The IAradSim – IA32 architecture under a high radiation environment simulator

Bartlomiej SWIERCZ, Dariusz MAKOWSKI, Andrzej NAPIERALSKI

Technical University of Lodz, Department of Microelectronics and Computer Science
Al. Politechniki 11, 93-590 Lodz, Poland, swierczu@dmcs.p.lodz.pl

ABSTRACT

The issue of designing hardened systems is crucial for modern physics and space applications. High energy particles (e.g. neutrons) can affect memory and registers in microprocessor systems and cause an illegal operation and, thus the loss of the system functionality. The fault model is the single transient bit-flip, which is known as a Single Event Upset (SEU). Hardened devices are designed to minimize the radiation impact on the system with hardware redundancy. Software redundancy that should reduce designing and testing cost of the main control system, is recommended to design instead of hardware. This paper presents the IAradSim software simulator of the Intel IA-32 architecture, which imitate behaviour of the electronic equipment (e.g. memory, CPU) working under a high radiation influence. Moreover, the IAradSim enables developers to debug the system easily.

Key words: Single Event Effect, Single Event Upset, Total Ionizing Dose, Radiation Environment, IA-32 Architecture, The Fault Tolerant System, Hardened.

1. INTRODUCTION

A complex distributed system is used to control the X-Ray Free Electron Laser (X-FEL) accelerator [5]. The system is based on a multiplatform environment which consists of SPARC Workstations, dedicated DSP and FPGA boards and VME technology. The total length of the accelerator's equipment will be approximately 3.3 km with dense concentration of electronic devices [4]. Many of that devices will be placed in the accelerator's tunnel and subjected to the neutrons and gamma influence [1, 2]. Neutron radiation is responsible for generating SEU effects in electronic devices [3] and therefore causes the system malfunction. One of possible methods of preventing SEU effect is to design electronic devices by using hardened parts and hardware redundancy that are immune to radiation [6]. Because of the system complexity and a high cost of dedicated solutions, radiation hardened devices can not be used. Radiation tolerant software algorithms and redundancy will be applied to design radiation immune system instead of hardware.

All devices are controlled by Distributed Object Oriented Control System DOOCS, which should be protected against faults. DOOCS is a collection of libraries. DOOCS would be fault tolerant without a modification if it run on fault tolerant radiation resistant operating system.

Traditional methods of developing software systems are insufficient for designing fault tolerant system. Software development requires various analyzes and testes. The analyzes should be carried out in a specific environment to imitate a destination one. Moreover, a performance test under the radiation environment is difficult to execute because the examined system can not be observed directly. When a system is researched under the radiation environment and it breaks down the information about event is only given without any information about a reason of fault. Standard tools, such as a debugger, can not be used to debug the application because the debugger works also on the same platform which is open to radiation impact. The situation is worse when the operating system is tested (it works on the lowest software level).

The IAradSim simulator was investigated to simplify the design of radiation tolerant software. The IAradSim emulates the PC architecture with the Intel IA-32 processor family and it makes possible to simulate the radiation impact on PC. The behaviour of the radiation impact is characterized in Python dynamic scripting language. The Description of the radiation environment can be easy modified without the recompiling of IAradSim.

The standard PC architecture, which is cheap and popular (also in a industrial market), was chosen to carry out research on the protection of the radiation impact.

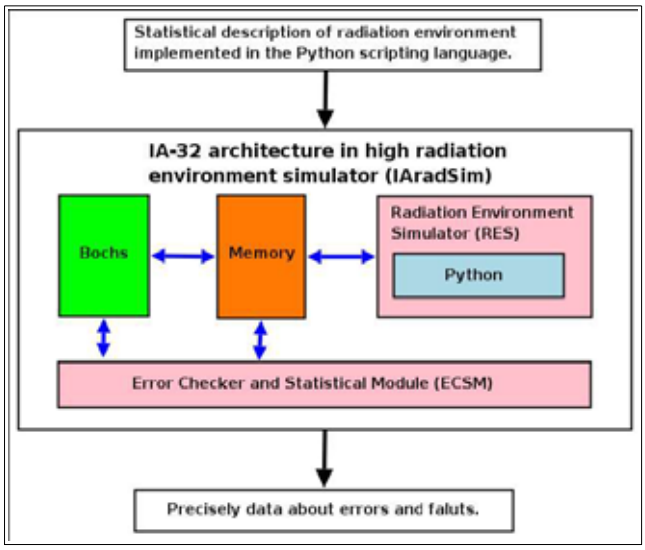

Figure 1: The structure of IAradSim.

2. THE STRUCTURE OF IAradSim

The IAradSim is a software simulator of the Intel IA-32 architecture. The principle of operation is based on Bochs' simulator [7]. Bochs was modified to imitate the behaviour of electronic equipment (e.g. memory, CPU, FPU) working under the radiation influence. The structure of IAradSim simulator is shown on figure 1. The IAradSim consists of two independent parts: Bochs simulator and Python's modules. Each part is connected by a shared memory.

2.1 Bochs

Bochs [7] is a software's PC architecture emulator. It virtulize the IA-32 processor (and also AMD64 CPU) with all peripheral devices: graphics, keyboard, mouse, mass storages, input/output ports, network card. Bochs is able to simulate a multiprocessor (SMP) system but IAradSim supports only single a processor now. Bochs is a open source project, distributed under the GNU GPL license.

Bochs uses an internal full software-realized processor to execute the program. The software's emulation of CPU is relatively slow but it allows to control every step of simulation, which is important for the developers of operating systems.

Bochs has also an internal debugger and is able to connect with an external debugger such as GNU GDB.

Two parts of the Bochs subsystem, memory and CPU objects are required to modify to do IAradSim project.

2.2 Python's modules

Python is used to describe the behaviour of the radiation environment and to manipulate hardware faults in a simulated system. This part of IAradSim is called the Radiation Environment Simulator RES (figure 1).

2.3 An internal communication inside the IAradSim

The Radiation Environment Simulator and Bochs (figure 1) are independent processes. They communicate using a shared memory. A large region of the memory (all memory in a virtualizing system) is required to share by IAradSim components. A standard mechanism of the Inter-Process Communication allows to share (in Linux system) only 4MB of memory, which is to not enough for IAradSim.

The IAradSim uses memory-maped files. This technique allows to share a large region of memory between processes is an efficient way to exchange data.

3. SIMULATION

In this section a hypothetical simulation will be described. The Real-Time Micro-Kernel sCore was used to exemplify the virtual control system.

After started sCore inside Bochs (figure 2), internal Bochs' debugger (table 1) is able to be used to control the simulated system.

```
<bochs:2> print-stack
  00301fec [00301fec]  301ffc
  00301ff0 [00301ff0]  202bf4
  00301ff4 [00301ff4]  202be8
```

Table 1: Actual stack information.

Figure 2: Micro-Kernel sCore running in Bochs.

When Bochs is running, the second part of IAradSim Radiation Environment Simulator might be used to investigate the influence of radiation in a simulated system. The RES system is initiated by importing the python's module IARADPY and creating an instance of class RADSIM (table 2).

.

```
Python 2.3.4 (#2, Jan  5 2005, 08:24:51)
[GCC 3.3.5 (Debian 1:3.3.5-5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import iaradpy
>>> rad = iaradpy.radsim (128)
>>> rad.connect ()
>>> rad.mem
array([32, 18,  8, ...,  0,  0,  0], type=Int8)
```

Table 2: Initialization of IAradSim environment.

The Simulated radiation environment can by described after the initiation of the RES module. Fore example, it is possible to overwrite an actual stack area that is read by using internal Bochs' debugger and change a return address (table 3).

```
>>> for i in range (10): rad.mem[0x301fec+i] = 0x0
```

Table 3: Overwriting stack region.

Changing the function's return address has an unforeseen consequence but it may occur frequently in target system working in the radiation environment. After

this modification function returned to random place of memory. In the presented case the function is returned to address 0x0 and the exception is thrown. The system sCore printed screen of death with information about general protection fault (figure 3).

The reason of crash can by analyzed after system breakdown. All system memory and processor's registers (table 4) might be read by the researcher.

Figure 3: Breakdown of sCore

The IAradSim, apart from the manual simulation supports an automatic simulation for collecting statistical data about system faults. Python's scripts are necessary to automatize the simulation and redirect Bochs' output to a file (table 5).

<bochs:9> ptime
ptime: 4373177083
<bochs:10> dump_cpu
eax:0x300400; ebx:0x0; ecx:0xb8e26; edx:0xb8ec6; ebp:0x301472
esi:0x0; edi:0x0; esp:0x30146e; eflags:0x2; eip:0x201fbc
cs:s=0x8, dl=0xffff, dh=0xcf9a00, valid=1
ss:s=0x10, dl=0xffff, dh=0xcf9200, valid=7
ds:s=0x10, dl=0xffff, dh=0xcf9300, valid=7
es:s=0x10, dl=0xffff, dh=0xcf9300, valid=1
fs:s=0x0, dl=0x0, dh=0x0, valid=0
gs:s=0x0, dl=0x0, dh=0x0, valid=0
ldtr:s=0x0, dl=0x0, dh=0x0, valid=0
tr:s=0x28, dl=0x4ac00068, dh=0x8920, valid=1
gdtr:base=0x204980, limit=0x2f
idtr:base=0x0, limit=0x7f8

Table 4: Information about state the of the system after breakdown. A manual, step-by-step, simulation.

The described technique of simulation is flexible and allows to simulate different computer systems. Moreover, many variants of hardware's faults can be simulated and the influence of them for software's system can be researched.

A software simulation has many advantages in comparison to hardware. The radiation source is not required and the running system can be debugged step-by-step. Furthermore, the last state of the simulated system before the breakdown is easy to restore.

04373177083i[ ] dbg: Quit
04373177083i[CPU0] protected mode
04373177083i[CPU0] CS.d_b = 32 bit
04373177083i[CPU0] SS.d_b = 32 bit
04373177083i[CPU0] | EAX=00300400 EBX=00000000 ECX=000b8e26 EDX=000b8ec6
04373177083i[CPU0] | ESP=0030146e EBP=00301472 ESI=00000000 EDI=00000000
04373177083i[CPU0] | IOPL=0 NV UP DI PL NZ NA PO NC
04373177083i[CPU0] | SEG selector base limit G D
04373177083i[CPU0] | SEG sltr(index|ti|rpl) base limit G D
04373177083i[CPU0] | DS:0010(0002| 0| 0) 00000000 000fffff 1 1
04373177083i[CPU0] | ES:0010(0002| 0| 0) 00000000 000fffff 1 1
04373177083i[CPU0] | FS:0000(0000| 0| 0) 00000000 000fffff 1 1
04373177083i[CPU0] | GS:0000(0000| 0| 0) 00000000 000fffff 1 1
04373177083i[CPU0] | SS:0010(0002| 0| 0) 00000000 000fffff 1 1
04373177083i[CPU0] | CS:0008(0001| 0| 0) 00000000 000fffff 1 1
04373177083i[CPU0] | EIP=00201fbc (00201fbc)
04373177083i[CPU0] | CR0=0x60000011 CR1=0x00000000 CR2=0x00000000
04373177083i[CPU0] | CR3=0x00000000 CR4=0x00000000

Table 5: Information about the last state of the system, batch simulation.

4. FUTURE WORK

Further research should concern improvements of the fault detection mechanism. A system breakdown is difficult to detect. For example, the system memory might be corrupted and the program executes an infinite loop. Normal program executes are indistinguishable from the abnormal. The implementation of a watchdog for the IAradSim seems to be the best solution. If simulated system does not reset watchdog the IAradSim should interrupt the simulation and dump the last systems state.

The mechanism of an automation system state analysis should be provided. It ought to be able to understand debug symbols and marks labels instead of addresses (e.g. useful to analysis back trace of stack).

The interface to other computer's peripherals, such as north and south bridge, network card and others, is planned to be developed. Every peripheral device has internal registers that are sensitive to radiation influence (SEU effect).

CONCLUSIONS

This paper presents a novel technique to simulate and test a fault tolerant software system. The IAradSim allows to develop a radiation resistant system without difficult and expensive research in a target environment. By applying the Bochs module the IAradSim enables to debug all parts of the simulated system. Moreover, it enables to dump registers and memory after system breaks down.

The main aim of developing the IAradSim was cost reduction of fault tolerant system research. The IAradSim also reduces the total time of development through delivery of sufficient information about systems faults. It is straightforward to restore a systems state before breakdown.

The IAradSim is intended to be used to develop a fault tolerant software system for X-FEL research project.

ACKNOWLEDGMENTS

We acknowledge the support of the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" program (CARE, contract number RII3-CT-2003-506395).

We also thank Bochs' open source community for development of a PC architecture simulator.

REFERENCES

[1] D. Makowski, B. Mukherjee, M. Grecki and S. Simrock, “SEE Induced in SRAM operating in a Superconducting Electron Linear Accelerator Environment”, XIV IEEE-SPIE, 2004.

[2] H.P. Chou, T.C. Chou and T.H. Hau, “Evaluation of high density DRAMs as a nuclear radiation detector”, Applied Radiation and Isotopes, Volume 48, Issues, pages 1601-1604, 1997.

[3] R. J. Peterson, ”Radiation-induced Errors in Memory Chips”, Brazilian Journal of Physics, vol.33 no.2, 2003.

[4] R. Brinkmann, K. Flöttmann, J. Roßbach, P. Schmüser, N. Walker and H. Weise, ”TESLA Technical Design Report - The Accelerator, part II”, DESY, 2001.

[5] G. Materlik and Th. Tschentscher, “TESLA Technical Design Report - The X-Ray Free Electron Laser, PART V”, DESY, 2001.

[6] Actel - White Paper, ”Effects of Neutrons on Programmable Logic”, 2002.

[7] http://bochs.sf.net