SEARCH ALGORITHM FOR SHORTEST SYNCHRONIZING SEQUENCES USING BOOLEAN SATISFIABILITY

Liudmila Cheremisinova

***Abstract**: The problem under consideration is to find a synchronizing sequence for a logic network with memory. A novel method is proposed that is based on formulation of the task as the Boolean satisfiability problem solved with any standard SAT solver. The developed method allows creating a Boolean equation presenting the problem in conjunctive normal form.*

***Keywords**: design automation, verification, testing.*

***ACM Classification Keywords**: B.6.2 [Logic Design]: Reliability and Testing; B.6.2 [Reliability and Testing]: Error-checking.*

Introduction

Advances in manufacturing process technology leads to very complex designs. But with the increasing complexity of integrated circuits the problem of ensuring that no faults exist which can cause the device to malfunction is on the rise. The problem of testing VLSI circuits with great design complexity is the dominating validation activity in industry today, its cost is quickly increasing. The necessity to improve the process of testing is urgent today.

The traditional method for testability verification is based, as a rule, on gatelevel simulation. A subset of the functional test patterns (sequences of input vectors) is applied to integrated circuits on the manufacturing testers. Those design parts that yield the expected values for the applied vectors are said to pass and are decided as good ones; those that do not pass, are said to be bad.

Purely combinational logic can be modeled with two states (0 and 1). The sequential circuit having memory can have internal states that make sequential circuit testing more complex than that of the combinational logic. That is because in typical case the state of internal memory is not known at the beginning of the simulation to test. This assumption is also a good representation of the reality, because when a circuit is powered up, its flip-flops can be in any of the two possible states. In order to begin execution of any test sequence it is necessary to bring a sequential circuit under test to some identified state from which it is known how to proceed. So we must to initialize the internal memory to some identified state. And only after the initialization it is possible to activate the faults and to test circuit. As input sequences that solve the problem of initialization homing and synchronizing sequences are used.

Test sequence (the sequence of input vectors) that brings a sequential circuit to some known state regardless of its initial state is the synchronizing sequence. A homing sequence is such an input sequence that the corresponding output sequence uniquely determines its final state regardless of its initial state. After applying a synchronizing sequence, the final state of the circuit is known without observing the outputs. So, every synchronizing sequence is also a homing sequence, but not conversely.

These two sequences play an important role in the testing of finite state systems and have been used in a number of applications such as hardware fault detection, protocol verification, learning algorithms, etc. In the past and existing literature the problem of homing and synchronizing sequences generation is usually considered for the case of finite state machines (FSM). Motivated mainly by automata theory, the problem was heavily studied many years ago [Kohavi, 1978], [Gill, 1962]. In most applications the underlying FSM is an automaton with abstract state, whose functionality is described by a transition and an output tables (or by state transition graph). Several approaches are used to obtain a synchronizing and homing sequences for a FSM. A survey of the main methods can be found in [Lee, 1996]. It should note that finding a shortest homing or synchronizing sequence (with minimum legth) is an NP-hard problem [Eppstein, 1990]. Further we will consider the problem of obtaining synchronizing sequence for sequential logic circuits.

Recent advances in solving Boolean satisfiability problems caused a significant resurgence of the application of satisfiability solvers (SAT-solvers) in different electronic design automation domains. In the last years, great improvements were achieved in both the speed and capacity of SAT-solvers [Eén], [Mahajan, 2005], [Goldberg, 2002], which are now very fast and can handle huge problems. The new efficient SAT-solvers open new possibilities for applying this technology by translating hard design problems to equivalent SAT problems. So the existence of effective SAT solvers makes it attractive to translate looking-for synchronizing sequence problem into Boolean problem solvable by SAT solvers. SAT solvers normally operate on Boolean formulas in Conjunctive Normal Form (CNF), so the method is proposed that allows to create a Boolean function presenting problem of search for synchronizing sequence in CNF form.

The case considered here concerns to synchronous logical circuits having flip-flop primitives of type D as memory elements. For such a case a method of finding synchronizing sequence is proposed, the method allows creating a Boolean equation presenting the problem of search for synchronizing sequence in form of CNF.

The problem statement

The proposed method works on a synchronous circuit that consists of combinational logic and flip-flops, and is often represented in the form of two blocks. The first block is purely combinational, some its outputs feed a set of flip-flops, which in turn control some inputs of the combinational block. So the combinational block has two types of inputs: external inputs known as primary inputs and internal inputs, they present the internal state of the circuit and are supplied by the flip-flops. Similarly, the combinational block has two types of outputs: externally observable and known as primary outputs, and internal outputs, they present excitation functions for flip-flops.

The combinational block is modeled as an interconnect of primitive gates such as AND, OR, NOT, NAND, NOR, XOR, and in addition to them sometimes complex gates such as blocks implementing combinational circuits are also allowed.

The second block consists of register of the frequently used data flip-flops – D flip-flops. The example of such a logic circuit is shown in Figure 1.

The task is to be initialize memory elements to some reset states, that is, to find out the synchronizing sequence. Informally, a synchronizing sequence is a sequence of input sets that, when fed the sequential circuit, is guaranteed to bring it to some specified final state.

Let we have a circuit with n primary inputs and m D flip-flops. An input sequence, $\boldsymbol{X} = (\boldsymbol{x}^1, \boldsymbol{x}^2, \dots, \boldsymbol{x}^k)$ (where $\boldsymbol{x}^i = (x_1{}^i, x_2{}^i, \dots, x_n{}^i)$ is a vector of input signals that is fed the circuit in the time moment i) is said to be a synchronizing sequence of a sequential circuit, if the final circuit internal state after feeding it on the sequence can be determined uniquely regardless of the circuit initial state.

In general case there exists more than one synchronizing sequence for a finite state machine. We classify a synchronizing sequence $\boldsymbol{X}$ for a circuit as optimal if it is the shortest for all synchronizing sequences accepted by the circuit, that is $\boldsymbol{X}$ is of the shortest length k. It can be not alone too. The task is to find one of the shortest synchronizing sequences.

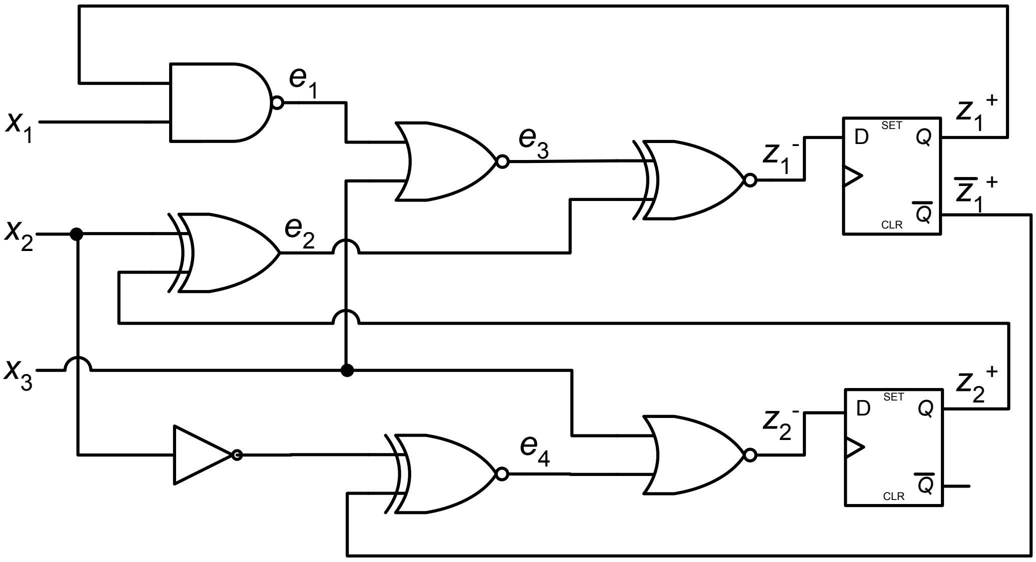

Figure 1. The example circuit under test

The behavior of the D flip-flop is described as the behavior of Moore's automaton with two states: 0 and 1. The symbol at its output coincides with the symbol of the state, in which the flip-flop is at the current time instant. Usually, the structure of an output symbol is given as two Boolean variables: Q and $\overline{Q}$ (Figure 1). The D flip-flop acts as a delay, i.e. at the next instant after the excitation signal has come it comes to the state corresponding to this signal. The search for known D flip-flop states is reduced to search for predefined values of excitation functions of the flip-flops. So further we are allowed to consider only combinational block having $n + m$ primary inputs $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+k}$ corresponding n primary inputs and k flip-flop outputs, and k primary outputs $y_1, y_2, \dots, y_k$ corresponding to flip-flop inputs defining their excitation functions. The circuit part defining primary outputs of ancestor sequential circuit is deleted as it is shown in Figure 2.

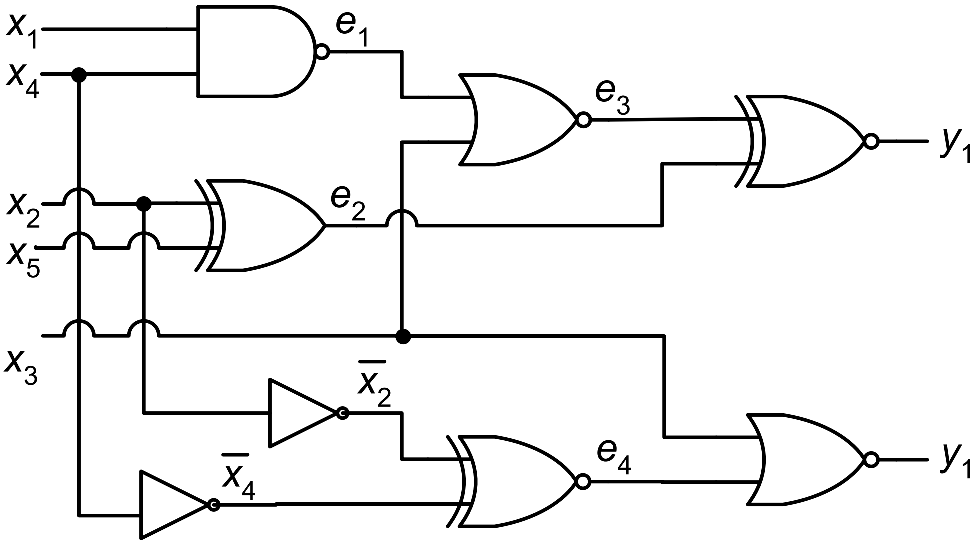

Figure 2. Combinational part of the sequential circuit shown in Figure 1

SAT-based approach to solving the problem of synchronizing sequence search

A CNF represents a Boolean function as conjunction of one or more clauses, each being in its turn a disjunction of literals (from now on literal is a Boolean variable or its inversion). A CNF denotes a unique completely specified Boolean function. 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 $\overline{x}_j$) the variable x_j appears or does not appear in *i*-th clause of *C*.

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 a truth assignment of literals which simultaneously satisfies each of CNF clauses. If such an assignment exists the CNF is referred to as satisfiable, and the assignment is known as a satisfying assignment.

A variable value assignment ***a*** (a set of *n* equalities of type $a_i = \sigma_i$, where $\sigma_i \in \{0, 1\}$) for the vector ***x*** can be complete if all x_i are assigned or partial otherwise. A complete variable value assignment represents a minterm and a partial assignment represents a cube that could be thought as a product of literals.

The topological description of a combinational circuit can be represented using a directed acyclic graph, where nodes correspond to the mentioned gates, primary inputs and outputs; edges correspond to circuit wires connecting the nodes. Incoming edges of a circuit node are called its fanins and outgoing edges are called fanouts. A node in the circuit is multiple fan-out if its output is a fan-in to different gates. The node and its output signal are named the same. Let us call the functionality of a circuit node in terms of its immediate fanins as the local function of the node. The functionality of the circuit in terms of its primary inputs is the system of global functions implemented on primary outputs.

Majority of SAT applications derived from circuit representation produce so called conventional CNF describing all combinations of signal values on all circuit terminals. The conventional CNF of a combinational circuit specifies all combinations of signal values of its terminals that can take place when it functions.

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. The procedure of derivation of conventional CNF is known, it associates a CNF formula with each circuit gate that captures the consistent assignments between gate primary inputs and output. All such gate local CNFs are joined then in the overall circuit conventional CNF by using the conjunction operation. CNF for a gate representing a local function $y = f(z_1, z_2, \ldots, z_k)$ is based on defining and representing in a CNF form a new Boolean function $\varphi(y, f) = y \sim f(z_1, z_2, \ldots, z_k)$ [Kunz, 2002] that is true in the only case when both functions y and $f(z_1, z_2, \ldots, z_k)$ assume the same value.

Here are the conventional CNF representations of NOT, 2EXOR, *n*AND, *n*OR functions and their inversions:

$$y = \bar{z} \rightarrow (z \vee y)(\bar{z} \vee \bar{y});$$

$$y = z_1 \oplus z_2 \rightarrow (z_1 \vee z_2 \vee \bar{y})(\bar{z}_1 \vee \bar{z}_2 \vee \bar{y})(z_1 \vee \bar{z}_2 \vee y)(\bar{z}_1 \vee z_2 \vee y);$$

$$y = z_1 \wedge z_2 \wedge \ldots \wedge z_n \rightarrow (z_1 \vee \bar{y})(z_2 \vee \bar{y}) \ldots (z_n \vee \bar{y})(\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee y);$$

$$y = z_1 \vee z_2 \vee \ldots \vee z_n \rightarrow (\bar{z}_1 \vee y)(\bar{z}_2 \vee y) \ldots (\bar{z}_n \vee y)(z_1 \vee z_2 \vee \ldots \vee z_n \vee \bar{y});$$

$$y = \overline{z_1 \oplus z_2} \rightarrow (z_1 \vee z_2 \vee y)(\bar{z}_1 \vee \bar{z}_2 \vee y)(z_1 \vee \bar{z}_2 \vee \bar{y})(\bar{z}_1 \vee z_2 \vee \bar{y});$$

$$y = \overline{z_1 z_2 \ldots z_n} \rightarrow (z_1 \vee y)(z_2 \vee y) \ldots (z_n \vee y)(\bar{z}_1 \vee \bar{z}_2 \vee \ldots \vee \bar{z}_n \vee \bar{y});$$

$$y = \overline{z_1 \vee z_2 \vee \ldots \vee z_n} \rightarrow (\bar{z}_1 \vee \bar{y})(\bar{z}_2 \vee \bar{y}) \ldots (\bar{z}_n \vee \bar{y})(z_1 \vee z_2 \vee \ldots \vee z_n \vee y).$$

It is possible to eliminate the output variable y of NOT gate and two appropriate clauses if to subsume it in its fan-out gates replacing all instances of y with the negated input variable x of this gate.

The obtained 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.

For example, four additional variables e_1, e_2, e_3 and e_4 were supplemented the circuit shown in Figure 2. The corresponding conventional CNF is shown at the second column in Table 1.

Given a conventional CNF formula the SAT problem may be restated as the problem of finding a variable value assignment that satisfies every clause, taking into account that a clause is satisfied if at least one its literal is equal to 1. Recall, a CNF formula is satisfiable if and only if there is a satisfying assignment of its literals which simultaneously satisfies each of its member clauses.

The method of searching for synchronizing sequence for a logic circuit via Boolean satisfiability

Boolean SAT formulations are binary in essence. Introduced Boolean variables represent solution alternatives, and Boolean formulas represent constraints imposes by the solved problem. All variable assignments satisfying Boolean formulas are equivalent when solving the satisfiability problem. During the search for SAT solution of the synchronizing sequence problem there is no cost mechanisms to favor one solution over another. Thus formulating SAT problem of searching the shortest synchronizing sequence, we are able only to get the answer whether some solution (of the predefined length) of our problem exists. That is why the problem of optimal synchronizing sequence finding is solved regarding a priori assigned synchronizing sequence length. The problem is formulated as a problem of Boolean satisfiability, deriving a Boolean function such that an assignment of variables that satisfy it (if it exists) defines a synchronizing sequence of the predefined size.

Thus, we are forced to reformate continuously the folding problem with increasing values of synchronizing sequence length until a satisfiable problem formulation arises. Such a reformulation of the problem based on enumeration of sequence length values seems cumbersome for logic circuits of great size, but below it will be shown that the process of alternate CNF building for increasing synchronizing sequence length is iterative.

At the beginning we search for a synchronizing sequence $\boldsymbol{X}^1 = (\boldsymbol{x}^1)$ of the length 1 and form conventional CNF C^1 for the combinational circuit under test assuming that its primary inputs corresponding to primary inputs of ancestor sequential circuit are $(x_1^1, x_2^1, \ldots, x_n^1)$ and primary inputs correspond to flip-flop outputs of ancestor sequential circuit are $(x_{n+1}^1, x_{n+2}^1, \ldots, x_{n+k}^1)$. The values of the last variables are accepted to be don't-care: x_{n+1}^1 = "–", x_{n+2}^1 = "–", …, x_{n+k}^1 = "–" because we don't know their initial values.

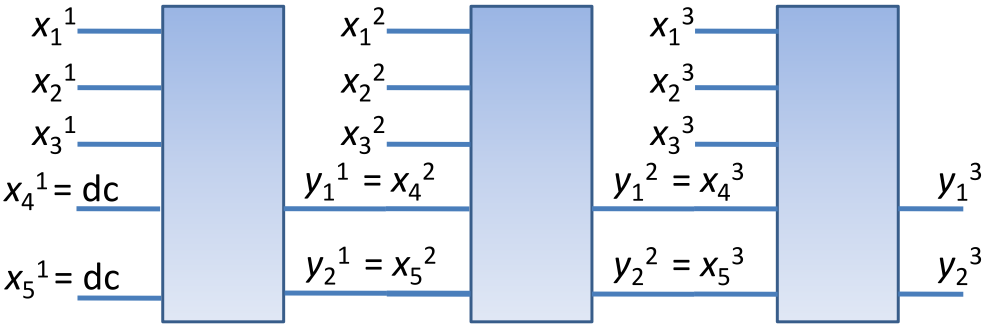

Figure 3. The process of augmentation of the number of blocks for the circuit of Figure 2

If such a manner formed CNF C^1 is satisfiable we will obtain synchronizing sequence $\boldsymbol{X}^1 = (x_1^1, x_2^1, \ldots, x_n^1)$ of unit length and the corresponding values of the circuit primary outputs $y_1^1, y_2^1, \ldots, y_k^1$ – defining values of excitation functions. Otherwise we should augment the experiment length and form the conventional CNF C^2 to search for a two cycle synchronizing sequence $\boldsymbol{X}^2 = (\boldsymbol{x}^1, \boldsymbol{x}^2)$. So two block combinational circuit will be considered: the first block is the circuit considered on the first step and the second one is a circuit identical to the first one but having primary inputs $(x_1^2, x_2^2, \ldots, x_n^2)$ and $x_{n+1}^2 = y_1^1$, $x_{n+2}^2 = y_2^1$, …, $x_{n+k}^2 = y_k^1$. Its k primary inputs $x_{n+1}^2, x_{n+2}^2, \ldots, x_{n+k}^2$ are connected with primary outputs $y_1^1, y_2^1, \ldots, y_k^1$ of the first block circuit, identifying the variables $y_1^1, y_2^1, \ldots, y_k^1$ and $x_{n+1}^2, x_{n+2}^2, \ldots, x_{n+k}^2$, so we use $x_{n+1}^2, x_{n+2}^2, \ldots, x_{n+k}^2$ instead of $y_1^1, y_2^1, \ldots, y_k^1$ have been introduced earlier. Then we again verify whether such a formed CNF C^2 is satisfiable to test whether there exists synchronizing sequence $\boldsymbol{X}^2 = (\boldsymbol{x}^1, \boldsymbol{x}^2)$ of the length 2 and so on as long (Figure 3) as we will obtain satisfiable CNF or the number of iterations exceeds the limit of iterations predefined in advance. In the last case our search fails and we don't find out any synchronizing sequence.

The peculiarities of search for synchronizing sequence via Boolean satisfiability

Here for illustration we consider the mentioned above example circuit (Figures 1 and 2) and find a synchronizing sequence for it. Taking in mind the proposed method we construct one-block circuit (Figure 4) and the conventional CNF C^1 corresponding to it (the second column of Table 1). For the convenience of its future augmentation (in the case if CNF C^1 will be unsatisfiable) we place the inner variables x_4^1, x_5^1 forward of the primary input variables x_1^1, x_2^1, x_3^1. CNF C^1 has 21 clauses. Before testing its satisfiability let substitute don't-care values for inner variables x_4^1, x_5^1 because their values are unknown and we don't entitled to assign them as distinct from values of primary input variables x_1^1, x_2^1, x_3^1. Replacement of the values will simplify the conventional

CNF C^1. For example, three clauses of CNF – $(x_1^1 \vee e_1^1)(x_4^1 \vee e_1^1)(\bar{x}_4^1 \vee \bar{x}_1^1 \vee \bar{e}_1^1)$ for NAND gate e_1^1 are replaced with $(x_1^1 \vee e_1^1)\, e_1^1\, (\bar{x}_1^1 \vee \bar{e}_1^1)$ or after simplification with only two simple clauses – e_1^1 and $\bar{x}_4^1$.

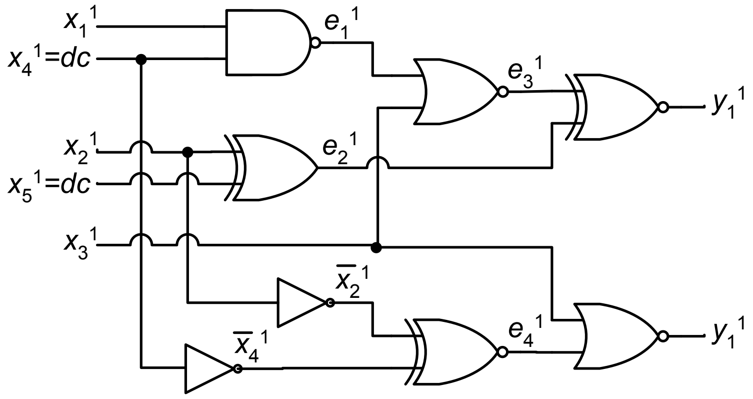

Figure 4. One-block combinational circuit for searching part of the sequential circuit shown in Figure 1

It should note that there exist some primitive gates that always have don't-care on its outputs when they have don't-care at least in one input. Gates realizing XOR and equivalency functions are such ones. For example, four clauses CNF for NXOR gate e_2^1 – $(x_5^1 \vee x_2^1 \vee e_2^1)(\bar{x}_5^1 \vee \bar{x}_2^1 \vee e_2^1)(x_5^1 \vee \bar{x}_2^1 \vee \bar{e}_1^1)(\bar{x}_5^1 \vee x_2^1 \vee \bar{e}_1^1)$ is replaced with CNF $(x_2^1 \vee e_2^1)(\bar{x}_2^1 \vee e_2^1)(\bar{x}_2^1 \vee \bar{e}_1^1)(x_2^1 \vee \bar{e}_1^1)$ that is always unsatisfiable. So, we may delete fragments associated with such gates from CNF C^1 taking the values of their output variables to be don't-care.

Simplified conventional CNF C^1_{sim} for our example circuit (Figure 4) is shown in the second column of the second row in Table 1. The CNF is unsatisfiable, so we have no synchronizing sequence of the unit length. More precisely in the case in question we cannot initialize the first D flip-flop in the only cycle, while the synchronizing sequence of the unit length for the second D flip-flop exists.

Table 1

Conventional conjunctive normal forms for example combinational circuits

	CNF C^1 for one-block circuit					CNF C^2 for two-block circuit						
	x_4^1	x_5^1	x_1^1	x_2^1		x_4^1	x_5^1	x_1^1	x_2^1	x_3^1	e_1^1	e_2^1
1		x_3^1	e_1^1	e_2^1			e_3^1	e_4^1	x_4^2	x_5^2	x_1^2	x_2^2
2		e_3^1	e_4^1	y_1^1			x_3^2	e_1^2	e_2^2	e_3^2	e_4^2	y_1^2
3		y_2^1					y_2^2					
4	–	–	1	–	–	–	–	–	–	–	1	–
5		1	–	–	–		–	–	–	–	–	–
6		–	–				–	–	–	–	–	–
7		e_1^1					–		e_1^1			
8	1	–	–	–	–	–	–	0	–	–	–	–
9		1	–	–	–		–	–	–	–	–	–
		–	–				–	–	–	–	–	–
10	0	–	0	–	–		–					
11		0	–	–	–	–	–	–	–	–	0	–
12		–	–				0	–	–	–	–	–
13	–	1	–	1	–		–	–	–	–	–	–
14		–	1	–	–		–		e_3^1			
15		–	–			–	–	–	–	0	–	–
16		e_2^1					0	–	–	–	–	–
17	–	0	–	0	–		–	–	–	–	–	–
18		–	1	–	–		–					
19		–	–			–	–	–	–	1	1	–
20	–	1	–	0	–		1	–	–	–	–	–
21		–	0	–	–		–	–	–	–	–	–

22 23 24 25 26 27 28	– – – 0 – 1 – – 0 – – – – – – – – – 0 – 0 – – – e_3^1 – – – – 0 – – 0 – – – – – – – 1 1 – 1 – – – 0 – – 0 – – – – 1 – – e_4^1 1 – – 1 – – – – 1 – – 0 – – 1 – – – – 0 – – 1 – – 0 – – – – 0 – – – – – – – – 1 1 – 1 – y_1^1 – – – – – – 0 0 – 1 – – – – – – – 1 0 – 0 – – – – – – – 0 1 – 0 – – – – – 0 – – – – – 0 y_2^1 – – – – – – – – 0 – 0 – – – – 1 – – – 1 – 1	– – – – – – – – – – – 0 – – – – – – – – – y_2^1 – – – – 1 – 1 – – 1 – – – – – e_1^2 – – – – – – – – – 1 – – – – 1 – – – – – – – – – – – – – – 0 – 0 – – 0 – – – – – – – – – – – – – – – 1 – 1 – – 1 – – – – e_2^2 – – – – – – – – – – 0 – 0 – – 1 – – – – – – – – – – – – – – 1 – 0 – – 0 – – – – – – – – – – – – – – 0 – 1 – – 0 – – – – – – – – – – – – – – – – – – 0 – 0 – – – e_3^2 – – – – – – – – – – – – – 0 – – 0 – – – – – – – – – – – – – – – – 1 1 – 1 – – – – – – – – – – – – 0 – – 0 – – – 1 – – e_4^2 – – – – – – – – – 1 – – 1 – – – – 1 – – – – – – – – – – – 0 – – 1 – – – – 0 – – – – – – – – – – – 1 – – 0

		– – – – 0 – – – – – – – – – – – – – – – – – 1 1 – 1 – y_1^2 – – – – – – – – – – – – – – – 0 0 – 1 – – – – – – – – – – – – – – – – 1 0 – 0 – – – – – – – – – – – – – – – – 0 1 – 0 – – – – – – – – – – – – – – 0 – – – – – 0 y_2^2 – – – – – – – – – – – – – – – – – 0 – 0 – – – – – – – – – – – – – 1 – – – 1 – 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22	CNF C^1_{sim} after C^1 simplification x_4^1 x_5^1 x_1^1 x_2^1 x_3^1 e_1^1 e_2^1 e_3^1 e_4^1 y_1^1 y_2^1 – – – – – 1 – – – – – e_1^1 – – 0 – – – – – – – – – – – – – 0 – 0 – – – e_3^1 – – – – 0 – – 0 – – – – – – – 1 1 – 1 – – – – – – – – – – – – – – y_1^1 – – – – – – – – – – 0 y_2^1 – – – – 1 – – – – – –	CNF C^2_{sim} after C^2 simplification x_4^1 x_5^1 x_1^1 x_2^1 x_3^1 e_1^1 e_2^1 e_3^1 e_4^1 x_4^2 x_5^2 x_1^2 x_2^2 x_3^2 e_1^2 e_2^2 e_3^2 e_4^2 y_1^2 y_2^2 – – – – – 1 – – – – – – – – – – – – – – e_1^1 – – 0 – 0 – 0 – – – – – – – – – – – – e_3^1 – – – – 0 – – 0 – – – – – – – – – – – – – – – – 1 1 – 1 – 0 – – – – – – – – – y_2^1 – – – – 1 – – – – – – – – – – – – – – –

		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	1	–	–	–	–
			–		e_1^2			
		–	–	–	–	–	–	–
			–	–	–	–	0	–
			–	–	–	–	–	–
			–					
		–	–	–	–	–	–	–
			–	–	–	1	–	1
			–	–	1	–	–	–
			–		e_2^2			
		–	–	–	–	–	–	–
			–	–	–	0	–	0
			–	–	1	–	–	–
			–					
		–	–	–	–	–	–	–
			–	–	–	1	–	0
			–	–	0	–	–	–
			–					
		–	–	–	–	–	–	–
			–	–	–	0	–	1
			–	–	0	–	–	–
			–					
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	0	–	0	–	–
			–		e_3^2			
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			0	–	–	0	–	–
			–					
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			1	1	–	1	–	–
			–					
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	–	1	1	–	1
			–		y_1^2			
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	–	0	0	–	1
			–					
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	–	1	0	–	0
			–					
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	–	0	1	–	0
			–					
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			–	–	–	–	–	–
			0		y_2^2			
		–	–	–	–	–	–	–
			–	–	–	–	–	–
			1	–	–	–	–	–
			–					

Further we construct two-block circuit (Figure 5). Its conventional CNF C^2 (the third column of Table 1) turns out simple enough: CNF C^1 after renumbering its variable labels (the superscripts 2 are replaced for 1) is attached to CNF C^1_{sim}. At that, the columns labeled y_1^1, y_2^1 of CNF C^1_{sim} and the columns x_4^2, x_5^2 of CNF C^2 are merged. So CNF C^2 has 20 columns (instead of 11 columns of CNF C^1 and C^1_{sim}) and 28 rows. Instead of including the sixth row of CNF C^1_{sim} in CNF C^2 we substitute x_4^2 with don't-care everywhere in C^2. After that we obtain the simplified form of CNF C^2 – CNF C^2_{sim} (the third column of Table 1). After testing CNF C^2_{sim} we discover that there exists its satisfying assignment:

x_4^1	x_5^1	x_1^1	x_2^1	x_3^1	$e_1^1e_2^1e_3^1e_4^1x_4^2$		x_5^2	x_1^2	x_2^2
	x_3^2	$e_1^2e_2^2e_3^2e_4^2y_1^2$		y_2^2					
–	–	0	–	1	1	–	0	–	–
	0	0	0	1	1	1	0	–	0
	0 .								

That is $\bar{x}_1^1 x_3^1 e_1^1\ \bar{e}_3^1\ \bar{x}_5^2\ \bar{x}_1^2\ \bar{x}_2^2 x_3^2 e_1^2 e_2^2\ \bar{e}_3^2\ \bar{y}_1^2\ \bar{y}_2^2$ in the form of conjunction. So we obtain the following synchronizing sequence of the length 2:

$$\boldsymbol{X}^2 = ((0–1, 001).$$

Thus to initialize the sequential circuit (Figure 1) we should feed it at the first cycle with input signals $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ or $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, then at the next cycle with input signals $x_1 = 0$, $x_2 = 0$, $x_3 = 1$. After that both D flip-flops pass into the state 0.

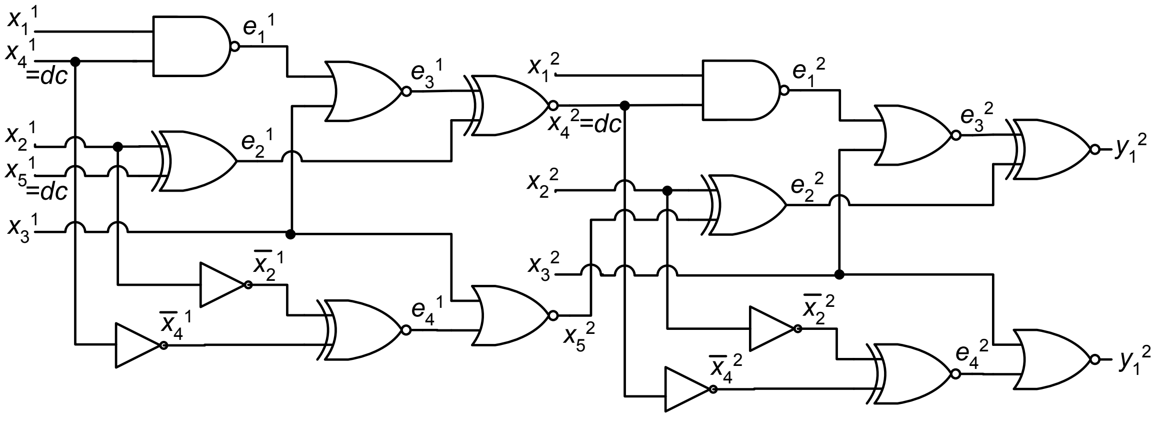

Figure 5. Two-block combinational circuit for searching part of the sequential circuit shown in Figure 1

Here one should draw attention that there are some peculiarities of searching for synchronizing sequence via Boolean satisfiability. They result from existence of don't-care signals in tested circuit that can cause local fragments of conventional CNF to be unsatisfiable though the tested circuit has a synchronizing sequence. That is because don't-care signal run through this circuit fragment from input to output. The characteristic example of such a case is some gate, such as XOR, or some subcircuit realizing such a function. To don't miss a synchronizing sequence for a circuit having such fragments, when testing CNF satisfiability it should use SAT solvers (for instance, SAT solver PicoSAT [Biere, 2008]) that permit to give proof traces from which it is possible to extract a reason why the tested CNF is erroneous. In that case we can substitute the unsatisfiable fragment with assigning don't-care value to the variable corresponding to the appropriate fragment output signal, just as we have substituted CNF fragments concerned with XOR gate.

Conclusion

In this paper the problem of search for synchronizing sequence for logic circuits with memory elements is considered. A novel reformulation of the problem as the Boolean satisfiability problem solved with any existing SAT-solver was developed. The proposed method is used when testing a memory block whether D flip-flops are good or faulty.

Bibliography

[Kohavi, 1978] Z. Kohavi. Switching and Finite Automata Theory. The McGraw-Hill College, 2 edition, 1978.

[Gill, 1962] A. Gill. Introduction to the Theory of Finite-state Machines, McGraw-Hill, 1962.

[Lee, 1996] D. Lee and M. Yannakakis. Principles and methods of testing finite state machine – a survey. In: Proceedings of the IEEE, 84(8), August 1996, pp. 1090–1123.

[Eppstein, 1990] D. Eppstein. Reset sequences for monotonic automata. In: SIAM J. on Computing, vol. 19, no. 3, 1990, pp. 500–510.

[Eén] N. Eén, and N. Sörensson. MiniSat. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat.

[Mahajan, 2005] Y. Mahajan, Z. Fu, and S. Malik. Zchaff2004: An Efficient SAT Solver. In: Theory and Applications of Satisfiability Testing (2004 SAT Solver Competition and QBF Solver Evaluation (Invited Papers)), Springer Berlin / Heidelberg, 2005, pp. 360–375.

[Goldberg, 2002] E. Goldberg, and Y. Novikov. BerkMin: A Fast and Robust SAT-Solver. In: Design, Automation, and Test in Europe, March 2002, pp. 142–149.

[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.

[Biere, 2008] A. Biere. PicoSAT Essentials. In: Journal on Satisfiability. Boolean Modeling and Computation, 2008, vol. 4, pp. 75–97.

The research was partially supported by the Fond of Fundamental Researches of Belarus (Project Ф11ОБ–041).

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*

Major Fields of Scientific Research: Logic Design, CAD systems, optimization