A Hybrid Fault Tolerant Architecture for Robustness Improvement of Digital Circuits

D. A. Tran A. Virazel A. Bosio L. Dilillo
P. Girard S. Pravossoudovitch
LIRMM – University of Montpellier / CNRS
Montpellier, France
{tran, virazel, bosio, dilillo, girard, pravo}@lirmm.fr

H.-J. Wunderlich
Institut für Technische Informatik
Stuttgart, Germany
wu@informatik.uni-stuttgart.de

***Abstract*—In this paper, a novel hybrid fault tolerant architecture for digital circuits is proposed in order to enable the use of future CMOS technology nodes. This architecture targets robustness, power consumption and yield at the same time, at area costs comparable to standard fault tolerance schemes. The architecture increases circuit robustness by tolerating both transient and permanent online faults. It consumes less power than the classical Triple Modular Redundancy (TMR) approach while utilizing comparable silicon area. It overcomes many permanent faults occurring throughout manufacturing while still tolerating soft errors introduced by particle strikes. These can be done by using scalable redundancy resources, while keeping the hardened combinational logic circuits intact. The technique combines different types of redundancy: information redundancy for error detection, temporal redundancy for soft error correction and hardware redundancy for hard error tolerance. Results on largest ISCAS and ITC benchmark circuits show that our approach has an area cost negligible of about 2% to 3% with a power consumption saving of about 30% compared to TMR. Finally, it deals with aging phenomenon and thus, increases the expected lifetime of logic circuits.**

Keywords-transient error, permanent error, robustness, fault tolerance, TMR, power consumption, aging phenomenon.

I. INTRODUCTION

CMOS technology scaling allows the realization of more and more complex systems, reduces production costs and optimizes performances and power consumption. Today, each CMOS technology node is facing reliability problems [1] whilst there is currently no alternative technology as effective as CMOS in terms of cost and efficiency. Therefore, it becomes essential to develop methods that can guarantee a high robustness for future CMOS technology nodes.

A high integration density affects the robustness of a circuit during its functioning as well as during its manufacturing. Smaller size of transistors makes the circuit more vulnerable to soft errors where devices are not permanently damaged. Moreover, a high integration density causes a high defect density, which results in hard errors and a lower manufacturing yield.

To increase the robustness of future CMOS circuits and systems, fault tolerant architectures might be one solution. In fact, these architectures are commonly used to tolerate on-line faults, *i.e.* faults that appear during the normal functioning of the system, irrespective of their transient or permanent nature [2]. Moreover, it has been shown in [3, 4, 5] that they could also tolerate permanent defects and thus help improving the manufacturing yield.

Various solutions using fault tolerant techniques for robustness improvement have been studied, of which they mainly target the tolerance of transient and/or permanent faults. While increasing fault tolerance capability, keeping a low area overhead is also a main optimization criterion of these works. In [3, 4, 5], the authors proposed manufacturing yield improvement as a new goal. Beside, other aspects such as power consumption, aging and expected lifetime of circuits are of the same importance. Although being the subjects of research in fault tolerant communication, *e.g.* in network on chip [6], these aspects have not been studied for random logic cores. Here for the first time, our study provides a fault tolerant architecture that targets different goals: increasing circuit robustness, keeping a low area overhead, saving power consumption and extending the expected lifetime of logic circuits. This paper will focus on the derivation of a scalable fault tolerant method, which achieves significant power saving with an area overhead comparable to existing solution. The aging phenomenon discussed in the last section will be further studied in the upcoming paper.

Our hybrid fault tolerant architecture uses three types of redundancy: information redundancy for error detection, temporal redundancy for transient error tolerance and hardware redundancy for permanent error correction. Similar to TMR, this architecture consists of implementing three times the combinational logic part of the circuit. However, only two of them are running in parallel during functional mode of operation. A Finite State Machine (FSM) performs the selection of the running logic parts. It changes the architecture configuration with respect to the error detection made by a comparator. This architecture is compared to the classical TMR structure in terms of area overhead and power consumption. Results on ISCAS'85, ISCAS'89 and ITC'99 benchmark circuits show that our approach has negligible area cost of about 2% to 3% while TMR consumes 30% more power.

The remaining parts of this paper are organized as follows: Section II briefly reviews different fault tolerant techniques. Section III provides the principle as well as the functioning of the hybrid fault tolerant architecture. Comparisons with the TMR approach in terms of area and power consumption are discussed in Section IV. Section V analyzes impacts of our architecture on aging phenomenon. Finally, Section VI concludes the paper and provides some perspectives.

II. FAULT TOLERANT TECHNIQUES

Existing fault tolerant techniques are commonly used to tolerate on-line faults [7]. They use redundancy, *i.e.*, the property of having spare resources that perform a given

1081-7735/11 $26.00 © 2011 IEEE
DOI 10.1109/ATS.2011.89

function and tolerate defects. These techniques are generally classified by the type of redundancy used. Basically, three types of redundancy are considered: information, temporal and hardware [2].

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 [2]. Error-correcting codes are widely used in memories, but their application to logic cores requires a significant design effort associated with a high area overhead used for code prediction and computation [8, 9, 10].

Temporal redundancy consists of forcing the system to repeat a given operation and then compare the result obtained with that of the previous operation [11, 12]. Such a redundancy is able to tolerate soft and timing errors but not permanent errors.

Finally, hardware redundancy consists of modifying the design by adding extra 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 [2].

Each presented redundancy has different pros and cons with respect to the tolerance of soft and hard errors. For example, temporal redundancy targets only soft error while hardware redundancy is too expensive to target such error type. Consequently, it is judicious to use combination of redundancies in order to take advantage of each one. In this context, authors in [13] have proposed a hybrid fault tolerant architecture that uses information and hardware redundancies. In the same way, we build our hybrid fault tolerant structure in order to target robustness improvement (soft and hard error tolerance), manufacturing defect tolerance, power saving and circuit expected lifetime enhancement (aging phenomenon).

III. The hybrid fault tolerant architecture

Our goal is to increase the robustness of emerging CMOS circuits and systems by tolerating transient and permanent faults. While solutions for timing violations and soft errors in sequential elements can be found in the literature such as razor register proposed in [14, 15], this paper targets only robustness improvement in combinational part of digital circuits.

Based on the redundancy types presented in Section II, we present a new hybrid fault tolerant architecture. It uses three types of redundancy: information redundancy for error detection, temporal redundancy for transient error tolerance and hardware redundancy for permanent error correction. In the following sub-sections we present the principle and the possible configurations of the architecture.

A. Principle

Information redundancy for error detection coupled with temporal redundancy for error correction is a solution to tolerate transient errors. This first level architecture is shown in Figure 1. The information redundancy module is running in parallel with the Logic Circuit (LC). The Checker enables the error detection. Its output signal is used to activate the temporal redundancy module, which enables/disables the input/output registers. Consequently, in case of error detection, the logic computation could be re-run with the same input data and only transient errors are tolerated.

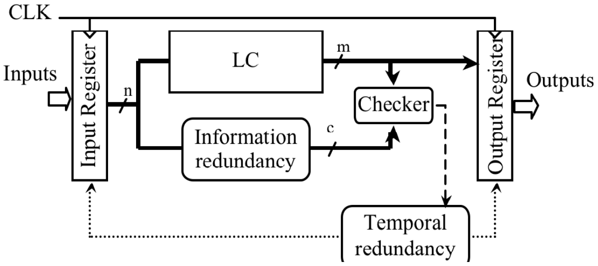

Figure 1. Architecture for transient error tolerance

Information redundancy for logic circuits has been studied in [8, 9, 16] where authors targeted low hardware code predictors. Although effective, these techniques require structural modifications of the original logic circuit. In [10], a synthesizing flow is proposed to generate parity code predictors without modification on the logic circuit structure. However, the main conclusion of this study is that parity code prediction for logic circuit is costly (in terms of area comparable to the area of the logic circuit) and not sufficiently effective in terms of error detection (in case of multiple errors). Accordingly, we use the duplication and comparison as information redundancy for our hybrid architecture. Therefore, the information redundancy module in Figure 1 will be replaced by a duplication of LC, while an output comparator replaces the checker.

As we have discussed in Section II, hardware redundancy is necessary for permanent error tolerance. A third circuit must be embedded to tolerate the malfunction. The global architecture is presented in Figure 2. As for TMR architectures, the LC is implemented three times (LC1, LC2 and LC3) but only two of them are working in parallel and are selected with the help of two multiplexors (MUX_IN, MUX_OUT). Consequently, three configurations are possible.

The comparator verifies the good functioning of the current configuration and its output (Ok signal) controls the enable input of the registers. During fault free operations, the Ok signal is true and the current configuration does not change. As long as no error is detected, only two circuits are running. The third one is in standby state. Since only two circuits are active our architecture will consume less power than classical TMR.

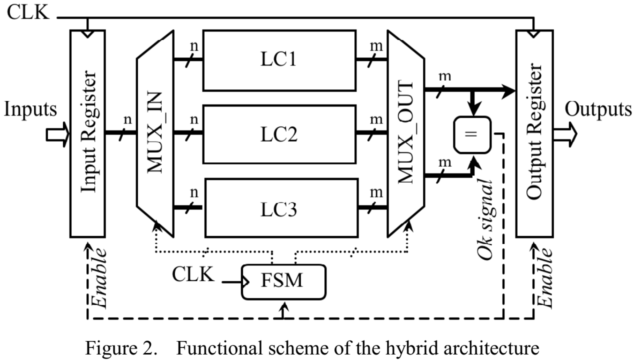

Figure 2. Functional scheme of the hybrid architecture

If the comparator detects an error, the OK signal becomes false and the registers are disabled. The Finite State Machine (FSM) changes the configuration to tolerate the detected error by controlling the multiplexors. Different configuration procedures allowed by the FSM will be discussed in the next sub-section. Note that, the architecture does not modify the original LC structure compared to the hybrid solution proposed in [13].

B. Configurations

As mentioned above, the FSM manages the configuration of the architecture by selecting a couple of circuits to run in parallel. When an error is detected, two tolerant schemes are possible:

- FSM1: the FSM does not change the configuration when the first error occurs. The two running circuits re-compute the same input data. If the error still remains at the second computation, the FSM changes the configuration. This solution puts priority in the tolerance of transient errors and requires more time for tolerating permanent faults.
- FSM2: the FSM changes the configuration each time an error is detected. This solution focuses on tolerating permanent faults and needs more time for tolerating transient faults.

The procedures for FSM1 and FSM2 are illustrated by diagrams in Figure 3 and Figure 4 and examples in Table I (for FSM1) and Table II (for FSM2).

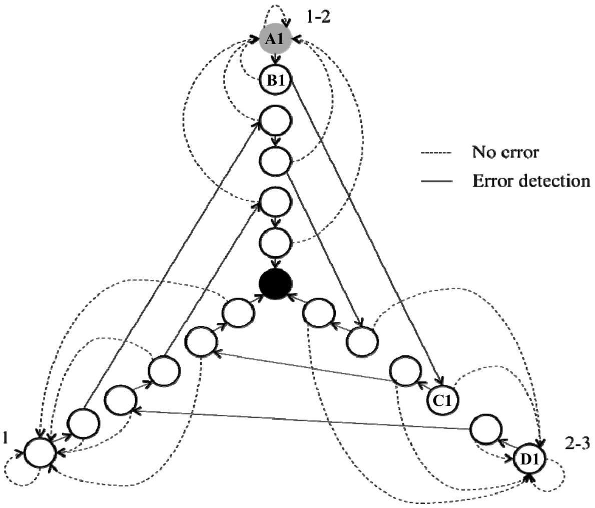

Figure 3. State diagram of FSM1

For each diagram, the initial state is in gray while the final state (the one indicating that no more correction/tolerance is possible) is in black. The branches correspond to the three configurations (1-2, 2-3 and 3-1). The edges illustrate the transitions in the graph for each new clock period. Continuous edges correspond to the case of error detection while dotted edges are used for no error occurrence.

In Table I and II, the first row indicates the clock period; the second one provides the architecture configuration, for example 1-2 means that only LC1 and LC2 are running; the third row highlights the input sequence; the last row indicates the current state in the corresponding diagram. In the two examples, when the first error occurs (error1), the same input vector V3 is repeated during the next clock period. For FSM1, the configuration still remains at branch 1-2 (state A1 and B1).

For FSM2, the configuration changes from state A2 of the branch 1-2 to state B2 of the branch 2-3. When two consecutive errors occur (error2 and error3), the input vector V5 is repeated during the two next clock periods. FSM1 only changes the configuration (from state B1 of the branch 1-2 to state C1 of the branch 2-3) at the second error detection (error3) while FSM2 reconfigures the architecture twice (from state C2 of the branch 2-3 to state D2 of the branch 3-1, then to state E2 of the branch 1-2).

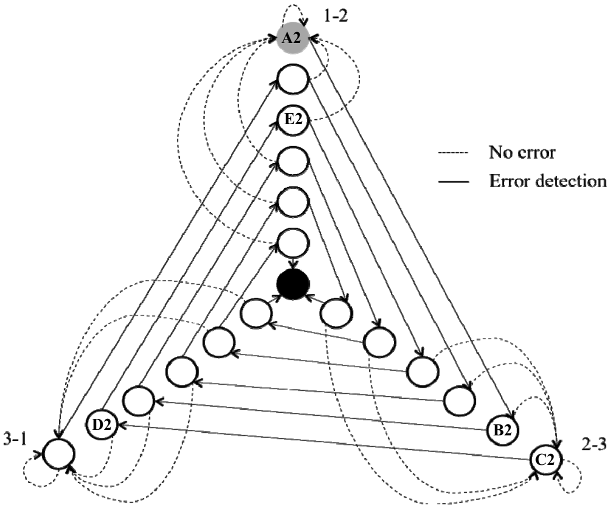

Figure 4. State diagram of FSM2

TABLE I. FSM1 FUNCTIONING EXAMPLE

Clock period	1	2	3	4	5	6	7	8	9
Configuration	1-2	1-2	1-2	1-2	1-2	1-2	1-2	2-3	2-3
Input vector	V1	V2	V3	V3	V4	V5	V5	V5	V6
Current State	A1	A1	A1	B1	A1	B1	C1	D1	D1

⇑ error1 ⇑ error2 ⇑ error3

TABLE II. FSM2 FUNCTIONING EXAMPLE

Clock period	1	2	3	4	5	6	7	8	9
Configuration	1-2	1-2	1-2	2-3	2-3	2-3	3-1	1-2	1-2
Input vector	V1	V2	V3	V3	V4	V5	V5	V5	V6
Current State	A2	A2	A2	B2	C2	C2	D2	E2	A2

⇑ error1 ⇑ error2 ⇑ error3

IV. COMPARISONS WITH THE TMR ARCHITECTURE

In order to evaluate our fault tolerant architecture, we compare it with the classical TMR solution using bit-wise voter in terms of power consumption and required area. Logic circuits used in these comparisons are ISCAS'85 benchmark circuits and combinational parts of ISCAS'89 and ITC'99 benchmark circuits.

A. Power consumption

In this sub-section we compare TMR and the hybrid fault tolerant architecture regarding power consumption. In order to perform such comparison, both architectures were synthesized using a 90nm technology with RTL Compiler™ [18]. Then, the power consumption of each architecture was evaluated with

NanoSim™ [19]. Note that, our power saving relies on the fact that only two out of three logic circuits are running in parallel while the third one is in standby. Therefore, we consider only dynamic power in this comparison. Results are summarized in Table III. The first four columns present the LC characteristics: name, number of input, number of output and transistor count. The last column show the additional power required by the TMR implementation, expressed in percentage of the new architecture's power consumption.

TABLE III. POWER SAVING WITH THE HYBRID ARCHITECTURE COMPARED TO TMR

Circuit	n	m	N_{LC}	Power saving
c5315	178	123	4183	19%
c6288	32	32	8846	57%
c7552	206	107	4960	25%
s15850	611	684	9851	9%
s35932	1763	2048	25976	13%
s38417	1664	1742	27717	11%
s38584	1464	1730	34546	9%
b14s	277	299	13328	33%
b15s	485	519	27347	27%
b17s	1452	1512	81557	25%
b18	3357	3342	210655	28%
b19	6666	6669	424235	27%
b20s	522	512	27397	32%
b21s	522	512	28523	35%
b22s	767	757	42330	35%

As shown in Table IV, for largest circuits, the hybrid architecture save about 30% of power consumption compared to TMR except for ISCAS'89 benchmark circuits. In fact, these circuits have many more inputs/outputs than other circuits of the same size. Consequently, for these circuits, the consumption of the logic part does not dominate the overall architecture power consumption. Therefore, the fact that only two LCs are running instead of three does not reduce the power consumption as expected.

B. *Area*

The second evaluation of the hybrid architecture is related to its silicon area compared to TMR. To perform this comparison we first have to estimate the cost of each module used by each architecture. Synthesis tools mentioned in the last sub-section use an optimized module to realize the voter of TMR, while no such optimization is provided for the comparator of the hybrid architecture. Therefore, for fairness, we use the transistor count method (*i.e.* the area of each module is estimated by the number of required transistors) instead of synthesis tools to evaluate the area of both architectures. In the following calculations, we consider LC with n inputs and m outputs.

1) *Transistor count of a TMR architecture*

The transistor count N_{TMR} of a TMR architecture is defined in the following equation:

$$N_{TMR} = 3 \times N_{LC} + N_{REG} + N_{VOTER} \tag{1}$$

where N_{LC}, N_{REG} and N_{VOTER} are the transistor count of the LC, the input/output registers and the voter respectively.

N_{LC} is computed for the targeted circuit given the transistor count of each logic gates.

N_{REG} is computed with respect to the number of LC input/output and with the transistor cost of a D flip-flop. From the classical transistor view of a D flip-flop with an enable command, we estimate its cost to 14 transistors. Consequently, the transistor count N_{REG} is calculated by the following equation:

$$N_{REG} = 14 \times (n + m) \tag{2}$$

Outputs of the three LCs are voted separately by m one-bit-voters. Each one-bit-voter receives three signal a, b, c and provides the output signal v as follows:

$$v = f(a,b,c) = ab + bc + ca = \overline{\overline{ab}.\overline{bc}.\overline{ca}} \tag{3}$$

Therefore, a one-bit-voter can be built by three 2-input NAND gates and one 3-input NAND gate. So, the cost of a one-bit-voter is 18 transistors. The transistor count N_{VOTER} is calculated by the following equation:

$$N_{VOTER} = 18 \times m \tag{4}$$

Finally, we have:

$$N_{TMR} = 3 \times N_{LC} + 14 \times (n + m) + 18 \times m \tag{5}$$

2) *Transistor count of the hybrid architecure*

The transistor count N_{HFT} of the hybrid fault tolerant architecture for robustness improvement is defined by the following equation:

$$\begin{aligned} N_{HFT} &= 3 \times N_{LC} + N_{REG} + N_{MUX\_IN} \\ &+ N_{MUX\_OUT} + N_{COMP} + N_{FSM} \end{aligned} \tag{6}$$

where N_{MUX_IN}, N_{MUX_OUT}, N_{COMP} and N_{FSM} are the transistor count of the input multiplexors, the output multiplexors, the comparator and the FSM respectively.

The input multiplexor allows the selection of which couple of LCs is running. In addition, it keeps the third LC in a standby state by simply connecting its inputs to the ground. Using transmission gates, this module requires 9 transistors per output.

$$N_{MUX\_IN} = 9 \times n \tag{7}$$

The output multiplexors allow the selection of two LC outputs to make the comparison. This is simply done by using two one-bit multiplexors (2:1) for each LC output. The transistor count for output multiplexors is therefore:

$$N_{MUX\_OUT} = 8 \times m \tag{8}$$

The comparator is made of two stages. The first one performs m bit–comparisons of the running LCs output. These comparisons are done by XOR functions, which can be realized with a low area gate of 4 transistors [17]. The second stage consists of building the global comparison signal (Ok signal in Figure 2). This can be done by an OR tree. For different LC output number, we evaluated the transistor count to build the comparator function. Results obtained show a linear relation between m and N_{COMP}:

$$N_{COMP} = 6.67 \times m \tag{9}$$

Finally, the two versions of the FSM (FSM1 and FSM2 presented in Section III) were synthesized. As the transistor count obtained for FSM1 and FSM2 were respectively 320 and 366, we use the following average cost:

$$N_{FSM} \approx 340 \tag{10}$$

As a result, we have:

$$N_{HTF} \approx 3 \times N_{LC} + 9 \times n + 14.67 \times m + 340 \tag{11}$$

3) Area comparisons

With the help of Equations (5) and (11), we compared the area cost of our architecture with TMR for ISCAS'85 benchmark circuits and combinational parts of ISCAS'89 and ITC'99 benchmark circuits used as LCs. Results are reported in Table IV.

TABLE IV. AREA OVERHEAD OF THE HYBRID ARCHITECTURE COMPARED TO TMR

Circuit	**n**	**m**	**N_{LC}**	**N_{TMR}**	**N_{HFT}**	**Overhead**
c5315	178	123	4183	18977	20509	7%
c6288	32	32	8846	28010	28531	2%
c7552	206	107	4960	21188	23026	8%
s15850	611	684	9851	59995	63556	6%
s35932	1763	2048	25976	168146	177533	5%
s38417	1664	1742	27717	162191	171706	6%
s38584	1464	1730	34546	179494	187249	4%
b14s	277	299	13328	53430	55267	3%
b15s	485	519	27347	105439	108416	3%
b17s	1452	1512	81557	313383	321756	3%
b18	3357	3342	210655	785907	805331	2%
b19	6666	6669	424235	1579437	1617563	2%
b20s	522	512	27397	105883	109216	3%
b21s	522	512	28523	109261	112594	3%
b22s	767	757	42330	161952	166674	3%

The three first columns present respectively the name (Circuit), the number of input (n) and the number of output (m) of each LC. The three next columns show the transistor count of the LC (N_{LC}), of the TMR architecture (N_{TMR}) and of the hybrid architecture (N_{HFT}). Finally, the last column gives the area overhead of our architecture with respect to the TMR architecture. This overhead is expressed in percentage of the new architecture's area. For all cases, the overhead required is trivial.

C. Discussion

In the sub-section above, we have compared the hybrid fault-tolerant architecture with the traditional TMR structure using bit-wise voter. The result showed that for large circuits, the hybrid architecture can save in average 30% dynamic power consumption while requiring only 3% more silicon area.

We can also reduce static power consumption of the architecture by keeping the non-running logic circuit at standby using a specific input vector instead of the vector all zero. This additional feature requires only a small change in the "Input Multiplexor" (MUX_IN in Figure 1), which will not affect the area overhead or the dynamic power consumption of the architecture.

With regard to the performance, the hybrid architecture guarantees that at least two logic circuits have the same output vector before providing this vector. This guarantee can only be achieved with TMR using word-wise voter, whose voter is larger and consumes more power than that of TMR using bit-wise voter.

V. IMPACT OF THE HYBRID FAULT TOLERANT ARCHITECTURE ON AGING PHENOMENON

In this section we discuss the ability of the hybrid architecture to deal with aging phenomenon. In fact, since only two LCs are running, the remaining one does not compute any data and hence has no activity. Consequently, for a fault free functioning, the two running circuits are those that suffer the most from the aging phenomenon. The one in standby mode normally will have a higher expected aging time and may even recover from previous activity. Let us consider an example where 1-2 is the initial configuration. After a long fault-free running period, LC1 and LC2 will become older than LC3. When both LC1 and LC2 suffer from errors due to aging phenomenon, LC3 will not be able to tolerate such problem since the hybrid architecture require two fault-free operations to be compared.

Our architecture must be modified in a way to balance the using time period of each LC. This can be done by modifying the FSM in a way to change the configuration periodically using one of the following methods:

- Time: The configuration is changed after a certain number of fault-free clock periods. This solution requires a simple counter.
- Pattern: The configuration is changed each time specific input patterns are applied. This solution requires a small memory to store these patterns.

As second discussion on aging phenomenon, we analyze further the impact of using FSM1 or FSM2 to control the configuration. Remember that FSM1 changes the configuration when two consecutive errors are detected while FSM2 changes the configuration each time an error occurs. Let us consider the following notations:

- P_i means that there is a permanent fault affecting LCi.
- S_{jk} means that a transient fault occurs for the first period of use of j-k configuration.

Tables V and VI present the fault tolerance capability of FSM1 and FSM2 for seven error scenarios. These scenarios cover some single error cases and also combinations of two errors (soft/permanent). In each table, the first row indicates the LC configuration used at each clock period and the remaining rows indicate the simulated scenarios. For example, scenario P_1-S_{23} means that LC1 is affected by a permanent fault and that a soft error occurs when changing the configuration to 2-3.

TABLE V. FAULT TOLERANCE CAPABILITY OF FSM1

	1-2	**1-2**	**2-3**	**2-3**	**3-1**	**3-1**
S_{12}	SE*	**OK***				
P_1	HE*	HE	**OK**			
P_2	HE	HE	HE	HE	**OK**	
P_3	**OK**					
P_1-S_{23}	HE	HE	SE	**OK**		
P_2-S_{31}	HE	HE	HE	HE	SE	**OK**
P_3-S_{12}	SE	**OK**				

*SE: Soft Error detection OK: Error tolerated
HE: Hard Error detection

TABLE VI. FAULT TOLERANCE CAPABILITY OF FSM2

	1-2	**2-3**	**3-1**	**1-2**	**2-3**	**3-1**
S_{12}	SE	**OK**				
P_1	HE	**OK**				
P_2	HE	HE	**OK**			
P_3	**OK**					
P_1-S_{23}	HE	SE	HE	HE	**OK**	
P_2-S_{31}	HE	HE	SE	HE	HE	**OK**
P_3-S_{12}	SE	HE	HE	**OK**		

*SE: Soft Error detection OK: Error tolerated
HE: Hard Error detection

From these results, it appears that FSM1 tolerates soft errors in presence of hard errors faster (*i.e.* in less clock periods) than FSM2. At the end of the manufacturing process, LCs can be affected by some manufacturing defects (*i.e.* permanent faults). Therefore, the probability of having a combination of hard and soft errors is higher than the probability of having hard error only. Therefore, using FSM1 to control the configuration seems to be the best solution for the beginning life cycle of the circuit.

During subsequent life cycles, aging phenomenon will increase the number of permanent faults in the circuit. Then, the probability to have hard errors will increase. FSM2 becomes more suitable in this case as it enables the tolerance of hard errors faster than FSM1.

VI. CONCLUSION

In this paper, we have proposed a hybrid architecture to improve the robustness of logic CMOS circuits. This architecture combines different types of redundancy to tolerate transient as well as permanent faults: information redundancy for error detection, temporal redundancy for transient error correction and hardware redundancy for hard error tolerance. Our approach has major benefits in terms of power consumption compared to classical TMR structure. Comparison using ISCAS'85, ISCAS'89 and ITC'99 benchmark circuits showed that adding only 2% to 3% of area compared to TMR, the hybrid architecture can save on average 30% of power consumption. In addition, it has been shown that its expected lifetime can be improved with regards to that of TMR fault tolerant structure.

REFERENCES

[1] Semiconductor Industry Association (SIA), "International Technology Roadmap for Semiconductors (ITRS)", 2010.

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

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

[4] J. Vial, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch and A. Virazel, "Using TMR Architectures for Yield Improvement", Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 7-15, 2008.

[5] 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, pp. 581-592, November 2009.

[6] A. Pullini et al. , "Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes", in Proc. of Annual Symposium on Integrated Circuits and System Design, pp. 224-229, 2007.

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

[8] N. A. Touba and E. J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, No 7, pp. 783-789, July 1997.

[9] S.-B. Ko and J.-C. Lo, "Efficient Realization of Parity Functions in FPGAs", Journal of Electronic Testing Theory and Application, vol. 20, No 3, pp. 489-499, October 2004.

[10] D. A. Tran, A. Virazel A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch and H.-J. Wunderlich, "Parity Prediction Synthesis for Nano-Electronic Gate Designs", Int. Test Conference, poster 23, 2010.

[11] 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.

[12] Y. H. Choi, M. Malek, "A Tolerant FFT Processor", in Proc. of Faut Tolerant Computing Symposium, pp. 814-823, 1985.

[13] S. Almukhaizim and Y. Makris, "Fault Tolerant Design of Combinational and Sequential Logic Based on a Parity Check Code", in Proc. of IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 563-570, 2003.

[14] T. Austin et al., "Making Typical Silicon Matter with Razor", IEEE Computer, vol. 37, No 3, pp. 57–65, 2004.

[15] S. Das, C. Tokunaga, S. Pant, W-H. Ma, S. Kalaiselvan, K. Lai, D.M. Bull and D.T. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance", IEEE Journal of Solid-State Circuits, vol. 44, No 1, pp. 32-48, 2009.

[16] K. De, C. Nataraian, D. Nair and P. Banerjee, "Synthesis of Reliable Multilevel Circuits", IEEE Transactions on VLSI Sytems, vol. 2, No 2, pp. 186-195, 1994.

[17] M. J. Sharifil and D. Bahrepour, "A New XOR Structure Based on Resonant-Tunneling High Electron Mobility Transistor", VLSI design journal, January 2009.

[18] Cadence Inc., RTL Compiler, User Guide 2008.

[19] Synopsys Inc., NanoSim™, User Guide 2006.