1 Problem Set Part A

Chapter 1: Introduction

1.1

The events of Example 1.1 are redefined as follows:

- PQ: chip is good
- P: chip passes the test
- FQ: chip is bad
- F: chip fails the test

A 70% yield means, \( \text{Prob}(PQ) = 0.7 \) and \( \text{Prob}(FQ) = 0.3 \). Following the analysis of Example 1.1, \( \text{Prob}(P) = 0.68 \). Then,

\[
\text{Defect level} = \frac{\text{Bad chips that pass tests}}{\text{All chips that pass tests}} = \frac{\text{Prob}(FQ|P)}{\text{Prob}(P)} = \frac{\text{Prob}(P|FQ)\text{Prob}(FQ)}{\text{Prob}(P)}
\]

\[
= \frac{0.05 \times 0.3}{0.68} = 0.022
\]

The defect level is 22,000 ppm (parts per million).

1.4

Following Example 1.2 of the book (pp. 10-11), we obtain

\[
\text{ATE purchase price} = 1.2M + 256 \times 3,000 = 1.968M
\]

Assuming a 20% per year linear rate of depreciation, a maintenance cost of 2% of the price, and an annual operating cost of \$0.5M,

\[
\text{Running cost} = 1.968M \times 0.2 + 1.968M \times 0.02 + 0.5M = 932,960/\text{year}
\]
Testing cost of the self-test design is 2.96 cents per second, down from 4.56 cents per second calculated in Example 1.2

Chapter 4: Fault Modeling

4.2 Initialization faults

In the circuit of Figure 4.1 (p. 62 of the book), let $Q_p$ denote the present state at the output of the $FF$. Let the next state, i.e., the output of the AND gate, be $Q_n$. We can write the next state function, as

$$Q_n = (Q_p + A)(\overline{A} + B)$$

If we set $A = 1$, the next state function, $Q_n = B$, becomes independent of the present state. That is, irrespective of the present state, the next state can be set to a value, which is uniquely determined by primary inputs. This makes the fault-free circuit initializable. When the fault $A$ s-a-0 is present, the above equation reduces to $Q_n = Q_p$. Thus, starting with $Q_p = X$, $Q_n$ can never be changed to any value other than $X$ and, therefore, the circuit will remain uninitialized in the presence of this fault.

Using the next-state expression, we can easily determine that no other single stuck-at fault in this circuit will prevent initialization. For example, consider the s-a-0 fault on the top branch of the fanout of $A$. The faulty next state function is $Q_n = Q_p(\overline{A} + B)$, which can be set to 0, when $Q_p = X$, by applying $A = 1$, $B = 0$.

4.4 Fault counting

For the circuit of Figure 4.6 (p. 72 of book), we have

$$\text{Number of fault sites} = \text{PIs} + \text{gates} + \text{fanout branches} = 2 + 4 + 6 = 12$$

Therefore,

$$\text{Number of single and multiple faults} = 3^{\text{number of fault sites}} - 1 = 3^{12} - 1 = 531,440$$

The circuit has 531,440 single and multiple stuck-at faults.

4.5 CMOS faults

(a) A two-input NAND gate is shown in the above figure. The following table gives tests for transistor stuck-open (sop) faults:
Circuit for Problem 4.5.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Fault</th>
<th>Test: Vector 1, Vector 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1 sop</td>
<td>11, 01</td>
</tr>
<tr>
<td>2</td>
<td>P2 sop</td>
<td>11, 10</td>
</tr>
<tr>
<td>3</td>
<td>N1 sop</td>
<td>01, 11 or 10, 11 or 00, 11</td>
</tr>
<tr>
<td>4</td>
<td>N2 sop</td>
<td>01, 11 or 10, 11 or 00, 11</td>
</tr>
</tbody>
</table>

Notice that the sop faults of N1 and N2 have exactly the same tests. These two faults are equivalent.

(b) The following sequence of four vectors contains one vector pair for each fault in the above table:

11, 01, 11, 10

Notice that this sequence also detects all single stuck-at faults in the logic model of the NAND gate.

(c) A stuck-at fault in a signal affects two transistors in the two-input NAND gate. For example, the fault A s-a-1 will mean that N1 remains permanently shorted (N1-ssh) and P1 remains permanently open (P1-sop). The following table gives all equivalences:

<table>
<thead>
<tr>
<th>Stuck-at fault</th>
<th>Equivalent transistor faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>A s-a-1</td>
<td>N1-ssh and P1-sop</td>
</tr>
<tr>
<td>B s-a-1</td>
<td>N1-ssh and P2-sop</td>
</tr>
<tr>
<td>C s-a-1</td>
<td>(P1-ssh or P2-ssh) and (N1-sop or N2-sop)</td>
</tr>
<tr>
<td>A s-a-0</td>
<td>N1-sop and P1-ssh</td>
</tr>
<tr>
<td>B s-a-0</td>
<td>N2-sop and P2-ssh</td>
</tr>
<tr>
<td>C s-a-0</td>
<td>N1-ssh, N2-ssh, P1-sop and P2-sop</td>
</tr>
</tbody>
</table>

Notice that the three equivalent faults, A s-a-0, B s-a-0 and C s-a-0, are actually caused by different faulty transistors. They are detected by the same test (11).
4.8 Functional equivalence

Faulty functions for the circuit of Figure 4.12 corresponding to the two faults are:

\[ i(c \bar{s} - a - 0) = b(\overline{ab}) = \overline{ab} \]
\[ i(f \bar{s} - a - 1) = (a + b)\overline{a} = \overline{ab} \]

The two faulty functions are indistinguishable and hence **the two faults are equivalent**.

4.9 Functional equivalence

Faulty functions for the circuit of Figure 4.6 corresponding to the two faults are:

\[ z(c \bar{s} - a - 1) = \overline{ab}.(\overline{ab}.b) = \overline{ab} \]
\[ z(f \bar{s} - a - 1) = \overline{ab} \]

The two faulty functions are indistinguishable and hence **the faults are equivalent**.

4.10 Fault collapsing for test generation

The circuit of Figure 4.9 has 18 single stuck-at faults. Gate-level fault equivalence, as shown in the following figure, reduces the number to 12. The faults in shaded boxes have been collapsed as shown by arrows. Many ATPG and fault simulation programs will collapse faults as shown above. However, functional fault collapsing can further reduce the number of faults to 10. As shown in Example 4.11 (see page 75 of the book), the s-a-1 faults on \( A1 \) and \( B1 \) are equivalent, and so are the s-a-1 faults on \( A2 \) and \( B2 \).

**Whether we take the set of 12 faults or the set of 10 faults, their detection requires all four input vectors.**

4.11 Equivalence and dominance fault collapsing

(a) The given circuit is shown below with fault sites marked by numbers. The number of potential fault sites is 18. The total number of faults is 36.
(b) The figure shows deletion of equivalent faults using an output to input pass. Of the 36 faults, 20 remain, giving a collapse ratio \( \frac{20}{36} = 0.56 \).

(c) Checkpoint lines are shown by boldface numbers. These are three PIs and seven fanout branches. Line 2 fans out to 4 and 5. Line 3 fans out to 6, 7 and 8. Line 10 fans out to 12 and 13. There are ten checkpoints and 20 checkpoint faults. Further, s-a-0 faults on lines 6 and 12 are equivalent and any one of them can be chosen. Similarly, s-a-0 faults on 7 and 13 are equivalent, and so are s-a-0 on 5 and s-a-1 on 8. Thus, the size of the fault set is reduced to 17, giving a collapse ratio \( \frac{17}{36} = 0.47 \).

4.12 Dominance fault collapsing

(a) Checkpoints are defined for the signals in a combinational circuit. These signals are the interconnects between Boolean gates, a fact not always explicitly stated. To avoid ambiguity, the definition on page 78 of the book should read as:

**Definition 4.7 Checkpoints.** Primary inputs and fanout branches of a combinational circuit consisting only of Boolean gates are called the checkpoints.

To find checkpoints of the circuit of Figure 4.12, we must replace the exclusive-OR (XOR) function by a primitive Boolean gate implementation. AND, OR, NAND, NOR and NOT are called the primitive Boolean gates. Functions such as XOR are sometimes referred to as complex gates. In the following figure, we have assumed one such implementation. Our result is, therefore, based on this assumption. Other implementations of the XOR function are possible and can give a different set of checkpoints.

There are nine checkpoints in this circuit. These include three primary inputs, \( a, b \) and \( c \), and six fanout branches, \( d1, d2, f, e1, e2 \) and \( g \). The checkpoint fault set consists of eighteen faults – s-a-0 and s-a-1 faults on the nine lines.
Notice that lines $d$ and $e$ of the original circuit are not checkpoints. If we did not model the XOR block with Boolean gates, then those lines will appear to be checkpoints, whose number will be fourteen. However, detection of those faults will not guarantee detection of faults on the fanouts that are internal to the XOR block. Considering the Boolean gate structure, a fault on $d$ corresponds to a simultaneous (multiple) fault on $d_1$ and $d_2$ and, in general, the detection of a multiple fault is not equivalent to detection of the component faults.

(b) We evaluate the output function $k$ corresponding to the two faults:

\[
k(d s - a - 0) = \overline{c} + \overline{b} + \overline{a} + \overline{b} = \overline{c} + \overline{b} + ab
\]

\[
k(g s - a - 1) = \overline{c} + \overline{ab} + \overline{a} + a = \overline{c} + \overline{ab} + a
\]

The two faulty functions are shown by Karnaugh maps below. In both cases, the functions have exactly one false minterm, $\overline{abc}$. Since the two faulty functions are identical the corresponding faults are equivalent.

Note: this type of fault equivalence is functional and is often difficult to find by typical fault analysis tools, which rely on structurally identifiable equivalences.
Chapter 7: Combinational Circuit ATPG

7.1 Cubes

**AND gate:**

\[
\begin{array}{c}
\text{Singular cover:} \\
\begin{array}{ccc}
0 & X & 0 \\
X & 0 & 0 \\
1 & 1 & 1
\end{array}
\end{array}
\]

Propagation \( D \) cubes (last two cubes are not propagation \( D \)-cubes since they do not propagate \( D \) or \( \overline{D} \)):

\[
\begin{array}{ccc}
1 & D & D \\
D & 1 & D \\
D & D & D \\
1 & \overline{D} & \overline{D}
\end{array}
\]

Primitive \( D \) cube of failure for \( a \) sal:

\[
\begin{array}{ccc}
0 & 1 & \overline{D}
\end{array}
\]

**Exclusive-OR gate:**

\[
\begin{array}{c}
\text{Singular cover:} \\
\begin{array}{ccc}
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
0 & 0 & 0
\end{array}
\end{array}
\]

Propagation \( D \) cubes (last four cubes are not propagation \( D \)-cubes since they do not propagate \( D \) or \( \overline{D} \)):

\[
\begin{array}{ccc}
0 & D & D \\
D & 0 & D \\
D & 1 & \overline{D} \\
1 & D & \overline{D} \\
0 & \overline{D} & \overline{D} \\
\overline{D} & 0 & \overline{D} \\
1 & \overline{D} & D \\
\overline{D} & 1 & D
\end{array}
\]

Primitive \( D \) cubes of failure for \( a \) sal:

\[
\begin{array}{ccc}
0 & 0 & \overline{D} \\
0 & 1 & D
\end{array}
\]
7.2 Stuck-at fault testing

(a) Three tests for a two-input OR gate:

\[ a \quad b \rightarrow c \]

<table>
<thead>
<tr>
<th>Vector number</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>Collapsed faults tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( \overline{D} )</td>
<td>( a , s_{1}, , b , s_{1}, , c , s_{1} )</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>( D )</td>
<td>( b , s_{0}, , c , s_{0} )</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>( D )</td>
<td>( a , s_{0}, , c , s_{0} )</td>
</tr>
</tbody>
</table>

(b) Gate replacements:

<table>
<thead>
<tr>
<th>OR replaced by:</th>
<th>Test results</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>pass, fail, fail</td>
</tr>
<tr>
<td>NAND</td>
<td>fail, pass, pass</td>
</tr>
<tr>
<td>NOR</td>
<td>fail, fail, fail</td>
</tr>
</tbody>
</table>

The three-vector test will detect the error if the OR gate were to be replaced by an AND, NAND or NOR gate.

(c) OR gate replaced by an exclusive-OR gate: All three vectors will produce the same output as that of the OR gate. Therefore, this error will not be detected. It is necessary to include a fourth vector 11 to detect this error. The addition of the fourth vector makes the vector set exhaustive, which completely verifies the truth table of the gate.

7.3 D-ALG

We level order the signals and proceed as follows:

<table>
<thead>
<tr>
<th>Step no.</th>
<th>Action</th>
<th>Signals</th>
<th>( D ) front.</th>
<th>Impl. stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault Activation</td>
<td>( 0 , 0 , \overline{D} )</td>
<td>( k )</td>
<td>( g = 0 )</td>
</tr>
<tr>
<td></td>
<td>Immediate impl.</td>
<td>( 0 , 0 , 0 , \overline{D} )</td>
<td>( k )</td>
<td>( g = 0 )</td>
</tr>
<tr>
<td></td>
<td>Immediate impl.</td>
<td>( 1 , 1 , 0 , 0 , 0 , \overline{D} )</td>
<td>( k )</td>
<td>( g = 0 )</td>
</tr>
<tr>
<td></td>
<td>Immediate impl.</td>
<td>( 1 , 1 , 0 , 0 , 0 , 0 , \overline{D} )</td>
<td>( k )</td>
<td>( g = 0 )</td>
</tr>
<tr>
<td></td>
<td>Immediate impl.</td>
<td>( 1 , 1 , 0 , 0 , 0 , 0 , \overline{D} )</td>
<td>( \phi )</td>
<td>( g = 0 )</td>
</tr>
<tr>
<td></td>
<td>Immediate impl.</td>
<td>( 1 , 1 , 0 , 0 , 0 , 0 , \overline{D} )</td>
<td>( \phi )</td>
<td>( g = 0 )</td>
</tr>
</tbody>
</table>
The fault is redundant, because the $D$-frontier disappeared. No backtracks. Signals are shown in the following figure.

![Diagram](image.png)

### 7.4 D-ALG

We level order the signals and proceed as follows:

<table>
<thead>
<tr>
<th>Step no.</th>
<th>Action</th>
<th>Signals</th>
<th>$D$ front.</th>
<th>Impl. stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault activation</td>
<td>$A B C d e f g Y h k Z$</td>
<td>$k$</td>
<td>$g = 1$</td>
</tr>
<tr>
<td>2</td>
<td>$D$-drive $h \rightarrow k$</td>
<td>$A B C d e f g Y h k Z$</td>
<td>$Z$</td>
<td>$f = 1$ $g = 1$</td>
</tr>
<tr>
<td>3</td>
<td>$D$-drive $k \rightarrow Z$</td>
<td>$A B C d e f g Y h k Z$</td>
<td>$PO$</td>
<td>$f = 0$ $g = 1$</td>
</tr>
<tr>
<td></td>
<td>Immediate Impl.</td>
<td>$A B C d e f g Y h k Z$</td>
<td>$PO$</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>Immediate Impl.</td>
<td>$A B C d e f g Y h k Z$</td>
<td>$PO$</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>Immediate Impl.</td>
<td>$A B C d e f g Y h k Z$</td>
<td>$PO$</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

The test is: $A = X$, $B = 0$, $C = 1$ as shown in the following figure; 0 backtracks.

![Diagram](image.png)

### 7.7 PODEM and FAN

The following figure shows the SCOAP testability measures used for guiding PODEM and FAN.
The following table gives the steps that PODEM takes:

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Objective</th>
<th>Action</th>
<th>Imp. stack</th>
<th>Implied signal values</th>
<th>D front.</th>
<th>X path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$r = 0$</td>
<td>Backtrace</td>
<td>$A = 0$</td>
<td>$r = 0(D), u = 0$</td>
<td>$Z$</td>
<td>ok</td>
</tr>
<tr>
<td>2</td>
<td>$w = 1$</td>
<td>Backtrace</td>
<td>$B = 1$</td>
<td>$A = 0, B = 1, p = 0, q = 1, s = 1$</td>
<td>PO</td>
<td>ok</td>
</tr>
</tbody>
</table>

Algorithm termination: Fault detected with 0 backtracks. Test is $\{ABCDEF\} = \{01\overline{XXX}\}$

For an explanation of $X$-path, see Problem 7.5.

(a) FAN ATPG. Step 1 is the same as for PODEM.

Step 2. Goal: propagate $\overline{D}$ from $r$ to $Z$.
Goal: set $w = 1$, 1 vote for 1, set $q = 1$, 1 vote for 1, set $s = 1$, 1 vote for 1, set $B = 1$, 2 votes for 1, no votes for 0. Rest of step 2 is exactly the same as PODEM.

Headlines are $m$ and $l$.
Initial objective: set $r = 0$.
Final objective: set $B = 1$.
Head objectives: not used. 0 backtracks.

(b) The following (on the next page) lists dominators for all signals.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Dominators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z$</td>
<td>$-$</td>
</tr>
<tr>
<td>$r$</td>
<td>$Z$</td>
</tr>
<tr>
<td>$u$</td>
<td>$Z$</td>
</tr>
<tr>
<td>$w$</td>
<td>$Z$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gate</th>
<th>Dominators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$r, Z$</td>
</tr>
<tr>
<td>$q$</td>
<td>$w, Z$</td>
</tr>
<tr>
<td>$s$</td>
<td>$w, Z$</td>
</tr>
<tr>
<td>$m$</td>
<td>$Z$</td>
</tr>
<tr>
<td>$l$</td>
<td>$s, w, Z$</td>
</tr>
</tbody>
</table>
7.8 PODEM

The figure below shows the SCOAP testability measures used for guiding PODEM.

![PODEM Diagram]

The following table gives the steps of PODEM (see Problem 7.5 for an explanation of X-path):

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Objective</th>
<th>Action</th>
<th>Imp. stack</th>
<th>Implied signal values</th>
<th>( D ) front.</th>
<th>( X ) path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( g = 0(D) )</td>
<td>Backtrace</td>
<td>( C = 0 )</td>
<td>( C = 0, h = 0 )</td>
<td>( \phi )</td>
<td>ok</td>
</tr>
<tr>
<td>2</td>
<td>( g = 0(D) )</td>
<td>Backtrace</td>
<td>( D = 0 )</td>
<td>( C = 0, D = 0, g = 0(D) )</td>
<td>( \phi )</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>( g = 0(D) )</td>
<td>Backtrack</td>
<td>( D = 1 )</td>
<td>( C = 0, D = 1, g = 1, h = 0 )</td>
<td>( \phi )</td>
<td>none</td>
</tr>
<tr>
<td>4</td>
<td>( g = 0(D) )</td>
<td>Backtrace</td>
<td>( C = 1 )</td>
<td>( C = 1, g = 1, h = 1, m = 1 )</td>
<td>( \phi )</td>
<td>none</td>
</tr>
<tr>
<td>5</td>
<td>( g = 0(D) )</td>
<td>Backtrack</td>
<td>Empty</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Algorithm termination: \( g = 0(D) \) with X-path impossible; fault \( g \) s-a-1 is redundant. 3 backtracks.

7.10 FAN

The following figure shows the SCOAP testability measures used for FAN. The headlines are \( m \) and \( l \).

**Step 1:** Goal – set \( r = 1 \) → set \( A = p = 1 \) → set \( A = m = B = 0 \)

\( D \)-frontier = \( \phi \).

Conflict at stem \( A \), choose \( A = 1 \)

Forward imply \( A = 1, B = 0, p = 0, r = 0 \), fault \( r \) s-a-0 cannot be sensitized—is redundant (0 backtracks).
7.11 SOCRATES

The following figure shows the SCOAP testability measures used for SOCRATES.

**Step 1:** Objective – set \( m = 1 \)

Implication stack – \( C = 1 \)
Assignments – \( C = 1, g = 1, h = 1, m = 1, q = 1 \)

\( D \)-frontier – \( p \)
Dynamic learning – \( (k = 0) \Rightarrow (D = 0), (w = 0) \Rightarrow (l = 0) \)

**Step 2:** Objective – set \( A = 0 \)

Implication stack – \( A = 0, C = 1 \)
Assignments – \( C = 1, g = 1, h = 1, m = 1, q = 1, A = 0, r = 0, u = 0 \)

X-path check – fault propagation path blocked – alternative assignment \( A = 1 \) infeasible

*Fault \( m – p \) s-a-0 is redundant (no backtracks).*

No applications of Modus Tollens or constructive dilemma.
### Static learning

<table>
<thead>
<tr>
<th>Signal</th>
<th>Learned implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B = 1$</td>
<td>$(w = 0) \Rightarrow (B = 0)$</td>
</tr>
<tr>
<td>$g = 0$</td>
<td>$(m = 1) \Rightarrow (g = 1)$</td>
</tr>
<tr>
<td>$C = 1$</td>
<td>$(h = 0) \Rightarrow (C = 0)$</td>
</tr>
<tr>
<td>$D = 1$</td>
<td>$(k = 0) \Rightarrow (D = 0)$</td>
</tr>
</tbody>
</table>

#### 7.15 FAN

The following figure shows the SCOAP testability measures used to guide the ATPG. The signal values are those determined by the steps described below.

There are no headlines.

**Step 1:** Goal – sensitize fault

Implication stack – $B = 0$

Implied signals – $B = 0, d = e = f = g = 0, h = l = 0, k = \overline{D}, m = n = p$

$X = 1, Y = 1$

**Step 2:** Goal – propagate fault to $Z$

Implication stack – $B = 0, C = 1$

Implied signals – $B = 0, C = 0, d = e = f = g = 0, h = l = 0, k = \overline{D}, m = n = p = D, q = \overline{D}, r = \overline{D}, s = t = u = D, v = \overline{D}$

$X = 1, Y = 1, Z = 1$

**Sensitization and propagation conditions do not intersect. Hence, the fault is proved redundant in two steps without any backtracks.**
7.17 SOCRATES

To obtain a test for the fault n s-a-1 in the circuit of Figure 7.24 (see page 190 of the book and the figure below), we perform static learning:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Learned implications</th>
<th>Signal</th>
<th>Learned implications</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B = 0$</td>
<td>$(m = 0) \Rightarrow (B = 1)$</td>
<td>$X = 0$</td>
<td>$(r = 1) \Rightarrow (X = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(q = 1) \Rightarrow (B = 1)$</td>
<td></td>
<td>$(Y = 1) \Rightarrow (X = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(r = 1) \Rightarrow (B = 1)$</td>
<td></td>
<td>$(v = 1) \Rightarrow (X = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(s = 0) \Rightarrow (B = 1)$</td>
<td></td>
<td>$(q = 1) \Rightarrow (X = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(v = 1) \Rightarrow (B = 1)$</td>
<td></td>
<td>$(s = 1) \Rightarrow (X = 1)$</td>
</tr>
<tr>
<td>$d = 1$</td>
<td>$(m = 0) \Rightarrow (d = 0)$</td>
<td>$Y = 0$</td>
<td>$(m = 0) \Rightarrow (Y = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(q = 1) \Rightarrow (d = 0)$</td>
<td></td>
<td>$(v = 1) \Rightarrow (Y = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(r = 0) \Rightarrow (d = 0)$</td>
<td></td>
<td>$(r = 0) \Rightarrow (Y = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(s = 1) \Rightarrow (d = 0)$</td>
<td></td>
<td>$(d = 0) \Rightarrow (Y = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(X = 1) \Rightarrow (d = 0)$</td>
<td></td>
<td>$(m = 0) \Rightarrow (Y = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(Y = 1) \Rightarrow (d = 0)$</td>
<td></td>
<td>$(v = 1) \Rightarrow (Y = 1)$</td>
</tr>
<tr>
<td></td>
<td>$(v = 1) \Rightarrow (d = 0)$</td>
<td></td>
<td>$(Z = 0) \Rightarrow (d = 0)$</td>
</tr>
<tr>
<td>$Z = 1$</td>
<td>$(q = 1) \Rightarrow (Z = 1)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Step 1:** Goal – sensitize fault, $m = 0$

Static learning – $B = 1$

Implications – $d = 0, X = 1, Y = 1, A = 0, r = D, q = 1, m = 0, s = \overline{D}$, $v = D, Z = 1$

$D$-frontier – $\phi$ (null)

Redundant fault, because $D$-frontier vanishes at gate $Z$, no decision alternatives. No need for dynamic learning, no use of the constructive dilemma or Modus Tollens. No backtracks.

[Diagram of the circuit]
7.19 Redundancy proofs

(1) Proof of $d$ s-a-0 redundant using PODEM.

Implication – $d = D$. $D$-frontier – $g$.

**Step 2:** Goal – propagate fault. Objective – $e = 0$. Backtrace – Implication stack – $a = 1, b = 0$.
Implications – $d = D, g = D, h = 1, n = \overline{D}, p = D, q = 1$.
$D$-frontier – $\phi$.

Fault proved redundant because $D$-frontier disappears at $q$ – no alternative assignments possible.

(2) Proof of $m$ s-a-0 testable using PODEM.

**Step 1:** Goal – sensitize fault. Objective – $g = 1$.
Backtrace – Implication stack – $a = 1$. Implications – $g = 1, m = D, n = 0$.
$D$-frontier – $p$.

**Step 2:** Goal – propagate fault. Objective – $h = 1$.
Backtrace – Implication stack – $a = 1, b = 0$. Implications – $g = 1, m = D, n = 0, h = 1, p = D, q = D$.
$D$-frontier – $\phi$ fault at PO.
Test found – $a = 1, b = 0$; $q = D$.

(3) Redundancy removal.
A. Start with redundant fault \( d \text{ sa0} \).

B. Set fault site to the faulty state and find all implications. For \( d = 0 \), we find \( g = b \). Thus, OR gate \( g \) is removed and \( k \) and \( m \) become fanouts of PI \( b \). The reduced circuit is shown on the left in the following figure.

C. Examine the reduced circuit for another redundant fault. We find that \( m \text{ sa0} \), which was testable in the original circuit, is now redundant.

D. Repeat steps B and C until all faults in the reduced circuit are testable.

The above procedure leads to the circuit, \( q = a \oplus b \), as shown on the right in the above figure.

Note: This procedure removes only one redundant fault at a time and requires repeated use of ATPG. It is possible to remove several redundant faults together, provided they are selected such that the circuit function is preserved. Removal of a single redundant fault leaves the circuit function unchanged.

### 7.26 Static compaction

<table>
<thead>
<tr>
<th>Forward order</th>
<th>Reverse order</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 \cap t_2 = 1010 )</td>
<td>( t_5 \cap t_4 = 1100 )</td>
</tr>
<tr>
<td>( t_3 \cap t_4 = 0100 )</td>
<td>( t_3 \cap t_1 = 0010 )</td>
</tr>
<tr>
<td>( t_5 = 1100 )</td>
<td>( t_5 \cap t_4 \cap t_2 = 1100 )</td>
</tr>
</tbody>
</table>

Compacted vector sets:

- Forward order compaction: \( 1010, 0100, 1100 \)
- Reverse order compaction: \( 0010, 1100 \)

Reverse order is better, as it gives 1 less vector.
Chapter 6: Testability Measures

6.3 SCOAP

Circuit of Figure 6.21 with combinational SCOAP measures.

6.5 SCOAP

Circuit of Figure 6.23 with combinational SCOAP measures.

6.7 SCOAP

The steps of calculation for SCOAP testability measures are shown in the three figures that follow. Combinational measures are shown as \((CC0, CC1)CO\) and sequential measures as \([SC0, SC1]SO\).
Circuit of Figure 6.25: PI and PO initialization and first controllability pass.

Circuit of Figure 6.25: Converged controllability values.

Circuit of Figure 6.25: All controllability and observability values.
2 Problem Set Part B

I. (Functional Fault Equivalence)

The sum-of-products formulation of an XOR function can be written as \( A'B + AB' \). You can see that there exist 22 stuck-at faults in this 2 2-input AND, 1 2-input OR and 2 INVERTER implementation.

(Part A) Please identify equivalent faults within this set of 22 faults which can be obtained solely through functional equivalence.

\[
\begin{align*}
\text{b s-a-1 is equivalent to f s-a-1} \\
\text{c s-a-0 is equivalent to g s-a-0 (or equally d s-a-1 and h s-a-1)}
\end{align*}
\]

(Part B) The XNOR function can be written in a product-of-sums form in a rather similar representation as \((A' + B) \times (A + B')\), implemented in 2 2-input OR, 1 2-input AND and 2 INVERTER form. Please identify equivalent faults within this set of 22 faults which can be again obtained solely through functional equivalence.

\[
\begin{align*}
\text{b s-a-0 is equivalent to f s-a-0} \\
\text{c s-a-1 is equivalent to g s-a-1 (or equally d s-a-0 and h s-a-0)}
\end{align*}
\]

(Part C) The XOR function can be written in a product-of-sums form as well, albeit in a slightly different formulation as \((A' + B') \times (A + B)\), implemented in 2 2-input OR, 1 2-input AND and 2 INVERTER form. Please identify equivalent faults within this set of 22 faults which can be obtained solely through functional equivalence.

\[
\begin{align*}
\text{c s-a-0 is equivalent to h s-a-0 (and b s-a-1)} \\
\text{g s-a-0 is equivalent to d s-a-0 (and f s-a-1)}
\end{align*}
\]
II. (D-algorithm)

The carry and sum functions for a single-bit adder are defined as:

\[
\begin{align*}
\text{sum} &= x \oplus y \oplus c_{in} \\
c_{out} &= xy + xc_{in} + yc_{in}
\end{align*}
\]

Behaviorally speaking, a carry \((c_{out})\) has the value of 1 if 2 or more of the three input bits to a single bit adder \((x, y, c_{in})\) are set, while the sum \((\text{sum})\) is 1 if an odd number of the three inputs is set.

In the context of this question, these \((\text{sum})\) and \((c_{out})\) functions are shown in the design as single components, necessitating the development of D-algorithm procedures to handle them. This question asks you to extend your understanding of D-algorithm procedures that we discussed in the context of 2-input Boolean gates to such slightly larger Boolean functions.

(Part A) Please define the singular covers for the \((\text{sum})\) and \((c_{out})\) functions.

(Part B) Using the singular covers, please derive the propagation D-cubes for the \((\text{sum})\) and \((c_{out})\) functions.

(Part C) The Fault Modeling team has identified a new fault type that has been plaguing the carry \((c_{out})\) function implementation. Apparently, the carry \((c_{out})\), in an inexplicable way, is manifesting a tendency of mutating to the \((\text{sum})\) function. Please develop primitive D-cubes of failure for this fault by using cube sets \(\alpha_0\), \(\alpha_1\), \(\beta_0\), and \(\beta_1\).
III. (Advanced Combinational Test)

In this question you will be working on this circuit that should be familiar to you from our discussions in class.

(Part A) Please identify all static learning implications that SOCRATES will identify. Please refrain from including implications that will be straightforwardly derived through local propagation rule application.

The implication (k=0) => (n=0) causes SOCRATES to learn (n=1) => (k=1).
Other implications are not worth learning because they can be derived from local propagation rules together with the aforementioned implication. For example, (n=1) implies (k=1), which further implies AND((f=1), (h=1), (g=0), (D=0), (d=1), (c=1), (B=1), (C=1)) by local propagation rules. Another correct way is to write down a list of implications from (n=1) to each element in the previous AND list, although it is non-optimal in terms of the storage space.

(Part B) Please identify any headlines in this circuit. Please show how FAN will utilize such headlines in resolving a test for the s-a-0 fault at the primary output (i.e. line n).

Headlines are points where the circuit can be partitioned such that a cone of logic driven by PIs can be isolated from the rest of the circuit by cutting a single line. In the circuit above, the headlines are k (and maybe trivially A and E depending on how one interprets the definition).

To sensitize the s-a-0 fault at n, we need the correct circuit to produce a 1 at n. One way (by flipping a coin) to achieve this is to set (l=1) and (m=0). By local propagation rules and immediate implication, we get the following assignments: (A = 1), (k=1), and (E = 0). Generally, FAN defers backwards search from the headline k until the ATPG algorithm knows that it has a viable assignment for the headline. Since all of the to-be-justified signals are headlines, (k=1) is certainly a viable assignment so FAN continues to search the input space to justify (k=1) which gives us (B=1), (C=1), and (D= 0).