System-on-Chip Oriented
Fault-Tolerant Sequential Systems Implementation Methodology

S. Pontarelli, G.C. Cardarilli, A. Malvoni, M. Ottavi, M. Re, A. Salsano

{ottavi,pontarelli,salsano}@ing.uniroma2.it
{marco.re, g.cardarilli}@ieee.org
malvoni@wappi.com

Department of Electronic Engineering University of Rome "Tor Vergata", Italy

Via di Tor Vergata 110
00133-Rome-ITALY

ABSTRACT

This paper presents a design methodology for fault tolerant sequential systems implemented on System on Chip (SoC). In the paper, as an example, a complex fault tolerant finite state machine has been mapped on the FPGA contained in the SoC. The fault identification has been obtained by using a checker permitting the identification of class of faults.
When a fault is detected, an interrupt for the microcontroller is generated and the interrupt handling routine partially reprograms the FPGA to override the part of memory configuring the faulty block.
The architectures of the SoCs recently appeared on the market are characterized by a very efficient interaction between the microcontroller and the FPGA allowing a very efficient implementation of the fault detection and fault recovery strategy. A test bed of the proposed methodology has been implemented on the recently presented Atmel AT94K FPSLIC (Field Programmable System Level Integrated Circuits).

1. INTRODUCTION

Different approaches to the design of high reliability digital electronic systems for hostile environments have been proposed in the past. The kind of used technique depends on the level of reliability and security required for the specific application.
The main approaches to the design of high reliability electronic systems are mainly two: Fault avoidance and Fault Tolerance. The former can be accomplished by using a technology based approach e.g. Radiation-Hard components (like S.O.I. or similar) [1] while the latter can be accomplished by using suitable system level based techniques [2].
Radiation-Hard (RH) components are built by using expensive technological processes and guarantee the required protection against high-energy radiation. The main drawbacks of using radiation hard components is the component cost and the unavailability of the state-of- the-art parts in the RH version.
As stated before, fault tolerance is obtained by using system level strategies like

- Triple Modular Redundancy (TMR) or in general N Modular Redundancy (NMR) originally suggested by Von Neumann [1]
- Fault detection, fault masking and recovery [2]

Nowadays, a big research effort is spent on the fault detection approach. This method present the benefit of a lower hardware redundancy with respect to the TMR approach but introduces the necessity of reconfiguration algorithms (software redundancy) and consequently downtime caused by the MTTR (Mean Time to Repair).
Moreover, in space applications, where the number of manufactured parts is small, the use of standard components, which can be reconfigured by a simple and inexpensive procedure, is becoming increasingly important. Currently, some companies like Atmel [4] and Triscend [5] are proposing solutions (called System on Chip or SoC) that integrate on the same chip a considerable amount of memory, a field programmable logic array

(FPGA) a microcontroller and some peripherals. The interaction between microcontroller and FPGA can dramatically decrease the MTTR simplifying the reconfiguration algorithms when fault detection, fault masking and reconfiguration are the chosen techniques to approach the reliability issues.
The SoC is a very good candidate for the implementation of fault tolerant systems; in fact, we can exploit the intrinsic flexibility offered by such architectures. In particular:

- The microcontroller can easily reconfigure the FPGA to mask faults occurred in the FPGA
- The FPGA can directly communicate with the microcontroller if an error occurs in a functional block

The fault models applicable to such device, when used in spatial environments, can be divided into two main categories [6]:

- Transient faults: e. g. Single Event Upset (SEU), that causes a bit-flip in the FPGA configuration memory.
- Permanent faults: e. g. Total Ionizing Dose (TID), that can modify the threshold voltage of the MOS transistor. This physical effect, that causes the transistor to loose its switching capability, can be modeled as a stuck-at 0 or 1 (S-A-0/1) in the circuit nodes and stuck open/close on the circuit interconnections.

The paper is organized as follows: in Section 2 the fault detection methodology is illustrated. Section 3 and 4 describe respectively the fault detection using parity check codes and the fault recovery for transient faults. In section 5 the fault diagnosis using the microcontroller as a test pattern is described while in section 6 is presented the fault recovery strategy in the presence of permanent faults. The conclusions are drawn in Section 7.

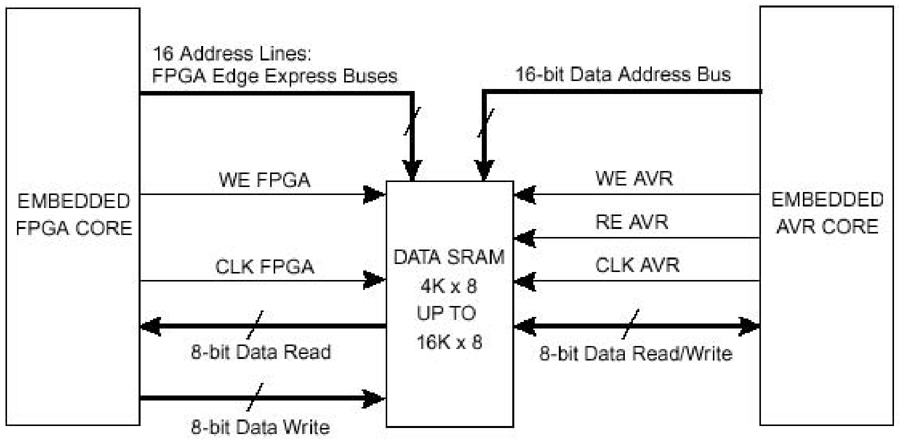


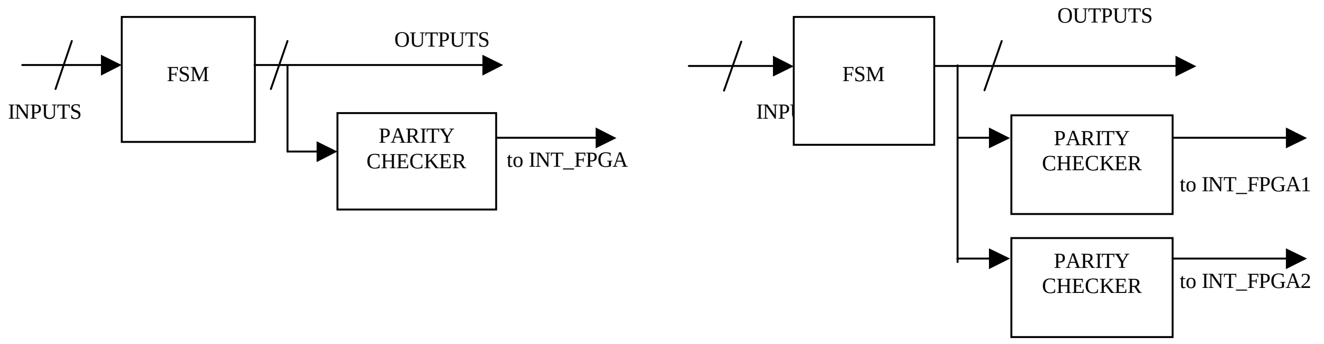
Fig. 1: Atmel AT94K FPSLIC

2. FAULT DETECTION AND FAULT COVERAGE STRATEGY

To obtain fault detection we use a scheme based on error detection code. In particular, a self-checking FSM, synthesized like in [7], has been mapped in the FPGA contained in the SoC. The control of the outputs correctness is achieved by using parity based code. The faults, detected by the checker, cause interrupts sent toward the microcontroller. The unused Complex Logic Blocks (CLB) of the FPGA are used as spares to replace the faulty blocks. The microcontroller (supposed to be fault free) is used to obtain fault recovery. The fault recovery strategy is simply achieved through the interaction between the FPGA and the microcontroller. In fact, when a fault causes the occurrence of the first incorrect output in the FSM, the microcontroller starts the FPGA reconfiguration procedure. The microcontroller reconfigures the FPGA, initially reprogramming the part of memory storing the configuration of the faulty block. This procedure corrects the occurrence of transient faults like SEU. If an error is detected again in the same functional block, and it occurs in a time interval shorter than the estimated mean time between faults (MTBF), we suppose the occurrence of a permanent fault. In this case the microcontroller maps the faulty block in an unused area of the FPGA and marks as unusable the block (Faulty Block) where the error persisted. In the following section, more particular are given on the fault detection phase.

3 FAULT DETECTION USING PARITY CHECK CODES

In the presented methodology, the self-checking property has been obtained by using parity check codes and the FSM (mapped into the FPGA) is synthesized by using the technique presented in [7]. The use of parity check codes means that output correct code words show always the same parity. The architecture of the circuit mapped in the FPGA is show in Fig.2a. A fault occurred in the FSM is detected by the parity checker, and consequently an interrupt is sent to the microcontroller. The fault diagnosis and recovery routine described in the following section masks the fault occurrence. However, with this method the occurrence of a fault in the Parity Checker (e.g. a stuck-at in the INT_FPGA line) could be undetected, so a second fault in the FSM could give undetected errors. To avoid this we developed a check routine based on fault diagnosis to detect the faults in the parity checker. The routine is executed from the microcontroller with a period shorter than the Mean Time Between Faults (MTBF). Using this routine a fault in the parity checker is detected by the µC before the FSM produces an incorrect output. Moreover, a modify in the basic scheme is show in Fig.2b. This scheme simply duplicates the parity checker. In this case we program the AVR/FPGA interface to connect both of the two checker outputs to an interrupt line of the AVR microcontroller (called INT_FPGAn).



a) FSM with one parity checker b) FSM with two parity checkers

Fig. 2: FSM with parity checkers

Fig. 3: FPGA Interrupt Routine

```
void int1_routine (void) {   /* External interrupt on INT1 */
 flag++;
 for (i=0;i<10;i++) {/* Do nothing waiting the other interrupt */}
if (flag==2){ /* Fault detected in FSM: Reprogramming */
   riprog_fsm();
   flag=0;
   exit(1);
}
if (flag==1){    /* The other interrupt is not arrived Fault detected in Parity Checker 1: Reprogramming */
    riprog_parity1();
    flag=0;
    exit(1) ;
    }
if (flag==0) exit(1);                          /* Routine executed by the other Interrupt */
}
```

A fault occurred in the FSM is detected by the two checkers, so two interrupts are sent toward the microcontroller. This change prevents the case of a fault in FSM is undetected because the parity checker is unavailable during the check routine. This modify allows to check one parity checker while the other is able to detect faults in the FSM. The routine, in the case of two parity checkers is described in Fig.3.

The routine handles the two interrupts and launches the fault diagnosis and recovery routine described below. Using interrupts the µC is able to do its normal operation during the normal conditions (no fault detected). Only during the fault diagnosis and recovery steps the µC is exploited to obtain fault tolerance capability.

4. FAULT RECOVERY FOR TRANSIENT FAULTS

The second step, the system recovery after a SEU like fault, is easily managed by µC, refreshing the programming bit of the FPGA. After the second step, the timer controlling the MTBF is initialized to discriminate permanent from transient fault. To refresh the FPGA we use the routine described in Fig 4.

```
void refresh() {
  for ( windows = 1; windows <= num_windows; windows++) {
      FPGAX_INI = bitstreams[i++]; FPGAY_INI = bitstreams[i++];
      FPGAX_END = bitstreams[i++]; FPGAY_END = bitstreams[i++];
         for ( x= FPGAX_INI ; x<= FPGAX_END ; x++ )
           for ( y= FPGAY_INI ; y<= FPGAY_END ; y++ ) {
            FPGAX = x; FPGAY = y; FPGAD = bitstreams[i++] }
  }
}
```

Fig. 4: FPGA Reconfiguration Routine

If another error occurs in an interval shorter than MTBF, we suppose an occurrence of a permanent fault. In this case, the µC locate the faulty CLB using the diagnosis routine described in the following section.

5. FAULT DIAGNOSIS USING MICROCONTROLLER AS TEST PATTERN GENERATOR

In literature, the proposed method to detect a faulty CLBs are divided in two family. The first family is based on methods like described in [8]. A group of six CLBs, called BISTER, is configured as a *Test Pattern Generator* (TPG), two identically configured *blocks under test* (BUTs), and an *Output Response Analyzer* (ORA). The two major drawbacks of this technique are:

1) The necessity to test every group with various configuration, in fact every CLB must be confronted with two other CLB to locate the faulty BISTER.

2) The use of adaptive divide-and-conquer strategies to locate exactly the faulty CLB in the faulty BISTER.

The other methods proposed use external hardware to test the CLBs.

Our approach uses the µC as a Test Pattern Generator and as a Response Analyzer. The µC begins the test diagnosis configuring a group o CLB like in Fig. 5.

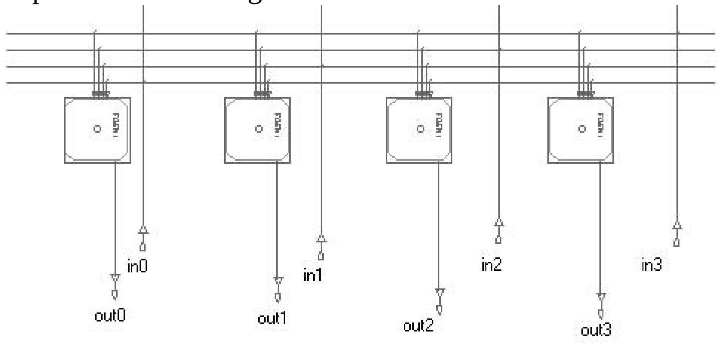


Fig. 5: Test Configuration Routing for CLBs

The input lines come from the AVR to the FPGA and give the test bench to the CLB. The output lines come from the FPGA to the AVR to detect faults. The 4-input LUT in the CLB are configured twice. In the first configuration the LUT contains a 16 bit word “0110 1001 1001 0110” (LUT word) and the 16 input vectors are passed to the CLB. In The second configuration the LUT word is inverted (“1001 0110 0110 1001”) and the test vector is passed again. This configuration check every S-A-0/1 faults in the programming SRAM of the LUT. It also checks the S-A-0/1 in the input and output lines. Moreover, because the LUT are realized with 2-input multiplexer, like in many LUT-based FPGAs, this configuration detect also S-A-0/1 in select and ouput lines of the internal 2-input mux. In the Fig. 6 we show the waveform of a not faulty LUT, a S-A-0 in the programming SRAM and a fault in a internal 2-input mux of the LUT.

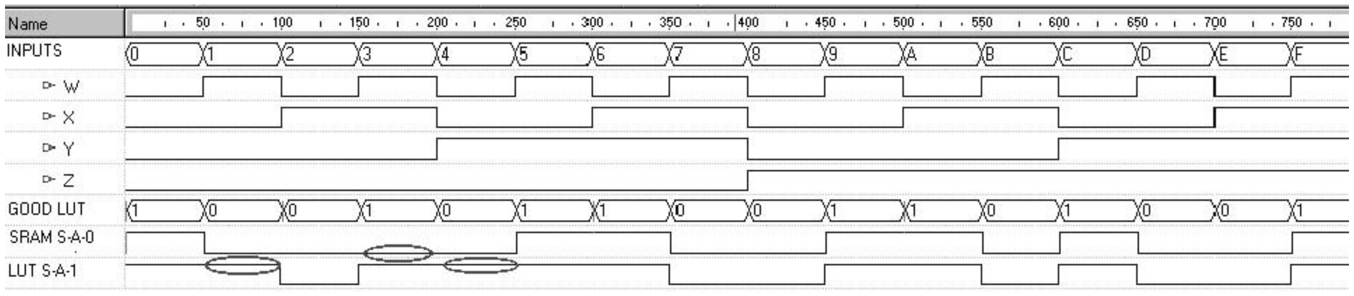
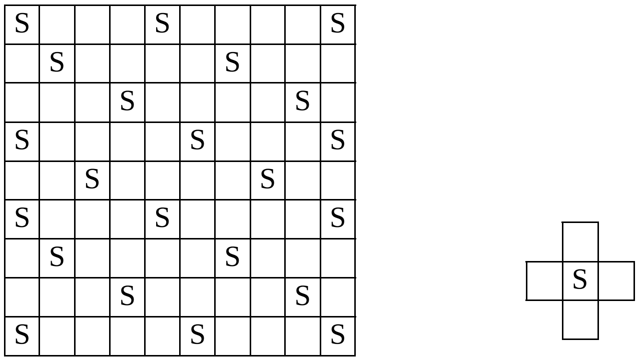


Fig. 6: Output of good and faulty LUTs

6. FAULT RECOVERY FOR PERMANENT FAULTS

To recover the FSM, after the diagnosis of the faulty CLBs we use the spare CLBs distributed like in Fig. 7a. Ignoring the boundary of the FPGA the spare CLBs are the 20% for the CLBs presents in the FPGA. Every Spare CLB can substitute the adjacent one like in Fig. 7b.



S				S					S
	S					S			
			S					S	
S					S				S
		S					S		
S				S					S
	S					S			
			S					S	
S					S				S

a) b)

Fig. 7: Positions of Spare CLBs

The choose of this topology is justified by the particular routing resource of the Atmel FPGA. In fact, direct routing line of the AT40K family CLBs can directly connect also the diagonal adjacent CLB. In particular the X input of the CLB can directly be connected with the output of the closest horizontal and vertical CLBs. Similarly the Y input of the CLB can be connected with the output closest diagonal ones. So many direct connection can the restored in the new configuration simply modifying the output direction from diagonal to horizontal/vertical, and exchanging in the LUT word the X input with the Y input. A Similar exchanging can be done for horizontal/vertical to diagonal connection. Fig.8 shows the routing of a CLB before and after remapping. The precompiled bitstream are very small because only few programming bit must be modified with such topology.

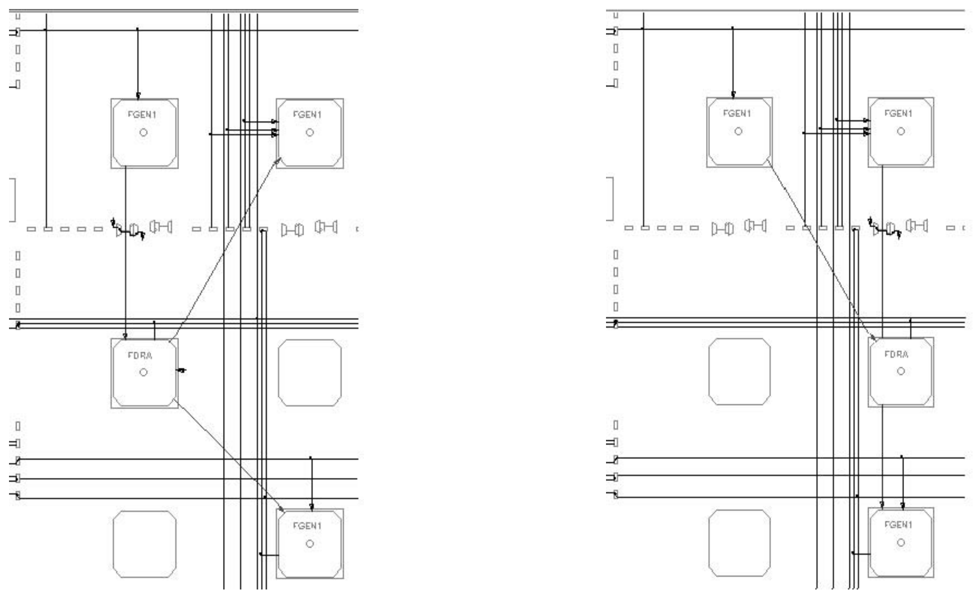


Fig. 8: Routed CLBs before and post remapping

7. CONCLUSIONS AND FUTURE WORKS

In this paper a 4-step strategy to achieve fault tolerance using SoC is presented. In the first step make use of fault detection capability of the self-checking FSM. The second step, the system recovery after a SEU like fault, is easily managed by μC, refreshing the programming bit of the FPGA. After the second step, the timer controlling the MTBF is initialized to discriminate permanent from transient fault. If a permanent fault is detected, in the third the μC executes a fault diagnosis routine able to detect the single faulty CLBs. In the last step the μC reprogram the FPGA using the spare CLBs. The proposed methodology is able to tolerate an high number of permanent faults exploiting the SoC features. Moreover, the exact localization of the faulty CLBs allows a very graceful degradation of the FPGA contained in the SoC. At present, our work is oriented toward the development of self-checking algorithms for the microcontroller auto-diagnosis.

8. REFERENCES

[1] J. Von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components". Automata Studies, in Annals of Mathematical Studies No. 34, pp. 43-98, Princeton University Press 1956.

[2] P. K. Lala, "Fault tolerant and fault testable hardware design". Prentice-Hall International

[3] Triscend Home Page: http://www.triscend.com

[4] Atmel Home Page: http://www.atmel.com

[6] H. Johnston, "Radiation Effects in Advanced Microelectronics Technologies", IEEE Trans. Nuc. Sci., Vol. 45, No. 3, June 1998

[7] Z. Chaohuang, N. Saxena, E.J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection", Test Conference, 1999. Proceedings. International

[8] J.Emmert, C. Stroud, B.Skaggs; M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration", 2000 IEEE Symposium on Field-Programmable Custom Computing Machines