MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

KHARKOV NATIONAL UNIVERSITY OF RADIOELECTRONICS

ISBN 966-659-113-8

Proceedings of IEEE East-West Design & Test Workshop (EWDTW'06)

Sochi, Russia, September 15 – 19, 2006

CONTENTS

Path Sensitization at Functional Verification of HDL-Models

Alexandr Shkil, Yevgeniya Syrevitch, Andrey Karasyov, Denis Cheglikov
Design Automation Department, Kharkov National University of Radio Electronics
14, Lenin Av., Kharkov, 61166, Ukraine
E-mail: Syr_Jane@rambler.ru, andreyk@ aldec.com.pl

Abstract

Strategy of verification of digital devices models, represented in hardware description languages, is considered. The basic idea is in using path sensitization method in a graph model, generating distinguishing tests for separate functional elements, superposing these tests and interactive calculating etalon reactions.

Key-words: *Functional verification, graph model, path sensitization, distinguishing sequences.*

1. Introduction

The necessity of researches in the field of verification is caused by the lack of effective methods and tools of functional verification of digital devices (DD) models on a step of describing them on behavioral level.

World companies – vendors of digital circuits, are forced to decrease their time-to-market. According to vendors' evaluations, verification (functional as well) takes up to 80% of labor expenditures in the design cycle [1]. There is a big demand for tools of functional verification of devices models on a step of their description in hardware description language (HDL) on behavioral level.

If model description in hardware description languages (HDL) is considered as a software program, then, from one point of view it is necessary to execute software verification, but the other point of view it is not always optimal. Software verification considers all modes with all data testing. While checking correspondence between code, which describes a device, and its specification on all possible data for all reachable inputs, to hold on verification during appropriate time with 100% completeness it is not possible.

Assume that all in-build operators are combinational elements. It means that to check them it is necessary to drive 2^n values, where n – total dimension of inputs. To get high quality of verification for appropriate time it is necessary to decrease number of driven tests.

Despite variety of publications, connected with verification and diagnostics of digital devices, today instrumental tools of automatic test generation for complex DD functional verification are actual and claimed.

Lack of automatic test generators is felt in many well-known companies: Aldec, Altera, Actel, Xilinx, Synopsis.

Thus, this work aim is to develop HDL-models verification strategy, which allows decreasing time on design cycle by decreasing number of test vectors.

2. Problem Definition

Digital device model, written in synthesable subset of HDL is given. Syntax mistakes are checked and corrected on a compilation step. Specification on this device, on a base of which the model was build, is also given. On the assumption of the set aim, to reach it, it is necessary to solve the following problems:

1) to develop internal model of a DD, represented in hardware description language (VHDL) in an effort to carry verification, and also procedures of direct implication and backtracing on this model;

2) to develop methods of creating distinguishing sequences for different types of arithmetic and logic functions;

3) to develop methods and strategies of functional verification of digital devices on a step of design entry and simulation.

3. Description of verification strategy

Today the most widespread approach to verification consists of the following [2]. There are models M1 and M2. Here, M1-reference model, and M2-verified. Tests for model M1 are generated, and as a result of simulating M1 on the received test sets, reference values (similarly to process of test diagnosing) are defined. Then the same tests move on M2, them also it is simulated and experimental responses are received. Reactions of models M1 and M2 are compared. Model M1 is equivalent M2 if reactions coincide, and contains a mistake otherwise. If M1 is set by informal or in part by formal way, the earlier described approach to verification becomes ineffective.

In modern technologies of designing instead of a real physical device its representation in HDL is used. Such representation is usually based on the data received from the specification. Thus, the specification forms M1, the description in HDL -M2

Process of tests generation is based on the following aspects: an internal model, which allows carrying out procedures of direct implication and backtracing; design error models, tests and algorithm of their generation.

In the following chapters the verification strategy, based on graph model of the verified device, functional design models and distinguishing sequences, is considered.

To build internal model the origin description is transformed into graph model, which is decomposition if two graphs. First - information - describes dataflow and their conversion (similarly to an operational automaton in classical composite model with microprogram handle) without the registration of conditional branches. The second graph is developed as a network of conditions [3].

The dataflow I-graph contains vertexes of 2 types: operands and functions. Types of operands: integer, unsigned bit and std_logic vectors Types of functions are restricted to the synthesizable subset of VHDL. Arcs connect vertexes in the following manner: a source vertex is connected to a functional vertex, then the arc goes out of the functional vertex and comes into a destination vertex. The arcs, which come into destination vertexes, can be conditional or non-conditional. Conditional arcs correspond to the operators, which are inside conditional expressions of VHDL. Conditional arcs contain labels, which code conditions of arc transition operation.

In its turn, the second graph (a control one) contains conditional constructions (like *case*, *if...then...*, *with ... select*) from the origin device description. Each predicate in the condition is modeled as a subgraph, which has a specific label. The result of C-graph simulation is a set of labels (a label), according to which it is necessary to fulfill transition along arcs in I-graph.

To form the verification strategy methods of structural testing is used, based on path sensitization in a device model, which after adaptation can be used for functional verification.

To form a system of proving, which allows to evaluate the quality of the offered strategy, consider most important lemmas, statements, and theorems.

Lemma 1. If conditions of functional element (FE) activation in error-free HDL-code exist, then they can be obtained using P procedure (direct implication and backtracing) in graph model.

Proof. Considering that there are only in-build operators in HDL-code, then conditions of distinguishing sequences DS (distinguishing sequences are those, for which on different functions from a given subset same input pattern or patterns will give different results) are defined by simple listing of in-build operators' types and procedures of distinguishing sequences creation for them. Propagation (justification) conditions for activated element are built with the help of direct implication and backtracing. Assume that the HDL-code is error-free. In this case tolerance ranges (TR) on element's inputs and feasible regions on its outputs are those, that if distinguishing sequences belong to TR or an activated functional element, they can be justified on external inputs of the graph model. It is caused by an assumption that each functional element in error-free HDL-code executes its function in TR limits. On the assumption of Roth's known theorem about possibility of activated path construction for a net of logic elements. Condition of such path construction is in saving all temporal decisions of backtracing. On the assumption of backtracing procedure, all temporal decisions are saved at backtracing for all types of build-in operations, which were developed in [2, 3], and, consequently, necessary solution can be found between them. Such assertion may not be applied for HDL-models with design errors.

Corollary 1. Functional element may be activated, if distinguishing sequences exist and justification and propagation conditions are kept.

Corollary 2. If it is impossible to identify a functional element, then there is an error in the code (admitted place of the error - subset of its successors).

Consider an example. In fig.1 circles correspond to operand vertexes, rectangles – functional elements. Assume that there is a mistake in the HDL-code: addition operator with inputs 1 and 2 was replaced with multiplication operator. At this on input 1 '0' value is driven. While identifying functional element, which coincides to division operation on the 6-th line, there should be value '0'. But at justification of this condition there is a contradiction, because there is constant '0' on the 5-th line. Circles are operand vertexes, and rectangles are functional elements.

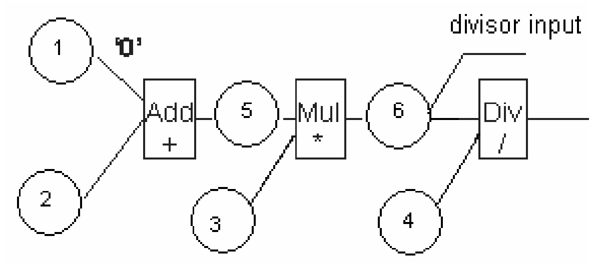

Figure 1. A fragment of corrected I-graph

Thus, information about design errors can be obtained before diagnostic experiment carrying out in the course of test building.

Lemma 2. All functional elements activation in I-graph is a check of data operation mechanism in HDL-model.

Proof. Summation of FEs on an activated path defines an algorithm of operation of data, obtained from external inputs. It is obvious, because data go from external inputs to external outputs through functional elements – operators. From the other point of view it is possible to say, that summation of algorithms of data operation corresponds to a list of device modes, thus its specification. Similar statement was proved in [4] regarding to data in microprogram-controlled devices.

Statement 1. If propagation conditions through FE f_i coincide (belong) with DS for FE f_j (which precede f_i on the activated path), then the type of FE f_j also is considered being identified. These propagation conditions are called C_p.

Proof. It is obvious, that if DS for (i+1)th element include results of direct implication of DS on ith element, then these results can be used to activate (i+1)th element.

Lemma 3. All FEs on the activated path are identifiable.

Proof. If at path sensitization conditions, which include C_p, were used, then all FEs on the activated path are said to be identified. For FES with 2 inputs, inputs of which are not set, activation condition can always be generated. If there is a solution, it can be obtained by listing all possible values.

If at activated path construction justification conditions are conflicting, then it is necessary to insert extra control point into HDL-model.

Theorem 1. To identify all FEs in informational I-graph, it is necessary and enough to activate all paths in graph, which cover it from 1st rank FEs to external outputs or control points.

Proof. Based on lemmas 1 - 3 and corollary 1 it is obvious that all FEs, which lie on activated path from 1st rank functional element, are activated too, and, consequently, they are identifiable. Similar results of non-redundant digital circuits were obtained in [5]. Adding extra control points solves a question of HDL-code redundancy, and consequently, I-graph redundancy.

Corollary 3. Identification of all FE in HDL-model checks data processing mechanisms of the model.

On a base of lemma 2 activation of all FE is a check of data operation mechanism. Summation of checked data processing mechanisms corresponds to a checking of HDL-model correspondence to specification.

4. Distinguishing sequences creation

At tests generation for some primitive element, it is necessary to take into account the following. As it has already been told, HDL operators are chosen as functional elements (primitives). From the point of view of hardware verification, operators HDL so as functional elements, do not contain mistakes inside and work correctly. If it were not so, HDL model verification would include checking simulation system. And different experts, a programmer and a designer, should carry out these two problems. Taking into account, that operators of language (primitives) do not contain mistakes inside themselves, it is obvious, that driving tests checking the law of primitive functioning is inexpedient. Therefore the sense of functional elements testing will consist not in check of functioning, but in check of its type. Thus, it is necessary to drive such test sets onto a primitive that after the analysis of reactions to them it is possible to identify type (function) of the primitive and to distinguish it from the others functional elements.

Such distinguishing sequences (DS) allow finding mistakes connected with replacement of operators in the HDL-code. At that, in particular it finds mistakes of replacements of logic operators only by logic ones, arithmetic only by arithmetic, because cross replacement can be revealed, as a rule, during compilation.

Consider creation of distinguishing sequences for logic operators. Assume, that number of inputs is n. Number of bits of each input is m. All operators in VHDL (except negation, sign, absolute value, and putting to some power) have $n \geq 2$. Consider the following set of logic operations – {**and, or, nand, nor, xor, xnor**}. Depending on operands dimension there can be from 1 to 3 vectors in a distinguishing sequence.

Logic operations on each bit are done independently. Three possible conditions, which influence number of vectors in distinguishing sequences, can be defined:

1) n>=2, m=1;

2) n>=2, m=2;

3) n>=2, m>2.

In case 1 maximum number of vectors are 3, in case 2 maximum is 2, in case 3 maximum is 1.

Distinguishing sequence for logic elements is build after analysis of fault-free checking tests for functional elements. Distinguishing sequence must fulfill the following requirements: it must distinguish types of logic elements from each other, and number of vectors must be minimal.

In fig. 2 there is an approximate schematics of forming distinguishing sequences for one-bit multi inputs logic elements. The following chain of actions is shown: all one-bit inputs of unknown logic element are driven with «0». Depending on the reaction two subsets are formed. Then the element is driven a sequence, where on of inputs are '1', all the rest are '0'. Two more subsets are formed. And third pattern – two values of '1' in any two bits, and again two subsets are formed. At last, after driving sequences and results analysis, 6 subsets are formed, which contain one element.

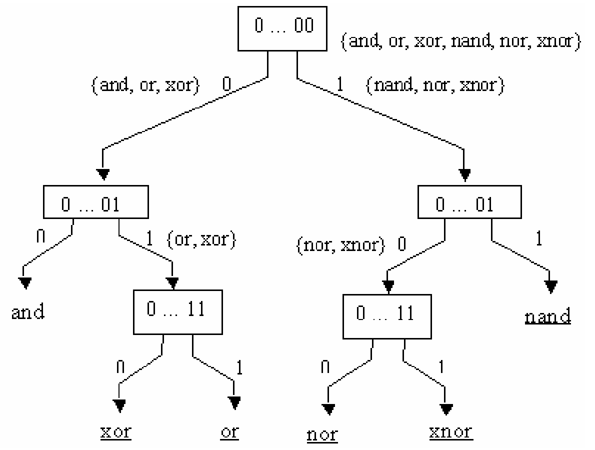

Figure 2. Distinguishing sequences for logic operators (m=1, n>2)

Multibit logic functional elements operate with each bit separately. So multibit functional element can be represented as a set of one-bit ones.

Consider algorithm of distinguishing sequence creation for arithmetic operations of "all-out-of-all" from a given subset {add (+), sub (-), mul (*), div (/)}. Arithmetic operations can be distinguished with a condition, that operands' dimension is enough to represent numbers, bigger then '1'.

So that to distinguish operation "plus" from a subset {addition, subtraction, multiplication, divide), it is necessary to submit a zero on one of inputs of the functional element, and on the other input – value, greater than '1'. Then break the given set into 2 subsets (addition, subtraction) and (multiplication, division) by the following way: if the result of the previous step equals to '0', then tested functional element is in the subset (multiplication, division); if not equal to '0', then it is in the subset (addition, subtraction). Then drive equal values greater than '1' onto both inputs. If the result is not '0', the functional element is ADDITION.

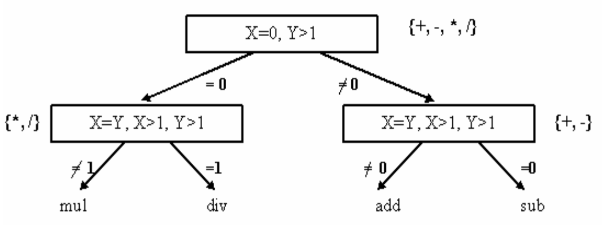

Figure 3. Distinguishing sequences for arithmetic operators

If distinguishing sequences for i+1 element includes results of direct implication of i^{th} element distinguishing sequences, then these results can be used to activate i+1 element, in other words propagation conditions through the element can contain distinguishing sequences.

It is necessary to point that mentioned reasoning shows only one of possible ways of constructing DS.

Sequence and time of driving vectors from DS for FE doesn't influence the result of distinguishing. It means that vectors from DS can be driven in any order and arbitrary number of other vectors can be inserted between them. But FE's reactions on vectors, which belong to DS, should be kept in mind.

According to lemma 3, to identify all FEs in I-graph, it is necessary and enough to activate all paths in the graph, starting from 1^{st} rank FEs to external outputs or control points in a way so, that every FE from a set of 1^{st} rank FEs belongs to one path. Path sensitization in its turn implies justifying propagation conditions. Propagation conditions include input sequence, justification condition, and algorithm of output modification. At path sensitization algorithm forming consider the following strategy.

1. Choose i^{th} operator (non-activated functional element FE of any rank, starting with the 1^{st}). A distinguishing sequence (DS) is driven n it.
2. Calculate reactions of the i^{th} activated operator on the given DS, thus this FE becomes activated.
3. If reactions of the i^{th} activated operator completely coincide with DS for j^{th} operator (its successor), they are used as DS for j^{th} operator. If reactions do not coincide: a search of i^{th} element reactions, which belong to DS of the j^{th} element (successor), is carried out, if there are any. Missing vectors from DS for the successor is provided with the help of backtracing.
4. Steps 1-3 are repeated until a set point is reached (control point CT or external output).

If on ith step activation of next element is impossible, so it is necessary to use extra control point on (i-1)th step of FE. CT is not a single-valued criterion of activation end. If activation is possible, so a value in CT is registered, but activation is continued.

5. General verification strategy

On the assumption of formed and proved lemmas, statements, and theorems, general verification strategy, based on functional elements sensitization, starting from 1st-rank element. It consists of the following steps.

1. Activation of the ith FE of 1st rank is carried out. Distinguishing sequences are driven directly from external inputs of a graph model. External inputs (outputs) of I-graph are operand vertexes, which are ports in HDL-model.
2. Sensitized path is build from activated FE to either external outputs in a graph (output ports in HDL-models) or CT.
3. Activation is finished, if set operand vertex is reached or path sensitization is impossible.
4. Steps 1-4 repeat for all FEs of the 1st rank.
5. After finishing the 1st rank FEs activation next k^{ary} FE of the p^{ary}-rank ($p>1, k=\overline{1,n}$), which do not belong to any already sensitized paths, is activated.
6. Activation is carrying out until all FEs are activated.

In order to start test generation, it is necessary:

- transform synthesizable behavioral description of DD model in VHDL into internal graph model;
- set a list of functional elements – operators, which have to be checked, and define their ranks on a base of graph model;
- for each ith FE find a set of DS, which distinguish the given functional element from the other ones from a given subset (which identify th given operator);
- activate ith FE by driving DS on it; calculate FE reactions on DS with the help of direct implication and backtracing for the corresponding element;
- execute justification of DS by backtracing from activated FE to external inputs of I-graph, combining with simulation on C – graph, which provide transition of data along arcs with labels (simulation is done by using backtracing procedures, defined for each type of FEs);
- obtain justification results – test vector on external inputs of graph model;
- simulate obtained vector to control points or external outputs of graph model;
- if there is exact values in internal points (on operand vertexes), which are successors of the ith FE, then it is necessary to simulate till these points;
- compare results, obtained after simulation, and etalon reactions from specification at same input stimuli.

In described above procedure number of obtained DS equal to number of operators in HDL-code. This number can be decreased due to usage of parallel tested operators.

Consider a problem of test vectors forming on external inputs. A set of test vectors on external inputs of graph model corresponds to one DS. At that, during passing along the model from external inputs to a set point (from 1st rank FE to CT or external output) test range on inputs enlarges, because each DS produces a quantity of justifying vectors, which is shown in fig.4. Reactions on DS are called DS_i'. In fig.4 circles are operand vertexes, and rectangles are functional elements.

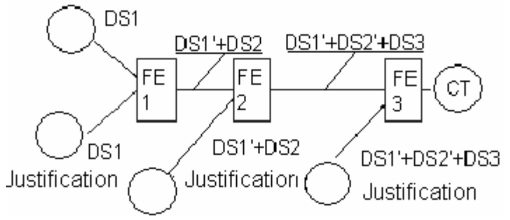

Figure 4. Test vectors production

Production of test vectors at justification can be evaluated by limit (best and worst) rating.

Best case is absence of common vectors in DS reactions of ith FE and DS of j-th (its successor), namely, $DS_i' \cap DS_j = \varnothing$.

Best case is inclusion of DS_i' into DS_j.

6. Etalons obtaining

Etalon obtaining at functional verification consists in the fact, that specification is usually informal (there is no unique correspondence between input stimuli and output reactions) and incomplete (not all modes of a code are described). In particular these reasons lead to the situation, when it is impossible to obtain reactions in an explicit form. Etalon calculation problem is one of basic during test verification. Three basic ways of obtaining reactions can be defined:

1. From specification. In specification etalons are set in explicit or implicit forms. Specification on a digital device is set in an informal, usually verbal, form. On a base of his experience, design engineer can mark out etalon reactions or calculate them, if they are given in a form of operation algorithm, timing diagrams, etc.

2. Using external simulation system. If specification contains operation algorithm, then either software simulation system or emulator can be created. These systems allow automatic calculation of etalon

reactions by driving obtained tests onto them. At the same time it is necessary to make an assumption, that such external system is ideal.

3. Using interactive iterative process. This approach is based on a fact, that the designer for any input data can calculate reactions of a developed device by hand or somehow, if input test data corresponds to states (device modes), described in the specification.

The sequence of verification engineer's activities at interactive process of etalon obtaining is set by the following procedures:

1) to receive from a design engineer compiled HDL-code that describes a device or its logically (functionally) finished blocks;

2) to build tests on the received HDL-code by generation of sequences distinguishing a given operator and path sensitization, to set control points;

3) to return received tests to a design engineer for calculation of corresponding output values and/or values in set control points;

4) to simulate generated tests on the HDL-code in order to get experimental values;

5) to compare values received after simulating the HDL-code, and etalon values received from the design engineer;

6) if they are not equal, a conclusion of a design error, present in the HDL-code of a DD model is made.

7. Conclusion

In the given work strategy of verification of digital devices models, represented in hardware description languages, is offered with usage of special distinguishing tests. General strategy of verification of digital devices models and algorithm of distinguishing sequences generation are considered. Completeness of obtained tests and their distinguishing properties are proved. Particularly, ways of getting output reactions from informal specification are discussed. Table 1 contains results of diagnostic experiment above 3 types of devices: control device b06 from ITC benchmark library, sectional microprocessor KP1804BC1, and sequential device s27 from ISCAS'95 benchmark library.

Table 1. Comparative analysis

Name	NCL	NEO	NT	NPT	FC	FCG
B06	127	70	16	16	25	100
KP1804BC1	46	17	8	32768	70	99
s27	50	19	14	32	89	92

In the given table: NCL-number of code lines, NEO-number of executed operators, NT-number of generated tests, NPT- number of NP-complete tests, FC-fault coverage of NP-complete testing (in percentage), FCG-fault coverage of the given method (in percentage).

Science novelty of the offered strategy consists in applying path sensitization method at test generation for HDL-models verification. The greatest effect this strategy gives for functionally finished DD, whose specification is set in an implicit form in the form of mathematical dependences. In this case listing of all possible data on all modes is replaced by path sensitization and FE identification in graph model of DD and analysis of tolerance range of functional elements

Practical value of the work is set by software realization of tests creation method, based on VHDL-model of a device, being verified, and their simulation in the course of diagnostic experiment.

8. References

[1] *Bergeron J.* Writing Testbenches: Functional Verification Of Hdl Models, 2003, Kluwer Academic Publishers, 354 p.

[2] *Syrevitch Yev. Yef.* HDL-models verification. TCSET '2006, February 28 - March 4. 2006, Slavske, Ukraine, p. 570-573.

[3] *Rustinov V.A., Syrevitch Y.Y., Syrevitch A.V., Cheglikov D.I.* Protsedury implikatsii na arifmeticheskih operatsiyah pri sinteze testov verifikatsii // Radioelektronika I informatika. Vyp. 130. 2005. S. 4-13 (in Russian).

[4] *Sharshunov S.G.* Razrabotka funktsionalnih testov RISC-mikroprotsessorov // Avtomatika I telemehanika. 2004. N 11. S. 174-189 (in Russian).

[5] *Shkil A.S., Kizub V.A., krivoulya G.F.* Generatsiya testov v sisteme avtomatizirovannogo proektirovaniya diagnosticheskogo obespecheniya // Uprav. sistemy i mashiny. 1987. №4. C.44-48 (in Russian).

AUTHORS INDEX