An Automatic Test Generation Algorithm for Hardware Description Languages

Forrest E. Norrod
Hewlett Packard
3404 Harmony Road
Fort Collins, CO 80525

Abstract

A new approach to test generation from Hardware Description Language circuit models has been developed and implemented. The described *E-algorithm* generates tests for control, operation, and data faults in sequential and combinational logic modeled at the functional level. A symbolic variable notation is introduced to permit systematic fault propagation through control structures. Results of the implementation are given for a set of test cases and the application of the algorithm to a semi-custom ASIC are discussed.

Introduction

Traditional test generation techniques for digital circuits have been rendered inadequate by the increasing levels of integration achieved by VLSI technology. As the number of active components on a chip reaches the hundreds of thousands, the computational cost of using gate level automatic test pattern generation (ATPG) algorithms becomes prohibitive. Adequate tests for these complex circuits must, however, still be devised. This paper details an ATPG technique, the E-algorithm, that accepts functional-level circuit models described using a Hardware Description Language (HDL). A fault model has been developed that addresses data path faults, faults in control structures, and faults in functional operators. The E-algorithm is able to generate tests for all modeled fault types, and handles a wide variety of circuit types, including sequential circuits. The algorithm has been implemented; initial results are given.

Previous Research

As the computational costs of gate-level test generation have grown excessive, designers and test engineers have turned to other testing methods. Often, these techniques are somewhat ad-hoc such as: write test vectors to "toggle every circuit node". These approaches require a large investment in engineering time and yet fail to guarantee adequate fault coverage. A popular alternative is to add logic that makes the circuit easier to test; this method is particularly useful for sequential circuits.

In recent attempts to devise systematic ATPG systems, functional approaches to circuit description and test generation have been examined. In these functional approaches, the circuit input to the ATPG system is described at the behavioral level rather than structural level, reducing the amount of detail which must be considered during the design and test process. Test generation with such models requires a fault model based on aberrations in the circuit function rather than structural defects. Faults are modeled as perturbations in the functional or control blocks of the circuit.

Levendel and Menon [1] proposed extensions to the gate-level D-algorithm to handlen faults in functional blocks of HDL's. Their approach gives strategies for D-propagation through functional blocks, but has an high computational cost for complex control structures. Several other approaches to test generation for functional circuit models have been presented [2 - 8,16] with varying results. A number of these approaches have produced test vector suites with gate-level fault coverages in the 95 - 100% range. However, a drawback of most functional level ATPG systems is the lack of a clearly-defined fault model, making functional fault coverage estimates difficult to produce. Barclay and Armstrong [9,12] devised an ATPG method for HDL circuit models which incorporates a *Chip-level* fault model; the method was implemented as a heuristic goal-oriented ATPG system. Their technique generates tests for faults in control structures as well as data path faults, but is computationally expensive and can't handle reconvergent fanout. O'Neill analyzed and eliminated some of these limitations [10].

Test Generation Approach & Circuit Models

The test generation approach detailed herein is extension of the D-algorithm [11] that generates tests for digital circuits described by a non-procedural HDL. A graph transformation is applied to the HDL circuit model to yield a representation that eases the description and implementation of the E-algorithm. This representation also forms the basis for a fault model for control faults.

The process of generating a set of test vectors for a given fault in a circuit can be broken into three main steps. The first step, *Sensitization*, is where the effect of the fault is made to manifest itself at the fault site. The second step is *Fault Propagation* where the effect of the fault is moved through the circuit until it can be observed

The research described in this paper was supported in part by Control Data, IBM, MCC, NSF, and the Virginia Center for Innovative Technology.

at an output. The third step in the procedure works the signals required to establish the propagation path back to the inputs and is called *Justification*.

The E-algorithm employs the standard notation, using the logical values 0,1,X,D,$\bar{D}$ where D is herein assumed to = 1 (good signal value) and 0 (faulty value). During test generation, the *D-frontier* lists the signals that manifest the fault as propagation proceeds. A set of symbolic variables E and $\bar{E}$ are introduced to propagate fault syndromes through control structures. A further extension to the standard notation is mandated by the properties of sequential circuits. Each signal assignment or justification requirement is given a time tag to specify when the assignment is done or the signal needed.

A subset of VHDL (VHSIC Hardware Description Language) is used to model digital circuits. Circuit descriptions are limited to non-procedural data-flow representations; these consist of groups of data signal assignments and transformations using functional blocks such as AND, and ADD operators. Signals can be of the type BIT or BIT-VECTOR, from which other types can be formed. Control statements such as IF..THEN..ELSE and CASE are used to govern the flow of execution. Algorithmic structures such as FOR loops or REPEAT...UNTIL blocks are excluded.

A simple timing model is employed to permit sequential behavior. All sequential logic is assumed to be synchronous. Inputs are set at the beginning of a master clock period and outputs sampled at the end. Explicit signal transitions are modeled with the signal'STABLE boolean operation which is false immediately after signal changes value. Thus, the use of the 'STABLE operation implies two time periods. A typical use of 'STABLE is to model clocked logic with statements such as: IF CLOCK AND NOT CLOCK'STABLE THEN ... where assignments under control of the IF will execute on the leading edge of CLOCK.

Graph Transformation & Fault Model

A data-flow graph of the circuit is derived from the VHDL description. The graph is composed of operation blocks corresponding to the basic data functions (eg. NOR or SUBTRACT) specified in the VHDL description. Operation blocks are interconnected by data paths, data buffers, and control lines. Internal signals are transformed into signal blocks. Each assignment in the VHDL model to an internal signal is connected through a data buffer to the input of the corresponding graph signal block. The data buffers each have a control port that regulates the flow of information into the connected signal block. Whenever the graph branch attached to the buffer control port has a logic value of '1', the data at the input of the buffer is loaded into the signal block. The control branch will have a '1' value when the conditions controlling the data assignment in the VHDL model are met. See Figure 1 for an example of VHDL-to-graph transformation.

The fault model for the E-algorithm considers two types of faults: defects in functional operation blocks, and stuck-at faults in data or control lines. Each operation block is faulted by having it fail to some other operation; the current implementation faults an operation to its dual or uses a user-defined fault syndrome table which allows arbitrary faults to be modeled. Research has been conducted to find what operation substitution(s) yield the best fault coverage [13].

Data and control lines are faulted by impressing stuck-at faults on the lines. For data paths, these faults usually affect the entire path, ie. all bits in a vector-wide data line are faulted to all 0's, all 1's, and alternating 0-1 patterns. Control faults are modeled as stuck-at-1 and stuck-at-0 faults on control branches. A stuck-at-0 fault on a control branch will prevent data transfers through the attached control buffer(s). In the VHDL description this corresponds to a assignment statement being prevented from executing. Similarly, a stuck-at-1 fault on a control branch causes the associated data transfer(s) to occur at every time period, thus modeling assignment statements which execute continuously.

VHDL

```
IF A XOR (B AND C) THEN
  COM <= DT1;
ELSE
  COM <= DT2;
ENDIF;
OUTPORT <= COM;
```

Graph Model

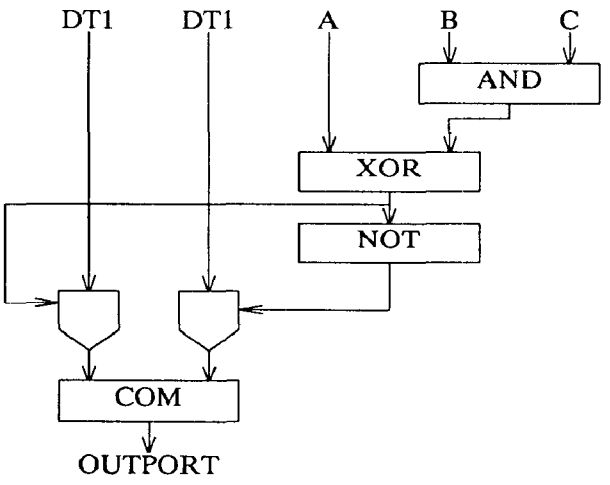


Figure 1. VHDL to Graph transformation.

Sensitization & Propagation

For a test to be generated for a given fault, the fault site must be sensitized so that at least one circuit line or block produces or carries a value containing D or $\bar{D}$. Sensitizing the fault is accomplished by choosing from a fault table a primitive cube that specifies what conditions must be set at the inputs to the faulted line or block to produce the given faulty value. The faulty output of the block or line is a set of values known as the initial D-vector for the fault.

After the fault has been sensitized, unless the fault occurs at a circuit output, the effect of the fault (the fault syndrome) must be moved to an output. Propagating the fault syndrome consists of moving the D-vector of the fault through the graph until an output is reached. The D-vector is driven through each successive data operation or signal block in the path by using propagation rules proposed by Levendel and Menon [1]. Propagating fault syndromes through control branches is discussed in the next section.

The E-algorithm uses a derivative of the D-algorithm to propagate fault syndromes through data structures (function or signal blocks):

1. Sensitize the fault: select fault cube; set D-frontier to initial D-vector; set justification list to the requirements of primitive fault cube; assign initial time-tags to the D-vector and justification list.
2. At limits of the D-frontier, select the next object(s) in path(s) to be propagated through. This selection is based on stored observability measures for each graph block. Typically, single-path propagation will be tried first, but if no single consistent propagation path can be found then multiple paths will be examined.
3. Propagate the fault syndrome through the selected object by intersecting the D-frontier and its time-tags with the propagation requirements of the object. This creates a new D-vector at the outputs of the object. If propagation is through the data path of a control buffer (and under a 'STABLE), then a new time tag is created for the new D-vector.
4. Check propagation requirements for consistency with justification list. If inconsistencies arise, attempt to reschedule data signal loads. Also check signal values against an *Illegal conditions* table. The illegal conditions table may be used to enumerate illegal combinations of state lines, thus preventing the system from wasting time generating a test using a set of values which cannot be justified.
5. If no propagation cube is consistent, try another path. GOTO 2.
6. Add propagation requirements to propagation list, construct a new D-frontier.
7. If primary output has been reached, resolve justification requirements back to primary inputs and STOP - test has been generated. (If justification fails, backup and try another path. GOTO 2)
8. If a primary output has not been reached, continue propagation from new D-frontier. GOTO 2.

Whenever a path turns out to be a dead-end, the algorithm backs up to the last untried path in the circuit and propagates through that path. The choice of paths is guided primarily by observability measures. The observability measures are estimates of the difficulty in reaching the output along a certain path; rules similar to those discussed by Fong [15] are used by a pre-processor to generate the observability of each circuit object.

Faults are propagated through simple logic elements such as AND and OR gates by intersecting the D-frontier with the gate's propagation cubes. Propagating fault syndromes through more complex operation blocks may be accomplished with simple D-cube intersection [1], but in a few cases (ex. DIVIDE and multi-bit PARITY functions) a model of the operation is simulated to derive the propagated syndrome (D-vector) and justification requirements. The object's propagated syndrome and justification requirements are assigned time tags that specify when they are available or required. The time tags will be the same as the tag of the D-frontier elements at the input of the object unless the object is a control buffer or line under a 'STABLE operation. In these cases, at least two time periods are implied and new tags (set to be before or after the D-frontier elements as appropriate) are created for the signals.

Once the fault syndrome reaches a primary output, a complete propagation path has been formed and the justification algorithm is invoked. If an irresolvable inconsistency is detected during justification, then the propagation routine tries to find a different propagation path. This can be, of course, rather expensive and a focus of future research is implementing a set of guidelines for pin-pointing the source of irresolvable conflicts.

Consistency checks are performed during propagation to check for direct contradictions (trying to set a line to '0' and '1' simultaneously for example), and also check for signal combinations which are explicitly defined a priori as illegal. This feature is used primarily to identify illegal states in state machines. Without this feature it is possible that the system will try generating a test where an illegal state is to be justified on state lines. This condition could endless looping during justification, searching in vain for an initialization sequence.

Example

Figure 2 shows a VHDL fragment and its associated graph. A fault has occurred which prevents the first clause in the CASE statement from ever executing (a DEADCLAUSE fault). In the graph model, this fault corresponds to the control branch being stuck-at-0. This fault prevents the connected data buffer from loading the signal *C*. *OP* and *G* are primary outputs, while *A*, *B*, *H*, *P*, *OC*, *CK*, and *CD* are primary inputs.

To generate a test for this fault, the fault is sensitized by setting the output of *eql* to 1 at an initial time tag; this produces a *D* on the faulted line. Propagation of the fault syndrome begins by finding the most observable object with an input connected to the faulted branch. In this case, there exists no path through data paths - the only route is through the control port of *buf1*. Unfortunately, there is no way for the fault syndrome to directly affect the data in the signal block, thus the effect of the fault can not be propagated by using D-propagation rules and a path for the syndrome must be found some other way.

Propagation Through Control Branches

When a D-vector reaches a control branch, the data path propagation algorithm invokes the control branch propagation process to find a path for the fault:

1. Select a signal block connected to a data buffer whose control port is attached to the branch exhibiting the fault syndrome.
2. Load the selected signal block with a vector (as wide as the signal block) containing subscripted E-bits. The subscripts on the E bits indicate the bit's position in the signal block.
3. Propagate the E-vector towards the circuit outputs using the D-propagation rules, but with 'E' values instead of 'D' values.
4. When the E-vector reaches the output or a second unavoidable control branch, invoke the *E-Justification* routine to select a value or sequence of values to be instantiated for the E-bits in the original selected signal block.

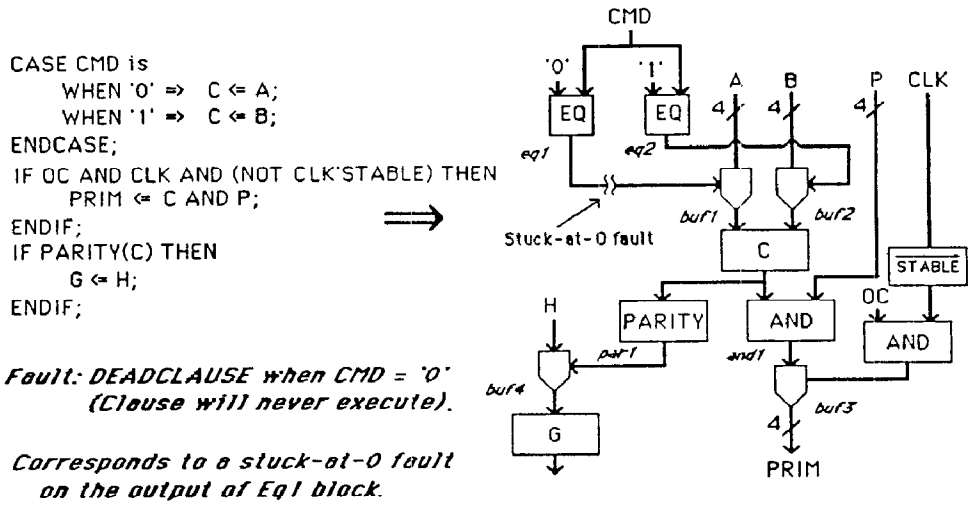


Figure 2. Example of control branch fault.

The E-vector is used to establish a propagation path for the fault syndrome from the signal block under the faulty control branch. As the E-vector is propagated from the signal block, it accumulates information on the effect that values in the signal block have on other circuit elements. Therefore, when the E-vector has reached an output, the output value can be specified as a function of the E-bits of the E-vector and other signal. The effect of a certain value in the selected data block upon the output can now be discerned.

To observe the fault syndrome, conditions must be established so that when the fault is present one value is loaded into the signal block, and when no fault exists a different value is loaded. Values are chosen after examining the E-vector at the output to determine which bits of the signal block reach the output and in what form. As different signal block values may produce identical output values, an inequality satisfaction routine is applied to the E-vector at the output to determine what E-bit values in the signal block will produce different outputs. The E-Justification routine chooses the values to be loaded into the block. The sequence of values to be instantiated for the E-vector at are chosen depending on the topology of the faulty control branch and signal block. Figure 3 shows the 5 basic cases of propagation through control branches and their E-load strategies. Note in Figure 3 that E and $\bar{E}$ correspond to the two different actual values (derived from the inequality satisfaction routine) loaded into the bits of the E-vector which reach an output.

In most cases sequences of values are loaded into the E-vectors as a consequence of the nature of control faults. If a single signal block has separate load buffers, with one controlled by a branch with a value of 'D' and another with a control branch value of '$\bar{D}$' (Figure 3.a) then a test that would detect this fault consists of placing one value on the input to one buffer, and the inverse value on the input to the other buffer. The value of the output will indicate which load occurred, and thus the presence or absence of the fault. However, if the D-frontier of the fault doesn't reach at least two buffers of the same signal block, then this strategy won't work. Instead, these fault cases are covered by checking if a load under control of the faulty branch can be properly executed. A known value is loaded into the selected signal block, and the output then observed for a change or non-change in the value produced by faulty control branch. Figures 3.b - 3.e show these cases. If the faulty control branch is the only one that can be used to load the signal block, then it is first assumed to be fault-free, a preload value loaded, and then a test for incorrect control branch value made.

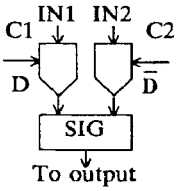
Armed with the E-propagation and justification routines, we may now return to the example of Figure 2. The D @ $t0$ at the input to *buf1* causes C to be loaded with $E_{c(1-4)}$. This E-vector is now propagated using data path rules. The most observable object under C is *and1*; *par1* is not chosen as it requires propagation through another control branch. $E_{c(1-4)}$ is intersected with the propagation cubes of *and1*. A '1' in any bit of P would allow the syndrome to propagate through *and1* but assume here that the cube chosen for intersection requires P = '1111' and yields $Output(and1) = E_{c(1-4)}$ @ $t0$. Only one route is available for further propagation at this point, so $E_{c(1-4)}$ @ $t0$ is intersected with the cube of *buf3*, giving $OP = E_{c(1-4)}$ @ $t1$. Note the output has now been expressed as a function of the E-vector $E_{c(1-4)}$. The new tag $t1$ is produced since *buf3* is under a of CK.

As the E-vector has reached an output, the E-Justification procedure is invoked. The topology of the C signal block / data buffer where $E_{c(1-4)}$ is loaded corresponds to that of Figure 3.b. With the appropriate substitutions (eg. $IN1 = A$, $IN2 = B$ etc.), the sequence of Figure 3.b is used and the values for the E-vectors are chosen based on the constraints accumulated during E-vector propagation. In this case the choice is simple: any two different values ex.: E_c = '1111', $\bar{E}_c$ = '0000'. After justifying all required signals, a complete test for the modeled fault is given by:

tag	CK	CD	A	B	P	II	OP
t0	0	1	xxxx	1111	1111	x	- / -
t1	1	1	xxxx	1111	1111	x	1111/1111
t2	0	0	0000	xxxx	1111	x	1111/1111
t3	1	0	0000	xxxx	1111	x	0000/1111

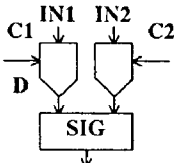
If the E-vector encounters another control branch instead of an output, E-Justification of the vector proceeds as before and then a signal block under control of the encountered control branch is loaded with a different E-vector, Enew. E-propagation is then reinitiated. However, since the fault syndrome at the new control branch isn't given by D or $\bar{D}$, different

strategies must be used during E-Justification to instantiate the E_{new} vector values when the output or next control branch is reached. The values are chosen based on the sequence of explicit values the previous E-vector induced at the new control branch. Figure 4 lists the possible control branch conditions and the corresponding E_{new} load strategies.



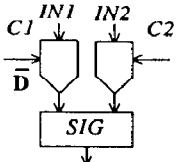
Tag	C1	C2	IN1	IN2	Out f(SIG)
t0	1	0	E	Ē	E / Ē (good/faulty)

Figure 3.a



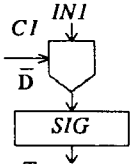
Tag	C1	C2	IN1	IN2	Out f(SIG)
t0	0	1	X	E	E / E(preload)
t1	1	0	Ē	X	Ē / E(test)

Figure 3.b



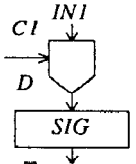
Tag	C1	C2	IN1	IN2	Out f(SIG)
t0	0	1	X	E	E / -(preload)
t1	0	0	Ē	X	E / Ē(test)

Figure 3.c



Tag	C1	IN1	Out f(SIG)
t0	1	E	E / E(preload)
t1	0	Ē	E / Ē(test)

Figure 3.d



Tag	C1	IN1	Out f(SIG)
t0	1	E	E / -
t1	1	Ē	Ē /unchanged from t0

Figure 3.e

Figure 3. E-Justification strategies for the five basic control branch / signal block topologies.

An example of propagation through a second control branch can be seen from the example of Figure 2: If the path through *buf3* were removed, then the fault syndrome would have to propagate through the control branch of *buf4*. The E-vector value at the output of par1 can be given as $Output(par1) = E_{c1} \oplus E_{c2} \oplus E_{c3} \oplus E_{c4}$. When the inequality routine is invoked, values for the $E_{c(1\text{-}4)}$ vector loads are chosen to meet this condition, ex: A = 0001 and B = 0000. This yields the sequence at the input to *buf4* of: *0/0 @ t1 - t2; 1/0 @ t3*. In order to match this syndrome (given in Figure 3.b) with those given in Figure 4, the value *1/0* is to be held for two more time periods, giving *1/0 @ t3-t5* and matching the first strategy in Figure 4. This strategy requires the E_{new} vector to be loaded into G and propagated to an output (easy enough here). The values instantiated for E_{new} and $\bar{E}_{new}$ are chosen so as they differ (here just any two different values). Thus, H (the load path) will be justified to: $H = E_{new}$ @ *t3, t5*; $H = \bar{E}_{new}$ @ *t4*. The output will follow the same sequence and the syndrome can be observed.

The strategies given in Figure 4 apply in cases where the new control branch controls a buffer that may be loaded with different values. If, however, the branch controls the loading of a literal (eg. '00'), then the previous E-vectors are altered to produce a continuous '1/0' or '0/1' syndrome, effectively a D or $\bar{D}$, and the strategies of Figure 3 followed.

Time	Sequence at Branch	E-Vector Sequence	Output is function of
t0	1 / 0	$\underline{E}_{new}$	$\underline{E}_{new}$ / -
t1	1 / 0	E_{new}	E_{new} / -
t2	1 / 0	E_{new}	E_{new} / -
t0	1 / -	E_{new}	E_{new} / -
t1	0 / -	$\underline{E}_{new}$	$\underline{E}_{new}$ / -
t2	1 / -	E_{new}	E_{new} / -
t0	1 / 1	$\underline{E}_{new}$	$\underline{E}_{new}$ / E_{new}
t1	1 / 0	E_{new}	E_{new} / E_{new}
t0	1 / 1	$\underline{E}_{new}$	E_{new} / $\underline{E}_{new}$
t1	0 / 1	E_{new}	E_{new} / E_{new}

Figure 4. E-Justification strategies for successive control branch propagation

Justification

Signal values required to sensitize the fault, establish the propagation path, or load E-vectors are moved to the circuit inputs with the justification procedure. Justification proceeds in the standard manner; signals are worked backwards by intersecting them with the singular covers of operation blocks. If different paths for justifying a signal are available, the more controllable is chosen. If a signal is justified back through a signal block and buffer, then the value of the signal must be available at the buffer and loaded into the block at *or before* the time the value is needed at the output of the signal block. If conflicts develop during justification, the system attempts to re-schedule signal block loads to eliminate the contention. When all required values have been moved back to the inputs and no inconsistency has arisen, then a test for the fault has been found.

Implementation and Results

The E-algorithm has been implemented in PASCAL running in a MS-DOS shell on a 25 MHz 80386 workstation. The complete system includes of a pre-processor that derives the graph from the VHDL model and calculates the controllability and observability measures. A fault list is generated for the circuit that includes faulting each operation block and placing stuck-at faults on all control and data lines. Tests are generated for each fault on the fault list except for faults in 'STABLE operation blocks which cause continuous clocking.

Tests for several large MSI circuits have been generated, including a registered ALU and an 8-bit controlled counter. The ALU model used is equivalent to the positive logic portion of a 74181, but with output

registers added. Tests for all 127 chip-level faults were generated in 72 seconds of CPU time; this figure includes the time consumed during pre-processing. The 8-bit controlled counter produced tests for 74 modeled faults; tests for two faults failed due to poor controllability and 6 faults were excluded as they produced continuous clocking. The total CPU time consumed during pre-processing and test generation was 110 seconds.

Preliminary comparisons of generated test sets with gate-level implementations of the circuits indicate gate-level fault coverage on the order of 94%. Complete test results and circuit models are found in [14].

More recently, a test set was generated for a Hewlett-Packard gate array, the BLENDER chip, to serve as a production test set. The BLENDER gate array is a 1400 gate chip used in a high-performance 3-D superworkstation to digitally blend multiple high-resolution images in real time. The BLENDER chip contains a number of complex functional blocks, including three carry-look-ahead adders. The BLENDER contains 157 modeled, non-dominated and non-equivalent chip-level faults. A test vector set to cover those faults was generated in 537 CPU seconds, including preprocessing and graph compilation time. A test suite of 143 vectors was produced after redundant vector elimination and compacting. The vectors were fault graded on a HILO-3 system and yielded equivalent gate-level stuck-at fault coverage of 93%. All undetected faults occurred in the adders, the adder fault model is being updated to increase the fault coverage.

The algorithm performs well on complex sequential circuits if the controllability of internal registers is good. It does, however, suffer from a common limitation of ATPG systems: when controllability of registers is poor, test generation performance suffers. This limitation is the result of the lack of high-level state transition information, the system may have to exhaustively search to find the sequence of inputs to move internal registers to a state required for test generation. The addition of higher-level state transition information (ie. a library of initialization or homing sequences for circuit state machines) would greatly expedite test generation in many cases involving sequential machines.

Conclusions

This paper presented the E-algorithm, a method for generating tests for general logic circuits modeled using a Hardware Description Language. A fault model based on faulting structures in a graph derived from the HDL model, and a test generation algorithm that generates tests for the modeled faults have been developed and implemented. The E-algorithm has the ability to propagate fault syndromes through high-level control structures as well as data operations. Moreover, tests for sequential logic can be generated. Preliminary results are highly encouraging, yielding high fault coverages and reasonable performance. Future work will attempt test generation for larger circuits, and examine the problem of embedding state-transition knowledge.

Acknowledgements

The author would like to thank Dr. James Armstrong for his support and guidance. Mike O'Neill, David Miller, and Erann Gat all provided many helpful comments. Joel Gengler assisted in the generation of vectors for the BLENDER chip. Finally, the support of GTD management is greatly appreciated.

References

[1] Y. Levendel and P. Menon, "Test Generation Algorithms for Computer Hardware Description Languages,"*IEEE Trans. Comput.*,C-31, July 1982, pp.577-588.

[2] M.A. Breuer and A.D. Friedman,"Functional Level Primitives in Test Generation,"*IEEE Trans. Comput.*,C-29, March 1980, pp. 223-235.

[3] B.M. Huey and F.J. Hill,"Fault Test Generation Using A Design Language,"*Proc. Symp. CHDL*,1975, pp.91-95.

[4] S.Y.H. Su and T. Lin, "Functional Testing Techniques for Digital LSI/VLSI Systems," in *Proc. 21st Design Automation Conf.*, pp 517-528, June 1984.

[5] C. Liaw, S.Y.H. Su, and Y.K. Malaiya,"State Diagram Approach for Functional Testing of Control Section,"*Proc. Intl. Test Conf.*, 1981, pp.433-446.

[6] T. Lin and S.Y.H. Su, "VLSI Functional Test Pattern Generation - A Design and Implementation," *Proc. Intl. Test Conf.*, 1985, pp. 922-929.

[7] ----,"The S-Algorithm: A Promising Solution for Systematic Functional Test Generation,"*IEEE Trans. CAD*, CAD-4, July 1985, pp. 250-263.

[8] S.Y.H. Su and Y. Hsieh,"Testing Functional Faults in Digital Systems Described by Register Transfer Language,"*Proc. Intl. Test Conf.*, 1981,pp. 447-457.

[9] D. Barclay and J.R. Armstrong,"A Heuristic Chip-Level Test Generation Algorithm,"*Proc. 23rd DAC*,1986, pp. 257-262.

[10] M. O'Neill,"An Improved Chip-Level Test Generation Algorithm", MS thesis, Dept. of Electrical Engineering, VPI & SU, Blacksburg, Jan. 1988.

[11] J.P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, vol. 10, pp.278-291, July 1966.

[12] D. Barclay, "An Automatic Test Generation Method for Chip-Level Circuit Descriptions," MS thesis, Dept. of Electrical Engineering, VPI & SU, Blacksburg, Jan. 1987.

[13] C.H. Chao and F.G. Gray, "Micro-Operation Perturbations in Chip Level Fault Modeling",*Proc. 25th DAC*, 1988,pp. 579-582.

[14] F.E. Norrod, "The E-Algorithm, an Automatic Test Generation Algorithm for Hardware Description Languages," MS thesis, Dept. of Electrical Engineering, VPI & SU, Blacksburg, February 1988.

[15] J.Y.O. Fong, "On Functional Controllability and Observability Analysis", *Proc. Intl. Test Conf.*, 1982, pp. 170-175.

[16] S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors", *IEEE Trans. on Computers*, June 1980, pp. 429-441.