Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

7th International Conference
on Electrical Engineering
ICEENG 2010

Assertion based HDL-models testing for SoC components

By

Vladimir Hahanov * Eugenia Litvinova * Wajeb Gharibi **
Olesya Guz *** Ngene Christopher Umerah * Tiecoura Yves *

Abstract:

The testing and verification technology for system HDL models, focused to the significant improvement of the quality of design components for digital systems on chips and reduction the development time (time-to-market) by using the simulation environment, testable analysis of the logical structure HDL-program and the optimal placement of assertion engine is proposed.

Keywords:

Testing, verification, HDL-model, assertion engine.

* Kharkov National University of Radioelectronics, Kharkov, Ukraine
** Jazan University, Kingdom of Saudi Arabia
*** Donetsk Academy of Road Transport, Donetsk, Ukraine

1. Introduction:

The novel testing and verification technology for system HDL models allows searching for errors in the HDL-code with a given thoroughness for an acceptable time by means of the introduction assertion redundancy to the critical points of the software model, which are defined by the synthesized logic functions of the testability. The controllability and observability criteria, used in hardware design and test, are applied to estimate the quality of software code in order to improve it and effective diagnose semantic errors.
The objective is improvement of the testing and verification technology for digital systems to diagnose and correct of errors for HDL-models by sharing of the assertion engine and testable design technologies.
The research tasks: 1. Assertions based verification and testing environment for system HDL-model. 2. Development of testability evaluation metrics on the basis of new logic testability function. 3. Application of a technological assertion model to verify an IP-core filter on the basis of discrete cosine transform. 4. Practical results and directions for further research.
The research sources: 1. Technologies and tools of test and testbench creation are represented in the papers [1-3]. 2. Models and methods for verification of the system models on the basis of assertions are described in [4-7]. Testable software design uses the IEEE standards [8-10], as well as innovative solutions to verify and testability analysis for the system HDL-models [11-18].

2. Infrastructure of Design Verification

The verification model can be represented as a generalized equation for fault detection $T \oplus S = L$ at the system level or more detail in the components:

$$(T, F) \oplus (S, A) = L_S.$$

Here T, F are test patterns and functional coverage of the reference model with expected responses; S, A are testable HDL-model and assertion engine for the verification and exact diagnosis of errors in software code. Hardware testing is based on utilize the analytic expression $(T) \oplus (S, B) = L_h$, where B is the boundary scan register defined by the IEEE 1500 standard and used as a complement of the model to obtain the desired diagnosis thoroughness. At that L_s, L_h are lists of errors and faults, obtained on the stages of design verification and product testing.
Verification and testing strategies have different models of technology application, focused to reduction of time-to-market. The iterative verification process seeks to

correct the errors of system level HDL-model, obtained from the project specification (Fig. 1). The end result is a netlist or debugged HDL-model of register level. The subsequent iterative process is synthesis and implementation of a design in silicon chip. Here the testing checks the correction of hardware implementation for HDL-model to the register or gate level in an FPGA. For ASICs such technology would not be practical because the reprogramming of errors would cost up to one million dollars.

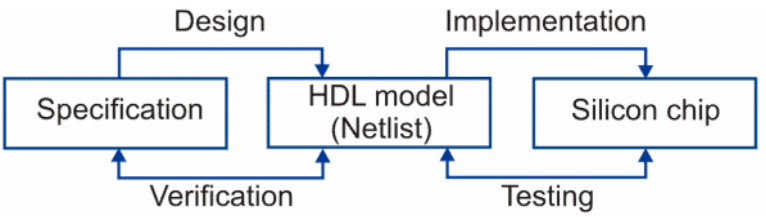

***Figure (1):** Strategy of project designing*

Taking into account the definitions and explanations model of the environment or macroprocess of verification for software stage of a project is focused to reduce the time of product creation and increase the yield for the use of code redundancy in the form of assertion engine and use testbench in conjunction with the metrics for determining the quality of the test or the functional completeness.

Testing and verification infrastructure for HDL-model is shown in Fig. 2, where the specification of a design described in the formal high-level programming language, is the initial information for the follows: creation of the metrics for the estimation of test quality as functional coverage, HDL-model of a design, testbench, assertion structure that is complement to the basic model (it is necessary to accelerate testing and debugging of a design).

The verification environment is presented by simulation system (for instance, Questa, Mentor Graphics), test system (Testbench), assertion engine and system code of the model by using languages VHDL, Verilog, System Verilog. Testbench sets input stimuli and reference responses for them, defined in the HDL-languages, focused to functionality check (variables, functions, sequences), parameters of which are defined in a functional bin.

Assertion engine is model redundancy, complementary testbench to verify the time internal design status, represented by input-output assertions and designed to accelerate the testing, verification, diagnosis and correction of design errors in the system code. Assertions can be generated not only by specification, but also by HDL-model, removing the unnecessary construction, and the rest need to be modified to the assertion form. At the same time there is the probability of software error recurrence in an assertion, which will not be identified in the simulation.

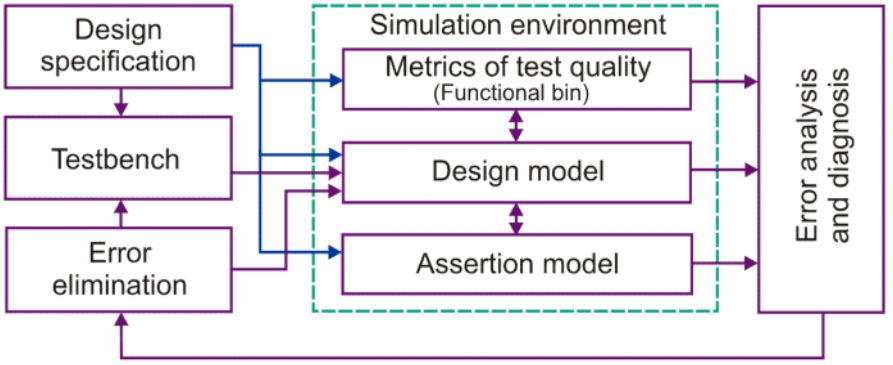

Figure (2): *Design verification environment*

A simplified interaction structure for the assertion engine and other components of the verification and diagnosis environment is presented in Fig. 3. Testbench is a reference model of design object in the form of input and output responses. Often, already proven and accessible model of another company, which is checked on the Model Under Verification (MUV), is used instead of testbench. In this case, a generator of input sequences or testbench without output responses is required. Assertion Engine is a superstructure for the modules Testbench and MUV and it is designed to compare the simulation results in order to create a binary output response vector (assertion state vector):

$$A = (A_1, A_2, ..., A_i, ..., A_n),\ A_i = T_i \oplus S_i,\ A_i = \{0,1,X\}.$$

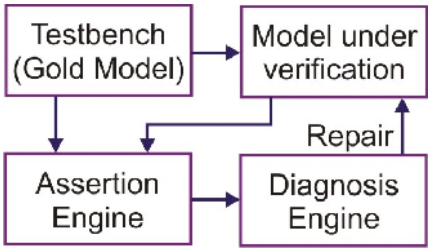

Figure (3): *Assertion utilize technology for design verification*

Sequence of operations for creation the verification environment is interesting. Here sole argument is a design specification; all other is derivative from it. The interconnection structure for the design and diagnosis with subsequent correction of errors in the HDL-code is shown in Fig. 4.

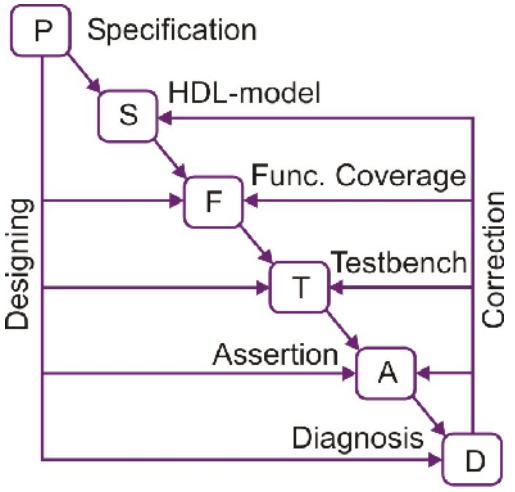

Figure (4): *Design verification flow*

3. Analytical Model of Verification Infrastructure

For identification of a generalized state of HDL-model the reference signatures are formed for critical points (variables, registers, memory) in time and space. Then the analysis of HDL-model for its diagnosis is performed by comparing the reference and experimental signatures to form the assertion vector describing the comparison state (reference and real) for the object components in time and space. It is advisable to form the assertion vector regardless of design HDL-model. Assertion and functional models are processed concurrently and independently of one another by simulation environment.

Assertion model determines the deviations from the behavior of the object in significant points of the space-time reference structure. The assertion format must conform to the format of the HDL-model. Assertions are focused to the diagnosis of semantic errors for HDL-model by means of Testbench and MUV.

The analytical model of the verification infrastructure is represented by the following expressions (P– design specification, S– soft-model, A– assertion model, T– Testbench, F– functional bin, D – module for error diagnosis and C – conditions for error diagnosis):

$$
\begin{aligned}
&M = \{P, S, A, T, F, d, C\},\\
&1)\ S = f_1(P) = \left|S_{ij}\right|;\\
&2)\ F = f_2(P, S) = \{F_1, F_2, ..., F_i, ..., F_n\};\\
&3)\ T = f_3(P, S, F) = \{T_1, T_2, ..., T_i, ..., T_n\};\\
&4)\ A = f_4(P, S, F, T) = \left|A_{ij}\right|;\\
&5)\ D = f_5(P, S, F, T, A) = \left|L_{ij}\right| \in \{L_s, L_h\};\\
&6)\ C = [\underset{i=1}{\overset{n}{Y}} F_i \in F = P] \wedge [\underset{i=1}{\overset{n}{Y}} T_i \in T = F];\\
&7)\ L_s = (T, F) \oplus (S, A);\\
&8)\ L_h = (T) \oplus (S, B).
\end{aligned}
\tag{1}
$$

Here, the expression 6 defines the conditions of test confidence for functional bin concerning the specification. Line 7 defines the function of determination the design errors in the transition from the system level to register one, using all attributes of verification infrastructure. The function 8 regulates the detection of faults in operation stage of digital system-on-a-chip.

The assertion redundancy is a function of critical points of HDL-model, the maximum number of which may be equal to the number of time frames for functional components identified by the specification.

A priori, the coordinates of the assertion vector are assigned the value X. Then the critical coordinates are identified, a number of which would be sufficient to carry out a verification experiment in the search for faulty software blocks with specified diagnosis thoroughness. These coordinates are identified by the unities. In the process of simulation vector coordinates are modified to reduce the unities. Each coordinate of the vector A is in compliance with a list of all nodes- ancestors of software transaction graph. The reachability matrix of a transaction graph or lists of nodes- ancestors correspond to the vector coordinates.

For the actual binary state the elements of the vector A unconditional diagnosis procedure is performed for the list L to diagnose faulty software blocks d (A), defined by the following expressions [18]:

$$\begin{cases} L_s(A) = \left(\underset{\forall i(A_i = 1)}{\mathrm{I}} A_i \right) \setminus \left(\underset{\forall i(A_i = 0)}{\mathrm{Y}} A_i \right); \\ L_m(A) = \left(\underset{\forall i(A_i = 1)}{\mathrm{Y}} A_i \right) \setminus \left(\underset{\forall i(A_i = 0)}{\mathrm{Y}} A_i \right). \end{cases} \quad (2)$$

The system of equations is to search for single and multiple errors by using the assertion state vector. The length of assertion vector is equal to the number of nodes in a graph or the number of program blocks in the functional-logical structure of HDL-code. Vector model for verification environment is presented below:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_i \\ T_n \end{bmatrix} \oplus \begin{bmatrix} S_1 \\ S_2 \\ S_i \\ S_n \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ A_i \\ A_n \end{bmatrix} \overset{d}{\rightarrow} \begin{bmatrix} L_1 \\ L_2 \\ L_i \\ L_n \end{bmatrix} \quad (3)$$

In the simulation the comparison of responses for testbench and HDL-model is carried out; it forms the coordinate states of assertion vector.

$$A_i = f(T_i, S_i) = T_i \oplus S_i,\ A_i = \{0,1,X\}$$

Then, essential (0,1)- coordinates of an assertion vector mask a reachability matrix for obtaining a list of program blocks with errors by performing one of the procedures set out in (2).

4. Testability analysis for software HDL-models

Substantial redundancy of the HDL-model assumes its efficient use in order to improve testability of structure for designed code. Existing standards for hardware testable design can be adapted to verify the HDL-code of system and register software models.

S.G. Sharshunov's register or transaction transfer graph that provides information about interrelations between Boolean and register variables, memory and interface buses, can be used. This data can be obtained automatically, by analyzing the syntax of HDL-code lines. Generated graph should cover the functionalities of the software model components and define all the existing links for the reception, transmission and transformation of information between the nodes of a transaction graph (TG). For integral estimating the testability Q of transaction graph the following criteria are introduced:

$$Q = \frac{Z(S)}{Z(S) + Z(F) + Z(T) + Z(A)} \times \frac{1}{n} \sum_{i=1}^{n} (U_i \times N_i);$$
$$U_i = \frac{1}{T} \sum_{j=1}^{x_i} T_j^i \times \frac{1}{d_i^x \vee t_i^x}; \quad N_i = \frac{1}{T} \sum_{j=1}^{y_i} T_j^i \times \frac{1}{d_i^y \vee t_i^y}. \qquad \textbf{(4)}$$

Controllability and observability is the metrics for testability estimation not interconnections, but components of HDL-code, such as: registers, counters, memory or arrays, input-output buses, vectors, logical or arithmetic variables.

These characteristics have a functional dependence on the structural granularity of a component (relative to the inputs) or the length of conjunctive term – $d_i^x \vee t_i^x$, ($d_i^y \vee t_i^y$ – relative to the outputs), as well as on the percentage of the command quantity $\frac{1}{T} \sum_{j=1}^{x_i} T_j^i$, having input (output – $\frac{1}{T} \sum_{j=1}^{y_i} T_j^i$) access to a node at the analysis of the software. Testability Q, presented in (4), depends on the controllability U, observability N, as well as model redundancy (Z) provided by the components: metrics of the functional coverage (F), testbench (B), assertion engine (A). The controllability (observability) is a function of the number of operators, incoming into the node (outcoming from the node) of a transaction graph, as well as the structural granularity of the element – the distance from the inputs (outputs) or the number of time cycles, required for the component control (observation) in given state in the time axis.

This testability criterion can also be used to estimate the quality of the control flow graph for computational processes. Only the statement nodes, loaded with the input conditions, as well as the node position relative to the beginning or the end of control circuit are considered. The position of the statement node is correlated with control cycle of computational process. Number of conditions for the execution of an operation set at each node, joint by operation Or, increases the testability of a graph in controllability. Similarly the observability is determined, on which affect not only the structural granularity, but the power of conditions, created by the functional operations And, Or. The testability of a node for oriented graph may be described by logical function, defined as a conjunctive normal form CNF. At that the testability (controllability and observability) will be determined by Quine estimation of CNF computational complexity. In general, the logic functions of controllability and observability for current node of the transaction graph are determined by conjunction of the disjunctive terms – the first line of (5):

$$\begin{aligned} &1)\ U_r^f = \bigwedge_{i=1}^{n_r^x}\left(\bigvee_{j=1}^{x_i} T_{rij}^x\right);\ N_r^f = \bigwedge_{i=1}^{n_r^y}\left(\bigvee_{j=1}^{y_i} T_{rij}^y\right); \\ &2)\ U_r^f = \bigvee_{i=1}^{x_r}\left(\bigwedge_{j=1}^{n_i^x} T_{ij}^x\right);\ N_r^f = \bigvee_{i=1}^{y_r}\left(\bigwedge_{j=1}^{n_i^y} T_{ij}^y\right). \end{aligned} \quad (5)$$

Here the controllability U_r^f (observability N_r^f) function is determined by conjunction of all nodes-ancestors n_r^x (successors n_r^y), where each of them have x_i incoming (y_i outgoing) arcs-transactions connected by disjunction signs.

The power of disjunctive terms corresponds to the number of arcs incoming into the node, but the number of conjunctions is the structural granularity of the component location.

Then the conjunctive form is transformed to DNF – the second line of (5), where the number of terms for the controllability (observability) function $x_r (y_r)$ is equal to all possible ways of forming the state for a node, and the length of controllability (observability) term $n_i^x (n_i^y)$ is a condition for the reachability of a node – the structural granularity from the inputs (outputs).

The interesting solution when the controllability and observability for the current node of transaction graph is determined on the basis of the logical functions of the observability and controllability ($^{U_i, N_i}$) and the integral testability estimation (Q), when using the apparatus – the algebraic form of graph representation [18]. The formulas for calculating these criteria are the following:

$$U_i = \frac{1}{t^x_{max} \times n^x_t} \times \sum_{i=1}^{n^x_t} \sum_{j=1}^{k^x_i} (t^x_{max} - \left| t^x_{ij} \right| + 1);$$

$$N_i = \frac{1}{t^y_{max} \times n^y_t} \times \sum_{i=1}^{n^y_t} \sum_{j=1}^{k^y_i} (t^y_{max} - \left| t^y_{ij} \right| + 1); \quad (6)$$

$$Q = \frac{1}{n} \sum_{i=1}^{n} (U_i \times N_i),$$

where $t^x_{max}, n^x_t, k^x_i, \left| t^x_{ij} \right|$ are the conjunctive term of maximum length for determining the controllability criterion; a number of terms in a logical controllability function; a number of transactions (letters) in current function term; power of the transaction in a term. Similar designations are used for calculating the observability criterion for each node of transaction graph – $t^y_{max}, n^y_t, k^y_i, \left| t^y_{ij} \right|$.

5. Verification of DCT IP-core, Xilinx

The proposed verification models for software HDL-code are tested on a real project Xilinx IP-core in order to determine the presence of errors. At that the positive result about the semantics of the program is obtained for subsequent correction of the code. Fragment of the discrete cosine transform module is presented in Listing 1 [Xilinx.com]. The whole HDL-model consists of 900 lines of System Verilog code.

Listing 1.

```
module Xilinx
‘timescale 1ns/10ps
module dct ( CLK, RST, xin,dct_2d,rdy_out);
output [11:0] dct_2d;
input CLK, RST;
```

```
input[7:0] xin; /* input */
output rdy_out;
wire[11:0] dct_2d;
……
/* The first 1D-DCT output becomes valid after 14 +64 clk cycles. For the first 2D-DCT output
to be valid it takes 78 + 1clk to write into the ram + 1clk to write out of the ram + 8 clks to shift
in the 1D-DCT values + 1clk to register the 1D-DCT values + 1clk to add/sub + 1clk to take
compliment + 1 clk for multiplying +  2clks to add product. So the 2D-DCT output will be valid
at the 94th clk. rdy_out goes high at 93rd clk so that the first data is valid for the next block*/
endmodule
```

In accordance with the rules of testability analysis described above, the transaction graph is designed presented in Fig. 5, which for the module Xilinx has 28 nodes-components (input and output buses, logic and register variables, vectors and memory). Arc identifier has the upper index meaning the number of transactions in the program between the outcoming and incoming nodes. For each node the logical function of controllability and observability are constructed. Example of the controllability function for the node B_2 is the following:

$$B_2 = T_{28}^{12} T_{27}^{2} T_{22}^{2} T_{25}^{2} T_{24}^{4} (T_5^1 \vee T_6^1 \vee T_{12}^4 T_{13}^2 T_{14}^2 T_{15}^2 T_{15}^{64} T_{17}^{64} T_{18}^4 T_{19}^4 T_{20}^{64} T_{21}^8 T_{22}^8 \wedge$$

$$\wedge T_{23}^8 (T_1^{64} T_7^4 \vee T_{11}^4 T_2^{64} T_8^8 T_9^8 T_{10}^4 \vee T_{11}^4 T_3^1 \vee T_{11}^4 T_4^1)) =$$

$$= T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 T_5^1 \vee T_6^1 T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 \vee T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 T_{12}^4 T_{13}^2 T_{14}^2 T_{15}^2 T_{15}^{64} T_{17}^{64} T_{18}^4 \wedge$$

$$\wedge T_{19}^4 T_{20}^{64} T_{21}^8 T_{22}^8 T_{23}^8 T_1^{64} T_7^4 \vee T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 T_{12}^4 T_{13}^2 T_{14}^2 T_{15}^2 T_{15}^{64} T_{17}^{64} \wedge$$

$$\wedge T_{18}^4 T_{19}^4 T_{20}^{64} T_{21}^8 T_{22}^8 T_{23}^8 T_{11}^4 T_2^{64} T_8^8 T_9^8 T_{10}^4 \vee T_{11}^4 T_3^1 T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 T_{12}^4 T_{13}^2 T_{14}^2 T_{15}^2 \wedge$$

$$\wedge T_{15}^{64} T_{17}^{64} T_{18}^4 T_{19}^4 T_{20}^{64} T_{21}^8 T_{22}^8 T_{23}^8 \vee T_{11}^4 T_4^1 T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 T_{12}^4 T_{13}^2 T_{14}^2 \wedge$$

$$\wedge T_{15}^2 T_{15}^{64} T_{17}^{64} T_{18}^4 T_{19}^4 T_{20}^{64} T_{21}^8 T_{22}^8 T_{23}^8.$$

For the node L_1 DNF of observability function is presented in the following form:

$$N(L_1) = T_{28}^{12} T_{27}^2 T_{22}^2 T_{25}^2 T_{24}^4 T_{23}^8 T_{22}^8 T_{21}^8 T_{20}^{64} T_{19}^4 T_{18}^4 T_{17}^{64} \wedge T_{16}^{64} T_{15}^2 T_{14}^2 T_7^4 T_1^{64}.$$

The result of calculating the controllability, observability and testability for the graph (Fig. 5) is shown in Fig. 6.

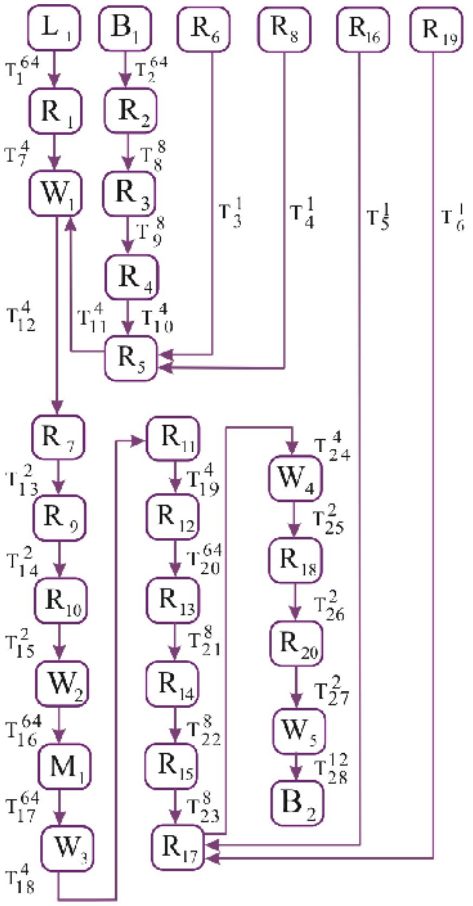

Figure (5): *Transaction graph for Xilinx model*

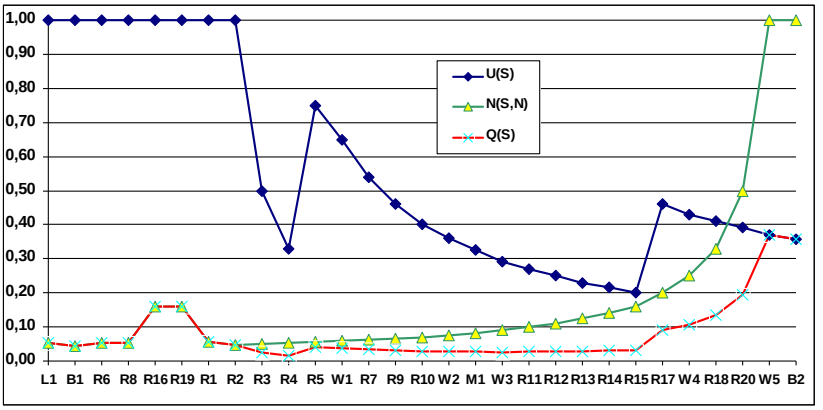

Figure (6): *Testability curve for Xilinx model*

After determination of controllability and observability for transaction graph nodes the calculation of the generalized testability criterion for each component of software code is carried out in accordance with expression (6). For the Xilinx model such estimation is equal to 0,382. It describes the quality of the design variant, which is very important when comparing several alternative solutions. As an example of the positive use of developed models and methods the testability analysis for software code of discrete cosine transform from the Xilinx library is performed. It was done construction of transaction model, calculation of the testability characteristics, and identification of critical points.

6. Conclusions:

1. An universal model of software component in the form of transaction graph, which can be used to solve the problems of testability analysis in order to obtain the required diagnosis granularity for HDL-code is presented.
2. The logic testability functions for HDL-models on the basis of transaction graph are proposed in order to determine the testability estimates (controllability and observability) for software components and HDL-design as a whole.
3. Examples and testability curves (controllability and observability) for software models represented by transaction graphs are made.
4. The practical significance of the proposed methods and models is the market appeal and high interest from the technology companies to the innovation solutions of the effective testing and verification of hardware and software components for system-level design in order to reduce time-to-market and raise yield.
5. Further research will be focused on the development of standard interfaces for integration of models, methods and software in the technological design flows of digital systems on chips.

References:

[1] M. Abramovici, M.A. Breuer and A.D. Friedman, *Digital System Testing and Testable Design*, Computer Science Press, 1998, 652 p.

[2] Ismet Bayraktaroglu, Alex Orailoglu, *The Construction of Optimal Deterministic Partitionings in Scan-Based BIST Fault Diagnosis: Mathematical Foundations and Cost-Effective Implementations,* IEEE Transactions on Computers.– P.61-75, 2005.

[3] Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli, *A Platform-Based taxonomy for ESL Design*, Design&Test of computers, P. 359-373, September-October 2006.
[4] Francisco DaSilva, Yervant Zorian, Lee Whetsel, Karim Arabi, Rohit Kapur, *Overview of the IEEE P1500 Standard*, ITC International Test Conference, P. 988–997, 2003.
[5] P. Rashinkar, P. Paterson, L. Singh, *System-on-chip Verification: Methodology and Techniques*, Kluwer Academic Publishers, 2002, 324 p.
[6] Yervant Zorian, *What is Infrustructure IP?*, IEEE Design & Test of Computers, P. 5-7, 2002.
[7] Yervant Zorian, Dmytris Gizopoulos, *Gest editors' introduction: Design for Yield and reliability*, IEEE Design & Test of Computers, P. 177-182, 2004.
[8] Yervant Zorian, *Guest Editor's Introduction: Advances in Infrastructure IP*, IEEE Design and Test of Computers, 2003, 49 p.
[9] S.M. Thatte, J.A. Abraham, *Test generation for microprocessors*, IEEE Trans. Comput., C-29, No 6, P. 429-441, 1980.
[10] S.G. Sharshunov, *Test Genagation for Microprocessors. 1. General Model. Check of data processing*, Automation and Telemechanics, №11, P.145-155, 1985.
[11] A.A. Jerraya, *System Level Synthesis SLS. TIMA Laboratory. Annual Report*, P. 65-75, 2002.
[12] Frank Ghenassia, *Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems*, Published by Springer, 2005, 282 p.
[13] Janick Bergeron, *Writing testbenches: functional verification of HDL models*, Boston: Kluwer Academic Publishers, 2001, 354 p.
[14] Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale, *Verification Methodology. Manual for SystemVerilog*, Springer, 2005, 528 p.
[15] Harry Foster, Adam Krolnik, David Lacey, *Assertion-based design*, Second edition, Kluwer Academic Publishers, Springer, 2005, 392 p.
[16] P. Rashinkar, P. Paterson, L. Singh, *System-on-chip Verification: Methodology and Techniques*, Kluwer Academic Publishers, 2002, 393 p.
[17] A.S. Meyer, *Principles of Functional Verification*, Elsevier Science, 2004, 206 p.
[18] V.I. Hahanov, E.I. Litvinova, O.A. Guz, *Design and Testing of Digital System-on-Chip*.– Kharkov: Novoye Slovo, 2009, 484 p.