IHS 3: Test of Digital Systems

R. Ubar, A. Jutman, H-D. Wuttke
Overview

1. Introduction
2. Theory: Boolean differential algebra
3. Theory: Decision diagrams
4. Fault modelling
5. Test generation
6. Fault simulation
7. Fault diagnosis
8. Testability measuring
9. Design for testability
10. Built in Self-Test
Logic Synthesis

- translation of Boolean expressions into a netlist of components from a given library of logic gates such as NAND, NOR, EXOR, etc.

-> see logic synthesis section for details
Register-transfer Synthesis

- start with a set of states and a set of register-transfers in each state
- one state typically corresponds to a clock cycle (clock-accurate description)
- register-transfer synthesis generates the corresponding structures in two parts

(a) a data path which is a structure of storage elements and functional units that perform the given register transfers, and

(b) a control unit that controls the sequencing of the states in the register-transfer description

---

**Diagram**

- High-level synthesis
- Logic synthesis
- Circuit synthesis
- Behavioral domain
- Register transfers
- Boolean expressions
- Transistor functions
- Transistor layout
- Cells
- Chips
- Boards
- Physical domain (layout)

**States and Transfers**
- Registers, ALUs, MUXs
- Gates, flip-flops
- Processors, memories, buses

---

**Legend**
- Red arrows: register-transfer synthesis
- Blue arrows: high-level synthesis
- Gray arrows: behavioral domain

---

**Notes**
- Integrating hardware and software systems
- Andreas Mitschele-Thiel
- 19-Feb-14
BDDs and DDs

Test generation at logic level (BDD)

Test generation at higher levels (DD)
## Fault Modeling on High Level DDs

### High-level DDs (RT-level):

- **Terminal nodes represent:**
  - RTL-statement faults:
    - data storage,
    - data transfer,
    - data manipulation faults

- **Nonterminal nodes represent:**
  - RTL-statement faults:
    - label,
    - timing condition,
    - logical condition, register decoding,
    - operation decoding,
    - control faults

### SSBDDs (Gate-level):

- **Terminal nodes represent:**
  - Path value faults \{0,1\}
    - Stuck at 1
    - Stuck at 0

- **Nonterminal nodes represent:**
  - Path signal faults:
    - All stuck at faults along a corresponding signal path
RT-Level Design

• data path and control path on RT-level
• RT level simulation
• Functional units (F1,…,F4)
• Register
• Multiplexer / Demultiplexer
Register Level Fault Models

RTL statement:

\[ K: (\text{If } T, C) \quad R_D \leftarrow F(R_{S1}, R_{S2}, \ldots R_{Sm}), \quad \rightarrow N \]

Components (variables) of the statement:

- \( K \) - label
- \( T \) - timing condition
- \( C \) - logical condition
- \( R_D \) - destination register
- \( R_S \) - source register
- \( F \) - operation (microoperation)
- \( \leftarrow \) - data transfer
- \( \rightarrow N \) - jump to the next statement

RT level faults:

- \( K \rightarrow K' \) - label faults
- \( T \rightarrow T' \) - timing faults
- \( C \rightarrow C' \) - logical condition faults
- \( R_D \rightarrow R_D' \) - register decoding faults
- \( R_S \rightarrow R_S' \) - data storage faults
- \( F \rightarrow F' \) - operation decoding faults
- \( \leftarrow \) - data transfer faults
- \( \rightarrow N' \) - control faults
- \( (F) \rightarrow (F)' \) - data manipulation faults
Register Level Fault Models

Components (variables):

- **K** - label: -
- **T** - timing condition: -
- **C** - logical condition: $y_1 \ y_2 \ y_3 \ y_4$
- **$R_D$** - destination register: $R_2$
- **$R_S$** - source register: $R_1, \ IN$
- **F** - operation: +, *
- $\leftarrow$ - data transfer:
- $\rightarrow \ N$ - jump to the next statement: -
Fault Modeling on High Level DDs

High-level DDs (RT-level):

<table>
<thead>
<tr>
<th>$y_1$</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$M_1 = R_1$</td>
</tr>
<tr>
<td>1</td>
<td>$M_1 = IN$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$y_2$</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$M_2 = R_1$</td>
</tr>
<tr>
<td>1</td>
<td>$M_2 = IN$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$y_3$</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$M_3 = M_1 + R_2$</td>
</tr>
<tr>
<td>1</td>
<td>$M_3 = IN$</td>
</tr>
<tr>
<td>2</td>
<td>$M_3 = R_1$</td>
</tr>
<tr>
<td>3</td>
<td>$M_3 = M_2^* R_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$y_4$</th>
<th>Operation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset</td>
<td>$R_2 = 0$</td>
</tr>
<tr>
<td>1</td>
<td>Hold</td>
<td>$R_2 = R'_{2}$</td>
</tr>
<tr>
<td></td>
<td>Load</td>
<td>$R_2 = M_3$</td>
</tr>
</tbody>
</table>

Function

0 $M_1 = R_1$
0 $M_2 = R_1$
2 $M_3 = R_1$
3 $M_3 = M_2^* R_2$

Operation

0 Reset $R_2 = 0$
1 Hold $R_2 = R'_{2}$
Fault Modeling on High Level DDs

**High-level DDs (RT-level):**

Terminal nodes represent:
- RTL-statement faults: data storage, (e.g. \#0)
- data transfer, (e.g. IN)
- data manipulation faults (e.g. \( R_1 \times R_2 \))

Nonterminal nodes represent:
- RTL-statement faults: label, timing condition, logical condition, register decoding, operation decoding, control faults
Superposition of High-Level DDs:
A single DD for a subcircuit

Instead of simulating all the components in the circuit, only a single path in the DD should be traced
Decision Diagrams for Microprocessors

High-Level DDs for a microprocessor (example):

Instruction set:

1: MVI A,D  A ← IN
2: MOV R,A  R ← A
3: MOV M,R  OUT ← R
4: MOV M,A  OUT ← A
5: MOV R,M  R ← IN
6: MOV A,M  A ← IN
7: ADD R    A ← A + R
8: ORA R    A ← A ∨ R
9: ANA R    A ← A ∧ R
10: CMA A   A ← ¬ A

DD-model of the microprocessor:
Decision Diagrams for Microprocessors

High-Level DD-based structure of the microprocessor (example):

DD-model of the microprocessor:
**Hierarchical Modelling on DDs**

**System:** High-level decision diagram

**Component:** Binary Decision Diagram

A small part is simulated at the lower level.

A small part is simulated at the higher level: to increase the speed of analysis.

Cause-effect analysis well formalized.
Circuit Synthesis

- generates a transistor schematic from a set of input-output current, voltage and frequency characteristics or equations
- transistor schematic contains transistor types, parameters and sizes
Mapping Transistor Faults to Logic Level

A transistor fault causes a change in a logic function not representable by SAF model

Function: \[ y = x_1 x_2 x_3 \lor x_4 x_5 \]

Faulty function: \[ y^d = (x_1 \lor x_4) (x_2 x_3 \lor x_5) \]

Defect variable: \[ d = \begin{cases} 
0 & \text{defect } d \text{ is missing} \\
1 & \text{defect } d \text{ is present} 
\end{cases} \]

Generic function with defect:

\[ y^* = (y \land \overline{d}) \lor (y^d \land d) \]

Mapping the physical defect onto the logic level by solving the equation:

\[ \frac{\partial y^*}{\partial d} = 1 \]
Mapping Transistor Faults to Logic Level

Function: \( y = x_1 x_2 x_3 \lor x_4 x_5 \)

Faulty function: \( y^d = (x_1 \lor x_4)(x_2 x_3 \lor x_5) \)

Generic function with defect:
\[
y^* = (y \land \overline{d}) \lor (y^d \land d)
\]

Test calculation by Boolean derivative:
\[
\frac{\partial y^*}{\partial d} = \frac{\partial ((x_1 x_2 x_3 \lor x_4 x_5)\overline{d} \lor (x_1 \lor x_4)(x_2 x_3 \lor x_5)d)}{\partial d} =
\]
\[
=x_1 x_2 x_4 x_5 \lor x_1 x_3 x_4 x_5 \lor x_1 x_2 x_3 x_4 x_5 = 1
\]
Uniform Fault Model

\[ y^* = F^* (x_1, \ldots, x_n, d) = \overline{dF} \lor dF^d \]

Fault model: \( \{W^d\} \rightarrow dy \)

Hierarchical diagnostics
Hierarchical Test Generation

Hierarchical test (fault propagation)
Overview

1. Introduction
2. Theory: Boolean differential algebra
3. Theory: Decision diagrams
4. Fault modelling
5. Test generation
6. Fault simulation
7. Fault diagnosis
8. Testability measuring
9. Design for testability
10. Built in Self-Test
High-Level Decision Diagrams Applet

- Design of data path and a micro-program (control path) on the RT-level
- RT level simulation
- Fault simulation and test coverage evaluation
- Test generation
- Design for testability and BIST
Applets for Learning RT-Level Test

Functional Test mode:
The test vectors are operands results can be observed at the output, no extra test parts

Deterministic Test mode:
Gate level test for each FU separately, generate test vectors, local test panel, cumulative FC calculation

BIST mode:
Functional-, Logical- Circular Built In Self Test via extra test part: Test Pattern Generator and Signature analyzer

Because of interaction the learning process becomes more efficient

The game-like character should raise the students' curiosity
Functional Test

- The test vectors are operands, results can be observed at the output,
- no extra test parts needed
Deterministic Test

Adder (F2)
- Schematic level
- Gate level test
- Boolean derivation
- SSBDDs
- Path activation
- Fault propagation
- Fault observation
Deterministic Test

- Fault coverage of the vector (FC vec) and
- Fault coverage of the sequence (FC)

\[
\text{Fault Coverage} = \frac{\text{Number of detected Faults}}{\text{Total Number of Faults}}
\]

\[
\text{Fault Efficiency} = \frac{\text{Number of detected Faults}}{\text{Total Number of Faults} - \text{Number of undetectable Faults}}
\]
Deterministic Test

- Insert test vectors manually
- simulate the fault coverage of the vector (FC vec) and the sequence (FC)
Built In Self Test (BIST)

- **Functional**
  - No additional part
- **Logical**
  - Additional TPG
    - Test Pattern Generator
- **Circular**
  - Additional TPG/SA
    - Test Pattern Generator
    - Signature Analyzer
Functional BIST

- Find a configuration of the LFSR
  - (Linear Feedback Shift Register) that generates optimal operands (= test vectors) (TPG/SA)

- Parameters:
  - Initial State $S_0$
  - Feedback polynomial $P$
Logical BIST

• Find a configuration \((S_0, P)\) of the LFSR (Linear Feedback Shift Register) that generates optimal operands (= test vectors).

• LFSR is generator and analyzer in one.
Test Pattern Generation

- LFSR generator

(a) standard LFSR circuit

(b) modular LFSR circuit

I_1  I_2  I_3  I_4  I_5  
1 1 0 0 1

I_1  I_2  I_3  I_4  I_5  
1 1 0 0 1
High-Level Decision Diagrams Applet

- Micro operation: MUX, F1,…
- Control signal: micro operation
- Cost: number of gates that implement a micro operation
- Choose between high speed and low cost

- Implementing an algorithm
- F can be transparent / disabled
  - Only F2 M-Automaton
  - F1, F2, F3 sequence
  - F1, F2, F3; F4 parallel
High-Level Decision Diagrams Applet

- Algorithm: average
  - \((A+B)/2\)

- a) high speed
  - Max resources
  - Max parallel

- b) low cost
  - Min resources
  - Sequential
High-Level Decision Diagrams Applet

- **Algorithm:** high speed average
- **(A+B)/2** (4 steps Cost: 83)

- **Simulation**
  - Load from Input Reg1
  - Load from Input Reg2
  - Add, Div and store in 1 clock cycle
  - Output and End

<table>
<thead>
<tr>
<th>Ad</th>
<th>Next</th>
<th>F1 (in)</th>
<th>F2 (in)</th>
<th>F4</th>
<th>F4</th>
<th>IN</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F3 (out)</th>
<th>F4</th>
<th>OUT</th>
<th>Input</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>REG1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>REG1 REG2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
High-Level Decision Diagrams Applet

- Algorithm: average
- \((A+B)/2\)

- a) high speed
  - 2 operation units (F2, F3)
  - Add and Div in in 1 clock cycle
  - 4 steps
  - Cost: 83
High-Level Decision Diagrams Applet

- Algorithm: average
- \((A+B)/2\)

- Test

<table>
<thead>
<tr>
<th>Test Nr.</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
High-Level Decision Diagrams Applet

- Algorithm: average
- \((A+B)/2\)

- To continue the test
- Follow the example at
High-Level Decision Diagrams Applet

- **Algorithm**: average
  - $(A+B)/2$

- **b) low cost**
  - Only 1 Unit (F4)
  - 2 clock cycles for operation
  - 5 clocks
  - Cost: 73