Causality Based Generation Of Directed Test Cases

Nina Saxena
Comp. Eng. Res. Ctr.
University Of Texas At Austin
Austin, TX 78712
saxena@cerc.utexas.edu

Jacob A. Abraham
Comp. Eng. Res. Ctr.
University Of Texas At Austin
Austin, TX 78712
jaa@cerc.utexas.edu

Avijit Saha
IBM, Austin
11400 Burnet Rd.
Austin, TX 78758
saha@austin.ibm.com

Abstract— Simulation is considered to be the irreplaceable part of design verification. However, the efficiency of this method depends greatly on the test cases used. Random test cases can be generated quickly but have poor coverage. Directed test cases on the other hand require time and manual effort. This paper presents a method for generating directed test cases automatically by making use of signal relationships in the specification. An algorithm is presented that was applied to the GL85 microprocessor, a clone of Intel's 8085. The results are compared with other methods to see the gain with the proposed method.

I. Introduction

The outcome of simulation greatly depends on the test cases used to validate the design. Test cases can be random and generated using automatic methods or they could be directed and generated manually. Random test cases, though easy to generate, do not give a good coverage as they do not exercise different paths or simulation runs very effectively. This means that it is harder to catch bugs using randomly generated test cases. Directed test cases, on the other hand, require time and expertise, thus adding to the overall verification time.

The problem, therefore, remains of being able to generate directed test cases in as little time as possible. Sometimes this is done by using a set of parameters to generate test cases targetting the relevant logic. This system however still requires manual interaction since the verification engineer has to analyse properties and supply the parameters to the generator.

II. Causality Based Biasing

Directed test cases are developed manually and require several days to generate. Therefore the generally accepted technique in industry is to use random tests biased by constraints or parameters. These constraints are often coded manually and do not correspond to any particular bug. Our approach in this work is to utilize the information encoded in formal specifications to select the parameters and thereby generate bug specific, directed tests.

A causality based approximation is used in our approach. Causality relationships between signals is dictated by the implication in the formal specifications used to validate the design. These specifications are then converted to a checker for simulation as discussed in the next section.

A mapper was developed to interpret specifications and to select parameters based on the information derived from the implications within each specification. This mapper detects causal relationships between signals and picks parameters to exercise the part of the logic it believes the specification to be targetting. These parameters are then provided to the test generator as input. Since the use of parameter based test generator is fairly common, details of any particular test generator have not been discussed in this paper.

CAUSE / EFFECT	SIGNAL1	SIGNAL2	• • •	SIGNALn
SIGNAL1	PARAM4		• • •	PARAM3 PARAM4
SIGNAL2	PARAM5		• • •	PARAM1 PARAM5
⋮	⋮	⋮	⋱	⋮
SIGNALn		PARAM2	• • •	PARAM1

Fig. 1. Look-up Table For Parameters Based On Causal Relationships Between Signals

Fig. 1 shows a lookup table for finding parameters based on the causal relationship between signals. If $SIGNAL1$ is a signal in the causal part of the specification and $SIGNAL2$ is a signal in the effect part of the specification then the parameter $param(SIGNAL1, SIGNAL2)$ to be applied is stored in the entry denoted by the column for $SIGNAL1$ and the row for $SIGNAL2$ in the matrix shown.

Some of these parameters could correspond to sets of biases instead of single biases. Some entries are not specified if the signals have no causal relationships between them, and in that case nothing is appended to the set of

 ISBN 0-7803-5974-7

parameters. Also, the matrix is not symmetrical since the causality relation between signals is not symmetrical.

```
Begin
    τ = {}      /* set of parameters */
    χ = set of causal signals in rule;
    ε = set of effect signals in rule;
    for each i in χ do
    begin
        for each j in ε do
        begin
            τ = τ ∪ param(i,j);
        end
    end
End
```

Fig. 2. Test Case Generation Algorithm

Fig. 2 gives the Test case Generation Algorithm which is used to deduce the set of parameters given to the test case generator to generate test cases for a particular specification. Fig. 3 shows an example of how a signal relationship is mapped on to parameters for test generation. Since the size of the matrix used has n^2 elements, where n is the number of signals in the design, the worst case complexity of this algorithm is $O(n^2)$. This is the time required to build the matrix. The time for lookup for each parameter is constant.

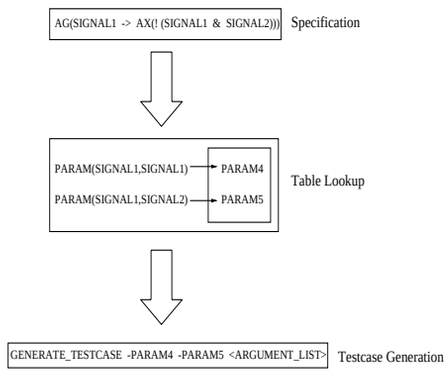

Fig. 3. Example Of Test Case Generation

III. Merged Verification

For our experiments and a fair idea of the performance of the proposed method, it was also desirable to include formal verification as part of the validation process. Usually industries apply both simulation and formal methods to their designs [6] to reduce the chances of undetected bugs. These efforts, however, are usually decoupled and carried on separate platforms by different teams. Due to this gap between the formal verification efforts and simulation efforts, there is no way to reduce repetitions in locating bugs. Consequently, much time is wasted and the cost of verification runs high. Combining simulation and model checking on a common platform would not only save time but also reduce repetitive or redundant verification runs and make it easy to manage large designs.

One way to create such a platform is to use executable specifications. These executable specifications can be written to serve as an abstract model of the actual design. Specifications written using a hardware description language or a programming language [4, 5, 3] could be used. The problem with such specifications would be that of uncertainty due to flexibility. The high level and flexible description language would allow for multiple ways of expressing the same thing, which would make interpretations and changes difficult to do. Besides, though simulation is easy using such specifications, formal verification would not be possible.

The other option would be that of using a language that preserves the formality and allows specifications to be written in a concise and specific fashion. We used a system where specifications were written in CTL-like rules and could be used either for formal verification or as checkers for simulation. When used in simulation, the specifications are used in a checker and checked every cycle to see if the behavior of the simulation run is adhering to that of the rules.

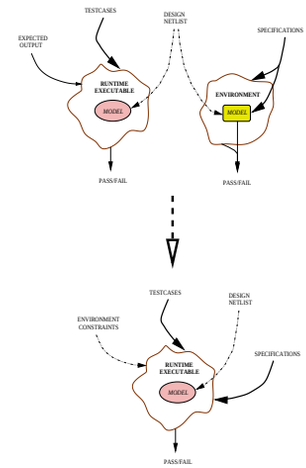

Fig. 4. Integrating Simulation And Model Checking

Fig. 4 depicts how the simulation and model checking are integrated into a single unified approach. In simulation, the model, derived from the design netlist, is used with test cases and the output is compared with the expected values for validity check. The runtime executable works like an operating system with the model. It is responsible for taking in test cases, feeding them to the model and making the output available to the checker.

Model checking uses a model, also derived from design netlist, and checks it against the specifications, for the specified environment. The unified approach uses all the same material as simulation, except for the outcome comparisons which are substituted by behavior check at every cycle to determine the correctness of the simulation run. *Sherlock* [2, 7], a platform for writing checkers, was used as the link between simulation and model checking. This platform uses CTL-like rules to check the output of the simulator. Specifications written in CTL could therefore be used for both model checking as well as simulation using Sherlock.

IV. Test Case Generation

The design used for experiments was the GL85 microprocessor circuit which is a clone of Intel's 8085 microprocessor but not pin-compatible with it. GL85 uses separate 8-bit input and output buses instead of an 8-bit bidirectional ADDRESS/DATA bus used by the 8085microprocessor.

Test cases for GL85 were generated in a modular fashion. The method has been depicted in Fig. 5. First vectors corresponding to each instruction were mapped against each instruction. A mapper file was used for this purpose. Then multiple instructions were grouped into sets depending on the types and categories. These sets were mapped on to parameters. The relationship between various signals was then noted and parameters assigned accordingly. Each parameter corresponded to multiple sets of instructions, belonging to the same category, but in different order.

The first part of each test case was a header that started with a RESET, followed by the disabling of the RESET and a few NOPs. This was followed by loads and stores for each register and PUSH and POP instructions for the stack pointer. After this general test, came the part that differed according to the specification being verified. For generating this part, first the signals in the specification were noted and their role in the specification was obesrved. Parameters were then assigned according to the causality relationship between signals in the specification. These parameters gave the sets of instructions to be used which in turn translated into vectors. The length of these test cases could be varied while mapping parameters to sets of instructions.

V. Experimental Platform

All experiments were done on RS6000 machines, with AIX 4.1.4 operating systems. The algorithms were implemented using C and Perl. The translator for environment and specifications was written using Lex and Yacc. The verification methods primarily used for experiments and comparisons were model checking and simulation. Simulation was used with both directed test cases and undirected test cases. A common model or prototype of the design was used for both simulation and model checking. The model checking tool uses the HIS compiler for obtaining the model from a behavioral description of the design.

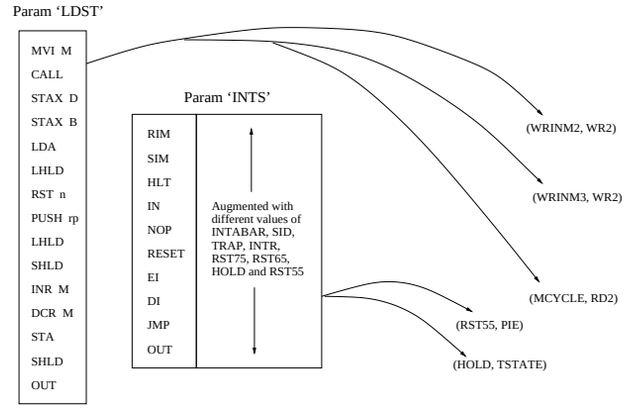

Fig. 5. Generating Test Cases For GL85

IBM's *Rulebase* was used as the model checking tool and *Mvlsim*, also from IBM, was used for simulation. Rulebase was developed in Haifa Research Laboratory in Israel. The specification language for Rulebase, called *Sugar* is is an extension of the Computational Tree Logic or CTL [1]. Though it has the same expressive power as CTL, it is more usable and accepts high-level constructs. Rulebase has its own Environment Description Language or EDL to describe the design environment. Mvlsim is a cycle simulator. Other than the design model, test cases, parameters and checkers, it uses a runtime-executable (RTX) that works like the operating system for the model and feeds the test case and parameters to it. Apart from initializing the hardware, it also links and calls the IVPC (Implementation Verification Program using C) or the checker. An event trace and simulation statistics are generated as outputs. Mvlsim's user interface allows it to run test cases individually as well as in batch mode.

VI. Results

Table I shows the specifications used and their brief description.

Two bugs were found in the design. These are depicted in Fig. 6 and are as follows.
(i) Specification IIHT: If the machine is in HALT state then a valid interrupt will always cause VINT to be reset and the interrupt to be processed
Violation of IIHT: This did not happen if VINT was set in the same cycle as HOLD even if before HOLD
(ii) Specification IIHM: If the machine is in last machine cycle of an instruction then a valid interrupt will always cause VINT to be reset and the interrupt to be processed

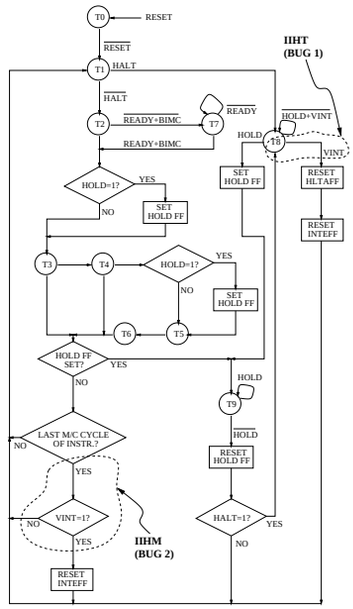

Fig. 6. Bug Report

TABLE I
SPECIFICATIONS

Spec	Description
ROIA	Reset On Interrupt Ack
TOPE	Trap On Priority Encoding
IOPE	Intr On Priority Encoding
IELC	Intr Enable Latch Check
PEOA	Priority Encode On Ack
MCHK	Mask Check
MUPD	Mask Update
RORW	Reset On Read Write
RLIT	RW Low In T3
RCHK	Readbar Check
LD	Load
ST	Store
ROTF	Reset On Tstates Flow
TUPD	Tstates Update
RWIO	Reset While Interrupt On
ICHK	Interrupt Check
ROIE	Reset On Interrup Enable
ECHK	Enable Check
IIHT	Interrupt In Halt Tstate
IIHM	Interrupt In Last Mcycle

Violation of IIHM: Design does not check VINT in the last machine cycle of each instr.

VII. Observations

The following tables show the time taken for verifying various specifications using the different methods. Columns named "M.C.", "R.S.", "D.S." and "Prop." respectively represent model checking, simulation with random test cases, simulation with directed test cases generated manually, and simulation using directed test cases generated using the proposed algorithm. The GL85 implementation that was used has 238 flip flops, 10084 gates and 18 inputs. The model checking tool, after optimizations, still had from 193 to 197 flip flops, apart from 18 to 23 environment variables and about 8000 gates, for most specifications.

The following tables depict the performance of each method. The specifications corresponding to each name are as given in Table I. Table II[1] shows CPU time in seconds, Table III[1] shows User time in seconds, Table IV[1] shows Elapsed time in seconds, Table V[1] shows memory used in MB, Table VI[1] shows the overall performance of the various methods; the second entry in case of IIHT and IIHM corresponds to the number of cycles to catch the bug. In the case of model checking, providing further conditions or overspecification could not be done since it did not prune or reduce the logic any further.

It was noted that the simulation time corresponding to failed specifications is much lower since the effect of cumulative errors causes the simulation run to end before the limiting number of cycles. The memory used in simulation is fixed and hence remains constant irrespective of the type of test case or the specification being verified.

VIII. Comparisons

From the tables given in the previous section, we see that our method has about the same performance as that of simulation with directed test cases, but uses very little time for generating test cases. In simulation, very little time is spent in comparing the values at each cycle compared to the cycle time. Therefore the CPU and user time remain more or less constant irrespective of the specification being validated and the test case being used, as long as the test case is longer than the MAX cycles specified to the simulator. Similarly, the memory used in simulation also remains constant. In considering the level of manual effort required for each method we disregarded the time required to code specifications since specifications and checkers were common for all methods and required almost the same time for each method. This is why model checking has been shown to require low effort as compared to simulation with directed test case.

Given enough space and time, model checking can find any bug. The same is true for simulation with directed test case. However, our purpose was to see if the tools

[1]These results were approximated using *time* and *usage* commands

TABLE II
CPU Time In Seconds

Spec	M.C.	R.S.	D.S.	Prop.
ROIA	> 85.6	23.72	22.55	22.22
TOPE	> 77.4	24.14	24.24	24.25
IOPE	> 77.5	22.22	23.78	22.18
IELC	> 77.9	18.28	19.28	19.20
PEOA	> 81.0	24.12	22.29	22.22
MCHK	> 81.9	22.18	21.19	20.88
MUPD	> 79.2	24.18	22.28	21.30
RORW	> 94.2	21.12	22.29	22.39
RLIT	> 94.0	20.20	21.30	21.29
RCHK	> 82.0	20.16	20.28	20.29
LD	> 91.0	22.28	22.39	22.30
ST	> 91.2	22.36	22.21	22.35
ROTF	> 81.5	22.38	22.38	22.35
TUPD	> 78.1	20.20	20.37	21.34
RWIO	> 78.6	20.98	19.36	20.46
ICHK	> 78.0	22.66	22.25	22.47
ROIE	> 79.4	23.10	22.36	22.65
ECHK	> 78.2	24.16	23.35	23.21
IIHT	> 77.9	20.16	6.27	6.25
IIHM	> 77.9	20.78	5.45	5.48

TABLE III
User Time In Seconds

Spec	M.C.	R.S.	D.S.	Prop.
ROIA	> 52.1	11.48	11.49	11.46
TOPE	> 46.4	11.28	11.36	12.11
IOPE	> 46.6	10.00	11.49	11.53
IELC	> 48.3	9.38	9.37	9.32
PEOA	> 54.6	10.68	10.37	10.32
MCHK	> 55.9	11.24	10.47	09.45
MUPD	> 42.4	11.22	11.34	11.36
RORW	> 58.1	10.10	11.39	11.39
RLIT	> 59.0	11.10	10.46	9.39
RCHK	> 46.1	9.48	9.35	10.37
LD	> 56.7	12.66	12.58	12.58
ST	> 57.9	12.90	12.58	12.5
ROTF	> 48.9	10.24	11.49	11.38
TUPD	> 49.6	12.68	9.38	9.47
RWIO	> 50.4	10.88	9.37	10.38
ICHK	> 48.4	12.50	11.47	11.47
ROIE	> 48.3	10.68	11.46	11.36
ECHK	> 48.4	11.90	12.48	11.36
IIHT	> 48.3	12.12	4.18	3.99
IIHM	> 48.4	12.14	4.78	3.98

can find bugs within set time constraints. Therefore we did not let the model checking tool run to completion but aborted the run whenever the process slowed down to very low speed because of the excessive memory being used. While model checking tool continued to run rules without reporting violations and simulation with random test cases did not search the right paths, simulation with directed test cases and our proposed method both reported violations within fairly reasonable time. Over directed test cases, however, our method has the advantage of requiring less time because of the reduction in the number of man hours required.

IX. Discussion

The GL85 design, being reasonably large when unpartitioned, stressed the model checking tool for even simple specifications. This happened because the model checking process involves a breadth first search of the states, whereas most specifications can be proved wrong, if not right, with a simple depth first search, provided we are looking into the right paths. This is the purpose of directed test cases. However, a directed search can be taxing in terms of effort and time. Our method fetches the advantages of directed search without the associated overhead.

The consistency checking and troubleshooting methods were applied to the specifications in trying to come up with the right rules for the specifications. An ad hoc method was used for biasing test cases for the GL85 design. However, somewhat more mature test case generators are expected to be available, for verifying large industrial designs. Hence in an actual life scenario it is expected to be fairly easy to bias the test cases using signal information as discussed earlier.

X. Conclusion

An algorithm was presented for automatically generating directed test cases using specifications. The method uses signal relationships and causality to generate the test cases. The technique was applied to the GL85 microprocessor and the results presented. These were compared with the results obtained by using other methods.

Acknowledgement

This work was supported in part by IBM under Project No. 08503. The specifications for GL8085 were provided by Alexander Miczo.

TABLE IV
ELAPSED TIME IN SECONDS

Spec	M.C.	R.S.	D.S.	Prop.
ROIA	> 432837	98.14	96.12	94.12
TOPE	> 455638	102.57	103.57	105.99
IOPE	> 454778	101.28	102.04	105.40
IELC	> 455050	84.68	82.35	82.46
PEOA	> 466960	98.28	97.44	95.66
MCHK	> 432666	94.24	93.44	93.44
MUPD	> 444160	92.48	98.66	98.10
RORW	> 429469	96.34	97.39	96.39
RLIT	> 459421	94.90	94.98	95.99
RCHK	> 465998	90.12	92.15	91.17
LD	> 465896	92.18	94.58	93.77
ST	> 456322	92.38	92.66	93.44
ROTF	> 444112	93.46	94.26	93.76
TUPD	> 432000	93.20	91.38	92.11
RWIO	> 442180	92.10	91.22	91.20
ICHK	> 436118	93.80	94.25	94.66
ROIE	> 408108	97.88	96.66	96.49
ECHK	> 432567	102.48	102.10	102.0
IIHT	> 467124	108.28	57.78	57.44
IIHM	> 467927	110.26	48.14	48.10

TABLE V
MEMORY USED IN MB

Spec	M.C.	R.S.	D.S.	Prop.
ROIA	> 117	5.189	5.189	5.189
TOPE	> 115	5.189	5.189	5.189
IOPE	> 115	5.189	5.189	5.189
IELC	> 115	5.189	5.189	5.189
PEOA	> 117	5.189	5.189	5.189
MCHK	> 117	5.189	5.189	5.189
MUPD	> 116	5.189	5.189	5.189
RORW	> 121	5.189	5.189	5.189
RLIT	> 121	5.189	5.189	5.189
RCHK	> 119	5.189	5.189	5.189
LD	> 120	5.189	5.189	5.189
ST	> 120	5.189	5.189	5.189
ROTF	> 117	5.189	5.189	5.189
TUPD	> 116	5.189	5.189	5.189
RWIO	> 116	5.189	5.189	5.189
ICHK	> 116	5.189	5.189	5.189
ROIE	> 116	5.189	5.189	5.189
ECHK	> 116	5.189	5.189	5.189
IIHT	> 116	5.189	5.189	5.189
IIHM	> 116	5.189	5.189	5.189

REFERENCES

[1] Ilan Beer, Shoham Ben-David, Cindy Eisner and Avner Landver, "Rulebase: An Industry-oriented Formal Verification Tool", *Design Automation Conference*, 1996.

[2] W. Canfield, E. A. Emerson and A. Saha, "Checking Formal Specifications Under Simulation", *IEEE International Conference on Computer Design*, October 1997.

[3] Y. V. Hoskote, J. A. Abraham and D. S. Fussell, "Automated Verification of Temporal Properties Specified as State Machines in VHDL", *Proceedings Fifth Great Lakes Symposium on VLSI*, March 1995.

[4] Y. Hoskote, J. Abraham, D. Fussell and J. Moondanos, "Automatic Verification of Implementations of Large Circuits Against HDL Specifications", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, March 1997.

[5] D. Moundanos, J. Abraham and Y. Hoskote, "A Unified Framework for Design Validation and Manufacturing Test", *Proceedings IEEE International Test Conference*, October 1996.

[6] S. P. Rajan and Masahiro Fujita, "Integration of High-level Modeling, Formal Verification, and High-level Synthesis in ATM Switch Design", *IEEE International Conference on VLSI Design*, January 1998.

[7] N. Saxena, J. Baumgartner, A. Saha and J. Abraham, "To Model Check Or Not To Model Check", *ICCD*, October 1998.

TABLE VI
OVERALL COMPARISON

Feature	M.C.	R.S.	D.S.	Prop.
CPU Time	High	Low	Low	Low
User Time	High	Low	Low	Low
Memory Usage	High	Low	Low	Low
Bugs Found	No	No	Yes	Yes
Manual Effort	Low	Low	High	Low