AN HDL SIMULATION OF THE EFFECTS OF SINGLE EVENT UPSETS ON MICROPROCESSOR PROGRAM FLOW

Authors:[1]
K. W. Li
J. R. Armstrong
J. G. Tront

Department of Electrical Engineering
Virginia Tech
Blacksburg, VA 24061

ABSTRACT

Simulation experiments for determining the effects of single event upsets on microprocessor program flow are described. A 16 bit microprocessor is modeled using a hardware description language. Using pseudorandom selection of event time and effected flip-flop, SEU's are injected into the microprocessor model. Upset detectors are modeled along with the microprocessor for determination of fault coverage of several candidate fault detection techniques.

INTRODUCTION

Due to the limited signal observability of VLSI devices, direct laboratory testing of microprocessors in a radiation environment yields only gross indication of the effect that single event upsets have on microprocessor program flow [1]. In this paper, we present the results of simulations that were used to assess the effects that single event upsets have on microprocessor program flow and also the effectiveness of upset detection mechanisms.

The microprocessor that was modeled was the Texas Instrument SBR9000, a radiation hardened, I^2L (2 μm) version of the SBP9989. The SBR9000 was chosen as the candidate 16 bit architecture for this study due to the fact that Texas Instruments made available to the authors a detailed HDL description of this microprocessor.

THE SIMULATION MODEL

The intent of the simulation was to model the execution of a microprocessor's program in the presence of upsets. In order to get meaningful results the simulation must model the execution of a program whose dynamic instruction mix [2] reflects the actual system application. The system application that we modeled was that of a satellite control program. This control program consisted of a main loop whose primary function was to wait for interrupts which would transfer execution to the mission software. In the simulation model, a 50 instruction sequence was used whose dynamic mix matched the dynamic mix of the application program. This mix took into account not only the relative frequency of instructions within a given routine but also the duty cycle of execution of that routine within the structure of the program [3]. Since this execution sequence was repeated for every simulated fault, it was essential that the simulation be efficient. Our previous research experience in modeling microprocessor program flow had shown the use of gate level logic simulators to be prohibitively expensive for this application [4], therefore, it was necessary to employ a functional level model in the simulation experiments. The functional level model for the microprocessor in our studies was represented in HDL, a hardware description language developed by Texas Instruments. Details of HDL are given in [7].

UPSET DETECTION MECHANISMS

The purpose of our simulation experiment was to assess the effect that transient upsets have on program flow. The term "program flow" refers to the fact that program execution produces a certain track through the address space of the computer. Transient upsets disturb this

---

[1] K. W. Li is with Assurance Technology in Carlisle, MA. J. R. Armstrong and J. G. Tront are with the Department of Electrical Engineering, at VA Tech in Blacksburg, VA. This work was funded by the Naval Research Laboratories under contract FE 178718.

0018-9499/84/1200-1139$1.00©1984 IEEE

track. This type of error is more serious than those which merely alter data words. Also, coding techniques which allow detection and correction of data word errors are already well known [8]. On the other hand, the effectiveness of techniques for detecting transient induced program flow disturbances has not been well established.

Indications of program flow disturbance can be obtained from the normal display capability of the HDL simulator. However, in addition to detecting flow disturbances, the simulation experiments were also used to evaluate upset detector designs. Our previous research had developed a list of potential mechanisms for detecting disturbances in program flow. We now describe each of them individually.

1. Illegal opcode detection

In most microprocessor systems, the processor recognizes only a certain subset of bit combinations as legal opcodes; all other combinations are illegal. Some microprocessors have a built-in illegal opcode detection mechanism, e.g., the Motorola MC68000 and the Texas Instruments SBR9000. When a processor fetches an illegal opcode, it executes an interrupt using the instruction stored in a specific interrupt vector. An appropriate service routine addressed by this vector will then flag the fault.

2. Invalid Opcode Address Detection

When a microprocessor incorrectly fetches an instruction from an invalid opcode address, the content of the address may be data or the second or third word of a multi-byte instruction. If the bit pattern fetched is an illegal opcode, it will be detected by the invalid opcode detection mechanism, which is discussed above. If the bit pattern falls within the range of legal opcode patterns, the processor will execute it as a legal operation, and no error will be detected. This type of error can be detected by applying the "SAFE ROM" [6] technique.

> "The SAFE ROM is a 1-bit wide memory, with the same number of locations as the main ROM. The single added bit is used to distinguish between valid and erroneous instruction fetches. The first byte of multiple byte instructions is marked as valid; all other locations which are addresses of the second or third byte of an instruction, data, or look-up tables are marked as erroneous."

The illegal operations detected by the SAFE ROM technique are:

a. In an opcode fetch cycle, the microprocessor accesses a non-opcode address,

b. In a non-opcode fetch cycle, the microprocessor accesses an opcode address.

3. Invalid read address detection

In a microprocessor system, instructions and look-up tables are stored in ROM while data and variables are stored in RAM. Some memory addresses do not physically exist, and some existing memory locations are not used. Also, if memory mapped I/O [9] is used in the system, some addresses are assigned for input and output. An invalid read address detector should flag an error when an instruction attempts to read a data value from an address in the instruction area, from memory locations used for output, from non-existent memory space, or from unused memory space. The detection mechanism checks the read address. If this address is not within the memory locations assigned to variable storage, data storage, look-up tables or input, the error is detected.

4. Invalid write address detection

The invalid write address detection mechanism detects any attempt to write into a memory location not designated as alterable or not assigned for output. In a microprocessor system, the instructions are stored in ROM, and the data is stored in RAM. Only the RAM area and the output locations are alterable. Any write instruction accessing an address outside the RAM area and the output area is an erroneous operation and the invalid write address detection mechanism detects this.

5. Non-existent memory and unused memory detection

In a microprocessor system, not all the memory locations in the address space physically exist and some existing memory locations are not used by the operational program. Any operations referencing these addresses are erroneous operations and are detected as such.

6. Transient induced loop change detection

In a microprocessor-based control system, the control program is composed of loops. Normal program execution transfers from one loop to another loop in a fixed pattern and the entry point of each loop is predetermined. When a transient fault produces a program flow disturbance, the program flow deviates from the

regular path and the starting point of execution in a loop is not at the desired point [7]. In order to detect the transient induced loop change, each loop is assigned a different tag value and is divided into two parts. The first part is the entry point of the loop, and at this point a register is set to a value equal to the loop tag. The second part is the main body of the loop. Once the looping has started, the program flow quickly bypasses the first part and the execution remains in the second part of the loop. The register storing the loop tag is continually checked for the correct value in the loop. If a transient fault induces a loop change, then the entry point into the loop will most likely fall in the main body of the loop, since this part of the loop contains much more code than the loop entry point section. When this occurs, the register is not updated by the first part of the loop and when the register is checked for the loop tag during the looping, an error is detected.

The loop change detection technique requires software implementation. When applying this technique, the program of the system must be constructed in a loop structure. Details of how to implement a loop change detection technique for a satellite microprocessor system are given in [10].

In addition to the microprocessor model, the simulation model contained a model of a detector containing all of the above features. A major goal of the simulation was to rate the relative effectiveness of the various detection mechanisms in detecting transient upsets.

THE SIMULATION EXPERIMENT

In Figure 1 is shown the diagram of the simulation experiment. The single event upset can be injected by using pseudo-random or deterministic techniques to select a flip-flop internal to the processor and, also the time during each run that the flip-flop's state is to be altered. In our experiments, the time of upsets was always randomly selected. In selecting the critical flip-flop we used random selection initially, but after several hundred runs, subsequent analysis of the logic revealed that due to the nature of the instruction mix simulated (which reflected the application program), it was impossible for transient faults in certain flip-flops to actually effect the computation. Because of this the list of candidate flip-flops was reduced somewhat and random selection was used within this list. While to some observers it might seem that this approach limits the generality of the results, it should be noted that for a given microprocessor in an interrupt driven, real-time control environment, the instruction mix will not vary much from one application to another [2].

SIMULATION RESULTS

In all 1236 transient faults were injected into the simulation model and of these 1236 injected faults only 298 actually disturbed the program flow. Of the 298 cases where the program flow was disturbed, it was important to know which of these 298 program flow disturbances would be detected and by which mechanism. Figure 2 gives this data. The height of each bar in the figure indicates the percentage of failures that were detected by a given detector. The detection mechanisms are numbered as follows: (1) SAFE ROM, (2) invalid write address, (3) invalid read address, (4) loop detector, (5) invalid opcode (6) non-existent and unused memory.

One of the important bits of information not illustrated in Figure 2 is the percentage of flow disturbances detected by some subset of the six detection mechanisms. In fact, very high coverages of flow disturbances can be achieved. Table 1 below shows the percent coverage for different subsets of detector mechanisms. Note that use of detector mechanisms 1-2-3-4-5 can provide 96% coverage of program flow disturbances.

Table 1 also gives the detection latency for the various combinations of mechanisms. Detection latency is the time from injection of the transient fault to detection by a mechanism. For a group of mechanisms the latency figure represents an average for the group.

Also given is the average power in milliwatts to implement a subset of detection mechanisms in low power, Schottky TTL, a circuit technology of interest to our sponsors. Mechanisms listed as requiring zero power are those built into the microprocessor chip or implemented in software. While not in line with the central theme of this paper, we include this information since power is frequently a critical resource in satellite systems. With the information given in Table 1 one can trade off detection level for power consumption. For example, a 96% detection level requires just over a watt of power while an 84% level of detection can be attained with an expenditure of just 20 milliwatts.

In summary, a high percentage (96%) of program flow disturbances can be detected. An analysis of the 4% of cases where the flow disturbances were not detected revealed that they consisted of such things as execution halt, illegal jump to an opcode address within a loop, and skipping of execution cycles.

Of particular interest to the microprocessor designer is the distribution of faults among the processor registers, i.e., which registers were the most sensitive in causing program flow disturbances. Of the 298 faults which caused program flow disturbances, 67% of the faults were injected in the major 16

bit registers: program counter register, workspace register, memory address register and instruction register; 16% of faults were injected in the 8 bit control register JA; 17% were in the other flip-flops. The distribution of error causing faults among registers is given in Table 2.

The value of this information is that if the microprocessor designer knows which registers in the processor architecture cause the most disturbances, provision might be made in the design process to make those registers more resistant to the radiation environment.

CONCLUSIONS

HDL simulations of microprocessor program execution provide an effective means for evaluating the effects that single event upsets have on microprocessor program flow. Of particular value is the ability to rate potential detectors of these disturbances. In fact, the results presented in this paper are the first time that data has been presented which rate the effectiveness of detectors in detecting transients internal to a processor.

It should be emphasized that the effectiveness of some detection mechanisms is dependent upon how the address space of the microprocessor system is layed out, e.g., if most of the address space is utilized by the application program, an unused memory address detector will not be very effective.

The simulations also provide information on which registers cause the most flow disturbances, information which should be of real value to chip designers.

The simulation experiment described in this paper was performed on the Texas Instruments SBR9000, a microprocessor architecture in which there are no arithmetic registers on-board the chip. As a follow on to this effort, similar experiments should be carried out on a microprocessor architecture which has an on board register array.

Finally, in the simulation experiments described here, the simulation model used a single 50 instruction sequence whose dynamic weight matched that of the application program. The execution of this sequence was repeated for each injected fault. It would be interesting to repeat the experiments described here using instruction mixes with the same relative frequency of instruction execution but having a different execution order to see if this would effect the results. Cost considerations inhibited us from doing this in the work described here.

Acknowledgement
The authors wish to thank George Flach of the Naval Research Labs for his enthusiastic support of this work.

References:

1. Price, W.E., Pickel, J.C., Ellis T., Frazee, F.B.,"Cosmic Ray Induced Errors in I^2L Microprocessors and Logic Devices", IEEE Transactions on Nuclear Science, vol. NS-28, No.2, December 1981, pp 3946-3954.
2. Kuck, D.J., "The Structure of Computers and Computation", Wiley and Sons, 1978, pp 288-294.
3. Li, K. W., Detection of Transient Faults in Microprocessors by Means of External Hardware, Masters Thesis, Electrical Engineering Department, Va. Tech., March 1984.
4. Armstrong, J.R. and Devlin, D.E., "GSP- A Logic Simulator for LSI", Proceedings of the 18th Annual Design Automation Conference.,pp 518-524, Nashville, TN, June 1981.
5. Schmid, M.E., Traph, R.L., Davidoff, A. E., and Masson, G.M., "Upset Exposure by Means of Abstraction Verification", Proceedings of FTCS-12, pp 237-244.
6. Glaser, R.E., and Masson, G. M., "The Containment Set Approach to Crash-Proof Microprocessor Controller Design", IEEE Transactions on Computers, Vol. C-31, No. 7, July 1982, pp. 689-692.
7. Johnson, W.,Crowley, J.,Steger, M. and Woolsey,E.,"Mix Level Simulation from a Hierarchical Language", Journal of Digital Systems, vol. 4, Fall 1980, pp 305-336.
8. Siewiorek, D. P. and Swarz, R. S., "The Theory and Practice of Reliable System Design", Digital Press, 1982, pp 84-103.
9. Short, K. L., "Microprocessors and Programmed Logic", Prentice Hall, Inc., 1981, pp 249-287.
10. Tront, J. G., Armstrong, J. R., and Oak, J. V., Interim Report for Bendix Contract FF 189225; "Satellite Digital Subsystem Design and Development.

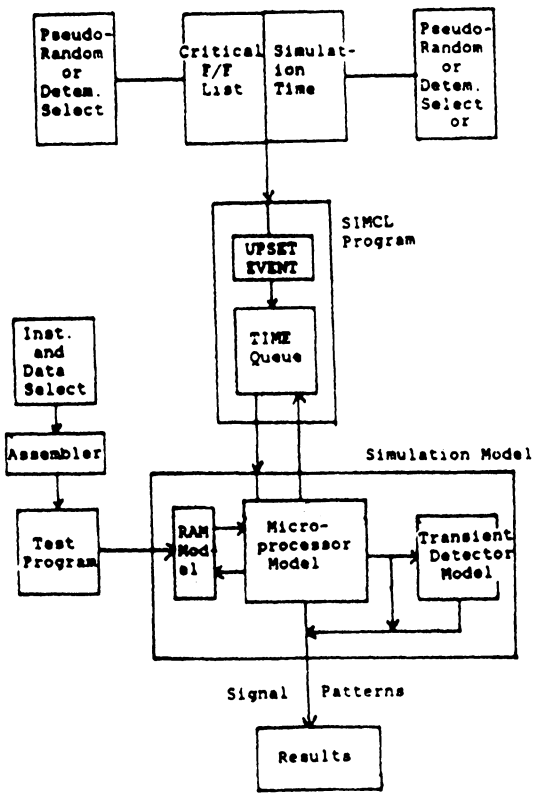

FIGURE 1

Simulation Experiment

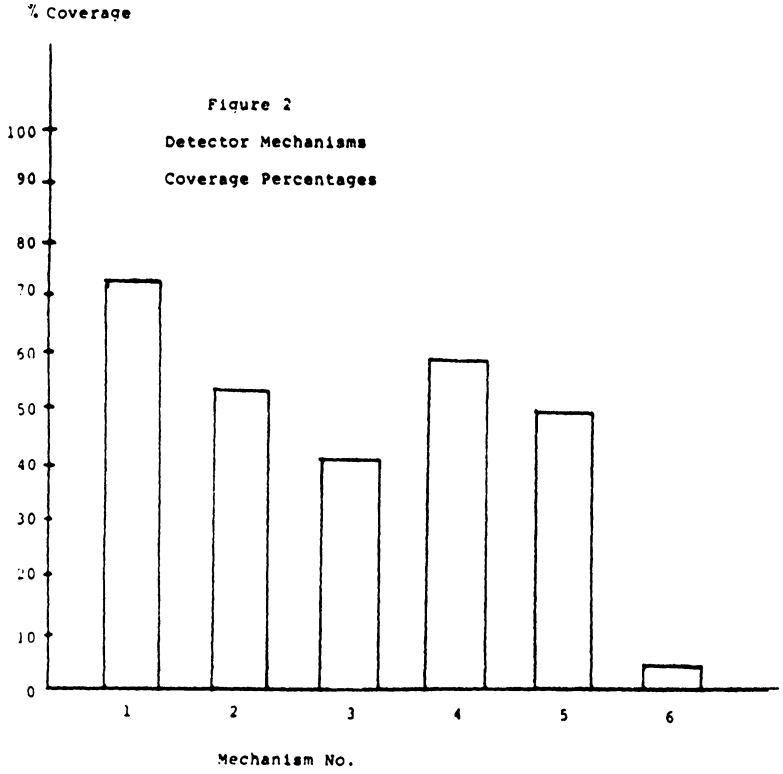

Figure 2

Detector Mechanisms

Coverage Percentages

TABLE 1

COVERAGE, LATENCY AND POWER CONSUMPTION OF THE MECHANISMS

MECHANISMS	COVERAGE %	LATENCY μsec	AVE. POWER mw.
1	69.4	9.6	476.6
2	51.0	8.5	20.6
3	38.6	9.8	576.4
4	62.4	10.1	0
5	47.2	42.2	0
6	2.9	5.4	14.0
1-2	86.9	7.6	497.2
1-3	78.8	7.2	1053.0
1-4	75.2	9.3	476.6
1-5	73.1	9.8	476.6
2-3	71.8	7.8	597.0
2-4	82.6	7.9	20.6
2-5	72.5	11.4	20.6
3-4	74.8	7.5	576.6
3-5	67.8	17.9	576.4
4-5	66.4	11.0	0
1-2-3	93.6	6.4	1073.6
1-2-4	89.5	7.8	497.2
1-2-5	87.9	7.7	497.2
1-3-4	82.5	6.8	1053.0
1-3-5	80.8	7.4	1053.0
1-4-5	76.1	9.6	476.6
2-3-4	90.9	6.4	597.0
2-3-5	86.6	9.6	597.0
2-4-5	84.6	8.0	20.6
3-4-5	77.9	8.4	576.4
1-2-3-4	95.0	6.2	1073.6
1-2-3-5	94.0	6.4	1073.6
1-2-4-5	90.0	7.9	497.2
1-3-4-5	83.2	7.1	1053.0
2-3-4-5	94.0	6.4	597.0
1-2-3-4-5	96.0	6.2	1073.6

TABLE 2

DISTRIBUTION OF ERROR CAUSING FAULTS AMONG REGISTERS

REGISTER (BIT)		% OF ERROR CAUSING FAULT
PROGRAM COUNTER	PC (16)	36%
WORKSPACE REGISTER	WP (16)	17%
MEMORY ADDRESS REGISTER	MA (16)	7%
INSTRUCTION REGISTER	IR (16)	7%
CONTROL REGISTER	JA (8)	16%
OTHERS		17%