Unveiling the ISCAS-85 Benchmarks:
A Case Study in Reverse Engineering

MARK C. HANSEN
Delphi Delco Electronics Systems
HAKAN YALCIN
Cadence Design Systems Inc.
JOHN P. HAYES
University of Michigan

Designing at higher levels of abstraction is key to managing the complexity of today's VLSI chips. The authors show how they reverse-engineered the ISCAS-85 benchmarks to add a useful, new high-level tool to the designer's arsenal.

DIGITAL CIRCUITS ARE COMMONLY designed at multiple levels of abstraction, including the layout, transistor, gate, register-transfer (RTL), and architecture (behavioral) levels. Designers describe circuits in a hierarchical, top-down fashion, typically using computer-aided design (CAD) tools. To simplify the design process, designers try to model circuits at a fairly abstract level, such as illustrated in the sidebar, "Modeling the 74283 adder."

Despite the simplified representation and computation they imply, however, circuits described at a high level still challenge the designer. Challenges exist because most CAD tools for design verification, test generation, and timing analysis cannot use high-level abstractions. Instead, the tools work primarily at the gate level, reflecting that level's well-developed Boolean algebra foundation and also its independence of IC technology details. Consequently, to use CAD tools, designers often "flatten" high-level designs to the gate level. This situation both increases circuit complexity and causes a loss of valuable high-level design data.

To resolve this situation, researchers are focusing on CAD tools that exploit multiple levels of hierarchy, but the lack of high-level benchmark circuits to evaluate tool quality and performance handicaps such research. This situation is what spurred us to obtain high-level circuits for use in calibrating new high-level methods in terms of well-understood, low-level approaches.

The widely accepted ISCAS-85 benchmark suite has been in use ever since being introduced in netlist form at the International Symposium of Circuits and Systems in 1985.[1] The circuits are industrial designs whose functions and high-level designs have not been published, both for confidentiality reasons and to allow them to be viewed as random logic circuits with no significant high-level structure.

In researching high-level test generation techniques,[2] we found that, in fact, the ISCAS-85 circuits have well-defined, high-level structures and functions based on common building blocks such as multiplexers, ALUs, and decoders. Because structural knowledge helps us analyze test requirements at the gate level, we began to reverse-engineer the benchmarks, starting with the original gate-level netlists. This process let us systematically recover the circuits' hidden functional and structural information. The high-level models we developed are now available to other researchers, as explained in the sidebar, "High-level ISCAS-85 models." This article describes our experience in reverse-engineering the ISCAS-85 circuits.

Reverse-engineering rationale

Designers typically use reverse engineering to determine a system's specifications, output functions, or other design characteristics from an existing implementation. This contrasts with the customary "forward"

0740-7475/99/$10.00 © 1999 IEEE

Modeling the 74283 adder

Figure A1 is a gate-level schematic diagram of the often-used 74283 adder. We have abstracted the 74283 adder to a single high-level module in Figure A2. This example illustrates some differences between high- and low-level models. To determine the four signal values labeled "?" in Figure A1, we must evaluate every gate in the circuit—a tedious computation. If instead we consider the gates' collective, high-level functionality, we can simplify the evaluation of the unknown signals. In this case, the circuit's function is the addition Z = A + B + C. It is obvious from Figure A2 that the required values are the solution to 13 = ? + 5 + 0, so A = 8 = 1000_2, and the unknown bits in Figure A1 are 1000.

This example also illustrates the fact that, despite their ease of representation and computation, circuits described at a higher level raise new challenges for fields such as testing. For instance, we can't readily tell how we should represent in Figure A2 the structural fault labeled SA0 (an internal line stuck at zero) in Figure A1.

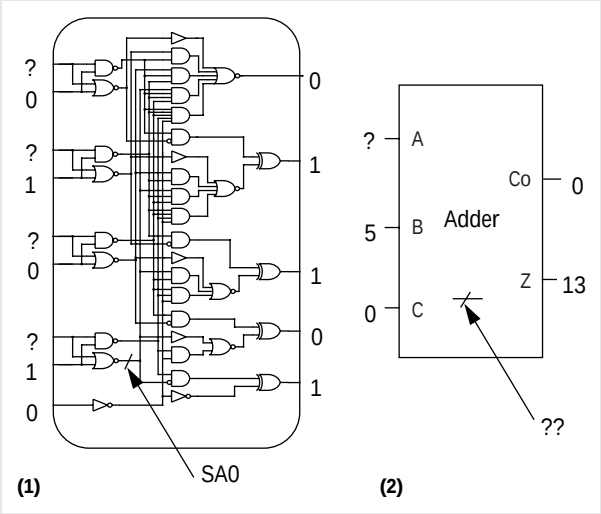


Figure A. *Functional modeling example: gate-level model of the 74283 adder (1), and equivalent high-level model (2).*

(specification to implementation) design process. Companies often reverse-engineer their competitors' products to discover how they are made or to evaluate their quality. In the software industry, for example, reverse engineering refers to updating, for reuse, programs whose specifications have been lost or inadequately documented.[3] In computer hardware, designers have used reverse engineering to extract gate-level models from transistor circuits.[4]

A limited form of reverse engineering applies to layout verification: Designers use CAD tools to extract a netlist from a circuit's layout implementation and compare it to the circuit's original representation, a problem commonly known as LVS (layout versus schematic).[5] However, extracting higher level functional information, such as an RTL or architectural description, from a gate-level circuit has rarely been addressed.

We realized that, to discover what useful high-level structure the ISCAS-85 circuits might contain, we would need to reverse-engineer their gate-level netlists. Figure 1a (next page), which shows part of the netlist defining one of the smaller ISCAS-85 benchmarks—the c880, with its 383 gates, 26 inputs, and 60 outputs—illustrates this task. The specific problem is to recognize that certain groups of gates have a well-defined function (carry look-ahead or CLA) that can be replaced by a high-level module (a CLA generator). The highlighted gates in Figure 1a show a CLA circuit that is transformed in Figure 1b into a high-level module (circled area).

High-level ISCAS-85 models

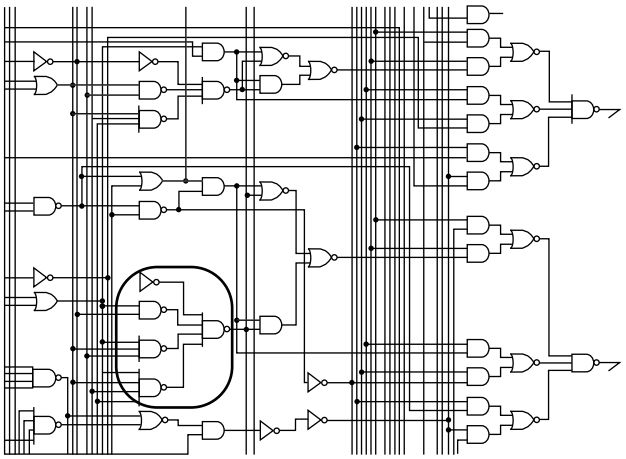
High-level models, in both structural and behavioral versions (in schematic circuits and Verilog simulation code), are available for all the ISCAS-85 benchmark circuits at http://www.eecs.umich.edu/~jhayes/iscas/benchmark.html. These models partition the original netlists into standard RTL blocks and identify the block functionality. The structural Verilog versions, for which we have a complete set, express the specific high-level structure implicit in the original gate-level designs. In some cases, we have also constructed behavioral Verilog models, which define high-level blocks as logical equations that can be synthesized into gates. The resulting suite of hierarchical designs greatly extends the ISCAS-85 circuits' usefulness and is available for experimentation in a variety of applications such as test generation, logic synthesis, and timing analysis.

While the high- and gate-level models of each circuit are intended to be both functionally and structurally equivalent, small and unavoidable structural discrepancies purposely exist between the models in some cases; these are noted at our Web site where they arise. Functional equivalence was checked using the logic verification tool VIS, as well as by comparing fault detection lists.

Logic from function. The well-known 74181 unit ALU and function generator further illustrates the problem of transforming low-level functions into high-level logic. Figure 2 (page 75) shows the 74181's standard logic circuit schemat-

```
...
737 = and (237, 662)
738 = not (670)
739 = and (228, 673)
740 = and (237, 670)
…
763 = nand (635, 644, 722)
764 = nand (609, 687)
765 = nand (600, 678)
766 = nand (600, 609, 687)
767 = buff (660)
768 = buff (661)
…
812 = nand (619, 796)
813 = nand (609, 796)
814 = nand (600, 609, 619, 796)
815 = nand (738, 765, 766, 814)
819 = nand (741, 764, 813)
…
```

(a)



(b)

Figure 1. *A portion (13%) of the ISCAS-85 c880 benchmark: netlist with the gates of a CLA generator highlighted* ***(a)*** *and equivalent gate-level schematic with the CLA generator circled* ***(b)****.*

ic. Researchers have characterized this circuit as a typical example of unstructured logic, yet as Figure 3 shows, the 74181 has a considerable amount of high-level structure.

Recognizing the logic that makes up a CLA block—the circled elements in Figure 2—is key to unlocking the 74181's secrets. Note how Figure 3b represents the four boxed circuits in Figure 2 as the single module M_1 with 4-bit I/O buses. A second quadruplicated circuit in the 74181 leads to the high-level module M_2, as in Figure 3c. The XOR gates are also grouped into 4-bit word gates. The 74181's original designers cleverly constructed the M_1 and M_2 logic so that with input line $M = 1$, each setting of the S (function select) bus produces one of 16 possible Boolean functions of the form $F(A,B)$.

Reverse-engineering techniques. We can identify high-level structures in logic circuits with a combination of techniques. These techniques do not completely solve the reverse-engineering task, but they do illustrate the principles we followed to derive the ISCAS circuit functions.

- *Library modules.* These common components, such as multiplexers, decoders, adders, and CLA generators, are found in IC manufacturers' databooks or cell libraries and in textbooks. The modules usually exist in variants due to differences in input size (fan-in or word length) and gate types. The circled logic elements in Figure 2 are variants of the CLA module marked in Figure 1. Note the different gate types and that the number of input bits of the CLA modules increases, from bottom to top, as we move from least significant to most significant bit positions.
- *Repeated modules.* Often a subcircuit whose logic function is not apparent occurs frequently, especially in data-path circuits where the same circuit slice repeats for different bits of input data. For example, Figure 2 shows four copies (boxed) of the three-gate M_1 circuit. Since the 74181 operates on 4-bit data words, such four-fold repetition is to be expected.
- *Expected global structures.* After recognizing several modules, the reverse engineer can look for common structures, signals, or functions that use these modules. For example, knowing that Figure 2 contains CLA generation logic, we would expect the circuit to produce certain functions such as the P (propagate), G (generate), and sum functions associated with CLA addition. Not surprisingly, M_1 (M_2) produces G (P) signals; the XOR gates produce the sum function.
- *Computed functions.* With a few structural clues to a subcircuit's role, we can compute its logic function in symbolic or binary (truth table) form, then relate it to known functions or to other circuit functions. This is feasible only for functions of typically no more than four or five signals. For example, each primary output F_i in Figure 2 can be expressed as

$$F_i = (C_i + M) \oplus D_i \oplus E_i \qquad (1)$$

  for $i = 0$, 1, 2, and 3, which helps us understand what D_i and E_i produce (the operands for addition).
- *Control functions.* We can often identify key control signals whose settings partition a complex function into simpler ones. In Equation 1, for instance, M is shared by each F_i, a typical characteristic of control signals. On setting $M = 0$, Equation 1 becomes

$$F_i = C_i \oplus D_i \oplus E_i$$

which is the logical expression for the arithmetic sum of two data bits and a carry bit. This suggests that $M = 1$ turns off the 74181's arithmetic functions and turns on its logic functions, which indeed it does.

- *Bus structures.* The outputs of repeated modules often can be grouped into buses. Further circuit partitioning can result from noting where these common signals lead.
- *Common names.* When analyzing netlists, we sometimes find a shared name among several elements. We may not know what that name implies, but grouping the elements together temporarily can lead to further structural insights.
- *Black boxes.* If all else fails, we can encapsu-

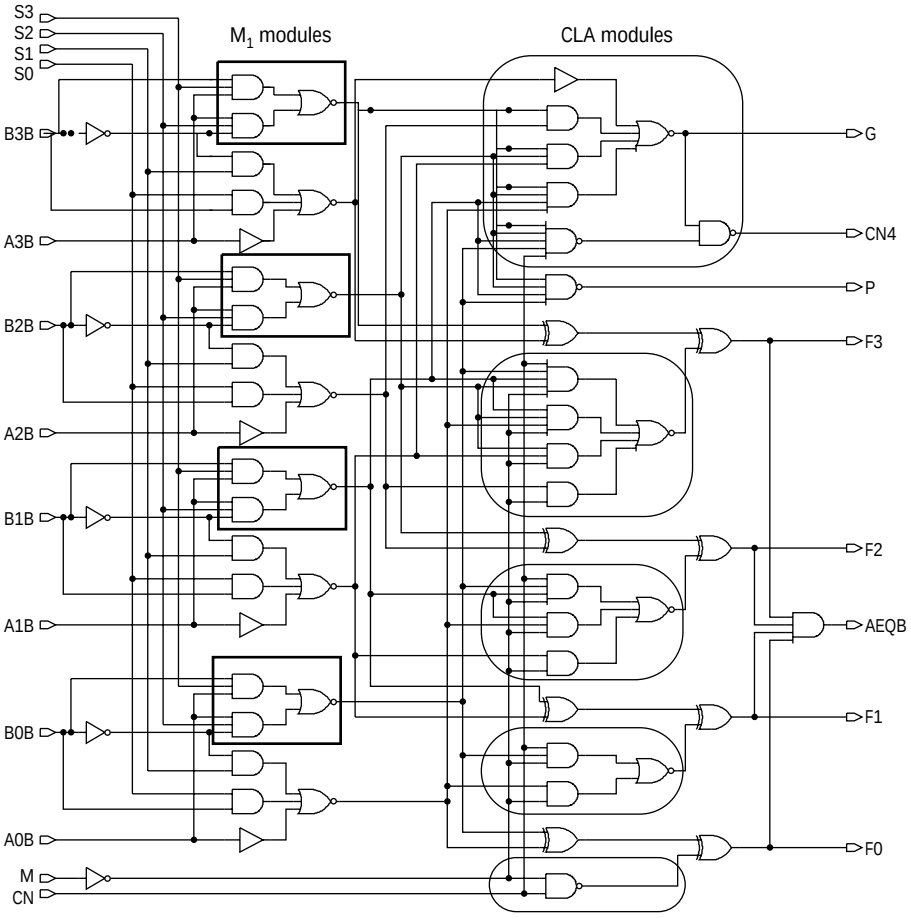


Figure 2. *Gate-level schematic of the 74181 ALU showing some of its high-level structure.*

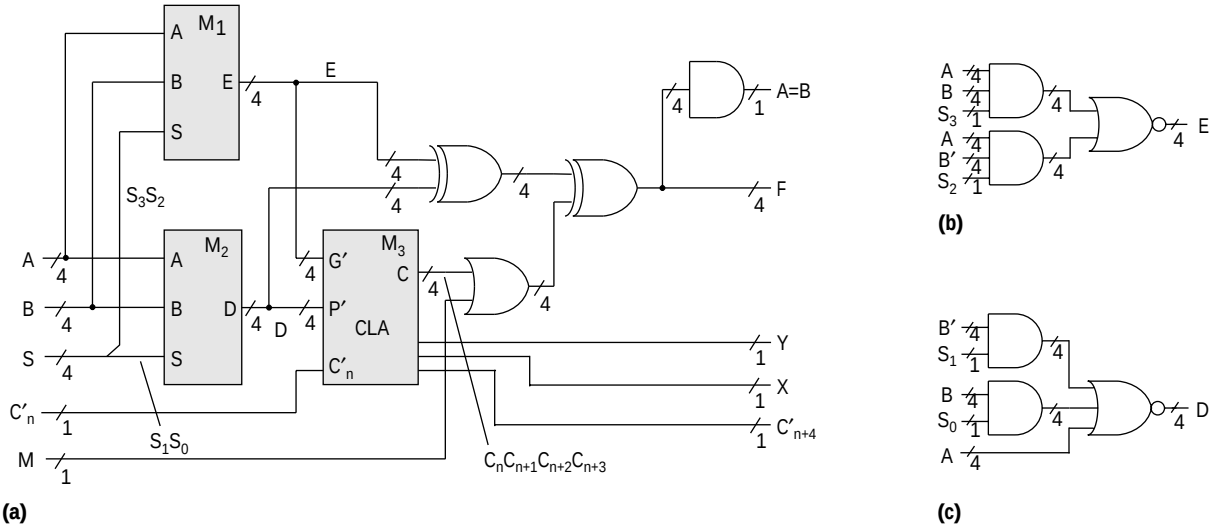


Figure 3. *74181 ALU: high-level model (a), logic of module M_1 (b), and logic of module M_2 (c).*

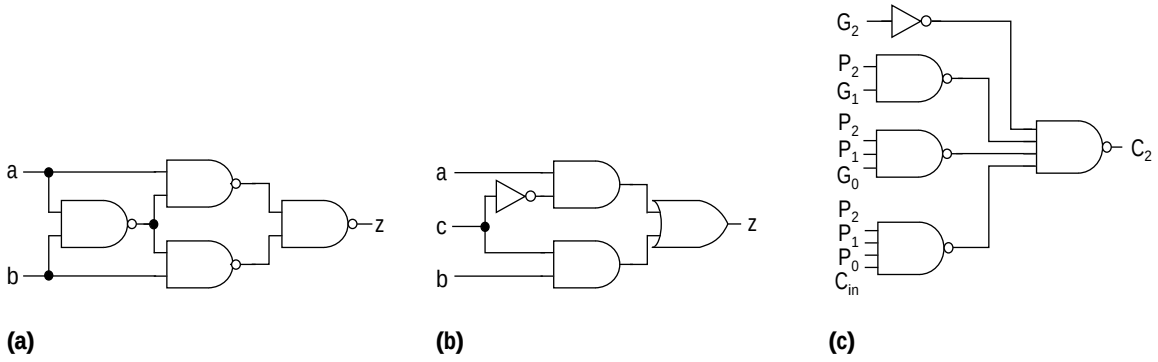


Figure 4. *Small functional blocks widely used in ISCAS-85 circuits: 2-input XOR (a), 2-to-1 multiplexer (b), and carry generator for the third bit of a CLA (c).*

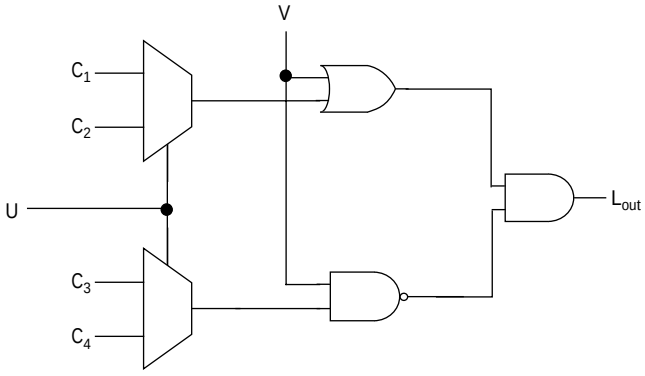


Figure 5. *Module generating all logic functions of inputs U and V.*

late a circuit as a module of unknown function or black box. This step is unavoidable when dealing with low-level control circuits consisting of truly random logic.

Extending the process. To extend the reverse-engineering process to larger designs, we converted the original netlists to schematics to visualize the circuits and track their interconnections. We used Synopsys's Design Analyzer CAD tool for this purpose. Next, we wrote a program to automate library module recognition. This program, which implements a simple pattern-matching method replaces certain subcircuits with a single module. Figure 4 shows three such examples.

Finally, we temporarily removed buffers from the original netlists, since they do not affect functionality, and thereby reduced the netlist size of the ISCAS-85 benchmarks by about 15% on average.

As we learn a circuit's functions, we can use them to understand adjoining logic. Identifying library building blocks such as multiplexers, flip-flops, decoders, and carry generators is essential. As reverse engineers, we can also find global interconnection structures, such as CLA or carry-select circuits, which helps establish the overall structure.

Another boost to the reverse-engineering process is the tendency of designers to follow published or textbook designs. For instance, part of the ISCAS-85 c3540 circuit realizes the same set of logical and arithmetic functions as the 74181 ALU/function generator, in essentially the same way. Similarly, the c499 is a single error-correcting circuit that implements a specific, published coding scheme.[6]

Trial and error. Reverse engineering is inevitably a trial-and-error process. To begin, the reverse engineer must make some functional assumptions about inputs or subcircuits, even if changes are later required. For example, the inputs to a large CLA adder cannot easily be properly ordered. After deriving the intermediate carry functions deep inside the circuit, the reverse engineer may need to reorder the inputs. Another source of difficulty is the occasional nonstandard use of standard modules. In Figure 5, for example, the control inputs *U* and *V* of the multiplexers are actually part of a data bus, and the data inputs *C*1–*C*4 are control lines, which can be determined only after analyzing the surrounding logic. This particular module—present in the ISCAS-85 c3540 and c7552 benchmarks—is part of an ALU that generates all 16 logic functions of its inputs *U* and *V*.

High-level models

In this section, we detail the models we obtained for the ISCAS-85 c880 and c1908 benchmarks; we summarize the rest. Table 1 lists the ISCAS-85 functions (complete details are available at our Web site, http://www.eecs.umich.edu/~jhayes/iscas/benchmark.html).

c880 ALU. Figure 6 shows the high-level model for this 383-gate circuit, which is an 8-bit ALU.

Because the common 74181 ALU has a CLA generator

Table 1. *Summary of the ISCAS-85 benchmark circuits. SEC/DED stands for "single-error-correcting and double-error detecting."*

Circuit	Function	No. of input lines	No. of output lines	No. of logic gates	No. of major functional blocks
c432	27-channel interrupt controller	36	7	160	5
c499	32-bit SEC circuit	41	32	202	2
c880	8-bit ALU	60	26	383	7
c1355	32-bit SEC circuit	41	32	546	2
c1908	16-bit SEC/DED circuit	33	25	880	6
c2670	12-bit ALU and controller	233	140	1,193	7
c3540	8-bit ALU	50	22	1,669	11
c5315	9-bit ALU	178	123	2,307	10
c6288	16 × 16 multiplier	32	32	2,406	240
c7552	32-bit adder/comparator	207	108	3,512	8

module, it's not surprising to find a similar module in the c880. The c880 core is an 8-bit adder, similar to the 74283. Module M_3 produces the generate, propagate, and sum signals; M_4 and M_5 are 4-bit CLA units. Multiplexers M_1 and M_6 select incoming and outgoing data buses, respectively. M_2 controls both multiplexers, such that an external source must ensure that only one function is activated at a time on $C(25:0)$. M_2 also contains 25 logic gates that generate control signals for circuits that presumably lie outside c880 itself.

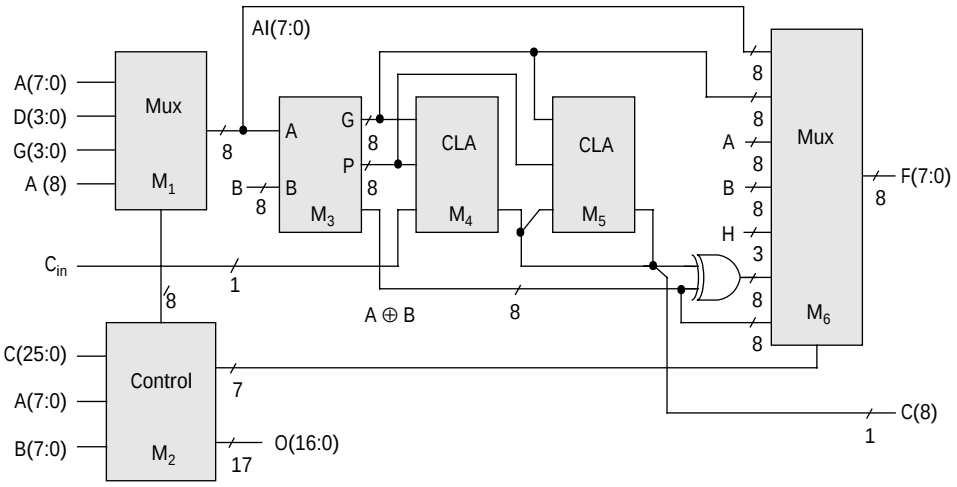


Figure 6. *High-level model for the c880 ALU.*

c1908 error-correcting circuit. Figure 7 (next page) shows the high-level model of the c1908 circuit, a 16-bit SEC/DED (single-error-correcting and double-error detecting) code-processing circuit with some byte-error-detection capability. It generates a 6-bit syndrome containing diagnostic information about the 16-bit data input *IN*, which the c1908 decodes to find the *IN* bit in error, if any. If it detects an error and the control inputs are set appropriately, the c1908 then corrects the error. The c1908 has an output indicating an uncorrectable error; this is set when more than one erroneous bit is detected. The circuit can also output the syndrome via the *SC* (syndrome checkbits) lines. These lines make it possible to cascade several copies of c1908 so that detection and correction can be done for words larger than 16 bits.

Other circuits. Figure 8a (next page) shows the high-level model for the c432 benchmark, which is an interrupt controller. It has three 9-bit request buses (channels), A, B, and C, and a channel-enable bus, E. Seven outputs *PA*, *PB*, *PC*, and *Chan* [3:0] specify which channels have acknowledged interrupt requests.

The 202-gate c499 benchmark is an SEC circuit, as Figure 8b shows. Its 41 combined inputs form an 8-bit internal bus *S*, which combines with 32 primary inputs to form the 32 primary outputs. The *S* lines generate a unique syndrome for each erroneous input line. The Boolean expressions defining *S* form the *H* matrix for a (40,32) Hamming code.[6] Module M_2 contains the necessary correction logic; however, no error-detection logic is present.

The c1355 benchmark has the same overall function as c499, except that all the XOR primitives of c499 are expand-

ed to their four-NAND-gate equivalents, as Figure 4a shows.

The c2670 benchmark is an ALU with a comparator, an equality checker, and several parity trees. The comparator has two 12-bit inputs, X and Y, and calculates $Y > X$ using a CLA adder that performs the addition $\overline{X} + Y$.

The 1,669-gate c3540 is an 8-bit ALU with binary and BCD arithmetic, and logic and shift operations. Logical operations are intermixed with arithmetic ones, much as in the 74181. BCD addition is done via a two's-complement adder by adding 6 to both digits of the first operand and then subtracting 6 from the digits of the result if they do not generate a carry.

The c5315 is an ALU that performs addition and logic operations simultaneously on two 9-bit input data words and also computes the results' parity. The parity logic for the sums, as well as the adders themselves, uses a hybrid carry-select/CLA scheme with 4-bit (low-order) and 5-bit (high-order) blocks.

The c6288 multiplier has 124 levels of logic gates. Its 2,406 gates form 240 full- and half-adder cells in a matrix. With billions of I/O paths, it has been a particularly difficult test case for existing timing analysis methods.

The c7552 is the largest of the ISCAS-85 benchmark circuits, with 3,512 gates. It is a 32-bit adder (expandable to 34 bits) and magnitude comparator that checks the parity of the adder output and its input buses. Like the c5315, it uses a hybrid carry-select/CLA adder with 4- and 5-bit blocks.

Our reverse-engineering approach applies equally well to sequential circuits, although that discussion is beyond the scope of this article.[2,7]

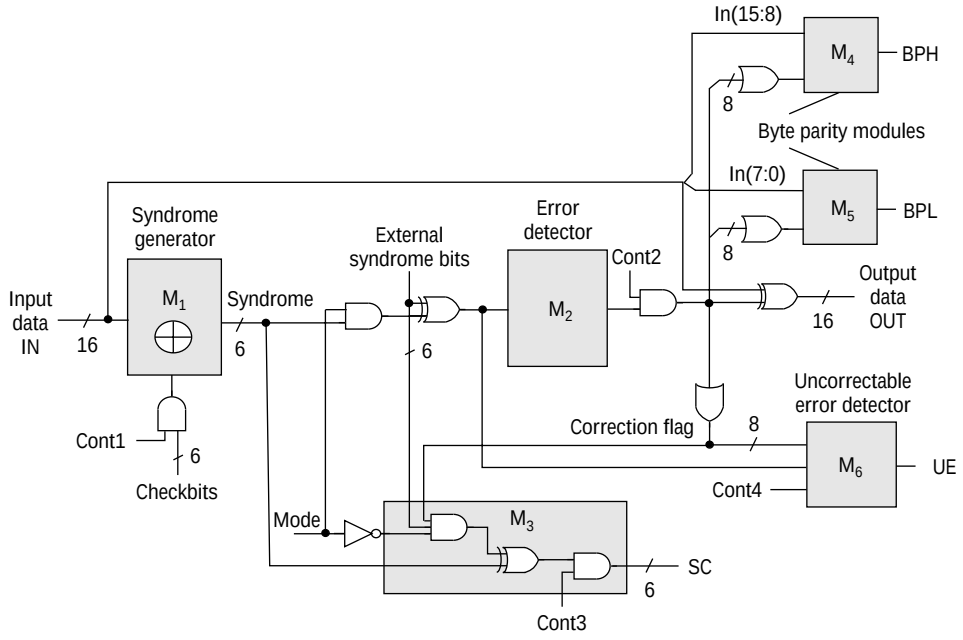


Figure 7. *High-level model of the c1908 error-correcting circuit. SC stands for syndrome checkbits; BPH, for byte parity high; BPL, for byte parity low; and UE, for uncorrectable error.*

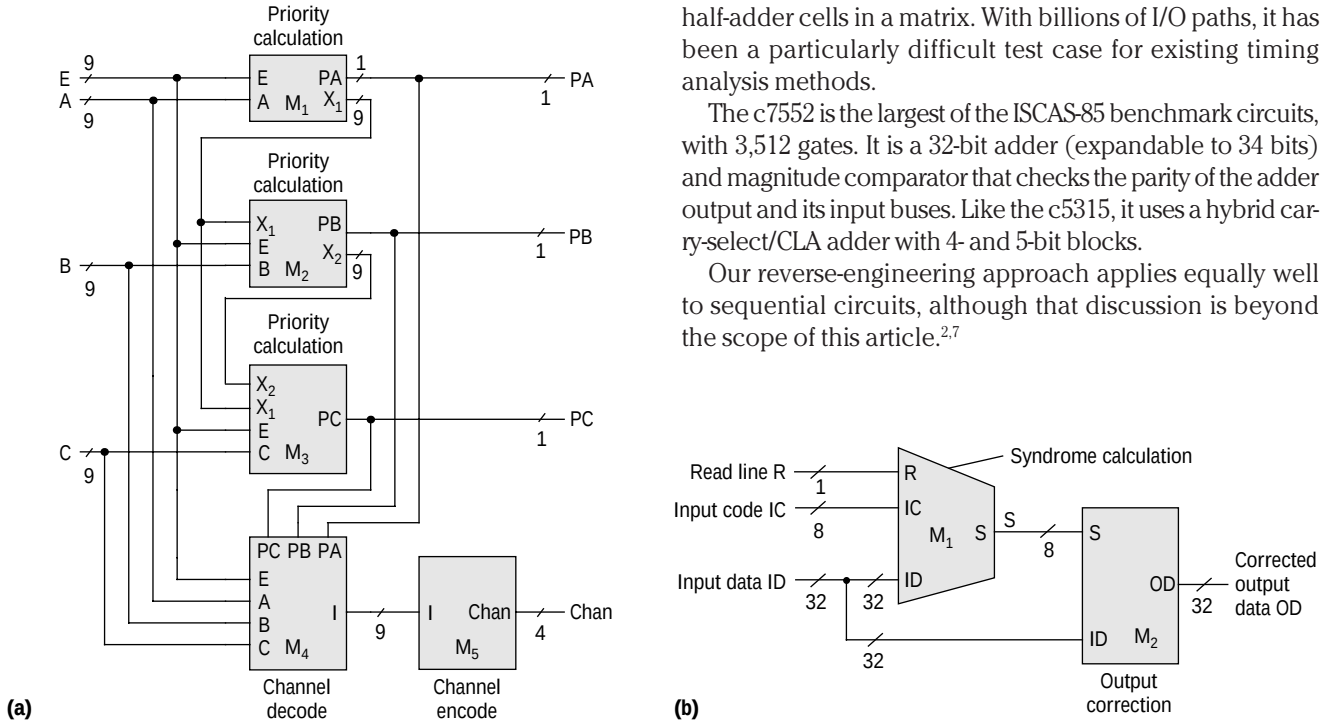


Figure 8. *High-level models of the c432 interrupt controller* ***(a)****, and of the c499 single-error-correcting circuit* ***(b)****.*

Applications

Reverse-engineering the ISCAS-85 benchmarks to discover their high-level structures let us work with only a small number of components and interconnections. The resulting simplification made it much easier to apply these benchmarks to test generation,[2] performance estimation,[8,9] and technology mapping,[10] and to reap benefits of speed and size.

For test pattern generation, the main advantages of the high-level approach are very compact test sets and reduced computation time, which can be several orders of magnitude faster than at gate level. Table 2 highlights some of our results for a few ISCAS-85 and ISCAS-89 benchmarks.[2]

Our future research in this area will utilize this functional information to develop high-level design-for-test methodologies and to locate redundant faults in circuits during test generation.

In another application, we[9] developed a hierarchical timing analysis method, exploiting high-level structure and function to calculate a circuit's longest delays. The c6288 multiplier, for example, is notoriously difficult to analyze with traditional, low-level approaches. In a high-level model, the number of paths of longest delay in this circuit is only 2, as opposed to 2^{33} (over 8 billion) at the gate level. The c6288 also contains many false paths: paths that appear to have the maximum delay but which cannot propagate signal changes.

We have also developed a fast delay calculation method[12] that distinguishes between data-like and control-like input signals. Such distinctions are entirely invisible in the ISCAS-85 gate-level models. The new method uses the reverse-engineered high-level models to identify the circuit inputs as control-like (called timing sensitive) or data-like (called timing insensitive). It assumes that delays are determined only by the logic values of timing-sensitive inputs. By ignoring the logic values of timing-insensitive inputs, the overall computational effort is greatly reduced, as Figure 9 shows.

Kukimoto et al.[8] have developed a similar approach to timing analysis using our ISCAS-85 models as benchmarks. They also report significant computation speedups.

The ISCAS-85 high-level models are similarly useful in other aspects of design. For example, Chowdhary and Hayes[10] use them in conjunction with a technology-mapping technique that embeds logic circuits in as few field-programma-

Table 2. *Test generation for gate-level faults in several ISCAS-85 and ISCAS-89 benchmark circuits. The c in front of the circuit number indicates ISCAS-85; the s indicates ISCAS-89.*

Circuit	Function	Previous smallest test set size from gate-level model	Test set size from high-level model[11]
c880	8-bit ALU	21	17
c6288	16 × 16 multiplier	14	12[1]
c6288 (modified)	16 × 16 multiplier	—	5[2]
s298	Traffic light controller	132	103[1]
s349	4-bit multiplier	84	45[1]

1 Best known to date
2 Minimum possible

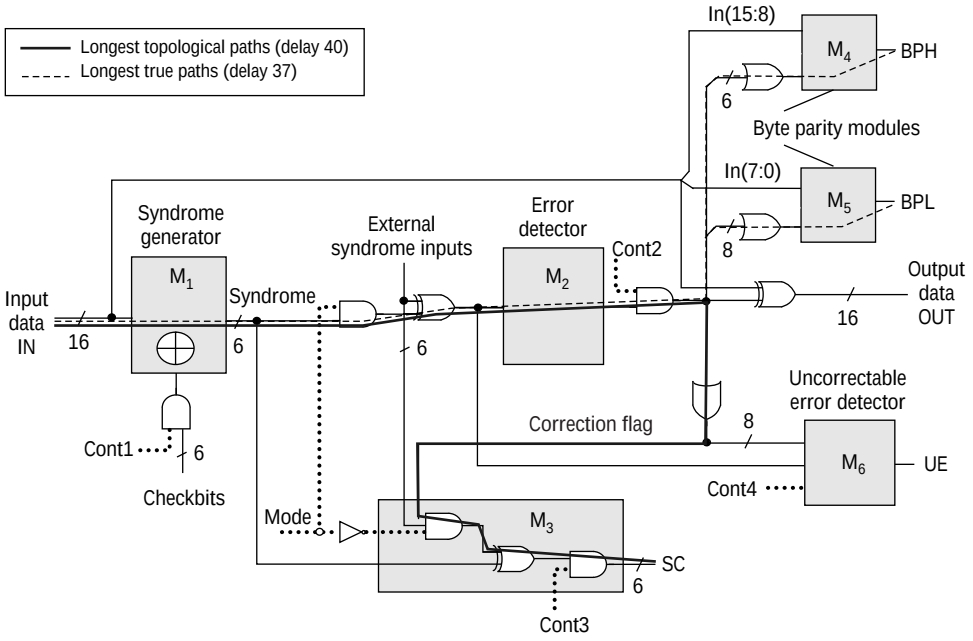


Figure 9. *Longest false (topological) and longest true paths of c1908 identified by the ACD method.[12] The dotted lines are timing-sensitive (control-like) inputs.*

ble gate arrays as possible. With larger circuits like most of the ISCAS-85 benchmarks, it is necessary to partition the given circuit into subcircuits of a few hundred gates or so, depending on the complexity of the FPGA's logic blocks, and map the subcircuits individually.

HIGH-LEVEL BENCHMARKS HAVE proven beneficial in test generation, timing analysis, and technology mapping. We encourage other researchers to use the schematic and Verilog versions of these ISCAS-85 models, available on our Web site, http://www.eecs.umich.edu/~jhayes/iscas/benchmark.html.

We have also reverse-engineered a few of the ISCAS-89 sequential circuits, available on our Web site. Ultimately, we hope that our high-level models, as well as others' work along similar lines, will contribute to the development of more-powerful, high-level CAD techniques and tools, as they are increasingly necessary in modern VLSI circuit design. D&T

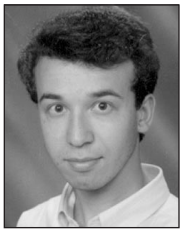
Acknowledgments

We thank Hyungwon Kim for his assistance in constructing some of the Verilog models and Hussain Al-Asaad and Jonathan Hauke for checking the models. We also thank Delphi Delco Electronics Systems, the National Science Foundation, and the Semiconductor Research Corporation for supporting portions of the research.

References

1. F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits," *Proc. IEEE Int'l Symp. Circuits and Systems*, IEEE Press, Piscataway, N.J., 1985, pp. 695–698; see also the ISCAS-85 benchmark directory at http://www.cbl.ncsu.edu/benchmarks.
2. M.C. Hansen and J.P. Hayes, "High-Level Test Generation Using Physically Induced Faults," *Proc. VLSI Test Symp.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1995, pp. 20–28.
3. E.J. Chikofsky and J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, Vol. 7, No. 3, Mar. 1990, pp. 13–17.
4. S. Kundu, "GateMaker: A Transistor to Gate Level Model Extractor for Simulation, Automatic Test Pattern Generation and Verification," *Proc. IEEE Int'l Test Conf.*, IEEE CS Press, 1998, pp. 372–381.
5. N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd ed., Addison-Wesley, Reading, Mass., 1992.
6. C.L. Chen and M.Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Research and Development*, Vol. 28, Mar. 1984, pp. 124–134.
7. F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. IEEE Int'l Symp. Circuits and Systems*, IEEE Press, 1989, pp. 1,929–1,934.
8. Y. Kukimoto et al., "Approximate Timing Analysis of Combinational Circuits Under the XBD0 Model," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, 1997, pp. 176–181.
9. H. Yalcin and J.P. Hayes, "Hierarchical Timing Analysis Using Conditional Delays," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, 1995, pp. 371–377.
10. A. Chowdhary and J.P. Hayes, "General Modeling and Technology-Mapping Technique for LUT-Based FPGAs," *Proc. Fifth Int'l Symp. FPGAs*, ACM Press, New York, 1997, pp. 43–47.
11. M.C. Hansen and J.P. Hayes, "High-Level Test Generation Using Symbolic Scheduling," *Proc. IEEE Int'l Test Conf.*, IEEE CS Press, 1995, pp. 586–595.
12. H. Yalcin, J.P. Hayes, and K.A. Sakallah, "An Approximate Timing Analysis Method for Datapath Circuits," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, 1996, pp. 114–118.

Mark C. Hansen supervises the IC Design Automation Operations groups at Delphi Delco Electronics Systems in Kokomo, Indiana. His research interests include functional-level fault modeling, design for test, ATPG, BIST, and analog test. He holds a dozen US patents. Hansen received a BSEE from Michigan State University, an MSEE from Carnegie Mellon University, and a PhD in computer science and engineering from the University of Michigan. He is a member of the IEEE and the IEEE Computer Society.



Hakan Yalcin is a senior member of the technical staff at Cadence Design Systems in San Jose, California. His research interests include timing analysis, characterization, and optimization of VLSI circuits. He received a BS in control and computer engineering from Istanbul Technical University, Turkey, and an MSE and a PhD in computer science and engineering from the University of Michigan. He is a member of the IEEE and the IEEE Circuits and Systems Society.

John P. Hayes is a professor of electrical engineering and computer science at the University of Michigan, Ann Arbor. He teaches and conducts research in the areas of computer-aided design, verification and testing, computer architecture, VLSI design, and fault-tolerant computing. He received a BE degree from the National University of Ireland, Dublin, and an MS and a PhD from the University of Illinois, all in electrical engineering. He is an IEEE fellow and a member of the ACM and Sigma Xi.

Address questions and comments about this article to John Hayes, Dept. of Electrical Eng. and Computer Science, Univ. of Michigan, Ann Arbor, MI 48109; jhayes@eecs.umich.edu.