Electronic Design Automation for Digital Circuits

Lecture Notes

Lecturer: U. Schlichtmann
Teaching Assistant: M. Barke
Room 2906, +49 89 289-23644
barke@tum.de

Address: Arcisstr. 21
80333 Munich
Germany
Telephone: +49 89 289-23666
Internet: http://eda.ei.tum.de

Contents of Lecture:

- Microelectronics overview, design flow for microelectronic systems, design space (Y-chart), implementation fabrics;
- Logic Synthesis, binary Boolean functions, optimization of combinational circuits (two-level, multi-level), FSMs, optimization of sequential circuits;
- Logic Simulation, event-driven simulation, modelling and simulation using VHDL;
- Testing of digital circuits, automatic test pattern generation (ATPG) for combinational circuits, fault simulation, test pattern generation for sequential circuits, scan path, built-in self-test.

Issue: February 14, 2012
## Important Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B^n$</td>
<td>$n$-dimensional Boolean Space</td>
<td>12</td>
</tr>
<tr>
<td>$\beta(x,a,b)$</td>
<td>Multiplexer function</td>
<td>20</td>
</tr>
<tr>
<td>$C_{M,N}$</td>
<td>Fault coverage</td>
<td>77</td>
</tr>
<tr>
<td>$c$</td>
<td>Cube</td>
<td>13</td>
</tr>
<tr>
<td>$c$</td>
<td>Implicant</td>
<td>21</td>
</tr>
<tr>
<td>cov</td>
<td>Cover</td>
<td>18</td>
</tr>
<tr>
<td>$D$</td>
<td>Don’t-care-set</td>
<td>14</td>
</tr>
<tr>
<td>$F$</td>
<td>On-set</td>
<td>14</td>
</tr>
<tr>
<td>$F_\nu$</td>
<td>Fault group</td>
<td>77</td>
</tr>
<tr>
<td>$f_{x_i}$</td>
<td>Cofactor w.r.t. $x_i$</td>
<td>15</td>
</tr>
<tr>
<td>$f_\mu$</td>
<td>Fault $\mu$</td>
<td>73</td>
</tr>
<tr>
<td>$MC$</td>
<td>Set of all cubes in $B^n$</td>
<td>13</td>
</tr>
<tr>
<td>$MI$</td>
<td>Set of all Implicants</td>
<td>21</td>
</tr>
<tr>
<td>$m_j$</td>
<td>Minterm, 0-cube</td>
<td>13</td>
</tr>
<tr>
<td>$MOC$</td>
<td>Set of all 0-cubes in $B^n$</td>
<td>13</td>
</tr>
<tr>
<td>$MPI$</td>
<td>Set of prime implicants</td>
<td>21</td>
</tr>
<tr>
<td>$p$</td>
<td>Prime implicant</td>
<td>21</td>
</tr>
<tr>
<td>POS</td>
<td>Product of sums</td>
<td>18</td>
</tr>
<tr>
<td>set($\chi$)</td>
<td>Set of variables</td>
<td>14</td>
</tr>
<tr>
<td>SOP</td>
<td>Sum of products</td>
<td>17</td>
</tr>
<tr>
<td>$T_C$</td>
<td>Test set</td>
<td>77</td>
</tr>
<tr>
<td>$T_\mu$</td>
<td>Test group</td>
<td>77</td>
</tr>
<tr>
<td>$t_\nu$</td>
<td>Test</td>
<td>72</td>
</tr>
<tr>
<td>$V_D$</td>
<td>Fault path</td>
<td>93</td>
</tr>
<tr>
<td>$x$</td>
<td>Variable</td>
<td>9</td>
</tr>
<tr>
<td>$\hat{x}$</td>
<td>Value of the variable (val($x$))</td>
<td>9</td>
</tr>
<tr>
<td>$y_\mu$</td>
<td>Output with fault $f_\mu$</td>
<td>73</td>
</tr>
</tbody>
</table>
# Contents

## I. Logic Synthesis

1. Logic Synthesis Basics ........................................................................ 9
2. Binary Boolean Functions .................................................................... 14
3. Synthesis of Two-Level Combinational Circuits .................................... 21
4. Heuristic Minimization of Two-Level Combinational Circuits .............. 34
5. Synthesis of Multi-Level Combinational Circuits .................................. 37
6. OBDD: Ordered Binary Decision Diagram ............................................ 44
7. Synthesis of Sequential Circuits ......................................................... 48

## II. Simulation of Digital Circuits

1. Introduction ......................................................................................... 57
2. Logic Simulation .................................................................................. 57
3. Simulation methods ............................................................................... 62
4. VHDL ................................................................................................. 64

## III. Testing of Digital Circuits

1. Introduction ......................................................................................... 69
2. Boolean Difference ............................................................................... 79
3. Fault Simulation ................................................................................... 90
4. Automatic Test Pattern Generation (ATPG) ....................................... 93
5. ATPG for Synchronous Sequential Circuits ....................................... 100

## Literature

108
I. Logic Synthesis

Task

<table>
<thead>
<tr>
<th>Behavioral description:</th>
<th>Design Requirements:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Finite State Machine</td>
<td>• Cell library data</td>
</tr>
<tr>
<td>• Boolean equations</td>
<td>• Layout design rules</td>
</tr>
<tr>
<td></td>
<td>• Testability</td>
</tr>
</tbody>
</table>

Optimization targets:

- Area
- Delay
- Power consumption

Structural description:

- Netlist using components of the target architecture
Typical Design Flow Steps

1. **Behavioral Specification**
   - Architectural Synthesis Tools
   - Manual Entry
   - RTL Description
     - Translation Tools
     - Unoptimized Logic Description
       - Logic Optimization
       - Technology Mapping
         - Optimized Logic Description
           - Physical Design Tools
             - Layout
             - Manufacturing
               - Test Generation
                 - Integrated Circuit

2. **Cell Library**
   - Custom Layout

3. **Test Generation**
Y-Chart

Behavioral Domain

System level

Algorithmic level

RT level

System Spec.

Algorithm

Register-Transfer Spec.

Boolean Eqn.

Differential Eqn.

Structural Domain

CPU, Memory

Processor, Subsystem

ALU, Reg., MUX

Gate, Flip-flop

Transistor

Rectangle/Polygon-Group

Standard-Cell / Subcell

Macro-Cell

Block / Chip

Chip / Board

Physical / Geometrical Domain
Design Space

Analysis
- Verification
- Extraction

Abstraction levels

System
1. System specification (C)
2. System architecture (CPUs, Busses)
3. Chip Board

Algorithm
4. Algorithms (VHDL, C)
5. Processor Subsystem
6. Block Chip

RT
7. RT Specification (Control/data-flow)
8. Modules (ALU, Registers, MUX)
9. Macro-Cell

Logic
10. Boolean Equations
11. Gates, Flip-Flops (Netlist)
12. Cells

Circuit
13. Differential Equations
14. Transistors (Netlist)
15. Mask Data Polygons

Behavior Structure Geometry

Synthesis
- Generation

Views

Introduction (4)
Combinational Circuit: 7−Segment Decoder

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Combinational Circuit: 7−Segment Decoder

<table>
<thead>
<tr>
<th>x3</th>
<th>x2</th>
<th>x1</th>
<th>x0</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
1 Logic Synthesis Basics

Digital circuits: binary signals

\[
\begin{array}{c|ccccc}
\times & 0 & 1 & 1 & 0 & 0 \\
y & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Output signal
(synchronous to clock)

\[
\begin{array}{c|ccccc}
x + y & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]

Input signals
Processing of signals
(synchronous to clock)

Logical (Signal-) Variable : \(x, y, \ldots\)

Value of the Variable : \(\text{value}(x) \triangleq \text{val}(x) \triangleq \hat{x}\)
\[\hat{x} = 0 \text{ oder } \hat{x} = 1 , \quad \hat{x} \in \{0, 1\}\]

Important Logical Operations

<table>
<thead>
<tr>
<th>Expression</th>
<th>Name</th>
<th>Gate symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bar{x})</td>
<td>NOT, Negation</td>
<td>![Gate Symbol]</td>
</tr>
<tr>
<td>(x \cdot y)</td>
<td>AND, Conjunction</td>
<td>![Gate Symbol]</td>
</tr>
<tr>
<td>(\bar{x} \cdot y)</td>
<td>NAND, (NOT AND)</td>
<td>![Gate Symbol]</td>
</tr>
<tr>
<td>(x + y)</td>
<td>OR, Disjunction</td>
<td>![Gate Symbol]</td>
</tr>
<tr>
<td>(\bar{x} + y)</td>
<td>NOR, (NOT OR)</td>
<td>![Gate Symbol]</td>
</tr>
<tr>
<td>(x \oplus y)</td>
<td>XOR, (excl. OR, Antivalence)</td>
<td>![Gate Symbol]</td>
</tr>
<tr>
<td>(\bar{x} \oplus y)</td>
<td>XNOR, (Equivalence)</td>
<td>![Gate Symbol]</td>
</tr>
</tbody>
</table>

Truth Table

<table>
<thead>
<tr>
<th>Value assignment</th>
<th>Truth values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>(y)</td>
</tr>
<tr>
<td>0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Operation is unary: \(\bar{x}\)

binary: \(x \cdot y\)

\(x + y\)

Logic Synthesis Basics (1)
Boolean Algebra

A) Laws of binary Boolean Algebra (Switching Algebra)

\( \text{BA} = (\{0, 1\}; \cdot, +, \overline{\cdot}) \)

Principle of duality

1. \( x \cdot y = y \cdot x \); \( x + y = y + x \)  
   Commutativity

2. \( (x \cdot y) \cdot z = x \cdot (y \cdot z) \); \( (x + y) + z = x + (y + z) \)  
   Associativity

3. \( x \cdot (y + z) = x \cdot y + x \cdot z \); \( x + y \cdot z = (x + y) \cdot (x + z) \)  
   Distributivity

4. \( x \cdot x = x \); \( x + x = x \)  
   Idempotence

5. \( x \cdot (x + y) = x \); \( x + x \cdot y = x \)  
   Absorption

6. \( x \cdot 1 = x \); \( x + 0 = x \)  
   Neutral Element

7. \( x \cdot 0 = 0 \); \( x + 1 = 1 \)  
   Dominance

8. \( x \cdot \overline{x} = 0 \); \( x + \overline{x} = 1 \)  
   Negation

9. \( \overline{x} = x \)  
   Double Negation

10. \( \overline{x \cdot y} = \overline{x} + \overline{y} \); \( \overline{x + y} = \overline{x} \cdot \overline{y} \)  
    De Morgan
B) Laws of Set Algebra

\[ \text{MA} = (P(G); \cap, \cup, \neg; G, \emptyset) \]

Principle of duality

Basic set \( G \); \( |G| = n \)

Power set \( P(G) \); \( |P(G)| = 2^n \)

(Model of a \( 2^n \)-valued Boolean algebra)

\[ A, B, C \in P(G) \]

1. \( A \cap B = B \cap A \); \( A \cup B = B \cup A \)  \hspace{1cm} \text{Commutativity}
2. \( (A \cap B) \cap C = A \cap (B \cap C) \) \hspace{1cm} \text{Associativity}
   \( (A \cup B) \cup C = A \cup (B \cup C) \)
3. \( A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \) \hspace{1cm} \text{Distributivity}
   \( A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \)
4. \( A \cap A = A \); \( A \cup A = A \) \hspace{1cm} \text{Idempotence}
5. \( A \cap (A \cup B) = A \) \hspace{1cm} \text{Absorption}
   \( A \cup (A \cap B) = A \)
6. \( A \cap G = A \); \( A \cup \emptyset = A \) \hspace{1cm} \text{Neutral Element}
7. \( A \cap \emptyset = \emptyset \); \( A \cup G = G \) \hspace{1cm} \text{Dominance}
8. \( A \cap \overline{A} = \emptyset \); \( A \cup \overline{A} = G \) \hspace{1cm} \text{Complement}
9. \( \overline{\overline{A}} = A \) \hspace{1cm} \text{Double Complement}
10. \( \overline{A \cap B} = \overline{A} \cup \overline{B} \) \hspace{1cm} \text{De Morgan}
    \( \overline{A \cup B} = \overline{A} \cap \overline{B} \)
**Boolean Space**

\( B^n : n \)-dimensional Boolean Space

(Signal-) Variable : \( x \)

Value of the variable : \( \text{val}(x) \triangleq \hat{x}; \ \hat{x} \in \{0, 1\} \)

Variable-\( n \)-tuple : \( \bar{x} = (x_1, \ldots, x_n) \)

Value assignment-\( n \)-tuple vertex in \( B^n \) : \( \hat{x} = (\hat{x}_1, \ldots, \hat{x}_n) \)

\( B = \{0, 1\}; \ \hat{x} \in B^n; \ |B^n| = 2^n; \ \hat{B}^n = \{\hat{x}_0, \ldots, \hat{x}_{2^n-1}\} \)

E.g. \( B^3 = B \times B \times B = \{(0, 0, 0), (0, 0, 1), \ldots, (1, 1, 1)\} \)

Notation : \( (1, 0, 1) \triangleq (101) \triangleq 101 = \hat{x}_5 \)

vector of bits,

bitpattern,

binary word,

binary number

\[ B^3 : 3 \text{-dimensional Boolean Space} \]

Literal \( l_i \) (positive or negative) : \( x_i \) or \( \bar{x}_i \)

Set of indices of literals : \( I_0 = \{1, \ldots, i, \ldots, n\}; \ |I_0| = n \)

Partial set of indices of literals : \( I_j; \ I_j \subseteq I_0 \)

Set of literals : \( L_0 = \{l_i \mid i \in I_0\} \)

Logic Synthesis Basics (4)
Minterm $m_j$, 0-Cube, AND clause, vertex

Cube $c_j$ in $B^n$, d-cube, Product term, AND clause

Dimension $d_j$ of cube $c_j$

Distance $\delta_{ij}$ between cubes $c_i$ and $c_j$

Example $n = 3$

Notation

Set $MC$ of all cubes in $B^n$

Set $MOC$ of all 0-cubes in $B^n$

Cube absorption

Logic Synthesis Basics (5)
2 Binary Boolean Functions
(binary switching functions)

\[ f(x) = f(x_1, \ldots, x_i, \ldots, x_n) \]

\( f : \mathcal{B}^n \to \mathcal{B} \)

\( n \)-dimensional
single output function

\( f : \mathcal{B}^n \to \mathcal{B}^m \)

multiple output function

sup(\( f \)) = set(\( x \)) = \{\( x_1, \ldots, x_i, \ldots, x_n \} : \) Set of variables

support of \( f \)

_\( f = (f_1, \ldots, f_j, \ldots, f_m) \) : \) Function-\( m \)-tuple

set(\( f \)) = \{\( f_1, \ldots, f_j, \ldots, f_m \} : \) Set of functions

\( f \in \text{set}(f) \) : \( f \) is a component of \( _\)

\[ f := \{ (\hat{x}, \hat{y}) \in \mathcal{B}^n \times \mathcal{B} \mid f(\hat{x}) = \hat{y} \} \]

\[ f(x) = y \iff (x, y) \in f \iff x\hat{y} \iff x \xrightarrow{f} y \]

on-set(\( f \)) = \{\( \hat{x} \in \mathcal{B}^n \mid f(\hat{x}) = 1 \} : \) On-set of \( f \)

off-set(\( f \)) = \{\( \hat{x} \in \mathcal{B}^n \mid f(\hat{x}) = 0 \} : \) Off-set of \( f \)

Convention:
\[ f := \text{on-set}(f) \]
\[ f := \{ \hat{x} \in \mathcal{B}^n \mid f(\hat{x}) = 1 \} \]

Incomplete specification of \( f \)

\( f : \mathcal{B}^n \to \{0, 1, *\} \)

on-set(\( f \)) = \( F \) = \{\( \hat{x} \in \mathcal{B}^n \mid f(\hat{x}) = 1 \}\)

don’t-care-set(\( f \)) = \( D \) = \{\( \hat{x} \in \mathcal{B}^n \mid f(\hat{x}) = * \}\)

off-set(\( f \)) = \( F \cup D \) = \{\( \hat{x} \in \mathcal{B}^n \mid f(\hat{x}) = 0 \}\)

Convention:
\[ F \subseteq f \subseteq F \cup D \]

Binary Boolean Functions (1)
Cofactor (restriction)

Cofactor of \( f(x_1, \ldots, x_i, \ldots, x_n) \)

w.r.t. \( x_i \):
\[
 f|_{x_i=1} = f_{x_i} = f(x_1, \ldots, 1, \ldots, x_n)
\]

w.r.t. \( \overline{x_i} \):
\[
 f|_{x_i=0} = f_{\overline{x_i}} = f(x_1, \ldots, 0, \ldots, x_n)
\]

Commutativity:
\[
(f_{x_i}x_j) = (f_{x_j}x_i) = f_{x_i}x_j
\]

Substitution rule:
\[
x_i \cdot f = x_i \cdot f_{x_i}, \quad \overline{x_i} \cdot f = \overline{x_i} \cdot f_{\overline{x_i}}
\]
\[
x_i + f = x_i + f_{x_i}, \quad \overline{x_i} + f = \overline{x_i} + f_{\overline{x_i}}
\]

Expansions of \( f(x) \) w.r.t. \( x_i \)
\[
f(x_1, \ldots, x_i, \ldots, x_n) = x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}}
\]
Shannon’s (Boole’s) expansion
\[
= (x_i + f_{\overline{x_i}}) \cdot (\overline{x_i} + f_{x_i})
\]
Muller-Reed’s (Davio’s) expansion
\[
= (f_{x_i} \oplus f_{\overline{x_i}}) \cdot x_i \oplus f_{x_i}
\]

Boolean difference of \( f(x) \) w.r.t. \( x_i \):
\[
\partial x_i(f) = \frac{\partial f}{\partial x_i} = f|_{x_i} = f_{x_i} \oplus f_{\overline{x_i}}
\]

Consensus of \( f(x) \) w.r.t. \( x_i \):
\[
C_{x_i}(f) = f_{x_i} \cdot f_{\overline{x_i}}
\]
(universal quantifier)

Smoothing of \( f(x) \) w.r.t. \( x_i \):
\[
S_{x_i}(f) = f_{x_i} + f_{\overline{x_i}}
\]
(existential quantifier)

\( f \) is a tautology:
\[\iff \forall \hat{x} \in \{0,1\}^n [f(\hat{x}) = 1] \triangleq f(x) = 1\]
\[
f = x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}} = 1 \iff f_{x_i} = 1 \text{ AND } f_{\overline{x_i}} = 1
\]
\( f \) is a contradiction:
\[\iff \forall \hat{x} \in \{0,1\}^n [f(\hat{x}) = 0] \triangleq f(x) = 0\]
\[
f = (\overline{x_i} + f_{x_i}) \cdot (x_i + f_{\overline{x_i}}) = 0 \iff f_{x_i} = 0 \text{ AND } f_{\overline{x_i}} = 0
\]
\( f \) is independent of \( x_i \):
\[\iff f_{x_i} = f_{\overline{x_i}} \iff f_{x_i} \oplus f_{\overline{x_i}} = 0
\]
\( f \) is dependent on \( x_i \):
\[\iff f_{x_i} \neq f_{\overline{x_i}} \iff f_{x_i} \oplus f_{\overline{x_i}} = 1
\]

DUAL\( (f(x_1, x_2, \ldots, x_n)) = \overline{f}(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})\)
f is positive symmetric w.r.t. $x_i$ and $x_j$ (positive symmetry):

\[ f(x, \ldots, x_i, \ldots, x_j, \ldots) = f(x, \ldots, x_j, \ldots, x_i, \ldots) \]

\[ \iff f(x_i, \ldots, x_j) = f(x_j, \ldots, x_i) \]

\[ \iff f(x) = \overline{f(x)} \]

\[ f \text{ is negative symmetric w.r.t. } x_i \text{ and } x_j \text{ (negative symmetry)}: \]

\[ f(x, \ldots, x_i, \ldots, x_j, \ldots) = f(x, \ldots, x_j, \ldots, x_i, \ldots) \]

\[ \iff f(x_i, \ldots, x_j) = f(x_j, \ldots, x_i) \]

\[ \iff f(x) = \overline{f(x)} \]

f is positive unate in $x_i$:

\[ f(x_i) \subseteq f(x) \iff f(x_i) = f(x) \]

\[ f(x_i) \subseteq f(x) \iff f(x) = x_i \cdot f(x) + \overline{f(x)} \]

\[ \iff f(x) = x_i \cdot f(x) + \overline{f(x)} \]

\[ \iff f(x) = x_i \cdot f(x) + f(x) \iff f(x) = 1 \]

\[ \text{IF} \quad f(x_i) \subseteq f(x), \quad \text{THEN} \quad f(x_i) \subseteq f \]

f is negative unate in $x_i$:

\[ f(x_i) \subseteq f(x) \iff f(x_i) = f(x) \]

\[ f(x_i) \subseteq f(x) \iff f(x) = x_i \cdot f(x) + \overline{f(x)} \]

\[ \iff f(x) = x_i \cdot f(x) + \overline{f(x)} \]

\[ \iff f(x) = x_i \cdot f(x) + f(x) \iff f(x) = 1 \]

\[ \text{IF} \quad f(x_i) \subseteq f(x), \quad \text{THEN} \quad f(x_i) \subseteq f \]

f is unate in $x_i$:

\[ (f(x_i) \subseteq f(x)) \lor (f(x_i) \not\subseteq f(x)) \]

f is binate in $x_i$:

\[ (f(x_i) \not\subseteq f(x)) \land (f(x_i) \not\subseteq f(x)) \]

f is unate:

\[ \forall x_i \in \text{set}(x) \quad f \text{ is unate in } x_i \]

Additional Comment:

To every set $M$, $M \subseteq G$, a binary Boolean function $f(x)$ can be assigned by a coding of its elements $z$, $z \in M$:

\[ z := \hat{x} \quad ; \quad |G| = 2^n \]

\[ z \in M \iff \hat{x} \in M \iff \hat{x} \in \text{on-set}(f) \iff f(\hat{x}) = 1 \]

i.e.: $M \triangleq \text{on-set}(f)$

\[ f(\hat{x}) : \text{ characteristic function of } M \]
Truth table, TT

Example for $f(x,y,z)$

<table>
<thead>
<tr>
<th>$m_κ$</th>
<th>xyz</th>
<th>$f(x,y,z)$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_0$</td>
<td>000</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$m_1$</td>
<td>001</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$m_2$</td>
<td>010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$m_3$</td>
<td>011</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$m_4$</td>
<td>100</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$m_5$</td>
<td>101</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$m_6$</td>
<td>110</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$m_7$</td>
<td>111</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$TT_1$

Reduced truth tables (Implicant tables)

(Canonical) sum of products formula, (C)SOP
(Canonical) disjunctive normal form, (C)DNF

$$CSOP : f = \sum_κ m_κ \triangleq \bigcup_κ m_κ ; \quad m_κ \subseteq f, \quad m_κ \in MOC$$

Example (1): $f = m_0 + m_2 + m_3 + m_7$

$TT_1$

$$f(x,y,z) = \overline{x} \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot y \cdot \overline{z} + \overline{x} \cdot y \cdot z + x \cdot y \cdot z$$

$\quad f = \{000\} \cup \{010\} \cup \{011\} \cup \{111\}$

$f = \{000,010,011,111\}$

Additional Comment:

$$f(x,y,z) = (x + \overline{x}) \cdot f = x \cdot f + \overline{x} \cdot f = x \cdot f_x + \overline{x} \cdot f_{\overline{x}}$$

$$\quad = (x + \overline{x}) \cdot (y + \overline{y}) \cdot (z + \overline{z}) \cdot f$$

$$f(x,y,z) = \overline{x} \cdot \overline{y} \cdot \overline{z} \cdot f(0,0,0) + \overline{x} \cdot \overline{y} \cdot z \cdot f(0,0,1)$$

$$\quad + \overline{x} \cdot y \cdot \overline{z} \cdot f(0,1,0) + \overline{x} \cdot y \cdot z \cdot f(0,1,1)$$

$$\quad + x \cdot \overline{y} \cdot \overline{z} \cdot f(1,0,0) + x \cdot \overline{y} \cdot z \cdot f(1,0,1)$$

$$\quad + x \cdot y \cdot \overline{z} \cdot f(1,1,0) + x \cdot y \cdot z \cdot f(1,1,1)$$

$e.g. \quad f(0,1,0) = f_{\overline{x}y\overline{z}}$
SOP : \[ f = \sum_{\kappa} \kappa \triangleq \bigcup_{\kappa} \kappa \; ; \; \kappa \subseteq f, \; \kappa \in MC \]

Example (2) : \[ f = c_0 + c_\alpha + c_\beta \triangleq c_0 \cup c_\alpha \cup c_\beta \]

TT 2 \[ f(x,y,z) = \bar{x} \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y + y \cdot z \]
\[ f = \{000\} \cup \{01\} \cup \{\ast11\} \]
\[ f = \{000\} \cup \{010,011\} \cup \{011,111\} \]

Example (3) : \[ f = c_\gamma + c_\beta \triangleq c_\gamma \cup c_\beta \]

TT 3 \[ f(x,y,z) = \bar{x} \cdot \bar{y} + y \cdot z \]
\[ f = \{0\ast0\} \cup \{\ast11\} = \{000,010\} \cup \{011,111\} \]

Definition : A cover of \( f \) is a set of product terms \( \text{cov}(f) = \{c_1, \ldots, c_k, \ldots, c_K\} \), such that: \( f = \sum_{\kappa \in \text{cov}(f)} c_\kappa \)

(Canonical) product of sums formula, (C)POS
(Canonical) conjunctive normal form, (C)CNF

CPOS : \[ f = \prod_{\kappa} \bar{m}_\kappa \triangleq \bigcap_{\kappa} \bar{m}_\kappa \; ; \; m_\kappa \subseteq \bar{f}, \; m_\kappa \in MOC \]

Example (1) : \[ \bar{f} = m_1 + m_4 + m_5 + m_6 \]

TT 1
\[ f = \bar{m}_1 \cdot \bar{m}_4 \cdot \bar{m}_5 \cdot \bar{m}_6 \]
\[ f(x,y,z) = (x+y+z) \cdot (x+y+z) \cdot (x+y+z) \cdot (x+y+z) \]

POS : \[ f = \prod_{\kappa} \bar{c}_\kappa \triangleq \bigcap_{\kappa} \bar{c}_\kappa \; ; \; c_\kappa \subseteq \bar{f}, \; c_\kappa \in MC \]

Example (2) : \[ f = \bar{c}_1 \cdot \bar{c}_\rho \cdot \bar{c}_\sigma \; ; \; \bar{f} = c_1 + c_\rho + c_\sigma \]

TT 2 \[ f(x,y,z) = (x+y+z) \cdot (x+y) \cdot (x+z) \]

Example (3) : \[ f = \bar{c}_\tau \cdot \bar{c}_\sigma \; ; \; \bar{f} = c_\tau + c_\sigma \]

TT 3 \[ f(x,y,z) = (y+z) \cdot (x+z) \]
\[ f(x,y,z) = \bar{x} \cdot \bar{z} + y \cdot z + \bar{x} \cdot y \] (SOP)
## Circuit Implementation

<table>
<thead>
<tr>
<th>Gate symbols</th>
<th>Operation</th>
<th>Function, On-set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>American</strong></td>
<td><strong>German</strong></td>
<td><strong>DIN-Norm 40900</strong></td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td>$y = f(x_1, x_2) = x_1 \cdot x_2$</td>
</tr>
<tr>
<td>OR</td>
<td>≥1</td>
<td>$y = f(x_1, x_2) = x_1 + x_2$</td>
</tr>
<tr>
<td>NOT</td>
<td>1</td>
<td>$y = f(x) = \overline{x}$</td>
</tr>
<tr>
<td>NAND</td>
<td>&amp;</td>
<td>$y = f(x_1, x_2) = x_1 \cdot x_2$</td>
</tr>
<tr>
<td>NOR</td>
<td>≥1</td>
<td>$y = f(x_1, x_2) = x_1 + x_2$</td>
</tr>
<tr>
<td>XOR</td>
<td>=1</td>
<td>$y = f(x_1, x_2) = x_1 \oplus x_2$</td>
</tr>
<tr>
<td>XNOR</td>
<td>=</td>
<td>$y = f(x_1, x_2) = x_1 \oplus x_2$</td>
</tr>
<tr>
<td>Subjunction (Implication)</td>
<td>≥1</td>
<td>$y = f(x_1, x_2) = x_1 \rightarrow x_2 = x_1 + x_2$</td>
</tr>
<tr>
<td>MUX</td>
<td>$a$, $b$: Data inputs</td>
<td>$y = \beta(x, a, b) = x \cdot a + \overline{x} \cdot b$</td>
</tr>
</tbody>
</table>

**Example (1):**

\[
w = f(x, y, z) = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot z + x \cdot y \cdot z
\]

Literal count: 12
(Effect, area consumption)

Two-level realization
Number of circuit levels: 2
(Signal delay)

\[
(x \rightarrow y) \equiv (x \cdot y + x \cdot y \cdot z)
\]
Example (2):

\[ w = f(x, y, z) = \bar{x} \cdot \bar{y} \cdot \bar{z} + x \cdot y + y \cdot z \]

(7 literals)

Example (3):

\[ w = f(x, y, z) = \bar{x} \cdot \bar{z} + y \cdot z \]

(4 literals)

Example (MUX):

[x, a, b]

y = β(x, a, b) = x \cdot a + \bar{x} \cdot b

β(1, a, b) = a , \quad β(0, a, b) = b

\[ x \cdot a + \bar{x} \cdot b \iff \text{if } x \text{ then } a , \text{ else } b \]

\[ β(x, a, b) \triangleq \text{ite}(x, a, b) \]

Multiplexer function β:

\[ y = β(x, a, b) = x \cdot a + \bar{x} \cdot b \]

\[ = (\bar{x} + a) \cdot (x + b) \]

\[ = (x \rightarrow a) \cdot (\bar{x} \rightarrow b) \]

\[ = [x \cdot (a \oplus b)] \oplus b \]

\[ f(x_1, \ldots, x_i, \ldots, x_n) = x_i \cdot f_{x_i} + \bar{x_i} \cdot f_{\bar{x_i}} \]

\[ = β(x_i, f_{x_i}, f_{\bar{x_i}}) \]
3 Synthesis of Two-Level Combinational Circuits

Definitions:

a) An implicant \( c \) of a function \( f \) is a product term \( c \in MC \) in \( B^n \), for which \( c \subseteq f \) holds.

Set of implicants : \( MI = \{ c \in MC \mid c \subseteq f \} \)

b) A prime implicant \( p \) of a function \( f \) is an implicant, which is not contained in any implicant of \( f \).

Set of prime implicants : \( MPI = \{ p \in MI \mid p \not\subseteq c, \text{ for all } c \in MI \} \)

\(|MPI| \lesssim \left\lceil \frac{3^n}{n} \right\rceil ; \quad MPI \subseteq MI \subseteq MC \)

Proposition:

Given \( p_j \in MI \) \((p_j \subseteq f)\) with \( p_j = \prod_{i \in I_j} l_i \), then:

\[ p_j \in MPI : \iff (p_j)_{l_i} \notin MI, \quad \text{for all } l_i \]

or \( p_j \notin MPI \iff (p_j)_{l_i} \in MI, \quad \text{for at least one } l_i. \)

Example (2) : \( f(x,y,z) = \overline{x} \cdot \overline{y} \cdot \overline{z} + y \cdot z + \overline{x} \cdot y \)

(1) \( y \cdot z \in MPI \)

Reasoning : \( (y \cdot z)_y = z ; \quad (y \cdot z)_z = y ; \)
\( z \notin MI \AND y \notin MI \text{ or } z \not\subseteq f \AND y \not\subseteq f, \text{ since} \)
\( f_z = y + \overline{x} \cdot y = y \neq 1 \AND f_y = z + \overline{x} \neq 1 \)

(2) \( \overline{x} \cdot \overline{y} \cdot \overline{z} \notin MPI \)

Reasoning : \( (\overline{x} \cdot \overline{y} \cdot \overline{z})_{\overline{x}} = \overline{x} \cdot \overline{z} ; \)
\( \overline{x} \cdot \overline{z} \in MI \text{ or } \overline{x} \cdot \overline{z} \subseteq f, \text{ since} \)
\( f_{\overline{x} \overline{z}} = \overline{y} + y = 1 \)

c) A minimal SOP (MinSOP) consists of a minimal number of literals.

d) Quine’s theorem about prime implicants:

A MinSOP always consists of prime implicants.

e.g. \( \text{SOP} : \quad f = x \cdot \overline{y} + y \equiv \{10, *1\} \)
\( \text{MinSOP} : \quad f = x + y \equiv \{1*, *1\} \)

Synthesis of Two-Level Combinational Circuits (1)
Quine’s Method to determine all prime implicants of \( f \)

Given: SOP or (reduced) truth table

Wanted: All prime implicants (Quine’s PI-Theorem)

**Step 1)** Determination of CSOP from SOP
(Disadvantage: number of minterms possibly very large, \( 2^n \))

e.g. SOP:  
\[
f(x, y, z) = x \cdot y \cdot z + x \cdot y \cdot (z + x) + y \cdot z \cdot (x + x) = 1
\]

CSOP:  
\[
f(x, y, z) = \bar{x} \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot \bar{z} + \bar{x} \cdot y \cdot z + x \cdot y \cdot z
\]

**Step 2)** Determination of all prime implicants of \( f \) by exhaustive application of two laws on the CSOP.

- Specialized rule of resolution (R):  
  \[
x \cdot a + \bar{x} \cdot a = a
\]
- Specialized consensus:  
  \[
\beta(x, a, a) = a
\]

- Law of absorption (A):  
  \[
a + a \cdot b = a
\]
  \[
a \cdot b \subseteq a
\]

Table of prime implicants for example (1):

| Number of positive literals | Minterm of \( f \)  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-cube</td>
</tr>
<tr>
<td>0</td>
<td>( \bar{x} \cdot \bar{y} \cdot \bar{z} \triangleq {000} = m_0 )</td>
</tr>
<tr>
<td>1</td>
<td>( \bar{x} \cdot \bar{y} \cdot \bar{z} \triangleq {010} = m_2 )</td>
</tr>
<tr>
<td>2</td>
<td>( \bar{x} \cdot \bar{y} \cdot \bar{z} \triangleq {111} = m_7 )</td>
</tr>
</tbody>
</table>

Definition: A complete SOP of \( f \) consists of all prime implicants of \( f \).

e.g.  
CompSOP, complete sum of products:  
\[
f = \bar{x} \cdot \bar{z} + \bar{x} \cdot y + y \cdot z = p_1 + p_2 + p_3
\]

prime cover:  
\[
\text{pcov}(f) = \{p_1, p_2, p_3\}
\]
Quine - McCluskey: Determination of MinSOP from CompSOP by solution of a covering problem

Example:

CSOP : \( f = m_0 + m_2 + m_3 + m_7 \);  
CompSOP : \( f = p_1 + p_2 + p_3 \)

For each minterm  
of \( f \): \( m \subseteq f \)  
For each prime implicant  
of \( f \): \( p \subseteq f \)

Condition \( C \) for covering all minterms by prime implicants:

For each minterm \( m \) at least one prime implicant \( p \) must exist in the MinSOP,  
such that: \( m \subseteq p \) , i.e.

\[
C = (m_0 \subseteq p_1) \cdot (m_2 \subseteq p_1 + m_2 \subseteq p_2) \cdot (m_3 \subseteq p_2 + m_3 \subseteq p_3) \cdot (m_7 \subseteq p_3)
\]

Absorption

\[
C = \tau_1 \cdot \tau_2 \cdot \tau_3
\]

\[
C = \tau_1 \cdot \tau_3 ; \quad \tau_\nu \text{ is selection variable for } p_\nu ; \quad \tau_\nu \in \{0, 1\}
\]

MinSOP : \( f = p_1 + p_3 = x \cdot z + y \cdot z \),  
compare circuit realization of example (3)

Comment: \( p_1 \) and \( p_3 \) must be selected, as they are essential prime implicants.

Covering table:

| \( p \) | \( m \) | \( m_0 = \overline{x} \cdot \overline{y} \cdot \overline{z} \) | \( m_2 = \overline{x} \cdot y \cdot \overline{z} \) | \( m_3 = \overline{x} \cdot y \cdot z \) | \( m_7 = x \cdot y \cdot z \) |
|---|---|---|---|---|
| \( p_1 = \overline{x} \cdot z \) | 1 | 1 | 0 | 0 |
| \( p_2 = \overline{x} \cdot y \) | 0 | 1 | 1 | 0 |
| \( p_3 = y \cdot z \) | 0 | 0 | 1 | 1 |

Selection of \( p_1 \) : Covering of \( m_0 \) and \( m_2 \)

Selection of \( p_3 \) : Covering of \( m_3 \) and \( m_7 \)

Reduction of problem and heuristic selection:

a) Determine of essential prime implicants (only '1' in a given column).

b) Cross out minterms (columns), which will be covered when a dominant minterm is covered.

\[ \text{e.g. } m_2 \text{ is dominated by } m_0 \quad \text{and} \quad m_3 \text{ is dominated by } m_7 . \]

c) Select prime implicants based on maximal number of '1's in rows (greedy algorithm).
Determination of all prime implicants using the resolution rule

Given : SOP or (reduced) truth table
Wanted : All prime implicants (Quine’s PI-Theorem)

Determination of all prime implicants of \( f \) by exhaustive application of two laws on the SOP (Determination of CSOP not required!)

Law of absorption (A) : \( a \cdot b \subseteq a \)
\( a + a \cdot b = a \)

General rule of resolution (R) in disjunctive form (consensus in general): \( \beta(x,a,b) = \beta(x,a,b) + a \cdot b \)
Resolvent
\( a \cdot b \subseteq \beta(x,a,b) \)

With \( x \in \text{sup}(f), a \in \text{MC}, b \in \text{MC}, \) and: \( \text{IF } a \cdot b \neq 0, \text{ THEN } \delta(x,a \cdot b) = 1. \)

Special cases of R: 1) \( \beta(x,a,a) = x \cdot a + \bar{x} \cdot a = x \cdot a + \bar{x} \cdot a + a = a \)
2) \( \beta(x,a,1) = x \cdot a + \bar{x} = x \cdot a + \bar{x} + a = \bar{x} + a \)
3) \( \beta(x,1,b) = x + \bar{x} \cdot b = x + \bar{x} \cdot b + b = x + b \)
4) \( \beta(x,a,a \cdot b) = x \cdot a + \bar{x} \cdot a \cdot b = x \cdot a + \bar{x} \cdot a \cdot b + a \cdot b = x \cdot a + a \cdot b \)
5) \( \beta(x,a \cdot b,a) = x \cdot a \cdot b + \bar{x} \cdot a = x \cdot a \cdot b + \bar{x} \cdot a + a \cdot b = \bar{x} \cdot a + a \cdot b \)

Resolution method (Layer algorithm)

a) Example SOP:
\( f(x,y,z) = \bar{x} \cdot y + x \cdot y \cdot z + \bar{x} \cdot \bar{y} \cdot \bar{z} \)

\[
\begin{array}{|c|}
\hline
f & \text{Layer} \\
\hline
\bar{x} \cdot y + x \cdot y \cdot z + \bar{x} \cdot \bar{y} \cdot \bar{z} & 0 \\
+ y \cdot z + \bar{x} \cdot \bar{z} & 1 \\
\hline
\end{array}
\]

No further resolvents possible. R and A laws have been applied exhaustively. Computation of Comp-SOP by determination of all possible prime implicants!

b) Example SOP:
\( t(x,y,z) = y \cdot z + \bar{x} \cdot \bar{z} + \bar{y} \cdot z + x \cdot \bar{z} \)

\[
\begin{array}{|c|}
\hline
\text{Layer} & \text{R} \\
\hline
\frac{y}{z} + \bar{x} \cdot \bar{z} + \bar{y} \cdot z + x \cdot \bar{z} & 0 \\
+ \bar{x} \cdot y + z + x \cdot y \cdot z + \bar{z} + x \cdot \bar{y} & 1 \\
+ y + \bar{x} + 1 + \ldots & 2 \\
\hline
\end{array}
\]

If repeated application of the resolution rule – starting from an arbitrary SOP – leads to a resolvent 1, then \( t = 1 \) has been proven. Tautology proof of \( t \) using the layer algorithm!
c) Deduction by the layer algorithm

<table>
<thead>
<tr>
<th>SOP</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ldots + x \cdot a + \bar{x} \cdot b + \ldots$</td>
<td>0</td>
</tr>
<tr>
<td>$\ldots + a \cdot b + \ldots$</td>
<td>1</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ddots$</td>
</tr>
<tr>
<td>$+1 \text{ or maximal number of prime implicants}$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

**Determination of all prime implicants from a conjunctive normal form (POS)**

**Given**: POS or (reduced) truth table

**Wanted**: All prime implicants

**Theorem**: $f_1 \cdot f_2$ is a CompSOP, IF $f_1$ and $f_2$ are each CompSOPs.

Determination of all prime implicants of $f$ from the POS representation by exhaustive application of laws of distribution and absorption. All other laws of Boolean algebra are to be considered (no determination of CPOS required!).

**Example (2):**

POS : $f(x,y,z) = (\overline{x+y}) \cdot (\overline{x+z}) \cdot (x+y+\overline{z})$

= $(\overline{x+y}) \cdot (\overline{y'+y+z}) \cdot (x+y+\overline{z})$

= $(\overline{x+y' \cdot z}) \cdot \overline{z} \cdot (x+y+\overline{z})$

= $\overline{z} \cdot x + \overline{x} \cdot y + \overline{x} \cdot z + \overline{y} \cdot z + y \cdot z + y \cdot z \cdot z$

CompSOP $f(x,y,z) = \overline{x} \cdot y + \overline{x} \cdot \overline{z} + y \cdot z$
Visualization of logic minimization in the Boolean space $\mathcal{B}^n$

a) **Karnaugh map**

Representation for $n = 3$

<table>
<thead>
<tr>
<th>$000$</th>
<th>$001$</th>
<th>$010$</th>
<th>$011$</th>
<th>$101$</th>
<th>$100$</th>
<th>$111$</th>
<th>$110$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$y$</td>
<td>$z$</td>
<td>$x$</td>
<td>$y$</td>
<td>$z$</td>
<td>$x$</td>
<td>$y$</td>
</tr>
</tbody>
</table>

Segment $m_v$: $v = 0, \ldots, 2^n - 1$

Set of segments: $MOC$

$m_\mu$ and $m_\nu$ are neighboring segments, IF $\delta_{\mu \nu} = 1$

$n = 2$

$n = 1$

b) **Cube graph $G_n = (MOC, E)$**

Representation for $n = 3$

vertex $m_\nu$

edge $k = (m_\mu, m_\nu)$

set of vertices: $MOC$

$m_\mu, m_\nu \in MOC$

set of undirected edges: $K$

$K \subseteq MOC \times MOC$

$(m_\mu, m_\nu) \in K \iff \delta_{\mu \nu} = 1$

Definition: A cube graph $G_n$ ($n$-cube) consists of $2^n$ vertices and $\frac{n}{2} \cdot 2^n$ edges. Each vertex is of degree $g = n$. 

Synthesis of Two-Level Combinational Circuits (6)
Worksheet (for copying)

Karnaugh maps and cube graphs for $n = 3$ and $n = 4$

Synthesis of Two-Level Combinational Circuits (7)
c) Example (1):

\( n = 3 \)

Given CSOP : \( f = \overline{x} \cdot y \cdot z + \overline{x} \cdot y \cdot z + \overline{x} \cdot y \cdot z + x \cdot y \cdot z \)

Wanted MinSOP : \( f = \overline{x} \cdot \overline{z} + y \cdot z \)

\( f = \{0 \ast 0, \ast 11\} \)
Example (4): $n = 4$

Given SOP: $f = \overline{y \cdot z} + \overline{y \cdot z \cdot w} + \overline{y \cdot z} \cdot w + x \cdot y \cdot w$

\[ \begin{array}{cccc}
0000 & 0001 & 0101 & 0100 \\
0010 & 0011 & 0111 & 0110 \\
1010 & 1011 & 1101 & 1110 \\
1100 & 1101 & 1001 & 1000 \\
\end{array} \]
Wanted MinSOP: \( f = \overline{y} \cdot \overline{z} + \overline{y} \cdot \overline{w} + x \cdot \overline{w} \)
Synthesis of multiple output functions

Nomenclature for functions with multiple outputs:

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>10</td>
</tr>
<tr>
<td>*11</td>
<td>*0</td>
</tr>
</tbody>
</table>

Meaning for the output part:
- 1: corresponding product term of input part is in on-set of this function component
- *: corresponding product term of input part is in DC-set of this function component
- 0: no statement about the corresponding product term of input part for this function component

Example (5):

\[ \begin{array}{c}
  x \\
  y \\
  z
\end{array} \xrightarrow{S_1} \begin{array}{c}
  f_1 \\
  f_2
\end{array} \quad f : \mathcal{B}^3 \to \mathcal{B}^2 \]

<table>
<thead>
<tr>
<th>SOP</th>
<th>MinSOP</th>
<th>Multiple implicant ( \bar{x}yz )</th>
<th>Irredundant Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( x\bar{y}z + yz )</td>
<td>( xz + yz + \bar{x}yz )</td>
<td>( xz + \bar{x}yz )</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( \bar{x}y + \bar{x}yz )</td>
<td>( \bar{x}y + \bar{x}\bar{z} + yz )</td>
<td>( \bar{x}\bar{z} + \bar{x}yz )</td>
</tr>
</tbody>
</table>

| Literals | 10 | 8 | 11 | 7 |

Steps:
1) Expand (implicants)
2) Reduce (implicants)
3) Remove (redundant implicants)

Expand : Implicant covers more minterms / consists of fewer literals (Minimization of literals)
Reduce : Implicant covers fewer minterms / consists of more literals (Preference for multiple implicants)
Remove : Removal of redundant implicants from \( \text{cov}(f) \) (Minimization of number of cubes)

Definition : An irredundant cover of \( f \) does not contain redundant implicants.

Then for each \( c_k \in \text{cov}(f) \):
\[ \text{cov}(f) \setminus \{c_k\} \neq \text{cov}(f) \]

Synthesis of Two-Level Combinational Circuits (11)
Realization with 2 MinSOPs

Realization with double implicants

Computation of double implicants from \( f_1 \) and \( f_2 \)

\[
\begin{align*}
f_1 &= \overline{x} \cdot y \cdot z + x \cdot y \cdot z + x \cdot \overline{y} \cdot z \\
f_2 &= \overline{x} \cdot y \cdot z + \overline{x} \cdot y \cdot z + \overline{x} \cdot \overline{y} \cdot z \\
f_1 \cdot f_2 &= (x \cdot z + y \cdot z) \cdot (\overline{x} \cdot y + \overline{x} \cdot z) \\
    &= y \cdot z \cdot \overline{x} \cdot y = \overline{x} \cdot y \cdot z \\
    &= \{11, 10\} \cdot \{01, 01\} = \{01, 11\}
\end{align*}
\]

Comment: SOP representations of Boolean functions (AND-OR-arrays) can be directly implemented in Programmable Logic Devices (PLDs). In case of a standard cell realization with library gates, NAND and NOR gates are preferred.

Equivalent Gates:

Realization with NAND Gates:

Synthesis of Two-Level Combinational Circuits (12)
Synthesis of Circuits from Incompletely Specified Functions

Example (6):

\[ f : \mathcal{B}^3 \rightarrow \mathcal{B}^2; \quad f(\hat{x}) = \hat{y} \iff (\hat{x}, \hat{y}) \in f \iff \hat{x} \rightarrow \hat{y} \]

\[
\begin{array}{c|ccc|c}
\hat{x}_1 & \hat{x}_2 & \hat{x}_3 & \hat{y}_1 & \hat{y}_2 \\
0 & * & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & * & 1 \\
* & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & * \\
\end{array}
\]

\[
\begin{align*}
F_1 & \subseteq f_1 \subseteq F_1 \cup D_1 \\
f_1 & = \{0 \ast 0, 101\}, \quad D_1 = \{001\} \\
f_1 & = \bar{x}_1 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \cdot x_3 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \\
F_2 & \subseteq f_2 \subseteq F_2 \cup D_2 \\
f_2 & = \{\ast 01, 010\}, \quad D_2 = \{011\} \\
f_2 & = \bar{x}_2 \cdot x_3 + \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot x_3 \\
\end{align*}
\]

\[ f := \{(\hat{x}, \hat{y}) \in \mathcal{B}^3 \times \mathcal{B}^2 | f(\hat{x}) = \hat{y}\} \]

Convention: tuple \((\hat{x}, \hat{y})\) with \(\hat{y} = 00\) will not be included in the set \(f\).

optional \(f = \{(0 \ast 0, 10), (101, 10), (001, \ast 1), (\ast 01, 01), (010, 01), (011, 0\ast)\}\)

optional

\[
\begin{align*}
\text{Expand} & \quad \text{Expand} \\
(001, 10), & \quad (001, 01) \\
(\ast 01, 10), & \quad (\ast 01, 01) \\
(01* , 01) & \quad (01*, 01) \\
\end{align*}
\]

\(\ast 01, 11\) multiple prime implicant

Synthesis of Two-Level Combinational Circuits (13)
4 Heuristic Minimization of Two-Level Combinational Circuits

Exact methods of minimization of combinational circuits with a large number of inputs (e.g. \( n > 16 \)) require very high computational effort (time, memory, resources), because the number of minterms and prime implicants grows exponentially with the number of variables. Therefore, heuristic methods have been developed for practical industrial application (e.g. ESPRESSO, MIS, SIS from University of California, Berkeley).

Combinatorial optimization (local search)

Configuration : feasible solution, \( \text{cov}(f) \)

Solution space (feasible region) : Set \( H \) of all configurations (all covers of \( f \))

\( H = \{1, \ldots, i, \ldots, N\}, \quad N < \infty \)

\( H \) is a set of numbers

Cost of a configuration : \( \varphi(i) \), e.g. number of literals in cover \( i \)

Objective function, cost function : \( \varphi : H \rightarrow \mathbb{R}^+ \)

Configuration \( j \) is global minimum : \( \iff \varphi(j) \leq \varphi(i) \), for all \( i \in H \):

\[ \forall i \in H \varphi(j) \leq \varphi(i) \]

Modification (Transition of configuration from \( i \) to \( j \) or from \( j \) to \( i \))

Important: feasibility check!

E.g.: In \( \text{cov}_i(f) \) one literal is removed from a product term, or in \( \text{cov}_j(f) \) one literal is added to a product term.

Set of all possible modifications : \( R \subseteq H \times H \), \( R \) is a binary relation

Configuration graph : \( G = (H, R) \), \( H \) : Set of vertices, \( R \) : Set of edges

Set of successor configurations of \( i \in H \)

Configuration \( i \) is a local minimum : \( \Gamma(i) = \emptyset \)

Arrow diagram of \( G = (H, R) \) :
Modification

Given : cov(f) or f in SOP-Form, \( F \subseteq f \subseteq F \cup D \)
\[ c \in \text{cov}(f) \; , \; [\text{cov}(f)] \setminus \{c\} = \text{cov}(h) \]
\[ f = c + h \; \; \; \text{NB:} \; \; c_l = c(l = 1) \; , \; c_l \cdot l = c \]

Wanted : successor or predecessor configuration of \( \text{cov}_i(f) \)

1. Expand (removal of a literal \( l \) from \( c \))
\[ \text{cov}_i(f) = \{c\} \cup \text{cov}(h) \rightarrow \{c_l\} \cup \text{cov}(h) = \text{cov}_j(f) \]
Feasibility conditions:
\[ c + h = c_l + h \]
\[ c + h = c_l \cdot l + c_l \cdot \bar{l} + h = c + c_l \cdot \bar{l} + h \]
\[ h = c_l \cdot \bar{l} + h \]
\[ c_l \cdot \bar{l} \subseteq h \]

2. Reduce (addition of a literal \( l \) to \( c \))
\[ \text{cov}_i(f) = \{c\} \cup \text{cov}(h) \rightarrow \{c \cdot l\} \cup \text{cov}(h) = \text{cov}_j(f) \]
Feasibility conditions:
\[ c + h = c \cdot l + h \]
\[ c \cdot l + c \cdot \bar{l} + h = c \cdot l + h \]
\[ c \cdot \bar{l} + h = h \]
\[ c \cdot \bar{l} \subseteq h \]

3. Remove (removal of a complete product term)
\[ \text{cov}_i(f) = \{c\} \cup \text{cov}(h) \rightarrow \text{cov}(h) = \text{cov}_j(f) \]
Feasibility conditions:
\[ c + h = h \]
\[ c \subseteq h \]

Checking for containment (as in 1., 2., and 3.) is very simple in special cases:
Case 1) \( c \subseteq h \), IF \( p \subseteq h \) AND \( c \subseteq p \) Absorption (A)
Case 2) \( c \subseteq h \), IF \( l \cdot c_1 + \bar{l} \cdot c_2 \subseteq h \) AND \( c \subseteq c_1 \cdot c_2 \) Resolution (R)

Heuristic minimization of two-level combinational circuits (2)
Comment on Combinatorial Optimization

The feasibility condition (containment check) can be expressed advantageously as a tautology check. Thus the problem of combinational optimization is transformed into a sequence of decision problems.

Proposition:

Let \( f : \mathbb{B}^n \rightarrow \mathbb{B} \) and \( c \in MC \) or \( c_1, c_2 \in MC \).

Then : \( c \subseteq f \iff f_c \equiv 1 \).

And : \( c_1 + c_2 \subseteq f \iff (f_{c_1} \equiv 1) \text{ AND } (f_{c_2} \equiv 1) \)

Example : \( f(x,a,b) = x \cdot a + \overline{x} \cdot b \); \( c = a \cdot b \)

\( a \cdot b \subseteq f \iff f_{ab} = 1 \iff f(x,1,1) = 1 \iff x + \overline{x} = 1 \)

Advantages of transforming a containment check into a tautology check:

a) By cofactoring a function, the function can be represented using fewer resources.

b) Proof of tautology is a standard problem which appears in many applications (SAT, satisfiability problem). It can be solved e.g. by the layer algorithm of the resolution method or by using binary decision diagrams (BDDs).

Proof of tautology for a function can recursively be reduced to proof of tautology for the cofactors of the function:

\[
\begin{align*}
 f(\overline{x}) = 1 & \iff (f_{x_i} = 1) \text{ AND } (f_{\overline{x}_i} = 1) \\
(f_{x_i} = 1) & \iff (f_{x_i \overline{x}_j}) = 1 \text{ AND } (f_{x_i} = 1) \text{ AND } (f_{\overline{x}_i} = 1) \text{ and so on}
\end{align*}
\]

If \( f \) is unate w.r.t. \( x_i \), then the proof of tautology can be simplified:

For \( f_{x_i} \subseteq f_{\overline{x}_i} : f(\overline{x}) = 1 \iff f_{\overline{x}_i} = 1 \),

for \( f_{\overline{x}_i} \subseteq f_{x_i} : f(x) = 1 \iff f_{x_i} = 1 \)

Example : \( f(x,y,z) = z \cdot (x \cdot \overline{y} + \overline{x} \cdot y) + \overline{x} \cdot \overline{y} + x \cdot y \)

From \( f_z \subseteq f_{\overline{x}} \text{ AND } f_z = \overline{x} + \overline{y} + x \cdot y = 1 \)

it follows \( f(x,y,z) = 1 \)

Heuristic minimization of two-level combinational circuits (3)
5 Synthesis of Multi-Level Combinational Circuits

Given : MinSOP of a two-level circuit
Wanted : Multi-level circuit with fewer gates / literals and more advantageous fanin / fanout structures

“General” representation of a combinational circuit

Primary inputs (PI) : $x_1, x_2, x_3, x_4, x_5, x_6, x_7$
Primary outputs (PO) : $y_1, y_2$
Internal signals (INT) : $z_1, z_2, z_3, z_4, z_5$

$z_1 = g_1(x_2, x_3, x_4)$; $z_4 = g_4(x_1, z_1)$; $y_1 = g_6(z_4, z_5)$
$z_2 = g_2(x_4, x_5, x_6)$; $z_5 = g_5(z_3, x_7)$; $y_2 = g_5(z_3, x_7)$
$z_3 = g_3(z_1, z_2)$

Chaining of subfunctions:

e.g. $y_2 = f_2(x) = g_5(g_3(g_1(x_2, x_3, x_4), g_2(x_4, x_5, x_6)), x_7)$
Boolean network $BN$ (of given 4-level circuit)

$BN = (V, E)$ is a directed acyclic graph (dag)

Set of vertices:

$V = V_{PI} \cup V_{INT} \cup V_{PO}$

$V_{PI} = \{x_1, \ldots, x_7\}$

$V_{PO} = \{y_1, y_2\}$

$V_{INT} = \{z_1, \ldots, z_5\}$

Set of edges:

$E = \{(x_1, z_4), \ldots, (z_5, y_2)\}$

- Directed edge $(z_2, z_3)$
- Initial vertex $z_2$
- Terminal vertex $z_3$

$(z_2, z_3) \in E \iff z_2$ is input signal and $z_3$ is output signal of the same module

Branch points: $x_4, z_1, z_5$ (fanout stems)

Reconvergence points: $z_3, y_1$

fanin($z_3$) = $\{z_1, z_2\}$ (predecessor of $z_3$)

transitive fanin($z_3$) = $\{z_1, z_2, x_2, x_3, x_4, x_5, x_6\}$ (ancestor of $z_3$)

(Limited) algebraic methods

Example (7):

\[
\begin{align*}
  y_1 &= y_2 + x_1 \cdot x_2 + x_1 \cdot \overline{x}_3 + x_1 \cdot x_4 \\
  y_2 &= x_2 \cdot \overline{x}_5 \cdot x_7 + x_2 \cdot x_6 \cdot x_7 + \overline{x}_3 \cdot \overline{x}_5 \cdot x_7 + \overline{x}_3 \cdot x_6 \cdot x_7 + x_4 \cdot x_7
\end{align*}
\]

By “clever” (systematic) factoring out we obtain:

\[
\begin{align*}
  y_1 &= y_2 + x_1 \cdot (x_2 + \overline{x}_3 + x_4) \\
  y_2 &= (x_2 + x_3 + x_4) \cdot (x_4 + \overline{x}_5 + x_6) \cdot x_7
\end{align*}
\]

Algebraic methods for synthesis of multi-level circuits have been developed especially at the University of California at Berkeley. They have been implemented in the tools MIS and SIS.

Synthesis of Multi-Level Combinational Circuits (2)
Functional Decomposition – Task

Given: Function \( f \) with \( |v| := |\text{set}(v)| = n \) inputs, and a partitioning of variables \( v \) into \( x \) and \( y \)

\[
V = \mathcal{B}^{|v|}, |V| = 2^{|v|} ; \quad W = \mathcal{B}^{|w|}, |W| = 2^{|w|}
\]

\[
X = \mathcal{B}^{|x|}, |X| = 2^{|x|} ; \quad Y = \mathcal{B}^{|y|}, |Y| = 2^{|y|}
\]

\[
|X \times Y| = |X| \cdot |Y| = 2^{|x|+|y|} = 2^{|v|}
\]

\[
w = f(v) ; \quad f : V \rightarrow W ; \quad \hat{v} \mapsto \hat{w}
\]

\[
w = f(x, y) \quad f : X \times Y \rightarrow W ; \quad (\hat{x}, \hat{y}) \mapsto \hat{w}
\]

Wanted: Decomposition of function \( f \) into functions \( g \) and \( h \)

\[
w = g(h(x), y) \quad g : Z \times Y \rightarrow W , \quad h : X \rightarrow Z
\]

\[
Z = \mathcal{B}^{|z|}, |Z| = 2^{|z|}
\]

Assumption: Disjoint partitioning of variables \( v \) into \( x \) and \( y \)

\[
\text{set}(x) \cup \text{set}(y) = \text{set}(v)
\]

\[
\text{set}(x) \cap \text{set}(y) = \emptyset
\]

Condition for advantageous decomposition:

\[
|z| \leq |x| - 1
\]

\[
|Z| \leq \frac{1}{2} |X|
\]

Comments:

The task of functional decomposition describes in a general manner the decomposition of combinatorial circuits into interlinked subcircuits (or the decomposition of a Boolean function into interlinked subfunctions, respectively).

In contrast to algebraic methods, all laws of Boolean algebra are used in functional decomposition.

The assumption of a disjoint partitioning of the variables does not limit the generality of this method, since e.g. common variables can be assigned completely to the set of bound variables. A beneficial partitioning of the variables can be determined e.g. by “quick exploration” using BDDs.
Functional Decomposition – Solution Steps

Example (8): \[ w = f(x, y) = f(x_1, x_2, x_3, y_1, y_2) \]
\[ w = x_1 x_3 \cdot \overline{y}_1 + x_3 \cdot \overline{y}_2 + x_1 x_2 x_3 \cdot y_1 + x_2 x_3 \cdot \overline{y}_1 + x_1 x_2 \cdot \overline{y}_2 + x_1 x_2 x_3 \cdot y_1 \]

Step 1: Evaluate \( f(\hat{x}_i, y) \) for \( i = 0, 1, \ldots, 2^3 - 1 \)

<table>
<thead>
<tr>
<th>((\hat{x}_1, \hat{x}_2, \hat{x}_3))</th>
<th>(f(\hat{x}_1, y))</th>
<th>(y_1)</th>
<th>(y_1 + \overline{y}_2)</th>
<th>(\overline{y}_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(00)</td>
<td>1 1 1 1 1 1 1</td>
<td>(\overline{y}_1)</td>
<td>(y_1 + \overline{y}_2)</td>
<td>(\overline{y}_2)</td>
</tr>
<tr>
<td>(01)</td>
<td>1 1 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10)</td>
<td>0 0 0 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(11)</td>
<td>0 0 0 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decomposition table for \( f(x, y) \)

\[ f(\hat{x}_i, y) = f(\hat{x}_j, y) \iff \hat{x}_i \sim \hat{x}_j ; \quad \hat{x}_i \text{ and } \hat{x}_j \text{ are equivalent, i.e. are elements of an equivalence class } X_k \text{ of the partition } \Pi \text{ of } X. \]

\[ \Pi(X) = \{X_1, X_2, X_3\} \]
\[ = \{\{000,010,100\}, \{001,111\}, \{011,101,110\}\} \]

Definition for \( \Pi(X) = \{X_1, X_2, X_3\} \)

\[ \Pi(X) \subseteq P(X) \text{ is a partition of } X \neq \emptyset \text{ iff} \]
\[ X_1, X_2, X_3 \neq \emptyset, X_1 \cup X_2 \cup X_3 = X \]
and \( X_1 \cap X_2, X_1 \cap X_3, X_2 \cap X_3 = \emptyset \)
(pairwise disjoint classes).

\( X_k \subseteq X, X_k \in P(X) : |\Pi(X)| = l = 3 ; \quad l \text{ is number of equivalence classes} \)

Result of step 1:

\[ w = f(x, y) = (m_0 + m_2 + m_4) \cdot \overline{y}_1 + (m_1 + m_7) \cdot (y_1 + \overline{y}_2) + (m_3 + m_5 + m_6) \cdot \overline{y}_2 \]

Synthesis of Multi-Level Combinational Circuits (4)
Step 2 : Construct the decomposition function \( h(x) \)

\[
\begin{align*}
z &= h(x), & h : X \rightarrow Z; & X = B^3, Z = B^{\lceil z \rceil} \\
\end{align*}
\]

Decomposition condition: \( l \leq 2^{\lceil z \rceil} \leq \frac{1}{2} \cdot 2^{|x|} \)

\(|z| = 2\), since \( l = 3 \) and \(|x| = 3\)

\[
\begin{align*}
z &= h(x) & \triangleq (z_1, z_2) = (h_1(x), h_2(x)) \\
z_1 &= h_1(x), & z_2 &= h_2(x) \\
\end{align*}
\]

value assignment : \( X \rightarrow Z \)

The mapping of value assignments \((\hat{z}_1, \hat{z}_2)_k\) with \( k = 0, 1, 2, 3 \), i.e. of 00, 01, 10, 11, to the available positions in \( Z \) is called coding.

Assignment condition:

\[
\begin{align*}
\text{IF } \hat{x}_i \sim \hat{x}_j, & \text{ THEN } h(\hat{x}_i) \neq h(\hat{x}_j) \\
\text{IF } \hat{x}_j \sim \hat{x}_j, & \text{ THEN } \hat{x}_k \neq \hat{x}_r, & k, r = 0, 1, 2, 3 \\
\hat{x}_i \sim \hat{x}_j : & \hat{x}_i \text{ and } \hat{x}_j \text{ are not equivalent} \\
\end{align*}
\]

Number of possible assignments (injective functions): \( l! \), if \( l = 2^{\lceil z \rceil} \)
Coding 1:

<table>
<thead>
<tr>
<th>$\hat{z}_i \in X$</th>
<th>$\hat{z}_k \in Z$</th>
<th>$z = h(x)$</th>
<th>$g(\hat{z}_k, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000, 010, 100</td>
<td>00</td>
<td>$z_1 \cdot z_2 = m_0 + m_2 + m_4$</td>
<td>$\overline{y}_1$</td>
</tr>
<tr>
<td>001, 111</td>
<td>10</td>
<td>$z_1 \cdot z_2 = m_1 + m_7$</td>
<td>$y_1 + \overline{y}_2$</td>
</tr>
<tr>
<td>011, 101, 110</td>
<td>11</td>
<td>$z_1 \cdot z_2 = m_3 + m_5 + m_6$</td>
<td>$\overline{y}_2$</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>$\hat{z}_1 \cdot z_2 = 1$</td>
<td>not possible</td>
</tr>
</tbody>
</table>

Assignment table

$z_1 = z_1 \cdot \hat{z}_2 + z_1 \cdot z_2 = m_1 + m_7 + m_3 + m_5 + m_6$

$z_2 = \hat{z}_1 \cdot z_2 + z_1 \cdot z_2 = m_3 + m_5 + m_6$

$z_1 = h_1(x) = x_1 \cdot x_2 + x_3$

$z_2 = h_2(x) = \overline{x}_1 \cdot x_2 \cdot x_3 + x_1 \cdot \overline{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x}_3$

Coding 2:

<table>
<thead>
<tr>
<th>$\hat{z}_i \in X$</th>
<th>$\hat{z}_k \in Z$</th>
<th>$z = h(x)$</th>
<th>$g(\hat{z}_k, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000, 010, 100</td>
<td>00</td>
<td>$z_1 \cdot z_2 = m_0 + m_2 + m_4$</td>
<td>$\overline{y}_1$</td>
</tr>
<tr>
<td>001, 111</td>
<td>10</td>
<td>$z_1 \cdot z_2 = m_1 + m_7$</td>
<td>$y_1 + \overline{y}_2$</td>
</tr>
<tr>
<td>011, 101</td>
<td>11</td>
<td>$z_1 \cdot z_2 = m_3 + m_5$</td>
<td>$\overline{y}_2$</td>
</tr>
<tr>
<td>110</td>
<td>01</td>
<td>$z_1 \cdot z_2 = m_6$</td>
<td>$\overline{y}_2$</td>
</tr>
</tbody>
</table>

Assignment table

$z_1 = z_1 \cdot \hat{z}_2 + z_1 \cdot z_2 = m_1 + m_7 + m_3 + m_5$

$z_2 = \hat{z}_1 \cdot z_2 + z_1 \cdot z_2 = m_3 + m_5 + m_6$

$z_1 = h_1(x) = x_3$

$z_2 = h_2(x) = \overline{x}_1 \cdot x_2 \cdot x_3 + x_1 \cdot \overline{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x}_3$
Step 3: Construct the composition function $g(z, y)$

Coding 1:

$$w = g(z, y) = z_1z_2 \cdot y_1 + z_1z_2 \cdot (y_1 + \overline{y}_2) + z_1z_2 \cdot \overline{y}_2$$

$$= z_1z_2 \cdot y_1 + z_1z_2 \cdot y_1 + z_1 \cdot \overline{y}_2$$

Since $z_1z_2 = 1$, i.e. the value assignment $(\hat{z}_1, \hat{z}_2) = 01$ cannot be obtained from any value assignment $\hat{x}$ to the input variable, every product term which contains $z_1z_2$ can be added to $w$ (internal don’t care values).

$$w = g(z, y) = \overline{z}_1 \overline{z}_2 \cdot \overline{y}_1 + (\overline{z}_1z_2 \cdot \overline{y}_1) + z_1z_2 \cdot y_1 + z_1 \cdot \overline{y}_2$$

$$= \overline{z}_1 \cdot \overline{y}_1 + z_1z_2 \cdot y_1 + z_1 \cdot \overline{y}_2$$

Coding 2:

$$w = g(z, y) = \overline{z}_1 \overline{z}_2 \cdot \overline{y}_1 + \overline{z}_1 \overline{z}_2 \cdot y_1 + z_1z_2 \cdot \overline{y}_2 + z_1 \cdot \overline{y}_2$$

$$w = g(z, y) = \overline{z}_1 \overline{z}_2 \cdot \overline{y}_1 + z_1z_2 \cdot y_1 + z_1 \cdot \overline{y}_2 + z_2 \cdot \overline{y}_2$$

Circuit realization for example (8) (Coding 1):

$$w = f(x, y) = f(x_1, x_2, x_3, y_1, y_2)$$

$$w = \overline{x}_1 \overline{x}_3 \cdot \overline{y}_1 + x_2 \cdot \overline{y}_2 + \overline{x}_1x_2x_3 \cdot y_1 + \overline{x}_2x_3 \cdot \overline{y}_1 + x_1x_2 \cdot \overline{y}_2 + x_1x_2x_3 \cdot y_1$$

Synthesis of Multi-Level Combinational Circuits (7)
6 OBDD: Ordered Binary Decision Diagram

Every Boolean function \( f \) can be represented by a recursive Shannon expansion.

E.g. \( w = f(x, y) = x \cdot f_x + \bar{x} \cdot f_{\bar{x}} = \beta(x, f_x, f_{\bar{x}}) = \beta(x, f(1, y), f(0, y)) \)
\[
\begin{align*}
\beta & = x \cdot (y \cdot f_{xy} + \bar{y} \cdot f_{\bar{y}x}) + \bar{x} \cdot (y \cdot f_{y\bar{x}} + \bar{y} \cdot f_{\bar{y}\bar{x}}) \quad \text{(Difference to SOP form!)} \\
w & = \beta \left[ x, \beta \left( y, f(1, 1), f(1, 0) \right), \beta \left( y, f(0, 1), f(0, 0) \right) \right]
\end{align*}
\]

\( \text{OBDD}(f) = (V,E) \) is a directed acyclic graph (dag) which recursively defines a Boolean function \( f \).

\[
\begin{array}{c}
\text{Shannon expansion} \\
\text{as root tree}
\end{array}
\begin{array}{c}
\text{Example for OBDDs:}
\end{array}
\begin{array}{c}
f(x) = x \\
f(x) = \bar{x} \\
f(x, y) = x \cdot y \\
f(x, y) = x + y \\
f(x, y) = x \oplus y
\end{array}
\]

Example (9): \( f(x,y,z) = x \cdot \bar{y} \cdot z + y \cdot z \)

\( \text{OBDD}(f) = (V,E) \) with variable ordering \( z < y < x \) is a acyclic root graph with a root vertex (vertex without a predecessor). Each non-terminal vertex (vertex with at least one successor) \( v \in V \) has an attribute index \( \text{index}(v) \in \{1,2,3\} \) associated, which points to a variable from \( \{x,y,z\} \), and two direct successors \( \text{high}(v), \text{low}(v) \in V \).

Each terminal vertex \( V \) has an attribute value \( \text{value}(v) \in \{0,1\} \) associated.

For the edges between non-terminal vertices \((v, \text{high}(v)) \lor (v, \text{low}(v)) \in E \) the following holds:

\( \text{index}(v) < \text{index}(\text{high}(v)) \)

respectively \( \text{index}(v) < \text{index}(\text{low}(v)) \)

\( \text{OBDD}: \text{Ordered Binary Decision Diagram} \)
Simplification Rules (SR1, SR2) for tree-like OBDDs

SR1 : Specialized rule of resolution
For $f(x, y, z)$:
For $f(1, y, z) = f(0, y, z) = g(y, z)$,
THEN $f(x, y, z) = g(y, z)$

SR2 : Merging of equivalent Sub-OBDDs

Comment : Exhaustive application of SR1, SR2 results in a Reduced OBDD (ROBDD).

Proposition by R. Bryant, CMU, 1986:
The representation of a Boolean function $f(x)$ by a ROBDD $(f)$ using a given variable ordering $x_1 \prec x_2 \prec \cdots \prec x_n$ is canonical.

It follows : $\text{ROBDD}(g) \sim \text{ROBDD}(h) \iff g(x) = h(x)$

Definition:
Two ROBDDs are equivalent, IF:
- Structural equivalence (Isomorphism)
- Same assignment of values to terminal vertices
- Same assignment of index values for non-terminal vertices
Example (9): \( f(x, y, z) = x \cdot \overline{y} \cdot z + y \cdot z, \ x \prec y \prec z \)

Example (2): \( f(x, y, z) = \overline{x} \cdot y \cdot \overline{z} + \overline{x} \cdot y + y \cdot z \)

Example (8): \( f(x_1, x_2, x_3, y_1, y_2) = x_1 \cdot \overline{x}_3 \cdot \overline{y}_1 + x_3 \cdot \overline{y}_2 + \overline{x}_1 \cdot x_2 \cdot x_3 \cdot y_1 + \overline{x}_2 \cdot \overline{x}_3 \cdot \overline{y}_1 + x_1 \cdot x_2 \cdot \overline{y}_2 + x_1 \cdot x_2 \cdot x_3 \cdot y_1 \)

OBDD: Ordered Binary Decision Diagram (3)
Since about 1990, the representation of Boolean functions by binary decision diagrams has improved the effectiveness of design and verification of digital systems significantly.

Properties of ROBDDs:

+ Compact representation of Boolean functions
  Only representation where size (number of vertices) usually does not grow exponentially with the number of variables. Functions depending on hundreds of variables can be represented.

+ Unique (canonical) representation
  Equivalence of two Boolean functions can be shown by isomorphism of their respective ROBDDs. Verification possible by equivalence checks.

+ Satisfiability check and tautology proof very simple
  Satisfiability: at least one terminal vertex has value 1
  Tautology: all terminal vertices have value 1 (function reduces to just one terminal vertex in this case)

+ Fast (partial) evaluation of function \( f \) possible
  Cofactors are easily accessible in ROBDD\((f)\)

+ Negation of \( f \) trivial
  Negation of values in terminal vertices

+ Boolean operations very simple on ROBDDs
  Simple computation of e.g. ROBDD\((f_1 \cdot f_2)\) from ROBDD\((f_1)\) and ROBDD\((f_2)\)

+ Composition of functions can be represented

+ Improvement of some discrete algorithms possible
  E.g. for covering problems, solution time is proportional to size of problem representation

+ Much larger state spaces of FSM can be treated
  – ROBDD size strongly dependent on variable ordering
    Determination of a favorable variable ordering not always easy
  – ROBDD size grows exponentially with number of variables for some functions
    E.g. for multiplier functions
  – SOP representation closer to implementation for some realization technologies (e.g. PLDs)
7 Synthesis of Sequential Circuits

Sequential Circuit (example)

DQ

11000110... 00101001...

Clock period

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(t)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>y(t)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>

\[
z(t) = y(t) = \overline{s(t) + x(t)}, \quad t \in \{0, 1, 2, 3, \ldots\}
\]

State: \( s(t) = z(t - 1) = y(t - 1), \quad y(-1) \triangleq s(0) \)
Next State: \( z(t) = s(t + 1) \)
\( y(t) = x(t) + y(t - 1) \)

Iterative combinational circuit (Alternative to sequential circuit)

\[
y(t) = f(s(0), x(0), x(1), \ldots, x(t)), \quad t \in \{0, 1, 2, 3, \ldots\}
\]
(Circuit with memory!)

Synthesis of Sequential Circuits (1)
Finite State Machine FSM (as a model of sequential circuits)
(Automaton with output)

FSM = (S, I, O, δ, λ, S₀) 6-tuple

1) S = {S₀, S₁, ..., Sᵣ₋₁} = \mathcal{B}^r : finite set of states, 0 < |S| < ∞
   = {S₀, S₁, ..., Sᵣ₋₁}
2) I = {I₀, I₁, ..., Iᵣ₋₁} = \mathcal{B}^n : set of input patterns
   = {X₀, X₁, ..., Xᵣ₋₁}
   input alphabet
3) O = {O₀, O₁, ..., Oᵣ₋₁} = \mathcal{B}^m : set of output patterns
   = {Y₀, Y₁, ..., Yᵣ₋₁}
   output alphabet
4) δ : S × I → S, (s, x) → z : (state-) transition function
5) λ : S × I → O, (s, x) → y : output function
6) S₀ ∈ S : initial state

Types of machines:
- Mealy automaton λ : S × I → O, y = λ(s, x)
- Moore automaton λ : S → O, y = λ(s)

Synthesis of Sequential Circuits (2)
General description of FSM

\[ X_j : \text{Input pattern} \quad Y_l : \text{Output pattern} \]
\[ S_i : \text{State} \quad S_k : \text{Next state} \quad i, j, k, l \in \{0, 1, 2, 3, \ldots \} \]

<table>
<thead>
<tr>
<th>\delta</th>
<th>\cdots</th>
<th>X_j</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>\vdots</td>
<td></td>
<td>\vdots</td>
<td></td>
</tr>
</tbody>
</table>
| \begin{array}{c}
S_i \\
\vdots
\end{array} | | \begin{array}{c}
S_k \\
\vdots
\end{array} | \cdots |

\begin{align*}
\delta & : S \times I \rightarrow S \\
S_k &= \delta(S_i, X_j)
\end{align*}

state transition function, next-state function

<table>
<thead>
<tr>
<th>\lambda</th>
<th>\cdots</th>
<th>X_j</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>\vdots</td>
<td></td>
<td>\vdots</td>
<td></td>
</tr>
</tbody>
</table>
| \begin{array}{c}
S_j \\
\vdots
\end{array} | | \begin{array}{c}
Y_l \\
\vdots
\end{array} | \cdots |

<table>
<thead>
<tr>
<th>\lambda</th>
<th>\cdots</th>
<th>X_j</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>\vdots</td>
<td></td>
<td>\vdots</td>
<td></td>
</tr>
</tbody>
</table>
| \begin{array}{c}
S_j \\
\vdots
\end{array} | | \begin{array}{c}
Y_l \\
\vdots
\end{array} | \cdots |

\begin{align*}
\lambda & : S \times I \rightarrow O \\
Y_l &= \lambda(S_i, X_j)
\end{align*}

output function

<table>
<thead>
<tr>
<th>\mu</th>
<th>\cdots</th>
<th>X_j</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>\vdots</td>
<td></td>
<td>\vdots</td>
<td></td>
</tr>
</tbody>
</table>
| \begin{array}{c}
S_i \\
\vdots
\end{array} | | \begin{array}{c}
(S_k, Y_l) \\
\vdots
\end{array} | \cdots |

<table>
<thead>
<tr>
<th>\mu</th>
<th>\cdots</th>
<th>X_j</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>\vdots</td>
<td></td>
<td>\vdots</td>
<td></td>
</tr>
</tbody>
</table>
| \begin{array}{c}
S_i \\
\vdots
\end{array} | | \begin{array}{c}
(S_k, Y_l) \\
\vdots
\end{array} | \cdots |

\begin{align*}
\mu & : S \times I \rightarrow S \times O \\
(S_k, Y_l) &= \mu(S_i, X_j)
\end{align*}

state output function

Synthesis of Sequential Circuits (3)
Example (10): \( I = O = \{0, 1\} \), \( n = m = 1; \) \( S = \{S_1, S_2, S_3\} \), \( S^0 = S_1 \)
\( (S_j, y) = \mu(S_i, \hat{x}) \), \( i, j = 1, 2, 3 \)

\[ \begin{array}{c|cc}
\mu & x = 0 & x = 1 \\
\hline
S_1 & (S_1, 0) & (S_2, 1) \\
S_2 & (S_1, 0) & (S_3, 0) \\
S_3 & (S_2, 0) & (S_1, 1) \\
\end{array} \]

state output table

\[ \begin{array}{c|cc}
x & s_1 & s_2 \\
\hline
0 0 0 & 0 0 0 \\
1 0 0 & 0 1 1 \\
0 0 1 & 0 0 0 \\
1 0 1 & 1 0 0 \\
0 1 0 & 1 0 0 \\
1 1 0 & 0 0 1 \\
0 1 1 & (1) * * (0) & * (0) \\
1 1 1 & (1) * * (0) & * (0) \\
\end{array} \]
coded cube table

\[ \begin{array}{c|cc}
x & z_1 & z_2 & y \\
\hline
0 0 0 & 00 & 0 & 0 \\
1 0 0 & 01 & 1 & 1 \\
0 0 1 & 00 & 0 & 0 \\
1 0 1 & 10 & 0 & 0 \\
0 1 0 & 10 & 0 & 0 \\
1 1 0 & 00 & 1 & 1 \\
0 1 1 & (1) * * (0) & * (0) & 0 \\
1 1 1 & (1) * * (0) & * (0) & 1 \\
\end{array} \]
cube table

\[ \begin{array}{c|cc}
\hat{x} \text{ state} & \text{next state} & \hat{y} \\
\hline
0 & S_1 & S_1 & 0 \\
1 & S_1 & S_2 & 1 \\
0 & S_2 & S_1 & 0 \\
1 & S_2 & S_3 & 0 \\
0 & S_3 & S_3 & 0 \\
1 & S_3 & S_1 & 1 \\
\end{array} \]
cube table

FSM for example (10):

FSM processes strings (words, concatenation of symbols). Starting with the initial state
\( S_1 \triangleq 00 \triangleq \bar{s}_1 \cdot \bar{s}_2 \), e.g. the string
\[ \text{str}(x) = x(0) \circ x(1) \circ x(2) \circ x(3) \circ x(4) \circ x(5) \]
\[ = 110101 \]
is transformed into the string
\[ \text{str}(y) = y(0) \circ y(1) \circ y(2) \circ y(3) \circ y(4) \circ y(5) \]
\[ = 100101 \]
one symbol at a time. The sequential system cycles through a number of system states during this processing.
\[ \text{str}(S) = S(0) \circ S(1) \circ S(2) \circ S(3) \circ S(4) \circ S(5) \circ S(6) \]
\[ = S_1 S_2 S_3 S_1 S_1 S_2 \quad (\text{see STG}) \]

Remark: State sequences or strings with an initial state describe paths in the state transition graph STG.
Operations on strings

Assumption: $I = O = \{0, 1\}, \quad n = m = 1, \quad$ Extension to $n > 1$ or $m > 1$ quite simple: $x \rightarrow x$ or $y \rightarrow y$

Notation
- $x(t) \triangleq x', \quad y(t) \triangleq y', \quad S(t) \triangleq S'$

String
- $\text{str}(x) = x(0) \circ x(1) \circ x(2) \circ \ldots \triangleq x^0 x^1 x^2 \ldots$

Length of string
- $k = |\text{str}(x)|$

String of length $k$
- $\text{str}(x)|_k$

Empty string
- $\varepsilon, \varepsilon = \text{str}(x)|_0$, e.g. $\varepsilon \circ \text{str}(x) = \text{str}(x)$

Extension of interpretation of $y = \lambda(S_i, x)$ to $\text{str}(y) = \lambda(S_i, \text{str}(x))$

\[ e.g. \quad y^0 y^1 y^2 = \lambda(S^0, x^0 x^1 x^2) = \lambda(S^0, x^0) \circ \lambda(\delta(S^0, x^0), x^1 x^2) = y^0 \circ \lambda(S^1, x^1 x^2) = y^0 y^1 \circ \lambda(S^2, x^2) \]

A directed path in state transition graph STG

\[ \hat{x}/\hat{y} \in \{0/0, 0/1, 1/0, 1/1\} \]

Extension of interpretation of $S^{t+1} = \delta(S^t, x')$ to $S^k = \delta(S^0, \text{str}(x)|_k)$

\[ e.g. \quad S^0 S^1 S^2 S^3 = S^0 \circ \delta(S^0, x^0) \circ \delta(S^0, x^0 x^1) \circ \delta(S^0, x^0 x^1 x^2) \]

Initial configuration of FSM
- $(S^0, \text{str}(x), \varepsilon)$

Final configuration of FSM
- $(S^r, \varepsilon, \text{str}(y))$

Configuration, situation of FSM
- $(S', \text{str}(x), \text{str}(y)), \quad S' : \text{current state}$
- $\text{str}(x) : \text{current input string}$
- $\text{str}(y) : \text{current output string}$

Configuration transition:  

\[ \begin{array}{c}
\text{e.g.} \quad (S^0, x^0 x^1 x^2 x^3, \varepsilon) \quad \rightarrow \quad (S^1, x^1 x^2 x^3, y^0) \\
\quad \rightarrow \quad (S^2, x^2 x^3, y^0 y^1) \\
\quad \rightarrow \quad (S^4, \varepsilon, y^1 y^2 y^3) \\
\quad \rightarrow \quad (S^k, x^k x^{k+1} \ldots, y^0 y^1 y^2 \ldots y^{k-1}) \\
\quad \rightarrow \quad (S^*, \varepsilon, y^1 y^2 \ldots y^*) 
\end{array} \]

Synthesis of Sequential Circuits (5)
State Minimization

The first step in state minimization is to eliminate non-reachable states.

Definition: A state $S_j$ is reachable from an initial state $S_i$, if a directed path from $S_i$ to $S_j$ exists in the state transition graph.

The key task of state minimization is to reduce the number of states by merging equivalent states.

Definition of equivalence of two states $S_i$ and $S_j$:

$$S_i \sim S_j \iff \lambda(S_i, \text{str}(x)) = \lambda(S_j, \text{str}(x)), \text{ for all } \text{str}(x)$$

Equivalent (non differentiable) identical output patterns for identical input patterns, originating from $S_i$ and $S_j$

The equivalence relation $\sim$ partitions the state set $S$ into disjoint subsets (equivalence classes). $S_i$ and $S_j$ are elements of an equivalence class $K_l$ of the partition $\Pi$ of $S$. $\Pi \subseteq P(S)$

$$\Pi(S) = \{K_1, \ldots, K_l, \ldots, K_L\}$$

Remark: $S_i \not\sim S_j \iff$ there exists a $\text{str}(x)$, such that $\lambda(S_i, \text{str}(x)) \neq \lambda(S_j, \text{str}(x))$

differentiable different output patterns

Definition of $k$-equivalence of two states $S_i$, $S_j$:

$$S_i \sim^k S_j \iff \lambda(S_i, \text{str}(x)|_k) = \lambda(S_j, \text{str}(x)|_k), \text{ for all } \text{str}(x)|_k$$

$$\Pi^{(k)}(S) = \{K_1^{(k)}, \ldots, K_l^{(k)}, \ldots, K_L^{(k)}\}; \quad S_i, S_j \in K_l^{(k)}$$

$$S_i \sim^1 S_j \iff \lambda(S_i, x^0) = \lambda(S_j, x^0), \text{ for all } x^0 \in \{1, 0\}$$

$$S_i \sim^1 S_j \iff [\lambda(S_i, 0) = \lambda(S_j, 0)] \land [\lambda(S_i, 1) = \lambda(S_j, 1)]$$

The iterative computation of $k$-equivalent (or simply: equivalent) states starts with the computation of the 1-equivalent states, since:

$$S_i \sim^0 S_j \iff \lambda(S_i, \varepsilon) = \lambda(S_j, \varepsilon) \iff \varepsilon = \varepsilon, \text{ for all } S_i, S_j \in S, \quad \Pi^{(0)} = \{S\}$$

coarsest partition of $S$
Example (11): $I = O = \{0, 1\}$, $n = m = 1$; $S = \{S_1, S_2, S_3, S_4\}$, $S^0 = S_1$ 
$(S_i, \hat{y}) = \mu(S_i, \hat{x})$, $i, j = 1, 2, 3, 4$

<table>
<thead>
<tr>
<th>$\mu$</th>
<th>$x = 0$</th>
<th>$x = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$(S_4, 0)$</td>
<td>$(S_2, 1)$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$(S_4, 0)$</td>
<td>$(S_3, 0)$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$(S_3, 0)$</td>
<td>$(S_1, 1)$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$(S_1, 0)$</td>
<td>$(S_2, 1)$</td>
</tr>
</tbody>
</table>

From the state output table we obtain the following relations:

$S_1 \sim S_3 \iff [0 = 0] \land [1 = 1]$, $S_1 \sim S_4 \quad (S_3 \sim S_4)$

$S_1 \not\sim S_2 \iff [0 = 0] \land [1 = 0]$, $S_1 \not\sim S_2 \quad (S_3 \not\sim S_2, S_4 \not\sim S_2)$

$\Pi(1) = \{\{S_1, S_3, S_4\}, \{S_2\}\}$

$S_i \overset{2}{\sim} S_j \iff [S_i \overset{1}{\sim} S_j] \land [\delta(S_i, 0) \overset{1}{\sim} \delta(S_j, 0)] \land [\delta(S_i, 1) \overset{1}{\sim} \delta(S_j, 1)]$

e.g. $S_1 \overset{2}{\sim} S_4 \iff [S_1 \overset{1}{\sim} S_4] \land [S_4 \overset{1}{\sim} S_1] \land [S_2 \overset{1}{\sim} S_2]$

$S_1 \not\overset{\text{false}}{\sim} S_3 \iff [S_1 \overset{1}{\sim} S_4] \land [S_4 \overset{1}{\sim} S_3] \land [S_2 \overset{1}{\sim} S_1]$, $S_1 \not\overset{\text{false}}{\sim} S_3$

$\Pi(2) = \{\{S_1, S_4\}, \{S_3\}, \{S_2\}\}$

$S_1 \overset{3}{\sim} S_4 \iff [S_1 \overset{2}{\sim} S_4] \land [\delta(S_1, 0) \overset{2}{\sim} \delta(S_4, 0)] \land [\delta(S_1, 1) \overset{2}{\sim} \delta(S_4, 1)]$

$\iff [S_1 \overset{2}{\sim} S_4] \land [S_4 \overset{2}{\sim} S_1] \land [S_2 \overset{2}{\sim} S_2]$

$\Pi(3) = \Pi(2) = \Pi(S)$

Result: state $S_1$ and $S_4$ are equivalent and can be merged in STG (see example (10)).

<table>
<thead>
<tr>
<th>$\mu$</th>
<th>$x = 0$</th>
<th>$x = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$(S_4, 0)$</td>
<td>$(S_2, 1)$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$(S_1, 0)$</td>
<td>$(S_2, 1)$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$(S_3, 0)$</td>
<td>$(S_1, 1)$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$(S_4, 0)$</td>
<td>$(S_3, 0)$</td>
</tr>
</tbody>
</table>

Synthesis of Sequential Circuits (7)
Iterative checking for equivalent states in an FSM:

\[ S_i \sim_k^{k+1} S_j \iff [S_i \sim S_j] \land [\delta(S_i, x^0) \sim_k \delta(S_j, x^0)], \quad \text{for all } x^0 \in \{0, 1\} \]

\[ S_i \sim_k^{k+1} S_j \iff [S_i \sim S_j] \land [\delta(S_i, 0) \sim_k \delta(S_j, 0)] \land [\delta(S_i, 1) \sim_k \delta(S_j, 1)] \]

Comment: \( S_i \sim_k^{k+1} S_j \iff [S_i \sim S_j] \land [\delta(S_i, 0) \sim_k \delta(S_j, 0)] \land [\delta(S_i, 1) \sim_k \delta(S_j, 1)] \)

Determination of equivalent states:

\[ \Pi = \Pi^{(k)}, \quad \text{IF} \quad \Pi^{(k+1)} = \Pi^{(k)}, \quad \text{with} \quad k \leq |S| - 1 \]

Additional remark: \( \Pi^{(k)} \) is a refinement of \( \Pi^{(k-1)}; \quad \Pi^{(k)}, \Pi^{(k-1)}, \ldots, \Pi^{(2)}, \Pi^{(1)} \)

Because \( S_i \sim_k S_j \implies S_i \sim_k S_j \implies \ldots \implies S_i \sim_k S_j \implies S_i \sim_k S_j \)

\( S_i \sim S_j \implies S_i \sim_k S_j \implies S_i \sim_k S_j \implies \ldots \implies S_i \sim_k S_j \)

Example (12): Completely specified FSM; \( I = O = \{0, 1\}, \quad n = m = 1; \)

\( S = \{A, B, C, D, E, F, G\} \)

<table>
<thead>
<tr>
<th>( \mu )</th>
<th>( x = 0 )</th>
<th>( x = 1 )</th>
<th>( \Pi^{(1)} )</th>
<th>( \Pi^{(2)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>((E, 0))</td>
<td>((C, 0))</td>
<td>{(A, D, G, F), {B, C}, {E}}</td>
<td>{(A), {D, G}, {F}, {B, C}, {E}}</td>
</tr>
<tr>
<td>D</td>
<td>((G, 0))</td>
<td>((A, 0))</td>
<td>{(A, D, G, F), {B, C}, {E}}</td>
<td>{(A), {D, G}, {F}, {B, C}, {E}}</td>
</tr>
<tr>
<td>G</td>
<td>((D, 0))</td>
<td>((G, 0))</td>
<td>{(A, D, G, F), {B, C}, {E}}</td>
<td>{(A), {D, G}, {F}, {B, C}, {E}}</td>
</tr>
<tr>
<td>F</td>
<td>((E, 0))</td>
<td>((D, 0))</td>
<td>{(A, D, G, F), {B, C}, {E}}</td>
<td>{(A), {D, G}, {F}, {B, C}, {E}}</td>
</tr>
<tr>
<td>B</td>
<td>((C, 0))</td>
<td>((A, 1))</td>
<td>{(A, D, G, F), {B, C}, {E}}</td>
<td>{(A), {D, G}, {F}, {B, C}, {E}}</td>
</tr>
<tr>
<td>C</td>
<td>((B, 0))</td>
<td>((G, 1))</td>
<td>{(A, D, G, F), {B, C}, {E}}</td>
<td>{(A), {D, G}, {F}, {B, C}, {E}}</td>
</tr>
</tbody>
</table>

\( \text{state output table} \)

(appropriately sorted)

Result: All states can be pairwise differentiated by appropriate strings with \( k \leq 3 \).

Remark: Typically, the state output table will be sorted again after each partitioning step.

Synthesis of Sequential Circuits (8)
General comments on machine models

Machine types:
- Mealy Automaton: \( \lambda: S \times I \to O \), \( Y = \lambda(S, X) \)
- Moore Automaton: \( \lambda: S \to O \), \( Y = \lambda(S) \)
- Medwedew Automaton: \( \lambda: I \to O \), \( Y = \lambda(X) \)

Properties of machines:
- completely specified – incompletely specified
- simplified
- minimal
- deterministic – non-deterministic

In a non-deterministic machine the state transition is not necessarily unambiguous anymore.

FSM design flow:

Synthesis of Sequential Circuits (9)
II. Simulation of Digital Circuits

1 Introduction

- Why use simulation?
  1. Hardware design validation (logic function, timing, power consumption)
  2. Fault simulation (fault coverage)
  3. Early co-validation of system hardware and software

- Simulation levels:

```
Device       Circuit           Switch-Level    Logic    Register-Transfer    Algorithm    System
```

- Abstraction levels:

```
Process     Circuit           Switch-Level    Logic    Register-Transfer    Algorithm    System
```

- Alternative/Enhancement: Formal (mathematical) verification

2 Logic Simulation

- Overview of simulation system:

```
Netlist     Stimuli           Simulator
```

```
Component models
- logic function
- timing behavior
```

```
Signal traces
```

```
# Transistors
1 10 10^2 10^3 10^4 10^5 10^6 10^7 10^8 10^9 10^10
```
Circuit Modeling

The circuit is represented by basic components of the simulation system

Modeling functionality and timing:

\[ y'(t) = x'(t) - \tau \]

\( \Delta \) : basic time unit

\( \tau \) : (signal-) delay

\( \tau_d = n \cdot \Delta \) : propagation delay (gate delay)

\( \tau_w = m \cdot \Delta \) : wire delay

Delay dependent effects:

a) Race: “run” between signal changes on different wires leading to a single component

b) Hazard: (short) signal pulse does not conform to the pure logic function of a circuit; caused by internal delays

Example with signal traces:

<table>
<thead>
<tr>
<th>x</th>
<th>0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>z1</td>
<td>1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>z2</td>
<td>0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>z3, z4</td>
<td>1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>z5</td>
<td>0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>y</td>
<td>0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Hazard
Simulation of functional and timing behavior

A) Logic evaluation (purely functional simulation)

Example:

Timing diagram:

\[
\begin{array}{cccccccc}
  x, z_5 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
  z_4 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
  y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

Boolean function:

\[ y = \overline{x} \cdot \overline{x} = x \cdot x = 0 \]

Typical signal modeling:

- Boolean logic \{ '0', '1' \}
- Three-valued logic \{ '0', '1', 'X' \}
- IEEE Standard logic \{ 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' \}

B) Timing evaluation (timing verification, race analysis)

"Logic" is not considered

Fan-in without specification of gate function

Assuming a signal change on \( x \) at \( t = 0 \), the following "time table" results for the propagation of this signal change to the circuit output.

At the output, the maximal transient uncertainty is now known (important to determine the clock frequency of synchronous circuits).
Simulation with slew rate (slope) uncertainty

A signal change does not have an ideal edge (infinite slope). Therefore, a transition period results, during which the signal value is undefined. In simulation this is represented by a new signal "X".

Example:

\[
\text{AND} \quad 0 \quad 1 \quad X \\
\begin{array}{ccc}
0 & 0 & 0 \\
1 & 0 & 1 \\
X & 0 & X \\
\end{array}
\]

\[
\text{OR} \quad 0 \quad 1 \quad X \\
\begin{array}{ccc}
0 & 0 & 1 \\
1 & 1 & 1 \\
X & X & 1 \\
\end{array}
\]

\[
\text{NOT} \quad 0 \quad 1 \\
\begin{array}{cc}
0 & 1 \\
1 & 0 \\
X & X \\
\end{array}
\]

Let the signal uncertainty be one basic time unit:

\[
\begin{array}{cccccccccccccccc}
x & 0 & 0 & 0 & X & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
z_1 & 1 & 1 & 1 & X & 0 & 0 & 0 & 0 & 0 & X & 1 & 1 & 1 & 1 & 1 \\
z_2 & 0 & 0 & 0 & X & 1 & 1 & 1 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
z_3, z_4 & 1 & 1 & 1 & X & 0 & 0 & 0 & 0 & X & 1 & 1 & 1 & 1 & 1 \\
z_5 & 0 & 0 & 0 & X & 1 & 1 & 1 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & X & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Hazard

Truth tables with the new signal value "X":

\[
\begin{array}{cccccccccccccccc}
\text{AND} & 0 & 1 & X \\
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 \\
1 & 0 & X & 0 \\
X & 0 & X & X \\
\end{array}
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
\text{OR} & 0 & 1 & X \\
\begin{array}{cccccccccccccccc}
0 & 0 & 1 & X \\
1 & 1 & X & 1 \\
X & X & 1 & X \\
\end{array}
\end{array}
\]

\[
\begin{array}{cccccccccccccccc}
\text{NOT} & \quad 0 & 1 \\
\begin{array}{cccccccccccccccc}
0 & 1 \\
1 & 0 \\
X & X \\
\end{array}
\end{array}
\]
Simulation with delay uncertainty

Let the minimum and maximum delay of a gate be known. In between the output value is undefined.

\[ \tau_{d_{\text{min}}} = \Delta \]
\[ \tau_{d_{\text{max}}} = 3\Delta \]

Example:

\[
\begin{array}{cccccccc}
  & x & \downarrow \tau_d & z_1 & \downarrow \tau_d & z_2 & \downarrow \tau_d & z_3 & \downarrow \tau_d & z_4 & \downarrow \tau_d & z_5 & \downarrow \tau_d & z_6 & \downarrow \tau_d & y \\
  \text{普通 modelling of timing behavior:} & \\
  \text{Gate and wire delays} & \\
  \text{Pin-to-pin delays} & \\
  & \tau_d & \tau_{w_1} & \tau_{w_2} & \\
\end{array}
\]

This method results in a worst-case analysis, due to the increasing uncertainty range.
3 Simulation methods

Simulation program

a) Modeling circuit components using basic elements (primitives) of the simulation system
b) Circuit replication (compilation or list creation)
c) Simulation execution

Compiler driven simulation
The circuit under test is compiled into an executable program. In each simulation cycle, the logic function of the complete circuit is evaluated, based on the input signals and the internal circuit states. (registers, feedback signals).
Very fast, usually no timing analysis (zero delay simulation).

Event driven simulation
Event driven simulation is based on signal changes within the circuit. In each simulation step, components are evaluated only if a signal change occurs on their corresponding inputs. Signal changes are coded as events.

- Signal change: \( \text{val}(z, t_N) \neq \text{val}(z, t_{N-1}) \)

- Event: A component evaluation at simulation time \( t_{\text{gen}} \) causes that signal \( z \) is set to a new value \( \text{val}(z, t_{\text{exe}}) \) at simulation time \( t_{\text{exe}} \).

\[
E = (z, \text{val}(z, t_{\text{exe}}), t_{\text{gen}}, t_{\text{exe}})
\]

- \( z \): signal name
- \( \text{val}(z, t_{\text{exe}}) \): value of signal \( z \) at time \( t_{\text{exe}} \)
- \( t_{\text{gen}} \): generation time of event
- \( t_{\text{exe}} \): execution time of event
- \( t_{\text{exe}} - t_{\text{gen}} = \tau \): imposed delay

- Simulation cycle:

(italic font indicates VHDL simulation terminology)
Example simulation run: Multiplexer

- Circuit:

```
  \( A \leftrightsquigarrow \text{Inv} \rightarrow N_{\text{and}}_c \)
  \( \tau \)

  \( \text{Sel} \rightarrow N_{\text{and}}_a \rightarrow \text{Sel}_n \rightarrow S_1 \rightarrow N_{\text{and}}_b \rightarrow S_2 \rightarrow Q \)

  \( \tau \)
```

- Simulation run:

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>B</th>
<th>Sel</th>
<th>Sel_n</th>
<th>S_1</th>
<th>S_2</th>
<th>Q</th>
<th>evaluated components</th>
<th>new events</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'0'</td>
<td>'0'</td>
<td>'1'</td>
<td>'0'</td>
<td>'1'</td>
<td>'1'</td>
<td>'0'</td>
<td>initial state (stimuli)</td>
<td>(A, '1', 0, 20); (B, '1', 0, 10); (Sel, '0', 0, 30)</td>
</tr>
<tr>
<td>10</td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_b</td>
<td>(S_2, '0', 10, 12)</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>'0'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_c</td>
<td>(Q, '1', 12, 14)</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>20</td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_a</td>
<td>-</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Inv, Nand_b</td>
<td>(Sel_n, '1', 30, 32); (S_2, '1', 30, 32)</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>'1'</td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_a, Nand_c</td>
<td>(S_1, '0', 32, 34); (Q, '0', 32, 34)</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>'0'</td>
<td>'0'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Nand_c</td>
<td>(Q, '1', 34, 36)</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td>'1'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

- Signal waveforms:

![Signal waveforms diagram]


4 VHDL

VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit)

- widely used hardware description language for simulation, documentation, and synthesis
- standardized format for libraries and interoperability of design tools
- similar to programming language; special extensions for modeling hardware:
  - Concurrency (description of concurrent processes)
  - Timing model (delays; event driven simulation)
- supports modeling of hardware at different levels of abstraction
- supports behavioral and structural description styles

VHDL model of a Multiplexer
1. Specification:

   Sel   A   B   Inputs: A, B, Sel
   |      |   |   Output: Q
   MUX   |   |   Sel = '0' ⇒ Q = A
      |   |   Sel = '1' ⇒ Q = B
   Q

2. Behavioral description (dataflow, zero delay):
   (A) Circuit interface:

   ENTITY Multiplexer IS
   PORT(A, B, Sel: in bit;
   Q: out bit);
   END Multiplexer;

   (B) Circuit implementation:

   ARCHITECTURE dataflow OF Multiplexer IS
   BEGIN
   Q <= A WHEN Sel = '0' ELSE B;
   END dataflow;

   VHDL (1)
3. Structural description:

(A) Circuit interface:

ENTITY Multiplexer IS
   PORT(A, B, Sel: in bit;
       Q: out bit);
END Multiplexer;

(B) Circuit implementation:

LIBRARY BasicGates;
USE BasicGates.ALL;

ARCHITECTURE structure OF Multiplexer IS
   COMPONENT Inverter
      PORT(i: in bit; o: out bit);
   END COMPONENT;
   COMPONENT Nand2
      PORT(i1,i2: in bit; o: out bit);
   END COMPONENT;
   SIGNAL Sel_n, S1, S2 : bit;
   BEGIN
      Inv: Inverter PORT MAP (Sel, Sel_n);
      Nand_a: Nand2 PORT MAP (A, Sel_n, S1);
      Nand_b: Nand2 PORT MAP (B, Sel, S2);
      Nand_c: Nand2 PORT MAP (S1, S2, Q);
   END structure;

4. Validation by simulation:
Testbench:

- applies stimuli to circuit inputs
- monitors circuit outputs
- checks correctness via assert statements (optional)

Configuration:

- binding of architectures to entities
- flexible mechanism for using alternative implementations
Testbench for Multiplexer

**Interface description:**

ENTITY Testbench IS
---
--- no inputs, no outputs.
---
END Testbench;

**Implementation (instantiation of multiplexer as COMPONENT):**

ARCHITECTURE functiontest OF Testbench IS
COMPONENT Multiplexer
  PORT(A, B, Sel: in bit;
    Q: out bit);
END COMPONENT;
SIGNAL A, B, Sel, Q: bit;
BEGIN
  MUX_1: Multiplexer PORT MAP(A, B, Sel, Q);
  A <= '0', '1' after 20 ns, '0' after 60 ns, '1' after 100 ns;
  B <= '1', '0' after 30 ns, '1' after 70 ns;
  Sel <= '0', '1' after 40 ns, '0' after 80 ns;
END functionstest;

**Configuration (Entity / Architecture Binding):**

CONFIGURATION Test_Setup OF Testbench IS
FOR functiontest
  FOR MUX_1: Multiplexer USE ENTITY WORK.Multiplexer(dataflow);
END FOR;
END FOR;
END Test_Setup;
Processes:

- Statements with a `PROCESS` are executed sequentially within an infinite loop.
- All Processes are concurrent.
- A `PROCESS` must have either a `WAIT`-statement or a `Sensitivity-List`.
- Each signal assignment outside of a `PROCESS` will be modified into a new `PROCESS` by the simulator.
- In a `PROCESS` variables can be used in addition to signals.

Example: Clock Generator

**Example for signal assignment**

```vhdl
ARCHITECTURE Simple OF CLKgen IS
BEGIN
  CLK <= NOT CLK AFTER 10 ns;
END Simple;
```

**Example for SENSITIVITY LIST**

```vhdl
ARCHITECTURE SList OF CLKgen IS
BEGIN
  PROCESS(CLK)
  BEGIN
    CLK <= NOT CLK AFTER 10 ns;
  END PROCESS;
END SList;
```

**Example for WAIT ON-statement**

```vhdl
ARCHITECTURE Wait1 OF CLKgen IS
BEGIN
  PROCESS
  BEGIN
    CLK <= NOT CLK AFTER 10 ns;
    WAIT ON CLK;
  END PROCESS;
END Wait1;
```

**Example for WAIT FOR-statement**

```vhdl
ARCHITECTURE Wait2 OF CLKgen IS
BEGIN
  PROCESS
  BEGIN
    CLK <= NOT CLK;
    WAIT FOR 10 ns;
  END PROCESS;
END Wait2;
```

Signals and Variables:

Signals:

- Have temporal “memory”. Future signal values are planner by entry of a transaction $(w_n, t_n)$ in the signal driver.
  $(w_n$: planned signal value, $t_n$: timestamp) $(\text{transaction} \neq \text{event})$
- In value assignments “$<=$ waveform” the new value will be planned. It will be incurred only when the `PROCESS` will be suspended (during the signal update phase).

Variables:

- Do not have temporal memory.
- In value assignments “$<=$expression” the new value will be incurred immediately.
Delay Mechanisms:

**Transport-Delay:** \( (\text{signal} \leq \text{transport waveform};) \)
- Modelling of an ideal component with unlimited bandwidth.
- Rule for insertion of new transaction \((w_n, t_n)\) into the signal driver:
  - All (later) transactions \((w_i, t_i)\) with \(t_i \geq t_n\) will be deleted.
  - The new transaction \((w_n, t_n)\) will be appended at the end of the list.

**Inertial-Delay:** \( (\text{signal} \leq \text{waveform};) \)
- Modelling of real components with finite bandwidth.
  (impulses which are shorter than the component’s delay, will be suppressed.)
- Rule for insertion of a new transaction \((w_n, t_n)\) into the signal driver:
  - All (later) transactions \((w_i, t_i)\) with \(t_i \geq t_n\) will be deleted.
  - The new transaction \((w_n, t_n)\) will be appended at the end of the list.
  - Transactions \((w_i, t_i)\) within the interval \(t_{\text{now}} \leq t_i \leq t_n\) will be deleted if \(w_i \neq w_n\).

**Event-driven simulation:**

Execution of all transactions at time step \(t_n\) in \(\Delta\)-cycles. Advancing the simulation time \(t_n \rightarrow t_{n+1}\) when all transactions at time \(t_n\) have been executed.

**\(\Delta\)-cycle:**
- **Signal-Update-Phase:**
  - Execution of all transactions for the current time step.
  - If a transaction changes a signal value, this is an event.
- **Process-Evaluation-Phase:**
  - Processes, for which an event is present, will be executed.
  - Value assignments to signals will be scheduled in the signal driver.
III. Testing of Digital Circuits

1 Introduction

Testing and Diagnosis

![Diagram of unit under test (UUT)]

Basics

fault: circuit performance differs from its specification

testing: few excitations should lead to responses, which yield much information on the correctness of a circuit

Testing Methods

static (DC) testing: testing digital circuit performance; cycle time ≫ circuit delay

dynamic (AC) testing,
at-speed testing: testing digital performance at operating speed (edges, races, hazards)
testing analog circuit performance: validate voltage and current values within the circuit

Fault Diagnosis During

design: design verification (logic simulation, formal verification), design for testability (DFT), test pattern generation, fault simulation

production: test automation

maintenance: fault search, fault detection, fault localizing

Introduction (1)
Fault Models

- represent effects of physical defects on the operation of a circuit
- allow to evaluate the quality of a test set, e.g. by computing the percentage of detected faults (fault coverage)
- allow algorithmic test pattern generation instead of heuristically determined functional test patterns or random test patterns

Levels of Abstraction – Fault Models
A fault model represents certain physical defects on a particular level of abstraction.

<table>
<thead>
<tr>
<th>Circuit Model</th>
<th>Fault Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Behavioral Circuit" /></td>
<td>faulty transition- and output-functions</td>
</tr>
<tr>
<td><img src="image2" alt="Register Transfer Circuit" /></td>
<td>faulty transfer and storage</td>
</tr>
<tr>
<td><img src="image3" alt="Functional Blocks" /></td>
<td>faults in truth table and faulty sequencing, resp.</td>
</tr>
<tr>
<td><img src="image4" alt="Logic Gates" /></td>
<td>stuck-at faults (0 or 1), delay faults, bridging- and crosstalk-faults between wires</td>
</tr>
<tr>
<td><img src="image5" alt="Transistors" /></td>
<td>faulty performance, e.g. amplification, leakage current, and voltage level</td>
</tr>
<tr>
<td><img src="image6" alt="Physical Level" /></td>
<td>static faults: process deviations dynamic faults: radiation, electromagnetic fields</td>
</tr>
</tbody>
</table>
Test Pattern Generation

- Algorithmic:
  - based on a defined fault model
  - quality with respect to the fault model can be determined

- Heuristic:
  - using either functional test or (pseudo) random test
  - quality has to be checked e.g. by fault simulation

Problems

- limited access to logic variables (signals)
- integrated circuits have only a limited number of pins
- time for test generation
- test time
- suitable fault model
- evaluation of test results
- test costs

Assumptions Made for This Lecture

fault model: logical stuck-at fault model with single fault assumption (signal x: x/0, x/1).

level of abstraction: gates having logical states 0 and 1.

methods: fault simulation

algorithmic test generation in combinational and sequential circuits: D-Algorithm, Boolean Difference
Definitions and Terminology

Fault Model – Terminology

example:

\[
\begin{align*}
x_1 & \quad \rightarrow \quad z_1 \\
x_2 & \quad \rightarrow \quad z_2 \\
x_3 & \quad \rightarrow \quad y \\
x_4 & \quad \rightarrow \quad z_2 \\
\end{align*}
\]

\[
y = z_1 + z_2 = x_1 x_2 + x_3 x_4 \\
z_1 = x_1 x_2 = x_1 + x_2 \\
z_2 = x_3 x_4 = x_3 + x_4
\]

input variable: \( x_1, x_2, x_3, x_4 \) \hspace{1cm} internal variable: \( z_1, z_2 \)

output variable: \( y \) \hspace{1cm} test point: \( y \)

test variable: \( x_1, x_2, x_3, x_4, z_1, z_2, y \)

For each test variable, two fault assumptions can be made (x/0 and x/1).

input value assignment: \( x_\nu \) (test \( t_\nu \))

convention: \( \nu \) is the digital value corresponding to the input value assignment if it is interpreted as a binary number.

set of all input value assignments (tests): \( T = \{ t_\nu | \nu \in \mathbb{N}\} \)

\( \mathbb{N} : \) set of test numbers, \( t_\nu \in T \iff \nu \in \mathbb{N} \)

\( n : \) number of input variables, \( |T| = |\mathbb{N}| = 2^n \)

set of all assumed faults: \( F = \{ f_\mu | \mu \in \mathbb{M}\} \)

\( \mathbb{M} : \) set of fault numbers, \( f_\mu \in F \iff \mu \in \mathbb{M} \)

\( m = |F| = |\mathbb{M}| : \) number of faults to be tested

Test: Input value assignment which detects one or more (single) faults.
Single Gates – Single Faults

\[ y(x_1, x_2) \]

\[ y(x=0/1) = y_\mu \]

fault-free circuit

fault \( f_\mu \)

conditions for input value assignments

\[ y \oplus y_\mu = 1 \]

\[ y_\mu \oplus y_\kappa = 1 \]

Fault Dictionaries and Fault Covering Tables for Single Gates

fault dictionary:

<table>
<thead>
<tr>
<th>AND</th>
<th>input value assignment</th>
<th>test point</th>
<th>( y )</th>
<th>( y(x_2=0) )</th>
<th>( y(y=0) )</th>
<th>( y(x_1=0) )</th>
<th>( y(x_2=1) )</th>
<th>( y(y=1) )</th>
<th>( y(x_1=1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 \cdot x_2 )</td>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_0 )</td>
<td>0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_1 )</td>
<td>0 1 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_2 )</td>
<td>1 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_3 )</td>
<td>1 1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

fault covering table:

<table>
<thead>
<tr>
<th>AND</th>
<th>test</th>
<th>( f_1 )</th>
<th>( f_2 )</th>
<th>( f_3 )</th>
<th>( f_4 )</th>
<th>( f_5 )</th>
<th>( f_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 \cdot x_2 )</td>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( y = x_1 \cdot x_2 )</td>
<td>( t_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Fault Dictionary:

<table>
<thead>
<tr>
<th>OR</th>
<th>Input Value Assignment</th>
<th>Test Point</th>
<th>y&lt;br&gt;(x_2=0)</th>
<th>y&lt;br&gt;(y=0)</th>
<th>y&lt;br&gt;(x_1=0)</th>
<th>y&lt;br&gt;(x_2=1)</th>
<th>y&lt;br&gt;(y=1)</th>
<th>y&lt;br&gt;(x_1=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1 \lor x_2</td>
<td>y</td>
<td>t_0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>y = x_1 + x_2</td>
<td></td>
<td>t_3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Fault Covering Table:

<table>
<thead>
<tr>
<th>OR</th>
<th>Test</th>
<th>f_1&lt;br&gt;x_2/0</th>
<th>f_2&lt;br&gt;y/0</th>
<th>f_3&lt;br&gt;x_1/0</th>
<th>f_4&lt;br&gt;x_2/1</th>
<th>f_5&lt;br&gt;y/1</th>
<th>f_6&lt;br&gt;x_1/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1 \lor x_2</td>
<td>y</td>
<td>t_0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>y = x_1 + x_2</td>
<td></td>
<td>t_3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Fault Dictionary:

<table>
<thead>
<tr>
<th>NOT</th>
<th>Input Value Assignment</th>
<th>Test Point</th>
<th>y&lt;br&gt;(x=0)</th>
<th>y&lt;br&gt;(y=1)</th>
<th>y&lt;br&gt;(x=1)</th>
<th>y&lt;br&gt;(y=0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>t_0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Fault Covering Table:

<table>
<thead>
<tr>
<th>NOT</th>
<th>Test</th>
<th>f_1&lt;br&gt;x/0</th>
<th>f_4&lt;br&gt;y/1</th>
<th>f_3&lt;br&gt;x/1</th>
<th>f_2&lt;br&gt;y/0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>t_0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t_1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Single Fault – Circuit

**Example:**

\[ y = z_1 \cdot z_2 = x_1 \cdot x_2 + x_3 \cdot x_4 \]

**Fault simulation:**

\[ y = z_1 = x_1 \cdot x_2 \]

\[ y = z_2 = x_3 \cdot x_4 \]

\[ y = 0 \]

\[ z_1 = 0 \]

\[ z_2 = 1 \]

\[ z_1 \cdot z_2 = 0 \]

<table>
<thead>
<tr>
<th>( y )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_5 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_6 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_7 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_8 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_9 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_{10} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_{11} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_{12} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_{13} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( y_{14} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Truth Table:**

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( t_8 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_9 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( t_{10} )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_{11} )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( t_{12} )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_{13} )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( t_{14} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_{15} )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Introduction (7)**
Fault covering table

<table>
<thead>
<tr>
<th></th>
<th>f_1</th>
<th>f_2</th>
<th>f_3</th>
<th>f_4</th>
<th>f_5</th>
<th>f_6</th>
<th>f_7</th>
<th>f_8</th>
<th>f_9</th>
<th>f_10</th>
<th>f_11</th>
<th>f_12</th>
<th>f_13</th>
<th>f_14</th>
<th>x_1/1</th>
<th>x_2/1</th>
<th>x_3/1</th>
<th>x_4/1</th>
<th>z_1/0</th>
<th>z_2/0</th>
<th>y_1/0</th>
<th>x_1/0</th>
<th>x_2/0</th>
<th>z_1/1</th>
<th>y/0</th>
<th>x_3/0</th>
<th>x_4/0</th>
<th>z_2/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>t_15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>t_15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Advantageously sorted fault covering table

<table>
<thead>
<tr>
<th></th>
<th>f_1</th>
<th>f_2</th>
<th>f_3</th>
<th>f_4</th>
<th>f_5</th>
<th>f_6</th>
<th>f_7</th>
<th>f_8</th>
<th>f_9</th>
<th>f_10</th>
<th>f_11</th>
<th>f_12</th>
<th>f_13</th>
<th>f_14</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_0</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>t_1</td>
<td>1</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>t_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>
</tr>
<tr>
<td>t_3</td>
<td>1</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>t_4</td>
<td></td>
<td>1</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>t_5</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>t_6</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>t_7</td>
<td></td>
<td>1</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>t_8</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>t_9</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>t_10</td>
<td>1</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>t_11</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>t_12</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>t_13</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>t_14</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>t_15</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>

Introduction (8)
Definitions for Test- and Fault-Sets

A) Test-Fault-Relation $R$:

ordered pair: $(t_\nu, f_\mu)$; first and second component

cartesian product: $T \times F = \{(t_\nu, f_\mu) | t_\nu \in T \land f_\mu \in F\}$

$R$ is a binary relation from set $T$ to set $F$ iff $R \subseteq T \times F$.

$R = \{(t_\nu, f_\mu) \in T \times F | t_\nu R f_\mu\}$;
$|R| \leq |T \times F| = |T| \cdot |F|$

$t_\nu R f_\mu \in \{1, 0\}$: Test $t_\nu$ is related to fault $f_\mu$.

B) Fault Detection: $t_\nu R f_\mu = y(x_\nu) \oplus y_\mu(x_\nu) = 1$

Fault $f_\mu$ is detected by the input value assignment $x_\nu$.

C) Fault Distinction: $y_\mu(x_\nu) \oplus y_\kappa(x_\nu) = 1$

Faults $f_\mu$ and $f_\kappa$ are distinguished by the input value assignment $x_\nu$.

$y_\mu \oplus y_\kappa = y \oplus y_\mu \oplus y \oplus y_\kappa = t_\nu R f_\mu \oplus t_\nu R f_\kappa$

D) Fault Group: $F_\nu = \{f_\mu \in F | t_\nu R f_\mu\}$

$F_\nu$ is the set of all faults detected by test $t_\nu$.

$F_\nu \subseteq F$

$t_\nu R f_\mu = y(x_\nu) \oplus y_\mu(x_\nu) = f_\mu \in F_\nu$

E) Fault Covering: $C_{M,N} = \bigwedge_{\mu \in M} \bigvee_{\nu \in N} t_\nu R f_\mu$

$C_{M,N} = \bigwedge_{\mu \in M} \bigvee_{\nu \in N} (f_\mu \in F_\nu) = \bigwedge_{\mu \in M} (f_\mu \in \bigcup_{\nu \in N} F_\nu) = [\bigcup_{\nu \in N} F_\nu = F]$

Fault covering means that the set union of fault groups contains all assumed faults.

complete test: $C_{M,N} = 1$

F) (Fault-Covering) Test Set: $T_C = \{t_\nu \in T | \nu \in I\}$ is true iff $(I \subseteq N) \land (\bigcup_{\nu \in I} F_\nu = F)$

$I$ : subset of test numbers of a complete test

The tests contained in the test set $T_C$ are sufficient for detection of all assumed faults.
G) Test Group: \[ T_\mu = \{ t_\nu \in T \mid t_\nu Rf_\mu \} \]

\( t_\nu Rf_\mu = t_\nu \in T_\mu \)

H) Set of Non-Distinguishable Faults:

\[ F_U = \{ f_\mu \in F \mid \mu \in U \} \]

is a set of non-distinguishable faults iff

\[ (U \subseteq M) \wedge \bigwedge_{\delta, \kappa \in U} (T_\delta = T_\kappa); \quad \delta \neq \kappa \]

\( U \) : subset of fault numbers of non-distinguishable faults

The corresponding test groups of non-distinguishable faults are identical.

I) Minimum fault covering test set:

\[ C_{M,N}(\ldots \tau_\nu \ldots) = \bigwedge_{\mu \in M} \bigvee_{\nu \in N} (t_\nu \in T_\mu) \cdot \tau_\nu \]

\( \tau_\nu \in \{1, 0\} : \) test \( t_\nu \) is member of the test set.

after transformation:

\[ C_{M,N}(\ldots \tau_\nu \ldots) = \bigvee_{\nu \in I_\kappa} \tau_\nu \]

\( I_{k_{\text{min}}} : |I_\kappa| = \text{Min} \)

the shortest terms describe the minimum test sets.

minimum test set: \[ T_{c_{\text{min}}} = \{ t_\nu \in T \mid \nu \in I_{k_{\text{min}}} \} \]

J) Algorithm for suboptimal computation of a test set (heuristic):

Selection of tests based on cardinality of fault groups

1. Determine cardinality of all fault groups
2. Select fault group \( F_\nu \) with largest cardinality
3. Add the corresponding test \( t_\nu \) to the test set
4. Eliminate all faults which are recognized by \( t_\nu \) and eliminate \( F_\nu \)
5. IF all faults have been recognized,
   THEN a test set has been found
   ELSE GOTO 1

K) Fault localization:

\[ L_{M,N} = C_{M,N} \wedge \left[ \bigwedge_{\mu, \kappa \in M} \bigvee_{\nu \in N} [(t_\nu \in T_\mu) \oplus (t_\nu \in T_\kappa)] \right]; \quad \mu \neq \kappa \]

\[ L_{M,N} = \bigwedge_{\mu \in M} (T_\mu \neq \emptyset) \wedge \bigwedge_{\mu, \kappa \in M} (T_\mu \triangle T_\kappa \neq \emptyset); \quad \mu \neq \kappa \]
2 Boolean Difference

Input value assignment: \( x \)
Test variable: \( z, z(x) \)
Test point: \( y(z(x), \bar{x}) \)

Boole’s Expansion Theorem (Shannon’s Expansion)

\[
y = z \cdot y(z = 1) + \overline{z} \cdot y(z = 0)
\]
\[
= [\overline{z} + y(z = 1)] \cdot [z + y(z = 0)]
\]
\[
= z \cdot y(z = 1) \oplus z \cdot y(z = 0)
\]
\[
= y_z \cdot z \oplus y(z = 0)
\]

Definition of the Boolean Difference

\[
y_z = y(z, \bar{x}) \oplus y(z, x)
\]

Test Condition

1) \( \bar{x} \cdot R \cdot z/0 = z \cdot y_z = 1 \)
2) \( \bar{x} \cdot R \cdot z/1 = z \cdot y_z = 1 \)

fault excitation for \( z/0 : z(x) = 1 \) (controllability!)
for \( z/1 : z(x) = 0 \)
sensitizing assignment : \( y_z(x) = 1 \) (observability!)
Theorems and Arithmetic Rules

Boolean Algebra

principle of duality: \( \{ 0, 1 \} ; \cdot, +, \overline{\cdot} \)

1. \( x \cdot y = y \cdot x \); \( x + y = y + x \) commutativity

2. \((x \cdot y) \cdot z = x \cdot (y \cdot z)\); \((x + y) + z = x + (y + z)\) associativity

3. \(x \cdot (y + z) = x \cdot y + x \cdot z\); \(x + y \cdot z = (x + y) \cdot (x + z)\) distributivity

4. \(x \cdot x = x\); \(x + x = x\) idempotence

5. \(x \cdot (x + y) = x\); \(x + x \cdot y = x\) absorption

6. \(x \cdot 1 = x\); \(x + 0 = x\) neutral element

7. \(x \cdot 0 = 0\); \(x + 1 = 1\)

8. \(x \cdot \overline{x} = 0\); \(x + \overline{x} = 1\) complement

9. \(\overline{\overline{x}} = x\) involution

10. \(\overline{x \cdot y} = \overline{x} + \overline{y}\); \(\overline{x + y} = \overline{x} \cdot \overline{y}\) De Morgan’s law
XOR, Alternative, Antivalence

\[
1 \oplus 0 = 1 ; \quad 0 \oplus 1 = 1 ; \quad 1 \oplus 1 = 0 ; \quad 0 \oplus 0 = 0
\]

1. \( x \oplus y = y \oplus x \) *commutativity*

2. \((x \oplus y) \oplus z = x \oplus (y \oplus z)\) *associativity*

3. \( x \cdot (y \oplus z) = x \cdot y \oplus x \cdot z \) *"." is distributive with respect to "\(\oplus\)"

4. \( x \oplus 0 = x \) *neutral element*

5. \( \overline{x} = x \oplus 1 \) *negation*
   \[
   \overline{x \oplus y} = x \oplus y \oplus 1
   \]

6. \( x \oplus \overline{x} = 1 \) *complement*
   \( x \oplus x = 0 \)

7. \( x \oplus x \oplus x = x \) *idempotence*

8. \( x \oplus y = \overline{x} \oplus \overline{y} \) *law of contraposition*

9. \( x \oplus y = x \cdot \overline{y} + x \cdot y \)

10. \( x \oplus y = (x + y) \cdot (\overline{x} + \overline{y}) \)

11. \( x + y = x \oplus y \oplus x \cdot y \)

12. \( x \cdot y = x \oplus y \oplus (x + y) \)
Arithmetic Rules for the Boolean Difference

1) \( y_x = 0 \) if \( y \neq f(x) \)
2) \( y_y = 1 \)
3) \( (\overline{y})_x = y_x \)
4) \( (z \oplus w)_x = z_x \oplus w_x \)
5) \( (z \cdot w)_x = z_x \cdot w_x \oplus z_x \cdot w_x \)
6) \( (z + w)_x = z_x \cdot w_x \oplus z_x \cdot w_x \)
7) \( y_x = y_z \cdot z_x \) if \( y = y(z(x)) \)
8) \( (yz)_w = (yw)_z \)

examples:

\[
\begin{align*}
  &z(x) \quad \text{w(x)} \quad \text{y(x)} \\
  z(x) \cdot \text{w(x)} \quad \text{y(x)} \\
  y = z \cdot w \\
  y_x = z \cdot w_x \oplus w \cdot z_x \oplus z_x \cdot w_x
\end{align*}
\]

\[
\begin{align*}
  &z(x) \quad \text{w(x)} \quad \text{y(x)} \\
  z(x) \oplus \text{w(x)} \quad \text{y(x)} \\
  y = z \oplus w \\
  y_x = z \oplus w_x \oplus w \cdot z_x \oplus z_x \cdot w_x
\end{align*}
\]

\[
\begin{align*}
  &z(x) \quad \text{w(x)} \quad \text{y(x)} \\
  z(x) \oplus \text{w(x)} \quad \text{y(x)} \\
  y = z \oplus w \\
  y_x = z \oplus w_x
\end{align*}
\]

case 1) \( z = z(x) \), i.e. \( z_x = 1 \) and \( w \neq w(x) \), i.e. \( w_x = 0 \)

\[
\begin{align*}
  &y_x = z \quad \text{w} \\
  w = 1 \rightarrow y_x = 1
\end{align*}
\]

\[
\begin{align*}
  &y_x = z \quad \text{w} \\
  w = 0 \rightarrow y_x = 1
\end{align*}
\]

\[
\begin{align*}
  &y_x = z \quad \text{w} \\
  y_x = 1
\end{align*}
\]

case 2) \( z = z(x) \), i.e. \( z_x = 1 \) and \( w = w(x) \), i.e. \( w_x = 1 \)

\[
\begin{align*}
  &y_x = z \quad \text{w} \\
  y_x = z \quad \text{w}
\end{align*}
\]

\[
\begin{align*}
  &y_x = z \quad \text{w} \\
  y_x = 0
\end{align*}
\]

If \( z \) and \( w \) are equally assigned, then \( y_x = 1 \).
Algebraic Test Generation Applying the Boolean Difference

Example:

\[
y = x_1 x_2 + x_3 x_4 = z_1 z_2 = z_1 + z_2 + 1
\]

\[
z_1 = x_1 x_2 = x_1 x_2 + 1
\]

\[
z_2 = x_3 x_4 = x_3 x_4 + 1
\]

a) test point \( y \); test variable \( y \)

\[
y = 1
\]

\[
y = 1
\]

\[
\rightarrow t_3, t_7, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}
\]

\[
\rightarrow t_0, t_1, t_2, t_4, t_5, t_6, t_8, t_9, t_{10}
\]

b) test point \( y \); test variables \( z_1, z_2 \)

\[
y = z_2 = x_3 + x_4
\]

\[
z_1 = x_1 + x_2
\]

\[
z_1 = x_1 + x_2 + 1
\]

\[
z_2 = x_1 + x_2
\]

\[
z_2 = x_1 + x_2 + 1
\]

\[
\rightarrow t_0, t_1, t_2, t_4, t_5, t_6, t_8, t_9, t_{10}
\]

\[
\rightarrow t_12, t_{13}, t_{14}
\]

c) test points \( z_1 \) resp. \( z_2 \); test variables \( x_1, x_2 \) resp. \( x_3, x_4 \)

\[
z_1 = x_2
\]

\[
x_1 = x_2 = 1
\]

\[
\rightarrow t_3, t_7, t_{11}, t_{15}
\]

\[
x_1 = x_2 = 1
\]

\[
\rightarrow t_2, t_6, t_{10}, t_{14}
\]

\[
z_1 = x_1
\]

\[
x_2 = x_1 = 1
\]

\[
\rightarrow t_3, t_7, t_{11}, t_{15}
\]

\[
x_2 = x_1 = 1
\]

\[
\rightarrow t_1, t_5, t_9, t_{13}
\]
\[ z_{23} = x_4 \]

\[ x_{3/0}: \quad x_3 \cdot x_4 = 1 \quad \rightarrow \quad t_{12}, t_{13}, t_{14}, t_{15} \]

\[ x_{3/1}: \quad \overline{x}_3 \cdot x_4 = 1 \quad \rightarrow \quad t_8, t_9, t_{10}, t_{11} \]

\[ z_{24} = x_3 \]

\[ x_{4/0}: \quad x_4 \cdot x_3 = 1 \quad \rightarrow \quad t_{12}, t_{13}, t_{14}, t_{15} \]

\[ x_{4/1}: \quad \overline{x}_4 \cdot x_3 = 1 \quad \rightarrow \quad t_4, t_5, t_6, t_7 \]

d) test point \( y \); test variables \( x_1, x_2, x_3, x_4 \)

\[ y_x_1 = y_{x_1} \cdot z_{1x_1} = z_{2x_2} \]

\[ x_{1/0}: \quad x_1 x_2 \overline{z}_2 = x_1 x_2 \overline{x}_3 + x_1 x_2 \overline{x}_4 = 1 \quad \rightarrow \quad t_3, t_7, t_{11} \]

\[ x_{1/1}: \quad \overline{x}_1 x_2 \overline{z}_2 = \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 x_2 \overline{x}_4 = 1 \quad \rightarrow \quad t_2, t_6, t_{10} \]

\[ y_{x_2} = y_{x_1} \cdot z_{2x_2} = z_{2x_1} \]

\[ x_{2/0}: \quad x_2 x_1 \overline{z}_2 = x_2 x_1 \overline{x}_3 + x_2 x_1 \overline{x}_4 = 1 \quad \rightarrow \quad t_3, t_7, t_{11} \]

\[ x_{2/1}: \quad x_2 x_1 \overline{z}_2 = x_2 x_1 \overline{x}_3 + x_2 x_1 \overline{x}_4 = 1 \quad \rightarrow \quad t_1, t_5, t_9 \]

\[ y_{x_3} = y_{x_2} \cdot z_{2x_3} = z_{1x_4} \]

\[ x_{3/0}: \quad x_3 x_4 \overline{z}_1 = x_3 x_4 \overline{x}_1 + x_3 x_4 \overline{x}_2 = 1 \quad \rightarrow \quad t_{12}, t_{13}, t_{14} \]

\[ x_{3/1}: \quad \overline{x}_3 x_4 \overline{z}_1 = \overline{x}_3 x_4 \overline{x}_1 + \overline{x}_3 x_4 \overline{x}_2 = 1 \quad \rightarrow \quad t_8, t_9, t_{10} \]

\[ y_{x_4} = y_{x_2} \cdot z_{2x_4} = z_{1x_3} \]

\[ x_{4/0}: \quad x_4 x_3 \overline{z}_1 = x_4 x_3 \overline{x}_1 + x_4 x_3 \overline{x}_2 = 1 \quad \rightarrow \quad t_{12}, t_{13}, t_{14} \]

\[ x_{4/1}: \quad \overline{x}_4 x_3 \overline{z}_1 = \overline{x}_4 x_3 \overline{x}_1 + \overline{x}_4 x_3 \overline{x}_2 = 1 \quad \rightarrow \quad t_4, t_5, t_6 \]

tests, input value assignments:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t_0</td>
<td>0 0 0 0</td>
<td>t_4</td>
<td>0 1 0 0</td>
<td>t_8</td>
<td>1 0 0 0</td>
<td>t_{12}</td>
<td>1 1 0 0</td>
<td></td>
</tr>
<tr>
<td>t_1</td>
<td>0 0 0 1</td>
<td>t_5</td>
<td>0 1 0 1</td>
<td>t_9</td>
<td>1 0 0 1</td>
<td>t_{13}</td>
<td>1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>t_2</td>
<td>0 0 1 0</td>
<td>t_6</td>
<td>0 1 1 0</td>
<td>t_{10}</td>
<td>1 0 1 0</td>
<td>t_{14}</td>
<td>1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>t_3</td>
<td>0 0 1 1</td>
<td>t_7</td>
<td>0 1 1 1</td>
<td>t_{11}</td>
<td>1 0 1 1</td>
<td>t_{15}</td>
<td>1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Boolean Difference (6)
Structure Oriented Computation of the Boolean Difference

expressions for $y_x$:

- Boolean difference of $y$ with respect to $x$
- observability of $x$ at $y$
- sensitivity of $y$ with respect to $x$

expressions for $y_x = 1$:

- $y$ is functionally dependent on $x$
- a faulty assignment at $x$ (signal $x$) leads to a faulty assignment at $y$ (signal $y$)
- a single stuck-at fault at $x$ is observable at $y$

\[
y_x = y(x) \oplus y(\overline{x})
\]
\[
y \oplus y_x = y(\overline{x}) = z(\overline{x}) \circ w(\overline{x})
\]
\[
y \oplus y_x = (z \oplus z_x) \circ (w \oplus w_x)
\]

**global:**
\[
y_x = [(z \oplus z_x) \circ (w \oplus w_x)] \oplus [z \circ w]
\]

**local:**
\[
y_z = (\overline{z} \circ w) \oplus (z \circ w)
\]
\[
y_w = (z \circ \overline{w}) \oplus (z \circ w)
\]
applying the expansion theorem we obtain

\[ y_x = z_x \cdot \overline{w_x} \cdot y_x(z_x = 1, w_x = 0) + \overline{z_x} \cdot w_x \cdot y_x(z_x = 0, w_x = 1) \\
+ z_x \cdot w_x \cdot y_x(z_x = 1, w_x = 1) + \overline{z_x} \cdot \overline{w_x} \cdot y_x(z_x = 0, w_x = 0) \]

General Computation Rule:

\[ y_x = y_z \cdot z_x \cdot \overline{w_x} + y_w \cdot w_x \cdot \overline{z_x} + z_x \cdot w_x \cdot [\overline{z \oplus w} \oplus z \oplus w] \]

a) for XOR gates it holds: \[ \overline{z \oplus w} \oplus z \oplus w = 0 \]
b) for AND, NAND, OR, and NOR gates it holds: \[ \overline{z \oplus w} \oplus z \oplus w = z \oplus w \]
c) \[ z \oplus w = 1 : \text{equal assignment of } z \text{ and } w \]
\[ z \oplus w = 0 : \text{unequal assignment of } z \text{ and } w \]
Computation in Combinational Circuits with Reconvergent Regions

\[ y = y(z(x), w(x)) = z \circ w \]

\( x \): branching point (fanout-stem) \hspace{1cm} \( y \): reconvergence point

\[ y_x = y_z \cdot z_x \cdot \overline{w_x} + y_w \cdot w_x \cdot \overline{z_x} + z_x \cdot w_x \cdot [\overline{z \circ w} \oplus z \circ w] \]

a) \( y_z \cdot z_x \cdot \overline{w_x} = 1 \): single-path sensitization
   (single) fault-path: \( x \rightarrow z \rightarrow y \)
b) \( y_w \cdot w_x \cdot \overline{z_x} = 1 \): single-path sensitization
   (single) fault-path: \( x \rightarrow w \rightarrow y \)
c) \( z_x \cdot w_x \cdot [\overline{z \circ w} \oplus z \circ w] = 1 \): multiple-path sensitization
   (multiple) fault-path: \( x \leftarrow z_{w} \rightarrow y \)

\[ z_x \cdot w_x \cdot [\overline{z \circ w} \oplus z \circ w] = 1 \): self-masking

A test for a fault at signal \( x \) propagates the fault-effect either along one of both fault-paths (\((y_z \cdot z_x \cdot \overline{w_x})\) or \((y_w \cdot w_x \cdot \overline{z_x})\)) or along both fault-paths simultaneously (\((z_x \cdot w_x \cdot [\overline{z \circ w} \oplus z \circ w])\)).
Therefore, in case of circuits having reconvergencies, a test set for input variables does not guarantee that internal variables are tested, too.
Computation in Combinational Circuits with Tree Structure (Fanout-Free)

\[ z_x \cdot w_x = 0 : \text{no multiple-path sensitization and no self-masking possible} \]

\[ y_x = y_z \cdot z_x \quad \text{(composition rule)} \]

\[ y_x = y_z \cdot z_u \cdot u_x \]
\[ y_x = y_z \cdot y_z \cdot z_u \cdot u_x \]
\[ y_x = y_z \cdot y_u \cdot u_x \]

If \( y_x = 1 \), then \( y_z \cdot y_u = 1 \)

(single) fault-path: \( x \longrightarrow u \longrightarrow z \longrightarrow y \)

In combinational circuits with a tree structure, a test set for input variables also tests all internal variables.
Construction of the fault tree

Fault tree:
- created from the outputs towards the inputs using the chain rule
- consists of single fault paths
- has to be contiguous
- depends on the signal value assignments

Construction:

a) \( y_z \cdot y_w = 1 \)  

b) \( y_z \cdot \overline{y_w} = 1 \)

\[ \begin{array}{c}
\text{Fault tree} \\
\text{Sensitization marker}
\end{array} \]

\[
\begin{array}{c}
\text{c) } \overline{y_z} \cdot y_w = 1 \\
\text{d) } \overline{y_z} \cdot \overline{y_w} = 1
\end{array}
\]

Example:

\[
\begin{array}{c}
\text{Circuit C} \\
\text{Fault tree construction in C}
\end{array}
\]
3 Fault Simulation

The Task of Fault Simulation

- Fault simulation determines those faults out of all assumed faults which are detected by a given test set
- Application:
  - Evaluation of the test-set quality,
  - Acceleration of automatic test pattern generation (ATPG)

An Efficient Fault Simulation Approach

- fault-free simulation computes controllabilities; at signal x, fault x/ν (ν = val(x)) is excited
- observability analysis computes the set of observable signals S^O
  - simulation of faults at fanout-stems
  - composition of local observabilities in fanout-free regions
- determination of tested faults: F^t = {x/ν | x ∈ S^O ∧ ν = val(x)}

Example

- fault-free simulation:

```
   a  1
   b  1
   c  0
   d  1
   e  1

   1     1     1     1     0   0
   p     q     r     w     z

1 0 1 0 1 1
```
• observability analysis:
  
  - simulation of faults at fanout stems (e.g. $h/0$):

  - fanout stem $q/1$:

  - fanout stem $c/1$: 

  self masking

multi path-sensitization
- composition of local observabilities in all observable fanout-free regions (composition rule: e.g. \( w_b = w_q \cdot q_b \)):

\[
\begin{align*}
&\text{determination of tested faults:} \\
&S^O = \{b, c, c_2, h, h_1, q, q_1, w, z\} \\
&F^I = \{b/0, c/1, c_2/1, h/0, h_1/0, q/1, q_1/1, w/0, z/0\}
\end{align*}
\]
4 Automatic Test Pattern Generation (ATPG)

The task of ATPG is to generate a test set, such that a fault-free and a faulty circuit can be distinguished. For each assumed stuck-at fault a test pattern is generated.

Structure Oriented Test Generation Applying the D-Algorithm

5-valued logic:

- 0: binary '0'
- 1: binary '1'
- X: undefined, unknown, don’t care
- D: '1' if variable is fault-free, '0' if variable is faulty
- D: negated value of D

D-path $V_D = \{x, \ldots, y\}$: (single/multiple) fault-path for fault-effect propagation from the test variable $x$ to an output variable $y$

Tables for the D-Algorithm

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>D</td>
<td>X</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>D</td>
<td>X</td>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>⊕</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

Initialization all signals in the circuit have logic value X

Implication

- e.g. $[w = \text{val}(w)] \land [z = \text{val}(z)] \implies [y = \text{val}(y)]$ \quad w, z, y $\in V$
- or $[y = \text{val}(y)] \implies [w = \text{val}(w)] \land [z = \text{val}(z)]$ \quad val(w), val(z), val(y) $\in \{0, 1, X, D, D\}$

- e.g. local forward implication
- local backward implication

$$[w = 0] \land [z = X] \implies [y = 0] \quad [y = 1] \implies [w = 1] \land [z = 1]$$
Mandatory Sensitization Assignments

with respect to fault-path \( V_D = \{x, \ldots, y\} \) at signal \( w \not\in V_D \)

If \( V_D \) is sensitizable and \( w = \text{val}(w) \) desensitizes the fault-path, then \( w = \text{val}(w) \) has to be assigned.

Examples:

1) assumed faults at \( x_1 \) : \( V_D = \{x_1, z_1, y\} \)

\[
\begin{array}{cccc}
  x_1 & D & D & z_1 \\
  x_2 & 1 & & \\
  x_3 & X/0 & 1 & z_2 \\
  x_4 & 0/X & & \\
\end{array}
\]

<table>
<thead>
<tr>
<th>( x_4 ), ( x_3 ), ( x_2 ), ( x_1 )</th>
<th>( x_1/0 )</th>
<th>( x_1/1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 1, 1</td>
<td>( t_3 )</td>
<td></td>
</tr>
<tr>
<td>0, 1, 1, 1</td>
<td></td>
<td>( t_7 )</td>
</tr>
<tr>
<td>1, 0, 1, 1</td>
<td></td>
<td>( t_{11} )</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cc}
  x_4 & x_3 \\
  0 & 0 \\
  0 & 1 \\
  1 & 0 \\
\end{array}
\]

possible input value assignments

<table>
<thead>
<tr>
<th>( x_4 ), ( x_3 )</th>
<th>possible value assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, X</td>
<td></td>
</tr>
<tr>
<td>X, 0</td>
<td></td>
</tr>
</tbody>
</table>

2) assumed faults at \( x_0 \) : \( V_{D1} = \{x_0, z_1, y\} \)
\( V_{D2} = \{x_0, z_2, y\} \)
\( V_{D3} = \{x_0, z_1, z_2, y\} \)

\[
\begin{array}{cccc}
  x_1 & D, 1, D & D, 1, D & z_1 \\
  x_0 & D, D, D & & \\
  x_4 & 0, 1, 1 & 1, D, D & z_2 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>( x_4 ), ( x_0 ), ( x_1 )</th>
<th>( x_0/0 )</th>
<th>( x_0/1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 1</td>
<td>( t_3 )</td>
<td></td>
</tr>
<tr>
<td>1, 1, 0</td>
<td></td>
<td>( t_7 )</td>
</tr>
<tr>
<td>1, 1, 1</td>
<td></td>
<td>( t_{11} )</td>
</tr>
<tr>
<td>0, 0, 1</td>
<td></td>
<td>( t_6 )</td>
</tr>
<tr>
<td>1, 0, 0</td>
<td></td>
<td>( t_{10} )</td>
</tr>
<tr>
<td>1, 0, 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decision Tree

- consists of vertices and edges
- a vertex contains one optional value assignment and the resulting implications
- edges show the sequence of taken optional value assignments

backtracking: undo already performed optional assignments and investigate the alternatives

Test Generation

1. initialization of all signals with value X
2. fault excitation and resulting implications
3. for each D-path:
   - mandatory sensitization assignments along D-path
   - implications
4. contradiction?
   - yes: test pattern generated
   - no: optional value assignment
5. unjustified signal value?
   - yes: fault not testable along D-path
   - no: backtracking possible?
     - yes: backtracking
     - no: test pattern generated
Examples for D-Algorithm:

A) combinational circuits with tree structure

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad \rightarrow z_1 \quad \rightarrow y \quad \rightarrow z_2 \]

assumed faults at \( x_1 \):

\[ x_1/0 : \]

\[ \begin{align*}
\text{F} & : x_1 = D \\
\text{S} & : x_2 = 1 \\
\text{I} & : z_1 = D \\
\text{S} & : z_2 = 1 \\
\text{O} & : y = D
\end{align*} \]

\[ x_3 = 0 \quad x_4 = X \]

\[ \begin{array}{c}
0 \\
1 \\
0
\end{array} \]

\[ \begin{array}{c}
0 \\
1
\end{array} \]

\[ \begin{array}{c}
1 \\
1
\end{array} \]

\[ t_3 \]

\[ t_{11} \]

\[ t_7 \]

tests for \( x_1/0 \):

\[ \begin{array}{cccc}
x_4 & x_3 & x_2 & x_1 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1
\end{array} \]

\[ t_3 \quad t_{11} \quad t_7 \]

assumed faults at \( z_1 \):

\[ z_1/0 : \]

\[ \begin{align*}
\text{F} & : z_1 = D \\
\text{S} & : z_2 = 1 \\
\text{I} & : y = \overline{D}
\end{align*} \]

\[ x_1 = 0 \\
\text{O} : x_2 = X \\
\text{O} : x_3 = 0 \\
\text{O} : x_4 = X \\
\text{O} : x_4 = 0 \\
\text{O} : x_4 = X \\
\text{O} : x_4 = 0 \\
\text{O} : x_4 = X \\
\text{O} : x_4 = 0
\]

\[ t_6 \]

tests for \( z_1/0 \):

\[ \begin{array}{cccc}
x_4 & x_3 & x_2 & x_1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0
\end{array} \]

\[ t_2 \quad t_{10} \quad t_6 \]
tests for $z_1/0$:

<table>
<thead>
<tr>
<th>$x_4$</th>
<th>$x_3$</th>
<th>$x_2$</th>
<th>$x_1$</th>
<th>$t_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$t_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$t_8$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$t_{10}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$t_4$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$t_6$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$t_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$t_9$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$t_5$</td>
</tr>
</tbody>
</table>

tests for $z_1/1$:

<table>
<thead>
<tr>
<th>$x_4$</th>
<th>$x_3$</th>
<th>$x_2$</th>
<th>$x_1$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$t_3$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$t_{11}$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$t_7$</td>
</tr>
</tbody>
</table>

B) combinational circuits with reconvergent regions

assumed fault: $h/0$
given multiple fault-path: $V_D = \{h, q, r, z\}$

A possible test pattern $t$ for detection of fault $h/0$ along the multiple fault-path $V_D = \{h, q, r, z\}$: $t = (a, b, c, d, e) = (X, 1, 0, X, 1)$

Please note: Test pattern $t$ also tests fault $h/0$ along the single fault-path $V_D' = \{h, q, w\}$.
Methods to accelerate ATPG

A) Global implications

Precondition: no fault effect on the considered part of the circuit

Law of contraposition: \( (P \implies Q) \iff (\neg Q \implies \neg P) \)

Learning procedure:

\[
\begin{align*}
( a = 0 ) \implies ( f = 0 ) & \iff \neg(f = 0) \implies \neg(a = 0) \\
( a = 0 ) \implies ( f = 0 ) & \iff ( f = 1 ) \implies ( a = 1 )
\end{align*}
\]

Remark: Only those global implications should be learned, which cannot be reduced to a sequence of local implications.

Learning criteria:

\[
\begin{align*}
\text{forward implication:} & \quad x = \text{val}(x) \implies y = \text{val}(y) \quad \text{val}(x), \text{val}(y) \in \{0, 1\} \\
\text{backward implication:} & \quad y = \text{val}(y) \implies x = \text{val}(x) \\
& \quad \text{is worthy of learning, if for the forward implication it holds: } y_z \cdot y_w = 1
\end{align*}
\]
B) Controllability

Selection ranking for optional value assignments using controllabilities \( C_0, C_1 \)

- \( C_0: \) zero controllability, \( 0 \leq C_0 \leq 1 \)
- \( C_1: \) one controllability, \( 0 \leq C_1 \leq 1 \)

\( C_0 + C_1 = 1 \)

\( C_0(x) = 0.5 \), \( C_1(x) = 0.5 \), \( x: \) input variable

\[
\begin{align*}
\text{AND:} & \quad C_1(c) &= C_1(a) \cdot C_1(b) \\
& \quad C_0(c) &= 1 - C_1(c) \\
\text{OR:} & \quad C_0(c) &= C_0(a) \cdot C_0(b) \\
& \quad C_1(c) &= 1 - C_0(c) \\
\text{XOR:} & \quad C_1(c) &= C_1(a) \cdot C_0(b) + C_0(a) \cdot C_1(b) \\
& \quad C_0(c) &= 1 - C_1(c) \\
\text{NOT:} & \quad C_0(c) &= C_1(a) = 1 - C_0(a) \\
& \quad C_1(c) &= C_0(a) = 1 - C_1(a)
\end{align*}
\]

Procedure: In case of alternatives to justify a value of a signal, the one having the largest controllability is chosen.

Calculating \( C_0 \) and \( C_1 \) is very fast. However, their values are only exact for circuits with a tree structure.

Example:

\[
\begin{align*}
C_0(x_1) &= C_1(x_1) = 0.5 \\
C_0(x_2) &= C_1(x_2) = 0.5 \\
C_0(x_3) &= C_1(x_3) = 0.5 \\
C_0(x_4) &= C_1(x_4) = 0.5 \\
C_0(z_1) &= C_1(x_1) \cdot C_1(x_2) = 0.25 \\
C_1(z_1) &= 1 - C_0(z_1) = 0.75 \\
C_0(z_2) &= C_1(x_3) \cdot C_1(x_4) = 0.25 \\
C_1(z_2) &= 1 - C_0(z_2) = 0.75 \\
C_0(y) &= C_1(z_1) \cdot C_1(z_2) = 0.5625 \\
C_1(y) &= 1 - C_0(y) = 0.4375
\end{align*}
\]
5 ATPG for Synchronous Sequential Circuits

Sequential Circuit

\[
\begin{align*}
  x &= \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \\
  s &= \begin{bmatrix} s_1 \\ \cdot \\ \cdot \\ s_m \end{bmatrix} \\
  x &\rightarrow \text{combinational block CB} \\
  s &\rightarrow \text{memory} \\
  y &= \begin{bmatrix} y_1 \\ \cdot \\ \cdot \end{bmatrix} \\
  z &= \begin{bmatrix} z_1 \\ \cdot \\ \cdot \end{bmatrix}
\end{align*}
\]

- memory units (flipflops) are generally not directly accessible from inputs \( x \)
- in general, testing takes multiple clock periods (circuit initialization, fault excitation, sensitization)

Iterative Logic Array (ILA):

- a single stuck-at fault exists at all time frames: single fault \( \rightarrow \) multiple fault

input variables: \( \sim X = \begin{bmatrix} \ldots x_{t-1}^t x_t x_{t+1}^t \ldots \end{bmatrix} \)

test points: \( \sim Y = \begin{bmatrix} \ldots y_{t-1}^t y_t y_{t+1}^t \ldots \end{bmatrix} \)

state variables: \( \sim S = \begin{bmatrix} \ldots s_{t-1}^t s_t s_{t+1}^t \ldots \end{bmatrix} \)

variables of next state: \( \sim Z = \begin{bmatrix} \ldots z_{t-1}^t z_t z_{t+1}^t \ldots \end{bmatrix} \)

\( \sim S \) can be controlled by \( \sim X \) and observed at \( \sim Y \).
Test Generation by Comparison of Fault-Free and Faulty Circuit

Condition for Fault Detection: \( y_1^t(X) \oplus y_{\mu_1}^t(X) = 1 \)

The value \( y_{\mu_1} \) of an output variable at time frame \( t \) has to differ in the fault-free and faulty circuit when applying the input sequence \( X \).

Example for Determination of a Test Sequence Using Boolean Algebra:

\[
\begin{align*}
X & = \begin{bmatrix} 1 & 0 & 1 \\ X & 1 & X \end{bmatrix} \\
Y & = \begin{bmatrix} X & 0 & 1 \end{bmatrix} \\
Y_{\mu} & = \begin{bmatrix} X & 0 & 0 \end{bmatrix}
\end{align*}
\]
Test Generation Using D-Algorithm

The approach remains the same as the one for combinational circuits, but must be extended to consider time frames before and after the test time-frame.

**Example:**

```
\[
\begin{align*}
  \text{F} & : z_1^t = D \\
  \text{I} & : x_1^t = 0, y_1^t = 0, x_2^t = 1, y_2^t = 0, z_2^{-1} = 1, s^{-1} = 1, x_2^{-1} = 1, y_2^{-1} = 1 \\
  \text{S} & : z_1^t = D, z_2^t = D, x_2^t = D, y_2^t = D, x_2^{-1} = D, y_2^{-1} = D \\
  \text{I} & : z_1^t = x_1^t = y_1^t = x_2^t = y_2^t = z_2^t = s^t = x_2^{-1} = y_2^{-1} = z_2^{-1} = s^{-1} = 1
\end{align*}
\]
```

- Perform fault excitation at test time-frame $t$ and make all sensitizing assignments; then, determine the resulting implications. Also propagate values from a flipflop input to its output until the fault-effect reaches a circuit output.
- Justify flipflop states by assignments in the previous time frame (circuit initialization). If there are alternatives use the one which results in the unknown state “$X$”.
- Values necessary for sensitization may appear at signals along the fault-path only if the sensitizing value is the same as its faulty value. E.g., for “$D$” (stuck-at 0) the sensitizing value “0” is allowed.

---

ATPG for Synchronous Sequential Circuits (3)
Design for Testability

A) Enhancement of Controllability and Observability

- single signals:
  Testability analysis provides information about the quality of controllability and observability of single signals. Signals, which are hard to control, are directly excited and signals, which are hard to observe, get an extra output.

- Additional reset wires are used for setting flipflops into defined (not arbitrary) states (initialization).

- partitioning a circuit into partitions:
  reduces the test complexity
  – fewer inputs \(2^n \gg x \cdot 2^{n/x}\)
  – fewer functions (gates, circuit depth)

  allows (functional) tests for functional units (e.g. μprocessor, RAM etc.)
Reducing the number of additionally needed circuit input- and output-pins by using multiplexers. Already existing pins are switched for testing purposes.

![Diagram](image)

**B) Scan-Path**

- All flipflops can directly be set and read. In case of a partial scan-path, the scan-chain does not contain all flipflops.
- In testing mode, the sequential circuit becomes combinational.
**Signature Analysis**

**Basics**
- modeling (0,1)-sequences by polynomials:
  \[\tilde{a}_{<m+1>} = \{a_m, a_{m-1}, \ldots, a_2, a_1, a_0\} \quad ; \quad a_i \in \{0,1\}\]
  \[a(X) = a_mX^m + a_{m-1}X^{m-1} + \ldots + a_2X^2 + a_1X + a_0\]
  e.g. sequences: \(\{1, 1, 0, 1, 0, 0, 0, 1, 0, 1\}\)
  polynomial: \(X^9 + X^8 + X^6 + X^2 + 1\) (first \(\ldots\) last element)

- modeling linear feedback shift registers (LFSR) by polynomials:

  \[
  \begin{array}{c}
  \text{input sequence} \\
  a(X) \\
  \downarrow \\
  \text{D} \\
  \downarrow \\
  b_0 \\
  \downarrow \\
  \text{D} \\
  \downarrow \\
  b_1 \\
  \downarrow \\
  \text{D} \\
  \downarrow \\
  \vdots \\
  \downarrow \\
  \text{D} \\
  \downarrow \\
  b_{n-1} \\
  \downarrow \\
  \text{D} \\
  \downarrow \\
  b_n \\
  \end{array}
  \rightarrow 
  \begin{array}{c}
  \text{output sequence} \\
  c(X) \\
  \end{array}
  \]

  characteristic polynomial \(b(X) = b_0 + b_1X + b_2X^2 + \ldots + b_{n-1}X^{n-1} + b_nX^n\)

  register state \(\tilde{s}_{<n>} = \{s_0, s_1, s_2, \ldots, s_{n-2}, s_{n-1}\}\)

**Applications of LFSRs**
- random pattern generator:

  For an input sequence \(a(X) = 0\) and an initial state \(s(X) \neq 0\), a LFSR can be used as a pseudo-random generator of (0,1)-sequences. For that purpose, LFSRs with irreducible polynomials are used which produce maximal-length sequences (\(2^n - 1\)) and go through every state except state \(s(X) = 0\).

- signature register:

  When applying the input sequence \(\tilde{a}_{<m+1>}\) to a LFSR \((b(X))\), the output sequence \(\tilde{c}_{<m-n+1>}\) is obtained after \(m + 1\) cycles and the register is in state \(\tilde{s}_{<n>}\).

  The state \(\tilde{s}_{<n>}\) is called the signature of \(\tilde{a}_{<m+1>}\) with respect to \((b(X))\) and can be computed by:

  \[
  \frac{a(X)}{b(X)} = \frac{c(X) + s(X)}{b(X)}
  \]

ATPG for Synchronous Sequential Circuits (6)
Testing by Means of Signature Analysis

The UUT is stimulated using pseudo-random patterns and its output sequence is compressed to its signature \( m \gg n \) which is compared to the fault-free signature.

This data compression leads to a loss of information since \( 2^{m+1} \) possible output sequences are mapped to \( 2^n \) signatures. Therefore, \( 2^{m+1-n} \) sequences have the same signature.

fault masking: faulty sequence has the same signature as the fault-free sequence

\[ p_f = \frac{2^{m+1-n} - 1}{2^{m+1} - 1} \]

\[ p_f \approx \frac{1}{2^n} \quad (m \gg n) \]
Built-In Self-Test (BIST)

Generation of test patterns and test evaluation are carried out within the circuit.

Advantages

- no automatic test equipment (ATE) necessary
- test hardware is equivalent to chip technology
- only few additional pins necessary
- test can be performed at any time (on-line, off-line)

Signature Analysis Using a BILBO (Built-In Logic Block Observer)

Existing flip-flops are arranged as BILBO elements and the circuit is partitioned into combinational blocks.

1. BILBO \rightarrow \text{comb. block 1} \rightarrow \text{BILBO} \rightarrow \text{comb. block 2}

internal structure of a BILBO:

\[ B_1 \quad \text{shift in} \quad B_2 \quad \text{shift out} \]

\[ \begin{array}{c}
\text{MUX} \\
\text{D} \\
\text{Q} \\
\text{O_1} \\
\text{O_2} \\
\text{O_n} \\
\end{array} \]

\[ \begin{array}{c}
\text{I_1} \\
\text{I_2} \\
\text{I_n} \\
\end{array} \]

- operation mode
- shift register; function depending on MUX:
  - scan path
  - random pattern generator
    - with parallel outputs
- signature register with parallel inputs
Literature

Logic Synthesis


Simulation


Available on WWW.

Testing
