Collection and Analysis of Microprocessor Design Errors

David Van Campenhout
Verisity Design

John P. Hayes
University of Michigan

Trevor Mudge
University of Michigan

Research on practical design verification techniques has long been impeded by the lack of published, detailed error data. We have systematically collected design error data over the last few years from a number of academic microprocessor design projects. We analyzed this data and report on the lessons learned in the collection effort.

With transistor budgets ever expanding, microprocessor architects are steadily integrating new and more sophisticated mechanisms into their designs to boost performance. To cope with this increase in complexity, successful processor verification efforts must employ a variety of complementary verification technologies to achieve an acceptable level of functional correctness in the final product. Research on practical verification techniques for microprocessors has long been impeded by the lack of published error data, despite the abundance of design errors in large-scale projects. It is common industry practice to record design errors, but this information is considered proprietary and, perhaps, embarrassing, so it rarely appears in public. Detailed error data is especially valuable to verification approaches that use error models to direct test generation.[1,2] Furthermore, sets of designs and corresponding errors can serve as benchmarks to compare different verification methods. Finally, statistical reliability analysis methods rely heavily on this type of data.[3]

These considerations led us to turn to academia as a source of error data from microprocessor design. We first report on the modest amount of error data published by industry. We then describe our method to systematically collect design error data from university design projects. After presenting and analyzing the data we collected, we offer some advice on the collection process based on lessons painfully learned.

Industrial Error Data

Although design errors that make their way into final products are common, microprocessor manufacturers have not always been forthcoming about them. This has changed since MIPS Technologies Inc., Mountain View, Calif., began to publish its bug list.[4] The notorious Pentium FDIV bug also influenced this change.[5] To give an idea of these errors, we discuss a few examples of design errors that have appeared in major commercial microprocessors and the data published about them.

The errata list for the MIPS R4000PC and R4000SC microprocessors (revisions prior to revision 3.0) documents 55 bugs.[4] Many of these require a rare combination of events

0740-7475/00/$10.00 © 2000 IEEE

```
lw        // data cache miss
nop       // one or two nop
jr        // last instruction in the page
------    // page boundary
nop       // first instruction (delay slot of jump)
          // on the next page
```

Figure 1. Example of instruction sequence that exposes an error.

before they become visible. Referring to Figure 1, the following is a representative bug: If an instruction sequence that contains a load causing a data cache miss is followed by a jump, and the jump instruction is the last instruction on the page, and further, the delay slot of the jump is not mapped at the time, then the virtual memory (VM) exception vector is incorrectly overwritten by the jump address. The R4000 will use the jump address as the exception vector. The workaround suggested ensures that jump instructions can never be stored in the last location of a page.[4]

Early versions of the Intel 8086 were shipped with the following bug.[6] The architecture specifies that for MOV and POP instructions to a segment register, interrupts are not to be sampled until completion of the following instruction.[7] This feature allows a 32-bit pointer to be loaded to the stack pointer registers SS and SP without the danger of an interrupt occurring between the two loads. However, early versions of the 8086 do not disable interrupts following a MOV to a segment register. This causes them to crash when an interrupt uses the stack between MOV SS, reg and MOV SP, op. A workaround is to insert instructions to temporarily disable the interrupts when reloading SS. An uncorrectable problem occurs when an unmaskable interrupt takes place while executing the instruction pair.

A detailed analysis of the errata lists of the Intel Pentium II microprocessor is presented by Avizienis et al.[8] This work also proposes a taxonomy for the study of design errors. However, these published bug lists are inadequate for error model construction for two reasons. First, the errata lists typically provide only a programmer's view of the errors. Error models depend on the design implementation. Therefore, more detailed information about the errors is required, namely the concrete modification to the implementation that fixes the error. Second, errata lists only concern errors in the final product. Microprocessor companies go to great efforts to validate the functionality of their designs. Those design errors that remain undetected before the product is shipped tend to be very subtle and difficult to detect. The majority of all design errors is detected before reaching the customer, and hence is not documented in errata lists. Consequently, these lists are not representative for the overall population of design errors. These shortcomings spurred us to begin collecting error data from academic design projects in the microprocessor area, quite a few of which can be found at the University of Michigan.

Collection Method

The most suitable point to collect design error data is immediately after the design error is discovered and corrected. At that point, all relevant information about the design error should be recorded. This record-keeping requirement conflicts with the interests of the designer, however. Overhead has to be reduced to a minimum to overcome designers' natural reluctance to cooperate.

Our error collection method uses the revision management program, CVS.[9] This tool supports the archiving of successive revisions to a design as they are created in a hardware design language (HDL) such as Verilog or VHDL. The designers were asked to submit a new revision of their design to CVS whenever a design error was corrected and whenever it interrupted work on the design. Some designers resisted the error collection process because they saw it as a way their work quality could be monitored. We defused this potential problem by providing designers with a handout explaining the use of the revision management system, and by explaining our objectives to obtain the designers' cooperation.

Our first tentative design error collection effort took place during the summer of 1996 and involved a few students engaged in the design of the PUMA research microprocessor. Only the bare revision management system was then in place. Experience with that project motivated the introduction of the handout mentioned here. It was clear that a standardized form was

needed to accompany each revision so that interesting revisions, such as those involving a design error correction, can be separated from other revisions. We therefore augmented the revision management system so that each time a new revision is submitted, the user is prompted to fill out a questionnaire. The questionnaire, in the form of a multiple choice form as shown in Figure 2, gathers 4 key pieces of information: 1) the motivation for revising the design. In the case of a bug, the following apply as well: 2) the method by which the bug was detected, 3) the class to which the bug belongs, 4) a short description of the bug. Design errors can be detected by reading the HDL code specifying the design (inspection), by syntax checking performed by the HDL simulator (compilation) or a synthesis tool (synthesis), or by logic simulation. The operation of our error collection method within the design cycle is illustrated in Figure 3.

From the raw revision management data, we identified the design modifications to fix each error by computing the differences between successive revisions. Analysis of the design error data led to a preliminary classification of design errors. This classification was used in our first major design error collection effort, which took place in the fall of 1996 and involved a design project included in a computer architecture course. Analysis of this design error data led us to revise our error classification scheme. The result is shown in Figure 2. The categories are not completely disjoint, so designers were asked to check all applicable categories.

Data Collected

Design error data was collected from both class design projects and research projects at the University of Michigan. All of the designs were described in Verilog. Table 1 lists these projects. LC2 refers to the design of the Little Computer 2 (LC-2), which is a small microprocessor used for teaching purposes at Michigan.[10] The design of both a behavioral and a synthesizable register-transfer-level model was carried out by a graduate student in the summer of 1997. DLX1, DLX2, and DLX3 refer to design projects that were undertaken as part of the senior/first-year graduate computer architecture course (EECS 470) in the fall of 1996.

```
(Replace the _ with X where
 appropriate)

   MOTIVATION:

   X bug correction
   _ design modification
   _ design continuation
   _ performance optimization
   _ synthesis simplification
   _ documentation

   BUG DETECTED BY:

   _ inspection
   _ compilation
   X simulation
   _ synthesis

   BUG CLASSIFICATION:

   Please try to identify the primary
   source of the error. If in doubt,
   check all categories that apply.

   X combinational logic:

      _ wrong signal source
      x missing input(s)
      _ unconnected (floating) input(s)
      _ unconnected  (floating)  output(s)
      _ conflicting outputs
      _ wrong gate/module type
      _ missing instance of gate/module

   _ sequential logic:

      _ extra latch/flipflop
      _ missing latch/flipflop
      _ extra state
      _ missing state
      _ wrong next state
      _ other  finite  state  machine  error

   _ statement:

      _ if statement
      _ case statement
      _ always statement
      _ declaration
      _ port list of module declaration

   _ expression (RHS of assignment):

      _ missing term/factor
      _ extra term/factor
      _ missing inversion
      _ extra inversion
      _ wrong operator
      _ wrong constant
      _ completely wrong

   _ buses:

      _ wrong bus width
      _ wrong bit order

   _ verilog syntax error

   _ conceptual error

   _ new category (describe below)

   BUG DESCRIPTION:

   Forgot  to select  NOP  in  case  of  stall
```

Figure 2. Example of a bug report.

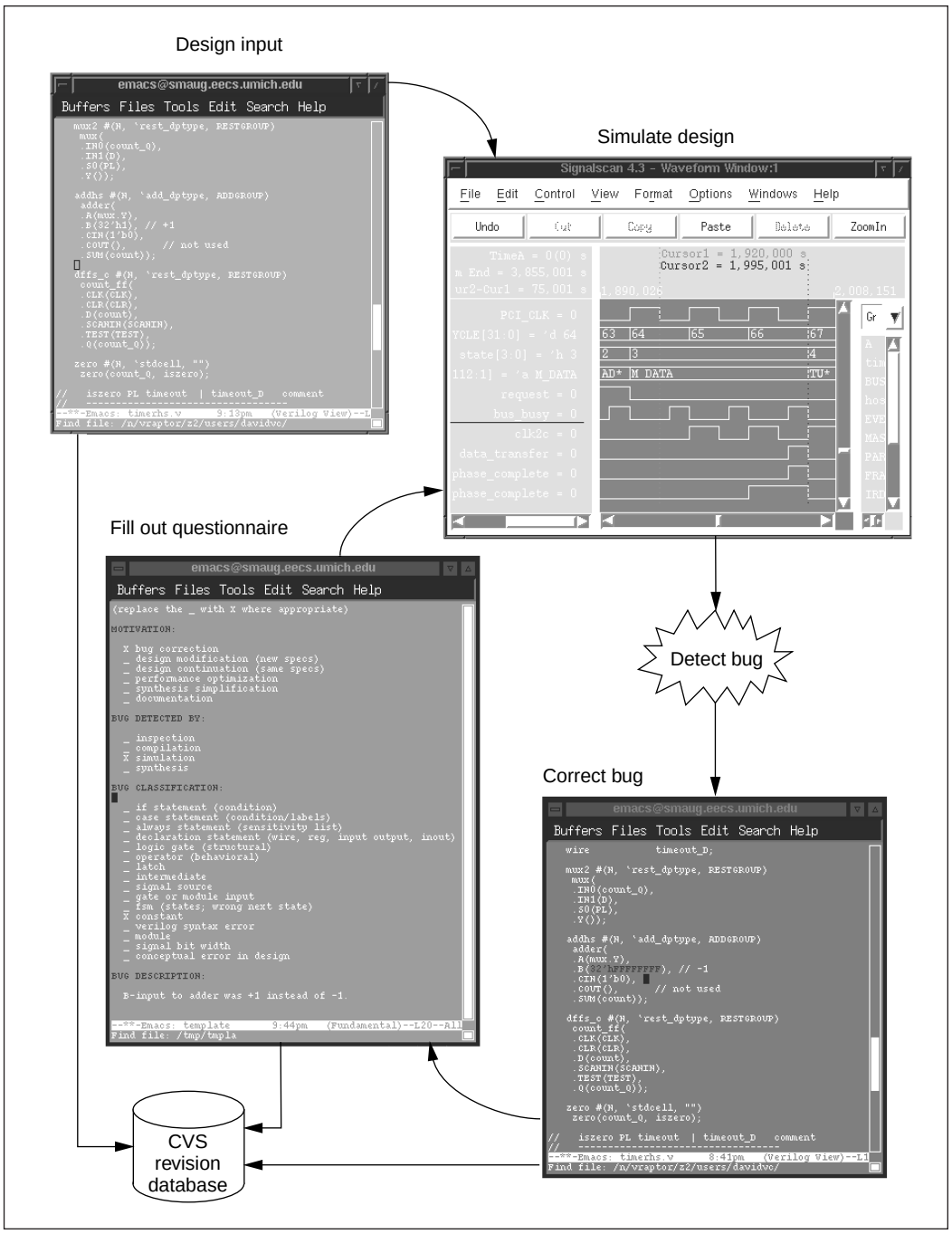

Figure 3. Error collection system.

Students designed a pipelined implementation of the DLX microprocessor at the structural level.[11] X86 concerns an EECS 470 design project carried out in the fall of 1997. Students designed a pipelined implementation of a subset of the Intel X86 architecture.[7] FPU refers to the design of a floating-point unit for the PUMA processor, which is a PowerPC microprocessor implemented in complementary GaAs (gallium arsenide) process technology, and was undertaken as part of the graduate level very large scale integration (VLSI) design class (EECS 627).[12] Both a purely behavioral and a mixed synthesizable behavioral/structural model were designed. FXU concerns the design of the fixed-point unit of the PUMA

Table 1. Design projects for which error data was collected.

Project	Class	Date	Duration (days)	No. of designers	Code size (lines)	No. of errors
LC2	N/A	Summer 1997	11	1	1,179	22
DLX1	EECS 470	Fall 1996	16	1	3,010	39
DLX2	EECS 470	Fall 1996	21	1	3,015	35
DLX3	EECS 470	Fall 1996	29	1	5,210	13
X86	EECS 470	Fall 1997	42	3	6,071	59
FPU	EECS 627 / PUMA	Fall 1996	96	2	5,607	17
FXU	PUMA	Fall 1996 - Winter 1997	135	2	27,587	113

Table 2: Design files written for the X86 project.

Design file	Code size (lines)	No. of revisions	Error detection method			
			Inspection	Compilation	Simulation	Synthesis
decode.v	984	63	1	2	18	0
datapath.v	530	54	0	9	12	0
stages1.v	294	19	0	1	9	0
modules1.v	1,750	27	1	3	8	0
smallmodules.v	1,010	21	0	4	2	0
fetch.v	140	23	1	2	2	0
datacaches.v	674	13	0	0	1	0
exe1.v	135	8	0	1	1	0
modules.v	554	27	3	1	0	0
Total	6,071	255	6	23	53	0

processor. Two graduate students wrote the synthesizable behavioral description in the fall of 1996. For each of the projects the table lists the number of designers, the duration of the design entry and logic debug part, the size of the design description, and the number of errors that were logged. Design verification for the class projects relied on simulating the design for a few handwritten assembly programs. Simulation outcome was checked by comparing the final state of the processor, and by examining internal signals over the duration of the simulation. For the FXU project, designers also wrote a random program generator and used that to augment handwritten test cases.

```
Index: project/decode.v
===================================================================
RCS file: /x/users/davidvc/repositories/repositories_470_f97/
jhauke/470_repository_98/project/decode.v,v
retrieving revision 1.50
retrieving revision 1.49
diff -r1.50 -r1.49
3c3
< $Revision: 1.50 $
---
> $Revision: 1.49 $
5c5
< $Date: 1997/12/13 22:45:54 $
---
> $Date: 1997/12/13 20:43:41 $
878c878
<    nor4$ Controls_NOPsel_nor2(Controls_NOPsel_nor2_out,
CounterInput,HLT_NOP,ScoreNOP,Stallin);
---
>    nor3$ Controls_NOPsel_nor2(Controls_NOPsel_nor2_out,
CounterInput,HLT_NOP,ScoreNOP);
```

Figure 4. Difference between two successive revisions.

Design project X86 was one of the latest projects from which we collected error data, and hence it benefited the most from past experience.

Table 2 lists the design files created in the course of the project. We list each file's size, the

Table 3. Error distribution in the X86 project.

Error category	Frequency (%)
Wrong signal source	32.8
Missing instance of gate/module	14.8
Missing input(s)	11.5
Wrong gate/module type	9.8
Unconnected (floating) input(s)	8.2
Missing latch/flip-flop	6.6
Conceptual error	4.9
Wrong next state	3.3
Other finite-state machine error	1.6
Extra term/factor	1.6
Extra inversion	1.6
Wrong bit order	1.6
Other	1.6

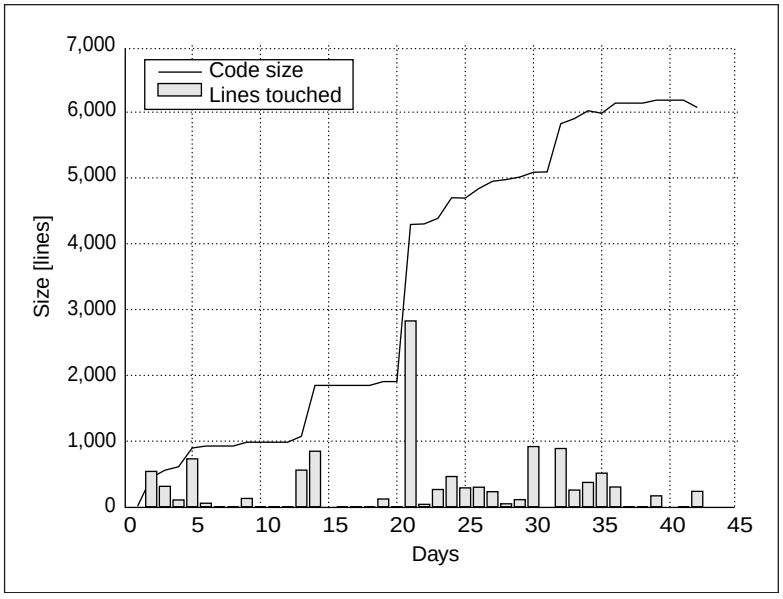

Figure 5. Project evolution: code size (lines) and lines touched over time.

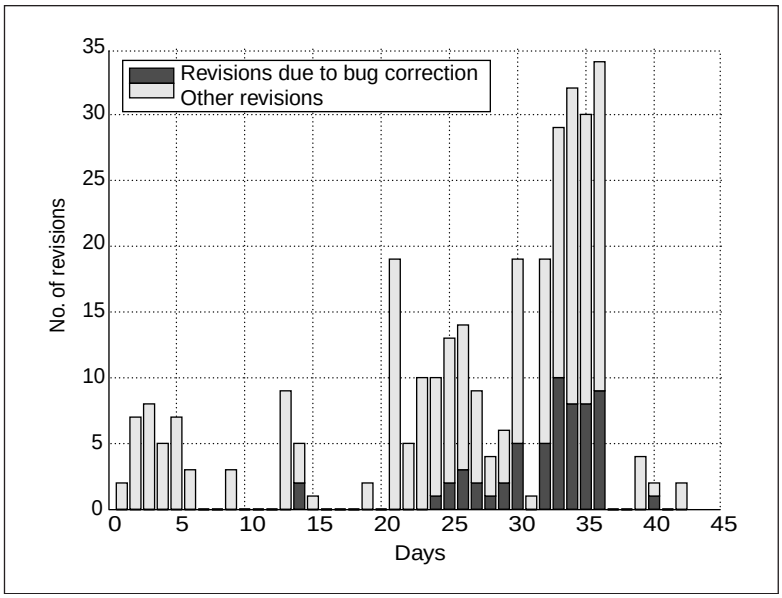

Figure 6. Revisions motivated by bug correction and other revisions over time.

total number of revisions it underwent, and the number of design bugs recorded, broken down by detection method. Note that no synthesis tools were used in this particular project, hence no errors were detected this way. The errors of most interest are those detected by inspection or simulation. The designers were aware that syntax errors are of little value to our work. We can therefore assume that many syntax errors were corrected without recording a new design revision, and hence do not appear in the table under the column "Compilation."

Figure 4 shows the difference between a design revision and the previous revision motivated by an error involving the X86's notoriously complex decoder logic. In revision 1.49, nor gate Controls_NOPsel_nor2 misses input Stallin. Revision 1.50 corrects this error.

Table 3 gives the distribution of design errors by error category. The dominant type of design error is a wrong signal source. Errors involving missing logic are also notable and amount to 33% of the total.

Figure 5 shows the evolution of the project over time. HDL coding and debugging spanned 42 days in this project. The chart shows the total size of the design at the end of each day. Also shown is the number of lines of code that were touched in the course of each day. Most of the design description was in place by day 21, permitting integration testing to start.

The number of revisions made in course of the project is shown in Figure 6. The number logged on any day is broken up into revisions that are due to bug corrections and those due to other reasons. Ideally, there is a one-to-one correspondence between uncovered design

errors and revisions motivated by error correction. Therefore, the bar for the number of revisions logged due to error corrections also gives the total number of bugs corrected on the corresponding day. It can be seen that most of the bugs were discovered and corrected in the second half of the project.

Figure 7 plots the time at which each error was corrected against the number of lines of code that were touched to correct the error. The vertical coordinate is an indication of the structural complexity of the error. Although easy to compute, this metric is far from ideal. It does not distinguish between lines of code that have merely been reformatted and lines that have truly been changed. More accurate measures, such as the minimum number of "atomic" modifications needed to remove an error from the control dataflow graph of the erroneous circuit, would be more appropriate but such measures are also much harder to compute. About half of the errors involved fewer than ten lines of code, and only four errors resulted in design modifications involving more than 100 lines of code.

We further characterize the design errors based on purely structural properties. We define the size of an error as the order of the polynomial that computes the number of similar errors as a function of the size of the circuit. For example, single-inversion errors and single-stuck errors both are of size 1, because there are $O(N^1)$ such errors in a circuit with N lines. Signal source errors are of size 2 as there are $O(N^2)$ such errors. We noted that some actual errors consist of multiple instances of the same type of error. An example is an inversion error on a port connection of a module instance that is repeated for all instances of the module. We define the multiplicity of an actual error as the number of identical and repeated instances of a simpler error that constitute the actual error. Figure 8 plots the frequency of design errors when binned according to size and multiplicity. We observe that design errors of higher multiplicity are rare. Errors of multiplicity 1 and size 1 or 2 account for more than half of all design errors. Only about 12% of the errors are very complex, as indicated by a size of 10 or greater.

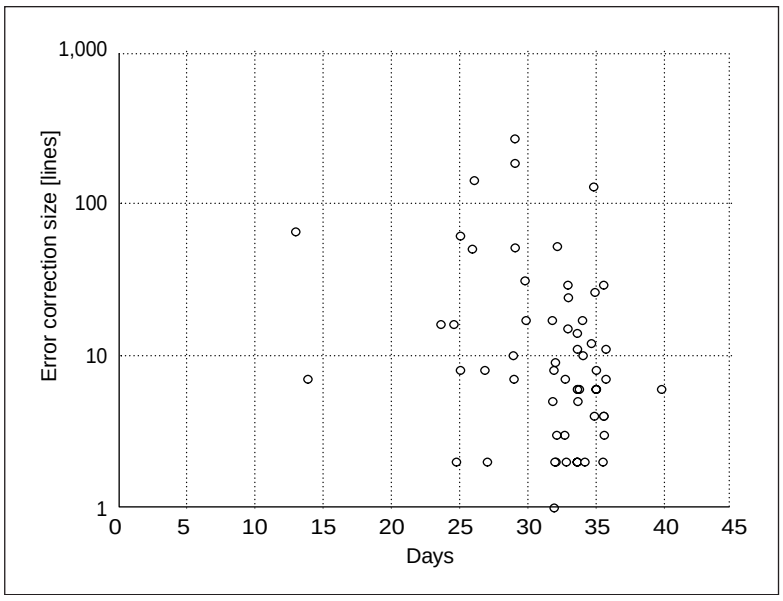

Figure 7. Design errors: time to discovery (days) compared to error size (lines).

Guidelines for Error Collection

A revision management system like CVS has proven to be an invaluable aid in design error collection. Not only did it allow detailed analysis of concrete errors but also eventually came to be appreciated by the designers. Nevertheless, a few designers saw the revision management system as a surreptitious way to monitor their work. Such reservations can usually be overcome by fully explaining the intent of the management system and the benefits accruing from its use.

A key factor is to remove the stigma usually associated with design errors. The participation of students from class projects in the error collection effort was on an entirely voluntary basis. We made an effort to make the participating students feel engaged with our research project, and carefully explained the value of collecting error data.

The need to minimize the overhead of error logging for the designer cannot be underestimated. Although the designer is, in principle, in the best position to classify each newly discovered error, this small effort, from which the designer may not see immediate benefit, might be felt as a burden or threat. Consequently, the designation of errors often becomes imprecise. We observed that for long periods some design-

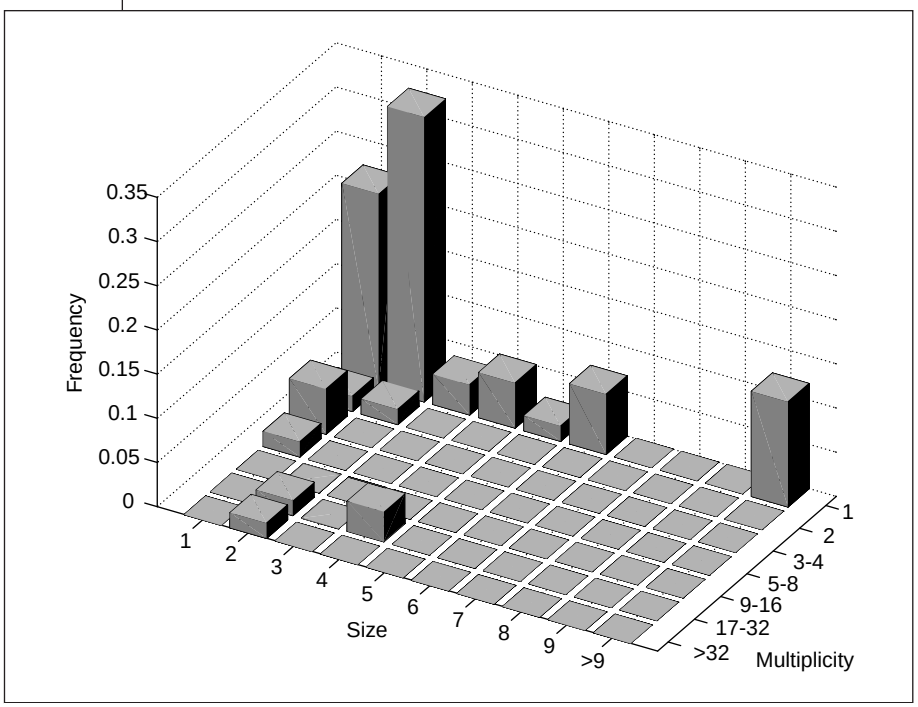

Figure 8. Frequency of design errors in function of their size and multiplicity.

designers.

Provided that a new design revision has been systematically recorded for each detected error, the task of classifying the errors with respect to their structural aspects (item 3 of our questionnaire) can be performed by engineers other than the original designers, perhaps with the help of automation.

Some additional practical considerations need to be pointed out. Fixing a single design error may require multiple modify/simulate cycles, and hence multiple revisions. The designer should record information to distinguish such revisions. Fixing a single design error may require modifications to multiple files. Designers should submit new revisions for all of these files together. Otherwise, the revisions data can wrongly be interpreted as concerning multiple errors.

ers marked all of their bugs as conceptual error, even if the actual error involved a single-inversion error. This led us to reassess the raw revision data and explains the discrepancies between the data reported here and that in earlier work.[2] The reassessment also corrected the counts assigned to errors that spanned multiple design files. Previously, these errors had been overrepresented. This adjustment primarily affects the bigger designs where such errors occurred more often.

An important element in an error collection effort is to encourage designers to adopt the habit of systematically recording every single design error that is not a syntax error. Simple errors such as single inversion errors do not require much explanation. For more elaborate errors a brief textual description of the error, even if is is already listed in the present error template, is very helpful to analyze the error afterward. Additional pieces of information could include a measure of the difficulty of detecting the error, and the root source of the error. Typical root sources include oversight, failure to consider certain behavior, wrongly implemented behavior, misunderstanding of specification, and miscommunication between

Discussion

Table 4 summarizes the error distributions for all the monitored projects. Also listed is the average error frequency over all projects. We observed that signal source errors were the most common type of error at 30%. Errors involving missing logic (missing instance, missing input, missing term, or missing state) were the second most common group at 26%. Also of note were apparently simple errors, such as extra/missing inversions and unconnected inputs, which account for 17% of all errors. More detailed analysis of these errors showed that some were detected quite late in the project. This indicates that behavior of some parts of the design is not properly exercised, since the simple errors do not require any activation conditions.

Our design error collection effort has several inherent limitations, so care should be taken in interpreting our data. First, student designers have limited experience, even in a university program with a strong design emphasis. Nevertheless, their errors may not be too far

Table 4. Design error distributions (%). Among the errors marked as new category are timing errors and errors that required very elaborate corrections.

Category	LC2	DLX1	DLX2	DLX3	X86	FPU	FXU	Average
Wrong signal source	27.3	31.4	25.7	46.2	32.8	23.5	25.7	30.4
Missing instance		28.6	20.0	23.1	14.8	5.9	15.9	15.5
Missing inversion		8.6				47.1	16.8	10.3
New category	9.1	8.6		7.7	6.6	11.8	4.4	6.9
Unconnected input(s)		8.6	14.3	7.7	8.2	5.9	0.9	6.5
Missing input(s)	9.1	8.6	5.7	7.7	11.5			6.1
Wrong gate/module type	13.6		11.4		9.8			5.0
Missing term/factor	9.1	2.9	5.7				4.4	3.2
Wrong constant	9.1		2.9				9.7	3.1
Always statement	9.1		2.9				2.7	2.1
Missing latch/flip-flop					4.9	5.9	0.9	1.7
Wrong bus width	4.5						7.1	1.7
Missing state	9.1							1.3
Conflicting outputs				7.7				1.1
Conceptual error			2.9		3.3		0.9	1.0
Signal declaration			5.7					0.8
Extra term/factor		2.9			1.6		0.9	0.8
Wrong operator							4.4	0.6
Gate or module input			2.9					0.4
Case statement							2.7	0.4
Other FSM error					1.6			0.2
Extra inversion					1.6			0.2
Wrong bit order					1.6			0.2
Wrong next state					1.6			0.2
Latch							0.9	0.1
If statement							0.9	0.1
Expression completely wrong							0.9	0.1

removed from those made by professional designers, since a considerable amount of industrial microprocessor design is done by recent graduates (but working under the supervision of experienced designers). Second, class projects are short in duration and the verification effort possible is modest. Consequently, this data may contain a disproportionately small number of hard-to-detect errors, compared to data from industrial design projects. This limitation also applies to the data derived from university research projects, but to a lesser extent. ■

■ References

1. M.S. Abadir, J. Ferguson, and T.E. Kirkland, "Logic Design Verification Via Test Generation," *IEEE Trans. Computer-Aided Design*, Vol. 7, No. 1, 1988, pp. 138-148.
2. D. Van Campenhout et al., "High-Level Design Verification of Microprocessors Via Error Modeling," *ACM Trans. Design Automation of Electronic Systems*, Vol. 3, No. 4, 1998, pp. 581-599.
3. Y. Malka and A. Ziv. "Design Reliability—Estimation Through Statistical Analysis of Bug Discovery Data," *Proc. Design Automation Conf.*, 1998, pp. 644-649.
4. MIPS Technologies Inc., *MIPS R4000PC/SC Errata*, Processor Rev. 2.2 and 3.0, 1994, http://www.mips.com/publications.
5. B. Beizer, "The Pentium Bug—an Industry Watershed," *Testing Techniques Newsletter, TTN Online Edition*, 1995. http://www.soft.com/News/TTN-Online/ttnsep95.html.

6. Hamarsoft, *Hamarsoft's 86BUGS List*, 4th ed., Heerlen, The Netherlands, 1994. http://www.xs4all.nl/~feldmann.
7. Intel Corp. *8086/8088 User's Manual*, *Programmer's and Hardware Reference Manual,* 1989.
8. A. Avizienis and Y. He, "Microprocessor entomology: a taxonomy of design faults in COTS microprocessors," *Proc. Dependable Computing for Critical Applications,* Vol. 7, 1999.
9. P. Cederqvist et al., "Version management with CVS," *Signum Support AB*, 1993. http://www.cvshome.org
10. M. Postiff. *LC-2 Programmer's Reference Manual*. Rev. 3.1. University of Michigan, Ann Arbor, 1996.
11. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Mateo, Calif., 1990.
12. R.B. Brown et al. "Complementary GaAs technology for a GHz microprocessor," *Proc. GaAs IC Symp.*, Tech. Digest, Orlando, Fla., 1996, pp. 313-316.

David Van Campenhout is a member of the technical staff at Versity Design, Mountain View, CA, where he has been working on functional test generation and temporal logic. His research interests include functional verification, formal methods, and VLSI design. He received an Engineering degree from Katholieke Universiteit Leuven in 1993 and MS and PhD degrees in electrical engineering from the University of Michigan in 1994 and 1999, respectively.

Trevor Mudge received the BSc degree in cybernetics from the University of Reading, England, in 1969, and the MS and PhD degrees in computer science from the University of Illinois, Urbana in 1973 and 1977, respectively. Since 1977, he has been on the faculty of the University Michigan, Ann Arbor. He is a professor of electrical engineering and computer science and recently served as Director of the Advanced Computer Architecture Laboratory. He is the author of articles on computer architecture, programming languages, VLSI design, and computer vision, and he holds a patent in computer aided-design of VLSI circuits. His research interests include computer architecture, computer-aided design, and compilers. Mudge is a Fellow of the IEEE, a member of the ACM, the IEE, and the British Computer Society.

John P. Hayes has been a professor in the Electrical Engineering and Computer Science Department of the University of Michigan, Ann Arbor, since 1982. He teaches and conducts research in the areas of computer-aided design verification and testing, computer architecture, VLSI design, and fault-tolerant computing. He is the author of several books including *Computer Architecture and Organization* and was the founding director of Michigan's Advanced Computer Architecture Laboratory. Hayes received a BE degree from the National University of Ireland, Dublin, and MS and PhD degrees from the University of Illinois, all in electrical engineering. He is an IEEE fellow, and a member of ACM and Sigma Xi.

■ Address concerns and questions to David Van Campenhout, Verisity Design Inc., 2041 Landings Avenue, Mountain View, CA 94043; e-mail dvc@verisity.com.