Deductive Fault Simulation for Asynchronous Sequential Circuits

Roland Dobai
Elena Gramatová

roland.dobai@savba.sk

Institute of Informatics,
Slovak Academy of Sciences, Slovakia

Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science,
November 13 – 15, 2009, Znojmo, Czechia
Outline

1. Introduction
2. Related work
3. Serial fault simulation
4. Deductive fault simulation
5. Results
6. Conclusion
Introduction
Asynchronous circuits

- Asynchronous circuits are fundamentally different from their synchronous counterparts.
- They do not use clock signal for synchronization.
Introduction
Asynchronous circuits

Asynchronous circuits are fundamentally different from their synchronous counterparts.
They do not use clock signal for synchronization.

This difference allows us to achieve:

- lower power consumption,
- higher operating speed,
- less electro-magnetic noise emission,
- better modularity,
- better robustness.

C-elements are used in the asynchronous circuits as basic construction elements.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>previous value</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>previous value</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

C-element — symbol and truth table
Introduction

Classes of the asynchronous circuits

Asynchronous circuits can be classified at the gate level as

- self-timed,
- speed-independent,
- delay-insensitive,
- quasi-delay-insensitive.
Introduction
Classes of the asynchronous circuits

Asynchronous circuits can be classified at the gate level as

- self-timed,
- speed-independent,
- delay-insensitive,
- quasi-delay-insensitive.

In this work only the speed-independent circuits are considered.

Other classes of asynchronous circuits can be represented under this model Shi et al.: Test Generation for Ultra-High-Speed Asynchronous Pipelines — [2].
Developed and implemented a **deductive fault simulator** for stuck-at faults of speed-independent circuits.
Introduction

About this work

- Developed and implemented a deductive fault simulator for stuck-at faults of speed-independent circuits.
- Implemented serial fault simulator.
  - Based on Shi et al.: SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits — [3].
  - Reason of the implementation: to compare the results with the deductive fault simulator in the same environment.
Introduction
About this work

- Developed and implemented a **deductive fault simulator** for stuck-at faults of speed-independent circuits.
- Implemented serial fault simulator.
  - Based on *Shi et al.: SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits* — [3].
  - Reason of the implementation: to compare the results with the deductive fault simulator in the same environment.
- Random test pattern generator was implemented based on *Shi et al.: Fault Simulation and Random Test Generation for Speed-Independent Circuits* — [4].
Fault simulation of the asynchronous circuits is more difficult.
Introduction
Fault simulation

Fault simulation of the asynchronous circuits is more difficult.

It needs to deal with

- hazards (unwanted glitches on signals),
- oscillations (feedback signals never stabilize),
- races (two or more feedback signals are changing simultaneously).

Shi et al.: SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits — [3]
Related work
Fault simulators

**SPIN-SIM** *Shi et al.: SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits — [3]*

- Serial fault simulator for the speed-independent circuits.
- Uses the 13-valued logic and the time stamps for hazard detection.
- Time stamps — determination of difference between signal transitions and hazards.
- Transforms circuits into combinational representation:
  - replaces C-elements with a set of simple gates,
  - cuts each feedback path into pseudo-primary inputs and outputs.
Related work
Fault simulators

**Related work**

**Fault simulators**

**SPIN-SIM** Shi et al.: *SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits* — [3]
- Serial fault simulator for the speed-independent circuits.
- Uses the 13-valued logic and the time stamps for hazard detection.
- Time stamps — determination of difference between signal transitions and hazards.
- Transforms circuits into combinational representation:
  - replaces C-elements with a set of simple gates,
  - cuts each feedback path into pseudo-primary inputs and outputs.

**Fsimac** Sur-Kolay et al.: *Fsimac: A fault simulator for asynchronous sequential circuits* — [5]
- Uses the 13-valued logic.
- Can not be used for the speed-independent circuits.
Serial fault simulation was implemented based on Shi et al.: SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits — [3].

Simulation of the faulty circuit with inserted faults one-by-one for the current test pattern.
Serial fault simulation

- Serial fault simulator was implemented based on Shi et al.:

  *SPIN-SIM: Logic and Fault Simulation for Speed-Independent Circuits* — [3].

Serial fault simulation

- Simulation of the faulty circuit with inserted faults one-by-one for the current test pattern.
Deductive fault simulation is not a new technique.

**Deductive fault simulation**

- Creation of detectable faults lists and their propagation to outputs for the current test pattern in a single simulation overpass.
Deductive fault simulation

- Deductive fault simulation is not a new technique.

**Deductive fault simulation**

- Creation of detectable **faults lists** and their propagation to outputs for the **current test pattern** in a **single simulation overpass**.

- It was used mainly for combinational circuits.
- A method for synchronous sequential circuits was also proposed *Walczak: Deductive Fault Simulation for Sequential Module Circuits — [6]*.
  - The fault list for the sequential blocks modelled by Moore-type automaton is propagated using the automaton states.
Our achievements

- Application for the asynchronous circuits.
- Fault list propagation algorithm for the complex gates.
- Hazard detection during deductive fault simulation.
The developed deductive fault simulator can propagate the fault lists in the speed-independent circuits with complex gates.

The fault propagation algorithm is universal and can be used for any gate which is represented by a Boolean function expressed by the Disjunctive Normal Form.

The proposed algorithm is presented in the paper.

- Main principles.
- Example.
Deductive fault simulation
Main principles

- Analyzing the Boolean function.
- Fault lists from inputs of the gate are placed in 4 temporary lists.
- The output fault list is generated based on the temporary lists.
The meaning of the variables used in the algorithm is as follows:

- **andIntersectionList** — the list of faults for the examined conjunction which contains the faults of inputs with a logic 0 which is the controlling value for the logic operation **AND**;

- **orIntersectionList** — the list of faults for the whole disjunction which contains the faults of conjunctions with a logic 1 which is the controlling value for the logic operation **OR**;
Deductive fault simulation
Main principles

- \textit{andUnionList} — the list of faults for the examined conjunction which contains the faults of inputs with a logic 1 which is the \textit{non-controlling} value for the logic operation \textit{AND};

- \textit{orUnionList} — the list of faults for the whole disjunction which contains the faults of conjunctions with a logic 0 which is the \textit{non-controlling} value for the logic operation \textit{OR};
Consider the complex gate with the function $Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C$, where $A$, $B$, $C$ are the inputs and $Y$ is the output of the gate. $A = B = C = 1$, $L_A = \{A_{sa0}, G_{sa0}, H_{sa1}\}$, $L_B = \{B_{sa0}, G_{sa0}, H_{sa1}\}$, $L_C = \{C_{sa0}, H_{sa1}, L_{sa1}\}$

1. $Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C$
   
   $A = 1 \Rightarrow andUnionList = \{A_{sa0}, G_{sa0}, H_{sa1}\}$

2. $Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C$
   
   $B = 1 \Rightarrow andUnionList = \{A_{sa0}, G_{sa0}, H_{sa1}, B_{sa0}\}$

3. $andControllingSetIsEmpty = true$ and $A \cdot B = 1$
   
   $\Rightarrow orIntersectionList = \{A_{sa0}, G_{sa0}, H_{sa1}, B_{sa0}\}$

($G, H, L$ are lines in the circuit which are situated before the examined complex gate.)
Deductive fault simulation

Example

\[ Y = A.B + \overline{A}.\overline{B}.C \]

1. \[ Y = A.B + \overline{A}.\overline{B}.C \]
   \[ \overline{A} = 0 \Rightarrow \text{andIntersectionList} = \{A_{sa0}, G_{sa0}, H_{sa1}\} \]

2. \[ Y = A.B + \overline{A}.\overline{B}.C \]
   \[ \overline{B} = 0 \Rightarrow \text{andIntersectionList} = \{G_{sa0}, H_{sa1}\} \]

3. \[ Y = A.B + \overline{A}.\overline{B}.C \]
   \[ C = 1 \Rightarrow \text{andUnionList} = \{C_{sa0}, H_{sa1}, L_{sa1}\} \]

\[ \text{andControllingSetIsEmpty} = \text{false} \quad \text{and} \quad \overline{A}.\overline{B}.C = 0 \]
\[ \Rightarrow \text{orUnionList} = \{G_{sa0}\} \]

\[ \text{orControllingSetIsEmpty} = \text{false} \Rightarrow \]
\[ \text{faultListOf}(Y) = \{A_{sa0}, H_{sa1}, B_{sa0}\} \]
Deductive fault simulation

Example

\[ Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C \]

1. \[ Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C \]
   \[ \overline{A} = 0 \Rightarrow \text{andIntersectionList} = \{ A_{sa0}, G_{sa0}, H_{sa1} \} \]

2. \[ Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C \]
   \[ \overline{B} = 0 \Rightarrow \text{andIntersectionList} = \{ G_{sa0}, H_{sa1} \} \]

3. \[ Y = A \cdot B + \overline{A} \cdot \overline{B} \cdot C \]
   \[ C = 1 \Rightarrow \text{andUnionList} = \{ C_{sa0}, H_{sa1}, L_{sa1} \} \]

4. \( \text{andControllingSetIsEmpty} = \text{false} \) and \( A \cdot \overline{B} \cdot C = 0 \)
   \[ \Rightarrow \text{orUnionList} = \{ G_{sa0} \} \]

5. \( \text{orControllingSetIsEmpty} = \text{false} \Rightarrow \)
   \[ \text{faultListOf}(Y) = \{ A_{sa0}, H_{sa1}, B_{sa0} \} \]

This example is very simple and not demonstrating all capabilities of the proposed algorithm.
Results

- Implementation in C++.
- Evaluation has been done over a set of speed-independent benchmark circuits synthesized by Petrify as described in the paper by Cortadella et al.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers — [7].
## Results

### Fault coverage comparison

<table>
<thead>
<tr>
<th>Circuit name</th>
<th>Number of faults</th>
<th>Average stuck-at fault coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_outbound</td>
<td>58</td>
<td>92%</td>
</tr>
<tr>
<td>chu133</td>
<td>60</td>
<td>97%</td>
</tr>
<tr>
<td>chu150</td>
<td>40</td>
<td>82%</td>
</tr>
<tr>
<td>converta</td>
<td>56</td>
<td>46%</td>
</tr>
<tr>
<td>dff</td>
<td>34</td>
<td>79%</td>
</tr>
<tr>
<td>ebergen</td>
<td>46</td>
<td>N/A</td>
</tr>
<tr>
<td>half</td>
<td>34</td>
<td>N/A</td>
</tr>
<tr>
<td>hazard</td>
<td>40</td>
<td>86%</td>
</tr>
<tr>
<td>master_read</td>
<td>132</td>
<td>46%</td>
</tr>
<tr>
<td>mp_forward_pkt</td>
<td>66</td>
<td>95%</td>
</tr>
<tr>
<td>nak_pa</td>
<td>76</td>
<td>91%</td>
</tr>
<tr>
<td>nowick</td>
<td>50</td>
<td>98%</td>
</tr>
<tr>
<td>ram_read_sbuf</td>
<td>84</td>
<td>89%</td>
</tr>
<tr>
<td>rcv_setup</td>
<td>36</td>
<td>93%</td>
</tr>
<tr>
<td>rpdf</td>
<td>26</td>
<td>92%</td>
</tr>
<tr>
<td>sbuf_ram_write</td>
<td>82</td>
<td>78%</td>
</tr>
<tr>
<td>sbuf_send_ctl</td>
<td>66</td>
<td>49%</td>
</tr>
</tbody>
</table>
## Results

### Time and memory comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ser.</td>
<td>ded.</td>
<td>ser.</td>
</tr>
<tr>
<td>alloc_outbound</td>
<td>0.19</td>
<td>0.07</td>
<td>63%</td>
</tr>
<tr>
<td>chu133</td>
<td>0.23</td>
<td>0.29</td>
<td>-26%</td>
</tr>
<tr>
<td>chu150</td>
<td>0.09</td>
<td>0.07</td>
<td>22%</td>
</tr>
<tr>
<td>converta</td>
<td>0.18</td>
<td>0.26</td>
<td>-44%</td>
</tr>
<tr>
<td>dff</td>
<td>0.05</td>
<td>0.00</td>
<td>100%</td>
</tr>
<tr>
<td>ebergen</td>
<td>0.07</td>
<td>0.01</td>
<td>86%</td>
</tr>
<tr>
<td>half</td>
<td>0.08</td>
<td>0.13</td>
<td>-63%</td>
</tr>
<tr>
<td>hazard</td>
<td>0.05</td>
<td>0.01</td>
<td>80%</td>
</tr>
<tr>
<td>master_read</td>
<td>0.70</td>
<td>0.39</td>
<td>44%</td>
</tr>
<tr>
<td>mp_forward_pkt</td>
<td>0.24</td>
<td>0.04</td>
<td>83%</td>
</tr>
<tr>
<td>nak_pa</td>
<td>0.32</td>
<td>0.08</td>
<td>75%</td>
</tr>
<tr>
<td>nowick</td>
<td>0.14</td>
<td>0.05</td>
<td>64%</td>
</tr>
<tr>
<td>ram_read_sbuf</td>
<td>0.25</td>
<td>0.06</td>
<td>76%</td>
</tr>
<tr>
<td>rcv_setup</td>
<td>0.07</td>
<td>0.02</td>
<td>71%</td>
</tr>
<tr>
<td>rpdft</td>
<td>0.04</td>
<td>0.00</td>
<td>100%</td>
</tr>
<tr>
<td>sbuf_ram_write</td>
<td>0.15</td>
<td>0.01</td>
<td>93%</td>
</tr>
<tr>
<td>sbuf_send_ctl</td>
<td>0.19</td>
<td>0.10</td>
<td>47%</td>
</tr>
</tbody>
</table>
Results

- In most of the cases the deductive fault simulator is by 60% – 80% faster than the serial one.
- For 3 circuits the deductive fault simulator was slower than the serial one.
  - The fault dropping technique was implemented in the serial fault simulator.
  - This problem will be eliminated by implementing a deterministic test pattern generator.
- For 2 circuits lower memory requirements are reported for the deductive fault simulator than for the serial one.
  - Could be caused by low execution time.
  - Could be caused by the relatively small size of the circuits.
Conclusion

Improvements of the deductive fault simulation technique

- Application for the asynchronous circuits.
- Fault list propagation algorithm for the complex gates.
- Hazard detection during deductive fault simulation.
Conclusion

- Deductive fault simulator was developed and implemented for the stuck-at fault of the speed-independent asynchronous circuits.
- The experimental results show 60% – 80% reduction of the computational time and max. 14% increase of the memory requirements.

Future work

- Deterministic test pattern generator based on the algorithm FAN.
  - Without using external tools — reduction of the test generation time.
- Test pattern generator for delay faults.


Thank you for attention
Fault propagation rules for single gates

The deductive fault simulator is based on the following *fault propagation rules* for the logic gates *OR* and *AND*, where $I_1$ is the set of logic gate inputs with logic one, $I_0$ is the set of logic gate inputs with logic zero and $L_j$ is the fault list for input $j$.

- **Fault list propagation through gate *AND***:
  - if $I_0 = \emptyset$ then \( \{ \bigcup_{j \in I_1} L_j \} \)
  - else \( \{ \bigcap_{j \in I_0} L_j \} - \{ \bigcup_{j \in I_1} L_j \} \)

- **Fault list propagation through gate *OR***:
  - if $I_1 = \emptyset$ then \( \{ \bigcup_{j \in I_0} L_j \} \)
  - else \( \{ \bigcap_{j \in I_1} L_j \} - \{ \bigcup_{j \in I_0} L_j \} \)
Boolean function for the D flip-flop substitute

- Cooperates with SPIN-SIM.
- **Uses ATALANTA**  *Lee et al.: On the Generation of Test Patterns for Combinational Circuits* — [9].
  - Automatic test pattern generator for stuck-at fault of combinational circuits.
  - Generate test sequence for the asynchronous circuit.
  - Uses the test patterns for the combinational representation.
Random test pattern generator was implemented based on Shi et al.: Fault Simulation and Random Test Generation for Speed-Independent Circuits — [4].

It has the following properties:

- To avoid long meaningless test sequences the circuit is reseted to its initial state in the presence of hazards.
- Faults can be detected on the hazard-free outputs even in the presence of hazards, so it is not always necessary to reset the hazardous circuit.
- Circuit is reseted with a computed probability based on the number of hazards.
Our proposal for improving the performance

- Simple equation for the probability computation.
- Our equation:
  \[ P = \frac{n_h}{n} \cdot 100\%, \]  
  where \( n \) is the total number of outputs and \( n_h \) is the number of outputs with a hazard.
  - Hazard-free circuit: \( P = 0\% \).
  - Hazard on all of the outputs: \( P = 100\% \).
  - Linear dependency.

- Simplification without decreasing of the fault coverage.
Pseudo-gate implementation was proposed previously only for C-elements.

Asynchronous circuits can contain other memory elements too, for example D-flip-flops.

**We propose** the set of gates for replacing D-flip-flops in the circuits.
Pseudo-gate implementation of the D flip-flop

- The role of the pseudo-gate $p\text{-}and\text{-}3$ is to eliminate the hazard which could occur on the output $Q$ when $d = 1$, $Q = 1$ and a signal transition $0 \rightarrow 1$ or $1 \rightarrow 0$ occurs on $clk$. 
The circuit contain complex gates which are represented using the Disjunctive Normal Form.

Fault injection is handled by inserting gates to circuit with the following Boolean functions:

- for stuck-at-one:
  \[ O = I + \bar{I}, \]  

- for stuck-at-zero:
  \[ O = I \cdot \bar{I}, \]  

where \( I \) is the gate input and \( O \) is the gate output.
The circuit contain complex gates which are represented using the Disjunctive Normal Form.

Fault injection is handled by inserting gates to circuit with the following Boolean functions:

- for stuck-at-one:
  \[ O = I + \bar{I}, \]  
  \[ (2) \]

- for stuck-at-zero:
  \[ O = I \cdot \bar{I}, \]  
  \[ (3) \]

where \( I \) is the gate input and \( O \) is the gate output.

**Advantage**

- The simulation of fault-free and faulty circuit can be done in the same way.