USING SAT FOR COMBINATIONAL IMPLEMENTATION CHECKING

Liudmila Cheremisinova, Dmitry Novikov

Abstract. The problem of checking whether a system of incompletely specified Boolean functions is implemented by the given combinational circuit is considered. The task is reduced to testing out if two given logical descriptions are equivalent on the domain of one of them having functional indeterminacy. We present a novel SAT-based verification method that is used for testing whether the given circuit satisfies all the conditions represented by the system of incompletely specified Boolean functions.

Keywords: design automation, verification, simulation, SAT.

ACM Classification Keywords: B.6.2 [Logic Design]: Reliability and Testing; G.4 [Mathematical Software]: Verification; B.6.2 [Reliability and Testing]: Error-checking.

Introduction

Verification takes place at all design levels: from the conceptual design down to the design of combinational circuits. In this process, the objective of verification is to ensure that implemented and specified behaviors are the same; at this point, the design is error-free. Validating the functionality of digital circuits and systems is an increasingly difficult task. Multiple chip design projects are reporting that approximately 70% of their design time is spent in verification. This is due to the growing complexity of the designs that has not been accompanied by improvements in functional verification techniques.

While design tools have been made, and they, at least partially, support the complexity of highly integrated designs, there is a lack of verification support. Widely used tools for verification are logic simulators. At present, logic simulation is the most widely used technique for ensuring the correctness of digital integrated circuits in industry because of its scalability and predictable run-time behavior. This technique is based on verifying a digital system by stimulating inputs of the circuit with binary signals values that propagate in the circuit leading to a corresponding activation of the outputs, whose values must be consistent with the expected ones.

The past ten years have seen efforts in developing commercial formal verification tools. Instead of testing all input combinations explicitly (as simulators do), they formally prove a circuit's functionality to be consistent with its specification. Formal verification techniques have the potential of providing more general results than traditional simulation methods: it is possible to guarantee that a specific property holds for a design under all possible input stimuli. Now combinational equivalence checking (CEC) plays an important role in VLSI design; its usual application is verifying functional equivalence of combinational circuits after multi-level logic synthesis. In a typical scenario, there are two structurally similar implementations of the same design, and the problem is to prove their functional equivalence. This problem was addressed in numerous research publications, some of them are referenced in [Drechsler, 2000; Kropf, 1999; Mishchenko, 2006; Ganai, 2002; Kuehlmann, 2002; Kunz, 2002]. In a modern CEC flow based on formal verification approach, both circuits to be verified are transformed into a single circuit called a miter. It is derived by combining the pairs of inputs with the same names and feeding the pairs of outputs with the same names into EXOR gates, which are ORed to produce the single output of the miter. The miter is a combinational circuit with the same inputs as the original circuit and there is constant 0 on its output if and only if the two original circuits produce identical output values under all possible input assignments.

In the paper, the verification task is examined for a case, when desired functionality of the system under design is incompletely specified, i.e. intended behavior of implemented design allows functional indeterminacy. Such a case usually occurs on early stages of designing when assignments to primary inputs of designed device exist which will never arise during normal mode of the device usage. Thus when hardware implementing this device, its outputs in response of these inputs may be arbitrary defined. In this case verification methodologies must consider only possible input scenarios to the design under verification and verify that every possible output signal of implemented behavior has its intended (described in initial specification) value.

The considered case could be thought as solvable by means of simulation-based tools of verification. But we propose a Boolean satisfiability-approach (SAT-approach) of checking whether a given combinational circuit implements a particular design specification. In this paper, we propose the following contributions to the problem of combinational verification: 1) the behaviour of the combinational circuit to be designed is specified with functional indeterminacy; 2) we show how it is possible to use SAT tools [Goldberg, 2002] for the considered case.

So in this paper we consider the verification problem for the case, when: 1) desired incompletely specified functionality is given in the form of a system of incompletely specified Boolean functions; 2) functions of the system are specified on intervals (cubes) of values of Boolean input variables and these intervals are large enough; 3) the system is implemented in the form of a combinational circuit in the basis of the elementary gates AND, OR and NOT.

We will discuss a novel SAT based method for testing whether the given circuit implements all multiple-output cubes representing the system of incompletely specified Boolean functions.

Background

Vectors and assignments. Let us consider a Boolean vector $x = (x_1, x_2, \dots, x_n)$ of input variables. Let us call a set of equalities of type $x_i = \sigma_i$ (where $\sigma_i \in \{0, 1\}$, $i = \{1, 2, \dots, n\}$) as a variable value assignment a for the vector x of input variables. A variable value assignment a for the vector x can be a complete if all x_i are assigned or a partial otherwise. In the last case some of variables may be don't-care, meaning that any assignment to these variables is permissible. A complete variable value assignment represents a minterm and partial assignment represents a cube in n-dimensional Boolean space $E^n = \{0,1\}^n$. A cube represents a product of literals (from now on, literal is a Boolean variable or its negation). A cube of the size k fixes values of exactly k variables and covers 2^{n-k} minterms. In general case a cube c_k (and the appropriate product) covers another cube c_l (and the appropriate product) if the literals of c_k are a subset of the literals in c_l.

Boolean functions. A completely specified Boolean function (CSF) $f(x) = f(x_1, x_2, \dots, x_n)$ is a many-to-one mapping from n-dimensional ($n \geq 0$) Boolean space into a single-dimensional one: $E^n \rightarrow E$. A don't-care for a logic function allows it to have either 0 or 1 as a possible value. If, for some input combinations (minterm), the output of the function is a don't-care, this Boolean function is called as incompletely specified one (ISF). ISF is a mapping $E^n \rightarrow \{0,1,–\}^m$, where the symbol "–" denotes don't-care condition. A CSF has only care minterms, which correspond to the assignments, for which it takes values 0 or 1. An ISF additionally has don't-care minterms, which correspond to the assignments, for which the function is flexible and can be either 0 or 1.

CSF $f(x)$ is specified by a pair of sets U_f^1 and U_f^0 of cubes (or minterms) that represent its on-set and off-set that divide Boolean space E^n into two parts. In the case of ISF there exists a don't-care set $E^n \setminus (U_f^1 \cup U_f^0)$ of cubes (that is not empty). A CSF $g(x)$ implements an ISF $f(x)$, if the CSF can be derived from the ISF by assigning either 0 or 1 to each don't-care minterm or, that is the same, if

$$U_f^1 \subseteq U_g^1, \; U_f^0 \subseteq U_g^0 \qquad (1)$$

A system of Boolean functions $F = \{f_1(X), f_2(X), \dots, f_m(X)\}$ (or $\boldsymbol{f}(x)$ in the vector notation, where x and $\boldsymbol{f}$ are vectors of input and output variables) of completely specified Boolean functions (CSF) is a mapping between

n-dimensional and m-dimensional Boolean spaces. In the case of ISF don't-care minterms may differ for different functions.

Let us specify a system $f(x)$ of ISFs as a set of multiple-output cubes. A multiple-output cube $(\boldsymbol{u}, \boldsymbol{t})$ is a pair of ternary vectors (products) of dimensions n and m that are called as its input and output parts correspondingly. The input part $\boldsymbol{u}$ represents a cube in E^n or a product of some literals $x_i \in X$. The output part $\boldsymbol{t}$ is a ternary vector of values of functions for the cube $\boldsymbol{u}$ or a product of some literals $f_i \in F$. For each $f_i \in F$ the j-th entry t^j of $\boldsymbol{t}$ is 1 or 0 ($t^j = 0$) if all the minterms covered by the cube $\boldsymbol{u}$ are in the on-set U_{fj}^1 or in the off-set U_{fj}^0 correspondingly; otherwise t^j is don't-care. For example, for the case $(\boldsymbol{u}, \boldsymbol{t})$ = (– 0 1 0 –; – 0) (or $(\boldsymbol{u}, \boldsymbol{t}) = (\bar{x}_2 x_3 \bar{x}_4;\ \bar{f}_2)$) we may state that all four minterms (belonging to the interval – 0 1 0 –) 0 0 1 0 0, 0 0 1 0 1, 1 0 1 0 0, 1 0 1 0 1 $\in U_{f2}^0$ and $f_2(x_1,0,1,0,x_5) = 0$ but we can say nothing about the value of the function f_1 on these minterms: some of them can be care some don't-care ones.

A Conjunctive normal form (CNF) represents a Boolean function as conjunction of one or more clauses, each being in its turn a disjunction of literals. From now on, we consider only clauses that do not simultaneously contain a literal and its negation.

A CNF denotes a unique completely specified Boolean function $f(x)$ and each of its clauses corresponds to an implicate of the function. A Boolean variable can be assigned a truth value (0 or 1). Also, clauses and CNF may assume values depending respectively on the values of the corresponding literals and clauses. CNF representation is popular among SAT algorithms because each clause must be satisfied (evaluate to 1) for the overall CNF to be satisfied. The SAT problem is concerned with finding an assignment $X' \rightarrow \{0,1\}$ to the variables of some subset $X' \subseteq X$ that makes CNF equal to 1 or proving that it is equal to the constant 0. If the first outcome takes place they say that the CNF is satisfied and refer to $X' \rightarrow \{0,1\}$ as a satisfying assignment.

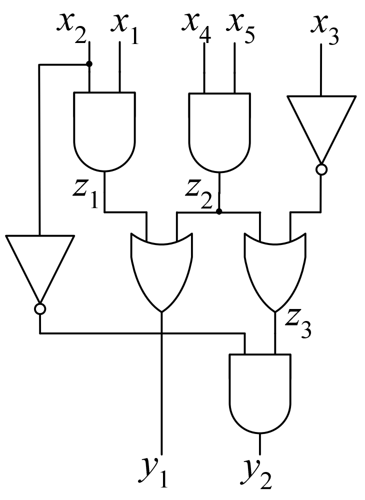
Matrix Models of Boolean functions and CNF. Matrix representation of CNF formula C containing k clauses and n distinct variables is a ternary matrix C having a row for each clause and a column for each variable. The entry c_i^j of the matrix in the i-th row and the j-th column is 1, 0 or "–" depending on in what a form (x_j or $\bar{x}_j$) the variable x_j appears or does not appear in i-th clause of C. The same manner, matrix representation of ISF $f(x)$ is a pair of ternary matrixes U_f^1 and U_f^0 having rows for all cubes from U_f^1 and U_f^0, correspondingly.

The system $f(x)$ of ISFs given by the set S of multiple-output cubes $(\boldsymbol{u}_i, \boldsymbol{t}_i)$ can be represented by a pair of ternary matrices U and T of the same cardinality (Fig. 1). The matrix U contains as its rows all input parts of multiple-output cubes from S; similarly matrix T specifies as its rows all output parts.

$$U = \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \\ – & – & 1 & 1 & 1 \\ 1 & 1 & – & – & – \\ – & 0 & 0 & 0 & – \\ 0 & 1 & – & 1 & 0 \\ – & 0 & 1 & 0 & – \\ – & 1 & – & 1 & 1 \end{array} \qquad T = \begin{array}{ccc} f_1 & f_2 & \\ 1 & – & 1 \\ 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 4 \\ – & 0 & 5 \\ – & 1 & 6 \end{array}$$

Figure 1: An example of ISF system

Representation of a function in multiple-output cubes form has the following distinctive features. Cubes $u_i, u_j \in U$ can intersect each other (in contrast to a representation in the minterm form). Don't-care value of an element t_i^j of the matrix T means that either the function f_j is don't-care on the whole cube u_i or f_j does not take the same value (1, 0 or "–") on the whole interval u_i, i.e. there exist at least two minterms covered by the cube u_i on which f_j has different values. For example, the cubes u_1 = – – 1 1 1 and u_2 = 1 1 – – – intersect on the minterm 1 1 1 1 1 so we may say that $f_2(1,1,1,1,1) = 0$ and f_2 do not take the same value on the whole interval u_1.



x_1	x_2	x_3	x_4	x_5	z_1	z_2	z_3	y_1	y_2	
1	–	–	–	–	0	–	–	–	–	1
–	1	–	–	–	0	–	–	–	–	2
0	0	–	–	–	1	–	–	–	–	3
–	–	–	1	–	–	0	–	–	–	4
–	–	–	–	1	–	0	–	–	–	5
–	–	–	0	0	–	1	–	–	–	6
–	–	0	–	–	–	1	0	–	–	7
–	–	1	–	–	–	–	1	–	–	8
–	–	–	–	–	–	0	1	–	–	9
–	–	–	–	–	1	1	–	0	–	10
–	–	–	–	–	0	–	–	1	–	11
–	–	–	–	–	–	0	–	1	–	12
–	0	–	–	–	–	–	–	–	0	13
–	–	–	–	–	–	–	1	–	0	14
–	1	–	–	–	–	–	0	–	1	15

Figure 2: An example of a combinational circuit: a) the circuit; b) the corresponding conventional CNF

Combinational circuit. A combinational circuit under consideration refers to a gate-level network where primary inputs are connected to primary outputs through an interconnection of basic gates that implement elementary Boolean functions such as AND, OR, NOT, NAND etc. As usual we consider further only acyclic circuits.

The topological description of an acyclic combinational circuit can be represented using a directed acyclic graph, where nodes correspond to the gates, primary inputs and outputs of the circuit; edges correspond to circuit wires connecting the nodes. Incoming edges of a node are called its fanins and outgoing edges are called fanouts. A node in the circuit is multiple fanout if its output is a fanin to different gates. The node and its output signal are named the same. Nodes without fanins are the sources of the graph, called as primary inputs of the circuit; nodes without fanouts are the sinks, called as the primary outputs. Internal nodes of the graph correspond to logical gates implementing elementary Boolean functions. An example of a circuit (that will be tested later whether it implements the system of two ISFs depicted in Fig. 1) with five inputs, two outputs and seven gates is shown in Fig. 2,*a*. Here AND, OR, NOT gates are used as the basic ones.

Let us call the functionality of a circuit node in terms of its immediate fanins as the local function of the node, and the functionality of a circuit node in terms of the circuit primary inputs as the global function of the node. Thus the functionality of the circuit in terms of its primary inputs is the system of global functions implemented on the circuit primary outputs. For example, the local function of the node connected with primary output y_1 is $y_1 = z_1 \vee z_2$ and the corresponding global function is $y_1 = x_1 x_2 \vee x_4 x_5$.

Simulation-based verification

At present, logic simulation is the most widely used technique for ensuring the correctness of digital integrated circuits in industry. This technique computes the values of the internal signals and primary outputs of a circuit, given the values of its primary inputs. One round of simulation begins with stimulating primary inputs of the simulated circuit with binary signals values simulation and then this one is propagated through the circuit leading to a corresponding activation of the circuit primary outputs, whose values must be consistent with the expected ones. The complexity of simulating a particular set of input values is linear in the simulated circuit size (let us remember we consider only combinational circuit, so there are no internal state variables). But the overall Boolean space of values of n primary input variables contains up to 2^n combinations. Due to the complexity of constructing all these combinations and verifying the compatibility between implementation and specification, simulation is infeasible for state-of-the-art designs.

A special type of simulation is of the most interest: guided simulation, when inputs are assigned based on certain information, provided by the design specification. In our case inputs could be assigned minterms covered by input parts u_i of multiple-output cubes of ISF-system. So the first step on the way of ISF-system verification consists in representing all the multiple-output cubes as multiple-output minterms. Then parallel binary simulation [Cheremisinova, 2008] of the combinational circuit can be performed under all input assignments corresponding to the minterms simultaneously.

Such a method could reduce in some cases the search space of simulators but only in the case when the input parts u_i of multiple-output cubes are “small enough” covering a small number of minterms. However though the specification of the designed circuit with n inputs would be specified with a small number of multi-output implicants, the overall size of Boolean space covered by them could be near to 2^n. So for the case when ISF system contains “big” multiple-output cubes covering a great number of minterms we propose a novel SAT-based method of testing whether the circuit implements such a multiple-output cubes.

SAT-based verification

SAT-solvers can be circuit-based or CNF-based. The former represent the SAT problem as a circuit composed of simple gates, while the latter use conjunctive-normal-form. To tackle the problems of circuit verification using the second type of SAT-solvers, they usually require their input to be in CNF because instances of SAT are usually represented as CNF formulas. This type of solvers is more general and can also be applied to circuits, by converting them into CNF form.

CNF encoding of combinational circuit. Majority of SAT applications derived from circuit representation rely on some a version of the Tseitin transformation for producing conventional CNF of the circuit. A circuit-to-CNF conversion uses as many variables as there are primary inputs and gates in the circuit.

When the conventional transformation is applied to a combinational circuit, for output of each gate (except output ones) its own internal Boolean variable is introduced and only local functions of the gates are considered. Then CNF formula is associated with each gate, and captures the consistent assignments between gate inputs and output. These all the gate local CNFs are joined then in the overall circuit CNF by using the conjunction operation. Both the size of the resulting CNF and the complexity of the conventional translation procedure are linear in the gate number of the original combinational circuit.

The derivation of CNF for a gate representing a local function $y = f(z_1, z_2, \dots, z_k)$ is based on defining a new Boolean function $\varphi(y, f) = y \sim f(z_1, z_2, \dots, z_k)$ (as in [Kunz, 2002]), that is true in the only case when both functions y and $f(z_1, z_2, \dots, z_k)$ assume the same value. Next, the function φ should be represented as a CNF form. As an example consider 2-input AND and OR gates, the formulas φ for them can be transformed the following way:

$$\varphi^{\wedge}(y, f^{\wedge}) = \varphi^{\wedge}(y, z_1,z_2) = y \sim f(z_1,z_2) = y \sim z_1 z_2 = (\bar{y} \vee z_1 z_2)(y \vee \bar{z}_1 \vee \bar{z}_2) = (\bar{y} \vee z_1)(\bar{y} \vee z_2)(y \vee \bar{z}_1 \vee \bar{z}_2);$$

$$\varphi^{\vee}(y, f) = \varphi^{\vee}(y, z_1,z_2) = y \sim f(z_1,z_2) = y \sim z_1 \vee z_2 = (\bar{y} \vee z_1 \vee z_2)(y \vee \bar{z}_1\ \bar{z}_2) = (\bar{y} \vee z_1 \vee z_2)(y \vee \bar{z}_1)(y \vee \bar{z}_2).$$

The above CNF formulas of 2-input AND and OR gates r could be obtained reasoning from the truth table for the relational representations of AND and OR gates, identifying what combinations of the inputs and the gate output are admissible or possible (Fig. 3). In other words, the rightmost column of the truth tables is true iff the characteristic function of $f(z_1,z_2)$ is equal to y [Kropf, 1999.]. Than we construct a clause for each row of the truth table where the final column has 0 by formulating a disjunction of literals z_1, z_2, y in the negated form relative to the values of the considered truth table. The disjunction of all these clauses results in the CNF formula. So for 2-input AND and OR gates we will obtain the following representations (the same as ones shown above):

$$\varphi^{\wedge}(y,z_1,z_2) = (\bar{y} \vee z_1 \vee z_2)(\bar{y} \vee z_1 \vee \bar{z}_2)(\bar{y} \vee \bar{z}_1 \vee z_2)(y \vee \bar{z}_1 \vee \bar{z}_2) = (\bar{y} \vee z_1)(\bar{y} \vee z_2)(y \vee \bar{z}_1 \vee \bar{z}_2);$$

$$\varphi^{\vee}(y,z_1,z_2) = (\bar{y} \vee z_1 \vee z_2)(y \vee z_1 \vee \bar{z}_2)(y \vee \bar{z}_1 \vee z_2)(y \vee \bar{z}_1 \vee \bar{z}_2) = (\bar{y} \vee z_1 \vee z_2)(y \vee \bar{z}_1)(y \vee \bar{z}_2).$$

z_1	z_2	$y^{\wedge}$	$\varphi^{\wedge}$	z_1	z_2	$y^{\vee}$	$\varphi^{\vee}$
0	0	0	1	0	0	0	1
0	1	0	1	0	1	1	1
1	0	0	1	1	0	1	1
1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	0
0	1	1	0	0	1	0	0
1	0	1	0	1	0	0	0
1	1	0	0	1	1	0	0

Figure 3: Truth tables of the 2-input AND and OR functions

For the general case here are the conventional CNF representations of NOT, *k*-input AND and OR gates comprising the example circuit in Fig. 2, *a*:

$$\varphi^{\neg}(y,z) = (z \vee y)(\bar{z} \vee \bar{y});$$

$$\varphi^{\wedge}(y,z_1,z_2,\dots,z_k) = (z_1 \vee \bar{y})(z_2 \vee \bar{y})\dots(z_k \vee \bar{y})(\bar{z}_1 \vee \bar{z}_2 \vee \dots \vee \bar{z}_k \vee y);$$

$$\varphi^{\vee}(y,z_1,z_2,\dots,z_k) = (\bar{z}_1 \vee y)(\bar{z}_2 \vee y)\dots(\bar{z}_k \vee y)(z_1 \vee z_2 \vee \dots \vee z_k \vee \bar{y}).$$

It is possible to eliminate the output variable y of NOT $y = \bar{z}$ gate and two appropriate clauses of circuit conventional CNF if to subsume y in fanout gates of the NOT gate replacing all instances of y with the negated input variable z of this gate. Fig. 2, *b* shows a circuit and its conventional CNF.

Conventional CNF analysis. SAT problem for a CNF formula is formulated as follows [Kunz, 2002]: given a CNF formula representing a Boolean function $y = f(x_1, x_2, \dots, x_n)$, the problem consists of identifying a set of assignments to the formula variables, $\{x_1 = a_1, x_2 = a_2, \dots, x_n = a_n\}$, such that all CNF clauses are satisfied (taking into account that a clause is satisfied if at least one its literal is equal to 1, i.e., $f(a_1, a_2, \dots, a_n) = 1$, or proving that no such assignment exists. Recall that a CNF formula is satisfiable if there exists an assignment $\{x_1 = a_1, x_2 = a_2, \dots, x_n = a_n\}$ providing $f(a_1, a_2, \dots, a_n) = 1$. This assignment is known as a satisfying assignment.

When we have a multiple-output circuit we can state a problem of finding out primary input assignments making some primary output to be one. To test whether the output be 1 the unit clause (a clause consisting of the only literal) corresponding to the tested output y_i is added to the circuit CNF. Once the overall problem is formulated in CNF, a SAT solver can be used to solve it. The resulting satisfying assignments of the circuit CNF and only they form the on-set of the global function y_i and furthermore only on-set minterms satisfy the CNF. For example, after adding to the CNF of the circuit (Fig. 2) the unit clause y_1 and testing CNF for existence partial assignments satisfying every clause one could find CNF partial assignments (for the subspace restricted by the function y_1) given in Fig. 5.

In general case let y^{σ} (where $\sigma \in \{0, 1\}$) be a literal of the variable y, precisely, $y^1 = y$ and $y^0 = \bar{y}$. Then the unit clause y^{σ} represents the assignment $y = \sigma$. Let we have to check whether an output y_i of a circuit to be constantly σ. We cannot directly use SAT solver to show that this statement is true but we could without problems to prove using SAT solver that y_i equals (in the case when y_i is not the constant σ) or does not equal to $\bar{\sigma}$ for some argument assignments. If we cannot prove y_i equals $\bar{\sigma}$ for some arguments assignments thereby we prove that y_i is the constant σ.

x_1	x_2	x_3	x_4	x_5	z_1	z_2	y_1
1	1	–	1	1	1	1	1
–	0	–	1	1	0	1	1
0	–	–	1	1	0	1	1
1	1	–	0	–	1	0	1
1	1	–	–	0	1	0	1

Figure 4: CNF partial assignments of the circuit CNF (Fig. 2,*b*) appended by the unit clause y_1

Such proving is known as proving by contradiction, often it is very convenient when using SAT solver. Thus to show that circuit primary output y_i is the constant σ using prove by contradiction we suppose it is not. At the first step we add to the circuit CNF the unit clause $y_i^{\bar{\sigma}}$ and search using SAT solver for a satisfying assignment *a* to CNF formula for which $y_i(a) = \bar{\sigma}$ is true. Such an assignment $a \cup \{y_i = \bar{\sigma}\}$ is called as a counter-example. If there exists no counter-example then the circuit implements the constant σ on the output y_i.

SAT-Based Model of Testing of Multiple-Output Cubes of ISF System

Assume we have some CNF formula *C* describing a circuit and a system $f(x)$ of ISFs the arguments and the functions of which correspond to primary inputs and outputs of the circuit. Hereinafter let us consider as an example the system $f(x)$ and the circuit (and CNF specifying it) shown in Figures 1, 2.

A problem under discussion is verifying if the given circuit implements the ISF system $f(x)$. It is right if it takes place for each pair of ISF $f_i(x) \in f(x)$ and the appropriate circuit output $y_i(x)$. The global CSF $y_i(x)$, realized by the *i*-th circuit output, implements a function $f_i(x) \in f(x)$ iff (1) takes place.

A multiple-output cube $(u_i, \mathbf{t}_i)$ of ISF system imposes conditions on values of some functions f_j: $f_j(u_i) = t_i^j$ for all *j* for which $t_i^j \in \{1, 0\}$. Further, we are interested in only those components of $\mathbf{t}_i$ which are not don't-care. The truth of the conditions (1) guarantees the circuit has the same functionality as the ISFs system: for every input stimulus implied by input part of any multiple-output cube $(u_i, \mathbf{t}_i)$ the Boolean vector of values of the circuit outputs is covered by the output part $\mathbf{t}_i$. In terms of circuit CNF the conditions (1) could be reformulated as follows. For every multiple-output cube $(u_i, \mathbf{t}_i) \in \mathbf{f}(x)$ a partial value assignment $u_i \cup \mathbf{t}_i$ of input and output variables should be satisfying assignment for circuit CNF. Below, cubes $(u_i, \mathbf{t}_i)$ will be checked if they are implemented by CNF one by one, with no particular order.

Let us consider in more details the procedure of SAT-solving for an elementary cube $(u_i, t_i^j) \in (u_i, \mathbf{t}_i)$, where $t_i^j = f_j(u_i) = \sigma$ and $\sigma \in \{1, 0\}$. At the first step, keeping in mind the prove by contradiction, we assign $f_j(u_i)$ to be $\bar{\sigma}$ (where $\sigma = t_i^j$) i.e. we suppose $(u_i, y_j^{\bar{\sigma}})$ takes place and reduce the circuit CNF *C* making a set of assignments setting all literals of $u_i \cup y_j^{\bar{\sigma}}$ to 1 obtaining $C(u_i \cup y_j^{\bar{\sigma}})$.

For instance, the set of initial assignments for (u_6, t_6^2) (Fig. 1) will be $x_2 = x_4 = x_5 = 1$, $y_2 = 0$. This means that all clauses of *C* having at least one of the literals x_2, x_4, x_5, $\bar{y}_2$ are discarded and literals $\bar{x}_2$, $\bar{x}_4$, $\bar{x}_5$ and y_2 are removed from all clauses having them. After making these assignments we search for a satisfying assignment *a* to the obtained CNF $C(u_i \cup y_j^{\bar{\sigma}})$. In our case for CNF $C(x_2 = 1, x_4 = 1, x_5 = 1, y_2 = 0)$ there exists such an assignment, for instance, $x_1 = z_1 = z_2 = z_3 = y_1 = 1$ (1––––1111–). That proves that there exists a counter-example for (u_6, t_6^2), proving that the global function $y_2(x)$ of the circuit does not implement $f_2(x)$ for some input pattern from u_6: $f_2(u_6) \neq y_2(u_6)$. This conflicting input pattern in our case is 11–11.

The above procedure is applicable for the case when the output part $\mathbf{t}_i$ of a multiple-output cube $(u_i, \mathbf{t}_i)$ consists of the only component having definite value (0 or 1). But in general case the output part $\mathbf{t}_i = y_{i1}^{\sigma 1} y_{i2}^{\sigma 2} \ldots y_{ik}^{\sigma k}$ of a multiple-output cube $(u_i, \mathbf{t}_i)$ consists of more than one component, for instance *k*, having definite values. Proof by contradiction which tries to find a counter-example forces to test the following assignment:

$$u_i \cup \overline{t_i} = u_i \cup \neg(y_{i1}^{\sigma_1} y_{i2}^{\sigma_2} \dots y_{ik}^{\sigma_k}) = u_i \cup (y_{i1}^{\overline{\sigma}_1} \vee y_{i2}^{\overline{\sigma}_2} \vee \dots \vee y_{ik}^{\overline{\sigma}_k}). \qquad (2)$$

So in this case we make initial assignments setting only literals of u_i to 1, then add to CNF $C(u_i)$ the clause $y_{i1}^{\overline{\sigma}_1} \vee y_{i2}^{\overline{\sigma}_2} \vee \dots \vee y_{ik}^{\overline{\sigma}_k}$ ($y_{ij}^{\overline{\sigma}_j} = 0, 1$). Or, it is the same, only add to CNF $l + 1$ clauses (where l is the number of literals in u_i): l unit clauses of the type x_j ($x_j \in u_i$) and a clause $y_{i1}^{\overline{\sigma}_1} \vee y_{i2}^{\overline{\sigma}_2} \vee \dots \vee y_{ik}^{\overline{\sigma}_k}$ of size k.

For example the cube (u_2, t_2) implies three clauses: x_1, x_2, $\overline{y}_1 \vee y_2$ to be added to CNF. For the extended CNF there exists no satisfying assignment, that fact proves the values of the global functions $y_1(x)$ and $y_2(x)$ are equal correspondingly to $f_1(u_2)$ and $f_2(u_2)$ for all input patterns from u_2: $f_1(u_2) = y_1(u_2)$, $f_2(u_2) = y_2(u_2)$.

Conclusion

In this paper, we propose the following contributions to the problem of combinational verification.

1. We consider a case when one of the compared descriptions is incompletely specified.

2. We show how it is possible to use SAT tools for the considered case.

3. We suppose the method of checking whether multiple-output cubes of ISF are implemented by circuit conventional CNF.

Bibliography

[Drechsler, 2000] R. Drechsler. Formal Verication of Circuits. Kluwer Academic Publishers, 2000.

[Kropf, 1999] T. Kropf. Introduction to Formal Hardware Verification. Springer, 1999.

[Mishchenko, 2006]. A. Mishchenko, S. Chatterjee, R. Brayton, N. Een. Improvements to Combinational Equivalence Checking. In: Proc. ICCAD'06, Nov. 5–9, 2006, San Jose, CA, 2006.

[Ganai, 2002]. M.K. Ganai, L. Zhang, P. Ashar, A. Gupta, Malik S. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In: Proc. ACM/IEEE Design Automation Conference, 2002, pp. 747–750.

[Kuehlmann, 2002] A. Kuehlmann, A.J. van Eijk Cornelis: Combinational and Sequential Equivalence Checking. In: Logic synthesis and Verification. Ed. S.Hassoun, T.Sasao and R.K.Brayton. Kluwer Academic Publishers, 2002, pp. 343–372.

[Kunz, 2002] W. Kunz, J. Marques-Silva, S. Malik. SAT and ATPG: Algorithms for Boolean Decision Problems. In: Logic synthesis and Verification. Ed. S.Hassoun, T.Sasao and R.K.Brayton. Kluwer Academic Publishers, 2002, pp. 309–341.

[Goldberg, 2002] E. Goldberg, E. Novikov. BerkMin: A fast and robust SAT-Solver. In: Proc. European Design and Test Conference, 2002, pp. 142–149.

[Cheremisinova, 2008] L. Cheremisinova, D. Novikov. Simulation-based approach to verification of logical descriptions with functional indeterminacy. In: Information Theories & Applications (IJ ITA), 2008, Vol. 15, No. 3, pp. 218–224.

Authors' Information

Liudmila Cheremisinova *– Principal Researcher, The United Institute of Informatics Problems of National Academy of Sciences of Belarus, Surganov str., 6, Minsk, 220012, Belarus, e-mail: cld@newman.bas-net.by*

Dmitry Novikov *– Post graduate student, The United Institute of Informatics Problems of National Academy of Sciences of Belarus, Surganov str., 6, Minsk, 220012, Belarus, e-mail: yakov_nov@tut.by*