Fault Injection Analysis on a Reed Solomon Decoder

M M Dhanvijay
Lecturer,
MMIT ,Department of Electronics and Telecommunicaton Engineering, University of Pune
Pune,India

ABSTRACT

This paper shows improvement in the efficiency of self checking circuit with the use of exhaustive fault injection with analysis of property of circuit is shown. Experimental results from fault injection on a Reed-Solomon Decoder demonstrated that by observing the occurred errors and the correspondent detection module has been possible to reduce the number of detection module, while paying a small reduction of the percentage of SEUs that can be detected.

General Terms

Single Event Upsets,simulation.

Keywords

Reed Solomon Codes, SEUs, Fault injection, Reed Solomon decoder

1. INTRODUCTION

Reed-Solomon (RS) codes are having the ability to correct single upsets per coded word with reduced area and performance overhead and are widely used for protecting memories against Single Event Upsets (SEUs). However, a SEU within the RS decoder can give a wrong data word even if no errors occurs during the codeword transmission. Therefore, also the RS decoder must be designed to with fault tolerance capabilities in order to realize high reliable systems. When the analysis of SEUs is the major concern ,simulation-based fault injection approaches allow early evaluation of the system dependability when only the model is available. However, considering the large complexity of such circuitry, a huge amount of CPU time may be required, thus limiting its usability to exhaustive fault tolerance capabilities evaluations. Instead, we used a partially reconfiguration based fault injection technique able to run in a fraction of the time simulation-based approaches require, supporting the execution of exhaustive fault injection campaigns.

2. FAULT INJECTION

The fault injection system we developed is composed by: a host computer; an FPGA board equipped with a Virtex II-Pro device, and a serial communication link to the host computer. The host computer is used for configuring the Virtex-II Pro, for the generation of a fault location list and to collect the results in terms of fault-effect classification.

2.1 Architectural Scheme

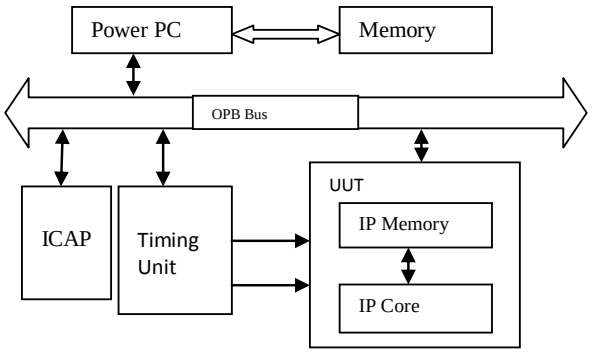

Fig. 1 Architectural scheme of proposed fault injection approach

The FPGA board is composed of four components interconnected by an On-chip Peripheral Bus (OPB) and its layout is depicted in Figure 1.

• Timing Unit (TU): it drives the UUT clock and reset. Aport connected to the OPB Bus defines its functionality.

• Unit Under Test (UUT): it is the circuit under test. Its input and output ports are connected to the OPB while the reset and clock signals are connected to the TU.

• ICAP: it is the Internal Configuration Access Port provided by last generations of Xilinx FPGAs. It allows the access to all the memory elements (Flip-Flops or Latches)
of the UUT to perform the partial reconfiguration.

• PowerPC microprocessor: it is hardwired in the FPGA device, performs the fault injection of SEUs in the UUT and communicates the fault-injection experiment results
to the host computer.

2.2 Fault Injection Execution Flow

The fault injection execution flow is a two-phase process composed of a preliminary phase followed by an execution phase.

The preliminary phase is illustrated in Figure 2. This phase is automatically executed by the host computer using either internal developed tools and commercial tools provided by Xilinx.

During this phase the Unit Under Test is inserted within the FPGA device layout circuit description. This actions is performed by the UUT Wrapper Inserting Tool that gene-rates a UUT wrapper inserted within the FPGA device layout description. This wrapper links the input and output ports with the OPB Bus and the clock and reset signals to the Timing Unit.

The FPGA bitstream is obtained following the FPGA implementation tools chain provided by Xilinx. In particular the BITGEN tool is used to obtain the bitstream that is loaded within the FPGA configuration memory and to generate a Logic Allocation file that contains a list of all the logic resources used within the FPGA.

The Fault Location List Generator reads the Logic Allocation file and generates a Fault Location List where each fault location is characterized by an identifier of the correspondent flip-flop or latches position in the FPGA.

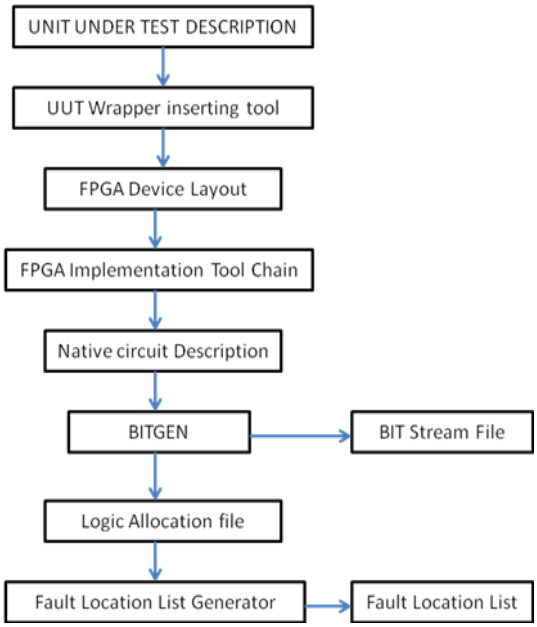

Fig. 2 Preliminary phase of fault injection execution flow

The execution phase is executed by the PowerPC and it consists of three parts: pre-running, campaign and fault injection results. At first, it loads within the PowerPC memory the test patterns that will be applied and initializes the UUT. Secondly, it performs a golden run of the UUT storing the Golden Output (GO) produced.

2.3 Injection of Each SEU

The Campaign performs the fault injection of the selected number of faults (NF). The following steps are executed for the injection of each SEU:

- The UUT is reseted.
- A fault injection time (FT) and a fault location (FL) are randomly selected.
- The execution of the UUT is started until reached the clock cycle FT. This operation is performed by configuring a Timing Unit's counter at the FT value.
- The fault location FL is read. This procedure reads directly the content of the flip-flop or latches from the configuration memory using the ICAP port.
- The FPGA is partially reconfigured writing the opposite value within the content of the flip-flop or latches identified by FL. Therefore a SEU is injected in the considered fault location.
- The execution of the UUT is continued until the end of the run. It monitors the UUT output ports and comparing their value with the UUT golden outputs. It finally updates a fault classification list (FCL) with the results obtained by the fault injection and classifying each injected SEU as silent, if the output produced by the UUT are equal to the GO; wrong answer, if a mismatch was detected.

3. FAULT DETECTION

An RS(n,k) code [4] is defined by representing the data symbols as elements of the Galois Field GF(2m) and the over-all data word is treated as a polynomial d(x) with coefficient in GF(2m). The Reed-Solomon codeword is then generated by using the generator polynomial g(x). All valid codewords are exactly divisible by g(x). The general form of g(x) is:

$g(x) = (x + \alpha i)(x + \alpha i+1) \ldots (x + \alpha i+2t)$

where $2t = n - k$ and α is a primitive element of the field. Therefore the codeword is a polynomial c(x) of degree n−1 such as c(x) mod g(x) = 0 .

3.1 RS Decoder

The RS decoder is able to correct up to t errors in a received word providing as output the corrected codeword. Two main properties can be defined for a fault free RS decoder .

Property 1: The decoder output is always a codeword.

Property 2: The Hamming distance between the received word and the output codeword is not greater than t.

Where the Hamming distance of two polynomials a(x) and b(x) of degree n is the number of coefficients of the same degree that are different. When the fault inside the decoder is activated, i.e. the output is different from the correct one due to the presence of the fault, two cases occur:

- The decoder gives as output a non codeword, and this case can be detected by property 1.
- If the output of the faulty decoder is a wrong codeword the detection of this fault is easily performed by evaluating the Hamming weight of the error polynomial e(x).

The error polynomial can be provided by the decoder as an additional output or can be evaluated by comparing the received polynomial and the provided output c(x).

4. SELF CHECKING RS DECODER

This approach is completely independent by the assumed fault set and it is based only on the assumption that the fault free behaviour of the decoder provides always a codeword as output. To check if properties 1 and 2 are respected some blocks can be added to the decoder. They are:

• An optional error polynomial recovery block is needed if the decoder do not provides at the output the error polynomial.

• Hamming weight counter, that checks the Hamming distance between the received word and the output word of the RS decoder.

• Codeword checker, that checks if the output data of the RS decoder form a correct codeword.

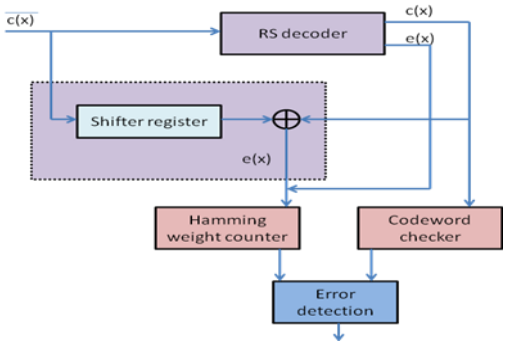

Fig. 3 Schematic of the self checking RS decoder

The codeword checker block checks if c(x) is exactly divisible for the generator polynomial g(x).

4.1 Syndrome Calculation

The syndrome calculation performs the evaluation of the received polynomial c(x) for the values of x that are roots of g(x). The received polynomial is a codeword if and only if all the computed syndromes are zero. Figure 4 shows the implementation of the syndrome calculation block.

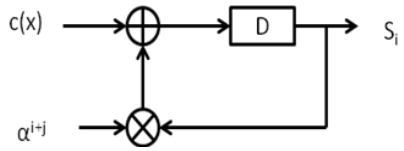

Fig.4 Syndrome calculation block

These blocks allow to detect a fault inside the RS decoder without any knowledge on the implementation details of he decoder. However, is possible that some of the blocks presented in this section are not really needed for a certain
implementation of the decoder. The fault injection experiments of the following section allows to identify which errors can occurs on the decoder output when a fault occurs inside it and herefore which blocks are really needed for fault detection.

5. EXPERIMENTATION

To perform the fault injection campaign an RS(255,239) decoder with the additional blocks described in the previous section has been implemented. The syndrome calculation block is composed by 16 elementary blocks described in
Fig. 4. For the fault injection campaign we call this vector S = S0...S15. When a fault inside the decoder is activated and property 1 is not respected albeit one byte of the syndrome vector is different from zero. The fault injection campaign has
been performed injecting in two runs.

5.1 First Run

During the first run of the fault injection experiment 1,000,000 SEU are injected and the outputs of the syndrome computation block has been monitored. Monitoring these outputs, we are able to identify if the fault is activated and if the erroneous output produced by the decoder violates properties 1 or 2. If all the bytes of the syndrome vectors are equal to zero, then the erroneous output violates property 2, else it violates property one. The results achieved from the fault injection campaign report that the number of activated faults is 6,873,492 and the number of faults detected by the RS decoder is 6,873,310 which results in a fault detection of about 99.9974%. This implies that excluding the Hamming counter from the scheme of the self-checking RS decoder shown in Figure 3.1 however we obtain a fault detection coverage of the RS decoder that is about 99.9974%.

5.2 Second Run

In the second run 100,000 SEU are injected and the 16 bytes of the syndrome Vector has been monitored to check which bytes are able to detect an activated fault. The aim of these run is to find a subset of the syndrome elements that detect a high percentage of fault. The overall activated faults was 98174 .

6. RESULT AND DISCUSSION

The fault detected by any single element of the syndrome vector S are reported in table I. The S1 element of the syndrome vector detects most of the SEUs provoking faults. Therefore use only the block computing S1 the system is able to detect 92% of the activated fault. Instead, using the subset composed only by S1 and S3 we are able to detect all the activated SEU faults. The fault injection experiments show that excluding the Hamming Distance Counter Block and using only the blocks computing S1 and S3 in the codeword Checker we are able to detect about 99.9974 % percentage of faults. In this case we save the logic resources needed to implement the Hamming Distance Counter Block and 14 syndrome element computation block. Instead, using also the Hamming Distance Counter Block we are able to cope 100%.

TABLE I
FAULT COVERAGE OF SYNDROME VECTOR ELEMENTS

Syndrome element	Detected SEU's	[%]	Syndrome element	Detected SEU's	[%]
S0	7825	8	S1	90186	92
S2	41783	42	S3	84994	86
S4	27550	28	S5	12145	12
S6	42177	42	S7	2114	2
S8	1134	1	S9	3811	4
S10	37268	38	S11	981	1
S12	6534	6	S13	719	0.7
S14	289	0.3	S15	119	0.1

7. CONCLUSIONS

The result presented here shown how some blocks added to the Reed Solomon decoder due to high level considerations about the decoder functionality are not really necessary to obtain an high fault detection coverage. The Hamming Counter block can be excluded from the self-checking scheme with a very small reduction of the fault tolerance capabilities. Moreover, a detailed analysis of the syndrome vector outputs allows to reduce also the blocks computing the syndrome only
to a small subset, without penalty in terms of percentage of detected faults. This methodology to improve the efficiency of self-checking structures is possible only if an extensive fault injection campaign can be performed in a fast and flexible
environment. The fault injection technique used in this seminar is a good candidate for this task, because is able to perform injection campaigns in a fraction of the time simulation-based approaches require.

8. ACKNOWLEDGMENTS

I am thankful to my guide Mr. S. M. Deokar and HOD Mr. Lokhande for their cooperation and support.
T am thankful to my kids and husband.

9. REFERENCES

[1] Hyunman Chang; Myung H. Sunwoo, "A low complexity Reed-Solomon architecture using the Euclid's algorithm" Circuits and Systems, 1999.ISCAS '99. Proceedings of the 1999 IEEE International Symposium on Volume 1, 30 May-2 June 1999 Page(s):513 - 516 vol.1

[2] Syed Shahzad Shah, Saqib Yaqub, and Faisal Suleman, Self-correcting codes conquer noise Part 2: Reed-Solomon codecs, Chameleon Logics, (Part 1: Viterbi Codecs), 2001.

[3] G.C. Cardarilli, A. Leandri, P. Marinucci, M. Ottavi, S. Pontarelli, M. Re,A. Salsano, "Design of a fault tolerant solid state mass memory", IEEE Transactions on Reliability, Vol. 52, Issue 4, Dec. 2004, pp. 476 – 491

[4] R.E. Blahut, "Theory and Practice of Error Control Codes", Addison-Wesley Publishing Company,1983.

[5] Manual, *Stratix Device Handbook*. May 2005.

[6] Hanho Lee, *A high-speed, low-complexity reed-solomon decoder for optical communications.*. IEEE Transactions on Circuits and Systems II, PP, 2005.

[7] Hanho Lee *High-Speed VLSI Architecture for Parallel Reed–Solomon Decoder* IEEE Transaction on very large scale integration (VLSI) systems, Vol. 11, No. 2, April 2003.

[8] Andr´e S¨ulflow Rolf Drechsler *Modeling a Fully Scalable Reed-Solomon Encoder/Decoder over GF(pm) in System C* Proceedings of the 37th International Symposium on Multiple-Valued Logic (ISMVL'07) 0-7695-2831-7/07, 2007.

[9] S. S. Shah, S. Yaqub, and F. Suleman. *Self-correctin codesconquer noise part2: Reed-solomon codecs*. *EDN*, March 15, 2001.

[10] B. Sklar. *Digital Communications: Fundamentals and Applications*. Prentice-Hall, 2001.