An Analysis of Fault Effects and Propagations in ZPU: the World's Smallest 32 bit CPU *

Mahroo Zandrahimi[1], Hamid R. Zarandi[1, 2], Alireza Rohani[1]
[1]Department of Computer Engineering and Information Technology, Amirkabir University
[2]School of Computer Science, Institute for Research in Fundamental Sciences (IPM)
E-mail: {mzand, h_zarandi, alirezarohani}@aut.ac.ir

Abstract

This paper presents an analysis of the effects and propagations of transient faults by simulation-based fault injection into the ZPU processor. This analysis is done by injecting 5800 transient faults into the main components of ZPU processor that is described in VHDL language. The sensitivity level of various points of ZPU processor such as PC, SP, IR, Controller, and ALU against fault manifestation is considered and evaluated. The behavior of ZPU processor against injected faults is reported. Besides, it is shown that about 50.25% of faults are recovered during simulation time; 46.47% of faults are effective and the remainders 3.28% of faults are latent. Moreover, a comparison of the behavior of ZPU processor in fault injection experiments against some common microprocessors is done. The results will be used in the future research for proposing a fault-tolerant mechanism for ZPU processor.

Keywords

Fault injection, Transient fault, Simulation-based

1. Introduction

In recent years reliability has become the major factor in designing microprocessor and microcontroller architectures [1]. After critical bugs in Pentium FDIV in 1994 and Sun Microsystems in early 2000 [2], designers have been concentrating on application of fault-tolerant techniques. In order to do so, they need an estimation of the behavior of designed architecture against various faults.

The first step in designing a fault-tolerant architecture is to analyze the effects and propagations of faults in the designed architecture. One of the best methods to estimate the behavior of architecture against various faults is simulation-based fault injection that describes the architecture in one of HDL languages such as VHDL or VERILOG, and then injects faults into the desired points, and ultimately observes the important targets. Simulation-based fault injection is used in this paper since it has capability of high controllability and observability in comparison with other fault injection methods.

Simulation-based fault injection is used in several works. In [3], an analysis of the effect of transient faults on Algha 21264 and AMD Athlon by simulation-based fault injection is characterized. An analysis of the effects of faults in the cache memories of the SPARC V8 architecture by simulation-based fault injection is reported in [4]. The effects of temporary faults located in the pipeline stages and cache memories are studied in [5]. In [6], fault injection by using heavy ions proves the efficiency of the LEON-FT microprocessor. The behavior of SEUs on the 8051 microcontroller protected by a single error correction code is analyzed in [7]. In [8 and 14] simulation-based fault injection in a 32-bit processor, namely: DP32 and an arithmetic processor called ARP are done and the effect of faults in the various points of these processors is reported. An analysis of the effects and propagation of faults in the 32-bit OpenRisc1200 microprocessor by simulation-based fault injection is done in [9]. Finally, an analysis of fault effects and propagations in AVR microcontroller is done in [10].

ZPU as the World's Smallest 32 bit CPU provides a 32 bit core with standard tools. Its flexible core can be used with many FPGA brand/type, memory and I/O systems [11]. The performance of ZPU is measured by DMIPS. This criterion is the popular performance index for deeply embedded processors. The performance of ZPU is between 1 DMIPS to 10 DMIPS depending on configuration and memory subsystem. That great Flexibility and high performance make ZPU one of the best solutions for embedded processors. Since the use of ZPU in embedded-based applications is increasing [11], recognizing effects of different faults in this CPU is mandatory.

The remainder of this paper is composed of four sections. Section 2 describes an overview of ZPU processor architecture; section 3 presents the characteristics of simulation-based fault injection and fault characteristics. Fault injection results and data analysis are presented in section 4 and finally section 5 concludes the paper.

2. ZPU Processor Architecture

ZPU is a small CPU, due to its low intake of resources and its small architecture. The latter feature can be important when learning about CPU architectures and implementing variations of ZPU where aspects of CPU design is examined.

ZPU is a zero operand or stack based CPU in which the opcodes have a fixed width of 8 bits. The choice of opcodes is intimately tied to the GCC toolchain capabilities; moreover, all operations are 32 bit wide.

ZPU supports interrupts. To trigger an interrupt, the interrupt signal must be asserted. ZPU has an edge triggered interrupt. When ZPU notices that the interrupt is asserted, it executes the interrupt instruction. The interrupt signal must stay asserted until the ZPU acknowledges it. When the interrupt instruction is executed, the PC is pushed onto the stack and is set to the interrupt vector address; therefore, stack Pointer (SP) plays a key role in executing instructions.

* This research was in part supported by a grant from IPM. (No. CS1388-4-03)

ZPU implements the minimum instruction set. It is optimized for size and simplicity so that it can serve as a reference in both regards [11 and 13]. All parts of this architecture, including PC, SP and IR are connected to each other through a shared bus.

This architecture uses a BRAM (dual port RAM with read/write to both ports) as data & code storage and is implemented as a simple state machine. Essentially it has four states which are indicated in table I.

ZPU core can generate different configurations depending on the attached peripherals, three of which are explained as follows:

a) Minimal (core + RAM)

The minimum design is a ZPU core with true dual port RAMs attached to it. This could become handy for size/fmax trial in a particular FPGA, and maybe HDL regression. However, it might not be a very useful starting point, unless you can DMA all your IO.

b) Basic (core + RAM + UART + Timer)

The minimum design required for standard benchmarks and DMIPS applications. This could become handy as a starting point for a new users design and for running DMIPS evaluation. It might also be used for HDL regression.

c) SOC (core + RAM + Wishbone)

The SOC design is required for large benchmarks as well as large design(s) for one or more chosen evaluation board. In this configuration, features are dictated by board and available IP. Figure 1 illustrates a typical architecture in this configuration. This paper uses this configuration for executing benchmarks.

ZPU does not currently have floating point support. Feedback from users indicates that single precision floating point support for addition, multiplication and float-to-integer conversion would be useful for small ZPU programs that sit in a tight control loop. It essentially can be useful when ZPU is measuring something which involves a few calculations and then modification of the control signal. Such control loops can be written in fixed point math. However, doing so adds to the engineering effort and reduces clarity of the software implementation and ultimately, the performance will probably be worse than when floating point version is used.

3. Fault Injection Characteristics

The two well-known workloads which are considered in fault injection experiments are namely: Quick Sort and Matrix Multiplication. By using these two well-known workloads, we are able to compare the results of fault injection experiments with other related works. The workloads are coded in C++ and are compiled with GCC toolchain Compiler which can generate machine code for ZPU processor.

SINJECT fault injection tool [12] is used for injecting various fault models into the VHDL description of ZPU [13]. The set of fault injections that are used in fault injection experiments are the same for two workloads.

Table I. ZPU states

State	Task
Fetch	starts fetch of next instruction
FetchNext	sets up operands for execute cycle
Decode	decodes instruction
Execute	executes instruction

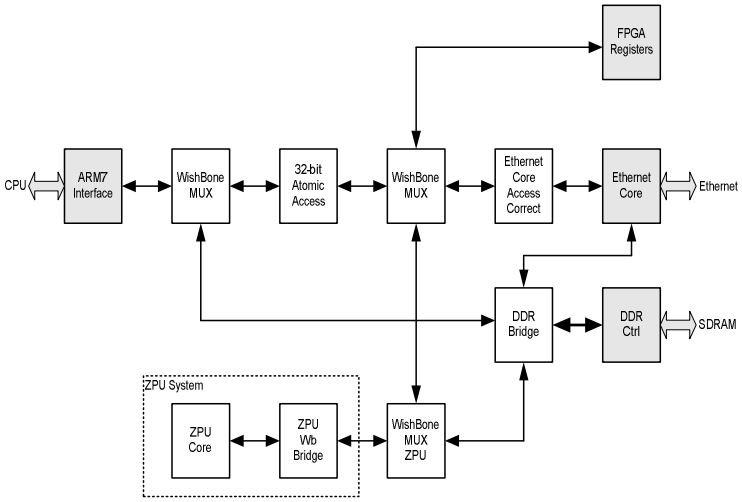

Figure I . SOC architecture for ZPU [11]

Different faults are injected only into the core of the processor in all experiments; however, these faults are not injected in peripheral devices such as UART and timer, because these parts do not play any role in executing implicit workloads. Faults are injected in different parts of the core including PC, SP, IR, Controller and ALU.

Different fault models are considered for various parts of ZPU processor; therefore, the fault injection experiments are comprehensive. Stuck-at fault model is considered for PC, SP and IR; furthermore, Bridging and SEU fault models are considered for these parts. The controller of this processor is described in behavioral level; hence, it is possible to inject Stuck-Then and Stuck-Else fault models in this part of the core. Also Micro-operation fault model is considered for ALU. Table II presents the characteristics of injected faults and their locations. Totally 5800 transient faults are injected into the mentioned parts of ZPU processor core. A 10ns clock period is used for the simulations, and the faults are randomly injected with the average duration of 10ns using exponential distribution. During the 10ms workload execution time, fault injection time is randomly distributed using uniform distribution. The experiments include 116 fault injection points; moreover, 50 transient faults are injected into each point, so totally 5800 fault injection experiments are carried out.

4. Experimental Results

The results of fault injection experiments are presented in this section of the paper. The results are grouped into two subsections. The first subsection describes the effects of injected faults in critical points of ZPU processor such as

PC, SP, I/O bus and etc; furthermore, propagations of injected faults are observed in this subsection. The second subsection presents the behavior of ZPU processor in the presence of injected faults. It is necessary to mention that in this paper a failure is considered as a wrong result which is stored in a proper region.

4.1. Fault Effects in Critical Points

In this part of the paper the propagation of injected faults is monitored, and the percentage of injected faults which reach the critical points of ZPU processor is determined.

Table III illustrates fault latencies and error manifestation in critical points of ZPU processor for each workload. This table shows how fault propagates through critical points of ZPU processor. The first column of this table indicates the observation points for each workload that are similar for all experiments. The next two columns show the minimum and maximum time for error detection in observation points respectively. Column 4 of this table labeled *mean* indicates the average time that an error is observed in an observation point; thus, a low number for this time shows the rapid transformation of injected faults to errors. The last column of table III labeled *error manifestation* includes the percentage of injected faults which produce errors in selected points for each workload. For Quick Sort workload, the percentage of error manifestation is 76 for PC, 72 for SP and 50 for address bus; hence, PC and SP are the most sensitive points for Quick Sort workload. The mean number represented in column 4 of the table is 10700.41ns for PC, 23600.49ns for SP and 15600.61ns for address bus which means that errors are detected first in PC and then in address bus and finally in SP. The mentioned information shows that PC, SP and address bus are the most sensitive points for Quick Sort workload. The same analysis could be done for Matrix Multiplication workload.

Figure 2 helps us distinguish the most sensitive points of ZPU processor for fault-tolerant purposes. This figure illustrates the average error manifestation of two workloads in all selected points of ZPU processor core. It can be understood from this figure that SP, PC and address bus have respectively about 84%, 80% and 67% contribution in transforming faults to errors; moreover, the average mean of two workloads is 61100.49ns for SP, 15600.46ns for PC and 17450.40ns for address bus. Consequently, fault propagation occurs first in PC and then in address bus and finally in SP. It is realized from figure 2 that SP, PC and address bus are the most sensitive points in ZPU processor. Therefore, they must be targeted for fault-tolerant purposes in future researches; otherwise, critical applications which use this processor may face serious damages.

4.2. Behavior of ZPU against Injected Faults

This subsection represents the analysis of two workloads to find out how ZPU processor behaves in the presence of injected faults. The results include the percentage of faults which are recovered in simulation time and the percentage that have no impact on the final output which are called

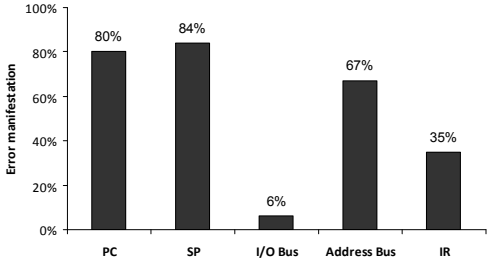

Figure II. Error manifestation in critical points

Table II. The characteristics of the injected faults

Fault injection points	# of injected faults	Fault models
PC	16	Bridging
PC	16	Stuck-at
PC	1	SEU
SP	16	Bridging
SP	16	Stuck-at
SP	1	SEU
IR	16	Bridging
IR	16	Stuck-at
IR	1	SEU
Controller	11	Stuck-Else Stuck-Then
ALU	6	Micro-operation

latent faults. The fault characteristics are all the same as are explained in section 3 of this paper.

Figure 3 shows the behavior of ZPU processor in the presence of faults for each workload. This figure indicates that about an average 7% of injected faults are recovered in the first 2000 clock pulses. The period of a clock pulse is 10ns, yet the fault recovery remains stable as time passes. Besides, this figure shows that about the average 50% of faults are recovered in 30000 cycles after fault activation, and about 35% of injected faults are recovered in the beginning of experiments which means that fault masking has occurred sometime after fault activation. The simulation continues until 10ms while fault recovery remains stable on 50% which shows that about 50% of injected faults are recovered in ZPU processor.

Table IV shows the detailed results of fault injection analysis for each workload. This table shows that about 45.54% of injected faults are recovered during runtime for Quick Sort workload; this number is 54.96% for Matrix Multiplication workload. Moreover, the average recovery coverage of two workloads is illustrated in the last row of this table which can be considered as a metric that shows the behavior of ZPU processor in the presence of faults. Column 2 of this table shows the percentage of faults that could not be recovered during runtime including 1) Latent faults which do not transform to errors during simulation time and 2) Effective faults which cause failures in the system as a wrong final result. The percentage of effective faults is 49.37 for Quick Sort workload and 43.57 for Matrix Multiplication workload since this workload uses SP

Table III. Fault effects, latencies and error manifestations

Parameter Workload	Observation Point	Fault Latency (ns)					manifested Errors (%)
		Minimum	Maximum	Median	Mean	Standard deviation	
Quick Sort	PC	1	14200	29300	10700.41	86900.26	72
	SP	1	41700	22200	23600.49	8800.75	76
	I/O bus	0	174400	0	22000.91	71300.8	5
	Address bus	1	51400	14600	15600.61	89600.74	58
	IR	1	214700	5600	18100.60	74400.96	30
Matrix Multiplication	PC	1	35100	43000	20500.51	79600.10	88
	SP	1	75900	15200	98600.49	9100.76	93
	I/O bus	0	439000	1010	23900.11	74800.15	7
	Address bus	1	78500	23000	19300.20	78100.41	77
	IR	1	63500	4900	25300.50	59900.26	40

Table IV. Behavior of faults in ZPU processor

Workload	Recovery Coverage (%)	Not Recovered Faults (%)		Average of Fault Recovery (ns)
		Latent Faults	Effective Faults	
Quick Sort	45.54	5.09	49.37	1025
Matrix Multiplication	54.96	1.47	43.57	8107
Average	50.25	3.28	46.47	4566

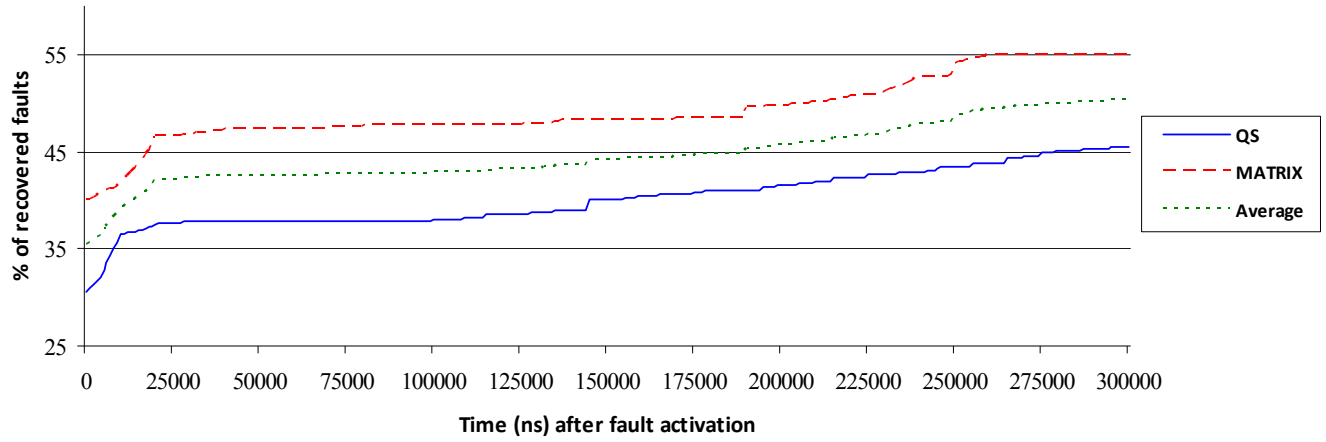

Figure III. Fault recovery in ZPU

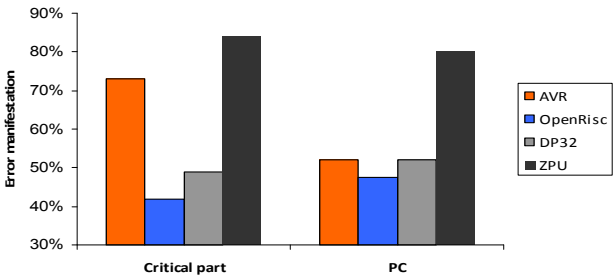

Figure IV. Error manifestation in critical points

extensively which is a sensitive part of ZPU processor as it was already mentioned in previous subsections.It is obvious from table IV that about 46.47% of faults in ZPU processor are effective; therefore, proposing a fault-tolerant architecture for ZPU processor to reduce this number is essential.

5. Conclusions

An analysis of the effects and propagations of injecting various models of transient faults in ZPU processor was done in this paper. A simulation-based fault injection tool named SINJECT was used for injecting faults into the HDL model of ZPU processor. A comparison among various points of ZPU processor against injected faults including PC, SP, IR, Controller and ALU was carried out in this paper. The results indicated that SP, PC and address bus are the most sensitive points of ZPU processor against injected faults; hence, they must be targeted for fault- tolerant purposes. Furthermore, through the analysis of the behavior of ZPU processor, it was revealed that about 50.25% of injected faults were recovered during simulation time. Also, 46.47% of faults were effective which caused failure as a final wrong result in ZPU processor; 3.28% of faults were latent which had no impact on final results during simulation time. As simulation-based fault injection method and Quick Sort and Matrix Multiplication workloads were used in analyzing the behavior of OpenRisc1200 microprocessor, DP32 microprocessor, AVR microcontroller and ZPU processor against faults, a comparison among all four mentioned microprocessors in error manifestation for critical point and PC was made possible (illustrated in figure 4). The comparison has shown that error manifestation in ZPU critical point is much higher than three other microprocessors; thus, it should be considered a fault-tolerant mechanism in ZPU for high-computational workloads. Also, figure 4 demonstrates that the sensitivity of PC in ZPU is much higher than the other three microprocessors which means that this processor is not suitable for sequential workloads which contain infrequent loops and branches.

9. References

[1] G. T. Subramanian, "Control Cashing: A Fault-tolerant Architecture for SEU Mitigation in Microprocessor Control Logic," *M.S thesis, Dep. Comp. Eng., Iowa State University*, Ames, Iowa, pp. 2-3, 2006.

[2] R. Baumann, "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends," *In the Proceedings of the Reliability Physics Tutorial Notes,* 2002.

[3] N. J. Wang, J. Quek, T. M. Rafacz and S. J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *In the Proceedings of IEEE International Conference on Dependable Systems and Networks*, pp. 61-70, 2004.

[4] M. Rebaudengo, M. Sonza Reorda and M. Violante, "An Accurate Analysis of the Effects of Soft Errors in the Instruction and Data Caches of a Pipelined Microprocessor," *In the proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 602-607, 2003.

[5] M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Analysis of SEU Effects in a Pipelined Processor," *In the Proceedings of the IEEE international On-Line Testing Workshop*, pp. 206-210, 2002.

[6] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture," *In the Proceedings of the International Conference on Dependable Systems and Networks*, pp. 409-415, 2002.

[7] F. Lima, L. Carro, R. Velazco and R. Reis, "Injecting Multiple Upsets in a SEU Tolerant 8051 Microcontroller", *In the Proceedings of the IEEE International On-Line Testing Workshop*, pp. 194-199, 2002.

[8] H. R. Zarandi, G. Miremadi and A. R. Ejlali, "Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models", *In the Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp, 485-492, 2003.

[9] N. Mehdizadeh, M. Shokrolah-Shirazi and G. Miremadi, "Analyzing Fault Effects in the 32-bit OpenRISC 1200 Microprocessor", *In the Proceedings of the International Conference on Availability, Reliability and Security*, vol. 00, pp. 648-652, 2008.

[10] A. Rohani and H. R. Zarandi, "An Analysis of Fault Effects and Propagation in AVR Microcontroller ATmega103(l)", *In the Proceedings of the International Conferences on Availability, Reliability and Security,* pp. 166-172, 2009.

[11] Zylin Site, www.zylin.com

[12] H. R. Zarandi, G. Miremadi and A. R. Ejlali, "Dependability Analysis Using a Fault Injection Tool Based on Synthesizability of HDL Models", *In the Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp, 485-492, 2003.

[13] Open Cores, *"www.opencores.org",* [accessed: July 2009].

[14] H. R. Zarandi, G. Miremadi, and A. R. Ejlali, "Fault Injection into Verilog Models for Dependability Evaluation of Digital Systems", *In the Proceedings of the International Symposium on Parallel and Distributed Computing,* pp. 281-287,2003.