Hierarchical Test Generation under Intensive Global Functional Constraints

Jaushin Lee and Janak H. Patel
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

Abstract

In hierarchical test generation, the test vectors for the low level structure of the module under test are computed and then justified at a high level. In the module test computation procedure, a low level ATPG tool is conventionally applied to the complete structure of that particular module without adding extra information. Due to the global circuit functional constraints applied to the inputs of that module, many of the test vectors being computed are not justifiable at the high level. Therefore high efficiency cannot be achieved without managing the functional constraint problem in the hierarchical ATPG process. In this paper, two schemes, valid control code abstraction and test cube justification, are proposed to overcome the global functional constraint problem and achieve very high efficiency in test computation. The proposed algorithms have been implemented in our hierarchical ATPG package and promising experimental results have been derived. We conclude that global functional constraints can be efficiently avoided through these techniques.

1. Introduction

Many novel approaches have been proposed for gate level sequential circuit test generation [1-4]. These approaches remain at gate level without looking at any of the inherent functional information in the sea of logic gates. The increase in VLSI circuit size will soon degrade the efficiency of the gate level ATPG approaches. The hierarchical circuit design style has been widely adopted in the design of modern VLSI systems. A number of approaches have been proposed to generate tests hierarchically and exploit high level functional information in circuits to speedup the test generation process [5-8]. However, the ATPG algorithms used in these approaches are extended from D-algorithm [10] or POEDIT [11], so that the module partition cannot go much further than gate level. In [6], the module partition could be at a higher level, but it should be flattened to gate level if any interior stuck-at fault is considered. Therefore, the speedup of these approaches over gate level ones is limited. The problem of system level functional constraints for these approaches does not exist because the module under test either contains very few logic gates, or is flattened to gate level. The functional constraint problem will be explained in more detail in the next section. Sarfert, et. al. updated SOCRATES[9] by including higher level modules which implement non-trivial functions, like multiplexer, decoder, etc [10]. However, their approach still adopted single bit-level signals at the high level and the speedup is limited. Kunda, et. al. take advantage of both higher level primitives and bit-vector representation of signals to speedup the test generation process [11]. Murray and Hayes[12] assume the test set for module under test is precomputed, and again the global functional constraint problem has not been addressed in any of them. Anirudhan and Menon[13] have addressed this problem in their paper. Symbolic constraints are derived in the top-down approach and the module tests are computed such that these constraints could be avoided. However, for complex circuits the symbolic constraint representation could be computational prohibitive, and the algorithm of computing module tests under such constraints is non-trivial. Lee and Patel have proposed a relaxation-based test generation algorithm at an architectural level. The global functional constraints were manually derived and put into the module test computation process [14].

In this paper, we propose two approaches to overcome architectural level functional constraint problem in hierarchical test generation. We separate these constraints into control constraints and bus constraints. For each module, the control input lines can be unambiguously separated from the bus input lines based on high level functions. Bus lines carry input data for either transferring or computation, and control inputs determine the transferring paths or computation functions. For control constraints, valid control codes occurring at the control lines of the modules of interest are determined by searching through control words of machine instruction. These valid control codes are stored in a table which are then intersected in the module test computation process. For bus constraints, we observed in some circuits that the relations among input buses might be very complex, and it is not practical to derive these exact relations given reasonable amount of time. Instead of deriving the exact bus constraints, we try to justify the partially specified test vectors at the high level. A partially specified vector has several of its bits set to value $X$ (don't care). Due to the fact that the bus constraints are less likely to be violated in most partially specified vectors, this approach is expected to be effective in avoiding bus constraints.

This research was supported by the Semiconductor Research Corporation under Contract SRC-91-DP-109.
In this paper, we assume that circuits are represented in two levels of hierarchy, that is, the architectural level and gate level. The relaxation-based architectural level test generator, ARTEST [14], and a gate level ATPG are used at these two levels. The contents of this paper are outlined as follows. Some preliminary definitions are given in Section 2. Section 3 introduces the derivation of control constraints and the constant value bus constraints, and the modification of gate level test generation algorithms. The approach of justifying partially assigned test vectors will be described in Section 4. Some experimental results will be shown in Section 5. Section 6 is followed by conclusions.

2. Definitions

In a hierarchical circuit design environment, some information barriers exist between levels of hierarchy. A phenomenon termed hierarchy blindness has been introduced by Lee and Patel [15], which illustrates two different kinds of information barriers. The first type of hierarchy blindness occurs when the low level circuit structure is not available to the high level. In this paper, it is assumed that the low level structure of the module under test is available. However, the complete low level structure of the circuit is never required in our approach. We are concentrating on the second type of hierarchy blindness, in which the high level architectural constraints are not visible to low level test computations. Figure 1 is an example of the second type of hierarchy blindness where only the four functions of an ALU are valid, when it is used as a module in a large circuit. In this case, test vectors for the ALU requiring functions other than these four ones will never be justified at the system level. These vectors are sometimes called "can't happen don't care" in the logic synthesis literature. The high level architectural constraints applied to the inputs of the module under test are termed global functional constraints for that particular module. The global functional constraints for a module can be subdivided into control constraints and bus constraints for control lines and bus lines. For the ALU in Figure 1, any module test requiring a function other than ADD, SUB, AND, and OR, or requiring a value at bus A not equal to 15 plus value at bus B can never be justified at the system level. Therefore, a gate level test generator must take into account these constraints when generating test vectors for the ALU. In this paper, only data path faults are considered. All control constraints derived are from the valid behaviors of the control machine.

An n-dimensional vector can be represented as \( V = <v_1, v_2, \ldots, v_n> \). Each \( v_i \in \mathbb{R} \), and \( \mathbb{R} \) is the range of data space at certain level of hierarchy. If considering gate level, \( R = (0,1) \). In logic simulation and test generation at gate level, another value \( X \), unassigned don't care, has been frequently used. So \( R \) can be updated to \((0.1, X)\). An n-dimensional Boolean cube is therefore defined as \( C = <c_1, c_2, \ldots, c_n> \) and \( c_i \in (0,1,X) \). A cube can be expanded to specific vectors in the dimensions with the data type X. In the extreme case that no \( X \) appears at any dimension of a cube, then this cube will also be a vector. An n-dimensional test vector at the gate level is defined as an input vector with all bits assigned values 0 or 1 for a circuit with \( n \) inputs. An n-dimensional test cube is a partially assigned test vector, if some input values remain \( X \). For high level modules, we can also define a control test subcube and bus test subcube for each module, which is simply the test cube for the circuit divided into two disjoint parts for control input lines and bus input lines.

The distance between two vectors or cubes means the number of dimensions in these two vectors or cubes having different values. The constraining weight of the test cube is number of bits having 0 or 1 in the test cube. Relative constraining weight of a test cube is the ratio of the weight to the dimension of the test cube. For test vectors, the relative constraining weight is 1. For test subcubes, the relative constraining weight can also be calculated as the ratio of the weight of the subcube to the number of bits in either bus or control. A test cover is a collection of cubes that covers a specified set of test vectors. A cover is a tautology if expanding all its cubes, the data space covered is all \( 2^n \) vectors.

Some terms need to be defined in the context of the discrete-relaxation algorithms in ARTEST at the high level. A fixed value at a high level is a fixed constraint implied by the injected test vector or the control words of the instruction being used. A value conflict happens when the current assigned values at some internal lines contradict some fixed values. A symbol is a variable attached to a control line or a data bus, and its value is bounded by the bus size. If the value assigned to a symbol is out of the predefined range, it will be considered as a value conflict also. A fixed symbol is defined as a symbol with a fixed value.

3. Functional Constraint Abstraction

For a high level module in data path, its control input lines usually come from the control unit so that the valid codes on these lines depend on the control words or micro orders of machine instructions. However, bus input lines would strongly depend on data path configurations which implicitly determine relations among buses. Based on this observation, the strategies used for global functional constraint abstraction should be different for control lines and bus lines.

3.1. Control Line Constraints

Following the assumptions we have been using in our previous work, we again assume that the high level circuit under test can be represented as an interconnection of a data path portion and a control machine. The instruction set of the control machine
and its corresponding control word sequences for each instruction should be available. For each module of interest, we can simply traverse the high level module diagram to ensure the connection of control input lines with the control unit. If they are connected, record the valid control codes from the control words of each instruction, and put those codes into the Boolean cover of that module. If any of the control input lines is not connected to the control unit, the corresponding bits in the code for that particular input line will be all Xs. In the preprocessing phase of the high level ATPG algorithm we are using, a symbolic simulation is applied to the module diagram to derive behavior information and a system of equations for each instruction. When each valid control code is found, it has to be approved by checking the observability of the output symbol of the module at that cycle of the instruction. If the symbol is not observable, then the derived control code will be judged as invalid.

After all valid control codes are derived, a minimal Boolean cover is formed on the control vectors, using any of the available minimization procedures. If the Boolean cover is found to be a tautology, then no control constraint exists at that particular module, i.e., all binary vectors on the control lines are valid control codes.

Invalid control codes are caused by global constraints of the circuit and sometimes termed "can't happen don't cares" in logic synthesis literature. It should be noted that it is very difficult to derive these "can't happen don't cares" in large circuits. This information is also very difficult to derive by simulating the complete gate level structure of the circuit. Our approach exploits the high level behavior information of the circuit so that the valid control codes can be easily derived.

In this paper, four high level circuits will be used for all experiments. The first one is a microprogram sequence controller with the same functional implementation as that of Am2910 [16]. The second one is a 16-bit general purpose microprocessor with nine instructions (MP2). The third one is an 8-bit 2's complement multiplier (MP1), and the last one is a digital signal filter (PCONT2). The control constraints have been derived for some modules in these circuits, and the information is shown in Table 1. CSP represents Cover Space Percentage, which means the percentage of space covered by all expanded cubes to $2^n$, where n is the number of control input lines. From Table 1, we know that Am2910 and PCONT2 have moderate control constraints, and MP1 has no control constraints. MP2 has intensive control constraints at the ALU, where only 4 control codes out of 64 are valid. This information will be used in the module test computation procedure which will be illustrated in section 3.3.

### 3.2. Bus Line Constraints

Bus constraints have significantly different characteristics than control constraints. Most of the bus constraints are contributed by the complex data path configuration. Take the multiplier MP1 as an example, the data path module diagram performs iterative additions of the partial sums for multiplication. There is an Adder embedded in the circuit performing the addition function. The values of the two input bus A and B of the Adder have a very complex relation during each cycle. There is only one instruction, Multiply, in MP1. Through symbolic simulation in the preprocessing phase, the bus constraints at the last cycle can be exactly expressed by about 50 module equations. These module equations look like those shown in Figure 2. They can also be expressed in a RTL format which clearly illustrates the implicit functions. These module equations in a RTL format are shown in Figure 3. The inter-dependent relation between bus A and B is very intensive and can be expressed by these 50 equations. The bus constraint in the example shown in Figure 1 is relatively much less intensive since only 1 equation is involved. These 50 module equations fully characterize the circuit behavior at the high level. Injecting the required values at A and B into the equations, a solution satisfying these equations justifies the values at bus A and B. So the problem of avoiding this bus constraint, when tests of the Adder is computed, is as complex as the complete high level ATPG problem [14]. Those vectors which do not satisfy these 50 equations will be classified as "can't happen don't cares". It is very difficult to derive all those "can't happen don't cares" for either control covers or bus constraints.

Instead of dumping the exact bus constraint information to the gate level test generator, we propose a bottom-up approach to

$$\begin{align*}
41 \text{ (MUX 2)(vs}_0, vs} & = (vs}_1 \\
42 \text{ (ADDER)(vs}_1, vs_2, 0) & = (vs_3) \\
43 \text{ (BusSplit_LSB)(vs}_{26} & = (vs_8, vs_{28}) \\
44 \text{ (BusJoin_MSB)(vs}_{26}, vs_3) & = (vs_{61}) \\
\end{align*}$$

Figure 2. An example of system of module equations expressing bus constraints.

$$\begin{align*}
v_3 & = v_8 \text{ if } v_{26} = 1, \text{ else } v_3 = 0; \\
v_{25} & = v_{27} + v_{37}; \\
v_{26} & = v_{27}[8:1]; \\
v_{20} & = v_{27}[0:0]; \\
v_{61}[7:7] & = v_{20}, v_{61}[6:0] & = v_{27}; \\
\end{align*}$$

Figure 3. Module equations in a RTL format.

<table>
<thead>
<tr>
<th>Control Constraints</th>
<th>modules</th>
<th># b bits</th>
<th># cubes</th>
<th>CSP(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Am2910</td>
<td>4-to-1 Mux</td>
<td>3</td>
<td>3</td>
<td>62.5</td>
</tr>
<tr>
<td>Incrementer</td>
<td>1</td>
<td>T</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>RegCn  CL1</td>
<td>3</td>
<td>2</td>
<td>73.0</td>
<td></td>
</tr>
<tr>
<td>MP2</td>
<td>16-bit ALU</td>
<td>6</td>
<td>4</td>
<td>63</td>
</tr>
<tr>
<td>MP1</td>
<td>8-bit Adder</td>
<td>1</td>
<td>T</td>
<td>100.0</td>
</tr>
<tr>
<td>8-bit XOR</td>
<td>1</td>
<td>T</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>2-to-1 Mux</td>
<td>1</td>
<td>T</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>PCONT2</td>
<td>8-bit Adder</td>
<td>2</td>
<td>1</td>
<td>25.0</td>
</tr>
</tbody>
</table>

† # e bits : number of control bits.
‡ T means the cover is Tautology.

Table 1. Control constraint information for modules.
avoid bus constraints. This approach will be addressed in the next section. However, sometimes some bus constraints are easily specified, e.g., fixed values on the data lines. Such constraints we do record and use in the gate level test generation.

3.3. Module Test Computation

Given the derived control constraints and constant value bus constraints, the gate level PODEM algorithm can be modified without making a major change. It is explained as follows:

1. Consider \( n \) input bits as an \( n \)-dimensional cube. When any assignment is made at an input bit, the cube is updated and intersected with the control cover of the corresponding module under test. If the intersection is null, then do backtracking.
2. If there exists any fixed value at an input bus, convert the fixed value to a vector and intersect with the cube when it is updated.
3. The \( X' \)s in the control test subcube need to be assigned with 0 or 1 according to the control cover such that it is easier to derive an instruction sequence at the high level. A decision stack is kept for these assignments. A new test vector can be derived by doing boundary backtracking on the decision stack. If constant value happens at a bus line, the \( X' \)s in the corresponding bus test subcube also need to be assigned with 0 or 1 according to the constant value.

The rules of bit-wise intersection of two cubes is shown in Table 2. The intersection of a test cube with the control cover is \( \emptyset \) if the intersection of each control cube in the cover with the test cube results in \( \emptyset \). If any bit in the resulting cube ends with \( \emptyset \), it means conflict. The techniques for both control constraint and constant value bus constraint abstractions have been implemented in the preprocessing phase of ARTEST. A modified PODEM algorithm has also been implemented for the module test computation.

4. High Level Test Cube Justification

4.1. Important Observations

Some important observations have been obtained from the discussions in previous sections. Based on which the test cube justification technique is developed.

1. Observing those test cubes generated by PODEM, we found that most of the control test subcubes have very high relative constraining weights, and contrarily most of the bus test subcubes have very low relative constraining weights. In other words, given a single stuck-at fault in the module, most of the time very few bits corresponding to input buses will be assigned to a 0 or 1 by a PODEM like test generator.

2. Most of the modern high level test generators need to justify complete test vectors since integer representation for bus signals are used to enlarge the module partitions and speedup the test generation process at a high level. Therefore, test cubes generated by PODEM are fully specified to obtain integer values. However, this cube expanding procedure overspecifies many \( X' \)s which would very likely cause conflicts with various implicit constraints at the high level. If a test cube can be justified through the high level algorithm without being expanded, most bus constraints could be avoided.

When the module tests are computed, only those \( X' \)s corresponding to the control lines are expanded according to the control cover. The bus test subcubes derived by PODEM will not be expanded and injected directly to the high level. The modified high level ATPG algorithm which can justify the test cubes at a high level is presented in the next section.

4.2. Modification of the Discrete-Relaxation Algorithm at High Level

In the relaxation-based ARTEST, the high level test generation problem has been transferred to the problem of solving a system of equations. The injected test vector for the module under test will be translated to values corresponding to each input bus and control line. These fixed values will be assigned to symbols in the system of equations, and the discrete-relaxation algorithm will be applied to solve this system of equations. Those corresponding symbols with fixed values will be marked as fixed symbols. During the equation solving process, if there is no conflict at any of the fixed symbols in the system of equations, the solution is derived.

When a test cube is injected at the high level, it is partitioned into test subcubes corresponding to each input line. If any test subcube has relative constraining weight 0, i.e., all bits are \( X \), this subcube will not be assigned to any symbol. If any test subcube is also a test subvector, i.e., all bits fully specified, the corresponding value will be calculated and assigned to a symbol as the previous approach. Except for the above two conditions, the test subcube in which some bits are specified and some are \( X \), will be assigned to the corresponding symbol and a special data structure will be maintained to store all those partially assigned symbols. Those partially assigned symbols will not be marked as fixed symbols, since multiple values can be applied to them. In each iteration of the discrete-relaxation algorithm, when the value of a symbol is updated, the symbol has to be checked whether it is a partially assigned symbol. If it is, the updated value needs to be translated to a subvector and then intersected with the subcube. If there is a conflict, a new updated value will be calculated based on some rules. The iteration can therefore continue until convergence.

When a new updated value for a partially assigned symbol conflicts with the corresponding test subcube, the algorithm for calculating a new updated value can be illustrated by Figure 4. The conflict value is 256, in which the bits conflicting with the assigned ones in the test subcube need to be updated based on the values at the same dimensions of the test subcube. The other bits
corresponding to Xs' are not updated. The new value is therefore 174. The concept is simple. We only update those dimensions having a conflict with the test subcube, and keep all other dimensions unchanged. From our experience, the exact new updated value does not seem to have a significant effect on the convergence of the algorithm. This heuristic has been proved to be effective in our experiments.

Test subcube: 0 0 0 1 0 1 1 0 1
Conflict value: 0 0 1 1 1 0 1 1 0 0 = 236
New updated value: 0 0 1 0 1 0 1 1 1 0 = 174

Figure 4. An example of conflict handling for test cubes.

5. Results and Comparisons

ARTEST has been continuously updated with the testcube justification technique to overcome intensive bus constraints. A highly efficient fault simulator, PROOFS [17], has been linked with this package to collapse module stuck-at faults. The four circuits introduced in the previous section are used and the test results for these techniques are shown in Table 3. All experiments were run on a SUN 4 machine. The first approach does not apply any technique to avoid the global functional constraints. The second approach exploits control cover, constant value bus constraints, and the test cube justification technique to overcome both control and bus constraints. It has been pointed out that MP1 has intensive bus constraints. PCONT2 is a digital signal filter and it also contains intensive bus constraints. MP2 has intensive control constraints at the ALU. Am2910 is a moderate circuit for either of the two types of functional constraints. The results show that these proposed techniques are very effective in avoiding global functional constraints. For the control constraint at the ALU in MP2, the speedup achieved by the valid control code abstraction technique is approaching 3.34 with much higher fault coverage. Considering the bus constraints at the Adder of MP1, the speedup achieved by the test cube justification technique is almost 66 times. The speedup for the Adder in PCONT2 is also extremely high. For the easy circuit, Am2910, the Approach 2 also derives speedup of 4.9 for RegCn_CL2.

To compare the complexity of these approaches, the statistics of total number of boundary backtracks and retrials of PODemo are recorded and shown in Table 4. When an injected test vector or cube fails at the high level, the gate level test generator will be retried until the implicit limit is exceeded. For this experiment, the maximum limits for boundary backtrack and retrial of the gate level ATPG are 50 and 20. Considering the total number of boundary backtracks and retrials of the gate level ATPG as an index of algorithm complexity, these data demonstrate the efficiency of Approach 2. For the Adder in MP1, the total number of boundary backtracks consumed by Approach 2 is 254 times less than that of Approach 1. In computing tests for RegCn_CL2 in Am2910 and Adders in PCONT2, Approach 2 also shows significant superiority over other approaches. If the ALU of MP2 or Adder of MP1 were embedded in other larger VLSI circuits, those amount of boundary backtracks in Approach 1 would not be affordable. Based on these experimental results, we can conclude that the test cube justification technique using discrete relaxation method is very promising in avoiding bus constraints. Combining the control cover abstraction technique for control constraints together with the test cube justification technique for bus constraints gives a complete solution to hierarchical test generation under intensive global functional constraints. This approach not only supports a higher level of module partitions for speedup, but is also unaffected by the drawbacks of overspecifying unassigned values at the module boundary for satisfying the requirements of high level ATPG algorithms.

<table>
<thead>
<tr>
<th>circuit</th>
<th>primitive</th>
<th>module(s)</th>
<th>faults</th>
<th>Approach 1</th>
<th>Approach 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td># drop</td>
<td># red</td>
</tr>
<tr>
<td>Am2910</td>
<td>MUX</td>
<td>1</td>
<td>294</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Increment</td>
<td>1</td>
<td>184</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RegCn_CL2</td>
<td>1</td>
<td>367</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>MP2</td>
<td>16-bit ALU</td>
<td>1</td>
<td>950</td>
<td>104</td>
<td>16</td>
</tr>
<tr>
<td>MP1</td>
<td>8-bit XOR</td>
<td>1</td>
<td>84</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2-to-1 Mux</td>
<td>6</td>
<td>408</td>
<td>83</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Adder</td>
<td>1</td>
<td>332</td>
<td>21</td>
<td>32</td>
</tr>
<tr>
<td>PCONT2</td>
<td>8-bit Adder</td>
<td>8</td>
<td>2240</td>
<td>345</td>
<td>254</td>
</tr>
</tbody>
</table>

Approach 1: without any functional constraint information.
Approach 2: with control cover, constant value bus constraint abstraction and test cube justification techniques.
# drop = number of faults dropped.
# red = number of faults identified as redundant.
eff. (%) = test generation efficiency = (#faults - #drop)/#faults.
RegCn_CL2: combinational part of the Register Counter.
6. Conclusions

In this paper, we have formally addressed the system level functional constraint problem for hierarchical test generation. We have also proposed several approaches to solve both control constraints and bus constraints. For control constraints, circuit behavior information is exploited to derive valid control Boolean covers for different modules. For bus constraints, we propose a constant value bus constraint abstraction technique and a test cube justification technique. These proposed algorithms have been implemented in our hierarchical test generation package, ARTEST, and four high level circuits with different constraint characteristics have been tried in the experiments. The experimental results show the effectiveness of combining the control cover abstraction technique and the test cube justification technique as a complete solution to the global functional constraint problem.

References


