95	58.45	94.85	92.46	81.10	84.92	75.95	80.84	87.06	92.46
c3540	49.50	89.54	54.09	48.78	91.22	88.36	62.03	84.67	54.09	61.35	87.15	88.36	83.06	61.59	54.09	82.64	67.70	88.36
c5315	48.08	92.69	93.20	47.73	93.35	94.30	60.68	89.31	93.20	60.35	90.28	94.30	82.24	73.10	93.20	82.03	75.54	94.30
c6288	49.63	89.24	38.02	49.38	89.84	76.35	62.15	84.23	38.02	61.92	85.11	76.35	83.13	60.49	38.02	82.99	62.66	76.35
c7552	48.90	90.95	84.92	48.66	91.48	94.40	61.46	86.76	84.92	61.24	87.53	94.40	82.71	66.74	84.92	82.58	68.65	94.40
b04	46.30	95.73	84.32	45.25	97.10	93.19	58.96	93.78	84.32	57.94	95.79	93.19	81.17	84.35	84.32	80.52	89.43	93.19
b05	48.08	92.69	88.65	47.73	93.35	89.09	60.68	89.31	88.65	60.35	90.28	89.09	82.24	73.10	88.65	82.03	75.54	89.09
b07	45.66	96.59	81.91	44.05	98.32	91.23	58.34	95.05	81.91	56.75	97.56	91.23	80.78	87.56	81.91	79.74	93.90	91.23
b11	47.73	93.35	74.50	45.66	96.59	87.75	60.35	90.28	74.50	58.34	95.05	87.75	82.03	75.54	74.50	80.78	87.56	87.75
b12	46.51	95.41	95.45	46.19	95.88	96.27	59.17	93.31	95.45	58.86	94.00	96.27	81.30	83.17	95.45	81.10	84.92	96.27
b13	44.54	97.86	96.94	44.35	98.05	97.36	57.24	96.90	96.94	57.05	97.18	97.36	80.06	92.23	96.94	79.94	92.93	97.36

TABLE IV. BASIC, DOUBLE AND TRIPLE TMR COMPARISONS

Logic core	M = 70%					
	Basic TMR		*Double TMR*		*Triple TMR*	
	$1/A_O$ (%)	R (%)	$1/A_O$ (%)	R (%)	$1/A_O$ (%)	R (%)
c1908	61.58	56.42	60.46	89.49	59.92	96.50
c2670	58.86	75.95	58.45	92.46	57.94	93.90
c3540	62.03	54.09	61.35	88.36	60.28	93.78
c5315	60.68	93.20	60.35	94.30	60.24	96.38
c6288	62.15	38.02	61.92	76.35	61.69	84.14
c7552	61.46	84.92	61.24	94.40	61.01	96.22
b04	58.96	84.32	57.94	93.19	57.44	95.53
b05	60.68	88.65	60.35	89.09	59.81	93.30
b07	58.34	81.91	56.75	91.23	56.47	94.11
b11	60.35	74.50	58.34	87.75	57.44	93.91
b12	59.17	95.45	58.86	96.27	58.24	97.79
b13	57.24	96.94	57.05	97.36	56.75	97.83

C. SoC yield improvement

To illustrate and prove the interest of using TMR architectures, we have considered, as case study, two benchmark circuits (c5315 with M = 70% and b05 with M = 90%) as the logic cores in a SoC. We have computed the yield improvement (Y_I) reachable if a fault-tolerant SoC (using TMR for logic cores) is manufactured instead of a classical SoC (only memory cores are fault-tolerant):

$$Y_I = 100 \times \frac{Y_{SoCTMR} - Y_{SoC} \times A_O}{Y_{SoC} \times A_O} \qquad (17)$$

The graphs in Figures 6 and 7 show the yield improvement for the two SoC examples. In each graph, *Basic* and *Triple TMR* implementations are considered. Moreover, voters are considered to be either possibly faulty or fault-free.

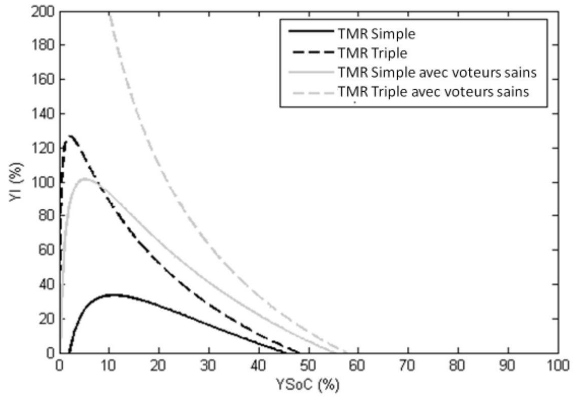

Figure 6. Yield improvement for the c5315 with M = 70%

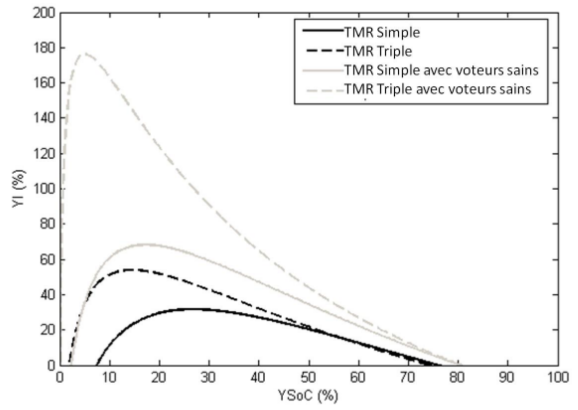

Figure 7. Yield improvement for the b05 with M = 90%

For example, let us consider the b05 benchmark circuit used as unique logic core. From Figure 7, if the initial yield Y_{SoC} = 30%, the yield improvement (Y_I) is about 31.28% for a Basic TMR implementation (59.17% when voters are fault-free) and about 42.69% for a Triple TMR implementation (90.49% when voters are fault-free). These results clearly show the interest of using TMR architectures for SoC yield improvement.

D. General SoC context

Until now, we have considered simple SoC examples composed by only one logic core. In the general SoC context, several logic cores are embedded. Consequently, the probability R (R_i) has to be computed for each logic core i translated into a TMR architecture. Then, we have to compute the value R (R_{eq}) representing the fault tolerance of the whole logic part by applying the following equation:

$$R_{eq} = 1 - \frac{\sum_{i=1}^{lc}\left[(1-R_i)\times C^2_{3\times n_i}\right]}{C^2_{\sum_{j=1}^{lc} n_j}} \qquad (19)$$

where lc is the number of logic cores and n_i is the number of single stuck-at faults in the logic core i. We also have to compute the $Rmin$ value ($Rmin_{eq}$), which represents the minimum value of R_{eq} above which it is beneficial to use TMR architectures, by using Eq. 15.

For example, let us assume a SoC with M = 70% and two logic cores C_1 (c7552) and C_2 (b05). From data reported in Table 1, we extract that R_1 = 84.92% and R_2 = 88.65%. If considered as independent, these two cores have no interest to successfully resort to TMR, *i.e.*, $R < Rmin$ in Table 1. Now, if these cores are embedded in the same SoC, R_{eq} = 90.90% and $Rmin_{eq}$ = 87.45%. So, as $R_{eq} > Rmin_{eq}$ the TMR versions of C_1 and C_2 are suitable for SoC yield improvement.

It is also important to notice that C_1 and C_2 can be viewed as two partitions of a logic core C. Each partition is not enough fault tolerant ($R_i < Rmin_i$) but C tolerates enough fault pairs as $R_{eq} > Rmin_{eq}$. Consequently, this example demonstrates the interest of using logic core partitioning to improve the fault tolerance of the TMR.

VII. Conclusion

In this paper analyzes the use of TMR architectures for logic cores to improve the overall SoC yield. We have computed the necessary conditions that make TMR architectures more attractive compared to non-tolerant logic cores. These conditions are related to *i)* the initial SoC yield (Y_{SoC}), *ii)* the memory ratio (M), *iii)* the area overhead needed to implement the TMR (A_O) and *iv)* the tolerance of the TMR implementation (R). A dedicated ATPG targeting TMR architectures has been used to evaluate the R parameter for different values of A_O and M. We have also analyzed the impact of the voter yield and the interest of partitioning logic cores to increase the fault tolerance of the TMR architecture. Results have shown that using TMR architectures for logic cores can be very fruitful to improve the overall SoC yield.

References

[1] Semiconductor Industry Association (SIA), "International Technology Roadmap for Semiconductors (ITRS)", http://itrs.net, 2007.

[2] S. Shoukourian et Al., "SoC Yield Optimization via an Embedded-Memory Test and Repair Infrastructure", Proc of IEEE Design & Test of Computer, pp. 200-207, 2004.

[3] C. E. Stroud and A. E. Barbour, "Design for Testability and Test Generation for Static Redundancy System Level Fault Tolerant Circuits", Proc. of IEEE International Test Conference, pp. 812-818, 1989.

[4] I. Koren and C. Krishna, "Fault Tolerant Systems", Morgan Kauffman Publisher, 2007.

[5] T. Tsai, "Fault tolerance via N-modular software redundancy", Proc. of IEEE International Symposium on Fault Tolerant Computing, pp. 201 1998.

[6] S. Laha and J. H. Patel, "Error correction in arithmetic operations using time redundancy", Proc. of IEEE Fault Tolerant Computing Symposium, pp. 298-305, 1983.

[7] J. Vial, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch and A. Virazel, "Yield Improvement, Fault-Tolerance to the Rescue?", Proc. of IEEE International On-Line Testing Symposium, pp. 165-166, 2008.

[8] J. Vial, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch and A. Virazel, "Using TMR Architectures for Yield Improvement", Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 7-15, 2008.

[9] J. Vial, A. Virazel, A. Bosio, P. Girard, C. Landrault and S. Pravossoudovitch, "Using TMR Architectures for SoC yield Improvement", International Conference on Advances in System Testing and Validation Lifecycle, pp 155-160, 2009.

[10] J. Vial, A. Virazel, A. Bosio, P. Girard, C. Landrault and S. Pravossoudovitch, "Is TMR Suitable for Yield Improvement?", IET Computers and Digital Techniques, Vol. 3, No 6, November 2009, pp. 581-592.

[11] P. K. Lala, "Self-Checking and Fault-Tolerant Digital Design", Morgan Kauffman Publisher, 2000.

[12] P. K. Lala, "Fault Tolerant and Fault Testable Hardware Design", Prentice-Hall International, 1985.

[13] R. E. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability", IBM Journal of Research and Development, Vol. 6, No 2, April 1962, pp.200-209.

[14] J. M. Cazeaux, D. Rossi and C. Metra, "New High Speed CMOS Self-Checking Voter", Proc. of IEEE International On-Line Testing Symposium, pp. 58-63, 2004.

[15] I. Koren, Z. Koren and C. H. Stapper, "A Unified Negative-Binomial Distribution for Yield Analysis of Defect-Tolerant Circuits", IEEE Transaction of Computers, Vol. 42, No 6, 1993, pp. 724-734.

[16] P. de Gyvez, "Integrated Circuit Manufacturability", John Wiley & Sons Inc, 1999.

[17] L. Fang and M. S. Hsiao, "Bilateral Testing of Nano-scale Fault Tolerant Circuits", Proc. of IEEE Defect and Fault Tolerance in VLSI Systems, pp. 309-317, 2006.

[18] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain", Technical Report, Department of Computer Science, University of Minnesota, 1998.