High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems

Gert Jervan
Embedded Systems Laboratory (ESLAB)
Linköping University

Thesis Overview

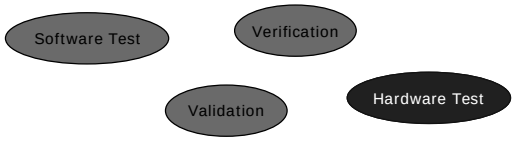
- Test and Design for Test of Digital Systems
- **High-Level Hierarchical Test Generation**
- **Hybrid Built-In Self-Test Technique**

Experimental Results
Conclusions
Future Work

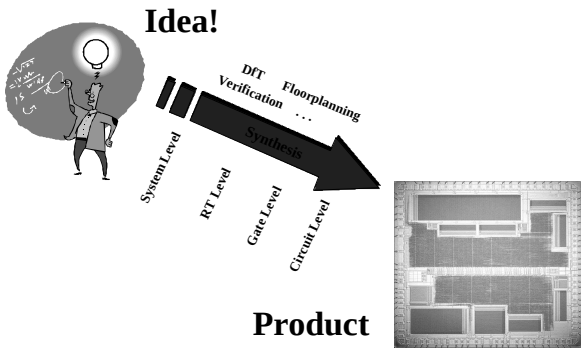
Test? What Test?

- To prove the truth, genuineness, or quality of by experiment, or by some principle or standard.

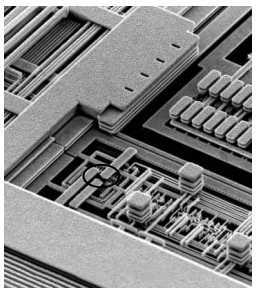
(Webster's Revised Unabridged Dictionary)



Semiconductor Design Flow



Defects

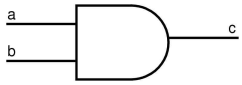


- Device level
  - shorts, cracks, leaks, impurities, bonding, ...
- Board level
  - component errors, track errors, ...
- Functional
  - Incorrect design
- Wearout/Environment
  - temperature, humidity, vibration, radiation...

Fault model -
abstraction mechanism for describing defects mathematically

Fault Models

- Single stuck-at (SSA) fault model



Possible Errors: 6
"a" s-a-1, "a" s-a-0
"b" s-a-1, "b" s-a-0
"c" s-a-1, "c" s-a-0

- stuck-open, bridging, delay, ...

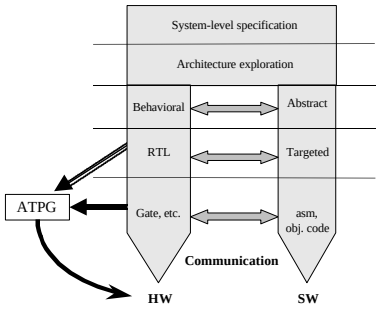
Test Generation

- Exhaustive tests
  - 2^n test patterns for an n-input combinational circuit

n	2	10	20	100
2^n	4	~10^3	~10^6	~10^{30}

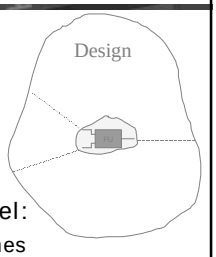
- Pseudorandom tests
- Deterministic tests
  - **Deterministic test generation is an NP-complete problem!!**
  - Different heuristics

Design Flow

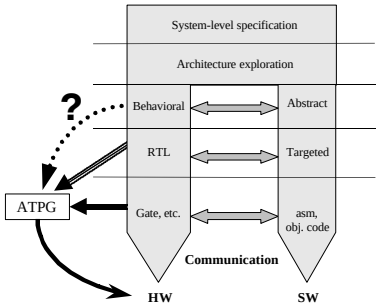


Hierarchical Test Generation

- Combining information from different levels of abstraction
- Top-down and bottom-up approaches
- Shortcomings at the RT-Level:
  - Very long test generation times
  - Large designs are not addressed
  - Requires lots of information about the final architecture and implementation



Design Flow



Behavioral Level

```
if (IN1 > 0)
    X = IN2 + 3;        --- q=1
else {
    if (IN2 >= 0)
        X = IN1 + IN2; --- q=2
    else
        X = IN1 * 5;    --- q=3
}

Y = X - 10;         -------- q=4
X = Y * 2;

OUT = X + Y;        -------- q=6
```

- Pure behavior
  - No allocation
  - No scheduling
  - No assignment

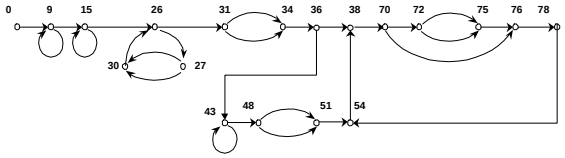
No information about the structure

High-level Testability

- Testability is an attribute of the implementation
- Theoretical framework concerning high-level testability is missing
- Proposed fault models efficiency is demonstrated only experimentally

High-level Fault Models

- Code coverage metrics
  - Path testing (P_∞)
  - Statement testing (P_1)
  - Branch testing (P_2)



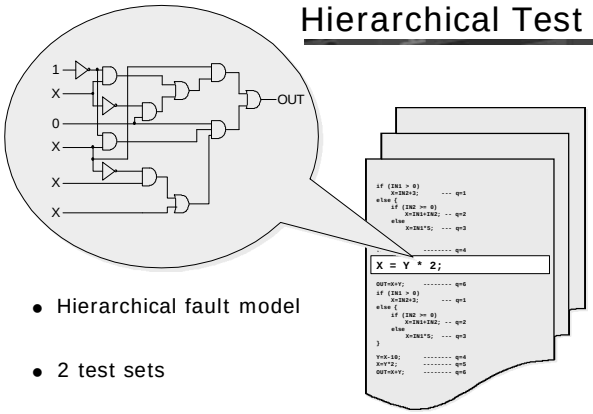
High-level Fault Models

- State coverage
- Condition coverage
  - Condition failures (stuck-at-true, stuck-at-false)
- Bit coverage
  - Bit failures in variables, signals and ports (stuck-at-0 and stuck-at-1)
  - Covers statement, branch, condition coverage and partially path coverage
- Mutation Fault Model

High-level Test Pattern Generation

- Gap between the fault coverage figures attained by test sequences generated purely on a high level and those by the gate-level ones
- Novel High-level Hierarchical ATPG
  - Combining behavioral level information with gate-level accuracy
  - Based on hierarchical test generation approach

Hierarchical Test

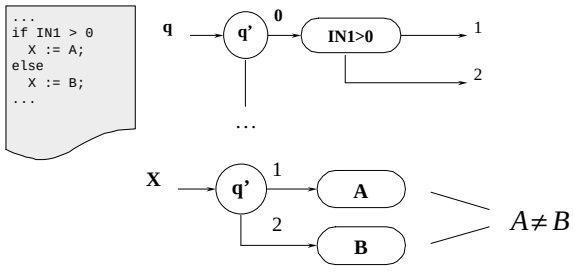


- Hierarchical fault model
- 2 test sets

Testing of Behavior

- Introduction of control states into the specification
- Conversion of modified specification to DDs
- Test generation based on code coverage metrics
- Employing commercial constraint solver (SICStus) for solving the constraints

Branch Activation Fault Model



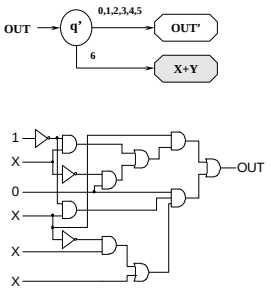
Testing FUs

- Library of FU-s
  - VHDL models on a behavioral/structural level
  - DD models on a structural level
- PODEM algorithm for gate-level test generation (external tool)
- "Intelligent" method to handle X values and constants
- Justification and propagation is performed on a behavioral level

Testing FUs

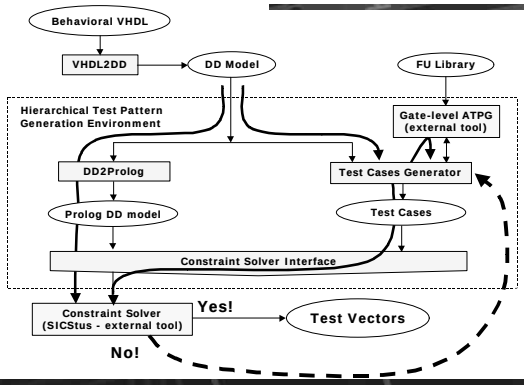
```
if (IN1 > 0)
    X=IN2+3;        --- q=1
else {
    if (IN2 >= 0)
        X=IN1+IN2; -- q=2
    else
        X=IN1*5;   --- q=3
}

Y=X-10;         -------- q=4
X=Y*2;          -------- q=5
OUT=X+Y;        -------- q=6
```

Behavioral Description



Gate-level netlist

Automatic HTG Environment



Experimental Results

- Classical High-level synthesis benchmark design
- Industrial design example (slice of F4)

Experimental Results

- DIFFEQ
  - # of lines (Behavioral VHDL): 59
  - # of gates: 4081
  - # of faults: 16222
  - # of untestable faults: 384
  - # of detected faults: 15277
  - Test generation time: 468 sec
  - # of unique test vectors: 199
  - # of applied test vectors: 20896 (includes vectors for reset etc.)
  - Fault efficiency: 96.46% / 98.05% (gate-level stuck-at coverage)

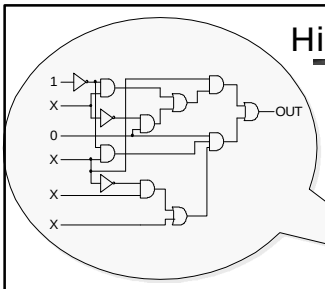
Experimental Results

	High level ATPG			Hierarchical ATPG			testgen		
	FC [%]	Len [#]	CPU [s]	FC [%]	Len [#]	CPU [s]	FC [%]	Len [#]	CPU [s]
DIFFEQ 1	97.25	553	954	98.05	199	468	99.62	1,177	4,792
DIFFEQ 2	94.57	553	954	96.46	199	468	96,75	923	4,475

- DIFFEQ 1 - optimized for speed
- DIFFEQ 2 - optimized for area

- High level ATPG - Politecnico di Torino
- testgen - Synopsys

Hierarchical ATPG




```
-- Tested 5616 faults
-- Untestable 0
-- Aborted 32
-- Fault coverage: 99.433428
-- 35 Vectors
```

```
X = Y * 2;
```

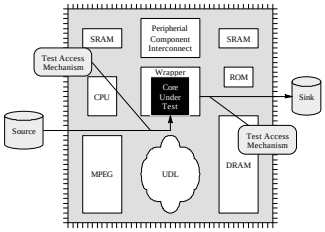

Experimental Results - F4

Design	VHDL Lines [#]	Stuck-at faults [#]	High-level HTG			Gate-level ATPG		
			Test length	**CPU [s]**	**FC [%]**	Test length	CPU [s]	FC [%]
F4_Input Handler_1	175	4872	**62**	**228**	**64.22%**	219	811	38.22%
F4_Output Handler_1	54	872	**26**	**1.52**	**76.26%**	170	5	81.30%

Conclusions

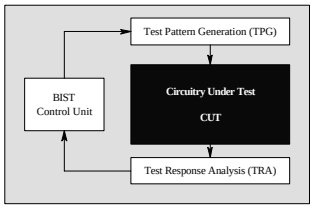
- Hierarchical test generation approach to incorporate structural information into the high-level test generation environment
- Improved fault coverage compared to the high-level ATPG
- Reduced test generation time and test length compared to the gate-level ATPG

Test Application



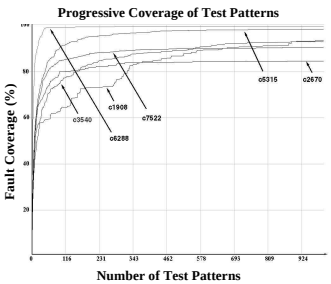
- Support from ATE
- Need for on-chip test: Appearance of Built-In Self-Test (BIST)

Built-In Self-Test



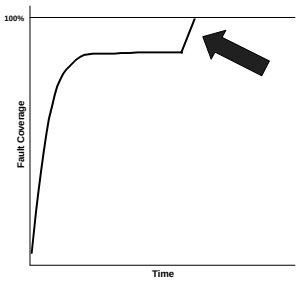
- Typically TPG & TRA are implemented as linear feedback shift registers (LFSR)

Problems with BIST



- LFSR generated vectors are pseudorandom by their nature
- Pseudorandom vectors:
  - Very long test application time
  - Not guaranteed high fault coverage
  - Area overhead
  - Additional delay

Problems with BIST



- Possible solution: Combination of pseudorandom test and deterministic test to form a **Hybrid BIST** solution in order to:
  - increase the fault coverage
  - reduce the test cost

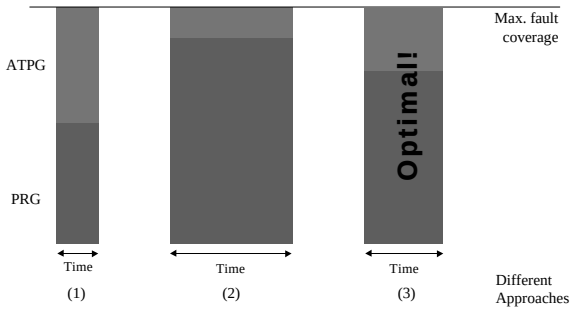
Related Work

- Improvement of pseudorandom test set
  - Multiple seed
    - Hellebrand et. al (ITC'92, JETTA 98)
    - Zacharia et. al (VTS'95)
  - Bit-flipping
    - Touba et. al (ITC'95), Chatterjee et. al (VTS'95)
- Minimization of test time
  - Sugihara et. al (DATE 2000)

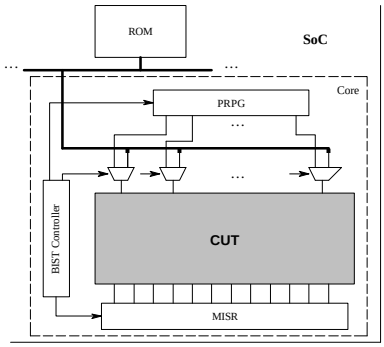
Objective

- Existing approaches:
  - To improve the fault coverage
  - To reduce the test application time
  - Test cost minimization has not been addressed directly
- Our objective:
  - Optimal balance between pseudorandom and deterministic test patterns in terms of time and memory without losing in fault coverage

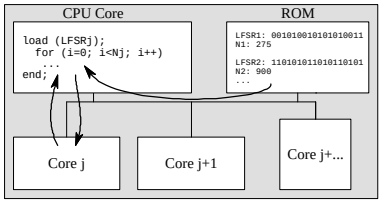
Hybrid BIST



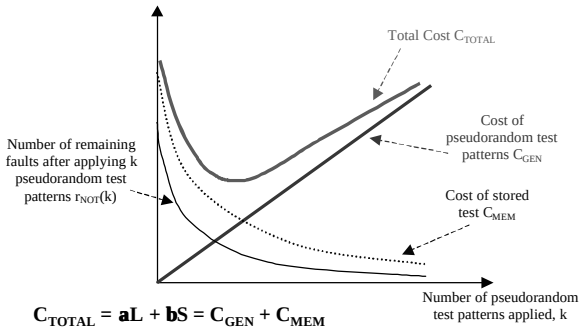
Hardware Based Hybrid BIST Architecture



LFSR Emulation

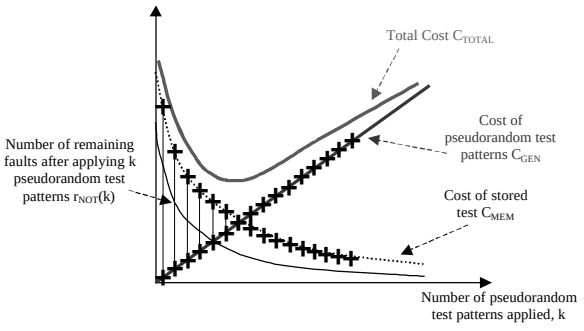


Cost Calculation for Hybrid BIST

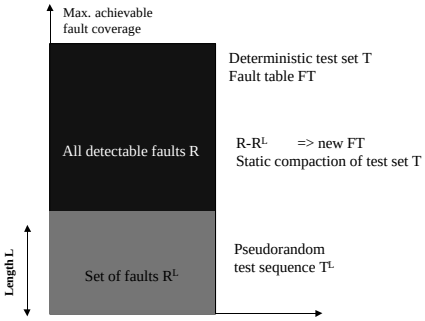


$$C_{TOTAL} = \mathbf{a}L + \mathbf{b}S = C_{GEN} + C_{MEM}$$

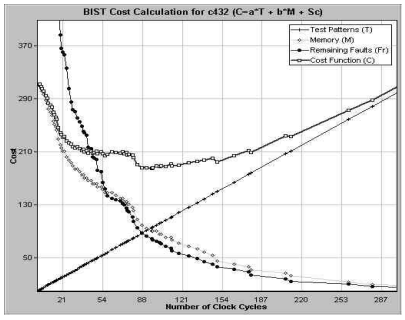
ATPG based approach



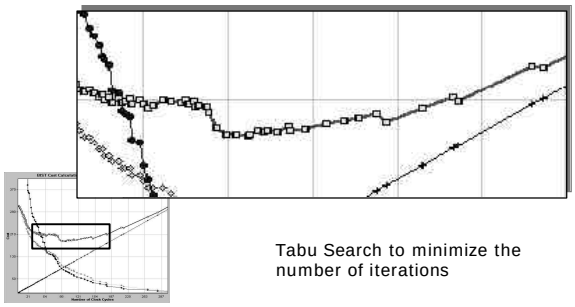
Fault Table Based Approach



Cost Calculation for Hybrid BIST



Cost Calculation for Hybrid BIST

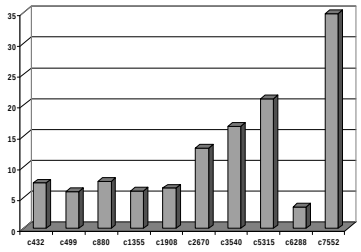


Tabu Search to minimize the number of iterations

Tabu search

- Iterative heuristic technique for optimization
- Gives good near-optimal solution
- Applies restrictions (“tabus”) to avoid local optima
- Takes into account history

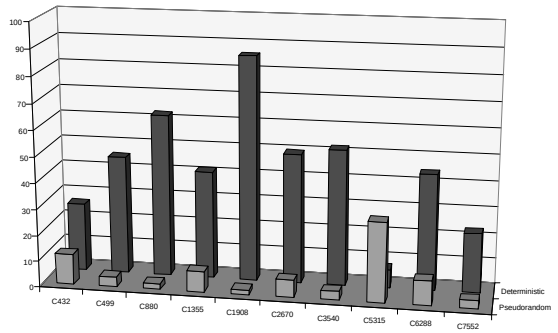
Experimental results



Acceleration of search speed by using Tabu search

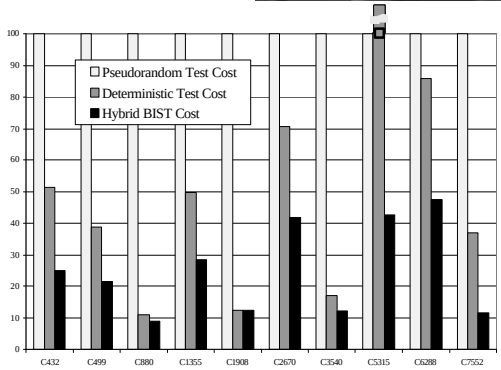
- List size: 3
- $V^* = 4$
- Step size: 3% of effective clocks
- E=7
- Accuracy of the final result at least 97,2%

Experimental Results



Percentage of the test vectors in the optimized test set compare to the original test set

Experimental Results



Conclusions - Hybrid BIST

- Hybrid BIST solution which combines pseudorandom and deterministic test patterns in cost effective way
- The BIST area overhead can be reduced by implementing some of the required test structures in software
- Hybrid BIST approach can lead to the significant reduction of test length without losing in the fault coverage

Conclusions

- A novel high-level hierarchical ATPG
  - Testability evaluation at early stages of the design flow
  - Efficient test patterns for manufacturing test
- Hybrid BIST scheme
  - Optimal combination between pseudorandom and deterministic test patterns
  - Method for test cost minimization

Future Work

- Testability of HW/SW systems
- High-level fault models
- Hybrid BIST for sequential circuits
- Self-test methods for other fault models