SIMULATION-BASED APPROACH TO VERIFICATION OF LOGICAL DESCRIPTIONS WITH FUNCTIONAL INDETERMINACY

Liudmila Cheremisinova, Dmitry Novikov

***Abstract**: A verification task of proving the equivalence of two descriptions of the same device is examined for the case, when one of the descriptions is partially defined. In this case, the verification task is reduced to checking out whether logical descriptions are equivalent on the domain of the incompletely defined one. Simulation-based approach to solving this task for different vector forms of description representations is proposed. Fast Boolean computations over Boolean and ternary vectors having big sizes underlie the offered methods.*

***Keywords**: design automation, verification, Boolean computations, simulation.*

***ACM Classification Keywords**: B.6.2 [Logic Design]: Reliability and Testing; G.4 [Mathematical Software]: Verification; I.6.8 [Simulation and Modeling]: Types of Simulation – Parallel*

Introduction

In a typical design project more than half of the efforts go not into designing but into verifying that the design is correct. In this process the design is checked against the specification to ensure that every requirement of the specification is satisfied by its implementation. The objective of formal verification is to prove behaviour equivalence of two descriptions representing different design stages of the same device. The task is well studied, if both descriptions are completely specified and are given as structural representations. This case is reduced to checking on whether two combinational circuits are equivalent. Efficient methods for equivalence checking were proposed (see [1 – 4] for example).

In the paper, the verification task is examined for a more general case, when one (initial) of logical descriptions is not completely specified. The case usually occurs on early stages of designing complex devices 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. The initial logic description of a verifiable combinational device is considered as a system of partially defined Boolean functions.

We are focusing on a case, when the combinational structure obtained in the process of device decomposition is a multi-block structure. Each block of the structure is described by a system of completely defined Boolean functions. The task of proving equivalence of two logical descriptions is transformed for this case into a task of checking out whether the resulting description which is defined functional completely is an extension of the initial one having functional indeterminacy; i.e. equivalence of logical descriptions is tested on the domain of the initial system of partially defined Boolean functions.

Different vector forms of representation of partially defined Boolean functions are considered, such as tuples and intervals of the Boolean space. Simulation-based approach to solving the verification task for different vector forms of description representations is proposed. The offered methods focusing on the mentioned forms of representation of Boolean functions are based on fast Boolean computations over Boolean and ternary vectors having big sizes.

Representation of a system of partially defined Boolean functions

In the paper we consider the case, when the first of the descriptions is a system $F = \{f_1(X), f_2(X), \ldots, f_m(X)\}$ ($X = \{x_1, x_2, \ldots, x_n\}$) of partially defined Boolean functions. Two cases of representing the system F are considered below:

1) on tuples (or patterns) of values of Boolean variables from X, i.e. complete Boolean assignments to the variables of X;

2) on intervals (or cubes) of values of Boolean variables from X, i.e. incomplete Boolean assignments to the variables of X.

In the first case, a partially defined Boolean function $f_i(X) \in F$ is specified by a pair of sets M_{fi}^1 and M_{fi}^0 that are on and off sets of the function f_i and consist of n-tuples $(b_1, b_2,\ldots, b_n) \in B^n$ ($b_i \in B = \{0,1\}$) of values of Boolean variables from $X = \{x_1, x_2, \ldots, x_n\}$ on which the function takes values 1 and 0 correspondingly. The value of the function is not defined (takes don't care value "–") on all other n-tuples that are neither in M_{fi}^1 nor in M_{fi}^0. In this paper, a case is considered when the system F is weakly defined, i.e. the cardinality of its domain $M = \bigcup_{i=1}^{m}(M_{f_i}^1 \cup M_{f_i}^0)$ is greatly less the cardinality of the whole Boolean space B^n, i.e. $|M| << 2^n$.

Such a system F can be represented by a pair of matrices $\boldsymbol{B}$ and $\boldsymbol{T}$ (Fig. 1,a) having the same number l of rows as the cardinality of the set $M = \bigcup_{i=1}^{m}(M_{fi}^1 \cup M_{fi}^0)$. The Boolean matrix $\boldsymbol{B}$ contains as its rows all n-tuples from the set M and the ternary matrix $\boldsymbol{T}$ specifies the values of the functions on these n-tuples. A value given on the cross of the i-th column and the j-th row of the matrix $\boldsymbol{T}$ is used to point out the value of the function $f_i(X)$ on the j-th tuple of M. The value is "1" or "0", if the function $f_i(X)$ is defined on the corresponding n-tuple, and it is "–" (don't care) otherwise. Matrices $\boldsymbol{B}$ and $\boldsymbol{T}$ have n and m columns respectively.

	x_1	x_2	x_3	x_4	x_5		f_1	f_2	f_3	
	0	1	0	1	0		–	1	0	*1*
	0	0	0	1	1		1	0	–	*2*
	1	1	0	0	0		1	–	0	*3*
B=	1	1	1	1	1	T=	–	0	1	*4*
	0	0	1	0	1		0	–	–	*5*
	1	0	0	1	1		0	1	1	*6*
	1	0	1	1	0		–	0	0	*7*
	1	0	0	0	1		0	–	1	*8*

a)

	x_1	x_2	x_3	x_4	x_5		f_1	f_2	f_3	
	0	1	0	–	–		0	–	–	*1*
	0	0	0	1	1		–	1	1	*2*
	1	1	–	0	0		1	–	0	*3*
U=	1	1	1	–	1	T=	1	0	–	*4*
	0	–	1	0	1		0	–	0	*5*
	1	0	0	1	1		1	1	–	*6*
	1	–	1	1	0		–	0	0	*7*
	1	0	0	0	–		1	1	–	*8*

b)

Fig. 1. Systems of partially defined Boolean functions represented: a) on tuples; b) on intervals

In the second case, a partially defined Boolean function $f_i(X)$ is specified by a pair of sets U_{fi}^1 and U_{fi}^0 of intervals that represent on and off sets of the function correspondingly. An interval generally specifies more than one tuple. More precisely an interval of the rank k consists of a set of 2^{n-k} n-tuples, taking into account that the rank of an interval is the number of the variables of X, having values 0 or 1. An interval represents a subcube in the Boolean space B^n and fixes the values of only k variables. The system F can be represented in these case by a pair of ternary matrices $\boldsymbol{U}$ and $\boldsymbol{T}$ of the same cardinality (Fig. 1,b): the matrix $\boldsymbol{U}$ contains as its rows intervals of the set $U = \bigcup_{i=1}^{m}(U_{fi}^1 \cup U_{fi}^0)$ and the matrix $\boldsymbol{T}$ specifies the values of the functions of F on the intervals of U as follows. A value given on the cross of the i-th column and the j-th row of the matrix $\boldsymbol{T}$ is used to point out the value of the function $f_i(X)$ on the j-th interval of U. The value is "1" or "0", if the function is defined on all the n-tuples of the j-th interval, and it is don't care otherwise.

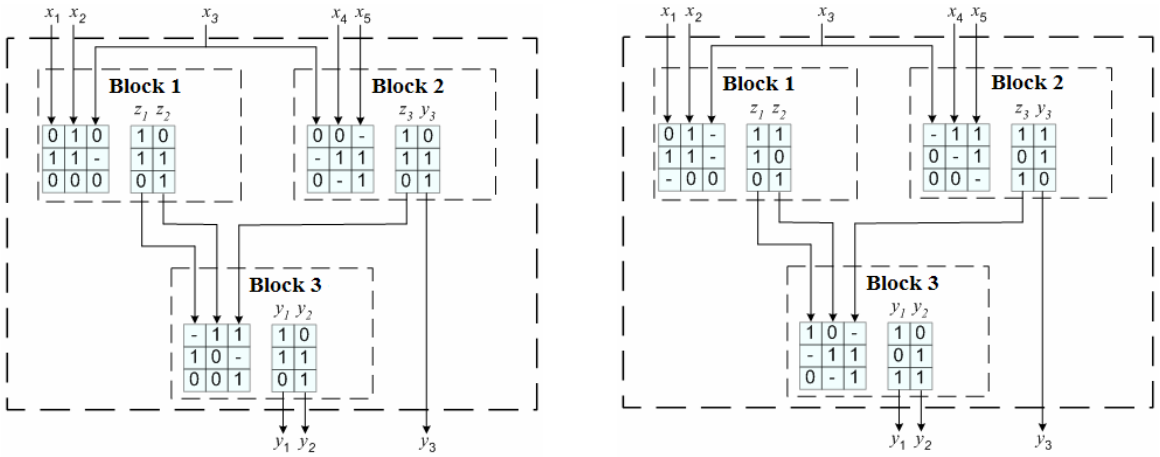
Representation of a function by the interval form has the following distinctive features. Intervals $u_i, u_j \in U$ can intersect each other (in contrast to a representation by the tuple form). The value "–" of an element t_i^j of the matrix $\boldsymbol{T}$ means that either the value of the function f_j can be not specified (is don't care) on the whole interval u_i or the function f_j does not take the same definite value (1 or 0) on the whole interval u_i, i.e. there exist at least two n-tuples belonging to the interval u_i on which the function f_j has different values from the set {1, 0, –}. Thus, the value "–" of the component t_i^j of the matrix $\boldsymbol{T}$ does not always mean that the value of the function f_j is not specified in the whole interval u_i (from such a viewpoint, that the value of f_j can be specified arbitrarily when it is implemented at the design step). So don't care value of the function f_j points only out that it can take different values inside of the interval u_i.

Transforming a representation of a multi-block structure

Each block of a multi-block structure S is a multi-output one (as in Fig. 1, 2) and is specified by a system of disjunctive normal forms (DNFs). The system of DNFs is represented by a pair of matrices: ternary and Boolean. The rows of the first one specify elementary conjunctions. The component on the cross of the i-th row and the j-th column of the Boolean matrix is equal 1, if the j-th DNF includes the i-th conjunction. The set of primary input variables of the structure S is the same as the set X of arguments of the system F. Each of m functions $y_i(X)$ implemented by the structure S must be an extension of the corresponding function $f_i(X)$ from F. A structure S implements a system F of partially defined functions if and only if for each $f_i(X) \in F$ and the corresponding $y_i(X)$ that is the i-th output of the structure S the next two relations hold:

$$M_{fi}^1 \subseteq M_{yi}^1 \text{ and } M_{fi}^0 \subseteq M_{yi}^0, \quad (1)$$

i. e. the values of f_i and y_i must be the same in the domain $M_{f_i}^1 \cup M_{f_i}^0$ of the function $f_i(X)$.



a) b)

Fig. 2. A three-block structures implementing systems of Boolean functions shown: a) in Fig. 1,a; b) in Fig. 1,b

Each block realizing a system of disjunctive normal forms can be considered as a three-level multi-output combinational circuit. Its first level consists of invertors which are used to invert those primary input variables of the block, which are represented in the inverse form at least in one conjunction. The second level is formed by multi-input AND gates implementing conjunctions, and the last (third) level is composed by multi-input OR gates. AND gates are specified in a natural way by the intervals from M. For example the interval 1 –0 1 – 0 corresponds to the conjunction $x_1\ \bar{x}_3\ x_4\ \bar{x}_6$ implemented by 4-input AND gate and two invertors fed upon variables x_3 and x_6.

Let us number gates inside of blocks of the multi-block structure and blocks themselves in a topological order, i.e. in such a way that any connection between gates will connect an output of a gate having a smaller number to an input of a gate having a bigger number. It should be pointed out that the necessary and sufficient condition for making the ordering is absence of feedbacks in the structure. The condition holds for each intrablock structure, and we assume that the structure itself satisfies the condition.

As the result the multi-block structure can be considered as a multi-output combinational circuit C which consists of invertors, AND and OR gates. Let us assign internal variables z_i to outputs of all the elements of C, marking out those of them that are used to implement m Boolean functions $y_j(X)$ (primary outputs). A method proposed below can be extended for nets consisting of any other elements implementing symmetric logic operations $\varphi(z_1, z_2, \ldots, z_k)$.

Solving the verification task for the tuple based representation

An idea of the proposed method for checking out whether a system F of partially defined Boolean functions is implemented by a multi-block structure S is to simulate a combinatorial circuit C corresponding to the structure on the domain of the system F. The system F is implemented by the structure S, if for each n-tuple $\boldsymbol{b}_j \in \boldsymbol{B}$ of values assigned to the variables from X the equality $f_i(\boldsymbol{b}_j) = y_i(\boldsymbol{b}_j)$ holds for all i for which the value $f_i(\boldsymbol{b}_j)$ is defined (i.e. is

equal to 0 or 1). In other words, the vector $\boldsymbol{f}(\boldsymbol{b}_j)$ being the j-th row of the matrix $\boldsymbol{T}$ must cover the vector $\boldsymbol{y}(\boldsymbol{b}_j)$ consisting of values generated by primary outputs of the circuit C under assignment $\boldsymbol{b}_j$ to its primary inputs. A ternary (or binary) vector $\boldsymbol{a}$ is covered by a ternary vector $\boldsymbol{b}$ of the same size, if for all components of $\boldsymbol{b}^j$ which are equal to 1 or 0 the equality $\boldsymbol{b}^j = \boldsymbol{a}^j$ holds.

Thus according to condition (1) we are interested in values of functions y_i on n-tuples $\boldsymbol{b}_j$ from the set M represented by rows of the matrix $\boldsymbol{B}$.

An idea of parallel binary simulation [5] is used. The combinational circuit C is simulated under all possible inputs simultaneously, i.e. all tuples from the set M are examined at the same time. When parallel simulation of the circuit is performed under all l n-tuples from M at the same time, a state of each node of the circuit (corresponding to a primary input or output of a gate) is represented by a Boolean vector. The latter has the size l and specifies the values of the same variable in all l tuples.

Thus, each Boolean vector represents states of a corresponding node for all l considered assignments to primary inputs of the simulated circuit, and the union of the i-th components of all the vectors describes the state of all nodes of the circuit for the i-th assignment to primary inputs.

At the beginning of the simulation, the ordered set of n Boolean vectors having the size l is taken; they correspond to the columns of the matrix $\boldsymbol{B}$. Then gates of the circuit C are simulated in the predefined topological order. Let a gate of the circuit implement a function $\varphi_i(z_{1i}, z_{2i}, \ldots, z_{ki})$. As each argument z_{ji} is related to a Boolean vector $\boldsymbol{z}_{ji}$ having been computed already, the simulation is reduced to computation of the operation φ_i over Boolean vectors $\boldsymbol{z}_{1i}, \boldsymbol{z}_{2i}, \ldots, \boldsymbol{z}_{ki}$ in the bitwise style. The result of the simulation is a new Boolean vector $\boldsymbol{z}_i$ of the same size l.

As soon as the last gate of the circuit has been simulated, value assignments to the primary outputs of the circuit C for all assignments to the primary inputs belonging to the domain M of the system F are found. At that each output function y_i of the circuit has a definite value (0 or 1) for each assignment used to the primary inputs, in particular for those assignments, for which corresponding function $f_i \in F$ has a definite value as well.

Thus, we need to check the orthogonality of the values of the functions y_i and f_i on the domain $M = M_{fi}^1 \cup M_{fi}^0$. This is reduced to checking out whether the following pairs of vectors are orthogonal: the ternary vector $\boldsymbol{t}^i$ corresponding to the i-th column of the matrix $\boldsymbol{T}$ and the Boolean vector $\boldsymbol{z}_{ip}$ corresponding to the primary output y_i of the circuit. The multi-block structure S implements the system F, if all these pairs of the vectors are not orthogonal. Otherwise, the block responsible for violating the implementing condition can be found by back traversal of the combinational circuit C.

To demonstrate the method proposed let us check out that the system of partially defined Boolean functions shown in Fig. 1,a is implemented by the three-block structure depicted in Fig. 2,a. A sequence of Boolean vectors generated by parallel simulation of the circuit C simultaneously for the whole domain of the system F is shown below. Boolean vectors representing values of circuit nodes are accompanied with operations performed over the vectors corresponding to their arguments. Here the variable corresponding to the i-th output of AND gate of j-th block of the structure is denoted by $k_{i,j}$ and Boolean vectors representing states of primary outputs of the structure C are printed in bold.

Inputs.	0 0 1 1 0 1 1 1	x_1
	1 0 1 1 0 0 0 0	x_2
	0 0 0 1 1 0 1 0	x_3
	1 1 0 1 0 1 1 0	x_4
	0 1 0 1 1 1 0 1	x_5
Block 1.	1 1 0 0 1 0 0 0	x_1
	0 1 0 0 1 1 1 1	x_2
	1 1 1 0 0 1 0 1	x_3
	1 0 0 0 0 0 0 0	$k_{1,1} = x_1 \wedge x_2 \wedge \overline{x_3}$
	0 0 1 1 0 0 0 0	$k_{2,1} = x_1 \wedge x_2$
	0 1 0 0 0 0 0 0	$k_{3,1} = x_1 \wedge \overline{x_2} \wedge \overline{x_3}$
	1 0 1 1 0 0 0 0	$z_1 = k_{1,1} \vee k_{2,1}$
	0 1 1 1 0 0 0 0	$z_2 = k_{2,1} \vee k_{3,1}$

Block 2.	0 0 1 0 1 0 0 1	x_4
	0 0 1 0 0 0 0 1	$k_{1,2} = \overline{x_3} \wedge \overline{x_4}$
	0 1 0 1 0 1 0 0	$k_{2,2} = x_4 \wedge x_5$
	0 1 0 0 0 1 0 1	$k_{3,2} = \overline{x_3} \wedge x_5$
	0 1 1 1 0 1 0 1	$z_3 = k_{1,2} \vee k_{2,2}$
	0 1 0 1 0 1 0 1	$\boldsymbol{y}_3 = k_{2,1} \vee k_{3,1}$
Block 3.	0 1 0 0 1 1 1 1	z_1
	1 0 0 0 1 1 1 1	z_2
	0 1 1 1 0 0 0 0	$k_{1,3} = z_2 \wedge z_3$
	1 0 0 0 0 0 0 0	$k_{2,3} = z_1 \wedge z_2$
	0 0 0 0 0 1 0 1	$k_{3,3} = \overline{z_1} \wedge z_2 \wedge z_3$
	1 1 1 1 0 0 0 0	$\boldsymbol{y}_1 = k_{1,3} \vee k_{2,3}$
	1 0 0 0 0 1 0 1	$\boldsymbol{y}_2 = k_{2,3} \vee k_{3,3}$

When comparing pairs, including ternary vector representing values of functions $f_i \in F$ and Boolean vector derived under simulation for the corresponding primary output y_i, we see that for all pairs the second one is covered by the first:

f_1: – 1 1 – 0 0 – 0	f_2: 1 0 – 0 – 1 0 –	f_3: 0 – 0 1 – 1 0 1
y_1: 1 1 1 1 0 0 0 0	y_2: 1 0 0 0 0 1 0 1	y_3: 0 1 0 1 0 1 0 1

The example demonstrates as the considered task is reduced to Boolean computations over vectors (sequences of bits) having the same (but arbitrary) size.

Solving the verification task for the interval based representation

In this case, the verification task can be solved by one of the following ways: 1) by unfolding intervals of the domain of the system into sets of tuples, i.e. by reducing the task to the case considered above (when a system of partially defined Boolean functions is in a tuple based form); 2) by solving the task directly by using the interval based representation. The first way can be used, when the number of intervals of the set M having ranks less than n is not big and these ranks are close to n. In this case, an n-tuple based representation of the domain of a given system of partially defined Boolean functions will be not much bigger than an interval based representation given. The second way can be used, when the number of intervals of the set M having ranks less than n is big and/or these ranks are much less than n. In this case, unfolding intervals could be impossible at practice. Further we discuss this second way.

As well as earlier we will perform parallel simulation of the circuit C, in the discussed case the simulation will be carried out on intervals given as rows of the matrix $\boldsymbol{U}$. Now during circuit simulation, a state of every node (including those corresponding to primary inputs) of the circuit is represented by a ternary vector. Thus, each ternary vector represents states of the corresponding node for all l considered partial assignments to primary inputs (specified by intervals from M) of the simulated circuit, and the union of the i-th components of all the state vectors describes the state of all nodes of the circuit for the i-th partial assignment to primary inputs (defined by the i-th interval of M). In this case, the don't care value of the i-th component of a state vector of the j-th node means only that a function implemented by the node can have different values for different n-tuples of the i-th interval of M.

At the beginning of the simulation, the ordered set of n of ternary vectors having the size l is taken. The vectors represent states of n primary inputs and correspond to the columns of the matrix $\boldsymbol{U}$. Then gates of the circuit C are simulated in the predefined topological order. Let a gate implementing the function $\varphi_i(z_{1i}, z_{2i}, \ldots, z_{ki})$ is simulated. As for each its argument z_{ji} a ternary vector $\boldsymbol{z}_{ji}$ having been computed already corresponds to, the simulation of the gate is reduced to performing the logic operation φ_i over ternary vectors $\boldsymbol{z}_{1i}, \boldsymbol{z}_{2i}, \ldots, \boldsymbol{z}_{ki}$ in the bitwise style. The result of the simulation is a new ternary vector $\boldsymbol{z}_i$ of the same size l. A definition of basic operations over ternary variables and vectors is given bellow (all possible combinations of two ternary values are considered and don't care is interpreted as uncertainty):

$\boldsymbol{a}$:	0 0 0 – – – 1 1 1
$\boldsymbol{b}$:	0 – 1 0 – 1 0 – 1
$\bar{\boldsymbol{a}}$:	1 1 1 – – – 0 0 0
$\boldsymbol{a} \vee \boldsymbol{b}$:	0 – 1 – – 1 1 1 1
$\boldsymbol{a} \wedge \boldsymbol{b}$:	0 0 0 0 – – 0 – 1

After the simulation process is over, it is necessary to check whether the system F of functions is implemented by the multi-block structure S. This checking can be time consuming. As soon as the last gate of the circuit has been simulated the following pairs of vectors are compared: the ternary vector $\boldsymbol{t}^p$ (p = 1, 2,…, m) corresponding to the p-th column of the matrix $\boldsymbol{T}$ (the column of values of the function $f_p \in F$) and the ternary vector $\boldsymbol{z}_{ip}$ corresponding to the primary output y_p of the circuit C. The following three cases are possible:

1. Vectors $\boldsymbol{t}^p$ and $\boldsymbol{z}_{ip}$ are orthogonal in some component. Hence, the circuit C does not implement the function f_p.

2. The vector $\boldsymbol{t}^p$ covers the vector $\boldsymbol{z}_{ip}$, i.e. values of all definite (1 or 0) components of the vector $\boldsymbol{t}^p$ are the same as the values of the corresponding components of $\boldsymbol{z}_{ip}$. In this case, the circuit C implements the function f_p.

3. The value of some j-th component (corresponding to the interval $\boldsymbol{u}_j$ of the matrix $\boldsymbol{U}$) of the vector $\boldsymbol{z}_{ip}$ is don't care, while the value of the corresponding component of the vector $\boldsymbol{t}^p$ is equal 1 or 0. In this case, there exists no unambiguous answer whether the circuit C implements the function f_p or does not.

In the third case, an additional analysis is needed to detect the reason of distinction of the values of the j-th components of the vectors $\boldsymbol{z}_{ip}$ and $\boldsymbol{t}^p$. The simplest way is to simulate the circuit C once more on all tuples of the interval $\boldsymbol{u}_j$ of the matrix $\boldsymbol{U}$. A more refined method is based on-the-fly analysis of the simulation process. If the output z_q of a gate of the circuit C gets don't care value, the interval corresponding to the value assignment to the inputs of the gate is to be decomposed into subintervals, in which the output z_q gets a definite value. This will help us to prevent propagation of indeterminacy during the simulation. For example, if $z_q = x_1 x_3 x_4 x_6$, and computations are to be performed for the interval u_i = 1 0 – 1 – –, resulting in $z_q(u_i)$ = –, then the interval is decomposed into three following ones: u_i^1 = 1 0 1 1 – 1, : u_i^2 = 1 0 0 1 – –, u_i^3 = 1 0 – 1 – 0. For these subintervals, z_q takes definite values: $z_q(u_i^1) = 1$, $z_q(u_i^2) = z_q(u_i^3) = 0$.

To demonstrate the method proposed let us check out that the system shown in Fig. 1,b of partially defined Boolean functions is implemented by a three-block structure depicted in Fig. 2,b. A sequence of ternary vectors generated by parallel simulation of the circuit C for intervals of the matrix $\boldsymbol{U}$ is shown below.

Inputs.	0 0 1 1 0 1 1 1	x_1
	1 0 1 1 – 0 – 0	x_2
	0 0 – 1 1 0 1 0	x_3
	– 1 0 – 0 1 1 0	x_4
	– 1 0 1 1 1 0 –	x_5
Block 1.	1 1 0 0 1 0 0 0	x_1
	0 1 0 0 – 1 – 1	x_2
	1 1 – 0 0 1 0 1	x_3
	1 0 0 0 – 0 0 0	$k_{1,1} = \bar{x}_1 \wedge x_2$
	0 0 1 1 0 0 – 0	$k_{2,1} = x_1 \wedge x_2$
	0 1 0 0 0 1 0 1	$k_{3,1} = \bar{x}_2 \wedge \bar{x}_3$
	1 0 1 1 – 0 – 0	$z_1 = k_{1,1} \vee k_{2,1}$
	1 1 0 0 – 1 0 1	$z_2 = k_{1,1} \vee k_{3,1}$
Block 2.	– 0 1 – 1 0 0 1	x_4
	– 1 0 – 0 1 0 0	$k_{1,2} = x_4 \wedge x_5$
	– 1 0 0 0 1 0 –	$k_{2,2} = \bar{x}_3 \wedge x_5$
	– 0 – 0 0 0 0 1	$k_{3,2} = \bar{x}_3 \wedge \bar{x}_4$
	– 1 – – 0 1 0 1	$z_3 = k_{1,2} \vee k_{3,2}$
	– 1 0 – 0 1 0 –	$\boldsymbol{y}_3 = k_{1,2} \vee k_{2,2}$
Block 3.	0 1 0 0 – 1 – 1	z_1
	0 0 1 1 – 0 1 0	z_2
	0 0 1 1 – 0 – 0	$k_{1,3} = z_1 \wedge \bar{z}_2$
	– 1 0 0 0 1 0 1	$k_{2,3} = z_2 \wedge z_3$
	0 1 0 0 0 1 0 1	$k_{3,3} = \bar{z}_1 \wedge z_3$
	0 1 1 1 – 1 – 1	$\boldsymbol{y}_1 = k_{1,3} \vee k_{3,3}$
	– 1 0 0 0 1 0 1	$\boldsymbol{y}_2 = k_{2,3} \vee k_{3,3}$

When comparing pairs, each of which includes ternary vector representing value of a function $f_i \in F$ and Boolean vector derived under simulation for the corresponding primary output y_i, we see that for all pairs but one the second of the vectors is covered by the first one:

f_1: 0 – 1 1 0 1 – 1	f_2: – 1 – 0 – 1 0 1	f_3: – 1 0 – 0 – 0 –
y_1: 0 1 1 1 – 1 – 1	y_2: – 1 0 0 0 1 0 1	y_3: – 1 0 – 0 1 0 –

The only exception is in the 5-th component of f_1 for which the case 3 takes place. By splitting the interval 0–101 into two tuples and simulating the structure on them we find out $y_1(00101) = y_2(01101) = 0$, i.e. y_1 implements f_1.

In order to avoid unnecessary decompositions of intervals during simulation, it makes sense to orthogonalize the initial system of partially defined Boolean functions. A system is orthogonalized, if any interval u_i of the domain M has the following property: every function of the system F has the same value (1, 0, –) for all tuples forming u_i . In this case, the value "–" of a function means that the value of this function is undefined for all tuples of the interval and may be arbitrarily redefined during implementing the system by circuit.

Orthogonalization of a system of partially defined Boolean functions

In [6, 7], an orthogonalization task is examined for a system of completely defined Boolean functions $F = \{f_1(X), f_2(X), \ldots, f_m(X)\}$ which is represented by a pair of matrices: ternary and Boolean. Rows of the ternary matrix represent intervals. If an element in the Boolean matrix takes the value 1, then the corresponding function takes the value 1 for the corresponding interval. The task is to find a set of pair-wisely orthogonal completely defined Boolean functions $\varphi_1(X), \varphi_2(X), \ldots, \varphi_r(X)$ such that any function $f_i \in F$ can be represented as disjunction of some pair-wisely orthogonal functions φ_j (j = 1, 2, …, r), and the number r of different functions φ_i must be minimal. Functions φ_j and φ_k are orthogonal, if the condition $\varphi_j \wedge \varphi_k = 0$ holds for any value assignment to variables of X. At that, if functions φ_i are all reciprocally orthogonal, the intervals for which different functions φ_j and φ_k are defined don't intersect each other, however intervals for which the same function is defined may intersect each other.

In a similar manner as for systems of completely defined Boolean functions the orthogonalization task can be formulated for a system of partially defined Boolean functions $F = \{f_1(X), f_2(X), \ldots, f_m(X)\}$, when each $f_i \in F$ is represented by a pair of functions f_i^1 и f_i^0. The on set of the function f_i^1 is the union of intervals for which f_i takes the value 1. The on set of the function f_i^0 is the union of intervals for which f_i takes value 0. In the parts of the Boolean space that are complementary to on and off sets, the functions f_i^1 и f_i^0 take the value 0. In this way, we come to the system F' of completely defined Boolean functions having twice more functions than the initial system F of partially defined Boolean functions. Now, the orthogonalization task can be solved for F' by some method from [6,7], and then a backward transition of F' into the orthogonalized system F can be performed.

A distinguishing feature of an orthogonalized system of partially defined Boolean functions is that any two intervals for which a function of the system takes different values are not intersect. In this case, the don't care value of a function of the system for any interval means that the function is not defined on all tuples of the interval, and a definite value of the function can be arbitrary assigned under a synthesis procedure.

Conclusion

Methods proposed in the paper are focused on systems of weakly defined Boolean functions which have much smaller domains than Boolean spaces in which the systems are specified. The methods can be used for verifying complex logical descriptions.

The task of checking out whether a multi-block structure implements a system of partially defined Boolean functions is reduced to Boolean computations over ternary and Boolean vectors having an arbitrary size. Methods can be implemented effectively by using classes CBV [8] и CTM [9] developed for performing operations over Boolean and ternary vectors of arbitrary sizes in C++. Computational complexity of the methods linearly depends on the total number of nodes of the simulated combinational circuit (more precisely, on the total number of inputs of all gates of the circuit) and on the number of bytes (or 32-bit words) used to represent an l-bit vector.

Bibliography

[1] Drechsler R. (Ed.). Advanced Formal Verification. – Kluwer Academic Publishers, 2004. – 260 p.

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

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

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

[5] Zakrevskii A.D., Pottosin U.V. Cheremisinova L.D. Foundations of logic design. Book 3. Design of logic control devices. – Minsk: National Academy of Sciences of Belarus, United Institute of Informatics Problems of National Academy of Sciences of Belarus, 2006. – 252 p. (in Russian).

[6] Kuznetzov O.P. Orthogonal systems of Boolean functions and their application to analysis and synthesis of logical nets // Automation and remote control, 1970. – N 10, – P. 117–128 (in Russian).

[7] Pottosin U.V., Shestakov E.A. Orthogonalization of systems of completely defined Boolean functions // Logic design. – Minsk: The Institute of Engineering Cybernetics of National Academy of Sciences of Belarus. – 2000. – Vol. 5. – P. 107–115 (in Russian).

[8] Vasilkova I.V., Romanov V.I. Boolean vectors and matrices in C++ // Logic design. – Minsk: The Institute of Engineering Cybernetics of National Academy of Sciences of Belarus. – 1997. – Vol. 2. – P. 150–158 (in Russian).

[9] Cheremisionov D.I., Cheremisinova L.D. Ternary vectors and matrices in C++ // Logic design. – Minsk: The Institute of Engineering Cybernetics of National Academy of Sciences of Belarus. – 1998. – Vol. 3. – P. 146–156 (in Russian).

Authors' Information

Liudmila Cheremisinova *– Doctor of Sciences, Principal researcher, The United Institute of Informatics Problems of National Academy of Sciences of Belarus, Surganov St., 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 St., 6, Minsk–220012, Belarus; e-mail: yakov_nov@tut.by*