Distributed Approach for Parallel Exact Critical Path Tracing Fault Simulation

Eero Ivask, Sergei Devadze, and Raimund Ubar

***Abstract*—Distributed computing attempts to aggregate different computing resources available in enterprises and in the Internet for computation intensive applications in a transparent and scalable way. Fault simulation used in digital design flow for test quality evaluation can require a lot of processor and memory resources. To speed up simulation and to overcome the problem of memory limits in the case of very large circuits, a method of model partitioning and the procedure of parallel reasoning for several distributed simulation agents was proposed. The concept and implementation of web-based distributed system was introduced.**

***Index Terms*—distributed computing; fault simulation; critical path tracing; digital test**

I. Introduction

FAULT simulation is a central task used in the digital design process in order to estimate the quality of tests prepared for digital electronic device. In addition, the procedure of fault simulation is often required for other test-oriented tasks such as fault diagnosis, automatic test pattern generation (ATPG), test compaction, design of reliable systems and others. For certain tasks (ATPG, built-in self test optimization, etc), the intermediate step of fault simulation need to be carried out many times hence making the simulation speed be key issue in the acceleration of the overall task performance.

Today, complexity of integrated circuits is still increasing according to Moore's law, which states that transistor density doubles about every two years. This trend is predicted to continue at least for another decade, posing serious testing problems. One approach to cope with the problem is to improve the fault analysis algorithms towards better scalability. However, the abundance of different fault simulation methods proposed during the last decades leaves almost no room for further improvement. Another way to gain practical speedup is to parallelize the task execution. There are several possibilities: algorithm can be parallelized, circuit model can be partitioned into separate components and simulated concurrently, fault set or test pattern set can be divided and simulated in parallel.

Several parallel processing algorithms have been proposed to speed up fault simulation [1]. The techniques that use fault partitioning is one reasonable way to decrease simulation time Another approach, also able to accelerate fault simulation, relies on test vectors partitioning. Combining fault parallelism with vector parallelism has been proved to be even more effective: easy-to-detect faults are identified with fast preprocessing and simulated in parallel among processors, remaining faults are targeted by all processors, each using only subset of test vectors corresponding to its partition [2].

Circuit partitioning has got less attention, as its speedup has been relatively small so far. Parallel fault simulation with circuit partitioning was used in [3,4] for vector-synchronous implementations on message passing multiprocessor systems. Circuit partitioning approach for shared memory systems was presented in [5]. The method in [6] distributes the component models of the circuit partitions to unique processors of a parallel processor system for concurrent and asynchronous execution. Partitioning issues are not handled here, however manual partitioning is supported.

Less effort has been spent in the area of algorithmic parallelism. For instance, the pipelined approach in which the specific simulation functions are assigned to different processors is given in [7,8]. The latter solution has been shown more effective than circuit partitioning. Recent attempts in the field are aimed to avoid redundant work by judicious task decomposition [1]. In addition, it adopts a cyclic fault partitioning method based on the LOG [9] partitioning and local redistribution, resulting in a well-balanced load distribution.

Recently, new trend has been the use of graphics processing units for general purpose computing by exploiting thread level parallelism. Fault simulation approach using faults and vectors partitioning is proposed in [20].

Current paper presents distributed loosely coupled asynchronous Internet-based fault simulation approach, which relies on model parallelism and test parallelism. Fault detectability computational model for the circuit is divided and at the same time, also test pattern set is divided. Sub-sets of test are evaluated on partial computational models concurrently on different computers in wide or local area network. Our approach has no specific fault list, instead faults reside in simulation model. During model partitioning some overlaping occurs as we want to avoid interdependences, because of the communication lags. Therefore these repetitive model parts are obviously simulated several times. In fault

This work was supported by Estonian SF grants 7068, 7483, EC FP7 IST project DIAMOND, ELIKO Development Centre and European Union through the European Regional Development Fund (Research Centre CEBE).

Authors are with Department of Computer Engineering Tallinn University of Technology Tallinn, Estonia, email: {ieero, serega, raiub}@pld.ttu.ee

coverage point of view this does not interfere – results will be accumulated. This only results in some speed penalty, but this will be offset by gain in memory reduction. Distributed approach can aggregate more computational resources having a potential to ease the large circuits fault simulation problem.

The presented fault analysis algorithm is based on well-known critical path tracing (CPT) technique [10]. Traditional CPT consists of simulating the fault-free circuit and uses the computed signal values for backtracking all sensitized paths from primary outputs to primary inputs in order to determine the detected faults. The trace continues until the paths become non-sensitive or end at network primary inputs. Faults on the sensitive (critical) paths are detected by the test.

Although by using CPT one can process all the faults by a single run for many test patterns in parallel, conventional CPT approach gives the exact results only for circuits without reconvergent fanouts. A modified CPT technique that is linear time, exact, and complete is proposed in [11]. However, the rule based strategy does not allow simultaneous parallel analysis of many patterns.

Parallel critical path tracing in fanout-free regions (FFR) combined with parallel simulation of stem faults was investigated in [12]. In [13] the concept of parallel critical path tracing was generalized for using it beyond FFRs. In addition, circuit in [13] is modeled by network of macros instead of network gates providing higher level of abstraction (hence higher simulation speed) but preserving gate-level fault modeling accuracy. To describe circuit on macro-level special class of binary decision diagrams called structurally synthesized BDDs [14] is used.

Current distributed framework was initially inspired from MOSCITO system [16], intended for local tools in LAN mainly. Major obstacle for Internet based use was TCP/IP socket based communication, which conflicted with firewalls. More flexible web-based solution for remote tool usage was proposed in [17]. In current paper this concept is revised and improved to support distributed fault simulation.

There exist also several general purpose frameworks for distributed computing like BOINC [18], Globus [28], and AliCE [19] for example. By far, most popular is BOINC (Berkeley Open Infrastructure for Network Computing), a non-commercial middleware system for volunteer computing, originally developed to support the SETI@home project, but intended to be useful for other applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics. The intent of BOINC is to make it possible for researchers to tap into the enormous processing power of personal computers around the world.

A major part of BOINC is the backend server. The server can be run on one or many machines to allow BOINC to be scalable for projects of any size. BOINC servers run on Linux based computers and use Apache, PHP, and MySQL as a basis for its web and database systems. Framework uses cross-platform WxWidgets toolkit for building GUI-s.

BOINC is the infrastructure which downloads distributed applications and input data (work units), manages scheduling of multiple BOINC projects on the same CPU, and provides a user interface to the integrated system.

Scientific computations are run on participants' computers and results are analyzed after they are uploaded from the user PC to a science investigator's database and validated by the backend server. The validation process involves running all tasks on multiple contributor PCs and comparing the results.

Major drawback of the BOINC infrastructure is the use of remote procedure call (RPC) mechanisms which is often felt to be security risk, because they can be the route by which hackers can intrude upon targeted computers (even if it's configured for connections from the same computer). Another disadvantage is that BOINC servers are not simple to deploy as they are based mainly on a large number of PHP scripts and project is poorly documented which makes creating a new BOINC project not easy. Use of PHP over Java can not considered as an advantage.

Globus is a collection of libraries and programs that address common problems that occur when building distributed services and applications. Issues relating to security, resource access, management, discovery, data transfer, service deployment, system components monitoring and user control are handled. Globus toolkit makes extensive use of Web Services [29] to implement these infrastructure services. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP [35]) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [29]. Initially, work on Globus was motivated by demand of virtual organizations in science, then business applications became also important. Now, Globus is deployed in many large projects like TeraGrid [30], Open Science Grid [31], LHC Computing Grid [32], etc. Globus services are used to support different communities, each of which then executes their own application specific code on top of those services. Disadvantage of Web service based solutions is their reliance on XML markup notation- nicely readable to human being and easy to parse for computer programs, but it requires more processing power and network bandwidth.

AliCE, developed in National University of Singapore, attempted to become grid development system instead of just being collection of grid tools. Similarly to Globus, AliCE core layer has components for resource management, discovery, and allocation, data management, monitoring and accounting, communication and security infrastructure. On top of that comes extensions layer consisting distributed-shared memory programming templates, runtime support infra–structure and advanced data services. Running system has consumer, producer and resource broker entities. Consumer submits application code to grid, resource broker directs the application to appropriate task farm manager which initiates the application and creates a pool of tasks. Task references are

returned to the resource broker, which schedules the tasks for execution on producers. Results are returned to the consumer. Communication supports the migration of codes, data and results via "Space"– a special form of shared memory. Communication is carried out with objects. Objects and code are serialised and packed into jar archive fail. AliCE is based on Java Jini technology [33] and JavaSpaces [34]. Java Native Interface (JNI) is used to invoke non-Java code. Authors have used the system in several projects, but it seems that activity around AliCE has lost its momentum at present. Reasons are not clear, since technology itself is still promising. Only drawback in our point of view is that Jini technology is based on Remote Method Invocation (RMI)- although elegant programming solution for distributed computing, were one program can remotely invoke methods physically residing in other machine, however, firewall traversal can be problematic as dedicated communication ports are needed. Strict security policy might not allow that.

The rest of the paper is organized as follows. Section II gives theoretical explanation of the presented fault simulation algorithm and describes the procedure of construction of computational model. The approach for computational model partitioning is given in Section III. Section IV presents web-based infrastructure for distributed simulation. Experimental results are discussed in Section V and finally conclusions about the presented method are drawn in Section VI.

II. Fault Simulation Algorithm

The overall goal of fault simulation is to evaluate the behavior of a circuit in case of presence of faults inside it. In particular, fault simulation has to determine whether the output response of a circuit is changing due to the influence of a fault or not. A fault which effect propagates to primary outputs under current input stimulus is referred as detected by the current test pattern.

Fault simulator typically works with a specific fault model. In this paper we will consider fault simulation algorithm that works with single stuck-at fault model (SAF). The presence of stuck-at fault in a digital circuit permanently fixes the value of corresponded signal line to logic one (stuck-at 1) or logic zero (stuck-at 0). The single stuck-at model that is commonly used in practice permits only sole stuck-at fault to present in a circuit at a time.

The input data of fault simulator is a set of test patterns together with the model of a circuit. In general case, the result of the execution of fault simulator is a fault table that shows what of the modeled faults are detectable by each of the given test patterns.

A. Theoretical background

Let us consider combinational circuit as a network of blocks where single block represents a subnetwork of gates with single output. Then, a fanout-free region (FFR) of combinational circuit is a block that does not contain reconverging fanout stems (i.e. represents a tree-like subcircuit). Since the traditional critical path tracing technique in FFR [10] is independent of the region size, we will consider in the following the combinational circuits as networks of FFRs with maximum size.

In [36] it has been shown that the set of faults on primary inputs and the faults at the fanout branches of a combinational circuit is the representative set of collapsed faults that has to be tested. Therefore, it is enough to consider only the faults that reside on inputs of FFR blocks in order to carry out complete fault simulation for an arbitrary circuit.

Consider a fanout-free region represented by a Boolean function $y = F(x_1,\ldots, x_i, x_j, \ldots x_n)$. The task of fault simulation can be reduced to calculation of Boolean derivatives: if $\partial y/\partial x_j = 1$ then the fault is propagated from x_j to y. This check can be performed in parallel for a set of test patterns. In order to extend the parallel critical path tracing beyond the fanout-free regions we use the concept of partial Boolean differentials.

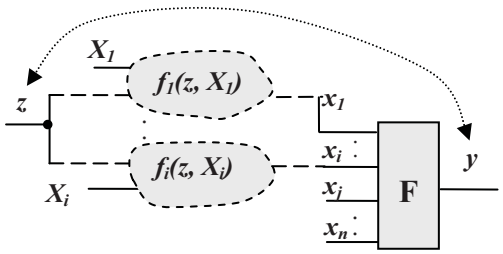


Figure 1. Reconvergent FFR in a circuit

Consider a fan-in subcircuit F of the converging fanout region depicted in Fig. 1, and represented by a function $y = F(x_1, \ldots, x_i, x_j, \ldots x_n)$. Each input of function F corresponds either to input without fanout or to branch of fanout input.

Assume that the inputs $x_1,\ldots,x_i$ of the subcircuit F are connected to the fanout stem z via subcircuits without reconvergencies and represented by functions $x_1 = f_1(z,X_1)$, …, $x_i = f_i(z,X_i)$, where X_i are vectors of variables. Then all possible fault propagation conditions for the circuit in Fig. 1 can be represented by the full Boolean differential:

$$dy = dF = y \oplus F((x_1 \oplus dx_1),\ldots,(x_i \oplus dx_i), (x_j \oplus dx_j),\ldots,(x_n \oplus dx_n)) \quad (1)$$

By the Boolean variable dx we denote the erroneous change of the value of x because of a propagated fault. In [13] we have shown that if a SAF is detected by a test pattern at y then the fault at the fanout stem z which converges in y at the inputs $x_1, \ldots, x_i$, is also detected iff

$$\frac{\partial y}{\partial z} = y \oplus F((x_1 \oplus \frac{\partial x_1}{\partial z}),\ldots,(x_i \oplus \frac{\partial x_i}{\partial z}), x_j,\ldots,x_n) = 1 \quad (2)$$

From (2), a method results for generalizing the parallel exact critical path tracing beyond the fanout-free regions. All the calculations in (2) can be carried out in parallel since they are Boolean operations.

In a general case of nested reconvergencies the formula (2) can be used recursively. If a stuck-at fault is detected by a test pattern on the output y of a subcircuit in Fig. 2 with two

nested reconvergencies, $y = F_y(x_1,z,X_y)$ and $z = F_z(x,X_z)$, where X_y and X_z are not depending on x, then the fault at the common reconverging fanout stem is also detected iff

$$d_x y = y \oplus F_y(x_1 \oplus \frac{\partial x_1}{\partial x}, z \oplus d_x F_z, X_y) = 1 \qquad (3)$$

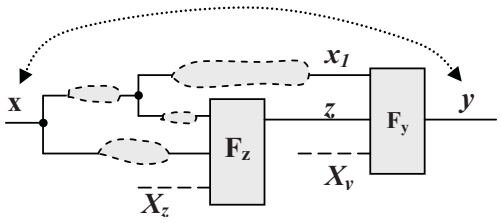


Figure 2. Nested reconvergencies in a circuit

The formula (3) can be used for calculating the influence of the fault at the common fanout stem x on the output y of the converging fanout region by calculation of partial Boolean differentials, first d_xF_z, and then d_xy. The formula (3) can be iteratively generalized for arbitrary configuration of nested reconvergencies by topological analysis of the circuit. On the other hand the derived full Boolean differentials can be easily transformed into fast computable critical path tracing procedures to be carried out in parallel for sets of test patterns.

The described exact parallel path tracing fault analysis is carried out in the following sessions:

- topological pre-analysis to create topology graph of source circuit
- construction of computational model of the circuit that consist of Boolean formulas for critical path tracing beyond FFRs.
- parallel fault backtracing on the created computational model

The topological pre-analysis and construction of computational model are performed only once to serve all the next sessions of the procedure.

B. Topological pre-analysis

The first procedure of the topological analysis is carried out in the direction from primary inputs to primary outputs of the circuit. By this procedure, all the fanout stems and all the reconvergent fan-in nodes of the circuit will be found. As the result of the procedure, a graph $G = (N, U)$ is created which represents a skeleton of the circuit. Let N be the set of nodes in G that represent either outputs of the gates with fanout branches or the outputs of fan-in gates where at least two paths from the same fanout stem converge.

Each edge $(x,y) \in U$ between two neighbour nodes x and y in the graph G represents a signal path in the circuit through the gates without fanouts and without fan-ins with reconvergencies. The subscripts at the node variables are introduced to distinguish the branches of the fanout nodes. The node label is interpreted as the signal variable of the corresponding gate: the variable x represents the output of a gate, and the variable x_j represents the j-th branch of the gate's fanout.

Denote by $RO \subseteq N$ the subset of all fan-out nodes which reconverge and by $RI \subseteq N$ the subset of all reconvergent fan-in nodes. To each $x \in RO$ we refer the set of nodes $RI(x) \subseteq RI$, so that for each $y \in RI(x)$ there exist at least two different converging paths from x to y.

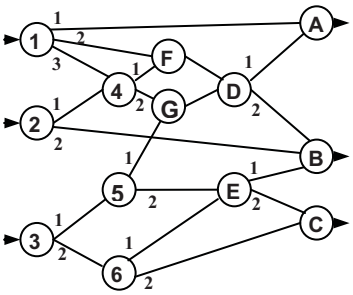


Figure 3. Reconvergency graph of a circuit

Consider in Fig. 3 a reconvergency graph which represents a topological skeleton of a circuit with primary outputs $\{A,B,C\}$.

During the topological analysis, all the paths in the circuit are traced, and the found reconvergencies are fixed in the form of subsets:

$RO = \{1,2,3,4,5,6\}$,
$RI = \{4,A,B,C,D,E,F,G\}$,
$RI(1) = \{A,D,F\}$, $RI(2) = \{B\}$,
$RI(3) = \{E, B,C\}$, $RI(4) = \{D\}$,
$RI(5) = \{B\}$, $RI(6) = \{C\}$.

Before creating a joint calculation model of the whole circuit for fault tracing purposes, the next step is to build an ordered set N^* of all the nodes in G. For reconvergency graph in Fig. 3 the following ordered set of nodes is constructed:

$N^* = (A,B,C,D,E,F,G,6,5,4,3,2,1)$.

C. Model creation procedure

Each edge (x_j,y) in the reconvergency graph G corresponds to a signal path in the circuit. Let us denote by the pair XY the formula of the Boolean derivative $\partial y/\partial x$. In this case, $\partial y/\partial x = 1$ iff flip of signal at x will also produce change at y. Thus the ultimate goal of the procedure of construction of calculation model is to build the formulas for calculation of $\partial y/\partial x$ for every node x in graph G and for each y that belongs to set of nodes representing primary outputs. Although the complete procedure of construction is described in [13] we will highlight the basic principles of the algorithm.

To calculate dependency of output of FFR block to its input the critical path tracing procedure in a reconvergent fanout is applied [12]. The latter corresponds to calculation of Boolean derivative $\partial y/\partial x_i$ where x_i is the i-th input of FFR with output y. For this case the formula x_jy is constructed where j is the index of the respective fanout branch which corresponds to the input x_i (in case if node X does not belong to the set RO this index can be omitted).

For instance, critical path tracing inside FFR could be used for calculating the dependency of output block A on the fault located at the first branch of fanout node *1* (see Fig. 3). For this purpose, we create the respective formula 1_1A and put it into computation model.

For Boolean derivative $\partial y/\partial z$ where the path between z and y consists of a simple chain of gates without reconvergencies we can build for the path a formula zy by the chain rule using logic AND operation of Boolean derivatives of the gates on the path:

$$\frac{\partial y}{\partial z_i} = \frac{\partial x_i}{\partial z_i} \wedge \frac{\partial y}{\partial x_i} \tag{4}$$

Here, z_i corresponds to i-th branch of fanout z that forms path to y. As an example, we can use this procedure to calculate the dependency of output of node B to the fault on output of node F (see Fig. 3) by construction of the following formula:

$$FB = F_1D \wedge D_2B$$

If two nodes z and y form a reconvergency, we use the formula obtained in [13]:

$$\frac{\partial y}{\partial z} = y \oplus F((x_1 \oplus \frac{\partial x_1}{\partial z_1}),...,(x_i \oplus \frac{\partial x_i}{\partial z_i}), x_j,...,x_n) \tag{5}$$

The subformulas $\partial x_i/\partial z_i$ in (5) are created step by step during path tracing. These formulas are used also for calculating the detectabilities of faults on the corresponding paths. We can calculate now for the given path *(z,y)* $\partial x_i/\partial z_i$ as a part of the formula (5), and then update the result according to (4) to get the values of $\partial y/\partial z_i$. In such a way we achieve in critical path tracing two goals: we calculate the activation of the faults on the paths up to the output of the gate with reconvergency, and up to the inputs of the same gate to be able to take into account the reconvergency effect.

In the case of the nested reconvergencies, we take them into account by superpositioning formulas (3) as shown in [13]. This operation follows also automatically during the backtracing analysis of the circuit.

Table I presents a part of computational model constructed for graph G in Fig 1. The formulas were created for nodes in the order of N* shown in the column "*Node*". The formulas z_iy and zy denote the derivatives $\partial y/\partial z_i$ and $\partial y/\partial z$, respectively. The notation $R_{zy}(z_1y,...,z_iy)$ is introduced to denote the formula (5), where the parameters z_jy represent the derivatives $\partial y/\partial z_i$, and i is the number of input of the gate y where the paths from fanout z reconverge at y along j-th fanout branch. The last rows of the table contain the formulas that unite the results of calculation of detectability of fanouts for all of primary outputs. For instance, the detectability of fault at node D can be expressed as a union of dependabilites of primary output A and B on signal flip at fanout D.

After creating the computational model, we proceed to the test pattern simulation phase. First, a subset of test patterns is simulated in parallel to determine the fault-free values of all signal lines in circuit. Second, based on these values and using the formulas in the computational model we determine which SAF faults are detected by this subset of test patterns.

TABLE I. COMPUTATIONAL MODEL FOR GRAPH G

#	Node	P	Formulas	#	Node	P	Formulas
1	A	1	1_1A	22	F	1	$1_2D=1_2F\wedge F_1D$
2	A	1	D_1A	23	F	1	$4_1D=4_1F\wedge F_1D$
3	B	*	D_2B	24	G	1	4_2G
4	B	1	2_2B	25	G	2	5_1G
5	B	2	E_1B	26	G	1	$4_2D=4_2G\wedge G_1D$
6	C	2	E_2C	27	G	2	$5_1A=5_1G\wedge G_1A$
7	C	2	6_2C	28	G	2	$5_1B=5_1G\wedge G_1B$
8	D	1	F_1D	29	6	2	6_13
9	D	1	G_1D	30	6	2	$R_{6C}(6_1C, 6_2C)$
10	D	1	$F_1A=F_1D\wedge D_1A$	31	5	2	3_15
11	D	1	$F_1B=F_1D\wedge D_2B$	32	5	2	$5B=R_{5B}(5_1B,5_2B)$
12	D	*	$G_1A=G_1D\wedge D_1A$	33	4	1	1_34
13	D	*	$G_1B=G_1D\wedge D_2B$	34	4	1	2_14
14	E	2	5_2E	35	4	1	$R_{4D}(4_1D, 4_2D)$
15	E	2	6_1E	36	4	1	$4A=R_{4D}\wedge D_1A$
16	E	2	$5_2B=5_2E\wedge E_1B$	37	4	1	$4B=R_{4D}\wedge D_2B$
17	E	2	$5_2C=5_2E\wedge E_2C$	…			
18	E	2	$6_1B=6_1E\wedge E_1B$		D	1	$D = D_1A \cup D_2B$
19	E	2	$6_1C=6_1E\wedge E_2C$		E	2	$E = E_1B \cup E_2C$
20	F	1	1_2F		F	1	$F = F_1A \cup F_2B$
21	F	1	4_1F	…			

III. MODEL PARTITIONING

The computational model constructed in the previous section allows to carry out fast efficient fault simulation of a circuit minimizing the number of repeated computations. The experimental results presented in Section V show that the proposed fault simulation method outperforms several commercial and academic tools. Nevertheless, the speed of simulation of very large designs on a single computer can be unacceptably slow even in case of efficient algorithms. In addition, the proposed technique requires certain amount of memory for storing the formulas used in the model. Again, for large circuits this requirement can exceed the amount of available memory. In the last case, the fault simulation cannot be performed for such circuit or the efficiency of fault simulator is extremely decreased.

To overcome these problems the method of splitting of computational model is proposed. By using the proposed approach it is possible to split the process fault simulation into a number of parallel sub-processes. Then for each sub-process, a partial computational model is constructed and the separate simulation procedure is run. Moreover, it can be easily seen, that such method gives an opportunity to use distributed environment for achieving higher simulation speed. Although, possible overlap between partial calculations may introduce certain costs, the overall performance of distributed simulation will overcome the speed of fault analysis on a single machine.

Let us define the set PI that is formed out of the nodes in G that are directly connected to primary inputs. Then $\pi(PI)$ denotes a partition of original set PI into non-overlapping subsets and B_i denotes an i^{th} subset of π. Then, $G'_i(N', U')$ is a subgraph of the graph $G(N, U)$ for which $N' \subseteq N$ and, $x \in N'$ only if there is a signal path from one of the primary inputs of B_i to x. Let us call G' as *partital reconvergency graph.* Then for each of the partial reconvergency graphs a separate

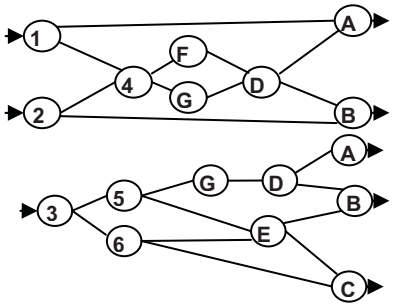


Figure 4. Partial reconvergency graphs G_1 and G_2

computational model is constructed and fault simulation is carried separately for each corresponding part of circuit.

For the reconvergency graph in Fig. 3 we have set $PI = \{1,2,3\}$. Let us define the partition π on the set PI as: $\pi = \{\overline{i_1, i_2}; \overline{i_3}\}$. The partial reconvergency graphs G_1 and G_2 are presented in Fig 4. The formulas belonging to the partial calculation model that corresponds to G_1 are marked by "*1*" in the columns "*P*" of Table I whereas formulas that correspond to G_2 are marked as "2" ("*" means that formula belongs to both models). Note that overlap is very likely to occur between formulas of partial computational models (see Table I).

Obviously, the effectiveness of the proposed method strongly depends on the initial partition of input nodes. As for the current implementation, no analysis is conducted to find the optimal selection of partition π for minimizing the size of overlapped area. Instead of that, π is selected randomly taking into account only the amount of available memory. For this purpose the algorithm that uses internal memory counter for keeping the currently allocated memory size stops the construction of partial calculation model when the maximally allowed amount of allocated memory is reached. The full description of algorithm is presented in [14].

IV. Distributed Simulation Environment

Our web-based infrastructure is built according to the client-server three-tier concept. There is a master server, several application servers and arbitrary number of users (Fig. 5). Master has a role of the mediator, it interacts both with users and simulation agents. Users and agents work in "polling" mode, whereas master is working in "answering" mode. Users can communicate with master only. Simulation agents reside on application servers. Agent consists of software layer wrapping the simulator tool and providing network communication abilities. On a request, agents will start instances of the simulator tool. Each user has own workspace in the server-side database, but large files are stored directly in file system for performance reasons – in the database only references to the file location are maintained.

At first, master server and agents must be started by system administrators. Invocation of the agents can be automated by use of system start-up scripts. Thereafter, users can submit tasks which are passed initially to the master and stored there until an idle agent will ask for a new task. When task is complete, agent passes the fault simulation results back to master, who assembles the partial fault tables, calculates the total fault coverage and stores the data. Results are delivered to user later when requested.

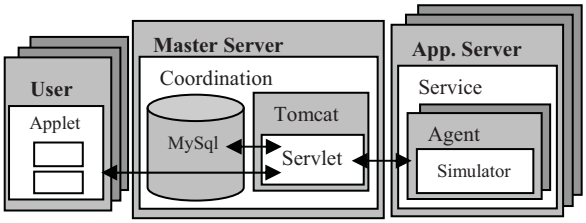


Figure 5. System components and communication

System components can be executed on different computing platforms, however the simulator instances must run on their native platform. Master servlet usually resides separately from agents. Moreover, master and agents can be located in different LANs. Firewall traversal is no problem as only one web server port must be configured on Master server.

A. Implementation

Web-based infrastructure is built on Java Applet/Servlet technology [24] and popular platform independent open source relational database MySQL[25]. Communication flow between system components and implementation details can be seen in Fig. 5. Servlet is a Java application that runs in a Web server or special application server and provides server side processing like different calculations, database access, e-commerce transactions, etc. Servlets are designed to handle HTTP requests and are the standard Java replacement for a variety of other methods, including CGI scripts, Active Server Pages (ASPs) and proprietary C/C++ plug-ins for specific Web servers (ISAPI). Because servlets are written in Java, they are portable between servers and operating systems. The servlet programming interface (Java Servlet API) is a standard part of the Java EE (Enterprise Edition of Java), the industry standard for enterprise Java computing. Tomcat[26] is open source servlet container (application server software) which is one way to run Java Servlets. Tomcat is developed by the Apache Software Foundation (ASF) and implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a pure Java HTTP web server environment to run a Java code.

In conclusion, Tomcat and servlets running on it play important role in order to access our intranet resources on application servers and the MySQL database. It is simple and light weight alternative to other full blown enterprise scale solutions.

Tool encapsulation

Our simulator is implemented in C language, it has no graphical interface and network communication abilities. In order to integrate it into web based environment, it is necessary to implement additional software (wrapper) layer. Simulator will be invoked from Java program (Agent), which allows to adapt the input data, convert the tool-specific data,

simulation results (log files, test vectors, etc), map the control information to the embedded tool, transfer and pass the status information (warning and error messages) to be submitted to the user, etc. Technically simplest way is to encapsulate a tool as an entire program. Tool has to be able to run as a batch job. Integration of other tools is then also possible similar way. Also embedding of a library (e.g. C, C++ routines) via the Java Native Interface (JNI) could be promising and also direct integration of Java-classes and applications (especially for Java software).

Data management

Data handling takes place in coordinator servlet. Problem is that web-based HTTP communication is stateless and session is valid for short time only, but simulation process may run much longer. Therefore, users must be identified, their tasks and results must be stored for later access.

Data module has three layers: presentation (user interface), business-logic (database queries, data processing) and physical database. First two layers are implemented in Java. User is accessing database only via presentation layer, which consists of several functions to run middle layer queries. Database access is implemented according to Data Access Object (DAO) design practice. Data access objects manage access to relational databases. For each table in a relational database there corresponds one Java class. Database table attributes map to Java class properties. For each property, there exist 'set' and 'get' methods. Additionally, DAO class has methods to insert, update and query the records in the database tables. For example, for table "Tasks", we will have at least properties like 'taskId', 'userId' and 'status'; methods could be like 'getTaskId', 'setStatus', 'insertTask', 'getCompletedTask', etc. Standard Java mechanism for accessing databases is using Java Database Connectivity (JDBC) API. For convenience purposes, we have captured basic DB connection code into single DB access class and every specific DAO class, like 'TasksDAO' class, extends that class – i.e. basic connection methods are inherited and used inside the class.

Alternatively, it could be possible to use popular Hibernate [21] and Spring [22] frameworks to simplify objects to relational DB mapping (ORM). For large and mission critical projects also Java EE technology like Enterprise JavaBeans is available[23]. However, for simple data persistency in current situation, proposed solution is adequate and was faster to implement. Setting up and closing DB connections is time consuming operation, therefore we have used Tomcat's native connection pooling to speed up DB transactions.

Communication

Use of applet/servlet approach means that general communication is based on HTTP protocol. The tools on different computers and on different computing platforms (UNIX, Linux, Windows) can easily exchange data as serialized Java objects. Data passing between components is implemented following Transfer Object (TO) design practice. Transfer object is a lightweight version of DAO object, it has only properties and 'get' and 'set' methods. Information is sent as data bundle as opposed to single strings.

HTTP protocol allows us also easy firewall traversal as we can use default web server port and Java servlet extensions on web servers as sort of proxies in order to reach intranet resources. There is no need for opening extra ports in the firewall on the user side as it is the case in TCP/IP socket based communication or when relying on Java RMI (which would be major restriction). Communication can be secured via SSL encryption by appropriate modifications in Tomcat configuration file, when necessary.

Graphical User interface

User interface (GUI) is based on Java Applet, which can be integrated into HTML page when needed. Java applets are very versatile in features and easy to develop. For rapid prototyping we have used NetBeans IDE[27], which supports visualised GUI development with drag and drop operations. Final tweeks to generated code still had to be done manually.

User GUI has fields to gather test tool's parameters, allows browsing for circuit model file, has button to start the tool, a console window to display all the messages from the running tool. When the task is complete, results download is enabled. User can browse and select the folder where to save results. Since local hard drive access for usual Java applets is restricted for security reasons, then GUI applet had to be signed digitally. We used so called self-signed certificate for simplicity. Certificate shows owner specific information. Only difference for end user is that when signed Applet is first time downloaded into user's computer, informative dialog box is displayed. It is user's responsibility to trust or untrust the origin and contents of the Applet. User can contact Applet owner about authenticity of certificate, when question arises. User needs the Java Runtime Environment (JRE) to run GUI.

B. *Workflow of distributed simulation*

At first, user specifies the parameters and source design for the simulation tool (see Fig. 5). In addition, the size of the simulation task can be predefined. Thereafter GUI module contacts with master server and circuit along with the parameters are passed to it automatically. The task coordinator process on master records all requests from user(s) to the DB. Test agents poll constantly the Master and if any of subtasks is scheduled by coordinator process, the agents receive the corresponding parameters and circuit file along the test patterns for starting native simulator tool.

In the beginning, simulator constructs calculation model taking into account the memory limit of the subtask. While reaching the limit, the simulator saves the breakpoint information into local file system. Simulation agent reads the breakpoint information and passes it to the Master server where it will be stored for other simulation agents. When next idle agent is polling, it will get the circuit model along with parameters and breakpoint information. The new instance of simulator started by the agent, constructs calculation model starting from the breakpoint hence, the updated breakpoint

information will be saved and passed to Master server by the agent. Simulator agents wait until their subtasks will be completed and report results back to Master server.

The process repeats until there are no subtasks left. Note that simulators need to be started subsequently but after that they run concurrently (the starting delay is small compared to runtime). However, finishing order of simulators may not be the same as starting order as simulation speed depends on the piece of the calculation model: some parts are more difficult to simulate. After all simulators are finished, the Master server assembles sub-results into the final result and stores it in the database. Then final result is passed to user when requested.

For each simulator there is a dedicated agent that must reside on the same computer. In case of multiprocessor computer, it is possible to run several agents and simulators concurrently on the single computer. Simulation agent will accept only one task at time. It is reasonable to have one agent for each processor because operating system typically assigns running tasks to available processors.

V. Experimental Results

Table II presents the single processor fault simulation results for the large circuits of three benchmark sets: ISCAS'85 and combinational versions of ISCAS'89 and ITC'99 (column 1). The second column shows the size of each circuit (number of equivalent 2-input gates). Following columns present the simulation results for various fault simulators: the approach described in [11] (column 3), two commercial fault simulators from major CAD vendors C1 and C2 and the proposed method (column 6). The last row shows the average speed gain in comparison with other methods.

TABLE II. Simulation results (no model partitioning)

Circuit	Gates	Fault simulation time, s			
		[11]	C1	C2	New
c1908	618	640	12	2.97	0.36
c2670	883	560	24	2.24	0.4
c3540	1270	770	43	7.48	0.9
c5315	2079	1270	57	5.55	0.76
c7552	2632	1480	88	8.14	1.17
s13207_C	3214	N/A	70	5.64	2.03
s15850_C	3873	N/A	111	12.06	2.63
s35932_C	12204	N/A	390	23.63	5.73
s38417_C	9849	N/A	310	31.44	6.85
s38584_C	13503	N/A	320	23.22	6.37
b14	9150	N/A	N/A	49.24	14.01
b15	8877	N/A	N/A	39.06	25.79
b17	31008	N/A	N/A	117.64	75.40
b18	104580	N/A	N/A	620	344.5
b19	210585	N/A	N/A	1353	750.1
Average speed gain		1393	53.3	4.4	1

The simulation was carried out for the sets of 10000 random patterns without usage of fault dropping (full fault table constructed). The experiments were run on a 1500MHz SUN UltraSparc IV+ server with Solaris 10 operating system, except the experiments data taken from [11] that were obtained on a 2.8GHz Pentium 4 under Windows XP.

TABLE III. Analysis of circuits structure

Circuit	Total #fanouts	Max depth	Fanouts for input	Fanout %
c1908	223	16	109	49
c2670	290	16	115	40
c3540	356	16	246	69
c5315	510	16	138	27
c7552	812	15	661	81
s13207_C	1224	16	131	11
s15850_C	1518	25	260	17
s35932_C	5295	10	1324	25
s38417_C	4569	16	233	5
s38584_C	3946	19	253	6
b14	2409	44	2023	84
b15	2353	58	1392	59
b17	8145	71	1518	19
b18	31066	79	6406	21
b19	63095	82	11541	18

Table III presents detailed characteristics of the circuits. Column 3 shows maximal depth (in fanouts) for each circuit. It can be seen that the maximal depth does not grow as fast as the number of fanouts (thus circuits are growing more in width dimension than in depth). Column 4 shows maximal count of fanouts that are driven (immediately or indirectly) by a single primary input thus roughly estimates the minimal size of slice we can get when dividing the calculation model (by current algorithm). This value also constraints the granularity of divided parts and defines a minimal amount of memory required for fault simulation. Again for many circuits the ratio of minimal slice to total fanout count decreases with the growing size of circuits (see last column). However there are exceptions (e.g. b14 and b15 benchmarks).

In experiments with distributed solution we measured the overall speed-up, memory reduction and communication overhead in order to determine how well the current task partitioning solution scales when the number of processing units increases. Simulation was carried out on the same UltraSPARC servers. Tomcat servlet engine and MySQL DB were running on 2-core AMD Athlon 64 6000+ 3GHz processor with 2GB memory. User applet was also executed on the similar Athlon machine. Circuit loading takes about a second for the input files on the user computer. File transfer to the database and user notification takes about 6 seconds. Thereafter, simulation agent receives files from Master server with 4-5 seconds delay. The total communication delay was approximately 12-16 seconds in case of distributed web-based solution. The total communication overhead was about 1% compared to single processor solution in case of largest circuits. The overhead depends on the size of the circuit and the number of test vectors simulated.

Distributed fault simulation results are presented in Table IV. 100K test patterns were applied to each circuit. As we can see, model build time for subtask (circuit slice) is very small (0.1% for b18 circuit) compared to simulation time. Final simulation time is dominated by the longest subtask simulation time. We see that there is some deviation from ideal mean time. This implies that model partitioning could be

still improved - model slices could be more balanced in size. This would lead to more equal and shorter simulation times and user would get final result faster. However, the possibilities of balancing the partitioning of the model depend essentially on the circuit structure.

TABLE IV. DISTRIBUTED SIMULATION RESULTS

Circuit	B17C	B18C	B21C	B22C
Max model partitions	13	8	12	13
Max model build, s	0.24	1.83	0.32	0.37
Max subtask simul., s	214	1534	146	195
Subtask simul. deviation, %	21.0	24.4	15.7	5.5
Model size reduction, x	4.1	2.8	2.5	2.6
Speedup by model partition	3.2	6.4	2.5	2.9
Speedup by test partition	10.3	7.9	8,7	10.0

The last rows of Table IV present simulation speed-up for the simulation distributed on several processors compared to single processor local simulation. Scalability in case of model partitioning is degrading due to model pieces overlapping. For the purpose of fair comparison, the speedup results in rows 6 and 7 are calculated for the same number of partitions (first row in Table IV) for both types of partitioning. It is interesting to see that in case of larger circuit b18 model partitioning speedup is quite close to test partitioning speedup. Figure 6 shows that initially, up to 6 processors model partitioning has an advantage compared to test partitioning. Integrated speedup for circuit b18 compared to single processor local simulation can be observed in figure 7. Using for example 8 processors (2 processors for model partitioning dimension and 4 processors for test partitioning dimension) would lead 15.6 time integrated speedup which is considerably better than just using model partitioning (4.1x) or test partitioning (7.9x).

Comparing simulation speed to GPU based solution in [20], our approach would require 5 processors to get similar result for b22 circuit (27599 gates, 32K vectors).

VI. CONCLUSIONS

Web-based distributed fault simulation approach has been proposed in this paper. In contrast to existing solutions, we have developed Internet based loosely coupled system, which potentially allows seamlessly aggregate computers of dislocated working groups into one powerful simulation application. Model partitioning has been proved to be useful as it allowed to speed up the simulation up to 6.4 times and at the same time to reduce the required memory amount 2.8 times on 8 processors compared to single processor simulation in case of the largest circuit b18. Model partitioning is able to outperform the test partitioning when number of processors is small. Further speedups (15.6 times for 8 processors, for example) can be achieved by combining model and test set partitioning. Model partitioning clearly helps to boost the simulation speed. Our approach favors larger circuits. Design pattern proposed in current paper can be easily used for other distributed applications, only task partitioning is specific.

REFERENCES

[1] Han and Soo-Young Lee, "A Parallel Implementation of Fault Simulation on a Cluster of Workstations," in *Proc. IEEE International Symposium Parallel and Distributed Processing IPDPS*, 2008

[2] E. M. Rudnick and J. H. Patel, "Overcoming the serial logic simulation bottleneck in parallel fault simulation," in *Proc. 10th Int. Conf. VLSI Design*, 1997, pp. 495-501.

[3] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano,and J. A. Abraham, "Portable parallel logic and fault simulation," in *Proc. Int. Conf. CAD*, 1989, pp. 506-509.

[4] J. F. Nelson, "Deductive fault simulation on hypercube multiprocessors," in *Proc. 9th ATT Conf. Electronic Testing*, 1987.

[5] S. Patil, P. Banerjee, and J. Patel, "Parallel test generation for sequential circuits on general purpose multiprocessors," in *Proc. 28th ACM/IEEE Design Automation Conf.*, San Fransisco, CA, 1991.

[6] S. Ghosh, "NODIFS: A noval, distributed circuit partitioning based algorithm for fault simulation of combinational and sequential digital designs on loosely coupled parallel processors," LEMS, Division of Engineering, Brown University, Providence, RI, Tech. Rep., 1991.

[7] P. Agrawal and V. D. Agrawal, K. T. Cheng, and R. Tutundjian, "Fault simulation in a pipelined multiprocessor system," in *Proc. Int. Test Conf.*, 1989, pp. 727-734.

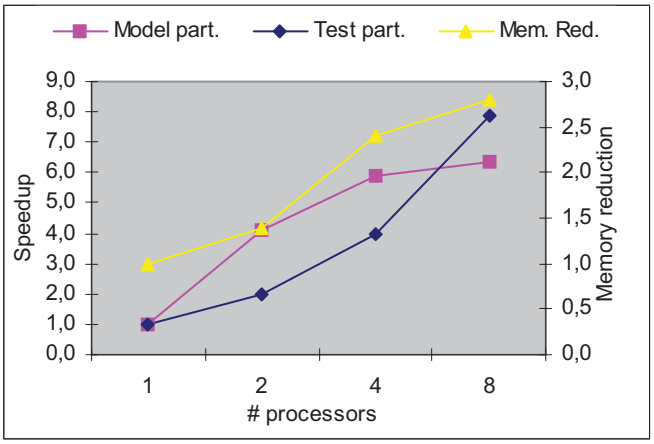


Figure 6. Circuit b18 experimental results

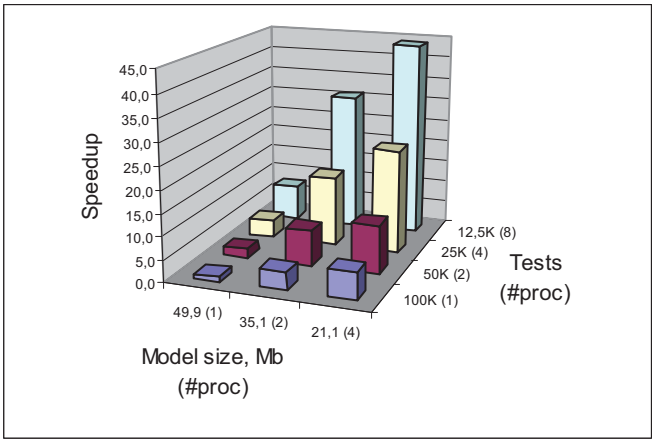


Figure 7. Integrated speedup for circuit b18

[8] S. Bose and P. Agrawal, “Concurrent fault simulation of logic gates and memory blocks on message passing multicomputers,” in *Proc. Design Automation Conf.*, 1992, pp. 332-335.

[9] M. B. Amin and B. Vinnakota, "Data Parallel-Fault Simulation,"*IEEE Trans. VLSI Systems*, vol. 7, no. 2, pp. 183-190, Jun. 1999.

[10] M. Abramovici, P.R. Menon and D.T. Miller, “Critical Path Tracing - an Alternative to Fault Simulation,” in *Proc. 20th Design Automation Conf.*, 1983, pp. 214-220.

[11] L. Wu and D.M.H. Walker, “A Fast Algorithm for Critical Path Tracing in VLSI”, in *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2005, pp.178-186.

[12] S. Devadze, J. Raik, A. Jutman and R. Ubar, “Fault Simulation with Parallel Critical Path Tracing for Combinational Circuits Using SSBDDs”, in *Proc. 7th IEEE LATW Conf.*, 2006, pp.97-102.

[13] R. Ubar, S. Devadze, J. Raik and A. Jutman, “Parallel Fault Backtracing for Calculation of Fault Coverage”, in *Proc. 13th Asia and South Pacific Design Automation Conference (ASPDAC)*, Korea, 2008, pp. 667-672.

[14] J.Raik and R.Ubar, “Feasibility of Structurally Synthesized BDD Models for Test Generation,” in *Proc. European Test Workshop*, Barcelona, 1998, pp. 145-146.

[15] S. Devadze, R. Ubar, J. Raik and A. Jutman, “Parallel Exact Critical Path Tracing Fault Simulation with Reduced Memory Requirements,” in *Proc. 4th IEEE Int. Conf. Design & Technology of Integrated Systems in Nanoscale Era*, Cairo, Egypt, 2009.

[16] A. Schneider et. al. “Internet-based Collaborative Test Generation with MOSCITO,” in *Proc. DATE*, Paris, France, 2002, pp. 221-226.

[17] E. Ivask, J. Raik, R. Ubar and A. Schneider, “WEB-Based Environment: Remote Use of Digital Electronics Test Tools,” in *Virtual Enterprises and Collaborative Networks*, Kluwer Academic Publishers, 2004, pp. 435-442.

[18] BOINC. http://boinc.berkeley.edu/

[19] Y.M. Teo and X. B. Wang, “AliCE: A Scalable Runtime Infrastructure for High Performance Grid Computing,” in Proc. *IFIP Int. Conf. Network and Parallel Computing*, Springer-Verlag Lecture notes in Computer Science, Wuhan, China, October 2004.

[20] K. Gulati, S. P. Khatri, “Towards Acceleration of Fault Simulation using Graphics Processing Units,” in *Proc. DAC, Anaheim*, California, 2008.

[21] C. Bauer and G. King,*Hibernate in Action*. Manning Publications, 2004

[22] C. Walls, *Spring in Action*, Third Edition. Manning Publications, 2011.

[23] D. Panda, R. Rahman and D. Lane, *EJB 3 in Action.* Manning Publications, First Edition, 2007.

[24] Java Servlet Technology. http://java.sun.com/products/servlet/overview.html

[25] Open source database MySQL. http://www.mysql.com/why-mysql/

[26] Apache Tomcat. http://tomcat.apache.org/

[27] NetBeans IDE. http://netbeans.org/features/

[28] Ian Foster. “Globus Toolkit Version 4: Software for Service-Oriented Systems”, *Journal of Computer Science and Technology,v*ol. 21, no.4, pp. 513-520, Jul. 2006.

[29] D. Booth, H. Haas, F. McCabe et. al., “Web Services Architecture,” W3C Working Group Note, 2004. http://www.w3.org/TR/ws-arch/

[30] TeraGrid. http://www.teragrid.org/about/

[31] Open Science Grid. http://www.opensciencegrid.org/

[32] Large Hadron Collider (LHC) Computing Grid. http://public.web.cern.ch/public/en/lhc/Computing-en.html

[33] Java Jini Technology. http://www.jini.org/wiki/

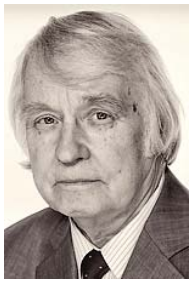
[34] JavaSpaces. http://www.jini.org/wiki/JavaSpaces_Specification

[35] Simple Object Access Protocol (SOAP). http://www.w3.org/TR/soap/

[36] M. Abramovici, M.A.Breuer and A.D. *Friedma, Digital systems testing and testable design*. IEEE Press, 1990

Eero Ivask has received his M.Sc. and Ph.D. degrees in computer engineering from Tallinn University of Technology, Estonia in 1998 and 2006 respectively and currently holds the position of researcher in the same university. His primary research interests include fault simulation, test generation, web based systems, distributed computing. He is a co-author of more than 30 scientific papers in international conference proceedings.

Sergei Devadze has received his M.Sc. and Ph.D. degrees in computer engineering from Tallinn University of Technology, Estonia in 2004 and 2009 respectively and currently holds the position of researcher in this university. His primary research interests embrace such topics as fault simulation, fault modeling, extended board-level test, decision diagrams and decomposition of finite-state machines. He is a co-author of over 25 scientific papers in the field of digital design and test published in international journals and refereed conference proceedings.



Raimund Ubar is a professor of computer engineering at Tallinn Technical University in Estonia. He received his PhD degree in 1971 at the Bauman Technical University in Moscow. His main research interests include computer science, electronics design, test generation, design for testability, fault-tolerance. He has published more than 300 papers and 4 books. R. Ubar has lectured as a visiting professor in more than 25 universities in 10 countries. He is a member of Estonian Academy of Sciences and a Golden Core member of the IEEE Computer Society.