Integrated Design and Test Generation under Internet-based Environment MOSCITO

A. Schneider, K.-H. Diener
Fraunhofer Institute for Integrated Circuits Branch Lab Design Automation (Germany)

E. Ivask, R. Ubar
Tallinn Technical University (Estonia)

E. Gramatova
Institute of Informatics (Slovak Republic)

T. Hollstein
Technical University of Dortmund (Germany)

W. Kuzmicz
Warsaw University of Technology (Poland)

Z. Peng
Linköping University (Sweden)
Outline

Introduction and motivation: Test generation and fault simulation variety of tools, complex workflow

What are the problems? platforms, performance, installation and administration

Our solution: MOSCITO basic idea, fundamental concept, implementation

Current Results: Collaborative work with MOSCITO VILAB workflow works between four European cities

Summary
Outline

Introduction and motivation: Test generation and fault simulation
variety of tools, complex workflow

What are the problems?
platforms, performance, installation and administration

Our solution: MOSCITO
basic idea, fundamental concept, implementation

Current Results: Collaborative work with MOSCITO
VILAB workflow works between four European cities

Summary
Starting Point: FPGA Design Flow

Synthesis with Synopsys

FPGA on board

Test pattern generation and fault simulation play an important role in FPGA design.
The C2VHDL Code Migration Tool is designed to convert C code to VHDL, enabling designers to use C specifications for hardware synthesis. The process involves the following steps:

1. **Conversion from C to VHDL**
2. **CST Analyzer**: Parses C processes and stores them in a C Structure Tree (CST).
3. **CST Object Library**: Stores CST data in a library.
5. **VHDL Backend**: Generates VHDL specification.

This tool facilitates the migration of C code to VHDL, streamlining the process of hardware synthesis and ensuring that designers can leverage their existing C specifications.
High-Level Synthesis System CAMAD

- uses ETPN (Extended Time Petri Net) as internal unified design representation
- RTL hardware implementation is generated (data path + control part)
- final RTL implementation is converted into structural VHDL (can be used for simulation)
Turbo Tester - A Set of Low-level ATPG Tools

- Turbo Tester tools operate on the SSBDD model (Structurally Synthesized Binary Decision Diagrams)

- set of tools for Automatic Test Pattern Generation (ATPG), Built-In Self-Test (BIST) emulation, fault simulation, test set optimization and multi-valued simulation
Defect-oriented ATPG with DefGen

- hierarchical ATPG for combinational circuits for $I_{DDQ}$ and voltage testing
- random and deterministic TPG and fault simulator are involved

ISCAS Netlists

- Random TPG
- Deterministic TPG
- Fault Simulation

possible faults, erroneous functions, input patterns, probabilities
Outline

Introduction and motivation: Test generation and fault simulation variety of tools, complex workflow

What are the problems? platforms, performance, installation and administration

Our solution: MOSCITO basic idea, fundamental concept, implementation

Current Results: Collaborative work with MOSCITO VILAB workflow works between four European cities

Summary
What are the problems?

Many tools from industry and academia are in use.

Synopsys™
Synthesis
Test Pattern Generation
TurboTester (Tallinn)
Test Pattern Generation
DefGen (Bratislava)
Translators
EDIF-SSBDD (Tallinn)
other commercial
design tools ...
What are the problems?

Many tools from industry and academia are in use.

Installation and administration cost a lot of man power.

- Synopsys
- Test Pattern Generation
- TurboTester (Tallinn)
- Test Pattern Generation
- DefGen (Bratislava)
- Translators
- EDIF-SSBDD (Tallinn)
- other commercial design tools ...
What are the problems?

Many tools from industry and academia are in use.

Installation and administration cost a lot of man power.

Time and money are wasted for studying documentations and trainings.
What are the problems?

Several hardware architectures are utilized.
What are the problems?

Several hardware architectures are utilized.

Appropriate resources are required for different design tasks.
What are the problems?

Several hardware architectures are utilized.

Appropriate resources are required for different design tasks.

Tools are running on different operating systems.
Introduction and motivation: Test generation and fault simulation. Variety of tools, complex workflow.

What are the problems? Platforms, performance, installation and administration.

Our solution: MOSCITO. Basic idea, fundamental concept, implementation.

Current Results: Collaborative work with MOSCITO. VILAB workflow works between four European cities.

Summary.
Basic Idea of MOSCITO

Available tools are applied.

- EDIF2SSBDD
- TurboTester
- Tst2Alb
- ALB
- DefGen
- EDIF2ISCAS
Basic Idea of MOSCITO

Available tools are applied.

Tools can be assembled into workflows as requested.
Basic Idea of MOSCITO

Available tools are applied.

Tools can be assembled into workflows as requested.

Coupling of different tools via Internet.
Available tools are applied.

Tools can be assembled into workflows as requested.

Coupling of different tools via Internet.

MOSCITO front end allows controlling the workflow and visualization of results.
Fundamental Concept
Each tool has to be encapsulated.
Each tool has to be encapsulated.

MOSCITO Agent provides a uniform interface for communication.
Each tool has to be encapsulated.

MOSCITO Agent provides a uniform interface for communication.

MOSCITO infrastructure implements data transfer via TCP/IP sockets.
Each tool has to be encapsulated.

MOSCITO Agent provides a uniform interface for communication.

MOSCITO infrastructure implements data transfer via TCP/IP sockets.

Data exchange is based on XML and Java serialization.
Fundamental Concept

Each tool has to be encapsulated.

MOSCITO Agent provides a uniform interface for communication.

MOSCITO infrastructure implements data transfer via TCP/IP sockets.

Data exchange is based on XML and Java serialization.

XML or Java serialized objects

MOSCITO Channel (TCP/IP socket based)

MOSCITO Datagram
Tool Encapsulation

The agent receives the input data and passes them to the embedded tool.
Tool Encapsulation

The agent receives the input data and passes them to the embedded tool.

Agent starts the tool and waits for result data.
Tool Encapsulation

The agent receives the input data and passes them to the embedded tool.

Agent starts the tool and waits for result data.

After conversion/serialization the result data will be sent to the following agent.
Tool Encapsulation

The agent receives the input data and passes them to the embedded tool.

Agent starts the tool and waits for result data.

After conversion/serialization the result data will be sent to the following agent.
Tool Encapsulation

The agent receives the input data and passes them to the embedded tool.

Agent starts the tool and waits for result data.

After conversion/serialization the result data will be sent to the following agent.
Tool Encapsulation

The agent receives the input data and passes them to the embedded tool.

Agent starts the tool and waits for result data.

After conversion/serialization the result data will be sent to the following agent.

Diagram:

- MOSCITO Agent
- COM
- Input data
- Result data
- Conversion
- Serialization
- Control information from user’s front end
Software Architecture

MOSCITO is implemented in Java. (Sun J2SE 1.4.x)
MOSCITO is implemented in Java.
(Sun J2SE 1.4.x)

Kernel provides basic functionality.
Software Architecture

MOSCITO is implemented in Java. (Sun J2SE 1.4.x)

Kernel provides basic functionality.

Using the open interfaces (API) users can add new components (agents, workflows, viewers).
MOSCITO is implemented in Java. (Sun J2SE 1.4.x)

Kernel provides basic functionality.

Using the open interfaces (API) users can add new components (agents, workflows, viewers).
MOSCITO is implemented in Java. (Sun J2SE 1.4.x)

Kernel provides basic functionality.

Using the open interfaces (API) users can add new components (agents, workflows, viewers).

Software Architecture

Operating system with Java Virtual Machine (Windows, Solaris, Linux, ...)

MOSCITO Kernel

Cycle4

Chain3

Singleton

VILAB

Workflow

TurboTester

C2VHDL

DefGen

Synopsys

Agent

...
MOSCITO is implemented in Java.
(Sun J2SE 1.4.x)

Kernel provides basic functionality.

Using the open interfaces (API) users can add new components (agents, workflows, viewers).
Implementation of agents (wrapper classes)

```java
public interface ATPG extends MoscitoAgent {
    public void configure()
    {
        ...;
    }

    public void initialize()
    {
        ...;
    }

    public MoscitoDatagram service(MoscitoDatagram data)
    {
        ...;
    }

    public void stop()
    {
        ...;
    }

    public void terminate()
    {
        ...;
    }
}
```
Outline

Introduction and motivation:
Test generation and fault simulation
variety of tools, complex workflow

What are the problems?
platforms, performance, installation and administration

Our solution: MOSCITO
basic idea, fundamental concept, implementation

Current Results: Collaborative work with MOSCITO
VILAB workflow works between four European cities

Summary
Internet-based collaborative test generation between partners from four European cities.

Each partner has to maintain their own tools.

Each partner can profit from all tools within the needed workflow.
MOSCITO

Front End
Front End

Browser
- Workflows
- Agents
Front End

Browser
- Workflows
- Agents

Scope
- Chart
- Text
- Image
Front End

Browser
- Workflows
- Agents

Scope
- Chart
- Text
- Image

Console
- Infos
- Errors
All available agents are listed in Browser window.
All available agents are listed in Browser window.

User can select the needed agents and can configure them via dialog windows.
Infos, warnings and error messages will be sent directly to Console window.

User can observe the tool status.
Each tool (agent) sent results to the database of front-end.

User can explore result data with Scope window.

Results can be stored to local file system.
Each tool (agent) sent results to the database of front-end.

User can explore result data with Scope window.

Results can be stored to local file system.
Summary

Test generation is based on various tools (pattern generators, fault simulators, translators).

Tools run on different platforms. Appropriate resources are needed.

MOSCITO provides an infrastructure for an Internet-based tool integration. Partners are enabled to cooperate remotely.

The maintenance costs for users are minimal. Test generation is available as a set of services.

Users with minimal knowledge in test engineering and tools involved there, can carry out complex test generation tasks.

MOSCITO Front End allows to observe the state of flow execution and results (files, images, numerical data).
Future work

Extending MOSCITO communication protocol for tunneling firewalls. (using proxies and/or HTTP)

Further tools will be integrated in MOSCITO to support more complex workflows.

The GUI of MOSCITO front end application will be improved (drag & drop, workflow editor, project manager will be added).

Direct communication between MOSCITO users will be supported (exchange of result data, chat rooms, ...).
Further information

http://vilab.dcs.elf.stuba.sk

http://ups.savba.sk/diag

http://www.eas.iis.fhg.de/solutions/moscito

http://www.pld.ttu.ee/tt
## Other Approaches and Architectures

<table>
<thead>
<tr>
<th>Approach</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CORBA</strong></td>
<td>• very complex</td>
</tr>
<tr>
<td><strong>Java RMI</strong></td>
<td>• could be used for communication and naming service</td>
</tr>
<tr>
<td><strong>COM/DCOM</strong></td>
<td>• available for Microsoft platform only</td>
</tr>
<tr>
<td><strong>.NET</strong></td>
<td>• under development</td>
</tr>
<tr>
<td><strong>HLA</strong></td>
<td>• targeted to simulation</td>
</tr>
<tr>
<td><strong>TENT (DLR)</strong></td>
<td>• similar to MOSCITO approach</td>
</tr>
<tr>
<td><strong>Lavana et al</strong></td>
<td>• fine-grained workflow concept</td>
</tr>
<tr>
<td><strong>Chamois (Won Kim et al)</strong></td>
<td>• ...</td>
</tr>
<tr>
<td><strong>James (Uhrmacher)</strong></td>
<td>• targeted to simulation of Multi Agent Systems</td>
</tr>
<tr>
<td><strong>Astai(R) (C-Lab)</strong></td>
<td>• CORBA-based</td>
</tr>
</tbody>
</table>