Logic Testing and Design for Testability

MIT Press, Sept. 1985

Hideo Fujiwara
The basis is necessary for development

- For a tree to grow larger, its root must grow bigger and deeper into the ground.
- Similarly, for test technologies (leaves) to develop, substantial results of the fundamental research (root) are necessary.
- The more enriched the **fundamental** research results become, the more enriched the **practical** research results become.
Fundamental problems of testing

What are the fundamentals of testing?

I had the opportunity to ask myself the same question when I was requested to write a book.

To educate the fundamentals of testing, I wrote a book.

Fundamental problems of testing

- Test generation problem
  - ATPG (Automatic Test Program Generation)

- Design-for-test problem
  - DFT

I LOGIC TESTING
1 Introduction to Logic Testing
1.1 Logic Circuits
1.2 Fault Modeling
1.3 Testing Problems
1.4 Testing Schemes
2 Test Generation
2.1 Boolean Difference
2.2 The D-Algorith
2.3 The PODEM Algorithm
2.4 The FAN Algorithm
2.5 Test Generation for Sequential Circuits
3 Fault Simulation
3.1 Simulation Methodology
3.2 Parallel Fault Simulation
3.3 Deductive Fault Simulation
3.4 Concurrent Fault Simulation
3.5 Hardware Simulators
4 The Complexity of Testing
4.1 NP-Completeness
4.2 Polynomial Time Class
4.3 Closedness under Faults

II DESIGN FOR TESTABILITY
5 Introduction to Design for Testability
5.1 Testability
5.2 Minimization of Testing Cost
5.3 Combinational Logic versus Sequential Logic
5.4 Ad Hoc Design and Structured Design
6 Design to Minimize the Cost of Test Application
6.1 EOR Embedding
6.2 Minimally Testable Design
6.3 Dual-Mode Logic
6.4 Testing with Fixed Reference Values
7 Design to Minimize the Cost of Test Generation
7.1 Partitioning and Exhaustive Testing
7.2 Syndrome-Testable Design
7.3 Reed-Muller Canonical Forms
7.4 Programmable Logic Arrays
8 Scan Design for Sequential Logic Circuits
8.1 State-Shiftable Machines
8.2 Scan Design Approaches
8.3 Variations of Scan Design
8.4 Incomplete Scan Design and Enhanced Scan Design
9 Design for Built-in Self-Testing
9.1 Signature Analysis
9.2 Built-In Logic Block Observer
9.3 Self-Test with Scan Design
9.4 Self-Verification
Practical items

- Test Generation
- Fault Simulation
- Scan Design
- Built-in Self-Testing

Fundamental problems of testing

Practical synthesis problems

- Test generation algorithms.
  - Invent efficient algorithms to provide high fault efficiency.
- Design-for-testability methods.
  - Optimize the DFT under various constraints.

Theoretical analysis problems

- Analysis of test generation complexity.
  - Clarify the complexity of test generation algorithms.
- Classification of sequential circuits.
  - Class of combinational test generation complexity.
  - Class of acyclic test generation complexity.
Theory is the mother of practice

Necessity is the mother of invention

Theoretical research / fundamental research are necessary for practical research.

Theoretical research

→

Practical ideas

Polynomial Time Class ???

FAN algorithm

Analysis of test generation complexity

→

Efficient ATPG

Classification of sequential circuits

→

Optimal design for testability
A Boolean expression is *satisfiable* iff there exists some assignment of zeros and ones to the variables that gives the expression the value 1.

*SAT (Satisfiability)*: Is a Boolean expression satisfiable?

*Theorem 1* [Cook 1971]: SAT is NP-complete.

*U-SAT*: Satisfiability problem for *unate* expression

*Theorem 2* [Fujiwara 1982]: U-SAT is solvable in time $O(L)$ where $L$ is the length of an expression.
Complexity of test generation

NP-completeness

- **Fault detection** (FD): Is a given single stuck-at fault detectable?
  - $kM$-FD: Fault detection problem for $k$-level monotone circuits
  - $kU$-FD: Fault detection problem for $k$-level unate circuits

- **Theorem 3** [Fujiwara 1982]:
  - 3M-FD is NP-complete.
  - Hence, 3U-FD and FD are NP-complete.
Complexity of test generation

SAT versus FD

Observation

- SAT is NP-complete.
- U-SAT is solvable in $O(L)$.
- M-SAT is solvable in $O(L)$.

- FD is NP-complete.
- U-FD is NP-complete.
- M-FD is NP-complete.

[Fujiwara, MIT Press, 1985]
A combinational circuit $C$ is said to be \textit{k-bounded} if there exists a partition $\Pi = \{B_1, B_2, \ldots, B_t\}$ such that
(1) the number of inputs of each block $B_i$ is at most $k$, and
(2) graph $G_\Pi$ has no cycle.
Complexity of test generation

**k-bounded circuits** [Fujiwara, MIT Press, 1985]

- **Theorem 4** [Fujiwara 1982]: Let C be a *k-bounded* circuit. Then there is an algorithm of time complexity $O(16^k m)$ to find a test for a single stuck-at fault in C, where m is the number of lines in C.

![Figure 4.3](image1.png)
**Ripple-carry adder**

![Figure 4.4](image2.png)
**Gate-minimum p-bit adder**

**3-bounded circuit**

**6-bounded circuit**
Complexity of test generation

k-fanout-limited vs. k-fanout-point-bounded

[Fujiwara, MIT Press, 1985]

- **k-FL-FD**: Fault detection problem for k-fanout-limited circuits
- **k-FPB-FD**: Fault detection problem for k-fanout-point-bounded circuits

- **Theorem 5** [Fujiwara 1982]
  - k-FL-FD is NP-complete if $k > 2$.
  - k-FPB-FD is solvable in $O(4^km)$ where $m$ is the number of lines in $C$.

In the proof of this theorem, an algorithm is presented which requires the enumeration of at most $4^k$ combinations of values (0,1,D,D') on fanout-points.

This algorithm is the origin of the FAN algorithm.
Observation:
- $k$-FL-FD is NP-complete even if $k$ is a constant.
- $k$-FPB-FD is solvable in $O(m)$ if $k$ is a constant.

The complexity of test generation is affected
- not by the number of fanout branches from a fanout point
- but by the number of fanout points.

[Fujiwara, MIT Press, 1985]
Complexity of test generation

Theory is the mother of practice

[Fujiwara, MIT Press, 1985]

The algorithm shown in the proof of the theorems is the origin of the FAN algorithm.

Analysis of test generation complexity  Efficient ATPG
Heuristics of the FAN algorithm

[Fujiwara, MIT Press, 1985]

- Strategy 1: In each step of the algorithm, determine as many signal values as possible that can be *uniquely implied*. 
Heuristics of the FAN algorithm

[Fujiwara, MIT Press, 1985]

- Strategy 1: In each step of the algorithm, determine as many signal values as possible that can be uniquely implied.

- Strategy 2: Assign a fault signal D or D’ that is uniquely determined or implied by the fault in question.
Strategy 1: In each step of the algorithm, determine as many signal values as possible that can be uniquely implied.

Heuristics of the FAN algorithm [Fujiwara, MIT Press, 1985]

Strategy 2: Assign a fault signal $D$ or $D'$ that is uniquely determined or implied by the fault in question.

Strategy 3: When the D-frontier consists of a single gate, apply a unique sensitization.

Unique assignment can reduce backtracks.

D-frontier becomes empty and backtrack occurs later.
Heuristics of the FAN algorithm

- **Strategy 1:** In each step of the algorithm, determine as many signal values as possible that can be uniquely implied.

- **Heuristics of the FAN algorithm** [Fujiwara, MIT Press, 1985]

- **Strategy 2:** Assign a fault signal $D$ or $D'$ that is uniquely determined or implied by the fault in question.

- **Strategy 3:** When the $D$-frontier consists of a single gate, apply a unique sensitization.

- **Strategy 4:** Stop the backtrace at a head line, and postpone the line justification for the head line to later.

This can reduce backtracks
Heuristics of the FAN algorithm

- **Strategy 1:** In each step of the algorithm, determine as many signal values as possible that can be uniquely implied.

- **Heuristics of the FAN algorithm** [Fujiwara, MIT Press, 1985]

- **Strategy 2:** Assign a fault signal D or D' that is uniquely determined or implied by the fault in question.

- **Strategy 3:** When the D-frontier consists of a single gate, apply a unique sensitization.

- **Strategy 4:** Stop the backtrace at a head line, and postpone the line justification for the head line to later.

- **Strategy 5:** **Multiple backtracing** (concurrent backtracing of more than one path) is more efficient than backtracing along a single path.

*This can reduce computation of backtrace as well as backtracks*
Heuristics of the FAN algorithm

[Fujiwara, MIT Press, 1985]

- **Strategy 1:** In each step of the algorithm, determine as many signal values as possible that can be *uniquely implied*.

- **Strategy 2:** Assign a fault signal $D$ or $D'$ that is *uniquely determined* or implied by the fault in question.

- **Strategy 3:** When the D-frontier consists of a single gate, apply a *unique sensitization*.

- **Strategy 4:** Stop the backtrace at a *head line*, and postpone the line justification for the head line to later.

- **Strategy 5:** *Multiple backtracing* (concurrent backtracing of more than one path) is more efficient than backtracing along a single path. 

PODEM assigns a binary value only to primary inputs. 
So, backtracks occur at primary inputs.

FAN assigns a binary value only to head lines and fanout points. 
So, backtracks occur at headlines and fanout points.

*This is effective to reduce the number of backtracks.*

- **Strategy 6:** In the multiple backtrace, if an objective at a fanout point $p$ has a contradictory requirement, stop the backtrace so as to *assign a binary value to the fanout point*. 
History of test generation algorithms

Combinational ATPG

1966
D-algorithm (Roth, IBM)

1981
PODEM (Goel)

1983
FAN (Fujiwara)

1985
SOCRATES (Schulz, et al.)

1988
Recursive Learning (Kunz, et al.)

1990
NEMESIS (Larrabee)

1992
TRAN (Chakradhar, et al.)

1993
ITC-99 benchmarks

1999
SPIRIT (Gizdarski & Fujiwara)

2000

2001

ISCAS-85 Benchmarks (Fujiwara & Brglez)

ITC-99 benchmarks
The first ATPG able to achieve 100% fault efficiency for ISCAS’85

The first ATPG able to achieve 100% fault efficiency for ITC’99

SAT-based ATPG
History of test generation algorithms

Combinational ATPG

- 1966: D-algorithm (Roth, IBM)
- 1981: PODEM (Goel)
- 1983: FAN (Fujiwara)
- 1985: SOCRATES (Schulz, et al.)
- 1988: Recursive Learning (Kunz, et al.)
- 1990: NEMESIS (Larrabee)
- 1992: TRAN (Chakradhar, et al.)
- 1999: SPIRIT (Gizdarski & Fujiwara)
- 2000: IGRAINE (Tafertshofer, et al.)
- 2001: ITC-99 benchmarks

Sequential ATPG

- 1968: Extended D-algorithm (Kubo, NEC)
- 1971: 9-Valued (Muth)
- 1976: Extended D-algorithm (Putzolu & Roth, IBM)
- 1989: GENTEST (Cheng & Chakraborty)
- 1991: HITEC (Niermann & Patel)
- 1997: STRATEGATE (Hsiao, et al.)
- 1999: ITC-99 benchmarks

https://mitpress.mit.edu/index.php?q=books/logic-testing-and-design-testability
Thank you