Instruction-Level Impact Comparison of RT- vs. Gate-Level Faults in a Modern Microprocessor Controller

Michail Maniatakos*, Naghmeh Karimi†, Chandra Tirumurti‡, Abhijit Jas§ and Yiorgos Makris*

*Department of Electrical Engineering, Yale University
†Department of Electrical and Computer Engineering, University of Tehran
‡Strategic CAD Labs - Intel Corporation
§Validation and Test Solutions - Intel Corporation

Abstract

We discuss the results of an extensive fault simulation study involving the control logic of a modern Alpha-like microprocessor. In this comparative study, faults are injected in both the RT- and the Gate-Level description of the design and are simulated under actual workload of the microprocessor, which is executing SPEC2000 benchmarks. The objective of this study is to analyze and contrast the impact of RT- and Gate-Level faults on the instruction execution flow of the microprocessor. The key observation is a pronounced consistency in the type and frequency of Instruction Level Errors (ILEs) arising due to RT- vs. Gate-Level faults. The motivation for this work stems from the need to understand the relative importance of low-level faults based on their instruction-level impact, in order to appropriately allocate error detection and/or correction resources. Hence, the consistency revealed through this study implies that such decisions can be made equally effective based on RT-Level fault simulation results, as with their far more computationally-expensive Gate-Level equivalents.

1. Introduction

Concurrent Error Detection (CED) [1], [2], [3] has recently enjoyed revived interest as a possible solution to various factors threatening the reliable operation of modern microprocessors, including not only Single Event Upsets (SEUs), but also design marginalities, Negative Bias Temperature Instability (NBTI), coupling, power supply noise, etc. [4], [5]. CED methods, however, incur area, power, and performance overhead and cannot be applied across the board. Rather, the limited budget available for CED features needs to be judiciously allocated. To this end, a method for assessing the relative importance of low-level faults with regards to their impact on the instruction-level execution of a typical microprocessor workload and allocating CED resources accordingly could prove invaluable.

Such information may, possibly, be obtained via extensive fault simulations. Performing these simulations at the Gate-Level could provide highly detailed and accurate information; yet it may be computationally prohibitive and rather late in the design cycle to effectively use this information to develop appropriate CED methods. Indeed, working at higher levels of abstraction, especially at the RT-Level, is preferred among designers, not only because they are faster to simulate but also because they are conceptually easier to follow, allow rapid prototyping of functions, offer portability/extendibility to similar/larger designs, and enable exploration of potential collateral benefits. However, obtaining the aforementioned information via RT-Level simulations raises the perennial question of accuracy when higher abstraction levels are used [6], [7].

Towards supporting efficient allocation of CED resources early in the design cycle, in this paper we investigate the accuracy of information obtained through RT-Level fault simulations. More specifically, we contrast the instruction-level impact caused by faults in the RT-Level description of a modern microprocessor controller, to that caused by faults in its Gate-Level equivalent. Our study focuses on faults in the control logic of modern microprocessors with advanced architectural features, for which CED solutions have recently started to appear [8], [9]. The underlying microprocessor model and the mechanism for capturing instruction-level impact of low-level faults are described in Section 2. Details of the target control modules are given in Section 3, while the RT- and Gate-Level fault injection and simulation capabilities developed for this study are discussed in Section 4. Results are presented and analyzed in Section 5, while conclusions are drawn and possible extensions are pinpointed in Section 6.

2. Study Framework

Among the very limited choice of publicly available microprocessor models with modern architectural features, we chose to work with IVM (Illinois Verilog Model) [10], [11]. This Verilog model resembles the functionality of an Alpha 21264 microprocessor, including out-of-order execution, 12-stage pipeline, hybrid branch prediction, speculative execution, etc. IVM complex control logic supports up to 132 instructions in-flight.

A key feature of IVM and our reason for choosing to work with it is the ability to simulate real workload (i.e. SPEC2000 benchmarks). Since the objective of our study is to contrast the instruction level impact of RT- vs. Gate-Level faults, being able to simulate actual workload rather than random traces boosts significantly our confidence in the obtained results.

In order to capture the instruction level impact of low-level faults, we employ the concept of Instruction Level Errors (ILEs). Specifically, we use the ILE Types defined in [12], which we summarize in Table 1. Fig. 1 shows an example, where a low-level fault (i.e. a bit-flip at the output of an inverter) causes an instruction to use a different register than the one intended (i.e. an ILE of Type 3). ILEs are not mutually exclusive (i.e. more than one ILE Type can be caused simulta-

1093-0167/09 $25.00 © 2009 IEEE
DOI 10.1109/VTS.2009.32

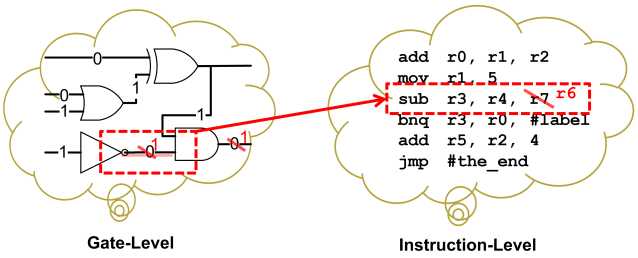

Fig. 1. Instruction level impact of low-level fault

neously by a low-level fault) and do not reflect every possible instruction level error, yet they provide a comprehensive basis for capturing incorrect behavior in the instruction execution flow. For proper classification of a low-level fault into one or more ILE Type, information is collected from various parts of the microprocessor, including the Scheduler, the ReOrder Buffer, the Execution Units, the Scoreboard etc., as shown in Table 2. In other words, the complete microprocessor model needs to be simulated to ensure accurate classification. Details about ILEs and the classification process can be found in [12].

The key limitation of the IVM version which supports simulation of actual workload is that it is not synthesizable, thus limiting our logic and fault simulation options to the RT-Level. Therefore, in order to perform a comparative study of the instruction-level impact of RT- vs. Gate-Level faults, we converted the target modules to synthesizable, incorporated their Gate-Level version in the RT-Level model of the microprocessor, and created the necessary infrastructure for using an RT-Level logic simulator to perform fault simulation of both their RT- and Gate-Level versions.

3. Target Modules

Since our focus is on microprocessor control logic, we target two key control modules: the Scheduler, which controls the allocation of instructions to execution units, and the ReOrder Buffer (ROB) which controls the order of instruction retirement. Both modules incorporate large buffers to support the number of instructions that can be in-flight, with a combined total of over 40,000 storage elements, as shown in Table 3. The Scheduler is relatively small; it contains 32 slots for instructions waiting to be executed and keeps the information needed to identify and correctly issue an instruction. The ROB is much larger because it contains a 64-slot instruction buffer, as well as complementary information about instruction retirement order.

In order to perform a comparative RT- vs. Gate-Level analysis, we converted the Verilog description of these two modules to a synthesizable version and employed Synopsys Design Compiler to synthesize them and obtain their Gate-Level description. The synthesized Scheduler consists of 170,099 standard cells, while the synthesized ROB consists of 228,881 standard cells. Even though the ROB has over 20K (228%) more storage elements than the Scheduler, it only uses 34% more standard cells. This is explained since, despite the fact that the ROB uses much larger buffers, the control logic involved is rather small and is only limited to the proper retirement of instructions. On the other hand, the Scheduler performs

Group 1: Operation Errors	**Type 1:**	Incorrect (yet valid) operation code used
	Type 2:	Invalid operation code used
Group 2: Operand Errors	**Type 3:**	Incorrect (yet valid) register addressed
	Type 4:	Invalid register addressed
	Type 5:	Premature use of register contents
	Type 6:	Incorrect immediate operand used
Group 3: Execution Errors	**Type 7:**	Incorrect functional unit type utilized
	Type 8:	Multiple functional units utilized
Group 4: Timing Errors	**Type 9:**	Early commencement
	Type 10:	Late or no commencement
	Type 11:	Longer duration
	Type 12:	Shorter duration
Group 5: Order Errors	**Type 13:**	Commitment order violation

TABLE 1. Instruction-Level Errors

Module	Information
Decoder	Opcode, Operands
Rename	Physical Registers
Execution	Functional Unit Utilization
Scheduler	Issue Time, Replays
Scoreboard	Availability of Registers
Reorder Buffer (ROB)	ROBid, Retirement Cycle/Order

TABLE 2. ILE classification information traced from various microprocessor modules

complicated tasks such as checking whether operands are ready, whether an instruction can be issued avoiding structural hazards etc., which involve much more control logic. This observation implies diversity, since both control logic-heavy and buffer-heavy modules are considered in the study. Finally, using an industrial-strength tool, we performed fault enumeration and collapsing on the two target modules and we provide the results in Table 4. All faults, including faults within the cells, are considered.

4. Fault Simulation

With the exception of the Scheduler and the ROB modules, for which we created synthesizable versions, the rest of the IVM microprocessor is not synthesizable; therefore, Gate-Level fault simulation tools cannot be used in our study. Even if the complete processor was synthesizable, fault-simulating the entire Gate-Level model would probably be impractical. Hence, we are limited to using RT-Level logic simulation tools, such as Synopsys VCS. This enables simulation of the RT-Level model of the microprocessor while it executes actual workload, but it does not offer fault injection capabilities. In this section, we describe the methods that we employ in order to inject faults in the RT- and Gate-Level models of the target modules, while performing RT-Level logic simulation. These methods have been fully automated through in-house software.

4.1. RT-Level Fault Injection

In order to support RT-Level fault injection, an extra module called *Fault Controller* is added to control all fault injection parameters, and the RT-Level model of each target module is mutated. Injection is performed in every storage element defined in the Verilog model, as shown in Table 3. Fault injection during RT-Level fault simulation is performed using

Module	# of Storage Elements
Scheduler	9,411
ReOrder Buffer (ROB)	30,735

TABLE 3. Statistics on RT-Level Modules

Fault Type	Scheduler	ReOrder Buffer
All faults	1,159,012	1,714,306
Unique faults	679,833	1,074,850
Untestable faults	320	19,845

TABLE 4. Statistics on Gate-Level Modules

a method similar to the parallel saboteurs technique described in [13]. Fig. 2 presents a simplified diagram of the method, which is capable of injecting either stuck-at faults, with user-defined start and stop times and duration (lighter color indicates hardware resources added for fault simulation purposes). Since we operate at the RT-Level model, only storage elements are fault-injected. Each storage element is driven by a `MUX` which is controlled by the Fault Controller. The latter also provides an additional *fault clock* signal which alters the value of the target storage element during the active fault injection window. Each storage element has a unique ID, so that the Fault Controller can pick the one to be injected each time, while the rest hold their value. The fault clock is different than the regular clock, with fault injection performed when it has a value of 1. Otherwise, the value set by the Fault Controller is ignored.

4.2. Gate-Level Fault Injection

Gate-Level fault injection during RT-Level fault simulation is also supported through addition of a *Fault Controller* module and mutation of the Gate-Level description of the target module. A simplified version of the method is depicted in Fig. 3, with lighter color indicating hardware resources added for Gate-Level fault injection. In order to avoid cluttering the figure, we only show the resources for 3 out of the 10 fault injection sites; the rest of the sites are injected in the exact same way. In this simulation environment every wire has an additional driver, which is controlled by the Fault Controller. In contrast to the RT-Level, we inject faults on wires rather than storage elements, hence we do not need a fault clock. Cells are treated as black boxes. The Fault Controller drives a high-impedance value `z` whenever a wire should not be fault-injected, so that the normal value of the wire is propagated. During fault-injection, a `supply0` or a `supply1` value is driven on the wire, resembling a stuck-at-0 or a stuck-at-1 fault, respectively. According to Verilog definition, if a wire has multiple drivers the strongest one prevails, with `z` being a neutral value. `Supply0` and `supply1` are the strongest signals, overwriting the regular `0` or `1` value of a signal and, thus, injecting a stuck-at fault.

We point out that additional buffers are used in the diagram of Fig. 3, in order to enable fault-injection in the various segments of a wire with fanout. These buffers enable individual addressing and, thus, fault-injection on each of the branches of a fanout net. Finally, we also note that in order to successfully simulate a Gate-Level module in an RT-Level environment, the delays of all the standard cells are set to zero, matching the default zero propagation delay of an RT-Level model.

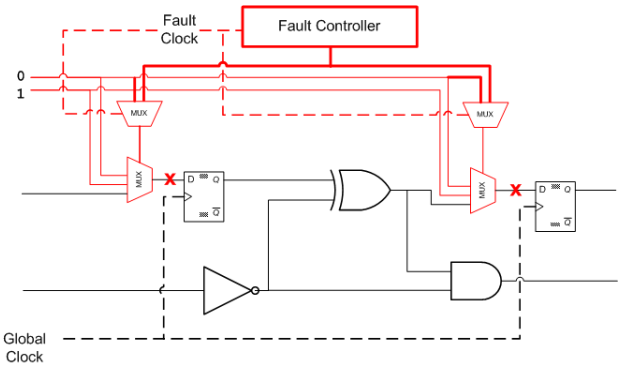

Fig. 2. Fault injection in latches of RT-Level model

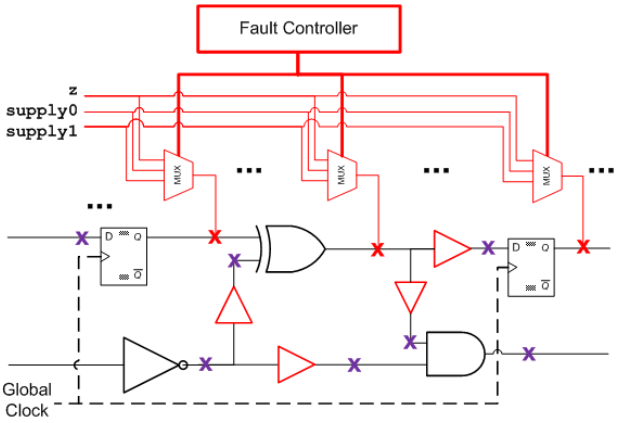

Fig. 3. Fault injection in wires of Gate-Level model (resources for 3 out of 10 fault injection sites shown)

5. Experimental Results

In this section, we outline the simulation setup used for our study and we present and discuss the obtained results.

5.1. Simulation Setup

Simulation Workload: Three different SPEC2000 benchmarks, namely `bzip2`, `mcf`, and `cc`, are used as the simulation workload for the IVM processor. The use of multiple benchmarks ensures variability on the instructions executed through the processor and the control logic that they exercise. Each benchmark is executed in the fault-free microprocessor for an initial warm-up period of 50,000 clock cycles, at which point a fault is injected and the execution continues for 2,000 cycles on the faulty microprocessor. On average, 1,297 instructions retire during benchmark execution.

Target Modules & Types of Injected Faults: We report results individually for the Scheduler and the ROB. We remind that the complete microprocessor needs to be simulated in order to accurately assess the instruction level impact of a low-level fault. Therefore, we employ the fault injection methods described in Section 4, which perform RT-Level simulation of the microprocessor model while employing either the RT- or the Gate-Level view of the target module. In both abstraction levels, the types of faults injected after the warm-up period are stuck-at faults. This choice is driven by the fact that

such injected faults resemble errors occurring due to design marginalities and in-field failures. Furthermore, their permanent nature increases the probability that they will cause an ILE, and, thereby, decreases the length of the fault simulations necessary to study their impact. We note, however, that the developed fault injection infrastructure also supports simulation of other fault types, such as single-cycle or multi-cycle transients, so a similar (yet much more computationally expensive) study can be performed for those.

Number of Injected Faults: While simulating the RT-Level versions of the two modules, all faults presented in Table 3 are injected. At the Gate-Level, however, the number of faults in Table 4 is very large so we resort to sampling, with a sample size of $c = 10\%$. For the Scheduler, where the total number of faults is $N_{Scheduler} = 1,159,012$, this translates to a sample size of $n_{Scheduler} = 115,901$ faults, while for the ROB, where $N_{ROB} = 1,714,306$ faults, this translates to a sample size of $n_{ROB} = 171,430$ faults. To assess the error incurred due to sampling, we use the following equation, defined in [14]:

$$C_{0.99} = c \pm \epsilon, \epsilon = \frac{a^2 k}{2N_i}\sqrt{1 + \frac{4N_i c(1-c)}{a^2 k}} \quad (1)$$

where ϵ is the incurred error, $C_{0.99}$ is the range of fault coverage within which the true coverage lies with a confidence interval of 99%, N_i is the total number of faults, c is the fraction of faults to be simulated, $a = 2.60$ to achieve a confidence interval of 99%, and $k = 1$ since the total population is large. Thus, our sample size of $c = 10\%$ yields an error of $\epsilon = 0.015\%$, which is adequate for our analysis.

Computational Power: The reported simulation times are obtained on a Quad-core Xeon 3.33GHz with 16GB of memory.

5.2. Results and Discussion

The first set of results investigates how the simulation speed is affected by the use of Gate-Level modules. Table 5 compares the average simulation time per fault for the various configurations. The first row provides the baseline when all modules are simulated at the RT-Level, while the following rows indicate the overhead when one or both of the target modules are simulated at the Gate-Level. As may be observed, this makes the simulation over an order of magnitude slower. Taking into account that, even with 10% sampling, the Gate-Level Scheduler and ROB require 72x and 55x more fault simulations than their RT-Level counterparts, a similar fault analysis at the Gate-Level is 675x more expensive, rendering it almost infeasible for extensive studies. Therefore, performing the simulations at the RT-Level is strongly desired, as long as the accuracy of the obtained results is up to par.

The next set of results compares the accuracy of assessing the impact of low-level faults on instruction execution at the RT- vs. the Gate-Level. Specifically, Fig. 4 presents the classification of RT- and Gate-Level Scheduler faults into the ILE types that they cause, for each of the three benchmarks, based on which we can make the following observations:

Fault Simulated Modules	**Seconds per Fault Simulation**	**Time Overhead** (Compared to RTL)
All RTL modules	3.26	-
Gate-level Scheduler	35.04	10.7x
Gate-level ROB	42.20	12.9x
Gate-level Scheduler & ROB	79.02	24.2x

TABLE 5. Fault simulation speed comparison

- The most important observation is a pronounced consistency in the types and frequency of ILEs caused by RT- vs. Gate-Level faults. Indeed, for each benchmark, the distribution of RT-Level faults to the 13 ILE types is strongly correlated with the distribution of Gate-Level faults to the same (correlation coefficient is 92% for `bzip2`, 92% for `mcf`, and 98% for `cc`). This critical observation corroborates that RT-Level fault analysis provides sufficiently accurate results with respect to its Gate-Level counterpart, yet it is much faster, as revealed by the simulation times of Table 5.
- The next observation is that results are consistent across the various benchmarks, even though each of them utilizes different instructions. The correlation coefficient is 92.85% between the results on `bzip2` and `mcf`, 98.70% between `mcf` and `cc`, and 93.49% and between the results on `mcf` and `cc`. Hence, it is evident that the type of ILE caused by a low-level fault is, typically, independent of the instruction subset that is utilized.
- A final observation is that some ILE types appear never to be caused by Gate-Level faults. These ILE types have already a very small frequency of occurrence due to RT-Level faults, which is further diminished during Gate-Level fault simulation because of sampling.

Similar observations hold true for the data shown in Fig. 5, which presents the results for the ROB. Therein, the correlation coefficient between the types and frequency of ILEs caused by RT- and Gate-Level faults is 94% for `bzip2`, 97% for `mcf`, and 97% for `cc`.

The next set of results compares the time that it takes for an RT- vs. a Gate-Level fault to cause its corresponding ILE. This information is useful since it reflects the window of opportunity for taking action prior to the fault corrupting the architectural state of the processor. Figs. 6 and 7 present the results for each ILE Type and for each of the three benchmarks for the Scheduler and the ROB, respectively, assuming that fault injection occurs at time zero. Based on these, we can make the following observations:

- There is some level of correlation between the appearance times of RT- and Gate-Level faults as an ILE Type. While this correlation does not appear to be as strong as one may expect, we raise caution to the following points that help explain the discrepancy: (i) some ILE Types appear never to be caused by Gate-Level faults due to sampling, as discussed above; yet the appearance times of the RT-Level faults causing the same ILE Type are taken into account, so the results may be slightly skewed, and (ii) faults injected at the RT-Level immediately impact the registers,

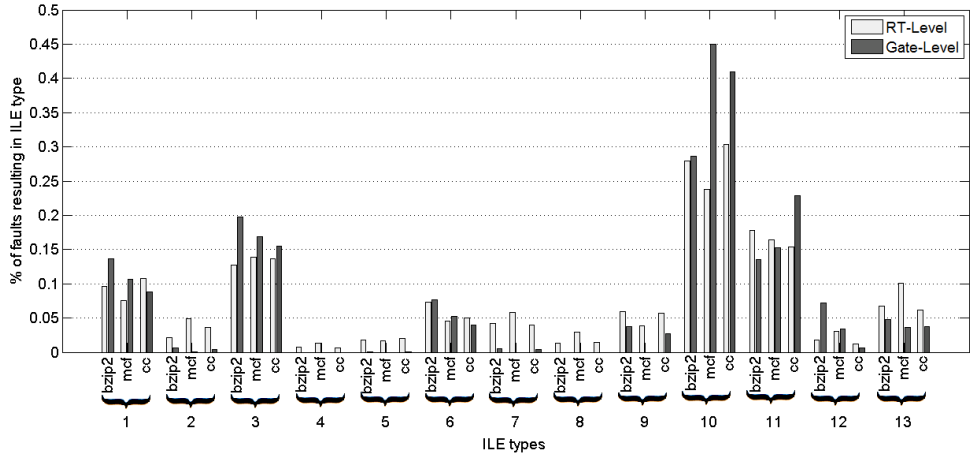

Fig. 4. Comparison between ILE Types caused by RT- vs. Gate-Level faults in the Scheduler

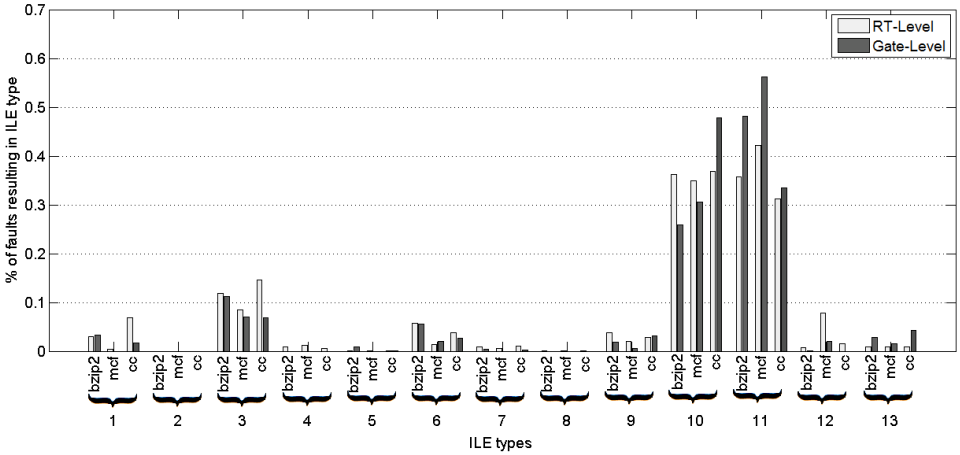

Fig. 5. Comparison between ILE Types caused by RT- vs. Gate-Level faults in the ROB

while faults injected at the Gate-Level have to traverse more logic which may require infrequent combinations of inputs, in order to reach a register.

- More importantly, due to point (ii) above, it is evident that it takes longer for a Gate-Level fault rather than an RT-Level fault to manifest as an ILE. Therefore, defining the window of opportunity for taking preventive/corrective action based on RT-Level fault simulation analysis is a pessimistic approach, whose validity and effectiveness holds for the Gate-Level faults as well.

6. Conclusion & Future Directions

Motivated by the need to develop cost-effective CED methods for the control logic of modern microprocessors, we performed a comparative investigation of the instruction level impact of RT- vs. Gate-Level faults. Our study revealed a very strong correlation in the type and frequency of ILEs caused by RT- and Gate-Level faults, respectively. Furthermore, we observed that the time needed for an injected RT-Level fault to appear as an ILE is a pessimistic estimate of the time it takes for a Gate-Level fault, which typically takes longer to manifest. In addition, we demonstrated that substitution of even a single RT-Level module with its Gate-Level equivalent results in fault simulation which is over an order of magnitude slower per fault, as well as a large increase in the number of faults that need to be simulated to ensure accuracy, even when sampling is employed. Based on these observations, we conclude that instruction-level impact analysis of RT-Level faults provides sufficiently accurate information with regards to the prevalent types of ILEs and the window of opportunity for averting them, yet earlier in the design cycle and at significantly lower computational cost than analysis of Gate-Level faults.

As a continuation of this study, we plan to expend the computational resources needed for fault-simulating the complete Gate-Level fault list as opposed to the current 10% sample, experiment with more SPEC2000 benchmarks to increase the variety of instructions executed by the microprocessor, and convert more control modules into synthesizable versions in order to extend the scope of our results.

Acknowledgements

The described research is supported through a generous gift from Intel Corporation, where it was partially carried out during an internship by the first author. The second author performed this research while being a visiting student at Yale University. The authors would like to thank Prof. Sanjay Patel and Nicholas

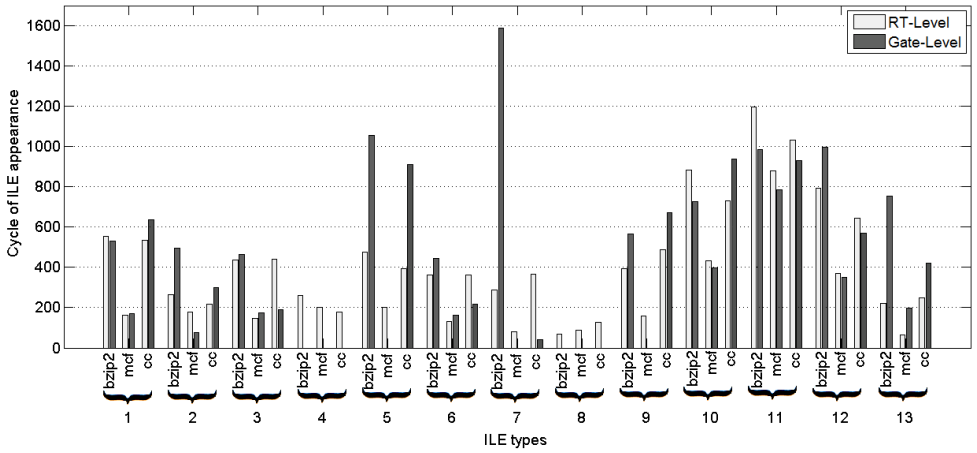

Fig. 6. Comparison of time-to-appearance of RT- vs. Gate-Level faults in the Scheduler as ILE Types

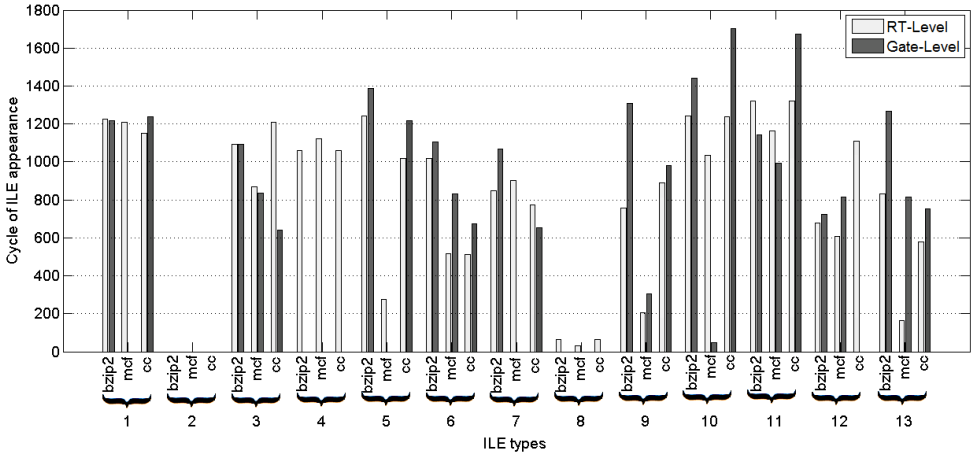

Fig. 7. Comparison of time-to-appearance of RT- vs. Gate-Level faults in the ROB as ILE Types

Wang from the University of Illinois at Urbana-Champaign for providing the IVM microprocessor and technical assistance.

References

[1] M. Goessel and S. Graf, *Error Detection Circuits*, McGraw-Hill, 1993.

[2] S. Mitra and E. J. McCluskey, "Which concurrent error detection scheme to choose?," in *International Test Conference*, 2000, pp. 985–994.

[3] S. Almukhaizim, P. Drineas, and Y. Makris, "Entropy-driven parity-tree selection for low-overhead concurrent error detection in finite state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 8, pp. 1547–1554, 2006.

[4] C. Metra, M. Favalli, and B. Ricco, "On-line detection of logic errors due to crosstalk, delay, and transient faults," in *International Test Conference*, 1998, pp. 524–533.

[5] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in cmos processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[6] F. Corno, G. Cumani, M. Sonza Reorda, and G. Squillero, "An RT-level fault model with high gate level correlation," *IEEE International High-Level Design Validation and Test Workshop*, 2000.

[7] W. Mao and R. K. Gulati, "Improving gate level fault coverage by RTL fault grading," *International Test Conference*, pp. 150–159, 1996.

[8] M. Maniatakos, N. Karimi, Y. Makris, A. Jas, and C. Tirumurti, "Design and evaluation of a timestamp-based concurrent error detection method (CED) in a modern microprocessor controller," *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pp. 454–462, Oct. 2008.

[9] C. Metra, Rossi D., Omana M., Jas A., and Galivanche R., "Function-inherent code checking: A new low cost on-line testing approach for high performance microprocessor control logic," *IEEE European Test Symposium*, pp. 171–176, 2008.

[10] 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 *International Conference on Dependable Systems and Networks*, 2004, pp. 61–70.

[11] N. J. Wang and S. J. Patel, "Restore: symptom based soft error detection in microprocessors," in *International Conference on Dependable Systems and Networks*, 2005, pp. 30–39.

[12] N. Karimi, M. Maniatakos, Y. Makris, and A. Jas, "On the correlation between controller faults and instruction-level errors in modern microprocessors," *International Test Conference*, pp. 24.1.1–24.1.10, 2008.

[13] J. C. Baraza, J. Gracia, S. Blanc, D. Gil, and P. J. Gil, "Enhancement of fault injection techniques based on the modification of VHDL code," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 693–706, 2008.

[14] D. Agrawal and H. Kato, "Fault sampling revisited," *IEEE Design and Test*, vol. 7, no. 4, pp. 32–35, 1990.