# Model Checker-Based Delay Fault Testing of Sequential Circuits

Savas Takan
Dept. of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey
Email: savastakan@iyte.edu.tr

Berkin Guler
Dept. of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey
Email: berkinguler@std.iyte.edu.tr

Tolga Ayav
Dept. of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey
Email: tolgaayav@iyte.edu.tr

***Abstract*—This paper applies model checker-based testing, a well-known method from software engineering, to the delay fault testing of synchronous sequential logic circuits. We first model the circuit as timed automata to reveal its timing characteristics. The model is repeatedly mutated by injecting the delay faults under a certain fault assumption and all the mutant models are checked against the given properties by exploiting a model checker. Counterexamples returned from the model checker form the basis of test input sequences. Finally, the test suite minimization is defined as an integer programming problem.**

## I. Introduction

The delay fault models consider that the fault does not cause a logical error in the circuit output, rather it causes a timing error. In case of a delay defect in a circuit, signals may not be stabilized to their final logic value until a deadline, *e.g.* the next clock edge for sequential circuits, and ignoring these defects may cause the violation of the timing properties in VLSI circuits [21][23]. To this end, this type of faults must be carefully handled in production lines as well.

Model based testing techniques, on the other hand, got much attention in the last decade since they provide cheap, flexible and optimized test cases in complex systems. The idea of this work is to get the benefit from those capabilities of model checkers in the effort of automatic test pattern generation for delay fault testing of VLSI circuits. To the best of our knowledge, this is the first study applying model based testing technique to the circuit testing.

Among different types of delay fault models and testing strategies, we select *Gate Delay* model and *At-Speed* testing strategy, and restrict our study to these assumptions. Gate delay model assumes that the delay is lumped at one gate in the circuit and it is defined as an interval. At-speed strategy means that the test sequence is applied to the circuit at the rated clock speed. The software tool developed in this study is able to transform the given sequential circuits to the timed automata model under those assumptions. By injecting the delay faults to the components of the circuit, our tool creates one mutant model for each component, which is directly sent to the model checker. The reason why we tackle the problem under the gate delay fault assumption is that its modeling is more straightforward and the number of mutants is linear in the the number of gates. The mutant models are checked against the given properties and possibly returned counterexamples constitute the test suite. Finally, we express the minimization of the test suite as a weighted set cover problem that can be solved either integer programming solvers or various heuristic algorithms. We demonstrate the methodology through a trivial, widely known sequential circuit, a traffic light controller.

The paper is structured as follows: Section II summarizes delay testing of VLSI circuits and Section III shortly explains the model based testing approach. The proposed approach is explained thoroughly in Section IV. Finally, Section V concludes the paper and gives insight into prospective future work.

## II. Delay Testing of VLSI

The aim of delay testing is to reveal the timing defects and make sure that the circuit satisfies the desired performance specifications. Unlike the other fault models, delay fault testing is highly tied to the testing strategy that defines how the tests are applied to the circuit. The speed of the testing equipment and the type of circuit such as combinational, scan, non-scan or partial scan sequential determine the testing strategy. Ordinarily, test vectors in the delay test should be applied to the circuit at the desired operating speed. However, high-speed test equipments cost much more. For that reason, testers are designed as several times slower than the actual circuit speed. Testing high-speed designs on slow testers require particular test generation and application schemes and developing new techniques for those is of paramount importance [18][27].

The delay defects in combinational circuits are observed such that a vector pair $V =< v_1, v_2 >$ is applied to the circuit inputs at certain times and the measured output is checked against its expected value. The first vector is applied for a sufficient amount of time to initialize the signals. Then, the second vector is applied for a certain amount of time, $T_c$ and the output is checked. Testing delay faults in sequential circuits is more difficult since application of arbitrary vector pairs is not possible. When studying sequential circuits, they are represented as an iterative array of combinational logic and each instance of the combinational logic is called a *time-frame*. In this case, vector pairs can be represented as $V =< i_1 + s_1, i_2, s_2 >$ where $i_1$, $i_2$ are the values of primary inputs and $s_1$, $s_2$ are the values of the state memory. $i_1$ is supposed to produce $s_2$ as the next state. There are several testing strategies such as enhanced scan, functional justification, slow-fast-slow, at-speed *etc.* and for further details one may refer to [18].

Delay faults were first used by Breuer in 1974 and have

ISBN 978-3-8007-3657-7

been extensively studied since early 1980s [9]. There are many delay fault models proposed since then. Each model has pros and cons in terms of complexity, fault coverage, test generation method *etc.* The common fault models are transition delay, gate delay, path delay, line delay and segment delay [17]. We shortly explain these models below.

Transition fault model supposes that the delay fault affects only one gate in the circuit. Each gate can have a slow-to-rise or a slow-to-fall transition fault. In fault-free circuit, each gate is supposed to have a nominal delay. In case of a fault, this delay is increased or decreased [18][26]. This model assumes that the delay is large enough to prevent transition from reaching any primary input at the time of observation. Thus, delay fault can be observed either it is propagated through a long path or a short path to any primary output.

The gate delay fault model is a quantitative model because it considers the circuit delays. Gate delay fault model assumes that the delay fault affects only one gate again and the delays of logic gates are characterized with some precision in terms of location and size. The gate delays are represented by intervals in this model. A fault is an extra delay of a particular size for the propagation of a rising or falling transition from the gate input to output [20]. In contrast to the transition model, the gate delay fault model does not suppose that the increased delay will impact the efficiency independent of the propagation path by using the fault site. The restrictions of the gate delay fault model resemble those for the transition fault model. Due to the single gate delay fault assumption a test may not able to identify delay faults that are a consequence of the sum of a few small delay defects [18]. Very long paths through the fault site might be expected to cause performance degradation. The main advantage of this model is the fact that the amount of faults in a circuit is proportional to the number of gates [20]. There are plenty of studies on the gate delay faults. For example, Ashar expressed a method for synthesizing gate-delay-fault testable multilevel circuits [1]. In 1995, Brakel showed a technique concerning the extension of delay fault test pattern generation to synchronous sequential circuits using scan techniques [25]. Cavallera pointed out the issue of simulating gate delay faults in synchronous sequential circuits. He provided a solution implemented in the fault simulator DFSIM [10]. Takahashi offered a method of diagnosing gate delay faults using delay fault simulation [24]. Irajpour created a technique for a considerable amount of gate delay faults in benchmark circuits, multiple tests collectively give more comprehensive coverage than any single test using an extended gate delay fault simulation algorithm [16]. In 2011, Bernardi explained a novel methodology that takes into account Gate Delay Fault as equal to a collection of Transition Delay Faults [7].

In path delay fault model, any path at a total delay exceeding the system clock interval is said to have a path delay fault. Distributed defect impact a whole path. Under the path delay fault model, a combinational circuit is considered faulty if the delay of any of its paths exceeds a given limit. The delay or length of the path shows the total of the delays of the gates and interconnections upon the path. A significant limitation of this fault model is the fact that the amount of paths in the circuit could be possibly exponential in the number of gates. That is why testing all path delay faults in the circuit are certainly not practical [18]. There are studies that propose to apply this testing through the critical paths only.

Line delay fault model checks a rising or falling delay fault on a given signal line in the circuit. The fault is propagated through the longest sensitizable path through the given line. Like the transition and gate delay fault models, line delay fault model assumes a single delay fault. The amount of faults equals twice the number of lines in the circuit. A test shall take care of several line delay faults. For this reason, this fault model can detect some distributed delay defects regarding the propagation paths. Since only one propagation path through each relative line is considered, it might probably not able to identify some defects [23] [19].

Segment delay fault model presents a trade-off amongst the transition delay fault model and the path delay fault model. The assumption in this model is the fact that the delay defect impacts a few gates in a local area. The idea of this fault model is to combine the benefits of the transition and path delay fault models while avoiding their limitations [14][9]. The location and length of the segment can be decided upon the basis of available statistics from manufacturing defects.

## III. Model Checker-Based Testing

Model Based Testing (MBT) is a testing technique that relies on modeling the system and verification of this model. Model checking is an automated technique [11]. It exhaustively and automatically checks whether the model of system meets a given specification [6]. If the state space exploration reveals no property violations, correctness of the property is proven. A fundamental function of model checkers is to produce witnesses and counterexamples for property satisfaction or violation, respectively. Whenever a model checker realizes that a property has failed, it immediately stops searching and returns a counterexample that illustrates the property violation. A human analyzer can utilize this counterexample to recognize and fix the design fault. MBT proposes that counterexamples form the basis of test cases [15].

The Kripke structure is the formalism widely used to describe model checking and also to determine the semantics of temporal logic.

**Definition 1** *(Kripke structure). A Kripke structure $K$ is a tuple $K = (S, S_0, T, L)$*

- *$S$ is a finite set of states.*
- *$S_0 \subseteq S$ is an initial state set.*
- *$T \subseteq S \times S$ is a total transition relation, that is, for every $s \subseteq S$ there is an $s'S$ such that $(s, s') \subseteq T$.*
- *L: $S \Rightarrow 2^{AP}$ is a labeling function that maps each state to a set of atomic propositions that hold in this state, where AP is a countable set of atomic propositions.*

A path is an infinite execution sequence for this model. A Kripke structure determines all feasible paths of a system.

**Definition 2** *(Path). A path $p :=< s_0, s_1, \ldots >$ of Kripke structure $K = (S, S_0, T, L)$ is an infinite sequence such that $\forall i \geq 0 : (s_i, s_{i+1}) \in T$ for $K$.*

ISBN 978-3-8007-3657-7

As infinite paths are not usable in practice, model checking utilizes finite sequences, known as traces.

**Definition 3** *(Trace) A trace* $t := < s_0, ...s_n >$ *of Kripke structure K is a finite sequence such that* $\forall 0 \leq i < n : (s_i, s_{i+1}) \in T$ *for K. There can be a dedicated state* $s_i$ *such that* $s_i = s_n$ *and* $i \neq n$*, which is a loop back state, and* $< s_0, \cdots, s_{i-1}, (s_i, \cdots, s_n)^\omega >$ *is a path of K.*

In model checking, properties over Kripke structures are formulated in temporal logic, which will be presented in subsection III-A. Model checking has some strengths and weak points [2]. The strengths are as follows:

- It is suitable for a broad range of applications such as software engineering, embedded systems, and hardware design.
- It supports partial verification so properties could be examined individually and enable concentrate on the essential properties first.
- No complete requirement specification is required, it is not susceptible to the likelihood that an error is exposed; this contrasts with testing and simulation that are targeted at tracing the many likely defects.
- It provides diagnostic information about instance a property is invalidated; this is quite helpful for debugging purposes.

The weak points of model checking are:

- It is primarily proper to control-intensive applications and less designed for data-intensive applications.
- Its applicability is susceptible to decidability problems; for infinite-state systems, or reasoning about abstract data types, model checking is not successfully computable.
- It verifies a system model, rather than the system that is actual product or prototype; any acquired outcome is not hence as useful as the system model.

### A. Timed Automata

Timed automata (TA) is a valuable tool for especially designing real-time systems. In this context, we transform the circuits to timed automata to express the timing behaviours of them. Let $X$ be a finite set of real valued clock variables and $V$ be a finite set of real valued data variables. A constraint $C$ is of the form:

$$C ::= z \odot k \quad | \quad z - y \odot k$$

where $z, y \in X$ or $V, k \in \mathbb{N}$ and $\odot \in \{\leq, <, =, >, \geq\}$.

**Definition 4** *(Timed Automaton). A timed automaton is a tuple* $(Q, q_0, X, \Sigma, \delta, I)$ *where:*

- $Q$ *is a finite set of locations.*
- $q_0 \in Q$ *is the initial location.*
- $X$ *is a finite set of clock variables.*
- $\Sigma$ *is the set of denoting actions.*
- $\delta \subseteq Q \times 2^C \times \Sigma \times 2^X \times Q$ *is the set of transitions.*
- $I : Q \to 2^C$ *assigns invariants to locations.*

A clock valuation is a function $u : X \to \mathbb{R}_{\geq 0}$ from the set of clocks to the non-negative reals. Let $\mathbb{R}^X$ be the set of all clock valuations. Let $u_0(x) = 0$ for all $x \in X$. We will abuse the notation by considering guards and invariants as sets of clock valuations, writing $u \in I(q)$ to mean that $u$ satisfies $I(q)$.

**Definition 5** *(Semantics of Timed Automaton). Let* $(Q, q_0, X, \Sigma, \delta, I)$ *be a timed automaton. The semantics is given by a transition system* $\langle S, s_0, \to \rangle$ *where* $S \subseteq L \times \mathbb{R}^X$ *is the set of states,* $s_0 = (q_0, u_0)$ *is the initial state and* $\to \subseteq S \times \{\mathbb{R}_{\geq 0} \cup \Sigma\} \times S$ *is the transition relation such that:*

- $(q, u) \xrightarrow{d} (q, u + d)$ *if* $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(q)$*, and*
- $(q, u) \xrightarrow{a} (q', u')$ *if* $\exists (q, g, a, r, q') \in \delta : u \in g$*,* $u' = [r \mapsto 0]u$ *and* $u' \in I(q)$*.*

*where for* $d \in \mathbb{R}_{\geq 0}$*,* $u + d$ *maps each clock* $x$ *in* $X$ *to the value* $u(x) + d$*, and* $[r \mapsto 0]u$ *denotes the clock valuation which maps each clock in* $r$ *to* $0$ *and agrees with* $u$ *over* $X \setminus r$*.*

Time may pass only if it satisfies the invariant of the current state. A transition of the automaton may occur if and only if its guard and the invariant of the new state are satisfied. The semantics of the automaton is the set of traces of the associated transition system. Timed automata are often composed into a network of timed automata over a common set of clocks and actions, consisting of $n$ timed automata.

**Definition 6** *(Network of Timed Automata). Let* $(Q_i, q_i^0, X_i, \Sigma_i, \delta_i, I_i)$ *be a network of* $n$ *timed automata. Let* $\bar{q}_0 = (q_1^0, q_2^0, \ldots, q_n^0)$ *be the initial location vector. The semantics is defined as the transition system* $\langle S, s_0, \to \rangle$*, where* $S = (Q_1 \times \ldots \times Q_n) \times \mathbb{R}^X$ *is the set of states,* $s_0 = (\bar{q}_0, u_0)$ *is the initial state and* $\to \subseteq S \times S$ *is the transition relation.*

### B. Specification Language

Specifications will be expressed in real-time temporal logic TCTL, which extends the the computation tree logic CTL with clock variables.

**Definition 7** *(Syntax of TCTL). The formulas* $\varphi$ *of TCTL are defined inductively by the grammar*

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \exists \mathcal{U}_I \varphi \mid \varphi \forall \mathcal{U}_I \varphi,$$

*where* $p \in Pr$ *is an atomic proposition and/or clock variables and* $I \in \mathcal{I}$ *is an interval in the set of intervals* $\mathcal{I}$ *appearing in* $\varphi$*.*

From the above syntax, we can derive the following operators (for further details on the semantics of TCTL and derivation of operators, one may refer to [8], [22] and [12]):

ISBN 978-3-8007-3657-7

$\exists \Diamond \psi$ (Possibly). There exists a path that property $\psi$ possibly holds.
$\forall \Box \psi$ (Invariantly). Property $\psi$ always holds.
$\exists \Box \psi$ (Potentially always). There exists a path along which property $\psi$ always holds.
$\forall \Diamond \psi$ (Eventually). Property $\psi$ eventually holds.
$\psi \rightsquigarrow \varphi$ (Leads-to). Whenever property $\psi$ holds, property $\varphi$ eventually holds.
$\psi \rightsquigarrow_{\leq t} \varphi$ (Time-bounded Leads-to). Whenever property $\psi$ holds, property $\varphi$ eventually holds in at most $t$ time units.

*Safety Properties*. Safety properties are of the form: "something bad will never happen". For instance, in a model of aircraft, a safety property might be that the altitude must never exceed its maximum value.

*Liveness Properties*. Liveness properties are of the form: "something will eventually happen", *e.g.* when pressing the button of the engine start, then eventually the engine should start.

*Bounded Liveness Properties*. In real-time systems, a liveness property is not sufficient and bounded times response should be investigated. Bounded time liveness property can be expressed with a time-bounded leads-to operator, *i.e.* $\varphi \rightsquigarrow_{\leq t} \psi$. These properties can be reduced to simple safety properties such that first the model under investigation is extended with a boolean variable $b$ and an additional clock $z$. The boolean variable $b$ must be initialized to `false`. Whenever $\varphi$ starts to hold $b$ is set to `true` and the clock $z$ is reset. When $\psi$ commences to hold $b$ is set to `false`. Thus the truth-value of $b$ indicates whether there is an obligation of $\psi$ to hold in the future and $z$ measures the accumulated time since this unfulfilled obligation started. The time-bounded leads-to property $\varphi \rightsquigarrow_{\leq t} \psi$ yields the verification of the safety property $\forall \Box\, b \Rightarrow z \leq t$. Similarly, we can define $\varphi \rightsquigarrow_{\geq t} \psi$ to express that $\psi$ must hold at least $t$ time units after $\varphi$ commences to hold.

### C. UPPAAL Model Checker

UPPAAL is a software tool for validation through a graphical simulation and verification via model checking. It was developed by Uppsala and Aalborg Universities and it has been widely used by both industry and academia in projects ranging from communication protocols to multimedia applications. The tool allows to verify systems modeled as network of timed automata augmented with variables, structured data types, user-defined features and channel synchronization [3][4]. UPPAAL accepts timed-automata models in XML format. It also has a Java-based GUI that allows to create and simulate the models. For a detailed explanation of UPPAL, one may refer to [5].

## IV. Model Based Testing of Delay Faults

In this section, we explain our methodology called Model Based Testing of Delay Faults (MBT-DF) and the developed software tool that applies model checker-based testing to the delay fault testing of synchronous sequential logic circuits. The methodology is shown in Figure 1. The first step is to model the circuit as timed automata to characterize precisely the timing properties of the circuit. The model is mutated by injecting the gate delay faults. Although the methodology cannot be tied to a certain type of fault model and testing strategy, we start this study with two assumptions: 1) Testing strategy is "At-Speed" and 2) Fault model is "Gate-Delay". Once all the mutant models are generated, they are checked against the given properties by exploiting the model-checker, UPPAAL. Counterexamples returned from the model checker form the basis of test input sequences. In the last step, the test suite is minimized. Gate delay fault model assumes the delay

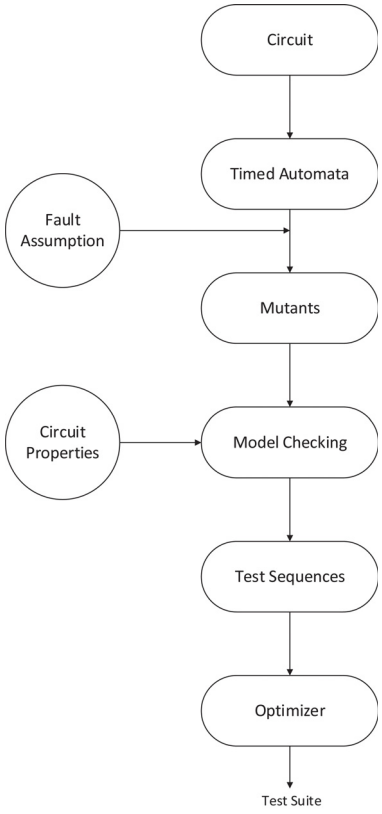



Fig. 1. MBT Methodology

is lumped at one gate in the circuit. The main advantage of this model is that the number of faults is linear in the number of gates in the circuit. The delays of gates are represented as intervals and each gate may have different delay intervals due to the manufacturing variations. For example, a 2-input AND gate can be expressed with equation $o(t) := i_1(t)_{\text{AND}}\, i_2(t)$. We assume that the input signals $i_1(k)$ and $i_2(k)$ are refreshed at times $t_1$ and $t_2$ and become $i_1(k{+}1)$ and $i_2(k{+}1)$ respectively. The propagation delay of the gate is given with the interval $\delta = [t_{min}, t_{max}]$. Therefore the new output is assumed to be stable at time $t = \max\{t_1, t_2\} + \delta$. The rule for sequential circuits is such that the outputs of the next-state logic circuit must be stable before the next clock transition. If sum of the delays of cascaded gates through the longest path from primary and state inputs to the next-state outputs exceeds one clock cycle, this may lead to incorrect state transitions and consequently incorrect primary outputs.

While transforming the circuit into the timed automata, we represent each gate with a timed automaton separately. Memory elements, flip-flops, clock and other several components of the circuit can be represented with appropriate timed automata as well. Therefore, the whole circuit is transformed

ISBN 978-3-8007-3657-7

to a network of timed automata. The automatons communicate via the synchronization channels. To give some intuition to the reader, we present a few examples below. Figure 2 shows

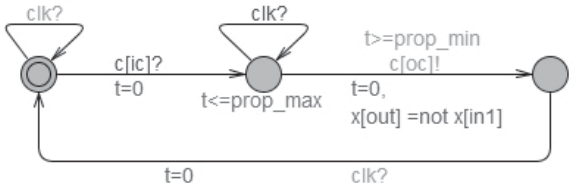



Fig. 2. Automaton for NOT gate

the TA of a NOT gate where $x[in1]$ is the input and $x[out]$ is the output signal. The gate waits at its initial state until the input is stable, *i.e.* $c[ic]$ is fired. Signal propagation is simulated with the communication channels. In the model of the circuit, one communication channel is defined for each wire. Once the input is stable, the gate is assumed to start working and the automaton waits for a propagation delay. After the delay has elapsed, it performs the logic function $x[out] = not\ x[in1]$, firing up the communication channel $c[oc]$. The model checker expands the system on this timing interval during the verification. Figure 3 shows the TA of 2-input AND gate. $x[in1]$, $x[in2]$ are the input signals and

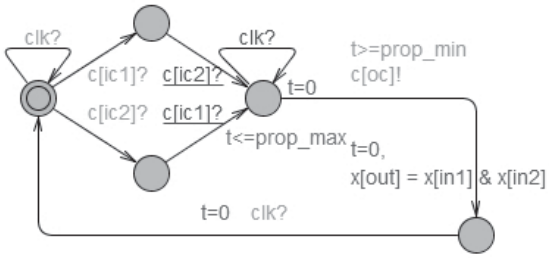



Fig. 3. Automaton for AND gate

$x[out]$ is the output signal of the gate. When both inputs are stable, *i.e.* both signals $c[ic1]$ and $c[ic2]$ are fired, the gate starts working and after a delay period it performs the logic function $x[out] = x[in1]\&x[in2]$, firing the output channel $c[oc]$. All 2-input gates will be the same except for the logic operator in the logic function. Figure 4 shows the

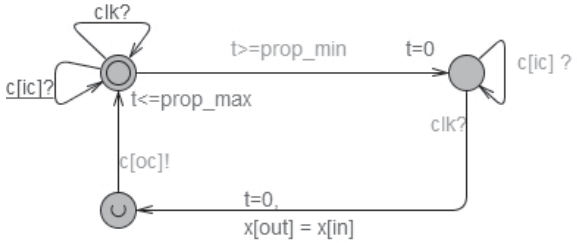



Fig. 4. Automaton to represent D Flip-Flop

automaton of DFF. It assigns the input to the output after the propagation delay elapses and a clock transition occurs regardless of whether the input is stable or not. Figure 5 is the TA for clock signal. The clock period is given with the parameter $T$. The invariant $t \leq T$ and the guard $t \geq T$

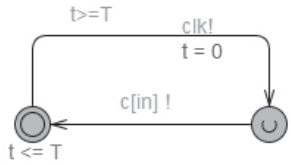



Fig. 5. Automaton to simulate clock

together forces the automaton to take the transition, firing the channel $clk$ at each clock period $T$. Then the additional channel $c[in]$ is immediately fired to trigger the gates with the primary and secondary inputs. Last, a critical automaton


Fig. 6. Automaton to simulate one primary input

for simulating the primary inputs of the circuit is given in Figure 6. One automaton must be defined for each primary input. The selection expression $i : int[0, 1]$ allows to assign an arbitrarily chosen boolean value 0 or 1 to the primary input $x[in]$. UPPAAL expands the search for both values during the verification, which guarantees that all possible combinations of the test inputs are taken into account.

We developed a Java-based tool that implements all the stages of the proposed methodology. Our tool has two dependencies, UPPAAL's verification program to generate the counterexamples and GLPK software [13] to optimize the test suite. The tool can parse the circuits given in Verilog format and produce the gate and wire lists. User is allowed to select/unselect the gates of which the mutants will be created. The clock period, the nominal and faulty delay ranges can also be defined through the GUI seen in Figure 7. User creates the mutants that are in fact TA models in UPPAAL-readable XML format. Properties in TCTL language must be supplied manually by the user. Finally, the program generates counterexamples by running UPPAAL's verifier and then optimizes the test suite by exploiting GLPK optimization software.

We express our methodology through a common example, Traffic Light Controller (TLC) used at a road intersection as seen in Figure 8. There are two types of road crossing: quiet crossings that use a simple sequence, and busy crossings require a longer (delayed green) sequence. Some junctions may use the busy sequence during the day and the quiet sequence at night. One digital input signal called J (for junction type) will indicate whether the road crossing is considered quiet. J=0 denotes a busy junction and J=1 a quiet one. Thus, we have a one-input, six-output synchronous system to design. The FSM and the circuit is given in Figure 9. There are two types of properties that can be defined for the sequential circuits. We call them Type-1 and Type-2 properties in this study. The Type-1 properties can be generated automatically from the FSM-level requirements. For example, the TLC has 8 states and 12 transitions. For the transition from State 1 to

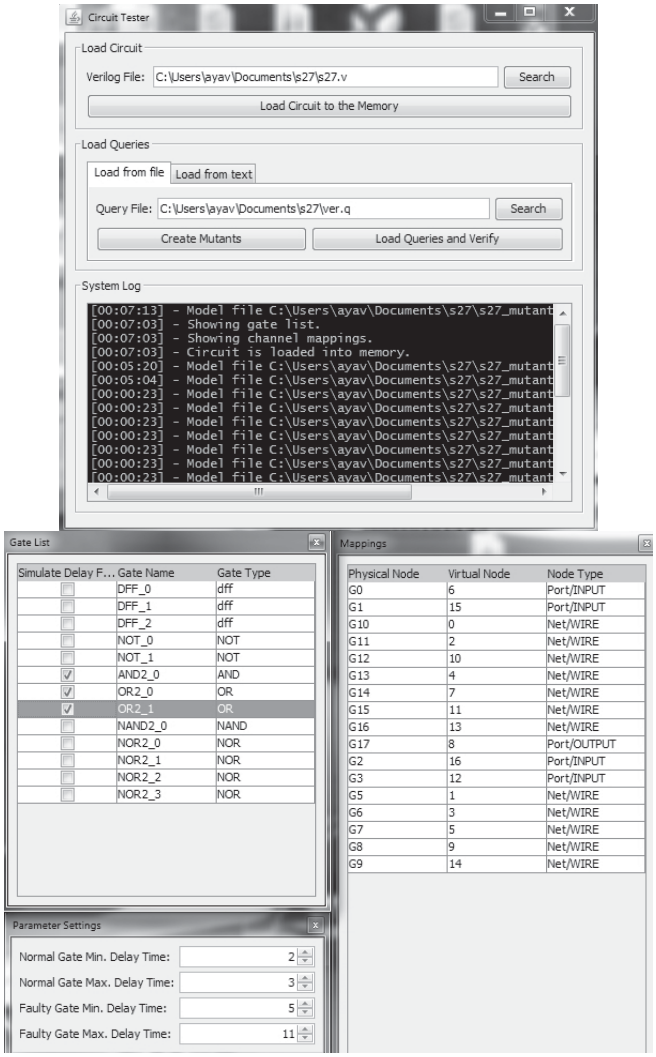



Fig. 7. ATPG Tool for MBTDF

State 2, the following property can be written:

$$\text{P1}: \quad (\text{Q1=0\&Q2=0\&Q3=1}) \rightsquigarrow_{\leq T_{clk}} (\text{Q1=0\&Q2=1\&Q3=0})$$

Type-2 properties are defined at a higher level and can be extracted from the system specifications manually. For instance, the following safety property is essential for the TLC and it must be satisfied; Two green lights must never be turned on at the same time:

$$\text{P2}: \quad \forall \ \Box \ \text{not (G1=1 and G2=1)}$$

The property simply says that condition (G1=1 and G2=1) must never be true. Playing with the delay parameters $t_{min}$ and $t_{max}$ for each gate, several mutants of the circuit can be generated. For instance, mutant M1 is such that the delay of AND gate that outputs G2 is sufficiently large. This mutant does not hold P2. If we run the program for this specific mutant against property P2, the following counterexample is returned:

$$T_1 = [(0/010000), (0/100100), (0/010100), (0/100100),$$

$$(0/001000), (0/001001)]$$

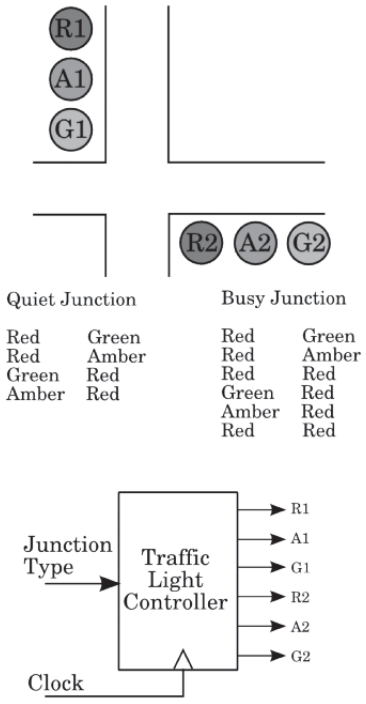
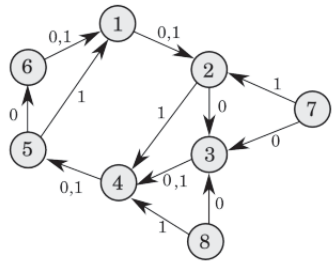



Fig. 8. Traffic Light Controller



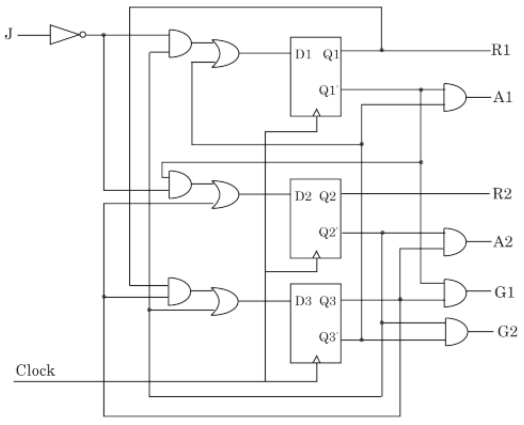




Fig. 9. FSM and Circuit of Traffic Light Controller

whereas the order of the input and outputs is like

(J/R1A1G1R2A2G2). As seen, the last state of the test sequence invalidates property P2 since both G1 and G2 are true.

For this circuit, several Type-1 and Type-2 properties can be written. Under the gate delay fault assumption, we can create as many mutants as the number of gates. Our automatic test pattern generation tool generates the counterexamples.

Once all the counterexamples are collected, the last stage is to optimize the test suite. We express the optimization of test suite as a weighted set cover problem. Let us assume that for $m$ mutants, we check them against $p$ properties and obtain test sequences $T_1, T_2, \ldots, T_n$. We expect that $n$ is approximately $m \times p$. A test suite with minimum number of test sequences or minimum size that reveal all the faults is of interest. We denote the weights with the lengths of test sequences:

$$\mathbf{w} = [w_1, w_2, w_3, \ldots, w_n]^T.$$

For example, the length of $T_1$ is 6. Let $\mathbf{x}$ be decision variables associating to the test sequences. If $x_i = 1$ then test sequence $i$ is included in the test suite. This is well-known weighted set cover problem and it can be defined formally as an integer programming problem:

$$\min \mathbf{w}^T \mathbf{x} \tag{1}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \geq \mathbf{1} \tag{2}$$

where $\mathbf{A}$ is a $m \times n$ binary matrix. Each row of the matrix is associated with a mutant and each column indicates whether its associating test sequence belongs to that mutant or not. The solution to this problem gives us the test suite with minimum size.

## V. Conclusion

We proposed a methodology applying the model based testing technique to the delay fault testing of VLSI circuits. The advantage of the methodology is due to the exhaustive search capability of model checking, which supposedly finds out the optimum test suite. The drawback seems to be the high cost of model checking in terms of computational and memory resource usage. The idea is to transform the circuit to its equivalent timed automata, to create the mutant models under certain fault assumptions and then generate the counterexamples using UPPAAL model checker. The fault model is *Gate-Delay* and the testing strategy is *At-Speed Testing*. The developed software can successfully generate the test suite in individual experiments with the traffic light controller and some simple ISCAS'89 circuits such as s27 and s344, yet a lot of experiments and extensive comparisons are needed to prove the success of the methodology. The future research will cover the deployment of other fault models and testing strategies as well. In order to improve the performance of the tool, additional effort in the circuit abstraction, *i.e.* compacting the timed automata model could be of utmost importance.

## References


[1] Pranav Ashar, Srinivas Devadas, and Kurt Keutzer. Gate-delay-fault testability properties of multiplexor-based networks. *Formal Methods in System Design*, 2:93–112, 1993.

[2] Christel Baier and Joost-Pieter Katoen. *Principles Of Model Checking*, volume 950. 2008.

[3] Gerd Behrmann, Alexandre David, and Kim Larsen. *A Tutorial on Uppaal*, volume 3185. 2004.

[4] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hå kansson, Paul Petterson, Yi Wang, and Martijn Hendriks. Uppaal 4.0. In *Third International Conference on the Quantitative Evaluation of Systems, QEST 2006*, pages 125–126, 2006.

[5] Gerd Behrmann, Re David, and Kim G. Larsen. A tutorial on uppaal. pages 200–236. Springer, 2004.

[6] Mordechai (Moti) Ben-Ari. A primer on model checking. *ACM Inroads*, 1(1):40, 2010.

[7] P. Bernardi, M. Sonza Reorda, a. Bosio, P. Girard, and S. Pravossoudovitch. On the modeling of Gate Delay Faults by means of Transition Delay Faults. *Proceedings - IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 226–232, 2011.

[8] Mustapha Bourahla and Mohamed Benmohamed. Verification of real-time systems by abstraction of time constraints. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 238.1, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Stefano Di Carlo and Paolo Prinetto. Models in Hardware Testing. In *Models in Hardware Testing*, volume 43. 2010.

[10] P. Cavallera, P. Girard, C. Landrault, and S. Pravossoudovitch. DFSIM: a gate-delay fault simulator for sequential circuits. *Proceedings ED&TC European Design and Test Conference*, 1996.

[11] Em Clarke, O Grumberg, and Da Peled. Model Checking. 1999.

[12] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal logic (vol. 1): mathematical foundations and computational aspects*. Oxford University Press, Inc., New York, NY, USA, 1994.

[13] GLPK. GNU Linear Programming Kit. `http://www.gnu.org/software/glpk`, 2000.

[14] K Heragu, J H Patel, and V D Agrawal. Segment delay faults: a new fault model. *VLSI Test Symposium, 1996., Proceedings of 14th*, pages 32–39, 1996.

[15] Document Id, Contact Person, and Contract Number. State of the Art Survey. pages 1–60, 2008.

[16] Shahdad Irajpour, Sandeep K. Gupta, and Melvin a. Breuer. Multiple tests for each gate delay fault: Higher coverage and lower test application cost. *Proceedings - International Test Conference*, 2005:1211–1219, 2005.

[17] Niraj K. Jha and Sandeep Gupta. *Testing of digital systems*. 2003.

[18] Angela Krstic and Kwang-Ting Cheng. *Delay Fault Testing for VLSI Circuits*, volume 26. 1998.

[19] Chin Jen Lin Chin Jen Lin and S.M. Reddy. On Delay Fault Testing in Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5), 1987.

[20] Ananta K. Majhi and Vishwani D. Agrawal. Tutorial: Delay Fault Models and Coverage. *Design*, 1:364–369, 1997.

[21] Alicja Pierzynska and Slawomir Pilarski. Pitfalls in delay fault testing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16:321–329, 1997.

[22] Y. Tachi and S. Yamane. Real-time symbolic model checking for hard real-time systems. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, page 496, Washington, DC, USA, 1999. IEEE Computer Society.

[23] Mihkel Tagel, Peeter Ellervee, and Gert Jervan. *Design and Test Technology for Dependable Systems-on-Chip*. 2010.

[24] Hiroshi Takahashi and Kwame Osei Boateng. Diagnosis of Single Gate Delay Faults in Combinational Circuits using Delay Fault Simulation. pages 108–112, 1998.

[25] G. van Brakel, U. Glaser, H.G. Kerkhoff, and H.T. Vierhaus. Gate delay fault test generation for non-scan circuits. *Proceedings the European Design and Test Conference. ED&TC 1995*, 1995.

[26] John A Waicukauski, Eric Lindbloom, Barry K Rosen, and Vijay S Iyengar. TRANSITION FAULT SIMULATION. *IEEE Design and Test of Computers*, 4:32–38, 1987.

[27] Laung-Terng Wang, Charles E. Stroud, and Nur a. Touba. *System-on-Chip Test Architectures: Nanometer Design for Testability*. 2010.