Design and Implementation of a Custom Verification Environment for Fault Injection and Analysis on an Embedded Microprocessor

Buse Ustaoglu
Istanbul Technical University
Istanbul, Turkey
Email: ustaoglubu@itu.edu.tr

Berna Ors
Istanbul Technical University
Istanbul, Turkey
Email: siddika.ors@itu.edu.tr

***Abstract*—Embedded microprocessors are widely used in most of the safety critical digital system applications. A fault in a single bit in the microprocessors may cause soft errors. It has different affects on the program outcome whether the fault changes a situation in the application. In order to analyse the behaviour of the applications under the faulty conditions we have designed a custom verification system. The verification system has two parts as Field Programmable Gate Array (FPGA) and personnel computer (PC). We have modified Natalius open source microprocessor in order to inject stuck-at-faults into it. We have handled a fault injection method and leveraged it to increase randomness. On FPGA, we have implemented modified Natalius microprocessor, the fault injection method and the communication protocol. Then the "Most Significant Bit First Multiplication Algorithm" has been implemented on the microprocessor as an application. We have prepared an environment which sends inputs to and gets outputs from the Natalius microprocessor on PC part. Finally, we have analysed our application by injecting faults in specific location and random location in register file to make some classifications for effects of the injected faults.**

***Keywords*—Fault Injection, Design, Analysis, Microprocessor**

I. Introduction

Digital systems complexity has been increasing. An explosive growth of embedded microprocessor distribution for a number of safety-critical applications suggests a strong need for understanding reliability as it applies to the embedded design space [4]. The importance of test and verification of digital systems are increasing in order to have high reliability. Hence, designers should have ability to test the systems under the realistic fault occurances. In order to produce faults during design time a fault injection method can be used. The fault injection environment plays a crucial role in the data collection and measurement.

Fault injection is a key to evaluating fault-tolerant techniques. Those approaches can be divided into hardware based fault injection [2] and software based fault injection [13]. Furthermore, software based fault injection can be classified as software-implemented fault injections and simulation based fault injections. One type of the simulation based fault injections, Verilog HDL,with the high observability and controllability can be obtained.

The steps of the fault injection to the microprocessor are as follows:

1) Selection of open source microprocessor
2) Design of fault injection circuit
3) Modification of the microprocessor
4) Writing an application code
5) Analyzing the results

Reduced Instruction Set Computer (RISC) architecture is used in embedded microprocessor design. One of the key RISC features is that the arithmetic operations are performed on register values [11]. This property makes the register file the most important part of the microprocessor.

The remainder of the paper is organized as follows. Section II gives the background information about Natalius microprocessor, the "Most Significant Bit First Multiplication Algorithm", general fault concepts, their types and effects. The detailed description of the system is discussed in Section III. Section IV presents the analyses of the system and Section V summaries the paper.

II. Background Information

A. Natalius Microprocessor

Natalius is a compact, capable and fully embedded 8 bit RISC processor core described 100% in Verilog. Natalius offers an assembler that can run on any python console. The instruction memory is implemented using two Xilinx Block-RAM Memories, it stores 2048 instructions, each instruction has a width of 16 bits (2048x16). Each instruction takes 3 clock cycles to be executed [7].

Features:

1) 8 Bit ALU
2) 8x8 Register File
3) 2048x16 Instruction Memory
4) 32x8 RAM Memory
5) 16x11 Stack Memory
6) Three CLK / Instruction
7) Carry and Zero flags
8) 8 bit Address Port (until 256 Peripherals)
9) 30 instruction set

B. Most Significant Bit First Multiplication

Most Significant Bit First Multiplication Algorithm is based on shift and add operations. The steps needed for

the multiplication are shown in Alg. 1. The algorithm starts by loading the multiplicand into the A register, loading the multiplier into the B register, and initializing the C register to 0. The counter i is initialized to $n-1$. The most significant bit of the multiplier register (b_{n-1}) determines whether the multiplicand is added to the product register [3].

Algorithm 1 Most Significant Bit First Multiplication

Require: A, $B = (b_{n-1}, b_{n-2}, \ldots, b_1, b_0)_2$ are positive integers
Ensure: $C = A \times B$
  $C = 0$
  for $i = n-1 : 0$ **do**
    $C = 2 \times C$
    if $b_i = 1$ **then**
      $C = C + A$
    end if
  end for

C. Definitions of Fault Terms

Fault is a physical defect, imperfection, or flaw that occurs within some hardware or software component. When a fault causes an incorrect change in a machine stage, an error occurs [14].

Hardware faults are classified with respect to fault duration. A permanent fault which is caused by some physical defects in hardware remains active until a corrective action is taken. A transient fault remains active for a short period of time [5]. Soft errors, also referred to as transient faults, are primarily caused by environmental effects when a transient fault occurs in a computer system, it can corrupt the application output or crash the system [8].

Faults can produce different errors and different failures at different moments of the system's life. These effects depend on the fault location and on the activity of the system during and after the fault's occurrence [6].

The most common model used for logical faults is the single stuck-at fault. It assumes that a fault in a logic gate results in one of its inputs or the output is fixed at either a logic 0 (stuck-at-0) or at logic 1 (stuck-at-1) [9].

III. System

Our system constitutes FPGA and PC part. Design Under Test (DUT) and Hardware Communication Module in FPGA part, Main Controller Unit(MCU) in PC part are the main units of the system as shown in Figure 1.

A. Design Under Test

1) Fault Injection Block: Fault Injection Block is composed of 3 8-bit linear feedback shift registers (LFSR), a fault decision and a fault location blocks. Figure 2a and 2b shows the schematic of the block and the overall fault injection algorithm.

Linear Feedback Shift Register Blocks

A major feature of the fault injection system is the ability to insert faults at desired intervals. To accomplish this task the injection system uses pseudo-random sequences. Pseudo-random sequences of maximal length are generated using LFSRs. The three 8-bit LFSRs run in parallel constantly generating pseudo-random sequences [10].

Fault Decision Block

The outputs of two LFSRs feed the fault decision circuit with a 3 bit control input. The least significant bits of the outputs are compared depending on the control input. Control inputs are for transient injection and range from "001" $\rightarrow$ 50% injection to "110" $\rightarrow$ ¡ 1% injection. A control input of "000" is 0% fault injection while a control code of "111" is 100% fault injection. The control input is used to control the point in time at which the fault is injected. By incrementing this control input by "1" for each 3 bit pattern, fault injection is dropped by $^1/_2$ from the previous rate.

Fault Location Block

The third LFSR determine the location of the 8x8 register file of an embedded microprocessor. The location output is 6 bit number and grouped as most significant and least significant 3 bits to make two dimensional corresponding to row and column. For example, if third LFSR output is "011010" (26 in decimal) the row is "011" and the column is "010". Thus, Figure 3 shows the fault signal strikes second bit of the third register.

2) Modified Natalius Microprocessor: In this step, some modifications have been made in order to inject faults into the microprocessor and the multiplication algorithm has been implemented on it.

Design Modifications

Figure 4a and Figure 4b show the original design of Natalius and its register file. Thus; the outputs of the fault injection unit have been added as inputs to Natalius and its register file that shown in Figure 4c and Figure 4d.

Implementation of Multiplication Algorithm

The code by using instruction set of Natalius has been shown in Figure 5. *r1* and *r2* registers correspond to A and B, *r3* is C, *r4* and *r6* provides loop operation describes i in the

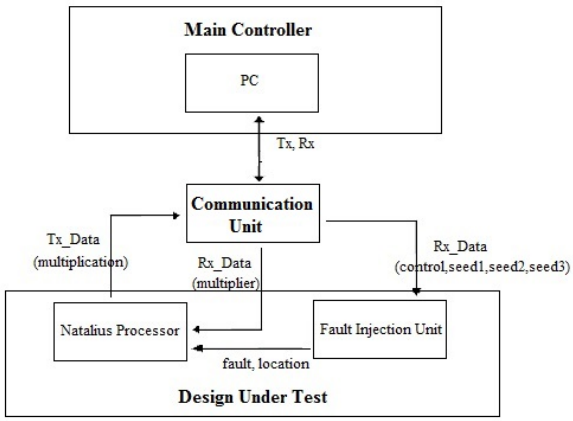
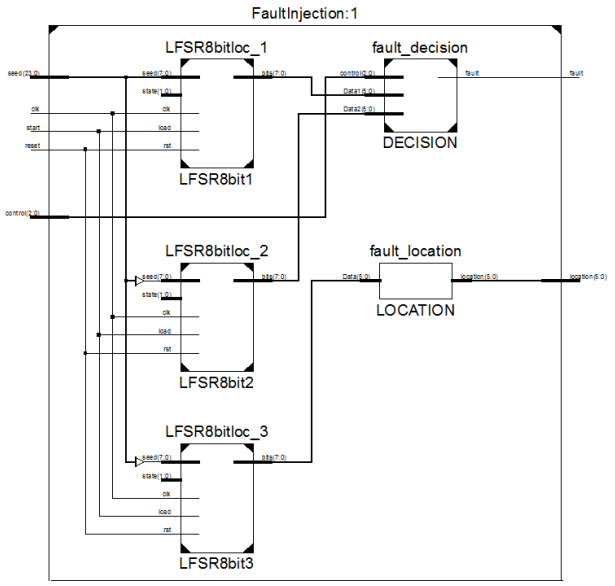
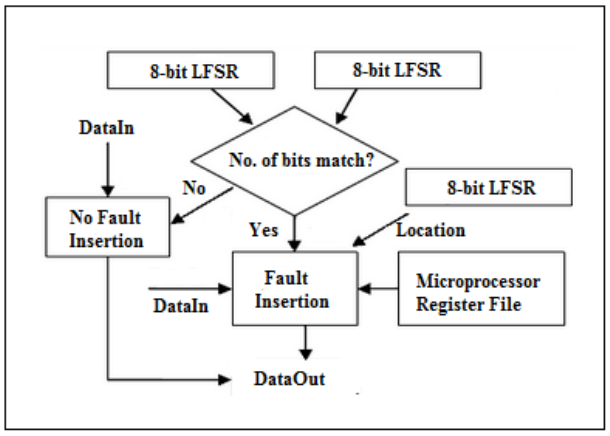


Fig. 1: System



(a) Fault Injection Block

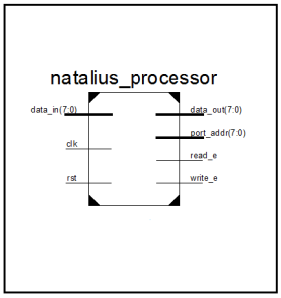


(b) Fault Injection Algorithm

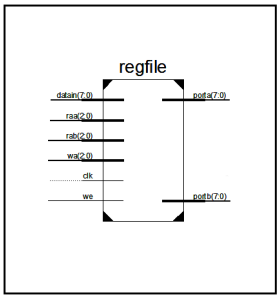
Fig. 2: Fault Injection Structure

Microprocessor 8x8 Register File								
r7								
r6								
r5								
r4								
r3						■		
r2								
r1								
r0								
Bits	111	110	101	100	011	010	001	000

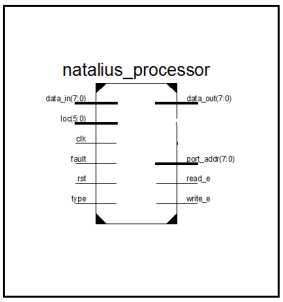
Fig. 3: Fault Location in Register File



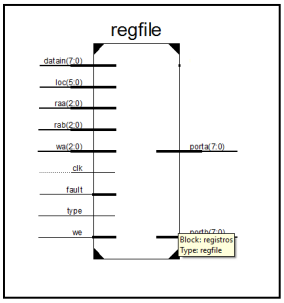
(a) Original Processor



(b) Original Register File



(c) Modified Processor



(d) Modified Register File

Fig. 4: Design Modifications on Natalius Microprocessor

algorithm explained in Alg. 1. *r5* is the constant and checks the most significant bit of *r2* and *r7* helps the checking.

B. Hardware Communication Unit

This unit is responsible for communication between MCU and DUT. Data transmission has been made by using Universal Asynchronous Receiver and Transmitter (UART) protocol and the baud rate is 9600 bps. Finite State Machine (FSM) has been designed and Figure 6 shows the flowchart of the mechanism. From state 0 to 7 seeds of the LFSR blocks and control, stuck-at-value, multiplier data are received from MCU. The multiplication results is waited and sent to MCU from state 8 to state 11, this process is repeated until the determined number of results are collected and FSM returns to its initial state wait new information.

C. Main Controller Unit

MCU can be seen in Figure 7 and described by using verification terminology. Verification environment is the top module and it contains one or more agents [1]. In our system bash script code works as a verification environment. It operates the system automatically depending on N which is 8 bit number from 0 to 255. The least significant four bits is multiplier, the most significant bit determine stuck-at-value input, the three bits between them are control input as seen in Table I. Agent written in C (communication.c) is a bridge in order to access Hardware Communication Module and includes sequencer, monitor, driver components. The main function generates three 8 bit random numbers to seed LFSR blocks. Moreover it acts as a sequencer because it calls sub-functions in sequence:

openport(): Do the serial port settings and start communication.

sendport(data): Send data to the serial. Every data length to be sent is one byte. The direction of data is from MCU to

ISBN: 978-1-4799-5679-1 ©2015 IEEE

```
forever   csr square
          jmp forever
square    ldm r1,1
          stm r1,1
          ldm r2,2
          csr multsoft
          stm r3,7
          ret
multsoft  ldi r3,0
          ldi r4,1
          ldi r6,9
          ldi r5,128
multloop  add r3,r3
          ldi r7,0
          oor r7,r2
          oor r7,r5
          cmp r2,r7
          jpz addbit
continue  add r2,r2
          sub r6,r4
          cmp r6,r4
          jnz multloop
          ret
addbit    add r3,r1
          jmp continue
```

Fig. 5: Assembly Code of the Multiplication Algorithm

TABLE I: N bit groups

7	6 5 4	3 2 1 0
type	control	multiplier
0	0..7	0...15
1	0..7	0...15

DUT, so its functionality is like driver.

readport(data): One byte data are read. The function behaves like monitor because the data are correspond to multiplication results coming from DUT.

closeport(): Close the serial port and ends communication.

Scoreboard has been written also C code (ratecalculator.c). First multiplication operation is made virtually then the real results from DUT and virtual results are compared. Finally the percentage of the correct results are calculated to analyse DUT.

IV. Analysis

DUT have been implemented into the Xilinx Spartan3s-500e board that has 20 ns clock period and in Figure 8. the occupied area has been given. Multiplier algorithm produce a result after 5940 ns because 99 instruction is operated, an instruction takes 3 clock cycles. Fault Injection Unit has been operated every instruction cycle. 256 measurements have been taken for every N three different times. Thus the analyses are the percentage of 768 results for every multiplier and the control inputs.

In the analysis, multiplicand and multiplier have been chosen same and 4-bit binary numbers in order not to occur overflow in the result because of 8-bit width register and stuck-at-0 type faults have been injected to the microprocessor. We have divided the analyses two parts.

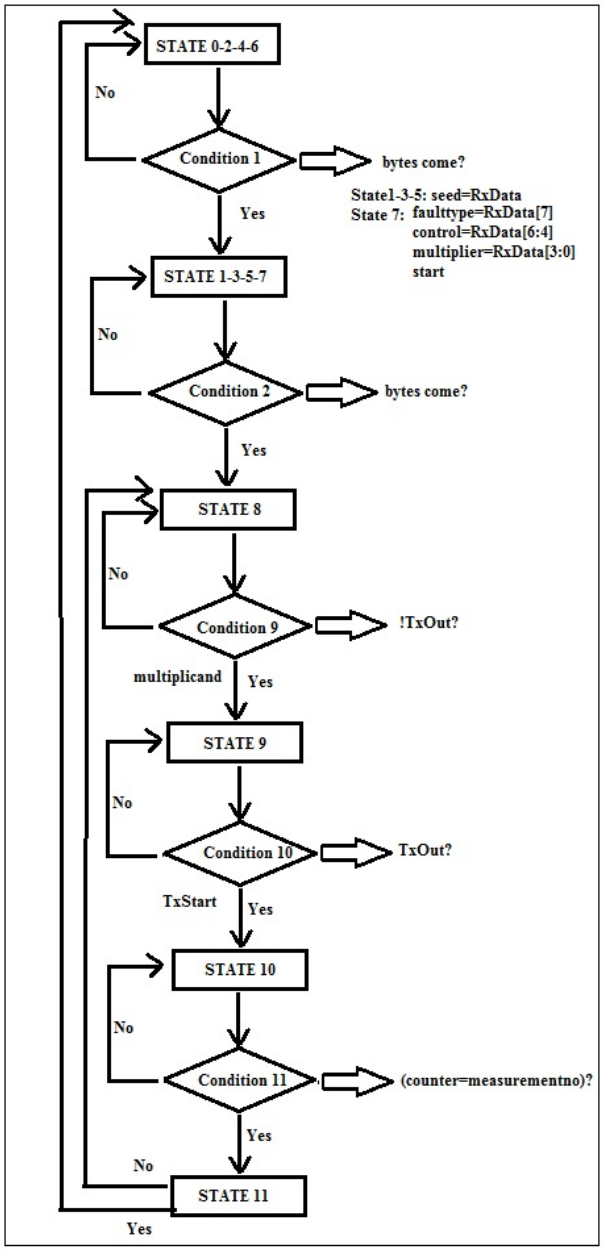


Fig. 6: Hardware Communication Unit Flowchart

A. In Specific Register

In this analysis, Fault Location Block has not been used and the fault signal has been given in the specific location. The number of ones and zeros in the multipliers affect the result. Multipliers have been grouped into datasets. Our aim has been to classify multipliers so the most significant bit of *r2* has been chosen in our application. The bit is the critical location because it has been checked every step in the algorithm. Figure 9 shows that the same set of data have the same behaviour. Dataset 0 results have not been affected and for the same control value from dataset 1 to dataset 4 the number of ones of the multiplier have increased and the percentage of the correct results have decreased because of the stuck-at-0 faults.

ISBN: 978-1-4799-5679-1 ©2015 IEEE

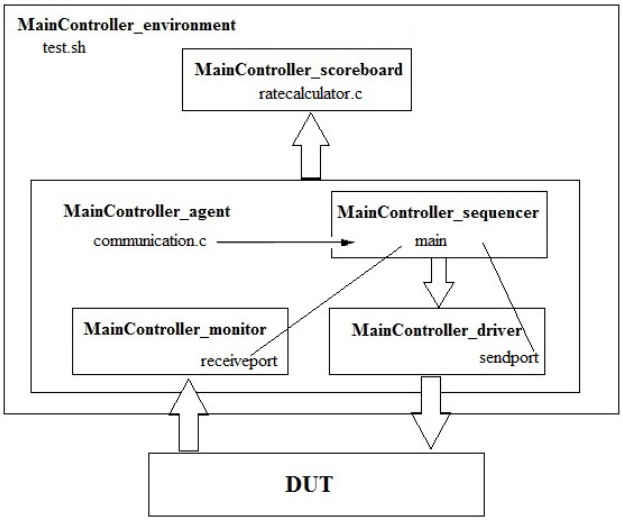
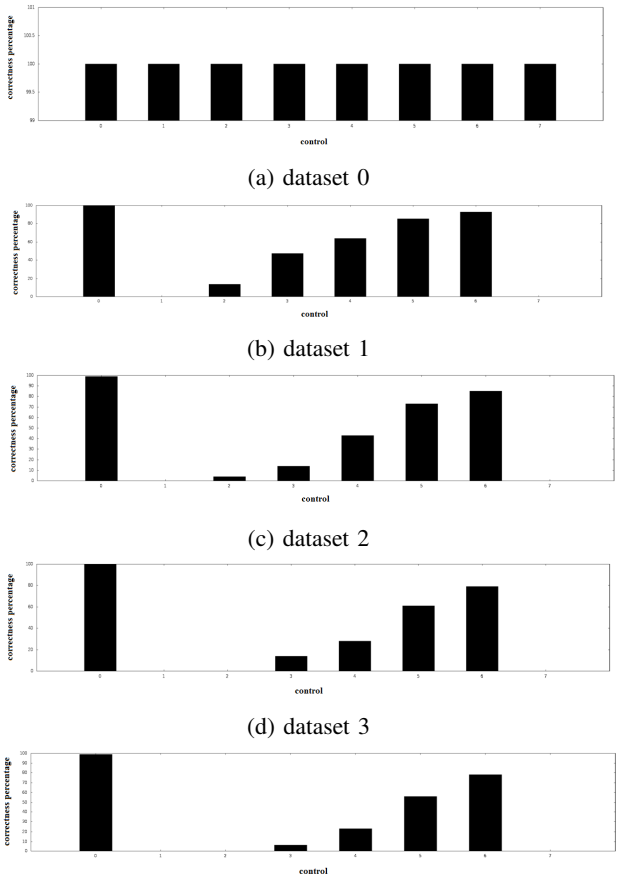


Fig. 7: Main Controller Unit

```
Design Summary Report:
 Number of External IOBs                  22 out of 232      9%
 Number of External Input IOBs            5
 Number of External Input IBUFs           5
 Number of LOCed External Input IBUFs     5 out of 5       100%
 Number of External Output IOBs           17
 Number of External Output IOBs           17
 Number of LOCed External Output IOBs     9 out of 17       52%
 Number of External Bidir IOBs            0
 Number of BUFGMUXs                       1 out of 24        4%
 Number of RAMB16s                        3 out of 20       15%
 Number of Slices                         727 out of 4656   15%
 Number of SLICEMs                        6 out of 2328      1%
```

Fig. 8: Design Summary



(a) dataset 0

(b) dataset 1

(c) dataset 2

(d) dataset 3

(e) dataset 4

Fig. 9: Datasets

B. In Random Register

The analysis is to show the fault is distinguishable or indistinguishable based on the application and fault location block is used. The distinguished faults are easily identified and indistinguished faults are not easily identified [12]. The maximum number of faulty registers based on the control input and fault rate is given in Table III if the fault location is different for every instruction cycle during the operation. Especially at low fault rate in our approach, the fault may be injected into an unused register or the stuck-at-value and faulty bit is the same, the result will not be affected so the fault is indistinguishable. But the unused register bit may be the most critical location or the stuck-at-value and faulty bit is different in other application. In our algorithm the multiplication is the result of cumulative faults. The percentage of correct result may not increase while the fault rate is getting lower. In addition, Figure 10 shows that the behaviour of the data in the same set are different .

TABLE II: Datasets

Datasets	Number of 1-0
0	0-8
1,2,4,8	1-7
3,5,6,9,10,12	2-6
7,11,13,14	3-5
15	4-4

TABLE III: Fault Rates Based on Control Input

Control Input	Fault Rate	Maximum Number of Faulty Registers
0	0%	0
1	50%	49
2	25%	25
3	12.5%	13
4	6.25%	6
5	3.125%	3
6	1.5675%	2
7	100%	64

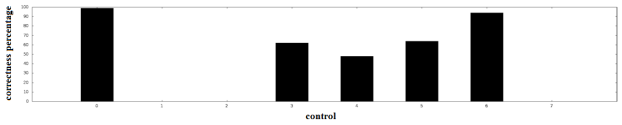
V. Conclusions

In this work, we have presented a custom verification system in order to analyse the fault effects on an embedded microprocessor applications.

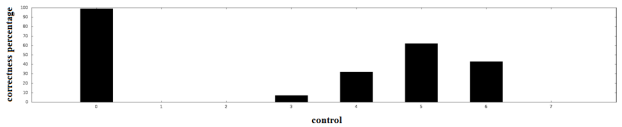
Experimental results have shown that the multiplication results are 100% correct when control input value is 0. This shows our system works as we have expected.

The value of the multiplier is the dominant factor because the same behaviour has been observed for the same datasets if the faults have been injected into the most critical register and also percentages of the correct results have increased from 3 to 6 control value for all datasets.

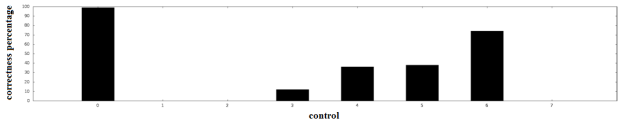
ISBN: 978-1-4799-5679-1 ©2015 IEEE



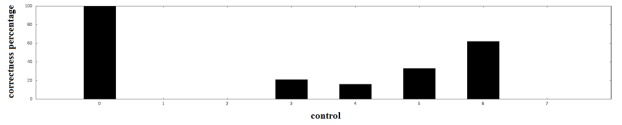
(a) multiplier value 7



(b) multiplier value 11



(c) multiplier value 13



(d) multiplier value 14

Fig. 10: The application behaviour of the same datasets

In contrast, the behaviour of the same dataset are different when the faults have occurred in the random registers. Moreover there is not always an increase from 3 to 6 control value because maximum number of faulty registers are getting closer. The fault may be indistinguishable on the program output depending on the location.

The percentages of the correct results are almost 0 and common for two analyses if the control values are 1 and 2. The algorithm results are the outputs of the cumulative faults because the fault rates are high and faults have been injected in the registers for every instruction cycle during the faulty period.

In future work, we will implement fault tolerant methods on $32/64$ bit embedded microprocessors and analyse more realistic safety critical applications and generalize the custom verification environment.

REFERENCES

[1] Accellera, 8698 Elk Grove Blvd. Suite 1, Elk Grove, CA. *Universal Verification Methodology (UVM) 1.1 User's Guide*, 1.1 edition, May 2011. www.accellera.org/downloads/standards/uvm.

[2] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *Software Engineering, IEEE Transactions on*, 16(2):166–182, Feb 1990.

[3] Z. F. Baruch. *Structure of Computer Systems*. U. T. PRES, Cluj-Napoca, Romania, 2002.

[4] J. A. Blome, S. Gupta, S. Feng, S. Mahlke, and D. Bradley. Cost-efficient soft error protection for embedded microprocessors. In *Proceedings of the International Conference On Compilers, Architecture, And Synthesis For Embedded Systems (CASES)*, pages 421 – 431, Seoul, Korea, October 23 25 2006. ACM.

[5] E. Dubrova. *Fault-Tolerant Design*. Springer, 2013.

[6] O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante. *Software-Implemented Hardware Fault Tolerance*. Springer, 233 Spring Street, New York, NY 10013, USA, 2006.

[7] F. A. Guzman Figueroa. Natalius 8 bit RISC processor. Opencores. http://opencores.org/project,natalius_8bit_risc.

[8] D. S. Khudia, G. Wright, and S. Mahlke. Efficient soft error protection for commodity embedded microprocessors using profile information. In *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded System (LCTES'12)*, pages 99 – 108, New York, USA, December 6 2012. ACM.

[9] P. K. Lala. *An Introduction to Logic Circuit Testing*. Morgan & Claypool, 2009.

[10] P. K. Lala. Transient and permanent fault injection in VHDL description of digital circuits. *Circuits and Systems*, 3:192 – 199, 2012.

[11] J. Nurmi. *Processor Design System-on-Chip Computing for ASICs and FPGAs*. Springer, P.O. Box 17, 3300 AA Dordrecht, The Netherlands, 2007.

[12] T. L. Srinidhi and M. Devanathan. Efficient diagnostic method for detecting the distinguished and indistinguished faults. *International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)*, 2:284 – 288, March 2013.

[13] H.R. Zarandi, S.G. Miremadi, and A. Ejlali. Fault injection into verilog models for dependability evaluation of digital systems. In *Parallel and Distributed Computing, 2003. Proceedings. Second International Symposium on*, pages 281–287, Oct 2003.

[14] H. Ziade, R. Ayoubi, and R. Velazco. A survey on fault injection techniques. *The International Arab Journal of Information Technology*, 1(2):171 – 186, July 2004.

ISBN: 978-1-4799-5679-1 ©2015 IEEE